COMPANION WEBSITE

WRITTEN BY
CHRISTOPHER NEGUS

Linux® BIBLE

THE COMPREHENSIVE TUTORIAL RESOURCE

EXPLORE THE LATEST TOOLS AND FEATURES | MASTER INSIDE TRICKS AND BEST PRACTICES | DISCOVER WHAT YOU NEED, WHEN YOU NEED IT

TENTH EDITION

WILEY

Linux®
Bible

Tenth Edition

Linux®
BIBLE

Tenth Edition

Christopher Negus

WILEY

Copyright © 2020 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-119-57888-8
ISBN: 978-1-119-57891-8 (ebk)
ISBN: 978-1-119-57889-5 (ebk)

Manufactured in the United States of America

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at http://booksupport.wiley.com. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2019956690

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Linux is a registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

As always, I dedicate this book to my wife, Sheree.

About the Author

Chris Negus is a principal technical writer for Red Hat, Inc. In more than a decade with Red Hat, Chris has taught hundreds of IT professionals to become Red Hat Certified Engineers (RHCEs), and he has written scores of documents on everything from Linux to virtualization to cloud computing and containerization.

Before joining Red Hat, Chris wrote or co-wrote dozens of books on Linux and UNIX, including the *Red Hat Linux Bible* (all editions), *Docker Containers*, *CentOS Bible*, *Fedora Bible*, *Linux Troubleshooting Bible*, *Linux Toys*, *Linux Toys II*, and, nine editions of this *Linux Bible*. Chris also co-authored several books for the Linux Toolbox series for power users: *Fedora Linux Toolbox*, *SUSE Linux Toolbox*, *Ubuntu Linux Toolbox*, *Mac OS X Toolbox*, and *BSD UNIX Toolbox*.

Before becoming an independent author, Chris worked for eight years with the organization at AT&T that developed the UNIX operating system before moving to Utah to help contribute to Novell's UnixWare project in the early 1990s. When not writing about Linux, Chris enjoys playing soccer, hanging out with his wife, Sheree, and spending what time he can with his sons, Seth and Caleb.

About the Technical Editors

Jason W. Eckert is an experienced technical trainer, consultant, and best-selling author in the Information Technology (IT) industry. With 45 industry certifications, over 30 years of IT experience, 4 published apps, and 24 published textbooks covering topics such as UNIX, Linux, security, Windows Server, Microsoft Exchange Server, PowerShell, BlackBerry Enterprise Server, and video game development, Mr. Eckert brings his expertise to every class that he teaches at triOS College in his role as the Dean of Technology. For more information about Mr. Eckert, visit `jasoneckert.net`.

Derrick Ornelas is a senior software maintenance engineer at Red Hat, Inc. In his current role as a product lead for Red Hat container technologies, including OpenShift Container Platform and Red Hat Enterprise Linux CoreOS, Derrick works to ensure both the supportability and quality of Red Hat's products. Previously, he worked as a senior technical support lead for Red Hat virtualization technologies, such as libvirt, KVM, and the Red Hat Virtualization product.

During his 12 years at Red Hat, Derrick earned the Red Hat Certified Engineer and Red Hat Certified Virtualization Administrator certifications, and he has applied his broad Linux knowledge to architect, deploy, and maintain various hardware labs and applications.

Derrick's nearly two decades of Linux experience began while earning his BS in Computer Science from Appalachian State University. As a devoted Linux supporter, he enjoys teaching and assisting new Linux users both on and off the clock. When he's not working on his monitor tan, Derrick enjoys mountain biking, motorcycling, and backpacking with his wife, Carolyn.

Acknowledgments

When I was hired at Red Hat about a dozen years ago, I didn't know that Red Hat would grow to about seven times its size, be bought by IBM for $34 billion, and (so far) still maintain the spirit of openness and excitement that it had when I first signed on. Every day when I come to work, I interact with many of the greatest Linux and cloud developers, testers, instructors, and support professionals in the world.

While I can't thank everyone individually, I would like to salute the culture of cooperation and excellence at Red Hat that serves to improve my own Linux skills every day. I don't speak well of Red Hat because I work there; I work at Red Hat because it lives up to the ideals of open source software in ways that match my own beliefs.

That said, there are a few Red Hatters that I want to acknowledge in particular. At Red Hat, I'm able to take on so many cool and challenging projects because of the freedom that I receive from the people to whom I report. They include Michelle Bearer, Dawn Eisner, and Sam Knuth. Sam, in particular, has had my back and encouraged my work for more than a decade.

In my daily work, I want to give a shout out to Red Hatters Scott McCarty, Ben Breard, Laurie Friedman, Dave Darrah, Micah Abbott, Steve Milner, and Ian McLeod (container tools, RHCOS, and OpenShift teams), and Tom McKay, Joey Schorr, Bill Dettelback, Richa Marwaha, and Dirk Herrmann (Quay team). Finally, a special thank you to Vikram Goyal, who luckily lives in Australia, so he is always available to bail me out when I blow up git in the middle of the night.

When it comes to support for writing this book, I have had the luxury of two excellent technical editors: Jason Eckert and Derrick Ornelas. I didn't know Jason before he took on this role, but his broad experience with different Linux systems has helped call me out when I get too Red Hat centric. Derrick, who I see almost every day, was asked to do this work because of his attention to detail and deep understanding of how Linux works and what people need to know to use it. Anyone reading this book will have a better experience because of the work that Jason and Derrick have done reviewing it.

As for the people at Wiley, thanks for letting me continue to develop and improve this book over the years. Thanks to Gary Schwartz, who applies constant, gentle pressure to keep me working on this book at times when I had no spare cycles to work on it. When Gary's pressure wasn't enough, Devon Lewis would step in to paint a clearer picture about the importance of deadlines. Thanks also to Margot Maley Hutchison from Waterside Productions for contracting the book for me with Wiley and always looking out for my best interests.

Finally, thanks to my wife, Sheree, for sharing her life with me and doing such a great job raising Seth and Caleb.

—Christopher Negus

Contents at a Glance

Acknowledgments xi
Introduction xxxv

Part I: Getting Started 1
Chapter 1: Starting with Linux 3
Chapter 2: Creating the Perfect Linux Desktop 27

Part II: Becoming a Linux Power User 59
Chapter 3: Using the Shell 61
Chapter 4: Moving Around the Filesystem 93
Chapter 5: Working with Text Files 113
Chapter 6: Managing Running Processes 131
Chapter 7: Writing Simple Shell Scripts 147

Part III: Becoming a Linux System Administrator 165
Chapter 8: Learning System Administration 167
Chapter 9: Installing Linux 195
Chapter 10: Getting and Managing Software 221
Chapter 11: Managing User Accounts 249
Chapter 12: Managing Disks and Filesystems 273

Part IV: Becoming a Linux Server Administrator 305
Chapter 13: Understanding Server Administration 307
Chapter 14: Administering Networking 339
Chapter 15: Starting and Stopping Services 369
Chapter 16: Configuring a Print Server 403
Chapter 17: Configuring a Web Server 427
Chapter 18: Configuring an FTP Server 455
Chapter 19: Configuring a Windows File Sharing (Samba) Server 475
Chapter 20: Configuring an NFS File Server 499
Chapter 21: Troubleshooting Linux 523

Part V: Learning Linux Security Techniques 563
Chapter 22: Understanding Basic Linux Security 565
Chapter 23: Understanding Advanced Linux Security 599
Chapter 24: Enhancing Linux Security with SELinux 635
Chapter 25: Securing Linux on a Network 663

Part VI: Engaging with Cloud Computing 691
Chapter 26: Shifting to Clouds and Containers 693
Chapter 27: Using Linux for Cloud Computing 709
Chapter 28: Deploying Linux to the Cloud 729
Chapter 29: Automating Apps and Infrastructure with Ansible 749
Chapter 30: Deploying Applications as Containers with Kubernetes 765

Part VII: Appendixes 785
Appendix A: Media 787
Appendix B: Exercise Answers 797

Index 863

Contents

Acknowledgments xi

Introduction xxxv

Part I: Getting Started 1

Chapter 1: Starting with Linux 3

Understanding What Linux Is 4
Understanding How Linux Differs from Other Operating Systems 6
Exploring Linux History 7
Free-flowing UNIX culture at Bell Labs 7
Commercial UNIX 9
Berkeley Software Distribution arrives 9
UNIX Laboratory and commercialization 10
GNU transitions UNIX to freedom 11
BSD loses some steam 12
Linus builds the missing piece 13
OSI open source definition 14
Understanding How Linux Distributions Emerged 16
Choosing a Red Hat distribution 16
Using Red Hat Enterprise Linux 17
Using Fedora 18
Choosing Ubuntu or another Debian distribution 19
Finding Professional Opportunities with Linux Today 19
Understanding how companies make money with Linux 20
Becoming Red Hat certified 21
RHCSA topics 22
RHCE topics 23
Summary 25

Chapter 2: Creating the Perfect Linux Desktop 27

Understanding Linux Desktop Technology 28
Starting with the Fedora GNOME Desktop Live image 30
Using the GNOME 3 Desktop 31
After the computer boots up 31
Navigating with the mouse 32
Navigating with the keyboard 36
Setting up the GNOME 3 desktop 38

Extending the GNOME 3 desktop 39
Using GNOME shell extensions 39
Using the GNOME Tweak Tool 40
Starting with desktop applications 41
Managing files and folders with Nautilus 42
Installing and managing additional software 43
Playing music with Rhythmbox 45
Stopping the GNOME 3 desktop 46
Using the GNOME 2 Desktop 46
Using the Metacity window manager 48
Changing GNOME's appearance 49
Using the GNOME panels 50
Using the Applications and System menus 51
Adding an applet 51
Adding another panel 52
Adding an application launcher 52
Adding a drawer 53
Changing panel properties 54
Adding 3D effects with AIGLX 54
Summary 57
Exercises 57

Part II: Becoming a Linux Power User 59

Chapter 3: Using the Shell. 61

About Shells and Terminal Windows 62
Using the shell prompt 63
Using a Terminal window 63
Using virtual consoles 65
Choosing Your Shell 65
Running Commands 66
Understanding command syntax 67
Locating commands 70
Recalling Commands Using Command History 72
Command-line editing 73
Command-line completion 75
Command-line recall 76
Connecting and Expanding Commands 78
Piping between commands 78
Sequential commands 79
Background commands 79
Expanding commands 80
Expanding arithmetic expressions 80
Expanding variables 80

Using Shell Variables....81
Creating and using aliases....81
Exiting the shell....83
Creating Your Shell Environment....84
Configuring your shell....84
Setting your prompt....85
Adding environment variables....87
Getting Information about Commands....88
Summary....90
Exercises....90

Chapter 4: Moving Around the Filesystem....93

Using Basic Filesystem Commands....96
Using Metacharacters and Operators....98
Using file-matching metacharacters....98
Using file-redirection metacharacters....99
Using brace expansion characters....101
Listing Files and Directories....101
Understanding File Permissions and Ownership....105
Changing permissions with chmod (numbers)....106
Changing permissions with chmod (letters)....107
Setting default file permission with umask....108
Changing file ownership....109
Moving, Copying, and Removing Files....109
Summary....111
Exercises....111

Chapter 5: Working with Text Files....113

Editing Files with vim and vi....113
Starting with vi....115
Adding text....115
Moving around in the text....116
Deleting, copying, and changing text....117
Pasting (putting) text....118
Repeating commands....118
Exiting vi....118
Skipping around in the file....119
Searching for text....119
Using ex mode....120
Learning more about vi and vim....120
Finding Files....120
Using locate to find files by name....121
Searching for files with find....122
Finding files by name....123
Finding files by size....124

Finding files by user 124
Finding files by permission 125
Finding files by date and time 125
Using 'not' and 'or' when finding files 126
Finding files and executing commands 127
Searching in files with grep 128
Summary 129
Exercises 129

Chapter 6: Managing Running Processes 131

Understanding Processes 131
Listing Processes 132
Listing processes with ps 132
Listing and changing processes with top 134
Listing processes with System Monitor 136
Managing Background and Foreground Processes 137
Starting background processes 138
Using foreground and background commands 139
Killing and Renicing Processes 140
Killing processes with kill and killall 140
Using kill to signal processes by **PID** 140
Using killall to signal processes by name 141
Setting processor priority with nice and renice 142
Limiting Processes with cgroups 143
Summary 144
Exercises 145

Chapter 7: Writing Simple Shell Scripts 147

Understanding Shell Scripts 147
Executing and debugging shell scripts 148
Understanding shell variables 149
Special shell positional parameters 150
Reading in parameters 151
Parameter expansion in bash 151
Performing arithmetic in shell scripts 152
Using programming constructs in shell scripts 153
The "if. . .then" statements 153
The case command 156
The "for. . .do" loop 157
The "while. . .do" and "until. . .do" loops 158
Trying some useful text manipulation programs 159
The general regular expression parser 159
Remove sections of lines of text (cut) 159
Translate or delete characters (tr) 160
The stream editor (sed) 160

Using simple shell scripts ... 161
Telephone list ... 161
Backup script ... 162
Summary ... 163
Exercises ... 163

Part III: Becoming a Linux System Administrator 165

Chapter 8: Learning System Administration ... 167

Understanding System Administration ... 167
Using Graphical Administration Tools ... 169
Using Cockpit browser-based administration ... 169
Using system-config-* tools ... 171
Using other browser-based admin tools ... 173
Using the root User Account ... 174
Becoming root from the shell (su command) ... 175
Allowing administrative access via the GUI ... 176
Gaining administrative access with sudo ... 176
Exploring Administrative Commands, Configuration Files, and Log Files ... 178
Administrative commands ... 178
Administrative configuration files ... 179
Administrative log files and systemd journal ... 183
Using journalctl to view the systemd journal ... 184
Managing log messages with rsyslogd ... 184
Using Other Administrative Accounts ... 185
Checking and Configuring Hardware ... 186
Checking your hardware ... 187
Managing removable hardware ... 189
Working with loadable modules ... 191
Listing loaded modules ... 191
Loading modules ... 192
Removing modules ... 192
Summary ... 193
Exercises ... 193

Chapter 9: Installing Linux ... 195

Choosing a Computer ... 196
Installing Fedora from Live Media ... 198
Installing Red Hat Enterprise Linux from Installation Media ... 201
Understanding Cloud-Based Installations ... 204
Installing Linux in the Enterprise ... 205
Exploring Common Installation Topics ... 207
Upgrading or installing from scratch ... 207
Dual booting ... 208

Installing Linux to run virtually 209
Using installation boot options 210
Boot options for disabling features 210
Boot options for video problems 210
Boot options for special installation types 210
Boot options for kickstarts and remote repositories 211
Miscellaneous boot options 212
Using specialized storage 213
Partitioning hard drives 214
Understanding different partition types 215
Tips for creating partitions 215
Using the GRUB boot loader 217
Summary 219
Exercises 219

Chapter 10: Getting and Managing Software 221

Managing Software on the Desktop 221
Going Beyond the Software Window 223
Understanding Linux RPM and DEB Software Packaging 224
Understanding DEB packaging 225
Understanding RPM packaging 226
What is in an RPM? 226
Where do RPMs come from? 227
Installing RPMs 228
Managing RPM Packages with YUM 229
Transitioning from yum to dnf 229
Understanding how yum works 229
Using YUM with third-party software repositories 233
Managing software with the yum command 233
Searching for packages 234
Installing and removing packages 236
Updating packages 238
Updating groups of packages 239
Maintaining your RPM package database and cache 240
Downloading RPMs from a YUM repository 241
Installing, Querying, and Verifying Software with the rpm Command 241
Installing and removing packages with rpm 241
Querying rpm information 242
Verifying RPM packages 244
Managing Software in the Enterprise 245
Summary 246
Exercises 247

Chapter 11: Managing User Accounts 249
Creating User Accounts 249
Adding users with useradd 252
Setting user defaults 255
Modifying users with usermod 257
Deleting users with userdel 258
Understanding Group Accounts 259
Using group accounts 259
Creating group accounts 260
Managing Users in the Enterprise 261
Setting permissions with Access Control Lists 262
Setting ACLs with setfacl 262
Setting default ACLs 264
Enabling ACLs 265
Adding directories for users to collaborate 267
Creating group collaboration directories (set GID bit) 267
Creating restricted deletion directories (sticky bit) 268
Centralizing User Accounts 269
Summary 270
Exercises 270

Chapter 12: Managing Disks and Filesystems 273
Understanding Disk Storage 273
Partitioning Hard Disks 275
Understanding partition tables 275
Viewing disk partitions 276
Creating a single-partition disk 277
Creating a multiple-partition disk 281
Using Logical Volume Manager Partitions 285
Checking an existing LVM 286
Creating LVM logical volumes 289
Growing LVM logical volumes 290
Mounting Filesystems 291
Supported filesystems 291
Enabling swap areas 293
Disabling swap area 294
Using the fstab file to define mountable file systems 295
Using the mount command to mount file systems 297
Mounting a disk image in loopback 298
Using the umount command 299
Using the mkfs Command to Create a Filesystem 300
Managing Storage with Cockpit 301
Summary 303
Exercises 303

Part IV: Becoming a Linux Server Administrator 305

Chapter 13: Understanding Server Administration 307

Starting with Server Administration 308
Step 1: Install the server 308
Step 2: Configure the server 310
Using configuration files 310
Checking the default configuration 310
Step 3: Start the server 311
Step 4: Secure the server 312
Password protection 312
Firewalls 313
TCP Wrappers 313
SELinux 313
Security settings in configuration files 314
Step 5: Monitor the server 314
Configure logging 314
Run system activity reports 314
Watch activity live with Cockpit 314
Keep system software up to date 315
Check the filesystem for signs of crackers 315
Checking and Setting Servers 316
Managing Remote Access with the Secure Shell Service 316
Starting the openssh-server service 317
Using SSH client tools 318
Using ssh for remote login 318
Using ssh for remote execution 320
Copying files between systems with scp and rsync 321
Interactive copying with sftp 324
Using key-based (passwordless) authentication 324
Configuring System Logging 326
Enabling system logging with rsyslog 326
Understanding the rsyslog.conf file 327
Understanding the messages log file 329
Setting up and using a loghost with rsyslogd 330
Watching logs with logwatch 331
Checking System Resources with sar 332
Checking System Space 334
Displaying system space with df 334
Checking disk usage with du 334
Finding disk consumption with find 335
Managing Servers in the Enterprise 336
Summary 336
Exercises 337

Chapter 14: Administering Networking 339
Configuring Networking for Desktops 340
Checking your network interfaces 342
Checking your network from NetworkManager 342
Checking your network from Cockpit 343
Checking your network from the command line 345
Configuring network interfaces 349
Setting IP addresses manually 349
Setting IP address aliases 350
Setting routes 351
Configuring a network proxy connection 352
Configuring Networking from the Command Line 353
Configure networking with nmtui 354
Editing a NetworkManager TUI connection 354
Understanding networking configuration files 355
Network interface files 356
Other networking files 358
Setting alias network interfaces 360
Setting up Ethernet channel bonding 361
Setting custom routes 363
Configuring Networking in the Enterprise 364
Configuring Linux as a router 364
Configuring Linux as a DHCP server 365
Configuring Linux as a DNS server 365
Configuring Linux as a proxy server 366
Summary 366
Exercises 367
Chapter 15: Starting and Stopping Services 369
Understanding the Initialization Daemon (init or systemd) 370
Understanding the classic init daemons 371
Understanding systemd initialization 377
Learning systemd basics 377
Learning systemd's backward compatibility to SysVinit 382
Checking the Status of Services 384
Checking services for SysVinit systems 385
Stopping and Starting Services 387
Stopping and starting SysVinit services 387
Stopping a service with systemd 389
Starting a service with systemd 389
Restarting a service with systemd 389
Reloading a service with systemd 390
Enabling Persistent Services 391
Configuring persistent services for SysVinit 391

Enabling a service with systemd 392
Disabling a service with systemd 393
Configuring a Default Runlevel or Target Unit 394
Configuring the SysVinit default runlevel 394
Adding New or Customized Services 396
Adding new services to SysVinit 396
Step 1: Create a new or customized service script file 396
Step 2: Add the service script to /etc/rc.d/init.d 398
Step 3: Set appropriate permission on the script 398
Step 4: Add the service to runlevel directories 398
Adding new services to systemd 399
Step 1: Create a new or customized service configuration unit file 399
Step 2: Move the service configuration unit file 399
Step 3: Add the service to the Wants directory 400
Summary 401
Exercises 401

Chapter 16: Configuring a Print Server 403

Common UNIX Printing System 403
Setting Up Printers 405
Adding a printer automatically 405
Using web-based CUPS administration 406
Allow remote printing administration 406
Add a printer not automatically detected 407
Using the Print Settings window 409
Configuring local printers with the Print Settings window 409
Configuring remote printers 412
Adding a remote CUPS printer 413
Adding a remote UNIX (LDP/LPR) printer 413
Adding a Windows (SMB) printer 414
Working with CUPS Printing 415
Configuring the CUPS server (cupsd.conf) 415
Starting the CUPS server 417
Configuring CUPS printer options manually 417
Using Printing Commands 418
Printing with lp 419
Listing status with lpstat -t 419
Removing print jobs with lprm 419
Configuring Print Servers 420
Configuring a shared CUPS printer 420
Configuring a shared Samba printer 422
Understanding `smb`.conf for printing 422
Setting up SMB clients 423
Summary 424
Exercises 424

Chapter 17: Configuring a Web Server 427
Understanding the Apache Web Server 427
Getting and Installing Your Web Server 428
Understanding the httpd package 428
Installing Apache 431
Starting Apache 432
Securing Apache 433
Apache file permissions and ownership 433
Apache and firewalls 433
Apache and SELinux 434
Understanding the Apache configuration files 435
Using directives 435
Understanding default settings 438
Adding a virtual host to Apache 440
Allowing users to publish their own web content 442
Securing your web traffic with SSL/TLS 443
Understanding how SSL is configured 445
Generating an SSL key and self-signed certificate 447
Generating a certificate signing request 448
Troubleshooting Your Web Server 449
Checking for configuration errors 449
Accessing forbidden and server internal errors 451
Summary 453
Exercises 453

Chapter 18: Configuring an FTP Server 455
Understanding FTP 455
Installing the vsftpd FTP Server 457
Starting the vsftpd Service 458
Securing Your FTP Server 461
Opening up your firewall for FTP 461
Configuring SELinux for your FTP server 463
Relating Linux file permissions to vsftpd 465
Configuring Your FTP Server 465
Setting up user access 465
Allowing uploading 467
Setting up vsftpd for the Internet 468
Using FTP Clients to Connect to Your Server 469
Accessing an FTP server from Firefox 470
Accessing an FTP server with the lftp command 470
Using the gFTP client 472
Summary 473
Exercises 473

Chapter 19: Configuring a Windows File Sharing (Samba) Server. 475

Understanding Samba 475
Installing Samba 476
Starting and Stopping Samba 478
Starting the Samba (smb) service 478
Starting the NetBIOS (nmbd) name server 480
Stopping the Samba (smb) and NetBIOS (nmb) services 481
Securing Samba 482
Configuring firewalls for Samba 482
Configuring SELinux for Samba 484
Setting SELinux Booleans for Samba 484
Setting SELinux file contexts for Samba 485
Configuring Samba host/user permissions 486
Configuring Samba 486
Configuring the [global] section 486
Configuring the [homes] section 487
Configuring the [printers] section 489
Creating a Samba shared folder 489
Adding the shared folder to Samba 490
Checking the Samba share 490
Accessing Samba Shares 493
Accessing Samba shares in Linux 493
Accessing Samba shares from a Linux file manager 493
Mounting a Samba share from a Linux command line 495
Accessing Samba shares in Windows 496
Using Samba in the Enterprise 497
Summary 497
Exercises 498

Chapter 20: Configuring an NFS File Server. 499

Installing an NFS Server 502
Starting the NFS service 502
Sharing NFS Filesystems 503
Configuring the /etc/exports file 504
Hostnames in /etc/exports 505
Access options in /etc/exports 506
User mapping options in /etc/exports 506
Exporting the shared filesystems 507
Securing Your NFS Server 508
Opening up your firewall for NFS 508
Allowing NFS access in TCP wrappers 510
Configuring SELinux for your NFS server 511
Using NFS Filesystems 512
Viewing NFS shares 512

Manually mounting an NFS filesystem ... 512
Mounting an NFS filesystem at boot time ... 513
Mounting noauto filesystems ... 514
Using mount options ... 515
Using autofs to mount NFS filesystems on demand ... 517
Automounting to the /net directory ... 517
Automounting home directories ... 518
Unmounting NFS filesystems ... 520
Summary ... 521
Exercises ... 521

Chapter 21: Troubleshooting Linux ... 523

Boot-Up Troubleshooting ... 523
Understanding Startup Methods ... 524
Starting with System V init scripts ... 524
Starting with systemd ... 525
Starting from the firmware (BIOS or UEFI) ... 526
Troubleshooting BIOS setup ... 526
Troubleshooting boot order ... 527
Troubleshooting the GRUB boot loader ... 528
The GRUB Legacy boot loader ... 528
GRUB 2 Boot loader ... 530
Starting the kernel ... 532
Troubleshooting the initialization system ... 533
Troubleshooting System V initialization ... 533
Troubleshooting rc.sysinit ... 533
Troubleshooting runlevel processes ... 534
Troubleshooting systemd initialization ... 538
Troubleshooting Software Packages ... 542
Fixing RPM databases and cache ... 545
Troubleshooting Networking ... 547
Troubleshooting outgoing connections ... 547
View network interfaces ... 547
Check physical connections ... 548
Check routes ... 548
Check hostname resolution ... 549
Troubleshooting incoming connections ... 550
Check if the client can reach your system at all ... 551
Check if the service is available to the client ... 551
Check the firewall on the server ... 552
Check the service on the server ... 553
Troubleshooting Memory ... 553
Uncovering memory issues ... 554
Checking for memory problems ... 556
Dealing with memory problems ... 558

Troubleshooting in Rescue Mode 559
Summary 561
Exercises 561

Part V: Learning Linux Security Techniques 563

Chapter 22: Understanding Basic Linux Security 565

Implementing Physical Security 565
Implementing disaster recovery 566
Securing user accounts 566
One user per user account 567
Limiting access to the root user account 567
Setting expiration dates on temporary accounts 567
Removing unused user accounts 568
Securing passwords 570
Choosing good passwords 570
Setting and changing passwords 571
Enforcing best password practices 572
Understanding the password files and password hashes 574
Securing the filesystem 576
Managing dangerous filesystem permissions 576
Securing the password files 577
Locking down the filesystem 578
Managing software and services 579
Updating software packages 579
Keeping up with security advisories 580
Advanced implementation 580
Monitoring Your Systems 580
Monitoring log files 581
Monitoring user accounts 584
Detecting counterfeit new accounts and privileges 584
Detecting bad account passwords 586
Monitoring the filesystem 587
Verifying software packages 588
Scanning the filesystem 589
Detecting viruses and rootkits 590
Auditing and Reviewing Linux 595
Conducting compliance reviews 595
Conducting security reviews 596
Summary 596
Exercises 597

Chapter 23: Understanding Advanced Linux Security 599

Implementing Linux Security with Cryptography 599
Understanding hashing 600

Understanding encryption/decryption 602
Understanding cryptographic ciphers 602
Understanding cryptographic cipher keys 603
Understanding digital signatures 608
Implementing Linux cryptography 610
Ensuring file integrity 610
Encrypting a Linux filesystem at installation 611
Encrypting a Linux directory 613
Encrypting a Linux file 616
Encrypting Linux with miscellaneous tools 616
Using Encryption from the Desktop 617
Implementing Linux Security with PAM 618
Understanding the PAM authentication process 619
Understanding PAM contexts 619
Understanding PAM control flags 620
Understanding PAM modules 621
Administering PAM on your Linux system 622
Managing PAM-aware application configuration files 622
Managing PAM system event configuration files 623
Implementing time restrictions with PAM 626
Enforcing good passwords with PAM 628
Encouraging sudo use with PAM 632
Obtaining more information on PAM 633
Summary 633
Exercises 633

Chapter 24: Enhancing Linux Security with SELinux 635

Understanding SELinux Benefits 635
Understanding How SELinux Works 637
Understanding Type Enforcement 637
Understanding Multi-Level Security 638
Implementing SELinux security models 639
Understanding SELinux operational modes 639
Understanding SELinux security contexts 640
Understanding SELinux Policy types 643
Understanding SELinux policy rule packages 644
Configuring SELinux 645
Setting the SELinux mode 645
Setting the SELinux policy type 647
Managing SELinux security contexts 648
Managing the user security context 649
Managing the file security context 650
Managing the process security context 651
Managing SELinux policy rule packages 651
Managing SELinux via Booleans 653
Monitoring and Troubleshooting SELinux 654

Understanding SELinux logging 654
Reviewing SELinux messages in the audit log 655
Reviewing SELinux messages in the messages log 655
Troubleshooting SELinux logging 656
Troubleshooting common SELinux problems 657
Using a nonstandard directory for a service 657
Using a nonstandard port for a service 658
Moving files and losing security context labels 658
Booleans set incorrectly 658
Putting It All Together 659
Obtaining More Information on SELinux 659
Summary 660
Exercises 660

Chapter 25: Securing Linux on a Network 663

Auditing Network Services 663
Evaluating access to network services with nmap 665
Using nmap to audit your network services advertisements 668
Working with Firewalls 672
Understanding firewalls 673
Implementing firewalls 674
Starting with firewalld 675
Changing firewall rules with Cockpit 677
Understanding the iptables utility 678
Using the iptables utility 680
Summary 688
Exercises 688

Part VI: Engaging with Cloud Computing 691

Chapter 26: Shifting to Clouds and Containers 693

Understanding Linux Containers 694
Namespaces 695
Container registries 695
Base images and layers 696
Starting with Linux Containers 697
Pulling and running containers 697
Pulling a container 697
Running a shell from a container 698
Running an FTP server from a container 699
Starting and stopping containers 701
Building a container image 702
Build a simple container image 702
Build an FTP container from GitHub 703

Tagging and pushing an image to a registry 705
Using containers in the enterprise 706
Summary 706
Exercises 707

Chapter 27: Using Linux for Cloud Computing 709

Overview of Linux and Cloud Computing 710
Cloud hypervisors (aka compute nodes) 710
Cloud controllers 711
Cloud storage 711
Cloud authentication 712
Cloud deployment and configuration 712
Cloud platforms 712
Trying Basic Cloud Technology 713
Setting Up a Small Cloud 714
Configuring hypervisors 715
Step 1: Get Linux software 715
Step 2: Check your computers 715
Step 3: Install Linux on hypervisors 716
Step 4: Start services on the hypervisors 717
Step 5: Edit /etc/hosts, or set up DNS 718
Configuring storage 718
Step 1: Install Linux software 718
Step 2: Configure NFS share 719
Step 3: Start the NFS service 719
Step 4: Mount the NFS share on the hypervisors 720
Creating virtual machines 720
Step 1: Get images to make virtual machines 721
Step 2: Check the network bridge 721
Step 3: Start Virtual Machine Manager (virt-manager) 722
Step 4: Check connection details 722
Step 5: Create a new virtual machine 722
Managing virtual machines 724
Migrating virtual machines 725
Step 1: Identify other hypervisors 726
Step 2: Migrate running VM to Other hypervisor 726
Summary 727
Exercises 727

Chapter 28: Deploying Linux to the Cloud 729

Getting Linux to Run in a Cloud 729
Creating Linux Images for Clouds 731
Configuring and running a cloud-init cloud instance 731
Investigating the cloud instance 733
Cloning the cloud instance 734
Expanding your cloud-init configuration 735

Adding ssh keys with cloud-init ... 736
Adding software with cloud-init ... 737
Using cloud-init in enterprise computing ... 738
Using OpenStack to Deploy Cloud Images ... 739
Starting from the OpenStack Dashboard ... 739
Configuring your OpenStack virtual network ... 739
Configuring keys for remote access ... 741
Launching a virtual machine in OpenStack ... 742
Accessing the virtual machine via ssh ... 743
Using Amazon EC2 to Deploy Cloud Images ... 744
Summary ... 746
Exercises ... 746

Chapter 29: Automating Apps and Infrastructure with Ansible ... 749

Understanding Ansible ... 750
Exploring Ansible Components ... 751
Inventories ... 751
Playbooks ... 752
Plays ... 752
Tasks ... 752
Modules ... 752
Roles, imports, and includes ... 753
Stepping Through an Ansible Deployment ... 753
Prerequisites ... 754
Setting up SSH keys to each node ... 754
Installing Ansible ... 756
Creating an inventory ... 756
Authenticating to the hosts ... 757
Creating a playbook ... 757
Run the playbook ... 758
Running Ad-Hoc Ansible Commands ... 760
Trying ad-hoc commands ... 761
Automating Tasks with Ansible Tower Automation Framework ... 762
Summary ... 763
Exercises ... 763

Chapter 30: Deploying Applications as Containers with Kubernetes ... 765

Understanding Kubernetes ... 766
Kubernetes masters ... 766
Kubernetes workers ... 767
Kubernetes applications ... 767
Kubernetes interfaces ... 768
Trying Kubernetes ... 768
Getting Kubernetes ... 769
Starting the Kubernetes Basics Tutorial ... 769
Starting Minikube ... 770

Running the Kubernetes Basics tutorial 771
Get information about your cluster 771
Deploy a Kubernetes application 772
Get information on the deployment's pods 773
Expose applications with services 776
Label a service 777
Delete a service 778
Scale up an application 779
Check the load balancer 780
Scale down an application 781
Enterprise-Quality Kubernetes with OpenShift 782
Summary 783
Exercises 783

Part VII: Appendixes **785**

Appendix A: Media 787

Appendix B: Exercise Answers 797

Index 863

Introduction

You can't learn Linux without using it.

I've come to that conclusion after more than two decades of teaching people how to use Linux. You can't just read a book; you can't just listen to a lecture. You need someone to guide you, and you need to jump in and do it yourself.

In 1999, I wrote my first Linux book, the *Red Hat Linux Bible*. The book's huge success gave me the opportunity to become a full-time, independent Linux author. For about a decade, I wrote dozens of Linux books and explored the best ways to explain Linux from the quiet of my small home office.

In 2008, I hit the road. I was hired by Red Hat, Inc., as a full-time instructor, teaching Linux to professional system administrators seeking Red Hat Certified Engineer (RHCE) certification. In my three years as a Linux instructor, I honed my teaching skills in front of a live audience whose Linux experience ranged from none to experienced professionals. Over time, I was able to broaden my own knowledge of Linux by acquiring about 10 certifications, including the Red Hat Certified Architect (RHCA) certification.

In the previous edition of the *Linux Bible*, I turned my teaching experience into text to take a reader from someone who has never used Linux to someone with the foundational skills to become a Linux professional. The skills that you could acquire from that edition remain in effect in this edition as well. They include the following:

Beginner to certified professional: As long as you have used a computer, mouse, and keyboard, you can start with this book. I tell you how to get Linux, begin using it, step through critical topics, and ultimately excel at administering and securing it.

System administrator focused: When you are finished with this book, you will know how to use Linux and how to modify and maintain it. Almost all of the topics needed to become a Red Hat Certified Engineer are introduced in this book. That said, many software developers have also used this book to understand how to work on a Linux system as a development platform or target for their applications.

Emphasis on command-line tools: Although point-and-click windows for managing Linux have improved greatly in recent years, many advanced features can only be utilized by entering commands and editing configuration files manually. I teach you how to become proficient with the Linux command-line shell, and I occasionally compare shell features with graphical tools for accomplishing the same tasks.

Aimed at fewer Linux distributions: In past editions, I described about 18 different Linux distributions. With only a few notable exceptions, most popular Linux distributions are either Red Hat based (Red Hat Enterprise Linux, Fedora, CentOS,

and so on) or Debian based (Ubuntu, Linux Mint, KNOPPIX, and so forth). Although this book most thoroughly covers Red Hat distributions, I increased the coverage of Ubuntu throughout the book, because that's where many of the biggest Linux fans begin.

Many, many demos and exercises: Instead of just telling you what Linux does, I actually show you what it does. Then, to make sure that you got it, you have the opportunity to try Linux exercises yourself. Every procedure and exercise has been tested to work in Fedora or Red Hat Enterprise Linux. Most work in Ubuntu as well.

For this 10th edition, major enhancements include a focus on simplified Linux administration, automating tasks, and managing containerized applications (individually or at scale):

Cockpit administration web UI: Since Linux was created, people have tried to develop simple graphical or browser-based interfaces for managing Linux systems. I believe that Cockpit is the best web UI ever created for managing most basic Linux features. Throughout this book, I have replaced most older system-config* tool descriptions with those focusing on Cockpit. With Cockpit, you can now add users, manage storage, monitor activities, and do many other administrative tasks through a single interface.

Lead into cloud technologies: After introducing cloud technologies in the previous edition, I've expanded on that coverage here. This coverage includes setting up your own Linux host for running virtual machines and running Linux in a cloud environment, such as Amazon Web Services. Linux is at the heart of most technological advances in cloud computing today. That means you need a solid understanding of Linux to work effectively in tomorrow's data centers. I help you learn Linux basics in the front of this book. Then in the last few chapters, I demonstrate how you can try out Linux systems as hypervisors, cloud controllers, and virtual machines as well as manage virtual networks and networked storage.

Ansible: Automating tasks for managing systems is becoming more and more essential in modern data centers. Using Ansible, you can create playbooks that define the state of a Linux system. This includes things like setting which packages are installed, which services are running, and how features are configured. A playbook can configure one system or a thousand systems, be combined to form a set of system services, and be run again to return a system to a defined state. In this edition, I introduce you to Ansible, help you create your first Ansible playbook, and show you how to run ad-hoc Ansible commands.

Containers: Packaging and running applications in containers is becoming the preferred method for deploying, managing, and updating small, scalable software services and features. I describe how to pull containers to your system, run them, stop them, and even build your own container images using `podman` and `docker` commands.

Kubernetes and OpenShift: While containers are nice on their own, to be able to deploy, manage, and upgrade containers in a large enterprise, you need an orchestration platform. The Kubernetes project provides that platform. For a commercial, supported Kubernetes platform, you can use a product such as OpenShift.

How This Book Is Organized

The book is organized to enable you to start off at the very beginning with Linux and grow to become a professional Linux system administrator and power user.

Part I, "Getting Started," includes two chapters designed to help you understand what Linux is and get you started with a Linux desktop:

- Chapter 1, "Starting with Linux," covers topics such as what the Linux operating system is, where it comes from, and how to get started using it.
- Chapter 2, "Creating the Perfect Linux Desktop," provides information on how you can create a desktop system and use some of the most popular desktop features.

Part II, "Becoming a Linux Power User," provides in-depth details on how to use the Linux shell, work with filesystems, manipulate text files, manage processes, and use shell scripts:

- Chapter 3, "Using the Shell," includes information on how to access a shell, run commands, recall commands (using history), and do tab completion. The chapter also describes how to use variables, aliases, and man pages (traditional Linux command reference pages).
- Chapter 4, "Moving Around the Filesystem," includes commands for listing, creating, copying, and moving files and directories. More advanced topics in this chapter include filesystem security, such as file ownership, permissions, and access control lists.
- Chapter 5, "Working with Text Files," includes everything from basic text editors to tools for finding files and searching for text within files.
- Chapter 6, "Managing Running Processes," describes how to see what processes are running on your system and change them. Ways of changing processes include killing, pausing, and sending other types of signals.
- Chapter 7, "Writing Simple Shell Scripts," includes shell commands and functions that you can gather together into a file to run as a command itself.

In Part III, "Becoming a Linux System Administrator," you learn how to administer Linux systems:

- Chapter 8, "Learning System Administration," provides information on basic graphical tools, commands, and configuration files for administering Linux systems. It introduces the Cockpit web UI for simplified, centralized Linux administration.
- Chapter 9, "Installing Linux," covers common installation tasks, such as disk partitioning and initial software package selection, as well as more advanced installation tools, such as installing from kickstart files.
- Chapter 10, "Getting and Managing Software," provides an understanding of how software packages work and how to get and manage software packages.
- Chapter 11, "Managing User Accounts," discusses tools for adding and deleting users and groups as well as how to centralize user account management.

- Chapter 12, "Managing Disks and Filesystems," provides information on adding partitions, creating filesystems, and mounting filesystems, as well as working with logical volume management.

In Part IV, "Becoming a Linux Server Administrator," you learn to create powerful network servers and the tools needed to manage them:

- Chapter 13, "Understanding Server Administration," covers remote logging, monitoring tools, and the Linux boot process.
- Chapter 14, "Administering Networking" discusses how to configure networking.
- Chapter 15, "Starting and Stopping Services," provides information on starting and stopping services.
- Chapter 16, "Configuring a Print Server," describes how to configure printers to use locally on your Linux system or over the network from other computers.
- Chapter 17, "Configuring a Web Server," describes how to configure an Apache web server.
- Chapter 18, "Configuring an FTP Server," covers procedures for setting up a vsftpd FTP server that can be used to enable others to download files from your Linux system over the network.
- Chapter 19, "Configuring a Windows File Sharing (Samba) Server," covers Windows file server configuration with Samba.
- Chapter 20, "Configuring an NFS File Server," describes how to use Network File System features to share folders of files among systems over a network.
- Chapter 21, "Troubleshooting Linux," covers popular tools for troubleshooting your Linux system.

In Part V, "Learning Linux Security Techniques," you learn how to secure your Linux systems and services:

- Chapter 22, "Understanding Basic Linux Security," covers basic security concepts and techniques.
- Chapter 23, "Understanding Advanced Linux Security," provides information on using Pluggable Authentication Modules (PAM) and cryptology tools to tighten system security and authentication.
- Chapter 24, "Enhancing Linux Security with SELinux," shows you how to enable Security Enhanced Linux (SELinux) to secure system services.
- Chapter 25, "Securing Linux on a Network," covers network security features, such as `firewalld` and `iptables` firewalls, to secure system services.

In Part VI," Engaging with Cloud Computing" the book pivots from a single-system focus toward containerization, cloud computing, and automation:

- Chapter 26, "Shifting to Clouds and Containers," describes how to pull, push, start, stop, tag, and build container images.

- Chapter 27, "Using Linux for Cloud Computing," introduces concepts of cloud computing in Linux by describing how to set up hypervisors, build virtual machines, and share resources across networks.
- Chapter 28, "Deploying Linux to the Cloud," describes how to deploy Linux images to different cloud environments, including OpenStack, Amazon EC2, or a local Linux system that is configured for virtualization.
- Chapter 29, "Automating Apps and Infrastructure with Ansible," tells you how to create Ansible playbooks and run ad-hoc Ansible commands to automate the configuration of Linux systems and other devices.
- Chapter 30, "Deploying Applications as Containers with Kubernetes," describes the Kubernetes project and how it is used to orchestrate container images, with the potential to massively scale up for large data centers.

Part VII contains two appendixes to help you get the most from your exploration of Linux. Appendix A, "Media," provides guidance on downloading Linux distributions. Appendix B, "Exercise Answers," provides sample solutions to the exercises included in Chapters 2 through 30.

Conventions Used in This Book

Throughout the book, special typography indicates code and commands. Commands and code are shown in a monospaced font:

```
This is how code looks.
```

In the event that an example includes both input and output, the monospaced font is still used, but input is presented in bold type to distinguish the two. Here's an example:

```
$ ftp ftp.handsonhistory.com
Name (home:jake): jake
Password: ******
```

As for styles in the text:

- New terms and important words appear in *italic* when introduced.
- Keyboard strokes appear like this: Ctrl+A. This convention indicates to hold the Ctrl key as you also press the "a" key.
- Filenames, URLs, and code within the text appear as follows: `persistence.properties`.

The following items call your attention to points that are particularly important.

NOTE
A Note box provides extra information to which you need to pay special attention.

Tip
A Tip box shows a special way of performing a particular task.

Caution
A Caution box alerts you to take special care when executing a procedure or damage to your computer hardware or software could result.

Jumping into Linux

If you are new to Linux, you might have vague ideas about what it is and where it came from. You may have heard something about it being free (as in cost) or free (as in freedom to use it as you please). Before you start putting your hands on Linux (which we will do soon enough), Chapter 1 seeks to answer some of your questions about the origins and features of Linux.

Take your time and work through this book to get up to speed on Linux and how you can make it work to meet your needs. This is your invitation to jump in and take the first step toward becoming a Linux expert!

Visit the *Linux Bible* website

To find links to various Linux distributions, tips on gaining Linux certification, and corrections to the book as they become available, go to `www.wiley.com/go/linuxbible10e`.

How to Contact Wiley or the Author

You can contact Christopher Negus at `striker57@gmail.com`.

If you believe you have found an error in this book, and it is not listed on the book's page at `www.wiley.com`, you can report the issue to our customer technical support team at `support.wiley.com`.

Part I

Getting Started

IN THIS PART

Chapter 1
Starting with Linux

Chapter 2
Creating the Perfect Linux Desktop

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER 1

Starting with Linux

IN THIS CHAPTER

Learning what Linux is

Learning where Linux came from

Choosing Linux distributions

Exploring professional opportunities with Linux

Becoming certified in Linux

The operating systems war is over, and Linux has won. Proprietary operating systems simply cannot keep up with the pace of improvements and quality that Linux can achieve with its culture of sharing and innovation. Even Microsoft, whose former CEO Steve Ballmer once referred to Linux as "a cancer," now says that Linux's use on its Microsoft's Azure cloud computing service has surpassed the use of Windows.

Linux is one of the most important technological advancements of the twenty-first century. Beyond its impact on the growth of the Internet and its place as an enabling technology for a range of computer-driven devices, Linux development has become a model for how collaborative projects can surpass what single individuals and companies can do alone.

Google runs thousands upon thousands of Linux servers to power its search technology. Its Android phones are based on Linux. Likewise, when you download and run Google's Chrome OS, you get a browser that is backed by a Linux operating system.

Facebook builds and deploys its site using what is referred to as a *LAMP stack* (Linux, Apache web server, MySQL database, and PHP web scripting language)—all open source projects. In fact, Facebook itself uses an open source development model, making source code for the applications and tools that drive Facebook available to the public. This model has helped Facebook shake out bugs quickly, get contributions from around the world, and fuel its exponential growth.

Financial organizations that have trillions of dollars riding on the speed and security of their operating systems also rely heavily on Linux. These include the New York Stock Exchange, Chicago Mercantile Exchange, and the Tokyo Stock Exchange.

As *cloud* continues to be one of the hottest buzzwords today, a part of the cloud groundswell that isn't hype is that Linux and other open source technologies continue to be the foundation on which

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

today's greatest cloud innovations are being built. Every software component that you need to build a private or public cloud (such as hypervisors, cloud controllers, network storage, virtual networking, and authentication) is freely available for you to start using from the open source world.

The widespread adoption of Linux around the world has created huge demand for Linux expertise. This chapter starts you down a path to becoming a Linux expert by helping you understand what Linux is, where it came from, and what your opportunities are for becoming proficient in it.

The rest of this book provides you with hands-on activities to help you gain that expertise. Finally, I show you how to apply that expertise to cloud technologies, including automation tools, such as Ansible, and containerization orchestration technologies, such as Kubernetes and OpenShift.

Understanding What Linux Is

Linux is a computer operating system. An operating system consists of the software that manages your computer and lets you run applications on it. The features that make up Linux and similar computer operating systems include the following:

Detecting and preparing hardware: When the Linux system boots up (when you turn on your computer), it looks at the components on your computer (CPU, hard drive, network cards, and so on) and loads the software (drivers and modules) needed to access those particular hardware devices.

Managing processes: The operating system must keep track of multiple processes running at the same time and decide which have access to the CPU and when. The system also must offer ways of starting, stopping, and changing the status of processes.

Managing memory: RAM and swap space (extended memory) must be allocated to applications as they need memory. The operating system decides how requests for memory are handled.

Providing user interfaces: An operating system must provide ways of accessing the system. The first Linux systems were accessed from a command-line interpreter called a *shell*. Today, graphical desktop interfaces are commonly available as well.

Controlling filesystems: Filesystem structures are built into the operating system (or loaded as modules). The operating system controls ownership and access to the files and directories (folders) that the filesystems contain.

Providing user access and authentication: Creating user accounts and allowing boundaries to be set between users is a basic feature of Linux. Separate user and group accounts enable users to control their own files and processes.

Offering administrative utilities: In Linux, hundreds (perhaps thousands) of commands and graphical windows are available to do such things as add users, manage

disks, monitor the network, install software, and generally secure and manage your computer. Web UI tools, such as Cockpit, have lowered the bar for doing complex administrative tasks.

Starting up services: To use printers, handle log messages, and provide a variety of system and network services, processes called *daemon processes* run in the background, waiting for requests to come in. Many types of services run in Linux. Linux provides different ways of starting and stopping these services. In other words, while Linux includes web browsers to view web pages, it can also be the computer that serves up web pages to others. Popular server features include web, mail, database, printer, file, DNS, and DHCP servers.

Programming tools: A wide variety of programming utilities for creating applications and libraries for implementing specialty interfaces are available with Linux.

As someone managing Linux systems, you need to learn how to work with those features just described. While many features can be managed using graphical interfaces, an understanding of the shell command line is critical for someone administering Linux systems.

Modern Linux systems now go way beyond what the first UNIX systems (on which Linux was based) could do. Advanced features in Linux, often used in large enterprises, include the following:

Clustering: Linux can be configured to work in clusters so that multiple systems can appear as one system to the outside world. Services can be configured to pass back and forth between cluster nodes while appearing to those using the services that they are running without interruption.

Virtualization: To manage computing resources more efficiently, Linux can run as a virtualization host. On that host, you could run other Linux systems, Microsoft Windows, BSD, or other operating systems as virtual guests. To the outside world, each of those virtual guests appears as a separate computer. KVM and Xen are two technologies in Linux for creating virtual hosts.

Cloud computing: To manage large-scale virtualization environments, you can use full-blown cloud computing platforms based on Linux. Projects such as OpenStack and Red Hat Virtualization (and its upstream oVirt project) can simultaneously manage many virtualization hosts, virtual networks, user and system authentication, virtual guests, and networked storage. Projects such as Kubernetes can manage containerized applications across massive data centers.

Real-time computing: Linux can be configured for real-time computing, where high-priority processes can expect fast, predictable attention.

Specialized storage: Instead of just storing data on the computer's hard disk, you can store it on many specialized local and networked storage interfaces that are available in Linux. Shared storage devices available in Linux include iSCSI, Fibre Channel, and Infiniband. Entire open source storage platforms include projects such as Ceph (`https://ceph.io`) and GlusterFS (`https://www.gluster.org`).

Some of these advanced topics are not covered in this book. However, the features covered here for using the shell, working with disks, starting and stopping services, and configuring a variety of servers should serve as a foundation for working with those advanced features.

Understanding How Linux Differs from Other Operating Systems

If you are new to Linux, chances are good that you have used a Microsoft Windows or MacOS operating system. Although MacOS had its roots in a free software operating system, referred to as the Berkeley Software Distribution (more on that later), operating systems from both Microsoft and Apple are considered proprietary operating systems. What that means is the following:

- You cannot see the code used to create the operating system, and therefore, you cannot change the operating system at its most basic levels if it doesn't suit your needs, and you can't use the operating system to build your own operating system from source code.
- You cannot check the code to find bugs, explore security vulnerabilities, or simply learn what that code is doing.
- You may not be able to plug your own software easily into the operating system if the creators of that system don't want to expose the programming interfaces you need to the outside world.

You might look at those statements about proprietary software and say, "What do I care? I'm not a software developer. I don't want to see or change how my operating system is built."

That may be true. However, the fact that others can take free and open source software and use it as they please has driven the explosive growth of the Internet (think Google), mobile phones (think Android), special computing devices (think TiVo), and hundreds of technology companies. Free software has driven down computing costs and allowed for an explosion of innovation.

Maybe you don't want to use Linux—as Google, Facebook, and other companies have done—to build the foundation for a multi-billion-dollar company. Nonetheless, those companies and others who now rely on Linux to drive their computer infrastructures need more and more people with the skills to run those systems.

You may wonder how a computer system that is so powerful and flexible has come to be free as well. To understand how that could be, you need to see where Linux came from. Thus the next sections of this chapter describe the strange and winding path of the free software movement that led to Linux.

Exploring Linux History

Some histories of Linux begin with the following message entitled "What would you like to see most in minix?" posted by Linus Torvalds to the `comp.os.minix` newsgroup on August 25, 1991, at

`https://groups.google.com/forum/#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ`

> Linus Benedict Torvalds
>
> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons, among other things). . .Any suggestions are welcome, but I won't promise I'll implement them :-)
>
> Linus (`torvalds@kruuna.helsinki.fi`)
>
> PS. Yes — it's free of any minix code, and it has a multi-threaded fs. It is NOT protable[sic] (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Minix was a UNIX-like operating system that ran on PCs in the early 1990s. Like Minix, Linux was also a clone of the UNIX operating system. With few exceptions, such as Microsoft Windows, most modern computer systems (including MacOS and Linux itself) were derived from UNIX operating systems, created originally by AT&T.

To truly appreciate how a free operating system could have been modeled after a proprietary system from AT&T Bell Laboratories, it helps to understand the culture in which UNIX was created and the chain of events that made the essence of UNIX possible to reproduce freely.

Note

To learn more about how Linux was created, pick up the book *Just for Fun: The Story of an Accidental Revolutionary* by Linus Torvalds (Harper Collins Publishing, 2001).

Free-flowing UNIX culture at Bell Labs

From the very beginning, the UNIX operating system was created and nurtured in a communal environment. Its creation was not driven by market needs but by a desire to overcome impediments to producing programs. AT&T, which owned the UNIX trademark originally, eventually made UNIX into a commercial product. By that time, however, many of the concepts (and even much of the early code) that made UNIX special had fallen into the public domain.

If you are not old enough to remember when AT&T split up in 1984, you may not remember a time when AT&T was *the* phone company. Up until the early 1980s, AT&T didn't have to think much about competition because if you wanted a phone in the United States, you had to go to AT&T. It had the luxury of funding pure research projects. The mecca for such projects was the Bell Laboratories site in Murray Hill, New Jersey.

After a project called Multics failed around 1969, Bell Labs employees Ken Thompson and Dennis Ritchie set off on their own to create an operating system that would offer an improved environment for developing software. Up to that time, most programs were written on paper punch cards that had to be fed in batches to mainframe computers. In a 1980 lecture on "The Evolution of the UNIX Time-sharing System," Dennis Ritchie summed up the spirit that started UNIX:

> What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing as supplied by remote-access, time-shared machines is not just to type programs into a terminal instead of a keypunch, but to encourage close communication.

The simplicity and power of the UNIX design began breaking down barriers that, until this point, had impeded software developers. The foundation of UNIX was set with several key elements:

The UNIX filesystem: Because it included a structure that allowed levels of subdirectories (which, for today's desktop users, look like folders inside of folders), UNIX could be used to organize the files and directories in intuitive ways. Furthermore, complex methods of accessing disks, tapes, and other devices were greatly simplified by representing those devices as individual device files that you could also access as items in a directory.

Input/output redirection: Early UNIX systems also included input redirection and pipes. From a command line, UNIX users could direct the output of a command to a file using a right-arrow key (`>`). Later, the concept of pipes (`|`) was added where the output of one command could be directed to the input of another command. For example, the following command line concatenates (`cat`) file1 and file2, sorts (`sort`) the lines in those files alphabetically, paginates the sorted text for printing (`pr`), and directs the output to the computer's default printer (`lpr`):

```
$ cat file1 file2 | sort | pr | lpr
```

This method of directing input and output enabled developers to create their own specialized utilities that could be joined with existing utilities. This modularity made it possible for lots of code to be developed by lots of different people. A user could just put together the pieces they needed.

Portability: Simplifying the experience of using UNIX also led to it becoming extraordinarily portable to run on different computer hardware. By having device drivers (represented by files in the filesystem tree), UNIX could present an interface to applications in such a way that the programs didn't have to know about the details

> of the underlying hardware. To port UNIX later to another system, developers had only to change the drivers. The application programs didn't have to change for different hardware!

To make portability a reality, however, a high-level programming language was needed to implement the software needed. To that end, Brian Kernighan and Dennis Ritchie created the C programming language. In 1973, UNIX was rewritten in C. Today, C is still the primary language used to create the UNIX (and Linux) operating system kernels.

As Ritchie went on to say in a 1979 lecture (`https://www.bell-labs.com/usr/dmr/www/hist.html`):

> Today, the only important UNIX program still written in assembler is the assembler itself; virtually all the utility programs are in C, and so are most of the application's programs, although there are sites with many in Fortran, Pascal, and Algol 68 as well. It seems certain that much of the success of UNIX follows from the readability, modifiability, and portability of its software that in turn follows from its expression in high-level languages.

If you are a Linux enthusiast and are interested in what features from the early days of Linux have survived, an interesting read is Dennis Ritchie's reprint of the first UNIX programmer's manual (dated November 3, 1971). You can find it at Dennis Ritchie's website: `https://www.bell-labs.com/usr/dmr/www/1stEdman.html`. The form of this documentation is UNIX man pages, which is still the primary format for documenting UNIX and Linux operating system commands and programming tools today.

What's clear as you read through the early documentation and accounts of the UNIX system is that the development was a free-flowing process, lacked ego, and was dedicated to making UNIX excellent. This process led to a sharing of code (both inside and outside of Bell Labs), which allowed rapid development of a high-quality UNIX operating system. It also led to an operating system that AT&T would find difficult to reel back in later.

Commercial UNIX

Before the AT&T divestiture in 1984, when it was split up into AT&T and seven "Baby Bell" companies, AT&T was forbidden to sell computer systems. Companies that would later become Verizon, Qwest, Nokia, and Alcatel-Lucent were all part of AT&T. As a result of AT&T's monopoly of the telephone system, the US government was concerned that an unrestricted AT&T might dominate the fledgling computer industry.

Because AT&T was restricted from selling computers directly to customers before its divestiture, UNIX source code was licensed to universities for a nominal fee. This allowed UNIX installations to grow in size and mindshare among top universities. However, there was still no UNIX operating system for sale from AT&T that you didn't have to compile yourself.

Berkeley Software Distribution arrives

In 1975, UNIX V6 became the first version of UNIX available for widespread use outside of Bell Laboratories. From this early UNIX source code, the first major variant of UNIX was

created at University of California, Berkeley. It was named the Berkeley Software Distribution (BSD).

For most of the next decade, the BSD and Bell Labs versions of UNIX headed off in separate directions. BSD continued forward in the free-flowing, share-the-code manner that was the hallmark of the early Bell Labs UNIX, whereas AT&T started steering UNIX toward commercialization. With the formation of a separate UNIX Laboratory, which moved out of Murray Hill and down the road to Summit, New Jersey, AT&T began its attempts to commercialize UNIX. By 1984, divestiture was behind AT&T and it was really ready to start selling UNIX.

UNIX Laboratory and commercialization

The UNIX Laboratory was considered a jewel that couldn't quite find a home or a way to make a profit. As it moved between Bell Laboratories and other areas of AT&T, its name changed several times. It is probably best remembered by the name it had as it began its spin-off from AT&T: UNIX System Laboratories (USL).

The UNIX source code that came out of USL, the legacy of which was sold in part to Santa Cruz Operation (SCO), was used for a time as the basis for ever-dwindling lawsuits by SCO against major Linux vendors (such as IBM and Red Hat, Inc.). Because of that, I think the efforts from USL that have contributed to the success of Linux are lost on most people.

During the 1980s, of course, many computer companies were afraid that a newly divested AT&T would pose more of a threat to controlling the computer industry than would an upstart company in Redmond, Washington. To calm the fears of IBM, Intel, Digital Equipment Corporation, and other computer companies, the UNIX Lab made the following commitments to ensure a level playing field:

Source code only: Instead of producing its own boxed set of UNIX, AT&T continued to sell source code only and to make it available equally to all licensees. Each company would then port UNIX to its own equipment. It wasn't until about 1992, when the lab was spun off as a joint venture with Novell (called Univel), and then eventually sold to Novell, that a commercial boxed set of UNIX (called UnixWare) was produced directly from that source code.

Published interfaces: To create an environment of fairness and community to its OEMs (original equipment manufacturers), AT&T began standardizing what different ports of UNIX had to be able to do to still be called UNIX. To that end, Portable Operating System Interface (POSIX) standards and the AT&T UNIX System V Interface Definition (SVID) were specifications UNIX vendors could use to create compliant UNIX systems. Those same documents also served as road maps for the creation of Linux.

Note

In an early email newsgroup post, Linus Torvalds made a request for a copy, preferably online, of the POSIX standard. I think that no one from AT&T expected someone actually to be able to write their own clone of UNIX from those interfaces without using any of its UNIX source code.

Technical approach: Again, until the very end of USL, most decisions on the direction of UNIX were made based on technical considerations. Management was promoted up through the technical ranks, and to my knowledge there was never any talk of writing software to break other companies' software or otherwise restrict the success of USL's partners.

When USL eventually started taking on marketing experts and creating a desktop UNIX product for end users, Microsoft Windows already had a firm grasp on the desktop market. Also, because the direction of UNIX had always been toward source-code licensing destined for large computing systems, USL had pricing difficulties for its products. For example, on software that it was including with UNIX, USL found itself having to pay out per-computer licensing fees that were based on $100,000 mainframes instead of $2,000 PCs. Add to that the fact that no application programs were available with UnixWare and you can see why the endeavor failed.

Successful marketing of UNIX systems at the time, however, was happening with other computer companies. SCO had found a niche market, primarily selling PC versions of UNIX running dumb terminals in small offices. Sun Microsystems was selling lots of UNIX workstations (originally based on BSD but merged with UNIX in SVR4) for programmers and high-end technology applications (such as stock trading).

Other commercial UNIX systems were also emerging by the 1980s. This new ownership assertion of UNIX was beginning to take its toll on the spirit of open contributions. Lawsuits were being initiated to protect UNIX source code and trademarks. In 1984, this new, restrictive UNIX gave rise to an organization that eventually led the path to Linux: the Free Software Foundation.

GNU transitions UNIX to freedom

In 1984, Richard M. Stallman started the GNU project (`https://gnu.org`), recursively named by the phrase GNU is Not UNIX. As a project of the Free Software Foundation (FSF), GNU was intended to become a recoding of the entire UNIX operating system that could be freely distributed.

The GNU Project page (`https://gnu.org/gnu/thegnuproject.html`) tells the story of how the project came about in Stallman's own words. It also lays out the problems that proprietary software companies were imposing on those software developers who wanted to share, create, and innovate.

Although rewriting millions of lines of code might seem daunting for one or two people, spreading the effort across dozens or even hundreds of programmers made the project possible. Remember that UNIX was designed to be built in separate pieces that could be piped together. Because they were reproducing commands and utilities with well-known, published interfaces, that effort could easily be split among many developers.

It turned out that not only could the same results be gained by all new code, but in some cases that code was better than the original UNIX versions. Because everyone could see

the code being produced for the project, poorly written code could be corrected quickly or replaced over time.

If you are familiar with UNIX, try searching the hundreds of GNU software packages, which contain thousands of commands, for your favorite UNIX command from the Free Software Directory (`https://directory.fsf.org/wiki/GNU`). Chances are good that you will find it there, along with many, many other available software projects.

Over time, the term *free software* has been mostly replaced by the term *open source software*. The term *free software* is preferred by the Free Software Foundation, while *open source software* is promoted by the Open Source Initiative (`https://opensource.org`).

To accommodate both camps, some people use the term *Free and Open Source Software (FOSS)* instead. An underlying principle of FOSS, however, is that although you are free to use the software as you like, you have some responsibility to make the improvements that you make to the code available to others. This way, everyone in the community can benefit from your work, as you have benefited from the work of others.

To define clearly how open source software should be handled, the GNU software project created the GNU Public License, or GPL. Although many other software licenses cover slightly different approaches to protecting free software, the GPL is the most well known—and it's the one that covers the Linux kernel itself. The GNU Public License includes the following basic features:

Author rights: The original author retains the rights to their software.

Free distribution: People can use the GNU software in their own software, changing and redistributing it as they please. They do, however, have to include the source code with their distribution (or make it easily available).

Copyright maintained: Even if you were to repackage and resell the software, the original GNU agreement must be maintained with the software, which means that all future recipients of the software have the opportunity to change the source code, just as you did.

There is no warranty on GNU software. If something goes wrong, the original developer of the software has no obligation to fix the problem. However, many organizations, large and small, offer paid support (often in subscription form) for the software when it is included in their Linux or other open source software distribution. (See the section "OSI open source definition" later in this chapter for a more detailed definition of open source software.)

Despite its success in producing thousands of UNIX utilities, the GNU project itself failed to produce one critical piece of code: the kernel. Its attempts to build an open source kernel with the GNU Hurd project (`https://gnu.org/software/hurd/`) were unsuccessful at first, so it failed to become the premier open source kernel.

BSD loses some steam

The one software project that had a chance of beating out Linux to be the premier open source kernel was the venerable BSD project. By the late 1980s, BSD developers at

University of California (UC), Berkeley realized that they had already rewritten most of the UNIX source code they had received a decade earlier.

In 1989, UC Berkeley distributed its own UNIX-like code as Net/1 and later (in 1991) as Net/2. Just as UC Berkeley was preparing a complete, UNIX-like operating system that was free from all AT&T code, AT&T hit them with a lawsuit in 1992. The suit claimed that the software was written using trade secrets taken from AT&T's UNIX system.

It's important to note here that BSD developers had completely rewritten the copyright-protected code from AT&T. Copyright was the primary means AT&T used to protect its rights to the UNIX code. Some believe that if AT&T had patented the concepts covered in that code, there might not be a Linux (or any UNIX clone) operating system today.

The lawsuit was dropped when Novell bought UNIX System Laboratories from AT&T in 1994. Nevertheless, during that critical period there was enough fear and doubt about the legality of the BSD code that the momentum that BSD had gained to that point in the fledgling open source community was lost. Many people started looking for another open source alternative. The time was ripe for a college student from Finland who was working on his own kernel.

Note

Today, BSD versions are available from three major projects: FreeBSD, NetBSD, and OpenBSD. People generally characterize FreeBSD as the easiest to use, NetBSD as available on the most computer hardware platforms, and OpenBSD as fanatically secure. Many security-minded individuals still prefer BSD to Linux. Also, because of its licensing, BSD code can be used by proprietary software vendors, such as Microsoft and Apple, who don't want to share their operating system code with others. MacOS is built on a BSD derivative.

Linus builds the missing piece

Linus Torvalds started work on Linux in 1991, while he was a student at the University of Helsinki, Finland. He wanted to create a UNIX-like kernel so that he could use the same kind of operating system on his home PC that he used at school. At the time, Linus was using Minix, but he wanted to go beyond what the Minix standards permitted.

As noted earlier, Linus announced the first public version of the Linux kernel to the `comp.os.minix` newsgroup on August 25, 1991, although Torvalds guesses that the first version didn't actually come out until mid-September of that year.

Although Torvalds stated that Linux was written for the 386 processor and probably wasn't portable, others persisted in encouraging (and contributing to) a more portable approach in the early versions of Linux. By October 5, 1991, Linux 0.02 was released with much of the original assembly code rewritten in the C programming language, which made it possible to start porting it to other machines.

The Linux kernel was the last—and the most important—piece of code that was needed to complete a whole UNIX-like operating system under the GPL. So when people started

putting together distributions, the name Linux and not GNU is what stuck. Some distributions, such as Debian, however, refer to themselves as GNU/Linux distributions. (Not including GNU in the title or subtitle of a Linux operating system is also a matter of much public grumbling by some members of the GNU project. See `https://gnu.org`.)

Today, Linux can be described as an open source UNIX-like operating system that reflects a combination of SVID, POSIX, and BSD compliance. Linux continues to aim toward compliance with POSIX as well as with standards set by the owner of the UNIX trademark, The Open Group (`https://opengroup.org`).

The nonprofit Open Source Development Labs, renamed the Linux Foundation after merging with the Free Standards Group (`https://linuxfoundation.org`), which employs Linus Torvalds, manages the direction today of Linux development efforts. Its sponsors list is like a Who's Who of commercial Linux system and application vendors, including IBM, Red Hat, SUSE, Oracle, HP, Dell, Computer Associates, Intel, Cisco Systems, and hundreds of others. The Linux Foundation's primary charter is to protect and accelerate the growth of Linux by providing legal protection and software development standards for Linux developers.

Although much of the thrust of corporate Linux efforts is on enterprise computing, huge improvements are continuing in the desktop arena as well. The KDE and GNOME desktop environments continuously improve the Linux experience for casual users. Newer lightweight desktop environments such as Chrome OS, Xfce, and LXDE now offer efficient alternatives that today bring Linux to thousands of netbook owners.

Linus Torvalds continues to maintain and improve the Linux kernel.

Note

For a more detailed history of Linux, see the book *Open Sources: Voices from the Open Source Revolution* (O'Reilly, 1999). The entire first edition is available online at

`https://oreilly.com/openbook/opensources/book/`

OSI open source definition

Linux provides a platform that lets software developers change the operating system as they like and get a wide range of help creating the applications they need. One of the watchdogs of the open source movement is the *Open Source Initiative, or OSI* (`https://opensource.org`).

Although the primary goal of open source software is to make source code available, other goals of open source software are also defined by OSI in its open source definition. Most of the following rules for acceptable open source licenses serve to protect the freedom and integrity of the open source code:

Free distribution: An open source license can't require a fee from anyone who resells the software.

Source code: The source code must be included with the software, and there can be no restrictions on redistribution.

Derived works: The license must allow modification and redistribution of the code under the same terms.

Integrity of the author's source code: The license may require that those who use the source code remove the original project's name or version if they change the source code.

No discrimination against persons or groups: The license must allow all people to be equally eligible to use the source code.

No discrimination against fields of endeavor: The license can't restrict a project from using the source code because it is commercial, or because it is associated with a field of endeavor that the software provider doesn't like.

Distribution of license: No additional license should be needed to use and redistribute the software.

License must not be specific to a product: The license can't restrict the source code to a particular software distribution.

License must not restrict other software: The license can't prevent someone from including the open source software on the same medium as non-open source software.

License must be technology neutral: The license can't restrict methods in which the source code can be redistributed.

Open source licenses used by software development projects must meet these criteria to be accepted as open source software by OSI. About 70 different licenses are accepted by OSI to be used to label software as "OSI Certified Open Source Software." In addition to the GPL, other popular OSI-approved licenses include the following:

LGPL: The GNU Lesser General Public License (LGPL) is often used for distributing libraries that other application programs depend upon.

BSD: The Berkeley Software Distribution License allows redistribution of source code, with the requirement that the source code keep the BSD copyright notice and not use the names of contributors to endorse or promote derived software without written permission. A major difference from GPL, however, is that BSD does not require people modifying the code to pass those changes on to the community. As a result, proprietary software vendors such as Apple and Microsoft have used BSD code in their own operating systems.

MIT: The MIT license is like the BSD license, except that it doesn't include the endorsement and promotion requirement.

Mozilla: The Mozilla license covers the use and redistribution of source code associated with the Firefox web browser and other software related to the Mozilla project

(https://www.mozilla.org/en-US/). It is a much longer license than the others just mentioned because it contains more definitions of how contributors and those reusing the source code should behave. This includes submitting a file of changes when submitting modifications and that those making their own additions to the code for redistribution should be aware of patent issues or other restrictions associated with their code.

The end result of open source code is software that has more flexibility to grow and fewer boundaries in how it can be used. Many believe that the fact that numerous people look over the source code for a project results in higher-quality software for everyone. As open source advocate Eric S. Raymond says in an often-quoted line, "Given enough eyeballs, all bugs are shallow."

Understanding How Linux Distributions Emerged

Having bundles of source code floating around the Internet that could be compiled and packaged into a Linux system worked well for geeks. More casual Linux users, however, needed a simpler way to put together a Linux system. To respond to that need, some of the best geeks began building their own Linux distributions.

A *Linux distribution* consists of the components needed to create a working Linux system and the procedures needed to get those components installed and running. Technically, Linux is really just what is referred to as the *kernel*. Before the kernel can be useful, you must have other software, such as basic commands (GNU utilities), services that you want to offer (such as remote login or web servers), and possibly a desktop interface and graphical applications. Then you must be able to gather all that together and install it on your computer's hard disk.

Slackware (http://www.slackware.com) is one of the oldest Linux distributions still supported today. It made Linux friendly for less technical users by distributing software already compiled and grouped into packages. (Those packages of software components were in a format called *tarballs*.) Then you would use basic Linux commands to do things like format your disk, enable swap, and create user accounts.

Before long, many other Linux distributions were created. Some Linux distributions were created to meet special needs, such as KNOPPIX (a live CD Linux), Gentoo (a cool customizable Linux), and Mandrake (later called Mandriva, which was one of several desktop Linux distributions). But two major distributions rose to become the foundation for many other distributions: Red Hat Linux and Debian.

Choosing a Red Hat distribution

When Red Hat Linux appeared in the late 1990s, it quickly became the most popular Linux distribution for several reasons:

RPM package management: Tarballs are fine for dropping software on your computer, but they don't work as well when you want to update, remove, or even find out

about that software. Red Hat created the RPM packaging format so that a software package could contain not only the files to be shared but also information about the package version, who created it, which files were documentation or configuration files, and when it was created. By installing software packaged in RPM format, you could store that information about each software package in a local RPM database. It became easy to find what was installed, update it, or remove it.

Simple installation: The Anaconda installer made it much simpler to install Linux. As a user, you could step through some simple questions, in most cases accepting defaults, to install Red Hat Linux.

Graphical administration: Red Hat added simple graphical tools to configure printers, add users, set time and date, and do other basic administrative tasks. As a result, desktop users could use a Linux system without even having to run commands.

For years, Red Hat Linux was the preferred Linux distribution for both Linux professionals and enthusiasts. Red Hat, Inc., gave away the source code, as well as the compiled, ready-to-run versions of Red Hat Linux (referred to as the binaries). But as the needs of its Linux community users and big-ticket customers began to move further apart, Red Hat abandoned Red Hat Linux and began developing two operating systems instead: Red Hat Enterprise Linux and Fedora.

Using Red Hat Enterprise Linux

In March 2012, Red Hat, Inc., became the first open source software company to bring in more than $1 billion in yearly revenue. It achieved that goal by building a set of products around Red Hat Enterprise Linux (RHEL) that would suit the needs of the most demanding enterprise computing environments. After expanding its product line to include many components of hybrid cloud computing, Red Hat was purchased by IBM in July 2019 for $34 billion.

While other Linux distributions focused on desktop systems or small business computing, RHEL worked on those features needed to handle mission-critical applications for big business and government. It built systems that could speed transactions for the world's largest financial exchanges and be deployed as clusters and virtual hosts.

Instead of just selling RHEL, Red Hat offers an ecosystem of benefits upon which Linux customers could draw. To use RHEL, customers buy subscriptions that they can use to deploy any version of RHEL that they desire. If they decommission a RHEL system, they can use the subscription to deploy another system.

Different levels of support are available for RHEL, depending on customer needs. Customers can be assured that, along with support, they can get hardware and third-party software that is certified to work with RHEL. They can get Red Hat consultants and engineers to help them put together the computing environments they need. They can also get training and certification exams for their employees (see the discussion of RHCE certification later in this chapter).

Red Hat has also added other products as natural extensions to Red Hat Enterprise Linux. JBoss is a middleware product for deploying Java-based applications to the Internet or company intranets. Red Hat Virtualization comprises the virtualization hosts, managers, and guest computers that allow you to install, run, manage, migrate, and decommission huge virtual computing environments.

In recent years, Red Hat has extended its portfolio into cloud computing. Red Hat OpenStack Platform and Red Hat Virtualization offer complete platforms for running and managing virtual machines. However, the technology with the biggest impact in recent years is *Red Hat OpenShift*, which provides a hybrid cloud suite of software that has Kubernetes, the most popular container orchestration platform project, as its foundation. With the Red Hat acquisition, IBM has set a goal to containerize most of its applications to run on OpenShift.

There are those who have tried to clone RHEL, using the freely available RHEL source code, rebuilding and rebranding it. Oracle Linux is built from source code for RHEL but currently offers an incompatible kernel. CentOS is a community-sponsored Linux distribution that is built from RHEL source code. Recently, Red Hat took over support of the CentOS project.

I've chosen to use Red Hat Enterprise Linux for many of the examples in this book because, if you want a career working on Linux systems, there is a huge demand for those who can administer RHEL systems. If you are starting out with Linux, however, Fedora can provide an excellent entry point to the same skills that you need to use and administer RHEL systems.

Using Fedora

While RHEL is the commercial, stable, supported Linux distribution, Fedora is the free, cutting-edge Linux distribution that is sponsored by Red Hat, Inc. Fedora is the Linux system that Red Hat uses to engage the Linux development community and encourage those who want a free Linux for personal use and rapid development.

Fedora includes tens of thousands of software packages, many of which keep up with the latest available open source technology. As a user, you can try the latest Linux desktop, server, and administrative interfaces in Fedora for free. As a software developer, you can create and test your applications using the latest Linux kernel and development tools.

Because the focus of Fedora is on the latest technology, it focuses less on stability. So, expect that you might need to do some extra work to get everything working and that not all the software will be fully baked.

I recommend that you use Fedora or RHEL for most of the examples in this book for the following reasons:

- Fedora is used as a proving ground for Red Hat Enterprise Linux. Red Hat tests many new applications in Fedora before committing them to RHEL. By using Fedora, you will learn the skills you need to work with features as they are being developed for Red Hat Enterprise Linux.

- For learning, Fedora is more convenient than RHEL, yet still includes many of the more advanced, enterprise-ready tools that are in RHEL.
- Fedora is free, not only as in "freedom," but also as in "you don't have to pay for it."

Fedora is extremely popular with those who develop open source software. However, in the past few years, another Linux distribution has captured the attention of many people starting out with Linux: Ubuntu.

Choosing Ubuntu or another Debian distribution

Like Red Hat Linux, the Debian GNU/Linux distribution was an early Linux distribution that excelled at packaging and managing software. Debian uses the deb packaging format and tools to manage all of the software packages on its systems. Debian also has a reputation for stability.

Many Linux distributions can trace their roots back to Debian. According to DistroWatch (`https://distrowatch.com`), more than 130 active Linux distributions can be traced back to Debian. Popular Debian-based distributions include Linux Mint, elementary OS, Zorin OS, LXLE, Kali Linux, and many others. However, the Debian derivative that has achieved the most success is Ubuntu (`https://ubuntu.com`).

By relying on stable Debian software development and packaging, the Ubuntu Linux distribution (sponsored by Canonical Ltd.) was able to come along and add those features that Debian lacked. In pursuit of bringing new users to Linux, the Ubuntu project added a simple graphical installer and easy-to-use graphical tools. It also focused on full-featured desktop systems while still offering popular server packages.

Ubuntu was also an innovator in creating new ways to run Linux. Using live CDs or live USB drives offered by Ubuntu, you could have Ubuntu up and running in just a few minutes. Often included on live CDs were open source applications, such as web browsers and word processors, that actually ran in Windows. This made the transition to Linux from Windows easier for some people.

If you are using Ubuntu, don't fear. Most of subject matter covered in this book will work as well in Ubuntu as it does in Fedora or RHEL.

Finding Professional Opportunities with Linux Today

If you want to develop an idea for a computer-related research project or technology company, where do you begin? You begin with an idea. After that, you look for the tools that you need to explore and eventually create your vision. Then you look for others to help you during that creation process.

Today, the hard costs of starting a company like Google or Facebook include just a computer, a connection to the Internet, and enough caffeinated beverage of your choice to

keep you up all night writing code. If you have your own world-changing idea, Linux and thousands of software packages are available to help you build your dreams. The open source world also comes with communities of developers, administrators, and users who are available to help you.

If you want to get involved with an existing open source project, projects are always looking for people to write code, test software, or write documentation. In those projects, you will find people who use the software, work on that software, and are usually willing to share their expertise to help you as well.

Whether you seek to develop the next great open source software project, or you simply want to gain the skills needed to compete for the thousands of well-paying Linux administrator or development jobs, it will help you to know how to install, secure, and maintain Linux systems.

In March 2020, more than 60,000 jobs requiring Linux skills were listed at Indeed.com. Nearly half of those offered pay of $100,000 per year or more. Site such as Fossjobs.net provide a venue for posting and finding jobs associated with Linux and other free and open source software skills.

The message to take from these job sites is that Linux continues to grow and create demands for Linux expertise. Companies that have begun using Linux have continued to move forward with it. Those using Linux continue to expand its use and find that cost savings, security, and the flexibility it offers continue to make Linux a good investment.

Understanding how companies make money with Linux

Open source enthusiasts believe that better software can result from an open source software development model than from proprietary development models. So, in theory, any company creating software for its own use can save money by adding its software contributions to those of others to gain a much better end product for themselves.

Companies that want to make money by selling software need to be more creative than they were in the old days. Although you can sell the software you create, which includes GPL software, you must pass the source code of that software forward. Of course, others can then recompile that product, basically using and even reselling your product without charge. Here are a few ways that companies are dealing with that issue:

Software subscriptions: Red Hat, Inc., sells its Red Hat Enterprise Linux products on a subscription basis. For a certain amount of money per year, you get binary code to run Linux (so you don't have to compile it yourself), guaranteed support, tools for tracking the hardware and software on your computer, access to the company's knowledge base, and other assets.

Although Red Hat's Fedora project includes much of the same software and is also available in binary form, there are no guarantees associated with the software or

future updates of that software. A small office or personal user might take a risk on using Fedora (which is itself an excellent operating system), but a big company that's running mission-critical applications will probably put down a few dollars for RHEL.

Training and certification: With Linux system use growing in government and big business, professionals are needed to support those systems. Red Hat offers training courses and certification exams to help system administrators become proficient using Red Hat Enterprise Linux systems. In particular, the Red Hat Certified Engineer (RHCE) and Red Hat Certified System Administrator (RHCSA) certifications have become popular (`https://www.redhat.com/en/services/training-and-certification/why-get-certified`). More on RHCE/RHCSA certifications later in this chapter.

Other certification programs are offered by Linux Professional Institute (`https://www.lpi.org`) and CompTIA (`wwww..comptia.org/`). LPI and CompTIA are professional computer industry associations.

Bounties: Software bounties are a fascinating way for open source software companies to make money. Suppose that you are using XYZ software package and you need a new feature right away. By paying a software bounty to the project itself, or to other software developers, you can have your required improvements moved to the head of the queue. The software you pay for will remain covered by its open source license, but you will have the features you need—probably at a fraction of the cost of building the project from scratch.

Donations: Many open source projects accept donations from individuals or open source companies that use code from their projects. Amazingly, many open source projects support one or two developers and run exclusively on donations.

Boxed sets, mugs, and T-shirts: Some open source projects have online stores where you can buy boxed sets (some people still like physical DVDs and hard copies of documentation) and a variety of mugs, T-shirts, mouse pads, and other items. If you really love a project, for goodness sake, buy a T-shirt!

This is in no way an exhaustive list, because more creative ways are being invented every day to support those who create open source software. Remember that many people have become contributors to and maintainers of open source software because they needed or wanted the software themselves. The contributions they make for free are worth the return they get from others who do the same.

Becoming Red Hat certified

Although this book is not focused on becoming certified in Linux, it touches on the activities that you need to be able to master to pass popular Linux certification exams. In

particular, most of what is covered in the Red Hat Certified Engineer (RHCE) and Red Hat Certified System Administrator (RHCSA) exams for Red Hat Enterprise Linux 8 is described in this book.

If you are looking for a job as a Linux IT professional, RHCSA or RHCE certification is often listed as a requirement, or at least a preference, for employment. The RHCSA exam (EX200) provides basic certification, covering such topics as configuring disks and filesystems, adding users, setting up a simple web and FTP server, and adding swap space. The RHCE exam (EX300) tests for more advanced server configuration as well an advanced knowledge of security features, such as SELinux and firewalls.

Those of us who have taught RHCE/RHCSA courses and given exams (as I did for three years) are not allowed to tell you exactly what is on the exam. However, Red Hat gives an overview of how the exams work as well as a list of topics that you can expect to see covered in the exam. You can find those exam objectives on the following sites:

RHCSA

`https://redhat.com/en/services/training/ex200-red-hat-certified-system-administrator-rhcsa-exam`

RHCE

`https://redhat.com/en/services/training/ex294-red-hat-certified-engineer-rhce-exam-red-hat-enterprise-linux-8`

As the exam objectives state, the RHCSA and RHCE exams are performance based, which means that you are given tasks to do and you must perform those tasks on an actual Red Hat Enterprise Linux system, as you would on the job. You are graded on how well you obtained the results of those tasks.

If you plan to take the exams, check back to the exam objectives pages often because they change from time to time. Also keep in mind that the RHCSA is a standalone certification; however, you must pass the RHCSA and the RHCE exams to get an RHCE certification. Often, the two exams are given on the same day.

You can sign up for RHCSA and RHCE training and exams at `https://redhat.com/en/services/training-and-certification`. Training and exams are given at major cities all over the United States and around the world. The skills that you need to complete these exams are described in the following sections.

RHCSA topics

As noted earlier, RHCSA exam topics cover basic system administration skills. These are the current topics listed for Red Hat Enterprise Linux 8 at the RHCSA exam objectives site (again, check the exam objectives site in case they change) and where in this book you can learn about them:

- **Understand essential tools**: You are expected to have a working knowledge of the command shell (bash), including how to use proper command syntax and do input/

output redirection (< > >>). You need to know how to log in to remote and local systems. Expect to have to create, edit, move, copy, link, delete, and change permission and ownership on files. Likewise, you should know how to look up information on man pages and /usr/share/doc. Most of these topics are covered in Chapter 3 and Chapter 4 in this book. Chapter 5 describes how to edit and find files.

Operate running systems: In this category, you must understand the Linux boot process, and how to shut down, reboot, and change to different targets (previously called *runlevels*). You need to identify processes and kill processes as requested. You must be able to find and interpret log files. Chapter 15 describes how to change targets and manage system services. See Chapter 6 for information on managing and changing processes. Logging is described in Chapter 13.

Configure local storage: Setting up disk partitions includes creating physical volumes and configuring them to be used for Logical Volume Management (LVM) or encryption (LUKS). You should also be able to set up those partitions as filesystems or swap space that can be mounted or enabled at boot time. Disk partitioning and LVM are covered in Chapter 12. LUKS and other encryption topics are described in Chapter 23, "Understanding Advanced Linux Security."

Create and configure filesystems: Create and automatically mount different kinds of filesystems, including regular Linux filesystems (ext2, ext3, or ext4) and network filesystems (NFS). Create collaborative directories using the set group ID bit feature. You must also be able to use LVM to extend the size of a logical volume. Filesystem topics are covered in Chapter 12. See Chapter 20 for NFS coverage.

Deploy, configure, and maintain systems: This covers a range of topics, including configuring networking and creating cron tasks. For software packages, you must be able to install packages from Red Hat Content Delivery Network (CDN), a remote repository, or the local filesystem. The cron facility is described in Chapter 13.

Manage users and groups: You must know how to add, delete, and change user and group accounts. Another topic that you should know is password aging, using the chage command. See Chapter 11 for information on configuring users and groups.

Manage security: You must have a basic understanding of how to set up a firewall (firewalld, system-config-firewall, or iptables) and how to use SELinux. You must be able to set up SSH to do key-based authentication. Learn about SELinux in Chapter 24. Firewalls are covered in Chapter 25. Chapter 13 includes a description of key-based authentication.

Most of these topics are covered in this book. Refer to Red Hat documentation (https://access.redhat.com/documentation) under the Red Hat Enterprise Linux heading for descriptions of features not found in this book. In particular, the system administration guides contain descriptions of many of the RHCSA-related topics.

RHCE topics

RHCE exam topics cover more advanced server configuration, along with a variety of security features for securing those servers in Red Hat Enterprise Linux 8. Again, check the

RHCE exam objectives site for the most up-to-date information on topics that you should study for the exam.

System configuration and management

The system configuration and management requirement for the RHCE exam covers a range of topics, including the following:

Firewalls: Block or allow traffic to selected ports on your system that offer services such as web, FTP, and NFS, as well as block or allow access to services based on the originator's IP address. Firewalls are covered in Chapter 25, "Securing Linux on a Network."

Kerberos authentication: Use Kerberos to authenticate users on a RHEL system.

System reports: Use features such as `sar` to report on system use of memory, disk access, network traffic, and processor utilization. Chapter 13 describes how to use the `sar` command.

Shell scripting: Create a simple shell script to take input and produce output in various ways. Shell scripting is described in Chapter 7.

SELinux: With Security Enhanced Linux in Enforcing mode, make sure that all server configurations described in the next section are properly secured with SELinux. SELinux is described in Chapter 24.

Ansible: Understand core Ansible components (inventories, modules, playbooks, and so on). Be able to install and configure an Ansible control node. Work with Ansible roles and use advanced Ansible features. See Chapter 29 for information on using Ansible playbooks to install and manage Linux systems.

Installing and configuring network services

For each of the network services in the list that follows, make sure you can go through the steps to install packages required by the service, set up SELinux to allow access to the service, set the service to start at boot time, secure the service by host or by user (using iptables or features provided by the service itself), and configure it for basic operation. These are the services:

Web server: Configure an Apache (HTTP/HTTPS) server. You must be able to set up a virtual host, deploy a CGI script, use private directories, and allow a particular Linux group to manage the content. Chapter 17 describes how to configure a web server.

DNS server: Set up a DNS server (bind package) to act as a caching-only name server that can forward DNS queries to another DNS server. No need to configure master or slave zones. DNS is described from the client side in Chapter 14. For information on configuring a DNS server with Bind, see the RHEL Networking Guide at

`https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/networking_guide/index`

NFS server: Configure an NFS server to share specific directories to specific client systems so they can be used for group collaboration. Chapter 20 covers NFS.

Windows file sharing server: Set up Linux (Samba) to provide SMB shares to specific hosts and users. Configure the shares for group collaboration. See Chapter 19 to learn about configuring Samba.

Mail server: Configure postfix or sendmail to accept incoming mail from outside of the local host. Relay mail to a smart host. Mail server configuration is not covered in this book (and should not be done lightly). See the RHEL System Administrator's Guide for information on configuring mail servers at:

`https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/system_administrators_guide/index#ch-Mail_Servers`

Secure Shell server: Set up the SSH service (sshd) to allow remote login to your local system as well as key-based authentication. Otherwise, configure the `sshd.conf` file as needed. Chapter 13 describes how to configure the sshd service.

Network Time server: Configure a Network Time Protocol server (ntpd) to synchronize time with other NTP peers.

Database server: Configure the MariaDB database and manage it in various ways. Learn how to configure the MariaDB from the `MariaDB.org` site (`https://mariadb.com/kb/en/library/documentation/`).

Although there are other tasks in the RHCE exam, as just noted, keep in mind that most of the tasks have you configure servers and then secure those servers using any technique that you need. Those can include firewall rules (iptables), SELinux, or any features built into configuration files for the particular service.

Summary

Linux is an operating system that is built by a community of software developers around the world, which is led by its creator, Linus Torvalds. It is derived originally from the UNIX operating system but has grown beyond UNIX in popularity and power over the years.

The history of the Linux operating system can be tracked from early UNIX systems that were distributed free to colleges and improved upon by initiatives such as the Berkeley Software Distribution (BSD). The Free Software Foundation helped make many of the components needed to create a fully free UNIX-like operating system. The Linux kernel itself was the last major component needed to complete the job.

Most Linux software projects are protected by one of a set of licenses that fall under the Open Source Initiative umbrella. The most prominent of these is the GNU Public License (GPL). Standards such as the Linux Standard Base and world-class Linux organizations and

companies (such as Canonical Ltd. and Red Hat, Inc.) make it possible for Linux to continue to be a stable, productive operating system into the future.

Learning the basics of how to use and administer a Linux system will serve you well in any aspect of working with Linux. The remaining chapters provide a series of exercises with which you can test your understanding. That's why, for the rest of the book, you will learn best with a Linux system in front of you so that you can work through the examples in each chapter and complete the exercises successfully.

The next chapter explains how to get started with Linux by describing how to get and use a Linux desktop system.

CHAPTER 2

Creating the Perfect Linux Desktop

IN THIS CHAPTER

Understanding the X Window System and desktop environments

Running Linux from a Live DVD image

Navigating the GNOME 3 desktop

Adding extensions to GNOME 3

Using Nautilus to manage files in GNOME 3

Working with the GNOME 2 desktop

Enabling 3D effects in GNOME 2

Using Linux as your everyday desktop system is becoming easier to do all the time. As with everything in Linux, you have choices. There are fully featured GNOME or KDE desktop environments or lightweight desktops such as LXDE or Xfce. There are even simpler standalone window managers.

After you have chosen a desktop, you will find that almost every major type of desktop application you have on a Windows or Mac system has equivalent applications in Linux. For applications that are not available in Linux, you can often run a Windows application in Linux using Windows compatibility software.

The goal of this chapter is to familiarize you with the concepts related to Linux desktop systems and to give you tips for working with a Linux desktop. In this chapter you do the following:

- Step through the desktop features and technologies that are available in Linux
- Tour the major features of the GNOME desktop environment
- Learn tips and tricks for getting the most out of your GNOME desktop experience

To use the descriptions in this chapter, I recommend that you have a Fedora system running in front of you. You can get Fedora in lots of ways, including the following:

Running Fedora from a live medium Refer to Appendix A for information on downloading and burning Fedora Live image to a DVD or USB drive so that you can boot it live to use with this chapter.

Installing Fedora permanently Install Fedora to your hard disk and boot it from there (as described in Chapter 9, "Installing Linux").

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Because the current release of Fedora uses the GNOME 3 interface, most of the procedures described in this chapter work with other Linux distributions that have GNOME 3 available. If you are using an older Red Hat Enterprise Linux system (RHEL 6 uses GNOME 2, but RHEL 7 and RHEL 8 use GNOME 3), I added descriptions of GNOME 2 that you can try as well.

NOTE

Ubuntu switched from its own Unity desktop as its default to GNOME 3 with release 17.10. Unity is still available for newer releases, but only from the unsupported, community-maintained *Universe* repository.

Understanding Linux Desktop Technology

Modern computer desktop systems offer graphical windows, icons, and menus that are operated from a mouse and keyboard. If you are under 40 years old, you might think that there's nothing special about that. However, the first Linux systems did not have graphical interfaces available. Also, many Linux servers today that are tuned for special tasks (for example, functioning as a web server or file server) don't have desktop software installed.

Nearly every major Linux distribution that offers desktop interfaces is based on the X Window System originally from the X.Org Foundation (`http://www.x.org`). The X Window System provides a framework on which different types of desktop environments or simple window managers can be built. A replacement for X.Org called Wayland (`http://wayland.freedesktop.org`) is being developed. Although Wayland is the default X server for Fedora now, you can still choose X.Org instead.

The X Window System (sometimes simply called *X*) was created before Linux existed, and it even predates Microsoft Windows. It was built to be a lightweight, networked desktop framework.

X works in sort of a backward client/server model. The X server runs on the local system, providing an interface to your screen, mouse, and keyboard. X clients (such as word processors, music players, and image viewers) can be launched from the local system or from any system on your network to which the X server gives permission to do so.

X was created in a time when graphical terminals (thin clients) simply managed the keyboard, mouse, and display. Applications, disk storage, and processing power were all on larger centralized computers. So, applications ran on larger machines but were displayed and managed over the network on the thin client. Later, thin clients were replaced by desktop personal computers. Most client applications on PCs ran locally using local processing power, disk space, memory, and other hardware features, while applications that did not start from the local system were not allowed.

X itself provides a plain gray background and a simple "X" mouse cursor. There are no menus, panels, or icons on a plain X screen. If you were to launch an X client (such as a terminal window or word processor), it would appear on the X display with no border around it to move, minimize, or close the window. Those features are added by a window manager.

A *window manager* adds the capability to manage the windows on your desktop and often provides menus for launching applications and otherwise working with the desktop. A full-blown desktop environment includes a window manager, but it also adds menus, panels, and usually an application programming interface that is used to create applications that play well together.

So how does an understanding of how desktop interfaces work in Linux help you when it comes to using Linux? Here are some of the ways:

- Because Linux desktop environments are not required to run a Linux system, a Linux system may have been installed without a desktop. It might offer only a plain-text, command-line interface. You can choose to add a desktop later. After it is installed, you can choose whether to start up the desktop when your computer boots or start it as needed.
- For a very lightweight Linux system, such as one meant to run on less powerful computers, you can choose an efficient, though less feature-rich, window manager (such as `twm` or `fluxbox`) or a lightweight desktop environment (such as LXDE or Xfce).
- For more robust computers, you can choose more powerful desktop environments (such as GNOME and KDE) that can do things such as watch for events to happen (such as inserting a USB flash drive) and respond to those events (such as opening a window to view the contents of the drive).
- You can have multiple desktop environments installed and you can choose which one to launch when you log in. In this way, different users on the same computer can use different desktop environments.

Many different desktop environments are available to choose from in Linux. Here are some examples:

GNOME GNOME is the default desktop environment for Fedora, Red Hat Enterprise Linux, and many others. Think of it as a professional desktop environment focusing on stability more than fancy effects.

K Desktop Environment KDE is probably the second most popular desktop environment for Linux. It has more bells and whistles than GNOME and offers more integrated applications. KDE is also available with Fedora, Ubuntu, and many other Linux systems. For RHEL 8, KDE was dropped from the distribution.

Xfce The Xfce desktop was one of the first lightweight desktop environments. It is good to use on older or less powerful computers. It is available with Fedora, Ubuntu, and other Linux distributions.

LXDE The Lightweight X11 Desktop Environment (LXDE) was designed to be a fast-performing, energy-saving desktop environment. Often, LXDE is used on less-expensive devices (such as netbook computers) and on live media (such as a live CD or live USB stick). It is the default desktop for the KNOPPIX live CD distribution. Although LXDE is not included with RHEL, you can try it with Fedora or Ubuntu.

GNOME was originally designed to resemble the MacOS desktop, while KDE was meant to emulate the Windows desktop environment. Because it is the most popular desktop environment, and the one most often used in business Linux systems, most desktop procedures and exercises in this book use the GNOME desktop. Using GNOME, however, still gives you the choice of several different Linux distributions.

Starting with the Fedora GNOME Desktop Live image

A live Linux ISO image is the quickest way to get a Linux system up and running so that you can begin trying it out. Depending on its size, the image can be burned to a CD, DVD, or USB drive and booted on your computer. With a Linux live image, you can have Linux take over the operation of your computer temporarily without harming the contents of your hard drive.

If you have Windows installed, Linux just ignores it and takes control of your computer itself. When you are finished with the Linux live image, you can reboot the computer, pop out the CD or DVD, and go back to running whatever operating system was installed on the hard disk.

To try out a GNOME desktop along with the descriptions in this section, I suggest that you get a Fedora Live DVD (as described in Appendix A). Because a live DVD does all its work from the DVD and in memory, it runs slower than an installed Linux system. Also, although you can change files, add software, and otherwise configure your system, by default, the work you do disappears when you reboot unless you explicitly save that data to your hard drive or external storage.

The fact that changes you make to the live environment go away on reboot is very good for trying out Linux but not that great if you want an ongoing desktop or server system. For that reason, I recommend that if you have a spare computer, you install Linux permanently on that computer's hard disk to use with the rest of this book (as described in Chapter 9).

After you have a live CD or DVD in hand, do the following to get started:

1. **Get a computer**. If you have a standard PC (32-bit or 64-bit) with a CD/DVD drive, at least 1GB of memory (RAM), and at least a 1GHz processor, you are ready to start. (Just make sure that the image you download matches your computer's architecture—a 64-bit medium does not run on a 32-bit computer. Fedora 31 and RHEL 7 dropped 32-bit support, so you would need older versions of those distributions to run on those older machines.)
2. **Start the live CD/DVD**. Insert the live CD/DVD or USB drive into your computer and reboot your computer. Depending on the boot order set on your computer, the live image might start up directly from the BIOS (the code that controls the computer before the operating system starts).

Note

If, instead of the live medium booting, your installed operating system starts up instead, you need to perform an additional step to start the live CD/DVD. Reboot again, and when you see the BIOS screen, look for some words that say something like "Boot Order." The onscreen instructions may say to press the F12 or F1 key. Press that key immediately from the BIOS screen. Next, you should see a screen that shows available selections. Highlight an entry for CD/DVD or USB drive, and press Enter to boot the live image. If you don't see the drive there, you may need to go into BIOS setup and enable the CD/DVD or USB drive there.

3. **Start Fedora**. If the selected drive is able to boot, you see a boot screen. For Fedora, with Start Fedora highlighted, press Enter to start the live medium.
4. **Begin using the desktop**. For Fedora, the live medium lets you choose between installing Fedora or booting it directly from the medium to a GNOME 3 desktop.

You can now proceed to the next section, "Using the GNOME 3 Desktop" (which includes information on using GNOME 3 in Fedora, Red Hat Enterprise Linux, and other operating systems). Following that, I'll cover the GNOME 2 desktop.

Using the GNOME 3 Desktop

The GNOME 3 desktop offers a radical departure from its GNOME 2.*x* counterparts. GNOME 2.*x* is serviceable, but GNOME 3 is elegant. With GNOME 3, a Linux desktop now appears more like the graphical interfaces on mobile devices, with less focus on multiple mouse buttons and key combinations and more focus on mouse movement and one-click operations.

Instead of feeling structured and rigid, the GNOME 3 desktop seems to expand as you need it to. As a new application is run, its icon is added to the Dash. As you use the next workspace, a new one opens, ready for you to place more applications.

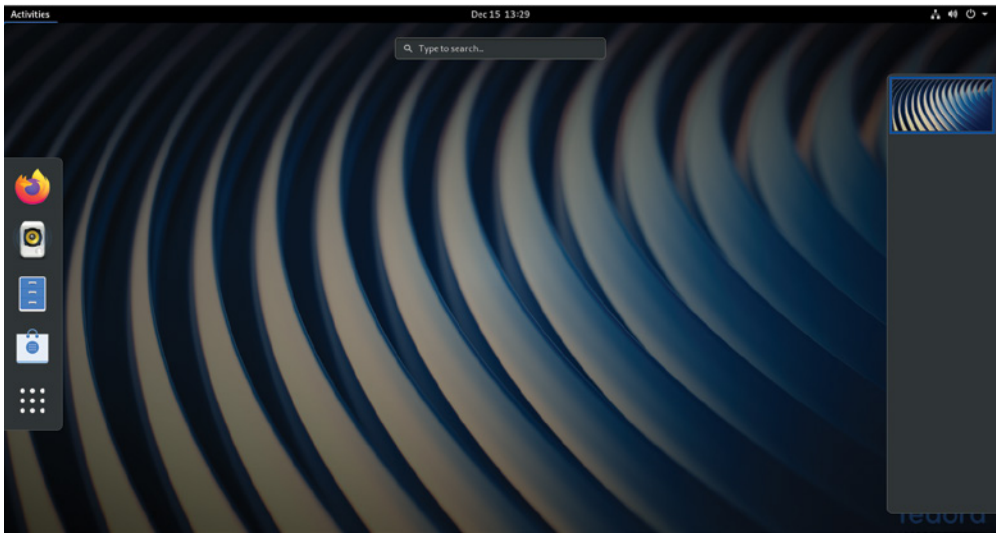
After the computer boots up

If you booted up a live image, when you reach the desktop, you are assigned as the Live System User for your username. For an installed system, you see the login screen, with user accounts on the system ready for you to select and enter a password. Log in with the username and password that you have defined for your system.

Figure 2.1 is an example of the GNOME 3 desktop screen that appears in Fedora. Press the Windows key (or move the mouse cursor to the upper-left corner of the desktop) to toggle between a blank desktop and the Overview screen.

FIGURE 2.1

Starting with the GNOME 3 desktop in Fedora.



There is very little on the GNOME 3 desktop when you start out. The top bar has the word "Activities" on the left, a clock in the middle, and some icons on the right for such things as adjusting audio volume, checking your network connection, and viewing the name of the current user. The Overview screen is where you can select to open applications, active windows, or different workspaces.

Navigating with the mouse

To get started, try navigating the GNOME 3 desktop with your mouse:

1. **Toggle activities and windows**. Move your mouse cursor to the upper-left corner of the screen near the Activities button. Each time you move there, your screen changes between showing you the windows that you are actively using and a set of available Activities. (This has the same effect as pressing the Windows key.)
2. **Open windows from applications bar**. Click to open some applications from the Dash on the left (Firefox, File Manager, Rhythmbox, or others). Move the mouse to the upper-left corner again, and toggle between showing all active windows minimized (Overview screen) and showing them overlapping (full-sized). Figure 2.2 shows an example of the miniature windows view.
3. **Open applications from Applications list**. From the Overview screen, select the Application button from the bottom of the left column (the button has nine dots in a box). The view changes to a set of icons representing the applications installed on your system, as shown in Figure 2.3.

FIGURE 2.2

Show all windows on the desktop minimized.

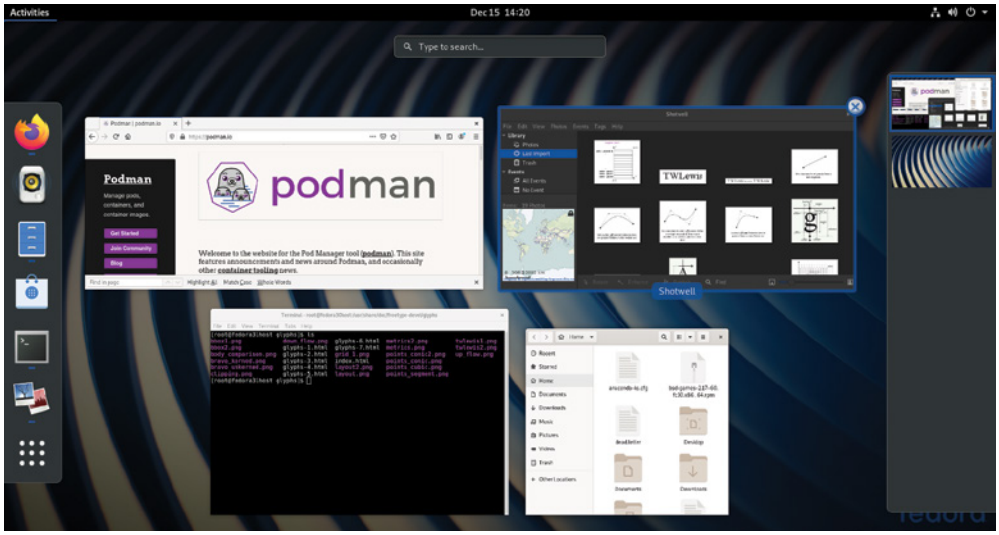


FIGURE 2.3

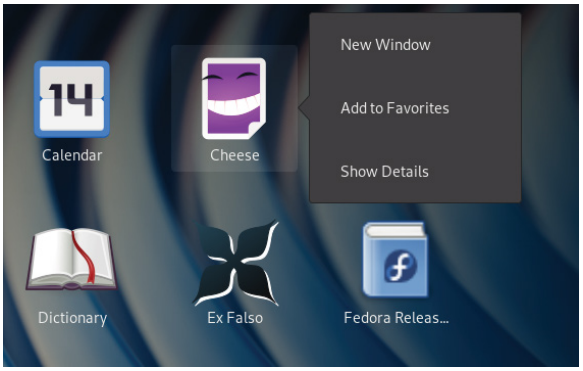
Show the list of available applications.



4. **View additional applications**. From the Applications screen, you can change the view of your applications in several ways, as well as launch them in different ways:
   a. **Page through**. To see icons representing applications that are not onscreen, use the mouse to click dots on the right to page through applications. If you have a wheel mouse, you can use that instead to scroll the icons.
   b. **Frequent**. Select the Frequent button on the bottom of the screen to see often-run applications or the All button to see all applications again.
   c. **Launching an application**. To start the application you want, left-click its icon to open the application in the current workspace. Right-click to open a menu that lets you choose to open a New Window, add or remove the application from Favorites (so the application's icon appears on the Dash), or Show Details about the application. Figure 2.4 shows an example of the menu.

FIGURE 2.4

Click the middle mouse button to display an application's selection menu.

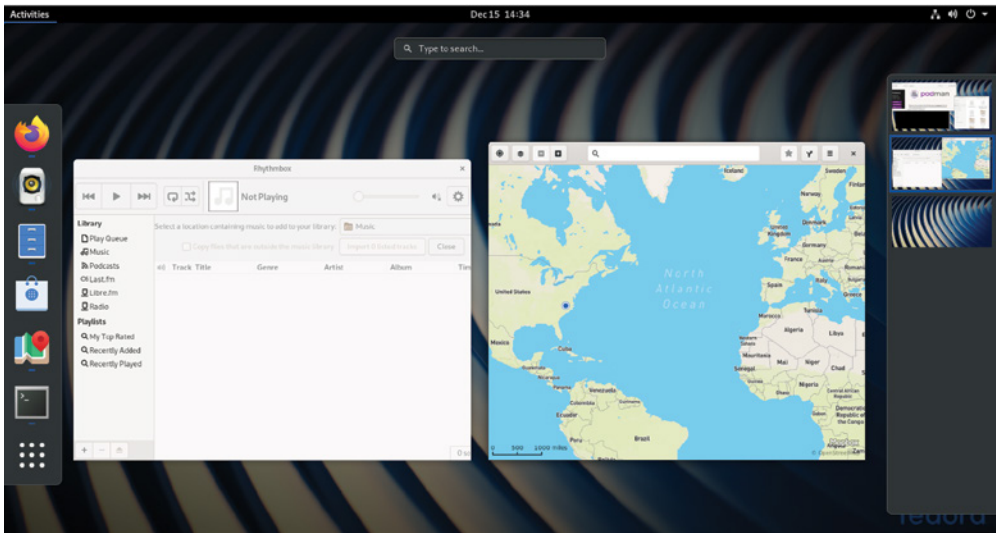


5. **Open additional applications**. Start up additional applications. Notice that as you open a new application, an icon representing that application appears in the Dash bar on the left. Here are other ways to start applications:
   a. **Application icon**. Click any application icon to open that application.
   b. **Drop Dash icons on workspace**. From the Windows view, you can drag any application icon from the Dash by pressing and holding the left mouse button on it and dragging that icon to any of the miniature workspaces on the right.
6. **Use multiple workspaces**. Move the mouse to the upper-left corner again to show a minimized view of all windows. Notice all of the applications on the right jammed into a small representation of one workspace while an additional workspace is empty. Drag and drop a few of the windows to an empty desktop space. Figure 2.5 shows what the small workspaces look like. Notice that an additional empty

workspace is created each time the last empty one is used. You can drag and drop the miniature windows to any workspace and then select the workspace to view it.

FIGURE 2.5

As new desktops are used, additional ones appear on the right.



7. **Use the window menu**. Move the mouse to the upper-left corner of the screen to return to the active workspace (large window view). Right-click the title bar on a window to view the window menu. Try these actions from that menu:

   a. **Minimize**. Remove window temporarily from view.

   b. **Maximize**. Expand window to maximum size.

   c. **Move**. Change window to moving mode. Moving your mouse moves the window. Click to fix the window to a spot.

   d. **Resize**. Change the window to resize mode. Moving your mouse resizes the window. Click to keep the size.

   e. **Workspace selections**. Several selections let you use workspaces in different ways. Select Always on Top to make the current window always on top of other windows in the workspace. Select Always on Visible Workspace to always show the window on the workspace that is visible, or select Move to Workspace Up or Move to Workspace Down to move the window to the workspace above or below, respectively.

If you don't feel comfortable navigating GNOME 3 with your mouse, or if you don't have a mouse, the next section helps you navigate the desktop from the keyboard.

Navigating with the keyboard

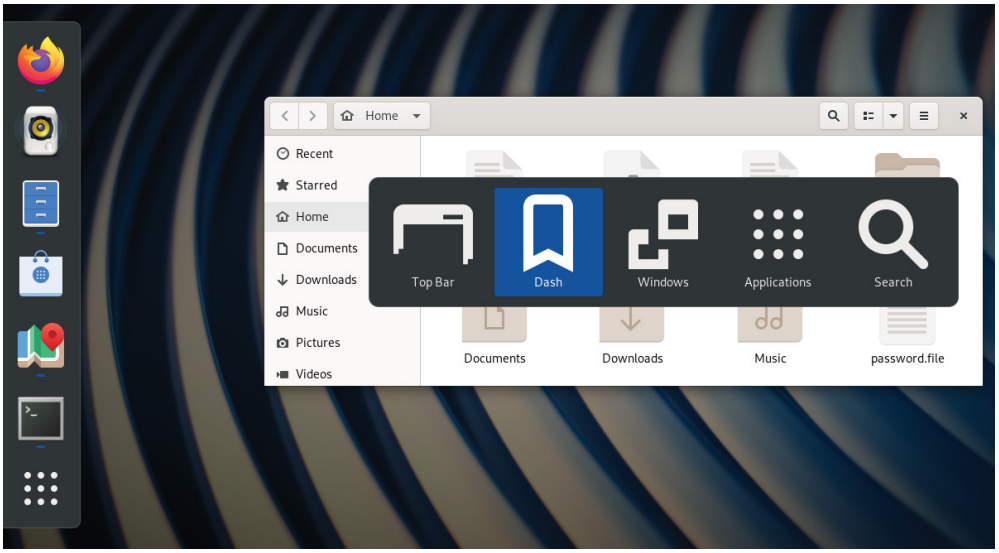
If you prefer to keep your hands on the keyboard, you can work with the GNOME 3 desktop directly from the keyboard in a number of ways, including the following:

Windows key. Press the Windows key on the keyboard. On most PC keyboards, this is the key with the Microsoft Windows logo on it next to the Alt key. This toggles the mini-window (Overview) and active-window (current workspace) views. Many people use this key often.

Select different views. From the Windows or Applications view, hold Ctrl+Alt+Tab to see a menu of the different views (see Figure 2.6). Still holding the Ctrl+Alt keys, press Tab again to highlight one of the following icons from the menu and release to select it:

FIGURE 2.6

Press Ctrl+Alt+Tab to display additional desktop areas to select.



Top Bar. Highlights the top bar. After it is selected, you can tab between items on that bar (Activities, Calendar, and the Top Bar menu).

Dash. Highlights the first application in the application bar on the left. Use arrow keys to move up and down that menu, and press Enter to open the highlighted application.

Windows. Selects the Windows view.

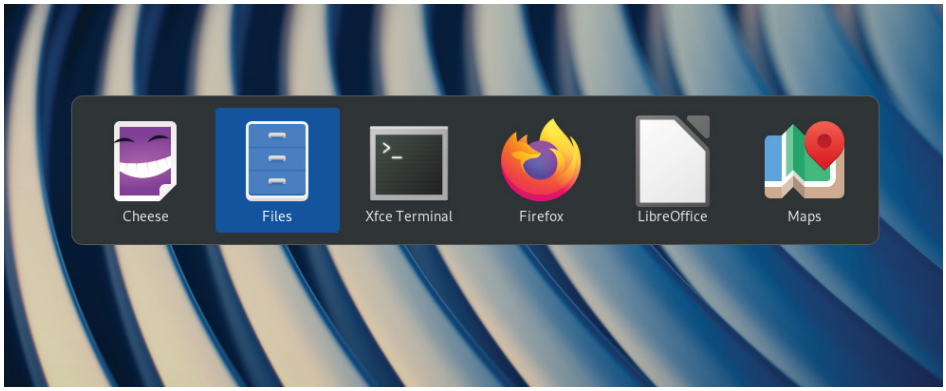
Applications. Selects the Applications view.

Search. Highlights the search box. Type a few letters to show only icons for applications that contain the letters you type. When you have typed enough letters to uniquely identify the application you want, press Enter to launch the application.

Select an active window. Return to any of your workspaces (press the Windows key if you are not already on an active workspace). Press Alt+Tab to see a list of all active windows (see Figure 2.7). Continue to hold the Alt key as you press the Tab key (or right or left arrow keys) to highlight the application that you want from the list of active desktop application windows. If an application has multiple windows open, press Alt+` (back-tick, located above the Tab key) to choose among those sub-windows. Release the Alt key to select it.

FIGURE 2.7

Press Alt+Tab to select which running application to go to.



Launch a command or application. From any active workspace, you can launch a Linux command or a graphical application. Here are some examples:

Applications. From the Overview screen, press Ctrl+Alt+Tab and continue to press Tab until the Applications icon is highlighted; then release Ctrl+Alt. The Applications view appears, with the first icon highlighted. Use the Tab key or arrow keys (up, down, right, and left) to highlight the application icon you want, and press Enter.

Command box. If you know the name (or part of a name) of a command that you want to run, press Alt+F2 to display a command box. Type the name of the command that you want to run into the box (try **gnome-calculator** to open a calculator application, for example).

Search box. From the Overview screen, press Ctrl+Alt+Tab and continue to press Tab until the magnifying glass (Search) icon is highlighted; then release Ctrl+Alt. In the search box now highlighted, type a few letters in an application's name or description (type **scr** to see what you get). Keep typing until the application you want is highlighted (in this case, Screenshot), and press Enter to launch it.

Dash. From the Overview screen, press Ctrl+Alt+Tab and continue to press Tab until the star (Dash) icon is highlighted; then release Ctrl+Alt. From the Dash, move the up and down arrows to highlight an application that you want to launch and press Enter.

Escape. When you are stuck in an action that you don't want to complete, try pressing the Esc key. For example, after pressing Alt+F2 (to enter a command), opening an icon from the top bar, or going to an overview page, pressing Esc returns you to the active window on the active desktop.

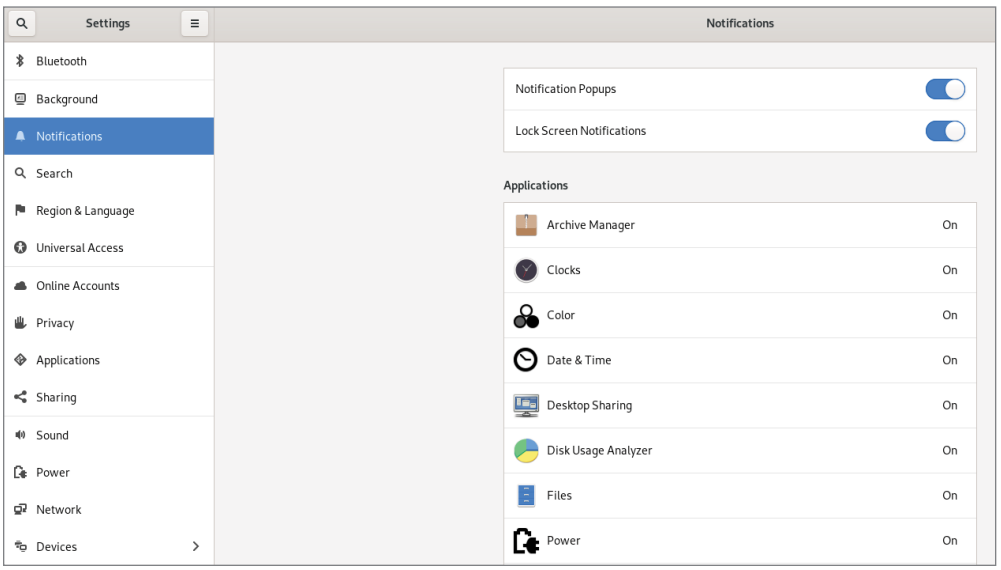
I hope you now feel comfortable navigating the GNOME 3 desktop. Next, you can try running some useful and fun desktop applications from GNOME 3.

Setting up the GNOME 3 desktop

Much of what you need GNOME 3 to do for you is set up automatically. However, you need to make a few tweaks to get the desktop the way you want. Most of these setup activities are available from the System Settings window (see Figure 2.8). Open the Settings icon from the Applications list.

FIGURE 2.8

Change desktop settings from the System Settings window.



Here are some suggestions for configuring a GNOME 3 desktop:

Configure networking. A wired network connection is often configured automatically when you boot up your Fedora system. For wireless, you probably have to select your wireless network and add a password when prompted. An icon in the top bar lets you do any wired or wireless network configuration that you need to do. Refer to Chapter 14, "Administering Networking," for further information on configuring networking.

Bluetooth. If your computer has Bluetooth hardware, you can enable that device to communicate with other Bluetooth devices (such as a Bluetooth headset or printer).

Devices. From the Devices screen, you can configure your keyboard, mouse and touchpad, printers, removable media, and other settings.

Sound. Click the Sound settings button to adjust sound input and output devices on your system.

Extending the GNOME 3 desktop

If the GNOME 3 shell doesn't do everything you'd like, don't despair. You can add extensions to provide additional functionality to GNOME 3. Also, a tool called GNOME Tweaks lets you change advanced settings in GNOME 3.

Using GNOME shell extensions

GNOME shell extensions are available to change the way your GNOME desktop looks and behaves. Visit the GNOME Shell Extensions site (`http://extensions.gnome.org`) from your Firefox browser on your GNOME 3 desktop. That site tells you what extensions you have installed and which ones are available for you to install. (You must select to allow the site to see those extensions.)

Because the extensions page knows what extensions you have and the version of GNOME 3 that you are running, it can present only those extensions that are compatible with your system. Many of the extensions help you add back in features from GNOME 2, including the following:

Applications Menu. Adds an Applications menu to the top panel, just as it was in GNOME 2.

Places Status Indicator. Adds a systems status menu, similar to the Places menu in GNOME 2, to let you navigate quickly to useful folders on your system.

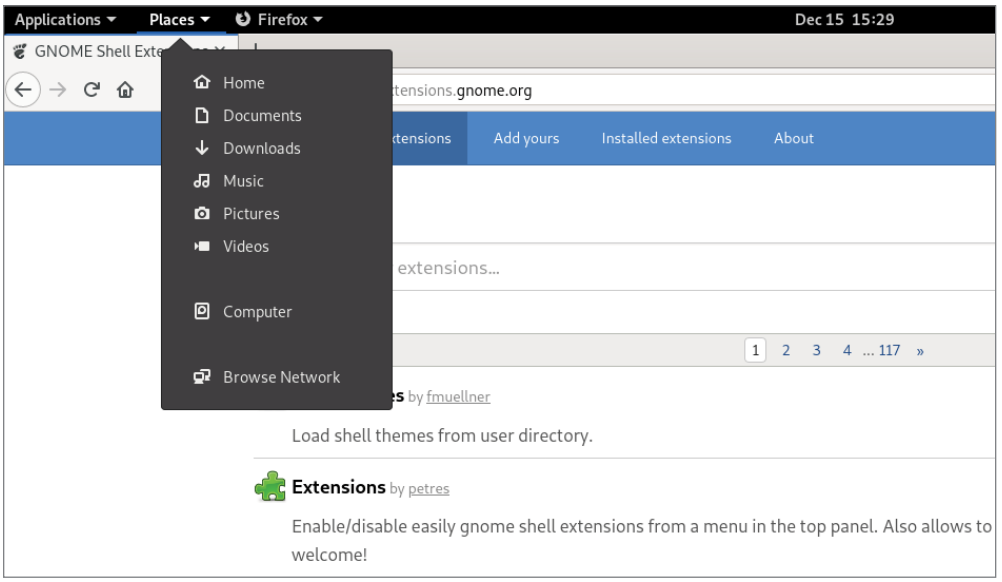
Window list. Adds a list of active windows to the top panel, similar to the Window list that appeared on the bottom panel in GNOME 2.

To install an extension, simply select the ON button next to the name. Or, you can click the extension name from the list to see the extension's page and click the button on that page from OFF to ON. Click Install when you are asked if you want to download and install the extension. The extension is then added to your desktop.

Figure 2.9 shows an example of the Applications menu Window List (showing several active applications icons), and Places Status Indicator (with folders displayed from a drop-down menu) extensions installed.

FIGURE 2.9

Extensions add features to the GNOME 3 desktop.



More than 100 GNOME shell extensions are available now, and more are being added all the time. Other popular extensions include Notifications Alert (which alerts you of unread messages), Presentation Mode (which prevents the screensaver from coming on when you are giving a presentation), and Music Integration (which integrates popular music players into GNOME 3, so that you are alerted about songs being played).

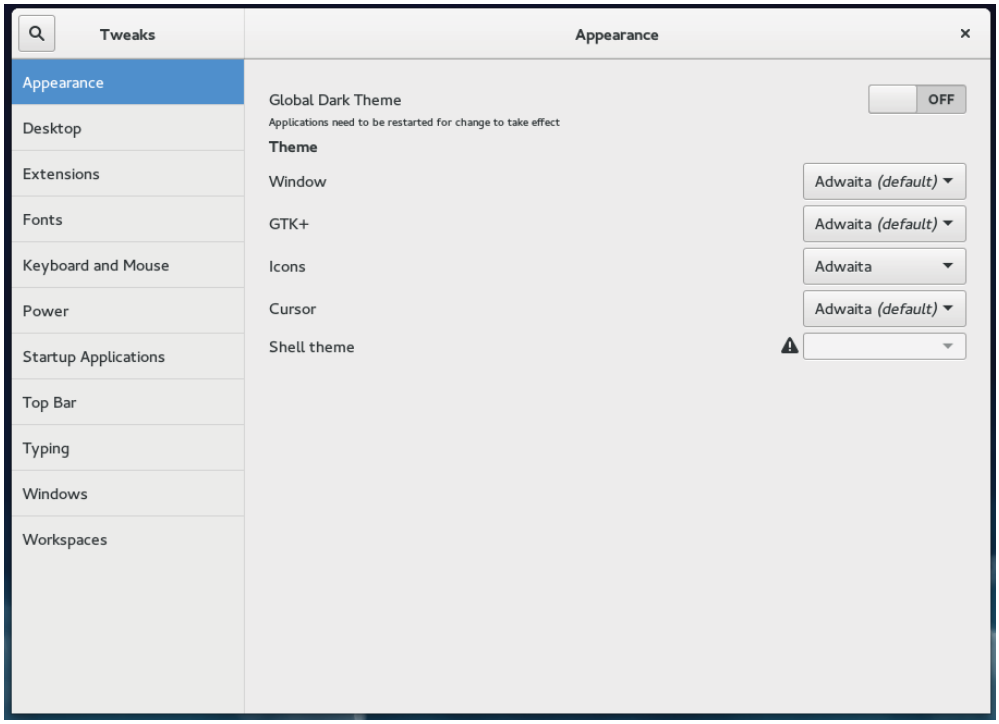
Because the Extensions site can keep track of your extensions, you can click the Installed extensions button at the top of the page and see every extension that is installed. You can turn the extensions off and on from there and even delete them permanently.

Using the GNOME Tweak Tool

If you don't like the way some of the built-in features of GNOME 3 behave, you can change many of them with the GNOME Tweak Tool. This tool is not installed by default with the Fedora GNOME Live CD, but you can add it by installing the `gnome-tweaks` package. (See Chapter 10, "Getting and Managing Software," for information on how to install software packages in Fedora.) After installation, the GNOME Tweak Tool is available by launching the Advanced Settings icon from your Applications screen. Start with the Desktop category to consider what you might want to change in GNOME 3. Figure 2.10 shows the Tweak Tool displaying Appearance settings.

FIGURE 2.10

Change desktop settings using the GNOME Tweak Tool (Appearance settings).



If fonts are too small for you, select the Fonts category and click the plus sign next to the Scaling Factor box to increase the font size, or change fonts individually for documents, window titles, or monospace fonts.

Under Top Bar settings, you can change how clock information is displayed in the top bar or set whether to show the week number in the calendar. To change the look of the desktop, select the Appearance category and change the Icons theme and GTK+ theme as you like from drop-down boxes.

Starting with desktop applications

The Fedora GNOME 3 desktop live DVD comes with some cool applications that you can start using immediately. To use GNOME 3 as your everyday desktop, you should install it permanently to your computer's hard disk and add the applications you need (a word processor, image editor, drawing application, and so on). If you are just getting started, the following sections list some cool applications to try out.

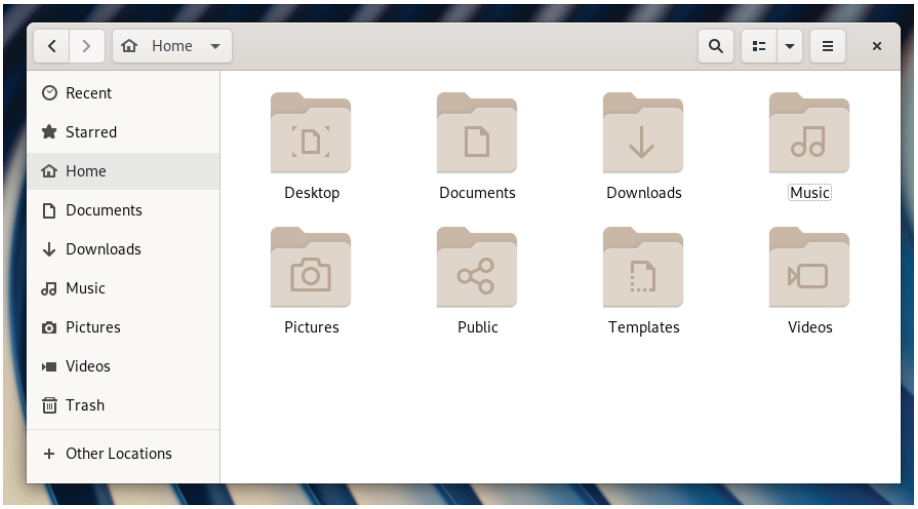
Managing files and folders with Nautilus

To move, copy, delete, rename, and otherwise organize files and folders in GNOME 3, you can use the Nautilus file manager. Nautilus comes with the GNOME desktop and works like other file managers that you may use in Windows or Mac.

To open Nautilus, click the Files icon from the GNOME Dash or Applications list. Your user account starts with a set of folders designed to hold the most common types of content: Music, Pictures, Videos, and the like. These are all stored in what is referred to as your Home directory. Figure 2.11 shows Nautilus open to a Home directory.

FIGURE 2.11

Manage files and folders from the Nautilus window.



When you want to save files that you downloaded from the Internet or created with a word processor, you can organize them into these folders. You can create new folders as needed, drag and drop files and folders to copy and move them, and delete them.

Because Nautilus is not much different from most file managers that you have used on other computer systems, this chapter does not go into detail about how to use drag-and-drop and traverse folders to find your content. However, I do want to make a few observations that may not be obvious about how to use Nautilus:

Home folder You have complete control over the files and folders that you create in your Home folder. Most other parts of the filesystem are not accessible to you as a regular user.

Filesystem organization Although it appears under the name Home, your Home folder is actually located in the filesystem under the `/home` folder in a folder named

after your username: for example, `/home/liveuser` or `/home/chris`. In the next few chapters, you learn how the filesystem is organized (especially in relation to the Linux command shell).

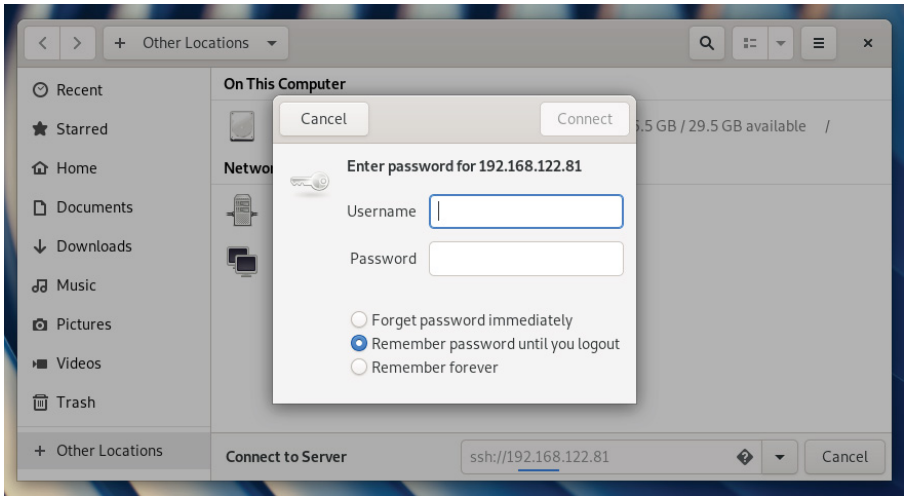
Working with files and folders Right-click a file or folder icon to see how you can act on it. For example, you can copy, cut, move to trash (delete), or open any file or folder icon.

Creating folders To create a new folder, right-click in a folder window and select New Folder. Type the new folder name over the highlighted Untitled Folder, and press Enter to name the folder.

Accessing remote content Nautilus can display content from remote servers as well as the local filesystem. In Nautilus, select Other Locations from the file menu. From the Connect to Server box that appears, you can connect to a remote server via SSH (secure shell), FTP with login, Public FTP, Windows share, WebDav (HTTP), or Secure WebDav (HTTPS). Add appropriate user and password information as needed, and the content of the remote server appears in the Nautilus window. Figure 2.12 shows an example of a Nautilus window prompting you for a password to log into a remote server over SSH protocol (`ssh://192.168.122.81`).

FIGURE 2.12

Access remote folders using the Nautilus Connect to Server feature.



Installing and managing additional software

The Fedora Live Desktop comes with a web browser (Firefox), a file manager (Nautilus), and a few other common applications. However, there are many other useful applications that, because of their size, just wouldn't fit on a live CD. If you install the live Fedora Workstation to your hard disk (as described in Chapter 9), you almost certainly will want to add some more software.

Note

You can try installing software if you are running the live medium. However, keep in mind that because writeable space on a live medium uses virtual memory (RAM), that space is limited and can easily run out. Also, when you reboot your system, anything that you install disappears.

When Fedora is installed, it is automatically configured to connect your system to the huge Fedora software repository that is available on the Internet. As long as you have an Internet connection, you can run the Add/Remove software tool to download and install any of thousands of Fedora packages.

Although the entire facility for managing software in Fedora (the `yum` and `rpm` features) is described in detail in Chapter 10, you can start installing some software packages without knowing much about how the feature works. Begin by going to the applications screen and opening the Software window. Figure 2.13 shows an example of the Software window.

With the Software window open, you can select the applications that you want to install by searching (type the name into the Find box) or choosing a category. Each category offers packages sorted by subcategories and featured packages in that category.

Select the spyglass icon in the upper-left corner, and then type a word associated with the software package that you want to install. You can read a description of each package that comes up in your search. When you are ready, click Install to install the package and any dependent packages needed to make it work.

FIGURE 2.13

Download and install software from the huge Fedora repository.



By searching for and installing some common desktop applications, you should be able to start using your desktop effectively. Refer to Chapter 10 for details on how to add software repositories and use `dnf`, `yum`, and `rpm` commands to manage software in Fedora and Red Hat Enterprise Linux.

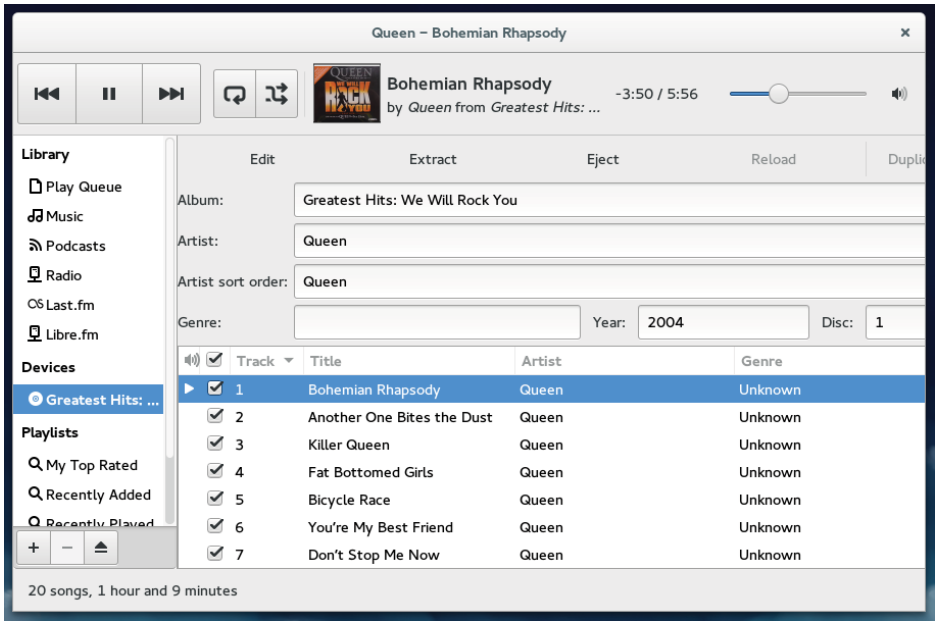
Playing music with Rhythmbox

Rhythmbox is the music player that comes on the Fedora GNOME Live Desktop. You can launch Rhythmbox from the GNOME 3 Dash and immediately play music CDs, podcasts, or Internet radio shows. You can import audio files in WAV and Ogg Vorbis formats or add plug-ins for MP3 or other audio formats.

Figure 2.14 shows an example of the Rhythmbox window with music playing from an imported audio library.

FIGURE 2.14

Play music, podcasts, and Internet radio from Rhythmbox.



Here are a few ways that you can get started with Rhythmbox:

Radio Double-click the Radio selection under Library and choose a radio station from the list that appears to the right.

Podcasts Search for podcasts on the Internet and find the URL for one that interests you. Right-click the Podcasts entry and select New Podcast Feed. Paste or type in the URL to the podcast and click Add. A list of podcasts from the site that you selected appears to the right. Double-click the one to which you want to listen.

Audio CDs Insert an audio CD, and press Play when it appears in the Rhythmbox window. Rhythmbox also lets you rip and burn audio CDs.

Audio files Rhythmbox can play WAV and Ogg Vorbis files. By adding plug-ins, you can play many other audio formats, including MP3. Because there are patent issues related to the MP3 format, the ability to play MP3s is not included with Fedora. In Chapter 10, I describe how to get software that you need that is not in the repository of your Linux distribution.

Plug-ins are available for Rhythmbox to get cover art, show information about artists and songs, add support for music services (such as Last.fm and Magnatune), and fetch song lyrics.

Stopping the GNOME 3 desktop

When you are finished with your GNOME 3 session, select the down arrow button in the upper-right corner of the top bar. From there, you can choose the On/Off button, which allows you to log out or switch to a different user account without logging out.

Using the GNOME 2 Desktop

The GNOME 2 desktop is the default desktop interface used up through Red Hat Enterprise Linux 6. It is well-known, stable, and perhaps a bit boring.

GNOME 2 desktops provide the more standard menus, panels, icons, and workspaces. If you are using a Red Hat Enterprise Linux system up to RHEL 6, or an older Fedora or Ubuntu distribution, you are probably looking at a GNOME 2 desktop. I will now provide a tour of GNOME 2, along with some opportunities for sprucing it up a bit. GNOME 2 releases include 3D effects (see "Adding 3D effects with AIGLX" later in this chapter).

To use your GNOME desktop, you should become familiar with the following components:

Metacity (window manager) The default window manager for GNOME 2 is Metacity. Metacity configuration options let you control such things as themes, window borders, and controls used on your desktop.

Compiz (window manager) You can enable this window manager in GNOME to provide 3D desktop effects.

Nautilus (file manager/graphical shell) When you open a folder (by double-clicking the Home icon on your desktop, for example), the Nautilus window opens

and displays the contents of the selected folder. Nautilus can also display other types of content, such as shared folders from Windows computers on the network (using SMB).

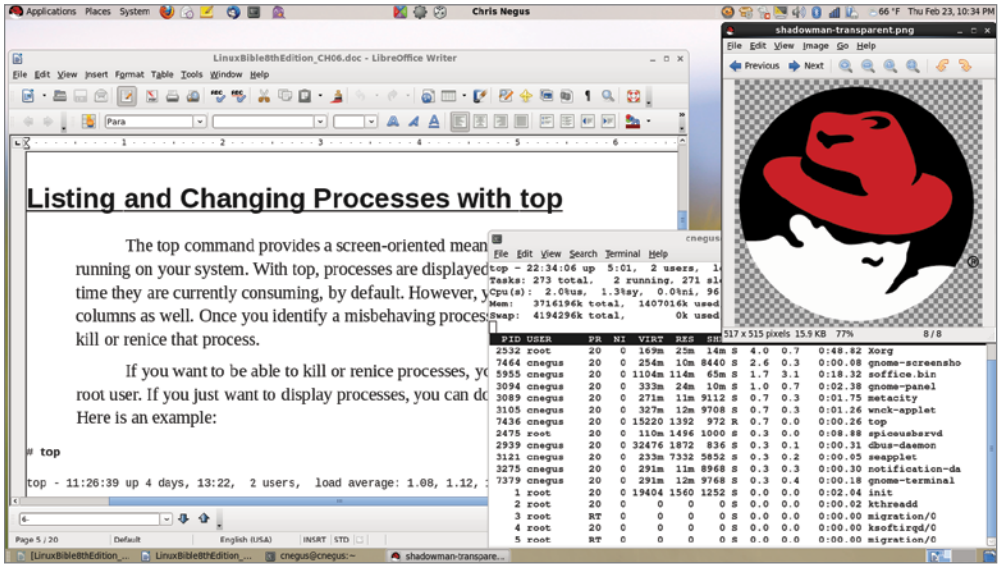
GNOME panels (application/task launcher) These panels, which line the top and bottom of your screen, are designed to make it convenient for you to launch the applications you use, manage running applications, and work with multiple virtual desktops. By default, the top panel contains menu buttons (Applications, Places, and System), desktop application launchers (Evolution email and Firefox web browser), a workspace switcher (for managing four virtual desktops), and a clock. Icons appear in the panel when you need software updates or SELinux detects a problem. The bottom panel has a Show Desktop button, window lists, a trash can, and workspace switcher.

Desktop area The windows and icons you use are arranged on the desktop area, which supports drag-and-drop between applications, a desktop menu (right-click to see it), and icons for launching applications. A Computer icon consolidates CD drives, floppy drives, the filesystem, and shared network resources in one place.

GNOME also includes a set of Preferences windows that enable you to configure different aspects of your desktop. You can change backgrounds, colors, fonts, keyboard shortcuts, and other features related to the look and behavior of the desktop. Figure 2.15 shows how the GNOME 2 desktop environment appears the first time you log in, with a few windows added to the screen.

FIGURE 2.15

The GNOME 2 desktop environment



The desktop shown in Figure 2.15 is for Red Hat Enterprise Linux. The following sections provide details on using the GNOME 2 desktop.

Using the Metacity window manager

The Metacity window manager seems to have been chosen as the default window manager for GNOME because of its simplicity. The creator of Metacity refers to it as a "boring window manager for the adult in you" and then goes on to compare other window managers to colorful, sugary cereal, whereas Metacity is characterized as Cheerios.

NOTE

To use 3D effects, your best solution is to use the Compiz window manager, described later in this chapter. You can't do much with Metacity (except get your work done efficiently). You assign new themes to Metacity and change colors and window decorations through the GNOME preferences (described later).

Basic Metacity functions that might interest you are keyboard shortcuts and the workspace switcher. Table 2.1 shows keyboard shortcuts to get around the Metacity window manager.

TABLE 2.1 Keyboard Shortcuts

Actions	Keystrokes
Cycle backward, without pop-up icons	Alt+Shift+Esc
Cycle backward among panels	Alt+Ctrl+Shift+Tab
Close menu	Esc

You can use other keyboard shortcuts with the window manager as well. Select System ➪ Preferences ➪ Keyboard Shortcuts to see a list of shortcuts, such as the following:

Run Dialog To run a command to launch an application from the desktop by command name, press Alt+F2. From the dialog box that appears, type the command and press Enter. For example, type `gedit` to run a simple graphical text editor.

Lock Screen If you want to step away from your screen and lock it, press Ctrl+Alt+L. You need to type your user password to open the screen again.

Show Main Menu To open an application from the Applications, Places, or System menu, press Alt+F1. Then use the up and down arrow keys to select from the current menu or use the right and left arrow keys to select from other menus.

Print Screen Press the Print Screen key to take a picture of the entire desktop. Press Alt+Print Screen to take a picture of the current window.

Another Metacity feature of interest is the workspace switcher. Four virtual workspaces appear in the workspace switcher on the GNOME 2 panel. You can do the following with the Workspace Switcher:

Choose current workspace Four virtual workspaces appear in the workspace switcher. Click any of the four virtual workspaces to make it your current workspace.

Move windows to other workspaces Click any window, each represented by a tiny rectangle in a workspace, to drag and drop it to another workspace. Likewise, you can drag an application from the Window list to move that application to another workspace.

Add more workspaces Right-click the Workspace Switcher and select Preferences. You can add workspaces (up to 32).

Name workspaces Right-click the Workspace Switcher and select Preferences. Click in the Workspaces pane to change names of workspaces to any names you choose.

You can view and change information about Metacity controls and settings using the `gconf-editor` window (type **`gconf-editor`** from a Terminal window). As the window says, it is not the recommended way to change preferences, so when possible, you should change the desktop through GNOME 2 preferences. However, `gconf-editor` is a good way to see descriptions of each Metacity feature.

From the `gconf-editor` window, select apps ➪ metacity, and choose from general, global_keybindings, keybindings_commands, window_keybindings, and workspace_names. Click each key to see its value, along with short and long descriptions of the key.

Changing GNOME's appearance

You can change the general look of your GNOME desktop by selecting System ➪ Preferences ➪ Appearance. From the Appearance Preferences window, select from three tabs:

Theme Entire themes are available for the GNOME 2 desktop that change the colors, icons, fonts, and other aspects of the desktop. Several different themes come with the GNOME desktop, which you can simply select from this tab to use. Or click "Get more themes online" to choose from a variety of available themes.

Background To change your desktop background, select from a list of backgrounds on this tab to have the one you choose immediately take effect. To add a different background, put the background you want on your system (perhaps download one by selecting "Get more backgrounds online" and downloading it to your Pictures folder). Then click Add and select the image from your Pictures folder.

Fonts Different fonts can be selected to use by default with your applications, documents, desktop, window title bar, and for fixed width.

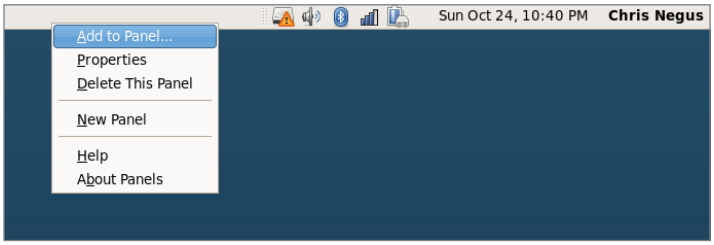
Using the GNOME panels

The GNOME panels are placed on the top and bottom of the GNOME desktop. From those panels, you can start applications (from buttons or menus), see what programs are active, and monitor how your system is running. You can also change the top and bottom panels in many ways—by adding applications or monitors or by changing the placement or behavior of the panel, for example.

Right-click any open space on either panel to see the Panel menu. Figure 2.16 shows the Panel menu on the top.

FIGURE 2.16

The GNOME Panel menu



From GNOME's Panel menu, you can choose from a variety of functions, including these:

Use the menus:

- The Applications menu displays most of the applications and system tools that you will use from the desktop.
- The Places menu lets you select places to go, such as the Desktop folder, Home folder, removable media, or network locations.
- The System menu lets you change preferences and system settings as well as get other information about GNOME.

Add to Panel. Add an applet, menu, launcher, drawer, or button.

Properties. Change the panel's position, size, and background properties.

Delete This Panel. Delete the current panel.

New Panel. Add panels to your desktop in different styles and locations.

You can also work with items on a panel. For example, you can do the following:

Move items. To move an item on a panel, right-click it, select Move, and drag and drop it to a new position.

Resize items. You can resize some elements, such as the Window list, by clicking an edge and dragging it to the new size.

Use the Window list. Tasks running on the desktop appear in the Window list area. Click a task to minimize or maximize it.

The following sections describe some things that you can do with the GNOME panel.

Using the Applications and System menus

Click Applications on the panel and you see categories of applications and system tools that you can select. Click the application that you want to launch. To add an item from a menu so that it can launch from the panel, drag and drop the item that you want to the panel.

You can add items to your GNOME 2 menus. To do that, right-click any of the menu names and select Edit Menus. The window that appears lets you add or delete menus associated with the Applications and System menus. You can also add items to launch from those menus by selecting New Item and typing the name, command, and comment for the item.

Adding an applet

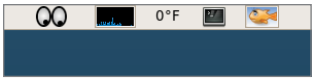
You can run several small applications, called *applets*, directly on the GNOME panel. These applications can show information that you may want to see on an ongoing basis or may just provide some amusement. To see what applets are available and to add applets that you want to your panel, follow these steps:

1. **Right-click an open space in the panel so that the Panel menu appears**.
2. **Click Add to Panel**. An Add to Panel window appears.
3. **Select from among several dozen applets, including a clock, dictionary lookup, stock ticker, and weather report**. The applet you select appears on the panel, ready for you to use.

Figure 2.17 shows (from left to right) eyes, system monitor, weather report, terminal, and Wanda the fish.

FIGURE 2.17

Placing applets on the panel makes accessing them easy.



After an applet is installed, right-click it on the panel to see what options are available. For example, select Preferences for the stock ticker and you can add or delete stocks whose prices you want to monitor. If you don't like the applet's location, right-click it, click Move, slide the mouse until the applet is where you want it (even to another panel), and click to set its location.

If you no longer want an applet to appear on the panel, right-click it, and click Remove From Panel. The icon representing the applet disappears. If you find that you have run out of room on your panel, you can add a new panel to another part of the screen, as described in the next section.

Adding another panel

If you run out of space on the top or bottom panels, you can add more panels to your desktop. You can have several panels on your GNOME 2 desktop. You can add panels that run along the entire bottom, top, or side of the screen. To add a panel, follow these steps:

1. **Right-click an open space in the panel so that the Panel menu appears**.
2. **Click New Panel**. A new panel appears on the side of the screen.
3. **Right-click an open space in the new panel and select Properties**.
4. **From the Panel Properties, select where you want the panel from the Orientation box** (Top, Bottom, Left, or Right).

After you've added a panel, you can add applets or application launchers to it as you did with the default panel. To remove a panel, right-click it and select Delete This Panel.

Adding an application launcher

Icons on your panel represent a web browser and several office productivity applications. You can add your own icons to launch applications from the panel as well. To add a new application launcher to the panel, follow these steps:

1. **Right-click in an open space on the panel**.
2. **Click Add to Panel ➪ Application Launcher from the menu**. All application categories from your Applications and System menus appear.
3. **Select the arrow next to the category of application you want, and then select Add**. An icon representing the application appears on the panel.

To launch the application that you just added, simply click the icon on the panel.

If the application that you want to launch is not on one of your menus, you can build a launcher yourself as follows:

1. **Right-click in an open space on the panel**.
2. **Click Add to Panel ➪ Custom Application Launcher ➪ Add**. The Create Launcher window appears.
3. **Provide the following information for the application you want to add**:
   a. **Type**. Select Application (to launch a regular GUI application) or Application in Terminal. Use Application in Terminal if the application is a character-based or ncurses application. (Applications written using the ncurses library run in a Terminal window but offer screen-oriented mouse and keyboard controls.)

b. **Name**. Choose a name to identify the application. (This appears in the tooltip when your mouse is over the icon.)

c. **Command**. This identifies the command line that is run when the application is launched. Use the full pathname, plus any required options.

d. **Comment**. Enter a comment describing the application. It also appears when you later move your mouse over the launcher.

4. **Click the Icon box (it might say No Icon), select one of the icons shown, and click OK**. Alternatively, you can browse your filesystem to choose an icon.
5. **Click OK**.

The application should now appear in the panel. Click it to start the application.

NOTE

Icons available to represent your application are contained in the `/usr/share/pixmaps` directory. These icons are either in PNG or XPM format. If there isn't an icon in the directory that you want to use, create your own (in one of those two formats) and assign it to the application.

Adding a drawer

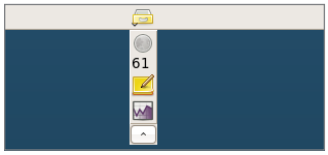
A *drawer* is an icon that you can click to display other icons representing menus, applets, and launchers; it behaves just like a panel. Essentially, any item that you can add to a panel you can add to a drawer. By adding a drawer to your GNOME panel, you can include several applets and launchers that together take up the space of only one icon. Click the drawer to show the applets and launchers as if they were being pulled out of a drawer icon on the panel.

To add a drawer to your panel, right-click the panel and select Add to Panel ➪ Drawer. A drawer appears on the panel. Right-click it and add applets or launchers to it as you would to a panel. Click the icon again to retract the drawer.

Figure 2.18 shows a portion of the panel with an open drawer that includes an icon for launching a weather report, sticky notes, and stock monitor.

FIGURE 2.18

Add launchers or applets to a drawer on your GNOME 2 panel.



Changing panel properties

You can change the orientation, size, hiding policy, and background properties of your desktop panels. To open the Panel Properties window that applies to a specific panel, right-click an open space on the panel and choose Properties. The Panel Properties window that appears includes the following values:

Orientation Move the panel to a different location on the screen by clicking a new position.

Size Select the size of your panel by choosing its height in pixels (48 pixels by default).

Expand Select this check box to have the panel expand to fill the entire side or clear the check box to make the panel only as wide as the applets it contains.

AutoHide Select whether a panel is automatically hidden (appearing only when the mouse pointer is in the area).

Show Hide buttons Choose whether the Hide/Unhide buttons (with pixmap arrows on them) appear on the edges of the panel.

Arrows on Hide buttons If you select Show Hide Buttons, you can choose to have arrows on those buttons.

Background From the Background tab, you can assign a color to the background of the panel, assign a pixmap image, or just leave the default (which is based on the current system theme). Click the Background Image check box if you want to select an Image for the background, and then select an image, such as a tile from `/usr/share/backgrounds/tiles` or another directory.

Tip

I usually turn on the AutoHide feature and turn off the Hide buttons. Using AutoHide gives you more desktop space with which you can work. When you move your mouse to the edge where the panel is located, the panel pops up—so you don't need Hide buttons.

Adding 3D effects with AIGLX

Several initiatives have made strides in recent years to bring 3D desktop effects to Linux. Ubuntu, openSUSE, and Fedora used AIGLX (`https://fedoraproject.org/wiki/RenderingProject/aiglx`).

The goal of the Accelerated Indirect GLX project (AIGLX) is to add 3D effects to everyday desktop systems. It does this by implementing OpenGL (`http://opengl.org`) accelerated effects using the Mesa (`http://www.mesa3d.org`) open source OpenGL implementation.

Currently, AIGLX supports a limited set of video cards and implements only a few 3D effects, but it does offer some insight into the eye candy that is in the works.

If your video card was properly detected and configured, you may be able simply to turn on the Desktop Effects feature to see the effects that have been implemented so far. To

turn on Desktop Effects, select System ➪ Preferences ➪ Desktop Effects. When the Desktop Effects window appears, select Compiz. (If the selection is not available, install the compiz package.)

Enabling Compiz does the following:

Starts Compiz Stops the current window manager and starts the Compiz window manager.

Enables the Windows Wobble When Moved effect With this effect on, when you grab the title bar of the window to move it, the window wobbles as it moves. Menus and other items that open on the desktop also wobble.

Enables the Workspaces on a Cube effect Drag a window from the desktop to the right or the left, and the desktop rotates like a cube, with each of your desktop workspaces appearing as a side of that cube. Drop the window on the workspace where you want it to go. You can also click the Workspace Switcher applet in the bottom panel to rotate the cube to display different workspaces.

Other nice desktop effects result from using the Alt+Tab keys to tab among different running windows. As you press Alt+Tab, a thumbnail of each window scrolls across the screen as the window it represents is highlighted.

Figure 2.19 shows an example of a Compiz desktop with AIGLX enabled. The figure illustrates a web browser window being moved from one workspace to another as those workspaces rotate on a cube.

FIGURE 2.19

Rotate workspaces on a cube with AIGLX desktop effects enabled.



The following are some interesting effects that you can get with your 3D AIGLX desktop:

Spin cube Hold Ctrl+Alt keys and press the right and left arrow keys. The desktop cube spins to each successive workspace (forward or back).

Slowly rotate cube Hold the Ctrl+Alt keys, press and hold the left mouse button, and move the mouse around on the screen. The cube moves slowly with the mouse among the workspaces.

Scale and separate windows If your desktop is cluttered, hold Ctrl+Alt and press the up arrow key. Windows shrink down and separate on the desktop. Still holding Ctrl+Alt, use your arrow keys to highlight the window you want and release the keys to have that window come to the surface.

Tab through windows Hold the Alt key and press the Tab key. You will see reduced versions of all your windows in a strip in the middle of your screen, with the current window highlighted in the middle. Still holding the Alt key, press Tab or Shift+Tab to move forward or backward through the windows. Release the keys when the one you want is highlighted.

Scale and separate workspaces Hold Ctrl+Alt and press the down arrow key to see reduced images of the workspace shown on a strip. Still holding Ctrl+Alt, use the right and left arrow keys to move among the different workspaces. Release the keys when the workspace you want is highlighted.

Send current window to next workspace Hold Ctrl+Alt+Shift keys together and press the left and right arrow keys. The next workspace to the left or right, respectively, appears on the current desktop.

Slide windows around Press and hold the left mouse button on the window title bar, and then press the left, right, up, or down arrow key to slide the current window around on the screen.

If you get tired of wobbling windows and spinning cubes, you can easily turn off the AIGLX 3D effects and return to using Metacity as the window manager. Select System ➪ Preferences ➪ Desktop Effects again, and toggle off the Enable Desktop Effects button to turn off the feature.

If you have a supported video card but find that you cannot turn on the Desktop Effects, check that your X server started properly. In particular, make sure that your `/etc/X11/xorg.conf` file is properly configured. Make sure that `dri` and `glx` are loaded in the Module section. Also, add an extensions section anywhere in the file (typically at the end of the file) that appears as follows:

```
Section "extensions"
 Option "Composite"
EndSection
```

Another option is to add the following line to the `/etc/X11/xorg.conf` file in the Device section:

```
Option "XAANoOffscreenPixmaps"
```

The `XAANoOffscreenPixmaps` option improves performance. Check your `/var/log/Xorg.log` file to make sure that DRI and AIGLX features were started correctly. The messages in that file can help you debug other problems as well.

Summary

The GNOME desktop environment has become the default desktop environment for many Linux systems, including Fedora and RHEL. The GNOME 3 desktop (now used in Fedora and RHEL 7 and RHEL 8) is a modern, elegant desktop, designed to match the types of interfaces available on many of today's mobile devices. The GNOME 2 desktop (used through RHEL 6) provides a more traditional desktop experience.

Besides GNOME desktops, you can try out other popular and useful desktop environments. The K Desktop Environment (KDE) offers many more bells and whistles than GNOME, and it is used by default in several Linux distributions. Netbooks and live CD distributions sometimes use the LXDE or Xfce desktops.

Now that you have a grasp of how to get and use a Linux desktop, it's time to start digging into the more professional administrative interfaces. Chapter 3, "Using the Shell," introduces you to the Linux command-line shell interface.

Exercises

Use these exercises to test your skill in using a GNOME desktop. You can use either a GNOME 2.*x* (Red Hat Enterprise Linux up until RHEL 6.*x*) or GNOME 3.*x* (Fedora 16 or later or Ubuntu up to 11.10, or later using the Ubuntu GNOME project) desktop. If you are stuck, solutions to the tasks for both the GNOME 2 and GNOME 3 desktops are shown in Appendix B.

1. Obtain a Linux system with either a GNOME 2 or GNOME 3 desktop available. Start the system and log in to a GNOME desktop.
2. Launch the Firefox web browser and go to the GNOME home page (`http://gnome.org`).
3. Pick a background you like from the GNOME art site (`http://gnome-look.org`), download it to your Pictures folder, and select it as your current background.
4. Start a Nautilus File Manager window and move it to the second workspace on your desktop.
5. Find the image that you downloaded to use as your desktop background and open it in any image viewer.

6. Move back and forth between the workspace with Firefox on it and the one with the Nautilus file manager.
7. Open a list of applications installed on your system and select an image viewer to open from that list. Use as few clicks or keystrokes as possible.
8. Change the view of the windows on your current workspace to smaller views you can step through. Select any window you'd like to make it your current window.
9. From your desktop, using only the keyboard, launch a music player.
10. Take a picture of your desktop, using only keystrokes.

Part II

Becoming a Linux Power User

IN THIS PART

Chapter 3
Using the Shell

Chapter 4
Moving Around the Filesystem

Chapter 5
Working with Text Files

Chapter 6
Managing Running Processes

Chapter 7
Writing Simple Shell Scripts

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER 3

Using the Shell

IN THIS CHAPTER

Understanding the Linux shell

Using the shell from consoles or Terminals

Using commands

Using command history and tab completion

Connecting and expanding commands

Understanding variables and aliases

Making shell settings permanent

Using man pages and other documentation

Before icons and windows took over computer screens, you typed commands to interact with most computers. On UNIX systems, from which Linux was derived, the program used to interpret and manage commands was referred to as the *shell*.

No matter which Linux distribution you are using, you can always count on the fact that the shell is available to you. It provides a way to create executable script files, run programs, work with filesystems, compile computer code, and manage the computer. Although the shell is less intuitive than common graphical user interfaces (GUIs), most Linux experts consider the shell to be much more powerful than GUIs. Shells have been around a long time, and many advanced features that aren't available from the desktop can be accessed by running shell commands.

The Linux shell illustrated in this chapter is called the *bash shell*, which stands for Bourne Again Shell. The name is derived from the fact that bash is compatible with the one of the earliest UNIX shells: the Bourne shell (named after its creator, Stephen Bourne, and represented by the `sh` command).

Although bash is included with most distributions and considered a standard, other shells are available, including the C shell (`csh`), which is popular among BSD UNIX users, and the Korn shell (`ksh`), which is popular among UNIX System V users. Ubuntu uses the dash shell by default at boot time, which is designed to perform faster than the bash shell. Linux also has a `tcsh` shell (an improved C shell) and an ash shell (another Bourne shell look-alike).

The odds are strong that the Linux distribution you are using has more than one shell available for your use. This chapter, however, focuses primarily on the bash shell. That is because the Linux

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

distributions featured in this book, Fedora, Ubuntu, and Red Hat Enterprise Linux, all use the bash shell by default when you open a Terminal window.

The following are a few major reasons to learn how to use the shell:

- *You will learn to get around any Linux or other UNIX-like system*. For example, I can log in to my Red Hat Enterprise Linux web server, my home multimedia server, my home router, or my wife's Mac and explore and use any of those computer systems from a shell. I can even log in and run commands on my Android phone. They all run Linux or similar systems on the inside.
- *Special shell features enable you to gather data input and direct data output between commands and Linux filesystems*. To save on typing, you can find, edit, and repeat commands from your shell history. Many power users hardly touch a graphical interface, doing most of their work from a shell.
- *You can gather commands into a file using programming constructs such as conditional tests, loops, and case statements to perform complex operations quickly, which would be difficult to retype over and over*. Programs consisting of commands that are stored and run from a file are referred to as *shell scripts*. Many Linux system administrators use shell scripts to automate tasks such as backing up data, monitoring log files, or checking system health.

The shell is a command language interpreter. If you have used Microsoft operating systems, you'll see that using a shell in Linux is similar to, but generally much more powerful than, the PowerShell interpreter used to run commands. You can happily use Linux from a graphical desktop interface, but as you grow into Linux you will surely need to use the shell at some point to track down a problem or administer some features.

How to use the shell isn't obvious at first, but with the right help you can quickly learn many of the most important shell features. This chapter is your guide to working with the Linux system commands, processes, and filesystem from the shell. It describes the shell environment and helps you tailor it to your needs.

About Shells and Terminal Windows

There are several ways to get to a shell interface in Linux. Three of the most common are the shell prompt, Terminal window, and virtual console, which you learn more about in the following sections.

To start, boot up your Linux system. On your screen, you should see either a graphical login screen or a plain-text login prompt similar to the following:

```
Red Hat Enterprise Linux Server release 8.0 (Ootpa)
Kernel 4.18.0-42.el8.x86_64 on an X86
mylinuxhost login:
```

In either case, you should log in with a regular user account. If you have a plain-text login prompt, continue to the next section, "Using the shell prompt." If you log in through a graphical screen, go to the section "Using a Terminal window" to see how to access a shell

from the desktop. In either case, you can access more shells as described in the section "Using virtual consoles," which appears shortly in this chapter.

Using the shell prompt

If your Linux system has no graphical user interface (or one that isn't working at the moment), you will most likely see a shell prompt after you log in. Typing commands from the shell will probably be your primary means of using the Linux system.

The default prompt for a regular user is simply a dollar sign:

```
$
```

The default prompt for the root user is a pound sign (also called a *number sign* or a *hash tag*):

```
#
```

In most Linux systems, the `$` and `#` prompts are preceded by your username, system name, and current directory name. For example, a login prompt for the user named `jake` on a computer named `pine` with `/usr/share/` as the current working directory would appear as follows:

```
[jake@pine share]$
```

You can change the prompt to display any characters you like and even read in pieces of information about your system. For example, you can use the current working directory, the date, the local computer name, or any string of characters as your prompt. To configure your prompt, see the section "Setting your prompt" later in this chapter.

Although a tremendous number of features are available with the shell, it's easy to begin by just entering a few commands. Try some of the commands shown in the remaining sections to become familiar with your current shell environment.

In the examples that follow, the dollar ($) and pound (#) symbols indicate a prompt. A `$` indicates that the command can be run by any user, but a # typically means that you should run the command as the root user; that is, many administrative tools require root permission to be able to run them. The prompt is followed by the command that you type (and then press Enter). The lines that follow show the output resulting from the command.

Note

Although we use # to indicate that a command be run as the root user, you do not need to log in as the root user to run a command as root. In fact, the most common way to run a command as a root user is to use the `sudo` command. See Chapter 8, "Learning System Administration," for further information about the `sudo` command.

Using a Terminal window

With the desktop GUI running, you can open a Terminal emulator program (sometimes referred to as a Terminal window) to start a shell. Most Linux distributions make it easy for

you to get to a shell from the GUI. Here are two common ways to launch a Terminal window from a Linux desktop:

Right-click the desktop. In the context menu that appears, if you see Open in Terminal, Shells, New Terminal, Terminal Window, Xterm, or some similar item, select it to start a Terminal window. (Some distributions have disabled this feature.)

Click the panel menu. Many Linux desktops include a panel at the top or bottom of the screen from which you can launch applications. For example, in some systems that use the GNOME 2 desktop, you can select Applications ➪ System Tools ➪ Terminal to open a Terminal window. In GNOME 3, click the Activities menu, type **`Terminal`**, and press Enter.

In all cases, you should be able to type a command as you would from a shell with no GUI. Different Terminal emulators are available with Linux. In Fedora, Red Hat Enterprise Linux (RHEL), and other Linux distributions that use the GNOME desktop, the default Terminal emulator window is the GNOME Terminal (started by the `gnome-terminal` command).

GNOME Terminal supports many features beyond the basic shell. For example, you can cut and paste text to or from a GNOME Terminal window, change fonts, set a title, choose colors or images to use as background, and set how much text to save when text scrolls off the screen.

To try some GNOME Terminal features, start up a Fedora or RHEL system and log in to the desktop. Then follow this procedure:

1. Select Applications ➪ Utilities ➪ Terminal (or click on the Activities menu and type **`Terminal`**). A Terminal window should open on your desktop.
2. Select Edit ➪ Profile Preferences or Preferences.
3. On the General tab or current profile (depending on your version of GNOME), check the "Custom font" box.
4. Select the Font field, try a different font and size, and then click Select. The new font appears in the Terminal window.
5. Unselect the "Custom font" box. This takes you back to the original font.
6. On the Colors tab, clear the "Use colors from system theme" check box. From here, you can try some different font and background colors.
7. Re-select the "Use colors from system theme" box to go back to the default colors.
8. Go to your Profile window. There are other features with which you may want to experiment, such as setting how much scrolled data is kept.
9. Close the Profile window when you are finished. You are now ready to use your Terminal window.

If you are using Linux from a graphical desktop, you will probably most often access the shell from a Terminal window.

Using virtual consoles

Most Linux systems that include a desktop interface start multiple virtual consoles running on the computer. Virtual consoles are a way to have multiple shell sessions open at once in addition to the graphical interface you are using.

You can switch between virtual consoles by holding the Ctrl and Alt keys and pressing a function key between F1 and F6. For example, in Fedora, press Ctrl+Alt+F1 (or F2, F3, F4, and so on up to F6 on most Linux systems) to display one of seven virtual consoles. The GUI is typically located on one of the first two virtual consoles, and the other six virtual consoles are typically text-based virtual consoles.

You can return to the GUI (if one is running) by pressing Ctrl+Alt+F1. On some systems, the GUI may run on a different virtual console, such as virtual console 2 (Ctrl+Alt+F2). Newer systems, such as Fedora 29, now start the gdm (the login screen) persistently on tty1 to allow multiple simultaneous GUI sessions: the gdm is on tty1, the first desktop is started on tty2, the second desktop is started on tty3, and so on.

Try it right now. Hold down the Ctrl+Alt keys and press F3. You should see a plain-text login prompt. Log in using your username and password. Try a few commands. When you are finished, type **`exit`** to exit the shell and then press Ctrl+Alt+F1 or Ctrl+Alt+F2 to return to your graphical desktop interface. You can go back and forth between these consoles as much as you like.

Choosing Your Shell

In most Linux systems, your default shell is the bash shell. To find out what is your default login shell, enter the following commands:

```
$ who am i
chris    pts/0        2019-10-21 22:45 (:0.0)
$ grep chris /etc/passwd
chris:x:13597:13597:Chris Negus:/home/chris:/bin/bash
```

Notice that the command-line examples shown here and throughout the book show the command followed by output from that command. When the command completes, you are presented with the command prompt again.

The `who am i` command shows your username, and the `grep` command (replacing `chris` with your username) shows the definition of your user account in the `/etc/passwd` file. The last field in that entry shows that the bash shell (`/bin/bash`) is your default shell (the one that starts up when you log in or open a Terminal window).

It's possible, although not likely, that you might have a different default shell set. To try a different shell, simply type the name of that shell (examples include `ksh`, `tcsh`, `csh`, `sh`, `dash`, and others, assuming that they are installed). You can try a few commands in that shell and type **`exit`** when you are finished to return to the bash shell.

You might choose to use different shells for the following reasons:

- You are used to using UNIX System V systems (often `ksh` by default) or Sun Microsystems and other Berkeley UNIX-based distributions (frequently `csh` by default), and you are more comfortable using default shells from those environments.
- You want to run shell scripts that were created for a particular shell environment, and you need to run the shell for which they were made so that you can test or use those scripts from your current shell.
- You simply prefer features in one shell over those in another. For example, a member of my Linux Users Group prefers `ksh` over `bash` because he doesn't like the way aliases are used with `bash`.

Although most Linux users have a preference for one shell or another, when you know how to use one shell, you can quickly learn any of the others by occasionally referring to the shell's man page (for example, type **`man bash`**). The man pages (described later in the section "Getting Information about Commands") provide documentation for commands, file formats, and other components in Linux. Most people use bash just because they don't have a particular reason for using a different shell. The rest of this chapter describes the bash shell.

Bash includes features originally developed for `sh` and `ksh` shells in early UNIX systems, as well as some `csh` features. Expect bash to be the default login shell in most Linux systems that you are using, with the exception of some specialized Linux systems (such as some that run on embedded devices) that may require a smaller shell that needs less memory and requires fewer features. Most of the examples in this chapter are based on the bash shell.

TIP

The bash shell is worth knowing not only because it is the default in most installations, but because it is the one you will use with most Linux certification exams.

Running Commands

The simplest way to run a command is just to type the name of the command from a shell. From your desktop, open a Terminal window. Then enter the following command:

```
$ date
Thu Jun 29 08:14:53 EDT 2019
```

Entering the `date` command, with no options or arguments, causes the current day, month, date, time, time zone, and year to be displayed as just shown.

Here are a few other commands you can try:

```
$ pwd
/home/chris
$ hostname
```

```
mydesktop
$ ls
Desktop    Downloads  Pictures  Templates
Documents  Music      Public    Videos
```

The `pwd` command shows your current working directory. Entering `hostname` shows your computer's hostname. The `ls` command lists the files and directories in your current directory. Although many commands can be run by just entering command names, it's more common to type other characters after the command to modify its behavior. The characters and words that you can type after a command are called *options* and *arguments*.

Understanding command syntax

Most commands have one or more *options* that you can add to change the command's behavior. Options typically consist of a single letter preceded by a hyphen. However, you can group single-letter options together or precede each with a hyphen to use more than one option at a time. For example, the following two uses of options for the `ls` command are the same:

```
$ ls -l -a -t
$ ls -lat
```

In both cases, the `ls` command is run with the `-l` (long listing), `-a` (show hidden dot files), and `-t` options (list by time).

Some commands include options that are represented by a whole word. To tell a command to use a whole word as an option, you typically precede it with a double hyphen (`--`). For example, to use the help option on many commands, you enter `--help` on the command line. Without the double hyphen, the letters `h`, `e`, `l`, and `p` would be interpreted as separate options. There are some commands that don't follow the double hyphen convention, using a single hyphen before a word, but most commands use double hyphens for word options.

NOTE

You can use the `--help` option with most commands to see the options and arguments that they support. For example, try typing **`hostname --help`**.

Many commands also accept arguments after certain options are entered or at the end of the entire command line. An *argument* is an extra piece of information, such as a filename, directory, username, device, or other item, that tells the command what to act on. For example, `cat /etc/passwd` displays the contents of the `/etc/passwd` file on your screen. In this case, `/etc/passwd` is the argument. Usually, you can have as many arguments as you want on the command line, limited only by the total number of characters allowed on a command line. Sometimes, an argument is associated with an option. In that case, the argument must immediately follow the option. With single-letter options, the

argument typically follows after a space. For full-word options, the argument often follows an equal sign (=). Here are some examples:

```
$ ls --hide=Desktop
Documents  Music     Public    Videos
Downloads  Pictures  Templates
```

In the previous example, the `--hide` option tells the `ls` command not to display the file or directory named `Desktop` when listing the contents of the directory. Notice that the equal sign immediately follows the option (no space) and then the argument (again, no space).

Here's an example of a single-letter option that is followed by an argument:

```
$ tar -cvf backup.tar /home/chris
```

In the `tar` example just shown, the options say to create (`c`) a file (`f`) named `backup.tar` that includes all of the contents of the `/home/chris` directory and its subdirectories and show verbose (`v`) messages as the backup is created. Because `backup.tar` is an argument to the `f` option, `backup.tar` must immediately follow the option.

Here are a few commands that you can try out. See how they behave differently with different options:

```
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates
Videos
$ ls -a
.              Desktop     .gnome2_private  .lesshst       Public
..             Documents   .gnote           .local         Templates
.bash_history  Downloads   .gnupg           .mozilla       Videos
.bash_logout   .emacs      .gstreamer-0.10  Music          .xsession-
errors
.bash_profile  .esd_auth   .gtk-bookmarks   Pictures       .zshrc
.bashrc        .fsync.log  .gvfs            Pictures
$ uname
Linux
$ uname -a
Linux mydesktop 5.3.7-301.fc31.x86_64 #1 SMP Mon Oct 21 19:18:58 UTC
     2019 x86_64 x86_64 x86_64 GNU/Linux
$ date
Wed 04 Mar 2020 09:06:25 PM EST
$ date +'%d/%m/%y'
04/03/20
$ date +'%A, %B %d, %Y'
Wednesday, March 04, 2020
```

The `ls` command, by itself, shows all regular files and directories in the current directory. By adding the `-a`, you can also see the hidden files in the directory (those beginning with a dot). The `uname` command shows the type of system you are running (Linux). When you add `-a`, you also can see the hostname, kernel release, and kernel version.

The `date` command has some special types of options. By itself, `date` simply prints the current day, date, and time as shown above. But the `date` command supports a special `+` format option, which lets you display the date in different formats. Enter **`date --help`** to see different format indicators you can use.

Try the `id` and `who` commands to get a feel for your current Linux environment, as described in the following paragraphs.

When you log in to a Linux system, Linux views you as having a particular identity, which includes your username, group name, user ID, and group ID. Linux also keeps track of your login session: It knows when you logged in, how long you have been idle, and where you logged in from.

To find out information about your identity, use the `id` command as follows:

```
$ id
uid=1000(chris) gid=1000(chris) groups=1005(sales), 7(lp)
```

In this example, the username is `chris`, which is represented by the numeric user ID (uid) `1000`. The primary group for chris also is called `chris`, which has a group ID (gid) of `1000`. It is normal for Fedora and Red Hat Enterprise Linux users to have the same primary group name as their username. The user `chris` also belongs to other groups called `sales` (gid `1005`) and `lp` (gid `7`). These names and numbers represent the permissions that `chris` has to access computer resources.

Note

Linux distributions that have Security Enhanced Linux (SELinux) enabled, such as Fedora and RHEL, show additional information at the end of the `id` output. That output might look something like the following:

```
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

SELinux provides a means of tightly locking down the security of a Linux system. See Chapter 24, "Enhancing Linux Security with SELinux," if you want to learn about SELinux.

You can see information about your current login session by using the `who` command. In the following example, the `-u` option says to add information about idle time and the process ID and `-H` asks that a header be printed:

```
$ who -uH
NAME       LINE     TIME             IDLE      PID    COMMENT
chris      tty1     Jan 13 20:57     .         2019
```

The output from this `who` command shows that the user `chris` is logged in on `tty1` (which is the first virtual console on the monitor connected to the computer) and his login session began at 20:57 on January 13. The `IDLE` time shows how long the shell has been open without any command being typed (the dot indicates that it is currently active). `PID` shows the process ID of the user's login shell. `COMMENT` would show the name of the remote computer from which the user had logged in, if that user had logged in from

another computer on the network, or the name of the local X display if that user were using a Terminal window (such as :0.0).

Locating commands

Now that you have typed a few commands, you may wonder where those commands are located and how the shell finds the commands you type. To find commands you type, the shell looks in what is referred to as your *path*. For commands that are not in your path, you can type the complete identity of the location of the command.

If you know the directory that contains the command that you want to run, one way to run it is to type the full, or absolute, path to that command. For example, you run the `date` command from the `/bin` directory by entering the following:

```
$ /bin/date
```

Of course, this can be inconvenient, especially if the command resides in a directory with a long pathname. The better way is to have commands stored in well-known directories and then add those directories to your shell's `PATH` environment variable. The path consists of a list of directories that are checked sequentially for the commands you enter. To see your current path, enter the following:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:↵
/home/chris/bin
```

The results show a common default path for a regular Linux user. Directories in the path list are separated by colons. Most user commands that come with Linux are stored in the `/bin`, `/usr/bin`, or `/usr/local/bin` directory. The `/sbin` and `/usr/sbin` directories contain administrative commands (some Linux systems don't put those directories in regular users' paths). The last directory shown is the `bin` directory in the user's `home` directory (`/home/chris/bin`).

Tip

If you want to add your own commands or shell scripts, place them in the `bin` directory in your home directory (such as `/home/chris/bin` for the user named `chris`). This directory is automatically added to your path in some Linux systems, although you may need to create that directory or add it to your `PATH` on other Linux systems. So, as long as you add the command to your `bin` with execute permission, you can begin using it by simply typing the command name at your shell prompt. To make commands available to all users, add them to `/usr/local/bin`.

Unlike some other operating systems, Linux does not, by default, check the current directory for an executable before searching the path. It immediately begins searching the path, and executables in the current directory are run only if they are in the `PATH` variable or you give their absolute (such as `/home/chris/scriptx.sh`) or relative (for example, `./scriptx.sh`) location.

The path directory order is important. Directories are checked from left to right. So, in this example, if there is a command called `foo` located in both the `/usr/bin` and `/bin` directories, the one in `/usr/bin` is executed. To have the other `foo` command run, you either type the full path to the command or change your `PATH` variable. (Changing your `PATH` and adding directories to it are described later in this chapter.)

Not all of the commands you run are located in directories in your `PATH` variable. Some commands are built into the shell. Other commands can be overridden by creating aliases that define any commands and options that you want the command to run. There are also ways of defining a function that consists of a stored series of commands. Here is the order in which the shell checks for the commands you type:

1. **Aliases**. These are names set by the `alias` command that represent a particular command and a set of options. Type **`alias`** to see what aliases are set. Often, aliases enable you to define a short name for a long, complicated command. (I describe how to create your own aliases later in this chapter.)
2. **Shell reserved word**. These are words reserved by the shell for special use. Many of these are words that you would use in programming-type functions, such as `do`, `while`, `case`, and `else`. (I cover some of these reserved words in Chapter 7, "Writing Simple Shell Scripts.")
3. **Function**. This is a set of commands that is executed together within the current shell.
4. **Built-in command**. This is a command built into the shell. As a result, there is no representation of the command in the filesystem. Some of the most common commands that you will use are shell built-in commands, such as `cd` (to change directories), `echo` (to output text to the screen), `exit` (to exit from a shell), `fg` (to bring a command running in the background to the foreground), `history` (to see a list of commands that were previously run), `pwd` (to list the present working directory), `set` (to set shell options), and `type` (to show the location of a command).
5. **Filesystem command**. This command is stored in and executed from the computer's filesystem. (These are the commands that are indicated by the value of the `PATH` variable.)

To determine the location of a particular command, you can use the `type` command. (If you are using a shell other than `bash`, use the `which` command instead.) For example, to find out where the bash shell command is located, enter the following:

```
$ type bash
bash is /bin/bash
```

Try these few words with the `type` command to see other locations of commands: `which`, `case`, and `return`. If a command resides in several locations, you can add the `-a` option to have all of the known locations of the command printed. For example, the command `type -a ls` should show an aliased and filesystem location for the `ls` command.

Tip

Sometimes, you run a command and receive an error message that the command was not found or that permission to run the command was denied. If the command was not found, check that you spelled the command correctly and that it is located in your `PATH` variable. If permission to run the command was denied, the command may be in the `PATH` variable but may not be executable. Also remember that case is important, so typing CAT or Cat will not find the `cat` command.

If a command is not in your `PATH` variable, you can use the `locate` command to try to find it. Using `locate`, you can search any part of the system that is accessible to you. (Some files are only accessible to the root user.) For example, if you wanted to find the location of the `chage` command, you could enter the following:

```
$ locate chage
/usr/bin/chage
/usr/sbin/lchage
/usr/share/man/fr/man1/chage.1.gz
/usr/share/man/it/man1/chage.1.gz
/usr/share/man/ja/man1/chage.1.gz
/usr/share/man/man1/chage.1.gz
/usr/share/man/man1/lchage.1.gz
/usr/share/man/pl/man1/chage.1.gz
/usr/share/man/ru/man1/chage.1.gz
/usr/share/man/sv/man1/chage.1.gz
/usr/share/man/tr/man1/chage.1.gz
```

Notice that `locate` not only found the `chage` command, it also found the `lchage` command and a variety of man pages associated with `chage` for different languages. The `locate` command looks all over your filesystem, not just in directories that contain commands. (If `locate` does not find files recently added to your system, run `updatedb` as root to update the locate database.)

In the coming chapters, you learn to use additional commands. For now, I want you to become more familiar with how the shell itself works. So next I discuss features for recalling commands, completing commands, using variables, and creating aliases.

Recalling Commands Using Command History

Being able to repeat a command you ran earlier in a shell session can be convenient. Recalling a long and complex command line that you mistyped can save you some trouble. Fortunately, some shell features enable you to recall previous command lines, edit those lines, or complete a partially typed command line.

The *shell history* is a list of the commands that you have entered before. Using the `history` command in a bash shell, you can view your previous commands. Then using various

shell features, you can recall individual command lines from that list and change them however you please.

The rest of this section describes how to do command-line editing, how to complete parts of command lines, and how to recall and work with the history list.

Command-line editing

If you type something wrong on a command line, the bash shell ensures that you don't have to delete the entire line and start over. Likewise, you can recall a previous command line and change the elements to make a new command.

By default, the bash shell uses command-line editing that is based on the emacs text editor. (Type **`man emacs`** to read about it, if you care to do so.) If you are familiar with emacs, you probably already know most of the keystrokes described here.

Tip

If you prefer the `vi` command for editing shell command lines, you can easily make that happen. Add the following line to the `.bashrc` file in your home directory:

```
set -o vi
```

The next time you open a shell, you can use `vi` commands to edit your command lines.

To do the editing, you can use a combination of control keys, meta keys, and arrow keys. For example, Ctrl+F means to hold down the Ctrl key, and type **f**. Alt+F means to hold down the Alt key, and type **f**. (Instead of the Alt key, your keyboard may use a Meta key or the Esc key. On a Windows keyboard, you can use the Windows key.)

To try out a bit of command-line editing, enter the following:

```
$ ls /usr/bin | sort -f | less
```

This command lists the contents of the `/usr/bin` directory, sorts the contents in alphabetical order (regardless of case), and pipes the output to `less`. The `less` command displays the first page of output, after which you can go through the rest of the output a line (press Enter) or a page (press spacebar) at a time. Simply press **`q`** when you are finished. Now, suppose that you want to change `/usr/bin` to `/bin`. You can use the following steps to change the command:

1. **Press the up arrow (↑) key**. This displays the most recent command from your shell history.
2. **Press Ctrl+A**. This moves the cursor to the beginning of the command line.
3. **Press Ctrl+F or the right arrow (→) key**. Repeat this command a few times to position the cursor under the first slash (`/`).
4. **Press Ctrl+D**. Type this command four times to delete `/usr` from the line.

5. **Press Enter**. This executes the command line.

As you edit a command line, at any point you can type regular characters to add those characters to the command line. The characters appear at the location of your text cursor. You can use right → and left ← arrows to move the cursor from one end to the other on the command line. You can also press the up ↑ and down ↓ arrow keys to step through previous commands in the history list to select a command line for editing. (See the section "Command-line recall" for details on how to recall commands from the history list.) You can use many keystrokes to edit your command lines. Table 3.1 lists the keystrokes that you can use to move around the command line.

TABLE 3.1 Keystrokes for Navigating Command Lines

Keystroke	Full Name	Meaning
Ctrl+F	Character forward	Go forward one character.
Ctrl+B	Character backward	Go backward one character.
Alt+F	Word forward	Go forward one word.
Alt+B	Word backward	Go backward one word.
Ctrl+A	Beginning of line	Go to the beginning of the current line.
Ctrl+E	End of line	Go to the end of the line.
Ctrl+L	Clear screen	Clear screen and leave line at the top of the screen.

The keystrokes in Table 3.2 can be used to edit command lines.

TABLE 3.2 Keystrokes for Editing Command Lines

Keystroke	Full Name	Meaning
Ctrl+D	Delete current	Delete the current character.
Backspace	Delete previous	Delete the previous character.
Ctrl+T	Transpose character	Switch positions of current and previous characters.
Alt+T	Transpose words	Switch positions of current and previous words.
Alt+U	Uppercase word	Change the current word to uppercase.
Alt+L	Lowercase word	Change the current word to lowercase.
Alt+C	Capitalize word	Change the current word to an initial capital.
Ctrl+V	Insert special character	Add a special character. For example, to add a Tab character, press Ctrl+V+Tab.

Use the keystrokes in Table 3.3 to cut and paste text on a command line.

TABLE 3.3 Keystrokes for Cutting and Pasting Text from within Command Lines

Keystroke	Full Name	Meaning
Ctrl+K	Cut end of line	Cut text to the end of the line.
Ctrl+U	Cut beginning of line	Cut text to the beginning of the line.
Ctrl+W	Cut previous word	Cut the word located behind the cursor.
Alt+D	Cut next word	Cut the word following the cursor.
Ctrl+Y	Paste recent text	Paste most recently cut text.
Alt+Y	Paste earlier text	Rotate back to previously cut text and paste it.
Ctrl+C	Delete whole line	Delete the entire line.

Command-line completion

To save you a few keystrokes, the bash shell offers several different ways of completing partially typed values. To attempt to complete a value, type the first few characters and press Tab. Here are some of the values you can type partially from a bash shell:

Command, alias, or function If the text you type begins with regular characters, the shell tries to complete the text with a command, alias, or function name.

Variable If the text you type begins with a dollar sign ($), the shell completes the text with a variable from the current shell.

Username If the text you type begins with a tilde (~), the shell completes the text with a username. As a result, *~username* indicates the home directory of the named user.

Hostname If the text you type begins with the at symbol (@), the shell completes the text with a hostname taken from the `/etc/hosts` file.

Tip

To add hostnames from an additional file, you can set the `HOSTFILE` variable to the name of that file. The file must be in the same format as `/etc/hosts`.

Here are a few examples of command completion. (When you see *<Tab>*, it means to press the Tab key on your keyboard.) Enter the following:

```
$ echo $OS<Tab>
$ cd ~ro<Tab>
$ userm<Tab>
```

The first example causes `$OS` to expand to the `$OSTYPE` variable. In the next example, `~ro` expands to the root user's home directory (`~root/`). Next, `userm` expands to the `usermod` command.

Pressing Tab twice offers some wonderful possibilities. Sometimes, several possible completions for the string of characters you have entered are available. In those cases, you can check the possible ways that text can be expanded by pressing Tab twice at the point where you want to do completion.

The following shows the result you would get if you checked for possible completions on `$P`:

```
$ echo $P<Tab><Tab>
$PATH $PPID $PS1 $PS2 $PS4 $PWD
$ echo $P
```

In this case, there are six possible variables that begin with `$P`. After possibilities are displayed, the original command line returns, ready for you to complete it as you choose. For example, if you typed another P and hit Tab again, the command line would be completed with `$PPID` (the only unique possibility).

Command-line recall

After you type a command, the entire command line is saved in your shell's history list. The list is stored in the current shell until you exit the shell. After that, it is written to a history file, from which any command can be recalled to be run again in your next session. After a command is recalled, you can modify the command line, as described earlier.

To view your history list, use the `history` command. Enter the command without options or followed by a number to list that many of the most recent commands. For example:

```
$ history 8
 382 date
 383 ls /usr/bin | sort -a | more
 384 man sort
 385 cd /usr/local/bin
 386 man more
 387 useradd -m /home/chris -u 101 chris
 388 passwd chris
 389 history 8
```

A number precedes each command line in the list. You can recall one of those commands using an exclamation point (!). Keep in mind that when an exclamation point is used, the command runs blind without presenting an opportunity to confirm the command you're referencing. There are several ways to run a command immediately from this list, including the following:

!*n* *Run command number.* Replace the *n* with the number of the command line and that line is run. For example, here's how to repeat the `date` command shown as command number 382 in the preceding history listing:

```
$ !382
```

```
date
Fri Jun 29 15:47:57 EDT 2019
```

!!—!! *Run previous command*. Runs the previous command line. Here's how you would immediately run that same `date` command:

```
$ !!
date
Fri Jun 29 15:53:27 EDT 2019
```

!?*string*—? *Run command containing string*. This runs the most recent command that contains a particular string of characters. For example, you can run the `date` command again by just searching for part of that command line as follows:

```
$ !?dat?

date
Fri Jun 29 16:04:18 EDT 2019
```

Instead of just running a `history` command line immediately, you can recall a particular line and edit it. You can use the following keys or key combinations to do that, as shown in Table 3.4.

Another way to work with your history list is to use the `fc` command. Type **`fc`** followed by a history line number, and that command line is opened in a text editor (`vi` by default, type **`:wq`** to save and exit or **`:q!`** to just exit if you are stuck in `vi`). Make the changes that you want. When you exit the editor, the command runs. You can also give a range of line

TABLE 3.4 Keystrokes for Using Command History

Key(s)	Function Name	Description
Arrow keys (↑ and ↓)	Step	Press the up and down arrow keys to step through each command line in your history list to arrive at the one you want. (Ctrl+P and Ctrl+N do the same functions, respectively.)
Ctrl+R	Reverse incremental search	After you press these keys, you enter a search string to do a reverse search. As you type the string, a matching command line appears that you can run or edit.
Ctrl+S	Forward incremental search	This is the same as the preceding function but for forward search. (It may not work in all instances.)
Alt+P	Reverse search	After you press these keys, you enter a string to do a reverse search. Type a string and press Enter to see the most recent command line that includes that string.
Alt+N	Forward search	This is the same as the preceding function but for forward search. (It may not work in all instances.)

numbers (for example, `fc 100 105`). All of the commands open in your text editor and then run one after the other when you exit the editor.

After you close your shell, the history list is stored in the `.bash _ history` file in your home directory. Up to 1,000 history commands are stored for you by default.

NOTE

Some people disable the history feature for the root user by setting the `HISTFILE` shell variable to `/dev/null` or simply leaving `HISTSIZE` blank. This prevents information about the root user's activities from potentially being exploited. If you are an administrative user with root privileges, you may want to consider emptying your file upon exiting as well for the same reasons. Also, because shell history is stored permanently when the shell exits properly, you can prevent storing a shell's history by killing a shell. For example, to kill a shell with process ID 1234, type **`kill -9 1234`** from any shell.

Connecting and Expanding Commands

A truly powerful feature of the shell is the capability to redirect the input and output of commands to and from other commands and files. To allow commands to be strung together, the shell uses metacharacters. A *metacharacter* is a typed character that has special meaning to the shell for connecting commands or requesting expansion.

Metacharacters include the pipe character (|), ampersand (&), semicolon (;), right parenthesis ()), left parenthesis (), less than sign (<), and greater than sign (>). The next sections describe how to use metacharacters on the command line to change how commands behave.

Piping between commands

The pipe (|) metacharacter connects the output from one command to the input of another command. This lets you have one command work on some data and then have the next command deal with the results. Here is an example of a command line that includes pipes:

```
$ cat /etc/passwd | sort | less
```

This command lists the contents of the `/etc/passwd` file and pipes the output to the `sort` command. The `sort` command takes the usernames that begin each line of the `/etc/passwd` file, sorts them alphabetically, and pipes the output to the `less` command (to page through the output).

Pipes are an excellent illustration of how UNIX, the predecessor of Linux, was created as an operating system made up of building blocks. A standard practice in UNIX was to connect utilities in different ways to get different jobs done. For example, before the days of graphical word processors, users created plain-text files that included macros to indicate formatting. To see how the document really appeared, they would use a command such as the following:

```
$ gunzip < /usr/share/man/man1/grep.1.gz | nroff -c -man | less
```

In this example, the contents of the grep man page (grep.1.gz) are directed to the gunzip command to be unzipped. The output from gunzip is piped to the nroff command to format the man page using the manual macro (-man). To display the output, it is piped to the less command. Because the file being displayed is in plain text, you could have substituted any number of options to work with the text before displaying it. You could sort the contents, change or delete some of the content, or bring in text from other documents. The key is that, instead of all of those features being in one program, you get results from piping and redirecting input and output between multiple commands.

Sequential commands

Sometimes, you may want a sequence of commands to run, with one command completing before the next command begins. You can do this by typing several commands on the same command line and separating them with semicolons (;):

```
$ date ; troff -me verylargedocument | lpr ; date
```

In this example, I was formatting a huge document and wanted to know how long it would take. The first command (date) showed the date and time before the formatting started. The troff command formatted the document and then piped the output to the printer. When the formatting was finished, the date and time were printed again (so I knew how long the troff command took to complete).

Another useful command to add to the end of a long command line is mail. You could add the following to the end of a command line:

```
; mail -s "Finished the long command" chris@example.com
```

Then, for example, a mail message is sent to the user you choose after the command completes.

Background commands

Some commands can take a while to complete. Sometimes, you may not want to tie up your shell waiting for a command to finish. In those cases, you can have the commands run in the background by using the ampersand (&).

Text formatting commands (such as nroff and troff, described earlier) are examples of commands that can be run in the background to format a large document. You also might want to create your own shell scripts that run in the background to check continuously for certain events to occur, such as the hard disk filling up or particular users logging in.

The following is an example of a command being run in the background:

```
$ troff -me verylargedocument | lpr &
```

Don't close the shell until the process is completed or that kills the process. Other ways to manage background and foreground processes are described in Chapter 6, "Managing Running Processes."

Expanding commands

With command substitution, you can have the output of a command interpreted by the shell instead of by the command itself. In this way, you can have the standard output of a command become an argument for another command. The two forms of command substitution are `$(command)` and `` `command` `` (backticks, not single quotes).

The command in this case can include options, metacharacters, and arguments. The following is an example of using command substitution:

```
$ vi $(find /home | grep xyzzy)
```

In this example, the command substitution is done before the `vi` command is run. First, the `find` command starts at the `/home` directory and prints out all of the files and directories below that point in the filesystem. The output is piped to the `grep` command, which filters out all files except for those that include the string `xyzzy` in the filename. Finally, the `vi` command opens all filenames for editing (one at a time) that include `xyzzy`. (If you run this and are not familiar with `vi`, you can type **`:q!`** to exit the file.)

This particular example is useful if you want to edit a file for which you know the name but not the location. As long as the string is uncommon, you can find and open every instance of a filename existing beneath a point you choose in the filesystem. (In other words, don't use `grep` from the root filesystem or you'll match and try to edit several thousand files.)

Expanding arithmetic expressions

Sometimes, you want to pass arithmetic results to a command. There are two forms that you can use to expand an arithmetic expression and pass it to the shell: *`$[expression]`* or *`$(expression)`*. The following is an example:

```
$ echo "I am $[2020 - 1957] years old."
I am 63 years old.
```

The shell interprets the arithmetic expression first (`2020 - 1957`) and then passes that information to the `echo` command. The `echo` command displays the text with the results of the arithmetic (`63`) inserted.

Here's an example of the other form:

```
$ echo "There are $(ls | wc -w) files in this directory."
There are 14 files in this directory.
```

This lists the contents of the current directory (`ls`) and runs the word count command to count the number of files found (`wc -w`). The resulting number (14, in this case) is echoed back with the rest of the sentence shown.

Expanding variables

Variables that store information within the shell can be expanded using the dollar sign (`$`) metacharacter. When you expand an environment variable on a command line, the value of the variable is printed instead of the variable name itself, as follows:

```
$ ls -l $BASH
-rwxr-xr-x. 1 root root 1219248 Oct 12 17:59 /usr/bin/bash
```

Using $BASH as an argument to ls -l causes a long listing of the bash command to be printed.

Using Shell Variables

The shell itself stores information that may be useful to the user's shell session in what are called *variables*. Examples of variables include $SHELL (which identifies the shell you are using), $PS1 (which defines your shell prompt), and $MAIL (which identifies the location of your user's mailbox).

You can see all variables set for your current shell by typing the set command. A subset of your local variables is referred to as *environment variables*. Environment variables are variables that are exported to any new shells opened from the current shell. Type **env** to see environment variables.

You can type **echo $*VALUE***, where ***VALUE*** is replaced by the name of a particular environment variable you want to list. And because there are always multiple ways to do anything in Linux, you can also type **declare** to get a list of the current environment variables and their values along with a list of shell functions.

Besides those that you set yourself, system files set variables that store things such as locations of configuration files, mailboxes, and path directories. They can also store values for your shell prompts, the size of your history list, and type of operating system. You can refer to the value of any of those variables by preceding it with a dollar sign ($) and placing it anywhere on a command line. For example:

```
$ echo $USER
chris
```

This command prints the value of the USER variable, which holds your username (chris). Substitute any other value for USER to print its value instead.

When you start a shell (by logging in via a virtual console or opening a Terminal window), many environment variables are already set. Table 3.5 shows some variables that are either set when you use a bash shell or that can be set by you to use with different features.

Creating and using aliases

Using the alias command, you can effectively create a shortcut to any command and options that you want to run later. You can add and list aliases with the alias command. Consider the following examples of using alias from a bash shell:

```
$ alias p='pwd ; ls -CF'
$ alias rm='rm -i'
```

In the first example, the letter p is assigned to run the command pwd and then to run ls -CF to print the current working directory and list its contents in column form.

TABLE 3.5 Common Shell Environment Variables

Variable	Description
BASH	This contains the full pathname of the bash command. This is usually /bin/bash.
BASH_VERSION	This is a number representing the current version of the bash command.
EUID	This is the effective user ID number of the current user. It is assigned when the shell starts, based on the user's entry in the /etc/passwd file.
FCEDIT	If set, this variable indicates the text editor used by the fc command to edit history commands. If this variable isn't set, the vi command is used.
HISTFILE	This is the location of your history file. It is typically located at $HOME/.bash_history.
HISTFILESIZE	This is the number of history entries that can be stored. After this number is reached, the oldest commands are discarded. The default value is 1000.
HISTCMD	This returns the number of the current command in the history list.
HOME	This is your home directory. It is your current working directory each time you log in or type the cd command with any options.
HOSTTYPE	This is a value that describes the computer architecture on which the Linux system is running. For Intel-compatible PCs, the value is i386, i486, i586, i686, or something like i386-linux. For AMD 64-bit machines, the value is x86_64.
MAIL	This is the location of your mailbox file. The file is typically your username in the /var/spool/mail directory.
OLDPWD	This is the directory that was the working directory before you changed to the current working directory.
OSTYPE	This name identifies the current operating system. For Fedora Linux, the OSTYPE value is either linux or linux-gnu, depending on the type of shell you are using. (Bash can run on other operating systems as well.)
PATH	This is the colon-separated list of directories used to find commands that you type. The default value for regular users varies for different distributions but typically includes the following: /bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/X11R6/bin:~/bin. You need to type the full path or a relative path to a command that you want to run which is not in your PATH. For the root user, the value also includes /sbin, /usr/sbin, and /usr/local/sbin.
PPID	This is the process ID of the command that started the current shell (for example, the Terminal window containing the shell).

Variable	Description
PROMPT_COMMAND	This can be set to a command name that is run each time before your shell prompt is displayed. Setting PROMPT_COMMAND=date lists the current date/time before the prompt appears.
PS1	This sets the value of your shell prompt. There are many items that you can read into your prompt (date, time, username, hostname, and so on). Sometimes a command requires additional prompts, which you can set with the variables PS2, PS3, and so on.
PWD	This is the directory that is assigned as your current directory. This value changes each time you change directories using the cd command.
RANDOM	Accessing this variable causes a random number to be generated. The number is between 0 and 99999.
SECONDS	This is the number of seconds since the time the shell was started.
SHLVL	This is the number of shell levels associated with the current shell session. When you log in to the shell, the SHLVL is 1. Each time you start a new bash command (by, for example, using su to become a new user, or by simply typing bash), this number is incremented.
TMOUT	This can be set to a number representing the number of seconds the shell can be idle without receiving input. After the number of seconds is reached, the shell exits. This security feature makes it less likely for unattended shells to be accessed by unauthorized people. (This must be set in the login shell for it actually to cause the shell to log out the user.)

The second example runs the `rm` command with the `-i` option each time you type `rm`. (This is an alias that is often set automatically for the root user. Instead of just removing files, you are prompted for each individual file removal. This prevents you from automatically removing all of the files in a directory by mistakenly typing something such as `rm *`.)

While you are in the shell, you can check which aliases are set by typing the `alias` command. If you want to remove an alias, use **`unalias`**. (Remember that if the `alias` is set in a configuration file, it will be set again when you open another shell.)

Exiting the shell

To exit the shell when you are finished, type **`exit`** or press Ctrl+D. If you go to the shell from a Terminal window and you are using the original shell from that window, exiting causes the Terminal window to close. If you are at a virtual console, the shell exits and returns you to a login prompt.

If you have multiple shells open from the same shell session, exiting a shell simply returns you to the shell that launched the current shell. For example, the `su` command opens a shell as a new user. Exiting from that shell simply returns you to the original shell.

Creating Your Shell Environment

You can tune your shell to help you work more efficiently. You can set aliases to create shortcuts to your favorite command lines and environment variables to store bits of information. By adding those settings to shell configuration files, you can have the settings available every time you open a shell.

Configuring your shell

Several configuration files support how your shell behaves. Some of the files are executed for every user and every shell, whereas others are specific to the user who creates the configuration file. Table 3.6 shows the files that are of interest to anyone using the bash shell in Linux. (Notice the use of ~ in the filenames to indicate that the file is located in each user's home directory.)

To change the `/etc/profile` or `/etc/bashrc` files, you must be the root user. It is better to create an `/etc/profile.d/custom.sh` file to add system-wide settings instead of

TABLE 3.6 Bash Configuration Files

File	Description
`/etc/profile`	This sets up user environment information for every user. It is executed when you first log in. This file provides values for your path in addition to setting environment variables for such things as the location of your mailbox and the size of your history files. Finally, `/etc/profile` gathers shell settings from configuration files in the `/etc/profile.d` directory.
`/etc/bashrc`	This executes for every user who runs the bash shell each time a bash shell is opened. It sets the default prompt and may add one or more aliases. Values in this file can be overridden by information in each user's `~/.bashrc` file.
`~/.bash_profile`	This is used by each user to enter information that is specific to his or her use of the shell. It is executed only once—when the user logs in. By default, it sets a few environment variables and executes the user's `.bashrc` file. This is a good place to add environment variables because, once set, they are inherited by future shells.
`~/.bashrc`	This contains the information that is specific to your bash shells. It is read when you log in and also each time you open a new bash shell. This is the best location to add aliases so that your shell picks them up.
`~/.bash_logout`	This executes each time you log out (exit the last bash shell).

editing those files directly, however. Users can change the information in the $HOME/.bash_profile, $HOME/.bashrc, and $HOME/.bash_logout files in their own home directories.

Until you learn to use the vi editor, described in Chapter 5, "Working with Text Files," you can use a simple editor called nano to edit plain-text files. For example, enter the following to edit and add stuff to your $HOME/.bashrc file:

```
$ nano $HOME/.bashrc
```

With the file open in nano, move the cursor down to the bottom of the file (using the down arrow key). Type the line you want (for example, you could type **alias d='date +%D'**). To save the file, press Ctrl+O (the letter *O*); to quit, press Ctrl+X. The next time you log in or open a new shell, you can use the new alias (in this case, just type **d**). To have the new information you just added to the file available from the current shell, type the following:

```
$ source $HOME/.bashrc
$ d
06/29/19
```

The following sections provide ideas about items to add to your shell configuration files. In most cases, you add these values to the .bashrc file in your home directory. However, if you administer a system, you may want to set some of these values as defaults for all your Linux system's users.

Setting your prompt

Your prompt consists of a set of characters that appear each time the shell is ready to accept a command. The PS1 environment variable sets what the prompt contains and is what you will interact with most of the time. If your shell requires additional input, it uses the values of PS2, PS3, and PS4.

When your Linux system is installed, often a prompt is set to contain more than just a dollar sign or pound sign. For example, in Fedora or Red Hat Enterprise Linux, your prompt is set to include the following information: your username, your hostname, and the base name of your current working directory. That information is surrounded by brackets and followed by a dollar sign (for regular users) or a pound sign (for the root user). The following is an example of that prompt:

```
[chris@myhost bin]$
```

If you change directories, the bin name would change to the name of the new directory. Likewise, if you were to log in as a different user or to a different host, that information would change.

You can use several special characters (indicated by adding a backslash to a variety of letters) to include different information in your prompt. Special characters can be used to output your Terminal number, the date, and the time as well as other pieces of information. Table 3.7 provides some examples (you can find more on the bash man page).

TABLE 3.7 Characters to Add Information to Bash Prompt

Special Character	Description
\!	This shows the current command history number. This includes all previous commands stored for your username.
\#	This shows the command number of the current command. This includes only the commands for the active shell.
\$	This shows the user prompt ($) or root prompt (#), depending on which type of user you are.
\W	This shows only the current working directory base name. For example, if the current working directory was /var/spool/mail, this value simply appears as mail.
\[	This precedes a sequence of nonprinting characters. This can be used to add a Terminal control sequence into the prompt for such things as changing colors, adding blink effects, or making characters bold. (Your Terminal determines the exact sequences available.)
\]	This follows a sequence of nonprinting characters.
\\	This shows a backslash.
\d	This displays the day name, month, and day number of the current date, for example, Sat Jan 23.
\h	This shows the hostname of the computer running the shell.
\n	This causes a newline to occur.
\nnn	This shows the character that relates to the octal number replacing *nnn*.
\s	This displays the current shell name. For the bash shell, the value would be bash.
\t	This prints the current time in hours, minutes, and seconds, for example, 10:14:39.
\u	This prints your current username.
\w	This displays the full path to the current working directory.

TIP

If you are setting your prompt temporarily by typing at the shell, you should put the value of PS1 in quotes. For example, you could type **export PS1="[\t \w]\$ "** to see a prompt that looks like this:

```
[20:26:32 /var/spool]$.
```

To make a change to your prompt permanent, add the value of PS1 to your .bashrc file in your home directory (assuming that you are using the bash shell). There may already be a PS1 value in that file, which you can modify. Refer to the Bash Prompt HOWTO

(http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO) for information on changing colors, commands, and other features of your bash shell prompt.

Adding environment variables

You might want to consider adding a few environment variables to your .bashrc file. These can help make working with the shell more efficient and effective:

TMOUT This sets how long the shell can be inactive before bash automatically exits. The value is the number of seconds for which the shell has not received input. This can be a nice security feature, in case you leave your desk while you are still logged in to Linux. To prevent being logged off while you are working, you may want to set the value to something like TMOUT=1800 (to allow 30 minutes of idle time). You can use any Terminal session to close the current shell after a set number of seconds, for example, TMOUT=30.

PATH As described earlier, the PATH variable sets the directories that are searched for the commands that you use. If you often use directories of commands that are not in your path, you can permanently add them. To do this, add a PATH variable to your .bashrc file. For example, to add a directory called /getstuff/bin, add the following:

```
PATH=$PATH:/getstuff/bin ; export PATH
```

This example first reads all of the current path directories into the new PATH ($PATH), adds the /getstuff/bin directory, and then exports the new PATH.

3

CAUTION

Some people add the current directory to their PATH by adding a directory identified simply as a dot (.), as follows:

```
PATH=.:$PATH ; export PATH
```

This enables you to run commands in your current directory before evaluating any other command in the path (which people may be used to if they have used DOS). However, the security risk with this procedure is that you could be in a directory that contains a command that you don't intend to run from that directory. For example, a malicious person could put an ls command in a directory that, instead of listing the content of your directory, does something devious. Because of this, the practice of adding the dot to your path is highly discouraged.

WHATEVER You can create your own environment variables to provide shortcuts in your work. Choose any name that is not being used and assign a useful value to it. For example, if you do lots of work with files in the /work/time/files/info/memos directory, you could set the following variable:

```
M=/work/time/files/info/memos ; export M
```

You could make that your current directory by typing cd $M. You could run a program from that directory called hotdog by typing **$M/hotdog**. You could edit a file from there called bun by typing **vi $M/bun**.

Getting Information about Commands

When you first start using the shell, it can be intimidating. All that you see is a prompt. How do you know which commands are available, which options they use, or how to use advanced features? Fortunately, lots of help is available. Here are some places that you can look to supplement what you learn in this chapter:

- **Check the PATH.** Type **`echo $PATH`**. You see a list of the directories containing commands that are immediately accessible to you. Listing the contents of those directories displays most standard Linux commands. For example:

```
$ ls /bin
arch      dd            fusermount   loadkeys   mv
awk       df            gawk         login      nano
basename  dmesg         gettext      ls         netstat
bash      dnsdomainname grep         lsblk      nice
cat       domainname    gtar         lscgroup   nisdomainname
chgrp     echo          gunzip       lssubsys   ping
chmod     ed            gzip         mail       ping6
chown     egrep         hostname     mailx      ps
cp        env           ipcalc       mkdir      pwd
cpio      ex            kbd_mode     mknod      readlink
csh       false         keyctl       mktemp     red
cut       fgrep         kill         more       redhat_lsb_init
dash      find          link         mount      rm
date      findmnt       ln           mountpoint rmdir
```

- **Use the `help` command.** Some commands are built into the shell, so they do not appear in a directory. The `help` command lists those commands and shows options available with each of them. (Enter **`help | less`** to page through the list.) For help with a particular built-in command, enter **`help`** ***`command`***, replacing ***`command`*** with the name that interests you. The `help` command works with the bash shell only.
- **Use `--help` with the command.** Many commands include a `--help` option that you can use to get information about how the command is used. For example, if you enter **`date --help | less`**, the output shows not only options, but also time formats that you can use with the `date` command. Other commands simply use a `–h` option, like `fdisk -h`.

- **Use the `info` command.** The `info` command is another tool for displaying information about commands from the shell. The `info` command can move among a hierarchy of nodes to find information about commands and other items. Not all commands have information available in the info database, but sometimes more information can be found there than on a man page.
- **Use the `man` command.** To learn more about a particular command, enter **`man` *`command`***. (Replace ***`command`*** with the command name you want.) A description of the command and its options appears on the screen.

Man pages are the most common means of getting information about commands as well as other basic components of a Linux system. Each man page falls into one of the categories listed in Table 3.8. As a regular user, you will be most interested in man pages in section 1. As a system administrator, you will also be interested in sections 5 and 8, and occasionally section 4. Programmers will be interested in section 2 and 3 man pages.

TABLE 3.8 **Manual Page Sections**

Section Number	Section Name	Description
1	User Commands	Commands that can be run from the shell by a regular user (typically no administrative privilege is needed)
2	System Calls	Programming functions used within an application to make calls to the kernel
3	C Library Functions	Programming functions that provide interfaces to specific programming libraries (such as those for certain graphical interfaces or other libraries that operate in user space)
4	Devices and Special Files	Filesystem nodes that represent hardware devices (such as Terminals or CD drives) or software devices (such as random number generators)
5	File Formats and Conventions	Types of files (such as a graphics or word processing file) or specific configuration files (such as the `passwd` or `group` file)
6	Games	Games available on the system
7	Miscellaneous	Overviews of topics such as protocols, filesystems, character set standards, and so on
8	System Administration Tools and Daemons	Commands that require root or other administrative privileges to use

Options to the `man` command enable you to search the man page database or display man pages on the screen. Here are some examples of man commands and options:

```
$ man -k passwd
...
passwd               (1)  - update user's authentication tokens
passwd               (5)  - password file
$ man passwd
$ man 5 passwd
```

Using the `-k` option, you can search the name and summary sections of all man pages installed on the system. There are about a dozen man pages that include "`passwd`" in the name or description of a command.

NOTE

If `man -k` displays no output, it may be that the man page database has not been initialized. Type **`mandb`** as root to initialize the man page database.

Let's say that the two man pages in which I am interested are the `passwd` command (in section 1 of the man pages) and the `passwd` file (in section 5) man pages. Because just typing **`man passwd`** displays the section 1 page, I need to request explicitly the section 5 man page if I want to see that instead (**`man 5 passwd`**).

While you are displaying a man page, you can view different parts of the file using Page Down and Page Up keys (to move a page at a time). Use the Enter key or up and down arrows to move a line at a time. Press a forward slash (**`/`**) and type a term to search the document for that term. Press **`n`** to repeat the search forward or **`N`** to repeat the search backward. To quit the man page, type **`q`**.

Summary

To become an expert Linux user, you must be able to use the shell to type commands. This chapter focuses on the bash shell, which is the one that is most commonly used with Linux systems. You learned how commands are structured and how many special features, such as variables, command completion, and aliases, are used.

The next chapter describes how to move around the Linux filesystem from the shell command line.

Exercises

Use these exercises to test your knowledge of using the shell. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. From your desktop, switch to the third virtual console and log in to your user account. Run a few commands. Then exit the shell and return to the desktop.
2. Open a Terminal window and change the font color to red and the background to yellow.
3. Find the location of the `mount` command and the `tracepath` man page.
4. Type the following three commands, and then recall and change those commands as described:

```
$ cat /etc/passwd
$ ls $HOME
$ date
```

   a. Use the command-line recall feature to recall the `cat` command and change `/etc/passwd` to `/etc/group`.
   b. Recall the `ls` command, determine how to list files by time (using the man page), and add that option to the `ls $HOME` command line.
   c. Add format indicators to the `date` command to display the date output as *month/day/year*.

5. Run the following command, typing as few characters as possible (using tab completion):

```
basename /usr/share/doc/
```

6. Use the `cat` command to list the contents of the `/etc/services` file and pipe those contents to the `less` command so that you can page through it (press **q** to quit when you are finished).
7. Run the `date` command in such a way that the output from that command produces the current day, month, date, and year. Have that read into another command line, resulting in text that appears like the following (your date, of course, will be different): `Today is Thursday, December 19, 2019`.
8. Using variables, find out what your hostname, username, shell, and home directories are currently set to.
9. Create an alias called `mypass` that displays the contents of the `/etc/passwd` file on your screen in such a way that it is available every time you log in or open a new shell from your user account.
10. Display the man page for the `mount` system call.

CHAPTER 4

Moving Around the Filesystem

IN THIS CHAPTER

Learning about the Linux filesystem

Listing file and directory attributes

Making files and directories

Listing and changing permission and ownership

Making copies and moving files

The Linux filesystem is the structure in which all of the information on your computer is stored. In fact, one of the defining properties of the UNIX systems on which Linux is based is that nearly everything you need to identify on your system (data, commands, symbolic links, devices, and directories) is represented by items in the filesystems. Knowing where things are and understanding how to get around the filesystem from the shell are critical skills in Linux.

In Linux, files are organized within a hierarchy of directories. Each directory can contain files as well as other directories. You can refer to any file or directory using either a full path (for example, `/home/joe/myfile.txt`) or a relative path (for example, if `/home/joe` were your current directory, you could simply refer to the file as `myfile.txt`).

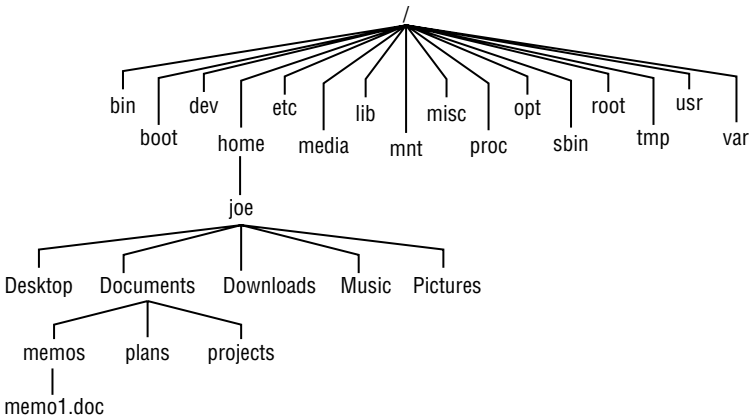
If you were to map out the files and directories in Linux, it would look like an upside-down tree. At the top is the *root* directory (not to be confused with the root user), which is represented by a single slash (`/`). Below that is a set of common directories in the Linux system, such as `bin`, `dev`, `home`, `lib`, and `tmp`, to name a few. Each of those directories, as well as directories added to the root directory, can contain subdirectories.

Figure 4.1 illustrates how the Linux filesystem is organized as a hierarchy. To demonstrate how directories are connected, the figure shows a `/home` directory that contains a subdirectory for the user `joe`. Within the `joe` directory are `Desktop`, `Documents`, and other subdirectories. To refer to a file called `memo1.doc` in the `memos` directory, you can type the full path of `/home/joe/Documents/memos/memo1.doc`. If your current directory is `/home/joe/`, refer to the file as `Documents/memos/memo1.doc`.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

FIGURE 4.1

The Linux filesystem is organized as a hierarchy of directories.



Some of these Linux directories may interest you:

/bin	Contains common Linux user commands, such as `ls`, `sort`, `date`, and `chmod`.
/boot	Has the bootable Linux kernel, initial RAM disk, and boot loader configuration files (GRUB).
/dev	Contains files representing access points to devices on your systems. These include terminal devices (`tty*`), hard disks (`hd*` or `sd*`), RAM (`ram*`), and CD-ROM (`cd*`). Users can access these devices directly through these device files; however, applications often hide the actual device names to end users.
/etc	Contains administrative configuration files. Most of these files are plain-text files that, given the user has proper permission, can be edited with any text editor.
/home	Contains directories assigned to each regular user with a login account. (The root user is an exception, using `/root` as his or her home directory.)
/media	Provides a standard location for automounting devices (removable media in particular). If the medium has a volume name, that name is typically used as the mount point. For example, a USB drive with a volume name of `myusb` would be mounted on `/media/myusb`.
/lib	Contains shared libraries needed by applications in `/bin` and `/sbin` to boot the system.
/mnt	A common mount point for many devices before it was supplanted by the standard `/media` directory. Some bootable Linux systems still use this directory to mount hard disk partitions and remote filesystems. Many people still use this directory to temporarily mount local or remote filesystems, which are not mounted permanently.
/misc	A directory sometimes used to automount filesystems upon request.
/opt	Directory structure available to store add-on application software.

/proc	Contains information about system resources.
/root	Represents the root user's home directory. The home directory for root does not reside beneath /home for security reasons.
/sbin	Contains administrative commands and daemon processes.
/sys	Contains parameters for such things as tuning block storage and managing cgroups.
/tmp	Contains temporary files used by applications.
/usr	Contains user documentation, games, graphical files (X11), libraries (lib), and a variety of other commands and files that are not needed during the boot process. The /usr directory is meant for files that don't change after installation (in theory, /usr could be mounted read-only).
/var	Contains directories of data used by various applications. In particular, this is where you would place files that you share as an FTP server (/var/ftp) or a web server (/var/www). It also contains all system log files (/var/log) and spool files in /var/spool (such as mail, cups, and news). The /var directory contains directories and files that are meant to change often. On server computers, it is common to create the /var directory as a separate filesystem, using a filesystem type that can be easily expanded.

The filesystems in the DOS or Microsoft Windows operating systems differ from Linux's file structure, as the sidebar "Linux Filesystems versus Windows-Based Filesystems" explains.

Linux Filesystems versus Windows-Based Filesystems

Although similar in many ways, the Linux filesystem has some striking differences when compared to filesystems used in MS-DOS and Windows operating systems. Here are a few of these differences:

- In MS-DOS and Windows filesystems, drive letters represent different storage devices. In Linux, all storage devices are connected to the filesystem hierarchy. So, the fact that all of /usr may be on a separate hard disk or that /mnt/remote1 is a filesystem from another computer is invisible to the user.
- Slashes, rather than backslashes, are used to separate directory names in Linux. So C:\home\joe in a Microsoft system is /home/joe in a Linux system.
- Filenames almost always have suffixes in DOS (such as .txt for text files or .docx for word-processing files). Although at times you can use that convention in Linux, three-character suffixes have no required meaning in Linux. They can be useful for identifying a file type. Many Linux applications and desktop environments use file suffixes to determine the contents of a file. In Linux, however, DOS command extensions such as .com, .exe, and .bat don't necessarily signify an executable. (Permission flags make Linux files executable.)
- Every file and directory in a Linux system has permissions and ownership associated with it. Security varies among Microsoft systems. Because DOS and Microsoft Windows began as single-user systems, file ownership was not built into those systems when they were designed. Later releases added features such as file and folder attributes to address this problem.

Using Basic Filesystem Commands

I want to introduce you to a few simple commands for getting around the filesystem to start out. If you want to follow along, log in and open a shell. When you log in to a Linux system and open a shell, you are placed in your home directory. As a Linux user, most of the files you save and work with will probably be in that directory or in subdirectories that you create. Table 4.1 shows commands to create and use files and directories.

TABLE 4.1 Commands to Create and Use Files

Command	Result
`cd`	Changes to another directory
`pwd`	Prints the name of the current (or present) working directory
`mkdir`	Creates a directory
`chmod`	Changes the permission on a file or directory
`ls`	Lists the contents of a directory

One of the most basic commands that you use from the shell is `cd`. The `cd` command can be used with no options (to take you to your home directory) or with full or relative paths. Consider the following commands:

```
$ cd /usr/share/
$ pwd
/usr/share
$ cd doc
$ pwd
/usr/share/doc
$ cd
$ pwd
/home/chris
```

The `/usr/share` option represents the *absolute path* to a directory on the system. Because it begins with a slash (`/`), this path tells the shell to start at the root of the filesystem and take you to the `share` directory that exists in the `usr` directory. The `doc` option to the `cd` command looks for a directory called `doc` that is relative to the current directory. So that command made `/usr/share/doc` your current directory.

After that, by typing `cd` alone, you are returned to your home directory. If you ever wonder where you are in the filesystem, the `pwd` command can help you. Here are a few other interesting `cd` command options:

```
$ cd ~
$ pwd
/home/chris
```

```
$ cd ~/Music
$ pwd
/home/chris/Music
$ cd ../../../usr
$ pwd
/usr
```

The tilde (~) represents your home directory. So `cd ~` takes you there. You can use the tilde to refer to directories relative to your home directory as well, such as `/home/chris/Music` with `~/Music`. Typing a name as an option takes you to a directory below the current directory, but you can use two dots (`..`) to go to a directory above the current directory. The example shown takes you up three directory levels (to `/`), and then takes you into the `/usr` directory.

The following steps lead you through the process of creating directories within your home directory and moving among your directories, with a mention of setting appropriate file permissions:

1. Go to your home directory. To do this, simply type **`cd`** in a shell and press Enter. (For other ways of referring to your home directory, see the sidebar "Identifying Directories.")
2. To make sure that you're in your home directory, type **`pwd`**. When I do this, I get the following response (yours will reflect your home directory):

   ```
   $ pwd
   /home/joe
   ```

3. Create a new directory called `test` in your home directory, as follows:

   ```
   $ mkdir test
   ```

4. Check the permissions of the directory:

   ```
   $ ls -ld test
   drwxr-xr-x 2 joe sales 1024 Jan 24 12:17 test
   ```

   This listing shows that `test` is a directory (`d`). The `d` is followed by the permissions (`rwxr-xr-x`), which are explained later in the section "Understanding File Permissions and Ownership." The rest of the information indicates the owner (`joe`), the group (`sales`), and the date that the files in the directory were most recently modified (Jan 24 at 12:17 p.m.).

Note

When you add a new user in Fedora and Red Hat Enterprise Linux, the user is assigned to a group of the same name by default. For example, in the preceding text, the user `joe` would be assigned to the group `joe`. This approach to assigning groups is referred to as the *user private group scheme*.

For now, enter the following:

```
$ chmod 700 test
```

This step changes the permissions of the directory to give you complete access and everyone else no access at all. (The new permissions should read `rwx------`.)

5. Make the test directory your current directory as follows:

```
$ cd test
$ pwd
/home/joe/test
```

If you followed along, at this point a subdirectory of your home directory called `test` is your current working directory. You can create files and directories in the `test` directory along with the descriptions in the rest of this chapter.

Using Metacharacters and Operators

Whether you are listing, moving, copying, removing, or otherwise acting on files in your Linux system, certain special characters, referred to as metacharacters and operators, help you to work with files more efficiently. Metacharacters can help you match one or more files without completely typing each filename. Operators enable you to direct information from one command or file to another command or file.

Using file-matching metacharacters

To save you some keystrokes and enable you to refer easily to a group of files, the bash shell lets you use metacharacters. Anytime you need to refer to a file or directory, such as to list, open, or remove it, you can use metacharacters to match the files you want. Here are some useful metacharacters for matching filenames:

*****	Matches any number of characters.
?	Matches any one character.
[...]	Matches any one of the characters between the brackets, which can include a hyphen-separated range of letters or numbers.

Try out some of these file-matching metacharacters by first going to an empty directory (such as the `test` directory described in the previous section) and creating some empty files:

```
$ touch apple banana grape grapefruit watermelon
```

The `touch` command creates empty files. The commands that follow show you how to use shell metacharacters with the `ls` command to match filenames. Try the following commands to see whether you get the same responses:

```
$ ls a*
apple
$ ls g*
grape grapefruit
$ ls g*t
```

```
grapefruit
$ ls *e*
apple grape grapefruit watermelon
$ ls *n*
banana watermelon
```

The first example matches any file that begins with `a` (`apple`). The next example matches any files that begin with `g` (`grape`, `grapefruit`). Next, files beginning with `g` and ending in `t` are matched (`grapefruit`). Next, any file that contains `e` in the name is matched (`apple`, `grape`, `grapefruit`, `watermelon`). Finally, any file that contains `n` is matched (`banana`, `watermelon`).

Here are a few examples of pattern matching with the question mark (`?`):

```
$ ls ????e
apple grape
$ ls g???e*
grape grapefruit
```

The first example matches any five-character file that ends in `e` (`apple`, `grape`). The second matches any file that begins with `g` and has `e` as its fifth character (`grape`, `grapefruit`).

The following examples use braces to do pattern matching:

```
$ ls [abw]*
apple banana watermelon
$ ls [agw]*[ne]
apple grape watermelon
```

In the first example, any file beginning with `a`, `b`, or `w` is matched. In the second, any file that begins with `a`, `g`, or `w` and also ends with either `n` or `e` is matched. You can also include ranges within brackets. For example:

```
$ ls [a-g]*
apple banana grape grapefruit
```

Here, any filenames beginning with a letter from `a` through `g` are matched.

Using file-redirection metacharacters

Commands receive data from standard input and send it to standard output. Using pipes (described earlier), you can direct standard output from one command to the standard input of another. With files, you can use less than (`<`) and greater than (`>`) signs to direct data to and from files. Here are the file-redirection characters:

- `<` Directs the contents of a file to the command. In most cases, this is the default action expected by the command and the use of the character is optional; using `less bigfile` is the same as `less < bigfile`.
- `>` Directs the standard output of a command to a file. If the file exists, the content of that file is overwritten.

2> Directs standard error (error messages) to the file.

&> Directs both standard output and standard error to the file.

>> Directs the output of a command to a file, adding the output to the end of the existing file.

The following are some examples of command lines where information is directed to and from files:

```
$ mail root < ~/.bashrc
$ man chmod | col -b > /tmp/chmod
$ echo "I finished the project on $(date)" >> ~/projects
```

In the first example, the content of the `.bashrc` file in the home directory is sent in a mail message to the computer's root user. The second command line formats the `chmod` man page (using the `man` command), removes extra back spaces (`col -b`), and sends the output to the file `/tmp/chmod` (erasing the previous `/tmp/chmod` file, if it exists). The final command results in the following text being added to the user's project file:

```
I finished the project on Sat Jun 15 13:46:49 EDT 2019
```

Another type of redirection, referred to as *here text* (also called *here document*), enables you to type text that can be used as standard input for a command. Here documents involve entering two less-than characters (`<<`) after a command, followed by a word. All typing following that word is taken as user input until the word is repeated on a line by itself. Here is an example:

```
$ mail root cnegus rjones bdecker << thetext
> I want to tell everyone that there will be a 10 a.m.
> meeting in conference room B. Everyone should attend.
>
> -- James
> thetext
$
```

This example sends a mail message to root, cnegus, rjones, and bdecker usernames. The text entered between `<<thetext` and `thetext` becomes the content of the message. A common use of here text is to use it with a text editor to create or add to a file from within a script:

```
/bin/ed /etc/resolv.conf <<resendit
a
nameserver 100.100.100.100
.
w
q
resendit
```

With these lines added to a script run by the root user, the `ed` text editor adds the IP address of a DNS server to the `/etc/resolv.conf` file.

Using brace expansion characters

By using curly braces ({}), you can expand out a set of characters across filenames, directory names, or other arguments to which you give commands. For example, if you want to create a set of files such as `memo1` through `memo5`, you can do that as follows:

```
$ touch memo{1,2,3,4,5}
$ ls
memo1 memo2 memo3 memo4 memo5
```

The items that are expanded don't have to be numbers or even single digits. For example, you could use ranges of numbers or digits. You could also use any string of characters, as long as you separate them with commas. Here are some examples:

```
$ touch {John,Bill,Sally}-{Breakfast,Lunch,Dinner}
$ ls
Bill-Breakfast Bill-Lunch John-Dinner Sally-Breakfast Sally-Lunch
Bill-Dinner John-Breakfast John-Lunch Sally-Dinner
$ rm -f {John,Bill,Sally}-{Breakfast,Lunch,Dinner}
$ touch {a..f}{1..5}
$ ls
a1 a3 a5 b2 b4 c1 c3 c5 d2 d4 e1 e3 e5 f2 f4
a2 a4 b1 b3 b5 c2 c4 d1 d3 d5 e2 e4 f1 f3 f5
```

In the first example, the use of two sets of braces means John, Bill, and Sally each have filenames associated with Breakfast, Lunch, and Dinner. If I had made a mistake, I could easily recall the command and change `touch` to `rm -f` to delete all of the files. In the next example, the use of two dots between letters `a` and `f` and numbers `1` and `5` specifies the ranges to be used. Note the files that were created from those few characters.

Listing Files and Directories

The `ls` command is the most common command used to list information about files and directories. Many options available with the `ls` command allow you to gather different sets of files and directories as well as to view different kinds of information about them.

By default, when you type the `ls` command, the output shows you all non-hidden files and directories contained in the current directory. When you type `ls`, however, many Linux systems (including Fedora and RHEL) assign an alias `ls` to add options. To see if `ls` is aliased, enter the following:

```
$ alias ls
alias ls='ls --color=auto'
```

The `--color=auto` option causes different types of files and directories to be displayed in different colors. So, return to the `$HOME/test` directory created earlier in the chapter, add a couple of different types of files, and then see what they look like with the `ls` command.

```
$ cd $HOME/test
$ touch scriptx.sh apple
```

```
$ chmod 755 scriptx.sh
$ mkdir Stuff
$ ln -s apple pointer_to_apple
$ ls
apple pointer_to_apple scriptx.sh Stuff
```

Although you can't see it in the preceding code example, the directory `Stuff` shows up in blue, `pointer_to_apple` (a symbolic link) appears as aqua, and `scriptx.sh` (which is an executable file) appears in green. All other regular files show up in black. Typing `ls -l` to see a long listing of those files can make these different types of files clearer still:

```
$ ls -l
total 4
-rw-rw-r--. 1 joe joe 0 Dec 18 13:38 apple
lrwxrwxrwx. 1 joe joe 5 Dec 18 13:46 pointer_to_apple -> apple
-rwxr-xr-x. 1 joe joe 0 Dec 18 13:37 scriptx.sh
drwxrwxr-x. 2 joe joe 4096 Dec 18 13:38 Stuff
```

As you look at the long listing, notice that the first character of each line shows the type of file. A hyphen (`-`) indicates a regular file, `d` indicates a directory, and `l` (lowercase *L*) indicates a symbolic link. An executable file (a script or binary file that runs as a command) has execute bits turned on (`x`). See more on execute bits in the upcoming section "Understanding File Permissions and Ownership."

You should become familiar with the contents of your home directory next. Use the `-l` and `-a` options to `ls`.

```
$ ls -la /home/joe
total 158
drwxrwxrwx 2     joe   sales   4096 May 12 13:55 .
drwxr-xr-x 3     root  root    4096 May 10 01:49 ..
-rw------- 1     joe   sales   2204 May 18 21:30 .bash_history
-rw-r--r-- 1     joe   sales     24 May 10 01:50 .bash_logout
-rw-r--r-- 1     joe   sales    230 May 10 01:50 .bash_profile
-rw-r--r-- 1     joe   sales    124 May 10 01:50 .bashrc
drw-r--r-- 1     joe   sales   4096 May 10 01:50 .kde
-rw-rw-r-- 1     joe   sales 149872 May 11 22:49 letter

^          ^     ^     ^     ^      ^            ^
col 1  col 2 col 3 col 4 col 5 col  6        col 7
```

Displaying a long list (`-l` option) of the contents of your home directory shows you more about file sizes and directories. The `total` line shows the total amount of disk space used by the files in the list (158 kilobytes in this example). Adding the all files option (`-a`) displays files that begin with a dot (`.`). Directories such as the current directory (`.`) and the parent directory (`..`)—the directory above the current directory—are noted as directories by the letter `d` at the beginning of each entry. Each directory begins with a `d` and each file begins with a dash (`-`).

The file and directory names are shown in column 7. In this example, a dot (.) represents /home/joe and two dots (..) represent /home—the parent directory of /joe. Most of the files in this example are dot (.) files that are used to store GUI properties (.kde directory) or shell properties (.bash files). The only non-dot file in this list is the one named letter. Column 3 shows the directory or file owner. The /home directory is owned by root, and everything else is owned by the user joe, who belongs to the sales group (groups are listed in column 4).

In addition to the d or -, column 1 on each line contains the permissions set for that file or directory. Other information in the listing includes the number of hard links to the item (column 2), the size of each file in bytes (column 5), and the date and time each file was most recently modified (column 6).

Here are a few other facts about file and directory listings:

- The number of characters shown for a directory (4096 bytes in these examples) reflects the size of the file containing information about the directory. Although this number can grow above 4096 bytes for a directory that contains lots of files, this number doesn't reflect the size of files contained in that directory.
- The format of the time and date column can vary. Instead of displaying "May 12," the date might be displayed as "2019-05-12," depending upon the distribution and the language setting (LANG variable).
- On occasion, instead of seeing the execute bit (x) set on an executable file, you may see an s in that spot instead. With an s appearing within either the owner (-rwsr-xr-x) or group (-rwxr-sr-x) permissions, or both (-rwsr-sr-x), the application can be run by any user, but ownership of the running process is assigned to the application's user/group instead of that of the user launching the command. This is referred to as a *set UID* or *set GID* program, respectively. For example, the mount command has permissions set as -rwsr-xr-x. This allows any user to run mount to list mounted filesystems (although you still have to be root to use mount to actually mount filesystems from the command line, in most cases).
- If a t appears at the end of a directory, it indicates that the *sticky bit* is set for that directory (for example, drwxrwxr-t). By setting the sticky bit on a directory, the directory's owner can allow other users and groups to add files to the directory but prevent users from deleting each other's files in that directory. With a set GID assigned to a directory, any files created in that directory are assigned the same group as the directory's group. (If you see a capital S or T instead of the execute bits on a directory, it means that the set GID or sticky bit permission, respectively, was set, but for some reason the execute bit was not also turned on.)
- If you see a plus sign at the end of the permission bits (for example, -rw-rw-r--+), it means that extended attributes (+), such as Access Control Lists (ACLs), are set on the file. A dot at the end (.) indicates that SELinux is set on the file.

Identifying Directories

When you need to identify your home directory on a shell command line, you can use the following:

$HOME	This environment variable stores your home directory name.
~	The tilde (~) represents your home directory on the command line.

You can also use the tilde to identify someone else's home directory. For example, `~joe` would be expanded to the `joe` home directory (probably `/home/joe`). So, if I wanted to go to the directory `/home/joe/test`, I could enter `cd ~joe/test` to get there.

Other special ways of identifying directories in the shell include the following:

.	A single dot (`.`) refers to the current directory.
..	Two dots (`..`) refer to a directory directly above the current directory.
$PWD	This environment variable refers to the current working directory.
$OLDPWD	This environment variable refers to the previous working directory before you changed to the current one. (Entering `cd -` returns you to the directory represented by `$OLDPWD`.)

As I mentioned earlier, there are many useful options for the `ls` command. Return to the `$HOME/test` directory in which you've been working. Here are some examples of `ls` options. Don't worry if the output doesn't exactly match what is in your directory at this point.

Any file or directory beginning with a dot (.) is considered hidden and is not displayed by default with `ls`. These dot files are typically configuration files or directories that need to be in your home directory but don't need to be seen in your daily work. The `-a` lets you see those files.

The `-t` option displays files in the order in which they were most recently modified. With the `-F` option, a backslash (/) appears at the end of directory names, an asterisk (*) is added to executable files, and an at sign (@) is shown next to symbolic links.

To show hidden and non-hidden files:

```
$ ls -a
. apple docs grapefruit pointer_to_apple .stuff watermelon
.. banana grape .hiddendir script.sh .tmpfile
```

To list all files by time most recently modified:

```
$ ls -at
.tmpfile .hiddendir .. docs watermelon banana script.sh
. .stuff pointer_to_apple grapefruit apple grape
```

To list files and append file-type indicators:

```
$ ls -F
apple banana docs/ grape grapefruit pointer_to_apple@ script.sh*
watermelon
```

To avoid displaying certain files or directories when you use `ls`, use the `--hide=` option. In the next set of examples, any file beginning with `g` does not appear in the output. Using a `-d` option on a directory shows information about that directory instead of showing the files and directories the directory contains. The `-R` option lists all files in the current directory as well as any files or directories that are associated with the original directory. The `-S` option lists files by size.

To exclude any files beginning with the letter `g` in the list:

```
$ ls --hide=g*
apple banana docs pointer_to_apple script.sh watermelon
```

To list info about a directory instead of the files it contains:

```
$ ls -ld $HOME/test/
drwxrwxr-x. 4 joe joe 4096 Dec 18 22:00 /home/joe/test/
```

To create multiple directory layers (`-p` is needed):

```
$ mkdir -p $HOME/test/documents/memos/
```

To list all files and directories recursively from current directory down:

```
$ ls -R
...
```

To list files by size:

```
$ ls -S
...
```

Understanding File Permissions and Ownership

After you've worked with Linux for a while, you are almost sure to get a `Permission denied` message. Permissions associated with files and directories in Linux were designed to keep users from accessing other users' private files and to protect important system files.

The nine bits assigned to each file for permissions define the access that you and others have to your file. Permission bits for a regular file appear as `-rwxrwxrwx`. Those bits are used to define who can read, write, or execute the file.

Note

For a regular file, a dash appears in front of the nine-bit permissions indicator. Instead of a dash, you might see a `d` (for a directory), `l` (for a symbolic link), `b` (for a block device), `c` (for a character device), `s` (for a socket), or `p` (for a named pipe).

Of the nine-bit permissions, the first three bits apply to the owner's permission, the next three apply to the group assigned to the file, and the last three apply to all others. The `r` stands for read, the `w` stands for write, and the `x` stands for execute permissions. If a dash appears instead of the letter, it means that permission is turned off for that associated read, write, or execute bit.

Because files and directories are different types of elements, read, write, and execute permissions on files and directories mean different things. Table 4.2 explains what you can do with each of them.

TABLE 4.2 Setting Read, Write, and Execute Permissions

Permission	File	Directory
Read	View what's in the file.	See what files and subdirectories it contains.
Write	Change the file's content, rename it, or delete it.	Add files or subdirectories to the directory. Remove files or directories from the directory.
Execute	Run the file as a program.	Change to the directory as the current directory, search through the directory, or execute a program from the directory. Access file metadata (file size, time stamps, and so on) of files in that directory.

As noted earlier, you can see the permission for any file or directory by typing the `ls -ld` command. The named file or directory appears as those shown in this example:

```
$ ls -ld ch3 test
-rw-rw-r-- 1 joe sales 4983 Jan 18 22:13 ch3
drwxr-xr-x 2 joe sales 1024 Jan 24 13:47 test
```

The first line shows that the `ch3` file has read and write permission for the owner and the group. All other users have read permission, which means that they can view the file but cannot change its contents or remove it. The second line shows the `test` directory (indicated by the letter `d` before the permission bits). The owner has read, write, and execute permissions while the group and other users have only read and execute permissions. As a result, the owner can add, change, or delete files in that directory, and everyone else can only read the contents, change to that directory, and list the contents of the directory. (If you had not used the `-d` options to `ls`, you would have listed files in the test directory instead of permissions of that directory.)

Changing permissions with chmod (numbers)

If you own a file, you can use the `chmod` command to change the permission on it as you please. In one method of doing this, each permission (read, write, and execute) is assigned a number—r=4, w=2, and x=1—and you use each set's total number to establish the

permission. For example, to make permissions wide open for yourself as owner, you would set the first number to 7 (4+2+1), and then you would give the group and others read-only permission by setting both the second and third numbers to 4 (4+0+0), so that the final number is 744. Any combination of permissions can result from 0 (no permission) through 7 (full permission).

Here are some examples of how to change permission on a file (named `file`) and what the resulting permission would be:

The following `chmod` command results in this permission: `rwxrwxrwx`

```
# chmod 777 file
```

The following `chmod` command results in this permission: `rwxr-xr-x`

```
# chmod 755 file
```

The following `chmod` command results in this permission: `rw-r--r--`

```
# chmod 644 file
```

The following `chmod` command results in this permission: `---------`

```
# chmod 000 file
```

The `chmod` command also can be used recursively. For example, suppose that you wanted to give an entire directory structure 755 permission (`rwxr-xr-x`), starting at the `$HOME/myapps` directory. To do that, you could use the `-R` option, as follows:

```
$ chmod -R 755 $HOME/myapps
```

All files and directories below, and including, the `myapps` directory in your home directory will have 755 permissions set. Because the numbers approach to setting permission changes all permission bits at once, it's more common to use letters to change permission bits recursively over a large set of files.

Changing permissions with chmod (letters)

You can also turn file permissions on and off using plus (+) and minus (–) signs, respectively, along with letters to indicate what changes and for whom. Using letters, for each file you can change permission for the user (`u`), group (`g`), other (`o`), and all users (`a`). What you would change includes the read (`r`), write (`w`), and execute (`x`) bits. For example, start with a file that has all permissions open (`rwxrwxrwx`). Run the following `chmod` commands using minus sign options. The resulting permissions are shown to the right of each command.

The following `chmod` command results in this permission: `r-xr-xr-x`

```
$ chmod a-w file
```

The following `chmod` command results in this permission: `rwxrwxrw-`

```
$ chmod o-x file
```

The following `chmod` command results in this permission: `rwx------`

```
$ chmod go-rwx file
```

Likewise, the following examples start with all permissions closed (---------). The plus sign is used with chmod to turn permissions on.

The following chmod command results in this permission: rw-------

```
$ chmod u+rw files
```

The following chmod command results in this permission: --x--x--x

```
$ chmod a+x files
```

The following chmod command results in this permission: r-xr-x---

```
$ chmod ug+rx files
```

Using letters to change permission recursively with chmod generally works better than using numbers because you can change bits selectively instead of changing all permission bits at once. For example, suppose that you want to remove write permission for "other" without changing any other permission bits on a set of files and directories. You could do the following:

```
$ chmod -R o-w $HOME/myapps
```

This example recursively removes write permissions for "other" on any files and directories below the myapps directory. If you had used numbers such as 644, execute permission would be turned off for directories; using 755, execute permission would be turned on for regular files. Using o-w, only one bit is turned off and all other bits are left alone.

Setting default file permission with umask

When you create a file as a regular user, it's given permission rw-rw-r-- by default. A directory is given the permission rwxrwxr-x. For the root user, file and directory permission are rw-r--r-- and rwxr-xr-x, respectively. These default values are determined by the value of umask. Enter **umask** to see what your umask value is. For example:

```
$ umask
0002
```

If you ignore the leading zero for the moment, the umask value masks what is considered to be fully opened permissions for a file 666 or a directory 777. The umask value of 002 results in permission for a directory of 775 (rwxrwxr-x). That same umask results in a file permission of 644 (rw-rw-r--). (Execute permissions are off by default for regular files.)

To change your umask value temporarily, run the umask command. Then try creating some files and directories to see how the umask value affects how permissions are set. For example:

```
$ umask 777 ; touch file01 ; mkdir dir01 ; ls -ld file01 dir01
d---------. 2 joe joe 6 Dec 19 11:03 dir01
----------. 1 joe joe 0 Dec 19 11:02 file01
$ umask 000 ; touch file02 ; mkdir dir02 ; ls -ld file02 dir02
```

```
drwxrwxrwx. 2 joe joe 6 Dec 19 11:00 dir02/
-rw-rw-rw-. 1 joe joe 0 Dec 19 10:59 file02
$ umask 022 ; touch file03 ; mkdir dir03 ; ls -ld file03 dir03
drwxr-xr-x. 2 joe joe 6 Dec 19 11:07 dir03
-rw-r--r--. 1 joe joe 0 Dec 19 11:07 file03
```

If you want to change your `umask` value permanently, add a `umask` command to the `.bashrc` file in your home directory (near the end of that file). The next time you open a shell, your `umask` is set to whatever value you chose.

Changing file ownership

As a regular user, you cannot change ownership of files or directories to have them belong to another user. You *can* change ownership as the root user. For example, suppose that you created a file called `memo.txt` in the user `joe`'s home directory while you were root user. Here's how you could change it to be owned by `joe`:

```
# chown joe /home/joe/memo.txt
# ls -l /home/joe/memo.txt
-rw-r--r--. 1 joe root 0 Dec 19 11:23 /home/joe/memo.txt
```

Notice that the `chown` command changed the user to `joe` but left the group as `root`. To change both user and group to `joe`, you could enter the following instead:

```
# chown joe:joe /home/joe/memo.txt
# ls -l /home/joe/memo.txt
-rw-r--r--. 1 joe joe 0 Dec 19 11:23 /home/joe/memo.txt
```

The `chown` command can be use recursively as well. Using the recursive option (`-R`) is helpful if you need to change a whole directory structure to ownership by a particular user. For example, if you inserted a USB drive, which is mounted on the `/media/myusb` directory, and you wanted to give full ownership of the contents of that drive to the user `joe`, you could enter the following:

```
# chown -R joe:joe /media/myusb
```

Moving, Copying, and Removing Files

Commands for moving, copying, and deleting files are fairly straightforward. To change the location of a file, use the `mv` command. To copy a file from one location to another, use the `cp` command. To remove a file, use the `rm` command. These commands can be used to act on individual files and directories or recursively to act on many files and directories at once. Here are some examples:

```
$ mv abc def
$ mv abc ~
$ mv /home/joe/mymemos/ /home/joe/Documents/
```

The first `mv` command moves the file `abc` to the file `def` in the same directory (essentially renaming it), whereas the second `mv` command moves the file `abc` to your home directory

(~). The next mv command moves the mymemos directory (and all its contents) to the /home/joe/Documents directory.

By default, the mv command overwrites any existing files if the file to which you are moving exists. However, many Linux systems alias the mv command so that it uses the -i option (which causes mv to prompt you before overwriting existing files). Here's how to check if that is true on your system:

```
$ alias mv
alias mv='mv -i'
```

Here are some examples of using the cp command to copy files from one location to another:

```
$ cp abc def
$ cp abc ~
$ cp -r /usr/share/doc/bash-completion* /tmp/a/
$ cp -ra /usr/share/doc/bash-completion* /tmp/b/
```

The first copy command (cp) copies abc to the new name def in the same directory, whereas the second copies abc to your home directory (~), keeping the name abc. The two recursive (-r) copies copy the bash-completion directory and all of the files it contains, first to new /tmp/a/ and /tmp/b/ directories. If you run ls -l on those two directories, you see that for the cp command run with the archive (-a) option, the date/time stamps and permissions are maintained by the copy. Without the -a, current date/time stamps are used, and permissions are determined by your umask.

The cp command typically also is aliased with the -i option in order to prevent you from inadvertently overwriting files.

As with the cp and mv commands, rm is also usually aliased to include the -i option. This can prevent the damage that can come from an inadvertent recursive remove (-r) option. Here are some examples of the rm command:

```
$ rm abc
$ rm *
```

The first remove command deletes the abc file; the second removes all of the files in the current directory (except that it doesn't remove directories and/or any files that start with a dot). If you want to remove a directory, you need to use the recursive (-r) option to rm or, for an empty directory, you can use the rmdir command. Consider the following examples:

```
$ rmdir /home/joe/nothing/
$ rm -r /home/joe/bigdir/
$ rm -rf /home/joe/hugedir/
```

The rmdir command in the preceding code only removes the directory (nothing) if it is empty. The rm -r example removes the directory bigdir and all of its contents (files and multiple levels of subdirectories), but it prompts you before each is removed. When you add the force option (-f), the hugedir directory and all of its contents are immediately removed, without prompting.

CAUTION

When you override the `-i` option on the `mv`, `cp`, and `rm` commands, you risk removing some (or lots) of files by mistake. Using wildcards (such as `*`) and no `-i` makes mistakes even more likely. That said, sometimes you don't want to be bothered to step through each file you delete. You have other options as follows:

- As noted with the `-f` option, you can force `rm` to delete without prompting. An alternative is to run `rm`, `cp`, or `mv` with a backslash in front of it (`\rm bigdir`). The backslash causes any command to run unaliased.
- Another alternative with `mv` is to use the `-b` option. With `-b`, if a file of the same name exists at the destination, a backup copy of the old file is made before the new file is moved there.

Summary

Commands for moving around the filesystem, copying files, moving files, and removing files are among the most basic commands that you need to work from the shell. This chapter covers lots of commands for moving around and manipulating files as well as commands for changing ownership and permission.

The next chapter describes commands for editing and searching for files. These commands include the `vim`/`vi` text editors, the `find` command, and the `grep` command.

Exercises

Use these exercises to test your knowledge of efficient ways to get around the Linux filesystem and work with files and directories. When possible, try to use shortcuts to type as little as possible to get the desired results. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Create a directory in your home directory called `projects`. In the `projects` directory, create nine empty files that are named `house1`, `house2`, `house3`, and so on up to `house9`. Assuming that there are lots of other files in that directory, come up with a single argument to `ls` that would list just those nine files.
2. Make the `$HOME/projects/houses/doors/` directory path. Create the following empty files within this directory path (try using absolute and relative paths from your home directory):

```
$HOME/projects/houses/bungalow.txt
$HOME/projects/houses/doors/bifold.txt
$HOME/projects/outdoors/vegetation/landscape.txt
```

3. Copy the files `house1` and `house5` to the `$HOME/projects/houses/` directory.
4. Recursively copy the `/usr/share/doc/initscripts*` directory to the `$HOME/projects/` directory. Maintain the current date/time stamps and permissions.
5. Recursively list the contents of the `$HOME/projects/` directory. Pipe the output to the `less` command so that you can page through the output.
6. Remove the files `house6`, `house7`, and `house8` without being prompted.
7. Move `house3` and `house4` to the `$HOME/projects/houses/doors` directory.
8. Remove the `$HOME/projects/houses/doors` directory and its contents.
9. Change the permissions on the `$HOME/projects/house2` file so that it can be read by and written to by the user who owns the file, only read by the group, and have no permission for others.
10. Recursively change permissions of the `$HOME/projects/` directory so that nobody has write permission to any files or directory beneath that point in the filesystem.

CHAPTER 5

Working with Text Files

IN THIS CHAPTER

Using **vim** and **vi** to edit text files

Searching for files

Searching in files

When the UNIX system, on which Linux was based, was created, most information was managed on the system in plain-text files. Thus, it was critical for users to know how to use tools for searching for and within plain-text files and to be able to change and configure those files.

Today, configuration of Linux systems can still be done by editing plain-text files. Whether you are modifying files in the `/etc` directory to configure a local service or editing Ansible inventory files to configure sets of host computers, plain-text files are still in common use for those tasks.

Before you can become a full-fledged system administrator, you need to be able to use a plain-text editor. The fact that most professional Linux servers don't even have a graphical interface available makes the need for editing of plain-text configuration files with a non-graphical text editor necessary.

After you know how to edit text files, you still might find it tough to figure out where the files are located that you need to edit. With commands such as `find`, you can search for files based on various attributes (filename, size, modification date, and ownership to name a few). With the `grep` command, you can search inside of text files to find specific search terms.

Editing Files with vim and vi

It's almost impossible to use Linux for any period of time and not need a text editor because, as noted earlier, most Linux configuration files are plain-text files that you will almost certainly need to change manually at some point.

If you are using a GNOME desktop, you can run `gedit` (type **`gedit`** into the Search box and press Enter, or select Applications ➪ Accessories ➪ gedit), which is fairly intuitive for editing text.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

You can also run a simple text editor called `nano` from the shell. However, most Linux shell users use either the `vi` or `emacs` command to edit text files.

The advantage of `vi` or `emacs` over a graphical editor is that you can use the command from any shell, character terminal, or character-based connection over a network (using `telnet` or `ssh`, for example)—no graphical interface is required. They each also contain tons of features, so you can continue to grow with them.

The following sections provide a brief tutorial on the `vi` text editor, which you can use to manually edit a text file from any shell. It also describes the improved versions of `vi` called `vim`. (If `vi` doesn't suit you, see the sidebar "Exploring Other Text Editors" for further options.)

The `vi` editor is difficult to learn at first, but after you know it, you never have to use a mouse or a function key—you can edit and move around quickly and efficiently within files just by using the keyboard.

Exploring Other Text Editors

Dozens of text editors are available for use with Linux. Some alternatives might be in your Linux distribution. You can try them out if you find `vi` to be too taxing. Here are some of the options:

`nano`: This popular, streamlined text editor is used with many bootable Linux systems and other limited-space Linux environments. For example, `nano` is available to edit text files during a Gentoo Linux install process.

`gedit`: The GNOME text editor runs on the desktop.

`jed`: This screen-oriented editor was made for programmers. Using colors, `jed` can highlight code that you create so that you can easily read the code and spot syntax errors. Use the Alt key to select menus to manipulate your text.

`joe`: The joe editor is similar to many PC text editors. Use control and arrow keys to move around. Press Ctrl+C to exit with no save or Ctrl+X to save and exit.

`kate`: This nice-looking editor comes in the `kdebase` package. It has lots of bells and whistles, such as highlighting for different types of programming languages and controls for managing word wrap.

`kedit`: This GUI-based text editor comes with the KDE desktop.

`mcedit`: In this editor, function keys help you get around, save, copy, move, and delete text. Like `jed` and `joe`, `mcedit` is screen oriented. It comes in the mc package in RHEL and Fedora.

`nedit`: This is an excellent programmer's editor. You need to install the optional `nedit` package to get this editor.

If you use `ssh` to log in to other Linux computers on your network, you can use any available text editor to edit files. If you use `ssh -X` to connect to the remote system, a GUI-based editor pops up on your local screen. When no GUI is available, you need a text editor that runs in the shell, such as `vi`, `jed`, or `joe`.

Starting with vi

Most often, you start `vi` to open a particular file. For example, to open a file called `/tmp/test`, enter the following command:

```
$ vi /tmp/test
```

If this is a new file, you should see something similar to the following:

```
□
~
~
~
~
~
"/tmp/test" [New File]
```

A blinking box at the top represents where your cursor is located. The bottom line keeps you informed about what is going on with your editing (here, you just opened a new file). In between, there are tildes (~) as filler because there is no text in the file yet. Now here's the intimidating part: There are no hints, menus, or icons to tell you what to do. To make it worse, you can't just start typing. If you do, the computer is likely to beep at you. (And some people complain that Linux isn't friendly.)

First, you need to know the two main operating modes: command and input. The `vi` editor always starts in command mode. Before you can add or change text in the file, you have to type a command (one or two letters, sometimes preceded by an optional number) to tell `vi` what you want to do. Case is important, so use uppercase and lowercase exactly as shown in the examples!

NOTE

On Red Hat Enterprise Linux, Fedora, and other Linux distributions, for regular users the `vi` command is aliased to run `vim`. If you type `alias vi`, you should see `alias vi='vim'`. The first obvious difference between `vi` and `vim` is that any known text file type, such as HTML, C code, or a common configuration file, appears in color. The colors indicate the structure of the file. Other features of `vim` that are not in `vi` include features such as visual highlighting and split-screen mode. By default, the root user doesn't have `vi` aliased to `vim`. If `vim` is not on your system, try installing the vim-enhanced package.

Adding text

To get into input mode, type an input command letter. To begin, type any of the following letters. When you are finished inputting text, press the Esc key (sometimes twice) to return to command mode. Remember the Esc key!

- **a**: The *add* command. With this command, you can input text that starts to the *right* of the cursor.
- **A**: The *add at end* command. With this command, you can input text starting at the end of the current line.

- `i`: The *insert* command. With this command, you can input text that starts to the *left* of the cursor.
- `I`: The *insert at beginning* command. With this command, you can input text that starts at the beginning of the current line.
- `o`: The *open below* command. This command opens a line below the current line and puts you in insert mode.
- `O`: The *open above* command. This command opens a line above the current line and puts you in insert mode.

Tip

When you are in insert mode, `-- INSERT --` appears at the bottom of the screen.

Type a few words, and press Enter. Repeat that a few times until you have a few lines of text. When you're finished typing, press Esc to return to command mode. Now that you have a file with some text in it, try moving around in your text with the keys or letters described in the next section.

Tip

Remember the Esc key! It always places you back into command mode. Remember that sometimes you must press Esc twice. For example, if you type a colon (`:`) to go into `ex` mode, you must press Esc twice to return to command mode.

Moving around in the text

To move around in the text, you can use the up, down, right, and left arrows. However, many of the keys for moving around are right under your fingertips when they are in typing position:

- **Arrow keys**: Move the cursor up, down, left, or right in the file one character at a time. To move left and right, you can also use Backspace and the spacebar, respectively. If you prefer to keep your fingers on the keyboard, move the cursor with h (left), l (right), j (down), or k (up).
- **w**: Moves the cursor to the beginning of the next word (delimited by spaces, tabs, or punctuation).
- **W**: Moves the cursor to the beginning of the next word (delimited by spaces or tabs).
- **b**: Moves the cursor to the beginning of the previous word (delimited by spaces, tabs, or punctuation).
- **B**: Moves the cursor to the beginning of the previous word (delimited by spaces or tabs).
- **0 (zero)**: Moves the cursor to the beginning of the current line.
- **$**: Moves the cursor to the end of the current line.
- **H**: Moves the cursor to the upper-left corner of the screen (first line on the screen).

M: Moves the cursor to the first character of the middle line on the screen.

L: Moves the cursor to the lower-left corner of the screen (last line on the screen).

Deleting, copying, and changing text

The only other editing that you need to know is how to delete, copy, or change text. The `x`, `d`, `y`, and `c` commands can be used to delete and change text. These can be used along with movement keys (arrows, PgUp, PgDn, letters, and special keys) and numbers to indicate exactly what you are deleting, copying, or changing. Consider the following examples:

x: Deletes the character under the cursor.

X: Deletes the character directly before the cursor.

d<?>: Deletes some text.

c<?>: Changes some text.

y<?>: Yanks (copies) some text.

The `<?>` after each letter in the preceding list identifies the place where you can use a movement command to choose what you are deleting, changing, or yanking. For example:

dw: Deletes (`d`) a word (`w`) after the current cursor position.

db: Deletes (`d`) a word (`b`) before the current cursor position.

dd: Deletes (`d`) the entire current line (`d`).

c$: Changes (`c`) the characters (actually erases them) from the current character to the end of the current line (`$`) and goes into input mode.

c0: Changes (`c`) (again, erases) characters from the previous character to the beginning of the current line (`0`) and goes into input mode.

cl: Erases (`c`) the current letter (`l`) and goes into input mode.

cc: Erases (`c`) the line (`c`) and goes into input mode.

yy: Copies (`y`) the current line (`y`) into the buffer.

y): Copies (`y`) the current sentence (`)`), to the right of the cursor, into the buffer.

y}: Copies (`y`) the current paragraph (`}`), to the right of the cursor, into the buffer.

Any of the commands just shown can be further modified using numbers, as you can see in the following examples:

3dd: Deletes (`d`) three (`3`) lines (`d`), beginning at the current line.

3dw: Deletes (`d`) the next three (`3`) words (`w`).

5cl: Changes (`c`) the next five (`5`) letters (`l`) (that is, removes the letters and enters input mode).

12j: Moves down (`j`) 12 lines (`12`).

5cw: Erases (`c`) the next five (`5`) words (`w`) and goes into input mode.

4y): Copies (`y`) the next four (`4`) sentences (`)`).

Pasting (putting) text

After text has been copied to the buffer (by deleting, changing, or yanking it), you can place that text back in your file using the letter `p` or `P`. With both commands, the text most recently stored in the buffer is put into the file in different ways.

`P`: Puts the copied text to the left of the cursor if the text consists of letters or words; puts the copied text above the current line if the copied text contains lines of text.

`p`: Puts the buffered text to the right of the cursor if the text consists of letters or words; puts the buffered text below the current line if the buffered text contains lines of text.

Repeating commands

After you delete, change, or paste text, you can repeat that action by typing a period (.). For example, with the cursor on the beginning of the name **`Joe`**, you type **`cw`** and then type **`Jim`** to change `Joe` to `Jim`. You search for the next occurrence of `Joe` in the file, position the cursor at the beginning of that name, and press a period. The word changes to `Jim`, and you can search for the next occurrence. You can go through a file this way, pressing `n` to go to the next occurrence and period (.) to change the word.

Exiting vi

To wrap things up, use the following commands to save or quit the file:

`ZZ`: Saves the current changes to the file and exits from `vi`.

`:w`: Saves the current file, but you can continue editing.

`:wq`: Works the same as `ZZ`.

`:q`: Quits the current file. This works only if you don't have any unsaved changes.

`:q!`: Quits the current file and *doesn't* save the changes you just made to the file.

Tip

If you've really trashed the file by mistake, the `:q!` command is the best way to exit and abandon your changes. The file reverts to the most recently changed version. So, if you just saved with `:w`, you are stuck with the changes up to that point. However, despite having saved the file, you can type `u` to back out of changes (all the way back to the beginning of the editing session if you like) and then save again.

You have learned a few `vi` editing commands. I describe more commands in the following sections. First, however, consider the following tips to smooth out your first trials with `vi`:

Esc: Remember that Esc gets you back to command mode. (I've watched people press every key on the keyboard trying to get out of a file.) Esc followed by `ZZ` gets you out of command mode, saves the file, and exits.

u: Press `u` to undo the previous change you made. Continue to press u to undo the change before that and the one before that.

Ctrl+R: If you decide that you didn't want to undo the previous undo command, use Ctrl+R for Redo. Essentially, this command undoes your undo.

Caps Lock: Beware of hitting Caps Lock by mistake. Everything that you type in `vi` has a different meaning when the letters are capitalized. You don't get a warning that you are typing capitals; things just start acting weird.

`:!command`: You can run a shell command while you are in `vi` using **`:!`** followed by a shell command name. For example, type **`:!date`** to see the current date and time, type **`:!pwd`** to see what your current directory is, or type **`:!jobs`** to see whether you have any jobs running in the background. When the command completes, press Enter and you are back to editing the file. You could even use this technique to launch a shell (`:!bash`) from `vi`, run a few commands from that shell, and then type `exit` to return to `vi`. (I recommend doing a save before escaping to the shell, just in case you forget to go back to `vi`.)

Ctrl+g: If you forget what you are editing, pressing these keys displays the name of the file that you are editing and the current line that you are on at the bottom of the screen. It also displays the total number of lines in the file, the percentage of how far you are through the file, and the column number the cursor is on. This just helps you get your bearings after you've stopped for a cup of coffee at 3 a.m.

Skipping around in the file

Besides the few movement commands described earlier, there are other ways of moving around a `vi` file. To try these out, open a large file that you can't damage too much. (Try copying `/var/log/messages` to `/tmp` and opening it in `vi`.) Here are some movement commands that you can use:

Ctrl+f: Pages ahead one page at a time.

Ctrl+b: Pages back one page at a time.

Ctrl+d: Pages ahead one-half page at a time.

Ctrl+u: Pages back one-half page at a time.

G: Goes to the last line of the file.

1G: Goes to the first line of the file.

35G: Goes to any line number (35, in this case).

Searching for text

To search for the next or previous occurrence of text in the file, use either the slash (/) or the question mark (?) character. Follow the slash or question mark with a pattern (string of text) to search forward or backward, respectively, for that pattern. Within the search, you can also use metacharacters. Here are some examples:

`/hello`: Searches forward for the word `hello`.

`?goodbye`: Searches backward for the word `goodbye`.

/The.*foot: Searches forward for a line that has the word `The` in it and also, after that at some point, the word `foot`.

?[pP]rint: Searches backward for either `print` or `Print`. Remember that case matters in Linux, so make use of brackets to search for words that could have different capitalization.

After you have entered a search term, simply type **n** or **N** to search again in the same direction (`n`) or the opposite direction (`N`) for the term.

Using ex mode

The `vi` editor was originally based on the `ex` editor, which didn't let you work in full-screen mode. However, it did enable you to run commands that let you find and change text on one or more lines at a time. When you type a colon and the cursor goes to the bottom of the screen, you are essentially in `ex` mode. The following are examples of some of those `ex` commands for searching for and changing text. (I chose the words `Local` and `Remote` to search for, but you can use any appropriate words.)

:g/Local: Searches for the word `Local` and prints every occurrence of that line from the file. (If there is more than a screenful, the output is piped to the `more` command.)

:s/Local/Remote: Substitutes `Remote` for the first occurrence of the word `Local` on the current line.

:g/Local/s//Remote: Substitutes the first occurrence of the word `Local` on every line of the file with the word `Remote`.

:g/Local/s//Remote/g: Substitutes every occurrence of the word `Local` with the word `Remote` in the entire file.

:g/Local/s//Remote/gp: Substitutes every occurrence of the word `Local` with the word `Remote` in the entire file and then prints each line so that you can see the changes (piping it through `less` if output fills more than one page).

Learning more about vi and vim

To learn more about the `vi` editor, try typing **vimtutor**. The `vimtutor` command opens a tutorial in the `vim` editor that steps you through common commands and features you can use in `vim`. To use `vimtutor`, install the `vim`-enhanced package.

Finding Files

Even a basic Linux installation can have thousands of files installed on it. To help you find files on your system, you can use commands such as `locate` (to find commands by name), `find` (to find files based on lots of different attributes), and `grep` (to search within text files to find lines in files that contain search text).

Using locate to find files by name

On most Linux systems (Fedora and RHEL included), the `updatedb` command runs once per day to gather the names of files throughout your Linux system into a database. By running the `locate` command, you can search that database to find the location of files stored in it.

Here are a few things that you should know about searching for files using the `locate` command:

- There are advantages and disadvantages to using `locate` to find filenames instead of the `find` command. A `locate` command finds files faster because it searches a database instead of having to search the whole filesystem live. A disadvantage is that the `locate` command cannot find any files added to the system since the previous time the database was updated.
- Not every file in your filesystem is stored in the database. The contents of the `/etc/updatedb.conf` file limit which filenames are collected by pruning out select mount types, filesystem types, file types, and mount points. For example, filenames are not gathered from remotely mounted filesystems (`cifs`, `nfs`, and so on) or locally mounted CDs or DVDs (`iso9660`). Paths containing temporary files (`/tmp`) and spool files (`/var/spool/cups`) are also pruned. You can add items to prune (or remove some items that you don't want pruned) the locate database to your needs. In RHEL 8, the `updatedb.conf` file contains the following:

```
PRUNE_BIND_MOUNTS = "yes"
PRUNEFS = "9p afs anon_inodefs auto autofs bdev binfmt_misc cgroup
cifs coda configfs cpuset debugfs devpts ecryptfs exofs fuse fuse
.sshfs fusectl gfs gfs2 gpfs hugetlbfs inotifyfs iso9660 jffs2
lustre mqueue ncpfs nfs nfs4 nfsd pipefs proc ramfs rootfs rpc_
pipefs securityfs selinuxfs sfs sockfs sysfs tmpfs ubifs udf usbfs
ceph fuse.ceph"
PRUNENAMES = ".git .hg .svn .bzr .arch-ids {arch} CVS"
PRUNEPATHS = "/afs /media /mnt /net /sfs /tmp /udev /var/cache/
ccache /var/lib/yum/yumdb /var/lib/dnf/yumdb /var/spool/cups /var/
spool/squid /var/tmp /var/lib/ceph"
```

As a regular user, you can't see any files from the locate database that you can't see in the filesystem normally. For example, if you can't type `ls` to view files in the `/root` directory, you can't locate files stored in that directory.

- When you search for a string, the string can appear anywhere in a file's path. For example, if you search for `passwd`, you could turn up `/etc/passwd`, `/usr/bin/passwd`, `/home/chris/passwd/pwdfiles.txt`, and many other files with `passwd` in the path.
- If you add files to your system after `updatedb` runs, you can't locate those files until `updatedb` runs again (probably that night). To get the database to contain all files up to the current moment, you can simply run `updatedb` from the shell as root.

Here are some examples of using the `locate` command to search for files:

```
$ locate .bashrc
/etc/skel/.bashrc
/home/cnegus/.bashrc
# locate .bashrc
/etc/skel/.bashrc
/home/bill/.bashrc
/home/joe/.bashrc
/root/.bashrc
```

When run as a regular user, `locate` only finds `.bashrc` in `/etc/skel` and the user's own home directory. Run as root, the same command locates `.bashrc` files in everyone's home directory.

```
$ locate dir_color
/usr/share/man/man5/dir_colors.5.gz
...
$ locate -i dir_color
/etc/DIR_COLORS
/etc/DIR_COLORS.256color
/etc/DIR_COLORS.lightbgcolor
/usr/share/man/man5/dir_colors.5.gz
```

Using locate `-i`, filenames are found regardless of case. So in the previous example, `DIR_COLORS` was found with `-i` whereas it wasn't found without the `-i` option.

```
$ locate services
/etc/services
/usr/share/services/bmp.kmgio
/usr/share/services/data.kmgio
```

Unlike the `find` command, which uses the `-name` option to find filenames, the `locate` command locates the string you enter if it exists in any part of the file's path. In this example, searching for `services` using the `locate` command finds files and directories containing the `services` text string.

Searching for files with find

The `find` command is the best one for searching your filesystem for files based on a variety of attributes. After files are found, you can act on those files as well (using the `-exec` or `-okay` option) by running any commands you want on them.

When you run `find`, it searches your filesystem live, which causes it to run slower than `locate`, but it gives you an up-to-the-moment view of the files on your Linux system. However, you can also tell `find` to start at a particular point in the filesystem so that the search can go faster by limiting the area of the filesystem being searched.

Nearly any file attribute that you can think of can be used as a search option. You can search for filenames, ownership, permission, size, modification times, and other attributes. You can even use combinations of attributes. Here are some basic examples of using the `find` command:

```
$ find
$ find /etc
# find /etc
$ find $HOME -ls
```

Run on a line by itself, the `find` command finds all files and directories below the current directory. If you want to search from a particular point in the directory tree, just add the name of the directory you want to search (such as `/etc`). As a regular user, `find` does not give you special permission to find files that have permissions that make them readable only by the root user. So, `find` produces a bunch of error messages. Run as the root user, `find /etc` finds all files under `/etc`.

A special option to the `find` command is `-ls`. A long listing (ownership, permission, size, and so on) is printed with each file when you add `-ls` to the `find` command (similar to output of the `ls -l` command). This option helps you in later examples when you want to verify that you have found files that contain the ownership, size, modification times, or other attributes that you are trying to find.

Note

If, as a regular user, you are searching an area of the filesystem where you don't have full permission to access all of the files it contains (such as the `/etc` directory), you might receive lots of error messages when you search with `find`. To get rid of those messages, direct standard errors to `/dev/null`. To do that, add the following to the end of the command line: `2> /dev/null`. The `2>` redirects standard error to the next option (in this case `/dev/null`, where the output is discarded).

Finding files by name

To find files by name, you can use the `-name` and `-iname` options. The search is done by base name of the file; the directory names are not searched by default. To make the search more flexible, you can use file-matching characters, such as asterisks (`*`) and question marks (`?`), as in the following examples:

```
# find /etc -name passwd
/etc/pam.d/passwd
/etc/passwd
# find /etc -iname '*passwd*'
/etc/pam.d/passwd
/etc/passwd-
/etc/passwd.OLD
/etc/passwd
/etc/MYPASSWD
/etc/security/opasswd
```

Using the -name option and no asterisks, the first example above lists any files in the /etc directory that are named passwd exactly. By using -iname instead, you can match any combination of upper- and lowercase. Using asterisks, you can match any filename that includes the word passwd.

Finding files by size

If your disk is filling up and you want to find out where your biggest files are located, you can search your system by file size. The -size option enables you to search for files that are exactly, smaller than, or larger than a selected size, as you can see in the following examples:

```
$ find /usr/share/ -size +10M
$ find /mostlybig -size -1M
$ find /bigdata -size +500M -size -5G -exec du -sh {} \;
4.1G    /bigdata/images/rhel6.img
606M    /bigdata/Fedora-16-i686-Live-Desktop.iso
560M    /bigdata/dance2.avi
```

The first example in the preceding code finds files larger than 10MB. The second finds files less than 1MB. In the third example, I'm searching for files that are between 500MB and 5GB. This includes an example of the -exec option (which I describe later) to run the du command on each file to see its size.

Finding files by user

You can search for a particular owner (-user) or group (-group) when you try to find files. By using -not and -or, you can refine your search for files associated with specific users and groups, as you can see in the following examples:

```
$ find /home -user chris -ls
131077    4 -rw-r--r--  1 chris   chris 379 Jun 29  2014 ./.bashrc
# find /home \( -user chris -or -user joe \) -ls
131077    4 -rw-r--r--  1 chris   chris 379 Jun 29  2014 ./.bashrc
181022    4 -rw-r--r--  1 joe     joe    379 Jun 15  2014 ./.bashrc
# find /etc -group ntp -ls
131438    4 drwxrwsr-x  3 root    ntp   4096 Mar  9 22:16 /etc/ntp
# find /var/spool -not -user root -ls
262100    0 -rw-rw----   1 rpc      mail    0 Jan 27  2014 /var/spool/mail/rpc
278504    0 -rw-rw----   1 joe      mail    0 Apr  3  2014 /var/spool/mail/joe
261230    0 -rw-rw----   1 bill     mail    0 Dec 18 14:17 /var/spool/mail/bill
277373 2848 -rw-rw----   1 chris    mail 8284 Mar 15  2014 /var/spool/mail/chris
```

The first example outputs a long listing of all of the files under the /home directory that are owned by the user chris. The next lists files owned by chris or joe. The find command of /etc turns up all files that have ntp as their primary group assignment. The last example shows all files under /var/spool that are not owned by root. You can see files owned by other users in the sample output.

Finding files by permission

Searching for files by permission is an excellent way to turn up security issues on your system or uncover access issues. Just as you changed permissions on files using numbers or letters (with the `chmod` command), you can likewise find files based on number or letter permissions along with the `-perm` options. (Refer to Chapter 4, "Moving Around the Filesystem," to see how to use numbers and letters with `chmod` to reflect file permissions.)

If you use numbers for permission, as I do below, remember that the three numbers represent permissions for the user, group, and other. Each of those three numbers varies from no permission (0) to full read/write/execute permission (7) by adding read (4), write (2), and execute (1) bits together. With a hyphen (-) in front of the number, all three of the bits indicated must match; with a forward slash (/) in front of it, any of the numbers can match for the search to find a file. The full, exact numbers must match if neither a hyphen nor a forward slash is used.

Consider the following examples:

```
$ find /usr/bin -perm 755 -ls
788884   28 -rwxr-xr-x   1 root     root        28176 Mar 10  2014 /bin/echo

$ find /home/chris/ -perm -222 -type d -ls
144503    4 drwxrwxrwx   8 chris  chris 4096 Jun 23  2014 /home/chris/OPENDIR
```

By searching for `-perm 755`, any files or directories with exactly `rwxr-xr-x` permission are matched. By using `-perm -222`, only files that have write permission for user, group, and other are matched. Notice that, in this case, the `-type d` is added to match only directories.

```
$ find /myreadonly -perm /222 -type f
685035    0 -rw-rw-r--  1 chris   chris      0 Dec 30 16:34 /myreadonly/abc

$ find . -perm -002 -type f -ls
266230    0 -rw-rw-rw-   1 chris   chris      0 Dec 30 16:28 ./LINUX_BIBLE/abc
```

Using `-perm /222`, you can find any file (`-type f`) that has write permission turned on for the user, group, or other. You might do that to make sure that all files are read-only in a particular part of the filesystem (in this case, beneath the `/myreadonly` directory). The last example, `-perm /002`, is very useful for finding files that have open write permission for "other," regardless of how the other permission bits are set.

Finding files by date and time

Date and time stamps are stored for each file when it is created, when it is accessed, when its content is modified, or when its metadata is changed. Metadata includes owner, group, time stamp, file size, permissions, and other information stored in the file's inode. You might want to search for file data or metadata changes for any of the following reasons:

- You just changed the contents of a configuration file, and you can't remember which one. So, you search `/etc` to see what has changed in the past 10 minutes:

```
$ find /etc/ -mmin -10
```

- You suspect that someone hacked your system three days ago. So, you search the system to see if any commands have had their ownership or permissions changed in the past three days:

```
$ find /bin /usr/bin /sbin /usr/sbin -ctime -3
```

- You want to find files in your FTP server (`/var/ftp`) and web server (`/var/www`) that have not been accessed in more than 300 days so that you can see if any need to be deleted:

```
$ find /var/ftp /var/www -atime +300
```

As you can glean from the examples, you can search for content or metadata changes over a certain number of days or minutes. The time options (`-atime`, `-ctime`, and `-mtime`) enable you to search based on the number of days since each file was accessed, changed, or had its metadata changed. The `min` options (`-amin`, `-cmin`, and `-mmin`) do the same in minutes.

Numbers that you give as arguments to the `min` and `time` options are preceded by a hyphen (to indicate a time from the current time to that number of minutes or days ago) or a plus (to indicate time from the number of minutes or days ago and older). With no hyphen or plus, the exact number is matched.

Using 'not' and 'or' when finding files

With the `-not` and `-or` options, you can further refine your searches. There may be times when you want to find files owned by a particular user but not assigned to a particular group. You may want files larger than a certain size but smaller than another size. Or you might want to find files owned by any of several users. The `-not` and `-or` options can help you do that. Consider the following examples:

- There is a shared directory called `/var/allusers`. This command line enables you to find files that are owned by either `joe` or `chris`.

```
$ find /var/allusers \( -user joe -o -user chris \) -ls
679967    0 -rw-r--r-- 1 chris chris    0 Dec 31 12:57
  /var/allusers/myjoe
679977 1812 -rw-r--r-- 1 joe   joe   4379 Dec 31 13:09
  /var/allusers/dict.dat
679972    0 -rw-r--r-- 1 joe   sales    0 Dec 31 13:02
  /var/allusers/one
```

- This command line searches for files owned by the user `joe`, but only those that are not assigned to the group `joe`:

```
$ find /var/allusers/ -user joe -not -group joe -ls
679972 0 -rw-r--r-- 1 joe sales  0 Dec 31 13:02 /var/allusers/one
```

- You can also add multiple requirements on your searches. Here, a file must be owned by the user `joe` and must also be more than 1MB in size:

```
$ find /var/allusers/ -user joe -and -size +1M -ls
679977 1812 -rw-r--r-- 1 joe root 1854379 Dec 31 13:09
  /var/allusers/dict.dat
```

Finding files and executing commands

One of the most powerful features of the `find` command is the capability to execute commands on any files that you find. With the `-exec` option, the command you use is executed on every file found, without stopping to ask if that's okay. The `-ok` option stops at each matched file and asks whether you want to run the command on it.

The advantage of using `-ok` is that, if you are doing something destructive, you can make sure that you okay each file individually before the command is run on it. The syntax for using `-exec` and `-ok` is the same:

```
$ find [options] -exec command {} \;
$ find [options] -ok command {} \;
```

With `-exec` or `-ok`, you run `find` with any options you like in order to find the files you are seeking. Then you enter the `-exec` or `-ok` option followed by the command you want to run on each file. The set of curly braces indicates where on the command line to read in each file that is found. Each file can be included in the command line multiple times if you like. To end the line, you need to add a backslash and semicolon (`\;`). Here are some examples:

- This command finds any file named `passwd` under the `/etc` directory and includes that name in the output of an `echo` command:

```
$ find /etc -iname passwd -exec echo "I found {}" \;
I found /etc/pam.d/passwd
I found /etc/passwd
```

- The following command finds every file under the `/usr/share` directory that is more than 5MB in size. Then it lists the size of each file with the `du` command. The output of `find` is then sorted by size, from largest to smallest. With `-exec` entered, all entries found are processed, without prompting:

```
$ find /usr/share -size +5M -exec du {} \; | sort -nr
116932  /usr/share/icons/HighContrast/icon-theme.cache
69048   /usr/share/icons/gnome/icon-theme.cache
20564   /usr/share/fonts/cjkuni-uming/uming.ttc
```

- The `-ok` option enables you to choose, one at a time, whether each file found is acted upon by the command you enter. For example, you want to find all files that belong to `joe` in the `/var/allusers` directory (and its subdirectories) and move them to the `/tmp/joe` directory:

```
# find /var/allusers/ -user joe -ok mv {} /tmp/joe/ \;
< mv ... /var/allusers/dict.dat > ? y
< mv ... /var/allusers/five > ? y
```

Notice in the preceding code that you are prompted for each file that is found before it is moved to the /tmp/joe directory. You would simply type **y** and press Enter at each line to move the file, or just press Enter to skip it.

For more information on the find command, enter **man find**.

Searching in files with grep

If you want to search for files that contain a certain search term, you can use the grep command. With grep, you can search a single file or search a whole directory structure of files recursively.

When you search, you can have every line containing the term printed on your screen (standard output) or just list the names of the files that contain the search term. By default, grep searches text in a case-sensitive way, although you can do case-insensitive searches as well.

Instead of just searching files, you can also use grep to search standard output. So, if a command turns out lots of text and you want to find only lines that contain certain text, you can use grep to filter just want you want.

Here are some examples of grep command lines used to find text strings in one or more files:

```
$ grep desktop /etc/services
desktop-dna    2763/tcp              # Desktop DNA
desktop-dna    2763/udp              # Desktop DNA

$ grep -i desktop /etc/services
sco-dtmgr      617/tcp               # SCO Desktop Administration Server
sco-dtmgr      617/udp               # SCO Desktop Administration Server
airsync        2175/tcp              # Microsoft Desktop AirSync Protocol
...
```

In the first example, a grep for the word desktop in the /etc/services file turned up two lines. Searching again, using the -i to be case-insensitive (as in the second example), there were 29 lines of text produced.

To search for lines that don't contain a selected text string, use the -v option. In the following example, all lines from the /etc/services file are displayed except those containing the text tcp (case-insensitive):

```
$ grep -vi tcp /etc/services
```

To do recursive searches, use the -r option and a directory as an argument. The following example includes the -l option, which just lists files that include the search text, without showing the actual lines of text. That search turns up files that contain the text peerdns (case-insensitive).

```
$ grep -rli peerdns /usr/share/doc/
/usr/share/doc/dnsmasq-2.66/setup.html
/usr/share/doc/initscripts-9.49.17/sysconfig.txt
...
```

The next example recursively searches the /etc/sysconfig directory for the term root. It lists every line in every file beneath the directory that contains that text. To make it easier to have the term root stand out on each line, the --color option is added. By default, the matched term appears in red.

```
$ grep -ri --color root /etc/sysconfig/
```

To search the output of a command for a term, you can pipe the output to the grep command. In this example, I know that IP addresses are listed on output lines from the ip command that include the string inet, so I use grep to display just those lines:

```
$ ip addr show | grep inet
inet 127.0.0.1/8 scope host lo
inet 192.168.1.231/24 brd 192.168.1.255 scope global wlan0
```

Summary

Being able to work with plain-text files is a critical skill for using Linux. Because so many configuration files and document files are in plain-text format, you need to become proficient with a text editor to use Linux effectively. Finding filenames and content in files are also critical skills. In this chapter, you learned to use the locate and find commands for finding files and grep for searching files.

The next chapter covers a variety of ways to work with processes. There, you learn how to see what processes are running, run processes in the foreground and background, and change processes (send signals).

Exercises

Use these exercises to test your knowledge of using the vi (or vim) text editor, commands for finding files (locate and find), and commands for searching files (grep). These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Copy the /etc/services file to the /tmp directory. Open the /tmp/services file in vim, and search for the term WorldWideWeb. Change that to read World Wide Web.
2. Find the following paragraph in your /tmp/services file (if it is not there, choose a different paragraph) and move it to the end of that file.

```
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, "Assigned Numbers" (October 1994). Not all ports
# are included, only the more common ones.
```

3. Using `ex` mode, search for every occurrence of the term `tcp` (case-sensitive) in your `/tmp/services` file and change it to `WHATEVER`.
4. As a regular user, search the `/etc` directory for every file named `passwd`. Redirect error messages from your search to `/dev/null`.
5. Create a directory in your home directory called `TEST`. Create files in that directory named `one`, `two`, and `three` that have full read/write/execute permissions on for everyone (user, group, and other). Construct a `find` command to find those files and any other files that have write permission open to "others" from your home directory and below.
6. Find files under the `/usr/share/doc` directory that have not been modified in more than 300 days.
7. Create a `/tmp/FILES` directory. Find all files under the `/usr/share` directory that are more than 5MB and less than 10MB and copy them to the `/tmp/FILES` directory.
8. Find every file in the `/tmp/FILES` directory, and make a backup copy of each file in the same directory. Use each file's existing name, and just append `.mybackup` to create each backup file.
9. Install the `kernel-doc` package in Fedora or Red Hat Enterprise Linux. Using `grep`, search inside the files contained in the `/usr/share/doc/kernel-doc*` directory for the term `e1000` (case-insensitive) and list the names of the files that contain that term.
10. Search for the `e1000` term again in the same location, but this time list every line that contains the term and highlight the term in color.

CHAPTER 6

Managing Running Processes

IN THIS CHAPTER

Displaying processes

Running processes in the foreground and background

Killing and renicing processes

In addition to being a multiuser operating system, Linux is a multitasking system. *Multitasking* means that many programs can be running at the same time. An instance of a running program is referred to as a *process*. Linux provides tools for listing running processes, monitoring system usage, and stopping (or killing) processes when necessary.

From a shell, you can launch processes and then pause, stop, or kill them. You can also put them in the background and bring them to the foreground. This chapter describes tools such as `ps`, `top`, `kill`, `jobs`, and other commands for listing and managing processes.

Understanding Processes

A process is a running instance of a command. For example, there may be one `vi` command on the system. But if `vi` is currently being run by 15 different users, that command is represented by 15 different running processes.

A process is identified on the system by what is referred to as a *process ID (PID)*. That PID is unique for the current system. In other words, no other process can use that number as its process ID while that first process is still running. However, after a process has ended, another process can reuse that number.

Along with a process ID number, other attributes are associated with a process. Each process, when it is run, is associated with a particular user account and group account. That account information helps determine what system resources the process can access. For example, a process run as the root user has much more access to system files and resources than a process running as a regular user.

The ability to manage processes on your system is critical for a Linux system administrator. Sometimes, runaway processes may be killing your system's performance. Finding and dealing with processes, based on attributes such as memory and CPU usage, are covered in this chapter.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Note

Commands that display information about running processes get most of that information from raw data stored in the `/proc` file system. Each process stores its information in a subdirectory of `/proc`, named after the process ID of that process. You can view some of that raw data by displaying the contents of files in one of those directories (using `cat` or `less` commands).

Listing Processes

From the command line, the `ps` command is the oldest and most common command for listing processes currently running on your system. The Linux version of `ps` contains a variety of options from old UNIX and BSD systems, some of which are conflicting and implemented in nonstandard ways. See the `ps` man page for descriptions of those different options.

The `top` command provides a more screen-oriented approach to listing processes, and it can also be used to change the status of processes. If you are using the GNOME desktop, you can use the System Monitor tool (`gnome-system-monitor`) to provide a graphical means of working with processes. These commands are described in the following sections.

Listing processes with ps

The most common utility for checking running processes is the `ps` command. Use it to see which programs are running, the resources they are using, and who is running them. The following is an example of the `ps` command:

```
$ ps u
USER   PID %CPU %MEM  VSZ    RSS   TTY    STAT  START  TIME  COMMAND
jake   2147 0.0  0.7 1836   1020   tty1   S+    14:50  0:00  -bash
jake   2310 0.0  0.7 2592    912   tty1   R+    18:22  0:00  ps u
```

In this example, the `u` option (equivalent to `-u`) asks that usernames be shown, as well as other information such as the time the process started and memory and CPU usage for processes associated with the current user. The processes shown are associated with the current terminal (`tty1`). The concept of a terminal comes from the old days when people worked exclusively from character terminals, so a terminal typically represented a single person at a single screen. Nowadays, you can have many "terminals" on one screen by opening multiple virtual terminals or Terminal windows on the desktop.

In this shell session, not much is happening. The first process shows that the user named `jake` opened a bash shell after logging in. The next process shows that `jake` has run the `ps u` command. The terminal device `tty1` is being used for the login session. The `STAT` column represents the state of the process, with `R` indicating a currently running process and `S` representing a sleeping process.

NOTE
Several other values can appear under the STAT column. For example, a plus sign (+) indicates that the process is associated with the foreground operations.

The USER column shows the name of the user who started the process. Each process is represented by a unique ID number referred to as a process ID, or PID. You can use the PID if you ever need to kill a runaway process or send another kind of signal to a process. The %CPU and %MEM columns show the percentages of the processor and random access memory, respectively, that the process is consuming.

VSZ (virtual set size) shows the size of the image process (in kilobytes), and *RSS (resident set size)* shows the size of the program in memory. The VSZ and RSS sizes may be different because VSZ is the amount of memory allocated for the process, whereas RSS is the amount that is actually being used. RSS memory represents physical memory that cannot be swapped.

START shows the time the process began running, and TIME shows the cumulative system time used. (Many commands consume very little CPU time, as reflected by 0:00 for processes that haven't even used a whole second of CPU time.)

Many processes running on a computer are not associated with a terminal. A normal Linux system has many processes running in the background. Background system processes perform such tasks as logging system activity or listening for data coming in from the network. They are often started when Linux boots up and run continuously until the system shuts down. Likewise, logging into a Linux desktop causes many background processes to kick off, such as processes for managing audio, desktop panels, authentication, and other desktop features.

To page through all of the processes running on your Linux system for the current user, add the pipe (|) and the less command to ps ux:

```
$ ps ux | less
```

To page through all processes running for all users on your system, use the ps aux command as follows:

```
$ ps aux | less
```

A pipe (located above the backslash character on the keyboard) enables you to direct the output of one command to be the input of the next command. In this example, the output of the ps command (a list of processes) is directed to the less command, which enables you to page through that information. Use the spacebar to page through and type **q** to end the list. You can also use the arrow keys to move one line at a time through the output.

The ps command can be customized to display selected columns of information and to sort information by one of those columns. Using the -o option, you can use keywords to indicate the columns you want to list with ps. For example, the next example lists every

running process (-e) and then follows the -o option with every column of information I want to display, including the process ID (pid), username (user), user ID (uid), group name (group), group ID (gid), virtual memory allocated (vsz), resident memory used (rss), and the full command line that was run (comm). By default, output is sorted by process ID number.

```
$ ps -eo pid,user,uid,group,gid,vsz,rss,comm | less
  PID  USER       UID GROUP      GID    VSZ   RSS COMMAND
    1  root         0 root         0 187660 13296 systemd
    2  root         0 root         0      0     0 kthreadd
```

If you want to sort by a specific column, you can use the sort= option. For example, to see which processes are using the most memory, I sort by the vsz field. That sorts from lowest memory use to highest. Because I want to see the highest ones first, I put a hyphen in front of that option to sort (sort=-vsz).

```
$ ps -eo pid,user,group,gid,vsz,rss,comm --sort=-vsz | head
    PID USER     GROUP      GID    VSZ   RSS COMMAND
 2366 chris    chris     1000 3720060 317060 gnome-shell
 1580 gdm      gdm         42 3524304 205796 gnome-shell
 3030 chris    chris     1000 2456968 248340 firefox
 3233 chris    chris     1000 2314388 316252 Web Content
```

Refer to the ps man page for information on other columns of information by which you can display and sort.

Listing and changing processes with top

The top command provides a screen-oriented means of displaying processes running on your system. With top, the default is to display processes based on how much CPU time they are currently consuming. However, you can sort by other columns as well. After you identify a misbehaving process, you can also use top to kill (completely end) or renice (reprioritize) that process.

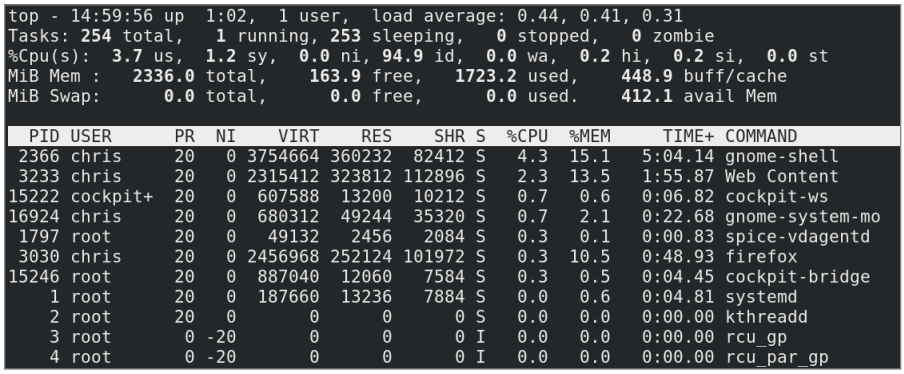
If you want to be able to kill or renice any processes, you need to run top as the root user. If you just want to display processes and possibly kill or change your own processes, you can do that as a regular user. Figure 6.1 shows an example of the top window.

General information about your system appears at the top of the top output, followed by information about each running process (or at least as many as will fit on your screen). At the top, you can see how long the system has been up, how many users are currently logged in to the system, and how much demand there has been on the system for the past 1, 5, and 10 minutes.

Other general information includes how many processes (tasks) are currently running, how much CPU is being used, and how much RAM and swap are available and being used.

FIGURE 6.1

Displaying running processes with `top`



Following the general information are listings of each process, sorted by what percent of the CPU is being used by each process. All of this information is redisplayed every 5 seconds, by default.

The following list includes actions that you can do with `top` to display information in different ways and modify running processes:

- Press **h** to see help options, and then press any key to return to the `top` display.
- Press **M** to sort by memory usage instead of CPU, and then press **P** to return to sorting by CPU.
- Press the number **1** to toggle showing CPU usage of all your CPUs if you have more than one CPU on your system.
- Press **R** to reverse sort your output.
- Press **u** and enter a username to display processes only for a particular user.

A common practice is to use `top` to find processes that are consuming too much memory or processing power and then act on those processes in some way. A process consuming too much CPU can be reniced to give it less priority to the processors. A process consuming too much memory can be killed. With `top` running, here's how to renice or kill a process:

Renicing a process Note the process ID of the process you want to renice and press **r**. When the `PID to renice` message appears, type the process ID of the process you want to renice. When prompted to `Renice PID to value`, type in a number from –20 to 19. (See "Setting processor priority with nice and renice" later in this chapter for information on the meanings of different renice values.)

Killing a process Note the process ID of the process you want to kill and press **k**. Type **15** to terminate cleanly or **9** to just kill the process outright. (See "Killing processes with kill and killall" later in this chapter for more information on using different signals you can send to processes.)

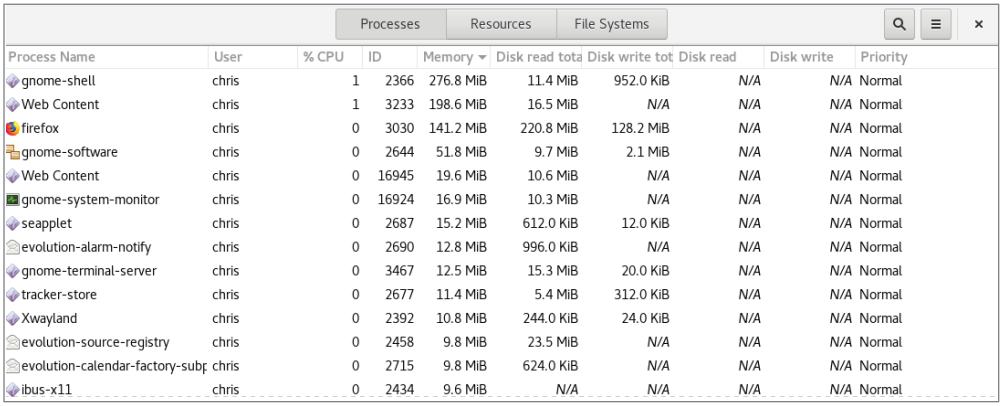
Listing processes with System Monitor

If you have GNOME desktop available on your Linux system, System Monitor (`gnome-system-monitor`) is available to provide a more graphical way of displaying processes on your system. You sort processes by clicking columns. You can right-click processes to stop, kill, or renice them.

To start System Monitor from the GNOME desktop, press the Windows key and then type **System Monitor** and press Enter. Then select the Processes tab. Figure 6.2 shows an example of the System Monitor window, displaying processes for the current user in order by memory use.

FIGURE 6.2

Use the System Monitor window to view and change running processes.



Process Name	User	% CPU	ID	Memory	Disk read tota	Disk write tot	Disk read	Disk write	Priority
gnome-shell	chris	1	2366	276.8 MiB	11.4 MiB	952.0 KiB	N/A	N/A	Normal
Web Content	chris	1	3233	198.6 MiB	16.5 MiB	N/A	N/A	N/A	Normal
firefox	chris	0	3030	141.2 MiB	220.8 MiB	128.2 MiB	N/A	N/A	Normal
gnome-software	chris	0	2644	51.8 MiB	9.7 MiB	2.1 MiB	N/A	N/A	Normal
Web Content	chris	0	16945	19.6 MiB	10.6 MiB	N/A	N/A	N/A	Normal
gnome-system-monitor	chris	0	16924	16.9 MiB	10.3 MiB	N/A	N/A	N/A	Normal
seapplet	chris	0	2687	15.2 MiB	612.0 KiB	12.0 KiB	N/A	N/A	Normal
evolution-alarm-notify	chris	0	2690	12.8 MiB	996.0 KiB	N/A	N/A	N/A	Normal
gnome-terminal-server	chris	0	3467	12.5 MiB	15.3 MiB	20.0 KiB	N/A	N/A	Normal
tracker-store	chris	0	2677	11.4 MiB	5.4 MiB	312.0 KiB	N/A	N/A	Normal
Xwayland	chris	0	2392	10.8 MiB	244.0 KiB	24.0 KiB	N/A	N/A	Normal
evolution-source-registry	chris	0	2458	9.8 MiB	23.5 MiB	N/A	N/A	N/A	Normal
evolution-calendar-factory-subp	chris	0	2715	9.8 MiB	624.0 KiB	N/A	N/A	N/A	Normal
ibus-x11	chris	0	2434	9.6 MiB	N/A	N/A	N/A	N/A	Normal

By default, only running processes associated with your user account are displayed. Those processes are listed alphabetically at first. You can resort the processes by clicking any of the field headings (forward and reverse). For example, click the %CPU heading to see which processes are consuming the most processing power. Click the Memory heading to see which processes consume the most memory.

You can change your processes in various ways by right-clicking a process name and selecting from the menu that appears (see Figure 6.3 for an example).

Here are some of the things you can do to a process from the menu you clicked:

Stop: Pauses the process so that no processing occurs until you select Continue Process. (This is the same as pressing Ctrl+Z on a process from the shell.)

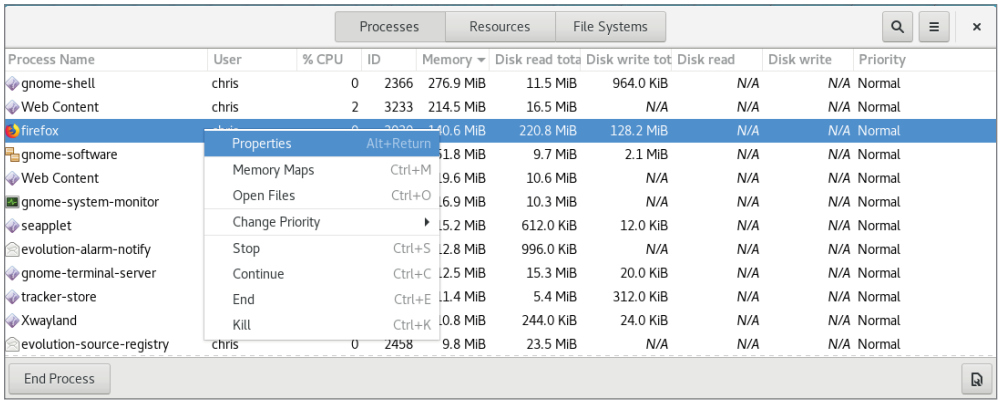
Continue: Continues running a paused process.

End: Sends a Terminate signal (15) to a process. In most cases, this terminates the process cleanly.

Kill: Sends a Kill signal (9) to a process. This should kill a process immediately, regardless of whether it can be done cleanly.

FIGURE 6.3

Renice, kill, or pause a process from the System Monitor window.



Change Priority: Presents a list of priorities from Very Low to Very High. Select Custom to see a slider bar from which you can renice a process. Normal priority is 0. To get better processor priority, use a negative number from –1 to –20. To have a lower processor priority, use a positive number (0 to 19). Only the root user can assign negative priorities, so when prompted you need to provide the root password to set a negative nice value.

Memory Maps: Lets you view the system memory map to see which libraries and other components are being held in memory for the process.

Open Files: Lets you view which files are currently being held open by the process.

Properties: Lets you see other settings associated with the process (such as security context, memory usage, and CPU use percentages).

You can display running processes associated with users other than yourself. To do that, highlight any process in the display (just click it). Then, from the menu button (the button with three bars on it), select All Processes. You can modify processes you don't own only if you are the root user or if you can provide the root password when prompted after you try to change a process. Sometimes, you won't have the luxury of working with a graphical interface. To change processes without a graphical interface, you can use a set of commands and keystrokes to change, pause, or kill running processes. Some of those are described next.

Managing Background and Foreground Processes

If you are using Linux over a network or from a *dumb terminal* (a monitor that allows only text input with no GUI support), your shell may be all that you have. You may be used to a

graphical environment in which you have lots of programs active at the same time so that you can switch among them as needed. This shell thing can seem pretty limited.

Although the bash shell doesn't include a GUI for running many programs at once, it does let you move active programs between the background and foreground. In this way, you can have lots of stuff running and selectively choose the one you want to deal with at the moment.

You can place an active program in the background in several ways. One is to add an ampersand (&) to the end of a command line when you first run the command. You can also use the `at` command to run commands in such a way that they are not connected to the shell.

To stop a running command and put it in the background, press Ctrl+Z. After the command is stopped, you can either bring it back into the foreground to run (the `fg` command) or start it running in the background (the `bg` command). Keep in mind that any command running in the background might spew output during commands that you run subsequently from that shell. For example, if output appears from a command running in the background during a `vi` session, simply press Ctrl+L to redraw the screen to get rid of the output.

Tip

To avoid having the output appear, you should have any process running in the background send its output to a file or to null (add `2> /dev/null` to the end of the command line).

Starting background processes

If you have programs that you want to run while you continue to work in the shell, you can place the programs in the background. To place a program in the background at the time you run the program, type an ampersand (&) at the end of the command line, like this:

```
$ find /usr > /tmp/allusrfiles &
[3] 15971
```

This example command finds all files on your Linux system (starting from `/usr`), prints those filenames, and puts those names in the file `/tmp/allusrfiles`. The ampersand (&) runs that command line in the background. Notice that the job number, `[3]`, and process ID number, `15971`, are displayed when the command is launched. To check which commands you have running in the background, use the `jobs` command, as follows:

```
$ jobs
[1]  Stopped (tty output)  vi /tmp/myfile
[2]  Running        find /usr -print > /tmp/allusrfiles &
[3]  Running        nroff -man /usr/man2/* >/tmp/man2 &
[4]- Running        nroff -man /usr/man3/* >/tmp/man3 &
[5]+ Stopped        nroff -man /usr/man4/* >/tmp/man4
```

The first job shows a text-editing command (`vi`) that I placed in the background and stopped by pressing Ctrl+Z while I was editing. Job 2 shows the `find` command I just ran.

Jobs 3 and 4 show `nroff` commands currently running in the background. Job 5 had been running in the shell (foreground) until I decided too many processes were running and pressed Ctrl+Z to stop job 5 until a few processes had completed.

The plus sign (+) next to number 5 shows that it was most recently placed in the background. The minus sign (-) next to number 4 shows that it was placed in the background just before the most recent background job. Because job 1 requires terminal input, it cannot run in the background. As a result, it is `Stopped` until it is brought to the foreground again.

Tip

To see the process ID for the background job, add a `-l` (the lowercase letter *L*) option to the jobs command. If you type `ps`, you can use the process ID to figure out which command is for a particular background job.

Using foreground and background commands

Continuing with the example, you can bring any of the commands on the jobs list to the foreground. For example, to edit `myfile` again, enter the following:

```
$ fg %1
```

As a result, the `vi` command opens again. All text is as it was when you stopped the `vi` job.

Caution

Before you put a text processor, word processor, or similar program in the background, make sure that you save your file. It's easy to forget that you have a program in the background, and you will lose your data if you log out or the computer reboots.

To refer to a background job (to cancel or bring it to the foreground), use a percent sign (`%`) followed by the job number. You can also use the following to refer to a background job:

`%`	Refers to the most recent command put into the background (indicated by the plus sign when you type the `jobs` command). This action brings the command to the foreground.
`%string`	Refers to a job where the command begins with a particular string of characters. The string must be unambiguous. (In other words, typing `%vi` when there are two `vi` commands in the background results in an error message.)
`%?string`	Refers to a job where the command line contains a string at any point. The string must be unambiguous or the match fails.
`%--`	Refers to the job stopped before the one most recently stopped.

If a command is stopped, you can start it running again in the background using the `bg` command. For example, refer back to job 5 from the jobs list in a previous example:

```
[5]+ Stopped nroff -man /usr/man4/* >/tmp/man4
```

Enter the following:

```
$ bg %5
```

After that, the job runs in the background. Its `jobs` entry appears as follows:

```
[5] Running nroff -man /usr/man4/* >/tmp/man4 &
```

Killing and Renicing Processes

Just as you can change the behavior of a process using graphical tools such as System Monitor (described earlier in this chapter), you can also use command-line tools to kill a process or change its CPU priority. The `kill` command can send a kill signal to any process to end it, assuming you have permission to do so. It can also send different signals to a process to otherwise change its behavior. The `nice` and `renice` commands can be used to set or change the processor priority of a process.

Killing processes with kill and killall

Although usually used for ending a running process, the `kill` and `killall` commands can actually be used to send any valid signal to a running process. Besides telling a process to end, a signal might tell a process to reread configuration files, pause (stop), or continue after being paused, just to name a few possibilities.

Signals are represented by both numbers and names. Signals that you might send most commonly from a command include `SIGKILL` (`9`), `SIGTERM` (`15`), and `SIGHUP` (`1`). The default signal is `SIGTERM`, which tries to terminate a process cleanly. To kill a process immediately, you can use `SIGKILL`. The `SIGHUP` signal can, depending on the program, tell a process to reread its configuration files. `SIGSTOP` pauses a process, while `SIGCONT` continues a stopped process.

Different processes respond to different signals. Processes cannot block `SIGKILL` and `SIGSTOP` signals, however. Table 6.1 shows examples of some signals (enter **`man 7 signal`** to read about other available signals).

Notice that there are multiple possible signal numbers for `SIGCONT` and `SIGSTOP` because different numbers are used in different computer architectures. For most x86 and Power architectures, use the middle value. The first value usually works for Alpha and SPARC, while the last one is for MIPS architecture.

Using kill to signal processes by **`PID`**

Using commands such as `ps` and `top`, you can find processes to which you want to send a signal. Then you can use the process ID of that process as an option to the `kill` command, along with the signal you want to send.

TABLE 6.1 **Signals Available in Linux**

Signal	Number	Description
SIGHUP	1	Hang-up detected on controlling terminal or death of controlling process.
SIGINT	2	Interrupt from keyboard.
SIGQUIT	3	Quit from keyboard.
SIGABRT	6	Abort signal from abort(3).
SIGKILL	9	Kill signal.
SIGTERM	15	Termination signal.
SIGCONT	19,18,25	Continue if stopped.
SIGSTOP	17,19,23	Stop process.

For example, you run the `top` command and see that the `bigcommand` process is consuming most of your processing power:

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10432 chris     20   0  471m 121m  18m S 99.9  3.2  77:01.76  bigcommand
```

Here, the `bigcommand` process is consuming 99.9 percent of the CPU. You decide that you want to kill it so that other processes have a shot at the CPU. If you use the process ID of the running `bigcommand` process, here are some examples of the `kill` command that you can use to kill that process:

```
$ kill 10432
$ kill -15 10432
$ kill -SIGKILL 10432
```

The default signal sent by kill is `15` (`SIGTERM`), so the first two examples have exactly the same results. On occasion, a `SIGTERM` doesn't kill a process, so you may need a `SIGKILL` to kill it. Instead of `SIGKILL`, you can use `-9` to get the same result.

Another useful signal is `SIGHUP`. If, for example, something on your GNOME desktop were corrupted, you could send the `gnome-shell` a `SIGHUP` signal to reread its configuration files and restart the desktop. If the process ID for `gnome-shell` were 1833, here are two ways you could send it a `SIGHUP` signal:

```
# kill -1 1833
# killall -HUP gnome-shell
```

Using killall to signal processes by name

With the `killall` command, you can signal processes by name instead of by process ID. The advantage is that you don't have to look up the process ID of the process that you want to kill. The potential downside is that you can kill more processes than you mean to if you are not careful. (For example, typing `killall bash` may kill a bunch of shells that you don't mean to kill.)

Like the `kill` command, `killall` uses `SIGTERM` (signal 15) if you don't explicitly enter a signal number. Also as with `kill`, you can send any signal you like to the process you name with `killall`. For example, if you see a process called `testme` running on your system and you want to kill it, you can simply enter the following:

```
$ killall -9 testme
```

The `killall` command can be particularly useful if you want to kill a bunch of commands of the same name.

Setting processor priority with nice and renice

When the Linux kernel tries to decide which running processes get access to the CPUs on your system, one of the things it takes into account is the nice value set on the process. Every process running on your system has a nice value between –20 and 19. By default, the nice value is set to 0. Here are a few facts about nice values:

- The lower the nice value, the more access to the CPUs the process has. In other words, the nicer a process is, the less CPU attention it gets. So, a –20 nice value gets more attention than a process with a 19 nice value.
- A regular user can set nice values only from 0 to 19. No negative values are allowed. So a regular user can't ask for a value that gives a process more attention than most processes get by default.
- A regular user can set the nice value higher, not lower. So, for example, if a user sets the nice value on a process to 10 and then later wants to set it back to 5, that action will fail. Likewise, any attempt to set a negative value will fail.
- A regular user can set the nice value only on the user's own processes.
- The root user can set the nice value on any process to any valid value, up or down.

You can use the `nice` command to run a command with a particular nice value. When a process is running, you can change the nice value using the `renice` command, along with the process ID of the process, as in the example that follows:

```
# nice -n +5 updatedb &
```

The `updatedb` command is used to generate the locate database manually by gathering names of files throughout the filesystem. In this case, I just wanted `updatedb` to run in the background (`&`) and not interrupt work being done by other processes on the system. I ran the `top` command to make sure that the nice value was set properly:

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20284 root    25   5 98.7m  932  644 D  2.7  0.0   0:00.96 updatedb
```

Notice that under the `NI` column, the nice value is set to 5. Because the command was run as the root user, the root user can lower the nice value later by using the `renice` command. (Remember that a regular user can't reduce the nice value or ever set it to a negative number.) Here's how you would change the nice value for the `updatedb` command just run to `–5`:

```
# renice -n -5 20284
```

If you ran the `top` command again, you might notice that the `updatedb` command is now at or near the top of the list of processes consuming CPU time because you gave it priority to get more CPU attention.

Limiting Processes with cgroups

You can use a feature like "nice" to give a single process more or less access to CPU time. Setting the nice value for one process, however, doesn't apply to child processes that a process might start up or any other related processes that are part of a larger service. In other words, "nice" doesn't limit the total amount of resources a particular user or application can consume from a Linux system.

As cloud computing takes hold, many Linux systems will be used more as hypervisors than as general-purpose computers. Their memory, processing power, and access to storage will become commodities to be shared by many users. In that model, more needs to be done to control the amount of system resources to which a particular user, application, container, or virtual machine running on a Linux system has access.

That's where *cgroups* come in.

Cgroups can be used to identify a process as a task, belonging to a particular control group. Tasks can be set up in a hierarchy where, for example, there may be a task called daemons that sets default limitations for all daemon server processes, then subtasks that may set specific limits on a web server daemon (`httpd`) for FTP service daemon (`vsftpd`).

As a task launches a process, other processes that the initial process launches (called child processes) inherit the limitations set for the parent process. Those limitations might say that all the processes in a control group only have access to particular processors and certain sets of RAM. Or they may only allow access to up to 30 percent of the total processing power of a machine.

The types of resources that can be limited by cgroups include the following:

Storage (`blkio`): Limits total input and output access to storage devices (such as hard disks, USB drives, and so on).

Processor scheduling (`cpu`): Assigns the amount of access a cgroup has to be scheduled for processing power.

Process accounting (`cpuacct`): Reports on CPU usage. This information can be leveraged to charge clients for the amount of processing power they use.

CPU assignment (`cpuset`): On systems with multiple CPU cores, assigns a task to a particular set of processors and associated memory.

Device access (`devices`): Allows tasks in a cgroup to open or create (`mknod`) selected device types.

Suspend/resume (`freezer`): Suspends and resumes cgroup tasks.

Memory usage (`memory`): Limits memory usage by task. It also creates reports on memory resources used.

Network bandwidth (`net_cls`): Limits network access to selected cgroup tasks. This is done by tagging network packets to identify the cgroup task that originated the packet and having the Linux traffic controller monitor and restrict packets coming from each cgroup.

Network traffic (`net_prio`): Sets priorities of network traffic coming from selected cgroups and lets administrators change these priorities on the fly.

Name spaces (`ns`): Separates cgroups into namespaces, so processes in one cgroup can only see the namespaces associated with the cgroup. Namespaces can include separate process tables, mount tables, and network interfaces.

At its most basic level, creating and managing cgroups is generally not a job for new Linux system administrators. It can involve editing configuration files to create your own cgroups (`/etc/cgconfig.conf`) or set up limits for particular users or groups (`/etc/cgrules.conf`). Or you can use the `cgcreate` command to create cgroups, which results in those groups being added to the `/sys/fs/cgroup` hierarchy. Setting up cgroups can be tricky and, if done improperly, can make your system unbootable.

The reason I bring up the concept of cgroups here is to help you understand some of the underlying features in Linux that can be used to limit and monitor resource usage. In the future, you will probably run into these features from controllers that manage your cloud infrastructure. You will be able to set rules like "Allow the Marketing department's virtual machines to consume up to 40 percent of the available memory" or "Pin the database application to a particular CPU and memory set."

Knowing how Linux can limit and contain the resource usage by the set of processes assigned to a task will ultimately help you manage your computing resources better. If you are interested in learning more about cgroups, you can refer to the following:

- **Red Hat Enterprise Linux Resource Management and Linux Containers Guide**:

  `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/resource_management_guide/index`

- **Kernel documentation on cgroups**: Refer to files in the `/usr/share/doc/kernel-doc-*/Documentation/cgroups` directory after installing the kernel-doc package.

Summary

Even on a Linux system where there isn't much activity, typically dozens or even hundreds of processes are running in the background. Using the tools described in this chapter, you can view and manage the processes running on your system.

Managing processes includes viewing processes in different ways, running them in the foreground or background, and killing or renicing them. More advanced features for limiting resource usage by selected processes are available using the cgroups feature.

In the next chapter, you learn how to combine commands and programming functions into files that can be run as shell scripts.

Exercises

Use these exercises to test your knowledge of viewing running processes and then changing them later by killing them or changing processor priority (nice value). These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, you can often use multiple ways to complete a task).

1. List all processes running on your system, showing a full set of columns. Pipe that output to the `less` command so that you can page through the list of processes.
2. List all processes running on the system and sort those processes by the name of the user running each process.
3. List all processes running on the system, and display the following columns of information: process ID, username, group name, virtual memory size, resident memory size, and the command.
4. Run the `top` command to view processes running on your system. Go back and forth between sorting by CPU usage and memory consumption.
5. Start the `gedit` process from your desktop. Make sure that you run it as the user you are logged in as. Use the System Monitor window to kill that process.
6. Run the `gedit` process again. This time, using the kill command, send a signal to the `gedit` process that causes it to pause (stop). Try typing some text into the `gedit` window and make sure that no text appears yet.
7. Use the `killall` command to tell the `gedit` command that you paused in the previous exercise to continue working. Make sure that the text you type in after `gedit` was paused now appears on the window.
8. Install the `xeyes` command (in Fedora, it is in the `xorg-x11-apps` package). Run the `xeyes` command about 20 times in the background so that 20 `xeyes` windows appear on the screen. Move the mouse around and watch the eyes watch your mouse pointer. When you have had enough fun, kill all `xeyes` processes in one command using `killall`.
9. As a regular user, run the `gedit` command so that it starts with a nice value of 5.
10. Using the `renice` command, change the nice value of the `gedit` command you just started to `7`. Use any command you like to verify that the current nice value for the `gedit` command is now set to `7`.

CHAPTER 7

Writing Simple Shell Scripts

IN THIS CHAPTER

Working with shell scripts

Doing arithmetic in shell scripts

Running loops and cases in shell scripts

Creating simple shell scripts

You'd never get any work done if you typed every command that needs to be run on your Linux system when it starts. Likewise, you could work more efficiently if you grouped together sets of commands that you run all the time. Shell scripts can handle these tasks.

A *shell script* is a group of commands, functions, variables, or just about anything else you can use from a shell. These items are typed into a plain-text file. That file can then be run as a command. Linux systems have traditionally used system initialization shell scripts during system startup to run commands needed to get services going. You can create your own shell scripts to automate the tasks that you need to do regularly.

For decades, building shell scripts was the primary skill needed to join together sets of tasks in UNIX and Linux systems. As demands for configuring Linux systems grew beyond single-system setups to complex, automated cluster configurations, more structured methods have arisen. These methods include Ansible playbooks and Kubernetes YAML files, described later in cloud-related chapters. That said, writing shell scripts is still the best next step from running individual commands to building repeatable tasks in Linux systems.

This chapter provides a rudimentary overview of the inner workings of shell scripts and how they can be used. You learn how simple scripts can be harnessed to a scheduling facility (such as `cron` or `at`) to simplify administrative tasks or just run on demand as they are needed.

Understanding Shell Scripts

Have you ever had a task that you needed to do over and over that took lots of typing on the command line? Do you ever think to yourself, "Wow, I wish I could just type one command to do all this"? Maybe a shell script is what you're after.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Shell scripts are the equivalent of batch files in Windows and can contain long lists of commands, complex flow control, arithmetic evaluations, user-defined variables, user-defined functions, and sophisticated condition testing. Shell scripts are capable of handling everything from simple one-line commands to something as complex as starting up a Linux system. Although dozens of different shells are available in Linux, the default shell for most Linux systems is called bash, the **B**ourne **A**gain **SH**ell.

Executing and debugging shell scripts

One of the primary advantages of shell scripts is that they can be opened in any text editor to see what they do. A big disadvantage is that large or complex shell scripts often execute more slowly than compiled programs. You can execute a shell script in two basic ways:

- The filename is used as an argument to the shell (as in `bash myscript`). In this method, the file does not need to be executable; it just contains a list of shell commands. The shell specified on the command line is used to interpret the commands in the script file. This is most common for quick, simple tasks.
- The shell script may also have the name of the interpreter placed in the first line of the script preceded by `#!` (as in `#!/bin/bash`) and have the execute bit of the file containing the script set (using `chmod +x filename`). You can then run your script just as you would any other program in your path simply by typing the name of the script on the command line.

When scripts are executed in either manner, options for the program may be specified on the command line. Anything following the name of the script is referred to as a *command-line argument*.

As with writing any software, there is no substitute for clear and thoughtful design and lots of comments. The pound sign (#) prefaces comments and can take up an entire line or exist on the same line after script code. It is best to implement more complex shell scripts in stages, making sure that the logic is sound at each step before continuing. Here are a few good, concise tips to make sure that things are working as expected during testing:

- In some cases, you can place an `echo` statement at the beginning of lines within the body of a loop and surround the command with quotes. That way, rather than executing the code, you can see what will be executed without making any permanent changes.
- To achieve the same goal, you can place dummy `echo` statements throughout the code. If these lines get printed, you know the correct logic branch is being taken.
- You can use `set -x` near the beginning of the script to display each command that is executed or launch your scripts using

```
$ bash -x myscript
```

- Because useful scripts have a tendency to grow over time, keeping your code readable as you go along is extremely important. Do what you can to keep the logic of your code clean and easy to follow.

Understanding shell variables

Often within a shell script, you want to reuse certain items of information. During the course of processing the shell script, the name or number representing this information may change. To store information used by a shell script in such a way that it can be easily reused, you can set *variables*. Variable names within shell scripts are case sensitive and can be defined in the following manner:

```
NAME=value
```

The first part of a variable is the variable name, and the second part is the value set for that name. Be sure that the `NAME` and *`value`* touch the equal sign, without any spaces. Variables can be assigned from constants, such as text, numbers, and underscores. This is useful for initializing values or saving lots of typing for long constants. The following examples show variables set to a string of characters (`CITY`) and a numeric value (`PI`):

```
CITY="Springfield"
PI=3.14159265
```

Variables can contain the output of a command or command sequence. You can accomplish this by preceding the command with a dollar sign and open parenthesis, following it with a closing parenthesis. For example, `MYDATE=$(date)` assigns the output from the `date` command to the `MYDATE` variable. Enclosing the command in back-ticks (`` ` ``) can have the same effect. In this case, the `date` command is run when the variable is set and not each time the variable is read.

Escaping Special Shell Characters

Keep in mind that characters such as the dollar sign ($), back-tick (`), asterisk (*), exclamation point (!), and others have special meaning to the shell, which you will see as you proceed through this chapter. On some occasions, you want the shell to use these characters' special meaning and other times you don't. For example, if you typed `echo $HOME`, the shell would think that you meant to display the name of your home directory (stored in the `$HOME` variable) to the screen (such as `/home/chris`) because a `$` indicates a variable name follows that character.

If you wanted literally to show `$HOME`, you would need to escape the `$`. Typing `echo '$HOME' or echo \$HOME` would literally show `$HOME` on the screen. So, if you want to have the shell interpret a single character literally, precede it with a backslash (\). To have a whole set of characters interpreted literally, surround those characters with single quotes (').

Using double quotes is a bit trickier. Surround a set of text with double quotes if you want all but a few characters used literally. For example, with text surrounded with double quotes, dollar signs ($), back-ticks (`), and exclamation points (!) are interpreted specially, but other characters (such as an asterisk) are not. Type these three lines to see the different output (shown on the right):

```
echo '$HOME * `date`'    $HOME * `date`
echo "$HOME * `date`"    /home/chris * Tue Jan 21 16:56:52 EDT 2020
echo $HOME * `date`      /home/chris file1 file2 Tue Jan 21 16:56:52 EDT 2020
```

Using variables is a great way to get information that can change from computer to computer or from day to day. The following example sets the output of the `uname -n` command to the `MACHINE` variable. Then I use parentheses to set `NUM_FILES` to the number of files in the current directory by piping (|) the output of the `ls` command to the word count command (`wc -l`):

```
MACHINE=`uname -n`
NUM_FILES=$(/bin/ls | wc -l)
```

Variables can also contain the value of other variables. This is useful when you have to preserve a value that will change so that you can use it later in the script. Here, `BALANCE` is set to the value of the `CurBalance` variable:

```
BALANCE="$CurBalance"
```

NOTE

When assigning variables, use only the variable name (for example, `BALANCE`). When you reference a variable, meaning that you want the *value* of the variable, precede it with a dollar sign (as in `$CurBalance`). The result of the latter is that you get the value of the variable, not the variable name itself.

Special shell positional parameters

There are special variables that the shell assigns for you. One set of commonly used variables is called *positional parameters* or *command-line arguments*, and it is referenced as $0, $1, $2, $3. . .$*n*. $0 is special, and it is assigned the name used to invoke your script; the others are assigned the values of the parameters passed on the command line in the order they appeared. For instance, let's say that you had a shell script named `myscript` which contained the following:

```
#!/bin/bash
# Script to echo out command-line arguments
echo "The first argument is $1, the second is $2."
echo "The command itself is called $0."
echo "There are $# parameters on your command line"
echo "Here are all the arguments: $@"
```

Assuming that the script is executable and located in a directory in your `$PATH`, the following shows what would happen if you ran that command with `foo` and `bar` as arguments:

```
$ chmod 755 /home/chris/bin/myscript
$ myscript foo bar
The first argument is foo, the second is bar.
The command itself is called /home/chris/bin/myscript.
There are 2 parameters on your command line
Here are all the arguments: foo bar
```

As you can see, the positional parameter `$0` is the full path or relative path to `myscript`, `$1` is `foo`, and `$2` is `bar`.

Another variable, `$#`, tells you how many parameters your script was given. In the example, `$#` would be 2. The `$@` variable holds all of the arguments entered at the command line. Another particularly useful special shell variable is `$?`, which receives the exit status of the last command executed. Typically, a value of zero means that the command exited successfully, and anything other than zero indicates an error of some kind. For a complete list of special shell variables, refer to the `bash` man page.

Reading in parameters

Using the `read` command, you can prompt the user for information and store that information to use later in your script. Here's an example of a script that uses the `read` command:

```
#!/bin/bash
read -p "Type in an adjective, noun and verb (past tense): " adj1 noun1 verb1
echo "He sighed and $verb1 to the elixir. Then he ate the $adj1 $noun1."
```

In this script, after the script prompts for an adjective, noun, and verb, the user is expected to enter words that are then assigned to the `adj1`, `noun1`, and `verb1` variables. Those three variables are then included in a silly sentence, which is displayed on the screen. If the script were called `sillyscript`, here's an example of how it might run:

```
$ chmod 755 /home/chris/bin/sillyscript
$ sillyscript
Type in an adjective, noun and verb (past tense): hairy football danced
He sighed and danced to the elixir. Then he ate the hairy football.
```

Parameter expansion in bash

As mentioned earlier, if you want the value of a variable, you precede it with a `$` (for example, `$CITY`). This is really just shorthand for the notation `${CITY}`; curly braces are used when the value of the parameter needs to be placed next to other text without a space. Bash has special rules that allow you to expand the value of a variable in different ways. Going into all of the rules is probably overkill for a quick introduction to shell scripts, but the following list presents some common constructs you're likely to see in bash scripts that you find on your Linux system.

`${var:-value}`: If variable is unset or empty, expand this to *value*.

`${var#pattern}`: Chop the shortest match for *pattern* from the front of *var*'s value.

`${var##pattern}`: Chop the longest match for *pattern* from the front of *var*'s value.

`${var%pattern}`: Chop the shortest match for *pattern* from the end of *var*'s value.

`${var%%pattern}`: Chop the longest match for *pattern* from the end of *var*'s value.

Try typing the following commands from a shell to test how parameter expansion works:

```
$ THIS="Example"
$ THIS=${THIS:-"Not Set"}
$ THAT=${THAT:-"Not Set"}
$ echo $THIS
Example
$ echo $THAT
Not Set
```

In the examples here, the `THIS` variable is initially set to the word `Example`. In the next two lines, the `THIS` and `THAT` variables are set to their current values or to `Not Set`, if they are not currently set. Notice that because I just set `THIS` to the string `Example`, when I echo the value of `THIS` it appears as `Example`. However, because `THAT` was not set, it appears as `Not Set`.

Note

For the rest of this section, I show how variables and commands may appear in a shell script. To try out any of those examples, however, you can simply type them into a shell, as shown in the previous example.

In the following example, `MYFILENAME` is set to `/home/digby/myfile.txt`. Next, the `FILE` variable is set to `myfile.txt` and `DIR` is set to `/home/digby`. In the `NAME` variable, the filename is cut down simply to `myfile`; then, in the `EXTENSION` variable, the file extension is set to `txt`. (To try these out, you can type them at a shell prompt as in the previous example and echo the value of each variable to see how it is set.) Type the code on the left. The material on the right side describes the action.

`MYFILENAME=/home/digby/myfile.txt`: Sets the value of `MYFILENAME`

`FILE=${MYFILENAME##*/}`: `FILE` becomes `myfile.txt`

`DIR=${MYFILENAME%/*}`: `DIR` becomes `/home/digby`

`NAME=${FILE%.*}`: `NAME` becomes `myfile`

`EXTENSION=${FILE##*.}`: `EXTENSION` becomes `txt`

Performing arithmetic in shell scripts

Bash uses *untyped variables*, meaning it normally treats variables as strings of text, but you can change them on the fly if you want it to.

Bash uses *untyped variables*, meaning that you are not required to specify whether a variable is text or numbers. It normally treats variables as strings of text, so unless you tell it otherwise with `declare`, your variables are just a bunch of letters to bash. However, when you start trying to do arithmetic with them, bash converts them to integers if it can. This makes it possible to do some fairly complex arithmetic in bash.

Integer arithmetic can be performed using the built-in `let` command or through the external `expr` or `bc` commands. After setting the variable `BIGNUM` value to `1024`, the

three commands that follow would all store the value 64 in the RESULT variable. The bc command is a calculator application that is available in most Linux distributions. The last command gets a random number between 0 and 10 and echoes the results back to you.

```
BIGNUM=1024
let RESULT=$BIGNUM/16
RESULT=`expr $BIGNUM / 16`
RESULT=`echo "$BIGNUM / 16" | bc`
let foo=$RANDOM; echo $foo
```

Another way to grow a variable incrementally is to use $(()) notation with ++I added to increment the value of I. Try typing the following:

```
$ I=0
$ echo "The value of I after increment is $((++I))"
The value of I after increment is 1

$ echo "The value of I before and after increment is $((I++)) and $I"
The value of I before and after increment is 1 and 2
```

Repeat either of those commands to continue to increment the value of $I.

Note

Although most elements of shell scripts are relatively freeform (where white space, such as spaces or tabs, is insignificant), both let and expr are particular about spacing. The let command insists on no spaces between each operand and the mathematical operator, whereas the syntax of the expr command requires white space between each operand and its operator. In contrast to those, bc isn't picky about spaces, but it can be trickier to use because it does floating-point arithmetic.

To see a complete list of the kinds of arithmetic that you can perform using the let command, type **help let** at the bash prompt.

Using programming constructs in shell scripts

One of the features that makes shell scripts so powerful is that their implementation of looping and conditional execution constructs is similar to those found in more complex scripting and programming languages. You can use several different types of loops, depending on your needs.

The "if. . .then" statements

The most commonly used programming construct is conditional execution, or the if statement. It is used to perform actions only under certain conditions. There are several variations of if statements for testing various types of conditions.

The first if...then example tests if VARIABLE is set to the number 1. If it is, then the echo command is used to say that it is set to 1. The fi statement then indicates that the if statement is complete, and processing can continue.

```
VARIABLE=1
if [ $VARIABLE -eq 1 ] ; then
echo "The variable is 1"
fi
```

Instead of using `-eq`, you can use the equal sign (=), as shown in the following example. The = works best for comparing string values, while `-eq` is often better for comparing numbers. Using the `else` statement, different words can be echoed if the criterion of the `if` statement isn't met (`$STRING = "Friday"`). Keep in mind that it's good practice to put strings in double quotes.

```
STRING="Friday"
if [ $STRING = "Friday" ] ; then
echo "WhooHoo.  Friday."
else
echo "Will Friday ever get here?"
fi
```

You can also reverse tests with an exclamation mark (!). In the following example, if `STRING` is not `Monday`, then "`At least it's not Monday`" is echoed.

```
STRING="FRIDAY"
if [ "$STRING" != "Monday" ] ; then
   echo "At least it's not Monday"
fi
```

In the following example, `elif` (which stands for "else if") is used to test for an additional condition (for example, whether `filename` is a file or a directory).

```
filename="$HOME"
if [ -f "$filename" ] ; then
   echo "$filename is a regular file"
elif [ -d "$filename" ] ; then
   echo "$filename is a directory"
else
   echo "I have no idea what $filename is"
fi
```

As you can see from the preceding examples, the condition you are testing is placed between square brackets `[ ]`. When a test expression is evaluated, it returns either a value of 0, meaning that it is true, or a 1, meaning that it is false. Notice that the `echo` lines are indented. The indentation is optional and done only to make the script more readable.

Table 7.1 lists the conditions that are testable and is quite a handy reference. (If you're in a hurry, you can type **`help test`** on the command line to get the same information.)

TABLE 7.1 Operators for Test Expressions

Operator	What Is Being Tested?
`-a` *file*	Does the file exist? (same as `-e`)
`-b` *file*	Is the file a block special device?
`-c` *file*	Is the file character special (for example, a character device)? Used to identify serial lines and terminal devices.
`-d` *file*	Is the file a directory?
`-e` *file*	Does the file exist? (same as `-a`)
`-f` *file*	Does the file exist, and is it a regular file (for example, not a directory, socket, pipe, link, or device file)?
`-g` *file*	Does the file have the set group id (SGID) bit set?
`-h` *file*	Is the file a symbolic link? (same as `-L`)
`-k` *file*	Does the file have the sticky bit set?
`-L` *file*	Is the file a symbolic link?
`-n` *string*	Is the length of the string greater than 0 bytes?
`-O` *file*	Do you own the file?
`-p` *file*	Is the file a named pipe?
`-r` *file*	Is the file readable by you?
`-s` *file*	Does the file exist, and is it larger than 0 bytes?
`-S` *file*	Does the file exist, and is it a socket?
`-t` *fd*	Is the file descriptor connected to a terminal?
`-u` *file*	Does the file have the set user id (SUID) bit set?
`-w` *file*	Is the file writable by you?
`-x` *file*	Is the file executable by you?
`-z` *string*	Is the length of the string 0 (zero) bytes?
expr1 `-a` *expr2*	Are both the first expression and the second expression true?
expr1 `-o` *expr2*	Is either of the two expressions true?
file1 `-nt` *file2*	Is the first file newer than the second file (using the modification time stamp)?
file1 `-ot` *file2*	Is the first file older than the second file (using the modification time stamp)?
file1 `-ef` *file2*	Are the two files associated by a link (a hard link or a symbolic link)?
var1 `=` *var2*	Is the first variable equal to the second variable?
var1 `-eq` *var2*	Is the first variable equal to the second variable?
var1 `-ge` *var2*	Is the first variable greater than or equal to the second variable?

Continues

TABLE 7.1 *(continued)*

Operator	What Is Being Tested?
var1 -gt *var2*	Is the first variable greater than the second variable?
var1 -le *var2*	Is the first variable less than or equal to the second variable?
var1 -lt *var2*	Is the first variable less than the second variable?
var1 != *var2*	Is the first variable not equal to the second variable?
var1 -ne *var2*	Is the first variable not equal to the second variable?

There is also a special shorthand method of performing tests that can be useful for simple *one-command actions*. In the following example, the two pipes (||) indicate that if the directory being tested for doesn't exist (`-d dirname`), then make the directory (`mkdir $dirname`):

```
# [ test ] || action
# Perform simple single command if test is false
dirname="/tmp/testdir"
[ -d "$dirname" ] || mkdir "$dirname"
```

Instead of pipes, you can use two ampersands to test if something is true. In the following example, a command is being tested to see if it includes at least three command-line arguments:

```
# [ test ] && {action}
# Perform simple single action if test is true
[ $# -ge 3 ] && echo "There are at least 3 command line arguments."
```

You can combine the `&&` and `||` operators to make a quick, one-line `if...then...else`. The following example tests that the directory represented by `$dirname` already exists. If it does, a message says the directory already exists. If it doesn't, the statement creates the directory:

```
# dirname=mydirectory
[ -e $dirname ] && echo $dirname already exists || mkdir $dirname
```

The case command

Another frequently used construct is the `case` command. Similar to a `switch` statement in programming languages, this can take the place of several nested `if` statements. The following is the general form of the `case` statement:

```
case "VAR" in
   Result1)
      { body };;
   Result2)
      { body };;
```

```
   *)
      { body } ;;
esac
```

Among other things, you can use the `case` command to help with your backups. The following `case` statement tests for the first three letters of the current day (`case 'date +%a' in`). Then, depending on the day, a particular backup directory (`BACKUP`) and tape drive (`TAPE`) are set.

```
# Our VAR doesn't have to be a variable,
# it can be the output of a command as well
# Perform action based on day of week
case `date +%a` in
   "Mon")
         BACKUP=/home/myproject/data0
         TAPE=/dev/rft0
# Note the use of the double semi-colon to end each option
         ;;
# Note the use of the "|" to mean "or"
   "Tue" | "Thu")
         BACKUP=/home/myproject/data1
         TAPE=/dev/rft1
         ;;
   "Wed" | "Fri")
         BACKUP=/home/myproject/data2
         TAPE=/dev/rft2
         ;;
# Don't do backups on the weekend.
   *)

BACKUP="none"
         TAPE=/dev/null
         ;;
esac
```

The asterisk (*) is used as a catchall, similar to the `default` keyword in the C programming language. In this example, if none of the other entries are matched on the way down the loop, the asterisk is matched and the value of `BACKUP` becomes `none`. Note the use of `esac`, or `case` spelled backwards, to end the `case` statement.

The "for. . .do" loop

Loops are used to perform actions over and over again until a condition is met or until all data has been processed. One of the most commonly used loops is the `for...do` loop. It iterates through a list of values, executing the body of the loop for each element in the list. The syntax and a few examples are presented here:

```
for VAR in LIST
do
    { body }
done
```

The `for` loop assigns the values in *`LIST`* to *`VAR`* one at a time. Then, for each value, the *`body`* in braces between `do` and `done` is executed. `VAR` can be any variable name, and `LIST` can be composed of pretty much any list of values or anything that generates a list.

```
for NUMBER in 0 1 2 3 4 5 6 7 8 9
do
   echo The number is $NUMBER
done

for FILE in `/bin/ls`
do
   echo $FILE
done
```

You can also write it this way, which is somewhat cleaner:

```
for NAME in John Paul Ringo George ; do
   echo $NAME is my favorite Beatle
done
```

Each element in the `LIST` is separated from the next by white space. This can cause trouble if you're not careful because some commands, such as `ls -l`, output multiple fields per line, each separated by white space. The string `done` ends the `for` statement.

If you're a die-hard C programmer, bash allows you to use C syntax to control your loops:

```
LIMIT=10
# Double parentheses, and no $ on LIMIT even though it's a variable!
for ((a=1; a <= LIMIT ; a++)) ; do
  echo  "$a"
done
```

The "while. . .do" and "until. . .do" loops

Two other possible looping constructs are the `while...do` loop and the `until...do` loop. The structure of each is presented here:

```
while condition      until condition
do                   do
   { body }            { body }
done                 done
```

The `while` statement executes while the condition is true. The `until` statement executes until the condition is true—in other words, while the condition is false.

Here is an example of a `while` loop that outputs the number 0123456789:

```
N=0
while [ $N -lt 10 ] ; do
   echo -n $N
   let N=$N+1
done
```

Another way to output the number 0123456789 is to use an `until` loop as follows:

```
N=0
until [ $N -eq 10 ] ; do
   echo -n $N
   let N=$N+1
done
```

Trying some useful text manipulation programs

Bash is great and has lots of built-in commands, but it usually needs some help to do anything really useful. Some of the most common useful programs you'll see used are `grep`, `cut`, `tr`, `awk`, and `sed`. As with all of the best UNIX tools, most of these programs are designed to work with standard input and standard output, so you can easily use them with pipes and shell scripts.

The general regular expression parser

The name *general regular expression print* (`grep`) sounds intimidating, but `grep` is just a way to find patterns in files or text. Think of it as a useful search tool. Gaining expertise with regular expressions is quite a challenge, but after you master it, you can accomplish many useful things with just the simplest forms.

For example, you can display a list of all regular user accounts by using `grep` to search for all lines that contain the text `/home` in the `/etc/passwd` file as follows:

```
$ grep /home /etc/passwd
```

Or you could find all environment variables that begin with `HO` using the following command:

```
$ env | grep ^HO
```

NOTE

The `^` in the preceding code is the actual caret character, `^`, not what you'll commonly see for a backspace, `^H`. Type `^`, `H`, and `O` (the uppercase letter) to see what items start with the uppercase characters HO.

To find a list of options to use with the `grep` command, type **`man grep`**.

Remove sections of lines of text (cut)

The `cut` command can extract fields from a line of text or from files. It is very useful for parsing system configuration files into easy-to-digest chunks. You can specify the field separator you want to use and the fields you want, or you can break up a line based on bytes.

The following example lists all home directories of users on your system. This `grep` command line pipes a list of regular users from the `/etc/passwd` file and displays the sixth field (`-f6`) as delimited by a colon (`-d':'`). The hyphen at the end tells `cut` to read from standard input (from the pipe).

```
$ grep /home /etc/passwd | cut  -d':' -f6 -
/home/chris
/home/joe
```

Translate or delete characters (tr)

The `tr` command is a character-based translator that can be used to replace one character or set of characters with another or to remove a character from a line of text.

The following example translates all uppercase letters to lowercase letters and displays the words `mixed upper and lower case` as a result:

```
$ FOO="Mixed UPpEr aNd LoWeR cAsE"
$ echo $FOO | tr [A-Z] [a-z]
mixed upper and lower case
```

In the next example, the `tr` command is used on a list of filenames to rename any files in that list so that any tabs or spaces (as indicated by the `[:blank:]` option) contained in a filename are translated into underscores. Try running the following code in a test directory:

```
for file in * ; do
   f=`echo $file | tr [:blank:] [_]`
   [ "$file" = "$f" ] || mv -i -- "$file" "$f"
done
```

The stream editor (sed)

The `sed` command is a simple scriptable editor, so it can perform only simple edits, such as removing lines that have text matching a certain pattern, replacing one pattern of characters with another, and so on. To get a better idea of how `sed` scripts work, there's no substitute for the online documentation, but here are some examples of common uses.

You can use the `sed` command essentially to do what I did earlier with the `grep` example: search the `/etc/passwd` file for the word `home`. Here the `sed` command searches the entire `/etc/passwd` file, searches for the word `home`, and prints any line containing the word `home`:

```
$ sed -n '/home/p' /etc/passwd
chris:x:1000:1000:Chris Negus:/home/chris:/bin/bash
joe:x:1001:1001:Joe Smith:/home/joe:/bin/bash
```

In this next example, `sed` searches the file `somefile.txt` and replaces every instance of the string `Mac` with `Linux`. Notice that the letter `g` is needed at the end of the substitution command to cause every occurrence of `Mac` on each line to be changed to `Linux`. (Otherwise, only the first instance of `Mac` on each line is changed.) The output is then sent to the `fixed_file.txt` file. The output from `sed` goes to `stdout`, so this command redirects the output to a file for safekeeping.

```
$ sed 's/Mac/Linux/g' somefile.txt > fixed_file.txt
```

You can get the same result using a pipe:

```
$ cat somefile.txt | sed 's/Mac/Linux/g' > fixed_file.txt
```

By searching for a pattern and replacing it with a null pattern, you delete the original pattern. This example searches the contents of the `somefile.txt` file and replaces extra blank spaces at the end of each line (`s/ *$`) with nothing (`//`). Results go to the `fixed_file.txt` file.

```
$ cat somefile.txt | sed 's/ *$//' > fixed_file.txt
```

Using simple shell scripts

Sometimes, the simplest of scripts can be the most useful. If you type the same sequence of commands repetitively, it makes sense to store those commands (once!) in a file. The following sections offer a couple of simple, but useful, shell scripts.

Telephone list

This idea has been handed down from generation to generation of old UNIX hacks. It's really quite simple, but it employs several of the concepts just introduced.

```
#!/bin/bash
# (@)/ph
# A very simple telephone list
# Type "ph new name number" to add to the list, or
# just type "ph name" to get a phone number

PHONELIST=~/.phonelist.txt

# If no command line parameters ($#), there
# is a problem, so ask what they're talking about.
if [ $# -lt 1 ] ; then
  echo "Whose phone number did you want? "
   exit 1
fi

# Did you want to add a new phone number?
if [ $1 = "new" ] ; then
  shift
  echo $* >> $PHONELIST
  echo $* added to database
  exit 0
fi

# Nope. But does the file have anything in it yet?
# This might be our first time using it, after all.
if [ ! -s $PHONELIST ] ; then
  echo "No names in the phone list yet! "
  exit 1
```

```
else
  grep -i -q "$*" $PHONELIST     # Quietly search the file
  if [ $? -ne 0 ] ; then         # Did we find anything?
    echo "Sorry, that name was not found in the phone list"
    exit 1
  else
    grep -i "$*" $PHONELIST
  fi
fi
exit 0
```

So, if you created the telephone list file as `ph` in your current directory, you could type the following from the shell to try out your `ph` script:

```
$ chmod 755 ph
$ ./ph new "Mary Jones" 608-555-1212
Mary Jones 608-555-1212 added to database
$ ./ph Mary
Mary Jones 608-555-1212
```

The `chmod` command makes the `ph` script executable. The `./ph` command runs the `ph` command from the current directory with the `new` option. This adds Mary Jones as the name and 608-555-1212 as the phone number to the database (`$HOME/.phonelist.txt`). The next `ph` command searches the database for the name Mary and displays the phone entry for Mary. If the script works, add it to a directory in your path (such as `$HOME/bin`).

Backup script

Because nothing works forever and mistakes happen, backups are just a fact of life when dealing with computer data. This simple script backs up all of the data in the home directories of all of the users on your Fedora or RHEL system.

```
#!/bin/bash
# (@)/my_backup
# A very simple backup script
#

# Change the TAPE device to match your system.
# Check /var/log/messages to determine your tape device.

TAPE=/dev/rft0

# Rewind the tape device $TAPE
mt $TAPE rew
# Get a list of home directories
HOMES=`grep /home /etc/passwd | cut -f6 -d':'`
# Back up the data in those directories
tar cvf $TAPE $HOMES
# Rewind and eject the tape.
mt $TAPE rewoffl
```

Summary

Writing shell scripts gives you the opportunity to automate many of your most common system administration tasks. This chapter covered common commands and functions that you can use in scripting with the bash shell. It also provided some concrete examples of scripts for doing backups and other procedures.

In the next chapter, you transition from learning about user features into examining system administration topics. Chapter 8, "Learning System Administration," covers how to become the root user, as well as how to use administrative commands, monitor log files, and work with configuration files.

Exercises

Use these exercises to test your knowledge of writing simple shell scripts. These tasks assume you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Create a script in your `$HOME/bin` directory called `myownscript`. When the script runs, it should output information that appears as follows:

   ```
   Today is Sat Jan 4 15:45:04 EST 2020.
   You are in /home/joe and your host is abc.example.com.
   ```

   Of course, you need to read in your current date/time, current working directory, and hostname. Also, include comments about what the script does and indicate that the script should run with the `/bin/bash` shell.

2. Create a script that reads in three positional parameters from the command line, assigns those parameters to variables named `ONE`, `TWO`, and `THREE`, respectively, and outputs that information in the following format:

   ```
   There are X parameters that include Y.
   The first is A, the second is B, the third is C.
   ```

   Replace X with the number of parameters and Y with all parameters entered. Then replace A with the contents of variable `ONE`, B with variable `TWO`, and C with variable `THREE`.

3. Create a script that prompts users for the name of the street and town where they grew up. Assign town and street to variables called `mytown` and `mystreet`, and output them with a sentence that reads as shown below (of course, `$mystreet` and `$mytown` will appear with the actual town and street the user enters):

   ```
   The street I grew up on was $mystreet and the town was
   $mytown
   ```

4. Create a script called `myos` that asks the user, "What is your favorite operating system?" Output an insulting sentence if the user types "Windows" or "Mac." Respond "Great choice!" if the user types "Linux." For anything else, say "Is <*what is typed in*> an operating system?"
5. Create a script that runs through the words *moose*, *cow*, *goose*, and *sow* through a `for` loop. Have each of those words appended to the end of the line "I have a. . . ."

Part III

Becoming a Linux System Administrator

IN THIS PART

Chapter 8
Learning System Administration

Chapter 9
Installing Linux

Chapter 10
Getting and Managing Software

Chapter 11
Managing User Accounts

Chapter 12
Managing Disks and Filesystems

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER

8

Learning System Administration

IN THIS CHAPTER

Doing graphical administration

Using the root login

Understanding administrative commands, config files, and log files

Working with devices and filesystems

Linux, like other UNIX-based systems, was intended for use by more than one person at a time. *Multiuser features* enable many people to have accounts on a single Linux system with their data kept secure from others. *Multitasking* enables many people to run many programs on the computer at the same time, with each person able to run more than one program. Sophisticated networking protocols and applications make it possible for a Linux system to extend its capabilities to network users and computers around the world. The person assigned to manage all of a Linux system's resources is called the *system administrator*.

Even if you are the only person using a Linux system, system administration is still set up to be separate from other computer use. To do most administrative tasks, you need to be logged in as the *root user* (also called the *superuser*) or to get root permission temporarily (usually using the `sudo` command). Regular users who don't have root permission cannot change, or in some cases cannot even see, some of the configuration information for a Linux system. In particular, security features such as stored passwords are protected from general view.

Because Linux system administration is such a huge topic, this chapter focuses on the general principles of Linux system administration. In particular, it examines some of the basic tools that you need to administer a Linux system for a personal desktop or on a small server. Beyond the basics, this chapter also teaches you how to work with filesystems and monitor the setup and performance of your Linux system.

Understanding System Administration

Separating the role of system administrator from that of other users has several effects. For a system that has many people using it, limiting who can manage it enables you to keep it more

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

secure. A separate administrative role also prevents others from casually harming your system when they are just using it to write a document or browse the Internet.

If you are the system administrator of a Linux system, you generally log in as a regular user account and then ask for administrative privileges when you need them. This is often done with one of the following:

`su` command: Often, `su` is used to open a shell as root user. After the shell is open, the administrator can run multiple commands and then exit to return to a shell as a regular user.

`sudo` command: With `sudo`, a regular user is given root privileges, but only when that user runs the `sudo` command to run another command. After running that one command with `sudo`, the user is immediately returned to a shell and acts as the regular user again. Ubuntu and Fedora by default assign `sudo` privilege to the first user account when those systems are installed. This is not done by default in RHEL, although during RHEL installation, you can choose for your first user to have `sudo` privilege if you'd like.

Cockpit browser-based administration: RHEL, Fedora, and other Linux distributions have committed to Cockpit as their primary browser-based system administration facility. With Cockpit enabled, you can monitor and change your system's general activities, storage, networking, accounts, services, and other features.

Graphical windows: Before Cockpit was widely available, RHEL, Fedora, and other Linux distributions offered individual graphical administration tools that were launched by commands beginning with `system-config-*`. Although most of these administration tools are not being offered in the latest release of RHEL and Fedora, they are noted here because they are still available in older Linux releases.

Tasks that can be done only by the root user tend to be those that affect the system as a whole or impact the security or health of the system. Following is a list of common features that a system administrator is expected to manage:

Filesystems: When you first install Linux, the directory structure is set up to make the system usable. However, if users later want to add extra storage or change the filesystem layout outside of their home directory, they need administrative privileges to do that. Also, the root user has permission to access files owned by any user. As a result, the root user can copy, move, or change any other user's files—a privilege needed to make backup copies of the filesystem for safekeeping.

Software installation: Because malicious software can harm your system or make it insecure, you need root privilege to install software so that it is available to all users on your system. Regular users can still install some software in their own directories and can list information about installed system software.

User accounts: Only the root user can add and remove user accounts and group accounts.

Network interfaces: In the past, the root user had to configure network interfaces and start and stop those interfaces. Now, many Linux desktops allow regular users to start and stop network interfaces from their desktop using Network Manager. This is particularly true for wireless network interfaces, which can come and go by location as you move your Linux laptop or handheld device around.

Servers: Configuring web servers, file servers, domain name servers, mail servers, and dozens of other servers requires root privilege, as does starting and stopping those services. Content, such as web pages, can be added to servers by non-root users if you configure your system to allow that. Services are often run as special administrative user accounts, such as `apache` (for the `httpd` service) and `rpc` (for the `rpcbind` service). So, if someone cracks a service, they can't get root privilege to other services or system resources.

Security features: Setting up security features, such as firewalls and user access lists, is usually done with root privilege. It's also up to the root user to monitor how the services are being used and to make sure that server resources are not exhausted or abused.

The easiest way to begin system administration is to use some graphical administration tools.

Using Graphical Administration Tools

Most system administration for the first Linux systems was done from the command line. As Linux became more popular, however, both graphical and command-line interfaces began to be offered for most common Linux administrative tasks.

The following sections describe some of the point-and-click types of interfaces that are available for doing system administration in Linux.

Using Cockpit browser-based administration

Cockpit is the best browser-based Linux system administration tool that I have ever seen. It brings together a range of Linux administrative activities into one interface and taps into a diverse set of Linux APIs using cockpit-bridge. As someone doing Linux administration, however, you just need to know that you will get a consistent and stable way of administering your systems with Cockpit.

Getting started with Cockpit is as simple as enabling the cockpit socket and pointing a web browser at the Cockpit service. Because of Cockpit's plug-in design, there are new tools being created all the time that you can add to your system's Cockpit interface.

If you are starting with the latest RHEL or Fedora systems, performing the following procedure lets you enable and start using Cockpit on your system.

Note

No configuration is required to start this procedure. However, you can configure Cockpit to use your own OpenSSL certificate instead of the self-signed one used by default. This lets you avoid having to accept the unverified self-signed certificate when you open the Cockpit interface from your browser.

1. If Cockpit is not already installed, do the following:

```
# dnf install cockpit
```

2. Log in as root user, and enable the Cockpit socket:

```
# systemctl enable --now cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/
cockpit.socket
    → /usr/lib/systemd/system/cockpit.socket.
```

3. Open your web browser to port 9090 on the system where you just enabled Cockpit. You can use the hostname or IP address. Port 9090 is configured for https by default, although you can reconfigure that if you like to use http. Here are examples of addresses to type into your browser's address bar:

```
https://host1.example.com:9090/
https://192.168.122.114:9090/
```

4. Assuming you didn't replace the self-signed certificate for Cockpit, you are warned that the connection is not safe. To accept it anyway, and depending on your browser, you must select Advanced and agree to an exception to allow the browser to use the Cockpit service.
5. Enter your username and password. Use the root user or a user with `sudo` privileges if you want to change your system configuration. A regular user can see but not change most settings. Figure 8.1 shows an example of this window.
6. Begin using Cockpit. The Cockpit dashboard contains a good set of features by default (you can add more later) on RHEL and Fedora systems. Figure 8.2 shows an example of the System area of the Cockpit dashboard:

Immediately after logging in to Cockpit, you begin seeing system activity related to CPU usage, memory and swap, disk input/output, and network traffic. Selections in the left navigation pane let you begin working with logs, storage, networking, user and group accounts, services, and many other features on your system.

As you proceed through the rest of this book, you will see descriptions of how to use the different features of Cockpit in the appropriate section. To dive deeper into any of the topics that you encounter with Cockpit, I recommend checking out the Cockpit project website: `https://cockpit-project.org`.

FIGURE 8.1

Logging in to Cockpit

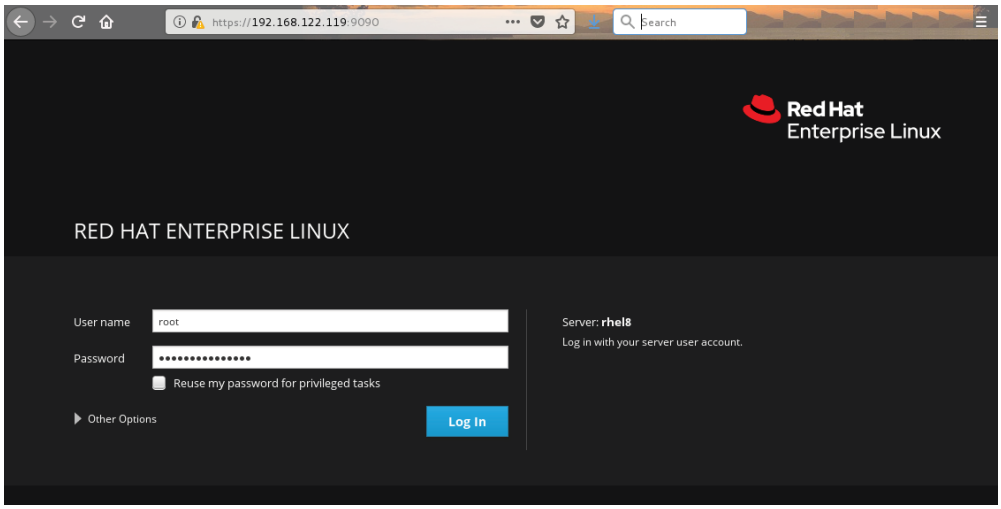


FIGURE 8.2

View system activity and other topics from the Cockpit dashboard.



Using system-config-* tools

On Fedora and RHEL systems prior to the release of Cockpit, a set of graphical tools was available from the Administration submenu of the System menu (GNOME 2), from the

Activities screen (GNOME 3), or from the command line. On these older Fedora and RHEL systems, you could operate these tools from the command line by running a set of commands that began with the `system-config*` string (such as `system-config-network`).

These `system-config*` tools require root permission. If you are logged in as a regular user, you must enter the root password before the graphical user interface (GUI) application's window opens or, in some cases, when you request to do some special activity.

The following list describes many of the graphical tools available in earlier Fedora or RHEL systems. (Some were only in Fedora and many are not installed by default.) The command that you would launch to get the feature is shown in parentheses (often, it is the same as the package name). The following graphical tools were available in Fedora:

Domain Name System (`system-config-bind`): Create and configure zones if your computer is acting as a DNS server.

HTTP (`system-config-httpd`): Configure your computer as an Apache web server.

NFS (`system-config-nfs`): Set up directories from your system to be shared with other computers on your network using the NFS service.

Root Password (`system-config-rootpassword`): Change the root password.

Samba NFS (`system-config-samba`): Configure Windows (SMB) file sharing. (To configure other Samba features, you can use the SWAT window.)

The following graphical tools were available in both Fedora and RHEL systems prior to RHEL 8:

Services (`system-config-services`): Display and change which services are running on your Fedora system at different runlevels from the Service Configuration window.

Authentication (`system-config-authentication`): Change how users are authenticated on your system. Typically, shadow passwords and MD5 passwords are selected. However, if your network supports LDAP, Kerberos, SMB, NIS, or Hesiod authentication, you can select to use any of those authentication types.

Date & Time (`system-config-date`): Set the date and time or choose to have an NTP server keep system time in sync.

Firewall (`system-config-firewall`): Configure your firewall to allow or deny services to computers from the network.

Language (`system-config-language`): Select the default language used for the system.

Printing (`system-config-printer`): Configure local and network printers.

SELinux Management (`system-config-selinux`): Set SELinux enforcing modes and default policy.

Users & Groups (`system-config-users`): Add, display, and change user and group accounts for your Fedora system.

Other administrative utilities were available from the Applications menu on the top panel. Select the System Tools submenu (in GNOME 2) or go to the Activities screen (in GNOME 3) to choose some of the following tools (if installed):

Configuration Editor (`gconf-editor`): Directly edit the GNOME configuration database.

Disk Usage Analyzer (`gnome-utils`): Display detailed information about your hard disks and removable storage devices.

Disk Utility (`gnome-disks`): Manage disk partitions and add filesystems (`gnome-disk-utility` package).

Kickstart (`system-config-kickstart`): Create a kickstart configuration file that can be used to install multiple Linux systems without user interaction.

Descriptions from previous editions of this book of most of these tools have been replaced by procedures using Cockpit instead.

Using other browser-based admin tools

To simplify the management of many enterprise-quality open source projects, those projects have begun offering browser-based graphical management tools. In most cases, command-line tools are offered for managing these projects as well.

For example, if you are using Red Hat Enterprise Linux, there are browser-based interfaces for managing the following projects:

Red Hat OpenShift: *OpenShift*, based on the Kubernetes project, offers a browser-based interface for deploying and managing a cluster of control plane and worker nodes as well as features for deploying and managing containers in what are referred to as *pods*. See the Red Hat OpenShift site at `www.openshift.com` or the upstream OKD site at `www.okd.io` for details.

Red Hat Enterprise Linux OpenStack Platform (RHELOSP): The OpenStack platform-as-a-service project lets you manage your own private, hybrid cloud through your browser. This includes the OpenStack dashboard from the OpenStack Horizon project (`http://horizondocs.openstack.org/horizon/latest`). That interface lets you launch and manage virtual machines and all of the resources around them: storage, networking, authentication, processing allocations, and so on. Refer to Chapter 27, "Using Linux for Cloud Computing," for a description of how to use the OpenStack Dashboard.

Red Hat Virtualization (RHV): With RHEV, the RHV manager provides the browser-based interface for managing virtual machines, including allocating storage and user access to resources. Many other examples of browser-based graphical administration tools are available with open source projects. If you are new to Linux, it can be easier to get started with these interfaces. However, keep in mind that often you need to use command-line tools if you need to troubleshoot problems because graphical tools are often limited in that area.

Using the root User Account

Every Linux system starts out with at least one administrative user account (the root user) and possibly one or more regular user accounts (given a name that you choose, or a name assigned by your Linux distribution). In most cases, you log in as a regular user and become the root user to do an administrative task.

The root user has complete control of the operation of your Linux system. That user can open any file or run any program. The root user also installs software packages and adds accounts for other people who use the system.

Tip

Think of the root user in Linux as similar to the Administrator user in Windows.

When you first install most Linux systems (although not all systems), you add a password for the root user. You must remember and protect this password; you need it to log in as root or to obtain root permission while you are logged in as some other user.

To become familiar with the root user account, you can simply log in as the root user. I recommend trying this from a virtual console. To do so, press Ctrl+Alt+F3. When you see the login prompt, type **`root`** (press Enter) and enter the password. A login session for root opens. When you are finished, type **`exit`**, and then press Ctrl+Alt+F1 to return to the regular desktop login.

After you have logged in as root, the home directory for the root user is typically `/root`. The home directory and other information associated with the root user account are located in the `/etc/passwd` file. Here's what the root entry looks like in the `/etc/passwd` file:

```
root:x:0:0:root:/root:/bin/bash
```

This shows that for the user named `root`, the user ID is set to `0` (root user), the group ID is set to `0` (root group), the home directory is `/root`, and the shell for that user is `/bin/bash`. (Linux uses the `/etc/shadow` file to store encrypted password data, so the password field here contains an `x`.) You can change the home directory or the shell used by editing the values in this file. A better way to change these values, however, is to use the `usermod` command (see the section "Modifying Users with usermod" in Chapter 11 for further information).

At this point, any command that you run from your shell is run with root privilege. So be careful. You have much more power to change (and damage) the system than you did as a regular user. Again, type **`exit`** when you are finished. If you are on a virtual console and have a desktop interface running on another console, press Ctrl+Alt+F1 to return to the graphical login screen if you are using a Linux desktop system.

NOTE
By default, the root account has no password set in Ubuntu. This means that even though the account exists, you cannot log in using it or use `su` to become the root user. This adds an additional level of security to Ubuntu and requires you to use `sudo` before each command that you want to execute as the root user.

Becoming root from the shell (su command)

Although you can become the superuser by logging in as root, sometimes that is not convenient.

For example, you may be logged in to a regular user account and just want to make a quick administrative change to your system without having to log out and log back in. You may need to log in over the network to make a change to a Linux system but find that the system doesn't allow root users in from over the network (a common practice for secure Linux systems). One solution is to use the `su` command. From any Terminal window or shell, you can simply type the following:

```
$ su
Password: ******
#
```

When you are prompted, type the root user's password. The prompt for the regular user (`$`) changes to the superuser prompt (#). At this point, you have full permission to run any command and use any file on the system. However, one thing that the `su` command doesn't do when used this way is read in the root user's environment. As a result, you may type a command that you know is available and get the message `Command Not Found`. To fix this problem, use the `su` command with the dash (-) option instead like this:

```
$ su -
Password: ******
#
```

You still need to type the password, but after that everything that normally happens at login for the root user happens after the `su` command is completed. Your current directory will be root's home directory (probably `/root`), and things such as the root user's `PATH` variable are used. If you become the root user by just typing `su`, rather than `su -`, you don't change directories or the environment of the current login session.

You can also use the `su` command to become a user other than root. This is useful for troubleshooting a problem that is being experienced by a particular user but not by others on the computer (such as an inability to print or send email). For example, to have the permissions of a user named *jsmith*, you'd type the following:

```
$ su - jsmith
```

Even if you were root user before you typed this command, afterward you would have only the permissions to open files and run programs that are available to jsmith. As root user, however, after you type the `su` command to become another user, you don't need a password to continue. If you type that command as a regular user, you must type the new user's password.

When you are finished using superuser permissions, return to the previous shell by exiting the current shell. Do this by pressing Ctrl+D or by typing **`exit`**. If you are the administrator for a computer that is accessible to multiple users, don't leave a root shell open on someone else's screen unless you want to give that person freedom to do anything he or she wants to the computer!

Allowing administrative access via the GUI

As mentioned earlier, when you run GUI tools as a regular user (from Fedora, Red Hat Enterprise Linux, or some other Linux systems), you are prompted for the root password before you are able to access the tool. By entering the root password, you are given root privilege for that task.

For Linux systems using the GNOME 2 desktop, after you enter the password, a yellow badge icon appears in the top panel, indicating that root authorization is still available for other GUI tools to run from that desktop session. For GNOME 3 desktops, you must enter the root password each time you start any of the system-config tools.

Gaining administrative access with sudo

Particular users can also be given administrative permissions for particular tasks or any task by typing `sudo` followed by the command they want to run, without being given the root password. The `sudoers` facility is the most common way to provide such privilege. Using `sudoers` for any users or groups on the system, you can do the following:

- Assign root privilege for any command they run with `sudo`.
- Assign root privilege for a select set of commands.
- Give users root privilege without telling them the root password because they only have to provide their own user password to gain root privilege.
- Allow users, if you choose, to run `sudo` without entering a password at all.
- Track which users have run administrative commands on your system. (Using `su`, all you know is that someone with the root password logged in, whereas the `sudo` command logs which user runs an administrative command.)

With the `sudoers` facility, giving full or limited root privileges to any user simply entails adding the user to `/etc/sudoers` and defining what privilege you want that user to have. Then the user can run any command they are privileged to use by preceding that command with the `sudo` command.

Here's an example of how to use the `sudo` facility to cause the user named *joe* to have full root privilege.

Tip

If you look at the `sudoers` file in Ubuntu, you see that the initial user on the system already has privilege, by default, for the `sudo` group members. To give any other user the same privilege, you could simply add the additional user to the admin group when you run `visudo`.

1. As the root user, edit the `/etc/sudoers` file by running the `visudo` command:

   ```
   # /usr/sbin/visudo
   ```

   By default, the file opens in `vi`, unless your `EDITOR` variable happens to be set to some other editor acceptable to `visudo` (for example, `export EDITOR=gedit`). The reason for using `visudo` is that the command locks the `/etc/sudoers` file and does some basic sanity checking of the file to ensure that it has been edited correctly.

Note

If you are stuck here, try running the `vimtutor` command for a quick tutorial on using `vi` and `vim`.

2. Add the following line to allow joe to have full root privileges on the computer:

   ```
   joe     ALL=(ALL)     ALL
   ```

   This line causes joe to provide a password (his own password, not the root password) in order to use administrative commands. To allow joe to have that privilege without using a password, type the following line instead:

   ```
   joe     ALL=(ALL)     NOPASSWD: ALL
   ```

3. Save the changes to the `/etc/sudoers` file (in `vi`, type **Esc** and then **:wq**). The following is an example of a session by the user joe after he has been assigned `sudo` privileges:

   ```
   [joe]$ sudo touch /mnt/testfile.txt
     We trust you have received the usual lecture
     from the local System Administrator. It usually
     boils down to these two things:
       #1) Respect the privacy of others.
       #2) Think before you type.
   Password: *********
   [joe]$ ls -l /mnt/testfile.txt
   -rw-r--r--. 1 root root 0 Jan  7 08:42 /mnt/testfile.txt
   [joe]$ rm /mnt/testfile.txt
   rm: cannot remove `/mnt/testfile.txt': Permission denied
   [joe]$ sudo rm /mnt/textfile.txt
   [joe]$
   ```

In this session, the user joe runs the `sudo` command to create a file (`/mnt/textfile.txt`) in a directory for which he doesn't have write permission. He is given a warning and asked to provide his password (this is joe's password, *not* the root password).

Even after joe has entered the password, he must still use the `sudo` command to run subsequent administrative commands as root (the `rm` fails, but the `sudo rm` succeeds). Notice that he is not prompted for a password for the second `sudo`. That's because after entering his password successfully, he can enter as many `sudo` commands as he wants for the next five minutes, on RHEL and Fedora systems, without having to enter it again. For Ubuntu, this is set to zero, for no time-out. (You can change the time-out value from five minutes to any length of time you want by setting the `passwd_timeout` value in the `/etc/sudoers` file.)

The preceding example grants a simple all-or-nothing administrative privilege to joe. However, the `/etc/sudoers` file gives you an incredible amount of flexibility in permitting individual users and groups to use individual applications or groups of applications. Refer to the `sudoers` and `sudo` man pages for information about how to tune your `sudo` facility.

Exploring Administrative Commands, Configuration Files, and Log Files

You can expect to find many commands, configuration files, and log files in the same places in the filesystem, regardless of which Linux distribution you are using. The following sections give you some pointers on where to look for these important elements.

Note

If GUI administrative tools for Linux have become so good, why do you need to know about administrative files? For one thing, while GUI tools differ among Linux versions, many underlying configuration files are the same. So if you learn to work with them, you can work with almost any Linux system. Also, if a feature is broken, or if you need to do something that's not supported by the GUI, when you ask for help, Linux experts almost always tell you how to run commands or change the configuration file directly.

Administrative commands

Only the root user is intended to use many administrative commands. When you log in as root (or use `su -` from the shell to become root), your `$PATH` variable is set to include some directories that contain commands for the root user. In the past, these have included the following:

`/sbin`: Originally contained commands needed to boot your system, including commands for checking filesystems (`fsck`) and turning on swap devices (`swapon`).

/usr/sbin: Originally contained commands for such things as managing user accounts (such as `useradd`) and checking processes that are holding files open (such as `lsof`). Commands that run as daemon processes are also contained in this directory. *Daemon processes* are processes that run in the background, waiting for service requests such as those to access a printer or a web page. (Look for commands that end in `d`, such as `sshd`, `pppd`, and `cupsd`.)

For the latest Ubuntu, RHEL and Fedora releases, all administrative commands from the two directories are stored in the `/usr/sbin` directory (which is symbolically linked from `/sbin`). Also, only `/usr/sbin` is added to the PATH of the root user, as well as the PATH of all regular users.

Some administrative commands are contained in regular user directories (such as `/bin` and `/usr/bin`). This is especially true of commands that have some options available to everyone. An example is the `/bin/mount` command, which anyone can use to list mounted filesystems but only root can use to mount filesystems. (Some desktops, however, are configured to let regular users use `mount` to mount CDs, DVDs, or other removable media.)

Note

See the section "Mounting Filesystems" in Chapter 12 for instructions on how to mount a filesystem.

To find commands intended primarily for the system administrator, check out the section 8 manual pages (usually in `/usr/share/man/man8`). They contain descriptions and options for most Linux administrative commands. If you want to add commands to your system, consider adding them to directories such as `/usr/local/bin` or `/usr/local/sbin`. Some Linux distributions automatically add those directories to your PATH, usually before your standard `bin` and `sbin` directories. In that way, commands installed to those directories are not only accessible, but can also override commands of the same name in other directories. Some third-party applications that are not included with Linux distributions are sometimes placed in the `/usr/local/bin`, `/opt/bin`, or `/usr/local/sbin` directory.

Administrative configuration files

Configuration files are another mainstay of Linux administration. Almost everything that you set up for your particular computer—user accounts, network addresses, or GUI preferences—results in settings being stored in plain-text files. This has some advantages and some disadvantages.

The advantage of plain-text files is that it's easy to read and change them. Any text editor will do. The downside, however, is that as you edit configuration files, traditionally no error checking is done. You sometimes have to run the program that reads these files (such as a network daemon or the X desktop) to find out whether you set up the files correctly.

While some configuration files use standard structures, such as XML for storing information, many do not. So, you need to learn the specific structure rules for each configuration

file. A comma or a quote in the wrong place can sometimes cause an entire interface to fail. You can check in many ways that the structure of many configuration files is correct.

Some software packages offer a command to test the sanity of the configuration file tied to a package before you start a service. For example, the `testparm` command is used with Samba to check the sanity of your `smb.conf` file. Other times, the daemon process providing a service offers an option for checking your config file. For example, run `httpd -t` to check your Apache web server configuration before starting your web server.

NOTE

Some text editors, such as the `vim` command (not `vi`), understand the structure of some types of configuration files. If you open such a configuration file in `vim`, notice that different elements of the file are shown in different colors. In particular, you can see comment lines in a different color than data.

Throughout this book, you'll find descriptions of the configuration files that you need to set up the different features that make up Linux systems. The two major locations of configuration files are your home directory (where your personal configuration files are kept) and the `/etc` directory (which holds system-wide configuration files).

Following are descriptions of directories (and subdirectories) that contain useful configuration files. The descriptions are followed by some individual configuration files in `/etc` that are of particular interest. Viewing the contents of Linux configuration files can teach you a lot about administering Linux systems.

`$HOME`: All users store in their home directories information that directs how their login accounts behave. Many configuration files are stored directly in each user's home directory (such as `/home/joe`) and begin with a dot (.), so they don't appear in a user's directory when you use a standard `ls` command (you need to type **`ls -a`** to see them). Likewise, dot files and directories won't show up in most file manager windows by default. There are dot files that define the behavior of each user's shell, the desktop look-and-feel, and options used with your text editor. There are even files such as those in each user's `$HOME/.ssh` directory that configure permissions for logging into remote systems. (To see the name of your home directory, type **`echo $HOME`** from a shell.)

`/etc`: This directory contains most of the basic Linux system configuration files.

`/etc/cron*`: Directories in this set contain files that define how the `crond` utility runs applications on a daily (`cron.daily`), hourly (`cron.hourly`), monthly (`cron.monthly`), or weekly (`cron.weekly`) schedule.

`/etc/cups`: Contains files used to configure the CUPS printing service.

`/etc/default`: Contains files that set default values for various utilities. For example, the file for the `useradd` command defines the default group number, home directory, password expiration date, shell, and skeleton directory (`/etc/skel`) used when creating a new user account.

/etc/httpd: Contains a variety of files used to configure the behavior of your Apache web server (specifically, the `httpd` daemon process). (On Ubuntu and other Linux systems, `/etc/apache` or `/etc/apache2` is used instead.)

/etc/mail: Contains files used to configure your `sendmail` mail transport agent.

/etc/postfix: Contains configuration files for the `postfix` mail transport agent.

/etc/ppp: Contains several configuration files used to set up Point-to-Point Protocol (PPP) so that you can have your computer dial out to the Internet. (PPP was more commonly used when dial-up modems were popular.)

/etc/rc?.d: There is a separate `rc?.d` directory for each valid system state: `rc0.d` (shutdown state), `rc1.d` (single-user state), `rc2.d` (multiuser state), `rc3.d` (multiuser plus networking state), `rc4.d` (user-defined state), `rc5.d` (multiuser, networking, plus GUI login state), and `rc6.d` (reboot state). These directories are maintained for compatibility with old UNIX SystemV init services.

/etc/security: Contains files that set a variety of default security conditions for your computer, basically defining how authentication is done. These files are part of the `pam` (pluggable authentication modules) package.

/etc/skel: Any files contained in this directory are automatically copied to a user's home directory when that user is added to the system. By default, most of these files are dot (.) files, such as `.kde` (a directory for setting KDE desktop defaults) and `.bashrc` (for setting default values used with the bash shell).

/etc/sysconfig: Contains important system configuration files that are created and maintained by various services (including `firewalld`, `samba`, and most networking services). These files are critical for Linux distributions, such as Fedora and RHEL, that use GUI administration tools but are not used on other Linux systems at all.

/etc/systemd: Contains files associated with the `systemd` facility, for managing the boot process and system services. In particular, when you run `systemctl` commands to enable and disable services, files that make that happen are stored in subdirectories of the `/etc/systemd system` directory.

/etc/xinetd.d: Contains a set of files, each of which defines an on-demand network service that the `xinetd` daemon listens for on a particular port. When the `xinetd` daemon process receives a request for a service, it uses the information in these files to determine which daemon processes to start to handle the request.

The following are some interesting configuration files in `/etc`:

aliases: Can contain distribution lists used by the Linux mail services. (This file is located in `/etc/mail` in Ubuntu when you install the `sendmail` package.)

bashrc: Sets system-wide defaults for bash shell users. (This may be called `bash.bashrc` on some Linux distributions.)

crontab: Sets times for running automated tasks and variables associated with the `cron` facility (such as the `SHELL` and `PATH` associated with `cron`).

csh.cshrc (or **cshrc**): Sets system-wide defaults for `csh` (C shell) users.

exports: Contains a list of local directories that are available to be shared by remote computers using the Network File System (NFS).

fstab: Identifies the devices for common storage media (hard disk, DVD, CD-ROM, and so on) and locations where they are mounted in the Linux system. This is used by the `mount` command to choose which filesystems to mount when the system first boots.

group: Identifies group names and group IDs (GIDs) that are defined on the system. Group permissions in Linux are defined by the second of three sets of `rwx` (read, write, execute) bits associated with each file and directory.

gshadow: Contains shadow passwords for groups.

host.conf: Used by older applications to set the locations in which domain names (for example, `redhat.com`) are searched for on TCP/IP networks (such as the Internet). By default, the local hosts file is searched and then any name server entries in `resolv.conf`.

hostname: Contains the hostname for the local system (beginning in RHEL 7 and recent Fedora and Ubuntu systems).

hosts: Contains IP addresses and hostnames that you can reach from your computer. (Usually this file is used just to store names of computers on your LAN or small private network.)

inittab: On earlier Linux systems, contained information that defined which programs start and stop when Linux boots, shuts down, or goes into different states in between. This configuration file was the first one read when Linux started the `init` process. This file is no longer used on Linux systems that support `systemd`.

mtab: Contains a list of filesystems that are currently mounted.

mtools.conf: Contains settings used by DOS tools in Linux.

named.conf: Contains DNS settings if you are running your own DNS server (`bind` or `bind9` package).

nsswitch.conf: Contains name service switch settings, for identifying where critical system information (user accounts, hostname-to-address mappings, and so on) comes from (local host or via network services).

ntp.conf: Includes information needed to run the Network Time Protocol (NTP).

passwd: Stores account information for all valid users on the local system. Also includes other information, such as the home directory and default shell. (Rarely includes the user passwords themselves, which are typically stored in the `/etc/shadow` file.)

printcap: Contains definitions for the printers configured for your computer. (If the `printcap` file doesn't exist, look for printer information in the `/etc/cups` directory.)

profile: Sets system-wide environment and startup programs for all users. This file is read when the user logs in.

protocols: Sets protocol numbers and names for a variety of Internet services.

rpc: Defines remote procedure call names and numbers.

services: Defines TCP/IP and UDP service names and their port assignments.

shadow: Contains encrypted passwords for users who are defined in the `passwd` file. (This is viewed as a more secure way to store passwords than the original encrypted password in the `passwd` file. The `passwd` file needs to be publicly readable, whereas the `shadow` file can be unreadable by all but the root user.)

shells: Lists the shell command-line interpreters (`bash`, `sh`, `csh`, and so on) that are available on the system as well as their locations.

sudoers: Sets commands that can be run by users, who may not otherwise have permission to run the command, using the `sudo` command. In particular, this file is used to provide selected users with root permission.

rsyslog.conf: Defines what logging messages are gathered by the `rsyslogd` daemon and in which files they are stored. (Typically, log messages are stored in files contained in the `/var/log` directory.)

xinetd.conf: Contains simple configuration information used by the `xinetd` daemon process. This file mostly points to the `/etc/xinetd.d` directory for information about individual services.

Another directory, `/etc/X11`, includes subdirectories that each contain system-wide configuration files used by X and different X window managers available for Linux. The `xorg.conf` file (configures your computer and monitor to make it usable with X) and configuration directories containing files used by `xdm` and `xinit` to start X are in here.

Directories relating to window managers contain files that include the default values that a user will get if that user starts one of these window managers on your system. The twm window manager may have system-wide configuration files in these directories.

Administrative log files and systemd journal

One of the things that Linux does well is keep track of itself. This is a good thing when you consider how much is going on in a complex operating system.

Sometimes you are trying to get a new facility to work and it fails without giving you the foggiest reason why. Other times, you want to monitor your system to see whether people are trying to access your computer illegally. In any of those cases, you want to be able to refer to messages coming from the kernel and services running on the system.

For Linux systems that don't use the systemd facility, the main utility for logging error and debugging messages is the rsyslogd daemon. (Some older Linux systems use syslogd and syslogd daemons.) Although you can still use rsyslogd with systemd systems, systemd has its own method of gathering and displaying messages called the systemd journal (journalctl command).

Using journalctl to view the systemd journal

The primary command for viewing messages from the systemd journal is the journalctl command. The boot process, the kernel, and all systemd-managed services direct their status and error messages to the systemd journal.

Using the journalctl command, you can display journal messages in many different ways. Here are some examples:

```
# journalctl
# journalctl --list-boots | head
-2 93bdb6164... Sat 2020-01-04 21:07:28 EST—Sat 2020-01-04 21:19:37 EST
-1 7336cb823... Sun 2020-01-05 10:38:27 EST—Mon 2020-01-06 09:29:09 EST
 0 eaebac25f... Sat 2020-01-18 14:11:41 EST—Sat 2020-01-18 16:03:37 EST
# journalctl -b 488e152a3e2b4f6bb86be366c55264e7
# journalctl -k
```

In these examples, the journalctl command with no options lets you page through all messages in the systemd journal. To list the boot IDs for each time the system was booted, use the –list-boots option. To view messages associated with a particular boot instance, use the -b option with one of the boot instances. To see only kernel messages, use the -k option. Here are some more examples:

```
# journalctl _SYSTEMD_UNIT=sshd.service
# journalctl PRIORITY=0
# journalctl -a -f
```

Use the _SYSTEMD_UNIT= options to show messages for specific services (here, the sshd service) or for any other systemd unit file (such as other services or mounts). To see messages associated with a particular syslog log level, set PRIORITY= to a value from 0 to 7. In this case, only emergency (0) messages are shown. To follow messages as they come in, use the -f option; to show all fields, use the -a option.

Managing log messages with rsyslogd

The rsyslogd facility, and its predecessor syslogd, gather log messages and direct them to log files or remote log hosts. Logging is done according to information in the /etc/rsyslog.conf file. Messages are typically directed to log files that are usually in the /

var/log directory, but they can also be directed to log hosts for additional security. Here are a few common log files:

boot.log: Contains boot messages about services as they start up.

messages: Contains many general informational messages about the system.

secure: Contains security-related messages, such as login activity or any other act that authenticates users.

Refer to Chapter 13, "Understanding Server Administration," for information on configuring the rsyslogd facility.

Using Other Administrative Accounts

You don't hear much about logging in with other administrative user accounts (besides root) on Linux systems. It was a fairly common practice in UNIX systems to have several different administrative logins that allowed administrative tasks to be split among several users. For example, people sitting near a printer could have lp permissions to move print jobs to another printer if they knew a printer wasn't working.

In any case, administrative logins are available with Linux; however, logging in directly as those users is disabled by default. The accounts are maintained primarily to provide ownership for files and processes associated with particular services. When daemon processes are run under separate administrative logins, having one of those processes cracked does not give the cracker root permission and the ability to access other processes and files. Consider the following examples:

lp: User owns such things as the /var/log/cups printing log file and various printing cache and spool files. The home directory for lp is /var/spool/lpd.

apache: User can set up content files and directories on an Apache web server. It is primarily used to run the web server processes (httpd) in RHEL and Fedora systems, while the www-data user runs the Apache service (apache2) on Ubuntu systems.

avahi: User runs the avahi daemon process to provide zeroconf services on your network.

chrony: User runs the chronyd daemon, which is used to maintain accurate computer clocks.

postfix: User owns various mail server spool directories and files. The user runs the daemon processes used to provide the postfix service (master).

bin: User owns many commands in /bin in traditional UNIX systems. This is not the case in some Linux systems (such as Ubuntu, Fedora, and Gentoo) because root owns most executable files. The home directory of bin is /bin.

news: User could do administration of Internet news services, depending on how you set permission for /var/spool/news and other news-related resources. The home directory for news is /etc/news.

rpc: User runs the remote procedure calls daemon (rpcbind), which is used to receive calls for services on the host system. The NFS service uses the RPC service.

By default, the administrative logins in the preceding list are disabled. You would need to change the default shell from its current setting (usually /sbin/nologin or /bin/false) to a real shell (typically /bin/bash) to be able to log in as these users. As mentioned earlier, however, they are really not intended for interactive logins.

Checking and Configuring Hardware

In a perfect world, after installing and booting Linux, all of your hardware is detected and available for access. Although Linux systems have become quite good at detecting hardware, sometimes you must take special steps to get your computer hardware working. Also, the growing use of removable USB devices (CDs, DVDs, flash drives, digital cameras, and removable hard drives) has made it important for Linux to do the following:

- Efficiently manage hardware that comes and goes
- Look at the same piece of hardware in different ways. (For example, it should be able to see a printer as a fax machine, scanner, and storage device as well as a printer.)

Linux kernel features added in the past few years have made it possible to change drastically the way that hardware devices are detected and managed. The Udev subsystem dynamically names and creates devices as hardware comes and goes.

If this sounds confusing, don't worry. It's designed to make your life as a Linux user much easier. The result of features built on the kernel is that device handling in Linux has become more automatic and more flexible:

More automatic For most common hardware, when a hardware device is connected or disconnected, it is automatically detected and identified. Interfaces to access the hardware are added so it is accessible to Linux. Then the fact that the hardware is present (or removed) is passed to the user level, where applications listening for hardware changes are ready to mount the hardware and/or launch an application (such as an image viewer or music player).

More flexible If you don't like what happens automatically when a hardware item is connected or disconnected, you can change it. For example, features built into GNOME and KDE desktops let you choose what happens when a music CD or data DVD is inserted, or when a digital camera is connected. If you prefer that a different program be launched to handle it, you can easily make that change.

The following sections cover several issues related to getting your hardware working properly in Linux. First, it describes how to check information about the hardware components of your system. It then covers how to configure Linux to deal with removable media. Finally, it describes how to use tools for manually loading and working with drivers for hardware that is not detected and loaded properly.

Checking your hardware

When your system boots, the kernel detects your hardware and loads drivers that allow Linux to work with that hardware. Because messages about hardware detection scroll quickly off the screen when you boot, to view potential problem messages you have to redisplay those messages after the system comes up.

There are a few ways to view kernel boot messages after Linux comes up. Any user can run the `dmesg` command to see what hardware was detected and which drivers were loaded by the kernel at boot time. As new messages are generated by the kernel, those messages are also made available to the `dmesg` command.

A second way to see boot messages is the `journalctl` command to show the messages associated with a particular boot instance (as shown earlier in this chapter).

Note

After your system is running, many kernel messages are sent to the `/var/log/messages` file. So, for example, if you want to see what happens when you plug in a USB drive, you can type `tail -f /var/log/messages` and watch as devices and mount points are created. Likewise, you could use the `journalctl -f` command to follow messages as they come into the `systemd` journal.

The following is an example of some output from the `dmesg` command that was trimmed down to show some interesting information:

```
$ dmesg | less
[    0.000000] Linux version 5.0.9-301.fc30.x86_64
   (mockbuild@bkernel04.phx2.fedoraproject.org) (gcc version 9.0.1
20190312
   (Red Hat 9.0.1-0.10) (GCC)) #1 SMP Tue Apr 23 23:57:35 UTC 2019
[    0.000000] Command line:
   BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.0.9-301.fc30.x86_64
   root=/dev/mapper/fedora_localhost--live-root ro
   resume=/dev/mapper/fedora_localhost--live-swap
   rd.lvm.lv=fedora_localhost-live/root
   rd.lvm.lv=fedora_localhost-live/swap rhgb quiet
...
             S31B1102 USB DISK         1100 PQ: 0 ANSI: 0 CCS
[79.177466] sd 9:0:0:0: Attached scsi generic sg2 type 0
```

```
[79.177854] sd 9:0:0:0: [sdb]
             8343552 512-byte logical blocks: (4.27 GB/3.97 GiB)
[79.178593] sd 9:0:0:0: [sdb] Write Protect is off
```

From this output, you first see the Linux kernel version, followed by kernel command-line options. The last few lines reflect a 4GB USB drive being plugged into the computer.

If something goes wrong detecting your hardware or loading drivers, you can refer to this information to see the name and model number of hardware that's not working. Then you can search Linux forums or documentation to try to solve the problem. After your system is up and running, some other commands let you look at detailed information about your computer's hardware. The `lspci` command lists PCI buses on your computer and devices connected to them. Here's a snippet of output:

```
$ lspci
00:00.0 Host bridge: Intel Corporation
     5000X Chipset Memory ControllerHub
00:02.0 PCI bridge: Intel Corporation 5000 Series Chipset
     PCI Express x4 Port 2
00:1b.0 Audio device: Intel Corporation 631xESB/632xESB
     High Definition Audio Controller (rev 09)
00:1d.0 USB controller: Intel Corporation 631xESB/632xESB/3100
     Chipset UHCI USBController#1 (rev 09)
07:00.0 VGA compatible controller: nVidia Corporation NV44
0c:02.0 Ethernet controller: Intel Corporation 82541PI
     Gigabit Ethernet Controller (rev 05)
```

The host bridge connects the local bus to the other components on the PCI bridge. I cut down the output to show information about the different devices on the system that handle various features: sound (Audio device), flash drives and other USB devices (USB controller), the video display (VGA compatible controller), and wired network cards (Ethernet controller). If you are having trouble getting any of these devices to work, noting the model names and numbers gives you something to Google.

To get more verbose output from `lspci`, add one or more `-v` options. For example, using `lspci -vvv`, I received information about my Ethernet controller, including latency, capabilities of the controller, and the Linux driver (e1000) being used for the device.

If you are specifically interested in USB devices, try the `lsusb` command. By default, `lsusb` lists information about the computer's USB hubs along with any USB devices connected to the computer's USB ports:

```
$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 413c:2105 Dell Computer Corp.
    Model L100 Keyboard
```

```
Bus 002 Device 004: ID 413c:3012 Dell Computer Corp.
    Optical Wheel Mouse
Bus 001 Device 005: ID 090c:1000 Silicon Motion, Inc. -
    Taiwan 64MB QDI U2 DISK
```

From the preceding output, you can see the model of a keyboard, mouse, and USB flash drive connected to the computer. As with `lspci`, you can add one or more `-v` options to see more details.

To see details about your processor, run the `lscpu` command. That command gives basic information about your computer's processors.

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
...
```

From the sampling of output of `lscpu`, you can see that this is a 64-bit system (x86-64), it can operate in 32-bit or 64-bit modes, and there are four CPUs.

Managing removable hardware

Linux systems such as Red Hat Enterprise Linux, Fedora, and others, which support full GNOME desktop environments, include simple graphical tools for configuring what happens when you attach popular removable devices to the computer. So, with a GNOME desktop running, you simply plug in a USB device or insert a CD or DVD, and a window may pop up to deal with that device.

Although different desktop environments share many of the same underlying mechanisms (in particular, Udev) to detect and name removable hardware, they offer different tools for configuring how they are mounted or used. Udev (using the `udevd` daemon) creates and removes devices (`/dev` directory) as hardware is added and removed from the computer. Settings that are of interest to someone using a desktop Linux system, however, can be configured with easy-to-use desktop tools.

The Nautilus file manager used with the GNOME desktop lets you define what happens when you attach removable devices or insert removable media into the computer from the File Management Preferences window. The descriptions in this section are based on GNOME 3.32 in Fedora 30.

From the GNOME 3.32 desktop, select Activities and type **Removable Media**. Then select the Removable Media Setting entry.

The following settings are available from the Removable Media window. These settings relate to how removable media are handled when they are inserted or plugged in. In most cases, you are prompted about how to handle a medium that is inserted or connected.

CD audio: When an audio CD is inserted, you can choose to be prompted for what to do (default), do nothing, open the contents in a folder window, or select from various audio CD players to be launched to play the content. Rhythmbox (music player), Audio CD Extractor (CD burner), and Brasero (CD burner) are among the choices that you have for handling an inserted audio CD.

DVD video: When a commercial video DVD is inserted, you are prompted for what to do with that DVD. You can change that default to launch Totem (videos), Brasero (DVD burner), or another media player you have installed (such as MPlayer).

Music player: When inserted media contains audio files, you are asked what to do. You can select to have Rhythmbox or some other music player begin playing the files by selecting that player from this box.

Photos: When inserted media (such as a memory card from a digital camera) contains digital images, you are asked what to do with those images. You can select to do nothing, or you can select to have the images opened in the Shotwell image viewer (the default application for viewing images on the GNOME desktop) or another installed photo manager.

Software: When inserted media contains installable software, the Software window opens by default. To change that behavior (to ask what to do, do nothing, or open the media contents in a folder), you can select the box for those choices.

Other Media: Select the Type box under the Other Media heading to select how less commonly used media are handled. For example, you can select what actions are taken to handle audio DVDs or blank Blu-ray discs, CDs, or DVDs. You can select what applications to launch for Blu-ray video disc, ebook readers, and Picture CDs.

Note that the settings described here are in effect only for the user who is currently logged in. If multiple users have login accounts, each can have their own way of handling removable media.

NOTE

The Totem movie player does not play movie DVDs unless you add extra software to decrypt the DVD. You should look into legal issues and other movie player options if you want to play commercial DVD movies from Linux.

The options to connect regular USB flash drives or hard drives are not listed on this window. If you connect one of those drives to your computer, however, devices are automatically created when you plug them in (named `/dev/sda`, `/dev/sdb`, and so on). Any filesystems found on those devices are automatically mounted on `/run/media/`*`username`*, and you are prompted if you want to open a Nautilus window to view files on those devices. This is done automatically, so you don't have to do any special configuration to make this happen.

When you are finished with a USB drive, right-click the device's name in the Nautilus file manager window and select Safely Remove Drive. This action unmounts the drive and

removes the mount point in the `/run/media/username` directory. After that, you can safely unplug the USB drive from your computer.

Working with loadable modules

If you have added hardware to your computer that isn't properly detected, you might need to load a module manually for that hardware. Linux comes with a set of commands for loading, unloading, and getting information about hardware modules.

Kernel modules are installed in `/lib/modules/` subdirectories. The name of each subdirectory is based on the release number of the kernel. For example, if the kernel were `5.3.8-200.fc30.x86_64`, the `/lib/modules/5.3.8-200.fc30.x86_64` directory would contain drivers for that kernel. Modules in those directories can then be loaded and unloaded as they are needed.

Commands for listing, loading, unloading, and getting information about modules are available with Linux. The following sections describe how to use those commands.

Listing loaded modules

To see which modules are currently loaded into the running kernel on your computer, use the `lsmod` command. Consider the following example:

```
# lsmod
Module                  Size  Used by
vfat                   17411  1
fat                    65059  1 vfat
uas                    23208  0
usb_storage            65065  2 uas
fuse                   91446  3
ipt_MASQUERADE         12880  3
xt_CHECKSUM            12549  1
nfsv3                  39043  1
rpcsec_gss_krb5        31477  0
nfsv4                 466956  0
dns_resolver           13096  1 nfsv4
nfs                   233966  3 nfsv3,nfsv4
.
.
.
i2c_algo_bit           13257  1 nouveau
drm_kms_helper         58041  1 nouveau
ttm                    80772  1 nouveau
drm                   291361  7 ttm,drm_kms_helper,nouveau
ata_generic            12923  0
pata_acpi              13053  0
e1000                 137260  0
i2c_core               55486  5 drm,i2c_i801,drm_kms_helper
```

This output shows a variety of modules that have been loaded on a Linux system, including one for a network interface card (e1000).

To find information about any of the loaded modules, use the modinfo command. For example, you can enter the following:

```
# /sbin/modinfo -d e1000
Intel(R) PRO/1000 Network Driver
```

Not all modules have descriptions available and, if nothing is available, no data are returned. In this case, however, the e1000 module is described as an Intel(R) PRO/1000 Network Driver module. You can also use the -a option to see the author of the module or -n to see the object file representing the module. The author information often has the email address of the driver's creator, so you can contact the author if you have problems or questions about it.

Loading modules

You can load any module (as root user) that has been compiled and installed (to a /lib/modules subdirectory) into your running kernel using the modprobe command. A common reason for loading a module is to use a feature temporarily (such as loading a module to support a special filesystem on some removable media you want to access). Another reason to load a module is to identify that module as one that will be used by a particular piece of hardware that could not be autodetected.

Here is an example of the modprobe command being used to load the parport module, which provides the core functions to share parallel ports with multiple devices:

```
# modprobe parport
```

After parport is loaded, you can load the parport_pc module to define the PC-style ports available through the interface. The parport_pc module lets you optionally define the addresses and IRQ numbers associated with each device sharing the parallel port, as in the following example:

```
# modprobe parport_pc io=0x3bc irq=auto
```

In this example, a device is identified as having an address of 0x3bc, and the IRQ for the device is autodetected.

The modprobe command loads modules temporarily—they disappear at the next reboot. To add the module to your system permanently, add the modprobe command line to one of the startup scripts run at boot time.

Removing modules

Use the rmmod command to remove a module from a running kernel. For example, to remove the module parport _ pc from the current kernel, type the following:

```
# rmmod parport_pc
```

If it is not currently busy, the `parport_pc` module is removed from the running kernel. If it is busy, try killing any process that might be using the device. Then run `rmmod` again. Sometimes, the module you are trying to remove depends on other modules that may be loaded. For instance, the `usbcore` module cannot be unloaded because it is a built-in module:

```
# rmmod usbcore
rmmod: ERROR: Module usbcore is builtin.
```

Instead of using `rmmod` to remove modules, you could use the `modprobe -r` command. With `modprobe -r`, instead of just removing the module you request, you can also remove dependent modules that are not being used by other modules.

Summary

Many features of Linux, especially those that can potentially damage the system or impact other users, require that you gain root privilege. This chapter describes different ways of obtaining root privilege: direct login, `su` command, or `sudo` command. It also covers some of the key responsibilities of a system administrator and components (configuration files, browser-based tools, and so on) that are critical to a system administrator's work.

The next chapter describes how to install a Linux system. Approaches to installing Linux that are covered in that chapter include how to install from live media and from installation media.

Exercises

Use these exercises to test your knowledge of system administration and to explore information about your system hardware. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. From a shell as root user (or using `sudo`), enable Cockpit (`cockpit.socket`) using the `systemctl` command.
2. Open your web browser to the Cockpit interface (9090) on your system.
3. Find all files under the `/var/spool` directory that are owned by users other than root and display a long listing of them.
4. Become the root user using the `su -` command. To prove that you have root privilege, create an empty or plain-text file named `/mnt/test.txt`. Exit the shell when you are finished. If you are using Ubuntu, you must set your root password first (`sudo passwd root`).

5. Log in as a regular user and become root using `su -`. Edit the `/etc/sudoers` file to allow your regular user account to have full root privilege via the `sudo` command.
6. As the user to whom you just gave `sudoers` privilege, use the `sudo` command to create a file called `/mnt/test2.txt`. Verify that the file is there and owned by the root user.
7. Run the `journalctl -f` command and plug a USB drive into a USB port on your computer. If it doesn't mount automatically, mount it on `/mnt/test`. In a second terminal, unmount the device and remove it, continuing to watch the output from `journalctl -f`.
8. Run a command to see what USB devices are connected to your computer.
9. Pretend that you added a TV card to your computer, but the module needed to use it (`bttv`) was not properly detected and loaded. Load the `bttv` module yourself, and then look to see that it was loaded. Were other modules loaded with it?
10. Remove the `bttv` module along with any other modules that were loaded with it. List your modules to make sure that this was done.

CHAPTER 9

Installing Linux

IN THIS CHAPTER

Choosing an installation method

Installing a single- or multi-boot system

Performing a Live media installation of Fedora

Installing Red Hat Enterprise Linux

Understanding cloud-based installations

Partitioning the disk for installation

Understanding the GRUB boot loader

Installing Linux has become a fairly easy thing to do—if you are starting with a computer that is up to spec (hard disk, RAM, CPU, and so on) and you don't mind totally erasing your hard drive. With cloud computing and virtualization, installation can be even simpler. It allows you to bypass traditional installation and spin a Linux system up or down within a few minutes by adding metadata to prebuilt images.

This chapter starts off with a simple installation on a physical computer from Live media and progresses to more complex installation topics.

To ease you into the subject of installing Linux, I cover three different ways of installing Linux and step you through each process:

Installing from Live media A Linux Live media ISO is a single, read-only image that contains everything you need to start a Linux operating system. That image can be burned to a DVD or USB drive and booted from that medium. With the Live media, you can totally ignore your computer's hard disk; in fact, you can run Live media on a system with no hard disk. After you are running the Live Linux system, some Live media ISOs allow you to launch an application that permanently installs the contents of the Live medium to your hard disk. The first installation procedure in this chapter shows you how to install Linux permanently from a Fedora Live media ISO.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Installing from an installation DVD An installation DVD, available with Fedora, RHEL, Ubuntu, and other Linux distributions, offers more flexible ways of installing Linux. In particular, instead of just copying the whole Live media contents to your computer, with an installation DVD you can choose exactly which software package you want. The second installation procedure I show in this chapter steps you through an installation process from a Red Hat Enterprise Linux 8 installation DVD.

Installing in the enterprise Sitting in front of a computer and clicking through installation questions isn't inconvenient if you are installing a single system. But what if you need to install dozens or hundreds of Linux systems? What if you want to install those systems in particular ways that need to be repeated over multiple installations? Later in this chapter, I describe efficient ways of installing multiple Linux systems using network installation features and kickstart files.

A fourth method of installation not covered in this chapter is to install Linux to a cloud environment (such as Amazon Web Services) or virtual machine on a virtualization host, such as Virtual Box or a VMware system. Chapter 27 and Chapter 28 describe ways of installing or deploying a virtual machine on a Linux KVM host or in a cloud environment.

To try the procedures in this chapter along with me, you should have a computer in front of you that you don't mind totally erasing. As an alternative, you can use a computer that has another operating system installed (such as Windows), as long as there is enough unused disk space available outside of that operating system. I describe the procedure, and risk of data loss, if you decide to set up one of these "dual boot" (Linux and Windows) arrangements.

Choosing a Computer

You can get a Linux distribution that runs on handheld devices or an old PC in your closet with as little as 24MB of RAM and a 486 processor. To have a good desktop PC experience with Linux, however, you should consider what you want to be able to do with Linux when you are choosing your computer.

Be sure to consider the basic specifications that you need for a PC-type computer to run the Fedora and Red Hat Enterprise Linux distributions. Because Fedora is used as the basis for Red Hat Enterprise Linux releases, hardware requirements are similar for basic desktop and server hardware for those two distributions.

Processor A 1GHz Pentium processor is the minimum for a GUI installation. For most applications, a 32-bit processor is fine (x86). However, if you want to set up the system to do virtualization, you need a 64-bit processor (x86_64).

Note

If you have a less powerful computer than the minimum described here, consider using a lightweight Linux distribution. Lightweight Ubuntu distributions include Peppermint OS (`https://peppermintos.com/`) and Lubuntu (`https://lubuntu.net/`). For a lightweight Fedora-based distribution, try the LXDE desktop (`https://spins.fedoraproject.org/lxde/`). For a Linux distribution requiring the least resources, you could try Tiny Core Linux (`http://tinycorelinux.net/`).

RAM Fedora recommends at least 1GB of RAM, but at least 2GB or 3GB would be much better. On my RHEL desktop, I'm running a web browser, word processor, and mail reader, and I'm consuming over 2GB of RAM.

DVD or USB drive You need to be able to boot up the installation process from a DVD or USB drive. In recent releases, the Fedora live media ISO has become too big to fit on a CD, so you need to burn it to a DVD or USB drive. If you can't boot from a DVD or USB drive, there are ways to start the installation from a hard disk or by using a PXE install. After the installation process is started, more software can sometimes be retrieved from different locations (over the network or from hard disk, for example).

Note

PXE (pronounced pixie) stands for *Preboot eXecution Environment (PXE)*. You can boot a client computer from a Network Interface Card (NIC) that is PXE-enabled. If a PXE boot server is available on the network, it can provide everything a client computer needs to boot. What it boots can be an installer. So, with a PXE boot, it is possible to do a complete Linux installation without a CD, DVD, or any other physical medium.

Network card You need wired or wireless networking hardware to be able to add more software or get software updates. Fedora offers free software repositories if you can connect to the Internet. For RHEL, updates are available as part of the subscription price.

Disk space Fedora recommends at least 20GB of disk space for an average desktop installation, although installations can range (depending on which packages you choose to install) from 600MB (for a minimal server with no GUI install) to 7GB (to install all packages from the installation DVD). Consider the amount of data that you need to store. Although documents can consume very little space, videos can consume massive amounts of space. (By comparison, you can install Tiny Core Linux to disk with only about 16MB of disk space, which includes a GUI.)

Special hardware features Some Linux features require special hardware features. For example, to use Fedora or RHEL as a virtualization host using KVM, the computer must have a processor that supports virtualization. These include AMD-V or Intel-VT chips.

If you're not sure about your computer hardware, there are a few ways to check what you have. If you are running Windows, the System Properties window can show you the processor you have as well as the amount of RAM that's installed. As an alternative, with the Fedora Live CD booted, open a shell and type **`dmesg | less`** to see a listing of hardware as it is detected on your system.

With your hardware in place, you can choose to install Linux from a Live CD or from installation media, as described in the following sections.

Installing Fedora from Live Media

In Chapter 2, you learned how to get and boot up Linux Live media. This chapter steps you through an installation process of a Fedora Live DVD so that it is permanently installed on your hard disk.

Simplicity is the main advantage of installing from Live media. Essentially, you are just copying the kernel, applications, and settings from the ISO image to the hard disk. There are fewer decisions that you have to make to do this kind of installation, but you also don't get to choose exactly which software packages to install. After the installation, you can add and remove packages as you please.

The first decisions that you must make about your Live media installation include where you want to install the system and whether you want to keep existing operating systems around when your installation is done:

Single-boot computer The easiest way to install Linux is to not have to worry about other operating systems or data on the computer and have Linux replace everything. When you are done, the computer boots up directly to Fedora.

Multi-boot computer If you already have Windows installed on a computer and you don't want to erase it, you can install Fedora along with Windows on that system. Then, at boot time, you can choose which operating system to start up. To be able to install Fedora on a system with another operating system installed, you must have either extra disk space available (outside the Windows partition) or be able to shrink the Windows system to gain enough free space to install Fedora. Because multi-boot computers are tedious to set up and risk damaging your installed system, I recommend installing Linux on a separate computer, even an old used one, or on a virtual machine, as opposed to multi-booting.

Bare metal or virtual system The resulting Fedora installation can be installed to boot up directly from the computer hardware or from within an existing operating system on the computer. If you have a computer that is running as a virtual host, you can install Fedora on that system as a virtual guest. Virtualization host software includes KVM, Xen, and VirtualBox (for Linux and UNIX systems as well as Windows and the Mac OS), Hyper-V (for Microsoft systems), and VMware (for Linux, Windows, and Mac OS). You can use the Fedora Live ISO image from disk or burned to a DVD to start an installation from your chosen hypervisor host. (Chapter 27, "Using Linux for Cloud Computing," describes how to set up a KVM virtualization host.)

The following procedure steps you through the process of installing the Fedora Live ISO described in Chapter 2 to your local computer. Because the Fedora 30 installation is very similar to the Red Hat Enterprise Linux 8 installation described later in this chapter, you can refer to that procedure if you want to go beyond the simple selections shown here (particularly in the area of storage configuration).

Caution

Before beginning the procedure, be sure to make backup copies of any data on the computer that you still want to keep. Although, you can choose not to erase selected disk partitions (as long as there is enough space available on other partitions), there is always a risk that data can be lost when you are manipulating disk partitions. Also, unplug any USB drives that you have plugged into your computer because they could be overwritten.

1. Get Fedora. Choose the Fedora Live media image that you want to use, download it to your local system, and burn it to an appropriate medium. See Appendix A for information on how to get the Fedora Live media and burn it to a DVD or USB drive.
2. Boot the Live image. Insert the DVD or USB drive. When the BIOS screen appears, look for a message that tells you to press a particular function key (such as F12) to interrupt the boot process and select the boot medium. Select the DVD or USB drive, depending on which you have, and Fedora should come up and display the boot screen. When you see the boot screen, select Start Fedora-Workstation-Live.
3. Start the installation. When the Welcome to Fedora screen appears, position your mouse over the Install to Hard Drive area and select it. Figure 9.1 shows an example of the Install to Hard Drive selection on the Fedora Live media.

FIGURE 9.1

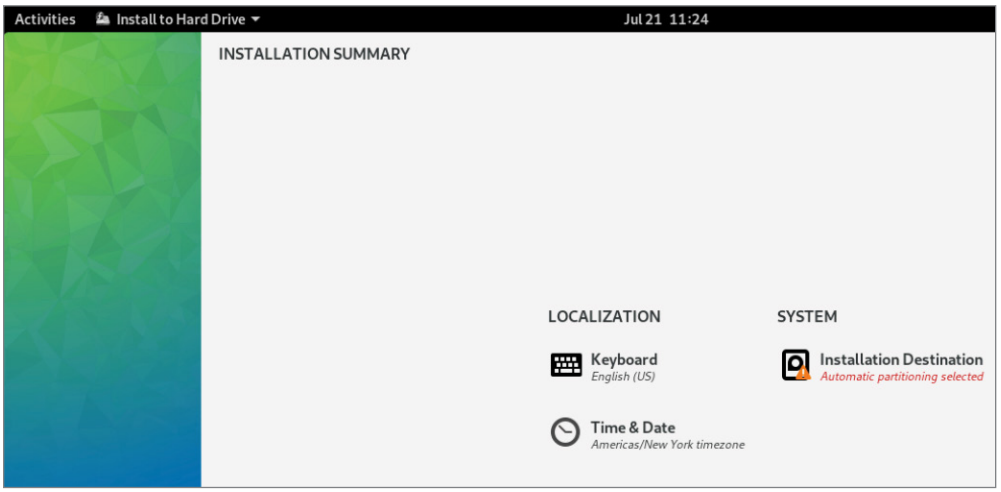
Start the installation process from Live media.



4. Select the language. When prompted, choose the language type that best suits you (such as U.S. English) and select Next. You should see the Installation summary screen, as shown in Figure 9.2.

FIGURE 9.2

Select configuration options from the Installation Summary screen.



5. Select Time & Date. From the Time & Date screen, you can select your time zone either by clicking the map or choosing the region and city from drop-down boxes. To set the date and time, if you have an Internet connection, you can select the Network Time button to turn it on, or you can select OFF and set the date and time manually from boxes on the bottom of the screen. Select Done in the upper-right corner when you are finished.
6. Select the installation destination. Available storage devices (such as your hard drive) are displayed, with your hard drive selected as the installation destination. If you want the installer to install Fedora automatically, reclaiming existing disk space, make sure that your disk is selected (not a USB drive or other device connected to your computer), then make the following selections:
   a. Automatic . . . If there is enough available disk space on the selected disk drive, you can continue with the installation by selecting Continue. Otherwise, you need to reclaim disk space as follows:

      I would like to make additional space available. . . If you want to erase the hard drive completely, select this check box and click Continue. You can erase some or all of the partitions that currently contain data.

   b. Reclaim Disk Space. From this screen, you can select Delete All. Then select Reclaim Space. Partitioning is set up automatically and you are returned to the Installation Summary screen.

7. Select the keyboard. You can just use the default English (U.S.) keyboard or select Keyboard to choose a different keyboard layout.
8. Begin installation. Select Begin Installation to begin installing to hard disk.
9. Finish the configuration. When the first part of the installation is complete, click Quit.
10. Reboot. Select the little on/off button from the menu on the top-right corner of the screen. When prompted, click the Restart button. Eject or remove the Live media when the system boot screen appears. The computer should boot to your newly installed Fedora system. (You may actually need to power off the computer for it to boot back up.)
11. Begin using Fedora. A first boot screen appears at this point, allowing you to create a user account and password, among other things. You are automatically logged in as that user account when configuration is done. That account has `sudo` privileges, so you can immediately begin doing administrative tasks as needed.
12. Get software updates. To keep your system secure and up to date, one of the first tasks that you should do after installing Fedora is to get the latest versions of the software you just installed. If your computer has an Internet connection (plugging into a wired Ethernet network or selecting an accessible wireless network from the desktop takes care of that), you can simply open a Terminal as your new user and type **`sudo dnf update`** to download and update all of your packages from the Internet. If a new kernel is installed, you can reboot your computer to have that new kernel take effect.

At this point, you can begin using the desktop, as described in Chapter 2. You can also use the system to perform exercises from any of the chapters in this book.

Installing Red Hat Enterprise Linux from Installation Media

In addition to offering a live DVD, most Linux distributions offer a single image or set of images that can be used to install the distribution. For this type of installation media, instead of copying the entire contents of the medium to disk, software is split up into packages that you can select to meet your exact needs. A full installation DVD, for example, can allow you to install anything from a minimal system to a fully featured desktop to a full-blown server that offers multiple services.

In this chapter, I use a Red Hat Enterprise Linux 8 installation DVD as the installation medium. Review the hardware information and descriptions of dual booting in the previous section before beginning your RHEL installation.

Follow this procedure to install Red Hat Enterprise Linux from an installation DVD.

1. Get the installation media. The process of downloading RHEL install ISO images is described on the Red Hat Enterprise Linux product page. If you are not yet a Red

Hat customer, you can apply for an evaluation copy here: `https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux`.

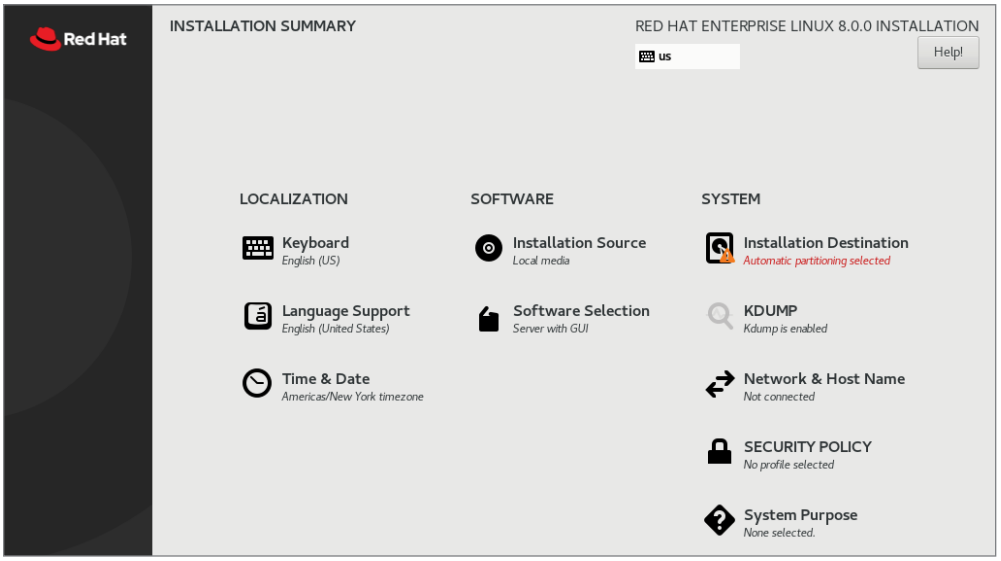
This requires that you create a Red Hat account. If that is not possible, you can download an installation DVD from a mirror site of the CentOS project to get a similar experience: `https://wiki.centos.org/Download`.

For this example, I used the 6.7G RHEL 8 DVD ISO `rhel-8.0-x86_64-dvd.iso`. After you have the DVD ISO, you can burn it to a physical USB drive or dual-layer DVD, as described in Appendix A.

2. Boot the installation media. Insert the USB drive or DVD into your computer and reboot. (If you need to, interrupt the boot prompt to select to boot from the selected USB or DVD.) The Welcome screen appears.
3. Select Install or Test Media. Select the Install or the "Test this media & install" entry to do a new installation of RHEL. The media test verifies that the DVD has not been corrupted during the copy or burning process. If you need to modify the installation process, you can add boot options by pressing the Tab key with a boot entry highlighted and typing in the options you want. See the section "Using installation boot options" later in this chapter.
4. Select a language. Select your language and choose Continue. The Installation Summary screen appears. From that screen, you can select to change any of the available Localization, Software, and System features, as shown in Figure 9.3.

FIGURE 9.3

Choose from Localization, Software, and System topics on the Installation Summary screen.



5. Keyboard. Choose from different types of keyboards available with the languages you selected earlier. Type some text to see how the keys are laid out.
6. Language Support. You have a chance to add support for additional languages (beyond what you set by default earlier). Select Done when you are finished.
7. Time & Date. Choose a time zone for your machine from either the map or the list shown (as described in the section "Installing Fedora from Live Media"). Either set the time manually with up/down arrows or select Network Time to have your system try to connect to networked time servers automatically to sync system time. Select Done when you are finished.
8. Installation Source. The installation DVD is used, by default, to provide the RPM packages that are used during installation. You have the option of selecting "On the network" and choosing a Web URL (`http`, `https`, or `ftp`) identifying where the Red Hat Enterprise Linux software repository is located. After choosing the DVD or a network location, you can add additional `yum` repositories to have those repositories used during installation as well. Select Done when you are finished.
9. Software Selection. The default "Server with GUI" selection provides a GNOME 3 desktop system on top of a basic server install. Other choices include "Server" (which has no GUI), "Minimal Install" (which starts with a basic package set), and "Workstation" (geared for end users). You can select to add other services or other base environments to include. Select Done when you are ready to continue.
10. Installation Destination. The new RHEL system is installed, by default, on the local hard drive using automatic partitioning. You also have the option of attaching network storage or special storage, such as Firmware RAID. (See the section "Partitioning hard drives" later in this chapter for details on configuring storage.) Click Done when you are finished. You may be asked to verify that it's okay to delete existing storage.
11. Kdump. Enabling `kdump` sets aside RAM to be used to capture the resulting kernel dump in the event that your kernel crashes. Without `kdump`, there would be no way to diagnose a crashed kernel. By default, enabling kdump sets aside 160MB plus 2 bits for every 4KB of RAM for saving kernel crashes.
12. Network & Host Name. Any network interface cards that are discovered can be configured at this point. If a DHCP service is available on the network, network address information is assigned to the interface after you select ON. Select Configure if you prefer to configure the network interface manually. Fill in the Hostname box if you want to set the system's hostname. Setting up your network and hostname during installation can make it easier to begin using your system after installation. Click Done to continue.
13. Security Policy. By choosing a security policy (none is chosen by default), you can ensure that your system complies with a selected security standard. All fields are optional and can be changed later.
14. System Purpose. This optional selection lets you choose the system's role, service-level agreement, and usage.

15. Begin the installation. Click the Begin Installation button to start the install process. A progress bar marks the progress of the installation. As the system is installing, you can set the root password and create a new user account for your new system.
16. Root Password. Set the password for the root user and verify it (type it again). Click Done to accept it. If the password is too short or too weak, you stay on the page (where you can set a new password). If you decide to keep the weak password instead, click Done again to accept the weak password.
17. User Creation. It is good practice to log into a Linux system with a non-root user account and request root privilege as needed. You can set up a user account, including a username, full name, and password. You can select "Make this user administrator" to give that user `sudo` privileges (allowing the account to act as the root user as needed). Select Done when you are finished. If the password you enter is too short or otherwise weak, you must change it or click Done again if you still want to use the weak password.
18. Complete the installation. When installation is finished, click Reboot. Pop out the DVD when the system restarts and Red Hat Enterprise Linux starts up from the hard disk.
19. Run firstboot. If you installed a desktop interface, the firstboot screen appears the first time you boot the system. Here's what you do:
    a. License Information. Read and click the check box to accept the license information, then click Done.
    b. Subscription Manager. When prompted, you can leave the default subscription management system in place (`subscription.rhn.redhat.com`) or enter the location of a Red Hat Satellite server to register your system. Click Next. Enter your Red Hat account and password, then click Register to register and entitle your system to updates. If the subscription found is acceptable, click Attach to enable the subscription.
20. Select Finish Configuration when you are done.

You should now be able to log in to your Red Hat Enterprise Linux system. One of the first things that you should do is to get software updates for the new system. Do this by logging into the system and running **`sudo dnf upgrade`** from a Terminal window.

Understanding Cloud-Based Installations

When you install a Linux system on a physical computer, the installer can see the computer's hard drive, network interfaces, CPUs, and other hardware components. When you install Linux in a cloud environment, those physical components are abstracted into a pool of resources. So, to install a Linux distribution in an Amazon EC2, Google Compute Engine, or OpenStack cloud platform, you need to go about things differently.

The common way of installing Linux in a cloud is to start with a file that is an image of an installed Linux system. Typically, that image includes all of the files needed by a basic, running Linux system. Metadata is added to that image from a configuration file or by filling out a form from a cloud controller that creates and launches the operating system as a virtual machine.

The kind of information added to the image might include a particular hostname, root password, and new user account. You might also want to choose to have a specific amount of disk space, a particular network configuration, and a certain number of CPU processors and RAM.

Methods for installing Linux in a local cloud-like KVM environment are discussed in Chapter 28, "Deploying Linux to the Cloud." That chapter covers how to run a Linux system as a virtual machine image on a KVM environment, Amazon EC2 cloud, or OpenStack environment.

Installing Linux in the Enterprise

If you were managing dozens, hundreds, even thousands of Linux systems in a large enterprise, it would be terribly inefficient to have to go to each computer to type and click through each installation. Fortunately, with Red Hat Enterprise Linux and other distributions, you can automate installation in such a way that all you need to do is to turn on a computer and boot from the computer's network interface card to get your desired Linux installation.

Although we have focused on installing Linux from a DVD or USB media, there are many other ways to launch a Linux installation and many ways to complete an installation. The following descriptions step through the installation process and describe ways of changing that process along the way:

Launch the installation medium. You can launch an installation from any medium that you can boot from a computer: CD, DVD, USB drive, hard disk, or network interface card with PXE support. The computer goes through its boot order and looks at the master boot record on the physical medium or looks for a PXE server on the network.

Start the anaconda kernel. The job of the boot loader is to point to the special kernel (and possibly an initial RAM disk) that starts the Linux installer (called anaconda). So, any of the media types just described simply needs to point to the location of the kernel and initial RAM disk to start the installation. If the software packages are not on the same medium, the installation process prompts you for where to get those packages.

Add kickstart or other boot options. Boot options (described later in this chapter) can be passed to the anaconda kernel to configure how it starts up. One option supported by Fedora and RHEL allows you to pass the location of a kickstart file to the installer. That kickstart can contain all of the information needed to complete the installation: root password, partitioning, time zone, and so on to configure the installed system further. After the installer starts, it either prompts for needed information or uses the answers provided in the kickstart file.

Find software packages. Software packages don't have to be on the installation medium. This allows you to launch an installation from a boot medium that contains only a kernel and initial RAM disk. From the kickstart file or from an option you enter manually to the installer, you can identify the location of the repository holding the RPM software packages. That location can be a local CD (`cdrom`), website (`http`), FTP site (`ftp`), NFS share (`nfs`), NFS ISO (`nfsiso`), or local disk (`hd`).

Modify installation with kickstart scripts. Scripts included in a kickstart can run commands you choose before or after the installation to further configure the Linux system. Those commands can add users, change permissions, create files and directories, grab files over the network, or otherwise configure the installed system exactly as you specify.

Although installing Linux in enterprise environments is beyond the scope of this book, I want you to understand the technologies that are available when you want to automate the Linux installation process. Here are some of those technologies available to use with Red Hat Enterprise Linux, along with links to where you can find more information about them:

Install server If you set up an installation server, you don't have to carry the software packages around to each machine where you install RHEL. Essentially, you copy all of the software packages from the RHEL installation medium to a web server (`http`), FTP server (`ftp`), or NFS server (`nfs`) and then point to the location of that server when you boot the installer. The RHEL 8 Installation Guide describes how to set up a local or network installation source:

`https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/performing_a_standard_rhel_installation/index#prepare-installation-source_preparing-for-your-installation`

PXE server If you have a computer with a network interface card that supports PXE booting (as most do), you can set your computer's BIOS to boot from that NIC. If you have set up a PXE server on that network, that server can present a menu to the computer containing entries to launch an installation process. The RHEL Installation Guide provides information on how to set up PXE servers for installation:

`https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/performing_a_standard_rhel_installation/index#booting-the-installation-using-pxe_booting-the-installer`

Kickstart files To automate an installation completely, you create what is called a *kickstart file*. By passing a kickstart file as a boot option to a Linux installer, you can provide answers to all of the installation questions that you would normally have to click through.

When you install RHEL, a kickstart file containing answers to all installation questions for the installation you just did is contained in the `/root/anaconda-ks.cfg` file. You can present that file to your next installation to repeat the installation configuration or use that file as a model for different installations.

See the Advanced RHEL Installation Guide for information on performing a kickstart installation: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/performing_an_advanced_rhel_installation/index/#performing_an_automated_installation_using_kickstart
. . . and creating your own kickstart files https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/performing_an_advanced_rhel_installation/index/#creating-kickstart-files_installing-rhel-as-an-experienced-user

Exploring Common Installation Topics

Some of the installation topics touched upon earlier in this chapter require further explanation for you to be able to implement them fully. Read through the following sections to get a greater understanding of specific installation topics.

Upgrading or installing from scratch

If you have an earlier version of Linux already installed on your computer, Fedora, Ubuntu, and other Linux distributions offer an upgrade option. Red Hat Enterprise Linux offers a limited upgrade path from RHEL 7 to RHEL 8.

Upgrading lets you move a Linux system from one major release to the next. Between minor releases, you can simply update packages as needed (for example, by typing `yum update`). Here are a few general rules before performing an upgrade:

Remove extra packages. If you have software packages that you don't need, remove them before you do an upgrade. Upgrade processes typically upgrade only those packages that are on your system. Upgrades generally do more checking and comparing than clean installs do, so any package that you can remove saves time during the upgrade process.

Check configuration files. A Linux upgrade procedure often leaves copies of old configuration files. You should check that the new configuration files still work for you.

Tip

Installing Linux from scratch goes faster than an upgrade. It also results in a cleaner Linux system. So, if you don't need the data on your system (or if you have a backup of your data), I recommend that you do a fresh installation. Then you can restore your data to a freshly installed system.

Some Linux distributions, most notably Gentoo, have taken the approach of providing ongoing updates. Instead of taking a new release every few months, you simply continuously grab updated packages as they become available and install them on your system.

Dual booting

It is possible to have multiple operating systems installed on the same computer. One way to do this is by having multiple partitions on a hard disk and/or multiple hard disks and then installing different operating systems on different partitions. As long as the boot loader contains boot information for each of the installed operating systems, you can choose which one to run at boot time.

CAUTION

Although tools for resizing Windows partitions and setting up multi-boot systems have improved in recent years, there is still some risk of losing data on Windows/Linux dual-boot systems. Different operating systems often have different views of partition tables and master boot records that can cause your machine to become unbootable (at least temporarily) or lose data permanently. Always back up your data before you try to resize a Windows filesystem to make space for Linux.

If the computer you are using already has a Windows system on it, quite possibly the entire hard disk is devoted to Windows. Although you can run a bootable Linux, such as KNOPPIX or Tiny Core Linux, without touching the hard disk, to do a more permanent installation, you'll want to find disk space outside of the Windows installation. There are a few ways to do this:

Add a hard disk. Instead of messing with your Windows partition, you can simply add a hard disk and devote it to Linux.

Resize your Windows partition. If you have available space on a Windows partition, you can shrink that partition so that free space is available on the disk to devote to Linux. Commercial tools such as Acronis Disk Director (`https://www.acronis.com/en-us/personal/disk-manager`) are available to resize your disk partitions and set up a workable boot manager. Some Linux distributions (particularly bootable Linux distributions used as rescue media) include a tool called GParted (which includes software from the Linux-NTFS project for resizing Windows NTFS partitions).

NOTE

Type `dnf install gparted` (in Fedora) or `apt-get install gparted` (in Ubuntu) to install GParted. Run `gparted` as root to start it.

Before you try to resize your Windows partition, you might need to defragment it. To defragment your disk on some Windows systems so that all your used space is put in order on the disk, open My Computer, right-click your hard disk icon (typically C:), select Properties, click Tools, and select Defragment Now.

Defragmenting your disk can be a fairly long process. The result of defragmentation is that all of the data on your disk are contiguous, creating lots of contiguous free space at the end of the partition. Sometimes, you have to complete the following special tasks to make this true:

- If the Windows swap file is not moved during defragmentation, you must remove it. Then, after you defragment your disk again and resize it, you need to restore the swap file. To remove the swap file, open the Control Panel, open the System icon, click the Performance tab, and select Virtual Memory. To disable the swap file, click Disable Virtual Memory.
- If your DOS partition has hidden files that are on the space you are trying to free up, you need to find them. In some cases, you can't delete them. In other cases, such as swap files created by a program, you can safely delete those files. This is a bit tricky because some files should not be deleted, such as DOS system files. You can use the `attrib -s -h` command from the root directory to deal with hidden files.

After your disk is defragmented, you can use commercial tools described earlier (Acronis Disk Director) to repartition your hard disk to make space for Linux. Or, you can use the open-source alternative GParted.

After you have cleared enough disk space to install Linux (see the disk space requirements described earlier in this chapter), you can install Ubuntu, Fedora, RHEL, or another Linux distribution. As you set up your boot loader during installation, you can identify Windows, Linux, and any other bootable partitions so that you can select which one to boot when you start your computer.

Installing Linux to run virtually

Using virtualization technology, such as KVM, VMware, VirtualBox, or Xen, you can configure your computer to run multiple operating systems simultaneously. Typically, you have a host operating system running (such as your Linux or Windows desktop), and then you configure guest operating systems to run within that environment.

If you have a Windows system, you can use commercial VMware products to run Linux on your Windows desktop. Get a trial of VMware Workstation (`https://www.vmware.com/try-vmware`) to see if you like it. Then run your installed virtual guests with the free VMware Player. With a full-blown version of VMware Workstation, you can run multiple distributions at the same time.

Open-source virtualization products that are available with Linux systems include VirtualBox (`https://www.virtualbox.org`), Xen (`https://xenproject.org`), and KVM (`https://www.linux-kvm.org`). Some Linux distributions still use Xen. However, all Red Hat systems currently use KVM as the basis for Red Hat's hypervisor features in RHEL, Red Hat Virtualization, and other cloud projects. See Chapter 28 for information on installing Linux as a virtual machine on a Linux KVM host.

Using installation boot options

When the anaconda kernel launches at boot time for RHEL or Fedora, boot options provided on the kernel command line modify the behavior of the installation process. By interrupting the boot loader before the installation kernel boots, you can add your own boot options to direct how the installation behaves.

When you see the installation boot screen, depending on the boot loader, press Tab or some other key to be able to edit the anaconda kernel command line. The line identifying the kernel might look something like the following:

```
vmlinuz initrd=initrd.img ...
```

The `vmlinuz` is the compressed kernel and `initrd.img` is the initial RAM disk (containing modules and other tools needed to start the installer). To add more options, just type them at the end of that line and press Enter.

So, for example, if you have a kickstart file available from `/root/ks.cfg` on a CD, your anaconda boot prompt to start the installation using the kickstart file could look like the following:

```
vmlinuz initrd=initrd.img ks=cdrom:/root/ks.cfg
```

For Red Hat Enterprise Linux 8 and the latest Fedora releases, kernel boot options used during installation are transitioning to a new naming method. With this new naming, a prefix of `inst.` can be placed in front of any of the boot options shown in this section that are specific to the installation process (for example, `inst.xdriver` or `inst.repo=dvd`). For the time being, however, you can still use the options shown in the next few sections with the `inst.` prefix.

Boot options for disabling features

Sometimes, a Linux installation fails because the computer has some non-functioning or non-supported hardware. Often, you can get around those issues by passing options to the installer that do such things as disable selected hardware when you need to select your own driver. Table 9.1 provides some examples.

Boot options for video problems

If you are having trouble with your video display, you can specify video settings as noted in Table 9.2.

Boot options for special installation types

By default, installation runs in graphical mode when you're sitting at the console answering questions. If you have a text-only console, or if the GUI isn't working properly, you can run an installation in plain-text mode: by typing **`text`**, you cause the installation to run in text mode.

TABLE 9.1 Boot Options for Disabling Features

Installer Option	Tells System
`nofirewire`	Not to load support for firewire devices
`nodma`	Not to load DMA support for hard disks
`noide`	Not to load support for IDE devices
`nompath`	Not to enable support for multipath devices
`noparport`	Not to load support for parallel ports
`nopcmcia`	Not to load support for PCMCIA controllers
`noprobe`	Not to probe hardware; instead prompt user for drivers
`noscsi`	Not to load support for SCSI devices
`nousb`	Not to load support for USB devices
`noipv6`	Not to enable IPV6 networking
`nonet`	Not to probe for network devices
`numa-off`	To disable the Non-Uniform Memory Access (NUMA) for AMD64 architecture
`acpi=off`	To disable the Advanced Configuration and Power Interface (ACPI)

TABLE 9.2 Boot Options for Video Problems

Boot Option	Tells System
`xdriver=vesa`	Use standard vesa video driver
`resolution=1024x768`	Choose exact resolution to use
`nofb`	Don't use the VGA 16 framebuffer driver
`skipddc`	Don't probe DDC of the monitor (the probe can hang the installer)
`graphical`	Force a graphical installation

If you want to start installation on one computer, but you want to answer the installation questions from another computer, you can enable a `VNC` (virtual network computing) installation. After you start this type of installation, you can go to another system and open a `vnc viewer`, giving the viewer the address of the installation machine (such as 192.168.0.99:1). Table 9.3 provides the necessary commands, along with what to tell the system to do.

Boot options for kickstarts and remote repositories

You can boot the installation process from an installation medium that contains little more than the kernel and initial RAM disk. If that is the case, you need to identify the repository where the software packages exist. You can do that by providing a kickstart file or by identifying the location of the repositories in some way. To force the installer to prompt for the repository location (CD/DVD, hard drive, NFS, or URL), add `askmethod` to the installation boot options.

TABLE 9.3 Boot Options for VNC Installations

Boot Option	Tells System
`vnc`	Run installation as a VNC server
`vncconnect=`*`hostname[:port]`*	Connect to VNC client hostname and optional port
`vncpassword=`*`password`*	Client uses password (at least 8 characters) to connect to installer

Using `repo=` options, you can identify software repository locations. The following examples show the syntax to use for creating `repo=` entries:

```
repo=hd:/dev/sda1:/myrepo
Repository in /myrepo on disk 1 first partition
repo=http://abc.example.com/myrepo
Repository available from /myrepo on web server
repo=ftp://ftp.example.com/myrepo
Repository available from /myrepo on FTP server
repo=cdrom
Repository available from local CD or DVD
repo=nfs::mynfs.example.com:/myrepo/
Repository available from /myrepo on NFS share
repo=nfsiso::nfs.example.com:/mydir/rhel7.iso
Installation ISO image available from NFS server
```

Instead of identifying the repository directly, you can specify it within a kickstart file. The following are examples of some ways to identify the location of a kickstart file.

```
ks=cdrom:/stuff/ks.cfg
Get kickstart from CD/DVD.
ks=hd:sda2:/test/ks.cfg
Get kickstart from test directory on hard disk(sda2).
ks=http://www.example.com/ksfiles/ks.cfg
Get kickstart from a web server.
ks=ftp://ftp.example.com/allks/ks.cfg
Get kickstart from a FTP server.
ks=nfs:mynfs.example.com:/someks/ks.cfg
Get kickstart from an NFS server.
```

Miscellaneous boot options

Here are a few other options that you can pass to the installer that don't fit in a category.

```
rescue
Instead of installing, run the kernel to open Linux rescue mode.

mediacheck
Check the installation CD/DVD for checksum errors.
```

For further information on using the anaconda installer in rescue mode (to rescue a broken Linux system), see Chapter 21, "Troubleshooting Linux." For information on the latest boot options use in RHEL 8, refer to the RHEL 8 Installation Guide:

```
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/
  html-single/performing_a_standard_rhel_installation/index#custom-boot-
  options_booting-the-installer
```

Using specialized storage

In large enterprise computing environments, it is common to store the operating system and data outside of the local computer. Instead, some special storage device beyond the local hard disk is identified to the installer, and that storage device (or devices) can be used during installation.

Once identified, the storage devices that you indicate during installation can be used the same way that local disks are used. You can partition them and assign a structure (filesystem, swap space, and so on) or leave them alone and simply mount them where you want the data to be available.

The following types of specialized storage devices can be selected from the Specialized Storage Devices screen when you install Red Hat Enterprise Linux, Fedora, or other Linux distributions:

Firmware RAID A firmware RAID device is a type of device that has hooks in the BIOS, allowing it to be used to boot the operating system, if you choose.

Multipath devices As the name implies, multipath devices provide multiple paths between the computer and its storage devices. These paths are aggregated, so these devices look like a single device to the system using them, while the underlying technology provides improved performance, redundancy, or both. Connections can be provided by iSCSI or Fibre Channel over Ethernet (FCoE) devices.

Other SAN devices Any device representing a Storage Area Network (SAN).

While configuring these specialized storage devices is beyond the scope of this book, know that if you are working in an enterprise where iSCSI and FCoE devices are available, you can configure your Linux system to use them at installation time. You need the following types of information to do this:

iSCSI devices Have your storage administrator provide you with the target IP address of the iSCSI device and the type of discovery authentication needed to use the device. The iSCSI device may require credentials.

Fibre Channel over Ethernet Devices (FCoE) For FCoE, you need to know the network interface that is connected to your FCoE switch. You can search that interface for available FCoE devices.

Partitioning hard drives

The hard disk (or disks) on your computer provide the permanent storage area for your data files, applications programs, and the operating system itself. *Partitioning* is the act of dividing a disk into logical areas that can be worked with separately. In Windows, you typically have one partition that consumes the whole hard disk. However, with Linux there are several reasons you may want to have multiple partitions:

Multiple operating systems If you install Linux on a PC that already has a Windows operating system, you may want to keep both operating systems on the computer. For all practical purposes, each operating system must exist on a completely separate partition. When your computer boots, you can choose which system to run.

Multiple partitions within an operating system To protect their entire operating system from running out of disk space, people often assign separate partitions to different areas of the Linux filesystem. For example, if `/home` and `/var` were assigned to separate partitions, then a gluttonous user who fills up the `/home` partition wouldn't prevent logging daemons from continuing to write to log files in the `/var/log` directory.

Multiple partitions also make doing certain kinds of backups (such as an image backup) easier. For example, an image backup of `/home` would be much faster (and probably more useful) than an image backup of the root filesystem (`/`).

Different filesystem types Different kinds of filesystems have different structures. Filesystems of different types must be on their own partitions. Also, you might need different filesystems to have different mount options for special features (such as read-only or user quotas). In most Linux systems, you need at least one filesystem type for the root of the filesystem (`/`) and one for your swap area. Filesystems on CD-ROM use the iso9660 filesystem type.

Tip

When you create partitions for Linux, you usually assign the filesystem type as Linux native (using the ext2, ext3, ext4, or xfs type on most Linux systems). If the applications that you are running require particularly long filenames, large file sizes, or many inodes (each file consumes an inode), you may want to choose a different filesystem type.

Coming from Windows

If you have only used Windows operating systems before, you probably had your whole hard disk assigned to C: and never thought about partitions. With many Linux systems, you have the opportunity to view and change the default partitioning based on how you want to use the system.

During installation, systems such as Fedora and RHEL let you partition your hard disk using graphical partitioning tools. The following sections describe how to partition your disk during a Fedora installation. See the section "Tips for creating partitions" for some ideas for creating disk partitions.

Understanding different partition types

Many Linux distributions give you the option of selecting different partition types when you partition your hard disk during installation. Partition types include the following:

Linux partitions Use this option to create a partition for an ext2, ext3, or ext4 filesystem type that is added directly to a partition on your hard disk (or other storage medium). The xfs filesystem type can also be used on a Linux partition. (In fact, xfs is now the default filesystem type for RHEL 8 systems.)

LVM partitions Create an LVM partition if you plan to create or add to an LVM volume group. LVMs give you more flexibility in growing, shrinking, and moving partitions later than regular partitions do.

RAID partitions Create two or more RAID partitions to create a RAID array. These partitions should be on separate disks to create an effective RAID array. RAID arrays can help improve performance, reliability, or both as those features relate to reading, writing, and storing your data.

Swap partitions Create a swap partition to extend the amount of virtual memory available on your system.

The following sections describe how to add regular Linux partitions and LVM, RAID, and swap partitions using the Fedora graphical installer. If you are still not sure when you should use these different partition types, refer to Chapter 12, "Managing Disks and Filesystems," for further information on configuring disk partitions.

Tips for creating partitions

Changing your disk partitions to handle multiple operating systems can be very tricky, in part because each operating system has its own ideas about how partitioning information should be handled as well as different tools for doing it. Here are some tips to help you get it right:

- If you are creating a dual-boot system, particularly for a Windows system, try to install the Windows operating system first after partitioning your disk. Otherwise, the Windows installation may make the Linux partitions inaccessible.
- The `fdisk` man page recommends that you use partitioning tools that come with an operating system to create partitions for that operating system. For example, the Windows `fdisk` knows how to create partitions that Windows will like, and the Linux `fdisk` will happily make your Linux partitions. After your hard disk is set up for dual boot, however, you should probably not go back to Windows-only partitioning tools. Use Linux `fdisk` or a product made for multi-boot systems (such as Acronis Disk Director).
- A master boot record (MBR) partition table can contain four primary partitions, one of which can be marked to contain 184 logical drives. On a GPT partition table, you can have a maximum of 128 primary partitions on most operating systems, including Linux. You typically won't need nearly that many partitions. If you need more partitions, use LVM and create as many logical volumes as you like.

If you are using Linux as a desktop system, you probably don't need lots of different partitions. However, some very good reasons exist for having multiple partitions for Linux systems that are shared by lots of users or are public web servers or file servers. Having multiple partitions within Fedora or RHEL, for example, offers the following advantages:

Protection from attacks Denial-of-service attacks sometimes take actions that try to fill up your hard disk. If public areas, such as `/var`, are on separate partitions, a successful attack can fill up a partition without shutting down the whole computer. Because `/var` is the default location for web and FTP servers, and is expected to hold lots of data, entire hard disks often are assigned to the `/var` filesystem alone.

Protection from corrupted filesystems If you have only one filesystem (`/`), its corruption can cause the whole Linux system to be damaged. Corruption of a smaller partition can be easier to fix and often allows the computer to stay in service while the correction is made.

Table 9.4 lists some directories that you may want to consider making into separate filesystem partitions.

TABLE 9.4 Assigning Partitions to Particular Directories

Directory	Explanation
`/boot`	Sometimes, the BIOS in older PCs can access only the first 1024 cylinders of your hard disk. To make sure that the information in your `/boot` directory is accessible to the BIOS, create a separate disk partition (by default, RHEL 8 sets this partition to 1024 MiB) for `/boot`. Even with several kernels installed, there is rarely a reason for `/boot` to be larger than 1024 MiB.
`/usr`	This directory structure contains most of the applications and utilities available to Linux users. The original theory was that if `/usr` were on a separate partition, you could mount that filesystem as read-only after the operating system had been installed. This would prevent attackers from replacing or removing important system applications with their own versions that may cause security problems. A separate `/usr` partition is also useful if you have diskless workstations on your local network. Using NFS, you can share `/usr` over the network with those workstations.
`/var`	Your FTP (`/var/ftp`) and web server (`/var/www`) directories are, by default in many Linux systems, stored under `/var`. Having a separate `/var` partition can prevent an attack on those facilities from corrupting or filling up your entire hard disk.
`/home`	Because your user account directories are located in this directory, having a separate `/home` account can prevent a reckless user from filling up the entire hard disk. It also conveniently separates user data from your operating system (for easy backups or new installs). Often, `/home` is created as an LVM logical volume, so it can grow in size as user demands increase. It may also be assigned user quotas to limit disk use.
`/tmp`	Protecting `/tmp` from the rest of the hard disk by placing it on a separate partition can ensure that applications that need to write to temporary files in `/tmp` can complete their processing, even if the rest of the disk fills up.

Although people who use Linux systems rarely see a need for lots of partitions, those who maintain and occasionally have to recover large systems are thankful when the system they need to fix has several partitions. Multiple partitions can limit the effects of deliberate damage (such as denial-of-service attacks), problems from errant users, and accidental filesystem corruption.

Using the GRUB boot loader

A boot loader lets you choose when and how to boot the operating systems installed on your computer's hard disks. The *GRand Unified Bootloader (GRUB)* is the most popular boot loader used for installed Linux systems. There are two major versions of GRUB available today:

- GRUB Legacy (version 1). This version of GRUB was used with earlier versions of RHEL, Fedora, and Ubuntu.
- GRUB 2. The current versions of Red Hat Enterprise Linux, Ubuntu, and Fedora use GRUB 2 as the default boot loader.

Note

SYSLINUX is another boot loader that you will encounter with Linux systems. The SYSLINUX boot loaders are not typically used for installed Linux systems. However, SYSLINUX is commonly used as the boot loader for bootable Linux CDs and DVDs. SYSLINUX is particularly good for booting ISO9660 CD images (isolinux) and USB sticks (syslinux) and for working on older hardware or for PXE booting (pxelinux) a system over the network.

If you want to boot to a particular run level, you can add the run level you want to the end of the kernel line. For example, to have RHEL boot to run level 3 (multiuser plus networking mode), add `3` to the end of the kernel line. You can also boot to single-user mode (`1`), multiuser mode (`2`), or X GUI mode (`5`). Level 3 is a good choice if your GUI is temporarily broken. Level 1 is good if you have forgotten your root password.

By default, you will see a splash screen as Linux boots. If you want to see messages showing activities happening as the system boots up, you can remove the option `rhgb quiet` from the kernel line. This lets you see messages as they scroll by. Pressing Esc during bootup gets the same result.

GRUB 2 represents a major rewrite of the GRUB Legacy project. It was adopted as the default boot loader for the latest Red Hat Enterprise Linux, Fedora, and Ubuntu releases. The major function of the GRUB 2 boot loader is still to find and start the operating system you want, but now much more power and flexibility is built into the tools and configuration files that get you there.

In GRUB 2, the configuration file is now named `/boot/grub2/grub.cfg` or `/etc/grub2-efi.cfg` (for systems booted with EFI). Everything from the contents of `grub.cfg` to the way `grub.cfg` is created is different from the GRUB Legacy `grub.conf` file.

Here are some things you should know about the `grub.cfg` file:

- Instead of editing `grub.cfg` by hand or having kernel RPM packages add to it, `grub.cfg` is generated automatically from the contents of the `/etc/default/grub` file and the `/etc/grub.d/` directory. You should modify or add to those files to configure GRUB 2 yourself.
- The `grub.cfg` file can contain scripting syntax, including such things as functions, loops, and variables.
- Device names needed to identify the location of kernels and initial RAM disks can be more reliably identified using labels or universally unique identifiers (UUIDs). This prevents the possibility of a disk device such as `/dev/sda` being changed to `/dev/sdb` when you add a new disk (which would result in the kernel not being found).
- For Fedora and RHEL systems, `*conf` files in the `/boot/loader/entries` directory are used to create entries that appear on the GRUB menu that appears at boot time.

You could create your own entry for the GRUB boot menu by following the format of an existing entry. The following file in the `/boot/loader/entries` directory creates a menu entry for booting a RHEL 8 kernel and `initrd`:

```
title Red Hat Enterprise Linux (4.18.0-80.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-80.el8.x86_64
linux /vmlinuz-4.18.0-80.el8.x86_64
initrd /initramfs-4.18.0-80.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20190313123447-4.18.0-80.el8.x86_64
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

The menu entry for this selection appears as `Red Hat Enterprise Linux (4.18.0-80.el8.x86_64) 8.0 (Ootpa)` on the GRUB 2 boot menu.

The `linux` line identifies the location of the kernel (`/vmlinuz-4.18.0-80.el8.x86_64`), followed by the location of the `initrd` (`/initramfs-4.18.0-80.el8.x86_64.img`).

There are many, many more features of GRUB 2 that you can learn about if you want to dig deeper into your system's boot loader. The best documentation for GRUB 2 is available by typing **`info grub2`** at the shell. The `info` entry for GRUB 2 provides lots of information for booting different operating systems, writing your own configuration files, working with GRUB image files, setting GRUB environment variables, and working with other GRUB features.

Summary

Although every Linux distribution includes a different installation method, you need to do many common activities, regardless of which Linux system you install. For every Linux system, you need to deal with issues of disk partitioning, boot options, and configuring boot loaders.

In this chapter, you stepped through installation procedures for a Fedora Workstation (using a live media installation) and Red Hat Enterprise Linux (from installation media). You learned how deploying Linux in cloud environments can differ from traditional installation methods by combining metadata with prebuilt base operating system image files to run on large pools of compute resources.

The chapter also covered special installation topics, including using boot options and disk partitioning. With your Linux system now installed, Chapter 10, "Getting and Managing Software," describes how to begin managing the software on your Linux system.

Exercises

Use these exercises to test your knowledge of installing Linux. I recommend that you do these exercises on a computer that has no operating system or data on it that you would fear losing (in other words, one you don't mind erasing). If you have a computer that allows you to install virtual systems, that is a safe way to do these exercises as well. These exercises were tested using Fedora 30 Workstation Live media and a RHEL 8 Installation DVD.

1. Start installing from Fedora Live media, using as many of the default options as possible.
2. After you have completely installed Fedora, update all of the packages on the system.
3. Start installing from an RHEL installation DVD but make it so that the installation runs in text mode. Complete the installation in any way you choose.
4. Start installing from an RHEL installation DVD and set the disk partitioning as follows: a 1024MB `/boot`, `/` (6G), `/var` (2G), and `/home` (2G). Leave the rest as unused space.

Caution

Completing Exercise 4 ultimately deletes all content on your hard disk. If you just want to use this exercise to practice partitioning, you can reboot your computer before clicking Accept Changes at the very end of this procedure without harming your hard disk. If you go forward and partition your disk, assume that all data that you have not explicitly changed has been deleted.

CHAPTER 10

Getting and Managing Software

IN THIS CHAPTER

Installing software from the desktop

Working with RPM packaging

Using YUM to manage packages

Using rpm to work with packages

Installing software in the enterprise

In Linux distributions such as Fedora and Ubuntu, you don't need to know much about how software is packaged and managed to get the software you want. Those distributions have excellent software installation tools that automatically point to huge software repositories. Just a few clicks and you're using the software in little more time than it takes to download it.

The fact that Linux software management is so easy these days is a credit to the Linux community, which has worked diligently to create packaging formats, complex installation tools, and high-quality software packages. Not only is it easy to get the software, but after it's installed, it's easy to manage, query, update, and remove it.

This chapter begins by describing how to install software in Fedora using the new Software graphical installation tool. If you are just installing a few desktop applications on your own desktop system, you may not need much more than that and occasional security updates.

To dig deeper into managing Linux software, next I describe what makes up Linux software packages (comparing `deb` and `rpm` formatted packaging), underlying software management components, and commands (`dnf`, `yum`, and `rpm`) for managing software in Fedora and Red Hat Enterprise Linux. That's followed by a description of how to manage software packages in enterprise computing.

Managing Software on the Desktop

The Fedora Software window offers an intuitive way of choosing and installing desktop applications that does not align with typical Linux installation practices. The Ubuntu Software window offers the same interface for Ubuntu users. In either case, with the Software window, the smallest

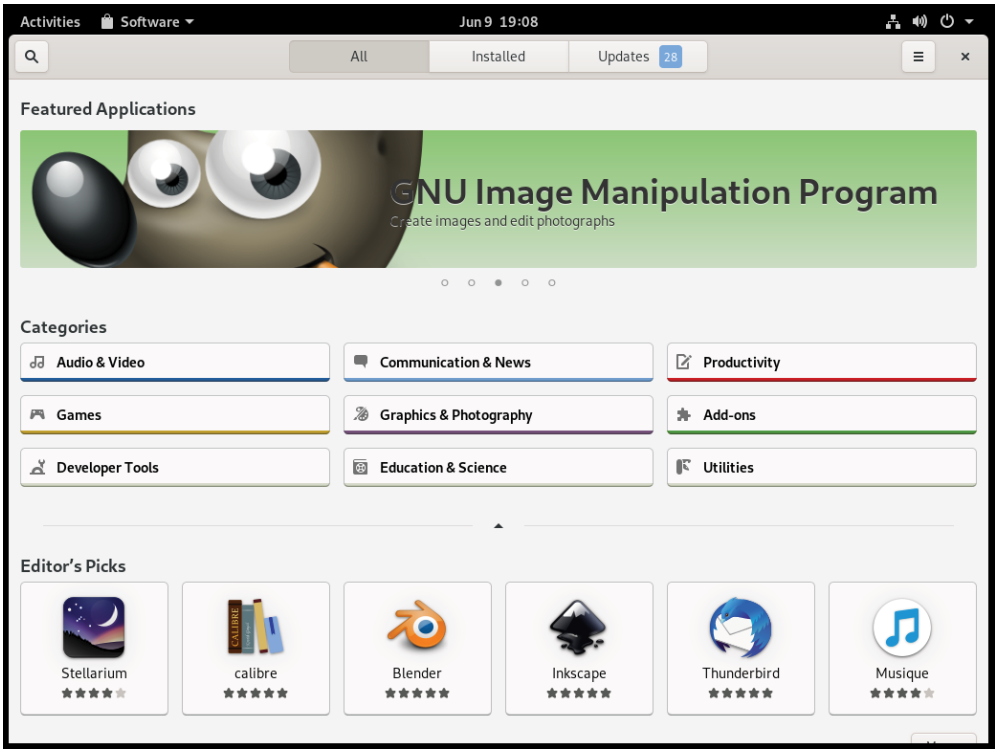
Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

software component you install is an application. With Linux, you install packages (such as `rpms` and `debs`).

Figure 10.1 shows an example of the Software window.

FIGURE 10.1

Install and manage software packages from the Software window.



To get to the Software window in either Fedora or Ubuntu, select Activities, then type **Software**, and press Enter. The first time you open this window, you can select Enable to allow third-party software repositories that are not part of the official redistributable Fedora repositories. Using the Software window is the best way to install desktop-oriented applications, such as word processors, games, graphics editors, and educational applications.

From the Software window, you can select the applications that you want to install from the Editor's Picks group (a handful of popular applications), choose from categories of applications (Audio & Video, Games, Graphics & Photography, and so on), or search by application

name or description. Select the Install button to have the Software window download and install all of the software packages needed to make the application work.

Other features of this window let you see all installed applications (Installed tab) or view a list of applications that have updated packages available for you to install (Updates tab). If you want to remove an installed application, simply click the Remove button next to the package name.

If you are using Linux purely as a desktop system, where you want to write documents, play music, and do other common desktop tasks, the Software window might be all you need to get the basic software you want. By default, your system connects to the main Fedora software repository and gives you access to hundreds of software applications. As noted earlier, you also have the option of accessing third-party applications that are still free for you to use but not redistribute.

Although the Software window lets you download and install hundreds of applications from the Fedora software repository, that repository actually contains tens of thousands of software packages. What packages can you not see from that repository, when might you want those other packages, and how can you gain access to those packages (as well as packages from other software repositories)?

Going Beyond the Software Window

If you are managing a single desktop system, you might be quite satisfied with the hundreds of packages that you can find through the Software window. Open-source versions of most common types of desktop applications are available to you through the Software window after you have a connection from Fedora to the Internet.

However, the following are some examples of why you might want to go beyond what you can do with the Software window:

More repositories Fedora and Red Hat Enterprise Linux distribute only open-source, freely distributable software. You may want to install some commercial software (such as Adobe Flash Player) or non-free software (available from repositories such as `rpmfusion.org`).

Beyond desktop applications Tens of thousands of software packages in the Fedora repository are not available through the Software window. Most of these packages are not associated with graphical applications at all. For example, some packages contain pure command-line tools, system services, programming tools, or documentation that doesn't show up in the Software window.

Flexibility Although you may not know it, when you install an application through the Software window, you may actually be installing multiple RPM packages. This set of packages may just be a default package set that includes documentation, extra fonts, additional software plug-ins, or multiple language packs that you

may or may not want. With `yum` and `rpm` commands, you have more flexibility on exactly which packages related to an application or other software feature is installed on your system.

More complex queries Using commands such as `yum` and `rpm`, you can get detailed information about packages, package groups, and repositories.

Software validation Using `rpm` and other tools, you can check whether a signed package has been modified before you installed it or whether any of the components of a package have been tampered with since the package was installed.

Managing software installation Although the Software window works well if you are installing desktop software on a single system, it doesn't scale well for managing software on multiple systems. Other tools are built on top of the `rpm` facility for doing that.

Before I launch into some of the command-line tools for installing and managing software in Linux, the next section describes how the underlying packaging and package management systems in Linux work. In particular, I focus on RPM packaging as it is used in Fedora, Red Hat Enterprise Linux, and related distributions as well as Deb packages, which are associated with Debian, Ubuntu, Linux Mint, and related distributions.

Understanding Linux RPM and DEB Software Packaging

On the first Linux systems, if you wanted to add software, you would grab the source code from a project that produced it, compile it into runnable binaries, and drop it onto your computer. If you were lucky, someone would have already compiled it in a form that would run on your computer.

The form of the package could be a tarball containing executable files (commands), documentation, configuration files, and libraries. (A *tarball* is a single file in which multiple files are gathered together for convenient storage or distribution.) When you install software from a tarball, the files from that tarball might be spread across your Linux system in appropriate directories (`/usr/share/man`, `/etc`, `/bin`, and `/lib`, to name just a few). Although it is easy to create a tarball and just drop a set of software onto your Linux system, this method of installing software makes it difficult to do these things:

Get dependent software You would need to know if the software you were installing depended on other software being installed for your software to work. Then you would have to track down that software and install that too (which might itself have some dependencies).

List the software Even if you knew the name of the command, you might not know where its documentation or configuration files were located when you looked for it later.

Remove the software Unless you kept the original tarball, or a list of files, you wouldn't know where all of the files were when it came time to remove them. Even if you knew, you would have to remove each one individually.

Update the software Tarballs are not designed to hold metadata about the contents that they contain. After the contents of a tarball are installed, you may not have a way to tell what version of the software you are using, making it difficult to track down bugs and get new versions of your software.

To deal with these problems, packages progressed from simple tarballs to more complex packaging. With only a few notable exceptions (such as Gentoo, Slackware, and a few others), the majority of Linux distributions went to one of two packaging formats—DEB and RPM:

DEB (.deb) packaging The Debian GNU/Linux project created `.deb` packaging, which is used by Debian and other distributions based on Debian (Ubuntu, Linux Mint, KNOPPIX, and so on). Using tools such as `apt-get`, `apt`, and `dpkg`, Linux distributions could install, manage, upgrade, and remove software.

RPM (.rpm) packaging Originally named Red Hat Package Manager, but later recursively renamed RPM Package Manager, RPM is the preferred package format for SUSE, Red Hat distributions (RHEL and Fedora), and those based on Red Hat distributions (CentOS, Oracle Linux, and so on). The `rpm` command was the first tool to manage RPMs. Later, `yum` was added to enhance the RPM facility, and now `dnf` is poised to eventually replace `yum`.

For managing software on individual systems, there are proponents on both sides of the RPM vs. DEB debate with valid points. Although RPM is the preferred format for managing enterprise-quality software installation, updates, and maintenance, DEB is very popular among many Linux enthusiasts. This chapter covers both RPM (Fedora and Red Hat Enterprise Linux) and (to some extent) DEB packaging and software management.

Understanding DEB packaging

Debian software packages hold multiple files and metadata related to some set of software in the format of an `ar` archive file. The files can be executables (commands), configuration files, documentation, and other software items. The metadata includes such things as dependencies, licensing, package sizes, descriptions, and other information. Multiple command-line and graphical tools are available for working with DEB files in Ubuntu, Debian, and other Linux distributions. Some of these include the following:

Ubuntu Software Center Select the Ubuntu Software application from the GNOME Activities menu. The window that appears lets you search for applications and packages that you want by searching for keywords or navigating categories.

aptitude The `aptitude` command is a package installation tool that provides a screen-oriented menu that runs in the shell. After you run the command, use arrow keys to highlight the selection you want, and press Enter to select it. You can upgrade packages, get new packages, or view installed packages.

apt* There is a set of `apt*` commands (`apt-get`, `apt`, `apt-config`, `apt-cache`, and so on) that can be used to manage package installation.

The Ubuntu Software Center is fairly intuitive for finding and installing packages. However, here are a few examples of commands that can help you install and manage packages with `apt*` commands. In this case, I'm looking for and installing the `vsftpd` package:

NOTE

Notice that the `apt*` commands are preceded by the `sudo` command in these examples. That's because it is common practice for an Ubuntu administrator to run administrative commands as a regular user with `sudo` privilege.

```
$ sudo apt-get update           Get the latest package versions
$ sudo apt-cache search vsftpd  Find package by key word (such as vsftpd)
$ sudo apt-cache show vsftpd    Display information about a package
$ sudo apt-get install vsftpd   Install the vsftpd package
$ sudo apt-get upgrade          Update installed packages if upgrade ready
$ sudo apt-cache pkgnames       List all packages that are installed
```

There are many other uses of `apt*` commands that you can try out. If you have an Ubuntu system installed, I recommend that you run `man apt` to get an understanding of what the `apt` and related commands can do.

Understanding RPM packaging

An *RPM package* is a consolidation of files needed to provide a feature, such as a word processor, a photo viewer, or a file server. The commands, configuration files, and documentation that make up the software feature can be inside an RPM. However, an RPM file also contains metadata that stores information about the contents of that package, where the package came from, what it needs to run, and other information.

What is in an RPM?

Before you even look inside an RPM, you can tell much about it by the name of the RPM package itself. To find out the name of an RPM package currently installed on your system (in this example the Firefox web browser), you could type the following from the shell in Fedora or Red Hat Enterprise Linux:

```
# rpm -q firefox
firefox-67.0-4.fc30.x86_64
```

From this, you can tell that the base name of the package is `firefox`. The version number is 67.0 (assigned by the upstream producer of Firefox, the Mozilla Project). The release

(assigned by the packager, Fedora, each time the package is rebuilt at the same release number) is 4. The `firefox` package was built for Fedora 30 (fc30) and is compiled for the x86 64-bit architecture (x86_64).

When the `firefox` package was installed, it was probably copied from the installation medium (such as a CD or DVD) or downloaded from a YUM repository (more on that later). If you had been given the RPM file and it was sitting in a local directory, the name would appear as `firefox-67.0-4.fc30.x86_64.rpm` and you could install it from there. Regardless of where it came from, once the package is installed, the name and other information about it are stored in an RPM database on the local machine.

To find out more about what is inside an RPM package, you can use options other than the `rpm` command to query that local RPM database, as in this example:

```
# rpm -qi firefox
Name        : firefox
Version     : 67.0
Release     : 4.fc30
Architecture: x86_64
Install Date: Sun 02 Jun 2019 09:37:25 PM EDT
Group       : Unspecified
Size        : 266449296
License     : MPLv1.1 or GPLv2+ or LGPLv2+
Signature   : RSA/SHA256, Fri 24 May 2019 12:09:57 PM EDT, Key ID
ef3c111fcfc659b9
Source RPM  : firefox-67.0-4.fc30.src.rpm
Build Date  : Thu 23 May 2019 10:03:55 AM EDT
Build Host  : buildhw-08.phx2.fedoraproject.org
Relocations : (not relocatable)
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : https://www.mozilla.org/firefox/
Bug URL     : https://bugz.fedoraproject.org/firefox
Summary     : Mozilla Firefox Web browser
Description :
Mozilla Firefox is an open-source web browser, designed for standards
compliance, performance and portability.
```

Besides the information you got from the package name itself, the `-qi` (query information) option lets you see who built the package (Fedora Project), when it was built, and when it was installed. The group the package is in (Unspecified), its size, and the licensing are listed. To enable you to find out more about the package, the URL points to the project page on the Internet and the Summary and Description lines tell you what the package is used for.

Where do RPMs come from?

The software included with Linux distributions, or built to work with those distributions, comes from thousands of open-source projects all over the world. These projects, referred to

as *upstream software providers*, usually make the software available to anyone who wants it, under certain licensing conditions.

A Linux distribution takes the source code and builds it into binaries. Then it gathers those binaries together with documentation, configuration files, scripts, and other components available from the upstream provider.

After all of those components are gathered into the RPM, the RPM package is signed (so that users can test the package for validity) and placed in a repository of RPMs for the specific distribution and architecture (32-bit x86, 64-bit x86, and so on). The repository is placed on an installation CD or DVD or in a directory that is made available as an FTP, web, or NFS server.

Installing RPMs

When you initially install a Fedora or Red Hat Enterprise Linux system, many individual RPM packages make up that installation. After Linux is installed, you can add more packages using the Software window (as described earlier). Refer to Chapter 9, "Installing Linux," for information on installing Linux.

The first tool to be developed for installing RPM packages, however, was the `rpm` command. Using `rpm`, you can install, update, query, validate, and remove RPM packages. The command, however, has some major drawbacks:

Dependencies For most RPM packages to work, some other software (library, executables, and so on) must be installed on the system. When you try to install a package with `rpm`, if a dependent package is not installed, the package installation fails, telling you which components were needed. At that point, you have to dig around to find what package contained that component. When you go to install it, that dependent package might itself have dependencies that you need to install to get it to work. This situation is lovingly referred to as "dependency hell" and is often used as an example of why DEB packages were better than RPMs. DEB packaging tools were made to resolve package dependencies automatically, well before RPM-related packaging tools could do that.

Location of RPMs The `rpm` command expects you to provide the exact location of the RPM file when you try to install it. In other words, you would have to give `firefox-67.0-4.fc30.x86_64.rpm` as an option if the RPM were in the current directory or `http://example.com/firefox-67.0-4.fc30.x86_64.rpm` if it were on a server.

As Red Hat Linux and other RPM-based applications grew in popularity, it became apparent that something had to be done to make package installation more convenient. The answer was the YUM facility.

Managing RPM Packages with YUM

The *YellowDog Updater Modified (YUM)* project set out to solve the headache of managing dependencies with RPM packages. Its major contribution was to stop thinking about RPM packages as individual components and think of them as parts of larger software repositories.

With repositories, the problem of dealing with dependencies fell not to the person who installed the software but to the Linux distribution or third-party software distributor that makes the software available. So, for example, it would be up to the Fedora Project to make sure that every component needed by every package in its Linux distribution could be resolved by some other package in the repository.

Repositories could also build on each other. So, for example, the `rpmfusion.org` repository could assume that a user already had access to the main Fedora repository. If a package being installed from `rpmfusion.org` needed a library or command from the main Fedora repository, the Fedora package could be downloaded and installed at the same time that you install the `rpmfusion.org` package.

The yum repositories could be put in a directory on a web server (`http://`), on an FTP server (`ftp://`), on local media such as a CD or DVD, or in a local directory (`file://`). The locations of these repositories would then be stored on the user's system in the `/etc/yum.conf` file or, more typically, in separate configuration files in the `/etc/yum.repos.d` directory.

Transitioning from yum to dnf

The `dnf` command-line interface represents the next generation of YUM. DNF, which refers to itself as *Dandified YUM*, (`https://github.com/rpm-software-management/dnf/`) has been part of Fedora since version 18 and has just been added as the default RPM package manager for RHEL 8. Like `yum`, `dnf` is a tool for finding, installing, querying, and generally managing RPM packages from remote software repositories to your local Linux system.

While `dnf` maintains a basic command-line compatibility with `yum`, one of its main differences is that it adheres to a strict API. That API encourages the development of extensions and plug-ins to `dnf`.

For our purposes, almost all of the `yum` commands described in this chapter can be replaced by `dnf` or used as they are. The `yum` command is a symbolic link to `dnf` in Fedora and RHEL, so typing either command has the same result. For more information on DNF, refer to the DNF page at `https://dnf.readthedocs.io/`.

Understanding how yum works

This is the basic syntax of the `yum` command:

```
# yum [options] command
```

Using that syntax, you can find packages, see package information, find out about package groups, update packages, or delete packages, to name a few features. With the YUM repository and configuration in place, a user can install a package by simply typing something like the following:

```
# yum install firefox
```

The user only needs to know the package name (which could be queried in different ways, as described in the section "Searching for packages" later in this chapter). The YUM facility finds the latest version of that package available from the repository, downloads it to the local system, and installs it.

To gain more experience with the YUM facility, and to see where there are opportunities for you to customize how YUM works on your system, follow the descriptions of each phase of the YUM install process described next.

NOTE

In the latest Fedora and RHEL systems, the YUM configuration files are now actually links to DNF files in the `/etc/dnf` directory. Besides the main `dnf` configuration file (`/etc/dnf/dnf.conf`), that directory primarily contains modules and plug-ins that add to your ability to manage RPM packages.

1. **Checking `/etc/yum.conf`:**

   When any `yum` command starts, it checks the file `/etc/yum.conf` for default settings. The `/etc/yum.conf` file is the basic YUM configuration file. You can also identify the location of repositories here, although the `/etc/yum.repos.d` directory is the more typical location for identifying repositories. Here's an example of `/etc/yum.conf` on a RHEL 8 system, with a few others added:

   ```
   [main]
   gpgcheck=1
   installonly_limit=3
   clean_requirements_on_remove=True
   best=True

   cachedir=/var/cache/yum/$basearch/$releasever
   keepcache=0
   debuglevel=2
   logfile=/var/log/yum.log
   exactarch=1
   plugins=1
   ```

   The `gpgcheck` is used to validate each package against a key that you receive from those who built the RPM. It is on by default (`gpgcheck=1`). For packages in Fedora or RHEL, the key comes with the distribution to check all packages. To install packages that are not from your distribution, you need either to import the key to verify those packages or to turn off that feature (`gpgcheck=0`).

The `install_onlylimit=3` setting allows up to three versions of the same package to be kept on the system (don't set this to less than `2`, to ensure that you always have at least two kernel packages). The `clean_requirements_on_remove=True` tells `yum` to remove dependent packages when removing a package, if those packages are not otherwise required. With `best=True`, when upgrading a package, always try to use the highest version available.

Other settings that you can add to `yum.conf` tell YUM where to keep cache files (`/var/cache/yum`) and log entries (`/var/log/yum.log`) and whether to keep cache files around after a package is installed (`0` means no). You can raise the `debuglevel` value in the `yum.conf` file to above `2` if you want to see more details in your log files.

Next, you can see whether the exact architecture (x86, x86_64, and so on) should be matched when choosing packages to install (`1` means yes) and whether to use plug-ins (`1` means yes) to allow for things such as blacklists, white lists, or connecting to Red Hat Network for packages.

To find other features that you can set in the `yum.conf` file, type **`man yum.conf`**.

2. **Checking `/etc/yum.repos.d/*.repo` files:**

   Software repositories can be enabled by dropping files ending in `.repo` into the `/etc/yum.repos.d/` directory that point to the location of one or more repositories. In Fedora, even your basic Fedora repositories are enabled from `.repo` files in this directory. Here's an example of a simple yum configuration file named `/etc/yum.repos.d/myrepo.repo`:

   ```
   [myrepo]
   name=My repository of software packages
   baseurl=http://myrepo.example.com/pub/myrepo
   enabled=1
   gpgcheck=1
   gpgkey=file:///etc/pki/rpm-gpg/MYOWNKEY
   ```

   Each repository entry begins with the name of the repository enclosed in square brackets. The `name` line contains a human-readable description of the repository. The `baseurl` line identifies the directory containing the RPM files, which can be an `httpd://`, `ftp://`, or `file://` entry.

   The `enabled` line indicates whether the entry is active. A `1` is active; `0` is inactive. If there is no `enabled` line, the entry is active. The last two lines in the preceding code indicate whether to check the signatures on packages in this repository. The `gpgkey` line shows the location of the key that is used to check the packages in this repository.

   You can have as many repositories enabled as you like. However, keep in mind that when you use `yum` commands, every repository is checked and metadata about all packages is downloaded to the local system running the `yum` command. So, to be more efficient, don't enable repositories that you don't need.

3. **Downloading RPM packages and metadata from a YUM repository**:

   After `yum` knows the locations of the repositories, metadata from the `repodata` directory of each repository is downloaded to the local system. In fact, it is the existence of a `repodata` directory in a directory of RPMs that indicates that it is a YUM repository.

   Metadata information is stored on the local system in the `/var/cache/yum` directory. Any further queries about packages, package groups, or other information from the repository are gathered from the cached metadata until a time-out period is reached.

   After the time-out period is reached, `yum` retrieves fresh metadata if the `yum` command is run. By default, the time-out is 6 hours for `yum` and 48 hours for `dnf` minutes. You can change that period by setting `metadata_expire` in the `/etc/yum.conf` file.

   Next, `yum` looks at the packages that you requested to install and checks if any dependent packages are needed by those packages. With the package list gathered, `yum` asks you if it is okay to download all of those packages. If you choose yes, the packages are downloaded to the cache directories and installed.

4. **Installing RPM packages to Linux file system:**

   After all of the necessary packages are downloaded to the cache directories, `yum` runs `rpm` commands to install each package. If a package contains preinstall scripts (which might create a special user account or make directories), those scripts are run. The contents of the packages (commands, config files, docs, and so on) are copied to the filesystem at locations specified in the RPM metadata. Then any post install scripts are run. (Post install scripts run additional commands needed to configure the system after each package is installed.)

5. **Storing YUM repository metadata to local RPM database**.

   The metadata contained in each RPM package that is installed is ultimately copied into the local RPM database. The RPM database is contained in files that are stored in the `/var/lib/rpm` directory. After information about installed packages is in the local RPM database, you can do all sorts of queries of that database. You can see what packages are installed, list components of those packages, and see scripts or change logs associated with each package. You can even validate installed packages against the RPM database to see if anyone has tampered with installed components.

   The `rpm` command (described in the section "Installing, Querying, and Verifying Software with the `rpm` Command" later in this chapter) is the best tool for querying the RPM database. You can run individual queries with `rpm` or use it in scripts to produce reports or run common queries over and over again.

Now that you understand the basic functioning of the `yum` command, your Fedora system should be automatically configured to connect to the main Fedora repository and the Fedora Updates repository. You can try some `yum` command lines to install packages right now. Or, you can enable other third-party YUM repositories to draw software from.

Using YUM with third-party software repositories

The Fedora and Red Hat Enterprise Linux software repositories have been screened to contain only software that meets criteria that make it open and redistributable. In some instances, however, you may want to go beyond those repositories. Before you do, you should understand that some third-party repositories have these limitations:

- They may have less stringent requirements for redistribution and freedom from patent constraints than the Fedora and RHEL repositories have.
- They may introduce some software conflicts.
- They may include software that is not open source and, although it may be free for personal use, may not be redistributable.
- They may slow down the process of installing all of your packages (because metadata is downloaded for every repository you have enabled).

For those reasons, I recommend that you either (1) don't enable any extra software repositories, or (2) enable only the RPM Fusion repository (`http://rpmfusion.org`) at first for Fedora and the EPEL repository (`http://fedoraproject.org/wiki/EPEL`) for Red Hat Enterprise Linux. RPM Fusion represents a fusion of several popular third-party Fedora repositories (Freshrpms, `Livna.org`, and Dribble). See the repository's FAQ for details (`http://rpmfusion.org/FAQ`). To enable the Free RPM Fusion repository in Fedora, do the following:

1. Open a Terminal window.
2. Type **`su`** and enter the root password when prompted.
3. Type the following command on one line with no space in between the slash and `rpmfusion` (if that doesn't work, go to the `fedora` directory and choose the RPM appropriate for your version of Fedora):

   ```
   # rpm -Uvh http://download1.rpmfusion.org/free/fedora/
   rpmfusion-free-release-stable.noarch.rpm
   ```

The RPM Fusion Nonfree repository contains such things as codecs needed to play many popular multimedia formats. To enable the Nonfree repository in Fedora, type the following (again, type the following two lines on a single line, with no space between the two):

```
# rpm -Uhv http://download1.rpmfusion.org/nonfree/fedora/
rpmfusion-nonfree-release-stable.noarch.rpm
```

Most of the other third-party repositories that might interest you contain software that is not open source. For example, if you want to install the Adobe Flash plug-in for Linux, download the YUM repository package from Adobe, and you can use the `yum` command to install the Flash plug-in and get updates later by running the `yum update` command when updates are available.

Managing software with the yum command

The `yum` command has dozens of subcommands that you can use to work with RPM packages on your system. The following sections provide some examples of useful `yum` command

lines to search for, install, query, and update packages associated with your YUM repositories. The section "Installing and removing packages" describes how to install and remove installed packages with the `yum` command.

NOTE

Metadata, describing the contents of YUM repositories, is downloaded from each of your enabled YUM repositories the first time you run a `yum` command. Metadata is downloaded again after the `metadata_expire` time is reached. The more YUM repositories that you enable and the larger they are, the longer this download can take. You can reduce this download time by increasing the expire time (in the `/etc/yum.conf` file) or by not enabling repositories you don't need.

Searching for packages

Using different searching subcommands, you can find packages based on key words, package contents, or other attributes.

Let's say that you want to try out a different text editor but you can't remember the name of the one you wanted. You could start by using the `search` subcommand to look for the term *editor* in the name or description:

```
# yum search editor
...
eclipse-veditor.noarch : Eclipse-based Verilog/VHDL plugin
ed.x86_64 : The GNU line editor
emacs.x86_64 : GNU Emacs text editor
```

The search uncovered a long list of packages containing *editor* in the name or description. The one I was looking for is named `emacs`. To get information about that package, I can use the `info` subcommand:

```
# yum info emacs
Name         : emacs
Epoch        : 1
Version      : 26.2
Release      : 1.fc30
Architecture : x86_64
Size         : 3.2 M
Source       : emacs-26.2-1.fc30.src.rpm
Repository   : updates
Summary      : GNU Emacs text editor
URL          : http://www.gnu.org/software/emacs/
License      : GPLv3+ and CC0-1.0
Description  : Emacs is a powerful, customizable, self-documenting,
modeless text
             : editor. Emacs contains special code editing features,
a scripting
```

```
              : language (elisp), and the capability to read mail,
news, and more
              : without leaving the editor.
```

If you know the command, configuration file, or library name you want but don't know what package it is in, use the `provides` subcommand to search for the package. Here you can see that the `dvdrecord` command is part of the `wodim` package:

```
# yum provides dvdrecord
wodim-1.1.11-41.fc30.x86_64 : A command line CD/DVD recording program
Repo        : fedora
Matched from:
Filename    : /usr/bin/dvdrecord
```

The `list` subcommand can be used to list package names in different ways. Use it with a package base name to find the version and repository for a package. You can list just packages that are `available` or `installed`, or you can list `all` packages.

```
# yum list emacs
emacs.i686     1:26.2-1.fc30     updates
# yum list available
CUnit.i686     2.1.3-17.el8   rhel-8-for-x86_64-appstream-rpms
CUnit.x86_64     2.1.3-17.el8   rhel-8-for-x86_64-appstream-rpms
GConf2.i686      3.2.6-22.el8   rhel-8-for-x86_64-appstream-rpms
LibRaw.i686      0.19.1-1.el8   rhel-8-for-x86_64-appstream-rpm
...
# yum list installed
Installed Packages
GConf2.x86_64       3.2.6-22.el8      @AppStream
ModemManager.x86_64 1.8.0-1.el8       @anaconda
...
# yum list all
...
```

If you find a package but you want to see what components that package is dependent on, you can use the `deplist` subcommand. With `deplist`, you can see the components (dependency) but also the package that component comes in (provider). Using `deplist` can help if no package is available to provide a dependency, but you want to know what the component is so that you can search other repositories for it. Consider the following example:

```
# yum deplist emacs | less
package: emacs-1:26.1-8.fc30.x86_64
  dependency: /bin/sh
   provider: bash-5.0.7-1.fc30.i686
   provider: bash-5.0.7-1.fc30.x86_64
  dependency: /usr/sbin/alternatives
   provider: alternatives-1.11-4.fc30.x86_64
```

Installing and removing packages

The `install` subcommand lets you install one or more packages, along with any dependent packages needed. With `yum install`, multiple repositories can be searched to fulfill needed dependencies. Consider the following example of `yum install`:

```
# yum install emacs
...
Package                 Architecture Version            Repository  Size
================================================================================
Installing:
 emacs                   x86_64       1:26.2-1.fc30      updates    3.2 M
Installing dependencies:
 emacs-common            x86_64       1:26.2-1.fc30      updates     38 M
 ImageMagick-libs        x86_64       1:6.9.10.28-1.fc30 fedora     2.2 M
 fftw-libs-double        x86_64       3.3.8-4.fc30       fedora     984 k
 ...

Transaction Summary
================================================================================
Install  7 Packages

Total download size: 45 M
Installed size: 142 M
Is this ok [y/N]: y
```

You can see here that `emacs` requires that `emacs-common` and several other packages be installed so all are queued up for installation. The six packages together are 45MB to download, but they consume 142MB after installation. Pressing **y** installs them. You can put a `-y` on the command line (just after the `yum` command) to avoid having to press **y** to install the packages. Personally, however, I usually want to see all of the packages about to be installed before I agree to the installation.

You can reinstall a package if you mistakenly delete components of an installed package. If you attempt a regular install, the system responds with "nothing to do." You must instead use the `reinstall` subcommand. For example, suppose that you installed the `zsh` package and then deleted `/bin/zsh` by mistake. You could restore the missing components by typing the following:

```
# yum reinstall zsh
```

You can remove a single package, along with its dependencies that aren't required by other packages, with the `remove` subcommand. For example, to remove the `emacs` package and dependencies, you could type the following:

```
# yum remove emacs
Removing:
 emacs                   x86_64       1:26.2-1.fc30      updates     38 M
```

```
Removing unused dependencies:
 ImageMagick-libs        x86_64       1:6.9.10.28-1.fc30 fedora     8.9 M
 emacs-common            x86_64       1:26.2-1.fc30      updates     89 M
 fftw-libs-double        x86_64       3.3.8-4.fc30       fedora     4.2 M
 ...

Transaction Summary
================================================================================
Remove  7 Packages

Freed space: 142 M
Is this ok [y/N]: y
```

Notice that the space shown for each package is the actual space consumed by the package in the file system and not the download size (which is considerably smaller).

An alternative method to remove a set of packages that you have installed is to use the `history` subcommand. Using `history`, you can see your `yum` activities and undo an entire transaction. In other words, all of the packages that you installed can be uninstalled using the `undo` option of the `history` subcommand, as in the following example:

```
# yum history
ID     | Command line   | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------
    12 | install emacs  | 2019-06-22 11:14 | Install        |    7
...
# yum history info 12
Transaction ID : 12
...
Command Line    : install emacs
...
# yum history undo 12
Undoing transaction 12, from Sat 22 Jun 2019 11:14:42 AM EDT

Install emacs-1:26.2-1.fc30.x86_64                 @updates
    Install emacs-common-1:26.2-1.fc30.x86_64          @updates
 ...
```

Before you undo the transaction, you can view the transaction to see exactly which packages were involved. Viewing the transaction can save you from mistakenly deleting packages that you want to keep. By undoing transaction `12`, you can remove all packages that were installed during that transaction. If you are trying to undo an install that included dozens or even hundreds of packages, `undo` can be a very useful option.

Updating packages

As new releases of a package become available, they are sometimes put into separate update repositories or simply added to the original repository. If multiple versions of a package are available (whether in the same repository or in another enabled repository), `yum` provides the latest version when you install a package. For some packages, such as the Linux kernel package, you can keep multiple versions of the same package.

If a new version of a package shows up later, you can download and install the new version of the package by using the `update` subcommand.

The `check-update` subcommand can check for updates. The `update` subcommand can be used to update a single package or to get updates to all packages that are currently installed and have an update available. Or, you can simply update a single package (such as the `cups` package), as in this example:

```
# yum check-update
...
file.x86_64       5.36-3.fc30     updates
file-libs.x86_64  5.36-3.fc30     updates
firefox.x86_64    67.0.4-1.fc30   updates
firewalld.noarch  0.6.4-1.fc30    updates
...
# yum update
Dependencies resolved.
================================================================
 Package             Arch   Version         Repository    Size
================================================================
Upgrading:
 NetworkManager      x86_64 1:1.16.2-1.fc30 updates      1.7 M
 NetworkManager-adsl x86_64 1:1.16.2-1.fc30 updates       25 k
...
Transaction Summary
================================================================
Install     7 Packages
Upgrade   172 Package(s)
Total download size: 50 M
Is this ok [y/N]: y
# yum update cups
```

The preceding command requested to update the `cups` package. If other dependent packages need to be updated to update `cups`, those packages would be downloaded and installed as well.

Updating groups of packages

To make it easier to manage a whole set of packages at once, YUM supports package groups. For example, you could install GNOME Desktop Environment (to get a whole desktop) or Virtualization (to get packages needed to set up the computer as a virtualization host). You can start by running the `grouplist` subcommand to see a list of group names:

```
# yum grouplist | less
Available Environment Groups:
   Fedora Custom Operating System
   Minimal Install
   Fedora Server Edition
...
Installed Groups:
   LibreOffice
   GNOME Desktop Environment
   Fonts
...
Available Groups:
   Authoring and Publishing
   Books and Guides
   C Development Tools and Libraries
...
```

Let's say that you wanted to try out a different desktop environment. You see LXDE, and you want to know what is in that group. To find out, use the `groupinfo` subcommand:

```
# yum groupinfo LXDE
Group: LXDE
 Description: LXDE is a lightweight X11 desktop environment...
 Mandatory Packages:
...
   lxde-common
   lxdm
   lxinput
   lxlauncher
   lxmenu-data
...
```

In addition to showing a description of the group, `groupinfo` shows Mandatory Packages (those that are always installed with the group), Default Packages (those that are installed by default but can be excluded), and Optional Packages (which are part of the group but not installed by default). When you use some graphical tools to install package groups, you can uncheck default packages or check optional packages to change whether they are installed with the group.

If you decide that you want to install a package group, use the `groupinstall` subcommand:

```
# yum groupinstall LXDE
```

This groupinstall resulted in 101 packages from the group being installed and 5 existing packages being updated. If you decide that you don't like the group of packages, you can remove the entire group at once using the groupremove subcommand:

```
# yum groupremove LXDE
```

Maintaining your RPM package database and cache

Several subcommands to yum can help you do maintenance tasks, such as checking for problems with your RPM database or clearing out the cache. The YUM facility has tools for maintaining your RPM packages and keeping your system's software efficient and secure.

Clearing out the cache is something that you want to do from time to time. If you decide to keep downloaded packages after they are installed (they are removed by default, based on the keepcache=0 setting in the /etc/yum.conf file), your cache directories (under /var/cache/yum) can fill up. Metadata stored in cache directories can be cleared, causing fresh metadata to be downloaded from all enabled YUM repositories the next time yum is run. Here are ways to clear that information:

```
# yum clean packages
14 files removed
# yum clean metadata
Cache was expired
16 files removed
# yum clean all
68 files removed
```

Although unlikely, it's possible that the RPM database can become corrupted. This can happen if something unexpected occurs, such as pulling out the power cord when a package is partially installed. You can check the RPM database to look for errors (yum check) or just rebuild the RPM database files. Here's an example of a corrupt RPM database and the rpm command that you can use to fix it:

```
# yum check
error: db5 error(11) from dbenv->open: Resource temporarily
unavailable
error: cannot open Packages index using db5-Resource temporarily
unavailable(11)
error: cannot open Packages database in /var/lib/rpm
Error: Error: rpmdb open failed
# rpm --rebuilddb
# yum check
```

The yum clean examples in the preceding command lines remove cached data from the /var/cache/yum subdirectories. The rpm --rebuilddb example rebuilds the database. The yum check example can check for problems with the local RPM cache and database but notice that we used the rpm command to fix the problem.

In general, the command best suited for working with the local RPM database is the rpm command.

Downloading RPMs from a YUM repository

If you just want to examine a package without actually installing it, you can download that package with the `dnf` command or, in earlier releases, use the `yumdownloader` command. Either case causes the package that you name to be downloaded from the YUM repository and copied to your current directory.

For example, to download the latest version of the Firefox web browser package with `yumdownloader` from the YUM repository to your current directory, type the following:

```
# yumdownloader firefox
firefox-67.0.4-1.fc30.x86_64.rpm    6.1 MB/s |  92 MB     00:14
```

To use the `dnf` command, type this:

```
# dnf download firefox
firefox-60.7.0-1.el8_0.x86_64.rpm   6.1 MB/s |  93 MB     00:15
```

With any downloaded RPM package now sitting in your current directory, you can use a variety of `rpm` commands to query or use that package in different ways (as described in the next section).

Installing, Querying, and Verifying Software with the rpm Command

There is a wealth of information about installed packages in the local RPM database. The `rpm` command contains dozens of options to enable you to find information about each package, such as the files it contains, who created it, when it was installed, how large it is, and many other attributes. Because the database contains fingerprints (md5sums) of every file in every package, it can be queried with RPM to find out if files from any package have been tampered with.

The `rpm` command can still do basic install and upgrade activities, although most people only use `rpm` in that way when there is a package sitting in the local directory, ready to be installed. So, let's get one in our local directory to work with. Type the following to download the latest version of the `zsh` package:

```
# dnf download zsh
zsh-5.5.1-6.el8.x86_64.rpm     3.0 MB/s | 2.9 MB   00:00
```

With the `zsh` package downloaded to your current directory, try some `rpm` commands on it.

Installing and removing packages with rpm

To install a package with the `rpm` command, type the following:

```
# rpm -i zsh-5.5.1-6.el8.x86_64.rpm
```

Notice that the entire package name is given to install with `rpm`, not just the package base name. If an earlier version of `zsh` were installed, you could upgrade the package using `-U`. Often, people use `-h` and `-v` options to get hash signs printed and more verbose output during the upgrade:

```
# rpm -Uhv zsh-5.5.1-6.el8.x86_64.rpm
Verifying...          ########################## [100%]
Preparing...          ########################## [100%]
   1:zsh-5.5.1-6.el8  ########################## [100%]
```

Although an install (`-i`) only installs a package if the package is not already installed, an upgrade (`-U`) installs the package even if it is already installed. A third type of install, called freshen (`-F`), installs a package only if an existing, earlier version of a package is installed on the computer, as in this example:

```
# rpm -Fhv *.rpm
```

You could use the previous freshen command if you were in a directory containing thousands of RPMs but only wanted to update those that were already installed (in an earlier version) on your system and skip those that were not yet installed.

You can add a few interesting options to any of your install options. The `--replacepkgs` option enables you to reinstall an existing version of a package (if, for example, you had mistakenly deleted some components), and the `--oldpackage` enables you to replace a newer package with an earlier version.

```
# rpm -Uhv --replacepkgs emacs-26.1-5.el8.x86_64.rpm
# rpm -Uhv --oldpackage zsh-5.0.2-25.el7_3.1.x86_64.rpm
```

You can remove a package with the `-e` option. You only need the base name of a package to remove it. Here's an example:

```
# rpm -e emacs
```

The `rpm -e emacs` command would be successful because no other packages are dependent on `emacs`. However, it would leave behind `emacs-common`, which was installed as a dependency to `emacs`. If you had tried to remove `emacs-common` first, that command would fail with a "Failed dependencies" message.

Querying rpm information

After the package is installed, you can query for information about it. Using the `-q` option, you can see information about the package including a description `(-qi)`, list of files `(-ql)`, documentation `(-qd)`, and configuration files `(-qc)`.

```
# rpm -qi zsh
Name        : zsh
Version     : 5.5.1
Release     : 6.el8
...
```

```
# rpm -ql zsh
/etc/skel/.zshrc
/etc/zlogin
/etc/zlogout
...
# rpm -qd zsh
/usr/share/doc/zsh/BUGS
/usr/share/doc/zsh/CONTRIBUTORS
/usr/share/doc/zsh/FAQ
...
# rpm -qc zsh
/etc/skel/.zshrc
/etc/zlogin
/etc/zlogout
...
```

You can use options to query any piece of information contained in an RPM. You can find what an RPM needs for it to be installed (`--requires`), what version of software a package provides (`--provides`), what scripts are run before and after an RPM is installed or removed (`--scripts`), and what changes have been made to an RPM (`--changelog`).

```
# rpm -q --requires emacs-common
/bin/sh
/usr/bin/pkg-config
/usr/sbin/alternatives
...
# rpm -q --provides emacs-common
config(emacs-common) = 1:26.2-1.fc30
emacs-common = 1:26.2-1.fc30
emacs-common(x86-64) = 1:26.2-1.fc30
emacs-el = 1:26.2-1.fc30
pkgconfig(emacs) = 1:26.2
# rpm -q --scripts httpd
postinstall scriptlet (using /bin/sh):
if [ $1 -eq 1 ] ; then
        # Initial installation
        systemctl --no-reload preset httpd.service...
...
# rpm -q --changelog httpd | less
* Thu May 02 2019 Lubos Uhliarik <luhliari@redhat.com> - 2.4.39-4
- httpd dependency on initscripts is unspecified (#1705188)
* Tue Apr 09 2019 Joe Orton <jorton@redhat.com> - 2.4.39-3
...
```

In the previous two examples, you can see that scripts inside the `httpd` package uses `systemctl` command to set up the `httpd` service. The `--changelog` option enables you to see why changes have been made to each version of the package.

Using a feature called `--queryformat`, you can query different tags of information and output them in any form you like. Run the `--querytags` option to be able to see all of the tags that are available:

```
# rpm --querytags | less
ARCH
ARCHIVESIZE
BASENAMES
BUGURL
...
# rpm -q binutils --queryformat "The package is %{NAME} \
     and the release is %{RELEASE}\n"
The package is binutils and the release is 29.fc30
```

All of the queries that you have done so far have been to the local RPM database. By adding a `-p` to those query options, you can query an RPM file sitting in your local directory instead. The `-p` option is a great way to look inside a package that someone gives you to investigate what it is before you install it on your system.

If you haven't already, get the `zsh` package and put it in your local directory (`dnf download zsh`). Then run some query commands on the RPM file.

```
# rpm -qip zsh-5.7.1-1.fc30.x86_64.rpm View info about the RPM file
# rpm -qlp zsh-5.7.1-1.fc30.x86_64.rpm List all files in the RPM file
# rpm -qdp zsh-5.7.1-1.fc30.x86_64.rpm Show docs in the RPM file
# rpm -qcp zsh-5.7.1-1.fc30.x86_64.rpm List config files in the RPM file
```

Verifying RPM packages

Using the `-V` option, you can check the packages installed on your system to see if the components have been changed since the packages were first installed. Although it is normal for configuration files to change over time, it is not normal for binaries (the commands in `/bin`, `/sbin`, and so on) to change after installation. Binaries that are changed are probably an indication that your system has been cracked.

In this example, I'm going to install the `zsh` package and mess it up. If you want to try along with the examples, be sure to remove or reinstall the package when you are finished.

```
# rpm -i zsh-5.7.1-1.fc30.x86_64.rpm
# echo hello > /bin/zsh
# rm /etc/zshrc
# rpm -V zsh
missing   c  /etc/zshrc
S.5....T.    /bin/zsh
```

In this output, you can see that the `/bin/zsh` file has been tampered with and `/etc/zshrc` has been removed. Each time that you see a letter or a number instead of a dot from

the `rpm -V` output, it is an indication of what has changed. Letters that can replace the dots (in order) include the following:

```
S    file Size differs
M    Mode differs (includes permissions and file type)
5    MD5 sum differs
D    Device major/minor number mismatch
L    readLink(2) path mismatch
U    User ownership differs
G    Group ownership differs
T    mTime differs
P    caPabilities differ
```

Those indicators are from the Verify section of the `rpm` man page. In my example, you can see the file size has changed (`S`), the md5sum checked against the file's fingerprint has changed (`5`), and the modification time (`T`) on the file differs.

To restore the package to its original state, use `rpm` with the `--replacepkgs` option, as shown next. (The `yum reinstall zsh` command would work as well.) Then check it with `-V` again. No output from `-V` means that every file is back to its original state.

```
# rpm -i --replacepkgs zsh-5.7.1-1.fc30.x86_64.rpm
# rpm -V zsh
```

It is good practice to back up your RPM database (from `/var/lib/rpm`) and copy it to some read-only medium (such as a CD). Then, when you go to verify packages that you suspect were cracked, you know that you aren't checking it against a database that has also been cracked.

Managing Software in the Enterprise

At this point, you should have a good working knowledge of how to install, query, remove, and otherwise manipulate packages with graphical tools, the `yum` command, and the `rpm` command. When you start working with RPM files in a large enterprise, you need to extend that knowledge.

Features used to manage RPM packages in the enterprise with Red Hat Enterprise Linux offer a bit more complexity and much more power. Instead of having one big software repository, as Fedora does, RHEL provides software using Red Hat Subscription Management, requiring paid subscriptions/entitlements that allow access to a variety of software channels and products (RHEL, Red Hat Virtualization, Red Hat Software Collections, and so on).

In terms of enterprise computing, one of the great benefits of the design of RPM packages is that their management can be automated. Other Linux packaging schemes allow packages to stop and prompt you for information when they are being installed (such as asking for a

directory location or a username). RPM packages install without interruption, offering some of the following advantages:

Kickstart files All of the questions that you answer during a manual install, and all of the packages that you select, can be added into a file called a *kickstart file*. When you start a Fedora or Red Hat Enterprise Linux installer, you can provide a kickstart file at the boot prompt. From that point on, the entire installation process completes on its own. Any modifications to the default package installs can be made by running pre and post scripts from the kickstart file to do such things as add user accounts or modify configuration files.

PXE boot You can configure a PXE server to allow client computers to boot an anaconda (installer) kernel and a select kickstart file. A completely blank computer with a network interface card (NIC) that supports PXE booting can simply boot from its NIC to launch a fresh installation. In other words, turn on the computer, and if it hits the NIC in its boot order, a few minutes later you can have a freshly installed system, configured to your exact specifications without intervention.

Satellite server (Spacewalk) Red Hat Enterprise Linux systems can be deployed using what is referred to as *Satellite Server*. Built into Satellite Server are the same features that you have from Red Hat CDN to manage and deploy new systems and updates. RHEL systems can be configured to get automatic software updates at times set from the satellite server. Sets of packages called Errata that fix specific problems can be quickly and automatically deployed to the systems that need them.

Container Images Instead of installing individual RPMs on a system, you can package up a few or a few hundred RPMs into a container image. The container image is like an RPM in that it holds a set of software, but unlike an RPM in that it is more easily added to a system, run directly, and deleted from a system than an RPM is.

Descriptions of how to use kickstart files, Satellite Servers, containers and other enterprise-ready installation features are beyond the scope of this book. But the understanding you have gained from learning about YUM and RPM remain critical components of all of the features just mentioned.

Summary

Software packaging in Fedora, Red Hat Enterprise Linux, and related systems is provided using software packages based on the RPM Package Manager (RPM) tools. Debian, Ubuntu, and related systems package software into DEB files. You can try easy-to-use graphical tools such as the Software window for finding and installing packages. The primary command-line tools include the `yum`, `dnf`, and `rpm` commands for Red Hat–related systems and `aptitude`, `apt*` and `dpkg` for Debian-related systems.

Using these software management tools, you can install, query, verify, update, and remove packages. You can also do maintenance tasks, such as clean out cache files and rebuild the

RPM database. This chapter describes many of the features of the Software window, as well as `yum`, `dnf`, and `rpm` commands.

With your system installed and the software packages that you need added, it's time to configure your Fedora, RHEL, Debian, or Ubuntu system further. If you expect to have multiple people using your system, your next task could be to add and otherwise manage user accounts on your system. Chapter 11, "Managing User Accounts," describes user management in Fedora, RHEL, and other Linux systems.

Exercises

These exercises test your knowledge of working with RPM software packages in Fedora or Red Hat Enterprise Linux. To do the exercises, I recommend that you have a Fedora system in front of you that has an Internet connection. (Most of the procedures work equally well on a registered RHEL system.)

You need to be able to reach the Fedora repositories (which should be set up automatically). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Search the YUM repository for the package that provides the `mogrify` command.
2. Display information about the package that provides the `mogrify` command, and determine what is that package's home page (URL).
3. Install the package containing the `mogrify` command.
4. List all of the documentation files contained in the package that provides the `mogrify` command.
5. Look through the `changelog` of the package that provides the `mogrify` command.
6. Delete the `mogrify` command from your system and verify its package against the RPM database to see that the command is indeed missing.
7. Reinstall the package that provides the `mogrify` command, and make sure that the entire package is intact again.
8. Download the package that provides the `mogrify` command to your current directory.
9. Display general information about the package you just downloaded by querying the package's RPM file in the current directory.
10. Remove the package containing the `mogrify` command from your system.

CHAPTER 11

Managing User Accounts

IN THIS CHAPTER

Working with user accounts

Working with group accounts

Configuring centralized user accounts

Adding and managing users are common tasks for Linux system administrators. *User accounts* keep boundaries between the people who use your systems and between the processes that run on your systems. *Groups* are a way of assigning rights to your system that can be assigned to multiple users at once.

This chapter describes not only how to create a new user, but also how to create predefined settings and files to configure the user's environment. Using tools such as the `useradd` and `usermod` commands, you can assign settings such as the location of a home directory, a default shell, a default group, and specific user ID and group ID values. With Cockpit, you can add and manage user accounts through a web UI.

Creating User Accounts

Every person who uses your Linux system should have a separate user account. Having a user account provides you with an area in which to store files securely as well as a means of tailoring your user interface (GUI, path, environment variables, and so on) to suit the way that you use the computer.

You can add user accounts to most Linux systems in several ways. Fedora and Red Hat Enterprise Linux systems offer *Cockpit*, which includes an Account selection for creating and managing user accounts. If Cockpit is not yet installed and enabled, do that as follows:

```
# yum install cockpit -y
# systemctl enable --now cockpit.socket
```


Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

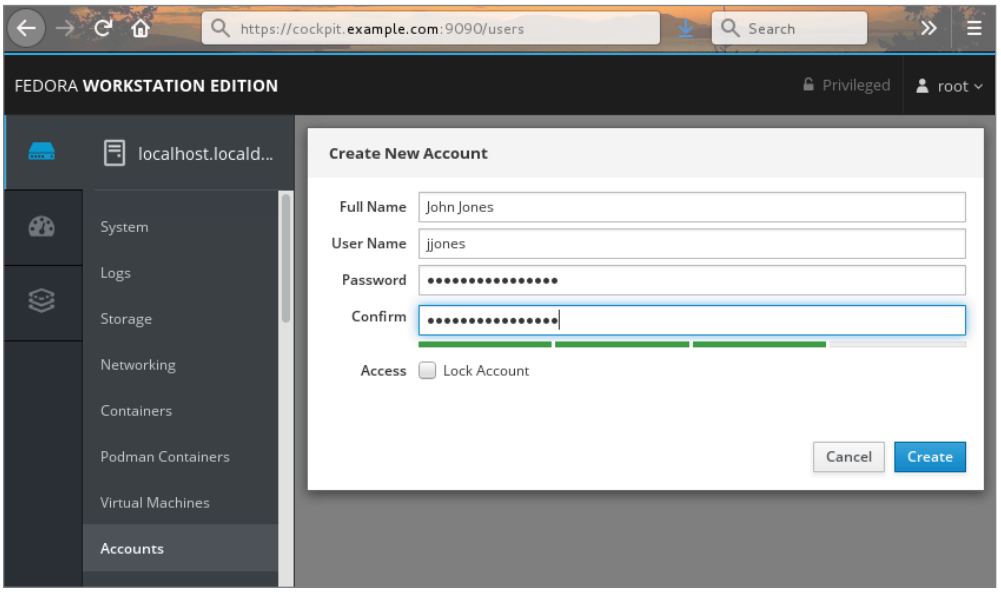
To create a user account through Cockpit, do the following:

1. Open the Cockpit interface from your web browser (`hostname:9090`).
2. Log in as root (or as a user with root privilege), select the "Reuse my password for privileged tasks" check box, and select Accounts.
3. Select Create New Account.

   Figure 11.1 shows an example of the Create New Account pop-up window:

FIGURE 11.1

Add and modify user accounts from Cockpit.



4. Begin adding a new user account to your Linux system. Here are the fields you need to fill in:

   Full Name Use the user's real name, typically used with uppercase and lowercase letters, as the user would write it in real life. Technically, this information is stored in the comment field of the `passwd` file, but by convention, most Linux and UNIX systems expect this field to hold each user's full name.

   User Name This is the name used to log in as this user. When you choose a username, don't begin with a number (for example, 26jsmith). Also, it's best to use all lowercase letters, no control characters or spaces, and a maximum of eight characters, by convention. The `useradd` command allows up to 32 characters,

but some applications can't deal with usernames that long. Tools such as `ps` display user IDs (UIDs) instead of names if names are too long. Having users named Jsmith and jsmith can cause confusion with programs (such as sendmail) that don't distinguish case.

Password, Confirm Enter the password you want the user to have in the Password and Confirm fields. The password should be at least eight characters long and contain a mixture of uppercase and lowercase letters, numbers, and punctuation. It should not contain real words, repeated letters, or letters in a row on the keyboard. Through this interface, you must set a password that meets the above criteria. (If you want to add a password that doesn't meet this criteria, you can use the `useradd` command, described later in this chapter.)Bars underneath the password fields turn from red to green as you improve the strength of your password.

Access To create an account that you are not quite ready to use, select the Lock Account check box. That prevents anyone from logging into the account until you uncheck that box or change that information in the `passwd` file.

5. Select Create to add the user to the system. An entry for the new user account is added to the `/etc/passwd` file and the new group account to the `/etc/group` file. (I will describe those later in this chapter.)

The Cockpit Accounts screen lets you modify a small set of information about a regular user after it has been created. To modify user information later, do the following:

1. Select the user account that you want to change. A screen appears with available selections for that user account.
2. You can delete but not modify the username, but you can change the following: information:

Full Name Because the user's full name is just a comment, you can change that as you please.

Roles By default, you have the opportunity to select check boxes that allow the user to be added to the role of Server Administrator (giving the user root privilege by being added to the wheel group) or Image Builder (allowing the user to build containers and other image types through the weldr group). Other roles might be added to this list by other Cockpit components. If the user is logged in, that user must log out to obtain those privileges.

Access You can choose Lock Account to lock the account, select to lock the account on a particular date, or never lock the account (setting no account expiration date).

Password You can choose Set Password to set a new password for that user or Force Change to force the user to change their password the next time they log in. By default, passwords never expire. You can change that to have the password expire every set number of days.

Authorized Public SSH Keys If you have a public SSH key for the user, you can select the plus sign (+) for this field, paste that key into the text box, and select Add key. With that key in place, the user with the associated private key is allowed to log into that user account via SSH without needing to enter a password.

3. Changes take effect immediately, so you can simply leave the window when you are done modifying the user account.

The Accounts area of the Cockpit web UI was designed to simplify the process of creating and modifying user accounts. More features associated with user accounts can be added or modified from the command line. The next part of this chapter describes how to add user accounts from the command line with `useradd` and change them with the `usermod` command.

Adding users with useradd

Sometimes, a Linux system doesn't have a desktop tool or web UI available for adding users. Other times, you might find it more convenient to add lots of users at once with a shell script or change user account features that are not available from Cockpit. For those cases, commands are available to enable you to add and modify user accounts from the command line.

The most straightforward method for creating a new user from the shell is the `useradd` command. After opening a Terminal window with root permission, you simply invoke `useradd` at the command prompt, with details of the new account as parameters.

The only required parameter is the login name of the user, but you probably want to include some additional information ahead of it. Each item of account information is preceded by a single-letter option code with a dash in front of it. The following options are available with `useradd`:

- **`-c "`*`comment`*`"`**: Provide a description of the new user account. Typically, this is the person's full name. Replace *`comment`* with the name of the user account (`-c Jake`). Use quotes to enter multiple words (for example,`-c "Jake Jackson"`).
- **`-d`** ***`home_dir`***: Set the home directory to use for the account. The default is to name it the same as the login name and to place it in `/home`. Replace *`home_dir`* with the directory name to use (for example, `-d /mnt/homes/jake`).
- **`-D`**: Rather than create a new account, save the supplied information as the new default settings for any new accounts that are created.
- **`-e`** ***`expire_date`***: Assign the expiration date for the account in *YYYY-MM-DD* format. Replace *`expire_date`* with a date that you want to use. (For example, to expire an account on May 5, 2022, use `-e 2022-05-05`.)
- **`-f -1`**: Set the number of days after a password expires until the account is permanently disabled. The default, `-1`, disables the option. Setting this to `0` disables the

account immediately after the password has expired. Replace `-1` (that's minus one) with the number to use.

-g *group*: Set the primary group (it must already exist in the `/etc/group` file) the new user will be in. Replace *group* with the group name (for example, `-g wheel`). Without this option, a new group is created that is the same as the username and is used as that user's primary group.

-G *grouplist*: Add the new user to the supplied comma-separated list of supplementary groups (for example, `-G wheel,sales,tech,lunch`). (If you use `-G` later with `usermod`, be sure to use `-aG` and not just `-G`. If you don't, existing supplementary groups are removed and the groups you provide here are the only ones assigned.)

-k *skel_dir*: Set the skeleton directory containing initial configuration files and login scripts that should be copied to a new user's home directory. This parameter can be used only in conjunction with the `-m` option. Replace *skel_dir* with the directory name to use. (Without this option, the `/etc/skel` directory is used.)

-m: Automatically create the user's home directory and copy the files in the skeleton directory (`/etc/skel`) to it. (This is the default action for Fedora and RHEL, so it's not required. It is not the default for Ubuntu.)

-M: Do not create the new user's home directory, even if the default behavior is set to create it.

-n: Turn off the default behavior of creating a new group that matches the name and user ID of the new user. This option is available with Fedora and RHEL systems. Other Linux systems often assign a new user to the group named `users` instead.

-o: Use with `-u uid` to create a user account that has the same UID as another username. (This effectively lets you have two different usernames with authority over the same set of files and directories.)

-p *passwd*: Enter a password for the account you are adding. This must be an encrypted password. Instead of adding an encrypted password here, you can simply use the `passwd` *user* command later to add a password for *user*. (To generate an encrypted MD5 password, type `openssl passwd`.)

-s *shell*: Specify the command shell to use for this account. Replace *shell* with the command shell (for example, `-s /bin/csh`).

-u *user_id*: Specify the user ID number for the account (for example, `-u 1793`). Without the `-u` option, the default behavior is to assign the next available number automatically. Replace `user_id` with the ID number. User IDs that are automatically assigned to regular users begin at 1000, so you should use IDs for regular users that are above that number in a way that doesn't collide with the automatic assignments.

Let's create an account for a new user. The user's full name is Sara Green, with a login name of sara. To begin, become root user and type the following command:

```
# useradd -c "Sara Green" sara
```

Next, set the initial password for sara using the passwd command. You're prompted to type the password twice:

```
# passwd sara
Changing password for user sara.
New password: **********
Retype new password: **********
```

NOTE

Asterisks in this example represent the password you type. Nothing is actually displayed when you type the password. Also, keep in mind that running passwd as root user lets you add short or blank passwords that regular users cannot add themselves.

In creating the account for sara, the useradd command performs several actions:

- Reads the /etc/login.defs and /etc/default/useradd files to get default values to use when creating accounts.
- Checks command-line parameters to find out which default values to override.
- Creates a new user entry in the /etc/passwd and /etc/shadow files based on the default values and command-line parameters.
- Creates any new group entries in the /etc/group file. (Fedora creates a group using the new user's name.)
- Creates a home directory based on the user's name in the /home directory.
- Copies any files located within the /etc/skel directory to the new home directory. This usually includes login and application startup scripts.

The preceding example uses only a few of the available useradd options. Most account settings are assigned using default values. You can set more values explicitly if you want to. Here's an example that uses a few more options to do so:

```
# useradd -g users -G wheel,apache -s /bin/tcsh -c "Sara Green" sara
```

In this case, useradd is told to make users the primary group sara belongs to (-g), add her to the wheel and apache groups, and assign tcsh as her primary command shell (-s). A home directory in /home under the user's name (/home/sara) is created by default. This command line results in a line similar to the following being added to the /etc/passwd file:

```
sara:x:1002:1007:Sara Green:/home/sara:/bin/tcsh
```

Each line in the /etc/passwd file represents a single user account record. Each field is separated from the next by a colon (:) character. The field's position in the sequence determines what it is. As you can see, the login name is first. The password field contains an x because, in this example, the shadow password file is used to store encrypted password data (in /etc/shadow).

The user ID selected by useradd is 1002. The primary group ID is 1007, which corresponds to a private sara group in the /etc/group file. The comment field was correctly set to Sara Green, the home directory was automatically assigned as /home/sara, and the command shell was assigned as /bin/tcsh, exactly as specified with the useradd options.

By leaving out many of the options (as I did in the first useradd example), defaults are assigned in most cases. For example, by not using -g sales or -G wheel,apache, the group name sara was assigned to the new user. Some Linux systems (other than Fedora and RHEL) assign users as the group name by default. Likewise, excluding -s /bin/tcsh causes /bin/bash to be assigned as the default shell.

The /etc/group file holds information about the different groups on your Linux system and the users who belong to them. Groups are useful for enabling multiple users to share access to the same files while denying access to others. Here is the /etc/group entry created for sara:

```
sara:x:1007:
```

Each line in the group file contains the name of a group, a group password (usually filled with an x), the group ID number associated with it, and a list of users in that group. By default, each user is added to their own group, beginning with the next available GID, starting with 1000.

Setting user defaults

The useradd command determines the default values for new accounts by reading the /etc/login.defs and /etc/default/useradd files. You can modify those defaults by editing the files manually with a standard text editor. Although login.defs is different on different Linux systems, the following is an example containing many of the settings that you might find in a login.defs file:

```
PASS_MAX_DAYS     99999
PASS_MIN_DAYS     0
PASS_MIN_LEN      5
PASS_WARN_AGE     7
UID_MIN                  1000
UID_MAX                 60000
SYS_UID_MIN               200
SYS_UID_MAX               999
GID_MIN                  1000
```

```
GID_MAX                 60000
SYS_GID_MIN               201
SYS_GID_MAX               999
CREATE_HOME yes
```

All uncommented lines contain keyword/value pairs. For example, the keyword `PASS_MIN_LEN` is followed by some white space and the value `5`. This tells `useradd` that the user password must be at least five characters. Other lines enable you to customize the valid range of automatically assigned user ID numbers or group ID numbers. (Fedora starts at UID 1000; earlier systems started with UID 100.) Permanent administrative user and group account numbers are reserved for up to 199 and 200, respectively. So, you can assign your own administrative user and group accounts starting at 200 and 201, respectively, up to ID number 999.

A comment section that explains the keyword's purpose precedes each keyword (which I edited out here to save space). Altering a default value is as simple as editing the value associated with a keyword and saving the file before running the `useradd` command.

If you want to view other default settings, find them in the `/etc/default/useradd` file. You can also see default settings by typing the `useradd` command with the `-D` option, as follows:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

You can also use the `-D` option to change defaults. When run with this flag, `useradd` refrains from actually creating a new user account; instead, it saves any additionally supplied options as the new default values in `/etc/default/useradd`. Not all `useradd` options can be used in conjunction with the `-D` option. You can use only the five options listed here.

- **`-b` *`default_home`***: Set the default directory in which user home directories are created. Replace *`default_home`* with the directory name to use (for example, `-b /garage`). Usually, this is `/home`.
- **`-e` *`default_expire_date`***: Set the default expiration date on which the user account is disabled. The *`default_expire_date`* value should be replaced with a date in the form YYYY-MM-DD (for example, `-e 2011-10-17`).
- **`-f` *`default_inactive`***: Set the number of days after a password has expired before the account is disabled. Replace *`default_inactive`* with a number representing the number of days (for example, `-f 7`).

- **-g *default_group***: Set the default group in which new users will be placed. Normally, `useradd` creates a new group with the same name and ID number as the user. Replace *default_group* with the group name to use (for example, `-g bears`).
- **-s *default_shell***: Set the default shell for new users. This is `/bin/bash`, typically. Replace *default_shell* with the full path to the shell that you want as the default for new users (for example, `-s /bin/ash`).

To set any of the defaults, give the `-D` option first and add the defaults that you want to set. For example, to set the default home directory location to `/home/everyone` and the default shell to `/bin/tcsh`, enter the following:

```
# useradd -D -b /home/everyone -s /bin/tcsh
```

In addition to setting up user defaults, an administrator can create default files that are copied to each user's home directory for use. These files can include login scripts and shell configuration files (such as `.bashrc`). Keep in mind that setting up these kinds of files is the purpose of the default `/etc/skel` directory.

Other commands that are useful for working with user accounts include `usermod` (to modify settings for an existing account) and `userdel` (to delete an existing user account).

Modifying users with usermod

The `usermod` command provides a simple and straightforward method for changing account parameters. Many of the options available with it mirror those found in `useradd`. The options that can be used with this command include the following:

- **-c *username***: Change the description associated with the user account. Replace *username* with the name of the user account (`-c jake`). Use quotes to enter multiple words (for example, `-c "Jake Jackson"`).
- **-d *home_dir***: Change the home directory to use for the account. The default is to name it the same as the login name and to place it in `/home`. Replace *home_dir* with the directory name to use (for example, `-d /mnt/homes/jake`).
- **-e *expire_date***: Assign a new expiration date for the account in *YYYY-MM-DD* format. Replace *expire_date* with a date you want to use. (For October 15, 2022, use `-e 2022-10-15`.)
- **-f -1**: Change the number of days after a password expires until the account is permanently disabled. The default, `-1`, disables the option. Setting this to `0` disables the account immediately after the password has expired. Replace `-1` with the number to use.
- **-g *group***: Change the primary group (as listed in the `/etc/group` file) the user will be in. Replace *group* with the group name (for example, `-g wheel`).

- **-G *grouplist***: Set the user's secondary groups to the supplied comma-separated list of groups. If the user is already in at least one group besides the user's private group, you must add the `-a` option as well (`-Ga`). If not, the user belongs to only the new set of groups and loses membership to any previous groups.
- **-l *login_name***: Change the login name of the account.
- **-L**: Lock the account by putting an exclamation point at the beginning of the encrypted password in `/etc/shadow`. This locks the account while still allowing you to leave the password intact (the `-U` option unlocks it).
- **-m**: Available only when `–d` is used. This causes the contents of the user's home directory to be copied to the new directory.
- **-o**: Use only with `-u uid` to remove the restriction that UIDs must be unique.
- **-s *shell***: Specify a different command shell to use for this account. Replace *shell* with the command shell (for example, `-s bash`).
- **-u *user_id***: Change the user ID number for the account. Replace *user_id* with the ID number (for example, `-u 1474`).
- **-U**: Unlocks the user account (by removing the exclamation mark at the beginning of the encrypted password).

The following are examples of the `usermod` command:

```
# usermod -s /bin/csh chris
# usermod -Ga sales,marketing, chris
```

The first example changes the shell to the `csh` shell for the user named `chris`. In the second example, supplementary groups are added for the user `chris`. The `-a` option (`-Ga`) makes sure that the supplementary groups are added to any existing groups for the user `chris`. If the `-a` is not used, existing supplementary groups for `chris` are erased and the new list of groups includes the only supplementary groups assigned to that user.

Deleting users with userdel

Just as `usermod` is used to modify user settings and `useradd` is used to create users, `userdel` is used to remove users. The following command removes the user `chris`:

```
# userdel -r chris
```

Here, the user `chris` is removed from the `/etc/password` file. The `–r` option removes the user's home directory as well. If you choose not to use `–r`, as follows, the home directory for `chris` is not removed:

```
# userdel chris
```

Keep in mind that simply removing the user account does not change anything about the files that user leaves around the system (except those that are deleted when you use `-r`). However, ownership of files left behind appear as belonging to the previous owner's user ID number when you run `ls -l` on the files.

Before you delete the user, you may want to run a `find` command to find all files that would be left behind by the user. After you delete the user, you could search on user ID to find files left behind. Here are two `find` commands to do those things:

```
# find / -user chris -ls
# find / -uid 504 -ls
```

Because files that are not assigned to any username are considered to be a security risk, it is a good idea to find those files and assign them to a real user account. Here's an example of a `find` command that finds all files in the filesystem that are not associated with any user (the files are listed by UID):

```
# find / -nouser -ls
```

Understanding Group Accounts

Group accounts are useful if you want to share a set of files with multiple users. You can create a group and change the set of files to be associated with that group. The root user can assign users to that group so they can have access to files based on that group's permission. Consider the following file and directory:

```
$ ls -ld /var/salesdocs /var/salesdocs/file.txt
drwxrwxr-x. 2 root sales 4096 Jan 14 09:32 /var/salesstuff/
-rw-rw-r--. 1 root sales    0 Jan 14 09:32 /var/salesstuff/file.txt
```

Looking at permissions on the directory `/var/salesdocs` (`rwxrwxr-x`), you see the second set of `rwx` shows that any member of the group (`sales`) has permission to read files in that directory (`r` is read), create and delete files from that directory (`w` is write), and change to that directory (`x` is execute). The file named `file.txt` can be read and changed by members of the sales group (based on the second `rw-`).

Using group accounts

Every user is assigned to a primary group. In Fedora and RHEL, by default, that group is a new group with the same name as the user. So, if the user were named `sara`, the group assigned to her would also be `sara`. The primary group is indicated by the number in the third field of each entry in the `/etc/passwd` file; for example, the group ID 1007 here:

```
sara:x:1002:1007:Sara Green:/home/sara:/bin/tcsh
```

That entry points to an entry in the `/etc/group` file:

```
sara:x:1007:
```

Let's turn to the `sara` user and group accounts for examples. Here are a few facts about using groups:

- When `sara` creates a file or directory, by default, that file or directory is assigned to `sara`'s primary group (also called `sara`).

- The user sara can belong to zero or more supplementary groups. If sara were a member of groups named sales and marketing, those entries could look like the following in the /etc/group file:

```
sales:x:1302:joe,bill,sally,sara
marketing:x:1303:mike,terry,sara
```

- The user sara can't add herself to a supplementary group. She can't even add another user to her sara group. Only someone with root privilege can assign users to groups.
- Any file assigned to the sales or marketing group is accessible to sara with group and other permissions (whichever provides the most access). If sara wants to create a file with the sales or marketing groups assigned to it, she could use the newgrp command. In this example, sara uses the newgrp command to have sales become her primary group temporarily and creates a file:

```
[sara]$ touch file1
[sara]$ newgrp sales
[sara]$ touch file2
[sara]$ ls -l file*
-rw-rw-r--. 1 sara sara  0 Jan 18 22:22 file1
-rw-rw-r--. 1 sara sales 0 Jan 18 22:23 file2
[sara]$ exit
```

It is also possible to allow users to become a member of a group temporarily with the newgrp command without actually being a member of that group. To do that, someone with root permission can use gpasswd to set a group password (such as gpasswd sales). After that, any user can type newgrp sales into a shell and temporarily use sales as their primary group by simply entering the group password when prompted.

Creating group accounts

As the root user, you can create new groups from the command line with the groupadd command. Also, as noted earlier, groups are created automatically when a user account is created.

Group ID numbers from 0 through 999 are assigned to special administrative groups. For example, the root group is associated with GID 0. Regular groups begin at 1000 for Red Hat Enterprise Linux and Fedora. On the first UNIX systems, GIDs went from 0 to 99. Other Linux systems reserve GIDs between 0 to 500 for administrative groups. A relatively new feature, described earlier, reserves administrative user and group accounts up to 199 and 200, respectively, and lets you create your own administrative accounts between those numbers and 999.

Here are some examples of creating a group account with the `groupadd` command:

```
# groupadd kings
# groupadd -g 1325 jokers
```

In the examples just shown, the group named `kings` is created with the next available group ID. After that, the group `jokers` is created using the `1325` group ID. Some administrators like using an undefined group number above 200 and under 1000 so that the group they create doesn't intrude on the group designations above 1000 (so UID and GID numbers can go along in parallel).

To change a group later, use the `groupmod` command, as in the following example:

```
# groupmod -g 330 jokers
# groupmod -n jacks jokers
```

In the first example, the group ID for `jokers` is changed to 330. In the second, the name `jokers` is changed to `jacks`. If you then wanted to assign any of the groups as supplementary groups to a user, you can use the `usermod` command (as described earlier in this chapter).

Managing Users in the Enterprise

The basic Linux method of handling user and group accounts has not changed since the first UNIX systems were developed decades ago. However, as Linux systems have become used in more complex ways, features for managing users, groups, and the permissions associated with them have been added on to the basic user/group model so that it could be more flexible and more centralized:

More flexible In the basic model, only one user and one group can be assigned to each file. Also, regular users have no ability to assign specific permissions to different users or groups and very little flexibility setting up collaborative files/directories. Enhancements to this model allow regular users to set up special collaborative directories (using features such as sticky bit and set GID bit directories). Using Access Control Lists (ACLs), any user can also assign specific permissions to files and directories to any users and groups they like.

More centralized When you have only one computer, storing user information for all users in the `/etc/passwd` file is probably not a hardship. However, if you need to authenticate the same set of users across thousands of Linux systems, centralizing that information can save lots of time and heartache. Red Hat Enterprise Linux includes features that enable you to authenticate users from LDAP servers or Microsoft Active Directories servers.

The following sections describe how to use features such as Access Control Lists (ACLs) and shared directories (sticky bit and set GID bit directories) to provide powerful ways to share files and directories selectively.

Setting permissions with Access Control Lists

The *Access Control List (ACL)* feature was created so that regular users could share their files and directories selectively with other users and groups. With ACLs, a user can allow others to read, write, and execute files and directories without leaving those filesystem elements wide open or requiring the root user to change the user or group assigned to them.

Here are a few things to know about ACLs:

- For ACLs to be used, they must be enabled on a filesystem when that filesystem is mounted.
- In Fedora and Red Hat Enterprise Linux, ACLs are automatically enabled on any filesystem created when the system is installed.
- If you create a filesystem after installation (such as when you add a hard disk), you need to make sure that the `acl` mount option is used when the filesystem is mounted (more on that later).
- To add ACLs to a file, you use the `setfacl` command; to view ACLs set on a file, you use the `getfacl` command.
- To set ACLs on any file or directory, you must be the actual owner (user) assigned to it. In other words, being assigned user or group permissions with `setfacl` does not give you permission to change ACLs on those files yourself.
- Because multiple users and groups can be assigned to a file/directory, the actual permission a user has is based on a union of all user/group designations to which they belong. For example, if a file had read-only permission (`r--`) for the sales group and read/write/execute (`rwx`) for the market group, and mary belonged to both, mary would have `rwx` permission.

NOTE

If ACLs are not enabled on the filesystem you are trying to use with `setfacl`, see the section "Enabling ACLs" later in this chapter for information on how to mount a filesystem with ACLs enabled.

Setting ACLs with setfacl

Using the `setfacl` command, you can modify permissions (`-m`) or remove ACL permissions (`-x`). The following is an example of the syntax of the `setfacl` command:

```
setfacl -m u:username:rwx filename
```

In the example just shown, the modify option (`-m`) is followed by the letter `u`, indicating that you are setting ACL permissions for a user. After a colon (`:`), you indicate the username, followed by another colon and the permissions that you want to assign. As with the `chmod` command, you can assign read (`r`), write (`w`), and/or execute (`x`) permissions to the user or group (in the example, full `rwx` permission is given). The last argument is replaced by the actual filename you are modifying.

The following are some examples of the user mary using the setfacl command to add permission for other users and groups on a file:

```
[mary]$ touch /tmp/memo.txt
[mary]$ ls -l /tmp/memo.txt
-rw-rw-r--. 1 mary mary 0 Jan 21 09:27 /tmp/memo.txt
[mary]$ setfacl -m u:bill:rw /tmp/memo.txt
[mary]$ setfacl -m g:sales:rw /tmp/memo.txt
```

In the preceding example, mary created a file named /tmp/memo.txt. Using the setfacl command, she modified (-m) permissions for the user named bill so that he now has read/write (rw) permissions to that file. Then she modified permissions for the group sales so that anyone belonging to that group would also have read/write permissions. Look at ls -l and getfacl output on that file now:

```
[mary]$ ls -l /tmp/memo.txt
-rw-rw-r--+ 1 mary mary 0 Jan 21 09:27 /tmp/memo.txt
[mary]$ getfacl /tmp/memo.txt
# file: tmp/memo.txt
# owner: mary
# group: mary
user::rw-
user:bill:rw-
group::rw-
group:sales:rw-
mask::rw-
other::r--
```

From the ls -l output, notice the plus sign (+) in the rw-rw-r--+ output. The plus sign indicates that ACLs are set on the file, so you know to run the getfacl command to see how ACLs are set. The output shows mary as owner and group (same as what you see with ls -l), the regular user permissions (rw-), and permissions for ACL user bill (rw-). The same is true for group permissions and permissions for the group sales. Other permissions are r--.

The mask line (near the end of the previous getfacl example) requires some special discussion. As soon as you set ACLs on a file, the regular group permission on the file sets a mask of the maximum permission an ACL user or group can have on a file. So, even if you provide an individual with more ACL permissions than the group permissions allow, the individual's effective permissions do not exceed the group permissions as in the following example:

```
[mary]$ chmod 644 /tmp/memo.txt
[mary]$ getfacl /tmp/memo.txt
# file: tmp/memo.txt
# owner: mary
# group: mary
user::rw-
user:bill:rw-   #effective:r--
```

```
group::rw-      #effective:r--
group:sales:rw- #effective:r--
mask::r--
other::r--
```

Notice in the preceding example that even though the user `bill` and group `sales` have `rw-` permissions, their effective permissions are `r--`. So, `bill` or anyone in `sales` would not be able to change the file unless `mary` were to open permissions again (for example, by typing `chmod 664 /tmp/memo.txt`).

Setting default ACLs

Setting default ACLs on a directory enables your ACLs to be inherited. This means that when new files and directories are created in that directory, they are assigned the same ACLs. To set a user or group ACL permission as the default, you add a `d:` to the user or group designation. Consider the following example:

```
[mary]$ mkdir /tmp/mary
[mary]$ setfacl -m d:g:market:rwx /tmp/mary/
[mary]$ getfacl /tmp/mary/
# file: tmp/mary/
# owner: mary
# group: mary
user::rwx
group::rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:sales:rwx
default:group:market:rwx
default:mask::rwx
default:other::r-x
```

To make sure that the default ACL worked, create a subdirectory. Then run getfacl again. You will see that default lines are added for user, group, mask, and other, which are inherited from the directory's ACLs:

```
[mary]$ mkdir /tmp/mary/test
[mary]$ getfacl /tmp/mary/test
# file: tmp/mary/test
# owner: mary
# group: mary
user::rwx
group::rwx
group:sales:rwx
group:market:rwx
mask::rwx
other::r-x
```

```
default:user::rwx
default:group::rwx
default:group:sales:rwx
default:group:market:rwx
default:mask::rwx
default:other::r-x
```

Notice that when you create a file in that directory, the inherited permissions are different. Because a regular file is created without execute permission, the effective permission is reduced to `rw-`:

```
[mary@cnegus ~]$ touch /tmp/mary/file.txt
[mary@cnegus ~]$ getfacl /tmp/mary/file.txt
# file: tmp/mary/file.txt
# owner: mary
# group: mary
user::rw-
group::rwx          #effective:rw-
group:sales:rwx     #effective:rw-
group:market:rwx    #effective:rw-
mask::rw-
other::r--
```

Enabling ACLs

In recent Fedora and RHEL systems, xfs and ext filesystem types (ext2, ext3, and ext4) are automatically created with ACL support. On other Linux systems, or on filesystems created on other Linux systems, you can add the `acl` mount option in several ways:

- Add the `acl` option to the fifth field in the line in the `/etc/fstab` file that automatically mounts the filesystem when the system boots up.
- Implant the `acl` line in the `Default mount options` field in the filesystem's super block, so that the `acl` option is used whether the filesystem is mounted automatically or manually.
- Add the `acl` option to the `mount` command line when you mount the filesystem manually with the `mount` command.

Keep in mind that in Fedora and Red Hat Enterprise Linux systems, you only have to add the `acl` mount option to those filesystems that were created elsewhere. The anaconda installer automatically adds ACL support to every filesystem it creates during install time and `mkfs` adds `acl` to every filesystem you create with that tool. To check that the `acl` option has been added to an ext filesystem, determine the device name associated with the filesystem, and run the `tune2fs -l` command to view the implanted mount options, as in this example:

```
# mount | grep home
/dev/mapper/mybox-home on /home type ext4 (rw)
# tune2fs -l /dev/mapper/mybox-home | grep "mount options"
Default mount options:    user_xattr acl
```

First, I typed the `mount` command to see a list of all filesystems that are currently mounted, limiting the output by grepping for the word `home` (because I was looking for the filesystem mounted on `/home`). After I saw the filesystem's device name, I used it as an option to `tune2fs -l` to find the default mount options line. There, I could see that mount options `user _ xattr` (for extended attributes such as SELinux) and `acl` were both implanted in the filesystem super block so that they would be used when the filesystem was mounted.

If the `Default mount options` field is blank (such as when you have just created a new filesystem), you can add the `acl` mount option using the `tune2fs -o` command. For example, on a different Linux system, I created a filesystem on a removable USB drive that was assigned as the `/dev/sdc1` device. To implant the `acl` mount option and check that it is there, I ran the following commands:

```
# tune2fs -o acl /dev/sdc1
# tune2fs -l /dev/sdc1 | grep "mount options"
Default mount options:    acl
```

You can test that this worked by remounting the filesystem and trying to use the `setfacl` command on a file in that filesystem.

A second way to add `acl` support to a filesystem is to add the `acl` option to the line in the `/etc/fstab` file that automatically mounts the filesystem at boot time. The following is an example of what a line would look like that mounts the ext4 filesystem located on the `/dev/sdc1` device to the `/var/stuff` directory:

```
/dev/sdc1      /var/stuff       ext4    acl         1 2
```

Instead of the `defaults` entry in the fourth field, I added `acl`. If there were already options set in that field, add a comma after the last option and add `acl`. The next time the filesystem is mounted, ACLs are enabled. If the filesystem were already mounted, I could type the following `mount` command as root to remount the filesystem using `acl` or any other values added to the `/etc/fstab` file:

```
# mount -o remount /dev/sdc1
```

A third way that you can add ACL support to a filesystem is to mount the filesystem by hand and specifically request the `acl` mount option. So, if there were no entry for the filesystem in the `/etc/fstab` file, after creating the mount point (`/var/stuff`), type the following command to mount the filesystem and include ACL support:

```
# mount -o acl /dev/sdc1 /var/stuff
```

Keep in mind that the `mount` command only mounts the filesystem temporarily. When the system reboots, the filesystem is not mounted again, unless you add an entry to the `/etc/fstab` file.

Adding directories for users to collaborate

A special set of three permission bits are typically ignored when you use the `chmod` command to change permissions on the filesystem. These bits can set special permissions on commands and directories. The focus of this section is setting the bits that help you create directories to use for collaboration.

As with read, write, and execute bits for `user`, `group`, and `other`, these special file permission bits can be set with the `chmod` command. If, for example, you run `chmod 775 /mnt/xyz`, the implied permission is actually `0775`. To change permissions, you can replace the number 0 with any combination of those three bits (4, 2, and 1), or you can use letter values instead. (Refer to Chapter 4, "Moving Around the Filesystem," if you need to be reminded about how permissions work.) The letters and numbers are shown in Table 11.1.

TABLE 11.1 Commands to Create and Use Files

Name	Numeric value	Letter value
Set user ID bit	4	u+s
Set group ID bit	2	g+s
Sticky bit	1	o+t

The bits in which you are interested for creating collaborative directories are the set group ID bit (2) and sticky bit (1). If you are interested in other uses of the set user ID and set group ID bits, refer to the sidebar "Using Set UID and Set GID Bit Commands."

Creating group collaboration directories (set GID bit)

When you create a set GID directory, any files created in that directory are assigned to the group assigned to the directory itself. The idea is to have a directory where all members of a group can share files but still protect them from other users. Here's a set of steps for creating a collaborative directory for all users in the group I created called `sales`:

1. Create a group to use for collaboration:

   ```
   # groupadd -g 301 sales
   ```

2. Add to the group some users with which you want to be able to share files (I used `mary`):

   ```
   # usermod -aG sales mary
   ```

3. Create the collaborative directory:

   ```
   # mkdir /mnt/salestools
   ```

Using Set UID and Set GID Bit Commands

The set UID and set GID bits are used on special executable files that allow commands set to be run differently than most. Normally, when a user runs a command, that command runs with that user's permissions. In other words, if I run the `vi` command as `chris`, that instance of the `vi` command would have the permissions to read and write files that the user `chris` could read and write.

Commands with the set UID or set GID bits set are different. It is the owner and group assigned to the command, respectively, that determines the permissions the command has to access resources on the computer. So, a set UID command owned by root would run with root permissions; a set GID command owned by apache would have apache group permissions.

Examples of applications that have set UID bits turned on are the `su` and `newgrp` commands. In both of those cases, the commands must be able to act as the root user to do their jobs. However, to actually get root permissions, a user must provide a password. You can tell `su` is a set UID bit command because of the `s` where the first execute bit (`x`) usually goes:

```
$ ls /bin/su
-rwsr-xr-x. 1 root root  30092 Jan 30 07:11 su
```

4. Assign the group `sales` to the directory:

```
# chgrp sales /mnt/salestools
```

5. Change the directory permission to 2775. This turns on the set group ID bit (2), full `rwx` for the `user` (7), `rwx` for `group` (7), and `r-x` (5) for `other`:

```
# chmod 2775 /mnt/salestools
```

6. Become `mary` (run `su - mary`). As `mary`, create a file in the shared directory and look at the permissions. When you list permissions, you can see that the directory is a set GID directory because a lowercase `s` appears where the group execute permission should be (`rwxrw`**`s`**`r-x`):

```
# su - mary
[mary]$ touch /mnt/salestools/test.txt
[mary]$ ls -ld /mnt/salestools/ /mnt/salestools/test.txt
drwxrwsr-x. 2 root sales 4096 Jan 22 14:32 /mnt/salestools/
-rw-rw-r--. 1 mary sales    0 Jan 22 14:32 /mnt/salestools/test.txt
```

Typically, a file created by `mary` would have the group `mary` assigned to it. But because `test.txt` was created in a set group ID bit directory, the file is assigned to the `sales` group. Now, anyone who belongs to the `sales` group can read from or write to that file, based on group permissions.

Creating restricted deletion directories (sticky bit)

A *restricted deletion directory* is created by turning on a directory's sticky bit. What makes a restricted deletion directory different than other directories? Normally, if write permission

is open to a user on a file or directory, that user can delete that file or directory. However, in a restricted deletion directory, unless you are the root user or the owner of the directory, you can never delete another user's files.

Typically, a restricted deletion directory is used as a place where lots of different users can create files. For example, the `/tmp` directory is a restricted deletion directory:

```
$ ls -ld /tmp
drwxrwxrwt. 116 root root 36864 Jan 22 14:18 /tmp
```

You can see that the permissions are wide open, but instead of an `x` for the execute bit for `other`, the `t` indicates that the sticky bit is set. The following is an example of creating a restricted deletion directory with a file that is wide open for writing by anyone:

```
[mary]$ mkdir /tmp/mystuff
[mary]$ chmod 1777 /tmp/mystuff
[mary]$ cp /etc/services /tmp/mystuff/
[mary]$ chmod 666 /tmp/mystuff/services
[mary]$ ls -ld /tmp/mystuff /tmp/mystuff/services
drwxrwxrwt. 2 mary mary   4096 Jan 22 15:28 /tmp/mystuff/
-rw-rw-rw-. 1 mary mary 640999 Jan 22 15:28 /tmp/mystuff/services
```

With permissions set to `1777` on the `/tmp/mystuff` directory, you can see that all permissions are wide open, but a `t` appears instead of the last execute bit. With the `/tmp/mystuff/services` file open for writing, any user could open it and change its contents. However, because the file is in a sticky bit directory, only root and `mary` can delete that file.

Centralizing User Accounts

Although the default way of authenticating users in Linux is to check user information against the `/etc/passwd` file and passwords from the `/etc/shadow` file, you can authenticate in other ways as well. In most large enterprises, user account information is stored in a centralized authentication server, so each time you install a new Linux system, instead of adding user accounts to that system, you have the Linux system query the authentication server when someone tries to log in.

As with local `passwd/shadow` authentication, configuring centralized authentication requires that you provide two types of information: account information (username, user/group IDs, home directory, default shell, and so on) and authentication method (different types of encrypted passwords, smart cards, retinal scans, and so on). Linux provides ways of configuring those types of information.

Authentication domains that are supported via the `authconfig` command include LDAP, NIS, and Windows Active Directory.

Supported centralized database types include the following:

LDAP The *Lightweight Directory Access Protocol (LDAP)* is a popular protocol for providing directory services (such as phone books, addresses, and user accounts). It is an open standard that is configured in many types of computing environments.

NIS The *Network Information Service (NIS)* was originally created by Sun Microsystems to propagate information such as user accounts, host configuration, and other types of system information across many UNIX systems. Because NIS passes information in clear text, most enterprises now use the more secure LDAP or Winbind protocols for centralized authentication.

Winbind Selecting *Winbind* from the Authentication Configuration window enables you to authenticate your users against a Microsoft Active Directory (AD) server. Many large companies extend their desktop authentication setup to do server configuration as well as using an AD server.

If you are looking into setting up your own centralized authentication services and you want to use an open-source project, I recommend looking into the 389 Directory Server (`https://directory.fedoraproject.org/`). Fedora and other Linux systems offer this enterprise-quality LDAP server.

Summary

Having separate user accounts is the primary method of setting secure boundaries between the people who use your Linux system. Regular users typically can control the files and directories within their own home directories but very little outside of those directories.

In this chapter, you learned how to add user and group accounts, how to modify them, and even how to extend user and group accounts beyond the boundaries of the local `/etc/passwd` file. You also learned that authentication can be done by accessing centralized LDAP servers.

The next chapter introduces another basic topic needed by Linux system administrators: how to manage disks. In that chapter, you learn how to partition disks, add filesystems, and mount them so the contents of the disk partitions are accessible to those using your system.

Exercises

Use these exercises to test your knowledge of adding and managing user and group accounts in Linux. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, you often have multiple ways to complete a task).

1. Add a local user account to your Linux system that has a username of `jbaxter` and a full name of John Baxter and that uses `/bin/sh` as its default shell. Let the UID be assigned by default. Set the password for `jbaxter` to: `My1N1te0ut!`
2. Create a group account named `testing` that uses group ID 315.
3. Add `jbaxter` to the `testing` group and the bin group.
4. Open a shell as `jbaxter` (either a new login session or using a current shell) and temporarily have the `testing` group be your default group so that when you type `touch /home/jbaxter/file.txt`, the `testing` group is assigned as the file's group.
5. Note what user ID has been assigned to `jbaxter`, and delete the user account without deleting the home directory assigned to `jbaxter`.
6. Find any files in the `/home` directory (and any subdirectories) that are assigned to the user ID that recently belonged to the user named `jbaxter`.
7. Copy the `/etc/services` file to the default skeleton directory so that it shows up in the home directory of any new user. Then add a new user to the system named `mjones`, with a full name of Mary Jones and a home directory of `/home/maryjones`.
8. Find all files under the `/home` directory that belong to `mjones`. Are there any files owned by `mjones` that you didn't expect to see?
9. Log in as `mjones`, and create a file called `/tmp/maryfile.txt`. Using ACLs, assign the `bin` user read/write permission to that file. Then assign the `lp` group read/write permission to that file.
10. Still as `mjones`, create a directory named `/tmp/mydir`. Using ACLs, assign default permissions to that directory so that the `adm` user has read/write/execute permission to that directory and any files or directories created in it. Create the `/tmp/mydir/testing/` directory and `/tmp/mydir/newfile.txt` file, and make sure that the `adm` user was also assigned full read/write/execute permissions. (Note that despite `rwx` permission being assigned to the `adm` user, the effective permission on `newfile.txt` is only `rw`. What could you do to make sure that `adm` gets execute permission as well?)

CHAPTER 12

Managing Disks and Filesystems

IN THIS CHAPTER

Working with shell scripts

Creating logical volumes with LVM

Adding filesystems

Mounting filesystems

Unmounting filesystems

Your operating system, applications, and data all need to be kept on some kind of permanent storage so that when you turn your computer off and then on again, it is all still there. Traditionally, that storage has been provided by a hard disk in your computer. To organize the information on that disk, the disk is usually divided into partitions, with most partitions given a structure referred to as a *filesystem*.

This chapter describes how to work with hard drives. Hard drive tasks include partitioning, adding filesystems, and managing those filesystems in various ways. Storage devices that are attached to the systems such as removable devices, including hard disk drives (HDDs) and solid-state drives (SSDs), and network devices can be partitioned and managed in the same ways.

After covering basic partitions, I describe how Logical Volume Manager (LVM) can be used to make it easier to grow, shrink, and otherwise manage filesystems more efficiently.

Understanding Disk Storage

The basics of how data storage works are the same in most modern operating systems. When you install the operating system, the disk is divided into one or more partitions. Each partition is formatted with a filesystem. In the case of Linux, some of the partitions may be specially formatted for elements such as swap area or LVM physical volumes. Disks are used for permanent storage; *random access memory (RAM)* and swap are used for temporary storage. For example, when you run a command, that command is copied from the hard disk into RAM so that your computer processor (CPU) can access it more quickly.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Your CPU can access data much faster from RAM than it can from a hard disk, although SSDs are more like RAM than HDDs. However, a disk is usually much larger than RAM, RAM is much more expensive, and RAM is erased when the computer reboots. Think of your office as a metaphor for RAM and disk. A disk is like a file cabinet where you store folders of information you need. RAM is like the top of your desk, where you put the folder of papers while you are using it but put it back in the file cabinet when you are not.

If RAM fills up by running too many processes or a process with a memory leak, new processes fail if your system doesn't have a way to extend system memory. That's where a swap area comes in. A *swap space* is a hard disk swap partition or a swap file where your computer can "swap out" data from RAM that isn't being used at the moment and then "swap in" the data back to RAM when it is again needed. Although it is better never to exceed your RAM (performance takes a hit when you swap), swapping out is better than having processes just fail.

Another special partition is a *Logical Volume Manager (LVM)* physical volume. LVM physical volumes enable you to create pools of storage space called *volume groups*. From those volume groups, you have much more flexibility for growing and shrinking logical volumes than you have resizing disk partitions directly.

For Linux, at least one disk partition is required, assigned to the root `(/)` of the entire Linux filesystem. However, it is more common to have separate partitions that are assigned to particular directories, such as `/home`, `/var`, and/or `/tmp`. Each of the partitions is connected to the larger Linux filesystem by mounting it to a point in the filesystem where you want that partition to be used. Any file added to the mount point directory of a partition, or a subdirectory, is stored on that partition.

NOTE

The word *mount* refers to the action of connecting a filesystem from a hard disk, USB drive, or network storage device to a particular point in the filesystem. This action is done using the `mount` command, along with options to tell the command where the storage device is located and to which directory in the filesystem to connect it.

The business of connecting disk partitions to the Linux filesystem is done automatically and invisibly to the end user. How does this happen? Each regular disk partition created when you install Linux is associated with a device name. An entry in the `/etc/fstab` file tells Linux each partition's device name and where to mount it (as well as other bits of information). The mounting is done when the system boots.

Most of this chapter focuses on understanding how your computer's disk is partitioned and connected to form your Linux filesystem as well as how to partition disks, format filesystems and swap space, and have those items used when the system boots. The chapter then covers how to do partitioning and filesystem creation manually.

Coming from Windows

Filesystems are organized differently in Linux than they are in Microsoft Windows operating systems. Instead of drive letters (for example, `A:`, `B:`, `C:`) for each local disk, network filesystem, CD-ROM, or other type of storage medium, everything fits neatly into the Linux directory structure.

Some drives are connected (mounted) automatically into the filesystem when you insert removable media. For example, a CD might be mounted on `/media/cdrom`. If the drive isn't mounted automatically, it is up to an administrator to create a mount point in the filesystem and then connect the disk to that point.

Linux can understand VFAT filesystems, which are often the default format when you buy a USB flash drive. A VFAT and exFAT USB flash drive provides a good way to share data between Linux and Windows systems. Linux kernel support is available for NTFS filesystems, which are usually used with Windows these days. However, NTFS, and sometimes exFAT, often require that you install additional kernel drivers in Linux.

VFAT file systems are often used when files need to be exchanged between different types of operating systems. Because VFAT was used in MS-DOS and early Windows operating systems, it offers a good lowest common denominator for sharing files with many types of systems (including Linux). NTFS is the file system type most commonly used with modern Microsoft Windows systems.

Partitioning Hard Disks

Linux provides several tools for managing your hard disk partitions. You need to know how to partition your disk if you want to add a disk to your system or change your existing disk configuration.

The following sections demonstrate disk partitioning using a removable USB flash drive and a fixed hard disk. To be safe, I use a USB flash drive that doesn't contain any data that I want to keep in order to practice partitioning.

Changing partitioning can make a system unbootable!

I don't recommend using your system's primary hard disk to practice changing partitioning because a mistake can make your system unbootable. Even if you use a separate USB flash drive to practice, a bad entry in `/etc/fstab` can hang your system on reboot. If after changing partitions your system fails to boot, refer to Chapter 21, "Troubleshooting Linux," for information on how to fix the problem.

Understanding partition tables

PC architecture computers have traditionally used *Master Boot Record (MBR) partition tables* to store information about the sizes and layouts of the hard disk partitions. There are

many tools for managing MBR partitions that are quite stable and well known. A few years ago, however, a new standard called *Globally Unique Identifier (GUID) partition tables* began being used on systems as part of the UEFI computer architecture to replace the older BIOS method of booting the system.

Many Linux partitioning tools have been updated to handle GUID partition tables (gpt). Other tools for handling GUID partition tables have been added. Because the popular `fdisk` command does not support gpt partitions, the `parted` command is used to illustrate partitioning in this chapter.

Limitations imposed by the MBR specification brought about the need for GUID partitions. In particular, MBR partitions are limited to 2TB in size. GUID partitions can create partitions up to 9.4ZB (zettabytes).

Viewing disk partitions

To view disk partitions, use the `parted` command with the `-l` option. The following is an example of partitioning on a 160GB fixed hard drive on a Red Hat Enterprise Linux 8 system:

```
# parted -l /dev/sda
Disk /dev/sda: 160.0 GB, 160000000000 bytes, 312500000 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x0008870c
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     1026047      512000   83  Linux
/dev/sda2         1026048   304281599   151627776   8e  Linux LVM
```

When a USB flash drive is inserted, it is assigned to the next available `sd` device. The following example shows the partitioning on the hard drive (`/dev/sda`) and a USB drive from a Fedora 30 system, where `/dev/sdb` is assigned as the USB device name (the second disk on the system). This USB drive is a new 128GB USB flash drive:

```
# fdisk -l /dev/sdb
```

Although this drive was assigned to`/dev/sdb`, your drive might be assigned to a different device name. Here are some things to look for:

- A SCSI or USB storage device, represented by an *sd?* device (such as `sda`, `sdb`, `sdc`, and so on) can have up to 16 minor devices (for example, the main `/dev/sdc` device and `/dev/sdc1` through `/dev/sdc15`). So, there can be 15 partitions total. A NVMe SSD storage device, represented by a `nvme` device (such as `nvme0`, `nvme1`, `nvme2`, and so on) can be divided into one or more namespaces (most devices just use the first namespace) and partitions. For example, `/dev/nvme0n1p1` represents the first partition in the first namespace on the first NVMe SSD.
- For x86 computers, disks can have up to four primary partitions. So, to have more than four total partitions, one must be an extended partition. Any partitions

beyond the four primary partitions are logical partitions that use space from the extended partition.

- The `id` field indicates the type of partition. Notice that there is a Linux LVM partition in the first example.

Your first primary hard disk usually appears as `/dev/sda`. With RHEL and Fedora installations, there is usually at least one LVM partition created by the installer, out of which other partitions can be assigned. So, the output of `fdisk` might be as simple as the following:

```
# parted -l
Disk /dev/sda: 500.1 GB, 500107862016 bytes
```

The first partition is roughly 210MB and is mounted on the `/boot/efi` directory. The second partition (1074MB) is mounted on the `/boot` partition. For older MBR partition tables, there is only a `/boot` partition. The `boot` under the Flags column indicates that the partition is bootable. The rest of the disk is consumed by the LVM partition, which is ultimately used to create logical volumes.

For the moment, I recommend that you leave the hard disk alone and find a USB flash drive that you do not mind erasing. You can try the commands I demonstrate on that drive.

Creating a single-partition disk

To add a new storage medium (hard disk, USB flash drive, or similar device) to your computer so that it can be used by Linux, you first need to connect the disk device to your computer and then partition the disk. Here's the general procedure:

1. Install the new hard drive or insert the new USB flash drive.
2. Partition the new disk.
3. Create the filesystems on the new disk.
4. Mount the filesystems.

The easiest way to add a disk or flash drive to Linux is to have the entire disk devoted to a single Linux partition. You can have multiple partitions, however, and assign them each to different types of filesystems and different mount points if you like.

The following process takes you through partitioning a USB flash drive to be used for Linux that has only one partition. If you have a USB flash drive (any size) that you don't mind erasing, you can work through this procedure as you read. The section following this describes how to partition a disk with multiple partitions.

Warning

If you make a mistake partitioning your disk with `parted`, make sure that you correct that change. Unlike `fdisk`, where you could just type **q** to exit without saving your changes, `parted` makes your changes immediately, so you are not able just to quit to abandon changes.

1. For a USB flash drive, just plug it into an available USB port. Going forward, I use a 128GB USB flash drive, but you can get a USB flash drive of any size.
2. Determine the device name for the USB drive. As root user from a shell, type the following `journalctl` command, and then insert the USB flash drive. Messages appear, indicating the device name of the drive you just plugged in (press Ctrl+C to exit the `tail` command when you are finished):

```
# journalctl -f
kernel: usb 4-1: new SuperSpeed Gen 1 USB device number 3 using
                 xhci_hcd
kernel: usb 4-1: New USB device found, idVendor=0781,
                 idProduct=5581, bcdDevice= 1.00
kernel: usb 4-1: New USB device strings: Mfr=1, Product=2,
                 SerialNumber=3
kernel: usb 4-1: Product: Ultra
kernel: usb 4-1: Manufacturer: SanDisk
...
kernel: sd 6:0:0:0: Attached scsi generic sg2 type 0
kernel:  sdb: sdb1
kernel: sd 6:0:0:0: [sdb] Attached SCSI removable disk
udisksd[809]: Mounted /dev/sdb1 at /run/media/chris/7DEB-B010
              on behalf of uid 1000
```

3. From the output, you can see that the USB flash drive was found and assigned to `/dev/sdb`. (Your device name may be different.) It also contains a single formatted partition: `sdb1`. Be sure you identify the correct disk or you could lose all data from disks you may want to keep!
4. If the USB flash drive mounts automatically, unmount it. Here is how to find the USB partitions in this example and unmount them:

```
# mount | grep sdb
/dev/sdb1 on /run/media...
# umount /dev/sdb1
```

5. Use the `parted` command to create partitions on the USB drive. For example, if you are formatting the second USB, SATA, or SCSI disk (`sdb`), you can type the following:

```
# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

   Now you are in `parted` command mode, where you can use the `parted` single-letter command set to work with your partitions.

6. If you start with a new USB flash drive, it may have one partition that is entirely devoted to a Windows-compatible filesystem (such as VFAT or fat32). Use `p` to view all partitions and `rm` to delete the partition. Here's what it looked like when I did that:

```
(parted) p
Model: SanDisk Ultra (scsi)
Disk /dev/sdb: 123GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End    Size   Type     File system  Flags
1      16.4kB  123GB  123GB  primary  fat32        lba
(parted) rm
Partition number? 1
```

7. Relabel the disk as having a gpt partition table.

```
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdb will be destroyed and all data
on this disk will be lost. Do you want to continue?
Yes/No? Yes
(parted)
```

8. To create a new partition, type **`mkpart`**. You are prompted for the file system type, then the start and end of the partition. This example names the partition `alldisk`, uses `xfs` as the file system type, starts the partition at 1M and ends at 123GB:

```
(parted) mkpart
Partition name?  []? alldisk
File system type?  [ext2]? xfs
Start? 1
End? 123GB
```

9. Double-check that the drive is partitioned the way you want by pressing **`p`**. (Your output will differ, depending on the size of your drive.)

```
(parted) p
Model: SanDisk Ultra (scsi)
Disk /dev/sdb: 123GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size   File system  Name     Flags
 1      1049kB  123GB  123GB  xfs          alldisk
```

10. Although the partitioning is done, the new partition is not yet ready to use. For that, you have to create a filesystem on the new partition. To create a filesystem on the new disk partition, use the `mkfs` command. By default, this command creates an `ext2` filesystem, which is usable by Linux. However, in most cases you want to use a journaling filesystem (such as ext3, ext4, or xfs). To create an xfs filesystem on the first partition of the second hard disk, type the following:

```
# mkfs -t xfs /dev/sdb1
```

TIP

You can use different commands, or options to this command, to create other filesystem types. For example, use `mkfs.exfat` to create a VFAT filesystem, `mkfs.msdos` for DOS, or `mkfs.ext4` for the ext4 filesystem type. You may want a VFAT or exFAT (available with Ubuntu) filesystem if you want to share files among Linux, Windows, and Mac systems.

11. To be able to use the new filesystem, you need to create a mount point and mount it to the partition. Here is an example of how to do that. You then check to make sure that the mount succeeded.

```
# mkdir /mnt/test
# mount /dev/sdb1 /mnt/test
# df -h /mnt/sdb1
Filesystem         Size  Used Avail Use% Mounted on
/dev/sdb1          115G   13M  115G   1% /mnt/test
```

The `df` command shows that `/dev/sdb1` is mounted on `/mnt/test` and that it offers about 115GB of disk space. The `mount` command shows all mounted filesystems, but here I just list `sdb1` to show that it is mounted.

Any files or directories that you create later in the `/mnt/test` directory, and any of its subdirectories, are stored on the `/dev/sdb1` device.

12. When you are finished using the drive, you can unmount it with the `umount` command, after which you can safely remove the drive (see the description of the `umount` command later if this command fails):

```
# umount /dev/sdb1
```

13. You don't usually set up a USB flash drive to mount automatically every time the system boots because it mounts automatically when you plug it in. But if you decide that you want to do that, edit `/etc/fstab` and add a line describing what and where to mount. Here is an example of a line you might add:

```
/dev/sdb1     /mnt/test       xfs     defaults     0 1
```

In this example, the partition (`/dev/sdb1`) is mounted on the `/mnt/test` directory as an `xfs` filesystem. The `defaults` keyword causes the partition to be

mounted at boot time. The number 0 tells the system not to back up files automatically from this filesystem with the `dump` command (`dump` is rarely used anymore, but the field is here). The `1` in the last column tells the system to check the partition for errors after a certain number of mounts.

At this point, you have a working, permanently mounted disk partition. The next section describes how to partition a disk that has multiple partitions.

Creating a multiple-partition disk

Now that you understand the basic process of partitioning a disk, adding a filesystem, and making that filesystem available (temporarily and permanently), it is time to try a more complex example. Taking that same 128GB USB flash drive, I ran the procedure described later in this section to create multiple partitions on one disk.

In this procedure, I configure a Master Boot Record (MBR) partition to illustrate how extended partitions work and to use the older `fdisk` command. I create two partitions of 5GB (`sdb1` and `sdb2`), two 3GB (`sdb3` and `sdb5`), and 4GB (`sdb6`). The `sdb4` device is an extended partition, which consumes all remaining disk space. Space from the `sdb5` and `sdb6` partitions is taken from the extended partition. This leaves plenty of space to create new partitions.

As before, insert the USB flash drive and determine the device name (in my case, `/dev/sdb`). Also, be sure to unmount any partitions that mount automatically when you insert the USB flash drive.

Tip

When you indicate the size of each partition, type the plus sign and the number of megabytes or gigabytes you want to assign to the partition. For example, `+1024M` to create a 1024-megabyte partition or `+10G` for a 10-gigabyte partition. Be sure to remember the plus sign (`+`) and the `M` or `G`! If you forget the `M` or `G`, `fdisk` thinks you mean sectors and you get unexpected results.

1. I started this procedure by overwriting the USB drive with the `dd` command (`dd if=/dev/zero of=/dev/sd<number> bs=1M count=100`). This allowed me to start with a fresh master boot record. Please be careful to use the right drive number, or you could erase your operating system!
2. Create six new partitions as follows.

```
# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.33.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x8933f665.
```

```
        Command (m for help): n
        Partition type
           p   primary (0 primary, 0 extended, 4 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (1-4, default 1): 1
        First sector (2048-240254975, default 2048):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-
240254975, default 240254975): +5G

        Created a new partition 1 of type 'Linux' and of size 5 GiB.

        Command (m for help): n
        Partition type
           p   primary (1 primary, 0 extended, 3 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (2-4, default 2): 2
        First sector (10487808-240254975, default 10487808):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (10487808-
240254975, default 240254975): +5G

        Created a new partition 2 of type 'Linux' and of size 5 GiB.

        Command (m for help): n
        Partition type
           p   primary (2 primary, 0 extended, 2 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (3,4, default 3): 3
        First sector (20973568-240254975, default 20973568):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (20973568-
240254975, default 240254975): +3G

        Created a new partition 3 of type 'Linux' and of size 3 GiB.

        Command (m for help): n
        Partition type
           p   primary (3 primary, 0 extended, 1 free)
           e   extended (container for logical partitions)
        Select (default e): e
        Selected partition 4
```

```
        First sector (27265024-240254975, default 27265024):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (27265024-
240254975, default 240254975): <ENTER>

        Created a new partition 4 of type 'Extended' and of size 101.6 GiB.

        Command (m for help): n
        All primary partitions are in use.
        Adding logical partition 5
        First sector (27267072-240254975, default 27267072):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (27267072-
240254975, default 240254975): +3G

        Created a new partition 5 of type 'Linux' and of size 3 GiB.

        Command (m for help): n
        All primary partitions are in use.
        Adding logical partition 6
        First sector (33560576-240254975, default 33560576):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (33560576-
240254975, default 240254975): +4G

        Created a new partition 6 of type 'Linux' and of size 4 GiB.
```

3. Check the partitioning before saving by typing **p**. Notice that there are five usable partitions (`sdc1`, `sdc2`, `sdc3`, `sdc5`, and `sdc6`) and that the sectors between the Start and End for `sdc4` are being consumed by `sdc5` and `sdc6`.

```
        Command (m for help): p
        ...
       Device     Boot    Start       End   Sectors   Size Id Type
       /dev/sdb1           2048  10487807  10485760     5G 83 Linux
       /dev/sdb2       10487808  20973567  10485760     5G 82 Linux
       /dev/sdb3       20973568  27265023   6291456     3G 83 Linux
       /dev/sdb4       27265024 240254975 212989952 101.6G  5 Extended
       /dev/sdb5       27267072  33558527   6291456     3G 83 Linux
       /dev/sdb6       33560576  41949183   8388608     4G 83 Linux
```

4. The default partition type is Linux. I decided that I wanted to use some of the partitions for swap space (`type 82`), FAT32 (`type x`), and Linux LVM (`type 8e`). To do that, I type **t** and indicate which partition type to use. Type **L** to see a list of partition types.

```
        Command (m for help): t
        Partition number (1-6): 2
        Hex code (type L to list codes): 82
```

```
Changed type of partition 'Linux' to 'Linux swap / Solaris'.

Command (m for help): t

Partition number (1-6): 5
Hex code (type L to list codes): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.

Command (m for help): t

Partition number (1-6): 6
Hex code (type L to list codes): 8e
Changed type of partition 'Linux' to 'Linux LVM'.
```

5. I check that the partition table is the way I want it and then write the changes:

```
Command (m for help): p
...
Device     Boot    Start       End   Sectors   Size Id Type
/dev/sdb1           2048  10487807  10485760     5G 83 Linux
/dev/sdb2       10487808  20973567  10485760     5G 82 Linux swap / Solaris
/dev/sdb3       20973568  27265023   6291456     3G 83 Linux
/dev/sdb4      27265024  240254975 212989952 101.6G  5 Extended
/dev/sdb5      27267072  33558527    6291456     3G  c W95 FAT32 (LBA)
/dev/sdb6    33560576  41949183 8388608    4G 8e Linux LVM

Command (m for help): w

The partition table has been altered!
The kernel still uses the old partitions. The new table
will be used at the next reboot.
Syncing disks
```

6. After the write is completed, check that the kernel knows about the changes to the partition table. To do that, search the `/proc/partitions` for `sdb`. If the new devices are not there, run the `partprobe /dev/sdb` command on the drive or reboot your computer.

```
# grep sdb /proc/partitions
   8       16  120127488 sdb
   8       17  120125440 sdb1
# partprobe /dev/sdb
# grep sdb /proc/partitions
   8       16  120127488 sdb
   8       17    5242880 sdb1
   8       18    5242880 sdb2
   8       19    3145728 sdb3
   8       20          1 sdb4
   8       21    3145728 sdb5
   8       22    4194304 sdb6
```

7. While the partitions are now set for different types of content, other commands are needed to structure the partitions into filesystems or swap areas. Here's how to do that for the partitions just created:

   sdb1: To make this into a regular Linux `ext4` filesystem, type the following:

   ```
   # mkfs -t ext4 /dev/sdb1
   ```

   sdb2: To format this as a swap area, type the following:

   ```
   # mkswap /dev/sdb2
   ```

   sdb3: To make this into an `ext2` filesystem (the default), type the following:

   ```
   # mkfs /dev/sdb3
   ```

   sdb5: To make this into a VFAT filesystem (the default), type the following:

   ```
   # mkfs -t vfat /dev/sdb5
   ```

   sdb6: To make this into a LVM physical volume, type the following:

   ```
   # pvcreate /dev/sdb6
   ```


These partitions are now ready to be mounted, used as swap areas, or added to an LVM volume group. See the next section, "Using Logical Volume Manager Partitions," to see how LVM physical volumes are used to ultimately create LVM logical volumes from volume groups. See the section "Mounting Filesystems" for descriptions of how to mount filesystems and enable swap areas.

Using Logical Volume Manager Partitions

Basic disk partitioning in Linux has its shortcomings. What happens if you run out of disk space? In the old days, a common solution was to copy data to a bigger disk, restart the system with the new disk, and hope that you didn't run out of space again anytime soon. This process meant downtime and inefficiency.

Logical Volume Manager (LVM) offers lots of flexibility and efficiency in dealing with constantly changing storage needs. With LVM, physical disk partitions are added to pools of space called volume groups. Logical volumes are assigned space from volume groups as needed. This gives you these abilities:

- Add more space to a logical volume from the volume group while the volume is still in use.
- Add more physical volumes to a volume group if the volume group begins to run out of space. The physical volumes can be from disks.
- Move data from one physical volume to another so you can remove smaller disks and replace them with larger ones while the filesystems are still in use—again, without downtime.

With LVM, it is also easier to shrink filesystems to reclaim disk space, although shrinking does require that you unmount the logical volume (but no reboot is needed). LVM also supports advanced features, such as mirroring and working in clusters.

Checking an existing LVM

Let's start by looking at an existing LVM example on a Red Hat Enterprise Linux system. The following command displays the partitions on my first hard disk:

```
# fdisk -l /dev/sda | grep /dev/sda
Disk /dev/sda: 160.0 GB, 160000000000 bytes
/dev/sda1   *        2048     1026047      512000    83   Linux
/dev/sda2   *     1026048   312498175   155736064    8e   Linux LVM
```

On this RHEL system, the 160GB hard drive is divided into one 500MB Linux partition (`sda1`) and a second (Linux LVM) partition that consumes the rest of the disk (`sda2`). Next, I use the `pvdisplay` command to see if that partition is being used in an LVM group:

```
# pvdisplay /dev/sda2
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               vg_abc
  PV Size               148.52 GiB / not usable 2.00 MiB
  Allocatable           yes (but full)
  PE Size               4.00 MiB
  Total PE              38021
  Free PE               0
  Allocated PE          38021
  PV UUID               wlvuIv-UiI2-pNND-f39j-oH0X-9too-AOII7R
```

You can see that the LVM physical volume represented by `/dev/sda2` has 148.52GiB of space, all of which has been totally allocated to a volume group named `vg_abc`. The smallest unit of storage that can be used from this physical volume is 4.0MiB, which is referred to as a *Physical Extent (PE)*.

Note

Notice that LVM tools show disk space in MiB and GiB. One MB is 1,000,000 bytes (10∧6), while a MiB is 1,048,576 bytes (2∧20). A MiB is a more accurate way to reflect how data are stored on a computer. But marketing people tend to use MB because it makes the hard disks, CDs, and DVDs they sell look like they have more capacity than they do. Keep in mind that most tools in Linux display storage data in MiB and GiB, although some can display MB and GB as well.

Next, you want to see information about the volume group:

```
# vgdisplay vg_abc
  --- Volume group ---
  VG Name               vg_abc
```

```
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                3
  Open LV               3
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               148.52 GiB
  PE Size               4.00 MiB
  Total PE              38021
  Alloc PE / Size       38021 / 148.52 GiB
  Free  PE / Size       0 / 0
  VG UUID               c2SGHM-KU9H-wbXM-sgca-EtBr-UXAq-UnnSTh
```


You can see that all of the 38,021 PEs have been allocated. Using `lvdisplay` as follows, you can see where they have been allocated (I have snipped some of the output):

```
# lvdisplay vg_abc
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_root
  VG Name                vg_abc
  LV UUID                33VeDc-jd0l-hlCc-RMuB-tkcw-QvFi-cKCZqa
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                50.00 GiB
  Current LE             12800
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_home
  VG Name                vg_abc
  ...
  LV Size                92.64 GiB
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_swap
  VG Name                vg_abc
  ...
  LV Size                5.88 GiB
```

There are three logical volumes drawing space from `vg_abc`. Each logical volume is associated with a device name that includes the volume group name and the logical volume

name: /dev/vg_abc/lv_root (50GB), /dev/vg_abc/lv_home (92.64GB), and /dev/vg_abc/lv_swap (5.88GB). Other devices linked to these names are located in the /dev/mapper directory: vg_abc-lv_home, vg_abc-lv_root, and vg_abc-lv_swap. Either set of names can be used to refer to these logical volumes.

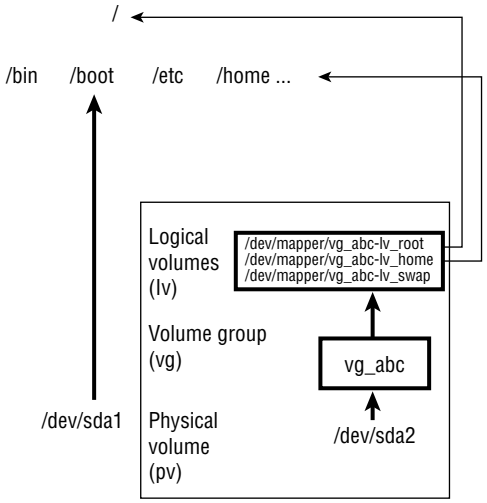
The root and home logical volumes are formatted as ext4 filesystems, whereas the swap logical volume is formatted as swap space. Let's look in the /etc/fstab file to see how these logical volumes are used:

```
# grep vg_ /etc/fstab
/dev/mapper/vg_abc-lv_root /      ext4  defaults    1 1
/dev/mapper/vg_abc-lv_home /home  ext4  defaults    1 2
/dev/mapper/vg_abc-lv_swap swap   swap  defaults    0 0
```

Figure 12.1 illustrates how the different partitions, volume groups, and logical volumes relate to the complete Linux filesystem. The sda1 device is formatted as a filesystem and mounted on the /boot directory. The sda2 device provides space for the vg_abc volume group. Then logical volumes lv_home and lv_root are mounted on the /home and / directories, respectively.

FIGURE 12.1

LVM logical volumes can be mounted like regular partitions on a Linux filesystem.



If you run out of space on any of the logical volumes, you can assign more space from the volume group. If the volume group is out of space, you can add another hard drive or network storage drive and add space from that drive to the volume group so more is available.

Now that you know how LVM works, the next section shows you how to create LVM logical volumes from scratch.

Creating LVM logical volumes

LVM logical volumes are used from the top down, but they are created from the bottom up. As illustrated in Figure 12.1, first you create one or more physical volumes (pv), use the physical volumes to create volume groups (vg), and then create logical volumes from the volume groups (lv).

Commands for working with each LVM component begin with the letters `pv`, `vg`, and `lv`. For example, `pvdisplay` shows physical volumes, `vgdisplay` shows volume groups, and `lvdisplay` shows logical volumes.

The following procedure takes you through the steps of creating LVM volumes from scratch. To do this procedure, you could use the USB flash drive and partitions that I described earlier in this chapter.

1. Obtain a disk with some spare space on it and create a disk partition on it of the LVM type (`8e`). Then use the `pvcreate` command to identify this partition as an LVM physical volume. The process of doing this is described in the section "Creating a multiple-partition disk" using the `/dev/sdb6` device in that example.
2. To add that physical volume to a new volume group, use the `vgcreate` command. The following command shows you how to create a volume group called `myvg0` using the `/dev/sdb6` device:

```
# vgcreate myvg0 /dev/sdc6
    Volume group "myvg0" successfully created
```

3. To see the new volume group, type the following:

```
# vgdisplay myvg0
  --- Volume group ---
  VG Name               myvg0
  ...
  VG Size               <4.00 GiB
  PE Size               4.00 MiB
  Total PE              1023
  Alloc PE / Size       0 / 0
  Free  PE / Size       1023 / <4.00 MiB
```

4. All of the 1023 physical extents (PEs, 4.00 MiB each) are available. Here's how to create a logical volume from some of the space in that volume group and then check that the device for that logical volume exists:

```
# lvcreate -n music -L 1G myvg0
  Logical volume "music" created
# ls /dev/mapper/myvg0*
/dev/mapper/myvg0-music
```

5. As you can see, the procedure created a device named `/dev/mapper/myvg0-music`. That device can now be used to put a filesystem on and mount, just as you did with regular partitions in the first part of this chapter. For example:

```
# mkfs -t ext4 /dev/mapper/myvg0-music
# mkdir /mnt/mymusic
# mount /dev/mapper/myvg0-music /mnt/mymusic
# df -h /mnt/mymusic
Filesystem               Size  Used  Avail  Use%  Mounted on
/dev/mapper/myvg0-music  976M  2.6M   987M    1%  /mnt/mymusic
```

6. As with regular partitions, logical volumes can be mounted permanently by adding an entry to the `/etc/fstab` file, such as

```
/dev/mapper/myvg0-music /mnt/mymusic  ext4 defaults 1 2
```

The next time you reboot, the logical volume is automatically mounted on `/mnt/mymusic`. (Be sure to unmount the logical volume and remove this line if you want to remove the USB flash drive from your computer.)

Growing LVM logical volumes

If you run out of space on a logical volume, you can add space to it without even unmounting it. To do that, you must have space available in the volume group, grow the logical volume, and grow the filesystem to fill it. Building on the procedure in the previous section, here's how to grow a logical volume:

1. Note how much space is currently on the logical volume, and then check that space is available in the logical volume's volume group:

```
# vgdisplay myvg0
...
  VG Size               <4.00 MiB
  PE Size               4.00 MiB
  Total PE              1023
  Alloc PE / Size       256 / 1.00 GiB
  Free  PE / Size       767 / <3.00 GiB
# df -h /mnt/mymusic/
Filesystem                 Size  Used Avail Use% Mounted on
/dev/mapper/myvg0-music    976M  2.6M  987M   1% /mnt/mymusic
```

2. Expand the logical volume using the `lvextend` command:

```
# lvextend -L +1G /dev/mapper/myvg0-music
  Size of logical volume myvg0/music changed
         from 1.00GiB to 2.00 GiB (512 extents).
  Logical volume myvg0/music successfully resized
```

3. Resize the filesystem to fit the new logical volume size:

```
# resize2fs -p /dev/mapper/myvg0-music
```

4. Check to see that the filesystem is now resized to include the additional disk space:

```
# df -h /mnt/mymusic/
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/myvg0-music  2.0G  3.0M  1.9G   1% /mnt/mymusic
```

You can see that the filesystem is now about 1G larger.

Mounting Filesystems

Now that you have had a chance to play with disk partitioning and filesystems, I'm going to step back and talk about how filesystems are set up to connect permanently to your Linux system.

Most of the hard disk partitions created when you install Linux are mounted automatically for you when the system boots. When you install Fedora, Ubuntu, Red Hat Enterprise Linux, and other Linux systems, you have the option to let the installer automatically configure your hard disk or create partitions yourself and indicate the mount points for those partitions.

When you boot Linux, usually all of the Linux partitions on your hard disk are listed in your `/etc/fstab` file and are mounted. For that reason, the following sections describe what you might expect to find in that file. It also describes how you can mount other partitions so that they become part of your Linux filesystem.

The `mount` command is used not only to mount local storage devices but also to mount other kinds of filesystems on your Linux system. For example, `mount` can be used to mount directories (folders) over the network from NFS or Samba servers. It can be used to mount filesystems from a new hard drive or USB flash drive that is not configured to automount. It can also mount filesystem image files using loop devices.

Note

With the addition of automatic mounting features and changes in how removable media are identified with the Linux 2.6 kernel (using features such as Udev and Hardware Abstraction Layer), you no longer need to mount removable media manually for many Linux desktop systems. Understanding how to mount and unmount filesystems manually on a Linux server, however, can be a very useful skill if you want to mount remote filesystems or temporarily mount partitions in particular locations.

Supported filesystems

To see filesystem types that are currently loaded in your kernel, type **`cat /proc/filesystems`**. The list that follows shows a sample of filesystem types that are currently supported in Linux, although they may not be in use at the moment or even available on the Linux distribution you are using.

befs: Filesystem used by the BeOS operating system.

btrfs: A copy-on-write filesystem that implements advanced filesystem features. It offers fault tolerance and easy administration. The btrfs file system has recently grown in popularity for enterprise applications.

cifs: Common Internet Filesystem (CIFS), the virtual filesystem used to access servers that comply with the SNIA CIFS specification. CIFS is an attempt to refine and standardize the SMB protocol used by Samba and Windows file sharing.

ext4: Successor to the popular ext3 filesystem. It includes many improvements over ext3, such as support for volumes up to 1 exbibyte and file sizes up to 16 tebibytes. (This replaced ext3 as the default filesystem used in Fedora and RHEL. It has since been supplanted by xfs as the default for RHEL.)

ext3: Ext filesystems are the most common in most Linux systems. Compared ext2, the ext3 filesystem, also called the third extended filesystem, includes journaling features that, compared to ext2, improve a filesystem's capability to recover from crashes.

ext2: The default filesystem type for earlier Linux systems. Features are the same as ext3, except that ext2 doesn't include journaling features.

ext: This is the first version of ext3. It is not used very often anymore.

iso9660: Evolved from the High Sierra filesystem (the original standard for CD-ROMs). Extensions to the High Sierra standard (called Rock Ridge extensions) allow iso9660 filesystems to support long filenames and UNIX-style information (such as file permissions, ownership, and links). Data CD-ROMs typically use this filesystem type.

kafs: AFS client filesystem. Used in distributed computing environments to share files with Linux, Windows, and Macintosh clients.

minix: Minix filesystem type, used originally with the Minix version of UNIX. It supports filenames of up to only 30 characters.

msdos: An MS-DOS filesystem. You can use this type to mount media that comes from old Microsoft operating systems.

vfat: Microsoft extended FAT (VFAT) filesystem.

exfat: Extended FAT (exFAT) file system that has been optimized for SD cards, USB drives, and other flash memory.

umsdos: An MS-DOS filesystem with extensions to allow features that are similar to UNIX (including long filenames).

proc: Not a real filesystem, but rather a filesystem interface to the Linux kernel. You probably won't do anything special to set up a proc filesystem. However, the `/proc` mount point should be a proc filesystem. Many utilities rely on `/proc` to gain access to Linux kernel information.

reiserfs: ReiserFS journaled filesystem. ReiserFS was once a common default filesystem type for several Linux distributions. However, ext and xfs filesystems are by far more common filesystem types used with Linux today.

swap: Used for swap partitions. Swap areas are used to hold data temporarily when RAM is used up. Data is swapped to the swap area and then returned to RAM when it is needed again.

squashfs: Compressed, read-only filesystem type. Squashfs is popular on live CDs, where there is limited space and a read-only medium (such as a CD or DVD).

nfs: Network Filesystem (NFS) type of filesystem. NFS is used to mount filesystems on other Linux or UNIX computers.

hpfs: Filesystem is used to do read-only mounts of an OS/2 HPFS filesystem.

ncpfs: A filesystem used with Novell NetWare. NetWare filesystems can be mounted over a network.

ntfs: Windows NT filesystem. Depending upon the distribution you have, it may be supported as a read-only filesystem (so that you can mount and copy files from it).

ufs: Filesystem popular on Sun Microsystems's operating systems (that is, Solaris and SunOS).

jfs: A 64-bit journaling filesystem by IBM that is relatively lightweight for the many features it has.

xfs: A high-performance filesystem originally developed by Silicon Graphics that works extremely well with large files. This filesystem is the default type for RHEL 7.

gfs2: A shared disk filesystem that allows multiple machines to all use the same shared disk without going through a network filesystem layer such as CIFS, NFS, and so on.

To see the list of filesystems that come with the kernel you are using, type **`ls /lib/modules/kernelversion/kernel/fs/`**. The actual modules are stored in subdirectories of that directory. Mounting a filesystem of a supported type causes the filesystem module to be loaded, if it is not already loaded.

Type **`man fs`** to see descriptions of Linux filesystems.

Enabling swap areas

A *swap area* is an area of the disk that is made available to Linux if the system runs out of memory (RAM). If your RAM is full and you try to start another application without a swap area, that application will fail.

With a swap area, Linux can temporarily swap out data from RAM to the swap area and then get it back when needed. You take a performance hit, but it is better than having processes fail.

To create a swap area from a partition or a file, use the `mkswap` command. To enable that swap area temporarily, you can use the `swapon` command. For example, here's how to check your available swap space, create a swap file, enable the swap file, and then check that the space is available on your system:

```
# free -m
         total    used     free    shared  buffers  cached
Mem:      1955     663     1291         0       42     283
-/+ buffers/cache:          337      1617
Swap:      819       0      819
# dd if=/dev/zero of=/var/tmp/myswap bs=1M count=1024
# mkswap /var/opt/myswap
# swapon /var/opt/myswap
# free -m
         total    used     free  shared  buffers     cached
Mem:      1955    1720      235       0       42       1310
-/+ buffers/cache:          367    1588
Swap:     1843       0     1843
```

The `free` command shows the amount of swap before and after creating, making, and enabling the swap area with the `swapon` command. That amount of swap is available immediately and temporarily to your system. To make that swap area permanent, you need to add it to your `/etc/fstab` file. Here is an example:

```
/var/opt/myswap  swap   swap    defaults   0 0
```

This entry indicates that the swap file named `/var/opt/myswap` should be enabled at boot time. Because there is no mount point for swap area, the second field is just set to swap, as is the partition type. To test that the swap file works before rebooting, you can enable it immediately (`swapon -a`) and check that the additional swap area appears:

```
# swapon -a
```

Disabling swap area

If at any point you want to disable a swap area, you can do so using the `swapoff` command. You might do this, in particular, if the swap area is no longer needed and you want to reclaim the space being consumed by a swap file or remove a USB drive that is providing a swap partition.

First, make sure that no space is being used on the swap device (using the `free` command), and then use `swapoff` to turn off the swap area so that you can reuse the space. Here is an example:

```
# free -m
         total    used     free     shared    buffers     cached
Mem:      1955    1720      235          0         42       1310
-/+ buffers/cache: 367     1588
```

```
Swap:     1843       0     1843
# swapoff /var/opt/myswap
# free -m
Mem:      1955    1720      235          0         42       1310
-/+ buffers/cache: 367     1588
Swap:      819       0      819
```

Notice that the amount of available swap was reduced after running the `swapoff` **command.**

Using the fstab file to define mountable file systems

The hard disk partitions on your local computer and the remote filesystems that you use every day are probably set up to mount automatically when you boot Linux. The `/etc/fstab` file contains definitions for each partition, along with options describing how the partition is mounted. Here's an example of an `/etc/fstab` file:

```
# /etc/fstab
/dev/mapper/vg_abc-lv_root     /        ext4   defaults       1 1
UUID=78bdae46-9389-438d-bfee-06dd934fae28 /boot ext4  defaults  1 2
/dev/mapper/vg_abc-lv_home     /home    ext4   defaults       1 2
/dev/mapper/vg_abc-lv_swap     swap     swap   defaults       0 0
# Mount entries added later
/dev/sdb1                      /win     vfat   ro             1 2
192.168.0.27:/nfsstuff         /remote  nfs    users,_netdev  0 0
//192.168.0.28/myshare         /share   cifs   guest,_netdev  0 0
# special Linux filesystems
tmpfs                          /dev/shm tmpfs   defaults      0 0
devpts                         /dev/pts devpts gid=5,mode=620 0 0
sysfs                          /sys     sysfs  defaults       0 0
proc                           /proc    proc   defaults       0 0
```

The `/etc/fstab` file just shown is from a default Red Hat Enterprise Linux 6 server install, with a few lines added.

For now, you can ignore the `tmpfs`, `devpts`, `sysfs`, and `proc` entries. Those are special devices associated with shared memory, terminal windows, device information, and kernel parameters, respectively.

In general, the first column of `/etc/fstab` shows the device or share (what is mounted), while the second column shows the mount point (where it is mounted). That is followed by the type of filesystem, any mount options (or defaults), and two numbers (used to tell commands such as `dump` and `fsck` what to do with the filesystem).

The first three entries represent the disk partitions assigned to the root of the filesystem (`/`), the `/boot` directory, and the `/home` directory. All three are `ext4` filesystems. The fourth line is a swap device (used to store data when RAM overflows). Notice that the device names for `/`, `/home`, and `swap` all start with `/dev/mapper`. That's because they are LVM logical volumes that are assigned space from a pool of space called an LVM volume group (more on LVM in the section "Using Logical Volume Manager Partitions" earlier in this chapter).

The /boot partition is on its own physical partition, /dev/sda1. Instead of using /dev/sda1, however, a unique identifier (UUID) identifies the device. Why use a UUID instead of /dev/sda1 to identify the device? Suppose you plugged another disk into your computer and booted up. Depending on how your computer iterates through connected devices on boot, it is possible that the new disk might be identified as /dev/sda, causing the system to look for the contents of /boot on the first partition of that disk.

To see all of the UUIDs assigned to storage devices on your system, type the blkid command, as follows:

```
# blkid
/dev/sda1:
  UUID="78bdae46-9389-438d-bfee-06dd934fae28" TYPE="ext4"
/dev/sda2:
  UUID="wlvuIv-UiI2-pNND-f39j-oH0X-9too-AOII7R" TYPE="LVM2_member"
/dev/mapper/vg_abc-lv_root:
  UUID="3e6f49a6-8fec-45e1-90a9-38431284b689" TYPE="ext4"
/dev/mapper/vg_abc-lv_swap:
  UUID="77662950-2cc2-4bd9-a860-34669535619d" TYPE="swap"
/dev/mapper/vg_abc-lv_home:
  UUID="7ffbcff3-36b9-4cbb-871d-091efb179790" TYPE="ext4"
/dev/sdb1:
  SEC_TYPE="msdos" UUID="75E0-96AA" TYPE="vfat"
```

Any of the device names can be replaced by the UUID designation in the left column of an /etc/fstab entry.

I added the next three entries in /etc/fstab to illustrate some different kinds of entries. I connected a hard drive from an old Microsoft Windows system and had it mounted on the /win directory. I added the ro option so it would mount read-only.

The next two entries represent remote filesystems. On the /remote directory, the /nfsstuff directory is mounted read/write (rw) from the host at address 192.168.0.27 as an NFS share. On the /share directory, the Windows share named myshare is mounted from the host at 192.168.0.28. In both cases, I added the _netdev option, which tells Linux to wait for the network to come up before trying to mount the shares. For more information on mounting CIFS and NFS shares, refer to Chapters 19, "Configuring a Windows File Sharing (Samba) Server," and 20, "Configuring an NFS File Server," respectively.

Coming from Windows

The section "Using the fstab file to define mountable file systems" shows mounting a hard disk partition from an old VFAT filesystem being used in Windows. Most Windows systems today use the NTFS filesystem. Support for this system, however, is not delivered with every Linux system. NTFS is available from Fedora in the ntfs-3g package.

To help you understand the contents of the /etc/fstab file, here is what is in each field of that file:

Field 1: Name of the device representing the filesystem. This field can include the LABEL or UUID option with which you can indicate a volume label or universally unique identifier (UUID) instead of a device name. The advantage to this approach is that because the partition is identified by volume name, you can move a volume to a different device name and not have to change the fstab file. (See the description of the mkfs command in the section "Using the mkfs Command to Create a Filesystem" later in this chapter for information on creating and using labels.)

Field 2: Mount point in the filesystem. The filesystem contains all data from the mount point down the directory tree structure unless another filesystem is mounted at some point beneath it.

Field 3: Filesystem type. Valid filesystem types are described in the section "Supported filesystems" earlier in this chapter (although you can only use filesystem types for which drivers are included for your kernel).

Field 4: Use defaults or a comma-separated list of options (no spaces) that you want to use when the entry is mounted. See the mount command manual page (under the -o option) for information on other supported options.

Tip

Typically, only the root user is allowed to mount a filesystem using the mount command. However, to allow any user to mount a filesystem (such as a filesystem on a CD), you could add the user option to Field 4 of /etc/fstab.

Field 5: The number in this field indicates whether the filesystem needs to be dumped (that is, have its data backed up). A 1 means that the filesystem needs to be dumped, and a 0 means that it doesn't. (This field is no longer particularly useful because most Linux administrators use more sophisticated backup options than the dump command. Most often, a 0 is used.)

Field 6: The number in this field indicates whether the indicated filesystem should be checked with fsck when the time comes for it to be checked: 1 means it needs to be checked first, 2 means to check after all those indicated by 1 have already been checked, and 0 means don't check it.

If you want to find out more about mount options as well as other features of the /etc/fstab file, there are several man pages to which you can refer, including man 5 nfs and man 8 mount.

Using the mount command to mount file systems

Linux systems automatically run mount -a (mount all filesystems from the /etc/fstab file) each time you boot. For that reason, you generally use the mount

command only for special situations. In particular, the average user or administrator uses `mount` in two ways:

- To display the disks, partitions, and remote filesystems currently mounted
- To mount a filesystem temporarily

Any user can type `mount` (with no options) to see what filesystems are currently mounted on the local Linux system. The following is an example of the `mount` command. It shows a single hard disk partition (`/dev/sda1`) containing the root (`/`) filesystem and proc and devpts filesystem types mounted on `/proc` and `/dev`, respectively.

```
$ mount
/dev/sda3 on / type ext4 (rw)
/dev/sda2 on /boot type ext4 (rw)
/dev/sda1 on /mnt/win type vfat (rw)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/cdrom on /media/MyOwnDVD type iso9660 (ro,nosuid,nodev)
```

Traditionally, the most common devices to mount by hand are removable media, such as DVDs or CDs. However, depending on the type of desktop you are using, CDs and DVDs may be mounted for you automatically when you insert them. (In some cases, applications are launched as well when media is inserted. For example, a music player or photo editor may be launched when your inserted USB medium has music or digital images on it.)

Occasionally, however, you may find it useful to mount a filesystem manually. For example, you want to look at the contents of an old hard disk, so you install it as a second disk on your computer. If the partitions on the disk did not automount, you could mount partitions from that disk manually. For example, to mount a read-only disk partition `sdb1` that has an older `ext3` filesystem, you could type this:

```
# mkdir /mnt/temp
# mount -t ext3 -o ro /dev/sdb1 /mnt/tmp
```

Another reason to use the `mount` command is to remount a partition to change its mount options. Suppose that you want to remount `/dev/sdb1` as read/write, but you do not want to unmount it (maybe someone is using it). You could use the remount option as follows:

```
# mount -t ext3 -o remount,rw /dev/sdb1
```

Mounting a disk image in loopback

Another valuable way to use the `mount` command has to do with disk images. If you download an SD card or DVD ISO image file from the Internet and you want to see what it contains, you can do so without burning it to DVD or other medium. With the image on your hard disk, create a mount point and use the `-o loop` option to mount it locally. Here's an example:

```
# mkdir /mnt/mydvdimage
# mount -o loop whatever-i686-disc1.iso /mnt/mydvdimage
```

In this example, the /mnt/mydvdimage directory is created, and then the disk image file (whatever-i686-disc1.iso) residing in the current directory is mounted on it. You can now cd to that directory, view the contents of it, and copy or use any of its contents. This is useful for downloaded DVD images from which you want to install software without having to burn the image to DVD. You could also share that mount point over NFS, so you could install the software from another computer. When you are finished, to unmount the image, type umount /mnt/mydvdimage.

Other options to mount are available only for specific filesystem types. See the mount manual page for those and other useful options.

Using the umount command

When you are finished using a temporary filesystem, or you want to unmount a permanent filesystem temporarily, use the umount command. This command detaches the filesystem from its mount point in your Linux filesystem. To use umount, you can give it either a directory name or a device name, as shown in this example:

```
# umount /mnt/test
```

This unmounts the device from the mount point /mnt/test. You can also unmount using this form:

```
# umount /dev/sdb1
```

In general, it's better to use the directory name (/mnt/test) because the umount command will fail if the device is mounted in more than one location. (Device names all begin with /dev.)

If you get the message device is busy, the umount request has failed because either an application has a file open on the device or you have a shell open with a directory on the device as a current directory. Stop the processes or change to a directory outside the device you are trying to unmount for the umount request to succeed.

An alternative for unmounting a busy device is the -l option. With umount -l (a lazy unmount), the unmount happens as soon as the device is no longer busy. To unmount a remote NFS filesystem that's no longer available (for example, the server went down), you can use the umount -f option to forcibly unmount the NFS filesystem.

Tip

A really useful tool for discovering what's holding open a device you want to unmount is the lsof command. Type lsof with the name of the partition that you want to unmount (such as lsof /mnt/test). The output shows you what commands are holding files open on that partition. The fuser-v /mnt/test command can be used in the same way.

Using the mkfs Command to Create a Filesystem

You can create a filesystem for any supported filesystem type on a disk or partition that you choose. You do so with the `mkfs` command. Although this is most useful for creating filesystems on hard-disk partitions, you can create filesystems on USB flash drives or rewritable DVDs as well.

Before you create a new filesystem, make sure of the following:

- You have partitioned the disk as you want (using the `fdisk` command).
- You get the device name correct, or you may end up overwriting your hard disk by mistake. For example, the first partition on the second SCSI or USB flash drive on your system is `/dev/sdb1` and the third disk is `/dev/sdc1`.
- To unmount the partition if it's mounted before creating the filesystem.

The following are two examples of using `mkfs` to create a filesystem on two partitions on a USB flash drive located as the first and second partitions on the third SCSI disk (`/dev/sdc1` and `/dev/sdc2`). The first creates an xfs partition, while the second creates an ext4 partition.

```
# mkfs -t xfs /dev/sdc1
meta-data=/dev/sda3        isize=256    agcount=4, agsize=256825 blks
         =                 sectsz=512   attr=2, projid32bit=1
         =                 crc=0
data     =                 bsize=4096   blocks=1027300, imaxpct=25
         =                 sunit=0      swidth=0 blks
naming   =version 2        bsize=4096   ascii-ci=0 ftype=0
log      =internal log     bsize=4096   blocks=2560, version=2
         =                 sectsz=512   sunit=0 blks, lazy-count=1
realtime =none             extsz=4096   blocks=0, rtextents=0

# mkfs -t ext4 /dev/sdc2
mke2fs 1.44.6 (5-Mar-2019)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 6379d82e-fa25-4160-8ffa-32bc78d410eee
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912
Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

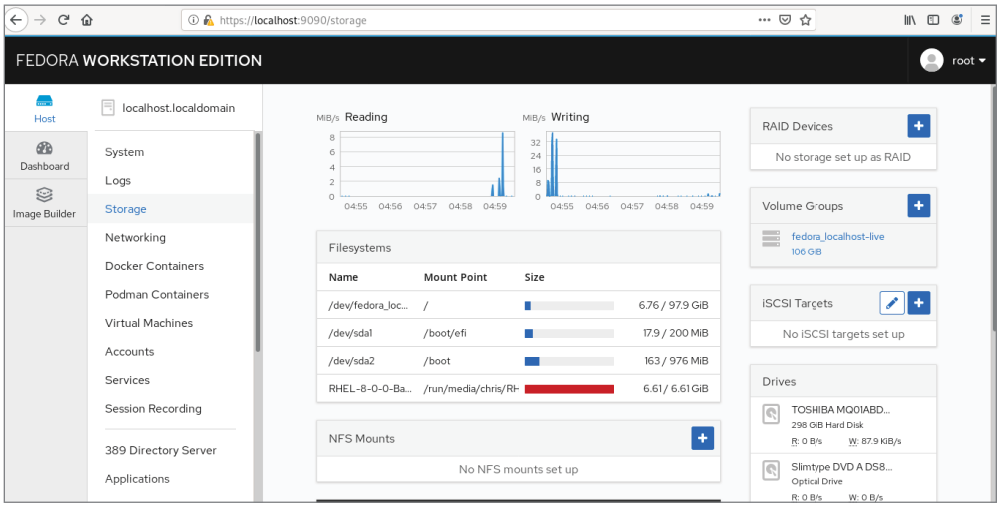
You can now mount either of these filesystems (for example, `mkdir /mnt/myusb ; mount /dev/sdc1 /mnt/myusb`), change to `/mnt/myusb` as your current directory (`cd /mnt/myusb`), and create files on it as you please.

Managing Storage with Cockpit

Most of the features described in this chapter for working with disk partitions and filesystems using command-line tools can be accomplished using the Cockpit web user interface. With Cockpit running on your system, open the web UI (`hostname:9090`) and select the Storage tab. Figure 12.2 shows an example of the Cockpit Storage tab on a Fedora system.

FIGURE 12.2

View storage devices, filesystems, and activities from the Cockpit Storage page.



The Storage tab provides a solid overview of your system's storage. It charts read and write activity of your storage devices every minute. It displays the local filesystems and storage (including RAID devices and LVM volume groups) as well as remotely mounted NFS shares and iSCSI targets. Each hard disk, DVD, and other physical storage device is also displayed on the Storage tab.

Select a mounted filesystem, and you can see and change partitioning for that filesystem. For example, by selecting the entry for a filesystem that was automatically mounted on `/run/media`, you can see all of the partitions for the device it is on (`/dev/sdb1` and `/dev/sdb2`). Figure 12.3 shows that there is an ISO9660 filesystem (typical for bootable media) and a smaller VFAT filesystem on the two partitions.

With the storage device information displayed, you could reformat the entire storage device (Create Partition Table) or, assuming that space is available on the device, add a new partition (Create Partition). Figure 12.4 shows an example of the window that appears when you select Create Partition Table.

FIGURE 12.3

View and change disk partitions for a select storage device.

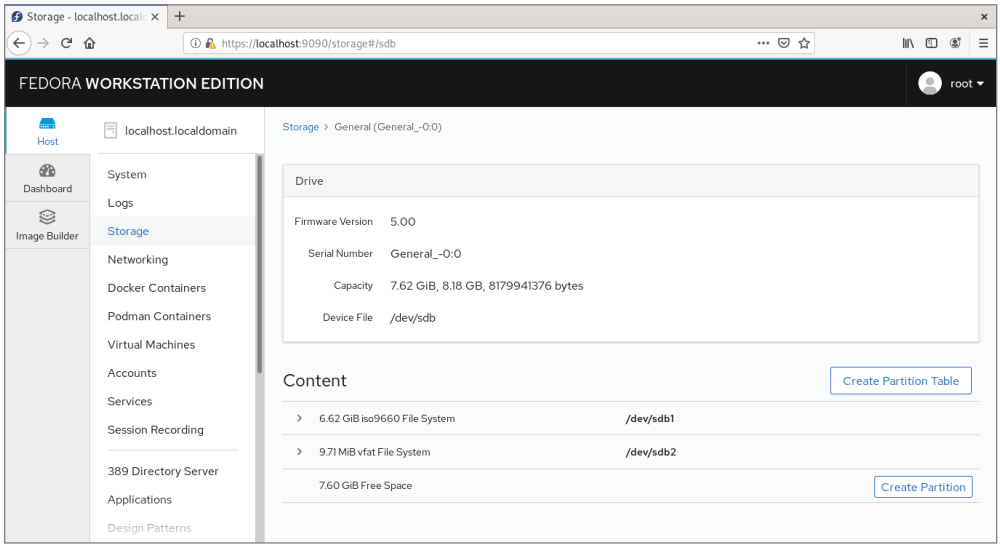
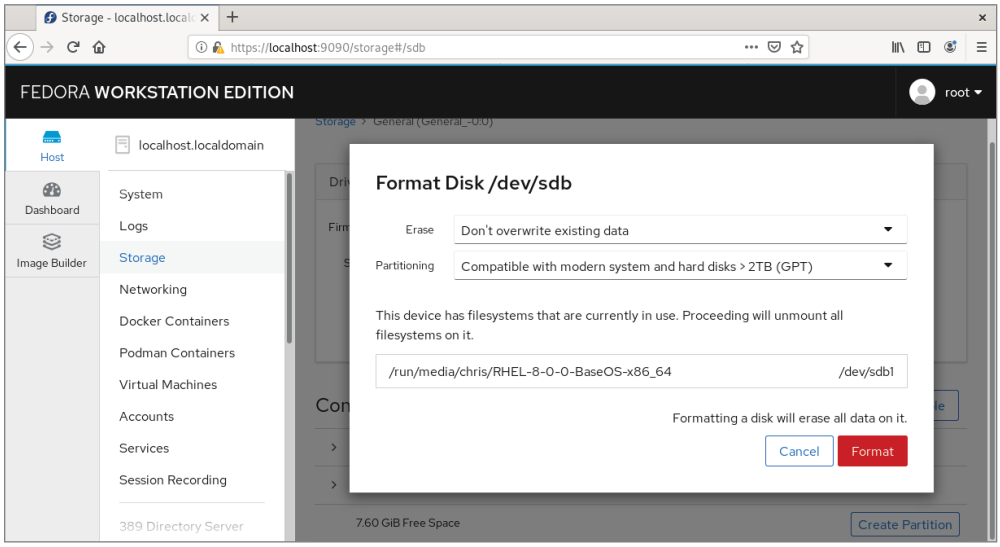


FIGURE 12.4

Creating a new partition table



If you decide that you want to format the disk or USB drive, change the Erase setting to allow all of the data on the drive to be overwritten and then choose the type of partitioning. Select Format to unmount any mounted partitions from the drive and create a new partition table. After that, you can add partitions to the storage device, choosing the size, filesystem type, and whether or not to encrypt data. You can even choose where in the operating system's filesystem to mount the new partition. With just a few selections, you can quickly create the disk layouts that you want in ways that are more intuitive than methods for doing comparable steps from the command line.

Summary

Managing filesystems is a critical part of administering a Linux system. Using commands such as `fdisk`, you can view and change disk partitions. Filesystems can be added to partitions using the `mkfs` command. Once created, filesystems can be mounted and unmounted using the `mount` and `umount` commands, respectively.

Logical Volume Manager (LVM) offers a more powerful and flexible way of managing disk partitions. With LVM, you create pools of storage, called volume groups, which can allow you to grow and shrink logical volumes as well as extend the size of your volume groups by adding more physical volumes.

For a more intuitive way of working with storage devices, Cockpit offers an intuitive, Web-based interface for viewing and configuring storage on your Linux system. Using the Web UI, you can see both local and networked storage as well as reformat disks and modify disk partitions.

With most of the basics needed to become a system administrator covered at this point in the book, Chapter 13, "Understanding Server Administration," introduces concepts for extending those skills to manage network servers. Topics in that chapter include information on how to install, manage, and secure servers.

Exercises

Use these exercises to test your knowledge of creating disk partitions, Logical Volume Manager, and working with filesystems. You need a USB flash drive that is at least 1GB, which you can erase for these exercises.

These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Run a command as root to watch the system journal in a Terminal as fresh data comes in and insert your USB flash drive. Determine the device name of the USB flash drive.

2. Run a command to list the partition table for the USB flash drive.
3. Delete all the partitions on your USB flash drive, save the changes, and make sure the changes were made both on the disk's partition table and in the Linux kernel.
4. Add three partitions to the USB flash drive: 100MB Linux partition, 200MB swap partition, and 500MB LVM partition. Save the changes.
5. Put an ext4 filesystem on the Linux partition.
6. Create a mount point called `/mnt/mypart` and mount the Linux partition on it.
7. Enable the swap partition and turn it on so that additional swap space is immediately available.
8. Create a volume group called `abc` from the LVM partition, create a 200MB logical volume from that group called `data`, add a VFAT partition, and then temporarily mount the logical volume on a new directory named `/mnt/test`. Check that it was successfully mounted.
9. Grow the logical volume from 200MB to 300MB.
10. Do what you need to do to remove the USB flash drive safely from the computer: unmount the Linux partition, turn off the swap partition, unmount the logical volume, and delete the volume group from the USB flash drive.

Part IV

Becoming a Linux Server Administrator

Chapter 13
Understanding Server Administration

Chapter 14
Administering Networking

Chapter 15
Starting and Stopping Services

Chapter 16
Configuring a Print Server

Chapter 17
Configuring a Web Server

Chapter 18
Configuring an FTP Server

Chapter 19
Configuring a Windows File Sharing (Samba) Server

Chapter 20
Configuring an NFS File Server

Chapter 21
Troubleshooting Linux

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER 13

Understanding Server Administration

IN THIS CHAPTER

Administering Linux servers

Communicating with servers over networks

Setting up logging locally and remotely

Monitoring server systems

Managing servers in the enterprise

Although some system administration tasks are needed even on a desktop system (installing software, setting up printers, and so on), many new tasks appear when you set up a Linux system to act as a server. That's especially true if the server that you configure is made public to anyone on the Internet, where you can be overloaded with requests from good guys while needing to be constantly on guard against attacks from bad guys.

Dozens of different kinds of servers are available for Linux systems. Most servers serve up data to remote clients, but others serve the local system (such as those that gather logging messages or kick off maintenance tasks at set times using the `cron` facility). Many servers are represented by processes that run continuously in the background and respond to requests that come to them. These processes are referred to as *daemon* processes.

As the name implies, servers exist to serve. The data that they serve can include web pages, files, database information, email, and lots of other types of content. As a server administrator, some of the additional challenges to your system administration skills include the following:

Remote access To use a desktop system, you typically sit at its console. Server systems, by contrast, tend to be housed in racks in climate-controlled environments under lock and key. More often than not, after the physical computers are in place, most administration of those machines is done using remote access tools. Often, no graphical interface is available, so you must rely on command-line tools or browser-based interfaces to do things such as remote login, remote copying, and remote execution. The most common of these tools are built on the Secure Shell (SSH) facility.

Diligent security To be useful, a server must be able to accept requests for content from remote users and systems. Unlike desktop systems, which can simply close down all network

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

ports that allow incoming requests for access, a server must make itself vulnerable by allowing some access to its ports. That's why as a server administrator, it is important to open ports to services that are needed and lock down ports that are not needed. You can secure services using tools such as `iptables` and `firewalld` (firewall tools) and Security Enhanced Linux (to limit the resources a service can access from the local system).

Continuous monitoring Although you typically turn off your laptop or desktop system when you are not using it, servers usually stay on 24×7, 365 days a year. Because you don't want to sit next to each server and continuously monitor it personally, you can configure tools to monitor each server, gather log messages, and even forward suspicious messages to an email account of your choice. You can enable system activity reporters to gather data around the clock on CPU usage, memory usage, network activity, and disk access.

In this chapter, I lay out some of the basic tools and techniques that you need to know to administer remote Linux servers. You learn to use SSH tools to access your server securely, transfer data back and forth, and even launch remote desktops or graphical applications and have them appear on your local system. You learn to use remote logging and system activity reports to monitor system activities continuously.

Starting with Server Administration

Whether you are installing a file server, web server, or any of the other server facilities available with Linux systems, many of the steps required for getting the server up and running are the same. Where server setup diverges is in the areas of configuration and tuning.

In later chapters, I describe specific servers and how they differ. In each of the server-related chapters that follow, you'll go through the same basic steps for getting that server started and available to be used by your clients.

Step 1: Install the server

Although most server software is not preinstalled on the typical Linux system, any general-purpose Linux system offers the software packages needed to supply every major type of server available.

Sometimes, multiple software packages associated with a particular type of server are gathered together in package groups (sometimes called *package collections*). At other times, you just need to install the server packages you want individually. Here are some server package categories in Fedora and some of the packages available in each category:

System logging server The `rsyslog` service allows the local system to gather log messages delivered from a variety of components on the system. It can also act as a remote logging server, gathering logging messages sent from other logging servers. (The `rsyslog` service is described later in this chapter.) In recent Ubuntu, Fedora,

and RHEL systems, log messages are gathered in the `systemd` journal, which can be picked up and redirected by the `rsyslog` service or displayed locally by the `journalctl` command.

Print server The Common UNIX Printing Service (`cups` package) is used most often to provide print server features on Linux systems. Packages that provide graphical administration of CUPS (`system-config-printer`) and printer drivers (`foomatic`, `hpijs`, and others) are also available when you install CUPS. (See Chapter 16, "Configuring a Print Server.")

Web server The Apache (`httpd` or `apache2` package) web server is the software used most often to serve web pages (HTTP content). Related packages include modules to help serve particular types of content (Perl, Python, PHP, and SSL connections). Likewise, there are packages of related documentation (`httpd-manual`), tools for monitoring web data (webalizer), and tools for providing web proxy services (squid). (See Chapter 17, "Configuring a Web Server.")

FTP server The Very Secure FTP daemon (`vsftpd` package) is the default FTP server used in Fedora and RHEL. Other FTP server packages include `proftpd` and `pure-ftpd`. (See Chapter 18, "Configuring an FTP Server.")

Windows file server Samba (`samba` package) allows a Linux system to act as a Windows file and print server. (See Chapter 19, "Configuring a Windows File Sharing [Samba] Server.")

NFS file server Network File System (NFS) is the standard Linux and UNIX feature for providing shared directories to other systems over a network. The `nfs-utils` package provides NFS services and related commands. (See Chapter 20, "Configuring an NFS File Server.")

Mail server These types of packages enable you to configure email servers, sometimes referred to as a Mail Transport Agent (MTA) server. You have several choices of email servers, including `sendmail`, `postfix` (default in Fedora and RHEL), and `exim`. Related packages, such as `dovecot`, allow the mail server to deliver email to clients.

Directory server Packages in this category provide remote and local authentication services. These include Kerberos (`krb5-server`), LDAP (`openldap-servers`), and NIS (`ypserv`).

DNS server The Berkeley Internet Name Domain service (`bind`) provides the software needed to configure a server to resolve hostnames into IP addresses.

Network Time Protocol server The `ntpd` or `chronyd` package provides a service that you can enable to sync up your system clock with clocks from public or private NTP servers.

SQL server The PostgreSQL (`postgresql` and `postgresql-server` packages) service is an object-relational database management system. Related packages provide PostgreSQL documentation and related tools. The MySQL (`mysql` and `mysql-server` packages) service is another popular open source SQL database server.

A community-developed branch of MySQL called MariaDB has supplanted MySQL on many Linux distributions.

Step 2: Configure the server

Most server software packages are installed with a default configuration that leans more toward security than immediate full use. Here are some things to think about when you set out to configure a server.

Using configuration files

Traditionally, Linux servers have been configured by editing plain-text files in the `/etc` directory (or subdirectories). Often, there is a primary configuration file; sometimes, there is a related configuration directory in which files ending in `.conf` can be pulled into the main configuration file.

The `httpd` package (Apache web server) is an example of a server package that has a primary configuration file and a directory where other configuration files can be dropped in and be included with the service. The main configuration file in Fedora and RHEL is `/etc/httpd/conf/httpd.conf`. The configuration directory is `/etc/httpd/conf.d/`.

After installing `httpd` and related packages, you will see files in the `/etc/httpd/conf.d/` directory that were placed there by different packages: `mod_ssl`, `mod_perl`, and so on. This is a way that add-on packages to a service can have their configuration information enabled in the `httpd` server, without the package trying to run a script to edit the main `httpd.conf` file.

The one downside to plain-text configuration files is that you don't get the kind of immediate error checking you get when you use graphical administration tools. You either have to run a test command (if the service includes one) or actually try to start the service to see if there is any problem with your configuration file.

Tip

Instead of using `vi` to edit configuration files, use `vim`. Using the `vim` command can help you catch configuration file errors as you are editing.

The `vim` command knows about the formats of many configuration files (`passwd`, `httpd.conf`, `fstab`, and others). If you make a mistake and type an invalid term or option in one of those files, or break the format somehow, the color of the text changes. For example, in `/etc/fstab`, if you change the option `defaults` to `default`, the word's color changes.

Checking the default configuration

Most server software packages in Fedora and RHEL are installed with minimal configuration and lean more toward being secure than totally useful out of the box. While installing a software package, some Linux distributions ask you things such as the directory in which you want to install it or the user account with which you want to manage it.

Because RPM packages are designed to be installed unattended, the person installing the package has no choice on how it is installed. The files are installed in set locations, specific user accounts are enabled to manage it, and when you start the service, it might well offer limited accessibility. You are expected to configure the software after the package is installed to make the server fully functional.

Two examples of servers that are installed with limited functionality are mail servers (`sendmail` or `postfix` packages) and DNS servers (`bind` package). Both of these servers are installed with default configurations and start up on reboot. However, both also only listen for requests on your `localhost`. So, until you configure those servers, people who are not logged in to your local server cannot send mail to the mail server or use your computer as a public DNS server, respectively.

Step 3: Start the server

Most services that you install in Linux are configured to start up when the system boots and then run continuously, listening for requests, until the system is shut down. There are two major facilities for managing services: `systemd` (used now by RHEL, Ubuntu, and Fedora) and SystemVinit scripts (used by Red Hat Enterprise Linux through RHEL 6.*x*).

Regardless of which facility is used on your Linux system, it is your job to do things such as set whether you want the service to come up when the system boots and to start, stop, and reload the service as needed (possibly to load new configuration files or temporarily stop access to the service). Commands for doing these tasks are described in Chapter 15, "Starting and Stopping Services."

Most, but not all, services are implemented as daemon processes. Here are a few things that you should know about those processes:

User and group permissions Daemon processes often run as users and groups other than root. For example, `httpd` runs as apache and `ntpd` runs as the ntp user. The reason for this is that if someone cracks these daemons, they would not have permissions to access files beyond what the services can access.

Daemon configuration files Often, a service has a configuration file for the daemon stored in the `/etc/sysconfig` directory. This is different than the service configuration file in that its job is often just to pass arguments to the server process itself rather than configure the service. For example, options you set in the `/etc/sysconfig/rsyslogd` file are passed to the `rsyslogd` daemon when it starts up. You can tell the daemon, for example, to output additional debugging information or accept remote logging messages. See the man page for the service (for example, `man rsyslogd`) to see what options are supported.

Port numbers Packets of data go to and from your system over network interfaces through ports for each supported protocol (usually UDP or TCP). Most standard services have specific port numbers to which daemons listen and to which clients connect. Unless you are trying to hide the location of a service, you typically don't change the ports on which a daemon process listens. When you go to secure

a service, you must make sure that the port to the service is open on the firewall (see Chapter 25, "Securing Linux on a Network," for information on `iptables` and `firewalld` firewalls). Also, if you change a port on which the service is listening, and SELinux is in enforcing mode, SELinux may prevent the daemon from listening on that port (see Chapter 24, "Enhancing Linux Security with SELinux," for more information on SELinux).

Note

One reason for changing port numbers on a service is "security by obscurity." For example, the `sshd` service is a well-known target for people trying to break into a system by guessing logins and passwords on TCP port 22.

I have heard about people changing their Internet-facing `sshd` service to listen on some other port number (perhaps some unused, very high port number). Then they tell their friends or colleagues to log in to their machine from SSH by pointing to this other port. The idea is that port scanners looking to break into a system might be less likely to scan the normally unused port.

Not all services run continuously as daemon processes. Some older UNIX services ran on demand using the `xinetd` super server. Other services just run once on startup and exit. Still others run only a set number of times, being launched when the `crond` daemon sees that the service was configured to run at the particular time.

In recent years, previous `xinetd` services in RHEL and Fedora, such as `telnet` and `tftp`, have been converted to `systemd` services. A number of services, including `cockpit`, use `systemd` sockets to achieve the same results.

Step 4: Secure the server

Opening your system to allow remote users to access it over the network is not a decision to be taken lightly. Crackers all over the world run programs to scan for vulnerable servers that they can take over for their data or their processing power. Luckily, there are measures that you can put in place on Linux systems to protect your servers and services from attacks and abuse.

Some common security techniques are described in the following sections. These and other topics are covered in more depth in Part V, "Learning Linux Security Techniques."

Password protection

Good passwords and password policies are the first line of defense in protecting a Linux system. If someone can log in to your server via ssh as the root user with a password of `foobar`, expect to be cracked. A good technique is to disallow direct login by root and require every user to log in as a regular user and then use `su` or `sudo` to become root.

You can also use the Pluggable Authentication Module (PAM) facility to adjust the number of times that someone can have failed login attempts before blocking access to that person. PAM also includes other features for locking down authentication to your Linux server. For a description of PAM, see Chapter 23, "Understanding Advanced Linux Security."

Of course, you can bypass passwords altogether by requiring public key authentication. To use that type of authentication, you must make sure that any user you want to have access to your server has their public key copied to the server (such as through `ssh-copy-id`). Then they can use `ssh`, `scp`, or related commands to access that server without typing their password. See the section "Using key-based (passwordless) authentication" later in this chapter for further information.

Firewalls

The `iptables` firewall service can track and respond to every packet coming from and going to network interfaces on your computer. Using `iptables`, you can drop or reject every packet making requests for services on your system except for those few that you have enabled. Further, you can tell `iptables` to allow service requests only from certain IP addresses (good guys) or not allow requests from other addresses (bad guys).

In recent RHEL and Fedora versions, the `firewalld` feature adds a layer of functionality to Linux firewall rules. With `firewalld`, you can not only insert firewall rules into the kernel, you can also organize firewall rules by dividing them up into zones and changing firewall rules on the fly to react to different events.

In each of the server chapters coming up, I describe what ports need to be open to allow access to services. Descriptions of how `iptables` and `firewalld` work are included in Chapter 25, "Securing Linux on a Network."

TCP Wrappers

TCP Wrappers, which uses `/etc/hosts.allow` and `/etc/hosts.deny` files to allow and deny access in a variety of ways to selected services, was used primarily to secure older UNIX services, and it is no longer considered to be very secure. While the use of the TCP Wrapper program (`/usr/sbin/tcpd`) is only common on systems that use `xinetd`, the `/etc/hosts.allow` and `/etc/hosts.deny` files that the TCP Wrapper program checked before granting access to network services are often checked by daemons that are configured to do so. The configuration option within the configuration files for these daemons is often labeled as TCP Wrapper support.

SELinux

Fedora, Red Hat Enterprise Linux, and other Linux distributions come with the Security Enhanced Linux (SELinux) feature included and in Enforcing mode. Although the default targeted mode doesn't have much impact on most applications that you run in Linux, it has a major impact on most major services.

A major function of SELinux is to protect the contents of your Linux system from the processes running on the system. In other words, SELinux makes sure a web server, FTP server, Samba server, or DNS server can access only a restricted set of files on the system (as defined by file contexts) and allows only a restricted set of features (as defined by Booleans and limited port access).

Details about how to use SELinux are contained in Chapter 24, "Enhancing Linux Security with SELinux."

Security settings in configuration files

Within the configuration files of most services are values that you can set to secure the service further. For example, for file servers and web servers, you can restrict access to certain files or data based on username, hostname, IP address of the client, or other attributes.

Step 5: Monitor the server

Because you can't be there to monitor every service, every minute, you need to put monitoring tools in place to watch your servers for you and make it easy for you to find out when something needs attention. Some of the tools that you can use to monitor your servers are described in the sections that follow.

Configure logging

Using the `rsyslog` service (`rsyslogd` daemon), you can gather critical information and error conditions into log files about many different services. By default, in RHEL log messages from applications are directed into log files in the `/var/log` directory. For added security and convenience, log messages can also be directed to a centralized server, providing a single location to view and manage logging for a group of systems.

Several different software packages are available to work with `rsyslog` and manage log messages. The `logwatch` feature scans your log files each night and sends critical information gathered from those files to an email account of your choice. The `logrotate` feature backs up log files into compressed archives when the logs reach a certain size or pass a set amount of time since the previous backup.

The features for configuring and managing system logging are described in the section "Configuring System Logging" later in this chapter.

Run system activity reports

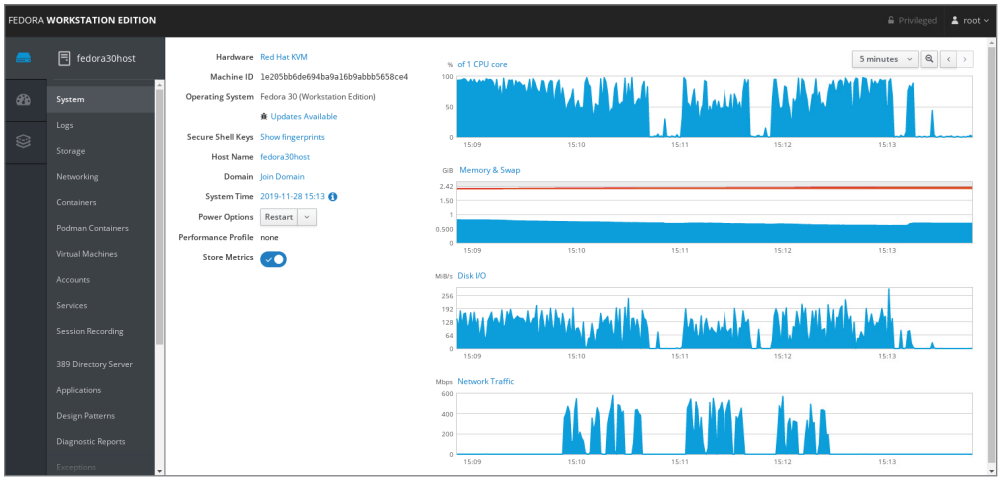
The `sar` facility (which is enabled by the `sysstat` package) can be configured to watch activities on your system such as memory usage, CPU usage, disk latency, network activities, and other resource drains. By default, the `sar` facility launches the `sadc` program every few minutes, day and night, to gather data. Viewing that data later can help you go back and figure out where and when demand is spiking on your system. The `sar` facility is described in the section "Checking System Resources with `sar`" later in this chapter.

Watch activity live with Cockpit

With Cockpit running on your system, you can watch system activity in real time. Open your web browser to display the Cockpit console (`https://localhost:9090`). In real time, you can watch percentage of CPU use, memory and swap consumption, how much data is written to and from disk (disk i/o), and network traffic as it is gathered and displayed across the screen. Figure 13.1 shows an example of the System area of the Cockpit console, displaying activity data.

FIGURE 13.1

Log in to Cockpit



Keep system software up to date

As security holes are discovered and patched, you must make sure that the updated software packages containing those patches are installed on your servers. Again, with mission-critical servers, the safest and most efficient way is to use subscribed Red Hat Enterprise Linux systems for your servers and then deploy security-related package updates to your system as soon as they are released and tested.

To keep your personal server and desktop systems up to date, there are various graphical tools to add software and to check for updates. You can also use the `yum` command to check for and install all packages that are available for your RHEL or Fedora systems (enter **`dnf update`** or **`yum update`**).

Check the filesystem for signs of crackers

To check your filesystem for possible intrusion, you can run commands such as `rpm -V` to check to see if any commands, document files, or configuration files have been tampered with on your system. For more information on `rpm -V`, refer to the description of `rpm -V` in Chapter 10, "Getting and Managing Software."

Now that you have an overview of how Linux server configuration is done, the next sections of this chapter focus on the tools that you need to access, secure, and maintain your Linux server systems.

Checking and Setting Servers

If you are tasked with managing a Linux server, the following sections include a bunch of items that you can check. Keep in mind that nowadays many servers in large data centers are deployed and managed by larger platforms. So, know how the server is managed before you make any changes to it. Your changes might be overwritten automatically if you changed the defined state of that system.

Managing Remote Access with the Secure Shell Service

The *Secure Shell tools* are a set of client and server applications that allow you to do basic communications between client computers and your Linux server. The tools include `ssh`, `scp`, `sftp`, and many others. Because communication is encrypted between the server and the clients, these tools are more secure than similar, older tools. For example, instead of using older remote login commands such as `telnet` or `rlogin`, you could use `ssh`. The `ssh` command can also replace older remote execution commands, such as `rsh`. Remote copy commands, such as `rcp`, can be replaced with secure commands such as `scp` and `rsync`.

With Secure Shell tools, both the authentication process and all communications that follow are encrypted. Communications from `telnet` and the older "`r`" commands expose passwords and all data to someone sniffing the network. Today, `telnet` and similar commands should be used only for testing access to remote ports, providing public services such as PXE booting, or doing other tasks that don't expose your private data.

NOTE
For a deeper discussion of encryption techniques, refer to Chapter 23, "Understanding Advanced Linux Security."

Most Linux systems include secure shell clients, and many include the secure shell server as well. If you are using the Fedora or RHEL distribution, for example, the client and server software packages that contain the `ssh` tools are `openssh`, `openssh-clients`, and `openssh-server` packages as follows:

```
# yum list installed | grep openssh
...
openssh.x86_64           7.9p1-5.fc30    @anaconda
openssh-clients.x86_64   7.9p1-5.fc30    @anaconda
openssh-server.x86_64    7.9p1-5.fc30    @anaconda
```

On Ubuntu, only the `openssh-clients` package is installed. It includes the functionality of the `openssh` package. If you need the server installed, use the `sudo apt-get install openssh-server` command.

```
$ sudo dpkg --list | grep openssh
openssh-client/bionic-updates,bionic-security,now 1:7.6p1-4ubuntu0.3 amd64
[installed]
  secure shell (SSH) client, for secure access to remote machines
openssh-client-ssh1/bionic 1:7.5p1-10 amd64
  secure shell (SSH) client for legacy SSH1 protocol

openssh-sftp-server/bionic-updates,bionic-security,now 1:7.6p1-4ubuntu0.3
amd64 [installed]
  secure shell (SSH) sftp server module, for SFTP access from remote machines
$ sudo apt-get install openssh-server
```

Starting the openssh-server service

Linux systems that come with the `openssh-server` package already installed sometimes are not configured for it to start automatically. Managing Linux services (see Chapter 15, "Starting and Stopping Services") can be very different depending on the different distributions. Table 13.1 shows the commands to use in order to ensure that the `ssh` server daemon, `sshd`, is up and running on a Linux system.

TABLE 13.1 Commands to Determine `sshd` Status

Distribution	Command to Determine `sshd` Status
RHEL 6	`chkconfig --list sshd`
Fedora and RHEL 7 or later	`systemctl status sshd.service`
Ubuntu	`systemctl status ssh.service`

If `sshd` is not currently running, you can start it by issuing one of the commands listed in Table 13.2. These commands need root privileges in order to work.

TABLE 13.2 Commands to Start `sshd`

Distribution	Command to Start `sshd`
RHEL 6	`service sshd start`
Fedora and RHEL 7 or later	`systemctl start sshd.service`
Ubuntu	`systemctl start ssh.service`

The commands in Table 13.2 only start the `ssh` or `sshd` service. They do not configure it to start automatically at boot. To make sure the server service is set up to start automatically, you need to use one of the commands in Table 13.3 using root privileges.

TABLE 13.3 Commands to Start `sshd` at Boot

Distribution	Command to Start `sshd` at Boot
RHEL 6	`chkconfig sshd on`
Fedora and RHEL 7 or later	`systemctl enable sshd.service`
Ubuntu	`systemctl enable ssh.service`

When you install `openssh-server` on Ubuntu, the `sshd` daemon is configured to start automatically at boot. Therefore, you may not need to run the command in Table 13.3 for your Ubuntu server.

Modify your firewall settings to allow the `openssh-client` to access port 22 (firewalls are covered in Chapter 25, "Securing Linux on a Network"). After the service is up and running and the firewall is properly configured, you should be able to use `ssh` client commands to access your system via the `ssh` server.

Any further configurations for what the `sshd` daemon is allowed to do are handled in the `/etc/ssh/sshd_config` file. At a minimum, set the `PermitRootLogin` setting to `no`. This stops anyone from remotely logging in as root.

```
# grep PermitRootLogin /etc/ssh/sshd_config
PermitRootLogin no
```

After you have changed the `sshd_config` file, restart the `sshd` service. After that point, if you use `ssh` to log in to that system from a remote client, you must do so as a regular user and then use `su` or `sudo` to become the root user.

Using SSH client tools

Many tools for accessing remote Linux systems have been created to make use of the SSH service. The most frequently used of those tools is the `ssh` command, which can be used for remote login, remote execution, and other tasks. Commands such as `scp` and `rsync` can copy one or more files at a time between SSH client and server systems. The `sftp` command provides an FTP-like interface for traversing a remote filesystem and getting and putting files between the systems interactively.

By default, all of the SSH-related tools authenticate using standard Linux usernames and passwords, all done over encrypted connections. However, SSH also supports key-based authentication, which can be used to configure key-based and possibly passwordless authentication between clients and SSH servers, as described in the section "Using key-based (passwordless) authentication" later in this chapter.

Using ssh for remote login

Use the `ssh` command from another Linux computer to test that you can log in to the Linux system running your `sshd` service. The `ssh` command is one that you will use often to access a shell on the servers you are configuring.

Try logging in to your Linux server from another Linux system using the `ssh` command. (If you don't have another Linux system, you can simulate this by typing `localhost` instead of the IP address and logging in as a local user.) The following is an example of remotely logging in to `johndoe`'s account on `10.140.67.23`:

```
$ ssh johndoe@10.140.67.23
The authenticity of host '10.140.67.23 (10.140.67.23)'
     can't be established.
RSA key fingerprint is
     a4:28:03:85:89:6d:08:fa:99:15:ed:fb:b0:67:55:89.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.140.67.23' (RSA) to the
     list of known hosts.
johndoe@10.140.67.23's password: *********
```

If this is the very first time that you have logged in to that remote system using the `ssh` command, the system asks you to confirm that you want to connect. Type `yes`, and press Enter. When prompted, enter the user's password.

When you type `yes` to continue, you accept the remote host's public key. At that point, the remote host's public key is downloaded to the client in the client's `~/.ssh/known_hosts` file. Now data exchanged between these two systems can be encrypted and decrypted using RSA asymmetric encryption (see Chapter 23, "Understanding Advanced Linux Security"). After you are logged in to the remote system, you can begin typing shell commands. The connection functions like a normal login. The only difference is that the data is encrypted as it travels over the network.

When you are finished, type `exit` to end the remote connection. The connection is closed, and you are returned to the command prompt on your local system. (If the local shell doesn't return after you exit the remote shell, typing `~.` usually closes the connection.)

```
$ exit
logout
Connection to 10.140.67.23 closed
```

After you have remotely connected to a system, a file in your local system subdirectory, `~.ssh/known_hosts`, will exist. This file contains the public key of the remote host along with its IP address. Your server's public and private keys are stored in the `/etc/ssh` directory.

```
$ ls .ssh
known_hosts
$ cat .ssh/known_hosts
10.140.67.23 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAoyfJK1YwZhNmpHE4yLPZAZ9ZNEdRE7I159f3I
yGiH21Ijfqs
NYFR10ZlBLlYyTQi06r/9O19GwCaJ753InQ8FWHW+OOYOG5pQmghhn
/x0LD2uUb6egOu6zim1NEC
JwZf5DWkKdy4euCUEMSqADh/WYeuOSoZ0pp2IAVCdh6
w/PIHMF1HVR069cvdv+OTL4vD0X8llSpw
```

```
0ozqRptz2UQgQBBbBjK1RakD7fY1TrWv
NQhYG/ugt gPaY4JDYeY6OBzcadpxZmf7EYUw0ucXGVQ1a
NP/erIDOQ9rA0YNzCRv
y2LYCm2/9adpAxc+UYi5UsxTw4ewSBjmsXYq//Ahaw4mjw==
```

TIP

Any later attempts by this user to contact the server at 10.140.67.23 are authenticated using this stored key. If the server should change its key (which happens if the operating system is reinstalled or if keys are rotated), attempts to `ssh` to that system result in a refused connection and dire warnings that you may be under attack. If the key has indeed changed, in order to be able to `ssh` to that address again, just remove the host's key (the whole line) from your `known _hosts` file and you can copy over the new key.

Using ssh for remote execution

Besides logging into a remote shell, the `ssh` command can be used to execute a command on the remote system and have the output returned to the local system. Here is an example:

```
$ ssh johndoe@10.140.67.23 hostname
johndoe@10.140.67.23's password: **********
host01.example.com
```

In the example just shown, the `hostname` command runs as the user `johndoe` on the Linux system located at IP address `10.140.67.23`. The output of the command is the name of the remote host (in this case, `host01.example.com`), which appears on the local screen.

If you run a remote execution command with `ssh` that includes options or arguments, be sure to surround the whole remote command line in quotes. Keep in mind that if you refer to files or directories in your remote commands, relative paths are interpreted in relation to the user's home directory, as shown here:

```
$ ssh johndoe@10.140.67.23 "cat myfile"
johndoe@10.140.67.23's password: **********
Contents of the myfile file located in johndoe's home directory.
```

The `ssh` command just shown goes to the remote host located at `10.140.67.23` and runs the `cat myfile` command as the user `johndoe`. This causes the contents of the `myfile` file from that system to be displayed on the local screen.

Another type of remote execution that you can do with `ssh` is X11 forwarding. If X11 forwarding is enabled on the server (`X11Forwarding yes` is set in the `/etc/sshd/sshd_config` file), you can run graphical applications from the server securely over the SSH connection using `ssh -X`. For a new server administrator, this means that if there are graphical administration tools installed on a server, you can run those tools without having to sit at the console, as in this example:

```
$ ssh -X johndoe@10.140.67.23 system-config-printer
johndoe@10.140.67.23's password: **********
```

After running this command, you are prompted for the root password. After that, the Printers window appears, ready for you to configure a printer. Just close the window when you are finished, and the local prompt returns. You can do this for any graphical administration tool or just regular X applications (such as the `gedit` graphical editor, so that you don't have to use `vi`).

If you want to run several X commands and don't want to have to reconnect each time, you can use X11 forwarding directly from a remote shell as well. Put them in the background and you can have several remote X applications running on your local desktop at once. Here's an example:

```
$ ssh -X johndoe@10.140.67.23
johndoe@10.140.67.23's password: **********
$ system-config-printer &
$ gedit &
$ exit
```

After you have finished using the graphical applications, close them as you would normally. Then type `exit`, as shown in the preceding code, to leave the remote shell and return to your local shell.

Copying files between systems with scp and rsync

The `scp` command is similar to the old UNIX `rcp` command for copying files to and from Linux systems, except that all communications are encrypted. Files can be copied from the remote system to the local system or local to remote. You can also copy files recursively through a whole directory structure if you choose.

The following is an example of using the `scp` command to copy a file called `memo` from the home directory of the user `chris` to the `/tmp` directory on a remote computer as the user `johndoe`:

```
$ scp /home/chris/memo johndoe@10.140.67.23:/tmp
johndoe@10.140.67.23's password: ***************
memo        100%|****************|   153    0:00
```

You must enter the password for `johndoe`. After the password is accepted, the file is copied to the remote system successfully.

You can do recursive copies with `scp` using the `-r` option. Instead of a file, pass a directory name to the `scp` command and all files and directories below that point in the filesystem are copied to the other system.

```
$ scp johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
volname.1.gz                                  100%  543    0.5KB/s   00:00
mtools.1.gz                                   100% 6788    6.6KB/s   00:00
roqet.1.gz                                    100% 2496    2.4KB/s   00:00
...
```

As long as the user johndoe has access to the files and directories on the remote system and the local user can write to the target directory (both are true in this case), the directory structure from /usr/share/man/man1 down is copied to the local /tmp directory.

The scp command can be used to back up files and directories over a network. However, if you compare scp to the rsync command, you see that rsync (which also works over SSH connections) is a better backup tool. Try running the scp command shown previously to copy the man1 directory (you can simulate the command using localhost instead of the IP address if you only have one accessible Linux system). Now enter the following on the system to which you copied the files:

```
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
-rw-r--r--.1 johndoe johndoe 2628 Apr 15 15:32 /tmp/man1/batch.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
      -> at.1.gz
```

Next, run the scp command again and list the files once more:

```
$ scp johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
-rw-r--r--.1 johndoe johndoe 2628 Apr 15 15:40 /tmp/man1/batch.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
      -> at.1.gz
```

The output of those commands tells you a few things about how scp works:

Attributes lost Permissions or date/time stamp attributes were not retained when the files were copied. If you were using scp as a backup tool, you would probably want to keep permissions and time stamps on the files if you needed to restore the files later.

Symbolic links lost The batch.1.gz file is actually a symbolic link to the at.1.gz file. Instead of copying the link, scp follows the link and actually copies the file. Again, if you were to restore this directory, batch.1.gz would be replaced by the actual at.1.gz file instead of a link to it.

Copy repeated unnecessarily If you watched the second scp output, you would notice that all files were copied again, even though the exact files being copied were already on the target. The updated modification date confirms this. By contrast, the rsync command can determine that a file has already been copied and not copy the file again.

The rsync command is a better network backup tool because it can overcome some of the shortcomings of scp just listed. Try running an rsync command to do the same action that scp just did, but with a few added options:

```
$ rm -rf /tmp/man1/
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
sending incremental file list
man1/
```

```
man1/HEAD.1.gz
man1/Mail.1.gz -> mailx.1.gz
...
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1/ /tmp/
johndoe@10.140.67.23's password: ***************
sending incremental file list
sent 42362 bytes  received 13 bytes  9416.67 bytes/sec
total size is 7322223  speedup is 172.80
$ ls -l /usr/share/man/man1/batch* /tmp/man1/batch*
lrwxrwxrwx.1 johndoe johndoe 7 Feb 14 17:49 /tmp/man1/batch.1.gz
       -> at.1.gz
lrwxrwxrwx.1 root root 7 Feb 14 17:49 /usr/share/man/man1/batch.1.gz
       -> at.1.gz
```

After removing the `/tmp/man1` directory, you run an `rsync` command to copy all of the files to the `/tmp/man1` directory, using `-a` (recursive archive), `-v` (verbose), and `-l` (copy symbolic links). Then run the command immediately again and notice that nothing is copied. The `rsync` command knows that all of the files are there already, so it doesn't copy them again. This can be a tremendous savings of network bandwidth for directories with gigabytes of files where only a few megabytes change.

Also notice from the output of `ls -l` that the symbolic links have been preserved on the `batch.1.gz` file and so has the date/time stamp on the file. If you need to restore those files later, you can put them back exactly as they were.

This use of `rsync` is good for backups. But what if you wanted to mirror two directories, making the contents of two directory structures exactly the same on two machines? The following commands illustrate how to create an exact mirror of the directory structure on both machines using the directories shown with the previous `rsync` commands.

First, on the remote system, copy a new file into the directory being copied:

```
# cp /etc/services /usr/share/man/man1
```

Next, on the local system, run `rsync` to copy across any new files (in this case, just the directory and the new file, `services`):

```
$ rsync -avl johndoe@10.140.67.23:/usr/share/man/man1 /tmp
johndoe@10.140.67.23's password:
***************
sending incremental file list
man1/
man1/services
```

After that, go back to the remote system and remove the new file:

```
$ sudo rm /usr/share/man/man1/services
```

Now, on the local system, run `rsync` again and notice that nothing happens. At this point, the remote and local directories are different because the local system has the services file and the remote doesn't. That is correct behavior for a backup directory. (You want to have

files on the backup in case something was removed by mistake.) However, if you want the remote and local directories to be mirrored, you would have to add the `--delete` option. The result is that the services file is deleted on the local system, making the remote and local directory structures in sync.

```
$ rsync -avl /usr/share/man/man1 localhost:/tmp
johndoe@10.140.67.23's password: ***************
sending incremental file list
man1/
$ rsync -avl --delete johndoe@10.140.67.23:/usr/share/man/man1 /tmp
johndoe@10.140.67.23's password: ***************
sending incremental file list
deleting man1/services
```

Interactive copying with sftp

If you don't know exactly what you want to copy to or from a remote system, you can use the `sftp` command to create an interactive FTP-style session over the SSH service. Using `sftp`, you can connect to a remote system over SSH, change directories, list directory contents, and then (given proper permission) get files from and put files on the server. Keep in mind that, despite its name, `sftp` has nothing to do with the FTP protocol and doesn't use FTP servers. It simply uses an FTP style of interaction between a client and a `sshd` server.

The following example shows the user `johndoe` connecting to `jd.example.com`:

```
$ sftp johndoe@jd.example.com
Connecting to jd.example.com
johndoe@jd.example.com's password: ***************
sftp>
```

At this point, you can begin an interactive FTP session. You can use `get` and `put` commands on files as you would with any FTP client, but with the comfort of knowing that you are working on an encrypted and secure connection. Because the FTP protocol passes usernames, passwords, and data in clear text, using `sftp` over SSH, if possible, is a much better alternative for allowing your users to copy files interactively from the system.

Using key-based (passwordless) authentication

If you are using SSH tools to connect to the same systems throughout the day, you might find it inconvenient to be entering your password over and over again. Instead of using password-based authentication, SSH allows you to set up key-based authentication to use instead. Here's how it works:

- You create a public key and a private key.
- You guard the private key but copy the public key across to the user account on the remote host to which you want to do key-based authentication.
- With your key copied to the proper location, you use any SSH tools to connect to the user account on the remote host, but instead of asking you for a password, the remote SSH service compares the public key and the private key and allows you access if the two keys match.

When you create the keys, you are given the option to add a passphrase to your private key. If you decide to add a passphrase, even though you don't need to enter a password to authenticate to the remote system, you still need to enter your passphrase to unlock your private key. If you don't add a passphrase, you can communicate using your public/private key pairs in a way that is completely passwordless. However, if someone should get ahold of your private key, they could act as you in any communication that required that key.

The following procedure demonstrates how a local user named `chris` can set up key-based authentication to a remote user named `johndoe` at IP address `10.140.67.23`. If you don't have two Linux systems, you can simulate this by using two user accounts on your local system. I start by logging in as the local user named `chris` and typing the following to generate my local public/private key pair:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chris/.ssh/id_rsa): ENTER
Enter passphrase (empty for no passphrase): ENTER
Enter same passphrase again: ENTER
Your identification has been saved in /home/chris/.ssh/id_rsa.
Your public key has been saved in /home/chris/.ssh/id_rsa.pub.
The key fingerprint is:
bf:06:f8:12:7f:f4:c3:0a:3a:01:7f:df:25:71:ec:1d chris@abc.example.com
The key's randomart image is:
 ...
```

I accepted the default RSA key (DSA keys are also allowed) and pressed Enter twice to have a blank passphrase associated with the key. As a result, my private key (`id_rsa`) and public key (`id_rsa.pub`) are copied to the `.ssh` directory in my local home directory. The next step is to copy that key over to a remote user so that I can use key-based authentication each time I connect to that user account with `ssh` tools:

```
 $ ssh-copy-id -i ~/.ssh/id_rsa.pub johndoe@10.140.67.23
johndoe@10.140.67.23's password:
***************
```

When prompted, I entered `johndoe`'s password. With that accepted, the public key belonging to `chris` is copied to the `authorized_keys` file in `johndoe`'s `.ssh` directory on the remote system. Now, the next time `chris` tries to connect to `johndoe`'s account, the SSH connection is authenticated using those keys. Because no passphrase is put on the private key, no passphrase is required to unlock that key when it is used.

Log into the machine with `ssh johndoe@10.140.67.23`, and check in the `$HOME/.ssh/authorized_keys` to make sure that you haven't added extra keys that you weren't expecting.

```
[chris]$ ssh johndoe@10.140.67.23
Last login: Sun Apr 17 10:12:22 2016 from  10.140.67.22
[johndoe]$
```

With the keys in place, `chris` could now use `ssh`, `scp`, `rsync`, or any other SSH-enabled command to do key-based authentication. Using these keys, for example, an `rsync` command could go into a `cron` script and automatically back up `johndoe`'s home directory every night.

Want to secure your remote system further? After you have the keys in place on your remote system for everyone you want to allow to log in to that system, you can set the `sshd` service on the remote system to not allow password authentication by changing the `PasswordAuthentication` setting in the `/etc/ssh/sshd_config` file to `no`, so that it appears as follows:

```
PasswordAuthentication no
```

Then restart the `sshd` service (`systemctl restart sshd`). After that, anyone with a valid key is still accepted. Anyone who tries to log in without a key gets the following failure message and doesn't even get a chance to enter a username and password:

```
Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

Configuring System Logging

With the knowledge of how to access your remote server using SSH tools, you can log in to the server and set up some of the services needed to make sure that it's running smoothly. System logging is one of the basic services configured for Linux to keep track of what is happening on the system.

The `rsyslog` service (`rsyslogd` daemon) provides the features to gather log messages from software running on the Linux system and direct those messages to local log files, devices, or remote logging hosts. Configuration of `rsyslog` is similar to the configuration of its predecessor, `syslog`. However, `rsyslog` allows you to add modules to manage and direct log messages more specifically.

In recent Red Hat Enterprise Linux and Fedora releases, the `rsyslog` facility leverages messages that are gathered and stored in the `systemd` journal. To display journal log messages directly from the `systemd` journal, instead of viewing them from files in the `/var/log` directory, use the `journalctl` command.

Enabling system logging with rsyslog

Most of the files in the `/var/log` directory are populated with log messages directed to them from the `rsyslog` service. The `rsyslogd` daemon is the system logging daemon. It accepts log messages from a variety of other programs and writes them to the appropriate log files. This is better than having every program write directly to its own log file because it enables you to manage centrally how log files are handled.

Configuring `rsyslogd` to record varying levels of detail in the log files is possible. It can be told to ignore all but the most critical messages, or it can record every detail.

The rsyslogd daemon can even accept messages from other computers on your network. This remote logging feature is particularly handy because it enables you to centralize the management and review of the log files from many systems on your network. There is also a major security benefit to this practice.

With remote logging, if a system on your network is broken into, the cracker cannot delete or modify the log files because those files are stored on a separate computer. It is important to remember, however, that those log messages are not, by default, encrypted (though encryption can be enabled). Anyone tapping into your local network can eavesdrop on those messages as they pass from one machine to another. Also, although crackers may not be able to change old log entries, they can affect the system such that any new log messages should not be trusted.

Running a dedicated loghost, a computer that serves no purpose other than to record log messages from other computers on the network, is not uncommon. Because this system runs no other services, it is unlikely that it will be broken into. This makes it nearly impossible for crackers to erase their tracks completely.

Understanding the rsyslog.conf file

The /etc/rsyslog.conf file is the primary configuration file for the rsyslog service. In the /etc/rsyslog.conf file, a modules section lets you include or not include specific features in your rsyslog service. The following is an example of the modules section of /etc/rsyslog.conf in RHEL 8:

```
module(load="imuxsock"
    # provides support for local system logging (e.g. via logger command)
     SysSock.Use="off") # Turn off message reception via local log socket;
                        # local messages are retrieved through imjournal now.
module(load="imjournal"
    # provides access to the systemd journal
       StateFile="imjournal.state") # File to store the position in the journal
#module(load="imklog")
    # reads kernel messages (the same are read from journald)
#module(load="immark")
    # provides --MARK-- message capability

# Provides UDP syslog reception
# for parameters see http://www.rsyslog.com/doc/imudp.html
#module(load="imudp") # needs to be done just once
#input(type="imudp" port="514")

# Provides TCP syslog reception
# for parameters see http://www.rsyslog.com/doc/imtcp.html
#module(load="imtcp") # needs to be done just once
#input(type="imtcp" port="514")
```

Entries beginning with module(load= load the modules that follow. Modules that are currently disabled are preceded by a pound sign (#). The imjournal module lets rsyslog

access the systemd journal. The imuxsock module is needed to accept messages from the local system. (It should not be commented out—preceded by a pound sign—unless you have a specific reason to do so.) The imklog module logs kernel messages.

Modules not enabled by default include the immark module, which allows --MARK-- messages to be logged (used to indicate that a service is alive). The imudp and imtcp modules and related port number entries are used to allow the rsyslog service to accept remote logging messages and are discussed in more detail in the section "Setting up and using a loghost with rsyslogd" later in this chapter.

Most of the work done in the /etc/rsyslog.conf configuration file involves modifying the RULES section. The following is an example of some of the rules in the RULES section of the /etc/rsyslog.conf file (note that in Ubuntu, you need to look in the /etc/rsyslog.d directory for this configuration information):

```
#### RULES ####
# Log all kernel messages to the console.

# Logging much else clutters up the screen.

#kern.*                                          /dev/console
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none        /var/log/messages
# The authpriv file has restricted access.
authpriv.*                                       /var/log/secure
# Log all the mail messages in one place.
mail.*                                           -/var/log/maillog
# Log cron stuff
cron.*                                           /var/log/cron
```

Rules entries come in two columns. In the left column are designations of what messages are matched; the right column shows where matched messages go. Messages are matched based on facility (mail, cron, kern, and so on) and priority (starting at debug, info, notice, and up to crit, alert, and emerg), separated by a dot (.). So mail.info matches all messages from the mail service that are info level and above.

As for where the messages go, most messages are directed to files in the /var/log directory. You can, however, direct messages to a device (such as /dev/console) or a remote loghost (such as @loghost.example.com). The at sign (@) indicates that the name that follows is the name of the loghost.

By default, logging is done only to local files in the /var/log directory. However, if you uncomment the kern.* entry, you can easily direct kernel messages of all levels to your computer's console screen.

The first working entry in the preceding example shows that info level messages from all services (*) are matched by that rule, with the exception of messages from mail,

authpriv, and cron services (which are excluded with the word none). All of the matched messages are directed to the /var/log/messages file.

The mail, authpriv (authentication messages), and cron (cron facility messages) services each has its own log files, as listed in the columns to their right. To understand the format of those and other log files, the format of the /var/log/messages file is described next.

Understanding the messages log file

Because of the many programs and services that record information to the messages log file, understanding the format of this file is important. You can get a good early warning of problems developing on your system by examining this file. Each line in the file is a single message recorded by some program or service. Here is a snippet of an actual messages log file:

```
Feb 25 11:04:32 toys network: Bringing up loopback:  succeeded
Feb 25 11:04:35 toys network: Bringing up interface eth0:  succeeded
Feb 25 13:01:14 toys vsftpd(pam_unix)[10565]: authentication failure;
      logname= uid=0 euid=0 tty= ruser= rhost=10.0.0.5  user=chris
Feb 25 14:44:24 toys su(pam_unix)[11439]: session opened for
      user root by chris(uid=500)
```

The default message format in the /var/log/messages file is divided into five main parts. This format is determined by the following entry in the /etc/rsyslog.conf file:

```
module(load="builtin:omfile" Template="RSYSLOG_TraditionalFileFormat")
```

When you view messages in files from the /var/log directory, from left to right, message parts are as follows:

- The date and time that the message was logged
- The name of the computer from which the message came
- The program or service name to which the message pertains
- The process number (enclosed in square brackets) of the program sending the message
- The actual text message

Take another look at the preceding file snippet. In the first two lines, you can see that the network was restarted. The next line shows that the user named chris tried and failed to get to the FTP server on this system from a computer at address 10.0.0.5. (He typed the wrong password and authentication failed.) The last line shows chris using the su command to become root user.

By occasionally reviewing the messages and secure files, you could catch a cracking attempt before it is successful. If you see an excessive number of connection attempts for a particular service, especially if they are coming from systems on the Internet, you may be under attack.

Setting up and using a loghost with rsyslogd

To redirect your computer's log files to another computer's `rsyslogd`, you must make changes to both the local and remote `rsyslog` configuration file, `/etc/rsyslog.conf`. Become root using the `su –` command, and then open the `/etc/rsyslog.conf` file in a text editor (such as `vi`).

On the client side

To send the messages to another computer (the loghost) instead of a file, start by replacing the log file name with the `@` character followed by the name of the loghost. For example, to direct the output of messages that are being sent to the `messages`, `secure`, and `maillog` log files to a loghost as well, add the lines in bold to the messages file:

```
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;news.none;authpriv.none;cron.none  /var/log/messages
*.info;mail.none;news.none;authpriv.none;cron.none  @loghost
# The authpriv file has restricted access.
authpriv.*                                          /var/log/secure
authpriv.*                                          @loghost
# Log all the mail messages in one place.
mail.*                                               -/var/log/maillog
mail.*                                              @loghost
```

The messages are now sent to the `rsyslogd` running on the computer named loghost. The name `loghost` was not an arbitrary choice. Creating such a hostname and making it an alias to the actual system acting as the loghost is customary. That way, if you ever need to switch the loghost duties to a different machine, you need to change only the loghost alias; you do not need to re-edit the `syslog.conf` file on every computer.

On the loghost side

The loghost that is set to accept the messages must listen for those messages on standard ports (514 UDP, although it can be configured to accept messages on 514 TCP as well). Here is how you would configure the Linux loghost that is also running the `rsyslog` service:

- Edit the `/etc/rsyslog.conf` file on the loghost system and uncomment the lines that enable the `rsyslogd` daemon to listen for remote log messages. Uncomment the first two lines to enable incoming UDP log messages on port 514 (default); uncomment the two lines after that to allow messages that use TCP protocol (also port 514):

```
module(load="imudp") # needs to be done just once
input(type="imudp" port="514")
module(load="imtcp") # needs to be done just once
input(type="imtcp" port="514")
```

- Open your firewall to allow new messages to be directed to your loghost. (See Chapter 25, "Securing Linux on a Network," for a description of how to open specific ports to allow access to your system.)
- Restart the `rsyslog` service (**`service rsyslog restart`** or **`systemctl restart rsyslog.service`**).
- If the service is running, you should be able to see that the service is listening on the ports that you enabled (UDP and/or TCP ports 514). Run the `netstat` command as follows to see that the `rsyslogd` daemon is listening on IPv4 and IPv6 ports 514 for both UDP and TCP services:

```
 # netstat -tupln | grep 514
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      25341/rsyslogd
tcp        0      0 :::514                  :::*                    LISTEN      25341/rsyslogd
udp        0      0 0.0.0.0:514             0.0.0.0:*                           25341/rsyslogd
udp        0      0 :::514                  :::*                                25341/rsyslogd
```

Watching logs with logwatch

The `logwatch` service runs in most Linux systems that do system logging with `rsyslog`. Because logs on busy systems can become very large over time, it doesn't take long for there to be too many messages for a system administrator to watch every message in every log. To install the `logwatch` facility, enter the following:

```
# yum install logwatch
```

What `logwatch` does is gather messages once each night that look like they might represent a problem, put them in an email message, and send it to any email address the administrator chooses. To enable `logwatch`, all you have to do is install the `logwatch` package.

The `logwatch` service runs from a `cron` job (`0logwatch`) placed in `/etc/cron.daily`. The `/etc/logwatch/conf/logwatch.conf` file holds local settings. The default options used to gather log messages are set in the `/usr/share/logwatch/default.conf/logwatch.conf` file.

Some of the default settings define the location of log files (`/var/log`), location of the temporary directory (`/var/cache/logwatch`), and the recipient of the daily `logwatch` email (the local `root` user). Unless you expect to log in to the server to read `logwatch` messages, you probably want to change the `MailTo` setting in the `/etc/logwatch/conf/logwatch.conf` file:

```
MailTo = chris@example.com
```

Look in `/usr/share/logwatch/default.conf/logwatch.conf` for other settings to change (such as detail level or the time range for each report). Then make your additions to `/etc/logwatch/conf/logwatch.conf` as mentioned.

When the service is enabled (which it is just by installing the `logwatch` package), you will see a message each night in the root user's mailbox. When you are logged in as root, you can use the old `mail` command to view the root user's mailbox:

```
# mail
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 2 messages 2 new
>N  1 logwatch@abc.ex  Sun Feb 15 04:02 45/664   "Logwatch for abc"
    2 logwatch@abc.ex  Mon Feb 16 04:02 45/664   "Logwatch for abc"
& 1
& x
```

In mail, you should see email messages from `logwatch` run each day (here at 4:02 a.m.). Type the number of the message that you want to view and page through it with the spacebar or line by line by pressing Enter. Type **x** to exit when you are finished.

The kind of information that you see includes kernel errors, installed packages, authentication failures, and malfunctioning services. Disk space usage is reported, so you can see if your storage is filling up. Just by glancing through this `logwatch` message, you should get an idea whether sustained attacks are under way or if some repeated failures are taking place.

Checking System Resources with sar

The System Activity Reporter (`sar`) is one of the oldest system monitoring facilities created for early UNIX systems—predating Linux by a few decades. The `sar` command itself can display system activity continuously, at set intervals (every second or two), and display it on the screen. It can also display system activity data that was gathered earlier.

The `sar` command is part of the `sysstat` package. When you install `sysstat` and enable the `sysstat` service, your system immediately begins gathering system activity data that can be reviewed later using certain options to the `sar` command.

```
# systemclt enable sysstat
# systemctl start sysstat
```

To read the data in the `/var/log/sa/sa??` files, you can use some of the following `sar` commands:

```
# sar -u | less
Linux 5.3.8-200.fc30.x86_64 (fedora30host) 11/28/2019   _x86_64_  (1 CPU)

23:27:46     LINUX RESTART (1 CPU)

11:30:05 PM  CPU       %user   %nice %system    %iowait    %steal      %idle
11:40:06 PM  all       0.90    0.00    1.81       1.44       0.28      95.57
Average:     all       0.90    0.00    1.81       1.44       0.28      95.57
```

The `-u` option shows CPU usage. By default, the output starts at midnight on the current day and then shows how much processing time is being consumed by different parts of the system. The output continues to show the activity every 10 minutes until the current time is reached.

To see disk activity output, run the `sar -d` command. Again, output comes in 10-minute intervals starting at midnight.

```
# sar -d | less
Linux 5.3.8-200.fc30.x86_64 (fedora30host)  11/28/2019 _x86_64_ (1 CPU)

23:27:46     LINUX RESTART  (1 CPU)

11:30:05 PM       DEV   tps   rkB/s   wkB/s   areq-sz    aqu-sz  await...
11:40:06 PM     dev8-0 49.31 5663.94   50.38    115.89      0.03   1.00
11:40:06 PM   dev253-0 48.99 5664.09    7.38    115.78      0.05   0.98
11:40:06 PM   dev253-1 10.84    0.01   43.34      4.00      0.04   3.29
Average:        dev8-0 49.31 5663.94   50.38    115.89      0.03   1.00
Average:      dev253-0 48.99 5664.09    7.38    115.78      0.05   0.98
Average:      dev253-1 10.84    0.01   43.34      4.00      0.04   3.29
```

If you want to run `sar` activity reports live, you can do that by adding counts and time intervals to the command line, as shown here:

```
# sar -n DEV 5 2
Linux 5.3.8-200.fc30.x86_64 (fedora30host)  11/28/2019  _x86_64_  (1 CPU)
11:19:36 PM IFACE rxpck/s txpck/s  rxkB/s  txkB/s rxcmp/s txcmp/s...
11:19:41 PM    lo    5.42    5.42    1.06    1.06    0.00    0.00...
11:19:41 PM  ens3    0.00    0.00    0.00    0.00    0.00    0.00...
...
Average: IFACE rxpck/s txpck/s rxkB/s txkB/ rxcmp/s txcmp/s rxmcst/s
Average:    lo    7.21    7.21   1.42  1.42    0.00    0.00      0.00
Average:  ens3    0.00    0.00   0.00  0.00    0.00    0.00      0.00
Average: wlan0    4.70    4.00   4.81  0.63    0.00    0.00      0.00

Average:  pan0    0.00    0.00   0.00  0.00    0.00    0.00      0.00
Average:  tun0    3.70    2.90   4.42  0.19    0.00    0.00      0.00
```

With the `-n Dev` example just shown, you can see how much activity came across the different network interfaces on your system. You can see how many packets were transmitted and received and how many KB of data were transmitted and received. In that example, samplings of data were taken every 5 seconds and repeated twice.

Refer to the `sar`, `sadc`, `sa1`, and `sa2` man pages for more information on how `sar` data can be gathered and displayed.

Checking System Space

Although `logwatch` can give you a daily snapshot of space consumption on your system disks, the `df` and `du` commands can help you immediately see how much disk space is available. The following sections show examples of those commands.

Displaying system space with df

You can display the space available in your filesystems using the `df` command. To see the amount of space available on all of the mounted filesystems on your Linux computer, type `df` with no options:

```
$ df
Filesystem  1k-blocks     Used  Available  Use%   Mounted on
/dev/sda3    30645460  2958356   26130408   11%   /
/dev/sda2       46668     8340      35919   19%   /boot
...
```

This example output shows the space available on the hard disk partition mounted on the / (root) directory (`/dev/sda1`) and `/boot` partition (`/dev/sda2`). Disk space is shown in 1KB blocks. To produce output in a more human-readable form, use the `-h` option:

```
$ df -h
Filesystem            Size  Used  Avail  Use%   Mounted on
/dev/sda3              29G  2.9G    24G   11%   /
/dev/sda2              46M  8.2M    25M   19%   /boot
...
```

With the `df -h` option, output appears in a friendlier megabyte or gigabyte listing. Other options with `df` enable you to do the following:

- Print only filesystems of a particular type (`-t type`).
- Exclude filesystems of a particular type (`-x type`). For example, type `df -x tmpfs -x devtmpfs` to exclude temporary filesystem types (limiting output to filesystems that represent real storage areas).
- Include filesystems that have no space, such as `/proc` and `/dev/pts` (`-a`).
- List only available and used inodes (`-i`).
- Display disk space in certain block sizes (`--block-size=#`).

Checking disk usage with du

To find out how much space is being consumed by a particular directory (and its subdirectories), use the `du` command. With no options, `du` lists all directories below the current directory, along with the space consumed by each directory. At the end, `du` produces total disk space used within that directory structure.

The `du` command is a good way to check how much space is being used by a particular user (`du /home/jake`) or in a particular filesystem partition (`du /var`). By default, disk space

is displayed in 1KB block sizes. To make the output friendlier (in kilobytes, megabytes, and gigabytes), use the `-h` option as follows:

```
$ du -h /home/jake
114k    /home/jake/httpd/stuff
234k    /home/jake/httpd
137k    /home/jake/uucp/data
701k    /home/jake/uucp
1.0M    /home/jake
```

The output shows the disk space used in each directory under the home directory of the user named `jake` (`/home/jake`). Disk space consumed is shown in kilobytes (`k`) and megabytes (`M`). The total space consumed by `/home/jake` is shown on the last line. Add the `–s` option to see total disk space used for a directory and its subdirectories.

Finding disk consumption with find

The `find` command is a great way to find file consumption of your hard disk using a variety of criteria. You can get a good idea of where disk space can be recovered by finding files that are over a certain size or were created by a particular person.

Note

You must be the root user to run this command effectively, unless you are just checking your personal files. If you are not the root user, there are many places in the filesystem for which you do not have permission to check. Regular users can usually check their own home directories but not those of others.

In the following example, the `find` command searches the root filesystem (`/`) for any files owned by the user named `jake` (`-user jake`) and prints the filenames. The output of the `find` command is organized in a long listing in size order (`ls -ldS`). Finally, that output is sent to the file `/tmp/jake`. When you view the file `/tmp/jake` (for example, `less /tmp/jake`), you will find all of the files that are owned by the user `jake` listed in size order. Here is the command line:

```
# find / -xdev -user jake -print | xargs ls -ldS > /tmp/jake
```

Tip

The `-xdev` option prevents filesystems other than the selected filesystem from being searched. This is a good way to cut out lots of junk that may be output from the `/proc` filesystem. It can also keep large, remotely mounted filesystems from being searched.

Here's another example, except that instead of looking for a user's files, we're looking for files larger than 100 kilobytes (`-size +100M`):

```
# find / -xdev -size +100M | xargs ls -ldS > /tmp/size
```

You can save yourself lots of disk space just by removing some of the largest files that are no longer needed. In this example, you can see that large files are sorted by size in the `/tmp/size` file.

Managing Servers in the Enterprise

Most of the server configuration covered in this book describes how to install systems manually and work directly on host computers. Having to set up each host individually would be far too inefficient for modern data centers consisting of dozens, hundreds, or even thousands of computers. To make the process of setting up Linux servers in a large data center more efficient, some of the following are employed:

Automated deployments One way to install systems without having to step through a manual install process is with PXE booting. By setting up a PXE server and booting a computer on that network from a PXE-enabled network interface card, you can start a full install of that system simply by booting the system. Once the install is done, the system can reboot to run from the installed system.

Generic host systems By making your host systems as generic as possible, individual installation, configuration, and upgrades can be greatly simplified. This can be automated in layers, where the base system is installed by PXE booting, configuration is done through features such as cloud-int, and applications can bring along their own dependencies when they run. On the application level, this can be done by running an application from inside a virtual machine or container. When the application is done running, it can be discarded without leaving its dependent software on the host.

Separation of management and worker systems Instead of individually managing host systems, a separate platform can offer a way to manage large sets of systems. To do this, a platform such as OpenStack or OpenShift can have management nodes (in some cases called *control plane* or *master nodes*) manage the machines where the workload actually runs (sometimes called *workers*, *slaves*, or just *nodes*). This separation of tasks by host type makes it possible to have applications deployed on any available worker that meets the needs of the application (such as available memory or CPU).

Keep in mind that understanding how individual applications are configured and services are run is still the foundation for these more advanced ways of managing data center resources. Although in-depth coverage of enterprise deployment and monitoring tools is outside the scope of this book, refer to Part VI, "Engaging with Cloud Computing," for an introduction to how different Linux-based cloud platforms manage these issues.

Summary

Although many different types of servers are available with Linux systems, the basic procedure for installing and configuring a server is essentially the same. The normal course

of events is to install, configure, start, secure, and monitor your servers. Basic tasks that apply to all servers include using networking tools (particularly SSH tools) to log in, copy files, or execute remote commands.

Because an administrator can't be logged in watching servers all the time, tools for gathering data and reviewing the log data later are very important when administering Linux servers. The `rsyslog` facility can be used for local and remote logging. The `sar` facility gathers live data or plays back data gathered earlier at 10-minute intervals. Cockpit lets you watch CPU, memory, disk, and networking activity live from a web user interface. To watch disk space, you can run `df` and `du` commands.

The skills described in this chapter are designed to help you build a foundation to do enterprise-quality system administration in the future. Although these skills are useful, to manage many Linux systems at the same time, you need to extend your skills by using automating deployment and monitoring tools, as described in the cloud computing section of this book.

Although it is easy to set up networking to reach your servers in simple, default cases, more complex network configuration requires a knowledge of networking configuration files and related tools. The next chapter describes how to set up and administer networking in Linux.

Exercises

The exercises in this section cover some of the basic tools for connecting to and watching over your Linux servers. As usual, you can accomplish the tasks here in several ways. So, don't worry if you don't go about the exercises in the same way as shown in the answers, as long as you get the same results. If you are stuck, solutions to the tasks are shown in Appendix B.

Some of the exercises assume that you have a second Linux system available that you can log in to and try different commands. On that second system, you need to make sure that the `sshd` service is running, that the firewall is open, and that `ssh` is allowed for the user account that you are trying to log in to (root is often blocked by `sshd`).

If you have only one Linux system, you can create an additional user account and simply simulate communications with another system by connecting to the name `localhost` instead, as shown in this example:

```
# useradd joe
# passwd joe
# ssh joe@localhost
```

1. Using the `ssh` command, log in to another computer (or the local computer) using any account to which you have access. Enter the password when prompted.
2. Using remote execution with the `ssh` command, display the contents of a remote `/etc/system-release` file and have its contents displayed on the local system.

3. Use the `ssh` command to use X11 forwarding to display a `gedit` window on your local system; then save a file in the remote user's home directory.
4. Recursively copy all of the files from the `/usr/share/selinux` directory on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files are updated to the time on the local system when they are copied.
5. Recursively copy all of the files from the `/usr/share/logwatch` directory on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files from the remote system are maintained on the local system.
6. Create a public/private key pair to use for SSH communications (no passphrase on the key), copy the public key file to a remote user's account with `ssh-copy-id`, and use key-based authentication to log in to that user account without having to enter a password.
7. Create an entry in `/etc/rsyslog.conf` that stores all authentication messages (`authpriv`) info level and higher into a file named `/var/log/myauth`. From one terminal, watch the file as data comes into it, and in another terminal, try to `ssh` into your local machine as any valid user with a bad password.
8. Use the `du` command to determine the largest directory structures under `/usr/share`, sort them from largest to smallest, and list the top ten of those directories in terms of size.
9. Use the `df` command to show the space that is used and available from all of the filesystems currently attached to the local system but exclude any `tmpfs` or `devtmpfs` filesystems.
10. Find any files in the `/usr` directory that are more that 10MB in size.

CHAPTER 14

Administering Networking

IN THIS CHAPTER

Automatically connecting Linux to a network

Using NetworkManager for simple network connectivity

Configuring networking from the command line

Working with network configuration files

Configuring routing, DHCP, DNS, and other networking infrastructure features for the enterprise

Connecting a single desktop system or laptop to a network, particularly one that connects to the Internet, has become so easy that I have put off writing a full chapter on Linux networking until now. If you are trying to connect your Fedora, RHEL, Ubuntu, or another Linux desktop system to the Internet, here's what you can try given an available wired or wireless network interface:

Wired network If your home or office has a wired Ethernet port that provides a path to the Internet and your computer has an Ethernet port, use an Ethernet cable to connect the two ports. After you turn on your computer, boot up Linux and log in. Clicking the NetworkManager icon on the desktop should show you that you are connected to the Internet or allow you to connect with a single click.

Wireless network For a wireless computer running Linux, log in and click the NetworkManager icon on the desktop. From the list of wireless networks that appear, select the one you want and, when prompted, enter the required password. Each time you log in from that computer from the same location, it automatically connects to that wireless network.

If either of those types of network connections works for you, and you are not otherwise curious about how networking works in Linux, that may be all you need to know. However, what if your Linux system doesn't automatically connect to the Internet? What if you want to configure your desktop to talk to a private network at work (VPN)? What if you want to lock down network settings on your server or configure your Linux system to work as a router?

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

In this chapter, topics related to networking are divided into networks for desktops, servers, and enterprise computing. The general approach to configuring networking in these three types of Linux systems is as follows:

Desktop/laptop networking On desktop or laptop systems, NetworkManager runs by default in order to manage network interfaces. With NetworkManager, you can automatically accept the address and server information that you need to connect to the Internet. However, you can also set address information manually. You can configure things such as proxy servers or virtual private network connections to allow your desktop to work from behind an organization's firewall or to connect through a firewall, respectively.

Server networking Although NetworkManager originally worked best on desktop and laptop network configurations, it now works extremely well on servers. Today, features that are useful for configuring servers, such as Ethernet channel bonding and configuring aliases, can be found in NetworkManager.

Enterprise networking Explaining how to configure networking in a large enterprise can fill several volumes itself. However, to give you a head start using Linux in an enterprise environment, I discuss basic networking technologies, such as DHCP and DNS servers, which make it possible for desktop systems to connect to the Internet automatically and find systems based on names and not just IP addresses.

Configuring Networking for Desktops

Whether you connect to the Internet from Linux, Windows, a smartphone, or any other kind of network-enabled device, certain things must be in place for that connection to work. The computer must have a network interface (wired or wireless), an IP address, an assigned DNS server, and a route to the Internet (identified by a gateway device).

Before I discuss how to change your networking configuration in Linux, let's look at the general activities that occur when Linux is set to connect to the Internet automatically with NetworkManager:

Activate network interfaces NetworkManager looks to see what network interfaces (wired or wireless) are set to start. By default, external interfaces are usually set to start automatically using DHCP, although static names and addresses can be set at install time instead.

Request DHCP service The Linux system acts as a DHCP client to send out a request for DHCP service on each enabled interface. It uses the MAC address of the network interface to identify itself in the request.

Get response from DHCP server A DHCP server, possibly running on a wireless router, cable modem, or other device providing a route to the Internet from your location, responds to the DHCP request. It can provide lots of different types of information to the DHCP client. That information probably contains at least the following:

IP address The DHCP server typically has a range of Internet Protocol (IP) addresses that it can hand out to any system on the network that requests an address. In more secure environments, or one in which you want to be sure that specific machines get specific addresses, the DHCP server provides a specific IP address to requests from specific MAC addresses. (MAC addresses are made to be unique among all network interface cards and are assigned by the manufacturer of each card.)

Subnet mask When the DHCP client is assigned an IP address, the accompanying subnet mask tells that client which part of the IP address identifies the subnetwork and which identifies the host. For example, an IP address of 192.168.0.100 and subnet mask of 255.255.255.0 tell the client that the network is 192.168.0 and the host part is 100.

Lease time When an IP address is dynamically allocated to the DHCP client (Linux system), that client is assigned a lease time. The client doesn't own that address but must lease it again when the time expires and request it once again when the network interface restarts. Usually, the DHCP server remembers the client and assigns the same address when the system starts up again or asks to renew the lease. The default lease time is typically 86,400 seconds (24 hours) for IPV4 addresses. The more plentiful IPV6 addresses are assigned for 2,592,000 seconds (30 days) by default.

Domain name server Because computers like to think in numbers (such as IP addresses like 192.168.0.100) and people tend to think in names (such as the hostname `www.example.com`), computers need a way to translate hostnames into IP addresses and sometimes the opposite as well. The domain name system (DNS) was designed to handle that problem by providing a hierarchy of servers to do name-to-address mapping on the Internet. The location of one or more DNS servers (usually two or three) is usually assigned to the DHCP client from the DHCP host.

Default gateway Although the Internet has one unique namespace, it is actually organized as a series of interconnected subnetworks. In order for a network request to leave your local network, it must know what node on your network provides a route to addresses outside of your local network. The DHCP server usually provides the "default gateway" IP address. By having network interfaces on both your subnetwork and the next network on the way to the ultimate destination of your communication, a gateway can route your packets to their destination.

Other information A DHCP server can be configured to provide all kinds of information to help the DHCP client. For example, it can provide the location of an NTP server (to sync time between clients), font server (to get fonts for your X display), IRC server (for online chats), or print server (to designate available printers).

Update local network settings After the settings are received from the DHCP server, they are implemented as appropriate on the local Linux system. For example, the IP address is set on the network interface, the DNS server entries are added to the local `/etc/resolv.conf` file (by NetworkManager), and the lease time is stored by the local system so it knows when to request that the lease be renewed.

All of the steps just described typically happen without your having to do anything but turn on your Linux system and log in. Now suppose that you want to be able to verify your network interfaces or change some of those settings. You can do that using the tools described in the next sections.

Checking your network interfaces

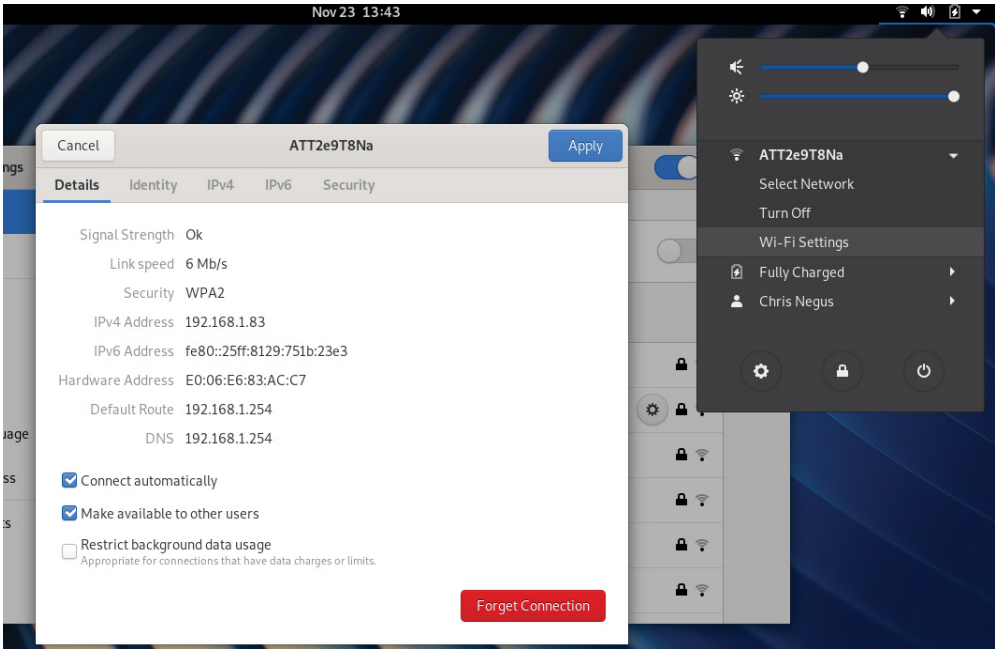
There are both graphical and command-line tools for viewing information about your network interfaces in Linux. From the desktop, NetworkManager tools and the Cockpit web user interface are good places to start.

Checking your network from NetworkManager

The easiest way to check the basic setting for a network interface is to open the pull-down menu at the upper-right corner of your desktop and select your active network interface. Figure 14.1 shows the Wi-Fi settings for an active network on a Fedora GNOME 3 desktop.

FIGURE 14.1

Checking network interfaces with NetworkManager

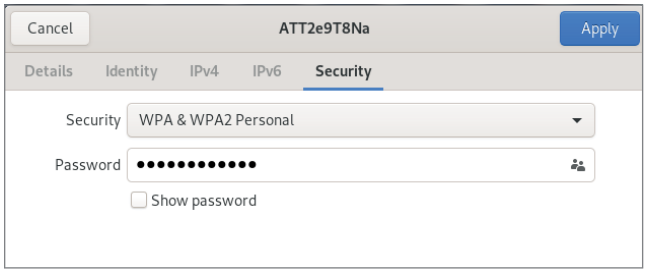


As you can see in Figure 14.1, both IPv4 and IPv6 addresses are assigned to the interface. The IP address 192.168.1.254 offers both a DNS service and a route to external networks.

To see more about how your Linux system is configured, click one of the tabs at the top of the window. For example, Figure 14.2 shows the Security tab, where you can select the type of security connection to the network and set the password needed to connect to that network.

FIGURE 14.2

Viewing network settings with NetworkManager



Checking your network from Cockpit

Provided you have enabled Cockpit, you can see and change information about your network interfaces through your web browser. On your local system, you open `https://localhost:9090/network` to go directly to the Cockpit Networking page for your local system. Figure 14.3 shows an example of this.

From the Cockpit Networking page, you can see information about all of your network interfaces at once. In this case, there are three network interfaces: `wlp2s0` (the active wireless interface), `enp4s0` (an inactive wired interface), and `virbr0` (an inactive interface into the network connected to any virtual machines running on the local system).

At the top of the Cockpit Networking page, you can see data being sent from and received on the local system. Select a network interface to see a page displaying activities for that particular interface.

Select Firewall to see the list of services that are allowed into your system. For example, Figure 14.4 shows that UDP ports are open for three services (DHCPv6 client, Multicast DNS, and Samba Client). DHCPv6 lets the system acquire an IPv6 address from the network. Multicast DNS and Samba Client services can allow the autodetection of printers, share file systems, and a variety of devices and shared resources.

The only TCP service shown here as open is ssh. With the ssh service (TCP port 22) open, the sshd service running on your local system is available for remote users to log into your local system.

FIGURE 14.3

Viewing and changing network settings from Cockpit

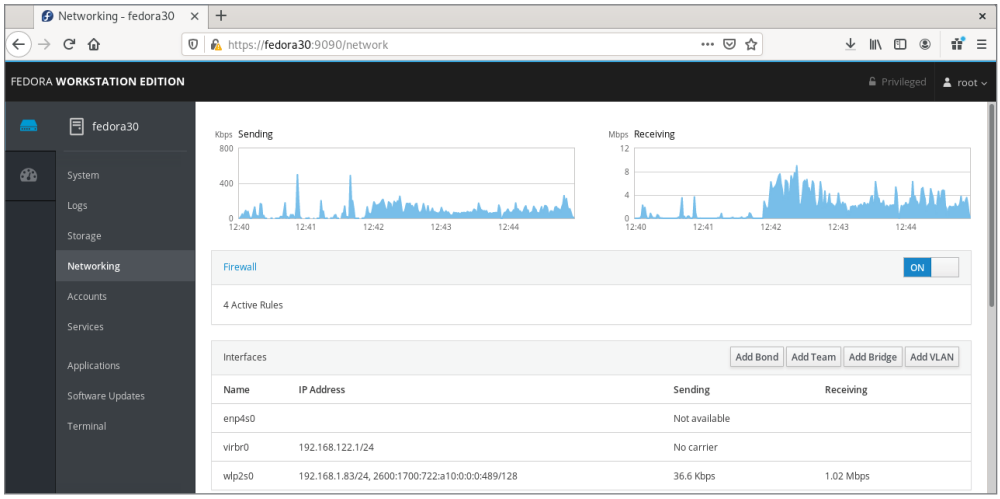
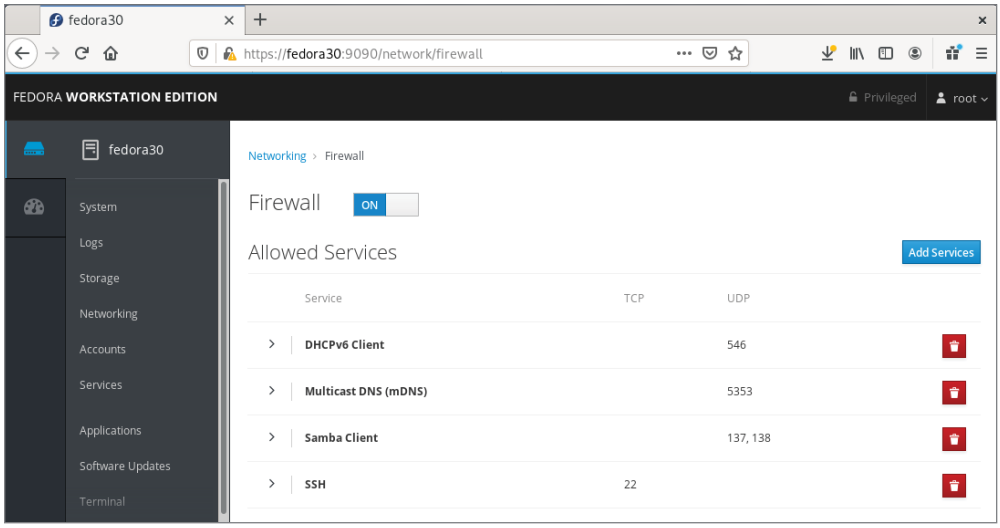


FIGURE 14.4

View services that are accessible through the firewall from Cockpit



The fact that those ports are open doesn't necessarily mean that the services are running. However, if they are running, the computer's firewall will allow access to them.

More advanced features available from the Cockpit Networking page allow you to add bonds, teams, bridges, and VLANs to your local network interfaces.

Checking your network from the command line

To get more detailed information about your network interfaces, try running some commands. There are commands that can show you information about your network interfaces, routes, hosts, and traffic on the network.

Viewing network interfaces

To see information about each network interface on your local Linux system, enter the following:

```
# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
        state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp4s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
        qdisc fq_codel state DOWN group default qlen 1000
    link/ether 30:85:a9:04:9b:f9 brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
        qdisc mq state UP group default qlen 1000
    link/ether e0:06:e6:83:ac:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.83/24 brd 192.168.1.255 scope global
        dynamic noprefixroute wlp2s0
       valid_lft 78738sec preferred_lft 78738sec
    inet6 2600:1700:722:a10::489/128 scope global dynamic
noprefixroute
       valid_lft 5529sec preferred_lft 5229sec
    inet6 fe80::25ff:8129:751b:23e3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
       qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:0c:69:0a brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
       valid_lft forever preferred_lft forever
```

The `ip addr show` output displays information about your network interfaces, in this case from a laptop running Fedora 30. The `lo` entry in the first line of the output shows the loopback interface, which is used to allow network commands run on the local system

to connect to the local system. The IP address for localhost is 127.0.0.1/8 (the /8 is CIDR notation, indicating that 127.0 is the network number and 0.1 is the host number). Add a `-s` option (`ip -s addr show`) to see statistics of packet transmissions and errors associated with each interface.

In this case, the wired Ethernet interface (`enp4s0`) is down (no cable), but the wireless interface is up (`wlp2s0`). The MAC address on the wireless interface (`wlp2s0`) is e0:06:e6:83:ac:c7 and the Internet (IPv4) address is 192.168.1.83. An IPv6 address is also enabled.

Older versions of Linux are used to assign more generic network interface names, such as `eth0` and `wlan0`. Now interfaces are named by their locations on the computer's bus. For example, the first port on the network card seated in the third PCI bus for a Fedora system is named `p3p1`. The first embedded Ethernet port would be `em1`. Wireless interfaces sometimes appear using the name of the wireless network as the device name.

Another popular command for seeing network interface information is the `ifconfig` command. By default, `ifconfig` shows similar information to that of `ip addr`, but `ifconfig` also shows the number of packets received (RX) and transmitted (TX) by default, as well as the amount of data and any errors or dropped packets:

```
# ifconfig wlp2s0
wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.83  netmask 255.255.255.0
             broadcast 192.168.1.255
        inet6 2600:1700:722:a10:b55a:fca6:790d:6aa6
             prefixlen 64  scopeid 0x0<global>
        inet6 fe80::25ff:8129:751b:23e3
             prefixlen 64  scopeid 0x20<link>
        inet6 2600:1700:722:a10::489
             prefixlen 128  scopeid 0x0<global>
        ether e0:06:e6:83:ac:c7  txqueuelen 1000  (Ethernet)
        RX packets 208402  bytes 250962570 (239.3 MiB)
        RX errors 0  dropped 4  overruns 0  frame 0
        TX packets 113589  bytes 13240384 (12.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
Checking connectivity to remote systems
```

To make sure that you can reach systems that are available on the network, you can use the `ping` command. As long as the computer responds to `ping` requests (not all do), you can use `ping` to send packets to that system in a way that asks them to respond. Here is an example:

```
$ ping host1
PING host1 (192.168.1.15 ) 56(84) bytes of data.
64 bytes from host1 (192.168.1.15 ): icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from host1 (192.168.1.15 ): icmp_seq=2 ttl=64 time=0.044 ms
^C
```

```
--- host1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1822ms
rtt min/avg/max/mdev = 0.044/0.053/0.062/0.009 ms
```

The `ping` command shown here continuously pings the host named `host1`. After a few pings, press Ctrl+C to end the pings, and the last few lines show you how many of the `ping` requests succeeded.

You could have used the IP address (192.168.1.15, in this case) to see that you could reach the system. However, using the hostname gives you the additional advantage of knowing that your name-to-IP-address translation (being done by your DNS server or local hosts file) is working properly as well. In this case, however, `host1` appeared in the local `/etc/hosts` file.

Checking routing information

Routing is the next thing that you can check with respect to your network interfaces. The following snippets show you how to use the `ip` and `route` commands to do that:

```
# ip route show
default via 192.168.122.1 dev ens3 proto dhcp metric 20100
192.168.122.0/24 dev ens3 proto kernel scope link src 192.168.122.194
metric 100
```

The `ip route show` command example illustrates that the 192.168.122.1 address provides the route to the host from a RHEL 8 VM over the `ens3` network interface. Communications to any address in the 192.168.122.0/24 range from the VM (192.168.122.194) goes over that interface. The `route` command can provide similar information:

```
# route
Kernel IP routing table
Destination   Gateway    Genmask        Flags Metric Ref Use Iface
default       homeportal 0.0.0.0        UG    600    0      0 wlp2s0
192.168.1.0   0.0.0.0    255.255.255.0  U     600    0      0 wlp2s0
192.168.122.0 0.0.0.0    255.255.255.0  U     0      0      0 virbr0
```

The output from the kernel routing table is from a Fedora system with a single active external network interface. The wireless network interface card is on PCI slot 2, port 1 (`wlp2`). Any packets destined for the 192.168.1 network use the `wlp2` NIC. Packets destined for any other location are forwarded to the gateway system at 192.168.1.0. That system represents my router to the Internet. Here's a more complex routing table:

```
# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref Use Iface
default       gateway      0.0.0.0         UG    600    0      0 wlp3s0
10.0.0.0      vpn.example. 255.0.0.0       U     50     0      0 tun0
10.10.135.0   0.0.0.0      255.255.217.0   U     50     0      0 tun0
vpn.example.  gateway      255.255.255.255 UGH   600    0      0 wlp3s0
```

```
172.17.0.0    0.0.0.0      255.255.0.0     U     0      0     0 docker0
192.168.1.0   *            255.255.255.0   U     600    0     0 wlp3s0
```

In the route example just shown, there is a wireless interface (`wlp3s0`) as well as an interface representing a virtual private network (VPN) tunnel. A VPN provides a way to have encrypted, private communications between a client and a remote network over an insecure network (such as the Internet). Here the tunnel goes from the local system over the `wlan0` interface to a host named `vpn.example.com` (some of the name is truncated).

All communication to the 192.168.1.0/24 network still goes directly over the wireless LAN. However, packets destined for the 10.10.135.0/24 and 10.0.0.0/8 networks are routed directly to `vpn.example.com` for communication with hosts on the other side of the VPN connection over the tunneled interface (`tun0`).

A special route is set up for communications to containers (`docker0`) running on the local system on network 172.17.0.0. All other packets go to the default route via the address 192.168.1.0. As for the flags shown in the output, a `U` says the route is up, a `G` identifies the interface as a gateway, and an `H` says the target is a host (as is the case with the VPN connection).

So far, I have shown you the routes to leave the local system. If you want to follow the entire route to a host from beginning to end, you can use the `traceroute` command (`dnf install traceroute`). For example, to trace the route a packet takes from your local system to the `google.com` site, type the following `traceroute` command:

```
# traceroute google.com
traceroute to google.com (74.125.235.136), 30 hops max, 60 byte pkts
...
 7  rrcs-70-62-95-197.midsouth.biz.rr.com (70.62.95.197)  ...
 8  ge-2-1-0.rlghncpop-rtr1.southeast.rr.com (24.93.73.62)  ...
 9  ae-3-0.cr0.dca10.tbone.rr.com (66.109.6.80) ...
10  107.14.19.133 (107.14.19.133)  13.662  ms  ...
11  74.125.49.181 (74.125.49.181)  13.912  ms ...
12  209.85.252.80 (209.85.252.80)  61.265  ms ...
13  66.249.95.149 (66.249.95.149)  18.308  ms ...
14  66.249.94.22 (66.249.94.22)  18.344  ms ...
15  72.14.239.83 (72.14.239.83)  85.342  ms ...
16  64.233.174.177 (64.233.174.177)  167.827  ms ...
17  209.85.255.35 (209.85.255.35)  169.995  ms  ...
18  209.85.241.129 (209.85.241.129)  170.322  ms  ...
19  nrt19s11-in-f8.1e100.net (74.125.235.136)  169.360ms  ...
```

I truncated some of the output to drop off some of the initial routes and the amount of time (in milliseconds) that the packets were taking to traverse each route. Using `traceroute`, you can see where the bottlenecks are along the way if your network communication is stalling.

Viewing the host and domain names

To see the hostname assigned to the local system, type `hostname`. To just see the domain portion of that name, use the `dnsdomainname` command:

```
# hostname
spike.example.com
# dnsdomainname
example.com
```

Configuring network interfaces

If you don't want to have your network interfaces assigned automatically from a DHCP server (or if there is no DHCP server), you can configure network interfaces manually. This can include assigning IP addresses, the locations of DNS servers and gateway machines, and routes. This basic information can be set up using NetworkManager.

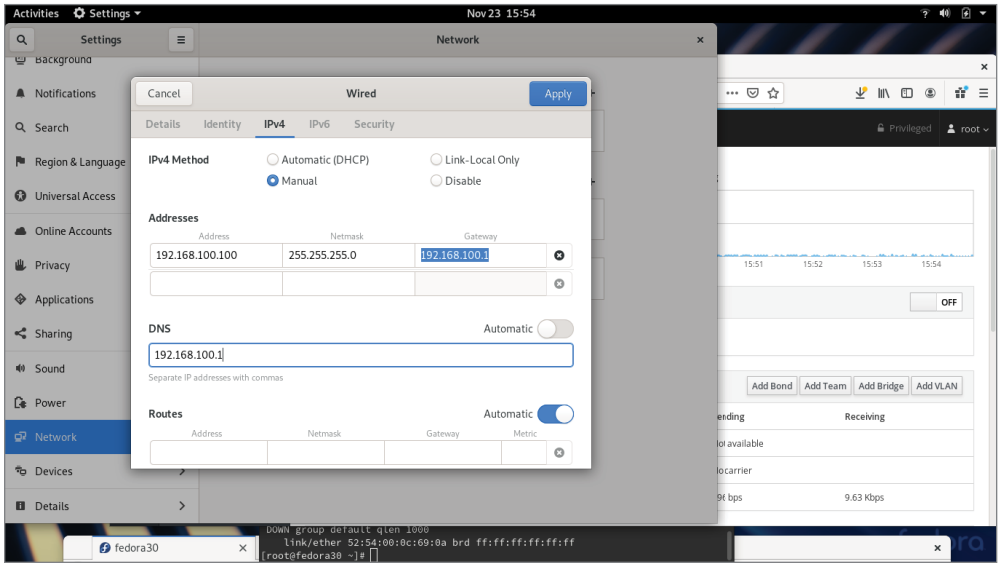
Setting IP addresses manually

To change the network configuration for your wired network interface through NetworkManager, do the following:

1. Select the Settings icon from the drop-down menu in the upper-right corner of the desktop and select Network.
2. Assuming that you have a wired NIC that is not yet in use, select the settings button (small gear icon) next to the interface that you want to change.
3. Choose IPv4 and change the IPv4 Method setting from Automatic (DHCP) to Manual.
4. Fill in the following information (only Address and Netmask are required):
   a. **Address**: The IP address that you want to assign to your local network interface. For example, 192.168.100.100.
   b. **Netmask**: The subnetwork mask that defines which part of the IP address represents the network and which part identifies the host. For example, a netmask of 255.255.255.0 would identify the network portion of the previous address as 192.168.100 and the host portion as 100.
   c. **Gateway**: The IP address of the computer or device on the network that acts as the default route. The default route will route packets from the local network to any address that is not available on the local network or via some other custom route.
   d. **DNS servers**: Fill in the IP addresses for the system providing DNS service to your computer. If there is more than one DNS server, add the others in a comma-separated list of servers.
5. Click the Apply button. The new information is saved, and the network is restarted using the new information. Figure 14.5 shows an example of those network settings.

FIGURE 14.5

Changing network settings with NetworkManager



Setting IP address aliases

You can attach multiple IP addresses to a single network interface. In the same NetworkManager screen, this is done by simply filling in a subsequent Addresses box and adding the new IP address information. Here are a few things you should know about adding address aliases:

- A netmask is required for each address, but a gateway is not required.
- The Apply button stays grayed out until you include valid information in the fields.
- The new address does not have to be on the same subnetwork as the original address, although it is listening for traffic on the same physical network.

After adding the address 192.168.100.103 to my wired interface, running `ip addr show enp4s0` displays the following indication of the two IP addresses on the interface:

```
2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
state UP group default qlen 1000
    link/ether 30:85:a9:04:9b:f9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.100/24 brd 192.168.100.255 scope
          global noprefixroute enp4s0
       valid_lft forever preferred_lft forever
    inet 192.168.100.103/24 brd 192.168.100.255 scope
          global secondary noprefixroute enp4s0
       valid_lft forever preferred_lft forever
```

For information on setting up aliases directly in configuration files, refer to the section "Setting alias network interfaces" later in this chapter.

Setting routes

When you request a connection to an IP address, your system looks through your routing table to determine the path on which to connect to that address. Information is sent in the form of packets. A packet is routed in the following different ways, depending on its destination:

- The local system is sent to the `lo` interface.
- A system on your local network is directed through your NIC directly to the intended recipient system's NIC.
- Any other system is sent to the gateway (router) that directs the packet on to its intended address on the Internet.

Of course, what I have just described here is one of the simplest cases. You may, in fact, have multiple NICs with multiple interfaces to different networks. You may also have multiple routers on your local network that provide access to other private networks. For example, suppose you have a router (or other system acting as a router) on your local network; you can add a custom route to that router via NetworkManager. Using the NetworkManager example shown previously, scroll down the page to view the Routes section. Then add the following information:

Address The network address of the subnetwork you route to. For example, if the router (gateway) will provide you access to all systems on the 192.168.200 network, add the address 192.168.200.0.

Netmask Add the netmask needed to identify the subnetwork. For example, if the router provides access to the Class C address 192.168.200, you could use the netmask 255.255.255.0.

Gateway Add the IP address for the router (gateway) that provides access to the new route. For example, if the router has an IP address on your 192.168.1 network of 192.168.1.199, add that address in this field.

Click Apply to apply the new routing information. You may have to restart the interface for this to take effect (for example, `ifup enp4s0`). Enter `route -n` to make sure the new routing information has been applied.

```
# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
0.0.0.0         192.168.100.1   0.0.0.0         UG    1024   0     0 p4p1
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0     0 p4p1

192.168.200.0   192.168.1.199   255.255.255.0   UG    1      0     0 p4p1
```

In the example just shown, you can see that the default gateway is 192.168.100.1. However, any packets destined for the 192.168.200 network are routed through the gateway host at IP address 192.168.1.199. Presumably that host has a network interface that faces the 192.168.200 network, and it is set up to allow other hosts to route through it to that network.

See the section "Setting custom routes" later in this chapter for information on how to set routes directly in configuration files.

Configuring a network proxy connection

If your desktop system is running behind a corporate firewall, you might not have direct access to the Internet. Instead, you might have to reach the Internet via a proxy server. Instead of allowing you full access to the Internet, a proxy server lets you make requests only for certain services outside of the local network. The proxy server then passes those requests on to the Internet or another network.

Proxy servers typically provide access to web servers (`http://` and `https://`) and FTP servers (`ftp://`). However, a proxy server that supports SOCKS can provide a proxy service for different protocols outside of the local network. (*SOCKS* is a network protocol made to allow client computers to access the Internet through a firewall.) You can identify a proxy server in NetworkManager and have communications for selected protocols go through that server (from the Settings window, select Network and then select Network Proxy).

Instead of identifying a proxy server to your network interfaces (via NetworkManager), you can configure your browser to use a proxy server directly by changing your Firefox preferences to use a proxy server. Here's how to define a proxy server from the Firefox window:

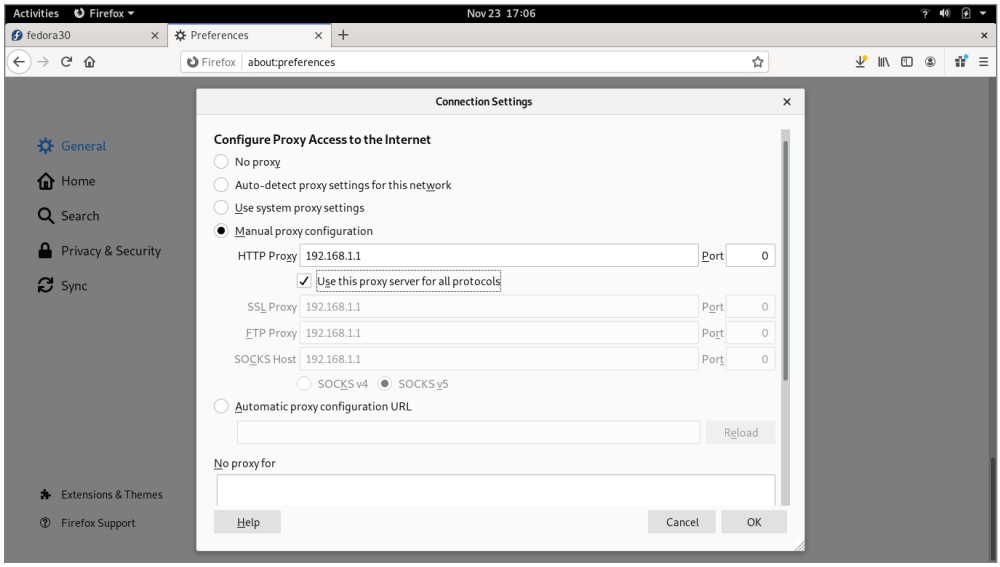
1. From Firefox, select Preferences. The Firefox Preferences window appears.
2. From the Firefox Preferences window, scroll down to Network Settings and select Settings.
3. From the Connection Settings window that appears, you can try to autodetect the proxy settings or, if you set the proxy in NetworkManager, you can choose to use system proxy settings. You can also select Manual Proxy Configuration, fill in the following information, and click OK.
    a. **HTTP Proxy**: The IP address of the computer providing the proxy service. This causes all requests for web pages (`http://` protocol) to be forwarded to the proxy server.
    b. **Port**: The port associated with the proxy service. By default, the port number is 3128, but it can differ.
    c. **Use this proxy server for all protocols**: Select this box to use the same proxy server and port associated with the HTTP proxy for all other service requests. This causes other proxy settings to be grayed out. (Instead of selecting this box, you can set those proxy services separately.)

d. **No Proxy for**: Add the hostname or IP address for any system that you want to be able to contact with Firefox directly without going through the proxy server. You don't need to add localhost and the local IP address (127.0.0.1) in this box, since those addresses are already set not to redirect.

Figure 14.6 shows an example of the Configure Proxy Access to the Internet window filled in to configure a connection to a proxy server located at IP address 192.168.1.1 for all protocols. After you click OK, all requests from the Firefox browser to locations outside of the local system are directed to the proxy server, which forwards those requests on to the appropriate server.

FIGURE 14.6

Setting up Firefox to use a proxy server



Configuring Networking from the Command Line

While NetworkManager does a great job of autodetecting wired networks or presenting you with lists of wireless networks, sometimes you need to abandon the NetworkManager GUI and commands or Cockpit to configure the features that you need. These are some of the networking features in RHEL and Fedora described in the coming sections:

Basic configuration: See how to use `nmtui` to configure basic networking with a menu-based interface from a shell. This tool provides an intuitive interface for configuring networking on servers that have no graphical interface for running GUI-based tools.

Configuration files: Understand configuration files associated with Linux networking and how to configure them directly.

Ethernet channel bonding: Set up Ethernet channel bonding (multiple network cards listening on the same IP address).

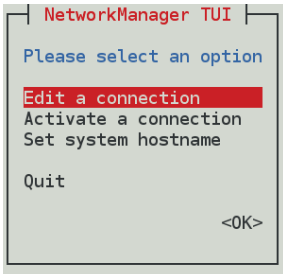
Configure networking with nmtui

Many servers don't have graphical interfaces available. So, if you want to configure networking, you must be able to do so from the shell. One way to do that is to edit networking configuration files directly. Another method is to use menu-based commands that let you press arrow and Tab keys to navigate and forms you fill in to configure your network interface.

The `nmtui` command (`yum install NetworkManager-tui`) provides a menu-based interface that runs in the shell. As root, enter **`nmtui`** to see a screen similar to the one presented in Figure 14.7.

FIGURE 14.7

Configuring networking with NetworkManager TUI



Use arrow keys and the Tab key to move around the interface. With the item you want to select highlighted, press Enter to select it. The interface is limited to modifying the following kinds of information: Edit or Activate a connection (network interface cards) and Set system hostname (hostname and DNS configuration).

Editing a NetworkManager TUI connection

From the NetworkManager TUI screen displayed, here is how to edit an existing connection.

1. **Edit a connection**: With "Edit a connection" highlighted, press Enter. A list of network devices (usually wired or wireless Ethernet cards) is displayed, along with any wireless networks to which you have connected in the past.
2. **Network devices**: Highlight one of the network devices (in my case, I chose a wired Ethernet interface) and press Enter.

3. **IPv4 Configuration**: Move to the IPv4 Configuration show button and press Enter. The Edit Connection window that appears lets you change information relating to the selected network device.
4. **Change to Manual**: You can leave the Profile Name and Device fields as they are. By default, Automatic is enabled. Automatic is what allows the network interface to come up automatically on the network if a DHCP service is available. To enter address and other information yourself, use the Tab key to highlight the Automatic field and press the spacebar; then use the arrow keys to highlight Manual and press Enter.
5. **Addresses**: Now fill in the address information (IP address and netmask). For example, 192.168.0.150/24 (where 24 is the CIDR equivalent for the 255.255.255.0 netmask).
6. **Gateway**: Type in the IP address for the computer or router that is supplying the route to the Internet.
7. **DNS servers**: Type in the IP addresses of either one or two DNS servers to tell the system where to go to translate hostnames you request into IP addresses.
8. **Search domains**: The Search domains entries are used when you request a host from an application without using a fully qualified domain name. For example, if you type `ping host1` with an `example.com` search path, the command would try to send ping packets to `host1.example.com`.
9. **Routing**: You can set custom routes by highlighting Edit in the Routing field and pressing Enter. Fill in the Destination/Prefix and Next Hop fields and select OK to save the new custom route.
10. **Other selections**: Of the other selections on the screen, consider setting "Never use this network for default route" if the network doesn't connect to wider networks and "Ignore automatically obtained routes" if you don't want those features to be set automatically from the network. Figure 14.8 shows the screen after Manual has been selected and the address information has been filled in.

Tab to the OK button and press the spacebar. Then click Quit to exit.

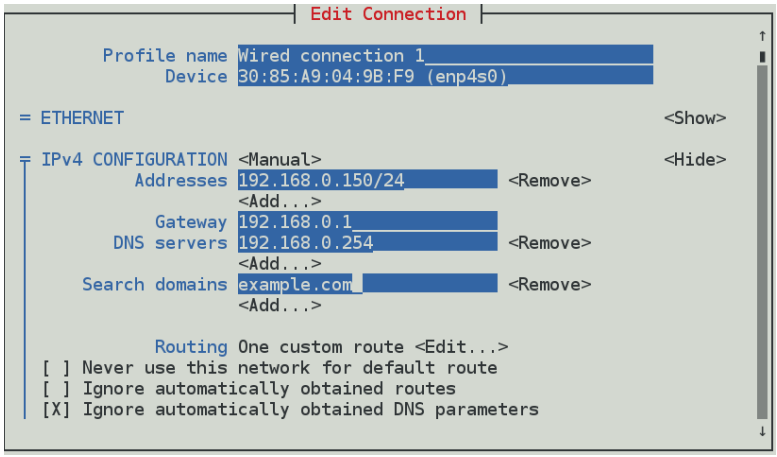
Understanding networking configuration files

Whether you change your network setup using NetworkManager or `nmtui`, most of the same configuration files are updated. In Fedora and RHEL, network interfaces and custom routes are set in files in the `/etc/sysconfig/network-scripts` directory.

Open the `/usr/share/doc/initscripts/sysconfig.txt` file for descriptions of network-scripts configuration files (available from the `initscripts` package).

FIGURE 14.8

Set static IP addresses by selecting Manual from the Edit Connection screen.



One thing to be careful about is that NetworkManager believes that it controls the files in the network-scripts directory. So keep in mind that if you set manual addresses on an interface that NetworkManager has configured for DHCP, it could overwrite changes that you made manually to the file.

Network interface files

Configuration files for each wired, wireless, ISDN, dialup, or other type of network interface are represented by files in the `/etc/sysconfig/network-scripts` directory that begin with `ifcfg-`*`interface`*. Note that *`interface`* is replaced by the name of the network interface.

Given a network interface for a wired NIC as `enp4s0`, here's an example of an `ifcfg-enp4s0` file for that interface, configured to use DHCP:

```
DEVICE=enp4s0
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFROUTE=yes
UUID=f16259c2-f350-4d78-a539-604c3f95998c
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
```

```
NAME="System enp4s0"
PEERDNS=yes
PEERROUTES=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

In this `ifcfg-enp4s0` example, the first two lines set the device name and the type of interface to `Ethernet`. The `BOOTPROTO` variable is set to `dhcp`, which causes it to request address information from a DHCP server. With `ONBOOT=yes`, the interface starts automatically at system boot time. IPV6 settings say to initialize IPV6 and use the IPV6 settings that are presented, but the interface will continue to initialize if there is no IPV6 network available. Other settings say to use peer DNS automatically and route values that are detected.

Here's what a simple `ifcfg-enp4s1` file might look like for a wired Ethernet interface that uses static IP addresses:

```
DEVICE=enp4s1
HWADDR=00:1B:21:0A:E8:5E
TYPE=Ethernet
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
IPADDR=192.168.0.140
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
```

In this `ifcfg-enp4s1` example, because this is setting the address and other information statically, `BOOTPROTO` is set to `none`. Other differences are needed to set the address information that is normally gathered from a DHCP server. In this case, the IP address is set to 192.168.0.140 with a netmask of 255.255.255.0. The `GATEWAY=192.168.0.1` identifies the address of the router to the Internet.

Here are a couple of other settings that might interest you:

- `PEERDNS`: Setting `PEERDNS=no` prevents DHCP from overwriting the `/etc/resolv.conf` file. This allows you to set which DNS servers your system uses without fear of that information being erased by data that is provided by the DHCP server.
- `DNS?`: If an `ifcfg` file is being managed by NetworkManager, it sets the address of DNS servers using DNS*?* entries. For example, `DNS1=192.168.0.2` causes that IP address to be written to `/etc/resolv.conf` as the first DNS server being used on the system. You can have multiple `DNS?` entries (`DNS2=`, `DNS3=`, and so on).

In addition to configuring the primary network interfaces, you can also create files in the `/etc/sysconfig/network-scripts` directory that can be used to set aliases (multiple IP addresses for the same interface), bonded interfaces (multiple NICs listening on the same address), and custom routes. Those are described later in this chapter.

Other networking files

In addition to the network interface files, there are other network configuration files that you can edit directly to configure Linux networking. Here are some of those files.

/etc/sysconfig/network file

System-wide settings associated with your local networking can be included in your `/etc/sysconfig/network` file. The system's hostname was commonly set in this file up to RHEL 6, but other settings can be added to this file as well. Here is an example of the contents of an `/etc/sysconfig/network` file:

```
GATEWAY=192.168.0.1
```

In the previous example, the default `GATEWAY` is set to 192.168.0.1. Different interfaces can use different `GATEWAY` addresses. For other settings that can appear in the `network` files, check the `sysconfig.txt` file in the `/usr/share/doc/initscripts` directory.

/etc/hostname file

In RHEL and Fedora releases, the system's hostname is stored in the `/etc/hostname` file. For example, if the file included the hostname `host1.example.com`, that hostname would be set each time the system booted up. You can check how the current hostname is set at any time by typing the `hostname` command.

/etc/hosts file

Before DNS was created, translating hostnames to IP addresses was done by passing around a single hosts file. While there were only a few dozen and then a few hundred hosts on the Internet, this approach worked pretty well. But as the Internet grew, the single hosts file became unscalable and DNS was invented.

The `/etc/hosts` file still exists on Linux systems. It can still be used to map IP addresses to hostnames. The `/etc/hosts` file is a way to set up names and addresses for a small local network or just create aliases in order to make it easier to access the systems that you use all the time.

Here's an example of an `/etc/hosts` file:

```
127.0.0.1  localhost localhost.localdomain
::1        localhost localhost.localdomain
192.168.0.201  node1.example.com node1 joe
192.168.0.202  node2.example.com node2 sally
```

The first two lines (`127.0.0.1` and `::1`) set addresses for the local system. The IPv4 address for the local host is `127.0.0.1`; the IPv6 address for the local host is `::1`. There are also entries for two IP addresses. You could reach the first IP address (192.168.0.201) by the names `node1.example.com`, `node1`, or `joe`. For example, typing `ping joe` results in packets being sent to 192.168.0.201.

/etc/resolv.conf file

DNS servers and search domains are set in the `/etc/resolv.conf` file. If NetworkManager is enabled and running, you should not edit this file directly. Using `DNS?=` entries from `ifcfg-*` files, NetworkManager overwrites the `/etc/resolv.conf` file so that you would lose any entries you add to that file. Here's an example of the `/etc/resolv.conf` file that was modified by NetworkManager:

```
# Generated by NetworkManager
nameserver 192.168.0.2
nameserver 192.168.0.3
```

Each nameserver entry identifies the IP address of a DNS server. The order defines the order in which the DNS servers are checked. It's normal to have two or three nameserver entries, in case the first is not available. More than that, and it can take too long for an unresolvable hostname to get checked for each server.

Another type of entry that you can add to this file is a search entry. A *search entry* lets you indicate domains to be searched when a hostname is requested by its base name instead of its entire fully qualified domain name. You can have multiple search entries by identifying one or more domain names after the search keyword, as in this example:

```
search example.com example.org example.net
```

The search options are separated by spaces or tabs.

/etc/nsswitch.conf

Unlike in earlier releases, the `/etc/nsswitch.conf` file is managed by the `authselect` command and should not be modified manually. To make changes, edit the `/etc/authselect/user-nsswitch.conf` file and run `authselect apply-changes`.

Settings in the `/etc/nsswitch.conf` file determine that hostname resolution is done by first searching the local `/etc/hosts` file (files) and then DNS servers listed in the `/etc/resolv.conf` file (`dns`). The `myhostname` value is used to ensure that an address is always returned for the host. This is how the hosts entry in the `/etc/resolv.conf` file appears in Red Hat Enterprise Linux:

```
hosts:      files dns myhostname
```

You can add other locations, such as Network Information Service (`nis` or `nisplus`) databases, for querying hostname-to-IP-address resolution. You can also change the order in which the different services are queried. You can check that hostname-to-IP-address resolution is working properly using different commands.

If you want to check that your DNS servers are being queried properly, you can use the `host` or `dig` commands, as in, for example:

```
$ host redhat.com
redhat.com has address 209.132.183.105
redhat.com mail is handled by 10 us-smtp-inbound-1.mimecast.com.
```

```
redhat.com mail is handled by 10 us-smtp-inbound-2.mimecast.com.
$ dig redhat.com
; <<>> DiG 9.11.11-RedHat-9.11.11-1.fc30 <<>> redhat.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9948
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;redhat.com.          IN  A
...
;; ANSWER SECTION:
redhat.com.     3600  IN  A  09.132.183.105
;; Query time: 49 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sat Nov 23 19:16:14 EST 2019
```

By default, the `host` command produces simpler output for DNS queries. It shows the IP address for `redhat.com` and the names of the mail servers (MX records) that serve `redhat.com`. The `dig` command shows information similar to what appears in the files that hold DNS records. The `QUESTION SECTION` part of the output shows that the address section asked for the address of `redhat.com` and the `ANSWER SECTION` part showed the answer (`209.132.183.105`). You can also see the address of the DNS server that was queried.

The `host` and `dig` commands are only used to query DNS servers. They don't check the `nsswitch.conf` file to find other places to query, such as the local hosts file. For that, you would have to use the `getent` command:

```
# getent hosts node1
192.168.0.201  node1
```

This `getent` example finds a host named `node1` that was entered into my local `/etc/hosts` file. The `getent` command can be used to query any information setup in the `nsswitch.conf` file. For example, typing `getent passwd root` shows the entry for the root user account in the local file, but it can also query a remote LDAP database for user information if you have configured that feature, as described in Chapter 11, "Managing User Accounts."

Setting alias network interfaces

Sometimes you might want your network interface card listening on multiple IP addresses. For example, if you were setting up a web server that was serving secure content (`https`) for multiple domains (`example.com`, `example.org`, and so on), each domain would require a separate IP address (associated with a separate certificate). In that case, instead of adding multiple network interface cards to the computer, you could simply create multiple aliases on a single NIC.

To create an alias network interface in RHEL 6 and earlier Fedora releases, you just have to create another `ifcfg-` file. Following the example of an `eth0` interface on a RHEL system,

you could create an eth0:0 interface associated with the same network interface card. To do this, create a file in the /etc/sysconfig/network-scripts directory called ifcfg-eth0:0 that contains information such as the following:

```
DEVICE=eth0:0
ONPARENT=yes
IPADDR=192.168.0.141
NETMASK=255.255.255.0
```

The example code creates an alias for the network interface eth0 called eth0:0. Instead of ONBOOT, the ONPARENT entry says to bring up this interface if the parent (eth0) is started and listen on address 192.168.0.141. You can bring up that interface by typing ifup eth0:0. You can then check that the interface came up using the ip command:

```
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
       pfifo_fast state UP qlen 1000
    link/ether f0:de:f1:28:46:d9 brd ff:ff:ff:ff:ff:ffinet
    192.168.0.140/24 brd 192.168.0.255 scope global
    eth0inet 192.168.0.141/24 brd 192.168.0.255 scope global secondary
    eth0:0inet6 fe80::f2de:f1ff:fe28:46d9/64 scope link
       valid_lft forever preferred_lft forever
```

You can see that the network interface card represented by eth0 is listening on two addresses: 192.168.0.140 (eth0) and 192.168.0.141 (eth0:0). So, this system will respond to packets destined for either of those two addresses. You could add more IP addresses to that interface by creating more ifcfg-eth0:? files (ifcfg-eth0:1, ifcfg-eth0:2, and so on).

In more recent RHEL and Fedora systems, you can create aliases directly in the primary ifcfg file for an alias. For example, a primary (192.168.0.187) and alias (192.168.99.1) address for a NIC interface named p4p1 might be represented by the following address settings in the ifcfg-p4p1 file:

```
IPADDR=192.168.0.187
PREFIX=24
IPADDR1=192.168.99.1
PREFIX1=24
```

Setting up Ethernet channel bonding

Ethernet channel bonding allows you to have more than one network interface card on a computer associated with a single IP address. There are several reasons you might want to do this:

High availability Multiple NICs on the same IP address can ensure that if one subnet goes down or one NIC breaks, the address can still be reached on a NIC connected to another subnet.

Performance If there is too much network traffic to be handled by one NIC, you can spread that traffic across multiple NICs.

In Red Hat Enterprise Linux and Fedora on a computer with multiple NICs, you can set up Ethernet channel bonding by creating a few `ifcfg` files and loading the necessary module. You can start with one bonding file (for example, `ifcfg-bond0`) and then point multiple `ifcfg-eth?` files at that bond interface. Then you can load the bond module.

Depending on the type of bonding that you want to do, you can set your bonding interface to different modes. Using the `BONDING_OPTS` variable, you define the mode and other bonding options (all of which are passed to the bonding module). You can read about the bonding module by entering `modinfo bonding` or by installing the `kernel-docs` package and reading the `bonding.txt` file from the `/usr/share/doc/kernel-doc*/Documentation/networking/`directory.

Here is an example of the file that defines a bonded interface. The file in this example is `/etc/sysconfig/network-scripts/ifcfg-bond0`:

```
DEVICE=bond0
ONBOOT=yes
IPADDR=192.168.0.50
NETMASK=255.255.255.0
BOOTPROTO=none
BONDING_OPTS="mode=active-backup"
```

The `bond0` interface in this example uses the IP address 192.168.0.50. It starts up on boot. The `BONDING _ OPTS` sets the bonding mode to active-backup. This means that only one NIC is active at a time, and the next NIC only takes over when the previous one fails (failover). No network interface card is associated with the `bond0` interface yet. For that, you must create separate `ifcfg` file options. For example, create an `/etc/sysconfig/network-scripts/ifcfg-eth0` that looks like the following (then create `eth1`, `eth2`, `eth3`, and so on for each NIC that you want to use in the bonding interface):

```
DEVICE=eth0
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
ONBOOT=yes
```

With the `eth0` interface used as part of the `bond0` interface, there is no IP address assigned. That's because the `eth0` interface uses the IP address from the `bond0` interface by defining itself as a slave (`SLAVE=yes`) to `bond0` (`MASTER=bond0`).

The last thing that you want to do is to make sure that the `bond0` interface is set to use the bonding module. To do that, create an `/etc/modprobe.d/bonding.conf` file that contains the following entry:

```
alias bond0 bonding
```

Because all of the interfaces are set to `ONBOOT=yes`, the `bond0` interface starts and all of the *`eth?`* interfaces are available as they are needed.

Setting custom routes

On a simple network configuration, communications that are destined for the local network are directed to the appropriate interface on your LAN, while communications for hosts outside of your LAN go to a default gateway to be sent on to remote hosts. As an alternative, you can set custom routes to provide alternative paths to specific networks.

To set a custom route in Fedora and RHEL, you create a configuration file in the `/etc/sysconfig/network-scripts` directory. In that route, you define the following:

GATEWAY? The IP address of the node on the local network that provides the route to the subnetwork represented by the static route.

ADDRESS? The IP address representing the network that can be reached by the static route.

NETMASK? The netmask that determines which part of the *ADDRESS?* represents the network and which represents the hosts that can be reached on that network.

The name of each custom route file is *route-interface*. So, for example, a custom route that can be reached through your `eth0` interface would be named `route-eth0`. You could have multiple custom routes in that file, with each route entry replacing the *?* with the interface number:

```
ADDRESS0=192.168.99.0
NETMASK0=255.255.255.0
GATEWAY0=192.168.0.5
```

In this example, any packet destined for a host on the 192.168.99 network would be sent through the local `eth0` interface and directed to the gateway node at 192.168.0.5. Presumably, that node would provide a route to another network containing hosts in the 192.168.99 address range. This route would take effect when the `eth0` network interface was restarted.

To check that the route is working after you restart the network interface, you could type the following:

```
# route
Kernel IP routing table
Destination  Gateway     Genmask         Flags Metric Ref Use Iface
default      192.168.0.1 0.0.0.0         UG    0      0      0 eth0
192.168.0.0  *           255.255.255.0   U     1      0      0 eth0
192.168.99.0 192.168.0.5 255.255.255.0   UG    0      0      0 eth0
```

The output from the `route -n` command shows that the default route (anything not destined for the local network 192.168.0 or the 192.168.99 network) is via the 192.168.0.1 address. Any packets destined for the 192.168.99 network are directed through the address 192.168.0.5.

If you wanted to add more custom routes, you could add them to this same `route-eth0` file. The next set of information would be named `ADDRESS1`, `NETMASK1`, `GATEWAY1`, and so on.

Configuring Networking in the Enterprise

So far, the network configuration described in this chapter has centered on setting up single systems to connect to a network. Features available in Linux can go well beyond that by providing software that supports the actual network infrastructure needed by host computers to communicate.

The following sections introduce you to a few of the network infrastructure types of services available in Linux. Full implementation of these features is beyond the scope of this book, but know that if you find yourself needing to manage network infrastructure features, the following sections will give you a sense of how those features are implemented in Linux.

Configuring Linux as a router

If you have more than one network interface on a computer (typically two or more NICs), you can configure Linux as a router. To make this happen, all that is needed is a change to one kernel parameter that allows packet forwarding. To turn on the `ip_forward` parameter immediately and temporarily, enter the following as root:

```
# cat /proc/sys/net/ipv4/ip_forward
0
# echo 1 > /proc/sys/net/ipv4/ip_forward
# cat /proc/sys/net/ipv4/ip_forward
1
```

Packet forwarding (routing) is disabled by default, with the value of `ip_forward` set to 0. By setting it to `1`, packet forwarding is immediately enabled. To make this change permanent, you must add that value to the `/etc/sysctl.conf` file, so that it appears as follows:

```
net.ipv4.ip_forward = 1
```

With that file modified as shown, each time the system reboots, the value for `ip_forward` is reset to `1`. (Notice that `net.ipv4.ip _ forward` reflects the actual location of the `ip _ forward` file, minus `/proc/sys`, and with dots replacing slashes. You can change any kernel parameters set in the `/proc/sys` directory structure in this way.)

When a Linux system is used as a router, it is often used as a firewall between a private network and a public network, such as the Internet. If that is the case, you might also want to use that same system as a firewall that does network address translation (NAT) and provides DHCP service, so the systems on the private network can route through the Linux system using private IP addresses. (See Chapter 25, "Securing Linux on a Network," for information on working with Linux firewall rules using the `iptables` facility.)

Configuring Linux as a DHCP server

Not only can a Linux system use a DHCP server to get its IP address and other information, it can also be configured to act as a DHCP server itself. In its most basic form, a DHCP server can hand out IP addresses from a pool of addresses to any system that requests an IP address. Usually, however, the DHCP server also distributes the locations of DNS servers and the default gateway.

Configuring a DHCP server is not something that should be done without some thought. Don't add a DHCP server on a network that is not under your control and that already has a working DHCP server. Many clients are set up to get address information from any DHCP server that will hand it out.

DHCP service is provided by the `dhcp-server` package in Fedora and RHEL. The service is named `dhcpd`. The primary configuration file is `/etc/dhcp/dhcpd.conf` for IPv4 networks (there is a `dhcpd6.conf` file in the same directory to provide DHCP service for IPv6 networks). By default, the `dhcpd` daemon listens on UDP port 67, so remember to keep that port open on your firewall.

To configure a DHCP server, you could copy the `dhcpd.conf.example` file from the `/usr/share/doc/dhcp-server` directory and replace the `/etc/dhcp/dhcpd.conf` file. Then modify it as you like. Before using that file, however, you want to change the domain-name options to reflect your domain and IP address ranges to suit those you are using. The comments in the file will help you do this.

When you install some virtualization and cloud services on a Linux system, a DHCP server is set up by default for you within that system. For example, when you install KVM and start the `libvirtd` service in RHEL or Fedora, it automatically configures a default private network in the 192.168.122.0/24 address range. When you launch virtual machines, they are given IP addresses in that range. When you install and start the Docker service on those Linux distributions, it likewise sets up a private network and hands out IP addresses to Docker containers launched on that system.

Configuring Linux as a DNS server

In Linux, most professional Domain Name System (DNS) servers are implemented using the Berkeley Internet Name Domain (BIND) service. This is implemented in Fedora and RHEL by installing the `bind`, `bind-utils`, and `bind-libs` packages. For added security, some people install the `bind-chroot` package.

By default, `bind` is configured by editing the `/etc/named.conf` file. Hostname-to-IP-address mapping is done in zone files located in the `/var/named` directory. If you install the `bind-chroot` package, `bind` configuration files are moved under the `/var/named/chroot` directory, which attempts to replicate the files from `/etc` and `/var` that are needed to configure `bind` so that the named daemon (which provides the service) is confined to the `/etc/named/chroot` directory structure.

If you are interested in trying out `bind`, I recommend that you first try it out by configuring DNS for a small home network behind a firewall as a way to make it easier for the people in your household to communicate with each other. You can lock down the IP addresses of the machines in your home by attaching MAC addresses of each computer's network interface card to specific IP addresses on a DHCP server and then mapping those names to addresses in a DNS server.

CAUTION

Before you create a public DNS server, keep in mind that it is very important to secure your DNS server properly. A cracked public DNS server can be used to redirect traffic to any server the bad guys choose. So, if you are using that server, you are in danger of being presented with sites that are not the sites you think they are.

Configuring Linux as a proxy server

A proxy server provides a means of restricting network traffic from a private network to a public one, such as the Internet. Such servers provide an excellent way to lock down a computer lab at a school or restrict websites that employees can visit from work.

By physically setting up Linux as a router but configuring it as a proxy server, all of the systems on your home or business network can be configured to access the Internet using only certain protocols and only after you filter the traffic.

Using the Squid Proxy Server, which comes with most Linux systems (`squid` package in Fedora and RHEL), you can enable the system to accept requests to web servers (HTTP and HTTPS), file servers (FTP), and other protocols. You can restrict which systems can use your proxy server (by hostname or IP address) and even limit which sites they can visit (by specific address, range of addresses, hostname, or domain names).

Configuring a squid proxy server can be as simple as installing the `squid` package, editing the `/etc/squid/squid.conf` file, and starting the `squid` service. The file comes with a recommended minimal configuration. However, you might want to define the hosts (based on IP address or name) that you want to allow to use the service. There are blacklists available with `squid` that allow you to deny access to whole sets of sites that might be inappropriate for children to visit.

Summary

Most network connections from a Linux desktop or laptop system can be made with little or no user intervention. If you use NetworkManager over a wired or wireless Ethernet connection, address and server information needed to start up can be automatically obtained from a DHCP server.

With NetworkManager's graphical interface, you can do some network configuration, if you like. You can set static IP addresses and select the name server and gateway computers to

use. To do more manual and complex network configuration, consider working more directly with network configuration files.

Network configuration files in Linux can be used to set up more advanced features such as Ethernet channel bonding.

Beyond the basics of network connectivity in Linux, features are available that enable you to provide network infrastructure types of services. This chapter introduced services and features such as routing, DHCP, and DNS that you need to know when working with more advanced networking features in Linux.

With your networking configured, you can now begin configuring services to run over your networks. Chapter 15, "Starting and Stopping Services," describes the tools that you need to enable, disable, start, stop, and check the status of the services that are configured for your Linux system.

Exercises

The exercises in this section help you to examine and change the network interfaces on your Linux system as well as understand how to configure more advanced networking features. Start these exercises on a Linux system that has an active network connection but that is *not* in the middle of some critical network activity.

I recommend that you do these exercises directly from your computer console (in other words, don't ssh into the computer to do them). Some of the commands that you run may interrupt your network connectivity, and some of the configuration you do, if you make a mistake, can result in your computer being temporarily unavailable from the network.

There are often multiple ways to complete the tasks described in these exercises. If you are stuck, refer to the task solutions that are provided in Appendix B.

1. Use the desktop to check that NetworkManager has successfully started your network interface (wired or wireless) to the network. If it has not, then try to start your network interface.
2. Run a command to check the active network interfaces available on your computer.
3. Try to contact `google.com` from the command line in a way that ensures that DNS is working properly.
4. Run a command to check the routes being used to communicate outside of your local network.
5. Trace the route being taken to connect to `google.com`.
6. View the network activity of your Linux system from the Cockpit web user interface.
7. Create a host entry that allows you to communicate with your local host system using the name `myownhost`.

8. Determine the addresses of the DNS name servers that are being used to resolve hostnames to IP addresses on your system, then check which is queried from your system to find the IP address for `google.com`.
9. Create a custom route that directs traffic destined for the 192.168.99.0/255.255.255.0 network to some IP address on your local network, such as 192.168.0.5 (first ensuring that the 192.168.99 network is not being used at your location).
10. Check to see if your system has been configured to allow IPv4 packets to be routed between network interfaces on your system.

CHAPTER 15

Starting and Stopping Services

IN THIS CHAPTER

Understanding the various Linux init services

Auditing Linux daemon-controlled services

Stopping and starting services

Changing the Linux server's default runlevel

Removing services

The primary job of a Linux server system is to offer services to local or remote users. A server can provide access to web pages, files, database information, streaming music, or other types of content. Name servers can provide access to lists of host computer or usernames. Hundreds of these and other types of services can be configured on your Linux systems.

Ongoing services offered by a Linux system, such as access to a printer service or login service, are typically implemented by what is referred to as a *daemon* process. Most Linux systems have a method of managing each daemon process as a *service* using one of several popular initialization systems (also referred to as init systems). Advantages of using init systems include the ability to do the following:

Identify runlevels: Put together sets of services in what are referred to as *runlevels* or *targets*.

Establish dependencies: Set service dependencies so, for example, a service that requires network interfaces won't start until all network startup services have started successfully.

Set the default runlevel: Select which runlevel or target starts up when the system boots (a *default runlevel*).

Manage services: Run commands that tell individual services to start, stop, pause, restart, or even reload configuration files.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Several different init systems are in use with Linux systems today. The one you use depends on the Linux distribution and release that you are using. In this chapter, I cover the following init systems that have been used in Fedora, Red Hat Enterprise Linux, Ubuntu, and many other Linux distributions:

SysVinit: This traditional init system was created for UNIX System V systems in the early 1980s. It offers an easy-to-understand method of starting and stopping services based on runlevel. Most UNIX and Linux systems up until a few years ago used SysVinit.

Systemd: The latest versions of Fedora and RHEL use the `systemd` init system. It is the most complex of the init systems, but it also offers much more flexibility. `Systemd` offers not only features for starting and working with services, but also lets you manage sockets, devices, mount points, swap areas, and other unit types.

Note

If you are using an older version of Ubuntu, you probably used Upstart as your initialization system. Beginning with Ubuntu 15.04 (released April 28, 2015), Upstart was replaced by the `systemd` initialization daemon. Thus, Upstart will not be described in this book.

This chapter describes the `SysVinit` and `systemd` init systems. In the process of using the init system that matches your Linux distribution, you learn how the boot process works to start services, how you can start and stop services individually, and how you enable and disable services.

Understanding the Initialization Daemon (init or systemd)

In order to understand service management, you need to understand the initialization daemon. The initialization daemon can be thought of as the "mother of all processes." This daemon is the first process to be started by the kernel on the Linux server. For Linux distributions that use SysVinit, the init daemon is literally named `init`. For `systemd`, the init daemon is named `systemd`.

The Linux kernel has a process ID (PID) of 0. Thus, the initialization process (`init` or `systemd`) daemon has a parent process ID (PPID) of 0, and a PID of 1. Once started, `init` is responsible for spawning (launching) processes configured to be started at the server's boot time, such as the login shell (`getty` or `mingetty` process). It is also responsible for managing services.

The Linux `init` daemon was based on the UNIX System V `init` daemon. Thus, it is called the SysVinit daemon. However, it was not the only classic `init` daemon. The `init` daemon is not part of the Linux kernel. Therefore, it can come in different flavors, and Linux distributions can choose which flavor to use. Another classic `init` daemon was based on

Berkeley UNIX, also called BSD. Therefore, the two original Linux `init` daemons were BSD `init` and SysVinit.

The classic `init` daemons worked without problems for many years. However, these daemons were created to work within a static environment. As new hardware, such as USB devices, came along, the classic `init` daemons had trouble dealing with these and other hot-plug devices. Computer hardware had changed from static to event based. New `init` daemons were needed to deal with these fluid environments.

In addition, as new services came along, the classic `init` daemons had to deal with starting more and more services. Thus, the entire system initialization process was less efficient and ultimately slower.

The modern initialization daemons have tried to solve the problems of inefficient system boots and non-static environments. The most popular of the new initialization daemons is `systemd`. Ubuntu, RHEL, and Fedora distributions have made the move to the `systemd` daemon while maintaining backward compatibility to the classic SysVinit, Upstart, or BSD `init` daemons.

The `systemd` daemon, available from `http://docs.fedoraproject.org/en-US/quick-docs/understanding-and-administering-systemd`, was written primarily by Lennart Poettering, a Red Hat developer. It is currently used by all of the latest versions of Fedora, RHEL, OpenSUSE, and Ubuntu.

In order to manage your services properly, you need to know which initialization daemon your server has. Figuring that out can be a little tricky. The initialization process running on a SysVinit or Upstart is named `init`. For the first `systemd` systems, it was also called `init` but is now named `systemd`. Running `ps -e` can immediately tell you if yours is a `systemd` system:

```
# ps -e | head
  PID TTY          TIME CMD
    1 ?        00:04:36 systemd
    2 ?        00:00:03 kthreadd
    3 ?        00:00:15 ksoftirqd/0
```

If PID 1 is the `init` daemon for your system, try looking on the init *Wikipedia* page (`http://wikipedia.org/wiki/Init`) under "Other implementations." This will help you understand if your `init` daemon is SysVinit, Upstart, or some other initialization system.

Understanding the classic init daemons

The classic `init` daemons, SysVinit and BSD `init`, are worth understanding, even if your Linux server has a different `init` daemon. Not only is backward compatibility to the classics often used in the newer `init` daemons, but many are based upon them.

The classic SysVinit and BSD `init` daemons operate in a very similar fashion. Although in the beginning they may have been rather different, over time very few significant differences remained. For example, the older BSD `init` daemon would obtain configuration

information from the /etc/ttytab file. Now, like the SysVinit daemon, the BSD init daemon's configuration information is taken at boot time from the /etc/inittab file. The following is a classic SysVinit /etc/inittab file:

```
# cat /etc/inittab
# inittab  This file describes how the INIT process should set up
# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, no NFS (Same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
pf::powerfail:/sbin/shutdown -f -h +2
"Power Failure; System Shutting Down"


# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c
"Power Restored; Shutdown Cancelled"


# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

TABLE 15.1 Standard Linux Runlevels

Runlevel #	Name	Description
0	Halt	All services are shut down, and the server is stopped.
1 or S	Single User Mode	The root account is automatically logged in to the server. Other users cannot log in to the server. Only the command-line interface is available. Network services are not started.
2	Multiuser Mode	Users can log in to the server, but only the command-line interface is available. On some systems, network interfaces and services are started; on others they are not. Originally, this runlevel was used to start dumb terminal devices so that users could log in (but no network services were started).
3	Extended Multiuser Mode	Users can log in to the server, but only the command-line interface is available. Network interfaces and services are started. This is a common runlevel for servers.
4	User Defined	Users can customize this runlevel.
5	Graphical Mode	Users can log in to the server. Command-line and graphical interfaces are available. Network services are started. This is a common runlevel for desktop systems.
6	Reboot	The server is rebooted.

The `/etc/inittab` file tells the `init` daemon which runlevel is the default runlevel. A *runlevel* is a categorization number that determines what services are started and what services are stopped. In the preceding example, a default runlevel of 5 is set with the line `id:5:initdefault:`. Table 15.1 shows the standard seven Linux runlevels.

Linux distributions can differ slightly on the definition of each runlevel as well as which runlevels are offered.

CAUTION

The only runlevels that should be used in the `/etc/inittab` file are 2 through 5. The other runlevels could cause problems. For example, if you put runlevel 6 in the `/etc/inittab` file as the default, when the server reboots, it would go into a loop and continue to reboot over and over again.

The runlevels are not only used as a default runlevel in the `/etc/inittab` file. They can also be called directly using the `init` daemon itself. Thus, if you want to halt your server immediately, you type `init 0` at the command line:

```
# init 0
...
System going down for system halt NOW!
```

The `init` command accepts any of the runlevel numbers in Table 15.1, allowing you to switch your server quickly from one runlevel category to another. For example, if you need

to perform troubleshooting that requires the graphical interface to be down, you can type **`init 3`** at the command line:

```
# init 3
INIT: Sending processes the TERM signal
starting irqbalance:                       [ OK ]
Starting setroubleshootd:
Starting fuse:  Fuse filesystem already available.
...
Starting console mouse services:           [ OK ]
```

To see your Linux server's current runlevel, simply type in the command **`runlevel`**. The first item displayed is the server's previous runlevel, which in the following example is `5`. The second item displayed shows the server's current runlevel, which in this example is `3`.

```
$ runlevel
5 3
```

In addition to the `init` command, you can use the `telinit` command, which is functionally the same. In the example that follows, the `telinit` command is used to reboot the server by taking it to runlevel `6`:

```
# telinit 6
INIT: Sending processes the TERM signal
Shutting down smartd:                          [ OK ]
Shutting down Avahi daemon:                    [ OK ]
Stopping dhcdbd:                               [ OK ]
Stopping HAL daemon:                           [ OK ]
...
Starting killall:
Sending all processes the TERM signal...       [ OK ]
Sending all processes the KILL signal...       [ OK ]
...
Unmounting filesystems                         [ OK ]
Please stand by while rebooting the system
...
```

On a freshly booted Linux server, the current runlevel number should be the same as the default runlevel number in the `/etc/inittab` file. However, notice that the previous runlevel in the example that follows is `N`. The `N` stands for "Nonexistent" and indicates that the server was freshly booted to the current runlevel.

```
$ runlevel
N 5
```

How does the server know which services to stop and which ones to start when a particular runlevel is chosen? When a runlevel is chosen, the scripts located in the `/etc/rc.d/rc#.d` directory (where # is the chosen runlevel) are run. These scripts are run whether the runlevel is chosen via a server boot and the `/etc/inittab initdefault` setting or the `init` or `telinit` command is used. For example, if runlevel `5` is chosen, then all of the

scripts in the /etc/rc.d/rc5.d directory are run; your list will be different, depending on what services you have installed and enabled.

```
# ls /etc/rc.d/rc5.d
K01smolt                      K88wpa_supplicant   S22messagebus
K02avahi-dnsconfd             K89dund             S25bluetooth
K02NetworkManager             K89netplugd         S25fuse
K02NetworkManagerDispatcher   K89pand             S25netfs
K05saslauthd                  K89rdisc            S25pcscd
K10dc_server                  K91capi             S26hidd
K10psacct                     S00microcode_ctl    S26udev-post
K12dc_client                  S04readahead_early  S28autofs
K15gpm                        S05kudzu            S50hplip
K15httpd                      S06cpuspeed         S55cups
K20nfs                        S08ip6tables        S55sshd
K24irda                       S08iptables         S80sendmail
K25squid                      S09isdn             S90ConsoleKit
K30spamassassin               S10network          S90crond
K35vncserver                  S11auditd           S90xfs
K50netconsole                 S12restorecond      S95anacron
K50tux                        S12syslog           S95atd
K69rpcsvcgssd                 S13irqbalance       S96readahead_later
K73winbind                    S13mcstrans         S97dhcdbd
K73ypbind                     S13rpcbind          S97yum-updatesd
K74nscd                       S13setroubleshoot   S98avahi-daemon
K74ntpd                       S14nfslock          S98haldaemon
K84btseed                     S15mdmonitor        S99firstboot
K84bttrack                    S18rpcidmapd        S99local
K87multipathd                 S19rpcgssd          S99smartd
```

Notice that some of the scripts within the /etc/rc.d/rc5.d directory start with a K and some start with an S. The K refers to a script that will kill (stop) a process. The S refers to a script that will start a process. Also, each K and S script has a number before the name of the service or daemon that they control. This allows the services to be stopped or started in a particular controlled order. You would not want your Linux server's network services to be started before the network itself was started.

An /etc/rc.d/rc#.d directory exists for all the standard Linux runlevels. Each one contains scripts to start and stop services for its particular runlevel.

```
# ls -d /etc/rc.d/rc?.d
/etc/rc.d/rc0.d  /etc/rc.d/rc2.d  /etc/rc.d/rc4.d  /etc/rc.d/rc6.d
/etc/rc.d/rc1.d  /etc/rc.d/rc3.d  /etc/rc.d/rc5.d
```

Actually, the files in the /etc/rc.d/rc#.d directories are not scripts but instead symbolic links to scripts in the /etc/rc.d/init.d directory. Thus, there is no need to have multiple copies of particular scripts.

```
# ls -l /etc/rc.d/rc5.d/K15httpd
lrwxrwxrwx 1 root root 15 Oct 10 08:15
```

```
 /etc/rc.d/rc5.d/K15httpd -> ../init.d/httpd
# ls /etc/rc.d/init.d
anacron             functions  multipathd               rpcidmapd
atd                 fuse       netconsole               rpcsvcgssd
auditd              gpm        netfs                    saslauthd
autofs              haldaemon  netplugd                 sendmail
avahi-daemon        halt       network                  setroubleshoot
avahi-dnsconfd      hidd       NetworkManager           single
bluetooth           hplip      NetworkManagerDispatcher smartd
btseed              hsqldb     nfs                      smolt
bttrack             httpd      nfslock                  spamassassin
capi                ip6tables  nscd                     squid
ConsoleKit          iptables   ntpd                     sshd
cpuspeed            irda       pand                     syslog
crond               irqbalance pcscd                    tux
cups                isdn       psacct                   udev-post
cups-config-daemon  killall    rdisc                    vncserver
dc_client           kudzu      readahead_early          winbind
dc_server           mcstrans   readahead_later          wpa_supplicant
dhcdbd              mdmonitor  restorecond              xfs
dund                messagebus rpcbind                  ypbind
firstboot           microcode  rpcgssd                  yum-updatesd
```

Notice that each service has a single script in `/etc/rc.d/init.d`. There aren't separate scripts for stopping and starting a service. These scripts will stop or start a service depending upon what parameter is passed to them by the `init` daemon.

Each script in `/etc/rc.d/init.d` takes care of all that is needed for starting or stopping a particular service on the server. The following is a partial example of the `httpd` script on a Linux system that uses the SysVinit daemon. It contains a case statement for handling the parameter (`$1`) that was passed to it, such as `start`, `stop`, `status`, and so on.

```
# cat /etc/rc.d/init.d/httpd
#!/bin/bash
#
# httpd        Startup script for the Apache HTTP Server
#
# chkconfig: - 85 15
# description: Apache is a World Wide Web server.
#              It is used to serve \
#              HTML files and CGI.
# processname: httpd
# config: /etc/httpd/conf/httpd.conf
# config: /etc/sysconfig/httpd
# pidfile: /var/run/httpd.pid

# Source function library.
. /etc/rc.d/init.d/functions
...
```

```
# See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status $httpd
        RETVAL=$?
        ;;
...
esac

exit $RETVAL
```

After the runlevel scripts linked from the appropriate `/etc/rc.d/rc#.d` directory are executed, the SysVinit daemon's process spawning is complete. The final step the `init` process takes at this point is to do anything else indicated in the `/etc/inittab` file (such as spawn `mingetty` processes for virtual consoles and start the desktop interface, if you are in runlevel `5`).

Understanding systemd initialization

The `systemd` initialization daemon is the newer replacement for the SysVinit and the Upstart `init` daemons. This modern initialization daemon was introduced in Fedora 15 and RHEL 7, and it is still in use today. It is backward compatible with both SysVinit and Upstart. System initialization time is reduced by `systemd` because it can start services in a parallel manner.

Learning systemd basics

With the SysVinit daemon, services are stopped and started based upon runlevels. The `systemd` service is concerned with runlevels, but it implements them in a different way with what are called *target units*. Although the main job of `systemd` is to start and stop services, it can manage other types of things referred to as units. A *unit* is a group consisting of a name, type, and configuration file, and it is focused on a particular service or action. There are 12 `systemd` unit types:

- `automount`
- `device`
- `mount`
- `path`
- `service`
- `snapshot`
- `socket`

- target
- timer
- swap
- slice
- scope

The two primary systemd units with which you need to be concerned for dealing with services are service units and target units. A *service unit* is for managing daemons on your Linux server. A *target unit* is simply a group of other units.

The example that follows shows several systemd service units and target units. The service units have familiar daemon names, such as cups and sshd. Note that each service unit name ends with .service. The target units shown have names like sysinit. (sysinit is used for starting up services at system initialization.) The target unit names end with .target.

```
# systemctl list-units | grep .service
...
cups.service           loaded active running CUPS Printing Service
dbus.service           loaded active running D-Bus Message Bus
...
NetworkManager.service loaded active running Network Manager
prefdm.service         loaded active running Display Manager
remount-rootfs.service loaded active exited  Remount Root FS
rsyslog.service        loaded active running System Logging
...
sshd.service           loaded active running OpenSSH server daemon
systemd-logind.service loaded active running Login Service
...
# systemctl list-units | grep .target
basic.target           loaded active active  Basic System
cryptsetup.target      loaded active active  Encrypted Volumes
getty.target           loaded active active  Login Prompts
graphical.target       loaded active active  Graphical Interface
local-fs-pre.target    loaded active active  Local File Systems (Pre)
local-fs.target        loaded active active  Local File Systems
multi-user.target      loaded active active  Multi-User
network.target         loaded active active  Network
remote-fs.target       loaded active active  Remote File Systems
sockets.target         loaded active active  Sockets
sound.target           loaded active active  Sound Card
swap.target            loaded active active  Swap
sysinit.target         loaded active active  System Initialization
syslog.target          loaded active active  Syslog
```

The Linux system unit configuration files are located in the `/lib/systemd/system` and `/etc/systemd/system` directories. You could use the `ls` command to look through those directories, but the preferred method is to use an option on the `systemctl` command as follows:

```
# systemctl list-unit-files --type=service
UNIT FILE                                   STATE
...
cups.service                                enabled
...
dbus.service                                static
...
NetworkManager.service                      enabled
...
poweroff.service                            static
...
sshd.service                                enabled
sssd.service                                disabled
...
276 unit files listed.
```

The unit configuration files shown in the preceding code are all associated with a service unit. Configuration files for target units can be displayed via the following method:

```
# systemctl list-unit-files --type=target
UNIT FILE                   STATE
anaconda.target             static
basic.target                static
bluetooth.target            static
cryptsetup.target           static
ctrl-alt-del.target         disabled
default.target              enabled
...
shutdown.target             static
sigpwr.target               static
smartcard.target            static
sockets.target              static
sound.target                static
swap.target                 static
sysinit.target              static
syslog.target               static
time-sync.target            static
umount.target               static
43 unit files listed.
```

Notice that both of the configuration units' file examples display units with a status of static, enabled, or disabled. The enabled status means that the unit is currently enabled. The disabled status means that the unit is currently disabled. The next status, static, is slightly confusing. It stands for "statically enabled," and it means that the unit is enabled by default and cannot be disabled, even by root.

The service unit configuration files contain lots of information, such as what other services must be started, when this service can be started, which environmental file to use, and so on. The following example shows the `sshd` daemon's unit configuration file:

```
# cat /lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=-/etc/crypto-policies/back-ends/opensshserver.config
EnvironmentFile=-/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS $CRYPTO_POLICY
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target

[Install]
WantedBy=multi-user.target
```

This basic service unit configuration file has the following options:

`Description`: A free-form description (comment line) of the service.

`Documentation`: Lists man pages for the `sshd` daemon and configuration file.

`After`: Configures ordering. In other words, it lists which units should be activated before this service is started.

`Environment File`: The service's configuration files.

`ExecStart`: The command used to start this service.

`ExecReload`: The command used to reload this service.

`WantedBy`: The target unit to which this service belongs.

Notice that the target unit, `multi-user.target`, is used in the `sshd` service unit configuration file. The `sshd` service unit is wanted by the `multi-user.target`. In other words, when the `multi-user.target` unit is activated, the `sshd` service unit is started.

You can view the various units that a target unit will activate by using the following command:

```
# systemctl show --property "Wants" multi-user.target
Wants=irqbalance.service firewalld.service plymouth-quit.service
systemd-update-utmp-runlevel.service systemd-ask-password-wall.path...
(END) q
```

Unfortunately, the `systemctl` command does not format the output for this well. It literally runs off the right edge of the screen so you cannot see the full results. Also, you must enter **q** to return to the command prompt. To fix this problem, pipe the output through some formatting commands to produce a nice, alphabetically sorted display, as shown in the example that follows:

```
# systemctl show --property "Wants" multi-user.target \
     | fmt -10 | sed 's/Wants=//g' | sort
atd.service
auditd.service
avahi-daemon.service
chronyd.service
crond.service
...
```

This display shows all of the services and other units that will be activated (started), including `sshd`, when the `multi-user.target` unit is activated. Remember that a target unit is simply a grouping of other units, as shown in the preceding example. Also notice that the units in this group are not all service units. There are path units and other target units as well.

A target unit has both *Wants* and requirements, called *Requires*. A *Wants* means that all of the units listed are triggered to activate (start). If they fail or cannot be started, no problem—the target unit continues on its merry way. The preceding example is a display of Wants only.

A Requires is much more stringent than a Wants and potentially catastrophic. A *Requires* means that all of the units listed are triggered to activate (start). If they fail or cannot be started, the entire unit (group of units) is deactivated.

You can view the various units a target unit Requires (must activate or the unit will fail), using the command in the example that follows. Notice that the Requires output is much shorter than the Wants for the `multi-user.target`. Thus, no special formatting of the output is needed.

```
# systemctl show --property "Requires" multi-user.target
Requires=basic.target
```

The `target` units also have configuration files, as do the `service` units. The following example shows the contents of the `multi-user.target` configuration file.

```
# cat /lib/systemd/system/multi-user.target
#  This file is part of systemd.
#
...

[Unit]
Description=Multi-User
Documentation=man:systemd.special(7)
Requires=basic.target
```

```
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

This basic target unit configuration file has the following options:

`Description`: This is just a free-form description of the target.

`Documentation`: Lists the appropriate systemd man page.

`Requires`: If this `multi-user.target` gets activated, the listed target unit is also activated. If the listed target unit is deactivated or fails, then `multi-user.target` is deactivated. If there are no After and Before options, then both `multi-user.target` and the listed target unit activate simultaneously.

`Conflicts`: This setting avoids conflicts in services. Starting `multi-user.target` stops the listed targets and services, and vice versa.

`After`: This setting configures ordering. In other words, it determines which units should be activated before starting this service.

`AllowIsolate`: This option is a Boolean setting of `yes` or `no`. If this option is set to `yes`, then this target unit, `multi-user.target`, is activated along with its dependencies and all others are deactivated.

To get more information on these configuration files and their options, enter **`man systemd.service`**, **`man systemd.target`**, and **`man systemd.unit`** at the command line.

For the Linux server using `systemd`, the boot process is easier to follow now that you understand `systemd` target units. At boot, `systemd` activates the `default.target` unit. This unit is aliased either to `multi-user.target` or `graphical.target`. Thus, depending upon the `alias` set, the services targeted by the target unit are started.

If you need more help understanding the `systemd` daemon, you can enter **`man -k systemd`** at the command line to get a listing of the various `systemd` utilities' documentation in the man pages.

Learning systemd's backward compatibility to SysVinit

The `systemd` daemon has maintained backward compatibility to the SysVinit daemon. This allows Linux distributions time to migrate slowly to `systemd`.

While runlevels are not truly part of `systemd`, the `systemd` infrastructure has been created to provide compatibility with the concept of runlevels. There are seven target unit configuration files specifically created for backward compatibility to SysVinit:

- `runlevel0.target`
- `runlevel1.target`
- `runlevel2.target`
- `runlevel3.target`
- `runlevel4.target`

- runlevel5.target
- runlevel6.target

As you probably have already figured out, there is a target unit configuration file for each of the seven classic SysVinit runlevels. These target unit configuration files are symbolically linked to target unit configuration files that most closely match the idea of the original runlevel. In the example that follows, the symbolic links are shown for runlevel target units. Notice that the runlevel target units for runlevel 2, 3, and 4 are all symbolically linked to `multi-user.target`. The `multi-user.target` unit is similar to the legacy extended multi-user mode.

```
# ls -l /lib/systemd/system/runlevel*.target
lrwxrwxrwx. 1 root root 15 Apr  9 04:25 /lib/systemd/system/runlevel0.target
    -> poweroff.target
lrwxrwxrwx. 1 root root 13 Apr  9 04:25 /lib/systemd/system/runlevel1.target
    -> rescue.target
lrwxrwxrwx. 1 root root 17 Apr  9 04:25 /lib/systemd/system/runlevel2.target
    -> multi-user.target
lrwxrwxrwx. 1 root root 17 Apr  9 04:25 /lib/systemd/system/runlevel3.target
    -> multi-user.target
lrwxrwxrwx. 1 root root 17 Apr  9 04:25 /lib/systemd/system/runlevel4.target
    -> multi-user.target
lrwxrwxrwx. 1 root root 16 Apr  9 04:25 /lib/systemd/system/runlevel5.target
    -> graphical.target
lrwxrwxrwx. 1 root root 13 Apr  9 04:25 /lib/systemd/system/runlevel6.target
    -> reboot.target
```

The `/etc/inittab` file still exists, but it contains only comments stating that this configuration file is not used, and it gives some basic `systemd` information. The `/etc/inittab` file no longer has any true functional use. This is an example of an `/etc/inittab` file on a Linux server that uses `systemd`.

```
# cat /etc/inittab
# inittab is no longer used.
#
# ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# Ctrl-Alt-Delete is handled by
# /etc/systemd/system/ctrl-alt-del.target
#
# systemd uses 'targets' instead of runlevels.
# By default, there are two main targets:
#
# multi-user.target: analogous to runlevel 3
# graphical.target: analogous to runlevel 5
#
# To view current default target, run:
# systemctl get-default
#
```

```
# To set a default target, run:
# systemctl set-default TARGET.target
```

The /etc/inittab explains that if you want something similar to a classic 3 or 5 runlevel as your default runlevel, you need run systemctl default.target to set the runlevel target to the one you want. To check what default.target is currently symbolically linked to (or in legacy terms, to check the default runlevel), use the command shown here. You can see that on this Linux server, the default is to start up at legacy runlevel 3.

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Mar 13 17:27
 /etc/systemd/system/default.target ->
   /lib/systemd/system/runlevel3.target
```

The capability to switch runlevels using the init or telinit command is still available. When issued, either of the commands is translated into a systemd target unit activation request. Therefore, typing **init 3** at the command line really issues the command systemctl isolate multi-user.target. Also, you can still use the runlevel command to determine the current legacy runlevel, but it is strongly discouraged.

The classic SysVinit /etc/inittab handled spawning the getty or mingetty processes. The systemd init handles this via the getty.target unit. The getty.target is activated by the multi-user.target unit. You can see how these two target units are linked by the following command:

```
# systemctl show --property "WantedBy" getty.target
WantedBy=multi-user.target
```

Now that you have a basic understanding of classic and modern init daemons, it's time to do some practical server administrator actions that involve the initialization daemon.

Checking the Status of Services

As a Linux administrator, you need to check the status of the services being offered on your server. For security reasons, you should disable and remove any unused system services discovered through the process. Most important for troubleshooting purposes, you need to be able to know quickly what should and should not be running on your Linux server.

Of course, knowing which initialization service is being used by your Linux server is the first piece of information to obtain. How to determine this was covered in the section "Understanding the Initialization Daemon" earlier in this chapter. The following sections are organized into subsections on the various initialization daemons.

Checking services for SysVinit systems

To see all of the services that are being offered by a Linux server using the classic SysVinit daemon, use the `chkconfig` command. The example that follows shows the services available on a classic SysVinit Linux server. Note that each runlevel (0–6) is shown for each service with a status of on or off. The status denotes whether a particular service is started (on) or not (off) for that runlevel.

```
# chkconfig --list
ConsoleKit      0:off  1:off  2:off  3:on   4:on   5:on   6:off
NetworkManager  0:off  1:off  2:off  3:off  4:off  5:off  6:off
...
crond           0:off  1:off  2:on   3:on   4:on   5:on   6:off
cups            0:off  1:off  2:on   3:on   4:on   5:on   6:off
...
sshd            0:off  1:off  2:on   3:on   4:on   5:on   6:off
syslog          0:off  1:off  2:on   3:on   4:on   5:on   6:off
tux             0:off  1:off  2:off  3:off  4:off  5:off  6:off
udev-post       0:off  1:off  2:off  3:on   4:on   5:on   6:off
vncserver       0:off  1:off  2:off  3:off  4:off  5:off  6:off
winbind         0:off  1:off  2:off  3:off  4:off  5:off  6:off
wpa_supplicant  0:off  1:off  2:off  3:off  4:off  5:off  6:off
xfs             0:off  1:off  2:on   3:on   4:on   5:on   6:off
ypbind          0:off  1:off  2:off  3:off  4:off  5:off  6:off
yum-updatesd    0:off  1:off  2:off  3:on   4:on   5:on   6:off
```

Some services in the example are never started, such as `vncserver`. Other services, such as the `cups` daemon, are started on runlevels 2 through 5.

Using the `chkconfig` command, you cannot tell if a service is currently running. To do that, you need to use the `service` command. To help isolate only those services that are currently running, the `service` command is piped into the `grep` command and then sorted, as follows:

```
# service --status-all | grep running... | sort
anacron (pid 2162) is running...
atd (pid 2172) is running...
auditd (pid 1653) is running...
automount (pid 1952) is running...
console-kit-daemon (pid 2046) is running...
crond (pid 2118) is running...
cupsd (pid 1988) is running...
...
sshd (pid 2002) is running...
syslogd (pid 1681) is running...
xfs (pid 2151) is running...
yum-updatesd (pid 2205) is running...
```

You can also use both the `chkconfig` and the `service` commands to view an individual service's settings. Using both commands in the example that follows, you can view the `cups` daemon's settings.

```
# chkconfig --list cups
cups            0:off   1:off   2:on    3:on    4:on    5:on    6:off
#
# service cups status
cupsd (pid 1988) is running...
```

You can see that the `cupsd` daemon is set to start on every runlevel but 0, 1, and 6, and from the `service` command, you can see that it is currently running. Also, the process ID (PID) number is given for the daemon.

To see all of the services that are being offered by a Linux server using `systemd`, use the following command:

```
# systemctl list-unit-files --type=service | grep -v disabled
UNIT FILE                                   STATE
abrt-ccpp.service                           enabled
abrt-oops.service                           enabled
abrt-vmcore.service                         enabled
abrtd.service                               enabled
alsa-restore.service                        static
alsa-store.service                          static
anaconda-shell@.service                     static
arp-ethers.service                          enabled
atd.service                                 enabled
auditd.service                              enabled
avahi-daemon.service                        enabled
bluetooth.service                           enabled
console-kit-log-system-restart.service      static
console-kit-log-system-start.service        static
console-kit-log-system-stop.service         static
crond.service                               enabled
cups.service                                enabled
...
sshd-keygen.service                         enabled
sshd.service                                enabled
system-setup-keyboard.service               enabled
...
134 unit files listed.
```

Remember that the three status possibilities for a `systemd` service are enabled, disabled, or static. There's no need to include disabled to see which services are set to be active, which is effectively accomplished by using the `-v` option on the `grep` command, as shown in the preceding example. The state of static is essentially enabled and thus should be included.

To see if a particular service is running, use the following command:

```
# systemctl status cups.service
cups.service - CUPS Scheduler
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: active (running) since Wed 2019-09-18 17:32:27 EDT; 3 days ago
     Docs: man:cupsd(8)
 Main PID: 874 (cupsd)
   Status: "Scheduler is running..."
    Tasks: 1 (limit: 12232)
   Memory: 3.1M
   CGroup: /system.slice/cups.service
           └─874 /usr/sbin/cupsd -l
```

The `systemctl` command can be used to show the status of one or more services. In the preceding example, the printing service was chosen. Notice that the name of the service is `cups.service`. A great deal of helpful information about the service is given here, such as the fact that it is enabled and active, its start time, and its process ID (PID) as well.

Now that you can check the status of services and determine some information about them, you need to know how to accomplish starting, stopping, and reloading the services on your Linux server.

Stopping and Starting Services

The tasks of starting, stopping, and restarting services typically refer to immediate needs—in other words, managing services without a server reboot. For example, if you want to stop a service temporarily, then you are in the right place. However, if you want to stop a service and not allow it to be restarted at server reboot, then you actually need to disable the service, which is covered in the section "Enabling Persistent Services" later in this chapter.

Stopping and starting SysVinit services

The primary command for stopping and starting SysVinit services is the `service` command. With the `service` command, the name of the service that you want to control comes second in the command line. The last option is what you want to do to the service: `stop`, `start`, `restart`, and so on. The following example shows how to stop the `cups` service. Notice that an `OK` is given, which lets you know that `cupsd` has been successfully stopped:

```
# service cups status
cupsd (pid 5857) is running...
# service cups stop
Stopping cups:        [  OK  ]
# service cups status
cupsd is stopped
```

To start a service, you simply use a `start` option instead of a `stop` option on the end of the `service` command, as follows:

```
# service cups start
Starting cups:          [  OK  ]
# service cups status
cupsd (pid 6860) is running...
```

To restart a SysVinit service, the `restart` option is used. This option stops the service and then immediately starts it again:

```
# service cups restart
Stopping cups:          [  OK  ]
Starting cups:          [  OK  ]
# service cups status
cupsd (pid 7955) is running...
```

When a service is already stopped, a `restart` generates a FAILED status on the attempt to stop it. However, as shown in the example that follows, the service is successfully started when a restart is attempted:

```
# service cups stop
Stopping cups:          [  OK  ]
# service cups restart
Stopping cups:          [FAILED]
Starting cups:          [  OK  ]
# service cups status
cupsd (pid 8236) is running...
```

Reloading a service is different from restarting a service. When you `reload` a service, the service itself is not stopped. Only the service's configuration files are loaded again. The following example shows how to reload the `cups` daemon:

```
# service cups status
cupsd (pid 8236) is running...
# service cups reload
Reloading cups:             [  OK  ]
# service cups status
cupsd (pid 8236) is running...
```

If a SysVinit service is stopped when you attempt to reload it, you get a FAILED status. This is shown in the following example:

```
# service cups status
cupsd is stopped
# service cups reload
Reloading cups: [FAILED]
Stopping and starting systemd services
```

For the `systemd` daemon, the `systemctl` command works for stopping, starting, reloading, and restarting services. The options to the `systemctl` command should look familiar.

Stopping a service with systemd

In the example that follows, the status of the `cups` daemon is checked and then stopped using the `systemctl stop cups.service` command:

```
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: active (running) since Mon, 20 Apr 2020 12:36:3...
  Main PID: 1315 (cupsd)
    CGroup: name=systemd:/system/cups.service
            1315 /usr/sbin/cupsd -f
# systemctl stop cups.service
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: inactive (dead) since Tue, 21 Apr 2020 04:43:4...
    Process: 1315 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
    CGroup: name=systemd:/system/cups.service
```

Notice that when the status is taken, after stopping the `cups` daemon, the service is inactive (dead) but still considered enabled. This means that the `cups` daemon is still started upon server boot.

Starting a service with systemd

Starting the `cups` daemon is just as easy as stopping it. The example that follows demonstrates this ease:

```
# systemctl start cups.service
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: active (running) since Tue, 21 Apr 2020 04:43:5...
  Main PID: 17003 (cupsd)
    CGroup: name=systemd:/system/cups.service
            └─ 17003 /usr/sbin/cupsd -f
```

After the `cups` daemon is started, using `systemctl` with the status option shows the service is active (running). Also, its process ID (PID) number, 17003, is shown.

Restarting a service with systemd

Restarting a service means that a service is stopped and then started again. If the service was not currently running, restarting it simply starts the service.

```
# systemctl restart cups.service
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; enabled)
   Active: active (running) since Tue, 21 Apr 2020 04:45:2...
```

```
  Main PID: 17015 (cupsd)
   CGroup: name=systemd:/system/cups.service
           └─ 17015 /usr/sbin/cupsd -f
```

You can also perform a conditional restart of a service using `systemctl`. A conditional restart only restarts a service if it is currently running. Any service in an inactive state is not started.

```
# systemctl status cups.service
cups.service - CUPS Printing Service
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: inactive (dead) since Tue, 21 Apr 2020 06:03:32...
 Process: 17108 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
  CGroup: name=systemd:/system/cups.service
# systemctl condrestart cups.service
# systemctl status cups.service
cups.service - CUPS Printing Service
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: inactive (dead) since Tue, 21 Apr 2020 06:03:32...
 Process: 17108 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
  CGroup: name=systemd:/system/cups.service
```

Notice in the example that the `cups` daemon was in an inactive state. When the conditional restart was issued, no error messages were generated! The `cups` daemon was not started because conditional restarts affect active services. Thus, it is always a good practice to check the status of a service after stopping, starting, conditionally restarting, and so on.

Reloading a service with systemd

Reloading a service is different from restarting a service. When you `reload` a service, the service itself is not stopped. Only the service's configuration files are loaded again. Note that not all services are implemented to use the reload feature.

```
# systemctl status sshd.service
sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since Wed 2019-09-18 17:32:27 EDT; 3 days ago
 Main PID: 1675 (sshd)
   CGroup: /system.slice/sshd.service
           └─1675 /usr/sbin/sshd -D
# systemctl reload sshd.service
# systemctl status sshd.service
sshd.service - OpenSSH server daemon
   Loaded: loaded (/lib/systemd/system/sshd.service; enabled)
   Active: active (running) since Wed 2019-09-18 17:32:27 EDT; 3 days ago
  Process: 21770 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
       (code=exited, status=0/SUCCESSd)
```

```
 Main PID: 1675 (sshd)
   CGroup: /system.slice/sshd.service
           └─1675 /usr/sbin/sshd -D ...
```

Doing a `reload` of a service, instead of a `restart` prevents any pending service operations from being aborted. A `reload` is a better method for a busy Linux server.

Now that you know how to stop and start services for troubleshooting and emergency purposes, you can learn how to enable and disable services.

Enabling Persistent Services

You use `stop` and `start` for immediate needs, not for services that need to be persistent. A *persistent service* is one that is started at server boot time or at a particular runlevel. Services that need to be set as persistent are typically new services that the Linux server is offering.

Configuring persistent services for SysVinit

One of the nice features of the classic SysVinit daemon is that making a particular service persistent or removing its persistence is very easy to do. Consider the following example:

```
# chkconfig --list cups
cups            0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

On this Linux server, the `cups` service is not started at any runlevel, as shown with the `chkconfig` command. You can also check and see if any start (S) symbol links are set up in each of the seven runlevel directories, `/etc/rc.d/rc?.d`. Remember that SysVinit keeps symbolic links here for starting and stopping various services at certain runlevels. Each directory represents a particular runlevel; for example, `rc5.d` is for runlevel 5. Notice that only files starting with a `K` are listed, so there are links for killing off the `cups` daemon. None are listed with `S`, which is consistent with `chkconfig` because the `cups` daemon does not start at any runlevel on this server.

```
# ls /etc/rc.d/rc?.d/*cups
/etc/rc.d/rc0.d/K10cups  /etc/rc.d/rc3.d/K10cups
/etc/rc.d/rc1.d/K10cups  /etc/rc.d/rc4.d/K10cups
/etc/rc.d/rc2.d/K10cups  /etc/rc.d/rc5.d/K10cups
/etc/rc.d/rc6.d/K10cups
```

To make a service persistent at a particular runlevel, the `chkconfig` command is used again. Instead of the `--list` option, the `--level` option is used, as shown in the following code:

```
# chkconfig --level 3 cups on
# chkconfig --list cups
cups            0:off  1:off  2:off  3:on   4:off  5:off  6:off
# ls /etc/rc.d/rc3.d/S*cups
/etc/rc.d/rc3.d/S56cups
```

The service's persistence at runlevel 3 is verified by using both the `chkconfig --list` command and looking at the `rc3.d` directory for any files starting with the letter *S*.

To make a service persistent on more than one runlevel, you can do the following:

```
# chkconfig --level 2345 cups on
# chkconfig --list cups
cups            0:off  1:off  2:on   3:on   4:on   5:on   6:off
# ls /etc/rc.d/rc?.d/S*cups
/etc/rc.d/rc2.d/S56cups  /etc/rc.d/rc4.d/S56cups
/etc/rc.d/rc3.d/S56cups  /etc/rc.d/rc5.d/S56cups
```

Disabling a service is just as easy as enabling one with SysVinit. You just need to change the `on` in the `chkconfig` command to `off`. The following example demonstrates using the `chkconfig` command to disable the `cups` service at runlevel 5:

```
# chkconfig --level 5 cups off
# chkconfig --list cups
cups            0:off  1:off  2:on   3:on   4:on   5:off   6:off
# ls /etc/rc.d/rc5.d/S*cups
ls: cannot access /etc/rc.d/rc5.d/S*cups: No such file or directory
```

As expected, there is now no symbolic link, starting with the letter *S*, for the `cups` service in the `/etc/rc.d/rc5.d` directory.

For the `systemd` daemon, again the `systemctl` command is used. With it, you can disable and enable services on the Linux server.

Enabling a service with systemd

Using the `enable` option on the `systemctl` command sets a service to always start at boot (be persistent). The following shows exactly how to accomplish this:

```
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; disabled)
   Active: inactive (dead) since Tue, 21 Apr 2020 06:42:38 ...
 Main PID: 17172 (code=exited, status=0/SUCCESS)
   CGroup: name=systemd:/system/cups.service
# systemctl enable cups.service
Created symlink /etc/systemd/system/printer.target.wants/cups.service
    → /usr/lib/systemd/system/cups.service.
Created symlink /etc/systemd/system/sockets.target.wants/cups.socket
    → /usr/lib/systemd/system/cups.socket.
Created symlink /etc/systemd/system/multi-user.target.wants/cups.path
    → /usr/lib/systemd/system/cups.path.
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; enabled)
   Active: inactive (dead) since Tue, 21 Apr 2020 06:42:38...
 Main PID: 17172 (code=exited, status=0/SUCCESS)
   CGroup: name=systemd:/system/cups.service
```

Notice that the status of cups.service changes from disabled to enabled after using the enable option on systemctl. Also, notice that the enable option simply creates a few symbolic links. You may be tempted to create these links yourself. However, the preferred method is to use the systemctl command to accomplish this.

Disabling a service with systemd

You can use the disable option on the systemctl command to keep a service from starting at boot. However, it does not immediately stop the service. You need to use the stop option discussed in the section "Stopping a service with systemd." The following example shows how to disable a currently enabled service:

```
# systemctl disable cups.service
rm '/etc/systemd/system/printer.target.wants/cups.service'
rm '/etc/systemd/system/sockets.target.wants/cups.socket'
rm '/etc/systemd/system/multi-user.target.wants/cups.path'
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; disabled)
   Active: active (running) since Tue, 21 Apr 2020 06:06:41...
 Main PID: 17172 (cupsd)
   CGroup: name=systemd:/system/cups.service
           17172 /usr/sbin/cupsd -f
```

The disable option simply removes a few files via the preferred method of the systemctl command. Notice also in the preceding example that although the cups service is now disabled, the cups daemon is still active (running) and needs to be stopped manually. With systemd, some services cannot be disabled. These services are static services. Consider the following service, dbus.service:

```
# systemctl status dbus.service
dbus.service - D-Bus System Message Bus
  Loaded: loaded (/lib/systemd/system/dbus.service; static)
  Active: active (running) since Mon, 20 Apr 2020 12:35:...
 Main PID: 707 (dbus-daemon)
...
# systemctl disable dbus.service
# systemctl status dbus.service
dbus.service - D-Bus System Message Bus
  Loaded: loaded (/lib/systemd/system/dbus.service; static)
  Active: active (running) since Mon, 20 Apr 2020 12:35:...
 Main PID: 707 (dbus-daemon)
...
```

When the systemctl disable command is issued on dbus.service, it is simply ignored. Remember that static means that the service is enabled by default and cannot be disabled, even by root. Sometimes, disabling a service is not enough to make sure that it does not run. For example, you might want network.service to replace NetworkManager.service for starting network interfaces on your system. Disabling NetworkManager

would keep the service from starting on its own. However, if some other service listed NetworkManager as a dependency, that service would try to start NetworkManager when it started.

To disable a service in a way that prevents it from ever running on your system, you can use the `mask` option. For example, to set the NetworkManager service so that it never runs, type the following:

```
# systemctl mask NetworkManager.service
ln -s '/dev/null' '/etc/systemd/system/NetworkManager.service'
```

As the output shows, the `NetworkManager.service` file in `/etc` is linked to `/dev/null`. So even if someone tried to run that service, nothing would happen. To be able to use the service again, you could type `systemctl unmask NetworkManager.service`.

Now that you understand how to enable individual services to be persistent (and how to disable or mask individual services), you need to look at service groups as a whole. Next, I cover how to start groups of services at boot time.

Configuring a Default Runlevel or Target Unit

Whereas a persistent service is one that is started at server boot time, a persistent (default) runlevel or target unit is a group of services that are started at boot time. Both classic SysVinit and Upstart define these groups of services as runlevels, while `systemd` calls them target units.

Configuring the SysVinit default runlevel

You set the persistent runlevel for a Linux server using SysVinit in the `/etc/inittab` file. A portion of this file is shown here:

```
# cat /etc/inittab
#
# inittab       This file describes how the INIT process should
#               set up the system in a certain run-level.
...
id:5:initdefault:
...
```

The `initdefault` line in the example shows that the current default runlevel is runlevel 5. To change this, simply edit the `/etc/inittab` file using your favorite editor and change the 5 to one of the following runlevels: 2, 3, or 4. Do not use the runlevels 0 or 6 in this file! This would cause your server either to halt or reboot when it is started up.

For systemd, the term *target units* refers to groups of services to be started. The following shows the various target units that you can configure to be persistent and their equivalent backward-compatible, runlevel-specific target units:.

- multi-user.target =
  - runlevel2.target
  - runlevel3.target
  - runlevel4.target
- graphical.target = runlevel5.target

The persistent target unit is set via a symbolic link to the default.target unit file. Consider the following:

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Mar 13 17:27
 /etc/systemd/system/default.target ->
 /lib/systemd/system/runlevel5.target
# ls -l /lib/systemd/system/runlevel5.target
lrwxrwxrwx. 1 root root 16 Mar 27 15:39
 /lib/systemd/system/runlevel5.target ->
 graphical.target
```

The example shows that the current persistent target unit on this server is runlevel5.target because default.target is a symbolic link to the runlevel5.target unit file. However, notice that runlevel5.target is also a symbolic link and it points to graphical.target. Thus, this server's current persistent target unit is graphical.target.

To set a different target unit to be persistent, you simply need to change the symbolic link for default.target. To be consistent, stick with the runlevel target units if they are used on your server.

The following systemctl example changes the server's persistent target unit from graphical.target to multi-user.target:

```
# systemctl get-default
graphical.target
#
 systemctl set-default runlevel3.target
 Removed /etc/systemd/system/default.target.
 Created symlink /etc/systemd/system/default.target →
/usr/lib/systemd/system/multi-user.target.
# systemctl get-default
 multi-user.target
```

When the server is rebooted, the multi-user.target is the persistent target unit. Any services in the multi-user.target unit are started (activated) at that time.

Adding New or Customized Services

Occasionally, you need to add a new service to your Linux server. Also, you may have to customize a particular service. When these needs arise, you must follow specific steps for your Linux server's initialization daemon to either take over the management of the service or recognize the customization of it.

Adding new services to SysVinit

When adding a new or customized service to a Linux SysVinit server, you must complete three steps in order to have the service managed by SysVinit:.

1. Create a new or customized service script file.
2. Move the new or customized service script to the proper location for SysVinit management.
3. Set appropriate permission on the script.
4. Add the service to a specific runlevel.

Step 1: Create a new or customized service script file

If you are customizing a service script, simply make a copy of the original unit file from `/etc/rc.d/init.d` and add any desired customizations.

If you are creating a new script, you need to make sure you handle all of the various options that you want the `service` command to accept for your service, such as `start`, `stop`, `restart`, and so on.

For a new script, especially if you have never created a service script before, it would be wise to make a copy of a current service script from `/etc/rc.d/init.d` and modify it to meet your new service's needs. Consider the following partial example of the `cupsd` service's script:

```
# cat /etc/rc.d/init.d/cups
#!/bin/sh
#
...
#   chkconfig: 2345 25 10

...
start () {
        echo -n $"Starting $prog: "
        # start daemon
        daemon $DAEMON
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && touch /var/lock/subsys/cups
        return $RETVAL
}
```

```
stop () {
        # stop daemon
        echo -n $"Stopping $prog: "
        killproc $DAEMON
        RETVAL=$?
       echo        [ $RETVAL = 0 ] && rm -f /var/lock/subsys/cups
}

restart() {
        stop
        start
}

case $1 in
...
```

The `cups` service script starts out by creating functions for each of the `start`, `stop`, and `restart` options. If you feel uncomfortable with shell script writing, review Chapter 7, "Writing Simple Shell Scripts," to improve your skills.

One line you should be sure to check and possibly modify in your new script is the `chkconfig` line that is commented out; for example:

```
#   chkconfig: 2345 25 10
```

When you add the service script in a later step, the `chkconfig` command reads that line to set runlevels at which the service starts (2, 3, 4, and 5), its run order when the script is set to start (25), and its kill order when it is set to stop (10).

Check the boot order in the default runlevel before adding your own script, as shown in this example:

```
# ls /etc/rc5.d
...
/etc/rc5.d/S22messagebus
/etc/rc5.d/S23NetworkManager
/etc/rc5.d/S24nfslock
/etc/rc5.d/S24openct
/etc/rc5.d/S24rpcgssd
/etc/rc5.d/S25blk-availability
/etc/rc5.d/S25cups
/etc/rc5.d/S25netfs
/etc/rc5.d/S26acpid
/etc/rc5.d/S26haldaemon
/etc/rc5.d/S26hypervkvpd
/etc/rc5.d/S26udev-post

...
```

In this case, the `chkconfig` line in the `S25My_New_Service` script will cause the script to be added after `S25cups` and before `S25netfs` in the boot order. You can change the `chkconfig` line in the service script if you want the service to start earlier (use a smaller number) or later (use a larger number) in the list of service scripts.

Step 2: Add the service script to /etc/rc.d/init.d

After you have modified or created and tested your service's script file, you can move it to the proper location, `/etc/rc.d/init.d`:

```
# cp My_New_Service /etc/rc.d/init.d
# ls /etc/rc.d/init.d/My_New_Service
/etc/rc.d/init.d/My_New_Service
```

Step 3: Set appropriate permission on the script

The script should be executable:

```
# chmod 755 /etc/rc.d/init.d/My_New_Service
```

Step 4: Add the service to runlevel directories

This final step sets up the service script to start and stop at different runlevels and checks that the service script works.

1. To add the script based on the `chkconfig` line in the service script, type the following:

   ```
   # chkconfig --add My_New_Service
   # ls /etc/rc?.d/*My_New_Service
   /etc/rc0.d/K10My_New_Service  /etc/rc4.d/S25My_New_Service
   /etc/rc1.d/K10My_New_Service  /etc/rc5.d/S25My_New_Service
   /etc/rc2.d/S25My_New_Service  /etc/rc6.d/K10My_New_Service
   /etc/rc3.d/S25My_New_Service
   ```

   Based on the previous example (`chkconfig: 2345 25 10`), symbolic links to the script set the service to start in the position 25 (`S25`) for runlevels 2, 3, 4, and 5. Also, links are set to stop (or not start) at runlevels 0, 1, and 6.

2. After you have made the symbolic link(s), test that your new or modified service works as expected before performing a server reboot.

   ```
   # service My_New_Service start
   Starting My_New_Service:       [  OK  ]
   # service My_New_Service stop
   ```

After everything is in place, your new or modified service starts at every runlevel that you have selected on your system. Also, you can start or stop it manually using the `service` command.

Adding new services to systemd

When adding a new or customized service to a Linux `systemd` server, you have to complete three steps in order to have the service managed by `systemd`:

1. Create a new or customized service configuration unit file for the new or customized service.
2. Move the new or customized service configuration unit file to the proper location for `systemd` management.
3. Add the service to a specific target unit's Wants to have the new or customized service start automatically with other services.

Step 1: Create a new or customized service configuration unit file

If you are customizing a service configuration unit file, simply make a copy of the original unit file from `/lib/systemd/system` and add any desired customizations.

For new files, obviously, you are creating a service unit configuration file from scratch. Consider the following basic service unit file template. At bare minimum, you need `Description` and `ExecStart` options for a service unit configuration file:

```
# cat My_New_Service.service
[Unit]
Description=My New Service
[Service]
ExecStart=/usr/bin/My_New_Service
```

For additional help on customizing or creating a new configuration unit file and the various needed options, you can use the man pages. At the command line, type **`man systemd.service`** to find out more about the various service unit file options.

Step 2: Move the service configuration unit file

Before you move the new or customized service configuration unit file, you need to be aware that there are two potential locations to store service configuration unit files. The one you choose determines whether the customizations take effect and if they remain persistent through software upgrades.

You can place your system service configuration unit file in one of the following two locations:

- `/etc/systemd/system`
  - This location is used to store customized local service configuration unit files.
  - Files in this location are not overwritten by software installations or upgrades.

    Files here are used by the system *even* if there is a file of the same name in the `/lib/systemd/system` directory.
- `/lib/systemd/system`
  - This location is used to store system service configuration unit files.
  - Files in this location are overwritten by software installations and upgrades.

Files here are used by the system only if there is no file of the same name in the `/etc/systemd/system` directory.

Thus, the best place to store your new or customized service configuration unit file is in `/etc/systemd/system`.

TIP

When you create a new or customized service, in order for the change to take effect without a server reboot, you need to issue a special command. At the command line, type **`systemctl daemon-reload`**.

Step 3: Add the service to the Wants directory

This final step is optional. It needs to be done only if you want your new service to start with a particular `systemd` target unit. For a service to be activated (started) by a particular target unit, it must be in that target unit's `Wants` directory.

First, add the line `WantedBy=desired.target` to the bottom of your service configuration unit file. The following example shows that the desired target unit for this new service is `multi-user.target`:

```
# cat /etc/systemd/system/My_New_Service.service
[Unit]
Description=My New Fake Service
[Service]
ExecStart=/usr/bin/My_New_Service
[Install]
WantedBy=multi-user.target
```

To add a new service unit to a target unit, you need to create a symbolic link. The following example shows the files located in the `multi-user.target` unit's `Wants` directory. Previously, in the section "Understanding `systemd` initialization," the `systemctl` command was used to list Wants, and it is still the preferred method. Notice that in this directory, the files are symbolic links pointing to service unit configuration files in the `/lib/systemd/system` directory.

```
# ls /etc/systemd/system/multi-user.target.wants
abrt-ccpp.service     cups.path           remote-fs.target
abrtd.service         fcoe.service        rsyslog.service
abrt-oops.service     irqbalance.service  sendmail.service
abrt-vmcore.service   lldpad.service      sm-client.service
atd.service           mcelog.service      sshd-keygen.service
auditd.service        mdmonitor.service   sshd.service
...
# ls -l /etc/systemd/system/multi-user.target.wants
total 0
lrwxrwxrwx. 1 root root 37 Nov  2 22:29 abrt-ccpp.service ->
    /lib/systemd/system/abrt-ccpp.service
lrwxrwxrwx. 1 root root 33 Nov  2 22:29 abrtd.service ->
    /lib/systemd/system/abrtd.service
...
```

```
lrwxrwxrwx. 1 root root 32 Apr 26 20:05 sshd.service ->
    /lib/systemd/system/sshd.service
```

The following illustrates the process of adding a symbolic link file for `My_New_Service`:

```
# ln -s /etc/systemd/system/My_New_Service.service
 /etc/systemd/system/multi-user.target.wants/My_New_Service.service
```

A symbolic link is created in the `multi-user.target.wants` directory. Now the new service, `My_New_Service`, is activated (started) when the `multi-user.target` unit is activated.

Tip

If you want to change the `systemd` target unit for a service, you need to change the symbol link to point to a new target `Wants` directory location. Use the `ln -sf` command to force any current symbolic link to be broken and the new designated symbolic link to be enforced.

Together, the three steps get your new or customized service added to a Linux `systemd` server. Remember that at this point, a new service is not running until a server reboot. To start the new service before a reboot, review the commands in the section "Stopping and Starting Services."

Summary

How you start and stop services is dependent upon what initialization daemon is used by your Linux server: SysVinit, Upstart, or Systemd. Before you do any service management, be sure to use the examples in this chapter to help you determine your Linux server's initialization daemon.

The concepts of starting and stopping services go along with other service management concepts, such as making a service persistent, starting certain services at server boot time, reloading a service, and restarting a service. Understanding these concepts is very helpful as you learn about configuring and managing a Linux print server in the next chapter.

Exercises

Refer to the material in this chapter to complete the tasks that follow. If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task). Try each of the exercises before referring to the answers. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well).

1. Determine which initialization daemon your server is currently using.
2. What command can you use to check the status of the `sshd` daemon, depending on the initialization daemon in use on your Linux server?

3. Determine your server's previous and current runlevel.
4. How can you change the default runlevel or target unit on your Linux server?
5. For each initialization daemon, what commands list services running (or active) on your server?
6. List the running (or active) services on your Linux server.
7. For each initialization daemon, what commands show a particular service's current status?
8. Show the status of the `cups` daemon on your Linux server.
9. Attempt to restart the `cups` daemon on your Linux server.
10. Attempt to reload the `cups` daemon on your Linux server.

CHAPTER 16

Configuring a Print Server

IN THIS CHAPTER

Understanding printing in Linux

Setting up printers

Using printing commands

Managing document printing

Sharing printers

You can configure your Linux system to use printers that are connected directly to it (via a USB port) or that are available for printing over the network. Likewise, any printer that you configure on your local system can be shared with users on other Linux, Windows, or Mac systems by opening up your printer as a print server.

You configure a printer as a native Linux printer in Fedora, RHEL, Ubuntu, and other Linux systems with the *Common UNIX Printing System (CUPS)*. To configure a printer to work as a Microsoft Windows style of print server, you can use the Samba service in Linux.

This chapter focuses on CUPS. In particular, it shows you the graphical front end to CUPS, called the *Print Settings window*, which comes with Fedora, Red Hat Enterprise Linux, and other Linux distributions. Using Print Settings, you can also configure your printers as print servers so that people can print to your printer from their own computers.

If you don't have a desktop, or you want to print from within a shell script, this chapter shows you how to use printing commands. From the command line, print commands such as `lp` are available for carrying out printing. Commands also exist for querying print queues (`lpq`), manipulating print queues (`cupsenable`, `cupsdisable`, and `cupsreject`), and removing print queues (`lprm`).

Common UNIX Printing System

CUPS has become the standard for printing from Linux and other UNIX-like operating systems. It was designed to meet today's needs for standardized printer definitions and sharing on Internet

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Protocol–based networks (as most computer networks are today). Nearly every Linux distribution today comes with CUPS as its printing service. Here are some of the service's features:

IPP CUPS is based on the Internet Printing Protocol (`http://www.pwg.org/ipp`), a standard that was created to simplify how printers can be shared over IP networks. In the IPP model, printer servers and clients who want to print can exchange information about the model and features of a printer using the HTTP (that is, web content) protocol. A server can also broadcast the availability of a printer so that a printing client can easily find a list of locally available printers without configuration.

Drivers CUPS also standardized how printer drivers are created. The idea was to have a common format that could be used by printer manufacturers so that a driver could work across all different types of UNIX systems. That way, a manufacturer had to create the driver only once to work for Linux, Mac OS X, and a variety of UNIX derivatives.

Printer classes You can use printer classes to create multiple print server entries that point to the same printer or one print server entry that points to multiple printers. In the first case, multiple entries can each allow different options (such as pointing to a particular paper tray or printing with certain character sizes or margins). In the second case, you can have a pool of printers so that printing is distributed. In this instance, a malfunctioning printer, or a printer that is dealing with very large documents, won't bring all printing to a halt. CUPS also supports *implicit classes*, which are print classes that form by merging identical network printers automatically.

Printer browsing With printer browsing, client computers can see any CUPS printers on your local network with browsing enabled. As a result, clients can simply select the printers that they want to use from the printer names broadcast on the network, without needing to know in advance what the printers are named and where they are connected. You can turn off the feature to prevent others on the local network from seeing a printer.

UNIX print commands To integrate into Linux and other UNIX environments, CUPS offers versions of standard commands for printing and managing printers that have been traditionally offered with UNIX systems.

Instead of using the Print Settings window, you can configure CUPS printing in other ways as well:

Configuring CUPS from a browser The CUPS project itself offers a web-based interface for adding and managing printers. With the `cupsd` service running, type **`localhost:631`** from a web browser on the computer running the CUPS service to manage printing. (See the section "Using web-based CUPS administration" later in this chapter.)

Configuring CUPS manually You also can configure CUPS manually (that is, edit the configuration files and start the `cupsd` daemon from the command line).

Configuration files for CUPS are contained in the `/etc/cups` directory. In particular, you might be interested in the `cupsd.conf` file, which identifies permissions, authentication, and other information for the printer daemon, and `printers.conf`, which identifies addresses and options for configured printers. Use the `classes.conf` file to define local printer classes.

Printing Directly from Windows to CUPS

You can print to CUPS from non-UNIX systems as well. For example, you can use a PostScript printer driver to print directly from a Windows system to your CUPS server. You can use CUPS without modification by configuring the Windows computer with a PostScript driver that uses `http://printservername:631/printers/targetPrinter` as its printing port.

You may also be able to use the native Windows printer drivers for the printer instead of the PostScript driver. If the native Windows driver does not work right out of the box on your CUPS print queue, you can create a Raw Print Queue under CUPS and use that instead. The Raw Print Queue directly passes through the data from the Windows native print driver to the printer.

To use CUPS, you must have the `cups` package installed in Fedora or RHEL. Most desktop Linux distributions include CUPS during the initial system install. If it is not installed in a Fedora or RHEL install, install it by typing the following:

```
# yum install cups cups-client
```

Setting Up Printers

Although using the printer administration tools specifically built for your distribution is usually best, many Linux systems simply rely on the tools that come with the CUPS software package.

The following sections explore how to use CUPS web-based administration tools that come with every Linux distribution. Then it examines the Print Settings tool `system-config-printer`, which is available with Fedora systems to enable you to set up printers. In some cases, no configuration is necessary, because connected printers can be automatically detected and configured. To install the Print Settings tool in Fedora, as root, enter the following `dnf` (or `yum`) command:

```
# yum install system-config-printer
```

Adding a printer automatically

CUPS printers can be configured to broadcast their availability on the network automatically so that a client system can detect and use them without configuration. Connect a USB printer to your computer, and the printer can be automatically detected and made

available. In fact, if you attach a local printer in Fedora and the print driver is not yet installed, you are prompted to install the software packages needed to use the printer.

The first time that you go to print a document or view your Print Settings tool, the printers are ready to use. Further configuration can be done using the web-based CUPS administration tool or the Print Settings window.

Using web-based CUPS administration

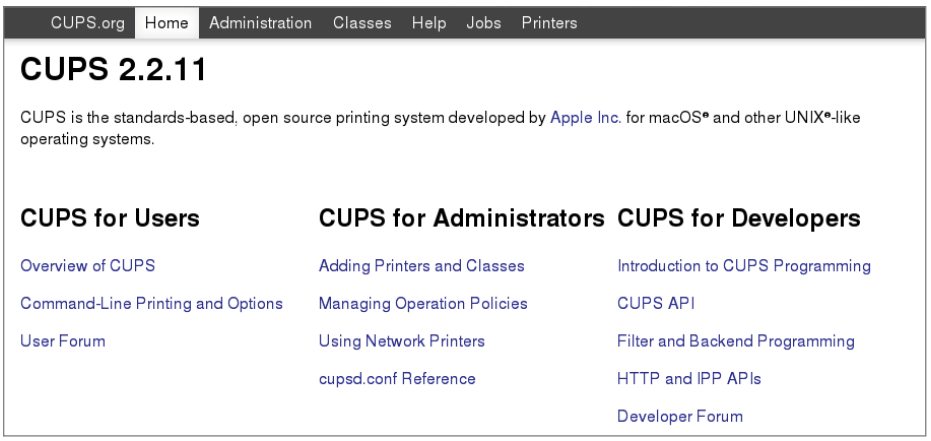
CUPS offers its own web-based administrative tool for adding, deleting, and modifying printer configurations on your computer. The CUPS print service (using the `cupsd` daemon) listens on port 631 to provide access to the CUPS web-based administrative interface and share printers.

If CUPS is already running on your computer, you can immediately use CUPS web-based administration from your web browser. To see whether CUPS is running and to start setting up your printers, open a web browser on the local computer and type this into its location box: **`http://localhost:631/`**.

A prompt for a valid login name and password may appear when you request functions that require it. If so, type the root login name and the root user's password and click OK. A screen similar to the one shown in Figure 16.1 appears.

FIGURE 16.1

CUPS provides a web-based administration tool.



Allow remote printing administration

By default, web-based CUPS administration is available only from the local host. To access web-based CUPS administration from another computer, from the main CUPS page:

1. Select the Administration tab.

2. Select the check box next to Allow remote administration.
3. Select the Change Settings button.

Then open your computer's firewall to allow connections to TCP port 631 to allow access to the service. After that, from any browser that has access to your local network, you can access the CUPS Administration page by going to port 631 on the CUPS server (for example, `http://host.example.com:631`).

You may need to restart CUPS for the change to take effect: **`systemctl restart cups.service`**. If you are not already running the browser as the root user, you must also enter the root username and password.

Add a printer not automatically detected

To configure a printer that is not automatically detected, you can add a printer from the Administration screen. With the Administration screen displayed, you can add a printer as follows:

1. Click the Add Printer button. The Add New Printer screen appears.
2. Select the device to which the printer is connected. The printer can be connected locally to a parallel, SCSI, serial, or USB port directly on the computer. Alternatively, you can select a network connection type for Apple printers (AppSocket or HP JetDirect), Internet Printing Protocol (`http`, `https`, `ipps`, or `ipp`), or a Windows printer (using Samba or SMB).
3. If prompted for more information, you may need to describe the connection to the printer further. For example, you might be asked for the network address for an IPP or Samba printer.
4. Type a name, location, and description for the printer; select if you want to share this printer and click Continue.
5. Select the make of the print driver. If you don't see the manufacturer of your printer listed, choose PostScript for a PostScript printer or HP for a PCL printer. For the manufacturer you choose, you can select a specific model.
6. Set options. If you are asked to set options for your printer, you may do so. Then select Set Printer Options to continue.
7. Your printer should be available. If the printer is added successfully, click the name of your printer to have the new printer page appear; from the printer page, you can select Maintenance or Administration to print a test page or modify the printer configuration.

With the basic printer configuration done, you can now do further work with your printers. Here are a few examples of what you can do:

List print jobs Click Show All Jobs to see what print jobs are currently active from any of the printers configured for this server. Click Show Completed Jobs to see information about jobs that are already printed.

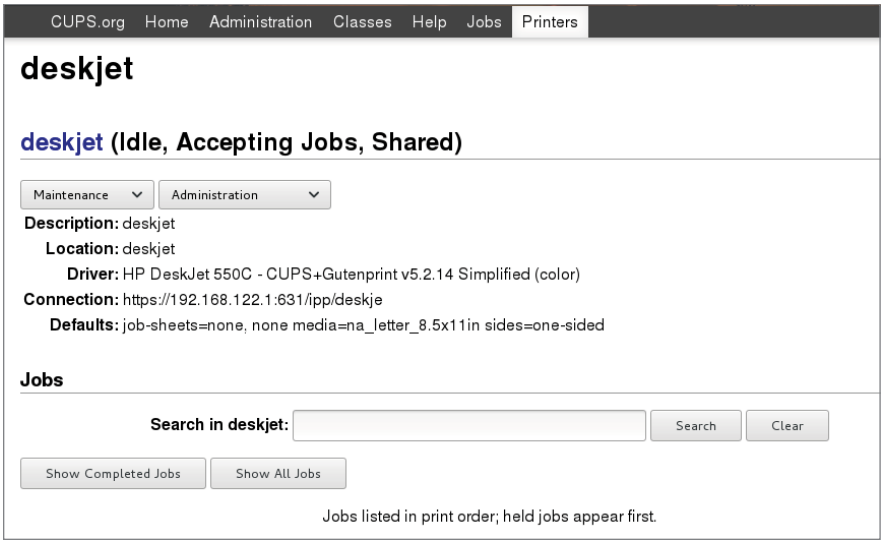
Create a printer class Click the Administration tab, choose Add Class, and identify a name, description, and location for a printer class. From the list of Printers (Members) configured on your server, select the ones to go into this class.

Cancel or move a print job If you print a 100-page job by mistake, or if the printer is spewing out junk, the Cancel feature can be very handy. Likewise, if you sent a print job to the wrong printer, the Move Job selection can be useful. From the Administration tab, click Manage Jobs; then click Show Active Jobs to see what print jobs are currently in the queue for the printer. Select the Cancel Job button next to the print job that you want to cancel or select Move Job to move the print job to a different printer.

View printers You can click the Printers tab from the top of any of the CUPS web-based administration pages to view the printers that you have configured. For each printer that appears, you can select Maintenance or Administrative tasks. Under Maintenance, click Pause Printer (to stop the printer from printing but still accept print jobs for the queue), Reject Jobs (to not accept any further print jobs for the moment), Move All Jobs (to move them to another printer), Cancel All Jobs (to delete all print jobs), or Print Test Page (to print a page). Figure 16.2 shows the information on the Printers tab for a specific printer.

FIGURE 16.2

You can do administration tasks from the Printers tab.



Using the Print Settings window

If you are using Fedora, you can use the Print Settings window to set up your printers. In fact, I recommend that you use it instead of CUPS web administration because the resulting printer configuration files are tailored to work with the way the CUPS service is started on those systems in Fedora. After the package is installed (`dnf install system-config-printer`), to install a printer from your GNOME desktop, start the Print Settings window by typing **Print Settings** from the Activity screen, or as root user by typing **`system-config-printer`**. This tool enables you to add and delete printers and edit printer properties. It also enables you to send test pages to those printers to make sure that they are working properly.

The key here is that you are configuring printers that are managed by your print daemon (`cupsd` for the CUPS service). After a printer is configured, users on your local system can use it. You can refer to the section "Configuring Print Servers" to learn how to make the server available to users from other computers on your network.

The printers that you set up can be connected directly to your computer (as on a USB port) or to another computer on the network (for example, from another UNIX system or Windows system).

Configuring local printers with the Print Settings window

Add a local printer (in other words, a printer connected directly to your computer) with the Printers window using the procedure that follows.

Adding a local printer

To add a local printer from a GNOME desktop in the latest version of Fedora, follow these steps:

1. **Type the following to open the Print Settings window**:

   ```
   # system-config-printer &
   ```

   The Printing window appears.
2. **Click Add**. (If asked, click the Adjust Firewall button to allow access to the printer port 631.) A New Printer window appears.
3. **If the printer that you want to configure is detected, simply select it and click Forward**. If it is not detected, choose the device to which the printer is connected (LPT #1 and Serial Port #1 are the first parallel and serial ports, respectively) and click Forward. (Type **`/usr/sbin/lpinfo -v | less`** in a shell to see printer connection types.) You are asked to identify the printer's driver.
4. **To use an installed driver for your printer, choose Select Printer From Database, and then choose the manufacturer of your printer.** As an alternative, you could select Provide PPD File and supply your own PPD file (for example, if you have

a printer that is not supported in Linux and you have a driver that was supplied with the printer). (PPD stands for PostScript Printer Description.) Select Forward to see a list of printer models from which you can choose.

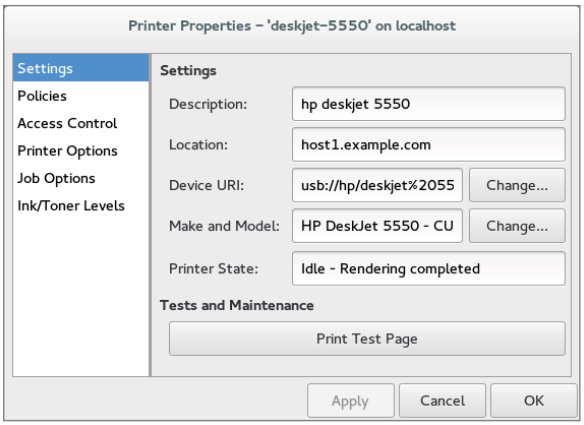
Tip

If your printer doesn't appear on the list but supports PCL (HP's Printer Control Language), try selecting one of the HP printers (such as HP LaserJet). If your printer supports PostScript, select PostScript printer from the list. Selecting Raw Print Queue enables you to send documents that are already formatted for a particular printer type to a specific printer.

5. **With your printer model selected, click the driver that you want to use with it and then click Forward to continue.**
6. **Add the following information, and click Forward:**
   a. **Printer Name:** Add the name that you want to give to identify the printer. The name must begin with a letter, but after the initial letter, it can contain a combination of letters, numbers, dashes (-), and underscores (_). For example, an HP printer on a computer named *maple* could be named *hp-maple*.
   b. **Description:** Add a few words describing the printer, such as its features (for example, an HP LaserJet 2100M with PCL and PS support).
   c. **Location:** Add some words that describe the printer's location (for example, "In Room 205 under the coffee maker").
7. **When the printer is added, click No or Yes if you're prompted to print a test page.** The new printer entry appears in the Print Settings window. Double-click the printer to see the Properties window for that printer, as shown in Figure 16.3.

FIGURE 16.3

The Printer Properties window after adding a printer



8. **If you want the printer to be your default printer, right-click the printer and select Set As Default.** As you add other printers, you can change the default printer by selecting the one you want and selecting Set As Default again.
9. **Make sure that the printing is working.** Open a Terminal window and use the `lp` command to print a file (such as `lp /etc/hosts`). (If you want to share this printer with other computers on your network, refer to the section "Configuring Print Servers" later in this chapter.)

Editing a local printer

After double-clicking the printer that you want to configure, choose from the following menu options to change its configuration:

Settings: The Description, Location, Device URI, and Make and Model information you created earlier are displayed in this dialog box.

Policies: Click Policies to set the following items:

State: Select check boxes to indicate whether the printer will print jobs that are in the queue (Enabled), accept new jobs for printing (Accepting Jobs), and be available to be shared with other computers that can communicate with your computer (Shared). You also must select Server Settings and click the "Share Published printers connected to this system" check box before the printer will accept print jobs from other computers.

Policies: In case of error, the stop-printer selection causes all printing to that printer to stop. You can also select to have the job discarded (abort-job) or retried (retry-job) in the event of an error condition.

Banner: There are no starting or ending banner pages by default for the printer. Choose starting or ending banner pages that include text such as *Classified*, *Confidential*, *Secret*, and so on.

Access Control: If your printer is a shared printer, you can select this window to create a list that either allows users access to the printer (with all others denied) or denies users access to the printer (with all others allowed).

Printer Options: Click Printer Options to set defaults for options related to the printer driver. The available options are different for different printers. Many of these options can be overridden when someone prints a document. Here are examples of a few of the options that you might (or might not) have available:

Watermark: Several Watermark settings are available to enable you to add and change watermarks on your printed pages. By default, Watermark and Overlay are off (None). By selecting Watermark (behind the text) or Overlay (over the text), you can set the other Watermark settings to determine how watermarks and overlays are done. Watermarks can go on every page (All) or only the first page (First Only). Select Watermark Text to choose what words are used for the watermark or overlay (Draft, Copy, Confidential, Final, and so on). You can then select the font type, size, style, and intensity of the watermark or overlay.

Resolution Enhancement: You can use the printer's current settings or choose to turn resolution enhancement on or off.

Page Size: The default is US letter size, but you can also ask the printer to print legal size, envelopes, ISO A4 standard, or several other page sizes.

Media Source: Choose which tray to print from. Select Tray 1 to insert pages manually.

Levels of Gray: Choose to use the printer's current levels of gray or have enhanced or standard gray levels turned on.

Resolution: Select the default printing resolution (such as 300, 600, or 1,200 dots per inch). Higher resolutions result in better quality but take longer to print.

EconoMode: Either use the printer's current setting or choose a mode where you save toner or one where you have the highest possible quality.

Job Options: Click Job Options to set common default options that will be used for this printer if the application printing the job doesn't already set them. These include Common Options (number of copies, orientation, scale to fit, and pages per side), Image Options (scaling, saturation, hue, and gamma), and Text Options (characters/inch, lines/inch, and margin settings).

Ink/Toner Levels: Click Ink/Toner Levels to see information on how much ink or toner your printer has left. (Not all printers report these values.)

Click Apply when you are satisfied with the changes you made to the local printer.

Configuring remote printers

To use a printer that is available on your network, you must identify that printer to your Linux system. Supported remote printer connections include Networked CUPS (IPP) printers, Networked UNIX (LPD) printers, Networked Windows (Samba) printers, and JetDirect printers. (Of course, both CUPS and UNIX print servers can be run from Linux systems as well as other UNIX systems.)

In each case, you need a network connection from your Linux system to the servers to which those printers are connected. To use a remote printer requires that someone set up that printer on the remote server computer. See the section "Configuring Print Servers" later in this chapter for information on how to do that on your Linux server.

Use the Print Settings window (`system-config-printer`) to configure each of the remote printer types. This is how it is done:

1. **From the GNOME 3 Activities screen, type** Print Settings **and press Enter**.
2. **Click Add**. The New Printer window appears.
3. **Depending on the type of ports that you have on your computer, select one of the following**:
   a. **LPT #1**: Use this for a printer connected to your parallel port.
   b. **Serial Port #1**: Use this for a printer connected to your serial port.

c. **Network Printer**: Under this heading, you can search for network printers (by hostname or IP address) or type in the URI for several different printer types:

   i. **Find Network Printer**: Instead of entering a printer URI, you can provide a hostname or IP address for the system that has the printer to which you want to print. Any printers found on that host appear on the window, ready for you to add.

   ii. **AppleSocket/HP JetDirect**: Use this for a JetDirect printer.

   iii. **Internet Printing Protocol (IPP)**: Use this for a CUPS or other IPP printer. Most Linux and Mac OS X printers fall into this category.

   iv. **Internet Printing Protocol (HTTPS)**: Use this for a CUPS or other IPP printer being shared over a secure connection (valid certificates required).

   v. **LPD/LPR Host or Printer**: Use this for a UNIX printer.

   vi. **Windows Printer via SAMBA**: Use this for a Windows system printer.

Continue with the steps in whichever of the following sections is appropriate.

Adding a remote CUPS printer

If you chose to add a CUPS (IPP) printer that is accessible over your local network from the Print Settings window, you must add the following information to the window that appears:

Host This is the hostname of the computer to which the printer is attached (or otherwise accessible). This can be an IP address or TCP/IP hostname for the computer. The TCP/IP name is accessible from your `/etc/hosts` file or through a DNS name server.

Queue This is the printer name on the remote CUPS print server. CUPS supports printer instances, which allows each printer to have several sets of options. If the remote CUPS printer is configured this way, you can choose a particular path to a printer, such as `hp/300dpi` or `hp/1200dpi`. A slash character separates the print queue name from the printer instance.

Complete the rest of the procedure as you would for a local printer (see the section "Adding a local printer" earlier in this chapter).

Adding a remote UNIX (LDP/LPR) printer

If you chose to add a UNIX printer (LPD/LPR) from the Print Settings window, you must add the following information to the window that appears:

Host This is the hostname of the computer to which the printer is attached (or otherwise accessible). This is the IP address or hostname for the computer (the hostname is accessible from your `/etc/hosts` file or through a DNS name server). Select the Probe button to search for the host.

Queue This is the printer name on the remote UNIX computer.

Complete the rest of the procedure as you would for a local printer (see the section "Adding a local printer" earlier in this chapter).

Tip

If the print job you send to test the printer is rejected, the print server computer may not have allowed you access to the printer. Ask the remote computer's administrator to add your hostname to the `/etc/lpd.perms` file. (Enter `lpstat -d` *`printer`* to see the status of your print job.)

Adding a Windows (SMB) printer

Enabling your computer to access an SMB printer (the Windows printing service) involves adding an entry for the printer in the Select Connection window.

When you choose to add a Windows printer to the Print Settings window (Windows Printer via Samba), select Browse to see a list of computers on your network that have been detected as offering SMB services (file and/or printing service). You can configure the printer from this window as follows:

1. **Type the URI of the printer, excluding the leading `smb://`.** For example, you might type `/host1/myprinter` or `/mygroup/host1/myprinter`.
2. **Select either "Prompt user if authentication is required" or "Set authentication details now.'**
3. **If you chose "Set authentication details now," fill in the username and password needed to access the SMB printer; then click Verify to check that you can authenticate to the server.**
4. **Click Forward to continue.**

Alternatively, you can identify a server that does not appear on the list of servers. Type the information needed to create an SMB URI that contains the following information:

Workgroup This is the workgroup name assigned to the SMB server. Using the workgroup name isn't necessary in all cases.

Server This is the NetBIOS name or IP address for the computer, which may or may not be the same as its TCP/IP name. To translate this name into the address needed to reach the SMB host, Samba checks several places where the name may be assigned to an IP address. Samba checks the following (in the order shown) until it finds a match: the local `/etc/hosts` file, the local `/etc/lmhosts` file, a WINS server on the network, and responses to broadcasts on each local network interface to resolve the name.

Share This is the name under which the printer is shared with the remote computer. It may be different from the name by which local users of the SMB printer know the printer.

User A username is required by the SMB server system to give you access to the SMB printer. A username is not necessary if you are authenticating the printer based on share-level rather than user-level access control. With share-level access, you can add a password for each shared printer or file system.

Password Use the password associated with the SMB username or the shared resource, depending on the kind of access control being used.

CAUTION
When you enter a username and password for SMB, the information is stored unencrypted in the `/etc/cups/printers.conf` file. Be sure that the file remains readable only by root.

The following is an example of the SMB URI that you could add to the `SMB://` box:

```
jjones:my9passswd@FSTREET/NS1/hp
```

The URI shown here identifies the username (`jjones`), the user's password (`my9passswd`), the workgroup (`FSTREET`), the server (`NS1`), and the printer queue name (`hp`).

Complete the rest of the procedure as you would for a local printer (see the section "Adding a local printer" earlier in this chapter).

If everything is set up properly, you can use the standard `lp` command to print the file to the printer. Using this example, employ the following form for printing:

```
$ cat file1.ps | lp -P NS1-PS
```

TIP
If you are receiving failure messages, make sure that the computer to which you are printing is accessible. For the Printer NS1 hp example, you can type **`smbclient -L NS1 -U jjones`**. Then type the password (`my9passswd`, in this case). The `-L` asks for information about the server; the `-U jjones` says to log in the user `jjones`. If you get a positive name query response after you enter a password, you should see a list of shared printers and files from that server. Check the names and try printing again.

Working with CUPS Printing

Tools such as CUPS web-based administration and the Print Settings window effectively hide the underlying CUPS facility. Sometimes, however, you want to work directly with the tools and configuration files that come with CUPS. The following sections describe how to use some special CUPS features.

Configuring the CUPS server (cupsd.conf)

The `cupsd` daemon process listens for requests to your CUPS print server and responds to those requests based on settings in the `/etc/cups/cupsd.conf` file. The configuration variables in the `cupsd.conf` file are in the same form as those in the Apache configuration file (`httpd.conf` or `apache2.conf`). Type **`man cupsd.conf`** to see details on any of the settings.

The Print Settings window adds access information to the `cupsd.conf` file. For other Linux systems, or if you don't have a desktop on your server, you may need to configure the `cupsd.conf` file manually. You can step through the `cupsd.conf` file to tune your CUPS server further. Most of the settings are optional or can just be left as the default. Let's look at some of the settings that you can use in the `cupsd.conf` file.

No classification is set by default. With the classification set to `topsecret`, you can have Top Secret displayed on all pages that go through the print server:

```
Classification topsecret
```

Other classifications you can substitute for `topsecret` include `classified`, `confidential`, `secret`, and `unclassified`.

The term *browsing* refers to the act of broadcasting information about your printer on your local network and listening for other print servers' information. The `cups-browsed` setting is used to browse shared, remote printers. Browsing is on by default for all local networks (`@LOCAL`). You can allow CUPS browser information (`BrowseAllow`) for additional selected addresses. Browsing information is broadcast, by default, on address `255.255.255.255`. Here are examples of several browsing settings:

```
Browsing On
BrowseProtocols cups
BrowseOrder Deny,Allow
BrowseAllow from @LOCAL
BrowseAddress 255.255.255.255
Listen *:631
```

To enable web-based CUPS administration and to share printers with others on the network, the `cupsd` daemon can be set to listen on port 631 for all network interfaces to your computer based on this entry: `Listen *:631`. By default, it listens on the local interface only on many Linux systems (`Listen localhost:631`). For Fedora, CUPS listens on all interfaces by default.

This is a good way to enable users on several connected LANs to discover and use printers on other nearby LANs.

You can allow or deny access to different features of the CUPS server. An access definition for a CUPS printer (created from the Print Settings window) might appear as follows:

```
<Location /printers/ns1-hp1>
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
AuthType None
</Location>
```

Here, printing to the `ns1-hp1` printer is allowed only for users on the local host (`127.0.0.1`). No password is needed (`AuthType None`). To allow access to the administration tool, CUPS must be configured to prompt for a password (`AuthType Basic`).

Starting the CUPS server

For Linux systems that use System V–style startup scripts (such as earlier releases of Fedora and RHEL), starting and shutting down the CUPS print service is pretty easy. Use the `chkconfig` command to turn on CUPS so that it starts at each reboot. Run the `cups` startup script to have the CUPS service start immediately. In RHEL 6.x or earlier, type the following as root user:

```
# chkconfig cups on
# service cups start
```

If the CUPS service was already running, you should use `restart` instead of `start`. Using the `restart` option is also a good way to reread any configuration options that you may have changed in the `cupsd.conf` file (although, if CUPS is already running, `service cups reload` rereads configuration files without restarting).

In Fedora 30 and RHEL 8, you use the `systemctl` command instead of `service` to start and stop services:

```
# systemctl status cups.service

* cups.service - CUPS Printing Service
   Loaded: loaded (/usr/lib/systemd/system/cups.service; enabled)
   Active: active (running) since Sat 2016-07-23 22:41:05 EDT; 18h ago
 Main PID: 20483 (cupsd)
   Status: "Scheduler is running..."
   CGroup: /system.slice/cups.service
           ├─20483 /usr/sbin/cupsd -f
```

You can tell the CUPS service is running because the status shows the `cupsd` daemon active with PID 20483. If that service were not running, you could start the CUPS service as follows:

```
# systemctl start cups.service
```

See Chapter 15, "Starting and Stopping Services," for more information on the `systemctl` and `service` commands for working with services.

Configuring CUPS printer options manually

If your Linux distribution doesn't have a graphical means of configuring CUPS, you can edit configuration files directly. For example, when a new printer is created from the Print Settings window, it is defined in the `/etc/cups/printers.conf` file. This is what a printer entry looks like:

```
<DefaultPrinter printer>
Info HP LaserJet 2100M
Location HP LaserJet 2100M in hall closet
DeviceURI parallel:/dev/lp0
State Idle
```

```
Accepting Yes
Shared No
JobSheets none none
QuotaPeriod 0
PageLimit 0
KLimit 0
</Printer>
```

This is an example of a local printer that serves as the default printer for the local system. The `Shared No` value is set because the printer is currently available only on the local system. The most interesting information relates to `DeviceURI`, which shows that the printer is connected to parallel port `/dev/lp0`. The state is `Idle` (ready to accept printer jobs), and the `Accepting` value is `Yes` (the printer is accepting print jobs by default).

The `DeviceURI` has several ways to identify the device name of a printer, reflecting where the printer is connected. Here are some examples listed in the `printers.conf` file:

```
DeviceURI parallel:/dev/plp
DeviceURI serial:/dev/ttyd1?baud=38400+size=8+parity=none+flow=soft
DeviceURI scsi:/dev/scsi/sc1d6l0
DeviceURI usb://hostname:port
DeviceURI socket://hostname:port
DeviceURI tftp://hostname/path
DeviceURI ftp://hostname/path
DeviceURI http://hostname[:port]/path
DeviceURI ipp://hostname/path
DeviceURI smb://hostname/printer
```

The first four examples show the form for local printers (`parallel`, `serial`, `scsi`, and `usb`). The other examples are for remote hosts. In each case, *`hostname`* can be the host's name or IP address. Port numbers or paths identify the locations of each printer on the host. For example, *`hostname`* could be `myhost.example.com:631` and *`path`* could be replaced by any name you like, such as `printers/myprinter`.

Using Printing Commands

To remain backward compatible with older UNIX and Linux printing facilities, CUPS supports many of the old commands for working with printing. Most command-line printing with CUPS can be performed with the `lp` command. Word processing applications such as LibreOffice, OpenOffice, and AbiWord are set up to use this facility for printing.

You can use the Print Settings window to define the filters needed for each printer so that the text can be formatted properly. Options to the `lp` command can add filters to process the text properly. Other commands for managing printed documents include `lpq` (for viewing the contents of print queues), `lprm` (for removing print jobs from the queue), and `lpstat -t` (for controlling printers).

Printing with lp

You can use the `lp` command to print documents to both local and remote printers (provided the printers are configured locally). Document files can be either added to the end of the `lp` command line or directed to the `lp` command using a pipe (|). Here's an example of a simple `lp` command:

```
$ lp doc1.ps
```

When you specify just a document file with `lp`, output is directed to the default printer. As an individual user, you can change the default printer by setting the value of the `PRINTER` variable. Typically, you add the `PRINTER` variable to one of your startup files, such as `$HOME/.bashrc`. Adding the following line to your `.bashrc` file, for example, sets your default printer to `lp3`:

```
export PRINTER=lp3
```

To override the default printer, specify a particular printer on the `lp` command line. The following example uses the `-P` option to select a different printer:

```
$ lp -P canyonps doc1.ps
```

The `lp` command has a variety of options that enable `lp` to interpret and format several different types of documents. These include `-#` *num*, where *num* is replaced by the number of copies to print (from 1 to 100) and `-l` (which causes a document to be sent in raw mode, presuming that the document has already been formatted). To learn more options to `lp`, type **man lp**.

Listing status with lpstat -t

Use the `lpstat -t` command to list the status of your printers. Here is an example:

```
$ /usr/sbin/lpstat -t
printer hp disabled since Wed 10 Jul 2019 10:53:34 AM EDT
printer deskjet-555 is idle.  enabled since Wed 10 Jul 2019
10:53:34 AM EDT
```

This output shows two active printers. The `hp` printer is currently disabled (offline). The `deskjet-555` printer is enabled.

Removing print jobs with lprm

Users can remove their own print jobs from the queue with the `lprm` command. Used alone on the command line, `lprm` removes all of the user's print jobs from the default printer. To remove jobs from a specific printer, use the `-P` option, as follows:

```
$ lprm -P lp0
```

To remove all print jobs for the current user, type the following:

```
$ lprm -
```

The root user can remove all of the print jobs for a specific user by indicating that user on the `lprm` command line. For example, to remove all print jobs for the user named `mike`, the root user types the following:

```
# lprm -U mike
```

To remove an individual print job from the queue, indicate its job number on the `lprm` command line. To find the job number, type the `lpq` command. Here's what the output of that command may look like:

```
# lpq
printer is ready and printing
Rank    Owner                Job Files                 Total Size Time
active root                  133 /home/jake/pr1          467
2       root                 197 /home/jake/mydoc      23948
```

The output shows two printable jobs waiting in the queue. (The printer is ready and printing the job listed as active.) Under the `Job` column, you can see the job number associated with each document. To remove the first print job, type the following:

```
# lprm 133
```

Configuring Print Servers

You've configured a printer so that you and the other users on your computer can print to it. Now you want to share that printer with other people in your home, school, or office. Basically, that means configuring the printer as a print server.

The printers configured on your Linux system can be shared in different ways with other computers on your network. Not only can your computer act as a Linux print server (by configuring CUPS), but it can also appear as an SMB (Windows) print server to client computers. After a local printer is attached to your Linux system and your computer is connected to your local network, you can use the procedures in the following sections to share the printer with client computers using a Linux (UNIX) or SMB interface.

Configuring a shared CUPS printer

Making the local printer added to your Linux computer available to other computers on your network is fairly easy. If a TCP/IP network connection exists between the computers sharing the printer, you simply grant permission to all hosts, individual hosts, or users from remote hosts to access your computer's printing service.

To configure a printer entry manually in the `/etc/cups/printers.conf` file to accept print jobs from all other computers, make sure that the `Shared Yes` line is set. The following example from a `printers.conf` entry earlier in this chapter demonstrates what the new entry would look like:

```
<DefaultPrinter printer>
Info HP LaserJet 2100M
Location HP LaserJet 2100M in hall closet
```

```
DeviceURI parallel:/dev/lp0
State Idle
Accepting Yes
Shared Yes
JobSheets none none
QuotaPeriod 0
PageLimit 0
KLimit 0
</Printer>
```

On Linux systems that use the Print Settings window described earlier in this chapter, it's best to set up your printer as a shared printer using that window. Here's how to do that using Fedora 30:

1. **From the Activities screen on a GNOME 3 desktop in Fedora, type** Print Settings **and press Enter.** The Print Settings window appears.
2. **To allow all of your printers to be shared, select Server ➪ Settings.** If you are not the root user, you are prompted for the root password. The Basic Server Settings pop-up appears.
3. **Select the check box next to "Publish shared printers connected to this system" and click OK.** You may be asked to modify your firewall to open the necessary ports for remote systems to access your printers.
4. **To allow or restrict printing for a particular printer further, double-click the name of the printer that you want to share.** (If the printer is not yet configured, refer to the section "Setting Up Printers" earlier in this chapter.)
5. **Choose the Policies heading and select Shared so that a check mark appears in the box.**
6. **If you want to restrict access to the printer to selected users, select the Access Control heading and choose one of the following options:**
    a. **Allow Printing for Everyone Except These Users.** With this selected, all users are allowed access to the printer. By typing usernames into the Users box and clicking Add, you exclude selected users.
    b. **Deny Printing for Everyone Except These Users.** With this selected, all users are excluded from using the printer. Type usernames into the Users box, and click Add to allow access to the printer for only those names that you enter.

Now you can configure other computers to use your printer, as described in the section "Setting Up Printers" of this chapter. If you try to print from another computer and it doesn't work, try these troubleshooting tips:

Open your firewall. If you have a restrictive firewall, it may not permit printing. You must enable access to TCP port 631 to allow access to printing on your computer.

Check names and addresses. Make sure that you entered your computer's name and print queue properly when you configured it on the other computer. Try using the IP

address instead of the hostname. (If that works, it indicates a DNS name resolution problem.) Running a tool such as `tcpdump` enables you to see where the transaction fails.

Check which addresses `cupsd` is listening on. The `cupsd` daemon must be listening outside of the localhost for remote systems to print to it. Use the `netstat` command (as the root user) as follows to check this. The first example shows `cupsd` only listening on local host (`127.0.0.1:631`); the second shows `cupsd` listening on all network interfaces (`0 0.0.0.0:631`):

```
# netstat -tupln | grep 631
tcp        0      0 127.0.0.1:631     0.0.0.0:*         LISTEN
6492/cupsd
# netstat -tupln | grep 631
tcp        0      0 0.0.0.0:631       0.0.0.0:*         LISTEN
6492/cupsd
```

Access changes to your shared printer are made in the `cupsd.conf` and `printers.conf` files in your `/etc/cups` directory.

Configuring a shared Samba printer

Your Linux printers can be configured as shared SMB printers so that they appear to be available from Windows systems. To share your printer as if it were a Samba (SMB) printer, simply configure basic Samba server settings as described in Chapter 19, "Configuring a Windows File Sharing (Samba) Server." All your printers should be shared on your local network by default. The next section shows what the resulting settings look like and how you might want to change them.

Understanding `smb`.conf for printing

When you configure Samba, the `/etc/samba/smb.conf` file is constructed to enable all of your configured printers to be shared. Here are a few lines from the `smb.conf` file that relate to printer sharing:

```
[global]
     ...
  load printers = yes
  cups options = raw
  printcap name = /etc/printcap
  printing = cups
    ...
[printers]
        comment = All Printers
        path = /var/spool/samba
        browseable = yes
        writeable = no
        printable = yes
```

You can read the comment lines to learn more about the file's contents. Lines beginning with a semicolon (;) indicate the default setting for the option on a comment line. Remove the semicolon to change the setting.

The selected lines show that printers from `/etc/printcap` were loaded and that the CUPS service is being used. With `cups options` set to `raw`, Samba assumes that print files have already been formatted by the time they reach your print server. This allows the Linux or Windows clients to provide their own print drivers.

The last few lines are the actual printers' definition. By changing the `browseable` option from `no` to `yes`, you give users the ability to print to all printers (`printable = yes`). You can also store Windows native print drivers on your Samba server. When a Windows client uses your printer, the driver automatically becomes available. You do not need to download a driver from the vendor's website. To enable the printer driver share, add a Samba share called `print$` that looks like the following:

```
[print$]
comment = Printer Drivers
path = /var/lib/samba/drivers
browseable = yes
guest ok = no
read only = yes
write list = chris, dduffey
```

After you have the share available, you can start copying Windows print drivers to the `/var/lib/samba/drivers` directory.

Setting up SMB clients

Chances are good that if you are configuring a Samba printer on your Linux computer, you want to share it with Windows clients. If Samba is set up properly on your computer and the client computers can reach you over the network, users should have no trouble finding and using your printer.

For many Windows 10 systems, click Start ➪ Printers and Scanners and select the printer from the list to configure it.

With Windows Vista, you open the Network icon. The name of your host computer (the NetBIOS name, which is probably also your TCP/IP name) appears on the screen or within a workgroup folder on the screen. Open the icon that represents your computer. The window that opens shows your shared printers and folders.

Tip

If your computer's icon doesn't appear in Network Neighborhood or My Network Places, try using the Search window. From Windows XP, choose Start ➪ Search ➪ Computer or People ➪ A Computer on the Network. Type your computer's name into the Computer Name box and click Search. Double-click your computer in the Search window results panel. A window displaying the shared printers and folders from your computer appears.

After your shared printer appears in the window, configure a pointer to that printer by opening (double-clicking) the printer icon. A message tells you that you must set up the printer before you can use it. Click Yes to proceed to configure the printer for local use. The Add Printer Wizard appears. Answer the questions that ask you how you intend to use the printer and add the appropriate drivers. When you are finished, the printer appears in your printer window.

Another way to configure an SMB printer from a Windows XP operating system is to go to Start ➪ Printers and Faxes. In the Printers and Faxes window that appears, click the Add a Printer icon in the upper-left portion of the window, and select Network Printer from the first window. From there, you can browse and/or configure your SMB printer.

Summary

Providing networked printing services is essential on today's business networks. With the use of a few network-attached devices, you can focus your printer spending on a few high-quality devices that multiple users can share instead of numerous lower-cost devices. In addition, a centrally located printer can make it easier to maintain the printer while still enabling everyone to get their printing jobs done.

The default printing service in nearly every major Linux distribution today is the Common UNIX Printing System (CUPS). Any Linux system that includes CUPS offers the CUPS web-based administrative interface for configuring CUPS printing. It also offers configuration files in the `/etc/cups` directory for configuring printers and the CUPS service (`cupsd` daemon).

In RHEL, Fedora, Ubuntu, and other Linux systems, you can configure your printer with the printing configuration windows available in both KDE and GNOME desktops. A variety of drivers makes it possible to print to different kinds of printers as well as to printers that are connected to computers on the network.

You can set up your computer as a Linux print server, and you can also have your computer emulate an SMB (Windows) print server. After your network is configured properly and a local printer is installed, sharing that printer over the network as a UNIX or SMB print server is not very complicated.

Exercises

Use these exercises to test your knowledge of configuring printers in Linux. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, you can often complete a task in multiple ways).

1. Use the Print Settings window (`system-config-printer` package) to add a new printer called `myprinter` to your system. (The printer does not have to be connected to set up a print queue for the new printer.) Make it a generic PostScript printer connected to a local serial, LPT, or other port.
2. Use the `lpstat -t` command to see the status of all of your printers.
3. Use the `lp` command to print the `/etc/hosts` file to that printer.
4. Check the print queue for that printer to see that the print job is there.
5. Remove the print job from the queue (cancel it).
6. Using the Printing window, set the basic server setting that publishes your printers so that other systems on your local network can print to your printers.
7. Allow remote administration of your system from a web browser.
8. Demonstrate that you can do remote administration of your system by opening a web browser to port 631 from another system to the Linux system running your print server.
9. Use the `netstat` command to see on which addresses the `cupsd` daemon is listening (the printing port is 631).
10. Delete the `myprinter` printer entry from your system.

CHAPTER 17

Configuring a Web Server

IN THIS CHAPTER

- Installing an Apache web server
- Configuring Apache
- Securing Apache with iptables and SELinux
- Creating virtual hosts
- Building a secure (HTTPS) website
- Checking Apache for errors

Web servers are responsible for serving up the content you view on the Internet every day. By far, the most popular web server is the Apache (HTTPD) web server, which is sponsored by the Apache Software Foundation (`http://apache.org`). Because Apache is an open source project, it is available with every major Linux distribution, including Fedora, RHEL, and Ubuntu.

You can configure a basic web server to run in Linux in just a few minutes. However, you can configure your Apache web server in a tremendous number of ways. You can configure an Apache web server to serve content for multiple domains (virtual hosting), provide encrypted communications (HTTPS), and secure some or all of a website using different kinds of authentication.

This chapter takes you through the steps to install and configure an Apache web server. These steps include procedures for securing your server as well as using a variety of modules so that you can incorporate different authentication methods and scripting languages into your web server. Then I describe how to generate certificates to create an HTTPS Secure Sockets Layer (SSL) website.

Understanding the Apache Web Server

Apache HTTPD (also known as the *Apache HTTPD Server*) provides the service with which the client web browsers communicate. The daemon process (`httpd`) runs in the background on your server and waits for requests from web clients. Web browsers provide those connections to the HTTP daemon and send requests, which the daemon interprets, sending back the appropriate data (such as a web page or other content).

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Apache HTTPD includes an interface that allows modules to tie into the process to handle specific portions of a request. Among other things, modules are available to handle the processing of scripting languages, such as Perl or PHP, within web documents and to add encryption to connections between clients and the server.

Apache began as a collection of patches and improvements from the National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign, to the HTTP daemon. The NCSA HTTP daemon was the most popular HTTP server at the time, but it had started to show its age after its author, Rob McCool, left NCSA in mid-1994.

NOTE
Another project that came from NCSA is Mosaic. Most modern web browsers can trace their origins to Mosaic.

In early 1995, a group of developers formed the Apache Group and began making extensive modifications to the NCSA HTTPD code base. Apache soon replaced NCSA HTTPD as the most popular web server, a title it still holds today.

The Apache Group later formed the Apache Software Foundation (ASF) to promote the development of Apache and other free software. With the start of new projects at ASF, the Apache server became known as Apache HTTPD, although the two terms are still used interchangeably. Currently, ASF has more than 350 open source initiatives, including Tomcat (which includes open source Java Servlet and JavaServer Pages technologies), Hadoop (a project providing highly available, distributed computing), and SpamAssassin (an email filtering program).

Getting and Installing Your Web Server

Although Apache is available with every major Linux distribution, it is often packaged in different ways. In most cases, all you need to start a simple Apache web server is the package containing the Apache daemon itself (`/usr/sbin/httpd`) and its related files. In Fedora, RHEL, and others, the Apache web server comes in the `httpd` package.

Understanding the httpd package

To examine the `httpd` package in Fedora or RHEL before you install it, download the package using the `yumdownloader` command and run a few `rpm` commands on it to view its contents:

```
# yumdownloader httpd
# rpm -qpi httpd-*rpm
Name        : httpd
Version     : 2.4.41
Release     : 1.fc30
Architecture: x86_64
Install Date: (not installed)
```

```
Group       : Unspecified
Size        : 5070831
License     : ASL 2.0
Signature   : RSA/SHA256, Mon 19 Aug 2019 06:06:09 AM EDT, Key ID
ef3c111fcfc659b9
Source RPM  : httpd-2.4.41-1.fc30.src.rpm
Build Date  : Thu 15 Aug 2019 06:07:29 PM EDT
Build Host  : buildvm-30.phx2.fedoraproject.org
Relocations : (not relocatable)
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : http://httpd.apache.org/
Bug URL     : https://bugz.fedoraproject.org/httpd
Summary     : Apache HTTP Server
Description :
The Apache HTTP Server is a powerful, efficient, and extensible
web server.
```

The `yumdownloader` command downloads the latest version of the `httpd` package to the current directory. The `rpm -qpi` command queries the `httpd` RPM package you just downloaded for information. You can see that the package was created by the Fedora project and that it is indeed the Apache HTTP Server package. Next, look inside the package to see the configuration files:

```
# rpm -qpc httpd-*rpm
/etc/httpd/conf.d/autoindex.conf
/etc/httpd/conf.d/userdir.conf
/etc/httpd/conf.d/welcome.conf
/etc/httpd/conf.modules.d/00-base.conf
/etc/httpd/conf.modules.d/00-dav.conf
...
/etc/httpd/conf/httpd.conf
/etc/httpd/conf/magic
/etc/logrotate.d/httpd
/etc/sysconfig/htcacheclean
```

The main configuration file is `/etc/httpd/conf/httpd.conf` for Apache. The `welcome.conf` file defines the default home page for your website, until you add some content. The `magic` file defines rules that the server can use to figure out a file's type when the server tries to open it.

The `/etc/logrotate.d/httpd` file defines how log files produced by Apache are rotated. The `/usr/lib/tmpfiles.d/httpd.conf` file defines a directory that contains temporary runtime files (no need to change that file).

Some Apache modules drop configuration files (`*.conf`) into the `/etc/httpd/conf.modules.d/` directory. Any file in that directory that ends in `.conf` is pulled into the main `httpd.conf` file and used to configure Apache. Most module packages that come with configuration files put those configuration files in the `/etc/httpd/conf.d` directory. For

example, the `mod_ssl` (for secure web servers) and `mod_python` (for interpreting python code) modules have related configuration files in the `/etc/httpd/conf.d` directory named `ssl.conf` and `python.conf`, respectively.

You can just install the `httpd` package to begin setting up your web server. However, you might prefer to add some other packages that are often associated with the `httpd` package. One way to do that is to install the entire Web Server (in Fedora) or Basic Web Server group (in RHEL), as in the following example:

```
# yum groupinstall "Web Server"
```

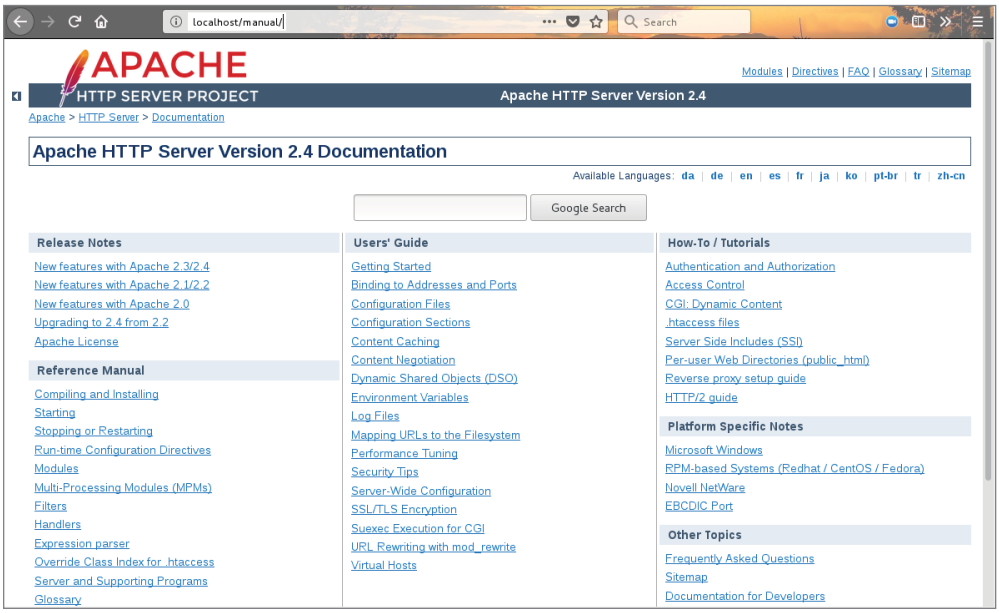
Besides installing some packages that are peripheral to `httpd` (such as `rsyslogd`, `irqbalance`, and others), here are other packages in the Web Server group in Fedora that you get by default along with `httpd`:

`httpd-manual` Fills the `/var/www/manual` directory with the Apache documentation manuals. After you start the httpd service (as shown in later steps), you can access this set of manuals from a web browser on the local machine by typing **`http://localhost/manual`** into the location box.

Externally, instead of localhost, you could use the fully qualified domain name or IP address of the system. The Apache Documentation screen then appears, as shown in Figure 17.1.

FIGURE 17.1

Access Apache documentation directly from the local Apache server.



mod_ssl Contains the module and configuration file needed for the web server to provide secure connections to clients using Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. These features are necessary if you need encrypted communications for online shopping or other data that you want to keep private. The configuration file is located at `/etc/httpd/conf.d/ssl.conf`.

crypto-utils Contains commands for generating keys and certificates needed to do secure communications with the Apache web server.

mod_perl Contains the Perl module (`mod_perl`), configuration file, and associated files needed to allow the Apache web server to execute any Perl code directly.

php Contains the PHP module and configuration file needed to run PHP scripts directly in Apache. Related packages include `php-ldap` (for running PHP code that needs to access LDAP databases) and `php-mysql` (to add database support to the Apache server).

php-ldap Adds support for Lightweight Directory Access Protocol (LDAP) to the PHP module, allowing directory service access over networks.

squid Provides proxy services for specific protocols (such as HTTP), as mentioned in Chapter 14, "Administering Networking." Although it doesn't provide HTTP content itself, a Squid proxy server typically forwards requests from proxy clients to the Internet or other network providing web content. This provides a means of controlling or filtering content that clients can reach from a home, school, or place of business.

webalizer Contains tools for analyzing web server data.

Optional packages in the Web Server group come from the web-server sub-group. Run `yum groupinfo web-server` to display those packages. Some of those packages offer special ways of providing content, such as wikis (`moin`), content management systems (`drupal7`), and blogs (`wordpress`). Others include tools for graphing web statistics (`awstats`) or offer lightweight web server alternatives to Apache (`lighttpd` and `cherokee`).

Installing Apache

Although you only need `httpd` to get started with an Apache web server, if you are just learning about Apache, you should install the manuals (`httpd-manual`) as well. If you are thinking of creating a secure (SSL) site and possibly generating some statistics about your website, you can just install the entire group in Fedora 30:

```
# yum groupinstall "Web Server"
```

Assuming that you have an Internet connection to the Fedora repository (or RHEL repository, if you are using RHEL), all of the mandatory and default packages from that group are installed. You have all of the software that you need to do the procedures and exercises described in this chapter.

Starting Apache

To get the Apache web server going, you want to enable the service to start on every reboot, and you want to start it immediately. In Red Hat Enterprise Linux (up to RHEL 6) and in older Fedora distributions, you could type the following as root:

```
# chkconfig httpd on
# service httpd start
Starting httpd:                [  OK  ]
```

In Fedora 30 and RHEL 8 systems, you enable and start `httpd` using the `systemctl` command:

```
# systemctl enable httpd.service
# systemctl start httpd.service
# systemctl status httpd.service
• httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled;
    vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
           └─php-fpm.conf
   Active: active (running) since Mon 2019-09-02 16:16:56 EDT;
    21min ago
     Docs: man:httpd.service(8)
 Main PID: 11773 (/usr/sbin/httpd)
   Status: "Total requests: 14; Idle/Busy workers 100/0;Requests/sec:
    0.0111; Bytes served/s>
    Tasks: 214 (limit: 2294)
   Memory: 24.6M
   CGroup: /system.slice/httpd.service
           ├─11773 /usr/sbin/httpd -DFOREGROUND
           ├─11774 /usr/sbin/httpd -DFOREGROUND
           ├─11775 /usr/sbin/httpd -DFOREGROUND
           ├─11776 /usr/sbin/httpd -DFOREGROUND
           ├─11777 /usr/sbin/httpd -DFOREGROUND
           └─11778 /usr/sbin/httpd -DFOREGROUND
    ...
```

When the `httpd` service starts, five or six `httpd` daemon processes are launched by default (depending on your Linux system) to respond to requests for the web server. You can configure more or fewer `httpd` daemons to be started based on settings in the `httpd.conf` file (described in the section "Understanding the Apache configuration files" later in this chapter).

To change the behavior of the `httpd` daemon, you can edit the `httpd` service by running `systemctl edit httpd`.

Because there are different versions of `httpd` around, check the man page (`man httpd`) to see what options can be passed to the `httpd` daemon. For example, run `systemctl edit httpd` and add an entry as follows:

```
[Service]
Environment=OPTIONS='-e debug'
```

Save the changes (Ctrl+O, Ctrl+X). Adding `-e debug` increases the log level so that the maximum number of Apache messages are sent to log files. Restart the `httpd` service for the changes to take effect. Type the `ps` command to make sure that the options took effect:

```
$ ps -ef | grep httpd
root   14575 1     0 08:49 ? 00:00:01 /usr/sbin/httpd -e debug
-DFOREGROUND
apache 14582 14575 0 08:49 ? 00:00:00 /usr/sbin/httpd -e debug
-DFOREGROUND
```

If you added a debug option (`-e debug`), remember to remove that option by running `systemctl edit httpd` again and removing the entry when you are done debugging Apache, and restart the service. Leaving debugging on will quickly fill up your log files.

Securing Apache

To secure Apache, you need to be aware of standard Linux security features (permissions, ownership, firewalls, and Security Enhanced Linux) as well as security features that are specific to Apache. The following sections describe security features that relate to Apache.

Apache file permissions and ownership

The `httpd` daemon process runs as the user `apache` and group `apache`. By default, HTML content is stored in the `/var/www/html` directory (as determined by the value of `DocumentRoot` in the `httpd.conf` file).

For the `httpd` daemon to be able to access that content, standard Linux permissions apply: If read permission is not on for "other" users, it must be on for the `apache` user or group for the files to be read and served to clients. Likewise, any directory the `httpd` daemon must traverse to get to the content must have execute permission on for the `apache` user, `apache` group, or other user.

Although you cannot log in as the `apache` user (`/sbin/nologin` is the default shell), you can create content as root and change its ownership (`chown` command) or permission (`chmod` command). Often, however, separate user or group accounts are added to create content that is readable by everyone (other) but only writable by that special user or group.

Apache and firewalls

If you have locked down your firewall in Linux, you need to open several ports for clients to be able to talk to Apache through the firewall. Standard web service (HTTP) is accessible over TCP port 80; secure web service (HTTPS) is accessible via TCP port 443. (Port 443 only appears if you have installed the `mod_ssl` package, as described later.)

To verify which ports are being used by the `httpd` server, use the `netstat` command:

```
# netstat -tupln | grep httpd
tcp6   0      0 :::80           :::*                LISTEN      29169/httpd
tcp6   0      0 :::443          :::*                LISTEN      29169/httpd
```

The output shows that the `httpd` daemon (process ID 29169) is listening on all addresses for port 80 (`:::80`) and port 443 (`:::443`). Both ports are associated with the TCP protocol (`tcp6`). To open those ports in Fedora or Red Hat Enterprise Linux, you need to add some firewall rules.

On a current Fedora 30 or RHEL 7 or 8 system, open the Firewall window (type **Firewall** and press Enter from the Activities screen on the GNOME 3 desktop). From there, select Permanent as the configuration. Then, with the public zone selected, click the check boxes next to the http and https service boxes. Those ports immediately become open.

For RHEL 6 or older Fedora releases, add rules to the `/etc/sysconfig/iptables` file (somewhere before a final `DROP` or `REJECT`) such as the following:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
```

Restart `iptables` (`service iptables restart`) for the new rules to take effect.

Apache and SELinux

If *Security Enhanced Linux (SELinux)* is set to enforcing (as it is by default in Fedora and Red Hat Enterprise Linux), SELinux adds another layer of security over your `httpd` service. In essence, SELinux actually sets out to protect the system from being damaged by someone who may have cracked the `httpd` daemon. SELinux does this by creating policies that do the following:

- Deny access to files that are not set to the right file contexts. For `httpd` in SELinux, there are different file contexts for content, configuration files, log files, scripts, and other `httpd`-related files. Any file that is not set to the proper context is not accessible to the `httpd` daemon.
- Prevent insecure features from being used, such as file uploading and clear-text authentication, by setting Booleans for such features to the off position. You can selectively turn on Booleans as they are needed—if they meet your security requirements.
- Keep the `httpd` daemon from accessing nonstandard features, such as a port outside of the default ports the service would expect to use.

A full description of SELinux is contained in Chapter 24, "Enhancing Linux Security with SELinux." However, here are a few specifics you should know about using SELinux with the Apache `httpd` service:

Turn off SELinux You don't have to use SELinux. You can set SELinux to permissive mode if you feel that it is too difficult and unnecessary to create the SELinux policies needed to get your web server to work with SELinux in enforcing mode. You can change the mode to permissive by editing the `/etc/sysconfig/selinux` file so that the `SELINUX` value is set as follows. With this set, the next time you reboot the system, it is in permissive mode. This means that if you break SELinux policies, that event is logged but not prevented (as it would be in enforcing mode).

```
SELINUX=permissive
```

Read the httpd_selinux man page Type **man httpd_selinux** from the shell. This man page shows you the proper file contexts and available Booleans. (If the man page is not there, install it with `yum install selinux-policy-doc`.)

Use standard locations for files When you create new files, those files inherit the file contexts of the directories in which they are stored. Because `/etc/httpd` is set to the right file context for configuration files, `/var/www/html` is right for content files, and so on. Simply copying files to or creating new files in those locations causes the file contexts to be set properly.

Modify SELinux to allow non-standard features You may want to serve web content from the `/mystuff` directory or put configuration files in the `/etc/whatever` directory. Likewise, you may want to allow users of your server to upload files, run scripts, or enable other features that are disabled by SELinux by default. In those cases, you can use SELinux commands to set the file contexts and Booleans that you need to get SELinux working the way you want.

Be sure to read Chapter 24, "Enhancing Linux Security with SELinux," to learn more about SELinux.

Understanding the Apache configuration files

The configuration files for Apache HTTPD are incredibly flexible, meaning that you can configure the server to behave in almost any manner you want. This flexibility comes at the cost of increased complexity in the form of a large number of configuration options (called *directives*). In practice, however, you need to be familiar with only a few directives in most cases.

NOTE

See `http://httpd.apache.org/docs/current/mod/directives.html` for a complete list of directives supported by Apache. If you have `httpd-manual` installed, you can reach descriptions of these directives and other Apache features by opening the manual from the server you have running Apache: `http://localhost/manual/`.

In Fedora and RHEL, the basic Apache server's primary configuration file is in `/etc/httpd/conf/httpd.conf`. Besides this file, any file ending in `.conf` in the `/etc/httpd/conf.d` directory is also used for Apache configuration (based on an `Include` line in the `httpd.conf` file). In Ubuntu, the Apache configuration is stored in text files read by the Apache server, beginning with `/etc/apache2/apache2.conf`. Configuration is read from start to finish, with most directives being processed in the order in which they are read.

Using directives

The scope of many configuration directives can be altered based on context. In other words, some parameters may be set on a global level and then changed for a specific file, directory, or virtual host. Other directives are always global in nature, such as those specifying on which IP addresses the server listens. Still others are valid only when applied to a specific location.

Locations are configured in the form of a start tag containing the location type and a resource location, followed by the configuration options for that location, and finishing with an end tag. This form is often called a *configuration block*, and it looks very similar to HTML code. A special type of configuration block, known as a *location block*, is used to limit the scope of directives to specific files or directories. These blocks take the following form:

```
<locationtag specifier>
(options specific to objects matching the specifier go within this
block)
</locationtag>
```

Different types of location tags exist and are selected based on the type of resource location that is being specified. The specifier included in the start tag is handled based on the type of location tag. The location tags that you generally use and encounter are `Directory`, `Files`, and `Location`, which limit the scope of the directives to specific directories, files, or locations, respectively.

- `Directory` tags are used to specify a path based on the location on the filesystem. For instance, `<Directory />` refers to the root directory on the computer. Directories inherit settings from directories above them, with the most specific `Directory` block overriding less-specific ones, regardless of the order in which they appear in the configuration files.
- `Files` tags are used to specify files by name. `Files` tags can be contained within a `Directory` block to limit them to files under that directory. Settings within a `Files` block override the ones in `Directory` blocks.
- `Location` tags are used to specify the URI used to access a file or directory. This is different from `Directory` in that it relates to the address contained within the request and not to the real location of the file on the drive. `Location` tags are processed last and override the settings in `Directory` and `Files` blocks.

Match versions of these tags—`DirectoryMatch`, `FilesMatch`, and `LocationMatch`—have the same function but can contain regular expressions in the resource specification. `FilesMatch` and `LocationMatch` blocks are processed at the same time as `Files` and `Location`, respectively. `DirectoryMatch` blocks are processed after `Directory` blocks.

Apache can also be configured to process configuration options contained within files with the name specified in the `AccessFileName` directive (which is generally set to `.htaccess`). Directives in access configuration files are applied to all objects under the directory they contain, including subdirectories and their contents. Access configuration files are processed at the same time as `Directory` blocks, using a similar "most specific match" order.

Note

Access control files are useful for allowing users to change specific settings without having access to the server configuration files. The configuration directives permitted within an access configuration file are determined by the `AllowOverride` setting on the directory in which they are contained. Some directives do not make sense at that level and generally result in a "server internal error" message when trying to access the URI. The `AllowOverride` option is covered in detail at `http://httpd.apache.org/docs/mod/core.html#allowoverride`.

Three directives commonly found in location blocks and access control files are `DirectoryIndex`, `Options`, and `ErrorDocument`:

- `DirectoryIndex` tells Apache which file to load when the URI contains a directory but not a filename. This directive doesn't work in `Files` blocks.
- `Options` is used to adjust how Apache handles files within a directory. The `ExecCGI` option tells Apache that files in that directory can be run as CGI scripts, and the `Includes` option tells Apache that server-side includes (SSIs) are permitted. Another common option is the `Indexes` option, which tells Apache to generate a list of files if one of the filenames found in the `DirectoryIndex` setting is missing. An absolute list of options can be specified, or the list of options can be modified by adding `+` or `-` in front of an option name. See `http://httpd.apache.org/docs/mod/core.html#options` for more information.
- `ErrorDocument` directives can be used to specify a file containing messages to send to web clients when a particular error occurs. The location of the file is relative to the `/var/www` directory. The directive must specify an error code and the full URI for the error document. Possible error codes include `403` (access denied), `404` (file not found), and `500` (server internal error). You can find more information about the `ErrorDocument` directive at `http://httpd.apache.org/docs/mod/core.html#errordocument`. As an example, when a client requests a URL from the server that is not found, the following `ErrorDocument` line causes the `404` error code to send the client an error message that is listed in the `/var/www/error/HTTP_NOT_FOUND.html.var` file.

```
ErrorDocument 404 /error/HTTP_NOT_FOUND.html.var
```

Another common use for location blocks and access control files is to limit or expand access to a resource. The `Allow` directive can be used to permit access to matching hosts, and the `Deny` directive can be used to forbid it. Both of these options can occur more than once within a block and are handled based on the `Order` setting. Setting `Order` to `Deny`, `Allow` permits access to any host that is not listed in a `Deny` directive. A setting of `Allow`, `Deny` denies access to any host not allowed in an `Allow` directive.

As with most other options, the most specific Allow or Deny option for a host is used, meaning that you can Deny access to a range and Allow access to subsets of that range. By adding the Satisfy option and some additional parameters, you can add password authentication. For more information on Allow or Deny, Satisfy, or other directives, refer to the Apache Directive Index: http://httpd.apache.org/docs/current/mod/directives.html.

Understanding default settings

The reason you can start using your Apache web server as soon as you install it is that the httpd.conf file includes default settings that tell the server where to find web content, scripts, log files, and other items that the server needs to operate. It also includes settings that tell the server how many server processes to run at a time and how directory contents are displayed.

If you want to host a single website (such as for the example.com domain), you can simply add content to the /var/www/html directory and add the address of your website to a DNS server so that others can browse to it. You can then change directives, such as those described in the previous section, as needed.

To help you understand the settings that come in the default httpd.conf file, I've displayed some of those settings with descriptions below. I have removed comments and rearranged some of the settings for clarity.

The following settings show locations where the httpd server is getting and putting content by default:

```
ServerRoot "/etc/httpd"
Include conf.d/*.conf
ErrorLog logs/error_log
CustomLog "logs/access_log" combined
DocumentRoot "/var/www/html"
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

The ServerRoot directive identifies /etc/httpd as the location where configuration files are stored.

At the point in the file where the Include line appears, any files ending in .conf from the /etc/httpd/conf.d directory are included in the httpd.conf file. Configuration files are often associated with Apache modules (and are often included in the software package with a module) or with virtual host blocks (which you might add yourself to virtual host configurations in separate files). See the section "Adding a virtual host to Apache" later in this chapter.

As errors are encountered and content is served, messages about those activities are placed in files indicated by the ErrorLog and CustomLog entries. From the entries shown here, those logs are stored in the /etc/httpd/logs/error_log and /etc/httpd/logs/access_log directories, respectively. Those logs are also hard linked to the /var/log/httpd directory, so you can access the same file from there as well.

The DocumentRoot and ScriptAlias directives determine where content that is served by your httpd server is stored. Traditionally, you would place an index.html file in the DocumentRoot directory (/var/www/html, by default) as the home page and add other content as needed. The ScriptAlias directive tells the httpd daemon that any scripts requested from the cgi-bin directory should be found in the /var/www/cgi-bin directory. For example, a client could access a script located in /var/www/cgi-bin/script.cgi by entering a URL such as http://example.com/cgi-bin/script.cgi.

In addition to file locations, you can find other information in the httpd.conf file. Here are some examples:

```
Listen 80
User apache
Group apache
ServerAdmin root@localhost
DirectoryIndex index.html index.php
AccessFileName .htaccess
```

The Listen 80 directive tells httpd to listen for incoming requests on port 80 (the default port for the HTTP web server protocol). By default, it listens on all network interfaces, although you could restrict it to selected interfaces by IP address (for example, Listen 192.168.0.1:80).

The User and Group directives tell httpd to run as apache for both the user and group. The value of ServerAdmin (root@localhost, by default) is published on some web pages to tell users where to email if they have problems with the server.

The DirectoryIndex lists files that httpd will serve if a directory is requested. For example, if a web browser requested *http://host/whatever/*, httpd would see whether /var/www/html/whatever/index.html existed and serve it if so. If it didn't exist, in this example, httpd would look for index.php. If that file couldn't be found, the contents of the directory would be displayed. An AccessFileName directive can be added to tell httpd to use the contents of the .htaccess file if it exists in a directory to read in settings that apply to access to that directory. For example, the file could be used to require password protection for the directory or to indicate that the contents of the directory should be displayed in certain ways. For this file to work, however, a Directory container (described next) would have to have AllowOverride opened. (By default, the AllowOverride None setting prevents the .htaccess file from being used for any directives.)

The following Directory containers define behavior when the root directory (/), /var/www, and /var/www/html directories are accessed:

```
<Directory/>
    AllowOverride none
    Require all denied
</Directory>
<Directory "/var/www">
    AllowOverride None
```

```
        # Allow open access:
        Require all granted
    </Directory>
    <Directory "/var/www/html">
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
```

The first `Directory` container (`/`) indicates that if `httpd` tries to access any files in the Linux filesystem, access is denied. The `AllowOverride none` directive prevents `.htaccess` files from overriding settings for that directory. Those settings apply to any subdirectories that are not defined in other `Directory` containers.

Content access is relaxed within the `/var/www` directory. Access is granted to content added under that directory, but overriding settings is not allowed.

The `/var/www/html Directory` container follows symbolic links and does not allow overrides. With `Require all granted` set, `httpd` doesn't prevent any access to the server.

If all of the settings just described work for you, you can begin adding the content that you want to the `/var/www/html` and `/var/www/cgi-bin html` directories. One reason you might not be satisfied with the default setting is that you might want to serve content for multiple domains (such as `example.com`, `example.org`, and `example.net`). To do that, you need to configure virtual hosts. Virtual hosts, which are described in greater detail in the next section, are a convenient (and almost essential) tool for serving different content to clients based on the server address or name to which a request is directed. Most global configuration options are applied to virtual hosts, but they can be overridden by directives within the `VirtualHost` block.

Adding a virtual host to Apache

Apache supports the creation of separate websites within a single server to keep content separate. Individual sites are configured on the same server in what are referred to as *virtual hosts*.

Virtual hosts are really just a way to have the content for multiple domain names available from the same Apache server. Instead of needing to have one physical system to serve content for each domain, you can serve content for multiple domains from the same operating system.

An Apache server that is doing virtual hosting may have multiple domain names that resolve to the IP address of the server. The content that is served to a web client is based on the name used to access the server.

For example, if a client got to the server by requesting the name `www.example.com`, the client would be directed to a virtual host container that had its `ServerName` set to

respond to www.example.com. The container would provide the location of the content and possibly different error logs or Directory directives from the global settings. This way, each virtual host could be managed as if it were on a separate machine.

To use name-based virtual hosting, add as many VirtualHost containers as you like. Here's how to configure a virtual host:

Note

After you enable your first VirtualHost, your default DocumentRoot (/var/www/html) is no longer used if someone accesses the server by IP address or some name that is not set in a VirtualHost container. Instead, the first VirtualHost container is used as the default location for the server.

1. **In Fedora or RHEL, create a file named /etc/httpd/conf.d/example.org.conf using this template:**

```
<VirtualHost *:80>
    ServerAdmin     webmaster@example.org
    ServerName      www.example.org
    ServerAlias     web.example.org
    DocumentRoot    /var/www/html/example.org/
DirectoryIndex  index.php index.html index.htm
</VirtualHost>
```

   This example includes the following settings:

   - The *:80 specification in the VirtualHost block indicates to what address and port this virtual host applies. With multiple IP addresses associated with your Linux system, the * can be replaced by a specific IP address. The port is optional for VirtualHost specifications but should always be used to prevent interference with SSL virtual hosts (which use port 443 by default).
   - The ServerName and ServerAlias lines tell Apache which names this virtual host should be recognized as, so replace them with names appropriate to your site. You can leave out the ServerAlias line if you do not have any alternate names for the server, and you can specify more than one name per ServerAlias line or have multiple ServerAlias lines if you have several alternate names.
   - The DocumentRoot specifies where the web documents (content served for this site) are stored. Although shown as a subdirectory that you create under the default DocumentRoot (/var/www/html), often sites are attached to the home directories of specific users (such as /home/chris/public_html) so that each site can be managed by a different user.

2. **With the host enabled, use `apachectl` to check the configuration, and then do a `graceful` restart:**

```
# apachectl configtest
Syntax OK
# apachectl graceful
```

Provided that you have registered the system with a DNS server, a web browser should be able to access this website using either `www.example.org` or `web.example.org`. If that works, you can start adding other virtual hosts to the system as well.

Another way to extend the use of your website is to allow multiple users to share their own content on your server. You can enable users to add content that they want to share via your web server in a subdirectory of their home directories, as described in the next section.

NOTE

Keeping individual virtual hosts in separate files is a convenient way to manage virtual hosts. However, you should be careful to keep your primary virtual host in a file that will be read before the others because the first virtual host receives requests for site names that don't match any in your configuration. In a commercial web-hosting environment, it is common to create a special default virtual host that contains an error message indicating that no site by that name has been configured.

Allowing users to publish their own web content

In situations where you do not have the ability to set up a virtual host for every user for whom you want to provide web space, you can easily make use of the `mod_userdir` module in Apache. With this module enabled (which it is not by default), the `public_html` directory under every user's home directory is available to the web at `http://`*`servername/~username/`*.

For example, a user named `wtucker` on `www.example.org` stores web content in `/home/wtucker/public_html`. That content would be available from `http://www.example.org/~wtucker`.

Make these changes to the `/etc/httpd/conf/httpd.conf` file to allow users to publish web content from their own home directories. Not all versions of Apache have these blocks in their `httpd.conf` file, so you might have to create them from scratch:

1. **Create a `<IfModule mod_userdir.c>` block.** Change `chris` to any username you want to allow users to create their own `public_html` directory. You can add multiple usernames.

```
<IfModule mod_userdir.c>
     UserDir enabled chris
     UserDir public_html
</IfModule>
```

2. **Create a `<Directory /home/*/public_html>` directive block and change any settings you like**. This is how the block will look:

```
<Directory "/home/*/public_html">
    Options Indexes Includes FollowSymLinks
    Require all granted
</Directory>
```

3. **Have your users create their own `public_html` directories in their own home directories**.

```
$ mkdir $HOME/public_html
```

4. **Set the execute permission (as root user) to allow the `httpd` daemon to access the home directory**:

```
# chmod +x /home /home/*
```

5. **If SELinux is in enforcing mode (which it is by default in Fedora and RHEL), a proper SELinux file context (`httpd_user_content_t`) should already be set on the following directories so that SELinux allows the `httpd` daemon to access the content automatically: `/home/*/www`, `/home/*/web`, and `/home/*/public_html`. If for some reason the context is not set, you can set it as follows:**

```
ttpd_user_content_t to /home/*/
# chcon -R --reference=/var/www/html/ /home/*/public_html
```

6. **Set the SELinux Boolean to allow users to share HTML content from their home directories**:

```
# setsebool -P httpd_enable_homedirs true
```

7. **Restart or reload the `httpd` service**.

   At this point, you should be able to access content placed in a user's `public_html` directory by pointing a web browser to `http://hostname/~user`.

Securing your web traffic with SSL/TLS

Any data that you share from your website using standard HTTP protocol is sent in clear text. This means that anyone who can watch the traffic on a network between your server and your client can view your unprotected data. To secure that information, you can add certificates to your site (so a client can validate who you are) and encrypt your data (so nobody can sniff your network and see your data).

Electronic commerce applications, such as online shopping and banking, should always be encrypted using either the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) specification. TLS is based on version 3.0 of the SSL specifications, so they are very similar in nature. Because of this similarity—and because SSL is older—the SSL acronym is often used to refer to either variety. For web connections, the SSL connection is established first, and then normal HTTP communication is "tunneled" through it.

Note

Because SSL negotiation takes place before any HTTP communication, name-based virtual hosting (which occurs at the HTTP layer) does not work easily with SSL. As a consequence, every SSL virtual host you configure should have a unique IP address. (See the Apache site for more information: `httpd.apache.org/docs/vhosts/name-based.html`.)

While you are establishing a connection between an SSL client and an SSL server, asymmetric (public key) cryptography is used to verify identities and establish the session parameters and the session key. A symmetric encryption algorithm is then used with the negotiated key to encrypt the data that are transmitted during the session. The use of asymmetric encryption during the handshaking phase allows safe communication without the use of a preshared key, and the symmetric encryption is faster and more practical for use on the session data.

For the client to verify the identity of the server, the server must have a previously generated private key as well as a certificate containing the public key and information about the server. This certificate must be verifiable using a public key that is known to the client.

Certificates are generally digitally signed by a third-party *certificate authority (CA)* that has verified the identity of the requester and the validity of the request to have the certificate signed. In most cases, the CA is a company that has made arrangements with the web browser vendor to have its own certificate installed and trusted by default client installations. The CA then charges the server operator for its services.

Commercial certificate authorities vary in price, features, and browser support, but remember that price is not always an indication of quality. Some popular CAs are InstantSSL (`https://www.instantssl.com`), Let's Encrypt (`https://www.letsencrypt.org`), and DigiCert (`https://www.digicert.com`).

You also have the option of creating self-signed certificates, although these should be used only for testing or when a very small number of people will be accessing your server and you do not plan to have certificates on multiple machines. Directions for generating a self-signed certificate are included in the section "Generating an SSL key and self-signed certificate" later in this chapter.

The last option is to run your own certificate authority. This is probably practical only if you have a small number of expected users and the means to distribute your CA certificate to them (including assisting them with installing it in their browsers). The process for creating a CA is too elaborate to cover in this book, but it is a worthwhile alternative to generating self-signed certificates.

The following sections describe how HTTPS communications are configured by default in Fedora and RHEL when you install the `mod_ssl` package. After that, I describe how to configure SSL communications better by generating your own SSL keys and certificates to use with the web server (running on a Fedora or RHEL system) configured in this chapter.

Understanding how SSL is configured

If you have installed the `mod_ssl` package in Fedora or RHEL (which is done by default if you installed the Basic Web Server group), a self-signed certificate and private key are created when the package is installed. This allows you to use HTTPS protocol immediately to communicate with the web server.

Although the default configuration of `mod_ssl` allows you to have encrypted communications between your web server and clients, because the certificate is self-signed, a client accessing your site is warned that the certificate is untrusted. To begin exploring the SSL configuration for your Apache web server, make sure that the `mod_ssl` package is installed on the server running your Apache `(httpd)` service:

```
# yum install mod_ssl
```

The `mod_ssl` package includes the module needed to implement SSLon your web server (`mod_ssl.so`) and a configuration file for your SSL hosts: `/etc/httpd/conf.d/ssl.conf`. There are many comments in this file to help you understand what to change. Those lines that are not commented out define some initial settings and a default virtual host. Here are some of those lines:

```
Listen 443 https
...
<VirtualHost _default_:443>
ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel warn
SSLEngine on
...
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
...
</VirtualHost>
```

The SSL service is set to listen on standard SSL port 443 on all the system's network interfaces.

A `VirtualHost` block is created that causes error messages and access messages to be logged to log files that are separate from the standard logs used by the server (`ssl_error_log` and `ssl_access_log` in the `/var/log/httpd/` directory). The level of log messages is set to `warn` and the SSLEngine is turned on.

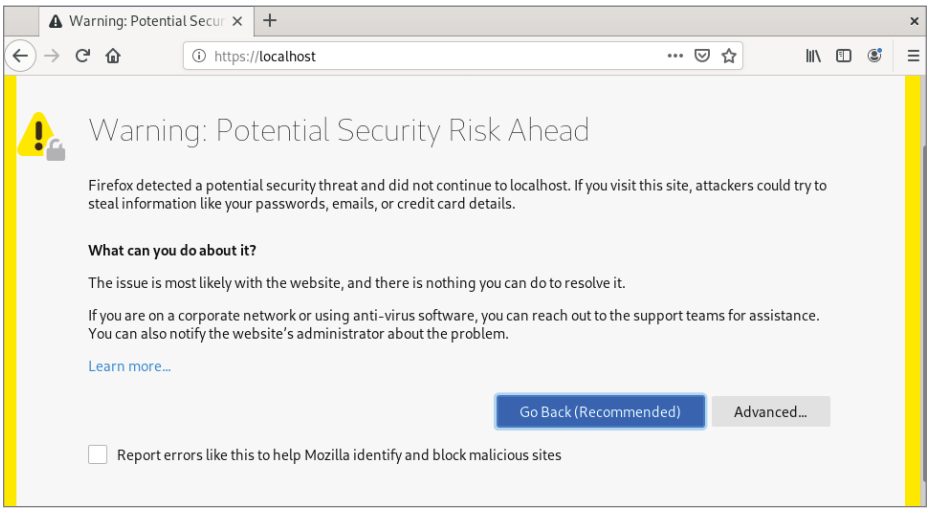
In the preceding sample code, two entries associated with SSL Certificates in the `VirtualHost` block identify the key and certificate information. As mentioned previously, a key is generated when `mod_ssl` is installed and placed in the file `/etc/pki/tls/private/localhost.key`. A self-signed certificate, `/etc/pki/tls/certs/localhost.crt`, is created using that key. When you create your own key and certificate later, you need to replace the values of `SSLCertificateFile` and `SSLCertificateKeyFile` in this file.

After installing the `mod_ssl` package and reloading the configuration file, you can test that the default certificate is working by following these steps:

1. **Open a connection to the website from a web browser, using the `HTTPS` protocol.** For example, if you are running Firefox on the system where the web server is running, type **`https://localhost`** into the location box and press Enter. Figure 17.2 shows an example of the page that appears.

FIGURE 17.2

Accessing an SSL website with a default certificate



2. This page warns you that there is no way of verifying the authenticity of this site. That is because there is no way to know who created the certificate that you are accepting.
3. **Because you are accessing the site via a browser on the local host, click Advanced and then View to see the certificate that was generated.** It includes your hostname, information on when the certificate was issued and when it expires, and lots of other organization information.
4. **Select Accept the Risk and Continue to allow connections to this site.**
5. **Close that window, and then select Confirm Security Exception to accept the connection.** You should now see your default web page using HTTPS protocol. From now on, your browser will accept HTTPS connections to the web server using that certificate and encrypt all communications between the server and browser.

Because you don't want your website to scare off users, the best thing to do is to get a valid certificate to use with your site. The next best thing to do is to create a self-signed certificate that at least includes better information about your site and organization. The following section describes how to do that.

Generating an SSL key and self-signed certificate

To begin setting up SSL, use the `openssl` command, which is part of the `openssl` package, to generate your public and private key. After that, you can generate your own self-signed certificate to test the site or to use internally.

1. **If the `openssl` package is not already installed, install it as follows:**

   ```
   # yum install openssl
   ```

2. **Generate a 2048-bit RSA private key and save it to a file:**

   ```
   # cd /etc/pki/tls/private
   # openssl genrsa -out server.key 2048
   # chmod 600 server.key
   ```


Note

You can use a filename other than `server.key` and should do so if you plan to have more than one SSL host on your machine (which requires more than one IP address). Just make sure that you specify the correct filename in the Apache configuration later.

Or, in higher-security environments, encrypting the key by adding the `-des3` argument after the `genrsa` argument on the `openssl` command line is a good idea. When prompted for a passphrase, press Enter:

```
# openssl genrsa -des3 -out server.key 1024
```

3. **If you don't plan to have your certificate signed, or if you want to test your configuration, generate a self-signed certificate and save it in a file named `server.crt` in the `/etc/pki/tls/certs` directory:**

   ```
   # cd /etc/pki/tls/certs
   # openssl req -new -x509 -nodes -sha1 -days 365 \
      -key /etc/pki/tls/private/server.key \
      -out server.crt
   Country Name (2 letter code) [AU]: US
   State or Province Name (full name) [Some-State]: NJ
   Locality Name (eg, city) [Default City]: Princeton
   Organization Name (eg, company) [Default Company Ltd
   Ltd]:TEST USE ONLY
   Organizational Unit Name (eg, section) []:TEST USE ONLY
   Common Name (eg, YOUR name) []:secure.example.org
   Email Address []:dom@example.org
   ```

4. **Edit the `/etc/httpd/conf.d/ssl.conf` file to change the key and certificate locations to use the ones that you just created.** For example:

   ```
   SSLCertificateFile /etc/pki/tls/certs/server.crt
   SSLCertificateKeyFile /etc/pki/tls/private/server.key
   ```

5. **Restart or reload the `httpd` server.**
6. **Open `https://localhost` from a local browser again, repeat the procedure to review, and accept the new certificate.**

For internal use or testing, a self-signed certificate might work for you. However, for public websites, you should use a certificate that is validated by a certificate authority (CA). The procedure for doing that is covered next.

Generating a certificate signing request

If you plan to have your certificate signed by a CA (including one that you run yourself), you can use your private key to generate a certificate signing request (CSR):

1. **Create a directory for storing your CSR.**

   ```
   # mkdir /etc/pki/tls/ssl.csr
   # cd /etc/pki/tls/ssl.csr/
   ```

2. **Use the `openssl` command to generate the CSR.** The result is a CSR file in the current directory named `server.csr`. When you enter the information, the Common Name entry should match the name that clients will use to access your server. Be sure to get the other details right so that it can be validated by a third-party CA. Also, if you had entered a passphrase for your key, you are prompted to enter it here to use the key.

   ```
   # openssl req -new -key ../private/server.key -out server.csr

   Country Name (2 letter code) [AU]:US
   State or Province Name (full name) [Some-State]:Washington
   Locality Name (eg, city) []:Bellingham
   Organization Name (eg, company) [Internet Widgits Pty
   Ltd]:Example Company, LTD.
   Organizational Unit Name (eg, section) []:Network
    Operations
   Common Name (eg, YOUR name) []:secure.example.org
   Email Address []:dom@example.org

   Please enter the following 'extra' attributes
   to be sent with your certificate request
   A challenge password []:
   An optional company name []:
   ```

3. **Visit the website of the certificate signing authority that you choose and request a signed certificate.** At some point, the CA site will probably ask you to

copy and paste the contents of your CSR (`server.csr` file in this example) into a form needed to make the request.

4. **When the CA sends you the certificate (probably via email), save it in the `/etc/pki/tls/certs/` directory using a name based on the site you are hosting — for example, `example.org.crt`.**
5. **Change the value of `SSLCertificateFile` in the `/etc/httpd/conf.d/ssl.conf` file to point to your new CRT file.** Or, if you have multiple SSL hosts, you might want to create a separate entry (possibly in a separate `.conf` file) that looks like the following:

```
Listen 192.168.0.56:443
<VirtualHost *:443>
    ServerName      secure.example.org
    ServerAlias     web.example.org
    DocumentRoot    /home/username/public_html/
    DirectoryIndex  index.php index.html index.htm
    SSLEngine       On
    SSLCertificateKeyFile /etc/pki/tls/private/server.key
    SSLCertificateFile /etc/pki/tls/certs/example.org.crt
</VirtualHost>
```

The IP address shown in the `Listen` directive should be replaced by the public IP address representing the SSL host you are serving. Remember that each SSL host should have its own IP address.

Troubleshooting Your Web Server

In any complex environment, you occasionally run into problems. The following sections include tips for isolating and resolving the most common errors that you may encounter.

Checking for configuration errors

You may occasionally run into configuration errors or script problems that prevent Apache from starting or that prevent specific files from being accessible. Most of these problems can be isolated and resolved using two Apache-provided tools: the `apachectl` program and the system error log.

When encountering a problem, first use the `apachectl` program with the `configtest` parameter to test the configuration. In fact, it's a good idea to develop the habit of running this every time you make a configuration change:

```
# apachectl configtest
Syntax OK
# apachectl graceful
/usr/sbin/apachectl graceful: httpd gracefully restarted
```

In the event of a syntax error, `apachectl` indicates where the error occurs and also does its best to give a hint about the nature of the problem. You can then use the `graceful` restart option (`apachectl graceful`) to instruct Apache to reload its configuration without disconnecting any active clients.

NOTE
The `graceful` restart option in `apachectl` automatically tests the configuration before sending the reload signal to `apache`, but getting in the habit of running the manual configuration test after making any configuration changes is still a good idea.

Some configuration problems pass the syntax tests performed by `apachectl` but cause the HTTP daemon to exit immediately after reloading its configuration. If this happens, use the `tail` command to check Apache's error log for useful information. On Fedora and RHEL systems, the error log is in `/var/log/httpd/error.log`. On other systems, you can find the location by looking for the `ErrorLog` directive in your Apache configuration.

You might encounter an error message that looks something like this:

```
[crit] (98)Address already in use: make_sock: could not bind to port
80
```

This error often indicates that something else is bound to port 80, that another Apache process is already running (`apachectl` usually catches this), or that you have told Apache to bind the same IP address and port combination in more than one place.

You can use the `netstat` command to view the list of programs (including Apache) with TCP ports in the `LISTEN` state:

```
# netstat -nltp
Active Internet connections (only servers)
Proto  Local Address  Foreign Address  State   PID/Program name
tcp6   :::80          :::*             LISTEN  2105/httpd
```

The output from `netstat` (which was shortened to fit here) indicates that an instance of the `httpd` process with a process ID of `2105` is listening (as indicated by the `LISTEN` state) for connections to any local IP address (indicated by `:::80`) on port 80 (the standard HTTP port). If a different program is listening to port 80, it is shown there. You can use the `kill` command to terminate the process, but if it is something other than `httpd`, you should also find out why it is running.

If you don't see any other processes listening on port 80, it could be that you have accidentally told Apache to listen on the same IP address and port combination in more than one place. Three configuration directives can be used for this: `BindAddress`, `Port`, and `Listen`:

- `BindAddress` enables you to specify a single IP address on which to listen, or you can specify all IP addresses using the `*` wildcard. You should never have more than one `BindAddress` statement in your configuration file.

- `Port` specifies on which TCP port to listen, but it does not enable you to specify the IP address. `Port` is generally not used more than once in the configuration.
- `Listen` enables you to specify both an IP address and a port to bind to. The IP address can be in the form of a wildcard, and you can have multiple `Listen` statements in your configuration file.

To avoid confusion, it is generally a good idea to use only one of these directive types. Of the three, `Listen` is the most flexible, so it is probably the one you want to use the most. A common error when using `Listen` is to specify a port on all IP addresses (`*:80`) as well as that same port on a specific IP address (`1.2.3.4:80`), which results in the error from `make_sock`.

Configuration errors relating to SSL commonly result in Apache starting improperly. Make sure that all key and certificate files exist and that they are in the proper format (use `openssl` to examine them).

For other error messages, try doing a web search to see whether somebody else has encountered the problem. In most cases, you can find a solution within the first few matches.

If you aren't getting enough information in the `ErrorLog`, you can configure it to log more information using the `LogLevel` directive. The options available for this directive, in increasing order of verbosity, are `emerg`, `alert`, `crit`, `error`, `warn`, `notice`, `info`, and `debug`. Select only one of these.

Any message that is at least as important as the `LogLevel` that you select are stored in the `ErrorLog`. On a typical server, `LogLevel` is set to `warn`. You should not set it to any value lower than `crit`, and you should avoid leaving it set to `debug` because that can slow down the server and result in a very large `ErrorLog`.

As a last resort, you can also try running `httpd -X` manually to check for crashes or other error messages. The `-X` runs `httpd` so that it displays debug and higher messages on the screen.

Accessing forbidden and server internal errors

The two common types of errors that you may encounter when attempting to view specific pages on your server are permission errors and server internal errors. Both types of errors can usually be isolated using the information in the error log. After making any of the changes described in the following list to attempt to solve one of these problems, try the request again and check the error log to see whether the message has changed (for example, to show that the operation completed successfully).

Note

"File not found" errors can be checked in the same way as "access forbidden" and "server internal errors." You may sometimes find that Apache is not looking where you think it is for a specific file. Generally, the entire path to the file shows up in the error log. Make sure that you are accessing the correct virtual host, and check for any `Alias` settings that might be directing your location to a place you don't expect.

File permissions A "File permissions prevent access" error indicates that the `apache` process is running as a user that is unable to open the requested file. By default, `httpd` is run by the `Apache` user and group. Make sure that the account has execute permissions on the directory, and every directory above it, as well as read permissions on the files themselves. Read permissions on a directory are also necessary if you want Apache to generate an index of files. See the manual page for `chmod` for more information about how to view and change permissions.

NOTE
Read permissions are not necessary for compiled binaries, such as those written in C or C++, but they can be safely added unless a need exists to keep the contents of the program secret.

Access denied A "Client denied by server configuration" error indicates that Apache was configured to deny access to the object. Check the configuration files for `Location` and `Directory` sections that might affect the file that you are trying to access. Remember that settings applied to a path are also applied to any paths below it. You can override these by changing the permissions only for the more specific path to which you want to allow access.

Index not found The "Directory index forbidden by rule" error indicates that Apache could not find an index file with a name specified in the `DirectoryIndex` directive and was configured not to create an index containing a list of files in a directory. Make sure that your index page, if you have one, has one of the names specified in the relevant `DirectoryIndex` directive, or add an `Options Indexes` line to the appropriate `Directory` or `Location` section for that object.

Script crashed "Premature end of script headers" errors can indicate that a script is crashing before it finishes. On occasion, the errors that caused this also show up in the error log. When using `suexec` or `suPHP`, this error may also be caused by a file ownership or permissions error. These errors appear in log files in the `/var/log/httpd` directory.

SELinux errors If file permissions are open but messages denying permission appear in log files, SELinux could be causing the problem. Set SELinux to permissive mode temporarily (`setenforce 0`) and try to access the file again. If the file is now accessible, set SELinux to enforcing mode again (`setenforce 1`) and check file contexts and Booleans. File contexts must be correct for `httpd` to be able to access a file. A Boolean might prevent a file being served from a remotely mounted directory or prevent a page from sending an email or uploading a file. Type **`man httpd_selinux`** for details about SELinux configuration settings associated with the `httpd` services. (Install the `selinux-policy-devel` package to have that man page added to your system.)

Summary

The open source Apache project is the world's most popular web server. Although Apache offers tremendous flexibility, security, and complexity, a basic Apache web server can be configured in just a few minutes in Fedora, RHEL, and most other Linux distributions.

The chapter described the steps for installing, configuring, securing, and troubleshooting a basic Apache web server. You learned how to configure virtual hosting and secure SSL hosts. You also learned how to configure Apache to allow any user account on the system to publish content from their own `public_html` directory.

Continuing on the topic of server configuration, in Chapter 18, "Configuring an FTP Server," you will learn how to set up an FTP server in Linux. The examples illustrate how to configure an FTP server using the `vsftpd` package.

Exercises

The exercises in this section cover topics related to installing and configuring an Apache web server. As usual, I recommend that you use a spare Fedora or Red Hat Enterprise Linux system to do the exercises. Don't do these exercises on a production machine because these exercises modify the Apache configuration files and service, and they could damage services that you have currently configured. Try to use a virtual machine or find a computer where it will do no harm to interrupt services on the system.

These exercises assume that you are starting with a Fedora or RHEL installation on which the Apache server (`httpd` package) is not yet installed.

If you are stuck, solutions to the tasks are shown in Appendix B. These show you one approach to each task, although Linux may offer multiple ways to complete a task.

1. From a Fedora system, install all of the packages associated with the Basic Web Server group.
2. Create a file called `index.html` in the directory assigned to `DocumentRoot` in the main Apache configuration file. The file should have the words "My Own Web Server" inside.
3. Start the Apache web server and set it to start up automatically at boot time. Check that it is available from a web browser on your local host. (You should see the words "My Own Web Server" displayed if it is working properly.)
4. Use the `netstat` command to see on which ports the `httpd` server is listening.
5. Try to connect to your Apache web server from a web browser that is outside of the local system. If it fails, correct any problems that you encounter by investigating the firewall, SELinux, and other security features.

6. Using the `openssl` or similar command, create your own private RSA key and self-signed SSL certificate.
7. Configure your Apache web server to use your key and self-signed certificate to serve secure (HTTPS) content.
8. Use a web browser to create an HTTPS connection to your web server and view the contents of the certificate that you created.
9. Create a file named `/etc/httpd/conf.d/example.org.conf`, which turns on name-based virtual hosting and creates a virtual host that does these things:
   - Listens on port 80 on all interfaces
   - Has a server administrator of `joe@example.org`
   - Has a server name of `joe.example.org`
   - Has a `DocumentRoot` of `/var/www/html/example.org`
   - Has a `DirectoryIndex` that includes at least `index.html`

   Create an `index.html` file in `DocumentRoot` that contains the words "Welcome to the House of Joe" inside.
10. Add the text `joe.example.org` to the end of the localhost entry in your `/etc/hosts` file on the machine that is running the web server. Then type `http://joe.example.org` into the location box of your web browser. You should see "Welcome to the House of Joe" when the page is displayed.

CHAPTER 18

Configuring an FTP Server

IN THIS CHAPTER

Learning how FTP works

Getting a vsftpd server installed

Choosing security settings for vsftpd

Setting up vsftpd configuration files

Running FTP clients

The *File Transfer Protocol (FTP)* is one of the oldest protocols in existence for sharing files over networks. Although there are more secure protocols for network file sharing, FTP is still used quite often for making files freely available on the Internet.

Several FTP server projects are available with Linux today. However, the one often used with Fedora, Red Hat Enterprise Linux, CentOS, Ubuntu, and other Linux distributions is the Very Secure FTP Daemon (`vsftpd` package). This chapter describes how to install, configure, use, and secure an FTP server using the `vsftpd` package.

Understanding FTP

FTP operates in a client/server model. An FTP server daemon listens for incoming requests (on TCP port 21) from FTP clients. The client presents a login and password. If the server accepts the login information, the client can interactively traverse the filesystem, list files and directories, and then download (and sometimes upload) files.

What makes FTP insecure is that everything sent between the FTP client and server is done in clear text. The FTP protocol was created at a time when most computer communication was done on private lines or over dial-up, where encryption was not thought to be critical. If you use FTP over a public network, someone sniffing the line anywhere between the client and server would be able to see not only the data being transferred but also the authentication process (login and password information).

So, FTP is not good for sharing files privately (use SSH commands such as `sftp`, `scp`, or `rsync` if you need private, encrypted file transfers). However, if you are sharing public documents, open source software repositories, or other openly available data, FTP is a good choice. Regardless of the operating system people use, they surely have an FTP file transfer application available to get files that you offer from your FTP server.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

When users authenticate to an FTP server in Linux, their usernames and passwords are authenticated against the standard Linux user accounts and passwords. There is also a special, non-authenticated account used by the FTP server called anonymous. The anonymous account can be accessed by anyone because it does not require a valid password. In fact, the term *anonymous FTP server* is often used to describe a public FTP server that does not require (or even allow) authentication of a legitimate user account.

NOTE

Although the ability to log in to the `vsftpd` server using a regular Linux user account is enabled by default in Fedora and Red Hat Enterprise Linux, if SELinux is set to enforcing mode, it prevents the logins and file transfers from succeeding. If you want to keep SELinux in enforcing mode yet still allow Linux logins, you can change a Boolean (see the section "Configuring SELinux for your FTP server" later in this chapter) to allow regular user logins to succeed.

After the authentication phase (on the control port, TCP port 21), a second connection is made between the client and server. FTP supports both active and passive connection types. With an *active FTP connection*, the server sends data from its TCP port 20 to some random port the server chooses above port 1023 on the client. With a *passive FTP connection*, the client requests the passive connection and requests a random port from the server.

Many browsers support passive FTP mode so that if the client has a firewall, it doesn't block the data port that the FTP server might use in active mode. Supporting passive mode requires some extra work on the server's firewall to allow random connections to ports above 1023 on the server. The section "Opening up your firewall for FTP," later in this chapter, describes what you need to do to your Linux firewall to make both passive and active FTP connections work.

After the connection is established between the client and server, the client's current directory is established. For the anonymous user, the `/var/ftp` directory is the home directory for Fedora or RHEL, and it's `/srv/ftp` for Ubuntu and most Debian-based distributions. The anonymous user cannot go outside of the `/var/ftp` directory structure. If a regular user, let's say joe, logs in to the FTP server, `/home/joe` is joe's current directory, but joe can change to any part of the filesystem for which he has permission.

Command-oriented FTP clients (such as `lftp` and `ftp` commands) go into an interactive mode after connecting to the server. From the prompt you see, you can run many commands that are similar to those that you would use from the shell. You could use `pwd` to see your current directory, `ls` to list directory contents, and `cd` to change directories. When you see a file that you want, you use the `get` and `put` commands to download files from or upload them to the server, respectively.

With graphical tools for accessing FTP servers (such as a web browser), you type the URL of the site that you want to visit (such as `ftp://docs.example.com`) into the location box of the browser. If you don't add a username or password, an anonymous connection is made and the contents of the home directory of the site are displayed. Click links to directories to change to those directories. Click links to files to display or download those files to your local system.

Armed with some understanding of how FTP works, you are now ready to install an FTP server (vsftpd package) on your Linux system.

Installing the vsftpd FTP Server

Setting up the Very Secure FTP server requires only one package in Fedora, RHEL, and other Linux distributions: vsftpd. Assuming you have a connection to your software repository, just type the following as root for Fedora or RHEL to install vsftpd:

```
# yum install vsftpd
```

If you are using Ubuntu (or another Linux distribution based on Debian packaging), type the following to install vsftpd:

```
$ sudo apt-get install vsftpd
```

Here are some commands that you can run after the vsftpd package is installed to familiarize yourself with the contents of that package. From Fedora or RHEL, run this command to get some general information about the package:

```
# rpm -qi vsftpd
...
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : https://security.appspot.com/vsftpd.html
Summary     : Very Secure Ftp Daemon
Description : vsftpd is a Very Secure FTP daemon. It was written
              completely from scratch.
```

If you want to get more information about vsftpd, follow the URL listed to the related website (https://security.appspot.com/vsftpd.html). You can get additional documentation and information about the latest revisions of vsftpd.

You can view the full contents of the vsftpd package (rpm -ql vsftpd), or you can view just the documentation (-qd) or configuration (-qc) files. To see the documentation files in the vsftpd package, use the following:

```
# rpm -qd vsftpd
/usr/share/doc/vsftpd/EXAMPLE/INTERNET_SITE/README
...
/usr/share/doc/vsftpd/EXAMPLE/PER_IP_CONFIG/README
...
/usr/share/doc/vsftpd/EXAMPLE/VIRTUAL_HOSTS/README
/usr/share/doc/vsftpd/EXAMPLE/VIRTUAL_USERS/README
...
/usr/share/doc/vsftpd/FAQ
...
/usr/share/doc/vsftpd/vsftpd.xinetd
/usr/share/man/man5/vsftpd.conf.5.gz
/usr/share/man/man8/vsftpd.8.gz
```

In the /usr/share/doc/vsftpd/EXAMPLE directory structure, there are sample configuration files included to help you configure vsftpd in ways that are appropriate for an Internet site, multiple IP address site, and virtual hosts. The main /usr/share/doc/vsftpd directory contains an FAQ (frequently asked questions), installation tips, and version information.

The man pages might have the most useful information when you set out to configure the vsftpd server. Type **man vsftpd.conf** to read about the configuration file and **man vsftpd** to read about the daemon process and how to manage it as a systemd service.

To list the configuration files, type the following:

```
# rpm -qc vsftpd
/etc/logrotate.d/vsftpd
/etc/pam.d/vsftpd
/etc/vsftpd/ftpusers
/etc/vsftpd/user_list
/etc/vsftpd/vsftpd.conf
```

The main configuration file is /etc/vsftpd/vsftpd.conf (in RHEL and Fedora) or /etc/vsftpd.conf (in Ubuntu). The ftpusers and user _ list (Fedora and RHEL, but not Ubuntu) files in the same directory store information about user accounts that are restricted from accessing the server. The /etc/pam.d/vsftpd file sets how authentication is done to the FTP server. The /etc/logrotate.d/vsftpd file configures how log files are rotated over time.

Now you have vsftpd installed and have taken a quick look at its contents. The next step is to start up and test the vsftpd service.

Starting the vsftpd Service

No configuration is required to launch the vsftpd service if you just want to use the default settings. If you start vsftpd as it is delivered with Fedora, the following is what you get:

- The vsftpd service starts the vsftpd daemon, which runs in the background.
- The standard port on which the vsftpd daemon listens is TCP port 21. By default, data is transferred to the user, after the connection is made, on TCP port 20. TCP port 21 must be open in the firewall to allow new connections to access the service. Both IPv4 and IPv6 connections are available by default. This procedure changes to the TCP IPv4 service. (See the section "Securing Your FTP Server" later in this chapter for details on opening ports, enabling connection tracking needed for passive FTP, and setting other firewall rules related to FTP.)
- The vsftpd daemon reads vsftpd.conf to determine what features the service allows.
- Linux user accounts (excluding administrative users) can access the FTP server. The anonymous user account (no password required) can be enabled. (If SELinux is in enforcing mode, you need to set a Boolean to allow regular users to log in to the FTP server. See the section "Securing Your FTP Server" for details.)

- The anonymous user has access only to the /var/ftp directory and its subdirectories. A regular user starts with their home directory as the current directory but can access any directory to which the user would be able to gain access via a regular login or SSH session. Lists of users in the /etc/vsftpd/user_list and /etc/vsftpd/ftpusers files define some administrative and special users who do not have access to the FTP server (root, bin, daemon, and others).
- By default, the anonymous user can download files from the server but not upload them. A regular user can upload or download files, based on regular Linux permissions.
- Log messages detailing file uploads or downloads are written in the /var/log/xferlogs file. Those log messages are stored in a standard xferlog format.

If you are ready to start your server using the defaults just described, the following examples show you how to do that. If you want to change some additional settings first, go to the section "Configuring Your FTP Server," later in this chapter, finalize your settings, and then come back here for instructions on how to enable and start your server.

1. Check the vsftpd service. Before you start the vsftpd service, you can check out whether it is running already. In Fedora or Red Hat Enterprise Linux 7 or 8, you do the following:

```
# systemctl status vsftpd.service
vsftpd.service - Vsftpd ftp daemon
       Loaded: loaded (/usr/lib/systemd/system/vsftpd.
service; disabled)
       Active: inactive (dead)
```

   In Red Hat Enterprise Linux 6, you need two commands to see the same information:

```
# service vsftpd status
vsftpd is stopped
# chkconfig --list vsftpd
vsftpd  0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

   In both the Fedora and RHEL examples above, the service, chkconfig, and systemctl commands show the status as stopped. You can also see that it is disabled in Fedora and RHEL 7 or 8 and off at every runlevel for RHEL 6. Disabled (off) means that the service will not turn on automatically when your start the system.

2. To start and enable vsftpd in Fedora or RHEL 7 or 8 (then check the status), type the following:

```
# systemctl start vsftpd.service
# systemctl enable vsftpd.service
```

```
        ln -s '/lib/systemd/system/vsftpd.service'
          '/etc/systemd/system/multi-user.target.wants/vsftpd.
service'
        # systemctl status vsftpd.service
        vsftpd.service - Vsftpd ftp daemon
               Loaded: loaded (/usr/lib/systemd/system/vsftpd.
service;
                  enabled vendor preset: disabled))
               Active: active (running) since Wed, 2019-09-18
00:09:54 EDT; 22s ago
               Main PID: 4229 (vsftpd)
                  Tasks: 1 (limit: 12232)
                 Memory: 536.0K
                  CGroup: /system.slice/vsftpd.service
                 └4229 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf
```

In Red Hat Enterprise Linux 6, start and turn on (enable) `vsftpd` (then check the status), as follows:

```
        # service vsftpd start
        Starting vsftpd for vsftpd:                [  OK  ]
        # chkconfig vsftpd on ; chkconfig --list vsftpd
        vsftpd       0:off    1:off    2:on    3:on    4:on    5:on
6:off
```

3. Now, on either system, you could check that the service is running using the `netstat` command:

```
        # netstat -tupln | grep vsftpd
        tcp  0    0 0.0.0.0:21  0.0.0.0:*  LISTEN   4229/vsftpd
```

   From the `netstat` output, you can see that the `vsftpd` process (process ID of 4229) is listening (`LISTEN`) on all IP addresses for incoming connections on port 21 (0.0.0.0:21) for the TCP (`tcp`) protocol.

4. A quick way to check that `vsftpd` is working is to put a file in the `/var/ftp` directory and try to open it from your web browser on the local host:

```
        # echo "Hello From Your FTP Server" > /var/ftp/hello.txt
```

   From a web browser on the local system, type the following into the location box of Firefox or another browser:

```
        ftp://localhost/hello.txt
```

If the text `Hello From Your FTP Server` appears in the web browser, the `vsftpd` server is working and accessible from your local system. Next, try this again from a web browser on another system, replacing `localhost` with your host's IP address or fully

qualified host name. If that works, the vsftpd server is publicly accessible. If it doesn't, which it quite possibly may not, see the next section, "Securing Your FTP Server." That section tells you how to open firewalls and modify other security features to allow access and otherwise secure your FTP server.

Securing Your FTP Server

Even though it is easy to get a vsftpd FTP server started, that doesn't mean that it is immediately fully accessible. If you have a firewall in place on your Linux system, it is probably blocking access to all services on your system except for those that you have explicitly allowed.

If you decide that the default vsftpd configuration works for you as described in the previous section, you can set to work allowing the appropriate access and providing security for your vsftpd service. To help you secure your vsftpd server, the next sections describe how to configure your firewall and SELinux (Booleans and file contexts).

Opening up your firewall for FTP

If you have a firewall implemented on your system, you need to add firewall rules that allow incoming requests to your FTP site and allow packets to return to your system on established connections. Firewalls are implemented using iptables rules and managed with the iptables service or firewalld service (see Chapter 25, "Securing Linux on a Network," for details about firewall services).

In Fedora and Red Hat Enterprise Linux, firewall rules have traditionally been stored in the /etc/sysconfig/iptables file and the underlying service was iptables (RHEL 6) or iptables.service (Fedora). Modules are loaded into your firewall from the /etc/sysconfig/iptables-config file. In RHEL 7 and Fedora 21 or later, the new firewalld service manages those rules and rules are stored in the /etc/firewalld/zones directory.

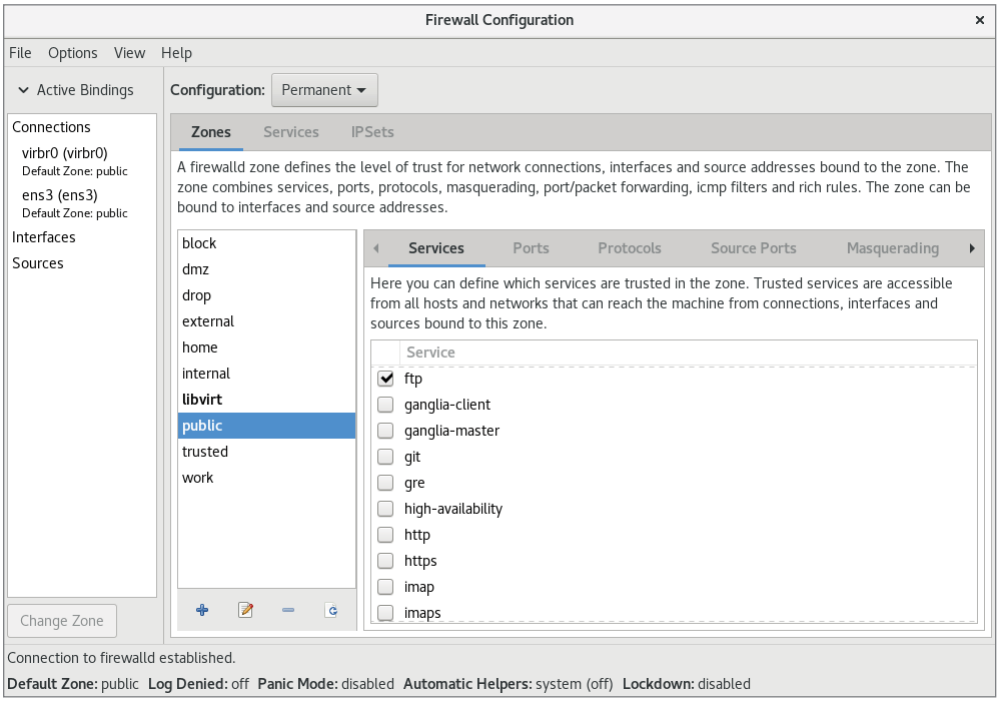
Note

It is best to work on your firewall directly from a system console, if possible, instead of over a remote login (such as ssh) because a small error can immediately lock you out of your server. After that, you must go over to the console to get back into the server and fix the problem. You need to add a few things to your firewall to allow access to your FTP server without opening up access to other services. First, you need to allow your system to accept requests on TCP port 21; then you need to make sure that the connection tracking module is loaded.

In RHEL 7 and Fedora 20 or later, you can use the Firewall Configuration window to enable your firewall and open access to your FTP service. Install the firewall-config package and run firewall-config to start the Firewall Configuration window, as shown in Figure 18.1.

FIGURE 18.1

Open access to your FTP service from the Firewall Configuration window.



Next, to open access permanently to your FTP service, click the Configuration box and select Permanent. Then select the check box next to ftp under the Services tab. This automatically opens TCP port 21 (FTP) on your firewall and loads kernel modules needed to allow access to passive FTP service. Select Options ➪ Reload Firewalld to apply the firewall rule permanently.

For RHEL 6 and earlier systems, add rules directly to the `/etc/sysconfig/iptables` file. If you are using a default firewall, rules in the beginning open access to requests for any services coming from the local host and allow packets to come in that are associated with, or related to, established connections. In the middle are rules that open ports for service requests that you have already allowed, such as the secure shell service (`sshd` on TCP port 22). At the end of the rules, a final rule usually `DROP`s or `REJECT`s any request that has not explicitly been allowed.

To allow public access to someone requesting your FTP server, you want to allow new requests to TCP port 21. You typically want to add the rule somewhere before the final `DROP` or `REJECT` rule. The following output shows partial contents of the `/etc/sysconfig/iptables` file with the rule allowing access to your FTP server in bold:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 21 -j ACCEPT
...
-A INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

This example shows that, for the filter table, the firewall accepts packets from established connections, connections from local hosts, and any new requests on TCP port 22 (SSH service). The line we just added (`--dport 21`) allows any packets on new connections to TCP port 21 to be accepted.

Note

It is important to have the `ESTABLISHED, RELATED` line in your `iptables` firewall rules. Without that line, users would be able to connect to your SSH (port 22) and FTP (port 21) services, but they would not be able to communicate after that. So, a user could get authenticated but not be able to transfer data.

The next thing that you must do on RHEL 6 and earlier systems is set up the FTP connection tracking module to be loaded each time the firewall starts up. Edit this line at the beginning of the `/etc/sysconfig/iptables-config` file to appear as follows:

```
IPTABLES_MODULES="nf_conntrack_ftp"
```

At this point, you can restart your firewall (keeping in mind that a mistake could lock you out if you are logged in remotely). Use one of the following two commands to restart your firewall, depending on whether your system is using the older `iptables` service or the newer `firewalld` service:

```
# service iptables restart
```

or

```
# systemctl restart firewalld.service
```

Try again to access your FTP server from a remote system (using a web browser or some other FTP client).

Configuring SELinux for your FTP server

If SELinux is set to permissive or disabled, it does not block access to the `vsftpd` service in any way. However, if SELinux is in enforcing mode, a few SELinux issues could cause your `vsftpd` server not to behave as you would like. Use the following commands to check the state of SELinux on your system:

```
# getenforce
Enforcing
# grep ^SELINUX= /etc/sysconfig/selinux
SELINUX=enforcing
```

The `getenforce` command shows how SELinux is currently set. (Here, it's in enforcing mode.) The `SELINUX=` variable in `/etc/sysconfig/selinux` shows how SELinux is set when the system comes up. If it is in enforcing mode, as it is here, check the `ftpd_selinux` man page for information about SELinux settings that can impact the operation of your `vsftpd` service. Install the `selinux-policy-doc` package to get the `ftpd_selinux` man page as well as man pages for other services with SELinux policies.

Here are some examples of file contexts that must be set for SELinux to allow files and directories to be accessed by `vsftpd`:

- To share content so that it can be downloaded to FTP clients, that content must be marked with a `public_content_t` file context. Files created in the `/var/ftp` directory or its subdirectories inherit `public_content_t` file context automatically. (Be sure to create new content or copy existing content to the `/var/ftp` directories. Moving the files there may not change the file context properly.)
- To allow files to be uploaded by anonymous users, the file context on the directory to which you upload must be set to `public_content_rw_t`. (Other permissions, SELinux Booleans, and `vsftpd.conf` settings must be in place for this to work as well.)

If you have files in the `/var/ftp` directory structure that have the wrong file contexts (which can happen if you move files there from other directories instead of copying them), you can change or restore the file context on those files so that they can be shared. For example, to recursively change the file context of the `/var/ftp/pub/stuff` directory so that the content can be readable from the FTP server through SELinux, enter the following:

```
# semanage fcontext -a -t public_content_t "/var/ftp/pub/stuff(/.*)?"
# restorecon -F -R -v /var/ftp/pub/stuff
```

If you wanted to allow users also to write to a directory as well as to read from it, you would need to assign the `public_content_rw_t` file context to the directory to which you want to allow uploads. This example tells SELinux to allow uploading of files to the `/var/ftp/pub/uploads` directory:

```
# semanage fcontext -a -t public_content_rw_t\
   "/var/ftp/pub/uploads(/.*)?"
# restorecon -F -R -v /var/ftp/pub/uploads
```

FTP server features that are considered insecure by SELinux have Booleans that let you allow or disallow those features. Here are some examples:

- For SELinux to allow anonymous users to read and write files and directories, you need to turn on the `allow_ftpd_anon_write` (RHEL 6) or `ftpd_anon_write` (RHEL 7 or later) Boolean:

```
# setsebool -P ftpd_anon_write on
```

- To be able to mount remote NFS or CIFS (Windows) shared filesystems and share them from your `vsftpd` server, you need to turn on the following two Booleans, respectively:

```
# setsebool -P allow_ftpd_use_nfs on
# setsebool -P allow_ftpd_use_cifs on
```

If you ever find that you cannot access files or directories from your FTP server that you believe should be accessible, try turning off SELinux temporarily:

```
# setpenforce 0
```

If you can access the files or directories with SELinux now in permissive mode, put the system back in enforcing mode (`setenforce 1`). Now you know you have to go back through your SELinux settings and find out what is preventing access. (See Chapter 24, "Enhancing Linux Security with SELinux," for more information on SELinux.)

Relating Linux file permissions to vsftpd

The `vsftpd` server relies on standard Linux file permissions to allow or deny access to files and directories. As you would expect, for an anonymous user to view or download a file, at least read permission must be open for `other` (------r--). To access a directory, at least execute permission must be on for `other` (--------x).

For regular user accounts, the general rule is that if a user can access a file from the shell, that user can access the same file from an FTP server. So, typically, regular users should at least be able to `get` (download) and `put` (upload) files to and from their own home directories, respectively. After permissions and other security provisions are in place for your FTP server, you may want to consider other configuration settings for your FTP server.

Configuring Your FTP Server

Most of the configuration for the `vsftpd` service is done in the `/etc/vsftpd/vsftpd.conf` file. Examples of `vsftpd.conf` for different types of sites are included in the `/usr/share/doc/vsftpd` directory. Depending on how you want to use your FTP site, the following sections discuss a few ways to configure your FTP server.

Remember to restart the `vsftpd` service after making any configuration changes.

Setting up user access

The `vsftpd` server comes with all local Linux users (those listed in the `/etc/passwd` file) configured to access the server and the anonymous user prevented. This is based on the following `vsftpd.conf` settings:

```
anonymous_enable=NO
local_enable=YES
```

Some web server companies let users use FTP to upload the content for their own web servers. In some cases, the users have FTP-only accounts, meaning that they cannot log in to a shell, but they can log in via FTP to manage their content. Creating a user account that has no default shell (actually, `/sbin/nologin`) is how you can keep a user from logging into a shell but still allow FTP access. For example, the `/etc/passwd` entry for the FTP-only user account `bill` might look something like the following:

```
bill:x:1000:1000:Bill Jones:/home/bill:/sbin/nologin
```

With the user account set with `/sbin/nologin` as the default shell, any attempts to log in from a console or via `ssh` as the user `bill` are denied. However, as long as `bill` has a password and local account access to the FTP server is enabled, `bill` should be able to log in to the FTP server via an FTP client.

Not every user with an account on the Linux system has access to the FTP server. The setting `userlist _ enable=YES` in `vsftpd.conf` says to deny access to the FTP server to all accounts listed in the `/etc/vsftpd/user _ list` file. That list includes administrative users `root`, `bin`, `daemon`, `adm`, `lp`, and others. You can add to that list other users to whom you would like to deny access.

If you change `userlist _ enable` to `NO`, the `user _ list` file becomes a list of only those users who do have access to the server. In other words, setting `userlist _ enable=NO`, removing all usernames from the `user _ list` file, and adding the usernames `chris`, `joe`, and `mary` to that file cause the server to allow only those three users to log in to the server.

No matter how the value of `userlist _ enable` is set, the `/etc/vsftpd/ftpusers` file always includes users who are denied access to the server. Like the `userlist _ enable` file, the `ftpusers` file includes a list of administrative users. You can add more users to that file if you want them to be denied FTP access.

One way to limit access to users with regular user accounts on your system is to use `chroot` settings. Here are examples of some `chroot` settings:

```
chroot_local_user=YES
chroot_list_enable=YES
chroot_list_file=/etc/vsftpd/chroot_list
```

With the settings just shown uncommented, you could create a list of local users and add them to the `/etc/vsftpd/chroot _ list` file. After one of those users logged in, that user would be prevented from going to places in the system that were outside of that user's home directory structure.

If uploads to your FTP server are allowed, the directories a user tries to upload to must be writeable by that user. However, uploads can be stored under a username other than that of the user who uploaded the file. This is one of the features discussed next, in the section "Allowing uploading."

Allowing uploading

To allow any form of writing to the vsftpd server, you must have write _ enable=YES set in the vsftpd.conf file (which it is, by default). Because of that, if local accounts are enabled, users can log in and immediately begin uploading files to their own home directories. However, anonymous users are denied the ability to upload files by default.

To allow anonymous uploads with vsftpd, you must have the first option in the following code example, and you may want the second line of code as well (both can be enabled by uncommenting them from the vsftpd.conf file). The first allows anonymous users to upload files; the second allows them to create directories:

```
anon_upload_enable=YES
anon_mkdir_write_enable=YES
```

The next step is to create a directory where anonymous users can write. Any directory under the /var/ftp directory that has write permissions for the user ftp, the ftp group, or other can be written to by an anonymous user. A common thing is to create an uploads directory with permission open for writing. The following are examples of commands to run on the server:

```
# mkdir /var/ftp/uploads
# chown ftp:ftp /var/ftp/uploads
# chmod 775 /var/ftp/uploads
```

As long as the firewall is open and SELinux Booleans are set properly, an anonymous user can cd to the uploads directory and put a file from the user's local system into the uploads directory. On the server, the file would be owned by the ftp user and ftp group. The permissions set on the directory (775) would allow you to see the files that were uploaded but not change or overwrite them.

One reason for allowing anonymous FTP, and then enabling it for anonymous uploads, is to allow people you don't know to drop files into your uploads folder. Because anyone who can find the server can write to this directory, some form of security needs to be in place. You want to prevent an anonymous user from seeing files uploaded by other users, taking files, or deleting files uploaded by other anonymous FTP users. One form of security is the chown feature of FTP.

By setting the following two values, you can allow anonymous uploads. The result of these settings is that when an anonymous user uploads a file, that file is immediately assigned ownership of a different user. The following is an example of some chown settings that you could put in your vsftpd.conf file to use with your anonymous upload directory:

```
chown_uploads=YES
chown_username=joe
```

If an anonymous user were to upload a file after vsftpd was restarted with these settings, the uploaded file would be owned by the user joe and the ftp group. Permissions would be read/write for the owner and nothing for anyone else (rw-------).

So far, you have seen configuration options for individual features on your vsftpd server. Some sets of vsftp.conf variables can work together in ways that are appropriate for certain kinds of FTP sites. The next section contains one of these examples, represented by a sample vsftpd.conf configuration file that comes with the vsftpd package. That file can be copied from a directory of sample files to the /etc/vsftpd/vsftpd.conf file to use for an FTP server that is available on the Internet.

Setting up vsftpd for the Internet

To share files from your FTP server safely to the Internet, you can lock down your server by limiting it to only allow downloads and only from anonymous users. To start with a configuration that is designed to share vsftpd files safely over the Internet, back up your current /etc/vsftpd/vsftpd.conf file and copy this file to overwrite your vsftpd.conf:

```
/usr/share/doc/vsftpd/EXAMPLE/INTERNET_SITE/vsftpd.conf
```

The following paragraphs describe the contents of that vsftpd.conf. Settings in the first section set the access rights for the server:

```
# Access rights
anonymous_enable=YES
local_enable=NO
write_enable=NO
anon_upload_enable=NO
anon_mkdir_write_enable=NO
anon_other_write_enable=NO
```

Turning on anonymous _ enable (YES)and turning off local _ enable (NO)ensures that no one can log in to the FTP server using a regular Linux user account. Everyone must come in through the anonymous account. No one can upload files (write _ enable=NO). Then, the anonymous user cannot upload files (anon _ upload _ enable=NO), create directories (anon _ mkdir _ write _ enable=NO), or otherwise write to the server (anon _ other _ write _ enable=NO). Here are the Security settings:

```
# Security
anon_world_readable_only=YES
connect_from_port_20=YES
hide_ids=YES
pasv_min_port=50000
pasv_max_port=60000
```

Because the vsftpd daemon can read files assigned to the ftp user and group, setting anon _ world _ readable _ only=YES ensures that anonymous users can see files where the read permission bit is turned on for other (------r--), but not write files. The connect _ from _ port _ 20=YES setting gives the vsftpd daemon slightly more permission to send data the way a client might request by allowing PORT-style data communications.

Using hide _ ids=YES hides the real permissions set on files, so to the user accessing the FTP site, everything appears to be owned by the ftp user. The two pasv settings restrict the range of ports that can be used with passive FTP (where the server picks a higher number port on which to send data) to between 50000 and 60000.

The next section contains features of the vsftpd server:

```
# Features
xferlog_enable=YES
ls_recurse_enable=NO
ascii_download_enable=NO
async_abor_enable=YES
```

With xferlog _ enable=YES, all file transfers to and from the server are logged to the /var/log/xferlog file. Setting ls _ recurse _ enable=NO prevents users from recursively listing the contents of an FTP directory (in other words, it prevents the type of listing that you could get with the ls -R command) because on a large site, that could drain resources. Disabling ASCII downloads forces all downloads to be in binary mode (preventing files from being translated in ASCII, which is inappropriate for binary files). The async _ abor _ enable=YES setting ensures that some FTP clients, which might hang when aborting a transfer, will not hang.

The following settings have an impact on performance:

```
# Performance
one_process_model=YES
idle_session_timeout=120
data_connection_timeout=300
accept_timeout=60
connect_timeout=60
anon_max_rate=50000
```

With one _ process _ model=YES set, performance can improve because vsftpd launches one process per connection. Reducing the idle _ session _ timeout from the default 300 seconds to 120 seconds causes FTP clients that are idle more than 2 minutes to be disconnected. Thus, less time is spent managing FTP sessions that are no longer in use. If a data transfer stalls for more than data _ connection _ timeout seconds (300 seconds here), the connection to the client is dropped.

The accept _ timeout setting of 60 seconds allows 1 minute for a PASV connection to be accepted by the remote client. The connect _ timeout sets how long a remote client has to respond to a request to establish a PORT-style data connection. Limiting the transfer rate to 50000 (bytes per second) with anon _ max _ rate can improve overall performance of the server by limiting how much bandwidth each client can consume.

Using FTP Clients to Connect to Your Server

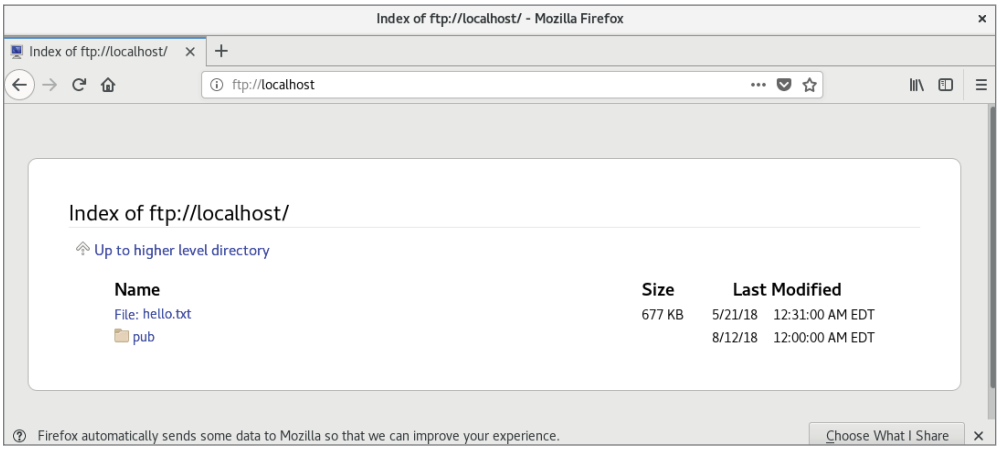
Many client programs come with Linux, which you can use to connect to your FTP server. If you simply want to do an anonymous download of some files from an FTP server, your Firefox web browser provides an easy interface to do that. For more complex interactions between your FTP client and server, you can use command-line FTP clients. The following sections describe some of these tools.

Accessing an FTP server from Firefox

The Firefox web browser provides a quick and easy way to test access to your FTP server or to access any public FTP server. On your own system, type **ftp://localhost** into the location box. You are prompted to log in, which you can do as a regular user or the anonymous user if your server is accessible via anonymous FTP. As the anonymous user, you should see something similar to the example shown in Figure 18.2.

FIGURE 18.2

Accessing an FTP server from Firefox



To log in to an FTP server as a particular user from Firefox, you can precede the host name with a `username:password@` notation, as shown in the following example:

```
ftp://chris:MypassWd5@localhost
```

If you provide the correct username and password, you should immediately see the contents of your home directory. Click a folder to open it. Click a file to download or view the file.

Accessing an FTP server with the lftp command

To test your FTP server from the command line, you can use the `lftp` command. To install the `lftp` command in Fedora or RHEL, enter the following from the command line:

```
# yum install lftp
```

If you use the `lftp` command with just the name of the FTP server you are trying to access, the command tries to connect to the FTP server as the anonymous user. By adding the `-u` *`username`*, you can enter the user's password when prompted and gain access to the FTP server as the user you logged in as.

After you have added your user and password information, you get an `lftp` prompt, ready for you to start typing commands. The connection is made to the server when you type your first command. You can use the commands to move around the FTP server and then use the `get` and `put` commands to download and upload files.

The following example shows how to use commands as just described. It assumes that the FTP server (and associated security measures) has been configured to allow local users to connect and to read and write files:

```
# lftp -u chris localhost
Password:
********
lftp chris@localhost:~> pwd
ftp://chris@localhost/%2Fhome/chris
lftp chris@localhost:~> cd stuff/state/
lftp chris@localhost:~/stuff/state> ls
-rw-r--r--    1 13597    13597        1394 Oct 23  2014
enrolled-20141012
-rw-r--r--    1 13597    13597         514 Oct 23  2014
enrolled-20141013
lftp chris@localhost:~/stuff/state> !pwd
/root
lftp chris@localhost:~/stuff/state> get survey-20141023.txt
3108 bytes transferred
lftp chris@localhost:~/stuff/state> put /etc/hosts
201 bytes transferred
lftp chris@localhost:~/stuff/state> ls
-rw-r--r--    1 13597    13597        1394 Oct 23  2014
enrolled-20141012
-rw-r--r--    1 13597    13597         514 Oct 23  2014
enrolled-20141013
-rw-r--r--    1 0        0             201 May 03 20:22 hosts
lftp chris@localhost:~/stuff/state> !ls
anaconda-ks.cfg          bin           install.log
dog                      Pictures      sent
Downloads                Public        survey-20141023.txt
lftp chris@localhost:~/stuff/state> quit
```

After providing the username (`-u chris`), `lftp` prompts for chris's Linux user password. Typing `pwd` shows that chris is logged in to the local host and that `/home/chris` is the current directory. Just as you would from a regular Linux command-line shell, you can use `cd` to change to another directory and `ls` to list that directory's contents.

To have the commands you run interpreted by the client system, you can simply put an exclamation mark (`!`) in front of a command. For example, running `!pwd` shows that the current directory on the system that initiated the `lftp` is `/root`. This is good to know because if you get a file from the server without specifying its destination, it goes to the client's current directory (in this case, `/root`). Other commands you might run so that they are interpreted by the client system include `!cd` (to change directories) and `!ls` (to list files).

Assuming that you have read permission for a file on the server and write permission from the current directory on the initiating system, you can use the `get` command to download a file from the server (`get survey-20141023.txt`). If you have write and upload permission on the current directory on the server, you can use `put` to copy a file to the server (`put /etc/hosts`).

Running an `ls` command shows that the `/etc/hosts` file was uploaded to the server. Running the `!ls` command lets you see that the `survey-20141023.txt` file was downloaded from the server to the initiating system.

Using the gFTP client

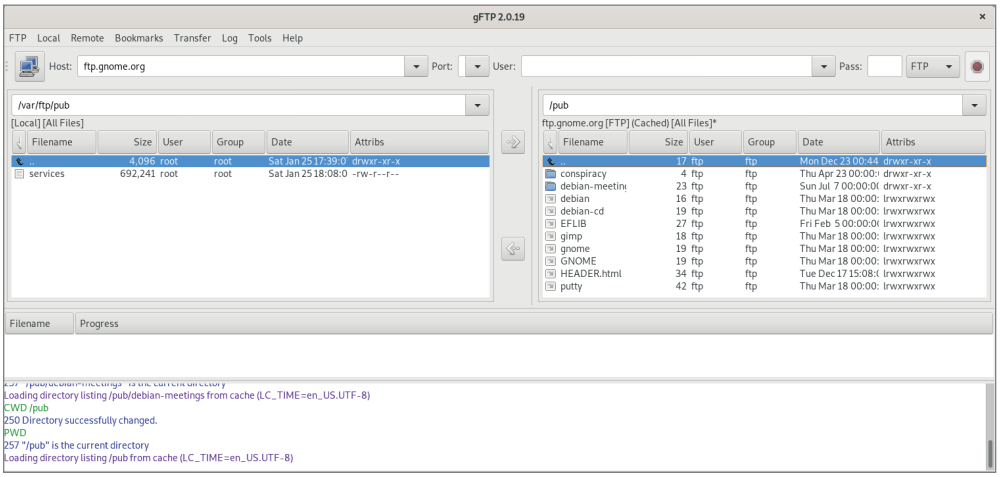
Many other FTP clients are available with Linux as well. Another FTP client that you could try is gFTP. The gFTP client provides an interface that lets you see both the local and remote sides of your FTP session. To install gFTP in Fedora, run the following command to install the `gftp` package:

```
# yum install gftp
```

To start gFTP, launch it from the applications menu or run `gftp &` from the shell. To use it, type the URL of the FTP server to which you wish to connect, enter the username you want to use (such as anonymous), and press Enter. Figure 18.3 shows an example of gFTP being used to connect to the `gnome.org` site: `ftp.gnome.org`.

FIGURE 18.3

The gFTP FTP client lets you see both sides of an FTP session.



To traverse the FTP site from gFTP, just double-click folders (just as you would from a file manager window). The full paths to the local directory (on the left) and remote directory (on the right) are shown above the listings of files and folders below.

To transfer a file from the remote side to the local side, select the file that you want from the right and click the arrow in the middle of the screen pointing to the left. Watch the progress of the file transfer from messages on the bottom of the screen. When the transfer completes, the file appears in the left pane.

You can bookmark the address information that you need to connect to an FTP site. That address is added to a set of bookmarks already stored under the Bookmarks menu. You can select sites from the list to try out the gFTP. Most of the sites are for Linux distributions and other open source software sites.

Summary

Setting up an FTP server is an easy way to share files over a TCP network. The Very Secure FTP Daemon (`vsftpd` package) is available for Fedora, Red Hat Enterprise Linux, Ubuntu, and other Linux systems.

A default `vsftpd` server allows anonymous users to download files from the server and regular Linux users to upload or download files (provided the correct security settings are applied). Moving around on an FTP server is similar to moving around a Linux filesystem. You move up and down the directory structure to find the content that you want.

There are both graphical and text-based FTP clients. A popular text-based client for Linux is `lftp`. As for graphical FTP clients, you can use a regular web browser, such as Firefox, or dedicated FTP clients, such as gFTP.

FTP servers are not the only way to share files over a network from Linux. The Samba service provides a way to share files over a network so that the shared Linux directory looks like a shared directory from a Windows system. Chapter 19, "Configuring a Windows File Sharing (Samba) Server," describes how to use Samba to offer Windows-style file sharing.

Exercises

The exercises in this section describe tasks related to setting up an FTP server in RHEL or Fedora and connecting to that server using an FTP client. If you are stuck, solutions to the tasks are shown in Appendix B. Keep in mind that the solutions shown in Appendix B are usually just one of multiple ways to complete a task.

Don't do these exercises on a Linux system running a public FTP server because they almost certainly will interfere with that server.

1. Determine which package provides the Very Secure FTP Daemon service.
2. Install the Very Secure FTP Daemon package on your system, and search for the configuration files in that package.

3. Enable anonymous FTP and disable local user login for the Very Secure FTP Daemon service.
4. Start the Very Secure FTP Daemon service and set it to start when the system boots.
5. On the system running your FTP server, create a file named `test` in the anonymous FTP directory that contains the words "Welcome to your vsftpd server."
6. From a web browser on the system running your FTP server, open the `test` file from the anonymous FTP home directory. Be sure that you can see that file's contents.
7. From a web browser outside of the system that is running the FTP server, try to access the `test` file in the anonymous FTP home directory. If you cannot access the file, check that your firewall, SELinux, and TCP wrappers are configured to allow access to that file.
8. Configure your `vsftpd` server to allow file uploads by anonymous users to a directory named `in`.
9. Install the `lftp` FTP client (if you don't have a second Linux system, install `lftp` on the same host running the FTP server). If you cannot upload files to the `in` directory, check that your firewall, SELinux, and TCP wrappers are configured to allow access to that file.
10. Using any FTP client you choose, visit the `/pub/debian-meetings` directory on the `ftp.gnome.org` site and list the contents of that directory.

CHAPTER 19

Configuring a Windows File Sharing (Samba) Server

IN THIS CHAPTER

Getting and installing Samba

Using Samba security features

Editing the `smb.conf` configuration file

Accessing Samba from Linux and Windows clients

Using Samba in the enterprise

Samba is the project that implements open source versions of protocols used to share files and printers among Windows systems as well as authenticate users and restrict hosts. Samba offers a number of ways to share files among Windows, Linux, and MacOS systems that are well known and readily available to users of those systems.

This chapter steps you through the process of installing and configuring a Samba server. It describes the security features that you need to know to share your file and printer resources and describes how to access those resources from Linux and Windows systems.

Understanding Samba

Samba (`https://samba.org`) is a suite of programs that allows Linux, UNIX, and other systems to interoperate with Microsoft Windows file and printer sharing protocols. Windows, MacOS, and other client systems can access Samba servers to share files and printers in the same ways that they would from Windows file and print servers.

With Samba, you can use standard TCP/IP networking to communicate with clients. For name service, Samba supports regular TCP/IP hostnames as well as NetBIOS names. For that reason, Samba doesn't require the NetBEUI (Microsoft Raw NetBIOS frame) protocol. File sharing is done using Server Message Block (SMB) protocol, which is sometimes referred to as the *Common Internet File System (CIFS)*.

The Samba project has gone to great lengths to make its software secure and robust. In fact, many people prefer using Samba servers over Windows file servers because of the added security that is inherent in running Windows-style file sharing services on Linux or other UNIX-like operating systems.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Beyond all of the technical mumbo-jumbo, however, the end result is that Samba makes it easy to share files and printers between Linux servers and Windows desktop systems. For the server, only a few configuration files and tools are needed to manage Samba. For the clients, shared resources just show up under Network selection in the File Explorer (formerly Windows Explorer) application or in the Network Neighborhood on older Windows systems.

To configure the Samba service, you directly edit Samba configuration files (particularly `smb.conf`) and run a few commands. Graphical and web-based interfaces, such as `system-config-samba` and Samba SWAT, are no longer included with the latest Fedora and RHEL systems.

To begin using Samba on your Linux system, you need to install a few software packages, as described in the next section.

Installing Samba

In Red Hat Enterprise Linux and Fedora, to configure a Samba file and print server, installing the `samba` package gets you everything you need to start. Among other components, the `samba` package includes the Samba service daemon (`/usr/sbin/smbd`) and NetBIOS name server daemon (`/usr/sbin/nmbd`). Installing the `samba` package pulls in the `samba-common` package, which contains server configuration files (`smb.conf`, `lmhosts`, and others) and commands for adding passwords and testing configuration files, along with other Samba features.

Features from other packages are referenced in this chapter, so I describe how to install those packages as well. Those packages include the following:

- **`samba-client` package**: Contains command-line tools such as `smbclient` (for connecting to Samba or Windows shares), `nmblookup` (for looking up host addresses), and `findsmb` (to find SMB hosts on the network).
- **`samba-winbind` package**: Includes components that allow your Samba server in Linux to become a complete member of a Windows domain, including using Windows user and group accounts in Linux.

To install all the packages just mentioned (`samba-common` is installed as a dependency of `samba`, so it doesn't need to be noted specifically), enter the following as root from the command line in Fedora or RHEL:

```
# yum install samba samba-client samba-winbind
...
Last metadata expiration check: 0:01:44 ago on Sun 24 Jan 2020
11:35:37 AM EST.
Dependencies resolved.
```

```
================================================================================
 Package       Architecture     Version  Repository
Size
================================================================================
Installing:
 samba         x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 739 k
 samba-winbind x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 570 k
Installing dependencies:
 samba-common-tools
               x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 469 k
 samba-libs    x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 185 k
 samba-winbind-modules
               x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 122 k
 samba-client  x86_64  4.10.4-101.el8_1  rhel-8-for-x86_64-baseos-
rpms 658 k

Transaction Summary
================================================================================
Install  6 Packages

Total download size: 2.5 M
Installed size: 6.8 M
Is this ok [y/d/N]: y
```

After you have installed the Samba packages, look at the configuration files in the `samba-common` package:

```
# rpm -qc samba-common
/etc/logrotate.d/samba
/etc/samba/lmhosts
/etc/samba/smb.conf
/etc/sysconfig/samba
```

The `/etc/logrotate.d/samba` and `/etc/sysconfig/samba` files are usually not modified. The first sets how files in `/var/log/samba` log files are rotated (copied to other files and removed) over time. The second is a file where you could put options that are passed to the `smbd`, `nmbd`, or `winbindd` daemon, so you could turn off features such as debugging.

Most configuration files that you would modify for Samba are in the `/etc/samba` directory. The `smb.conf` file is the primary configuration file where you put global settings for the Samba server as well as individual file and printer share information (more on that later). The `lmhosts` file enables the Samba NetBIOS hostname to be mapped into IP addresses.

Although it doesn't exist by default, you can create a file named /etc/samba/smbusers to map Linux usernames into Windows usernames. As you configure your Samba server, you can refer to the smb.conf man page (man smb.conf). There are also man pages for Samba commands, such as smbpasswd (to change passwords), smbclient (to connect to a Samba server), and nmblookup (to look up NetBIOS information).

After you have installed Samba packages and completed a quick survey of what they contain, try starting up the Samba service and see what you get in a default configuration.

Starting and Stopping Samba

With samba and samba-common installed, you can start the server and investigate how it runs in the default configuration. Two main services are associated with a Samba server, each of which has its own service daemon:

smb: This service controls the smbd daemon process, which provides the file and print sharing services that can be accessed by Windows clients.

nmb: This service controls the nmbd daemon. By providing NetBIOS name service name-to-address mapping, nmbd can map requests from Windows clients for NetBIOS names so that they can be resolved into IP addresses.

To share files and printers with other Linux systems with Samba, only the smb service is required. The next section describes how to start and enable the smb service.

Starting the Samba (smb) service

The smb service is what starts the smbd server and makes files and printers available from your local system to other computers on the network. As usual, services are enabled and started differently on different Linux systems. For different Linux systems, you need to find the name of the service and the correct tool to start the smbd daemon.

In Fedora and RHEL, to enable Samba to start immediately when the system boots, enter the following from the command line as root:

```
# systemctl enable smb.service
# systemctl start smb.service
# systemctl status smb.service
smb.service - Samba SMB Daemon
  Loaded: loaded (/usr/lib/systemd/system/smb.service; enabled)
  Active: active (running) since Fri 2020-01-31 07:23:37 EDT; 6s ago
     Docs: man:smbd(8)
           man:samba(7)
           man:smb.conf(5)
   Status: "smbd: ready to serve connections..."
    Tasks: 4 (limit: 12216)
   Memory: 20.7M
 Main PID: 4838 (smbd)
CGroup: /system.slice/smb.service
```

```
   ├ 4838 /usr/sbin/smbd --foreground --no-process-group
   └ 4840 /usr/sbin/smbd --foreground --no-process-group
```

The first `systemctl` command enables the service, the second starts it immediately, and the third shows the status. To investigate further, notice that the service file is located at `/usr/lib/systemd/system/smb.service`. Look at the contents of that file:

```
# cat /usr/lib/systemd/system/smb.service
[Unit]
Description=Samba SMB Daemon
Documentation=man:smbd(8) man:samba(7) man:smb.conf(5)
Wants=network-online.target
After=network.target network-online.target nmb.service winbind.
service
[Service]
Type=notify
NotifyAccess=all
PIDFile=/run/smbd.pid
LimitNOFILE=16384
EnvironmentFile=-/etc/sysconfig/samba
ExecStart=/usr/sbin/smbd --foreground --no-process-group $SMBDOPTIONS
ExecReload=/bin/kill -HUP $MAINPID
LimitCORE=infinity
Environment=KRB5CCNAME=FILE:/run/samba/krb5cc_samba
[Install]
WantedBy=multi-user.target
```

The Samba daemon process (`smbd`) starts up after the `network`, `network-online`, `nmb`, and `winbind` targets. The `/etc/sysconfig/samba` file contains variables that are passed as arguments to the `smbd`, `nmbd`, and `winbindd` daemons when they start. No options are set by default for any of those daemons. The `WantedBy` line indicates that `smb.service` should start when the system boots up into multi-user mode (`multi-user.target`), which it does by default.

In RHEL 6 and earlier, you can start the Samba service as follows:

```
# service smb start
Starting SMB services:        [  OK  ]
# chkconfig smb on
# service smb status
smbd (pid  28056) is running...
# chkconfig --list smb
smb             0:off  1:off  2:on  3:on  4:on  5:on  6:off
```

Whether you are running your Samba server on RHEL, Fedora, or another Linux system, you can check access to the Samba server using the `smbclient` command (from the `samba-client` package). You can get basic information from a Samba server using the following command:

```
# smbclient -L localhost
Enter SAMBA\root's password: <ENTER>
```

```
Anonymous login successful

   Sharename  Type    Comment
   ---------  ----    -------
   print$     Disk    Printer Drivers
   IPC$       IPC     IPC Service
  (Samba Server Version 4.10.10)
   deskjet    Printer deskjet
Reconnecting with SMB1 for workgroup listing.
Anonymous login successful
    Server               Comment
    ---------            -------
    Workgroup            Master
    ---------            -------
```

The `smbclient` output allows you to see what services are available from the server. By default, anonymous login is allowed when querying the server (so I just pressed Enter when prompted for a password).

You can discern a number of things about the default Samba server setup from this output:

- All printers that are shared via the CUPS server on your Linux system are, by default, also made available from the Samba server running on that same system.
- No directories are shared yet from the server.
- There is no NetBIOS name service running yet from the Samba server.

Next, you can decide whether you want to run the NetBIOS name service on your Samba server.

Starting the NetBIOS (nmbd) name server

If no Windows domain server is running on the network, as is the case here, you can start the `nmb` service on the Samba host to provide that service. To start the `nmb` service (`nmbd` daemon) in Fedora or RHEL 7, type the following:

```
# systemctl enable nmb.service
# systemctl start nmb.service
# systemctl status nmb.service
```

In RHEL 6 and earlier, you would type the following to start the `nmb` service:

```
# service nmb start
# service nmb status
# chkconfig nmb on
# chkconfig --list nmb
```

Regardless of how the NetBIOS service was started, the `nmbd` daemon should now be running and ready to serve NetBIOS name-to-address mapping. Run the `smbclient -L` command again, followed by the IP address of the server. This time, the last few lines of

the output should show the information obtained from the NetBIOS server now running on the Samba server. In this case, the last few lines look like this:

```
# smbclient -L localhost
    ...
    Workgroup   Master
    ---------   -------
    SAMBA       FEDORA30
```

You can see that the new NetBIOS server's name is `FEDORA30` and that it is the master server for the workgroup. To query the `nmbd` server for the IP address of `FEDORA30`, you would enter the following:

```
# nmblookup -U localhost FEDORA30
querying FEDORA30 on 127.0.0.1
192.168.122.81 FEDORA30<00>
```

You should be able to see your Samba server running from the local system now. The hostname assigned to the system (in this case `FEDORA30`) is assigned by default.

However, if you have a firewall configured or SELinux enabled, you may not be able to access the Samba server fully from a remote system yet. The next section should help you to open Samba to systems outside of the local system as well as to allow some Samba features that may be turned off by SELinux.

Stopping the Samba (smb) and NetBIOS (nmb) services

To stop the `smb` and `nmb` services in Fedora or RHEL, you can use the same `systemctl` command that you used to start them. You can use the same command to disable the services as well so that they do not start up again when the system boots. Here are examples of how to stop the `smb` and `nmb` services immediately:

```
# systemctl stop smb.service
# systemctl stop nmb.service
```

In RHEL 6 and earlier, you would enter the following to stop the `smb` and `nmb` services:

```
# service smb stop
# service nmb stop
```

To prevent the `smb` and `nmb` services from starting the next time the system reboots, enter the following commands in Fedora or RHEL:

```
# systemctl disable smb.service
# systemctl disable nmb.service
```

In Red Hat Enterprise Linux 6 and earlier, enter the following commands to disable the `smb` and `nmb` services:

```
# chkconfig smb off
# chkconfig nmb off
```

Of course, you only want to stop or disable the `smb` and `nmb` services if you no longer want to use the Samba service. If you are ready to continue to configure your Samba service, you can continue on and begin to configure your Linux security features to allow the Samba service to become available to others on your network.

Securing Samba

If you cannot access your Samba server immediately after starting it, you probably have some security work to do. Because many default installations of Linux prevent, rather than allow, access to the system, dealing with security for a service such as Samba usually has more to do with making it available than making it secure.

Here are the security features that you should be aware of when configuring your Samba system:

Firewalls The default firewall for Fedora, RHEL, and other Linux systems prevents any access to local services from outside systems. So, to allow users from other computers to access your Samba service, you must create firewall rules that open one or more ports for selected protocols (TCP in particular).

SELinux Many features of Samba are designated as potentially insecure by SELinux. Because the default SELinux Booleans (on/off switches for certain features) are set to provide the least access required, you need to turn Booleans on for features such as allowing users to access their own home directories with Samba. In other words, you can configure Samba to share user home directories, but SELinux prohibits someone from trying to use that feature unless you explicitly configure SELinux to allow that feature.

Host and user restrictions Within the Samba configuration files themselves, you can indicate which hosts and users can have access to the Samba server as a whole or to particular shared directories.

The next sections describe how to set up the security features just mentioned for Samba.

Configuring firewalls for Samba

If an `iptables` or `firewalld` firewall is configured for your system when you first install it, the firewall typically allows any requests for services from local users but none by outside users. That's why, at the end of the installation section of this chapter, you should have been able to test that Samba was working using the `smbclient` command from the local system. However, if the request originated from another system, it would have been rejected.

Configuring firewall rules for Samba mainly consists of opening up incoming ports on which the `smbd` and `nmbd` daemons are listening. These are the ports that you should open to get a working Samba service on your Linux system:

TCP port 445: This is the primary port on which the Samba `smbd` daemon listens. Your firewall must support incoming packet requests on this port for Samba to work.

TCP port 139: The `smbd` daemon also listens on TCP port 139 in order to handle sessions associated with NetBIOS hostnames. It is possible to use Samba over TCP without opening this port, but it is not recommended.

UDP ports 137 and 138: The `nmbd` daemon uses these two ports for incoming NetBIOS requests. If you are using the `nmbd` daemon, these two ports must be open for new packet requests for NetBIOS name resolution.

For Fedora and RHEL, allowing incoming access to those four ports is easy. Simply open the Firewall Configuration window, and select the check boxes next to the `samba` and `samba-client` entries on the public zone, Services tab. Those ports become immediately accessible (no restart of the `firewalld` service is required).

For earlier Fedora and RHEL systems that use `iptables` directly instead of the `firewalld` service, opening the firewall is a more manual process. Consider a default firewall from Fedora that allows incoming packets from the local host, from established connections, and related to established connections but denies all other incoming packets. The following example represents a set of firewall rules in the `/etc/sysconfig/iptables` file, with four new rules (highlighted in the example that follows) added to open ports for Samba:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-I INPUT -m state --state NEW -m udp -p udp --dport 137 -j ACCEPT
-I INPUT -m state --state NEW -m udp -p udp --dport 138 -j ACCEPT
-I INPUT -m state --state NEW -m tcp -p tcp --dport 139 -j ACCEPT
-I INPUT -m state --state NEW -m tcp -p tcp --dport 445 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Your firewall may include additional rules to allow incoming packet requests for other services, such as Secure Shell (`sshd`) or web (`httpd`) services. You can leave those in place. The main point is to have your Samba rules placed somewhere before the final `REJECT` rules.

If your `iptables` firewall is enabled, you can restart it to have the new rules take effect. To do that, type **`systemctl restart iptables.service`** (in older Fedora systems) or **`service restart iptables`** (in RHEL 6 or earlier). Try connecting to the Samba service by using the `smbclient` command again, or by using other techniques described in the section "Accessing Samba Shares" later in this chapter.

See Chapter 25, "Securing Linux on a Network," for more information on using `iptables`.

Configuring SELinux for Samba

There are both file context and Boolean considerations related to using Samba with SELinux in enforcing mode. File contexts must be properly set on a directory that is shared by Samba. Booleans allow you to override the secure-by-default approach to certain Samba features.

You can find information on how SELinux confines Samba on the `samba_selinux` man page (`man samba_selinux`). You must install the `selinux-policy-doc` package to get that man page. For a deeper understanding of SELinux, refer to Chapter 24, "Enhancing Linux Security with SELinux."

Setting SELinux Booleans for Samba

An easy way to list and change SELinux Booleans for Samba is from the command line. To use the `semanage` command to list Samba-related Booleans, enter the following:

```
# semanage boolean -l | egrep "smb|samba"
```

The following is a list of SELinux Booleans that apply to Samba and their descriptions. Most of the Booleans let you set which files and directories the Samba server can read and write on behalf of Samba users. Others let you allow potentially insecure features:

samba_run_unconfined: Allows samba to run unconfined scripts from Samba shares.

smbd_anon_write: Allows Samba to let anonymous users modify public files used for public file transfer services. Files and directories must be labeled `public_content_rw_t`.

samba_enable_home_dirs: Allows Samba to share users' home directories.

samba_export_all_ro: Allows Samba to share any file and directory read-only.

use_samba_home_dirs: Allows a remote Samba server to access home directories on the local machine.

samba_create_home_dirs: Allows Samba to create new home directories (for example, via PAM).

samba_export_all_rw: Allows Samba to share any file or directory read/write.

The following Booleans affect Samba's ability to share directories that are themselves mounted from other remote services (such as NFS) or to act as a Windows domain controller:

samba_share_fusefs: Allows Samba to export `ntfs/fusefs` volumes.

samba_share_nfs: Allows Samba to export NFS volumes.

samba_domain_controller: Allows Samba to act as the domain controller, add users and groups, and change passwords.

The `setsebool` command is used to turn the SELinux Booleans on or off. Used with the `-P` option, `setsebool` sets the Boolean you indicate permanently. For example, to allow

Samba to share any file or directory with read-only permission from the server, you could type the following from a shell as root user:

```
# setsebool -P samba_export_all_ro on
# getsebool samba_export_all_ro
samba_export_all_ro --> on
```

The `setsebool` command sets the Boolean in this case to `on`. The `getsebool` lets you see the value of the Boolean.

Setting SELinux file contexts for Samba

SELinux confines the files that the Samba service can access. Instead of allowing any file with the proper read and write permission to be shared by the Samba server, SELinux (when in enforcing mode) requires that files and directories have the correct file contexts set on them before the Samba service can even see that the files exist.

In order for the Samba service to function with SELinux immediately, some files and directories come preset with the proper file contexts. For example, Samba configuration files (`/etc/samba/*`), log files (`/var/log/samba/*`), and libraries (`/var/lib/samba/*`) have rules assigned to ensure that they get the proper file contexts. To find files and directories associated with the Samba service and `smbd` daemon that have file contexts preset, run the following:

```
# semanage fcontext -l | grep -i samba
# semanage fcontext -l | grep -i smb
```

The file context portion in which you are interested ends with `_t`: for example, `samba_etc_t`, `samba_log_t`, and `samba_var_t` for the `/etc/samba`, `/var/log/samba`, and `/var/lib/samba` directories, respectively.

You may find that you need to change file contexts—for example, when you put files in nonstandard locations (such as moving the `smb.conf` file to `/root/smb.conf`) or when you want to share a directory (other than home directories, which can be turned on by setting a Boolean). Unlike the `vsftpd` (FTP) and `httpd` (web) servers that come with Linux, Samba has no default shared content directories (those just mentioned used `/var/ftp` and `/var/www/html`).

You can change a file context permanently by creating a new file context rule and then applying that rule to the file or directory for which it is intended. You can do that with the `semanage` command (to make the rule) and `restorecon` command (to apply the rule). For example, if you wanted to share a directory, `/mystuff`, you would create that directory with the proper permissions and run the following command to make it available for read/write access from Samba:

```
# semanage fcontext -a -t samba_share_t "/mystuff(/.*)?"
# restorecon -v /mystuff
```

After those commands are run, the `/mystuff` directory, along with any files and directories below that point, have the file context of `samba_share_t`. It is then up to you

to assign the correct Linux ownership and file permissions to allow access to the users you choose. The upcoming section "Configuring Samba" provides an example of creating a share, and it shows you how to add permissions and ownership to a shared directory using standard Linux commands.

Configuring Samba host/user permissions

Within the `smb.conf` file itself, you can allow or restrict access to the entire Samba server or to specific shares based on the hosts or users trying to gain access. You can also restrict access to the Samba server by providing the service only on particular interfaces.

For example, if you have one network interface card connected to the Internet and another connected to the local network, you can tell Samba to serve requests only on the local network interface. The next section describes how to configure Samba, including how to identify which hosts, users, or network interfaces can access your Samba server.

Configuring Samba

Inside the `/etc/samba/smb.conf` file are settings for configuring your Samba server, defining shared printers, configuring how authentication is done, and creating shared directories. The file consists of the following predefined sections:

`[global]` Settings that apply to the Samba server as a whole are placed in this section. This is where you set the server's description, its workgroup (domain), the location of log files, the default type of security, and other settings.

`[homes]` This section determines whether users with accounts on the Samba server can see their home directories (browseable) or write to them.

`[printers]` In this section, settings tell Samba whether to make printers available through Samba that are configured for Linux printing (CUPS).

`[print$]` This section configures a directory as a shared printer drivers folder.

Inside the `smb.conf` file, lines beginning with pound signs (#) or semicolons (;) are comments. Removing the semicolons enables you to set up different kinds of shared information quickly. The # sign can also be used to comment out a line.

When you begin editing your `smb.conf` file, make a backup that you can go back to if something goes wrong. You can start by copying the `smb.conf.example` file to `smb.conf`, if you want to start with some examples.

Configuring the [global] section

Here is an example of a `[global]` section of the `smb.conf` file:

```
[global]
        workgroup = SAMBA
        security = user
```

```
        passdb backend = tdbsam
        printing = cups
        printcap name = cups
        load printers = yes
        cups options = raw

;       netbios name = MYSERVER
;       interfaces = lo eth0 192.168.12.2/24 192.168.13.2/24
;       hosts allow = 127. 192.168.12. 192.168.13.
```

The `workgroup` (also used as the domain name) is set to `SAMBA` in this example. When a client communicates with the Samba server, this name tells the client which workgroup the Samba server is in.

The default `security` type is set to `user` (Samba usernames and passwords).

The `passdb backend = tdbsam` specifies to use a Samba backend database to hold passwords. You can use the `smbpasswd` command to set each user's password (as described later).

Setting `printing = cups` and `printcap name = cups` indicates to use the `printcap` created by the CUPS printing service. When you set `load printers = yes`, Samba knows to share any printers configured by your local CUPS printing service from Samba.

The `cups options` lets you pass any options that you like to the CUPS printers served by your Samba server. By default, only `raw` is set, which allows Windows clients to use their own print drivers. Printers on your Samba server print the pages they are presented in raw form.

By default, your server's DNS hostname (enter `hostname` to see what it is) is used as your Samba server's NetBIOS name as well. You can override that and set a separate NetBIOS name by uncommenting the `netbios name` line and adding the server name you want. For example, `netbios name = myownhost`. `localhost` is used as your NetBIOS name if it has not otherwise been set.

If you want to restrict access to the Samba server so that it only responds on certain interfaces, you can uncomment the `interfaces` line and add either the IP address or name (`lo`, `eth0`, `eth1`, and so on) of the network interfaces you want.

You can restrict access to the Samba server to specific hosts as well. Uncomment the `hosts allow` line (remove the semicolon) and insert the IP addresses of the hosts that you want to allow. To enter a range of addresses, simply end the subnetwork portion of the address, followed by a dot. For example, `127.` is associated with IP addresses that point to the local host. The 192.168.12. entry matches all IP addresses from 192.168.12.1 to 192.168.12.254.

Configuring the [homes] section

The `[homes]` section is configured, by default, to allow any Samba user account to be able to access its own home directory via the Samba server. Here is what the default homes entry looks like:

```
[homes]
        comment = Home Directories
        valid users = %S, %D%w%S
        browseable = No
        read only = No
        inherit acls = Yes
```

Setting `valid users` to `%S` substitutes the current service name, which allows any valid users of the service to access their home directories. The valid users are also identified by domain or workgroup (`%D`), winbind separator (`%w`), and name of current service (`%S`).

The `browseable = No` setting prevents the Samba server from displaying the availability of the shared home directories. Users who can provide their own Samba usernames and passwords can read and write in their own home directories (`read only = no`). With `inherit acls` set to `Yes`, access control lists can be inherited to add another layer of security on the shared files.

If after starting the `smb` service you cannot log in using a valid user account, you may need to change some security features on your system. On Fedora and RHEL systems, in particular, SELinux features need to be changed to allow users to access their home directories if you are in SELinux enforcing mode.

For example, if you tried to use `smbclient` to log in to your home directory, the login would succeed, but when you tried to list the contents of the home directory, you might see the following message:

```
NT_STATUS_ACCESS_DENIED listing \*
```

To tell SELinux to allow Samba users to access their home directories as Samba shares, turn on the `samba _ enable _ home _ dirs` Boolean by entering the following as root from a shell:

```
# setsebool -P samba_enable_home_dirs on
```

The `setsebool` command turns on the capability of Samba to share home directories (which is off by default). First create a password for the user with `smbpasswd` and then log in with `smbclient`. The form for using the `smbclient` command to check access to the user's home directory, again for the user `chris`, would be the following (replacing the IP address with the name or address of your Samba server):

```
$ smbpasswd -a chris
New SMB password: *********
Retype new SMB password: *********
Added user chris.

$ smbclient -U chris //192.168.0.119/chris

Enter SAMBA\chris's password:
Try "help" to get a list of possible commands.
smb: \> ls file.txt
  file.txt 149946368  Sun Jan  4 09:28:53 2020
          39941 blocks of size 524288. 28191 blocks available
smb:\>  quit
```

The main point to remember is that, even though the share is not browseable, you can request it by giving the Samba server's hostname or IP address, followed by the user's name (here, `chris`), to access the user's home directory.

Configuring the [printers] section

Any printer that you configure for CUPS printing on your Linux system is automatically shared to others over Samba, based on the `[printers]` section that is added by default. The global `cups options = raw` setting makes all printers raw printers (meaning that the Windows client needs to provide the proper printer driver for each shared printer).

Here's what the default printers section looks like in the `smb.conf` file:

```
[printers]
        comment = All Printers
        path = /var/tmp
        printable = Yes
        create mask = 0600
        browseable = No
```

The `path` tells Samba to store temporary print files in `/var/tmp`. The `printable = Yes` line causes all of your CUPS printers on the local system to be shared by Samba. Printers are writeable and allow guest printing by default. The `create mask = 0600` setting used here has the effect of removing write and execute bits for group and other, within the ACL, when files are created in the path directory.

To see that local printers are available, you could run the `smbclient -L` command from a Linux system, as shown earlier. On a Windows system, you can select Network from the File Explorer window and select the icon representing your Samba server. All shared printers and folders appear in that window. (See the section "Accessing Samba Shares" later in this chapter for details on viewing and using shared printers.)

Creating a Samba shared folder

Before you can create a shared folder, that folder (directory) must exist and have the proper permissions set. In this example, the `/var/salesdata` directory is shared. You want the data to be writeable by the user named `chris` but visible to anyone on your network. To create that directory and set the proper permissions and SELinux file contexts, type the following as root user:

```
# mkdir /var/salesdata
# chmod 775 /var/salesdata
# chown chris:chris /var/salesdata
# semanage fcontext -a -t samba_share_t /var/salesdata
# restorecon -v /var/salesdata
# touch /var/salesdata/test
# ls -lZ /var/salesdata/test
-rw-r--r--. 1 root root
    unconfined_u:object_r:samba_share_t:s0 0 Dec 24 14:35
        /var/salesdata/test
```

Adding the shared folder to Samba

With the /var/salesdata directory created and properly configured to be shared by Samba, here is what the shared folder (called salesdata) might look like in the smb.conf file:

```
[salesdata]
        comment = Sales data for current year
        path = /var/salesdata
        read only = no
;       browseable = yes
        valid users = chris
```

Before this share was created, the /var/salesdata directory was created, with chris assigned as the user and group, and the directory was set to be readable and writeable by chris. (The SELinux file context must also be set if SELinux is in enforcing mode.) The Samba username chris must be presented along with the associated password to access the share. After chris is connected to the share, chris has read and write access to it (read only = no).

Now that you have seen the default settings for Samba and an example of a simple shared directory (folder), read the next few sections to see how to configure shares even further. In particular, the examples demonstrate how to make shares available to particular users, hosts, and network interfaces.

Checking the Samba share

For the changes to your Samba configuration to take effect, you need to restart the smb service. After that is done, check that the Samba share you created is available and that any user you assigned to the share can access it. To do those things, enter the following as root user from a shell on the Samba server:

```
# systemctl restart smb.service
# smbclient -L localhost -U chris
Enter SAMBA\chris's password: *******
    Sharename       Type      Comment
    ---------       ----      -------
    salesdata       Disk      Sales data for current year
    print$          Disk      Printer Drivers
    IPC$            IPC       IPC Service (Samba 4.10.4)
    chris           Disk      Home Directories
Reconnecting with SMB1 for workgroup listing.
    Server               Comment
    ---------            -------

    Workgroup            Master
    ---------            -------
    SAMBA                FEDORA30
...
```

Here you can see the share name (salesdata), the domain set to the workgroup name SAMBA, and the description entered earlier (Sales data for current year). Next, a quick way to test access to the share is to use the smbclient command. You can use the hostname or IP address with smbclient to access the share. Because I am on the local system in this example, I just use the name localhost and the user I added (chris):

```
# smbclient -U chris //localhost/salesdata
Enter SAMBA\chris's password: ********
Try "help" to get a list of possible commands.
smb: \> lcd /etc
smb: \> put hosts
putting file hosts as \hosts (43.5 kb/s) (average 43.5 kb/s)
smb: \> ls
  .                                   D        0  Sun Dec 29 09:52:51 2020
  ..                                  D        0  Sun Dec 29 09:11:50 2020
  hosts                               A       89  Sun Dec 29 09:52:51 2020
          39941 blocks of size 524288. 28197 blocks available
smb: \> quit
```

A Samba share is in the form //host/share or \\host\share. However, when you identify a Samba share from a Linux shell in the latter case, the backslashes need to be escaped. So, as an argument, the first example of the share would have to appear as \\\\localhost\\salesdata. Thus, the first form is easier to use.

Note

Escaping a character that you type from the shell is done by putting a backslash (\) in front of that character. It tells the shell to use the character following the backslash literally, instead of giving the character a special meaning to the shell. (The * and ? characters are examples of characters with special meaning.) Because the backslash itself has special meaning to the shell, if you want to use a backslash literally, you need to precede it with a backslash. That is why when you want to type a Samba address that includes two backslashes, you actually have to enter four backslashes.

When prompted, enter the Samba password for that user (it may be different from the Linux user's password). The Samba user's password was set earlier with smbpasswd in this example. You see the smb: \> prompt after that.

At this point, you have a session open to the Samba host that is similar to an ftp session for traversing an FTP server. The lcd /etc command makes /etc the current directory on the local system. The put hosts command uploads the hosts file from the local system to the shared directory. Typing ls shows that the file exists on the server. The quit command ends the session.

Restricting Samba access by network interface

To restrict access to all of your shares, you can set the global interfaces setting in the smb.conf file. Samba is designed more for local file sharing than for sharing over wide area networks. If your computer has a network interface connected to a local network and one connected to the Internet, consider allowing access only to the local network.

To set which interfaces Samba listens on, uncomment the `interfaces` line shown in an earlier example in the `[global]` section of the `smb.conf` file. Then add the interface names or IP address ranges of those computers that you want to allow access to your computer. Here is an example:

```
interfaces = lo 192.168.22.15/24
```

This interfaces entry allows access to the Samba service to all users on the local system (`lo`). It also allows access to any systems on the 192.168.22 network. See the `smb.conf` man page's description of different ways of identifying hosts and network interfaces.

Restricting Samba access by host

Host access to the Samba server can be set for the entire service or for single shares.

Here are some examples of `hosts allow` and `hosts deny` entries:

```
hosts allow = 192.168.22. EXCEPT 192.168.22.99
hosts allow = 192.168.5.0/255.255.255.0
hosts allow = .example.com market.example.net
hosts deny = evil.example.org 192.168.99.
```

These entries can be put in the `[global]` section or in any shared directory section. The first example allows access to any host in the 192.168.22. network except for 192.168.22.99, which is denied. Note that a dot is required at the end of the network number. The 192.168.5.0/255.255.255.0 example uses netmask notation to identify 192.168.5 as the set of addresses that are allowed.

In the third line of the sample code, any host from the `.example.com` network is allowed, as is the individual host `market.example.net`. The `hosts deny` example shows that you can use the same form to identify names and IP addresses in order to prevent access from certain hosts.

Restricting Samba access by user

Particular Samba users and groups can be allowed access to specific Samba shares by identifying those users and groups within a share in the `smb.conf` file. Aside from guest users, which you may or may not allow, the default user authentication for Samba requires you to add a Samba (Windows) user account that maps into a local Linux user account.

To allow a user to access the Samba server, you need to create a password for the user. Here is an example of how to add a Samba password for the user `jim`:

```
# smbpasswd -a jim
New SMB password: *******
Retype new SMB password: *******
```

After running that `smbpasswd` command, `jim` can use that username and password to access the Samba server. The `/var/lib/samba/private/passdb.tdb` file holds the password just entered for `jim`. After that, the user `jim` can change the password by simply typing `smbpasswd` when he is logged in. The root user can change the password by rerunning the command shown in the example but dropping the `-a` option.

If you wanted to give `jim` access to a share, you could add a `valid users` line to that shared block in the `smb.conf` file. For example, to provide both `chris` and `jim` access to a share, you could add the following line:

```
valid users = jim, chris
```

If the `read only` option is set to `no` for the share, both users could potentially write files to the share (depending on file permissions). If `read only` is set to `yes`, you could still allow access to `jim` and `chris` to write files by adding a `write list` line as follows:

```
write list = jim, chris
```

The write list can contain groups (that is, Linux groups contained in the `/etc/group` file) to allow write permission to any Linux user that belongs to a particular Linux group. You can add write permission for a group by putting a plus (+) character in front of a name. For example, the following adds write access for the `market` group to the share with which this line is associated:

```
write list = jim, chris, +market
```

There are many ways to change and extend the features of your shared Samba resources. For further information on configuring Samba, be sure to examine the `smb.conf` file itself (which includes many useful comments) and the `smb.conf` man page.

Accessing Samba Shares

After you have created some shared directories in Samba, many client tools are available in both Linux and Windows for accessing those shares. Command-line tools in Linux include the `smbclient` command, demonstrated earlier in this chapter. For a graphical means of accessing shares, you can use the file managers available in both Windows (File Explorer) and Linux (Nautilus, with the GNOME desktop).

Accessing Samba shares in Linux

Once a Samba share is available, it can be accessed from remote Linux and Windows systems using file managers or remote mount commands.

Accessing Samba shares from a Linux file manager

Opening a file manager in Linux can provide you with access to the shared directories from Linux (Samba) and Windows (SMB). How you access the file manager is different on different Linux desktops. In GNOME 3, you can click the Files icon. In other desktops, open the Home folder.

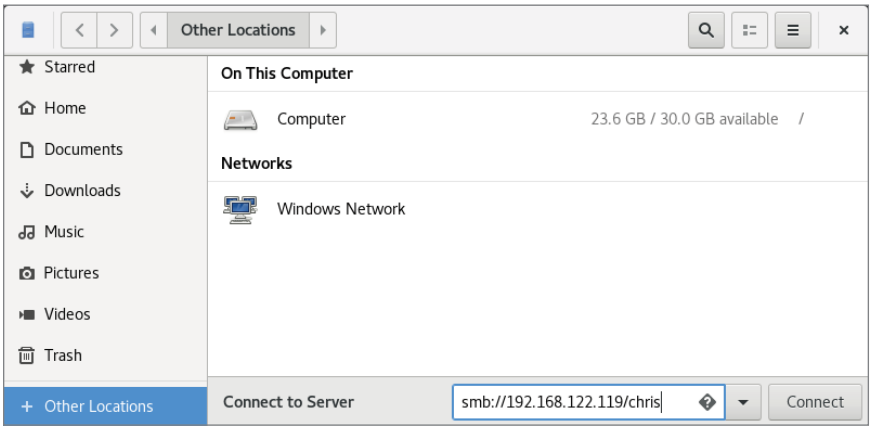
With the Nautilus window manager displayed, select Other Location in the left navigation bar. Available networks (such as Windows Network) should appear. Look to the box at the bottom of the window identified as Connect to Server, and then enter the location of an available Samba share. Given the previous examples, you would be able to use either of these shares:

```
smb://192.168.122.119/chris
smb://192.168.122.119/salesdata
```

The window should appear similar to Figure 19.1:

FIGURE 19.1

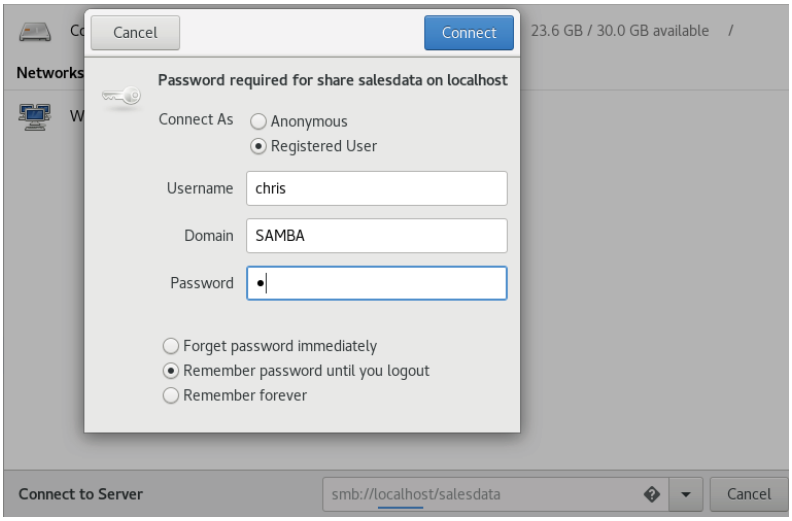
Identify a Samba share from the Nautilus Connect to Server box.



Click Connect. From the window that appears, you can select to connect as a registered user. If you do that, you can enter your username, Samba domain name, and the password for your user. You can also select whether or not to save that password. Figure 19.2 shows an example of that window:

FIGURE 19.2

Add your Samba credentials.

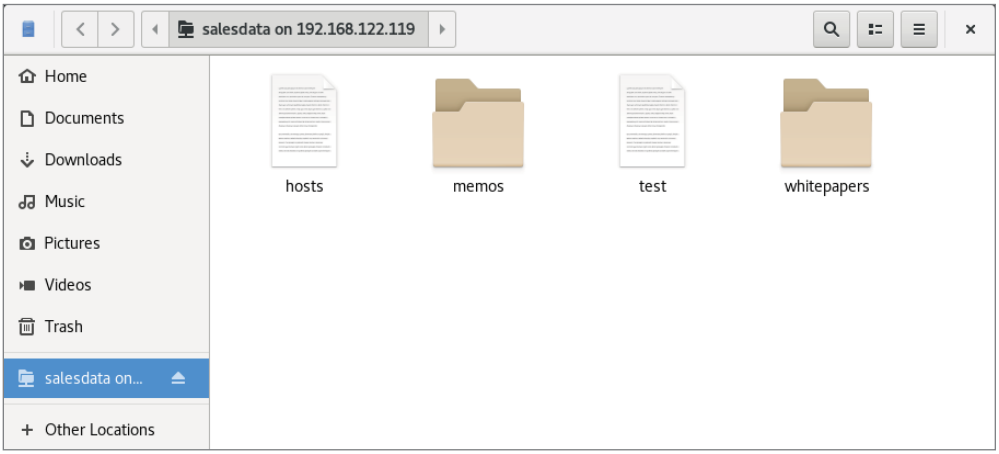


Click Connect.

If the user and password are accepted, you should see the contents of the remote directory. If you have write access to the share, you can open another Nautilus window and drag and drop files between the two systems. Figure 19.3 shows an example of the Nautilus window after I have connected to the `salesdata` share.

FIGURE 19.3

Displaying a Samba share from Connect to Server in Nautilus



Mounting a Samba share from a Linux command line

Because a Samba shared directory can be viewed as a remote filesystem, you can use common Linux tools to connect a Samba share (temporarily or permanently) to your Linux system. Using the standard `mount` command (with `cifs-utils` installed), you can mount a remote Samba share as a CIFS filesystem in Linux. This example mounts the `salesdata` share from the host at IP address 192.168.0.119 on the local directory `/mnt/sales`:

```
# yum install cifs-utils -y
# mkdir /mnt/sales
# mount -t cifs -o user=chris \
      //192.168.0.119/salesdata /mnt/sales
Password for chris@//192.168.122.119/salesdata: *******
# ls /mnt/sales
hosts   memos  test  whitepapers
```

When prompted, enter the Samba password for `chris`. Given that the user `chris` in this example has read-write permission to the shared directory, users on your system should be able to read and write to the mounted directory. Regardless of who saves files on the shared

directory, on the server those files are owned by the user `chris`. This mount lasts until the system is rebooted or you run the `umount` command on the directory. If you want the share to be mounted permanently (that is, every time the system boots up) in the same location, you can do some additional configuration. First, open the `/etc/fstab` file and add an entry similar to the following:

```
//192.168.0.119/salesdata /mnt/sales cifs credentials=/root/cif.txt 0 0
```

Next, create a credentials file (in this example, `/root/cif.txt`). In that file, put the name of the user and the user's password that you want to present when the system tries to mount the filesystem. Here is an example of the contents of that file:

```
user=chris
pass=mypass
```

Before you reboot to check that the entry is correct, try mounting it from the command line. A `mount -a` command tries to mount any filesystem listed in the `/etc/fstab` file that is not already mounted. The `df` command shows information about disk space for the mounted directory, as in the following example:

```
# mount -a
# df -h /mnt/sales
Filesystem                   Size  Used  Avail   Ues%  Mounted on
//192.168.0.119/salesdata     20G  5.7G    14G    30%  /mnt/sales
```

You should now be able to use the shared Samba directory as you do any directory on the local system.

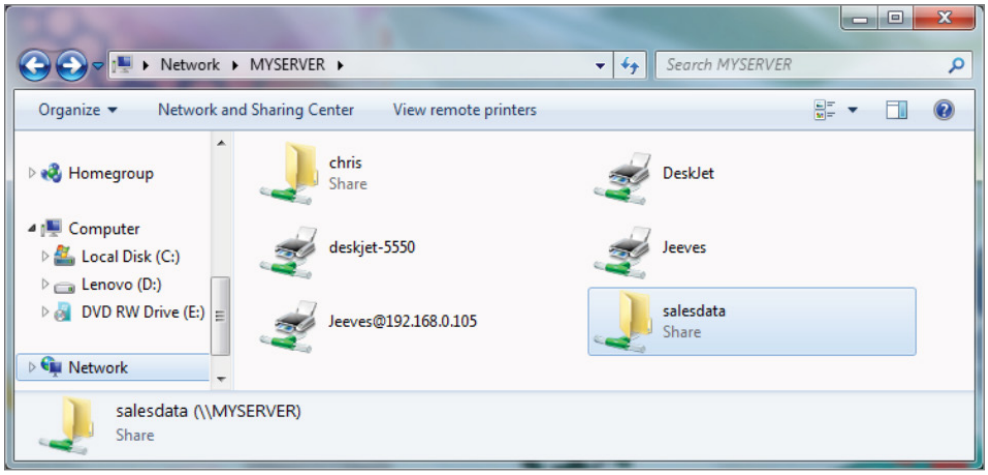
Accessing Samba shares in Windows

As with Linux, you can access Samba shares from the file manager window, in this case Windows File Explorer. To do this, open any folder in Windows, and select Network from the left panel. An icon representing the Samba server should appear on the screen. Click that icon and enter a password if prompted for one. You should see all shared printers and folders from that server (see Figure 19.4).

In Figure 19.4, you can see that there are two shared folders (directories): `chris` and `salesdata`. There are also several shared printers. To use the folders, double-click them and enter the required authentication information. Because printers are set up to use raw drivers by default, you need to obtain Windows drivers to use any of the Samba printers.

FIGURE 19.4

Accessing Samba shares from Windows



Using Samba in the Enterprise

Although it's beyond the scope of this book, Windows file and printer sharing via Samba servers is a very popular application in large enterprises. Despite the fact that Linux has begun to dominate the enterprise-quality server market, Microsoft Windows systems are still the predominant systems used on the desktop.

The major features needed to integrate Samba servers into a large enterprise with many Microsoft Windows desktops are related to authentication. Most large enterprises use Microsoft Active Directory Services (ADS) servers for authentication. On the Linux side, that means configuring Kerberos on the Linux system and using ADS (instead of user) for the type of security in the `smb.conf` file.

The advantage of central authentication is that users have to remember only one set of credentials throughout the enterprise and system administrators need to manage fewer user accounts and passwords.

Summary

Because of the popularity of Windows desktops, Samba servers have become popular for sharing files and printers among Windows and Linux systems. Samba provides a way to interoperate with Windows systems by implementing the Server Message Block (SMB) or Common Internet File (CIFS) protocol for sharing resources over a network.

This chapter stepped through the process of installing, starting, securing, configuring, and accessing Samba servers on a Linux system. Using command-line tools, I demonstrated how to set up a Samba server. I showed you both command-line and desktop tools for getting to Samba shares from Linux and Windows systems.

The next chapter describes the Network File System (NFS) facility. NFS is the native Linux facility for sharing and mounting filesystems over networks with other Linux and UNIX systems.

Exercises

The exercises in this section describe tasks related to setting up a Samba server in Linux and accessing that server using a Samba client. As usual, there are often several ways to accomplish some of the tasks here. So don't worry if you don't go about the exercises in exactly the same way as shown in the answers, as long as you get the same results. See Appendix B for suggested solutions.

Don't do these exercises on a Linux system running a Samba server because they will almost certainly interfere with that server. These exercises were tested on a Fedora system. Some of the steps might be slightly different on another Linux system.

1. Install the `samba` and `samba-client` packages.
2. Start and enable the `smb` and `nmb` services.
3. Set the Samba server's workgroup to `TESTGROUP`, the `netbios name` to `MYTEST`, and the server string to `Samba Test System`.
4. Add a Linux user named `phil` to your system, and add a Linux password and Samba password for `phil`.
5. Set the `[homes]` section so that home directories are browseable (`yes`) and writeable (`yes`), and `phil` is the only valid user.
6. Set any SELinux Boolean that is necessary to make it so that `phil` can access his home directory via a Samba client, then restart the `smb` and `nmb` services.
7. From the local system, use the `smbclient` command to list that the `homes` share is available.
8. From a Nautilus (file manager) window on the local system, connect to the `homes` share for the user `phil` on the local Samba server in a way that allows you to drag and drop files to that folder.
9. Open up the firewall so that anyone who has access to the server can access the Samba service (`smbd` and `nmbd` daemons).
10. From another system on your network (Windows or Linux), try to open the `homes` share again as the user `phil`, and again make sure that you can drag and drop files to it.

CHAPTER

20

Configuring an NFS File Server

IN THIS CHAPTER

Getting NFS server software

Enabling and starting NFS

Exporting NFS directories

Setting security features for NFS

Mounting remote NFS shared directories

Instead of representing storage devices as drive letters (A, B, C, and so on), as they are in Microsoft operating systems, Linux systems invisibly connect filesystems from multiple hard disks, USB drives, CD-ROMs, and other local devices to form a single Linux filesystem. The Network File System (NFS) facility enables you to extend your Linux filesystem to connect filesystems on other computers to your local directory structure.

An NFS file server provides an easy way to share large amounts of data among the users and computers in an organization. An administrator of a Linux system that is configured to share its filesystems using NFS has to perform the following tasks to set up NFS:

1. **Set up the network.** NFS is typically used on private networks as opposed to public networks, such as the Internet.
2. **Start the NFS service.** Several service daemons need to start up and run to have a fully operational NFS service. In Fedora and Red Hat Enterprise Linux, you can start up the `nfs-server` service.
3. **Choose what to share from the server.** Decide which directories (folders) on your Linux NFS server to make available to other computers. You can choose any point in the filesystem and make all files and directories below that point accessible to other computers.
4. **Set up security on the server.** You can use several different security features to apply the level of security with which you are comfortable. *Mount-level security* enables you to restrict the computers that can mount a resource and, for those allowed to mount it, enables you to specify whether it can be mounted read/write or read-only. In NFS, user-level security is implemented by mapping users from the client systems to users on the NFS server (based on

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

UID and not username) so that they can rely on standard Linux read/write/execute permissions, file ownership, and group permissions to access and protect files.

5. **Mount the filesystem on the client**. Each client computer that is allowed access to the server's NFS shared filesystem can mount it anywhere the client chooses. For example, you may mount a filesystem from a computer called `oak` on the `/mnt/oak` directory in your local filesystem. After it is mounted, you can view the contents of that directory by typing `ls /mnt/oak`.

Although it is often used as a file server (or other type of server), Linux is a general-purpose operating system, so any Linux system can share, or export, filesystems as a server or use another computer's filesystems (mount) as a client. In fact, both Red Hat Enterprise Linux 8 and Fedora 30 Workstation include the `nfs-server` service in their default installations.

NOTE

A *filesystem* is usually a structure of files and directories that exists on a single device (such as a hard disk partition or CD-ROM). The term *Linux filesystem* refers to the entire directory structure (which may include filesystems from several disk partitions, NFS, or a variety of network resources), beginning from root (`/`) on a single computer. A shared directory in NFS may represent all or part of a computer's filesystem, which can be attached (from the shared directory down the directory tree) to another computer's filesystem.

If you already have the NFS and Cockpit services running on your system, you can mount NFS shares and view mounted shares from the Cockpit Web UI. Here's how to do that:

1. Log in to your Cockpit interface (port 9090) through your web browser and select Storage. The URL to get to storage in the Cockpit service on your local system should be something like `https://host1.example.com:9090/storage`.
2. If there are mounted NFS shares on your system, they should appear under the NFS Mounts section. Figure 20.1 shows an example containing two mounted NFS shares.
3. To mount a remote NFS share, select the plus (+) sign on the NFS Mounts line. Fill in the address or hostname of the NFS server, the shared directory on the NFS share, and the point on the local file system where you will mount that share. Then select Add, as shown in Figure 20.2.

At this point, you should be able to access the content from the remote NFS share from the mount point on your local filesystem. By default, the NFS mount information is added to the `/etc/fstab` file, so the NFS share will be made available each time the system reboots. Now that you have seen the easy way to use NFS, the rest of the chapter describes how to use NFS from the ground up.

FIGURE 20.1

View NFS shares mounted locally using Cockpit Web UI

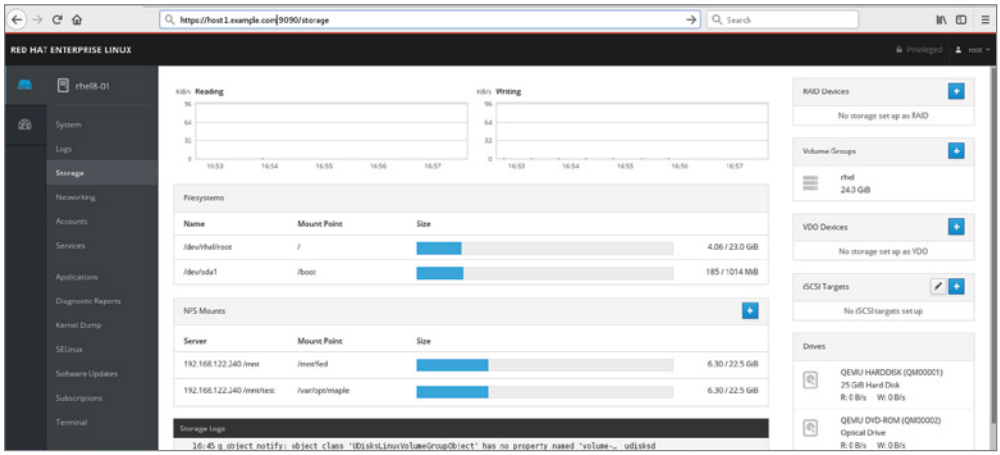
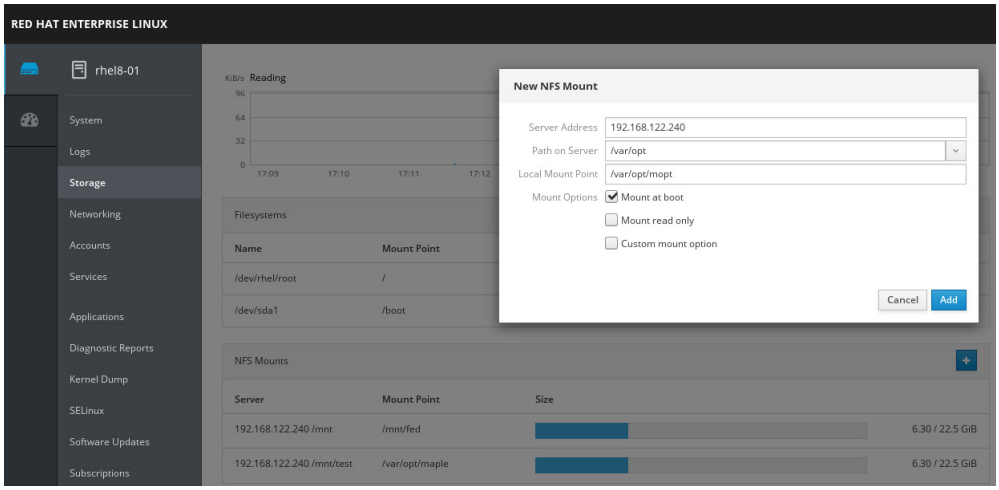


FIGURE 20.2

Add a new NFS mount using Cockpit Web UI



Installing an NFS Server

To run an NFS server, you need a set of kernel modules (which are delivered with the kernel itself) plus some user-level tools to configure the service, run daemon processes, and query the service in various ways.

For earlier releases of Fedora and RHEL, the components you need that are not already in the kernel can be added by installing the `nfs-utils` package. In RHEL 8 and Fedora 30, the required components are included in the following default installation:

```
# yum install nfs-utils
```

Besides a few documents in the `/usr/share/doc/nfs-utils` directory, most documentation in the `nfs-utils` package includes man pages for its various components. To see the list of documentation, type the following:

```
# rpm -qd nfs-utils | less
```

There are tools and man pages for both the NFS server side (for sharing a directory with others) and the client side (for mounting a remote NFS directory locally). To configure a server, you can refer to the exports man page (to set up the `/etc/exports` file to share your directories). The man page for the `exportfs` command describes how to share and view the list of directories that you share from the `/etc/exports` file. The `nfsd` man page describes the options that you can pass to the `rpc.nfsd` server daemon, which lets you do such things as run the server in debugging mode.

Man pages on the client side include the `mount.nfs` man page (to see what mount options you can use when mounting remote NFS directories on your local system). There is also an `nfsmount.conf` man page, which describes how to use the `/etc/nfsmount.conf` file to configure how your system behaves when you mount remote resources locally. The `showmount` man page describes how to use the `showmount` command to see what shared directories are available from NFS servers.

To find out more about the `nfs-utils` package, you can run the following commands to see information about the package, configuration files, and commands, respectively:

```
# rpm -qi nfs-utils
# rpm -qc nfs-utils
# rpm -ql nfs-utils | grep bin
```

Starting the NFS service

Starting the NFS server involves launching several service daemons. The basic NFS service in Fedora and RHEL 8 is called `nfs-server`. To start that service, enable it (so it starts each time your system boots) and check the status by running the following three commands:

```
# systemctl start nfs-server.service
# systemctl enable nfs-server.service
```

```
# systemctl status nfs-server.service
• nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled
             vendor preset: disabled)
     Active: active (exited) since Mon 2019-9-02 15:15:11 EDT; 24s
ago
   Main PID: 7767 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 12244)
     Memory: 0B
     CGroup: /system.slice/nfs-server.service
```

You can see from the status that the `nfs-server` service is enabled and active. The NFS service also requires that the RPC service be running (`rpcbind`). The `nfs-server` service automatically starts the `rpcbind` service, if it is not already running.

In Red Hat Enterprise Linux 6, you need the `service` and `chkconfig` commands to check, start, and enable the NFS service (`nfs`). The following commands show the `nfs` service not running currently and disabled:

```
# service nfs status
rpc.svcgssd is stopped
rpc.mountd is stopped
nfsd is stopped
# chkconfig --list nfs
nfs  0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

As mentioned earlier, the `rpcbind` service must be running for NFS to work. In RHEL 6, you could use the following commands to start and permanently enable both the `rpcbind` and `nfs` services:

```
# service rcpbind start
Starting rpcbind:                          [  OK  ]
# service nfs start
Starting NFS services:                     [  OK  ]
Starting NFS quotas:                       [  OK  ]
Starting NFS daemon:                       [  OK  ]
Starting NFS mountd:                       [  OK  ]
# chkconfig rpcbind on
# chkconfig nfs on
```

After the service is running, the commands (`mount`, `exportfs`, and so on) and files (`/etc/exports`, `/etc/fstab`, and so on) for actually configuring NFS are basically the same on every Linux system. So, after you have NFS installed and running, just follow the instructions in this chapter to start using NFS.

Sharing NFS Filesystems

To share an NFS filesystem from your Linux system, you need to export it from the server system. Exporting is done in Linux by adding entries into the `/etc/exports` file. Each

entry identifies a directory in your local filesystem that you want to share with other computers. The entry also identifies the other computers that can access the resource (or opens it to all computers) and includes other options that reflect permissions associated with the directory.

Remember that when you share a directory, you are sharing all files and subdirectories below that directory as well (by default). You need to be sure that you want to share everything in that directory structure. You can still restrict access within that directory structure in many ways; those are discussed later in this chapter.

Configuring the /etc/exports file

To make a directory from your Linux system available to other systems, you need to export that directory. Exporting is done on a permanent basis by adding information about an exported directory to the `/etc/exports` file.

Here's the format of the `/etc/exports` file:

```
Directory   Host(Options...)  Host(Options...)  # Comments
```

In this example, *`Directory`* is the name of the directory that you want to share, and Host indicates the client computer to which the sharing of this directory is restricted. *`Options`* can include a variety of options to define the security measures attached to the shared directory for the host. (You can repeat `Host` and `Option` pairs.) *`Comments`* are any optional comments that you want to add (following the # sign).

The `exports` man page (`man exports`) contains details about the syntax of the `/etc/exports` file. In particular, you can see the options that you can use to limit access and secure each shared directory.

As root user, you can use any text editor to configure `/etc/exports` to modify shared directory entries or add new ones. Here's an example of an `/etc/exports` file:

```
/cal     *.linuxtoys.net(rw)                # Company events
/pub     *(ro,insecure,all_squash)          # Public dir
/home    maple(rw,root_squash) spruce(rw,root_squash)
```

The `/cal` entry represents a directory that contains information about events related to the company. Any computer in the company's domain (`*.linuxtoys.net`) can mount that NFS share. Users can write files to the directory as well as read them (indicated by the `rw` option). The comment (`# Company events`) simply serves to remind you of what the directory contains.

The `/pub` entry represents a public directory. It allows any computer and user to read files from the directory (indicated by the `ro` option) but not to write files. The `insecure` option enables any computer, even one that doesn't use a secure NFS port, to access the directory. The `all _ squash` option causes all users (UIDs) and groups (GIDs) to be mapped to the `nobody` user (UID `65534`), giving them minimal permission to files and directories.

The /home entry enables a set of users to have the same /home directory on different computers. Suppose, for example, that you are sharing /home from a computer named oak. The computers named maple and spruce could each mount that directory on their own /home directories. If you gave all users the same username/UID on all machines, you could have the same /home/*user* directory available for each user, regardless of which computer they are logged into. The root _ squash is used to exclude the root user from another computer from having root privilege to the shared directory.

These are just examples; you can share any directories that you choose, including the entire filesystem (/). Of course, there are security implications of sharing the whole filesystem or sensitive parts of it (such as /etc). Security options that you can add to your /etc/exports file are described throughout the sections that follow.

Hostnames in /etc/exports

You can indicate in the /etc/exports file which host computers can have access to your shared directory. If you want to associate multiple hostnames or IP addresses with a particular shared directory, be sure to leave a space before each hostname. However, add no spaces between a hostname and its options. Here's an example:

```
/usr/local maple(rw) spruce(ro,root_squash)
```

Notice that there is a space after (rw) but none after maple. You can identify hosts in several ways:

Individual host Enter one or more TCP/IP hostnames or IP addresses. If the host is in your local domain, you can simply indicate the hostname. Otherwise, use the full host.domain format. These are valid ways to indicate individual host computers:

```
maple
maple.handsonhistory.com
10.0.0.11
```

IP network Allow access to all hosts from a particular network address by indicating a network number and its netmask, separated by a slash (/). Here are valid ways to designate network numbers:

```
10.0.0.0/255.0.0.0 172.16.0.0/255.255.0.0
192.168.18.0/255.255.255.0
192.168.18.0/24
```

TCP/IP domain Using wildcards, you can include all or some host computers from a particular domain level. Here are some valid uses of the asterisk and question mark wildcards:

```
*.handsonhistory.com
*craft.handsonhistory.com
???.handsonhistory.com
```

The first example matches all hosts in the handsonhistory.com domain. The second example matches woodcraft, basketcraft, or any other hostnames ending in craft in the handsonhistory.com domain. The final example matches any three-letter hostnames in the domain.

NIS groups You can allow access to hosts contained in an NIS group. To indicate an NIS group, precede the group name with an at (@) sign (for example, @group).

Access options in /etc/exports

You don't have to just give away your files and directories when you export a directory with NFS. In the options part of each entry in /etc/exports, you can add options that allow or limit access by setting read/write permission. These options, which are passed to NFS, are as follows:

ro: Client can mount this exported filesystem read-only. The default is to mount the filesystem read/write.

rw: Explicitly asks that a shared directory be shared with read/write permissions. (If the client chooses, it can still mount the directory as read-only.)

User mapping options in /etc/exports

In addition to options that define how permissions are handled generally, you can use options to set the permissions that specific users have to NFS shared filesystems.

One method that simplifies this process is to have each user with multiple user accounts have the same username and UID on each machine. This makes it easier to map users so they have the same permissions on a mounted filesystem as they do on files stored on their local hard disks. If that method is not convenient, user IDs can be mapped in many other ways. Here are some methods of setting user permissions and the /etc/exports option that you use for each method:

root user The client's root user is mapped by default into the nobody username (UID 65534). This prevents a client computer's root user from being able to change all files and directories in the shared filesystem. If you want the client's root user to have root permission on the server, use the no_root_squash option.

Tip

Keep in mind that even though root is squashed, the root user from the client can still become any other user account and access files for those user accounts on the server. So, be sure that you trust root with all of your user data before you share it read/write with a client.

nfsnobody or nobody user/group By using the 65534 user ID and group ID, you essentially create a user/group with permissions that do not allow access to files that belong to any real users on the server, unless those users open permission to everyone. However, files created by the 65534 user or group are available to anyone

assigned as the 65534 user or group. To set all remote users to the 65534 user/group, use the all _ squash option.

The 65534 UIDs and GIDs are used to prevent the ID from running into a valid user or group ID. Using anonuid or anongid options, you can change the 65534 user or group, respectively. For example, anonuid=175 sets all anonymous users to UID 175, and anongid=300 sets the GID to 300. (Only the number is displayed when you list file permission unless you add entries with names to /etc/passwd and /etc/group for the new UIDs and GIDs.)

User mapping If a user has login accounts for a set of computers (and has the same ID), NFS, by default, maps that ID. This means that if the user named mike (UID 110) on maple has an account on pine (mike, UID 110), he can use his own remotely mounted files on either computer from either computer.

If a client user who is not set up on the server creates a file on the mounted NFS directory, the file is assigned to the remote client's UID and GID. (An ls -l on the server shows the UID of the owner.)

Exporting the shared filesystems

After you have added entries to your /etc/exports file, run the exportfs command to have those directories exported (made available to other computers on the network). Reboot your computer or restart the NFS service, and the exportfs command runs automatically to export your directories. If you want to export them immediately, run exportfs from the command line (as root).

Tip

Running the exportfs command after you change the exports file is a good idea. If any errors are in the file, exportfs identifies them for you.

Here's an example of the exportfs command:

```
# /usr/sbin/exportfs -a -r -v
exporting maple:/pub
exporting spruce:/pub
exporting maple:/home
exporting spruce:/home
exporting *:/mnt/win
```

The -a option indicates that all directories listed in /etc/exports should be exported. The -r resyncs all exports with the current /etc/exports file (disabling those exports no longer listed in the file). The -v option says to print verbose output. In this example, the /pub and /home directories from the local server are immediately available for mounting by those client computers that are named (maple and spruce). The /mnt/win directory is available to all client computers.

Securing Your NFS Server

The NFS facility was created at a time when encryption and other security measures were not routinely built into network services (such as remote login, file sharing, and remote execution). Therefore, NFS (even up through version 3) suffers from some rather glaring security issues.

NFS security issues made it an inappropriate facility to use over public networks and even made it difficult to use securely within an organization. These are some of the issues:

Remote `root` users Even with the default `root_squash` (which prevents root users from having root access to remote shares), the root user on any machine to which you share NFS directories can gain access to any other user account. Therefore, if you are doing something like sharing home directories with read/write permission, the root user on any box to which you are sharing has complete access to the contents of those home directories.

Unencrypted communications Because NFS traffic is unencrypted, anyone sniffing your network can see the data that is being transferred.

User mapping Default permissions to NFS shares are mapped by user ID. So, for example, a user with UID 1000 on an NFS client has access to files owned by UID 1000 on the NFS server. This is regardless of the usernames used.

Filesystem structure exposed Up to NFSv3, if you shared a directory over NFS, you exposed the location of that directory on the server's filesystem. (In other words, if you shared the `/var/stuff` directory, clients would know that `/var/stuff` was its exact location on your server).

That's the bad news. The good news is that most of these issues are addressed in NFSv4 but require some extra configuration. By integrating Kerberos support, NFSv4 lets you configure user access based on each user obtaining a Kerberos ticket. For you, the extra work is configuring a Kerberos server. As for exposing NFS share locations, with NFSv4 you can bind shared directories to an `/exports` directory, so when they are shared, the exact location of those directories is not exposed.

Visit `https://help.ubuntu.com/community/NFSv4Howto` for details on NFSv4 features in Ubuntu.

As for standard Linux security features associated with NFS, `iptables` firewalls, TCP wrappers, and SELinux can all play a role in securing and providing access to your NFS server from remote clients. In particular, getting firewall features working with NFS can be particularly challenging. These security features are described in the sections that follow.

Opening up your firewall for NFS

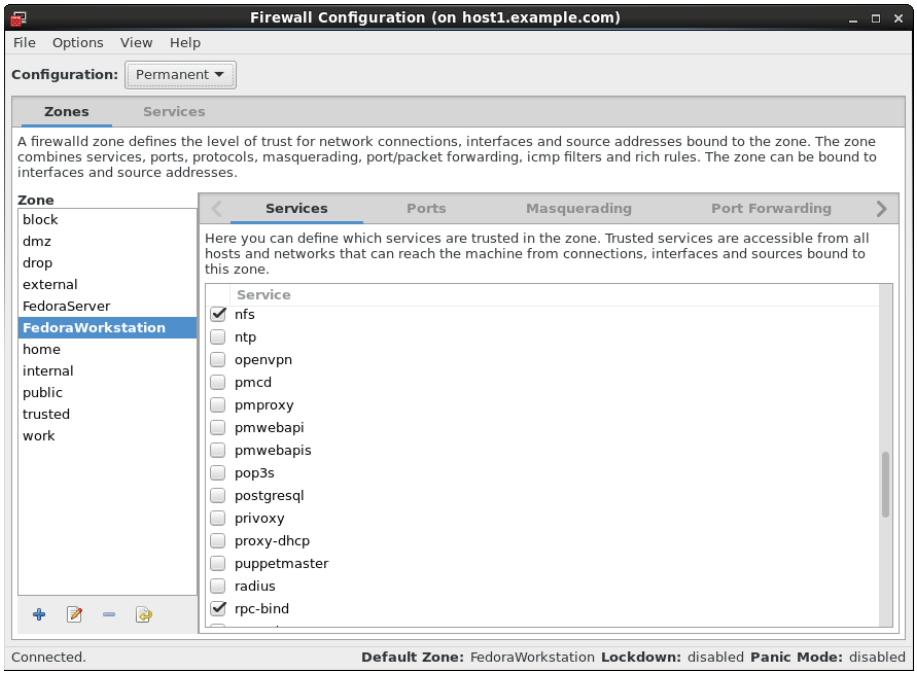
The NFS service relies on several different service daemons for normal operation, with most of these daemons listening on different ports for access. For the default NFSv4 used in Fedora, TCP and UDP ports 2049 (`nfs`) and 111 (`rpcbind`) must be open for an NFS server

to perform properly. The server must also open TCP and UDP ports 20048 for the `showmount` command to be able to query available NFS shared directories from `rpc.mountd` on the server.

For RHEL 8, Fedora 30, and other systems that use the `firewalld` service, you can use the Firewall Configuration window (`yum install firewall-config`) to open the firewall for your NFS service. Type `firewall-config`, then make sure that mountd, nfs, and rpc-bind are checked in the window to open the appropriate ports to allow access to your NFS service. Figure 20.3 shows an example of this window:

FIGURE 20.3

Use the Firewall Configuration window to open your firewall to allow access to the NFS service.



For RHEL 6 and other systems that use `iptables` service directly (prior to `firewalld` being added), to open ports on the NFS server's firewall, make sure `iptables` is enabled and started with firewall rules similar to the following added to the `/etc/sysconfig/iptables` file:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 2049 -j ACCEPT
```

```
-A INPUT -m state --state NEW -m udp -p udp --dport 2049 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 20048 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 20048 -j ACCEPT
```

In Red Hat Enterprise Linux 6.*x* and earlier, the firewall issue is a bit more complex. The problem, as it relates to firewalls, is that several different services are associated with NFS that listen on different ports, and those ports are assigned randomly. To get around that problem, you need to lock down the port numbers those services use and open the firewall so that those ports are accessible.

To make the process of locking down NFS server ports easier, entries in the `/etc/sysconfig/nfs` file can be added to assign specific port numbers to services. The following are examples of options in the `/etc/sysconfig/nfs` file with static port numbers set:

```
RQUOTAD_PORT=49001
LOCKD_TCPPORT=49002
LOCKD_UDPPORT=49003
MOUNTD_PORT=49004
STATD_PORT=49005
STATD_OUTGOING_PORT=49006
RDMA_PORT=49007
```

With those ports set, I restarted the `nfs` service (`service nfs restart`). Using the `netstat` command, you can see the resulting processes that are listening on those assigned ports:

```
tcp   0   0 0.0.0.0:49001   0.0.0.0:*     LISTEN    4682/rpc.rquotad
tcp   0   0 0.0.0.0:49002   0.0.0.0:*     LISTEN    -
tcp   0   0 0.0.0.0:49004   0.0.0.0:*     LISTEN    4698/rpc.mountd
tcp   0   0 :::49002        :::*          LISTEN    -
tcp   0   0 :::49004        :::*          LISTEN    4698/rpc.mountd
udp   0   0 0.0.0.0:49001   0.0.0.0:*               4682/rpc.rquotad
udp   0   0 0.0.0.0:49003   0.0.0.0:*               -
udp   0   0 0.0.0.0:49004   0.0.0.0:*               4698/rpc.mountd
udp   0   0 :::49003        :::*                    -
udp   0   0 :::49004        :::*                    4698/rpc.mountd
```

With those port numbers set and being used by the various services, you can now add `iptables` rules, as you did with ports 2049 and 111 for the basic NFS service.

Allowing NFS access in TCP wrappers

For services such as `vsftpd` and `sshd`, TCP wrappers in Linux enable you to add information to `/etc/hosts.allow` and `/etc/hosts.deny` files to indicate which hosts can or cannot access the service. Although the `nfsd` server daemon itself is not enabled for TCP wrappers, the `rpcbind` service is.

For NFSv3 and earlier versions, simply adding a line such as the following to the `/etc/hosts.deny` file would deny access to the `rpcbind` service, but it would also deny access to your NFS service:

```
rpcbind: ALL
```

For servers running NFSv4 by default, however, the `rpcbind: ALL` line just shown prevents outside hosts from getting information about RPC services (such as NFS) using commands like `showmount`. However, it does not prevent you from mounting an NFS shared directory.

Configuring SELinux for your NFS server

With SELinux set to permissive or disabled, it does not block access to the NFS service. In enforcing mode, however, you should understand a few SELinux Booleans. To check the state of SELinux on your system, enter the following:

```
# getenforce
Enforcing
# grep ^SELINUX= /etc/sysconfig/selinux
SELINUX=enforcing
```

If your system is in enforcing mode, as it is here, check the `nfs _ selinux` man page for information about SELinux settings that can impact the operation of your `vsftpd` service. Here are a few SELinux file contexts associated with NFS that you might need to know about:

- `nfs_export_all_ro`: With this Boolean set to on, SELinux allows you to share files with read-only permission using NFS. NFS read-only file sharing is allowed with this on regardless of the SELinux file context set on the shared files and directories.
- `nfs_export_all_rw`: With this Boolean set to on, SELinux allows you to share files with read/write permission using NFS. As with the previous Boolean, this works regardless of the file context set on the shared files and directories.
- `use_nfs_home_dirs`: To allow the NFS server to share your home directories via NFS, set this Boolean to on.

Of the Booleans just described, the first two are on by default. The `use_nfs_home_dirs` Boolean is off. To turn on the `use_nfs_home_dirs` directory, you could type the following:

```
# setsebool -P use_nfs_home_dirs on
```

You can ignore all of the Booleans related to NFS file sharing, however, by changing the file contexts on the files and directories you want to share via NFS. The `public_content_t` and `public_content_rw_t` file contexts can be set on any directory that you want to share via NFS (or other file share protocols, such as HTTP, FTP, and others, for that matter). For example, to set the rule to allow the `/whatever` directory and its subdirectories to be shared read/write via NFS, and then to apply that rule, enter the following:

```
# semanage fcontext -a -t public_content_rw_t "/whatever(/.*)?"
# restorecon -F -R -v /whatever
```

If you wanted to allow users just to be able to read files from a directory, but not write to it, you could assign the `public_content_t` file context to the directory instead.

Using NFS Filesystems

After a server exports a directory over the network using NFS, a client computer connects that directory to its own filesystem using the `mount` command. That's the same command used to mount filesystems from local hard disks, DVDs, and USB drives, but with slightly different options.

The `mount` command enables a client to mount NFS directories added to the `/etc/fstab` file automatically, just as it does with local disks. NFS directories can also be added to the `/etc/fstab` file in such a way that they are not automatically mounted (so you can mount them manually when you choose). With a `noauto` option, an NFS directory listed in `/etc/fstab` is inactive until the `mount` command is used, after the system is up and running, to mount the filesystem.

In addition to the `/etc/fstab` file, you can set mount options using the `/etc/nfsmount.conf` file. Within that file, you can set mount options that apply to any NFS directory you mount or only those associated with specific mount points or NFS servers.

Before you set about mounting NFS shared directories, however, you probably want to check out what shared directories are available via NFS using the `showmount` command.

Viewing NFS shares

From a client Linux system, you can use the `showmount` command to see what shared directories are available from a selected computer, such as in this example:

```
$ showmount -e server.example.com
/export/myshare client.example.com
/mnt/public     *
```

The `showmount` output shows that the shared directory named `/export/myshare` is available only to the host `client.example.com`. The `/mnt/public` shared directory, however, is available to anyone.

Manually mounting an NFS filesystem

After you know that the directory from a computer on your network has been exported (that is, made available for mounting), you can mount that directory manually using the `mount` command. This is a good way to make sure that it is available and working before you set it up to mount permanently. The following is an example of mounting the `/stuff` directory from a computer named `maple` on your local computer:

```
# mkdir /mnt/maple
# mount maple:/stuff /mnt/maple
```

The first command (`mkdir`) creates the mount point directory. (`/mnt` is a common place to put temporarily mounted disks and NFS filesystems.) The `mount` command identifies the remote computer and shared filesystem, separated by a colon (`maple:/stuff`), and the local mount point directory (`/mnt/maple`) follows.

NOTE

If the mount fails, make sure that the NFS service is running on the server and that the server's firewall rules don't deny access to the service. From the server, type **`ps ax | grep nfsd`** to see a list of `nfsd` server processes. If you don't see the list, try to start your NFS daemons as described earlier in this chapter. To view your firewall rules, type **`iptables -vnL`**. By default, the `nfsd` daemon listens for NFS requests on port number 2049. Your firewall must accept `udp` requests on ports 2049 (`nfs`) and 111 (`rpc`). In Red Hat Enterprise Linux 6 and earlier versions of Fedora, you may need to set static ports for related services and then open ports for those services in the firewall. Refer to the section "Securing Your NFS Server" earlier in this chapter to review how to overcome these security issues.

To ensure that the NFS mount occurred, type **`mount -t nfs4`**. This command lists all mounted NFS filesystems. Here is an example of the `mount` command and its output (with filesystems not pertinent to this discussion edited out):

```
# mount -t nfs4
192.168.122.240:/mnt on /mnt/fed type nfs4
(rw,relatime,vers=4.2,rsize=262144,wsize=262144,namlen=255,hard,
proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.168.122.63,
local_lock=none,addr=192.168.122.240)
```

The output from the `mount -t nfs4` command shows only those filesystems mounted from NFS file servers. The NFS filesystem is the `/mnt` directory from 192.168.122.240 (`192.168.122.240:/mnt`). It is mounted on `/mnt/fed`, and its mount type is `nfs4`. The filesystem was mounted read/write (`rw`), and the IP address of `maple` is `192.168.122.240` (`addr=192.168.122.240`). Many other settings related to the mount are shown as well, such as the read and write sizes of packets and the NFS version number.

The mount operation just shown temporarily mounts an NFS filesystem on the local system. The next section describes how to make the mount more permanent (using the `/etc/fstab` file) and how to select various options for NFS mounts.

Mounting an NFS filesystem at boot time

To set up an NFS filesystem to mount automatically on a specified mount point each time you start your Linux system, you need to add an entry for that NFS filesystem to the `/etc/fstab` file. That file contains information about all different kinds of mounted (and available to be mounted) filesystems for your system.

Here's the format for adding an NFS filesystem to your local system:

```
host:directory    mountpoint    nfs    options    0    0
```

The first item (*host:directory*) identifies the NFS server computer and shared directory. *mountpoint* is the local mount point on which the NFS directory is mounted. It is followed by the filesystem type (nfs). Any options related to the mount appear next in a comma-separated list. (The last two zeros configure the system not to dump the contents of the filesystem and not to run fsck on the filesystem.)

The following are examples of NFS entries in /etc/fstab:

```
maple:/stuff    /mnt/maple nfs   bg,rsize=8192,wsize=8192  0 0
oak:/apps       /oak/apps  nfs   noauto,ro                 0 0
```

In the first example, the remote directory /stuff from the computer named maple (maple:/stuff) is mounted on the local directory /mnt/maple (the local directory must already exist). If the mount fails because the share is unavailable, the bg causes the mount attempt to go into the background and retry again later.

The filesystem type is nfs, and read (rsize) and write (wsize) buffer sizes (discussed in the section "Using mount options," later in this chapter) are set at 8192 to speed data transfer associated with this connection. In the second example, the remote directory is /apps on the computer named oak. It is set up as an NFS filesystem (nfs) that can be mounted on the /oak/apps directory locally. This filesystem is not mounted automatically (noauto), however, and it can be mounted only as read-only (ro) using the mount command after the system is already running.

Tip

The default is to mount an NFS filesystem as read/write. However, the default for exporting a filesystem is read-only. If you are unable to write to an NFS filesystem, check that it was exported as read/write from the server.

Mounting noauto filesystems

Your /etc/fstab file may also contain devices for other filesystems that are not mounted automatically. For example, you might have multiple disk partitions on your hard disk or an NFS shared filesystem that you want to mount only occasionally. A noauto filesystem can be mounted manually. The advantage is that when you type the mount command, you can type less information and have the rest filled in by the contents of the /etc/fstab file. So, for example, you could type

```
# mount /oak/apps
```

With this command, mount knows to check the /etc/fstab file to get the filesystem to mount (oak:/apps), the filesystem type (nfs), and the options to use with the mount (in this case ro, for read-only). Instead of typing the local mount point (/oak/apps), you could have typed the remote filesystem name (oak:/apps) and had other information filled in.

Tip

When naming mount points, including the name of the remote NFS server in that name can help you remember where the files are actually being stored. This may not be possible if you are sharing home directories (`/home`) or mail directories (`/var/spool/mail`). For example, you might mount a filesystem from a machine called `duck` on the directory `/mnt/duck`.

Using mount options

You can add several `mount` options to the `/etc/fstab` file (or to a `mount` command line itself) to influence how the filesystem is mounted. When you add options to `/etc/fstab`, they must be separated by commas. For example, here the `noauto`, `ro`, and `hard` options are used when `oak:/apps` is mounted:

```
oak:/apps    /oak/apps  nfs   noauto,ro,hard    0 0
```

The following are some options that are valuable for mounting NFS filesystems. You can read about these and other NFS mount options you can put in the `/etc/fstab` file from the `nfs` man page (`man 5 nfs`):

`hard` If this option is used and the NFS server disconnects or goes down while a process is waiting to access it, the process hangs until the server comes back up. This is helpful if it is critical that the data with which you are working stay in sync with the programs that are accessing it. (This is the default behavior.)

`soft` If the NFS server disconnects or goes down, a process trying to access data from the server times out after a set period when this option is on. An input/output error is delivered to the process trying to access the NFS server.

`rsize` This is the size of the blocks of data (in bytes) that the NFS client will request be used when it is reading data from an NFS server. The default is 1024. Using a larger number (such as 8192) gets you better performance on a network that is fast (such as a LAN) and is relatively error-free (that is, one that doesn't have lots of noise or collisions).

`wsize` This is the size of the blocks of data (in bytes) that the NFS client will request to be used when it is writing data to an NFS server. The default is 1024. Performance issues are the same as with the `rsize` option.

`timeo=#` This sets the time after an RPC time-out occurs that a second transmission is made, where # represents a number in tenths of a second. The default value is seven-tenths of a second. Each successive time-out causes the time-out value to be doubled (up to 60 seconds maximum). Increase this value if you believe that time-outs are occurring because of slow response from the server or a slow network.

`retrans=#` This sets the number of minor time-outs and retransmissions that need to happen before a major time-out occurs.

retry=# This sets how many minutes to continue to retry failed mount requests, where # is replaced by the number of minutes to retry. The default is 10,000 minutes (which is about one week).

bg If the first mount attempt times out, try all subsequent mounts in the background. This option is very valuable if you are mounting a slow or sporadically available NFS filesystem. When you place mount requests in the background, your system can continue to mount other filesystems instead of waiting for the current one to complete.

NOTE
If a nested mount point is missing, a time-out to allow for the needed mount point to be added occurs. For example, if you mount `/usr/trip` and `/usr/trip/extra` as NFS filesystems and `/usr/trip` is not yet mounted when `/usr/trip/extra` tries to mount, `/usr/trip/extra` times out. If you're lucky, `/usr/trip` comes up and `/usr/trip/extra` mounts on the next retry.

fg If the first mount attempt times out, try subsequent mounts in the foreground. This is the default behavior. Use this option if it is imperative that the mount be successful before continuing (for example, if you were mounting `/usr`).

Not all NFS mount options need to go into the `/etc/fstab` file. On the client side, the `/etc/nfsmount.conf` file can be configured for Mount, Server, and Global sections. In the Mount section, you can indicate which mount options are used when an NFS filesystem is mounted to a particular mount point. The Server section lets you add options to any NFS filesystem mounted from a particular NFS server. Global options apply to all NFS mounts from this client.

The following entry in the `/etc/nfsmount.conf` file sets a 32KB read and write block size for any NFS directories mounted from the system named `thunder.example.com`:

```
[ Server "thunder.example.com" ]
  rsize=32k
  wsize=32k
```

To set default options for all NFS mounts for your systems, you can uncomment the `NFSMount _ Global _ Options` block. In that block, you can set such things as protocols and NFS versions as well as transmission rates and retry settings. Here is an example of an `NFSMount _ Global _ Options` block:

```
[ NFSMount_Global_Options ]
# This sets the default version to NFS 4
Defaultvers=4
# Sets the number of times a request will be retried before
# generating a timeout
Retrans=2
# Sets the number of minutes before retrying a failed
# mount to 2 minutes
Retry=2
```

In the example just shown, the default NFS version is 4. Data is retransmitted twice (2) before generating a time-out. The wait time is 2 minutes before retrying a failed transmission. You can override any of these default values by adding mount options to the /etc/fstab or to the mount command line when the NFS directory is mounted.

Using autofs to mount NFS filesystems on demand

Improvements to autodetecting and mounting removable devices have meant that you can simply insert or plug in those devices to have them detected, mounted, and displayed. However, to make the process of detecting and mounting remote NFS filesystems more automatic, you still need to use a facility such as autofs (short for *automatically mounted filesystems*).

The autofs facility mounts network filesystems on demand when someone tries to use the filesystems. With the autofs facility configured and turned on, you can cause any available NFS shared directories to mount on demand. To use the autofs facility, you need to have the autofs package installed. (For Fedora and RHEL, you can type **yum install autofs** or for Ubuntu or Debian **apt-get install autofs** to install the package from the network.)

Automounting to the /net directory

With autofs enabled, if you know the hostname and directory being shared by another host computer, simply change (cd) to the autofs mount directory (/net or /var/autofs by default). This causes the shared resource to be automatically mounted and made accessible to you.

The following steps explain how to turn on the autofs facility in Fedora or RHEL:

1. **In Fedora or RHEL, as root user from a Terminal window, open the /etc/auto.master file and look for the following line:**

   ```
   /net    -hosts
   ```

   This causes the /net directory to act as the mount point for the NFS shared directories that you want to access on the network. (If there is a comment character at the beginning of that line, remove it.)

2. **To start the autofs service in a Fedora 30, RHEL 7, or later system, type the following as root user:**

   ```
   # systemctl start autofs.service
   ```

3. **On a Fedora 30, RHEL 7, or later system, set up the autofs service to restart every time you boot your system:**

   ```
   # systemctl enable autofs
   ```

Believe it or not, that's all you have to do. If you have a network connection to the NFS servers from which you want to share directories, try to access a shared NFS directory. For

example, if you know that the /usr/local/share directory is being shared from the computer on your network named shuttle, you can do the following:

```
$ cd /net/shuttle/
```

If that computer has any shared directories that are available to you, you can successfully change to that directory.

You also can type the following:

```
$ ls
usr
```

You should be able to see that the usr directory is part of the path to a shared directory. If there were shared directories from other top-level directories (such as /var or /tmp), you would see those. Of course, seeing any of those directories depends on how security is set up on the server.

Try going straight to the shared directory, as shown in this example:

```
$ cd /net/shuttle/usr/local/share
$ ls
info man music television
```

At this point, the ls should reveal the contents of the /usr/local/share directory on the computer named shuttle. What you can do with that content depends on how it was configured for sharing by the server.

This can be a bit disconcerting because you don't see any files or directories until you actually try to use them, such as changing to a network-mounted directory. The ls command, for example, doesn't show anything under a network-mounted directory until the directory is mounted, which may lead to a sometimes-it's-there-and-sometimes-it's-not impression. Just change to a network-mounted directory, or access a file on such a directory, and autofs takes care of the rest.

In the example shown, the hostname shuttle is used. However, you can use any name or IP address that identifies the location of the NFS server computer. For example, instead of shuttle, you might have used shuttle.example.com or an IP address such as 192.168.0.122.

Automounting home directories

Instead of just mounting an NFS filesystem under the /net directory, you might want to configure autofs to mount a specific NFS directory in a specific location. For example, you could configure a user's home directory from a centralized server that could be automounted from a different machine when a user logs in. Likewise, you could use a central authentication mechanism, such as LDAP (as described in Chapter 11, "Managing User Accounts"), to offer centralized user accounts.

The following procedure illustrates how to set up a user account on an NFS server and share the home directory of a user named joe from that server so that it can be automounted when joe logs into a different computer. In this example, instead of using a central authentication server, matching accounts are created on each system.

1. On the NFS server (mynfs.example.com) that provides a centralized user home directory for the user named joe, create a user account for joe with a home directory of /home/shared/joe as its name. Also find joe's user ID number from the /etc/passwd file (third field) so that you can match it when you set up a user account for joe on another system.

```
# mkdir /home/shared
# useradd -c "Joe Smith" -d /home/shared/joe joe
# grep joe /etc/passwd
joe:x:1000:1000:Joe Smith:/home/shared/joe:/bin/bash
```

2. On the NFS server, export the /home/shared/ directory to any system on your local network (I use 192.168.0.* here), so that you can share the home directory for joe and any other users you create by adding this line to the /etc/exports file:

```
# /etc/exports file to share directories under /home/shared
# only to other systems on the 192.168.0.0/24 network:
/home/shared 192.168.0.*(rw,insecure)
```

Note

In the exports file example above, the insecure option allows clients to use ports above port 1024 to make mount requests. Some NFS clients require this because they do not have access to NFS-reserved ports.

3. On the NFS server, restart the nfs-server service, or if it is already running, you can simply export the shared directory as follows:

```
# exportfs -a -r -v
```

4. On the NFS server, make sure that the appropriate ports are open on the firewall. See the section "Securing Your NFS Server" earlier in this chapter for details.
5. On the NFS client system, add an entry to the /etc/auto.master file that identifies the mount point where you want the remote NFS directory to be mounted and a file (of your choosing) where you will identify the location of the remote NFS directory. I added this entry to the auto.master file:

```
/home/remote /etc/auto.joe
```

6. On the NFS client system, add an entry to the file you just noted (/etc/auto.joe is what I used) that contains an entry like the following:

```
joe       -rw     mynfs.example.com:/home/shared/joe
```

7. On the NFS client system, restart the autofs service:

```
# systemctl restart autofs.service
```

8. On the NFS client system, create a user named joe using the useradd command. For that command line, you need to get the UID for joe on the server (507 in this example) so that joe on the client system owns the files from joe's NFS home directory. When you run the following command, the joe user account is created, but you will see an error message stating that the home directory already exists (which is correct):

```
# useradd -u 507 -c "Joe Smith" -d /home/remote/joe joe
# passwd joe
Changing password for user joe.
New password: ********
Retype new password: ********
```

9. On the NFS client system, log in as joe. If everything is working properly, when joe logs in and tries to access his home directory (/home/remote/joe), the directory /home/share/joe should be mounted from the mynfs.example.com server. The NFS directory was both shared and mounted as read/write with ownership to UID 507 (joe on both systems), so the user joe on the local system should be able to add, delete, change, and view files in that directory.

After joe logs off (actually, when he stops accessing the directory) for a time-out period (10 minutes, by default), the directory is unmounted.

Unmounting NFS filesystems

After an NFS filesystem is mounted, unmounting it is simple. You use the umount command with either the local mount point or the remote filesystem name. For example, here are two ways that you could unmount maple:/stuff from the local directory /mnt/maple:

```
# umount maple:/stuff
# umount /mnt/maple
```

Either form works. If maple:/stuff is mounted automatically (from a listing in /etc/fstab), the directory is remounted the next time you boot Linux. If it was a temporary mount (or listed as noauto in /etc/fstab), it isn't remounted at boot time.

Tip

The command is umount, not unmount. This is easy to get wrong.

If you get the message device is busy when you try to unmount a filesystem, it means that the unmount failed because the filesystem is being accessed. Most likely, one of the directories in the NFS filesystem is the current directory for your shell (or the shell of someone else on your system). The other possibility is that a command is holding a file

open in the NFS filesystem (such as a text editor). Check your Terminal windows and other shells, and then `cd` out of the directory if you are in it, or just close the Terminal windows.

If an NFS filesystem doesn't unmount, you can force it (`umount -f /mnt/maple`) or unmount and clean up later (`umount -l /mnt/maple`). The `-l` option is usually the better choice because a forced unmount can disrupt a file modification that is in progress. Another alternative is to run `fuser -v` *`mountpoint`* to see what users are holding your mounted NFS share open and then `fuser -k` *`mountpoint`* to kill all of those processes.

Summary

Network File System (NFS) is one of the oldest computer file sharing products in existence today. It is still the most popular for sharing directories of files between UNIX and Linux systems. NFS allows servers to designate specific directories to make available to designated hosts and then allows client systems to connect to those directories by mounting them locally.

NFS can be secured using firewall (`iptables`) rules, TCP wrappers (to allow and deny host access), and SELinux (to confine how file sharing protocols can share NFS resources). Although NFS was inherently insecure when it was created (data is shared unencrypted and user access is fairly open), features in NFS version 4 have helped improve the overall security of NFS.

This NFS chapter is the last of the book's server chapters. Chapter 21, "Troubleshooting Linux," covers a wide range of desktop and server topics as it helps you understand techniques for troubleshooting your Linux system.

Exercises

Exercises in this section take you through tasks related to configuring and using an NFS server in Linux. If possible, have two Linux systems available that are connected on a local network. One of those Linux systems will act as an NFS server while the other will be an NFS client.

To get the most from these exercises, I recommend that you don't use a Linux server that has NFS already up and running. You can't do all of the exercises here without disrupting an NFS service that is already running and sharing resources.

See Appendix B for suggested solutions.

1. On the Linux system you want to use as an NFS server, install the packages needed to configure an NFS service.
2. On the NFS server, list the documentation files that come in the package that provides the NFS server software.

3. On the NFS server, determine the name of the NFS service and start it.
4. On the NFS server, check the status of the NFS service you just started.
5. On the NFS server, create the `/var/mystuff` directory and share it from your NFS server with the following attributes: available to everyone, read-only, and the root user on the client has root access to the share.
6. On the NFS server, make sure that the share you created is accessible to all hosts by opening TCP wrappers, `iptables`, and SELinux.
7. On a second Linux system (NFS client), view the shares available from the NFS server. (If you don't have a second system, you can do this from the same system.) If you do not see the shared NFS directory, go back to the previous question and try again.
8. On the NFS client, create a directory called `/var/remote` and temporarily mount the `/var/mystuff` directory from the NFS server on that mount point.
9. On the NFS client, unmount `/var/remote`, add an entry so that the same mount is done automatically when you reboot (with a `bg` mount option), and test that the entry you created is working properly.
10. From the NFS server, copy some files to the `/var/mystuff` directory. From the NFS client, make sure that you can see the files just added to that directory and make sure that you can't write files to that directory from the client.

CHAPTER 21

Troubleshooting Linux

IN THIS CHAPTER

Troubleshooting boot loaders

Troubleshooting system initialization

Fixing software packaging problems

Checking network interface issues

Dealing with memory problems

Using rescue mode

In any complex operating system, lots of things can go wrong. You can fail to save a file because you are out of disk space. An application can crash because the system is out of memory. The system can fail to boot up properly for, well, lots of different reasons.

In Linux, the dedication to openness, and the focus on making the software run with maximum efficiency, has led to an amazing number of tools that you can use to troubleshoot every imaginable problem. In fact, if the operating system isn't working as you would like, you even have the ultimate opportunity to rewrite the code yourself (although I don't cover how to do that here).

This chapter takes on some of the most common problems that you can run into on a Linux system, and it describes the tools and procedures that you can use to overcome those problems. Topics are broken down by areas of troubleshooting, such as the boot process, software packages, networking, memory issues, and rescue mode.

Boot-Up Troubleshooting

Before you can begin troubleshooting a running Linux system itself, that system needs to boot up. For a Linux system to boot up, a series of things has to happen. A Linux system installed directly on a PC architecture computer goes through the following steps to boot up:

- Turning on the power
- Starting the hardware (from BIOS or UEFI firmware)
- Finding the location of the boot loader and starting it

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

- Choosing an operating system from the boot loader
- Starting the kernel and initial RAM disk for the selected operating system
- Starting the initialization process (`init` or `systemd`)
- Starting all of the services associated with the selected level of activity (runlevel or default target)

The exact activities that occur at each of these points have undergone a transformation in recent years. Boot loaders are changing to accommodate new kinds of hardware. The initialization process is changing so that services can start more efficiently, based on dependencies and in reaction to the state of the system (such as what hardware is plugged in or what files exist) rather than a static boot order.

Troubleshooting the Linux boot process begins when you turn on your computer, and it ends when all of the services are up and running. At that point, typically a graphical or text-based login prompt is available from the console, ready for you to log in.

After reading the short descriptions of startup methods, go to the section "Starting from the firmware (BIOS or UEFI)" in order to understand what happens at each stage of the boot process and where you might need to troubleshoot. Because the general structure of the Linux boot process is the same for the three Linux systems featured here (Fedora, RHEL, and Ubuntu), I will go through the boot process only once, but I will describe the differences among them as I go.

Understanding Startup Methods

It's up to the individual Linux distribution how the services associated with the running Linux system are started. After the boot loader starts the kernel, how the rest of the activities (mounting filesystems, setting kernel options, running services, and so on) are done is all managed by the initialization process.

As I describe the boot process, I focus on two different types of initialization: System V `init` and `systemd`.

Starting with System V init scripts

The System V `init` facility consists of the `init` process (the first process to run after the kernel itself), an `/etc/inittab` file that directs all startup activities, and a set of shell scripts that starts each of the individual services. The first Fedora releases, and up to RHEL 5, used the System V `init` process. RHEL 6 contains a sort of hybrid of System V `init`, with the `init` process itself replaced by the Upstart `init` process.

System V `init` was developed for UNIX System V at AT&T in the mid-1980s when UNIX systems first incorporated the startup of network interfaces and the services connected to them. It has been supplanted only over the past decade by Upstart and `systemd` to better suit the demands of modern operating systems.

In System V `init`, sets of services are assigned to what is referred to as *runlevels*. For example, the multi-user runlevel can start basic system services, network interfaces,

and network services. Single-user mode just starts enough of the basic Linux system so that someone can log in from the system console without starting network interfaces or services.

After a System V `init` system is up and running, you can use commands such as `reboot`, `shutdown`, and `init` to change runlevels. You can use commands such as `service` and `chkconfig` to start/stop individual services or enable/disable services, respectively.

The System V `init` scripts are set to run in a specific order, with each script having to complete before the next can start. If a service fails, there is no provision for that service to restart automatically. In contrast, `systemd` was designed to address these and other System V `init` shortcomings.

Starting with systemd

The `systemd` facility is quickly becoming the present and future of the initialization process for many Linux systems. It was adopted in Fedora 15 and in RHEL 7 and replaced Upstart in Debian and Ubuntu 15.04. Although `systemd` is more complex than System V `init`, it also offers many more features, such as these:

Targets Instead of runlevels, `systemd` focuses on targets. A *target* can start a set of services as well as create or start other types of units (such as directory mounts, sockets, swap areas, and timers).

System V compatibility There are targets that align with System V runlevels, if you are used to dealing with runlevels. For example, `graphical.target` aligns with runlevel 5 while `multi-user.target` is essentially runlevel 3. However, there are many more targets than runlevels, giving you the opportunity to manage sets of units more finely. Likewise, `systemd` supports System V `init` scripts and commands, such as `chkconfig` and `service` for manipulating those services if System V `init` services happen to be installed.

Dependency-based startup When the system starts up, any service in the default target (`graphical.target` for desktops and `multi-user.target` for most servers) that has had its dependencies met can start. This feature can speed up the boot process by ensuring that a single stalled service doesn't stall other services from starting if they don't need the stalled service.

Resource usage With `systemd`, you can use `cgroups` to limit how much of your system's resources are consumed by a service. For example, you can limit the amount of memory, CPU, or other resources an entire service can consume, so a runaway process or a service that spins off an unreasonable number of child processes cannot consume more than the entire service is allowed.

When a `systemd`-enabled Linux system starts up, the first running process (PID 1) is the `systemd` daemon (instead of the `init` daemon). Later, the primary command for managing `systemd` services is the `systemctl` command. Managing `systemd` journal (log) messages is done with the `journalctl` command. You also have the ability to use old-style System V `init` commands such as `init`, `poweroff`, `reboot`, `runlevel`, and `shutdown` to manage services.

Starting from the firmware (BIOS or UEFI)

When you physically turn on a computer, firmware is loaded to initialize the hardware and find an operating system to boot. On PC architectures, that firmware has traditionally been referred to as *BIOS (Basic Input Output System)*. In recent years, a new type of firmware called *UEFI (Unified Extensible Firmware Interface)* has become available to replace BIOS on some computers. The two are mutually exclusive.

UEFI was designed to allow a secure boot feature, which can be used to ensure that only operating systems whose components have been signed can be used during the boot process. UEFI can still be used with non-signed operating systems by disabling the secure boot feature.

For Ubuntu, secure boot was first supported in 12.04.2. RHEL 7 and later versions also officially support secure boot. The main job of BIOS and UEFI firmware is to initialize the hardware and then hand off control of the boot process to a boot loader. The *boot loader* then finds and starts the operating system. After an operating system is installed, you should typically just let the firmware do its work and not interrupt it.

There are, however, occasions when you want to interrupt the firmware. For this discussion, we focus on how BIOS generally works. Right after you turn on the power, you should see a BIOS screen that usually includes a few words noting how to go into Setup mode and change the boot order. If you press the function key noted (often F1, F2, or F12) to choose one of those two items, here's what you can do:

Setup utility The setup utility lets you change settings in the BIOS. These settings can be used to enable or disable certain hardware components or turn on or off selected hardware features.

Boot order Computers are capable of starting an operating system, or more specifically, a boot loader that can start an operating system, from several different devices attached to the computer. Those devices can include a CD drive, DVD drive, hard disk, USB driver, or network interface card. The boot order defines the order in which those devices are checked. By modifying the boot order, you can tell the computer to ignore the default boot order temporarily and try to boot from the device that you select.

For my Dell workstation, after I see the BIOS screen, I immediately press the F2 function key to go into Setup or F12 to change the boot order temporarily. The next sections explore what you can troubleshoot from the Setup and Boot Order screens.

Troubleshooting BIOS setup

As I already noted, you can usually let the BIOS start without interruption and have the system boot up to the default boot device (probably the hard drive). However, here are some instances when you may want to go into Setup mode and change something in the BIOS:

To see an overview of your hardware If your troubleshooting problem is hardware related, the BIOS setup is a great place to start examining your system. The Setup

screen tells you the type of system, its BIOS version, its processors, its memory slots and types, whether it is 32-bit or 64-bit, which devices are in each slot, and many details about the types of devices attached to the system.

If you can't get an operating system booted at all, the BIOS Setup screen may be the only way to determine the system model, processor type, and other information you'll need to search for help or call for support.

To disable/enable a device Most devices connected to your computer are enabled and made available for use by the operating system. To troubleshoot a problem, you may need to disable a device.

For example, let's say that your computer has two network interface cards (NICs). You want to use the second NIC to install Linux over a network, but the installer keeps trying to use the first NIC to connect to the network. You can disable the first NIC so that the installer doesn't even see it when it tries to connect to the network. Or, you can keep the NIC visible to the computer but simply disable the NIC's ability to PXE boot.

Maybe you have an audio card, and you want to disable the integrated audio on the motherboard. That can be done in the BIOS as well. Conversely, sometimes you want to enable a device that has been disabled. Perhaps you were given a computer that had a device disabled in the BIOS. From the operating system, for example, it may look like you don't have front USB ports or a CD drive. Looking at the BIOS tells you whether those devices are not available simply because they have been disabled in the BIOS.

To change a device setting Sometimes, the default settings that come in your BIOS don't work for your situation. You might want to change the following settings in the BIOS:

NIC PXE boot settings Most modern NICs are capable of booting from servers found on the network. If you need to do that, and you find that the NIC doesn't come up as a bootable device on your Boot Order screen, you may have to enable that feature in the BIOS.

Virtualization settings If you want to run a Linux system as a virtual host, the computer's CPU must include Intel Virtual Technology or AMD Secure Virtual Machine (SVM) support. It is possible, however, that even if your CPU comes with this support, it may not be enabled in the BIOS. To enable it, go to the BIOS Setup screen and look for a Virtualization selection (possibly under the Performance category). Make sure that it is set to On.

Troubleshooting boot order

Depending on the hardware attached to your computer, a typical boot order might boot a CD/DVD drive first, then the hard drive, then a USB device, and finally the network interface card. The BIOS would go to each device, looking for a boot loader in the device's master boot record. If the BIOS finds a boot loader, it starts it. If no boot loader is located, the BIOS moves on to the next device until all are tried. If no boot loader is found, the computer fails to boot.

One problem that could occur with the boot order is that the device you want to boot may not appear in the boot order at all. In that case, going to the Setup screen, as described in the previous section, either to enable the device or change a setting to make it bootable, may be the thing to do.

If the device from which you want to boot does appear in the boot order, typically you just have to move the arrow key to highlight the device you want and press Enter. The following are reasons for selecting your own device to boot:

Rescue mode If Linux does not boot from the hard disk, selecting the CD drive or a USB drive allows you to boot to a rescue mode (described later in this chapter) that can help you repair the hard disk on an unbootable system. See the section "Troubleshooting in Rescue Mode" later in this chapter for further information.

Fresh install Sometimes, the boot order has the hard disk listed first. If you decide that you need to do a fresh install of the operating system, you need to select the boot device that is holding your installation medium (CD, DVD, USB drive, or NIC).

Assuming that you get past any problems you have with the BIOS, the next step is for the BIOS to start the boot loader.

Troubleshooting the GRUB boot loader

Typically, the BIOS finds the master boot record on the first hard disk and begins loading that boot loader in stages. Chapter 9, "Installing Linux," describes the GRUB boot loader that is used with most modern Linux systems, including RHEL, Fedora, and Ubuntu. The GRUB boot loader in RHEL 6, described here, is an earlier version than the GRUB 2 boot loader included with RHEL 7 and later, Fedora and Ubuntu. (Later in this chapter, I introduce you to the GRUB 2 boot loader as well.)

In this discussion, I am interested in the boot loader from the perspective of what to do if the boot loader fails or what ways you might want to interrupt the boot loader to change the behavior of the boot process.

The GRUB Legacy boot loader

Here are a few ways in which the boot loader might fail in RHEL 6 and some ways that you can overcome those failures:

Could not locate active partition When a boot loader is installed on a storage medium, the partition is usually marked as bootable. If you see this message, it means that no bootable partition was found. If you feel sure that the boot loader is on the disk, try using the `fdisk` command (probably from rescue media) to make the partition bootable and try again. See the section "Partitioning Hard Disks" of Chapter 12, "Managing Disks and Filesystems," for more information on the `fdisk` command.

Selected boot device not available You might see a message like this when the master boot record has been deleted from the hard drive, or it may just be that

the contents of the hard disk expect to be loaded from another boot loader, such as a boot CD. First, try seeing if the system will boot from other media. If it turns out that the master boot record was erased, you can try booting rescue media to attempt to recover the contents of the disk. However, if the master boot record is lost, it is possible that other data on the disk either is also erased or would require disk forensics to find it. If the master boot record was simply overwritten (which could happen if you installed another operating system on a different disk partition), it may be possible to reinstall the master boot record from rescue mode (described in the section "Troubleshooting in Rescue Mode" later in this chapter).

Text-based GRUB prompt appears It is possible for the BIOS to start GRUB and go straight to a GRUB prompt with no operating system selections available. This probably means that the master boot record portion of GRUB was found, but when GRUB looked on the hard drive to find the next stage of the boot process and a menu of operating systems to load, it could not find them. Sometimes this happens when the BIOS detects the disks in the wrong order and looks for the `grub.conf` file on the wrong partition.

One workaround to this problem, assuming that `grub.conf` is on the first partition of the first disk, is to list the contents of this file and enter the `root`, `kernel`, and `initrd` lines manually. To list the file, enter **`cat (hd0,0)/grub/grub.conf`**. If that doesn't work, try **`hd0,1`** to access the next partition on that disk (and so on) or **`hd1,0`** to try the first partition of the next disk (and so on). When you find the lines representing the `grub.conf` file, manually type the `root`, `kernel`, and `initrd` lines for the entry that you want (replacing the location of the hard drive you found on the root line), and then type **`boot`**. The system should start up, and you can manually fix your boot loader files. See Chapter 9 for more information on the GRUB boot loader.

If the BIOS finds the boot loader in the master boot record of the disk and that boot loader finds the GRUB configuration files on the disk, the boot loader starts a countdown of a few seconds, as determined by the `timeout` value in `/boot/grub/grub.conf` (it's 5 seconds in RHEL 6). During that countdown, you can interrupt the boot loader (before it boots the default operating system) by pressing any key.

When you interrupt the boot loader, you should see a menu of available entries to boot. Those entries can represent different available kernels to boot. However, they may also represent totally different operating systems (such as Windows, BSD, or Ubuntu).

Here are some reasons to interrupt the boot process from the boot menu to troubleshoot Linux:

To start in a different runlevel RHEL 6 systems typically start in runlevel 3 (boot to text prompt) or 5 (boot to graphical interface). You can override the default runlevel by highlighting the kernel line from the boot menu and putting a different runlevel number at the end. To do this, highlight the operating system entry you want,

type **e**, highlight the kernel, type **e**, and add the new runlevel to the end of the line (for example, add a space and the number **1** to go into single-user mode). Then press Enter, and type **b** to boot the new entry.

Why would you boot to different runlevels for troubleshooting? Runlevel 1 bypasses authentication, so you boot directly to a root prompt. This is good if you have forgotten the root password and need to change it (type **passwd** to do that). Runlevel 3 bypasses the start of your desktop interface. Go to runlevel 3 if you are having problems with your video driver and want to try to debug it without it trying to start up the graphical interface automatically.

To select a different kernel When RHEL installs a new kernel, it always keeps at least one older kernel around. If the new kernel fails, you can always boot the previous, presumably working, older kernel. To boot a different kernel from the GRUB menu, just use the arrow key to highlight the one you want, and press Enter to boot it.

To select a different operating system If you happen to have another operating system installed on your hard drive, you can select to boot that one instead of RHEL. For example, if you have Fedora and RHEL on the same computer, and RHEL isn't working, you can boot to Fedora, mount the RHEL filesystems that you need, and try to fix the problem.

To change boot options On the kernel line, notice that there are lots of options being passed to the kernel. At the very least, those options must contain the name of the kernel (such as `vmlinuz-2.6.32.el6.x86_64`) and the partition containing the root filesystem (such as `/dev/mapper/abc-root`). If you want, you can add other options to the kernel line.

You may want to add kernel options to add features to the kernel or temporarily disable hardware support for a particular component. For example, adding `init=/bin/bash` causes the system to bypass the `init` process and go straight to a shell (similar to running `init 1`). In RHEL 7, adding `1` as a kernel option is not supported, so `init=/bin/bash` is the best way to get into a sort of single-user mode. Adding `nousb` would temporarily disable the USB ports (presumably to make sure that anything connected to those ports would be disabled as well).

Assuming that you have selected the kernel you want, the boot loader tries to run the kernel, including the content of the initial RAM disk (which contains drivers and other software needed to boot your particular hardware).

GRUB 2 Boot loader

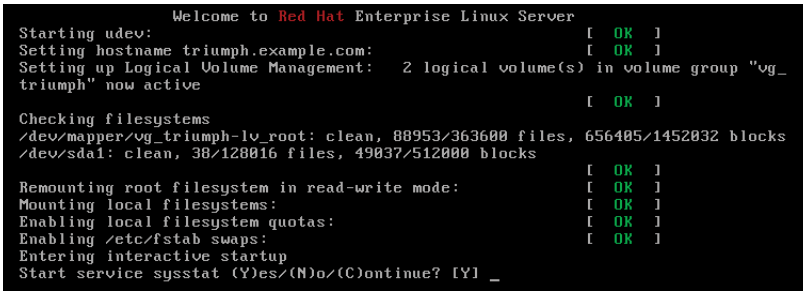
Techniques for troubleshooting Linux from the GRUB 2 boot prompt are similar to those in the legacy GRUB boot prompt. Follow these instructions for interrupting the GRUB boot prompt for the most recent Fedora, RHEL, and Ubuntu systems:

1. After you turn on your computer and just after you see the BIOS screen, press any key (such as the up arrow). You should see several menu items representing different kernels to boot.

2. From the available entries, the default is to boot the latest available kernel, which should be highlighted and ready to boot. However, you can choose a different entry if any of the following applies:
   - The current kernel is broken, and you want to choose an older kernel that you know is working.
   - You want to run an entry that represents a totally different operating system that is installed on your disk.
   - You want to run a rescue kernel.
3. Assuming you want to run a Linux kernel, highlight the kernel you want (using up and down arrows) and type **e**. You will see commands that are run to start the system, as shown in Figure 21.1.

FIGURE 21.1

Interrupt the GRUB bootloader to modify the boot process.



4. To add arguments to the kernel, move your cursor to the end of the line beginning with "linux" and type the arguments you want. See `https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt` for a list of kernel parameters. Here are two examples:
   - **selinux**: If there is a problem with SELinux that makes your system unusable, you can disable it as follows:

     ```
     selinux=0
     ```

   - **smt**: Simultaneous Multithreading (smt) allows a single CPU core to execute multiple threads. To fix some microprocessor flaws, you need to turn off that feature. You can do that at boot time by passing the `smt=1` or `nosmt` argument on the kernel command line.
5. Once you are done adding arguments, press Ctrl-x to boot the system with the kernel arguments you added.

Starting the kernel

After the kernel starts, there isn't much to do except to watch out for potential problems. For RHEL, you see a Red Hat Enterprise Linux screen with a slow-spinning icon. If you want to watch messages detailing the boot process scroll by, press the Esc key.

At this point, the kernel tries to load the drivers and modules needed to use the hardware on the computer. The main things to look for at this point (although they may scroll by quickly) are hardware failures that may prevent some feature from working properly. Although much rarer than it used to be, there may be no driver available for a piece of hardware, or the wrong driver may get loaded and cause errors.

In addition to scrolling past on the screen, messages produced when the kernel boots are copied to the *kernel ring buffer*. As its name implies, the kernel ring buffer stores kernel messages in a buffer, throwing out older messages after that buffer is full. After the computer boots up completely, you can log into the system and enter the following command to capture these kernel messages in a file (then view them with the `less` command):

```
# dmesg > /tmp/kernel_msg.txt
# less /tmp/kernel_msg.txt
```

I like to direct the kernel messages into a file (choose any name you like) so that the messages can be examined later or sent to someone who can help debug any problems. The messages appear as components are detected, such as your CPU, memory, network cards, hard drives, and so on.

In Linux systems that support `systemd`, kernel messages are stored in the `systemd` journal. So, besides the `dmesg` command, you can also run `journalctl` to see kernel messages from boot time to the present. For example, here are kernel messages output from a RHEL 7 system:

```
# journalctl -k
-- Logs begin at Sat 2019-11-23 10:36:31 EST,
      end at Sun 2019-12-08 08:09:42 EST. --
Nov 23 10:36:31 rhel81 kernel: Linux version 4.18.0-147.0.3.el8_1.x86_64
    (mockbuild@x86-vm-09.build.eng.bos.redhat.com)
       (gcc version 8.3.1 20190507 (Red Hat 8.3.1-4)
          (GCC)) #1 SMP Mon Nov 11 12:58:36 UTC 2019
Nov 23 10:36:31 rhel81 kernel: Command line:
  BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-147.0.3.el8_1.x86_64
    root=/dev/mapper/rhel-root ro resume=/dev/mapper/rhel-swap
       rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
...
Nov 23 10:36:31 rhel81 kernel: Hypervisor detected: KVM
Nov 23 10:36:31 rhel81 kernel: kvm-clock: Using msrs 4b564d01 ...
```

What you want to look for are drivers that fail to load or messages that show certain features of the hardware failing to be enabled. For example, I once had a TV tuner card (for watching television on my computer screen) that set the wrong tuner type for the card that was detected. Using information about the TV card's model number and the type of failure, I found

that passing an option to the card's driver allowed me to try different settings until I found the one that matched my tuner card.

In describing how to view kernel startup messages, I have gotten ahead of myself a bit. Before you can log in and see the kernel messages, the kernel needs to finish bringing up the system. As soon as the kernel is done initially detecting hardware and loading drivers, it passes off control of everything else that needs to be done to boot the system to the initialization system.

Troubleshooting the initialization system

The first process to run on a system where the kernel has just started depends on the initialization facility that system is using. For System V `init`, the first process to run is the `init` process. For `systemd`, the first process is `systemd`. Depending on which you see running on your system (type **`ps -ef | head`** to check), follow either the System V or `systemd` descriptions below. RHEL 6, which contains a hybrid of Upstart and System V `init`, is used in the example of System V initialization.

Troubleshooting System V initialization

Up to a few years ago, most Linux systems used System V `init` to initialize the services on the Linux system. In RHEL 6, when the kernel hands off control of the boot process to the `init` process, the `init` process checks the `/etc/inittab` file for directions on how to boot the system.

The `inittab` file tells the `init` process what the default runlevel is and then points to files in the `/etc/init` directory to do things such as remap some keystrokes (such as Ctrl+Alt+Delete to reboot the system), start virtual consoles, and identify the location of the script for initializing basic services on the system: `/etc/rc.sysinit`.

When you're troubleshooting Linux problems that occur after the `init` process takes over, two likely culprits are the processing by the `rc.sysinit` file and the runlevel scripts.

Troubleshooting rc.sysinit

As the name implies, the `/etc/rc.sysinit` script initializes many basic features on the system. When that file is run by `init`, `rc.sysinit` sets the system's hostname, sets up the `/proc` and `/sys` filesystems, sets up SELinux, sets kernel parameters, and performs dozens of other actions.

One of the most critical functions of `rc.sysinit` is to get the storage set up on the system. In fact, if the boot process fails during processing of `rc.sysinit`, in all likelihood the script was unable to find, mount, or decrypt the local or remote storage devices needed for the system to run.

The following is a list of some common failures that can occur from tasks run from the `rc.sysinit` file and ways of dealing with those failures.

Local mounts fail: If an entry in the `/etc/fstab` fails to mount, the boot process ends before runlevel services start. This typically happens when you add an entry to the `/etc/fstab` that has a mistake in it but you neglected to test it before you rebooted. When the `fstab` file fails, you are dropped to a shell for the root user with the root filesystem mounted read-only. To fix the problem, you need to remount the root filesystem, correct the `fstab` file, mount the filesystem entry to make sure that it now works, and reboot. Here's what that sequence of commands looks like:

```
# mount -o remount,rw /
# vim /etc/fstab
# mount -a
# reboot
```

NOTE

The `vim` command is used particularly when editing the `/etc/fstab` file because it knows the format of that file. When you use `vim`, the columns are in color and some error checking is done. For example, entries in the Mount Options field turn green when they are valid and black when they are not.

Hostname not set If your hostname is not set properly, you can check through the processing of `rc.sysinit` to see what might have gone wrong. To set the system's hostname, `rc.sysinit` uses the value of the `HOSTNAME=` in the `/etc/sysconfig/network` file. If that is not set, the name `localhost` is used instead. The hostname value can also be acquired from the DHCP server.

Cannot decrypt filesystem The `rc.sysinit` script looks in the `/etc/crypttab` file for information needed to decrypt encrypted filesystems. If that file becomes corrupted, you may need to find a backup of the file to be able to decrypt your filesystem. If you are prompted for a password and you don't know what that is, you might be out of luck.

Other features are set up by the `rc.sysinit` file as well. The `rc.sysinit` script sets the SELinux mode and loads hardware modules. The script constructs software RAID arrays and sets up Logical Volume Management volume groups and volumes. Troubles occurring in any of these areas are reflected in error messages that appear on the screen after the kernel boots and before runlevel processes start up.

Troubleshooting runlevel processes

In Red Hat Enterprise Linux 6.*x* and earlier, when the system first comes up, services are started based on the default runlevel. There are seven different runlevels, from 0 to 6. The default runlevel is typically 3 (for a server) or 5 (for a desktop). Here are descriptions of the runlevels in Linux systems up to RHEL 6:

0: Shutdown runlevel. All processes are stopped, and the computer is powered down.

1: Single-user runlevel. Only those processes that are needed to boot the computer (including mounting all filesystems) and have the system available from the console

are run. Networking and network services are not started. This runlevel bypasses normal authentication and boots up to a root user prompt (called `sulogin`). If you boot up to this mode, you can use it to become root user immediately in order to change a forgotten root password. (You could also use the word `single` instead of `1` to get to single-user runlevel. The difference between `single` and `1` is that `single` does not start scripts in the `/etc/rc1.d` directory.)

2: **Multiuser runlevel**. This runlevel is rarely used today. The original meaning of this runlevel has been lost. Early UNIX systems used this runlevel to start `tty` processes for systems where there were multiple dumb terminals connected to the system for people to use. This allowed many people to access a system simultaneously from character-based terminals (lots of people working from a shell with no graphical interface). Network interfaces were not started, usually because always-up network interfaces were not common. These days, runlevel 2 usually starts network interfaces, although not all network services are started.

3: **Multiuser plus networking runlevel**. This runlevel is typically used on Linux servers that do not boot up to a graphical interface but rather just a plain text prompt at the console. The network is started, as are all network services. A graphical desktop environment may or may not be installed (typically not) on machines that boot to runlevel 3, but the graphical environments must be started after boot time to be used.

4: **Undefined**. This runlevel tends to start the same services as runlevel 3. It can be used if you want to have different services available from runlevel 3 and runlevel 4. This runlevel is typically not used. Instead, runlevel 3 or runlevel 5 is used to boot to, with an administrator simply turning services on or off as required for the running system.

5: **Multiuser, networking, plus graphical interface runlevel**. This is the runlevel generally used with desktop Linux systems. It generally starts networking and all networked services; plus, it launches a graphical login prompt at the console. When the users log in, they see a graphical desktop environment.

6: **Reboot runlevel**. This is like runlevel 0 in that it brings down all services and stops all processes. However, runlevel 6 then starts the system back up again.

Runlevels are meant to set the level of activity on a Linux system. A default runlevel is set in the `/etc/inittab` file, but you can change the runlevel anytime you like using the `init` command. For example, as root, you might type **`init 0`** to shut down, **`init 3`** if you want to kill the graphical interface (from runlevel 5) but leave all other services up, or **`init 6`** to reboot.

Normal default runlevels (in other words, the runlevel to which you boot) are 3 (for a server) and 5 (for a desktop). Often, servers don't have desktops installed, so they boot to runlevel 3 so that they don't incur the processing overhead or the added security risks for having a desktop running on their web servers or file servers.

You can go either up or down with runlevels. For example, an administrator doing maintenance on a system may boot to runlevel 1 and then type **`init 3`** to boot up to the full

services needed on a server. Someone debugging a desktop may boot to runlevel 5 and then go down to runlevel 3 to try to fix the desktop (such as installing a new driver or changing the screen resolution) before typing **`init 5`** to return to the desktop.

The level of services at each runlevel is determined by the runlevel scripts that are set to start. There are `rc` directories for each runlevel: `/etc/rc0.d/`, `/etc/rc1.d/`, `/etc/rc2.d/`, `/etc/rc3.d/`, and so on. When an application has a startup script associated with it, that script is placed in the `/etc/init.d/` directory and then symbolically linked to a file in each */etc/rc?.d/* directory.

Scripts linked to each */etc/rc?.d* directory begin with either the letter `K` or `S`, followed by two numbers and the service name. A script beginning with `K` indicates that the service should be stopped, while one beginning with an `S` indicates that it should be started. The two numbers that follow indicate the order in which the service is started. Here are a few files that you might find in the `/etc/rc3.d/` directory, which are set to start up (with a description of each to the right):

`S01sysstat`: Start gathering system statistics.

`S08iptables`: Start `iptables` firewall.

`S10network`: Start network interfaces.

`S12rsyslog`: Start system logging.

`S28autofs`: Start automounter.

`S50bluetooth`: Start Bluetooth service.

`S55sshd`: Start the secure shell service.

`S58ntpd`: Start NTP time synchronization service.

`S85httpd`: Start the Apache web service.

`S90crond`: Start the `crond` service.

`S91smb`: Start the Samba service.

`S97rhnsd`: Start the Red Hat Network service.

`S99local`: Start user-defined local commands.

This example of a few services started from the `/etc/rc3.d` directory should give you a sense of the order in which processes boot up when you enter runlevel 3. Notice that the `sysstat` service (which gathers system statistics) and the `iptables` service (which creates the system's firewall) are both started before the networking interfaces are started. Those are followed by `rsyslog` (system logging service) and then the various networked services.

By the time the runlevel scripts start, you should already have a system that is basically up and running. Unlike some other Linux systems that start all of the scripts for runlevel 1, then 2, then 3, and so on, RHEL goes right to the directory that represents the runlevel, first stopping all services that begin with `K` and starting all those that begin with `S` in that directory.

As each `S` script runs, you should see a message saying whether the service started. Here are some things that might go wrong during this phase of system startup:

A service can fail. A service may require access to network interfaces to start properly or access to a disk partition that is not mounted. Most services time out, fail, and allow the next script to run. After you are able to log in, you can debug the service. Some techniques for debugging services include adding a debug option to the daemon process so that it spews more data into a log file or running the daemon process manually so that error messages come straight to your screen. See Chapter 15, "Starting and Stopping Services," for further information on starting services manually.

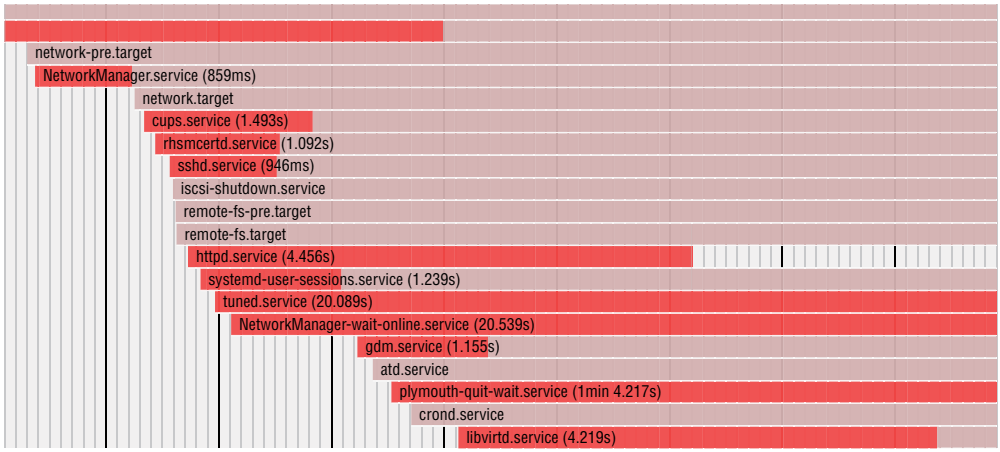
A service can hang. Some services that don't get what they need to start can hang indefinitely, keeping you from logging in to debug the problem. Some processes take longer to come up the first time after a fresh install, so you might want to wait for a few minutes to see if the script is still working and not just spinning forever.

If you cannot get past a hanging service, you can reboot into an *interactive startup mode*, where you are prompted before starting each service. To enter interactive startup mode in RHEL, reboot and interrupt the boot loader (press any key when you see the 5 second countdown). Highlight the entry you want to boot, and type **`e`**. Highlight the kernel line, and type **`e`**. Then add the word `confirm` to the end of the kernel line, press Enter, and type **`b`** to boot the new kernel.

Figure 21.2 shows an example of the messages that appear when RHEL boots up in interactive startup mode.

FIGURE 21.2

Confirm each service in RHEL interactive startup mode.



Most messages shown in Figure 21.2 are generated from `rc.sysinit`.

After the Welcome message, `udev` starts (to watch for new hardware that is attached to the system and load drivers as needed). The hostname is set, Logical Volume Management (LVM) volumes are activated, all filesystems are checked (with the added LVM volumes), any filesystems not yet mounted are mounted, the root filesystem is remounted read-write, and any LVM swaps are enabled. Refer to Chapter 12, "Managing Disks and Filesystems," for further information on LVM and other partition and filesystem types.

The last "Entering interactive startup" message tells you that `rc.sysinit` is finished and the services for the selected runlevel are ready to start. Because the system is in interactive mode, a message appears asking if you want to start the first service (`sysstat`). Type **`Y`** to start that service and go to the next one. After you see the broken service requesting to start, type **`N`** to keep that service from starting. If at some point you feel that the rest of the services are safe to start, type **`C`** to continue starting the rest of the services. After your system comes up with the broken services not started, you can go back and try to debug those individual services.

One last comment about startup scripts: The `/etc/rc.local` file is one of the last services to run at each runlevel. In runlevel 5, it is linked to `/etc/rc5.d/S99local`. Any command that you want to run every time your system starts up can be put in the `rc.local` file.

You might use `rc.local` to send an email message or run a quick iptables firewall rule when the system starts. In general, it's better to use an existing startup script or create a new one yourself (so you can manage the command or commands as a service). Know that the `rc.local` file is a quick and easy way to get some commands to run each time the system boots.

Troubleshooting systemd initialization

The latest versions of Fedora, RHEL, and Ubuntu use `systemd` instead of System V init as their initialization system. Although `systemd` is more complex than System V init, `systemd` also offers more ways to analyze what is happening during initialization.

Understanding the systemd boot process

When the `systemd` daemon (`/usr/lib/systemd/systemd`) is started after the kernel starts up, it sets in motion all of the other services that are set to start up. In particular, it keys off of the contents of the `/etc/systemd/system/default.target` file, as in this example:

```
# cat /etc/systemd/system/default.target
...
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
```

```
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
```

The `default.target` file is actually a symbolic link to a file in the `/lib/systemd/system` directory. For a server, it may be linked to the `multi-user.target` file; for a desktop, it is linked to the `graphical.target` file (as is shown here).

Unlike with the System V `init` facility, which just runs service scripts in alphanumeric order, the `systemd` service needs to work backward from the `default.target` to determine which services and other targets are run. In this example, `default.target` is a symbolic link to the `graphical.target` file. When you list the contents of that file, you can see the following:

- The `multi-user.target` is required to start first.
- The `display-manager.service` is started after that.

By continuing to discover what those two units require, you can find what else is required. For example, `multi-user.target` requires the `basic.target` (which starts a variety of basic services) and `display-manager.service` (which starts up the display manager, `gdm`) to launch a graphical login screen.

To see services the `multi-user.target` starts, list contents of the `/etc/systemd/system/multi-user.target.wants` directory, as in this example:

```
# ls /etc/systemd/system/multi-user.target.wants/
atd.service            ksm.service              rhsmcertd.service
auditd.service         ksmtuned.service         rpcbind.service
avahi-daemon.service   libstoragemgmt.service   rsyslog.service
chronyd.service        libvirtd.service         smartd.service
crond.service          mcelog.service           sshd.service
cups.path              mdmonitor.service        sssd.service
dnf-makecache.timer    ModemManager.service     tuned.service
firewalld.service      NetworkManager.service   vdo.service
irqbalance.service     nfs-client.target        vmtoolsd.service
kdump.service          remote-fs.target
```

These files are symbolic links to files that define what starts for each of those services. On your system, these may include remote shell (`sshd`), printing (`cups`), auditing (`auditd`), networking (`NetworkManager`), and others. Those links were added to that directory either when the package for a service was installed or when the service was enabled from a `systemctl enable` command.

Keep in mind that, unlike System V `init`, `systemd` can start, stop, and otherwise manage unit files that represent more than just services. It can manage devices, automounts, paths, sockets, and other things. After `systemd` has started everything, you can log into the system to investigate and troubleshoot any potential problems.

After you log in, running the `systemctl` command lets you see every unit file that `systemd` tried to start up. Here is an example:

```
# systemctl
UNIT                                          LOAD   ACTIVE SUB
      DESCRIPTION
proc-sys-fs-binfmt_misc.automount             loaded active waiting
      Arbitrary Executable File Formats File System
sys-devices-pc...:00:1b.0-sound-card0.device loaded active plugged
      631xESB/632xESB High Definition Audio Control
sys-devices-pc...:00:1d.2-usb4-4\x2d2.device loaded active plugged
      DeskJet 5550
...

-.mount                                    loaded active mounted   Root Mount
boot.mount                                 loaded active mounted   /boot
...
autofs.service                                loaded active running
      Automounts filesystems on demand
cups.service                                  loaded active running
      CUPS Scheduler
httpd.service                                 loaded failed failed
      The Apache HTTP Server
```

From the `systemctl` output, you can see whether any unit file failed. In this case, you can see that the `httpd.service` (your web server) failed to start. To investigate further, you can run `journalctl -u` for that service to see whether any error messages were reported:

```
# journalctl -u httpd.service
...
Dec 08 09:30:36 rhel81-bible systemd[1]: Starting The Apache HTTP
Server...
Dec 08 09:30:36 rhel81-bible httpd[16208]: httpd: Syntax error
   on line 105 of /etc/httpd/conf/httpd.conf:
   /etc/httpd/conf/httpd.conf:105: <Directory> was not closed.
Dec 08 09:30:36 rhel81-bible systemd[1]: httpd.service: Main process exited,
   code=exited, status=1/FAILURE
Dec 08 09:30:36 rhel81-bible systemd[1]: httpd.service:
   Failed with result 'exit-code'.
Dec 08 09:30:36 rhel81-bible systemd[1]:
   Failed to start The Apache HTTP Server.
```

From the output, you can see that there was a mismatch of the directives in the `httpd.conf` file (I failed to close a Directory section). After that was corrected, I could start the service (`systemctl start httpd`). If more unit files appear as failed, you can run the `journalctl -u` command again, using those unit filenames as arguments.

Analyzing the systemd boot process

To see exactly what happened during the boot process for a system using the `systemd` service, `systemd` provides the `systemd-analyze` tool. If you want to see if there

are services that are stalling, or you want to look for a place to put in your own `systemcd` service, you can use this command to analyze the entire startup process. Here are some examples:

```
# systemd-analyze time
Startup finished in 1.775s (kernel) + 21.860s (initrd)
    + 1min 42.414s (userspace) = 2min 6.051s
graphical.target reached after 1min 42.121s in userspace
```

The `time` option lets you see how long each phase of the startup process took, from the start of the kernel to the end of the default target. You can use `plot` to create an SVG graphic of each component of the startup process (I show `eog` here to display the output):

```
# systemd-analyze plot > /tmp/systemd-plot.svg
# eog /tmp/systemd-plot.svg
```

Figure 21.3 shows a small snippet of output from the much larger graphic.

FIGURE 21.3

Snippet from `systemd-analyze` startup plot



From this snippet, you can see services that start after the `NetworkManager.service` starts up. Parts in dark red show the time in which the service or target takes to start. If the service continues to run, that is shown in light red. In this example, the `httpd.service` failed after trying to start for 4.456 seconds. You can tell this because the bar to the right is white, showing that the service is not running. At this point, you could use the `journalctl` command, as described earlier, to debug the problem.

The next section describes how to troubleshoot issues that can arise with your software packages.

Troubleshooting Software Packages

Software packaging facilities (such as `yum` for RPM and `apt-get` for DEB packages) are designed to make it easier for you to manage your system software. (See Chapter 10, "Getting and Managing Software," for the basics on how to manage software packages.) Despite efforts to make it all work, however, sometimes software packaging can break.

NOTE

The `dnf` facility has replaced `yum` in recent Fedora and RHEL systems. This section often uses the `yum` command since it will work on both older and newer Fedora and RHEL systems in most cases, because `yum` is aliased to `dnf`. The `dnf` command is shown when it is explicitly required.

The following sections describe some common problems that you can encounter with RPM packages on a RHEL or Fedora system and how you can overcome those problems.

Sometimes, when you try to install or upgrade a package using the `yum` command, error messages tell you that the dependent packages that you need to do the installation you want are not available. This can happen on a small scale (when you try to install one package) or on a grand scale (where you are trying to update or upgrade your entire system).

Because of the short release cycles and larger repositories of Fedora and Ubuntu, inconsistencies in package dependencies are more likely to occur than they do in more stable, selective repositories (such as those offered by Red Hat Enterprise Linux). To avoid dependency failures, here are a few good practices that you can follow:

Use recent, well-tested repositories. There are thousands of software packages in Fedora. If you use the main Fedora repositories to install software from the current release, it is rare to have dependency problems.

When packages are added to the repository, as long as the repository maintainers run the right commands to set up the repository (and you don't use outside repositories), everything you need to install a selected package should be available. However, when you start using third-party repositories, those repositories may have dependencies on repositories that they can't control. For example, if a repository creates a new version of its own software that requires later versions of basic software (such as libraries), the versions that they need might not be available from the Fedora repository.

Consistently update your system. Running `dnf update` every night (or `yum update` on older systems) makes it less likely that you will encounter major dependency problems than if you update your system only every few months. In systems with a GNOME desktop, the Software window lets you check for and apply updates to installed software packages. On Fedora 22 and RHEL 8 (and later) systems, you can add AutoUpdates (`https://fedoraproject.org/wiki/AutoUpdates`). AutoUpdates automatically downloads updated packages when they are available and,

if configured, can install them as well. Another approach is to build a `cron` job to check for or run nightly updates. See the sidebar "Using `cron` for Software Updates" for details on how to do that.

Occasionally upgrade your system. Fedora and Ubuntu have new releases every 6 months. Fedora stops supplying updated packages for each version, 13 months after it is released. So, although you don't have to upgrade to the new release every 6 months, you should upgrade once a year or face possible dependency and security problems when Fedora stops supplying updates.

To get a whole new version of those distributions (such as Fedora 30 to Fedora 31), do the following:

1. Upgrade to the latest software for your current release:

```
# dnf upgrade --refresh -y
```

2. Install the dnf-plugin-system-upgrade plug-in:

```
# dnf install dnf-plugin-system-upgrade -y
```

3. Start upgrading to the new release:

```
# dnf system-upgrade download --releasever=31
```

4. Reboot to the upgrade process:

```
# dnf system-upgrade reboot
```

If you are looking for a stable system, Red Hat Enterprise Linux is a better bet because it provides updates for each major release for seven years or more.

Note

If you use the `apt-get` command in Ubuntu to update your packages, keep in mind that there are different meanings to the `update` and `upgrade` options in Ubuntu with `apt-get` than with the `dnf` or `yum` command (Fedora and RHEL).

In Ubuntu, `apt-get update` causes the latest packaging metadata (package names, version numbers, and so on) to be downloaded to the local system. Running `apt-get upgrade` causes the system to upgrade any installed packages that have new versions available, based on the latest downloaded metadata.

In contrast, every time that you run a `dnf` or `yum` command in Fedora or RHEL, the latest metadata about new packages for the current release is downloaded. When you then run `yum update`, you get the latest packages available for the current release of Fedora or RHEL. To go to the next release, you must run `dnf system-upgrade`, as described earlier.

If you encounter a dependency problem, here are a few things that you can do to try to resolve the problem:

Use stable repositories. For recent releases of well-known distributions (RHEL, Fedora, or Ubuntu, for example), dependency problems are rare and often fixed quickly. However, if you are relying on repositories for older releases or development-oriented repositories (such as Fedora's Rawhide repository), expect to find more dependency problems. Reinstalling or upgrading can often fix dependency problems.

Only use third-party apps and repositories when necessary. The further you are from the core of a Linux distribution, the more likely you are to have dependency problems someday. Always look in the main repositories for your distribution before you look elsewhere for a package or try to build one yourself.

Even if it works when you first install it, a package someone just handed to you might not have a way to be upgraded. A package from a third-party repository may break if the creators don't provide a new version when dependent packages change.

Solve kernel-related dependencies. If you get third-party RPM packages for such things as video cards or wireless network cards that contain kernel drivers and you install a later kernel, those drivers no longer work. The result might be that the graphical login screen doesn't start when the system boots or your network card fails to load, so you have no wireless networking.

Because most Linux systems keep the two most recent kernels, you can reboot, interrupt GRUB, and select the previous (still working) kernel from which you can boot. That gets your system up and running, with the old kernel and working drivers, while you look for a more permanent fix.

The longer-term solution is to get a new driver that has been rebuilt for your current kernel. Sites such as `rpmfusion.org` build third-party, non-open-source driver packages and upgrade those drivers when a new kernel is available. With the `rpmfusion.org` repository enabled, your system should pick up the new drivers when the new kernel is added.

As an alternative to sites such as `rpmfusion.org`, you can go straight to the website for the manufacturer and try to download its Linux drivers (Nvidia offers Linux drivers for its video cards), or if source code is available for the driver, you can try to build it yourself.

Exclude some packages from update. If you are updating lots of packages at once, you can exclude the packages that fail to get the others to work as you pursue the problem with the broken ones. Here's how to update all packages needing an upgrade, except for a package named *somepackage* (replace *somepackage* with the name of the package that you want to exclude):

```
# yum -y --exclude=somepackage update
```

Using `cron` for Software Updates

The `cron` facility provides a means of running commands at predetermined times and intervals. You can set the exact minute, hour, day, or month that a command runs. You can configure a command to run every five minutes, every third hour, or at a particular time on Friday afternoon.

If you want to use `cron` to set up nightly software updates, you can do that as the root user by running the `crontab -e` command. That opens a file using your default editor (`vi` command by default) that you can configure as a `crontab` file. Here's an example of what the `crontab` file you create might look like:

```
#  min   hour   day/month   month   day/week   command
   59    23     *           *       *          dnf -y update | mail root@localhost
```

A `crontab` file consists of five fields, designating day and time, and a sixth field, containing the command line to run. I added the comment line to indicate the fields. Here, the `dnf -y update` command is run, with its output mailed to the user `root@localhost`. The command is run at 59 minutes after hour 23 (11:59 p.m.). The asterisks (`*`) are required as placeholders, instructing `cron` to run the command on every day of the month, month, and day of the week.

When you create a `cron` entry, make sure that you either direct the output to a file or pipe the output to a command that can deal with the output. If you don't, any output is sent to the user that ran the `crontab -e` command (in this case, root).

In a `crontab` file, you can have a range of numbers or a list of numbers, or you can skip numbers. For example, `1, 5,` or `17` in the first field causes the command to be run 1, 5, and 17 minutes after the hour. An `*/3` in the second field causes the command to run every three hours (midnight, 3 a.m., 6 a.m., and so on). A `1-3` in the fourth field tells `cron` to run the command in January, February, and March. Days of the week and months can be entered as numbers or words.

For more information on the format of a `crontab` file, type **`man 5 crontab`**. To read about the `crontab` command, type **`man 1 crontab`**.

Fixing RPM databases and cache

Information about all of the RPM packages on your system is stored in your local RPM database. Although it happens much less often than it did with earlier releases of Fedora and RHEL, it is possible for the RPM database to become corrupted. This stops you from installing, removing, or listing RPM packages.

If you find that your `rpm` and `yum` commands are hanging or failing and returning an *`rpmdb open fails`* message, you can try rebuilding the RPM database. To verify that there is a problem in your RPM database, you can run the `yum check` command. Here is an example of what the output of that command looks like with a corrupted database:

```
# yum check
error: db4 error(11) from dbenv->open: Resource temporarily
unavailable
error: cannot open Packages index using db4 - Resource temporarily
     unavailable (11)
```

```
error cannot open Packages database in /var/lib/rpm
CRITICAL:yum.main:
Error: rpmdb open fails
```

The RPM database and other information about your installed RPM packages are stored in the /var/lib/rpm directory. You can remove the database files that begin with __db* and rebuild them from the metadata stored in other files in that directory.

Before you start, it's a good idea to back up the /var/lib/rpm directory. Then you need to remove the old __db* files and rebuild them. Type the following commands to do that:

```
# cp -r /var/lib/rpm /tmp
# cd /var/lib/rpm
# rm __db*
# rpm --initdb
```

New __db* files should appear after a few seconds in that directory. Try a simple rpm or yum command to make sure that the databases are now in order.

Just as RPM has databases of locally installed packages, the Yum facility stored information associated with Yum repositories in the local /var/cache/yum directory. With the introduction of dnf, the cache directory is now /var/cache/dnf. Cached data includes metadata, headers, packages, and yum plug-in data.

If there is ever a problem with the data cached by yum, you can clean it out. The next time that you run a yum command, necessary data is downloaded again. Here are a couple of reasons for cleaning out your yum cache:

Metadata is obsolete. The first time that you connect to a Yum repository (by downloading a package or querying the repository), metadata is downloaded to your system. The metadata consists of information on all of the available packages from the repository.

As packages are added and removed from the repository, the metadata has to be updated or your system will be working from old packaging information. By default, if you run a dnf command, dnf checks for new metadata if the old metadata is more than 48 hours old (or by however many minutes metadata_expire= is set to in the /etc/dnf/dnf.conf file).

If you suspect that the metadata is obsolete but the expire time has not been reached, you can run dnf clean metadata to remove all metadata, forcing new metadata to be uploaded with the next upload. Alternatively, you could run dnf makecache to get metadata from all repositories up to date.

You are running out of disk space. Normally, yum might cache as much as a few hundred megabytes of data in /var/cache/dnf directories. However, depending on the settings in your /etc/dnf/dnf.conf file (such as keepcache=1, which keeps all downloaded RPMs even after they are installed), the cache directories can contain multiple gigabytes of data.

To clean out all packages, metadata, headers, and other data stored in the /var/cache/dnf directory, type the following:

```
# yum clean all
```

At this point, your system gets up-to-date information from repositories the next time a `yum` command is run.

The next section covers information about network troubleshooting.

Troubleshooting Networking

With more and more of the information, images, video, and other content that we use every day now available outside of our local computers, a working network connection is required on almost every computer system. So, if you drop your network connection or can't reach the systems with which you wish to communicate, it's good to know that there are many tools in Linux for looking at the problem.

For client computers (laptops, desktops, and handheld devices), you want to connect to the network to reach other computer systems. On a server, you want your clients to be able to reach you. The following sections describe different tools for troubleshooting network connectivity for Linux client and server systems.

Troubleshooting outgoing connections

Let's say that you open your web browser but are unable to get to any website. You suspect that you are not connected to the network. Maybe the problem is with name resolution, but it may be with the connection outside of your local network.

To check whether your outgoing network connections are working, you can use many of the commands described in Chapter 14, "Administering Networking." You can test connectivity using a simple `ping` command. To see if name-to-address resolution is working, use `host` and `dig`.

The following sections cover problems that you can encounter with network connectivity for outgoing connections and what tools to use to uncover the problems.

View network interfaces

To see the status of your network interfaces, use the `ip` command. The following output shows that the loopback interface (`lo`) is up (so you can run network commands on your local system), but `eth0` (your first wired network card) is down (`state DOWN`). If the interface had been up, an `inet` line would show the IP address of the interface. Here, only the loopback interface has an `inet` address (127.0.0.1).

```
# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 state DOWN qlen
1000
    link/ether f0:de:f1:28:46:d9 brd ff:ff:ff:ff:ff:ff
```

By default in RHEL 8 and Fedora, network interfaces are now named based on how they are connected to the physical hardware. For example, in RHEL 8, you might see a network interface of enp11s0. That would indicate that the NIC is a wired Ethernet card (en) on PCI board 11 (p11) and slot 0 (s0). A wireless card would start with wl instead of en. The intention is to make the NIC names more predictable, because when the system is rebooted, it is not guaranteed which interfaces would be named eth0, eth1, and so on by the operating system.

Check physical connections

For a wired connection, make sure that your computer is plugged into the port on your network switch. If you have multiple NICs, make sure that the cable is plugged into the correct one. If you know the name of a network interface (`eth0`, `p4p1`, or other), to find which NIC is associated with the interface, enter **`ethtool -p eth0`** at the command line and look behind your computer to see which NIC is blinking (Ctrl+C stops the blinking). Plug the cable into the correct port.

If instead of seeing an interface that is down, the `ip` command shows no interface at all, check that the hardware isn't disabled. For a wired NIC, the card may not be fully seated in its slot or the NIC may have been disabled in the BIOS.

On a wireless connection, you may click the NetworkManager icon and not see an available wireless interface. Again, it could be disabled in the BIOS. However, on a laptop, check to see if there is a tiny switch that disables the NIC. I've seen several people shred their networking configurations only to find out that this tiny switch on the front or side of their laptops had been switched to the off position.

Check routes

If your network interface is up but you still can't reach the host you want to reach, try checking the route to that host. Start by checking your default route. Then try to reach the local network's gateway device to the next network. Finally, try to ping a system somewhere on the Internet:

```
# ip route show
default via 192.168.122.1 dev ens3 proto dhcp metric 100
192.168.122.0/24 dev ens3 proto kernel scope link src 192.168.122.194
metric 100
```

The default line shows that the default gateway is at address 192.168.122.1 and that the address can be reached over the `ens3` card. Because there is only the `ens3` interface here and only a route to the 192.168.122.0 network is shown, all communication not addressed to a host on the 192.168.122.0/24 network is sent through the default gateway (192.168.122.1). The default gateway is more properly referred to as a router.

To make sure that you can reach your router, try to ping it, as in this example:

```
# ping -c 2 192.168.122.1
PING 192.168.122.1 (192.168.122.1) 56(84) bytes of data.
64 bytes from 192.168.122.1: icmp_seq=1 ttl=64 time=0.757 ms
64 bytes from 192.168.122.1: icmp_seq=2 ttl=64 time=0.538 ms
```

```
--- 192.168.122.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 65ms
rtt min/avg/max/mdev = 0.538/0.647/0.757/0.112 ms
```

A "Destination Host Unreachable" message tells you that the router is either turned off or not physically connected to you (maybe the router isn't connected to the switch you share). If the ping succeeds and you can reach the router, the next step is to try an address beyond your router.

Try to ping a widely accessible IP address. For example, the IP address for the Google public DNS server is 8.8.8.8. Try to ping that (`ping -c2 8.8.8.8`). If that ping succeeds, your network is probably fine, and it is most likely your hostname-to-address resolution that is not working properly.

If you can reach a remote system but the connection is very slow, you can use the `traceroute` command to follow the route to the remote host. For example, this command shows each hop taken en route to `http://www.google.com`:

```
# traceroute www.google.com
```

The output shows the time taken to make each hop along the way to the Google site. Instead of `traceroute`, you can use the `mtr` command (`yum install mtr`) to watch the route taken to a host. With `mtr`, the route is queried continuously, so you can watch the performance of each leg of the journey over time.

Check hostname resolution

If you cannot reach remote hosts by name, but you can reach them by pinging IP addresses, your system is having a problem with hostname resolution. Systems connected to the Internet do name-to-address resolution by communicating to a domain name system (DNS) server that can provide them with the IP addresses of the requested hosts.

The DNS server your system uses can be entered manually or picked up automatically from a DHCP server when you start your network interfaces. In either case, the names and IP addresses of one or more DNS servers end up in your `/etc/resolv.conf` file. Here is an example of that file:

```
search example.com
nameserver 192.168.0.254
nameserver 192.168.0.253
```

When you ask to connect to a hostname in Fedora or Red Hat Enterprise Linux, the `/etc/hosts` file is searched; then the name server entries in `resolv.conf` are queried in the order that they appear. Here are some ways of debugging name-to-address resolution:

Check if DNS server can be reached. Knowing the name server addresses, you can try to ping each name server's IP address to see if it is accessible. For example: `ping -c 2 192.168.0.254`. If the IP address can be reached, it could be that you were either assigned the wrong address for the DNS server or it is currently down.

Check if DNS server is working. You specifically try to use each DNS server with the `host` or `dig` command. For example, either of these two commands can be used to see if the DNS server at `192.168.0.254` can resolve the hostname `www.google`.

com into an IP address. Repeat this for each name server's IP address until you find which ones work:

```
# host www.google.com 192.168.0.254
Using domain server:
Name: 192.168.0.254
Address: 192.168.0.254#53
Aliases:
www.google.com has address 172.217.13.228
www.google.com has IPv6 address 2607:f8b0:4004:809::2004
# dig @192.168.0.254 www.google.com
...
;; QUESTION SECTION:
;www.google.com.                IN  A

;; ANSWER SECTION:
www.google.com.         67  IN  A   172.217.13.228
...
```

Correct your DNS servers. If you determine that you have the wrong IP addresses set for your DNS servers, changing them can be a bit tricky. Search /var/log/messages or the output of journalctl for your DNS servers' IP addresses. If NetworkManager is used to start your networking and connect to a DHCP server, you should see name server lines with the IP addresses being assigned. If the addresses are wrong, you can override them.

With NetworkManager enabled, you can't just add name server entries to the /etc/resolv.conf file because NetworkManager overwrites that file with its own name server entries. Instead, add a PEERDNS=no line to the ifcfg file for the network interface (for example, ifcfg-eth0 in the /etc/sysconfig/network-scripts directory). Then set DNS1=192.168.0.254 (or whatever is your DNS server's IP address). The new address is used the next time you restart your networking.

If you are using the network service, instead of NetworkManager, you can still use PEERDNS=no to prevent the DHCP server from overwriting your DNS addresses. However, in that case, you can edit the resolv.conf file directly to set your DNS server addresses.

The procedures just described for checking your outgoing network connectivity apply to any type of system, whether it is a laptop, desktop, or server. For the most part, incoming connections are not an issue with laptops or desktops because most requests are simply denied. However, for servers, the next section describes ways of making your server accessible if clients are having trouble reaching the services you provide from that server.

Troubleshooting incoming connections

If you are troubleshooting network interfaces on a server, there are different considerations than on a desktop system. Because most Linux systems are configured as servers, you should know how to troubleshoot problems encountered by those who are trying to reach your Linux servers.

I'll start with the idea of having an Apache web server (`httpd`) running on your Linux system, but no web clients can reach it. The following sections describe things that you can try to locate the problem.

Check if the client can reach your system at all

To be a public server, your system's hostname should be resolvable so that any client on the Internet can reach it. That means locking down your system to a particular, public IP address and registering that address with a public DNS server. You can use a domain registrar (such as `http://www.networksolutions.com`) to do that.

When clients cannot reach your website by name from their web browsers, if the client is a Linux system, you can go through `ping`, `host`, `traceroute`, and other commands described in the previous section to track down the connectivity problem. Windows systems have their own version of `ping` that you can use from those systems.

If the name-to-address resolution is working to reach your system and you can `ping` your server from the outside, the next thing to try is the availability of the service.

Check if the service is available to the client

From a Linux client, you can check if the service you are looking for (in this case `httpd`) is available from the server. One way to do that is using the `nmap` command.

The `nmap` command is a favorite tool for system administrators checking for various kinds of information on networks. However, it is a favorite cracker tool as well because it can scan servers, looking for potential vulnerabilities. So, it is fine to use `nmap` to scan your own systems to check for problems, but know that using `nmap` on another system is like checking the doors and windows on someone's house to see if you can get in. You look like an intruder.

Checking your own system to see what ports to your server are open to the outside world (essentially, checking what services are running) is perfectly legitimate and easy to do. After `nmap` is installed (`yum install nmap`), use your system hostname or IP address to use `nmap` to scan your system to see what is running on common ports:

```
# nmap 192.168.0.119
Starting Nmap 6.40 ( http://nmap.org ) at 2019-12-08 13:28 EST
Nmap scan report for spike (192.168.0.119)
Host is up (0.0037s latency).
Not shown: 995 filtered ports
PORT    STATE  SERVICE
21/tcp  open   ftp
22/tcp  open   ssh
80/tcp  open   http
443/tcp open   https
631/tcp open   ipp
MAC Address: 00:1B:21:0A:E8:5E (Intel Corporate)
Nmap done: 1 IP address (1 host up) scanned in 4.77 seconds
```

The preceding output shows that TCP ports are open to the regular (http) and secure (https) web services. When you see that the state is open, it indicates that a service is listening on the port as well. If you get to this point, it means that your network connection is fine, and you should direct your troubleshooting efforts to how the service itself is configured (for example, you might look in /etc/httpd/conf/httpd.conf to see if specific hosts are allowed or denied access).

If TCP ports 80 and/or 443 are not shown, it means that they are being filtered. You need to check whether your firewall is blocking (not accepting packets to) those ports. If the port is not filtered but the state is closed, it means that the httpd service either isn't running or isn't listening on those ports. The next step is to log in to the server and check those issues.

Check the firewall on the server

From your server, you can use the iptables command to list the filter table rules that are in place. Here is an example:

```
# iptables -vnL
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source    destination
...
   0     0 ACCEPT tcp  --  *  *   0.0.0.0/0 0.0.0.0/0   state NEW tcp
dpt:80
   0     0 ACCEPT tcp  --  *  *   0.0.0.0/0 0.0.0.0/0   state NEW tcp
dpt:443
...
```

For the RHEL 8 and Fedora 30 systems where the firewalld service is enabled, you can use the Firewall configuration window to open the ports needed. With the public Zone and Services tab selected, click the check boxes for http and https to open those ports immediately for all incoming traffic. If your system is using the basic iptables service, there should be firewall rules like the two shown in the preceding code among your other rules. If there aren't any, add those rules to the /etc/sysconfig/iptables file. Here are examples of what those rules might look like:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
```

With the rules added to the file, clear out all of your firewall rules (systemctl stop iptables.service or service iptables stop) and then start them again (systemctl start iptables.service or service iptables start).

If the firewall is still blocking client access to the web server ports, here are a few things to check in your firewall:

Check rules order. Look at rules in /etc/sysconfig/iptables and see if a DROP or REJECT rule comes before the rules opening ports 80 and/or 443. Moving the rules to open those ports before any final DROP or REJECT lines can solve the problem.

Look for denied hosts. Check whether any rules drop or reject packets from particular hosts or networks. Look for rules that include `-s` or `--source` followed by an IP address or address range and then a `-j DROP` or `ACCEPT`. Modify the rule or add a rule prior to your rules to make an exception for the host you want to allow to access your service.

If the port is now open but the service itself is closed, check that the service itself is running and listening on the appropriate interfaces.

Check the service on the server

If there seems to be nothing blocking client access to your server through the actual ports providing the service that you want to share, it is time to check the service itself. Assuming that the service is running (depending on your system, type **`service httpd status`** or **`systemctl status httpd.service`** to check), the next thing to check is that it is listening on the proper ports and network interfaces.

The `netstat` command is a great general-purpose tool for checking network services. The following command lists the names and process IDs (`p`) for all processes that are listening (`l`) for TCP (`t`) and UDP (`u`) services, along with the port number (`n`) on which they are listening. The command line filters out all lines except those associated with the `httpd` process:

```
# netstat -tupln | grep httpd
tcp    0  0 :::80          :::*          LISTEN      2567/httpd
tcp    0  0 :::443         :::*          LISTEN      2567/httpd
```

The previous example shows that the `httpd` process is listening on port 80 and 443 for all interfaces. It is possible that the `httpd` process might be listening on selected interfaces. For example, if the `httpd` process were only listening on the local interface (127.0.0.1) for HTTP requests (port 80), the entry would appear as follows:

```
tcp    0  0 127.0.0.1:80 :::*          LISTEN      2567/httpd
```

For `httpd`, as well as for other network services that listen for requests on network interfaces, you can edit the service's main configuration file (in this case, `/etc/httpd/conf/httpd.conf`) to tell it to listen on port 80 for all addresses (`Listen 80`) or a specific address (`Listen 192.168.0.100:80`).

Troubleshooting Memory

Troubleshooting performance problems on your computer is one of the most important, although often elusive, tasks you need to complete. Maybe you have a system that was working fine, but it begins to slow down to a point where it is practically unusable. Maybe applications just begin to crash for no apparent reason. Finding and fixing the problem may take some detective work.

Linux comes with many tools for watching activities on your system and figuring out what is happening. Using a variety of Linux utilities, you can do things such as finding out which processes are consuming large amounts of memory or placing high demands on your processors, disks, or network bandwidth. Solutions can include the following:

Adding capacity Your computer may be trying to do what you ask of it, but failures might occur because you don't have enough memory, processing power, disk space, or network capacity to get reasonable performance. Even nearing the boundaries of resource exhaustion can cause performance problems. Improving your computer hardware capacity is often the easiest way of solving performance problems.

Tuning the system Linux comes with default settings that define how it internally saves data, moves data around, and protects data. System tunable parameters can be changed if the default settings don't work well for the types of applications you have on your system.

Uncovering problem applications or users Sometimes, a system performs poorly because a user or an application is doing something wrong. Misconfigured or broken applications can hang or gobble up all of the resources they can find. An inexperienced user might mistakenly start multiple instances of a program that drain system resources. As a system administrator, you want to know how to find and fix these problems.

To troubleshoot performance problems in Linux, you use some of the basic tools for watching and manipulating processes running on your system. Refer to Chapter 6, "Managing Running Processes," if you need details on commands such as `ps`, `top`, `kill`, and `killall`. In the following sections, I add commands such as `memstat` to dig a little deeper into what processes are doing and where things are going wrong.

The most complex area of troubleshooting in Linux relates to managing virtual memory. The next sections describe how to view and manage virtual memory.

Uncovering memory issues

Computers have ways of storing data permanently (hard disks) and temporarily (*random access memory, or RAM,* and *swap space*). Think of yourself as a CPU, working at a desk trying to get your work finished. You would put data that you want to keep permanently in a filing cabinet across the room (that's like hard disk storage). You would put information that you are currently using on your desk (that's like RAM memory on a computer).

Swap space is a way of extending RAM. It is really just a place to put temporary data that doesn't fit in RAM but is expected to be needed by the CPU at some point. Although swap space is on the hard disk, it is not a regular Linux filesystem in which data is stored permanently.

Compared to disk storage, random access memory has the following attributes:

Nearer the processor Like the desk being near to you as you work, memory is physically near the CPU on the computer's motherboard. So, any data the CPU needs, it can just grab immediately if the data is in RAM.

Faster Its proximity to the CPU and the way that it is accessed (solid state versus mechanical hard disks) makes it much faster for the CPU to get information from RAM than it can from a hard disk. It's quicker to look at a piece of paper on your desk (a small, close space) than to walk to a row of file cabinets and to start searching for what you want.

Less capacity A new computer might have a 1TB or larger hard drive but 8GB or 16GB of RAM. Although it would make the computer run faster to put every file and every piece of data that the processor may need into RAM, in most cases there just wouldn't be enough room. Also, both the physical memory slots on the computer and the computer system itself (64-bit computers can address more RAM than 32-bit computers) can limit how much RAM a computer is capable of having.

More expensive Although RAM is tremendously more affordable than it was a decade or two ago, it is still much more expensive (per GB) than hard disks.

Temporary RAM holds data and metadata that the CPU is using now for the work it is doing (plus some content the Linux kernel is keeping around because it suspects a process will need it before long). When you turn off the computer, however, everything in RAM is lost. When the CPU is done with data, that data is discarded if it is no longer needed, left in RAM for possible later use, or marked to be written to disk for permanent storage if it needs to be saved.

It is important to understand the difference between temporary (RAM) and permanent (hard disk) storage, but that doesn't tell the whole story. If the demand for memory exceeds the supply of RAM, the kernel can temporarily move data out of RAM to an area called *swap space*.

If we revisit the desk analogy, this would be like saying, "There is no room left on my desk, yet I have to add more papers to it for the projects I'm currently working on. Instead of storing papers I'll need soon in a permanent file cabinet, I'll have one special file cabinet (like a desk drawer) to hold those papers that I'm still working with but that I'm not ready to store permanently or throw away."

Refer to Chapter 12, "Managing Disks and Filesystems," for more information on swap files and partitions and how to create them. For the moment, however, there are a few things that you should know about these kinds of swap areas and when they are used:

- When data is swapped from RAM to a swap area (swapped out), you get a performance hit. Remember, writing to disk is much slower than writing to RAM.
- When data is returned from swap to RAM because it is needed again (swapped in), you get another performance hit.

- When Linux runs out of space in RAM, swapping is like losing a high gear on a car. The car might have to run in a lower gear, but it would not stop altogether. In other words, all your processes stay active, and they don't lose any data or fail completely, but the system performance can significantly slow down.
- If both RAM and swap are full and no data can be discarded or written to disk, your system can reach an *out-of-memory* (*OOM*) condition. When that happens, the kernel OOM killer kicks in and begins killing off processes, one by one, to regain as much memory as the kernel needs to begin functioning properly again.

The general rule has always been that swapping is bad and should be avoided. However, some would argue that, in certain cases, more aggressive swapping can actually improve performance.

Think of the case where you open a document in a text editor and then minimize it on your desktop for several days as you work on different tasks. If data from that document were swapped out to disk, more RAM would be available for more active applications that could put that space to better use. The performance hit would come the next time you needed to access the data from the edited document and the data was swapped in from disk to RAM. The settings that relate to how aggressively a system swaps are referred to as *swappiness*.

As much as possible, Linux wants to make everything that an open application needs immediately available. So, using the desk analogy, if I am working on nine active projects and there is space on the desk to hold the information I need for all nine projects, why not leave them all within reach on the desk? Following that same way of thinking, the kernel sometimes keeps libraries and other content in RAM that it thinks you might eventually need—even if a process is not looking for it immediately.

The fact that the kernel is inclined to store information in RAM that it expects may be needed soon (even if it is not needed now) can cause an inexperienced system administrator to think that the system is almost out of RAM and that processes are about to start failing. That is why it is important to know the different kinds of information being held in memory—so that you can tell when real out-of-memory situations can occur. The problem is not just running out of RAM; it is running out of RAM when only non-swappable data is left.

Keep this general overview of *virtual memory* (RAM and swap) in mind, as the next section describes ways to go about troubleshooting issues related to virtual memory.

Checking for memory problems

Let's say that you are logged in to a Linux desktop, with lots of applications running, and everything begins to slow down. To find out if the performance problems have occurred because you have run out of memory, you can try commands such as `top` and `ps` to begin looking for memory consumption on your system.

To run the top command to watch for memory consumption, type **top** and then type a capital **M**. Here is an example:

```
# top
top - 22:48:24 up  3:59,  2 users,  load average: 1.51, 1.37, 1.15
Tasks: 281 total,   2 running, 279 sleeping,   0 stopped,   0 zombie
Cpu(s): 16.6%us,  3.0%sy,  0.0%ni, 80.3%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   3716196k total,  2684924k used,  1031272k free,   146172k buffers
Swap:  4194296k total,        0k used,  4194296k free,   784176k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6679 cnegus    20   0 1665m 937m  32m S  7.0 25.8  1:07.95  firefox
 6794 cnegus    20   0  743m 181m  30m R 64.8  5.0  1:22.82  npviewer.bin
 3327 cnegus    20   0 1145m 116m  66m S  0.0  3.2  0:39.25  soffice.bin
 6939 cnegus    20   0  145m  71m  23m S  0.0  2.0  0:00.97  acroread
 2440 root      20   0  183m  37m  26m S  1.3  1.0  1:04.81  Xorg
 2795 cnegus    20   0 1056m  22m  14m S  0.0  0.6  0:01.55  nautilus
```

There are two lines (Mem and Swap) and four columns of information (VIRT, RES, SHR, and %MEM) relating to memory in the top output. In this example, you can see that RAM is not exhausted from the Mem line (only 2684924k of 3716196k is used) and that nothing is being swapped to disk from the Swap line (0k used).

However, adding up just these first six lines of output in the VIRT column, you would see that 4937MB of memory has been allocated for those applications, which exceeds the 3629MB of total RAM (3716196k) that is available. That's because the VIRT column shows only the amount of memory that has been promised to the application. The RES line shows the amount of non-swappable memory that is actually being used, which totals only 1364MB.

Notice that, when you ask to sort by memory usage by typing a capital **M**, top knows to sort on that RES column. The SHR column shows memory that could potentially be shared by other applications (such as libraries), and %MEM shows the percentage of total memory consumed by each application.

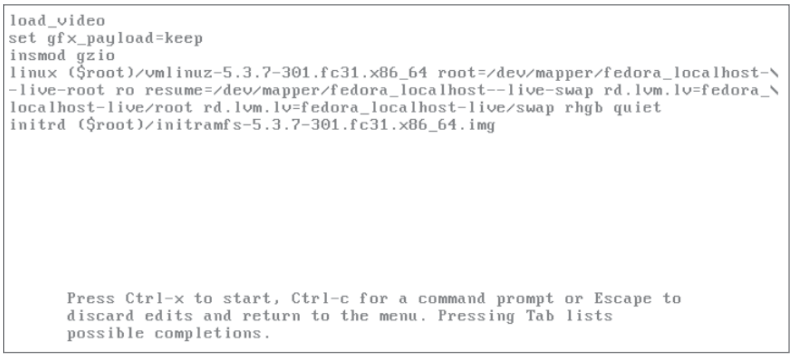
If you think that the system is reaching an out-of-memory state, here are a few things to look for:

- The free space shown on the Mem line would be at or near zero.
- The used space shown on the Swap line would be non-zero and would continue to grow. That should be accompanied with a slowdown of system performance.
- As the top screen redraws every few seconds, if there is a process with a memory leak (continuously asking for and using more memory, but not giving any memory back), the amount of VIRT memory grows, but more important, the RES memory continues to grow for that process.
- If the Swap space actually runs out, the kernel starts to kill off processes to deal with this out-of-memory condition.

If you have Cockpit installed and enabled, you can watch memory usage live from your Web browser. Open Cockpit and then select System ➪ Memory & Swap. Figure 21.4 shows a system where the memory is all being consumed by multiple video streams and has begun swapping.

FIGURE 21.4

Monitor RAM and Swap usage in real time with Cockpit.



Dealing with memory problems

In the short term, you can do several things to deal with this out-of-memory condition:

Kill a process If the memory problem is due to one errant process, you can simply kill that process. Assuming that you are logged in as root or as the user who owns the runaway process, type **k** from the top window, then enter the PID of the process that you want to kill, and choose 15 or 9 as the signal to send.

Drop page caches If you just want to clear up some memory right now as you otherwise deal with the problem, you can tell the system to drop inactive page caches. When you do this, some memory pages are written to disk; others are just discarded (because they are stored permanently and can be gotten again from disk when they are needed).

This action is the equivalent of cleaning your desk and putting all but the most critical information into the trash or into a file cabinet. You may need to retrieve information again shortly from a file cabinet, but you almost surely don't need it all immediately. Keep `top` running in one Terminal window to see the `Mem` line change as you type the following (as root) into another Terminal window:

```
# echo 3 > /proc/sys/vm/drop_caches
```

Kill an out-of-memory process Sometimes, memory exhaustion has made the system so unusable that you may not be able to get a response from a shell or GUI. In those cases, you might be able to use Alt+SysRq keystrokes to kill an out-of-memory process. The reason you can use Alt+SysRq keystrokes on an otherwise unresponsive system is that the kernel processes Alt+SysRq requests ahead of other requests.

To enable Alt+SysRq keystrokes, the system must have already set `/proc/sys/kernel/sysrq` to `1`. An easy way to do this is to add `kernel.sysrq = 1` to the `/etc/sysctl.conf` file. Also, you must run the Alt+SysRq keystrokes from a text-based interface (such as the virtual console you see when you press Ctrl+Alt+F2). With `kernel.sysrq` set to `1`, you can kill the process on your system with the highest OOM score by pressing Alt+SysRq+f from a text-based interface. A listing of all processes running on your system appears on the screen with the name of the process that was killed listed at the end. You can repeat those keystrokes until you have killed enough processes to be able to access the system normally from the shell again.

Note

There are many other Alt+SysRq keystrokes that you can use to deal with an unresponsive system. For example, Alt+SysRq+e terminates all processes except for the `init` process. Alt+SysRq+t dumps a list of all current tasks and information about those tasks to the console. To reboot the system, press Alt+SysRq+b. See the `sysrq.txt` file in the `/usr/share/doc/kernel-doc*/Documentation` directory for more information about Alt+SysRq keystrokes.

Troubleshooting in Rescue Mode

If your Linux system becomes unbootable, your best option for fixing it is probably to go into *rescue mode*. To go into rescue mode, you bypass the Linux system installed on your hard disk and boot some rescue medium (such as a bootable USB key or boot CD). After the rescue medium boots, it tries to mount any filesystems that it can find from your Linux system so that you can repair any problems.

For many Linux distributions, the installation CD or DVD can serve as boot media for going into rescue mode. Here's an example of how to use a RHEL installation DVD to go into rescue mode to fix a broken Linux system (burn the image to a USB drive if your computer doesn't have a DVD drive):

1. Get the installation CD or DVD image that you want to use and burn it to the appropriate medium (CD or DVD). See Appendix A, "Media," for information on burning CDs and DVDs. (For my example, I used a Red Hat Enterprise Linux 8 installation DVD.)
2. Insert the CD or DVD into the drive on the computer that has the broken Linux system installed and reboot.
3. The moment you see the BIOS screen, press the function key noted on that screen for selecting the boot device (possibly the F12 or F2 function key).
4. Choose the drive (CD or DVD) from the list of bootable devices, and press Enter.
5. When the RHEL 8 boot menu appears, use the arrow keys to highlight the word *Troubleshooting* and press Enter. In other Linux boot media, the selection could say

Rescue Mode or something similar. On the next screen that appears, select Rescue a Red Hat Enterprise Linux system and press Enter.

6. After a few moments, the Linux system on the rescue medium boots up. When prompted, select your language and keyboard. You are asked if you want to start network interfaces on the system.
7. If you think that you might need to get something from another system on your network (such as RPM packages or debugging tools), select Yes and try to configure your network interfaces. You are then asked if you want to try to mount filesystems from your installed Linux system under `/mnt/sysimage`.
8. Select Continue to have your filesystems mounted (if possible) under the `/mnt/sysimage` directory. If this is successful, a Rescue message appears, telling you that your filesystems have been mounted under `/mnt/sysimage`.
9. Select OK to continue. You should see a shell prompt for the root user (#). You are ready to begin troubleshooting from rescue mode. After you are in rescue mode, the portion of your filesystem that is not damaged is mounted under the `/mnt/sysimage` directory. Type **`ls /mnt/sysimage`** to check that the files and directories from the hard disk are there.

Right now, the root of the filesystem (`/`) is from the filesystem that comes on the rescue medium. To troubleshoot your installed Linux system, however, you can type the following command:

```
# chroot /mnt/sysimage
```

Now the `/mnt/sysimage` directory becomes the root of your filesystem (`/`) so that it looks like the filesystem installed on your hard disk. Here are some things that you can do to repair your system while you are in rescue mode:

Fix /etc/fstab. If your filesystems couldn't mount because of an error in your `/etc/fstab` file, you could try to correct any entries that might have problems (such as wrong device names or a mount point directory that doesn't exist). Type **`mount -a`** to make sure that all of the filesystems mount.

Reinstall missing components. It might be that the filesystems are fine, but the system failed to boot because some critical command or configuration file is missing. You might be able to fix the problem by reinstalling the package with the missing components. For example, if someone had deleted `/bin/mount` by mistake, the system would have no command to mount filesystems. Reinstalling the `util-linux` package would replace the missing `mount` command.

Check the filesystems. If your booting problems stem from corrupt filesystems, you can try running the `fsck` command (filesystem check) to see if there is any corruption on the disk partition. If there is, `fsck` attempts to correct problems it encounters.

When you are finished fixing your system, type **exit** to exit the `chroot` environment, and return to the filesystem layout that the live medium sees. If you are completely finished, type **reboot** to restart your system. Be sure to pop out the medium before the system restarts.

Summary

Troubleshooting problems in Linux can start from the moment you turn on your computer. Problems can occur with your computer BIOS, boot loader, or other parts of the boot process that you can correct by intercepting them at different stages of the boot process.

After the system has started, you can troubleshoot problems with software packages, network interfaces, or memory exhaustion. Linux comes with many tools for finding and correcting any part of the Linux system that might break down and need fixing.

The next chapter covers the topic of Linux security. Using the tools described in that chapter, you can provide access to those services that you and your users need while blocking access to system resources that you want to protect from harm.

Exercises

The exercises in this section enable you to try out useful troubleshooting techniques in Linux. Because some of the techniques described here can potentially damage your system, I recommend that you do not use a production system that you cannot risk damaging. See Appendix B for suggested solutions.

These exercises relate to troubleshooting topics in Linux. They assume that you are booting a PC with standard BIOS. To do these exercises, you need to be able to reboot your computer and interrupt any work it may be doing.

1. Boot your computer, and as soon as you see the BIOS screen, go into Setup mode as instructed on the BIOS screen.
2. From the BIOS Setup screen, determine if your computer is 32-bit or 64-bit, if it includes virtualization support, and if your network interface card is capable of PXE booting.
3. Reboot, and just after the BIOS screen disappears, when you see the countdown to booting the Linux system, press any key to get to the GRUB boot loader.
4. From the GRUB boot loader, add an option to boot up to runlevel 1 so that you can do some system maintenance.
5. After the system boots up, look at the messages that were produced in the kernel ring buffer that show the activity of the kernel as it booted up.
6. In Fedora or RHEL, run a trial `yum update` and exclude any kernel package that is available.

7. Check to see what processes are listening for incoming connections on your system.
8. Check to see what ports are open on your external network interface.
9. Run the `top` command in a Terminal window. Open a second Terminal window, clear your page cache, and note on the `top` screen if more RES memory is now available.
10. With Cockpit enabled on your system, access Cockpit to view details on the system's ongoing memory and swap usage.

Part V

Learning Linux Security Techniques

IN THIS PART

Chapter 22
Understanding Basic Linux Security

Chapter 23
Understanding Advanced Linux Security

Chapter 24
Enhancing Linux Security with SELinux

Chapter 25
Securing Linux on a Network

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER 22

Understanding Basic Linux Security

IN THIS CHAPTER

Implementing basic security

Monitoring security

Auditing and reviewing security

At its most basic level, securing a Linux system starts with physical security, data security, user accounts protection, and software security. Over time, you need to monitor that system to make sure it remains safe.

Some of the questions that you need to ask yourself include the following:

- Who can get to the system physically?
- Are backup copies of data being made in case of disaster?
- How well are user accounts secured?
- Does the software come from a secure Linux distribution, and are security patches up to date?
- Have you been monitoring the system to make sure that it has not been cracked or corrupted?

This chapter starts by covering basic Linux security topics. Subsequent chapters go deeper into advanced security mechanisms.

Implementing Physical Security

A lock on the computer server room door is the first line of defense. Although a very simple concept, it is often ignored. Access to the physical server means access to all of the data that it contains. No security software can fully protect your systems if someone with malicious intent has physical access to the Linux server.

Basic server room physical security includes items such as these:

- A lock or security alarm on the server room door
- Access controls that allow only authorized access and that identify who accessed the room and when the access occurred, such as a card key entry system

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

- A sign stating "no unauthorized access allowed" on the door
- Policies on who can access the room and when that access may occur for groups such as the cleaning crew, server administrators, and others

Physical security includes environmental controls. Appropriate fire suppression systems and proper ventilation for your server room must be implemented.

Implementing disaster recovery

Disaster recovery plans should include these things:

- What data is to be included in backups
- Where backups are to be stored
- How long backups are maintained
- How backup media is rotated through storage

Backup data, media, and software should be included in your Access Control Matrix checklist.

CAUTION

It is important to determine how many backup copies of each object should be maintained. While you may need only three backup copies of one particular object, another object may be important enough to require maintaining more copies.

Backup utilities on a Linux system include the following:

- `amanda` (Advanced Maryland Automatic Network Disk Archiver)
- `cpio`
- `dump/restore`
- `tar`
- `rsync`

The `cpio`, `dump/restore`, and `tar` utilities are typically pre-installed on a Linux distribution. A simple, yet effective tool for backing up data over networks is the `rsync` utility. With `rsync`, you can set up a `cron` job to keep copies of all data in selected directories or mirror exact copies of directories on remote machines.

Of the tools just mentioned, only `amanda` is not typically installed by default. However, `amanda` is extremely popular because it comes with a great deal of flexibility and can even back up a Windows system. If you need more information on the `amanda` backup utility, see `amanda.org`. Ultimately, the utility you select must meet your organization's particular security needs for backup.

Securing user accounts

User accounts are part of the authentication process allowing users into the Linux system. Proper user account management enhances a system's security. Setting up user accounts

was covered in Chapter 11, "Managing User Accounts." However, a few additional rules are necessary to increase security through user account management:

- One user per user account.
- Limit access to the root user account.
- Set expiration dates on temporary accounts.
- Remove unused user accounts.

One user per user account

Accounts should enforce accountability. Thus, multiple people should not be logging in to one account. When multiple people share an account, there is no way to prove a particular individual completed a particular action.

Limiting access to the root user account

If multiple people can log in to the root account, you have another repudiation situation. You cannot track individual use of the root account. To allow tracking of the root account use by individuals, a policy for using `sudo` (see Chapter 8, "Learning System Administration") instead of logging into root should be instituted.

Instead of giving multiple people root permission on a Linux system, you can grant root access on a per-command basis with the `sudo` command. Using `sudo` provides the following security benefits:

- The root password does not have to be given out.
- You can fine-tune command access.
- All `sudo` use (who, what, when) is recorded in `/var/log/secure`, including any failed `sudo` access attempts. Recent Linux systems store all `sudo` access in the systemd journal (type **`journalctl -f`** to watch live `sudo` access attempts, along with other system messages).
- After you grant someone `sudo` permission, you can try to restrict root access to certain commands in the `/etc/sudoers` file (with the `visudo` command). However, after you grant root permission to a user, even in a limited way, it is difficult to be sure that a determined user can't find ways to gain full root access to your system and do what they want to it.

One way to keep a misbehaving administrator in check is to have security messages intended for the `/var/log/secure` file sent to a remote log server to which none of the local administrators have access. In that way, any misuse of root privilege is attached to a particular user and is logged in a way that the user can't cover their tracks.

Setting expiration dates on temporary accounts

If you have consultants, interns, or temporary employees who need access to your Linux systems, it is important to set up their user accounts with expiration dates. The expiration date is a safeguard, in case you forget to remove their accounts when they no longer need access to your organization's systems.

To set a user account with an expiration date, use the `usermod` command. The format is `usermod -e yyyy-mm-dd user_name`. In the following code, the account `tim` has been set to expire on January 1, 2021.

```
# usermod -e 2021-01-01 tim
```

To verify that the account has been properly set to expire, double-check yourself by using the `chage` command. The `chage` command is primarily used to view and change a user account's password aging information. However, it also can access account expiration information. The `-l` option allows you to list various information to which `chage` has access. To keep it simple, pipe the output from the `chage` command into `grep` and search for the word *Account*. This produces only the user account's expiration date.

```
# chage -l tim | grep Account
Account expires                                         :  Jan  01,  2021
```

As you can see, the account expiration date was successfully changed for `tim` to January 1, 2021.

TIP

If you do not use the `/etc/shadow` file for storing your account passwords, the `chage` utility doesn't work. In most cases, this is not a problem because the `/etc/shadow` file is configured to store password information by default on most Linux systems.

Set account expiration dates for all transitory employees. In addition, consider reviewing all user account expiration dates as part of your security monitoring activities. These activities help to eliminate any potential backdoors to your Linux system.

Removing unused user accounts

Keeping old expired accounts around is asking for trouble. After a user has left an organization, it is best to perform a series of steps to remove their account along with data:

1. Find files on the system owned by the account, using the *`find / -user username`* command.
2. Expire or disable the account.
3. Back up the files.
4. Remove the files or reassign them to a new owner.
5. Delete the account from the system.

Problems occur when step 5 is forgotten and expired or disabled accounts are still on the system. A malicious user gaining access to your system could renew the account and then masquerade as a legitimate user.

To find these accounts, search through the `/etc/shadow` file. The account's expiration date is in the eighth field of each record. It would be convenient if a date format were used. Instead, this field shows the account's expiration date as the number of days since January 1, 1970.

You can use a two-step process to find expired accounts in the /etc/shadow file automatically. First, set up a shell variable (see Chapter 7, "Writing Simple Shell Scripts") with today's date in "days since January 1, 1970" format. Then, using the gawk command, you can obtain and format the information needed from the /etc/shadow file.

Setting up a shell variable with the current date converted to the number of days since January 1, 1970 is not particularly difficult. The date command can produce the number of seconds since January 1, 1970. To get what you need, divide the result from the date command by the number of seconds in a day: 86,400. The following demonstrates how to set up the shell variable TODAY.

```
# TODAY=$(echo $(($(date --utc --date "$1" +%s)/86400)))
# echo $TODAY
16373
```


Next, the accounts and their expiration dates are pulled from the /etc/shadow file using gawk. The gawk command is the GNU version of the awk program used in UNIX. The command's output is shown in the code that follows. As you would expect, many of the accounts do not have an expiration date. However, two accounts, Consultant and Intern, show an expiration date in the "days since January 1, 1970" format. Note that you can skip this step. It is just for demonstration purposes.

```
# gawk -F: '{print $1,$8}' /etc/shadow
...
chrony
tcpdump
johndoe
Consultant 13819
Intern 13911
```

The $1 and $8 in the gawk command represent the username and expiration date fields in the /etc/shadow file records. To check those accounts' expiration dates and see if they are expired, a more refined version of the gawk command is needed.

```
# gawk -F: '{if (($8 > 0) && ($TODAY > $8)) print $1}' /etc/shadow
Consultant
Intern
```

Only accounts with an expiration date are collected by the ($8 > 0) portion of the gawk command. To make sure that these expiration dates are past the current date, the TODAY variable is compared with the expiration date field, $8. If TODAY is greater than the account's expiration date, the account is listed. As you can see in the preceding example, two expired accounts still exist on the system and need to be removed.

That is all you need to do. Set up your TODAY variable and execute the gawk command. All of the expired accounts in the /etc/shadow file are listed for you. To remove these accounts, use the userdel command.

User accounts are only a portion of the authentication process allowing users into the Linux system. User account passwords also play an important role in the process.

Securing passwords

Passwords are the most basic security tool of any modern operating system and, consequently, the most commonly attacked security feature. It is natural for users to want to choose a password that is easy to remember, but often this means that they choose a password that is also easy to guess.

Brute force methods are commonly employed to gain access to a computer system. Trying the popular passwords often yields results. Some of the most common passwords are as follows:

- 123456
- Password
- princess
- rockyou
- abc123

Just use your favorite Internet search engine and look for "common passwords." If you can find these lists, then malicious attackers can too. Obviously, choosing good passwords is critical to having a secure system.

Choosing good passwords

In general, a password must not be easy to guess, be common or popular, or be linked to you in any way. Here are some rules to follow when choosing a password:

- Do not use any variation of your login name or your full name.
- Do not use a dictionary word.
- Do not use proper names of any kind.
- Do not use your phone number, address, family, or pet names.
- Do not use website names.
- Do not use any contiguous line of letters or numbers on the keyboard (such as "qwerty" or "asdfg").
- Do not use any of the above with added numbers or punctuation at the front or end or typed backward.

So now that you know what not to do, look at the two primary items that make a strong password:

1. A password should be at least 15 to 25 characters in length.
2. A password should contain all of the following:
   - Lowercase letters
   - Uppercase letters
   - Numbers
   - Special characters, such as : `! $ % * ( ) - + = , < > : : ″ ’`

Twenty-five characters is a long password. However, the longer the password, the more secure it is. What your organization chooses as the minimum password length depends on its security needs.

Tip

Gibson Research Center has some excellent material on strong passwords, including an article called "How big is your haystack. . .and how well hidden is your needle?" at `grc.com/haystack.htm`.

Choosing a good password can be difficult. It has to be hard enough not to be guessed and easy enough for you to remember. A good way to choose a strong password is to take the first letter from each word of an easily remembered sentence. Be sure to add numbers, special characters, and varied case. The sentence you choose should have meaning only to you and should not be publicly available. Table 22.1 lists examples of strong passwords and the tricks used to remember them.

TABLE 22.1 Ideas for Good Passwords

Password	How to Remember It
Mrci7yo!	My rusty car is 7 years old!
2emBp1ib	2 elephants make BAD pets, 1 is better
ItMc?Gib	Is that MY coat? Give it back

The passwords look like nonsense but are actually rather easy to remember. Of course, be sure not to use the passwords listed here. Now that they are public, they will be added to malicious attackers' dictionaries.

Setting and changing passwords

You set your own password using the `passwd` command. Type the `passwd` command and it allows you to change your password. First, it prompts you to enter your old password. To protect against someone shoulder surfing and learning your password, the password is not displayed as you type.

Assuming that you type your old password correctly, the `passwd` command prompts you for the new password. When you type your new password, it is checked using a utility called `cracklib` to determine whether it is a good or bad password. Non-root users are required to try a different password if the one they have chosen is not a good password.

The root user is the only user who is permitted to assign bad passwords. After the password has been accepted by `cracklib`, the `passwd` command asks you to enter the new password a second time to make sure that there are no typos (which are hard to detect when you can't see what you are typing).

When running as root, changing a user's password is possible by supplying that user's login name as a parameter of the `passwd` command, as in this example:

```
# passwd joe
Changing password for user joe.
New UNIX password: ********
Retype new UNIX password: ********
passwd: all authentication tokens updated successfully.
```

Here, the `passwd` command prompts you twice to enter a new password for `joe`. It does not prompt for his old password in this case.

Enforcing best password practices

Now you know what a good password looks like and how to change a password, but how do you enforce it on your Linux system? One place to start is with the PAM facility. With PAM, you can define exact requirements that passwords must meet. For example, to ensure that passwords must be 12 characters long, with at least 2 numbers, 3 uppercase letters, and 2 lowercase letters, and are different than the previous passwords, you can add the following line to either the `/etc/pam.d/common-password` or `/etc/pam.d/common-auth` file:

```
password requisite pam_cracklib.so minlen=12, dcredit=2, ucredit=3,
lcredit=2, difok=4
```

The next question is, How can you make people change passwords? It can become tiresome to come up with new, strong passwords every 30 days! That is why some enforcing techniques are often necessary.

Tip

If users are having a difficult time creating secure and unique passwords, consider installing the `pwgen` utility on your Linux system. This open source password generating utility creates passwords that are made to be pronounceable and memorable. You can use these generated words as a starting point for creating account passwords.

Default values in the `/etc/login.defs` file for new accounts were covered in Chapter 11. Within the `login.defs` file are some settings affecting password aging and length:

```
PASS_MAX_DAYS     30
PASS_MIN_DAYS     5PASS_MIN_LEN      16PASS_WARN_AGE     7
```

In this example, the maximum number of days, `PASS_MAX_DAYS`, until the password must be changed is 30. The number that you set here is dependent upon your particular account setup. For organizations that practice one person to one account, this number can be much larger than 30. If you do have shared accounts or multiple people know the root password, it is imperative that you change the password often. This practice effectively refreshes the list of those who know the password.

To keep users from changing their password to a new password and then immediately changing it right back, you need to set the `PASS_MIN_DAYS` to a number larger than 0. In the preceding example, the soonest a user could change their password again is 5 days.

The PASS_WARN_AGE setting is the number of days a user is warned before being forced to change their password. People tend to need lots of warnings and prodding, so the preceding example sets the warning time to 7 days.

Earlier in the chapter, I mentioned that a strong password is between 15 and 25 characters long. With the PASS_MIN_LEN setting, you can force users to use a certain minimum number of characters in their passwords. The setting you choose should be based upon your organization's security life cycle plans.

NOTE
Ubuntu does not have the PASS_MIN_LEN setting in its login.defs file. Instead, this setting is handled by the PAM utility. PAM is covered in Chapter 23, "Understanding Advanced Linux Security."

For accounts that have already been created, you need to control password aging via the chage command. The options needed to control password aging with chage are listed in Table 22.2. Notice that there is not a password length setting in the chage utility.

TABLE 22.2 chage **Options**

Option	Description
-M	Sets the maximum number of days before a password needs to be changed. Equivalent to PASS_MAX_DAYS in /etc/login.defs.
-m	Sets the minimum number of days before a password can be changed again. Equivalent to PASS_MIN_DAYS in /etc/login.defs.
-W	Sets the number of days a user is warned before being forced to change the account password. Equivalent to PASS_WARN_AGE in /etc/login.defs.

The example that follows uses the chage command to set password aging parameters for the tim account. All three options are used at once.

```
# chage -l tim | grep days
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
# chage -M 30 -m 5 -W 7 tim
# chage -l tim | grep days
Minimum number of days between password change          : 5
Maximum number of days between password change          : 30
Number of days of warning before password expires       : 7
```

You can also use the chage command as another method of account expiration, which is based upon the account's password expiring. Earlier, the usermod utility was used for account expiration. Use the chage command with the -M and the -I options to lock the

account. In the code that follows, the `tim` account is viewed using `chage -l`. Only the information for `tim`'s password settings are extracted.

```
# chage -l tim | grep Password
Password expires                    : never
Password inactive                   : never
```

You can see that there are no settings for password expiration (`Password expires`) or password inactivity (`Password inactive`). In the following code, the account is set to be locked 5 days after `tim`'s password expires by using only the `-I` option.

```
# chage -I 5 tim
# chage -l tim | grep Password
Password expires                    : never
Password inactive                   : never
```

Notice that no settings changed! Without a password expiration set, the `-I` option has no effect. Thus, using the `-M` option, the maximum number of days is set before the password expires and the setting for the password inactivity time should take hold.

```
# chage -M 30 -I 5 tim
# chage -l tim | grep Password
Password expires                    : Mar 03, 2017
Password inactive                   : Mar 08, 2017
```

Now, `tim`'s account will be locked 5 days after his password expires. This is helpful in situations where an employee has left the company but their user account has not yet been removed. Depending upon your organization's security needs, consider setting all accounts to lock a certain number of days after passwords have expired.

Understanding the password files and password hashes

Early Linux systems stored their passwords in the `/etc/passwd` file. The passwords were hashed. A *hashed password* is created using a one-way mathematical process. After you create the hash, you cannot re-create the original characters from the hash. Here's how it works.

When a user enters the account password, the Linux system rehashes the password and then compares the hash result to the original hash in `/etc/passwd`. If they match, the user is authenticated and allowed into the system.

The problem with storing these password hashes in the `/etc/passwd` file has to do with the filesystem security settings (see Chapter 4, "Moving Around the Filesystem"). The filesystem security settings for the `/etc/passwd` file are listed here:

```
# ls -l /etc/passwd
-rw-r--r--. 1 root root 1644 Feb  2 02:30 /etc/passwd
```

As you can see, everyone can read the password file. You might think that this is not a problem because the passwords are all hashed. However, individuals with malicious intent have created files called *rainbow tables*. A rainbow table is simply a dictionary of potential

passwords that have been hashed. For instance, the rainbow table would contain the hash for the popular password "Password," which is as follows:

```
$6$dhN5ZMUj$CNghjYIteau5xl8yX.f6PTOpendJwTOcXjlTDQUQZhhy
V8hKzQ6Hxx6Egj8P3VsHJ8Qrkv.VSR5dxcK3QhyMc.
```

Because of the ease of access to the password hashes in the `/etc/passwd` file, it is only a matter of time before a hashed password is matched in a rainbow table and the plain-text password is uncovered.

NOTE

Security experts will tell you that the passwords are not just hashed but also salted. *Salting a hash* means that a randomly generated value is added to the original password before it is hashed. This makes it even more difficult for the hashed password to be matched to its original password. However, in Linux, the hash salt is also stored with the hashed passwords. Thus, read access to the `/etc/passwd` file means that you have the hash value and its salt.

Thus, the hashed passwords were moved to a new configuration file, `/etc/shadow`, many years ago. This file has the following security settings:

```
# ls -l /etc/shadow
----------. 1 root root 1049 Feb  2 09:45 /etc/shadow
```

Despite having no permissions open, root, but no other user, can view this file. Thus, the hashed passwords are protected. Here is the tail end of a `/etc/shadow` file. You can see that there are long, nonsensical character strings in each user's record. Those are the hashed passwords.

```
# tail -2 /etc/shadow
johndoe:$6$jJjdRN9/qELmb8xWM1LgOYGhEIxc/:15364:0:99999:7:::
Tim:$6$z760AJ42$QXdhFyndpbVPVM5oVtNHs4B/:15372:5:30:7:16436::
```

CAUTION

You may inherit a Linux system that still uses the old method of keeping the hashed passwords in the `/etc/passwd` file. It is easy to fix. Just use the `pwconv` command, and the `/etc/shadow` file is created and hashed passwords moved to it.

The following are also stored in the `/etc/shadow` file, in addition to the account name and hashed password:

- Number of days (since January 1, 1970) since the password was changed
- Number of days before the password can be changed
- Number of days before a password must be changed
- Number of days to warn a user before a password must be changed
- Number of days after a password expires that an account is disabled
- Number of days (since January 1, 1970) that an account has been disabled

This should sound familiar, as they are the settings for password aging covered earlier in the chapter. Remember that the `chage` command does not work if you do not have an `/etc/shadow` file set up or if the `/etc/login.defs` file is not available.

Obviously, filesystem security settings are very important for keeping your Linux system secure. This is especially true with all Linux systems' configuration files and others.

Securing the filesystem

Another important part of securing your Linux system is setting proper filesystem security. The basics for security settings were covered in Chapter 4 and Access Control Lists (ACLs) in Chapter 11. However, there are a few additional points that need to be added to your knowledge base.

Managing dangerous filesystem permissions

If you gave full `rwxrwxrwx` (777) access to every file on the Linux system, you can imagine the chaos that would follow. In many ways, similar chaos can occur by not closely managing the set UID (`SUID`) and the set GID (`SGID`) permissions (see Chapter 4 and Chapter 11).

Files with the `SUID` permission in the `Owner` category and execute permission in the `Other` category allow anyone to become the file's owner temporarily while the file is being executed in memory. The riskiest case is if the file's owner is root.

Similarly, files with the `SGID` permission in the `Group` category and execute permission in the `Other` category allow anyone temporarily to become a group member of the file's group while the file is being executed in memory. `SGID` can also be set on directories. This sets the group ID of any files created in the directory to the group ID of the directory.

Executable files with `SUID` or `SGID` are favorites of malicious users. Thus, it is best to use them sparingly. However, some files do need to keep these settings. Two examples are the `passwd` and the `sudo` commands that follow. Each of these files should maintain their `SUID` permissions.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 28804 Aug 17 20:50 /usr/bin/passwd
$ ls -l /usr/bin/sudo
---s--x--x. 2 root root 77364 Nov 3 08:10 /usr/bin/sudo
```

Commands such as `passwd` and `sudo` are designed to be used as `SUID` programs. Even though those commands run as root user, as a regular user you can only change your own password with `passwd` and can only escalate to root permission with `sudo` if you were given permission in the `/etc/sudoers` file. A more dangerous situation would be if a hacker created a `SUID bash` command; anyone running that command could effectively change everything on the system that had root access.

Using the `find` command, you can search your system to see if there are any hidden or otherwise inappropriate `SUID` and `SGID` commands on your system. Here is an example:

```
# find / -perm /6000 -ls
4597316 52 -rwxr-sr-x 1 root games 51952 Dec 21 2013 /usr/bin/atc
4589119 20 -rwxr-sr-x 1 root tty   19552 Nov 18 2013 /usr/bin/write
4587931 60 -rwsr-xr-x 1 root root  57888 Aug  2 2013 /usr/bin/at
4588045 60 -rwsr-xr-x 1 root root  57536 Sep 25 2013 /usr/bin/crontab
4588961 32 -rwsr-xr-x 1 root root  32024 Nov 18 2013 /usr/bin/su
...
5767487 85 -rwsrwsr-x 1 root  root 68928 Sep 13 11:52 /var/.bin/myvi
...
```

Notice that `find` uncovers `SUID` and `SGID` commands that regular users can run to escalate their permission for particular reasons. In this example, there is also a file that a user tried to hide (`myvi`). This is a copy of the `vi` command that, because of permission and ownership, can change files owned by root. This is obviously a user doing something that they should not be doing.

Securing the password files

The `/etc/passwd` file is the file the Linux system uses to check user account information and was covered earlier in the chapter. The `/etc/passwd` file should have the following permission settings:

- Owner: root
- Group: root
- Permissions: (644) Owner: `rw-` Group: `r--` Other: `r--`

The example that follows shows that the `/etc/passwd` file has the appropriate settings:

```
# ls -l /etc/passwd
-rw-r--r--. 1 root root 1644 Feb  2 02:30 /etc/passwd
```

These settings are needed so that users can log in to the system and see usernames associated with user ID and group ID numbers. However, users should not be able to modify the `/etc/passwd` directly. For example, a malicious user could add a new account to the file if write access were granted to `Other`.

The next file is the `/etc/shadow` file. Of course, it is closely related to the `/etc/passwd` file because it is also used during the login authentication process. This `/etc/shadow` file should have the following permissions settings:

- Owner: root
- Group: root
- Permissions: (000) Owner: `---` Group: `---` Other: `---`

The code that follows shows that the `/etc/shadow` file has the appropriate settings.

```
# ls -l /etc/shadow
----------. 1 root root 1049 Feb  2 09:45 /etc/shadow
```

The `/etc/passwd` file has read access for the owner, group, and other. Notice how much more the `/etc/shadow` file is restricted than the `/etc/passwd` file. For the

/etc/shadow file, there is no access permission on, although the root user can still access the file. So, if only root can view this file, how can users change their passwords, which are stored in /etc/shadow? The passwd utility, /usr/bin/passwd, uses the special permission SUID. This permission setting is shown here:

```
# ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 28804 Aug 17 20:50 /usr/bin/passwd
```

Thus, the user running the passwd command temporarily becomes root while the command is executing in memory and can then write to the /etc/shadow file, but only to change the user's own password-related information.

NOTE

The root user does not have write access to the /etc/shadow permissions, so how does root write to the /etc/shadow file? The root user is all-powerful and has complete access to all files, whether the permissions are listed or not.

The /etc/group file (see Chapter 11) contains all of the groups on the Linux system. Its file permissions should be set exactly as the /etc/passwd file:

- Owner: root
- Group: root
- Permissions: (644) Owner: rw- Group: r-- Other: r--

Also, the group password file, /etc/gshadow, needs to be properly secured. As you would expect, the file permission should be set exactly as the /etc/shadow file:

- Owner: root
- Group: root
- Permissions: (000) Owner: --- Group: --- Other: ---

Locking down the filesystem

The filesystem table (see Chapter 12, "Managing Disks and Filesystems"), /etc/fstab, needs some special attention too. The /etc/fstab file is used at boot time to mount storage devices on filesystems. It is also used by the mount command, the dump command, and the fsck command. The /etc/fstab file should have the following permission settings:

- Owner: root
- Group: root
- Permissions: (644) Owner: rw- Group: r-- Other: r--

Within the filesystem table, there are some important security settings that need to be reviewed. Besides your root, boot, and swap partitions, filesystem options are fairly secure by default. However, you may want to also consider the following:

- Typically, you put the /home subdirectory, where user directories are located, on its own partition. When you add mount options to mount that directory in /etc/

fstab, you can set the nosuid option to prevent SUID and SGID permission-enabled executable programs from running from there. Programs that need SUID and SGID permissions should not be stored in /home and are most likely malicious. You can set the nodev option so that no device file located there will be recognized. Device files should be stored in /dev and not in /home. You can set the noexec option so that no executable programs, which are stored in /home, can be run.

- You can put the /tmp subdirectory, where temporary files are located, on its own partition and use the same options settings as for /home:
  - nosuid
  - nodev
  - noexec
- You can put the /usr subdirectory, where user programs and data are located, on its own partition and set the nodev option so that no device file located there is recognized. After software is installed, the /usr directory often has little or no change (sometimes, it is even mounted read-only for security reasons).
- If the system is configured as a server, you probably want to put the /var directory on its own partition. The /var directory is meant to grow, as log messages and content for web, FTP, and other servers are added. You can use the same mount options with the /var partition as you do for /home:
  - nosuid
  - nodev
  - noexec

Putting the preceding mount options into your /etc/fstab would look similar to the following:

```
/dev/sdb1    /home   ext4    defaults,nodev,noexec,nosuid   1 2
/dev/sdc1    /tmp    ext4    defaults,nodev,noexec,nosuid   1 1
/dev/sdb2    /usr    ext4    defaults,nodev                 1 2
/dev/sdb3    /var    ext4    defaults,nodev,noexec,nosuid   1 2
```

These mount options will help to lock down your filesystem further and add another layer of protection from those with malicious intent. Again, managing the various file permissions and fstab options should be part of your security policy. The items you choose to implement must be determined by your organization's security needs.

Managing software and services

Often, the administrator's focus is on making sure that the needed software and services are on a Linux system. From a security standpoint, you need to take the opposite viewpoint and make sure that the unneeded software and services are not on a Linux system.

Updating software packages

In addition to removing unnecessary services and software, keeping current software up to date is critical for security. The latest bug fixes and security patches are obtained via

software updates. Software package updates were covered in Chapter 9, "Installing Linux," and Chapter 10, "Getting and Managing Software."

Software updates need to be done on a regular basis. How often and when you do it, of course, depends upon your organization's security needs.

You can easily automate software updates, but like removing services and software, it would be wise to test the updates in a test environment first. When updated software shows no problems, you can then update the software on your production Linux systems.

Keeping up with security advisories

As security flaws are found in Linux software, the Common Vulnerabilities and Exposures (CVE) project tracks them and helps to quickly get fixes for those flaws worked on by the Linux community.

Companies such as Red Hat provide updated packages to fix the security flaws and deliver them in what is referred to as *errata*. Errata may consist of a single updated package or multiple updated packages. If you are running Red Hat Enterprise Linux, you search for, identify, and install the RPM (RPM Package Manager) packages associated with a particular CVE and delivered in errata.

As new forms of software packaging become available, make sure that the software in those packages is being checked for vulnerabilities. For example, the Red Hat Container Catalog (`https://access.redhat.com/containers`) lists Red Hat–supported container images along with associated errata and health indexes for each image.

For more information on how security updates are handled in Red Hat Enterprise Linux, refer to the Security Updates page on the Red Hat customer portal (`https://access.redhat.com/security/updates/`). The site contains a wealth of knowledge related to security vulnerabilities and how they are being handled. Being able to get timely security updates is one of the primary reasons companies subscribe critical systems to Red Hat Enterprise Linux.

Advanced implementation

You should be aware of several other important security topics as you are planning your deployments. They include cryptography, Pluggable Authentication Modules (PAM), and SELinux. These advanced and detailed topics have been put into separate chapters—Chapter 23 and Chapter 24.

Monitoring Your Systems

If you do a good job of planning and implementing your system's security, most malicious attacks will be stopped. However, if an attack should occur, you need to be able to recognize it. Monitoring is an activity that needs to be going on continuously.

Monitoring your system includes watching over log files, user accounts, and the filesystem itself. In addition, you need some tools to help you detect intrusions and other types of malware.

Monitoring log files

Understanding how message logging is done is critical to maintaining and troubleshooting a Linux system. Before the `systemd` facility was used to gather messages in what is referred to as the `systemd` journal, messages generated by the kernel and system services were directed to file in the `/var/log` directory. While that is still true to a great extent with `systemd`, you can now also view log messages directly from the `systemd` journal using the `journalctl` command.

The log files for your Linux system are primarily located in the `/var/log` directory. Most of the files in the `/var/log` directory are directed there from the `systemd` journal through the `rsyslogd` service (see Chapter 13, "Understanding Server Administration"). Table 22.3 contains a list of `/var/log` files and a brief description of each.

TABLE 22.3 **Log Files in the `/var/log` Directory**

System Log Name	Filename	Description
Apache Access Log	`/var/log/httpd/access_log`	Logs requests for information from your Apache web server.
Apache Error Log	`/var/log/httpd/error_log`	Logs errors encountered from clients trying to access data on your Apache web server.
Bad Logins Log	`btmp`	Logs bad login attempts.
Boot Log	`boot.log`	Contains messages indicating which system services have started and shut down successfully and which (if any) have failed to start or stop. The most recent bootup messages are listed near the end of the file.
Kernel Log	`dmesg`	Records messages printed by the kernel when the system boots.
Cron Log	`cron`	Contains status messages from the crond daemon.
dpkg Log	`dpkg.log`	Contains information concerning installed Debian packages.
FTP Log	`vsftpd.log`	Contains messages relating to transfers made using the vsftpd daemon (FTP server).
FTP Transfer Log	`xferlog`	Contains information about files transferred using the FTP service.
GNOME Display Manager Log	`/var/log/gdm/:0.log`	Holds messages related to the login screen (GNOME display manager). Yes, there really is a colon in the filename.
LastLog	`lastlog`	Records the last time an account logs in to the system.

(continued)

TABLE 22.3 *(Continued)*

System Log Name	Filename	Description
Login/out Log	`wtmp`	Contains a history of logins and logouts on the system.
Mail Log	`maillog`	Contains information about addresses to which and from which email was sent. Useful for detecting spamming.
MySQL Server Log	`mysqld.log`	Includes information related to activities of the MySQL database server (mysqld).
News Log	`spooler`	Provides a directory containing logs of messages from the Usenet News server if you are running one.
Samba Log	`/var/log/samba/smbd.log` `/var/log/samba/nmbd.log`	Shows messages from the Samba SMB file service daemon.
Security Log	`secure`	Records the date, time, and duration of login attempts and sessions.
Sendmail Log	`sendmail`	Shows error messages recorded by the sendmail daemon.
Squid Log	`/var/log/squid/access.log`	Contains messages related to the squid proxy/caching server.
System Log	`messages`	Provides a general-purpose log file where many programs record messages.
UUCP Log	`uucp`	Shows status messages from the UNIX to UNIX Copy Protocol daemon.
YUM Log	`yum.log`	Shows messages related to RPM software packages.
X.Org X11 Log	`Xorg.0.log`	Includes messages output by the X.Org X server.

The log files that are in your system's `/var/log` directory depend upon what services you are running. Also, some log files are distribution dependent. For example, if you use Fedora, you would not have the `dpkg` log file.

Most of the log files are displayed using the commands `cat`, `head`, `tail`, `more`, or `less`. However, a few of them have special commands for viewing (see Table 22.4).

TABLE 22.4 **Viewing Log Files That Need Special Commands**

Filename	View Command
`btmp`	`dump-utmp btmp`
`dmesg`	`dmesg`
`lastlog`	`lastlog`
`wtmp`	`dump-utmp wtmp`

With the change in Fedora, RHEL, Ubuntu, and other Linux distributions to `systemd` (which manages the boot process and services), as noted earlier, the mechanism for gathering and displaying log messages associated with the kernel and system services has changed as well. Those messages are directed to the `systemd` journal and can be displayed with the `journalctl` command.

You can view journal messages directly from the `systemd` journal instead of simply listing the contents of `/var/log` files. In fact, the `/var/log/messages` file, to which many services direct log messages by default, does not even exist in the latest Fedora release. Instead, you can use the `journalctl` command to display log messages in various ways.

To page through kernel messages, type the following command:

```
# journalctl -k
Logs begin at Sun 2019-06-09 18:59:23 EDT, end at
    Sun 2019-10-20 18:11:06 EDT.
Oct 19 11:43:04 localhost.localdomain kernel:
   Linux version 5.0.9-301.fc30.x86_64
   (mockbuild@bkernel04.phx2.fedoraproject.org)
   (gcc version 9.0.1 20190312 (Red Hat 9.0.1-0.10) (GCC))
        #1 SMP Tue Apr 23 23:57:35 UTC 2019
Oct 19 11:43:04 localhost.localdomain kernel: Command line:
        BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.0.9-301.fc30.x86_64
    root=/dev/mapper/fedora_localhost--live-root ro
    resume=/dev/mapper/fedora_localhost--live-swap
    rd.lvm.lv=fedora_localhost-live/root
    rd.lvm.lv=fedora_localhost-live/swap rhgb quiet
...
```

To view messages associated with a particular service, use the `-u` option followed by the service name to see log messages for any service, as in this example:

```
# journalctl -u NetworkManager.service
# journalctl -u httpd.service
# journalctl -u avahi-daemon.service
```

If you think that a security breach is in progress, you can watch all or selected messages as they come in by following messages. For example, to follow kernel messages or `httpd` messages as they come in, add the `-f` option (press Ctrl+C when you are finished):

```
# journalctl -k -f
# journalctl -f -u NetworkManager.service
```

To check just boot messages, you can list the boot IDs for all system boots and then boot the particular boot instance that interests you. The following examples display boot IDs and then shows boot messages for a selected boot ID:

```
# journalctl --list-boots
-3 6b968e820df345a781cb6935d483374c
   Sun 2019-08-25 12:42:08 EDT—Mon 2019-08-26 14:30:53 EDT
-2 f2c5a74fbe9b4cb1ae1c06ac1c24e89b
```

```
   Mon 2019-09-02 15:49:03 EDT—Thu 2019-09-12 13:08:26 EDT
-1 5d26bee1cfb7481a9e4da3dd7f8a80a0
   Sun 2019-10-13 12:30:27 EDT—Thu 2019-10-17 13:37:22 EDT
 0 c848e7442932488d91a3a467e8d92fcf
   Sat 2019-10-19 11:43:04 EDT—Sun 2019-10-20 18:11:06 EDT
# journalctl -b c848e7442932488d91a3a467e8d92fcf
-- Logs begin at Sun 2019-06-09 18:59:23 EDT,
   end at Sun 2019-10-20 18:21:18 EDT. --
Oct 19 11:43:04 localhost.localdomain kernel: Linux version
5.0.9-301.fc30.x86_64 (mockbuild@bkernel04.phx2.fedoraproject.org)
...
Oct 19 11:43:04 localhost.localdomain kernel: Command line:
   BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.0.9-301.fc30.x86_64
   root=/dev/mapper/fedora_local>
...
Oct 19 11:43:04 localhost.localdomain kernel:
   DMI: Red Hat KVM, BIOS 1.9.1-5.el7_3.3 04/01/2014
Oct 19 11:43:04 localhost.localdomain kernel: Hypervisor detected: KVM
```

Monitoring user accounts

User accounts are often used in malicious attacks on a system by gaining unauthorized access to a current account, by creating new bogus accounts, or by leaving an account behind to access later. To avoid such security issues, watching over user accounts is an important activity.

Detecting counterfeit new accounts and privileges

Accounts created without going through the appropriate authorization should be considered counterfeit. Also, modifying an account in any way that gives it a different unauthorized user identification (UID) number or adds unauthorized group memberships is a form of rights escalation. Keeping an eye on the `/etc/passwd` and `/etc/group` files will monitor these potential breaches.

To help you monitor the `/etc/passwd` and `/etc/group` files, you can use the audit daemon. The audit daemon is an extremely powerful auditing tool that allows you to select system events to track and record them, and it provides reporting capabilities.

To begin auditing the `/etc/passwd` and `/etc/group` files, you need to use the `auditctl` command. Two options at a minimum are required to start this process:

- **`-w`** ***`filename`***: Place a watch on ***`filename`***. The audit daemon tracks the file by its inode number. An *inode number* is a data structure that contains information concerning a file, including its location.
- **`-p`** ***`trigger(s—)`***: If one of these access types occurs (`r`=read, `w`=write, `x`=execute, `a`=attribute change) to *filename*, then trigger an audit record.

In the following example, a watch has been placed on the /etc/passwd file using the auditctl command. The audit daemon will monitor access, which consists of any reads, writes, or file attribute changes:

```
# auditctl -w /etc/passwd -p rwa
```

After you have started a file audit, you may want to turn it off at some point. To turn off an audit, use the command

```
# auditctl -W filename -p trigger(s)
```

To see a list of current audited files and their watch settings, type **auditctl -l** at the command line.

To review the audit logs, use the audit daemon's ausearch command. The only option needed here is the -f option, which specifies which records you want to view from the audit log. The following is an example of the /etc/passwd audit information:

```
# ausearch -f /etc/passwd
time->Fri Feb  7 04:27:01 2020
type=PATH msg=audit(1328261221.365:572):
item=0 name="/etc/passwd" inode=170549
dev=fd:01 mode=0100644 ouid=0 ogid=0
rdev=00:00 obj=system_u:object_r:etc_t:s0
type=CWD msg=audit(1328261221.365:572):  cwd="/"
...
time->Fri Feb  7 04:27:14 2020
type=PATH msg=audit(1328261234.558:574):
item=0 name="/etc/passwd" inode=170549
dev=fd:01 mode=0100644 ouid=0 ogid=0
rdev=00:00 obj=system_u:object_r:etc_t:s0
type=CWD msg=audit(1328261234.558:574):
cwd="/home/johndoe"
type=SYSCALL msg=audit(1328261234.558:574):
arch=40000003 syscall=5 success=yes exit=3
a0=3b22d9 a1=80000 a2=1b6 a3=0 items=1 ppid=3891
pid=21696 auid=1000 uid=1000 gid=1000 euid=1000
suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000
tty=pts1 ses=2 comm="vi" exe="/bin/vi"
 subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023"
----
```

This is a lot of information to review. A few items will help you see what audit event happened to trigger the bottom record:

time: The time stamp of the activity

name: The filename, /etc/passwd, being watched

inode: The /etc/passwd's inode number on this filesystem

uid: The user ID, 1000, of the user running the program

exe: The program, `/bin/vi`, used on the `/etc/passwd` file

To determine what user account is assigned the UID of 1000, look at the `/etc/passwrd` file. In this case, the UID of 1000 belongs to the user `johndoe`. Thus, from the audit event record displayed above, you can determine that account `johndoe` has attempted to use the `vi` editor on the `/etc/passwd` file. It is doubtful that this was an innocent action, and it requires more investigation.

NOTE
The `ausearch` command returns nothing if no watch events on a file have been triggered.

The audit daemon and its associated tools are extremely rich. To learn more about it, look at the man pages for the following audit daemon utilities and configuration files:

auditd: The audit daemon

auditd.conf: The audit daemon configuration file

autditctl: Controls the auditing system

audit.rule: Configuration rules loaded at boot

ausearch: Searches the audit logs for specified items

aureport: Report creator for the audit logs

audispd: Sends audit information to other programs

The audit daemon is one way to keep an eye on important files. You should also review your account and group files on a regular basis with a "human eye" to see if anything looks irregular.

Important files, such as `/etc/passwd`, do need to be monitored for unauthorized account creation. However, just as bad as a new unauthorized user account is an authorized user account with a bad password.

Detecting bad account passwords

Even with all your good efforts, bad passwords will slip in. Therefore, you do need to monitor user account passwords to ensure they that are strong enough to withstand an attack.

One password strength monitoring tool that you can use is the same one malicious users use to crack accounts, John the Ripper. John the Ripper is a free, open source tool that you can use at the Linux command line. It's not installed by default. For a Fedora distribution, you need to issue the command `yum install john` to install it.

TIP
To install John the Ripper on Ubuntu, use the command `sudo apt-get install john`.

In order to use John the Ripper to test user passwords, you must first extract account names and passwords using the `unshadow` command. This information needs to be redirected into a file for use by `john`, as shown here:

```
# unshadow /etc/passwd /etc/shadow > password.file
```

Now edit the `password.file` using your favorite text editor to remove any accounts without passwords. Because it is wise to limit John the Ripper to testing a few accounts at a time, remove any account names that you do not wish to test presently.

CAUTION

The `john` utility is extremely CPU-intensive. It does set its `nice` value to 19 in order to lower its priority. However, it would be wise to run it on a non-production system or during off-peak hours and for only a few accounts at a time.

Now use the `john` command to attempt password cracks. To run `john` against the created password file, issue the command `john` *`filename`*. In the following code snippet, you can see the output from running `john` against the sample `password.file`. For demonstration purposes, only one account was left in the sample file. Further, the account, `Samantha`, was given the bad password of `password`. You can see how little time it took for John the Ripper to crack the password.

```
# john password.file
Loaded 1 password hash (generic crypt(3) [?/32])
password         (Samantha)
guesses: 1  time: 0:00:00:44 100% (2)  c/s: 20.87
 trying: 12345 - missy
Use the "--show" option to display all of the
 cracked passwords reliably
```

To demonstrate how strong passwords are vital, consider what happens when the `Samantha` account's password is changed from `password` to `Password1234`. Even though `Password1234` is still a weak password, it takes longer than 7 days of CPU time to crack it. In the code that follows, `john` was finally aborted to end the cracking attempt.

```
# passwd Samantha
Changing password for user Samantha.
...
# john password.file
Loaded 1 password hash (generic crypt(3) [?/32])
...
time: 0:07:21:55 (3)  c/s: 119  trying: tth675 - tth787
Session aborted
```

As soon as passwords cracking attempts have been completed, the `password.file` should be removed from the system. To learn more about John the Ripper, visit `www.openwall.com/john`.

Monitoring the filesystem

Malicious programs often modify files. They also can try to cover their tracks by posing as ordinary files and programs. However, there are ways to uncover them through the various monitoring tactics covered in the following sections.

Verifying software packages

Typically, if you install a software package from a standard repository or download a reputable site's package, you won't have any problems. But it is always good to double-check your installed software packages to see if they have been compromised. The command to accomplish this is **`rpm -V`** *`package_name`*.

When you verify the software, information from the installed package files is compared against the package metadata (see Chapter 10, "Getting and Managing Software") in the `rpm` database. If no problems are found, the `rpm -V` command returns nothing. However, if there are discrepancies, you get a coded listing. Table 22.5 shows the codes used and a description of the discrepancy.

TABLE 22.5 Package Verification Discrepancies

Code	Discrepancy
S	File size
M	File permissions and type
5	MD5 check sum
D	Device file's major and minor numbers
L	Symbolic links
U	User ownership
G	Group ownership
T	File modified times (`mtime`)
P	Other installed packages this package is dependent upon (aka capabilities)

In the partial list that follows, all of the installed packages are given a verification check. You can see that the codes `5`, `S`, and `T` were returned, indicating some potential problems.

```
# rpm -qaV
5S.T.....  c /etc/hba.conf
...
...T.....    /lib/modules/3.2.1-3.fc16.i686/modules.devname
...T.....    /lib/modules/3.2.1-3.fc16.i686/modules.softdep
```

You do not have to verify all of your packages at once. You can verify just one package at a time. For example, if you want to verify your `nmap` package, you simply enter **`rpm -V nmap`**.

NOTE

To verify packages on Ubuntu, you need the `debsums` utility. It is not installed by default. To install `debsums`, use the command `sudo apt-get install debsums`. To check all installed packages, use the `debsums -a` command. To check one package, type `debsums` *`packagename`*.

Scanning the filesystem

Unless you have recently updated your system, binary files should not have been modified for any reason. Commands such as `find` and `rpm -V` can help you determine if a binary file has been tampered with.

To check for binary file modification, `find` can use the file's modify time, or `mtime`. The file `mtime` is the time when the contents of a file were last modified. Also, `find` can monitor the file's create/change time, or `ctime`.

If you suspect malicious activity, you can quickly scan your filesystem to see if any binaries were modified or changed today (or yesterday, depending upon when you think the intrusion took place). To do this scan, use the `find` command.

In the example that follows, a scan is made of the `/sbin` directory. To see if any binary files were modified less than 24 hours ago, the command `find /sbin -mtime -1` is used. In the example, several files are displayed, showing that they were modified recently. This indicates that malicious activity is taking place on the system. To investigate further, review each individual file's times, using the **`stat`** *`filename`* command, as shown here:

```
# find /sbin -mtime -1
/sbin
/sbin/init
/sbin/reboot
/sbin/halt
#
# stat /sbin/init
  File: '/sbin/init' -> '../bin/systemd'
  Size: 14    Blocks: 0      IO Block: 4096   symbolic link
Device: fd01h/64769d    Inode: 9551        Links: 1
Access: (0777/lrwxrwxrwx)
Uid: (    0/    root)   Gid: (    0/    root)
Context: system_u:object_r:bin_t:s0
Access: 2016-02-03 03:34:57.276589176 -0500
Modify: 2016-02-02 23:40:39.139872288 -0500
Change: 2016-02-02 23:40:39.140872415 -0500
 Birth: -
```

You could create a database of all of the binary's original `mtimes` and `ctimes` and then run a script to find current `mtimes` and `ctimes`, compare them against the database, and note any discrepancies. However, this type of program has already been created and works well. It's called an Intrusion Detection System, and it is covered later in this chapter.

You need to perform several other filesystem scans on a regular basis. Favorite files or file settings of malicious attackers are listed in Table 22.6. The table also lists the commands to perform the scans and why the file or file setting is potentially problematic.

TABLE 22.6 Additional Filesystem Scans

File or Setting	Scan Command	Problem with File or Setting
SUID permission	`find / -perm -4000`	Allows anyone to become the file's owner temporarily while the file is being executed in memory.
SGID permission	`find / -perm -2000`	Allows anyone to become a group member of the file's group temporarily while the file is being executed in memory.
rhost files	`find /home -name .rhosts`	Allows a system to trust another system completely. It should not be in `/home` directories.
Ownerless files	`find / -nouser`	Indicates files that are not associated with any username.
Groupless files	`find / -nogroup`	Indicates files that are not associated with any group name.

The `rpm -V` *package* command can tell you information about changes that have occurred to a file after it has been installed from an RPM package. For each file that has changed from the selected package since it was installed, you can see the following information:

```
S   Size of the file differs
M   Permissions or file type (Mode) of the file differs
5   Digest differs (formerly MD5 sum)
D   Device major/minor number is mismatched
L   The readLink(2) path is mismatch
U   User ownership differs
G   Group ownership differs
T   mTime differs
P   caPabilities differ
```

By default, only changed files appear. Add `-v` (verbose) to also show files that have not changed. Here is an example:

```
# rpm -V samba
S.5....T.    /usr/sbin/eventlogadm
```

In this example, I echoed a few characters into the `eventlogadm` binary. The `S` shows the size of the file changes, `5` shows the digest no longer matches the original digest, and `T` says the modification time on the file has changed.

These filesystem scans help monitor what is going on in your system and help detect malicious attacks. However, other types of attacks can occur to your files, including viruses and rootkits.

Detecting viruses and rootkits

Two popular malicious attack tools are viruses and rootkits because they stay hidden while performing their malicious activities. Linux systems need to be monitored for both such intrusions.

Monitoring for viruses

A *computer virus* is malicious software that can attach itself to already installed system software, and it has the ability to spread through media or networks. It is a misconception that there are no Linux viruses. The malicious creators of viruses do often focus on the more popular desktop operating systems, such as Windows. However, that does not mean that viruses are not created for the Linux systems.

Even more important, Linux systems are often used to handle services, such as mail servers, for Windows desktop systems. Therefore, Linux systems used for such purposes need to be scanned for Windows viruses as well.

Antivirus software scans files using virus signatures. A *virus signature* is a hash created from a virus's binary code. The hash will positively identify that virus. Antivirus programs have a virus signature database that is used to compare against files to see if there is a signature match. Depending upon the number of new threats, a virus signature database can be updated often to provide protection from these new threats.

A good antivirus software choice for your Linux system, which is open source and free, is ClamAV. To install ClamAV on a Fedora or RHEL system, type the command **`dnf install clamav`**. You can find out more about ClamAV at `clamav.net`, where there is documentation on how to set up and run the antivirus software.

Tip

You can review the packages available for Ubuntu installation by entering the command **`apt-cache search clamav`**. A couple of different packages are available for Ubuntu, so review the ClamAV website information before you choose a package.

Monitoring for rootkits

A rootkit is a little more insidious than a virus. A *rootkit* is a malicious program that does the following:

- Hides itself, often by replacing system commands or programs
- Maintains high-level access to a system
- Is able to circumvent software created to locate it

The purpose of a rootkit is to get and maintain root-level access to a system. The term was created by putting together *root*, which means that it has to have administrator access, and *kit*, which means it is usually several programs that operate in concert.

A rootkit detector that can be used on a Linux system is `chkrootkit`. To install `chkrootkit` on a Fedora or RHEL system, issue the command **`yum install chkrootkit`**. To install `chkrookit` on an Ubuntu system, use the command **`sudo apt-get install chkrootkit`**.

Tip

It is best to use a Live CD or flash drive to run `chkrootkit` so that the results are not circumvented by a rootkit. The Fedora Security Spin has `chkrootkit` on its Live CD. You can get this distribution at `labs.fedoraproject.org/en/security`.

Finding a rootkit with `chkrootkit` is simple. After installing the package or booting up the Live CD, type in **`chkrootkit`** at the command line. It searches the entire file structure denoting any infected files.

The code that follows shows a run of `chkrootkit` on an infected system. The `grep` command was used to search for the keyword `INFECTED`. Notice that many of the files listed as "infected" are bash shell command files. This is typical of a rootkit.

```
# chkrootkit | grep INFECTED
Checking 'du'... INFECTED
Checking 'find'... INFECTED
Checking 'ls'... INFECTED
Checking 'lsof'... INFECTED
Checking 'pstree'... INFECTED
Searching for Suckit rootkit... Warning: /sbin/init INFECTED
```

In the last line of the preceding `chkrootkit` code is an indication that the system has been infected with the Suckit rootkit. It actually is not infected with this rootkit. When running utilities, such as antivirus and rootkit-detecting software, you often get a number of false positives. A *false positive* is an indication of a virus, rootkit, or other malicious activity that does not really exist. In this particular case, this false positive is caused by a known bug.

The `chkrootkit` utility should have regularly scheduled runs and, of course, should be run whenever a rootkit infection is suspected. To find more information on `chkrootkit`, go to `chkrootkit.org`.

Tip

Another rootkit detector that might interest you is called Rootkit Hunter (`rkhunter`). Run the `rkhunter` script to check your system for malware and known rootkits. Configure `rkhunter` in the `/etc/rkhunter.conf` file. For a simple example, run `rkhunter -c` to check the filesystem for a variety of rootkits and vulnerabilities.

Detecting an intrusion

Intrusion Detection System (IDS) software—a software package that monitors a system's activities (or its network) for potential malicious activities and reports these activities—can help you monitor your system for potential intrusions. Closely related to Intrusion Detection System software is a software package that prevents an intrusion, called *Intrusion Prevention System* software. Some of these packages are bundled together to provide Intrusion Detection and Prevention.

Several Intrusion Detection System software packages are available for a Linux system. A few of the more popular utilities are listed in Table 22.7. You should know that `tripwire` is no longer open source. However, the original `tripwire` code is still available. See the `tripwire` website listed in Table 22.7 for more details.

TABLE 22.7 Popular Linux Intrusion Detection Systems

IDS Name	Installation	Website
aide	`yum install aide` `apt-get install aide`	`http://aide.sourceforge.net`
Snort	`rpm` or `tarball` packages from website	`http://snort.org`
tripwire	`yum install tripwire` `apt-get install tripwire`	`http://tripwire.org`

The Advanced Intrusion Detection Environment (aide) IDS uses a method of comparison to detect intrusions. When you were a child, you may have played the game of comparing two pictures and finding what was different between them. The `aide` utility uses a similar method. A "first picture" database is created. At some time later, another database "second picture" is created, and `aide` compares the two databases and reports what is different.

To begin, you need to take that "first picture." The best time to create this picture is when the system has been freshly installed. The command to create the initial database is `aide -i` and it takes a long time to run. Some of its output follows. Notice that `aide` tells you where it is creating its initial "first picture" database.

```
# aide -i
AIDE, version 0.16.11

### AIDE database at /var/lib/aide/aide.db.new.gz initialized.
```

The next step is to move the initial "first picture" database to a new location. This protects the original database from being overwritten. Plus, the comparison does not work unless the database is moved. The command to move the database to its new location and give it a new name is as follows:

```
# cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

When you are ready to check whether your files have been tampered with, you need to create a new database, "second picture," and compare it to the original database, "first picture." The check option on the `aide` command, `-C`, creates a new database and runs a comparison against the old database. The output shown next illustrates this comparison being done and the `aide` command reporting on some problems.

```
# aide -C
```

```
...
---------------------------------------------------
Detailed information about changes:
---------------------------------------------------
File: /bin/find
Size : 189736 , 4620
Ctime : 2020-02-10 13:00:44 , 2020-02-11 03:05:52
MD5 : <NONE> , rUJj8NtNa1v4nmV5zfoOjg==
RMD160 : <NONE> , 0CwkiYhqNnfwPUPM12HdKuUSFUE=
SHA256 : <NONE> , jg60Soawj4S/UZXm5h4aEGJ+xZgGwCmN

File: /bin/ls
Size : 112704 , 6122
Ctime : 2020-02-10 13:04:57 , 2020-02-11 03:05:52
MD5 : POeOop46MvRx9qfEoYTXOQ== , IShMBpbSOY8axhw1Kj8Wdw==
RMD160 : N3V3Joe5Vo+cOSSnedf9PCDXYkI= ,
 e0ZneB7CrWHV42hAEgT2lwrVfP4=
SHA256 : vuOFe6FUgoAyNgIxYghOo6+SxR/zxS1s ,
 Z6nEMMBQyYm8486yFSIbKBuMUi/+jrUi
...
File: /bin/ps
Size : 76684 , 4828
Ctime : 2020-02-10 13:05:45 , 2020-02-11 03:05:52
MD5 : 1pCVAWbpeXINiBQWSUEJfQ== , 4ElJhyWkyMtm24vNLya6CA==
RMD160 : xwICWNtQH242jHsH2E8rV5kgSkU= ,
 AZlI2QNlKrWH45i3/V54H+1QQZk=
SHA256 : ffUDesbfxx3YsLDhD0bLTW0c6nykc3m0 ,
 w1qXvGWPFzFir5yxN+n6t3eOWw1TtNC/
...
File: /usr/bin/du
Size : 104224 , 4619
Ctime : 2020-02-10 13:04:58 , 2020-02-11 03:05:53
MD5 : 5DUMKWj6LodWj4C0xfPBIw== , nzn7vrwfBawAeL8nkayICg==
RMD160 : Zlbm0f/bUWRLgi1B5nVjhanuX9Q= ,
 2e5S00lBWqLq4Tnac4b6QIXRCwY=
SHA256 : P/jVAKr/SO0epBBxvGP900nLXrRY9tnw ,
 HhTqWgDyIkUDxA1X232ijmQ/OMA/kRgl
File: /usr/bin/pstree
Size : 20296 , 7030
Ctime : 2020-02-10 13:02:18 , 2020-02-11 03:05:53
MD5 : <NONE> , ry/MUZ7XvU4L2QfWJ4GXxg==
RMD160 : <NONE> , tFZer6As9EoOi58K7/LgmeiExjU=
SHA256 : <NONE> , iAsMkqNShagD4qe7dL/EwcgKTRzvKRSe
...
```

The files listed by the `aide` check in this example are infected. However, `aide` can also display many false positives.

Where `aide` databases are created, what comparisons are made, and several other configuration settings are handled in the `/etc/aide.conf` file. The following is a partial display of the file. You can see the names of the database file and the log file directories set here:

```
# cat /etc/aide.conf
# Example configuration file for AIDE.

@@define DBDIR /var/lib/aide
@@define LOGDIR /var/log/aide

# The location of the database to be read.
database=file:@@{DBDIR}/aide.db.gz

# The location of the database to be written.
#database_out=sql:host:port:database:login_name:passwd:table
#database_out=file:aide.db.new
database_out=file:@@{DBDIR}/aide.db.new.gz
...
```

An Intrusion Detection System can be a big help in monitoring the system. When potential intrusions are detected, comparing the output to information from other commands (such as `rpm -V`) and log files can help you better understand and correct any attacks on your system.

Auditing and Reviewing Linux

You must understand two important terms when you are auditing the health of your Linux system. A *compliance review* is an audit of the overall computer system environment to ensure that the policies and procedures you have set for the system are being carried out correctly. A *security review* is an audit of current policies and procedures to ensure that they follow accepted best security practices.

Conducting compliance reviews

Similar to audits in other fields, such as accounting, audits can be conducted internally or by external personnel. These reviews can be as simple as someone sitting down and comparing implemented security to your company's stated policies. However, more popular is conducting audits using penetration testing.

Penetration testing is an evaluation method used to test a computer system's security by simulating malicious attacks. It is also called pen testing and ethical hacking. No longer do you have to gather tools and the local neighborhood hacker to help you conduct these tests.

Kali Linux (`https://www.kali.org/`) is a distribution created specifically for penetration testing. It can be used from a live DVD or a flash drive. Training on the use of Kali Linux is offered by Offensive Security (`https://www.offensive-security.com/information-security-training/`).

While penetration testing is lots of fun, for a thorough compliance review, a little more is needed. You should also use checklists from industry security sites.

Conducting security reviews

Conducting a security review requires that you know current best security practices. There are several ways to stay informed about best security practices. The following is a brief list of organizations that can help you.

- United States Cybersecurity and Infrastructure Security Agency (CISA)
  - URL: `www.us-cert.gov`
  - Offers the National Cyber Alert System
  - Offers RSS feeds on the latest security threats
- The SANS Institute
  - URL: `www.sans.org/security-resources`
  - Offers Computer Security Research newsletters
  - Offers RSS feeds on the latest security threats
- Gibson Research Corporation
  - URL: `www.grc.com`
  - Offers the *Security Now!* security netcast

Information from these sites will assist you in creating stronger policies and procedures. Given how fast the best security practices change, it would be wise to conduct security reviews often, depending upon your organization's security needs.

Now you understand a lot more about basic Linux security. The hard part is actually putting all of these concepts into practice.

Summary

Basic Linux security practices, such as managing user accounts, securing passwords, and managing software and services, form the foundation for all other security on your Linux system. With that foundation in place, ongoing monitoring of your system includes watching over system log files, checking for malicious intrusions, and monitoring the filesystem.

Reviews of your security policies are also important to keep up on a regular basis. Audits assist in ensuring that your Linux system is secured and the proper security policies and practices are in place.

You have completed your first step of gathering basic security procedures and principles knowledge. It is not enough just to know the basics. You need to add advanced Linux security tools to your security toolbox. In the next chapter, advanced security topics of cryptography and authentication modules are covered.

Exercises

Refer to the material in this chapter to complete the tasks that follow. If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task). Try each of the exercises before referring to the answers. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks will work on other Linux systems as well).

1. Check log messages from the `systemd` journal for the following services: `NetworkManager.service`, `sshd.service`, and `auditd.service`.
2. List the permissions of the file containing your system's user passwords and determine if they are appropriate.
3. Determine your account's password aging and if it will expire using a single command.
4. Start auditing writes to the `/etc/shadow` with the `auditd` daemon and then check your audit settings.
5. Create a report from the `auditd` daemon on the `/etc/shadow` file, and then turn off auditing on that file.
6. Install the lemon package, damage the `/usr/bin/lemon` file (perhaps copy `/etc/services` there), verify that the file has been tampered with, and remove the lemon package.
7. You suspect that you have had a malicious attack on your system today and important binary files have been modified. What command should you use to find these modified files?
8. Install and run `chkrootkit` to see if the malicious attack from the exercise above installed a rootkit.
9. Find files with the `SUID` or `SGID` permission set.
10. Install the `aide` package, run the `aide` command to initialize the aide database, copy the database to the correct location, and run the `aide` command to check if any important files on your system have been modified.

CHAPTER 23

Understanding Advanced Linux Security

IN THIS CHAPTER

Understanding hashing and encryption

Checking file integrity

Encrypting files, directories, and filesystems

Understanding pluggable authentication modules

Managing Linux security with PAM

Due to ever-changing and growing threats, implementing basic computer security is no longer enough. As malicious users gain access to and knowledge of advanced tools, so must a Linux system administrator. Understanding advanced computer security topics and tools must be part of your preparation.

In this chapter, you will learn about cryptography basics, such as ciphers and encryption. You will also learn how the authentication module utility can simplify your administrative duties, even though it is an advanced security topic.

Implementing Linux Security with Cryptography

Using cryptography enhances the security of your Linux system and its network communications. *Cryptography* is the science of concealing information. It has a long and rich history that goes back far before computers were around. Because of its heavy use of mathematical algorithms, cryptography has easily transitioned to computers. Linux comes with many cryptographic tools ready for you to use.

To understand cryptographic concepts and the various Linux tools, you should know a few cryptography terms:

Plain text: Text that a human or machine can read and comprehend

Ciphertext: Text that a human or machine cannot read and comprehend

Encryption: The process of converting plain text into ciphertext using an algorithm

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Decryption: The process of converting cipher text into plain text using an algorithm

Cipher: The algorithm used to encrypt plain text into ciphertext and decrypt ciphertext into plain text

Block cipher: A cipher that breaks data into blocks before encrypting

Stream cipher: A cipher that encrypts the data without breaking it up

Key: A piece of data required by the cipher to encrypt or decrypt data successfully

Parents of young children often use a form of cryptography. They spell words instead of speaking them. A parent may take the plain-text word "candy" and turn it into ciphertext by saying to the other parent "C-A-N-D-Y." The other parent decrypts the word by using the same spelling cipher and recognizes that the word is "candy." Unfortunately, it does not take children long to learn how to decrypt via the spelling cipher.

You may have noticed that hashing was not included in the preceding cryptography definition list. Hashing needs some special attention because it is often confused with encryption.

Understanding hashing

Hashing is not encryption, but it is a form of cryptography. Remember from Chapter 22, "Understanding Basic Linux Security," that *hashing* is a one-way mathematical process used to create ciphertext. However, unlike encryption, after you create a hash, you cannot dehash it back to its original plain text.

In order for a hashing algorithm to be used in computer security, it needs to be *collision-free*, which means that the hashing algorithm does not output the same hash for two totally different inputs. Each input must have a unique hashed output. Thus, *cryptographic hashing* is a one-way mathematical process that is collision-free.

By default, cryptography is already in use on a Linux system. For example, the `/etc/shadow` file contains hashed passwords. Hashing is used on Linux systems for the following:

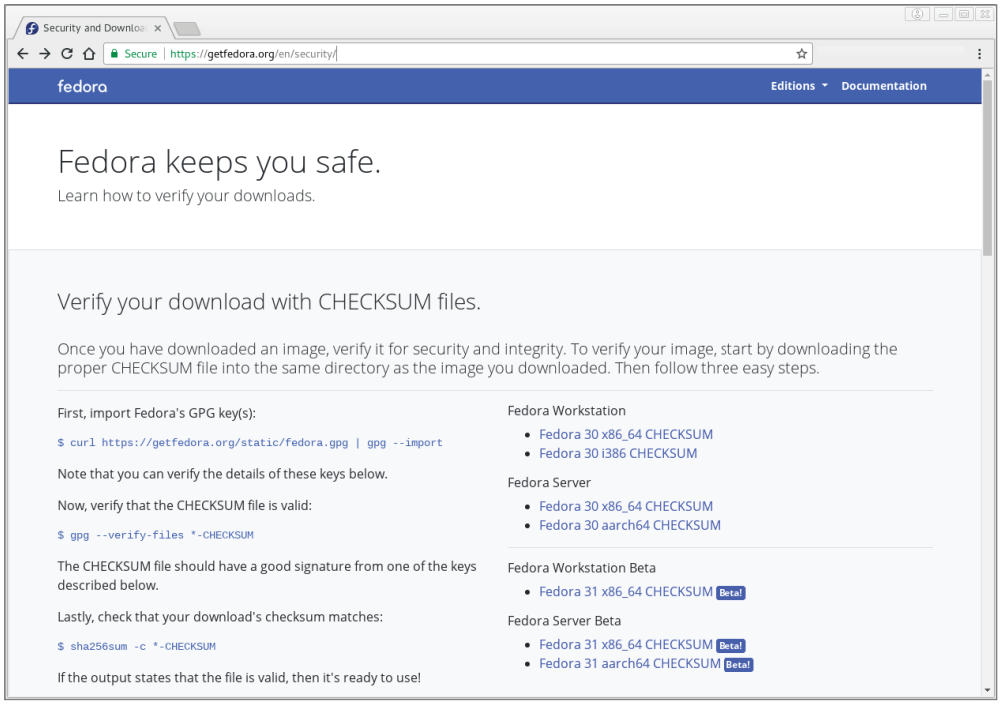
- Passwords (Chapter 22)
- Verifying files
- Digital signatures
- Virus signatures (Chapter 22)

A hash is also called a *message digest*, *checksum*, *fingerprint*, or *signature*. One Linux utility that produces message digests is the `sha256sum` utility. In Chapter 10, "Getting and Managing Software," you learned about getting software for your Linux system. When you download a software file, you can make sure that the file was not corrupted on download.

Figure 23.1 shows the website for downloading the Fedora distribution software (stored as a file in the form that is referred to as an *ISO image*). The web page describes how to get and use the `sha256sum` utility to ensure that the ISO image you downloaded was not corrupted during the download.

FIGURE 23.1

The Fedora ISO security page tells how to get and check with `sha256sum`.



A hash is made up of a software file at its original location, using the SHA-256 hash algorithm. The hash results can be posted in public, as was done in Figure 23.1. To ensure the integrity of your downloaded software file, you create an `sha256sum` hash of the software file at your location. You then compare the results of your hash to the posted hash results. If they match, the software file was not corrupted upon download.

To create your hash, run the `sha256sum` command on the ISO image after you download that image. The `sha256sum` hash results for the downloaded software file are shown in the code that follows:

```
$ sha256sum Fedora-Workstation-Live-x86_64-30-1.2.iso
a4e2c49368860887f1cc1166b0613232d4d5de6b46f29c9756bc7cfd5e13f39f
   Fedora-Workstation-Live-x86_64-30-1.2.iso
```

The resulting hash *does* match the one available from the website in Figure 23.1. This means that the downloaded ISO file has not been corrupted and is ready for use.

You can implement even more cryptography besides hashing on your Linux system. The Linux utilities to do so are very easy to use. However, first you need to understand a few more underlying cryptography concepts.

Understanding encryption/decryption

The primary use of cryptography on a Linux system is to encode data to hide it (encryption) from unauthorized eyes and then decode the data (decryption) for authorized eyes. On a Linux system, you can encrypt the following:

- Individual files
- Partitions and volumes
- Web page connections
- Network connections
- Backups
- Zip files

These encryption/decryption processes use special math algorithms to accomplish their task. The algorithms are called *cryptographic ciphers*.

Understanding cryptographic ciphers

One of the original ciphers, called the *Caesar Cipher*, was created and used by Julius Caesar. It was terribly easy to crack, however. Today, many more secure ciphers are available. Understanding how each cipher works is important because the strength of the cipher you choose should directly relate to the security needs of your data. Table 23.1 lists a few modern ciphers.

TABLE 23.1 Cryptography Ciphers

Method	Description
AES (Advanced Encryption Standard), also called Rijndael	*Symmetric cryptography.* Block cipher, encrypting data in 128-, 192-, 256-, 512-bit blocks using a 128-, 192-, 256, or 512-bit key for encrypting/decrypting.
Blowfish	*Symmetric cryptography.* Block cipher, encrypting data in 64-bit blocks using the same 32-bit to 448-bit keys for encrypting/decrypting.
CAST5	*Symmetric cryptography.* Block cipher, encrypting data in 64-bit blocks using the same up to 128-bit key for encrypting/decrypting.
DES (Data Encryption Standard)	No longer considered secure. *Symmetric cryptography.* Block cipher, encrypting data in 64-bit blocks using the same 56-bit key for encrypting/decrypting.
3DES	Improved DES cipher. *Symmetric cryptography.* Data is encrypted up to 48 times with three different 56-bit keys before the encryption process is completed.

Method	Description
El Gamal	*Asymmetric cryptography.* Uses two keys derived from a logarithm algorithm.
Elliptic Curve Cryptosystems	*Asymmetric cryptography.* Uses two keys derived from an algorithm containing two randomly chosen points on an elliptic curve.
IDEA	*Symmetric cryptography.* Block cipher, encrypting data in 64-bit blocks using the same 128-bit key for encrypting/decrypting.
RC4 also called ArcFour or ARC4	Stream cipher, encrypting data in 64-bit blocks using a variable key size for encrypting/decrypting.
RC5	*Symmetric cryptography.* Block cipher, encrypting data in 32-, 64-, or 128-bit blocks using the same up to 2,048-bit keys for encrypting/decrypting.
RC6	*Symmetric cryptography.* Same as RC5, but slightly faster.
Rijndael also called AES	*Symmetric cryptography.* Block cipher, encrypting data in 128-, 192-, 256-, 512-bit blocks using a 128-, 192-, 256-, or 512-bit key for encrypting/decrypting.
RSA	Most popular *asymmetric cryptography.* Uses two keys derived from an algorithm containing a multiple of two randomly generated prime numbers.

Understanding cryptographic cipher keys

Cryptographic ciphers require a piece of data, called a *key*, to complete their mathematical process of encryption/decryption. The key can be either a single key or a pair of keys.

Notice the different cipher key sizes listed in Table 23.1. The key size is directly related to how easily the cipher is cracked. The bigger the key size, the less the chance of cracking the cipher. For example, DES is no longer considered secure because of its small 56-bit key size. However, a cipher with a key size of 256 bits or 512 bits is considered secure because it would take trillions of years to brute-force crack such a keyed cipher.

Symmetric key cryptography

Symmetric cryptography, also called *secret key* or *private key* cryptography, encrypts plain text using a single keyed cipher. The same key is needed in order to decrypt the data. The advantage of symmetric key cryptography is speed. The disadvantage is the need to share the single key if the encrypted data is to be decrypted by another person.

An example of symmetric key cryptography on a Linux system is accomplished using the OpenPGP utility, GNU Privacy Guard, `gpg2`. The gnupg2 package is installed by default in Fedora and RHEL. For Ubuntu, you need to install the gnupg2 package to get the `gpg2` command.

Encrypting and decrypting a tar archive file

The example that follows shows the `tar` command used to create a compressed tar archive (`backup.tar.gz`) and the `gpg2` utility used to encrypt the file. With the `-c` option, `gpg2` encrypts the file with a symmetric key. The original file is kept and a new encrypted file, `backup.tar.gz.gpg`, is created.

```
# tar -cvzf /tmp/backup.tar.gz /etc
# gpg2 -c --force-mdc \
    -o /tmp/backup.tar.gz.gpg /tmp/backup.tar.gz
Enter passphrase: ******
Repeat passphrase: ******
# cd /tmp ; file backup*
/tmp/enc/backup.tar.gz:     gzip compressed data, last modified: Thu
    Jan 30 02:36:48 2020, from Unix, original size modulo 2^32
49121280
/tmp/enc/backup.tar.gz.gpg: GPG symmetrically encrypted data
(CAST5 cipher)
```

The single key used to encrypt the file is protected by a passphrase. This passphrase is simply a password or phrase chosen by the user at the time of encryption.

To decrypt the file, use the `gpg2` utility again. For example, if you were to hand the file to another user, that user could run `gpg2` with the `-d` option and provide the passphrase for the secret key.

```
$ gpg2 -d --force-mdc /tmp/backup.tar.gz.gpg > /tmp/backup.tar.gz
<A pop-up window asks for your passphrase>
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase
...
```

The result here is that the original tar file is decrypted and copied to `/tmp/backup.tar.gz`. If the `gpg-agent` daemon is running on the system, that passphrase is cached so that file could be decrypted again without entering the passphrase again.

Symmetric key cryptography is rather simple and easy to understand. Asymmetric cryptography is much more complicated and often is a point of confusion in cryptography.

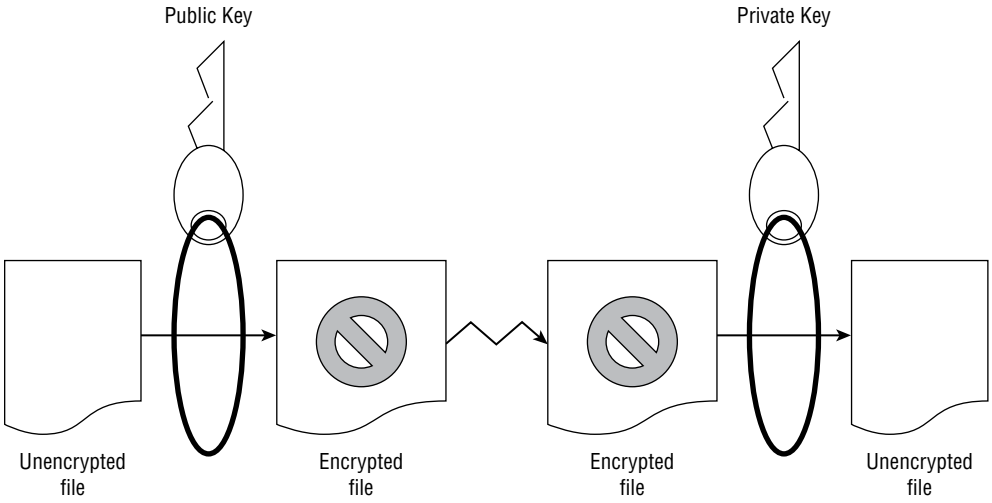
Asymmetric key cryptography

Asymmetric cryptography, also called private/public key cryptography, uses two keys, called a *key pair*. A key pair consists of a public key and a private key. The public key is just that—public. There is no need to keep it secret. The private key needs to be kept secret.

The general idea of asymmetric key cryptography is shown in Figure 23.2. A plain-text file is encrypted using a public key of a key pair. The encrypted file then can be securely transmitted to another person. To decrypt the file, the private key is used. This private key must be from the public/private key pair. Thus, data that has been encrypted with the public key can only be decrypted with its private key. The advantage of asymmetric cryptography is heightened security. The disadvantage is speed and key management.

FIGURE 23.2

Basic asymmetric key cryptography



Generating a key pair

You can perform asymmetric encryption on your Linux system using `gpg2`. It is a very versatile cryptographic utility. Before you can encrypt a file, you must first create your key pair and a "key ring." In the example that follows, the `gpg2 --gen-key` command was used. This command creates a public/private key pair for the user `johndoe`, according to his desired specifications. It also generates a key ring to store his keys.

```
$ gpg2 --gen-key
gpg (GnuPG) 2.2.9; Copyright (C)
 2018 Free Software Foundation, Inc.
...
GnuPG needs to construct a user ID to identify your key.
Real name: John Doe
Email address: jdoe@example.com
You selected this USER-ID:
    "John Doe <jdoe@gmail.com>"
Change (N)ame, (E)mail or (O)kay/(Q)uit? O
```

```
You need a Passphrase to protect your secret key.
<A pop-up window prompts you for a passphrase>
Enter passphrase: **********
Repeat passphrase: **********
...
gpg: /home/jdoe/.gnupg/trustdb.gpg: trustdb created
gpg: key 383D645D9798C173 marked as ultimately trusted
gpg: directory '/home/jdoe/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/jdoe/.gnupg/openpgp-
revocs.d/7469BCD3D05A4
3130F1786E0383D645D9798C173.rev'
public and secret key created and signed.
pub   rsa2048 2019-10-27 [SC] [expires: 2021-10-26]
      7469BCD3D05A43130F1786E0383D645D9798C173
uid                      John Doe <jdoe@example.com>
sub   rsa2048 2019-10-27 [E] [expires: 2021-10-26]
```

In the preceding example, the `gpg2` utility asks for several specifications to generate the desired public/private keys:

User ID: This identifies the public key portion of the public/private key pair.

Email Address: This is the email address associated with the key.

Passphrase: This is used to identify and protect the private key portion of the public/private key pair.

The user `johndoe` can check his key ring by using the `gpg2 --list-keys` command, as shown in the code that follows. Notice the User ID (UID) of the public key is displayed just as it was created, containing `johndoe`'s real name, comment, and email address.

```
$ gpg2 --list-keys
/home/jdoe/.gnupg/pubring.kbx
------------------------------
pub   rsa2048 2019-10-27 [SC] [expires: 2021-10-26]
      7469BCD3D05A43130F1786E0383D645D9798C173
uid           [ultimate] John Doe <jdoe@example.com>
sub   rsa2048 2019-10-27 [E] [expires: 2021-10-26]
```

After the key pair and key ring are generated, files can be encrypted and decrypted. First, the public key must be extracted from the key ring so that it can be shared. In the example that follows, the `gpg2` utility is used to extract the public key from `johndoe`'s key ring. The extracted key is put into a file to be shared. The filename can be any name you wish it to be. In this case, the user `johndoe` chose the filename `JohnDoe.pub`.

```
$ gpg2 --export John Doe > JohnDoe.pub
$ ls *.pub
JohnDoe.pub
$ file JohnDoe.pub
JohnDoe.pub: PGP/GPG key public ring (v4) created Sun Oct 27 16:24:27
2019 RSA (Encrypt or Sign) 2048 bits MPI=0xc57a29a6151b3e8d...
```

Sharing a public key

The file containing the public key can be shared in any number of ways. It can be sent as an attachment via email or even posted on a web page. The public key is considered public, so there is no need to hide it. In the example that follows, `johndoe` has given the file containing his public key to the user `jill`. She adds `johndoe`'s public key to her key ring using the `gpg2 --import` command. The user `jill` verifies that `johndoe`'s public key is added using the `gpg2 --list-keys` command to view her key ring.

```
$ ls *.pub
JohnDoe.pub
$ gpg2 --import JohnDoe.pub
gpg: directory '/home/jill/.gnupg' created
...
gpg: directory '/home/jill/.gnupg' created
gpg: keybox '/home/jill/.gnupg/pubring.kbx' created
gpg: /home/jill/.gnupg/trustdb.gpg: trustdb created
gpg: key 383D645D9798C173: public key "John Doe <jdoe@example.com>"
imported
gpg: Total number processed: 1
gpg:               imported: 1
$ gpg2 --list-keys
/home/jill/.gnupg/pubring.gpg
-----------------------------
pub   rsa2048 2019-10-27 [SC] [expires: 2021-10-26]
      7469BCD3D05A43130F1786E0383D645D9798C173
uid           [ unknown] John Doe <jdoe@example.com>
sub   rsa2048 2019-10-27 [E] [expires: 2021-10-26]
```

Encrypting an email message

After the key is added to the key ring, that public key can be used to encrypt data for the public key's original owner. In the example code that follows, note that `jill` has created a text file, `MessageForJohn.txt`, for user `johndoe`.

- She encrypts the file using *his* public key.
- The encrypted file, `MessageForJohn`, is created by the `--out` option.
- The option `--recipient` identifies `johndoe`'s public key using only the real name portion of his public key's UID in quotation marks, `"John Doe"`.

```
$ gpg2 --out MessageForJohn --recipient "John Doe" \
    --encrypt MessageForJohn.txt
...
$ ls
JohnDoe.pub  MessageForJohn  MessageForJohn.txt
```

The encrypted message file, `MessageForJohn`, created from the plain-text file, `MessageForJohn.txt`, can be securely sent to the user `johndoe`. In order to decrypt this message, `johndoe` uses *his* private key, identified and protected by the secret passphrase used to create the key originally. After `johndoe` provides the proper passphrase, `gpg2` decrypts

the message file and puts it into the file `JillsMessage`, designated by the `--out` option. Once it's decrypted, he can read the plaintext message.

```
$ ls MessageForJohn
MessageForJohn
$ gpg2 --out JillsMessage --decrypt MessageForJohn
<A pop-up window prompts you for a passphrase>
gpg: encrypted with 2048-bit RSA key, ID D9EBC5F7317D3830, created
2019-10-27
      "John Doe <jdoe@example.com>"
$ cat JillsMessage
I know you are not the real John Doe.
```

To review, the steps needed for encryption/decryption of files using asymmetric keys include the following:

1. Generate the key pair and the key ring.
2. Export a copy of your public key to a file.
3. Share the public key file.
4. Individuals who want to send you encrypted files add your public key to their key ring.
5. A file is encrypted using *your* public key.
6. The encrypted file is sent to you.
7. You decrypt the file using *your* private key.

You can see why asymmetric keys can cause confusion! Remember that in asymmetric cryptography, each public and private key is a paired set that works together.

Understanding digital signatures

A *digital signature* is an electronic originator used for authentication and data verification. A digital signature is not a scan of your physical signature. Instead, it is a cryptographic token sent with a file, so the file's receiver can be assured that the file came from you and has not been modified in any way.

When you create a digital signature, the following steps occur:

1. You create a file or message.
2. Using the `gpg2` utility, you create a hash or message-digest of the file.
3. The `gpg2` utility then encrypts the hash and the file, using an asymmetric key cipher. For the encryption, the private key of the public/private key pair is used. This is now a digitally signed encrypted file.
4. You send the encrypted hash (aka digital signature) and file to the receiver.
5. The receiver re-creates the hash or message digest of the received encrypted file.

6. Using the `gpg2` utility, the receiver decrypts the received digital signature using the public key, to obtain the original hash or message digest.
7. The `gpg2` utility compares the original hash to the re-created hash to see if they match. If they match, the receiver is told the digital signature is good.
8. The receiver can now read the decrypted file.

Notice in step 3 that the private key is used first. In the description of asymmetric key cryptography, the public key was used first. Asymmetric key cryptography is flexible enough to allow you to use your private key to encrypt and the receiver to use your public key to decrypt.

NOTE

Digital signatures have their own special ciphers. While several ciphers can handle both encryption and creating signatures, there are a few whose only job is to create digital signatures. Previously, the most popular cryptographic ciphers to use in creating signatures were RSA and Digital Signature Algorithm (DSA). The RSA algorithm can be used for both encryption and creating signatures, while DSA can be used only for creating digital signatures. Today, Ed25519 is considered to be more secure and faster than RSA, and ECDSA provides better protection than DSA.

As you can see, a digital signature contains both cryptographic hashing and asymmetric key cryptography. This complicated process is often handled by an application that has been configured to do so, instead of being directly handled by Linux system users. However, you can manually add your own digital signatures to documents.

Signing a file with a digital signature

Let's say that user `johndoe` is going to send a message to the user `christineb`, along with his digital signature. He has created a file containing the plain-text message to send. He uses the `gpg2` utility to create the signature file and encrypt the message file. The `--sign` option tells the `gpg2` utility that `MessageForChristine.txt` is the file to encrypt and use to create the digital signature. In response, the `gpg2` utility does the following:

- Creates a message digest (aka hash) of the message file
- Encrypts the message digest, which creates the digital signature
- Encrypts the message file
- Places the encrypted contents into the file specified by the `--output` option, `JohnDoe.DS`

The file `JohnDoe.DS` now contains an encrypted and digitally signed message. The following code demonstrates this process:

```
$ gpg2 --output JohnDoe.DS --sign MessageForJill.txt
```

After the user `jill` receives the signed and encrypted file, she can use the `gpg2` utility to check the digital signature and decrypt the file in one step. In the code that follows, the

`--decrypt` option is used along with the name of the digitally signed file, `JohnDoe.DS`. The file's message is decrypted and shown. The digital signature of the file is checked and found to be valid.

```
$ gpg2 --decrypt JohnDoe.DS
I am the real John Doe!
gpg: Signature made Sun 27 Oct 2019 07:03:21 PM EDT
gpg:                using RSA key
7469BCD3D05A43130F1786E0383D645D9798C173
gpg: Good signature from "John Doe <jdoe@example.com>" [unknown]
...
```

Without `johndoe`'s public key on her key ring, `jill` would not be able to decrypt this message and check the digital signature.

TIP

The previous example of digitally signing a document allows anyone with the public key the ability to decrypt the document. In order to keep it truly private, use the public key of the recipient to encrypt with the `gpg2` options: `--sign` and `--encrypt`. The recipient can decrypt with their private key.

Understanding a few cryptography basics will help you get started on securing your Linux system with encryption. Keep in mind that we've covered just the basics in this chapter. There are many more cryptography topics, such as digital certificates and public key infrastructure, that would be worth your time to learn.

Implementing Linux cryptography

Many cryptography tools are available on your Linux system. Which ones you choose to use depend upon your organization's security requirements. The following is a brief review of some of the Linux cryptography tools available.

Ensuring file integrity

Earlier in this chapter, an ISO's file integrity was checked using the message digest utility `sha256sum`.

Related message digest utilities include the following:

- `sha224sum`
- `sha256sum`
- `sha384sum`
- `sha512sum`

These tools work just like the `sha1sum` command, except, of course, they use the SHA-2 cryptographic hash standard. The only difference between the various SHA-2 tools is the key length they use. The `sha224sum` command uses a key length of 224 bits, the

`sha256sum` command uses a key length of 256 bits, and so on. Remember that the longer the key length, the less the chance of cracking the cipher.

The SHA-2 cryptographic hash standard was created by the National Security Agency (NSA). SHA-3 is another cryptographic hash standard, which was released by NIST in August 2015.

Encrypting a Linux filesystem at installation

You may need to encrypt an entire filesystem on your Linux server. This can be done in a number of different ways, including using a Free and Open Source Software (FOSS) third-party tool such as Linux Unified Key Setup (LUKS) (https://gitlab.com/cryptsetup/cryptsetup).

One of your options in Linux is to encrypt your root partition upon installation (see Chapter 9, "Installing Linux"). Many Linux distributions include an encryption option during their installation process. Figure 23.3 shows the encryption option during a Red Hat Enterprise Linux installation.

After you select this option during installation, you are asked for a password. This is symmetric key cryptography with a password protecting the single key. Figure 23.4 shows the installation asking for the key's password. The password must be at least eight characters long.

FIGURE 23.3

Red Hat Enterprise Linux installation encryption option

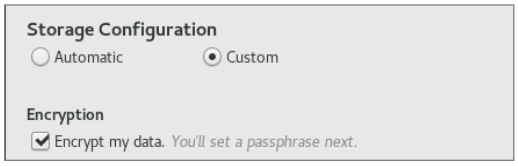


FIGURE 23.4

Linux Fedora encryption symmetric key password

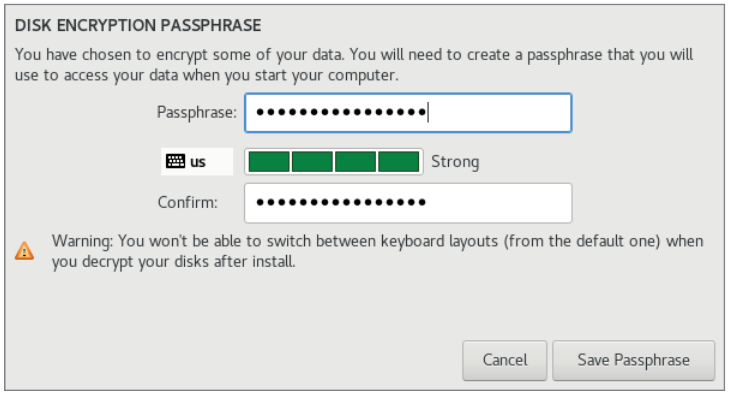
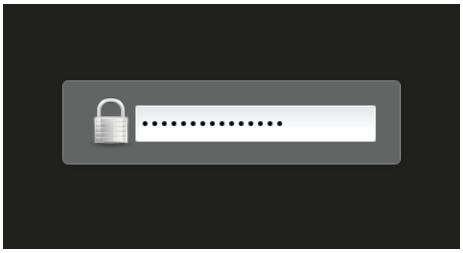


FIGURE 23.5

Asking for the encryption symmetric key password at boot



If you select this encryption option, whenever you boot the system, you are asked for the symmetric key password. Figure 23.5 shows what this looks like. This protects the root partition, should the disk it resides on be stolen.

If you inherit a system with an encrypted disk, using root privileges, you can use the `lvs` and `cryptsetup` commands and the `/etc/crypttab` file to help. In the following example, the `lvs` command shows all of the logical volumes currently on the system and their underlying device names. See Chapter 12, "Managing Disks and Filesystems," for a review of different LVM commands.

```
# lvs -o devices
  Devices
  /dev/mapper/luks-b099fbbe-0e56-425f-91a6-44f129db9f4b(56)
  /dev/mapper/luks-b099fbbe-0e56-425f-91a6-44f129db9f4b(0)
```

On this system, notice that the underlying device names start with `luks`. This indicates that the Linux Unified Key Setup (LUKS) standard for hard disk encryption has been used.

NOTE

Ubuntu does not have the `lvs` command installed by default. To install it, type **`sudo apt-get install lvm2`** at the command line.

The encrypted logical volumes are mounted at boot time using the information from the `/etc/fstab` file. However, contents of the `/etc/crypttab` file, which are used to trigger the capture of the password at boot time, will decrypt the `/etc/fstab` entries as they are mounted. This is shown in the following code. Notice that the `luks` names are the same as those listed by the `lvs` command in the previous example.

```
# cat /etc/crypttab
luks-b099fbbe-0e56-425f-91a6-44f129db9f4b
     UUID=b099fbbe-0e56-425f-91a6-44f129db9f4b none
```

You can also use the `cryptsetup` command to help you uncover more information about your Linux system's encrypted volumes. In the example that follows, the `status` option is used along with the `luks` device name to determine further information.

```
# cryptsetup status luks-b099fbbe-0e56-425f-91a6-44f129db9f4b
/dev/mapper/luks-b099fbbe-0e56-425f-91a6-44f129db9f4b
 is active and is in use.
  type:    LUKS1
  cipher:  aes-xts-plain64
  keysize: 512 bits
  device:  /dev/sda3
  offset:  4096 sectors
  size:    493819904 sectors
  mode:    read/write
```

Encrypting a Linux directory

You can also use the `ecryptfs` utility to encrypt on a Linux system. The `ecryptfs` utility is not a filesystem type, as the name would imply. Instead, it is a POSIX-compliant utility that allows you to create an encryption layer on top of any filesystem.

The `ecryptfs` utility is not installed by default on Fedora and not available in RHEL. To install that utility in Fedora, you use the command `dnf install ecryptfs-utils`. If it is not installed on a Debian system, use the command `sudo apt-get install ecrypt-utils`.

Tip

Because the `ecryptfs` utility is used for encryption, it is a common mistake to put the letter `n` after the letter `e` in the syntax `ecryptfs`. If you get an error while using the `ecryptfs` utilities, make sure that you did not use the syntax `encryptfs` by mistake.

In the example that follows, the user `johndoe` will have a subdirectory encrypted using the `ecryptfs` utility. First, there should be no files currently residing in the directory before it is encrypted. If there are files located there, move them to a safe place until after the encryption has been completed. If you do not move them, you cannot access them while the directory is encrypted.

Now, to encrypt the directory `/home/johndoe/Secret`, use the `mount` command. You must have root privileges to mount and unmount the encrypted directory in this method. Look at the `mount` command used in the example that follows. It is somewhat similar to the regular `mount` command, except that the partition type used is `ecryptfs`. The item to mount and its mount point are the same directory! You are literally encrypting the directory and mounting it upon itself. The other unusual item about this `mount` command is that it kicks off the `ecryptfs` utility, which asks a few interactive questions.

```
# mount -t ecryptfs /home/johndoe/Secret /home/johndoe/Secret
Select key type to use for newly created files:
 1) tspi
 2) passphrase
 3) pkcs11-helper
 4) openssl
```

```
Selection: 2
Passphrase: **********
Select cipher:
 1) aes: blocksize = 16;
 min keysize = 16; max keysize = 32 (loaded)
 2) blowfish: blocksize = 16;
 min keysize = 16; max keysize = 56 (not loaded)
 3) des3_ede: blocksize = 8;
 min keysize = 24; max keysize = 24 (not loaded)
 4) twofish: blocksize = 16;
 min keysize = 16; max keysize = 32 (not loaded)
 5) cast6: blocksize = 16;
 min keysize = 16; max keysize = 32 (not loaded)
 6) cast5: blocksize = 8;
 min keysize = 5; max keysize = 16 (not loaded)
Selection [aes]: 1
Select key bytes:
 1) 16
 2) 32
 3) 24
Selection [16]: 16
Enable plaintext passthrough (y/n) [n]: n
Enable filename encryption (y/n) [n]: n
Attempting to mount with the following options:
  ecryptfs_unlink_sigs
  ecryptfs_key_bytes=16
  ecryptfs_cipher=aes
  ecryptfs_sig=70993b8d49610e67
WARNING: Based on the contents of [/root/.ecryptfs/sig-cache.txt]
it looks like you have never mounted with this key
before. This could mean that you have typed your
passphrase wrong.

Would you like to proceed with the mount (yes/no)? : yes
Would you like to append sig [70993b8d49610e67] to
[/root/.ecryptfs/sig-cache.txt]
in order to avoid this warning in the future (yes/no)? : yes
Successfully appended new sig to user sig cache file
Mounted eCryptfs
```

The `ecryptfs` utility allows you to choose the following:

- Key type
- Passphrase
- Cipher
- Key size (in bytes)
- To enable or disable plain text to pass through
- To enable or disable filename encryption

It also warns you when you are first mounting this encrypted directory because the key has not been used before. The utility allows you to apply a digital signature to the mounted directory so that if you `mount` it again, it just mounts the directory and does not require a passphrase.

Tip
Write down the selections you make when you mount an `ecryptfs` folder for the first time. You need the exact selections you chose the next time you remount the folder.

To verify that the encrypted directory is now mounted, you can use the `mount` command again. In the example that follows, the `mount` command is used and then piped into `grep` to search for the `/home/johndoe/Secret` directory. As you can see, the directory is mounted with an `ecryptfs` type.

```
# mount | grep /home/johndoe/Secret

/home/johndoe/Secret on /home/johndoe/Secret type ecryptfs
(rw,relatime,ecryptfs_sig=70993b8d49610e67,ecryptfs_cipher=aes,
ecryptfs_key_bytes=16,ecryptfs_unlink_sigs)
```

So far, you have not seen the effects of this mounted and encrypted directory. In the text that follows, the file `my_secret_file` is copied to the encrypted directory. User `johndoe` can still use the `cat` command to display the file in plain text. The file is automatically decrypted by the `ecryptfs` layer.

```
$ cp my_secret_file Secret
$ cat /home/johndoe/Secret/my_secret_file
Shh... It's a secret.
```

The root user also can use the `cat` command to display the file in plain text.

```
# cat /home/johndoe/Secret/my_secret_file
Shh... It's a secret.
```

However, after the encrypted directory is unmounted using the `umount` command, the files are no longer automatically decrypted. The file `my_secret_file` is now gibberish and cannot be read, even by the root user.

```
# umount /home/johndoe/Secret
```

Thus, the `ecryptfs` utility allows you to create a location on the filesystem to encrypt and decrypt files quickly. However, after that directory is no longer mounted as an `ecryptfs` type, the files are secure and cannot be decrypted.

Tip
As a non-root user, you could use the `ecryptfs-setup-private` and `ecryptfs-mount-private` commands to configure a private cryptographic mountpoint as a non-root user.

Encrypting a Linux file

The most popular tool for file encryption on a Linux system is the OpenPGP utility GNU Privacy Guard, `gpg`. Its flexibility and variety of options, along with the fact that it is installed by default on most Linux distributions, add to its appeal.

CAUTION

If your organization uses a third-party cloud storage company, you need to know that some of these companies, such as Dropbox, do not encrypt the files until they are received. This means that the company has the keys required to decrypt your files and can leave your organization's data vulnerable. Encrypting files on your Linux system before they are sent to the cloud adds the extra layer of protection needed.

However, you can use several other cryptography tools on a Linux system to encrypt files. Just like `gpg`, many of these tools allow you to do much more than merely file encryption. The following are some of the popular Linux cryptography tools that you can use to encrypt files:

`aescrypt`: It uses the symmetric key cipher Rijndael, also called AES. This third-party FOSS tool is available for download from www.aescriypt.com.

`bcrypt`: This tool uses the symmetric key cipher *blowfish*. It is not installed by default. After `bcrypt` is installed, man pages are available.

- **For Fedora (not available in RHEL)**: `yum install bcrypt`
- **For Ubuntu**: `sudo apt-get install bcrypt`

`ccrypt`: This tool uses the symmetric key cipher Rijndael, also called AES. It was created to replace the standard Unix `crypt` utility and is not installed by default. After `ccrypt` is installed, man pages are available.

- **For Fedora (not available in RHEL)**: `yum install ccrypt`
- **For Ubuntu**: `sudo apt-get install ccrypt`

`gpg`: This utility can use either asymmetric key pairs or a symmetric key. It is installed by default, and it is the cryptography tool of choice for Linux servers. The default cipher to use is set in the `gpg.conf` file. There are man pages available as well as `info gnupg`.

Keep in mind that this list covers only the more popular tools. Also, remember that many of these tools can be used for more than just file cryptography.

Encrypting Linux with miscellaneous tools

You can apply *cryptography*, defined as the act of writing or generating codes meant to keep secrets, to just about everything in Linux. Besides filesystems, directories, and files, you can also encrypt backups, Zip files, network connections, and more.

TABLE 23.2 Linux Miscellaneous Cryptography Tools

Tool	Description
Duplicity	Encrypts backups. To install on Fedora, type **`yum install duplicity`**. To install on Ubuntu, type **`sudo apt-get install duplicity`** at the command line.
gpg-zip	Uses GNU Privacy Guard to encrypt or sign files into an archive. Installed by default.
Openssl	A toolkit that implements Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. These protocols require encryption. Installed by default.
Seahorse	A GNU Privacy Guard encryption key manager. Installed by default on Ubuntu. To install on Fedora and RHEL, type **`yum install seahorse`** at the command line.
Ssh	Encrypts remote access across a network. Installed by default.
Zipcloak	Encrypts entries in a Zip file. Installed by default.

Table 23.2 lists some of the miscellaneous Linux cryptography tools and what they do. If you want to see a full list of installed cryptography tools on your current Linux distribution, type **`man -k crypt`** at the command line.

Like many other items on a Linux system, the available cryptography tools are rich and plentiful. This gives you the flexibility and variety that you need in order to implement the cryptography standards your particular organization requires.

Using Encryption from the Desktop

The Passwords and Keys window provides a means of viewing and managing keys and passwords from the GNOME desktop. This window can be launched by selecting the Passwords and Keys icon from the Activities screen or by running the `seahorse` command. With the window that appears, you can work with the following:

Passwords: When you access a website, from a Chromium or Chrome web browser, and enter a username and password (and you select to save that password), it is stored on your system for the next time that you visit that site. Select the Login entry under the Passwords heading to see each of these saved usernames and passwords.

Certificates: You can view certificates associated with the Gnome2 Key Storage, User Key Storage, System Trust, and Default Trust.

PGP keys: You can view the GPG keys that you create by selecting the GnuPG keys entry.

Secure Shell: You can create public and private OpenSSH keys that let you log in to remote systems using those keys instead of passwords for authentication with `ssh`, `scp`, `rsync`, `sftp`, and related commands. Select OpenSSH keys to view any keys that you have created for this purpose. (See the section "Using key-based passwordless authentication" in Chapter 13 for information on creating these types of keys.)

Another extremely powerful security tool available on Linux is PAM. The next sections in this chapter cover basic PAM concepts and how you can use this tool to enhance even further your Linux system's security.

Implementing Linux Security with PAM

Pluggable Authentication Modules (PAM) was invented by Sun Microsystems and originally implemented in the Solaris operating system. The Linux-PAM project began in 1997. Today, most Linux distributions use PAM.

PAM simplifies the authentication management process. Remember that authentication (see Chapter 22, "Understanding Basic Linux Security") is the process of determining that a subject (aka user or process) is who they say they are. This process is sometimes called "identification and authentication." PAM is a centralized method of providing authentication for the Linux system and applications.

Applications can be written to use PAM; such applications are called "PAM-aware." A PAM-aware application does not have to be rewritten and recompiled to have its authentication settings changed. Any required changes are made within a PAM configuration file for the PAM-aware applications. Thus, authentication management for these applications is centralized and simplified.

You can see whether a particular Linux application or utility is PAM-aware. Check whether it is compiled with the PAM library, `libpam.so`. In the example that follows, the `crontab` application is being checked for PAM awareness. The `ldd` command checks a file's shared library dependencies. To keep it simple, `grep` is used to search for the PAM library. As you can see, `crontab` on this particular Linux system is PAM-aware.

```
# ldd /usr/bin/crontab | grep pam
libpam.so.0 => /lib64/libpam.so.0 (0x00007fbee19ce000)
```

The benefits of using PAM on your Linux system include the following:

- Simplified and centralized authentication management from the administrator viewpoint
- Simplified application development, because developers can write applications using the documented PAM library instead of writing their own authentication routines
- Flexibility in authentication:
  - Allow or deny access to resources based on traditional criteria, such as identification
  - Allow or deny access based on additional criteria, such as time-of-day restrictions
  - Set subject limitations, such as resource usage

Although the benefits of PAM simplify authentication management, the way that PAM actually works is not so simple.

Understanding the PAM authentication process

When a subject (user or process) requests access to a PAM-aware application or utility, two primary components are used to complete the subject authentication process:

- The PAM-aware application's configuration file
- The PAM modules the configuration file uses

Each PAM-aware application configuration file is at the center of the process. The PAM configuration files call upon particular PAM modules to perform the needed authentication. PAM modules authenticate subjects from system authorization data, such as a centralized user account using LDAP (see Chapter 11, "Managing User Accounts").

Linux comes with many applications that are PAM-aware, with their needed configuration files and PAM modules already installed. If you have any special authentication needs, you can most likely find a PAM module that has already been written for that need. However, before you start tweaking PAM, you need to understand more about how PAM operates.

A series of steps is taken by PAM using the modules and configuration files in order to ensure that proper application authentication occurs:

1. A subject (user or process) requests access to an application.
2. The application's PAM configuration file, which contains an access policy, is open and read.

   The access policy is set via a list of all the PAM modules to be used in the authentication process. This PAM module(s) list is called a *stack*.
3. The PAM modules in the stack are invoked in the order in which they are listed.
4. Each PAM module returns either a success or failure status.
5. The stack continues to be read in order, and it is not necessarily stopped by a single returned failure status.
6. The status results of all of the PAM modules are combined into a single overall result of authentication success or failure.

Typically, if a single PAM module returns a failure status, access to the application is denied. However, this is dependent upon the configuration file settings. Most PAM configuration files are located in `/etc/pam.d`. The general format of a PAM configuration file is

```
context   control flag   PAM module [module options]
```

Understanding PAM contexts

PAM modules have standard functions that provide different authentication services. These standard functions within a PAM module can be divided into function types called *contexts*. Contexts can also be called *module interfaces* or *types*. In Table 23.3, the different PAM contexts are listed along with what type of authentication service they provide.

TABLE 23.3 PAM Contexts

Context	Service Description
`auth`	Provides authentication management services, such as verifying account passwords
`account`	Provides account validation services, such as time-of-day access restrictions
`password`	Manages account passwords, such as password length restrictions

Understanding PAM control flags

In a PAM configuration file, control flags are used to determine the overall status, which are returned to the application. A control flag is either of the following:

Simple keyword: The only concern here is if the corresponding PAM module returns a response of either "failed" or "success." See Table 23.4 for how these statuses are handled.

TABLE 23.4 PAM Configuration Control Flags and Response Handling

`required`	If failed, returns a failure status to the application, after the rest of the contexts have been run in the stack. For example, a requisite control might cause a login to fail if someone types in an invalid user. But the user might not be told of the failure until after entering a password, hiding the fact that it was the bad username that caused the failure.
`requisite`	If failed, returns a failure status to the application immediately without running the rest of the stack. (Be careful where you place this control in the stack.) For example, a requisite control might require key-based authentication and fail immediately when a valid key is not provided. In that case, it could fail before even prompting for a username/password.
`sufficient`	If failed, the module status is ignored. If successful, then a success status is immediately returned to the application without running the rest of the stack. (Be careful where you place this control in the stack.)
`optional`	This control flag is important only for the final overall return status of success or failure. Think of it as a tiebreaker. When the other modules in the configuration file stack return statuses that are neither clear-cut failure nor success statuses, this optional module's status is used to determine the final status or break the tie. In cases where the other modules in the stack are returning a clear-cut path of failure or success, this status is ignored.
`include`	Get all the return statuses from this particular PAM configuration file's stack to include in this stack's overall return status. It's as if the entire stack from the named configuration file is now in this configuration file.
`substack`	Similar to the include control flag, except for how certain errors and evaluations affect the main stack. This forces the included configuration file stack to act as a substack to the main stack. Thus, certain errors and evaluations affect only the substack and not the main stack.

Series of actions: The returned module status is handled through the series of actions listed in the file.

Table 23.4 shows the various keyword control flags and their responses to the returned module status. Notice that a few of the control flags need to be carefully placed within the configuration file's stack. Some control flags cause the authentication process to stop immediately, and the rest of the PAM modules are not called. The control flags simply control how the PAM module status results are combined into a single overall result. Table 23.4 demonstrates how the status results are combined.

You should know that the PAM modules return many more status result codes than just "success" or "failure." For example, a module may return the status code of `PAM_ACCT_EXPIRED`, which means that the user account has expired. This would be deemed a "failure."

Understanding PAM modules

A PAM module is actually a suite of shared library modules (DLL files) stored in `/usr/lib64/security` (64-bit). You can see a list of the various installed PAM modules on your system by entering **`ls /usr/lib64/security/pam*.so`** at the command line.

NOTE
On Ubuntu, to find your PAM modules, type the command **`sudo find / -name pam*.so`** at the command line.

Your Linux system comes with many of the PAM modules needed already installed. If you do need a module not already installed, most likely someone else has already written it. Check out sources such as these:

- `http://www.openwall.com/pam/`
- http://puszcza.gnu.org.ua/software/pam-modules/download.html
- Understanding PAM system event configuration files

So far, the focus has been on PAM-aware applications and their configuration files. However, other system events, such as logging into the Linux system, also use PAM. Thus, these events also have configuration files.

The following is a partial directory listing of the PAM configuration file directory. Notice that there are PAM-aware application configuration files, such as `crond`, and system event configuration files, such as `postlogin-ac`.

```
# ls -l /etc/pam.d
total 204
-rw-r--r--. 1 root root  272 Nov 15 10:06 atd
...
-rw-r--r--. 1 root root  232 Jan 31 12:35 config-util
-rw-r--r--. 1 root root  293 Oct 26 23:10 crond
```

```
...
-rw-r--r--. 1 root root  109 Feb 28 01:33 postlogin

...
-rw-r--r--. 1 root root  981 Feb 28 01:33 system-auth
...
```

You can modify these system event configuration files to implement your organization's specific security needs. For example, the `system-auth` file can be modified to force certain password restrictions.

CAUTION

Modifying or deleting PAM system event configuration files incorrectly can lock you out of your own system. Make sure that you test any changes in a virtual or test environment before modifying your production Linux servers.

These PAM system event configuration files operate in exactly the same way as the PAM-aware application configuration files. They have the same format, use the same syntax, and call upon PAM modules. However, many of these files are symbolically linked (see Chapter 4, "Moving Around the Filesystem"). Therefore, these configuration files require a few extra steps when changes are made to them. The "how-tos" are covered later in this chapter.

TIP

Many of the PAM configuration files have a man page associated with them. For example, to find out more information on the `pam_unix` module, type **`man pam_unix`** at the command line of your Fedora and RHEL distribution. There are also module documentation files in the `/usr/share/doc/pam-*/txts/` directory.

Even though Linux comes with many PAM-aware applications, various configuration files, and PAM modules already installed, you cannot just hope that PAM will take care of itself. Certain administrative steps are needed to manage PAM.

Administering PAM on your Linux system

The task of administering PAM on your Linux system is rather minimal. You need to verify that PAM is properly implemented and make adjustments to meet your particular organization's security needs.

Also, PAM does a little more than just the application authentication steps described previously. PAM can also limit resources, restrict access times, enforce good password selection, and so on.

Managing PAM-aware application configuration files

You should review PAM configuration files for your PAM-aware applications and utilities in order to ensure that their authentication process matches your organization's desired authentication process. Your Access Control Matrix (see Chapter 22, "Understanding Basic

Linux Security") and the information on understanding PAM provided in this chapter should help you conduct an audit of the PAM configuration files.

Each PAM-aware application should have its very own PAM configuration file. Each configuration file defines what particular PAM modules are used for that application. If no configuration file exists, a security hole may be created for that application. This hole could be used for malicious intent. As a safety precaution, PAM comes with the "other" configuration file. If a PAM-aware application does not have a PAM configuration file, it defaults to using the "other" PAM configuration file.

You can verify whether your Linux system has the `/etc/pam.d/other` configuration file by using the `ls` command. The example that follows shows that the `/etc/pam.d/other` PAM configuration file does exist on this system.

```
$ ls /etc/pam.d/other
/etc/pam.d/other
```

The PAM `/etc/pam.d/other` configuration file should deny all access, which in terms of security is referred to as Implicit Deny. In computer security access control, *Implicit Deny* means that if certain criteria are not clearly met, access must be denied. In this case, if no configuration file exists for a PAM-aware application, all access to it is denied. The following shows an `/etc/pam.d/other` file's contents:

```
$ cat /etc/pam.d/other
#%PAM-1.0
auth     required       pam_deny.so
account  required       pam_deny.so
password required       pam_deny.so
session  required       pam_deny.so
```

Notice that all four PAM contexts—`auth`, `account`, `password`, and `session`—are listed. Each context uses the `required` control flag and the `pam_deny.so` module. The `pam_deny.so` PAM module is used to deny access.

Even with the "other" configuration file in place, if a PAM configuration file for a PAM-aware application is not there, it must be created. Add this item to your PAM audit checklist. You should also review your PAM "other" configuration file on your Linux system to ensure that it enforces Implicit Deny.

Managing PAM system event configuration files

Similar to PAM-aware application and utility configuration files, your PAM system event configuration files need to be audited with your organization's Access Control Matrix. However, for any needed modifications to these files, there are extra steps that must be taken.

In the material that follows, you will learn how to set up special security requirements via PAM on your Linux system, such as account login time restrictions. Many of the special requirements require you to make a change to PAM system event configuration files, such as `/etc/pam.d/system-auth`.

The problem with making changes to some of these PAM system event configuration files is that the utility `authselect` can rewrite these files and remove any locally made changes. Fortunately, each PAM configuration file that runs this risk has it documented in a comment line within. Using `grep`, you can quickly find which PAM configuration files have this potential problem.

```
# grep "authselect" /etc/pam.d/*
fingerprint-auth:# Generated by authselect on Mon Oct 21 19:24:36
2019
password-auth:# Generated by authselect on Mon Oct 21 19:24:36 2019
postlogin:# Generated by authselect on Mon Oct 21 19:24:36 2019
smartcard-auth:# Generated by authselect on Mon Oct 21 19:24:36 2019
system-auth:# Generated by authselect on Mon Oct 21 19:24:36 2019
```

These PAM system event configuration files use symbolic links (see Chapter 4, "Moving Around the Filesystem"). For example, in Fedora you can see that the file `system-auth` is actually a symbolic link pointing to the file `/etc/authselect/system-auth`. The first character in the file's security is an `l`. This indicates that the file is linked. The `->` symbol shows that the file is symbolically linked.

```
# ls -l system-auth
lrwxrwxrwx. 1 root root 27 Oct 1 15:24 system-auth -> /etc/
authselect/system-auth
```

Note

On some Linux distributions, the utility `pam-auth-config` is similar to the `authselect` utility in its ability to overwrite configuration files. This can happen if the command `pam-auth-config --force` is entered at the command line. Read the `man pam-auth-config` man page to learn more about this utility if it is installed on your system.

Implementing resources limits with PAM

Managing resources is not just a system administrative task. It is also a security administrative task. Setting resource limitations helps you avoid many adverse problems on your Linux system. Problems such as fork bombs can be averted by limiting the number of processes a single user can create. A *fork bomb* occurs when a process spawns one process after another in a recursive manner until system resources are consumed. Fork bombs can be malicious or just accidental; that is, created simply by poor program code development.

The PAM module `pam-limits` uses a special configuration file to set these resources limits: `/etc/security/limits.conf`. By default, this file has no resource limits set within it. Therefore, you need to review the file and set resources limits to match your organization's security needs.

NOTE
PAM configuration files are in the `/etc/pam.d` directory and the `/etc/security` directory.

The following snippet shows the `/etc/security/limits.conf` file. The file is well documented. You should read through the contents of that file for a thorough format description and examples of limits that can be set.

```
$ cat /etc/security/limits.conf
# /etc/security/limits.conf
#
#This file sets the resource limits for the users logged in via PAM.
#It does not affect resource limits of the system services.
#
#Also note that configuration files in /etc/security/limits.d
directory,
#which are read in alphabetical order, override the settings in this
#file in case the domain is the same or more specific.
...
#Each line describes a limit for a user in the form:
#
#<domain>        <type>  <item>  <value>
...
#*               soft    core            0
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#@student        -       maxlogins       4
# End of file
```

The format items *domain* and *type* need some further explanation than what is documented in the configuration file:

`domain`: The limit applies to the listed user or group. If the domain is `*`, it applies to *all* users.

`type`: A `hard` limit cannot be exceeded. A `soft` limit can be exceeded, but only temporarily.

Look at the `limits.conf` file setting example that follows. The group `faculty` is listed, but what you should notice is `nproc`. The `nproc` limit sets the maximum number of processes a user can start. This setting is what prevents a fork bomb. Notice that the `type` selected is `hard`; thus, the limit of 50 processes cannot be exceeded. Of course, this limit is not enforced because the line is commented out with a # symbol.

```
#@faculty        hard    nproc           50
```

Limit settings are set per login and only last for the duration of the login session. A malicious user could log in several times to create a fork bomb. Thus, setting the maximum number of logins for these user accounts is a good idea too.

Limiting the maximum number of logins may have to be done on a per-user basis. For example, `johndoe` needs to log in to the Linux system only once. To prevent others from using `johndoe`'s account, set his account's `maxlogins` to 1.

```
johndoe         hard    maxlogins           1
```

To override any settings in the `limits.conf` file, add files named `*.conf` to the `/etc/security/limits.d` directory. This is a convenient way to have an RPM file or other method to add and remove limits without needing to edit the `limits.conf` file directly.

The final step in limiting this resource is to ensure that the PAM module using `limits.conf` is included in one of the PAM system event configuration files. The PAM module using `limits.conf` is `pam_limits`. In the partial listing that follows, `grep` is used to verify that the PAM module is used within the system event configuration files.

```
# grep "pam_limits" /etc/pam.d/*
/etc/pam.d/fingerprint-auth:session required    pam_limits.so
/etc/pam.d/password-auth:session    required    pam_limits.so
/etc/pam.d/runuser:session          required    pam_limits.so
/etc/pam.d/system-auth:session      required    pam_limits.so
```

Time limits for access to services and accounts are not handled by the PAM `/etc/security/limits.conf` configuration file. Instead, it is handled by the `time.conf` file.

Implementing time restrictions with PAM

PAM can make your entire Linux system operate on "PAM time." Time restrictions such as access to particular applications during certain times of the day, or allowing logins only during specified days of the week, are all handled by PAM.

The PAM configuration file that handles these restrictions is located in the `/etc/security` directory. The following code shows the well-documented `/etc/security/time.conf` PAM configuration file.

```
$ cat /etc/security/time.conf
# this is an example configuration file for the pam_time module. Its
syntax
# was initially based heavily on that of the shadow package
(shadow-960129).

#
# the syntax of the lines is as follows:
#
#       services;ttys;users;times
...
```

I recommend that you read through the contents of the `time.conf` file. Note that the format for each valid entry follows this syntax: `services;ttys;users;times`. Fields are separated by semicolons. The valid field values are documented in the `time.conf` configuration file.

While `time.conf` is well-documented, an example is always helpful. For instance, you have decided that regular users should be allowed to log in on terminals on weekdays only (Monday through Friday). They can log in from 7 a.m. to 7 p.m. on these weekdays. The following list describes what elements need to be set:

`services`: Login

`ttys–*`: Indicates that all terminals are to be included

`users`: Everyone but root (`!root`)

`times`: Allowed on weekdays (`Wd`) from 7 a.m. (`0700`) to 7 p.m. (`1900`)

The entry in `time.conf` would look like the following:

```
login; * ; !root ; Wd0700-1900
```

The final step in implementing this example time restriction is to ensure that the PAM module using `time.conf` is included in one of the PAM system event configuration files. The PAM module using `time.conf` is `pam_time`. In the partial listing that follows, `grep` shows the PAM module; `pam_time` is not used within any of the system event configuration files.

```
# grep "pam_time" /etc/pam.d/*
config-util:auth              sufficient   pam_timestamp.so
config-util:session           optional     pam_timestamp.so
```

Because `pam_time` is not listed, you must modify the `/etc/pam.d/system-auth` file in order for PAM to enforce the time restrictions. The PAM configuration file `system-auth` is used by PAM at system login and during password modifications. This configuration file checks many items, such as time restrictions.

Add the following near the top of the "account" section of the configuration file. Now the `pam_time` module checks the login restrictions you set within the `/etc/security/time.conf` file.

```
account    required    pam_time.so
```

NOTE
On Ubuntu, you need to modify the `/etc/pam.d/common-auth` file instead of the `system-auth` configuration file.

Remember that `system-auth` is a symbolically linked file. If you modify this file, you must take extra steps to preserve the modifications from the `authselect` utility. You can employ additional PAM modules and configuration files to set even more restrictions on subjects. One important security module is `pam_cracklib`.

Enforcing good passwords with PAM

When a password is modified, the PAM module `pam_cracklib` is involved in the process. The module prompts the user for a password and checks its strength against a system dictionary and a set of rules for identifying poor choices.

NOTE

The `pam_cracklib` module is installed by default on Fedora and RHEL. For Ubuntu Linux systems, it is not installed by default. Therefore, to get access to the `pam_cracklib` module on Ubuntu, issue the command `sudo apt-get install libpam-cracklib`.

Using `pam_cracklib`, you can check a newly chosen password for the following:

- Is it a dictionary word?
- Is it a palindrome?
- Is it the old password with the case changed?
- Is it too much like the old password?
- Is it too short?
- Is it a rotated version of the old password?
- Does it use the same consecutive characters?
- Does it contain the username in some form?

You can change the rules `pam_cracklib` uses for checking new passwords by making modifications to the `/etc/pam.d/system-auth` file. You may think that the changes should be made in the PAM-aware `passwd` configuration file. However, `/etc/pam.d/passwd` includes the `system-auth` file in its stack.

```
# cat /etc/pam.d/passwd
#%PAM-1.0
# This tool only uses the password stack.
password   substack   system-auth
-password   optional        pam_gnome_keyring.so use_authtok
password   substack   postlogin
```

NOTE

On Ubuntu, you need to modify the `/etc/pam.d/common-password` file instead of the `system-auth` configuration file.

The current settings of the `system-auth` file are shown here. Currently, one entry calls the `pam_cracklib` PAM module.

```
# cat /etc/pam.d/system-auth
#%PAM-1.0
# Generated by authselect on Mon Oct 21 19:24:36 2019
```

```
# Do not modify this file manually.
auth        required      pam_env.so
auth        required      pam_faildelay.so delay=2000000
auth        sufficient    pam_fprintd.so
...
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so
auth        sufficient    pam_sss.so forward_pass
auth        required      pam_deny.so
account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 1000 quiet
account     [default=bad success=ok user_unknown=ignore] pam_sss.so
account     required      pam_permit.so
password    requisite     pam_cracklib.so try_first_pass retry=3
...
```

The `pam_cracklib` entry in the preceding listing uses the keyword `retry`. The following keywords are available for `cracklib`:

debug

Causes module to write information to syslog.

authtok_type=XXX

- Defaults to using `New UNIX password:` and `Retype UNIX password:` to request passwords.
- Replace `XXX` with a word to use instead of UNIX.

retry=*N*

- Default = 1
- Prompt user at most *N* times before returning with an error.

difok=*N*

- Default = 5
- The number of characters in the new password that must not be present in the old password.
- *Exception 1*: If half of the characters in the new password are different, then the new password is accepted.
- *Exception 2*: See `difignore`.

difignore=*N*

- Default = 23
- The number of characters that the password has before the `difok` setting is ignored.

minlen=*N*

- Default = 9
- The minimum acceptable size for the new password.
- See `dcredit`, `ucredit`, `lcredit`, and `ocredit` for how their settings affect `minlen`.

dcredit=*N*

- Default =1
- If (*N* >= 0): The maximum credit for having digits in the new password. If you have fewer than or *N* digits, each digit counts +1 toward meeting the current `minlen` value.
- If (*N* < 0): The minimum number of digits that must be met for a new password.

ucredit=*N*

- Default = 1
- If (*N* >= 0): The maximum credit for having uppercase letters in the new password. If you have fewer than or *N* uppercase letters, each letter counts +1 toward meeting the current `minlen` value.
- If (*N* < 0): The minimum number of uppercase letters that must be met for a new password.

lcredit=*N*

- Default = 1
- If (*N* >= 0): The maximum credit for having lowercase letters in the new password. If you have fewer than or *N* lowercase letters, each letter counts +1 toward meeting the current `minlen` value.
- If (*N* < 0): The minimum number of lowercase letters that must be met for a new password.

ocredit=*N*

- Default = 1
- If (*N* >= 0): The maximum credit for having other characters in the new password. If you have fewer than or *N* other characters, each character counts +1 toward meeting the current `minlen` value.
- If (*N* < 0): The minimum number of other characters that must be met for a new password.

minclass=*N*

- Default = 0
- *N* out of four character classes is required for the new password. The four classes are digits, uppercase letters, lowercase letters, and other characters.

maxrepeat=*N*

- Default = 0
- Reject passwords that contain more than *N* same consecutive characters.

reject_username

Check whether the name of the user in straight or reversed form is contained in the new password. If it is found, the new password is rejected.

try_first_pass

Try to get the password from a previous PAM module. If that does not work, prompt the user for the password.

use_authtok

This argument is used to *force* the module not to prompt the user for a new password. Instead, the new password is provided by the previously stacked *password* module.

dictpath=*/path*

Path to the `cracklib` dictionaries.

maxsequence=*N*

- Default = 0 (meaning this check is disabled)
- *N* set to any number other than 0 causes passwords with monotonic characters longer than that number to be rejected.

maxclassrepeat=*N*

- Default = 0 (meaning this check is disabled)
- *N* set to any number other than 0 causes passwords with consecutive characters in the same class that are longer than that number to be rejected.

gecoscheck=*N*

Causes passwords with more than three straight characters from the user's GECOS field, typically containing the user's real name, to be rejected.

enforce_for_root=*N*

- Default = 0 (meaning this check is disabled)
- *N* set to any number other than 0 causes passwords with consecutive characters in the same class that are longer than that number to be rejected.

enforce_for_root

Enforce failed password checks for the root user. This option is off by default.

For example, if your organization requires passwords to be 10 characters long and they must contain two digits, you would add a line similar to the following to the `/etc/pam.d/system-auth` file:

```
password required pam_cracklib.so minlen=10 dcredit=-2
```

The keywords used in this example with `pam_cracklib` are as follows:

- **`minlen=10`**: The new password must be at least 10 characters.
- **`dcredit=-2`**: The new password must contain two numbers.

NOTE
The `pam_cracklib` restrictions do not apply to the root user unless you apply the `enforce_for_root` option.

Encouraging sudo use with PAM

To allow tracking of root-account use by individuals and avoid a repudiation situation (see Chapter 22, "Understanding Basic Linux Security"), you should restrict the use of the `su` command and encourage the use of `sudo`. If your organization has such a policy, you can accomplish this with PAM in just a few steps.

The `su` command is PAM-aware, which greatly simplifies things. It uses the PAM module `pam_wheel` to check for users in the `wheel` group. The `/etc/pam.d/su` configuration file is shown here:

```
# cat /etc/pam.d/su
#%PAM-1.0
auth        required      pam_rootok.so
auth        sufficient    pam_rootok.so
# Uncomment the following line to implicitly trust users
# in the "wheel" group.
#auth        sufficient  pam_wheel.so trust use_uid
# Uncomment the following line to require a user to be
# in the "wheel" group.
#auth       required      pam_wheel.so use_uid
auth        substack      system-auth
auth        include       postlogin
account     sufficient    pam_succeed_if.so uid = 0 use_uid quiet
account     include       system-auth
password    include       system-auth
session     include       system-auth
session     include       postlogin
session     optional      pam_xauth.so
```

First, to restrict the use of `su`, if you are using the `wheel` group as your administrative group, you need to reassign your administrative group to a new group (see Chapter 11, "Managing User Accounts"). If you are not using the `wheel` group, just be sure not to assign anyone in the future to this group.

Next, you need to edit the `/etc/pam.d/su` configuration file. Remove the comment mark, #, from the following line:

```
#auth        required    pam_wheel.so use_uid
```

With these modifications, PAM disables the use of the `su` command. Administrative users now must use `sudo`, which the system tracks and provides a desired non-repudiation environment (see Chapter 22).

Obtaining more information on PAM

PAM is another rich and versatile security tool available to you on your Linux system. In your own Linux system's man pages, you can read about managing the PAM configuration files and about the modules in your `/usr/lib64/security` (64-bit) directory.

- To get more information on PAM configuration files, use the command **`man pam.conf`**.
- You can see all of the PAM modules available on your system by entering **`ls /usr/lib64/security/pam*.so`** at the command line. To get more information on each PAM module, enter **`man pam_`*`module_name`***. Be sure to leave off the file extension of `so` for the **`pam_`*`module_name`***. For example, enter **`man pam_lastlog`** to learn more about the `pam_lastlog.so` module. Several websites can provide additional information on PAM:
  - **The Official Linux-PAM website**: `http://linux-pam.org`
  - **The Linux-PAM System Administrator's Guide**: `http://linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html`
  - **PAM Module reference**: `http://linux-pam.org/Linux-PAM-html/sag-module-reference.html`

Summary

Cryptography tools offer ways of protecting and verifying the validity of the data you use on your Linux system. The PAM facility provides a means of creating policies to secure the tools that are used to authenticate users on your system.

Both the cryptography tools and PAM should be handled with care as you learn about Linux. Be sure to test any modifications that you make on a test Linux system or a virtualized Linux system before you implement them on a production machine.

The next chapter covers SELinux. While cryptography and PAM are tools that you can use on your Linux system, SELinux is an entire security enhancement layer.

Exercises

Use these exercises to test your knowledge of using cryptography tools and PAM. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Encrypt a file using the `gpg2` utility and a symmetric key.
2. Generate a public key ring using the `gpg2` utility.

3. List out the key ring you generated.
4. Encrypt a file and add your digital signature using the `gpg2` utility.
5. Go to the Fedora download page: `https://getfedora.org`. Select one of the Fedora distributions to download. When the download is complete, verify your image.
6. Using the command `which su`, determine the `su` command's full filename. Next, determine whether the `su` command on your Linux system is PAM-aware.
7. Does the `su` command have a PAM configuration file? If so, display the configuration file on the screen and list what PAM contexts it uses.
8. List out the various PAM modules on your system to your screen.
9. Find the PAM "other" configuration file on your system. Does it exist? Does it enforce Implicit Deny?
10. Find the PAM limits configuration file. Does it have a setting to keep a fork bomb from occurring on your system?

CHAPTER 24

Enhancing Linux Security with SELinux

IN THIS CHAPTER

Learning about SELinux benefits

Learning how SELinux works

Setting up SELinux

Fixing problems with SELinux

Getting additional information on SELinux

Security Enhanced Linux (SELinux) was developed by the National Security Agency (NSA) along with other security research organizations, such as the Secure Computing Corporation (SCC). SELinux was released to the open source community in 2000, and it became popular when Red Hat included SELinux in its Linux distributions. Now, SELinux is used by many organizations and is widely available.

Understanding SELinux Benefits

SELinux is a security enhancement module deployed on top of Linux. It provides additional security measures, is included by default, and is set to be in enforcing mode in Red Hat Enterprise Linux (RHEL) and Fedora.

SELinux provides improved security on the Linux system via role based access controls (RBACs) on subjects and objects (aka processes and resources). "Traditional" Linux security uses Discretionary Access Controls (DACs).

With DAC, a process can access any file, directory, device, or other resource that leaves itself open to access. With RBAC, a process only has access to resources that it is explicitly allowed to access, based on the assigned role. The way that SELinux implements RBAC is to assign an SELinux policy to a process. That policy restricts access as follows:

- Only letting the process access resources that carry explicit labels
- Making potentially insecure features, such as write access to a directory, available as Booleans, which can be turned on or off

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

A service, such as a web server, that includes an SELinux policy will often get installed with SELinux labels set on specific directories and files. This can make it so that the running server process can only read and write files from specific directories. If you want to change that, you need to add the correct SELinux labels on the files and directories that you want the process to access.

SELinux is not a replacement for DAC. Instead, it is an additional security layer.

- DAC rules are still used when using SELinux.
- DAC rules are checked first, and if access is allowed, then SELinux policies are checked.
- If DAC rules deny access, SELinux policies are not reviewed.

If a user tries to execute a file for which they do not have execute access to (`rw-`), the "traditional" Linux DAC controls deny access. Thus, the SELinux policies are not even checked.

NOTE

SELinux is the default security enhancement of Red Hat distributions, whereas AppArmor is the default security enhancement for Ubuntu. You can still install SELinux on Ubuntu by using the command `sudo apt-get install selinux` and then reboot. However, as of this writing, the Ubuntu Wiki page for SELinux suggests that you do not use Ubuntu's SELinux package (`https://wiki.ubuntu.com/SELinux`). If you want to learn more about AppArmor, go to `https://help.ubuntu.com/community/AppArmor`.

Even though "traditional" Linux security controls still work, there are several benefits to using SELinux. Following are a few of SELinux's benefits:

It implements the RBAC access control model. This is considered the strongest access control model.

It uses least privilege access for subjects (for example, users and processes). The term *least privilege* means that every subject is given a limited set of privileges that are only enough to allow the subject to be functional in its tasks. With least privilege implemented, a user or process is limited on the accidental (or on-purpose) damage to objects it can cause.

It allows process sandboxing. The term *process sandboxing* means that each process runs in its own area (sandbox). It cannot access other processes or their files unless special permissions are granted. These areas where processes run are called "domains."

It allows a test of its functionality before implementation. SELinux has a permissive mode, which allows you to see the effect of enforcing SELinux on your system. In permissive mode, SELinux still logs what it considers security breaches (called AVC denials), but it doesn't prevent them.

Another way to look at SELinux benefits is to examine what can happen if SELinux is not running on your Linux system. For example, getting back to the web server example,

your web server daemon (`httpd`) is listening on a port for something to happen. A simple request from a web browser comes in to view a home page. Going through its normal routine, the `httpd` daemon hears the request and only "traditional" Linux security is applied. Being unconstrained by SELinux, `httpd` can do these things:

- Access *any* file or directory, based on read/write/execute permissions for the associated owner and group.
- Perform potentially insecure activities, such as allowing a file upload or changing system limits.
- Listen on any port it likes for incoming requests.

On a system constrained by SELinux, the `httpd` daemon is much more tightly controlled. Using the preceding example, `httpd` can only listen on the port on which SELinux allows it to listen. SELinux prevents `httpd` from accessing any file that doesn't have the proper security context set and denies potentially insecure activities that are not explicitly enabled with Booleans in SELinux. In essence, SELinux severely limits what malicious code might gain access to and generally limits activity on your Linux system.

Understanding How SELinux Works

SELinux can be compared to a guard at a door: In this comparison, the subject (the user) wants to access the object (the file) inside the room. To gain access to this object:

1. The subject must present an ID badge to the guard.
2. The guard reviews the ID badge and access rules kept in a large manual.
    a. If the access rules allow this particular ID badge inside the door, the subject may enter the room to access the object.
    b. If the access rules do not allow this particular ID badge access to the object, then the guard refuses entry.

SELinux provides a combination of role based access control (RBAC) and either *type enforcement (TE)* or *Multi-Level Security (MLS)*. In role based access control, access to an object is based on a subject's assigned role in the organization. Therefore, it is not based on the subject's username or process ID. Each role is granted access rights.

Understanding Type Enforcement

Type enforcement (TE) is necessary to implement the RBAC model. Type enforcement secures a system through these methods:

- Labeling objects as certain security types
- Assigning subjects to particular domains and roles
- Providing rules allowing certain domains and roles to access certain object types

The example that follows uses the `ls -l` command to show the DAC controls on the file `my_stuff`. The output shows the file's owner (johndoe) and group (johndoe) as well as its assignments for permissions.

If you need a review of file permissions, see Chapter 4, "Moving Around the Filesystem."

```
$ ls -l my_stuff
-rw-rw-r--. 1 johndoe johndoe 0 Feb 12 06:57 my_stuff
```

The example that follows includes `ls -lZ` and the same file, `my _ stuff`, but instead of just the DAC controls, the `-Z` option displays the SELinux security RBAC controls too.

```
$ ls -lZ my_stuff
-rw-rw-r--. johndoe johndoe unconfined_u:object_r:user_home_t:s0
... my_stuff
```

The `ls -Z` example displays four items associated with the file that are specific to SELinux:

user (`unconfined_u`)

role (`object_r`)

type (`user_home_t`)

level (`s0`)

These four RBAC items (user, role, type, and level) are used in the SELinux access control to determine appropriate access levels. Together, the items are called the SELinux *security context*. A security context (ID badge) is sometimes called a security label.

These security context assignments are given to subjects (processes and users). Each security context has a specific name. The name given depends upon what object or subject it has been assigned: Files have a file context, users have a user context, and processes have a process context, also called a domain.

The rules allowing access are called allow rules or policy rules. A *policy rule* is the process SELinux follows to grant or deny access to a particular system security type. Returning to the comparison of SELinux with the guard, SELinux serves as the guard who must see the subject's security context (ID badge) and review the policy rules (access rules manual) before allowing or denying access to an object. Thus, type enforcement ensures that only certain "types" of subjects can access certain "types" of objects.

Understanding Multi-Level Security

With SELinux, the default policy type is called a *targeted policy*, which primarily controls how network services (such as web servers and file servers) can be accessed on a Linux system. The targeted policy places fewer restrictions on what valid user accounts can do on the system. For a more restricted policy, you can choose *Multi-Level Security (MLS)*. MLS uses type enforcement along with the additional feature of security clearances. It also offers Multi-Category Security, which gives classification levels to objects.

Tip

The Multi-Level Security (MLS) names can cause confusion. Multi-Category Security (MCS) is sometimes called Multi-Clearance Security. Because MLS offers MCS, it is sometimes called MLS/MCS.

Multi-Level Security enforces the Bell-LaPadula Mandatory Access security model. The Bell-LaPadula model was developed by the US government to impose information confidentiality. Enforcing this model is accomplished by granting object access based on a role's security clearance and an object's classification level.

Security clearance is an attribute granted to roles allowing access to classified objects. *Classification level* is an attribute granted to an object, providing protection from subjects who have a security clearance attribute that is too low. You most likely have heard of the classification level top secret. The fictional book and movie character James Bond had a top-secret security clearance, which granted him access to top-secret classified information. This is a classic example of the Bell-LaPadula model.

The combination of RBAC along with either type enforcement (TE) or Multi-Level Security (MLS) enables SELinux to provide such a strong security enhancement. SELinux also offers different operational modes for its use.

Implementing SELinux security models

The role-based access control model, type enforcement, Multi-Level Security, and Bell-LaPadula models are all interesting topics. SELinux implements these models through a combination of four primary SELinux pieces:

- Operational modes
- Security contexts
- Policy types
- Policy rule packages

Although we've touched upon some of these design elements, the following will give you an in-depth understanding of them. This understanding is needed before you can begin modifying the SELinux configuration on your system.

Understanding SELinux operational modes

SELinux comes with three operational modes: disabled, permissive, and enforcing. Each of these modes offers different benefits for Linux system security.

Using the disabled mode

In the *disabled mode*, SELinux is turned off. The default method of access control, Discretionary Access Control (DAC), is used instead. This mode is useful for circumstances in which enhanced security is not required.

If at all possible, Red Hat recommends setting SELinux to permissive mode rather than disabling it. However, there are occasions where disabling SELinux is appropriate.

If you are running applications that are working properly (from your perspective) but are generating massive amounts of SELinux AVC denial messages (even in permissive mode), you may end up filling up log files to the point of making your systems unusable. The better approach is to set the proper security context on the files your applications need to access. Nevertheless, disabling SELinux is the quicker fix.

Before you disable SELinux, however, think about whether you may ever want to enable it on that system again. If you decide to set it to enforcing or permissive at a later date, the next time you reboot your system, it will go through an automatic SELinux file relabel before it comes up.

Tip

If all you care about is turning SELinux off, you have found the answer. Just edit the configuration file `/etc/selinux/config` and change the text `SELINUX=` to `SELINUX=disabled`. SELinux will be disabled after a system reboot. You can now skip the rest of this chapter.

Using the permissive mode

In *permissive mode*, SELinux is turned on, but the security policy rules are not enforced. When a security policy rule should deny admission, access is still allowed. However, a message is sent to a log file denoting that access should have been denied.

SELinux permissive mode is used for the following:

- Auditing the current SELinux policy rules
- Testing new applications to see what effect SELinux policy rules will have on them
- Testing new SELinux policy rules to see what effect the new rules will have on current services and applications
- Troubleshooting why a particular service or application is no longer working properly under SELinux

In some cases, you can use the `audit2allow` command to read the SELinux audit logs and generate new SELinux rules to allow the denied actions selectively. This can be a quick way to get your applications working on your Linux system without disabling SELinux.

Using the Enforcing mode

The name says it all. In *enforcing mode*, SELinux is turned on and all of the security policy rules are enforced.

Understanding SELinux security contexts

As mentioned earlier, an SELinux security context is the method used to classify objects (such as files) and subjects (such as users and programs). The defined security context allows SELinux to enforce policy rules for subjects accessing objects. A security context consists of four attributes: `user`, `role`, `type`, and `level`.

User The `user` attribute is a mapping of a Linux username to an SELinux name. This is not the same as a user's login name, and it is referred to specifically as the SELinux user. The SELinux username ends with a `u`, making it easier to identify in the output. Regular unconfined users have an `unconfined_u` user attribute in the default targeted policy.

Role A designated role in the company is mapped to an SELinux role name. The `role` attribute is then assigned to various subjects and objects. Each role is granted access to other subjects and objects based on the role's security clearance and the object's classification level. More specifically, for SELinux, users are assigned a role and roles are authorized for particular types or domains. Using roles can force accounts, such as root, into a less privileged position. The SELinux role name has an `r` at the end. On a targeted SELinux system, processes run by the root user have a `system_r` role, while regular users run under the `unconfined_r` role.

Type This `type` attribute defines a domain type for processes, a user type for users, and a file type for files. This attribute is also called security type. Most policy rules are concerned with the security type of a process and what files, ports, devices, and other elements of the system that process has access to (based on their security types). The SELinux type name ends with a `t`.

Level The `level` is an attribute of Multi-Level Security (MLS), and it enforces the Bell-LaPadula model. It is optional in TE but is required if you are using MLS.

The MLS level is a combination of the sensitivity and category values that together form the security level. A level is written as `sensitivity : category`.

`sensitivity`

- Represents the security or sensitivity level of an object, such as confidential or top secret.
- Is hierarchical, with `s0` (unclassified) typically being the lowest.
- Is listed as a pair of sensitivity levels (*`lowlevel-highlevel`*) if the levels differ.
- Is listed as a single sensitivity level (`s0`) if there are no low and high levels. In some cases, however, even if there are no low and high levels, the range is still shown (`s0-s0`).

`category`

- Represents the category of an object, such as No Clearance, Top Clearance, and so on.
- Traditionally, the values are between `c0` and `c255`.
- Is listed as a pair of category levels (*`lowlevel.highlevel`*) if the levels differ.
- Is listed as a single category (level) if there are no low and high levels.

Users have security contexts

To see your SELinux user context, enter the `id` command at the shell prompt. The following is an example of the security context for user `johndoe`:

```
$ id
uid=1000(johndoe) gid=1000(johndoe) groups=1000(johndoe)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

The user's security context list shows the following:

user: The Linux user, johndoe, is mapped to the SELinux unconfined_u user.

role: The SELinux user, unconfined_u, is mapped to the role of the unconfined_r.

type: The user has been given the type of unconfined_t.

level:

sensitivity: The user has only one sensitivity level, and it is the lowest level of s0.

categories: The user has access to c0.c1023, which is all categories (c0 through to c1023).

Files have security contexts

A file also has a security context. To see an individual file's context, use the -Z option on the ls command. The following is a security context for the file my_stuff:

```
$ ls -Z my_stuff
-rw-rw-r--. johndoe johndoe
 unconfined_u:object_r:user_home_t:s0 my_stuff
```

The file context list shows the following:

user: The file is mapped to the SELinux unconfined_u user.

role: The file is mapped to the role of object_r.

type: The file is considered to be part of the user_home_t domain.

level:

sensitivity: The user has only one sensitivity level, and it is the lowest level of s0.

categories: MCS is not set for this file.

Processes have security contexts

A process's security context has the same four attributes as a user's and a file's context. To see process information on a Linux system, you typically use a variant of the ps command. In the following code, the ps -el command was used.

```
# ps -el | grep bash
0 S  1000  1589  1583  0  80   0 -  1653 n_tty_ pts/0    00:00:00
bash
0 S  1000  5289  1583  0  80   0 -  1653 wait   pts/1    00:00:00
bash
4 S     0  5350  5342  0  80   0 -  1684 wait   pts/1
00:00:00 bash
```

To see a process's security context, you need to use the `-Z` option on the `ps` command. In the example that follows, the `ps -eZ` command was used and then piped into `grep` to search only for processes running the bash shell.

```
# ps -eZ | grep bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 1589 pts/0
00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5289 pts/1
00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5350 pts/1
00:00:00 bash
```

The process context list shows the following:

`user`: Process is mapped to the SELinux `unconfined_u` user.

`role`: Process is running as the `unconfined_r` role.

`type`: Process is running in the `unconfined_t` domain.

`level`:

- **`sensitivity`**: Process has only level `s0`.
- **`categories`**: Process has access to `c0.c1023`, which is all categories (`c0` through to `c1023`).

These security contexts can all be changed to meet your organization's particular security needs. However, before you learn how to change the settings of these security contexts, you need to understand another piece of the SELinux puzzle, SELinux policy types.

Understanding SELinux Policy types

The *policy type* chosen directly determines what sets of policy rules are used to dictate what an object can access. The policy type also determines what specific security context attributes are needed. This is where you start to see the fine level of access control that can be implemented via SELinux.

Note

The policy types available on your distribution may not match the ones listed here. For instance, on older Linux distributions, the strict policy is still available. On newer distributions, the strict policy has been merged into the Targeted policy, with Targeted used by default.

SELinux has different policies among which you can choose:

- Targeted
- MLS
- Minimum

Each policy implements different access controls to match your organization's needs. It is critical to understand these policy types in order to select the correct one for your particular security requirements.

Targeted policy

The *Targeted policy*'s primary purpose is to restrict "targeted" daemons. However, it can also restrict other processes and users. Targeted daemons are sandboxed. A *sandbox* is an environment where programs can run but their access to other objects is tightly controlled.

A process running in such an environment is said to be "sandboxed." Thus, a targeted daemon is restricted so that no malicious attacks launched through them can affect other services or the Linux system as a whole. Targeted daemons make it safer for you to share your print server, file server, web server, or other services while limiting the risks that access to those services pose to other assets on your system.

All subjects and objects not targeted are run in the `unconfined_t` domain. The `unconfined_t` domain has no SELinux policy restrictions and thus only uses the "traditional" Linux security.

SELinux comes with the Targeted policy set as the default. Thus, by default, SELinux targets only a few daemons.

MLS (Multi-Level Security) policy

The *MLS policy*'s primary purpose is to enforce the Bell-LaPadula model. It grants access to other subjects and objects based upon a role's *security clearance* and the object's *classification level*.

In the MLS policy, a security context's MLS attribute is critical. Otherwise, the policy rules will not know how to enforce access restrictions.

Minimum policy

This policy is just as it sounds—minimal. It was originally created for low-memory machines or devices such as smartphones.

The *Minimum policy* is essentially the same as the Targeted policy, but only the base policy rule package is used. This "bare-bones" policy can be used to test out the effects of SELinux on a single designated daemon. For low-memory devices, the Minimum policy allows SELinux to run without consuming a great deal of resources.

Understanding SELinux policy rule packages

Policy rules, also called *allow rules*, are the rules used by SELinux to determine if a subject has access to an object. Policy rules are installed with SELinux and are grouped into packages, also called *modules*.

On your Linux system, there is user documentation on these various policy modules in the form of HTML files. To view this documentation on Fedora or RHEL, open your system's browser and type in the following URL: `file:///usr/share/doc/selinux-policy/html/index.html`. For Ubuntu, the URL is `file:///usr/share/doc/`

`selinux-policy-doc/html/index.html`. If you do not have the policy documentation on your system, you can install it on a Fedora or RHEL system by typing **`yum install selinux-policy-doc`** at the command line. On Ubuntu, type **`sudo apt-get install selinux-policy-doc`** at the command line.

You can review this policy documentation to see how policy rules are created and packaged.

The policy rule packages, along with the SELinux operational modes, policy type, and various security contexts, work together to secure your Linux system via SELinux. The following sections cover how to begin configuring SELinux to meet your particular organization's security needs.

Configuring SELinux

SELinux comes preconfigured. You can use the SELinux features without any configuration work. However, rarely do the preconfigured settings meet all of your Linux system's security needs.

SELinux configurations can only be set and modified by the root user. Configuration and policy files are located in the `/etc/selinux` directory. The primary configuration file is the `/etc/selinux/config` file, and it appears as follows:

```
# cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#      enforcing - SELinux security policy is enforced.
#      permissive - SELinux prints warnings instead of enforcing.
#      disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy.
#               Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

This main SELinux configuration file allows you to set the mode and the policy type.

Setting the SELinux mode

To see SELinux's current mode on your system, use the `getenforce` command. To see both the current mode and the mode set in the configuration file, use the `sestatus` command. Both commands are shown in the code that follows:

```
# getenforce
Enforcing
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
```

```
SELinux root directory:          /etc/selinux
Loaded policy name:              targeted
Current mode:                    enforcing
Mode from config file:           enforcing
Policy MLS status:               enabled
Policy deny_unknown status:      allowed
Memory protection checking:      actual (secure)
Max kernel policy version:       31
```

To change the mode setting, you can use the `setenforce` *`newsetting`*, where *`newsetting`* is either

- enforcing or 1
- permissive or 0

Notice that you cannot use the `setenforce` command to change SELinux to disabled mode.

The example that follows shows the SELinux mode being changed immediately to permissive mode via the `setenforce` command. The `sestatus` command shows the current operational mode and the mode in the configuration file, which has not been modified. When the system is rebooted, it determines the SELinux operational mode from the configuration file. Thus, the permissive mode set in the example that follows is temporary because the enforcing mode is set via the configuration file when the system is rebooted.

```
# setenforce 0
# getenforce
Permissive
# sestatus
SELinux status:                  enabled
SELinuxfs mount:                 /sys/fs/selinux
SELinux root directory:          /etc/selinux
Loaded policy name:              targeted
Current mode:                    permissive
Mode from config file:           enforcing
...
```

CAUTION

It is best to switch from the disabled to the enforcing mode by modifying the configuration file and rebooting. Switching from disabled to enforcing via the `setenforce` command may hang your system as a result of incorrect file labels. Keep in mind that, when rebooting after changing from `disabled` mode, there could be a long wait for your filesystem to be relabeled after the system comes back up in permissive or enforcing mode.

To disable SELinux, you must edit the SELinux configuration file. Rebooting the system always changes the mode back to what is set in that configuration file. The preferred method of changing the SELinux mode is to modify the configuration file and then reboot the system.

When switching from disabled to either enforcing or permissive mode, SELinux automatically relabels the filesystem after a reboot. This means that SELinux checks and changes the security contexts of any files with incorrect security contexts (for example, mislabeled files) that can cause problems in the new mode. Also, any files not labeled are labeled with contexts. This relabeling process can take a long time because each file's context is checked. Following is the message that you'll receive when a system is going through a relabeling process after a reboot:

```
*** Warning -- SELinux targeted policy relabel is required.
*** Relabeling could take a very long time, depending on file
*** system size and speed of hard drives.
```

To modify the mode in the `/etc/selinux/config` file, change the line `SELINUX=` to one of the following:

- `SELINUX=disabled`
- `SELINUX=enforcing`
- `SELINUX=permissive`

The SELinux configuration file example that follows shows that the mode has been set to permissive. Now, when a system reboot occurs, the mode is changed.

```
# cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy.
#               Only selected processes are protected.
#     mls - Multi Level Security protection
SELINUX=permissive
...
```

The primary SELinux configuration file does not just contain the mode setting. It also specifies the policy type, which will be enforced.

Setting the SELinux policy type

The policy type you choose determines whether SELinux enforces TE, MLS, or a base package. This type setting directly determines the sets of policy rules used to dictate what an object can access.

By default, the policy type is set to `targeted`. To change the default policy type, edit the `/etc/selinux/config` file. Change the line `SELINUXTYPE=` to one of the following:

- `SELINUX=targeted`
- `SELINUX=mls`
- `SELINUX=minimum`

If you set the SELinux type to mls or minimum, you need to make sure that you have their policy package installed first. Check by typing the following command:

```
yum list selinux-policy-mls or yum list selinux-policy-minimum
```

NOTE
To check the SELinux policy packages on Ubuntu, use the command `sudo apt-cache policy` *package_name*.

The example of the SELinux configuration file that follows shows that the type has been set to `mls`. Now when a system reboot occurs, the policy type is changed.

```
# cat /etc/selinux/config
# This file controls the state of SELinux on the system.
...
# SELINUXTYPE= type of policy in use. Possible values are:

#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy.
#               Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=mls
```

Managing SELinux security contexts

SELinux security contexts allow SELinux to enforce policy rules for subjects accessing objects. Your Linux system comes with security contexts already assigned.

To view current SELinux file and process security contexts, use the `secon` command. Table 24.1 lists available options on the `secon` command.

TABLE 24.1 `secon` **Command Options**

Option	Description
`-u`	Use this option to show the user of the security context.
`-r`	Use this option to show the role of the security context.
`-t`	Use this option to show the type of the security context.
`-s`	Use this option to show the sensitivity level of the security context.
`-c`	Use this option to show the clearance level of the security context.
`-m`	Use this option to show the sensitivity and clearance level of the security context as an MLS range.

If you use the `secon` command with no designation, it shows you the current process's security context. To see another process's security context, use the `-p` option. The example that follows shows you how to use `secon` to view the current and the `systemd` process's security context.

```
# secon -urt
user: unconfined_u
role: unconfined_r
type: unconfined_t
# secon -urt -p 1
user: system_u
role: system_r
type: init_t
```

To view a file's security context, you use the `-f` option, as shown here:

```
# secon -urt -f /etc/passwd
user: system_u
role: object_r
type: passwd_file_t
```

The `secon` command doesn't show your security context. To see your security context, use the `id` command.

Managing the user security context

Remember that every system user login ID is mapped to a particular SELinux user ID. To see a mapping list on your system, enter the `semanage login -l` command. The `semanage` command and its output are shown in the code that follows. If a user login ID is not listed, then it uses the "default" login mapping, which is the `Login Name` of `_default_`. Notice that the associated MLS/MCS settings for each SELinux user are shown as well.

```
# semanage login -l
Login Name           SELinux User         MLS/MCS Range     Service
__default__          unconfined_u         s0-s0:c0.c1023    *
root                 unconfined_u         s0-s0:c0.c1023    *
```

To see a current display of the SELinux users and their associated roles, use the command `semanage user -l`. The partial display that follows shows roles mapped to SELinux usernames:

```
# semanage user -l

                Labeling MLS/      MLS/
SELinux User    Prefix   MCS Level MCS Range       SELinux Roles
guest_u         user     s0        s0              guest_r
...
user_u          user     s0        s0              user_r
xguest_u        user     s0        s0              xguest_r
```

If you need to add a new SELinux username, the `semanage` utility is used again. This time, the command is `semanage user -a` *`selinux_username`*. To map a login ID to the newly-added SELinux username, the command is `semanage login -a -s` *`selinux_username loginID`*. The `semanage` utility is a powerful tool in managing your SELinux configuration. For more information on the `semanage` utility, see the man pages.

Managing the file security context

Labeling files is critical to maintaining proper access control to each file's data. SELinux does set file security labels upon installation and upon system reboot when the SELinux mode is switched from `disabled`. To see a file's current label (aka security context), use the `ls -Z` command, as shown here:

```
# ls -Z /etc/passwd
-rw-r--r--. root root system_u:object_r:etc_t:s0 /etc/passwd
```

You can use several commands to manage file security context labels, as shown in Table 24.2.

TABLE 24.2 File Security Context Label Management Commands

Utility	Description
`chcat`	Use this to change a file's security context label's category.
`chcon`	Use this to change a file's security context label.
`fixfiles`	This calls the `restorecon/setfiles` utility.
`restorecon`	This does the exact same thing as `setfiles` utility, but it has a different interface than `setfiles`.
`setfiles`	Use this to verify and/or correct security context labels. It can be run for file label verification and/or relabeling files when adding a new policy module to the system. Does exactly the same thing as the `restorecon` utility but has a different interface than `restorecon`.

The `chcat` and `chcon` commands, shown in Table 24.2, allow you to change a file's security context. In the following example, the `chcon` command is used to change the SELinux user associated with `file.txt` from `undefined _ u` to `system _ u`.

```
# ls -Z file.txt
-rw-rw-r--. johndoe johndoe
 unconfined_u:object_r:user_home_t:s0 file.txt
# chcon -u system_u file.txt
# ls -Z file.txt
-rw-rw-r--. johndoe johndoe
 system_u:object_r:user_home_t:s0 file.txt
```

Notice in Table 24.2 that `fixfiles`, `restorecon`, and `setfiles` are essentially the same utility. However, `restorecon` is the popular choice to use when fixing files' labels. The command `restorecon` *`filename`* changes a file back to its default security context.

Managing the process security context

The definition of a process is a running program. When you run programs or start services on a Linux system, each one is given a process ID (see Chapter 6, "Managing Running Processes"). On a system with SELinux, a process is also given a security context.

How a process gets its security context depends upon which process started it. Remember that `systemd` (previously `init`) is the "mother" of all processes (see Chapter 15, "Starting and Stopping Services"). Thus, many daemons and processes are started by `systemd`. The processes `systemd` starts are given new security contexts. For instance, when the `apache` daemon is started by `systemd`, it is assigned the type (aka domain) `httpd_t`. The context assigned is handled by the SELinux policy written specifically for that daemon. If no policy exists for a process, then it is assigned a default type, `unconfined_t`.

For a program or application run by a user (parent process), the new process (child process) inherits the user's security context. Of course, this occurs only if the user is allowed to run the program. A process can also run a program. The child process in this case also inherits its parent process's security context. Thus, the child process runs in the same domain.

So, a process's security context is set before the program is run and depends upon who started it. You can use a couple of commands to change the security contexts under which a program is run:

`runcon`: Run the program using options to determine the user, role, and type (aka domain).

`sandbox`: Run the program within a tightly controlled domain (aka sandbox).

You can cause several problems by using `runcon`, so use it with caution. However, `sandbox` offers a great deal of protection. It allows flexibility in testing out new programs on your Linux system.

Managing SELinux policy rule packages

Policy rules are the rules used by SELinux to determine whether a subject has access to an object. They are grouped into packages, also called modules, and are installed with SELinux. An easy way to view the modules on your system is to use the `semodule -l` command. It lists all of the policy modules along with their current version number. An example of the `semodule -l` command is shown here:

```
# semodule -l
abrt
accountsd
acct
...
```

```
xserver
zabbix
zarafa
zebra
zoneminder
zosremote
```

Several tools can help you manage and even create your own policy modules. Table 24.3 shows the various policy rule package tools available on a Fedora system.

TABLE 24.3 SELinux Policy Package Tools

Policy Tool	Description
`audit2allow`	Generates policy `allow`/`dontaudit` rules from logs of denied operations
`audit2why`	Generates a description of why the access was denied from logs of denied operations
`checkmodule`	Compiles policy modules
`checkpolicy`	Compiles SELinux policies
`load_policy`	Loads new policies into the kernel
`semodule_expand`	Expands a policy module package
`semodule_link`	Links policy module packages together
`semodule _ package`	Creates a policy module package

The following is an example policy typically used as a framework to create local policy rules. The example policy is rather long, so only a portion of it is shown.

```
# cat /usr/share/doc/selinux-policy/example.te

policy_module(myapp,1.0.0)

########################################
#
# Declarations
#

type myapp_t;
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t)

type myapp_log_t;
logging_log_file(myapp_log_t)
```

```
type myapp_tmp_t;
files_tmp_file(myapp_tmp_t)
...
allow myapp_t myapp_tmp_t:file manage_file_perms;
files_tmp_filetrans(myapp_t,myapp_tmp_t,file)
#
```

You can see from the preceding example code that a special syntax is used in policy code. To create and modify policy rules, you need to learn this policy rule language syntax, learn how to use the SELinux policy compilers, and learn how to link policy rule files together to form modules; you probably need to take a couple of day-long classes to accomplish this. You may be tempted to give up on SELinux at this point. However, it is much easier to use Booleans to modify policies.

Managing SELinux via Booleans

SELinux policy rule writing and module creation is a rather complicated and time-consuming activity. Creating incorrect policy rules could potentially compromise your Linux system's security. Thankfully, SELinux provides Booleans.

A Boolean is a toggle switch that toggles a setting on or off. A Boolean switch allows you to change parts of SELinux policy rules without any knowledge of policy writing. These policy changes can be done without a system reboot too!

To see a list of all of the current Booleans used in SELinux, use the `getsebool -a` command. The following is an example of the SELinux policy rules with Booleans on a Fedora Linux system:

```
# getsebool -a
abrt_anon_write --> off
abrt_handle_event --> off
...
xserver_object_manager --> off
zabbix_can_network --> off
```

To see a specific policy that can be modified by a Boolean, the `getsebool` command is used again. This time, the policy name is passed to it, as shown in the following example:

```
# getsebool httpd_can_connect_ftp
httpd_can_connect_ftp --> off
```

To toggle a policy, you can use the `setsebool` command. This command changes the policy rule temporarily. When the system is rebooted, the Boolean returns to its original setting. If you need this setting to be permanent, you can use `setsebool` with the `-P` option.

The `setsebool` command has six settings: three for turning a policy on (`on`, `1`, or `true`), and three for turning a policy off (`off`, `0`, or `false`).

One example where you might want to use `setsebool` relates to restricting the use of executable files. In some situations, it is not good security to allow users to execute programs

from their /home directory. To prevent this from happening, the allow_user_exec_content policy rule needs to be turned off. The example that follows shows the setsebool command being used to do just that. Notice that the -P option is used to make this setting permanent.

```
# setsebool -P allow_user_exec_content off
```

The getsebool command verifies that the Boolean setting has been correctly made:

```
# getsebool allow_user_exec_content
allow_user_exec_content --> off
```

Booleans make modifying current SELinux policy rules much easier. Overall, the SELinux command line configuration utilities, such as getsebool, are easy to use. However, if you want a GUI configuration tool, SELinux has one. It is installed via the command yum install policycoreutils-gui. On Ubuntu, use the command sudo apt-get install policycoreutils. To use this configuration tool, simply type in the command **system-config-selinux** and a GUI interface appears.

Monitoring and Troubleshooting SELinux

SELinux is another tool for monitoring your system. It logs all access denials, which can help you determine whether an attack is being attempted. These same SELinux log files are also useful in troubleshooting SELinux problems.

Understanding SELinux logging

SELinux uses a cache called the *Access Vector Cache (AVC)* when reviewing policy rules for particular security contexts. When access is denied, called an AVC denial, a denial message is put into a log file.

These logged denial messages can help you diagnose and address routine SELinux policy violations. Where these denial messages are logged depends upon the status of the auditd and rsyslogd daemons:

- If the auditd daemon is running, the denial messages are logged to /var/log/audit/audit.log.
- If auditd is not running, but the rsyslogd daemon is running, the denial messages are logged to /var/log/messages.

Note

If both auditd and rsyslogd are running, and you have the setroubleshootd daemon on your system, denial messages are sent to both the audit.log and messages log files. However, denial information in the messages log file is put into a more understandable format by the setroubleshootd daemon.

Reviewing SELinux messages in the audit log

If you have the `auditd` daemon running, you can quickly see if any AVC denials have been logged by using the `aureport` command. The example that follows shows the use of `aureport` and `grep` to search for AVC denials. At least one denial has been logged to `/var/log/audit/audit.log`:

```
# aureport | grep AVC
Number of AVC's: 1
```

After you discover that an AVC denial has been logged in `audit.log`, you can use `ausearch` to review the denial message(s). The example that follows shows the `ausearch` command being used to review the logged AVC denial message:

```
# ausearch -m avc
type=AVC msg=audit(1580397837.344:274): avc: denied { getattr } for
pid=1067
   comm="httpd" path="/var/myserver/services" dev="dm-0" ino=655836
   scontext=system_u:system_r:httpd_t:s0
   tcontext=unconfined_u:object_r:var_t:s0 tclass=file permissive=0
```

The display provides information on who was attempting access, along with their security context when attempting it. Look for these key words in an AVC denial message:

- `type=AVC`
- `avc: denied`
- `com="httpd"`
- `path="/var/myserver/services"`

This can give you enough data either to begin fixing a problem or to track down malicious activity. Here, the `/var/myserver/services` directory has the wrong SELinux file context to be read by the `httpd` service.

Reviewing SELinux messages in the messages log

If you have the `auditd` service running, you can find AVC denial messages by searching through the `/var/log/audit/audit.log` file using `grep`. For the latest RHEL and Fedora systems, or any Linux system using `systemd`, you can run the `journalctl` command to check for AVC denial log messages as well. Within each log message is an AVC message that you can view to get information about that AVC denial, as in the following example:

```
# journalctl | grep AVC
type=AVC msg=audit(1580397837.346:275): avc: denied { getattr }for
pid=1067
   comm="httpd" path="/var/myserver/services" dev="dm-0" ino=655836
   scontext=system_u:system_r:httpd_t:s0
   tcontext=unconfined_u:object_r:var_t:s0 tclass=file permissive=0
```

Since you know that there are AVC denials, you can pass the entire `/var/log/audit/audit.log` file to `sealert` to step through the issues:

```
# sealert -a /var/log/audit/audit.log
SELinux is preventing httpd from getattr access on the file
/var/myserver/services.

*****  Plugin catchall (100. confidence) suggests   *************

If you believe that httpd should be allowed getattr access on the
services file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# ausearch -c 'httpd' --raw | audit2allow -M my-httpd
# semodule -X 300 -i my-httpd.pp

Additional Information:
Source Context                system_u:system_r:httpd_t:s0
Target Context                unconfined_u:object_r:var_t:s0
Target Objects                /var/myserver/services [ file ]
...
Raw Audit Messages
type=AVC msg=audit(1580397837.346:275): avc:  denied  { getattr }
for  pid=1067 comm="httpd" path="/var/myserver/services" dev="dm-0"
ino=655836 scontext=system_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:var_t:s0 tclass=file permissive=0
Hash: httpd,httpd_t,var_t,file,getattr
```

In this case, if you want to allow access by the `httpd` service to the content in the directory being denied, you can run the `ausearch` and `semodule` commands shown in the output. This creates and applies a new SELinux policy to allow access to the content. Provided there are no other permission problems, `httpd` should be able to access that content.

Troubleshooting SELinux logging

Obviously, the log files are extremely important for diagnosing and addressing SELinux policy violations. The log files, or directly querying the `systemd` journal (`journalctl` command), are your first steps in troubleshooting SELinux. Thus, it is important to make sure that your Linux system is logging messages in the first place.

A quick way to determine if the logging is taking place is to check if the proper daemons are running: `auditd`, `rsyslogd`, and/or `setroubleshootd`. Use an appropriate command, such as `systemctl status auditd.service`. Of course, the command you use depends on your Linux distribution and its version. See Chapter 15 for more details. If the daemon is not running, start it so that logging may begin to occur.

CAUTION

Sometimes AVC denials are not logged because of `dontaudit` policy rules. Although the `dontaudit` rules help reduce false positives in the logs, they can cause problems when troubleshooting. To fix this, temporarily disable all `dontaudit` policy rules using the command `semodule -DB`.

Troubleshooting common SELinux problems

When you begin working with SELinux, it is easy to overlook the obvious. Whenever access is denied, you should first check the "traditional" Linux DAC permissions. For example, use the `ls -l` command and double-check that a file's owner; group; and read, write, and execute assignments are correct.

With SELinux, several regular items can cause problems:

- Using a nonstandard directory for a service
- Using a nonstandard port for a service
- Moving files that result in losing their security context labels
- Having Booleans set incorrectly

Each one of these problems can be solved fairly quickly.

Using a nonstandard directory for a service

For various reasons, you may decide to store a service's files in a nonstandard directory. When you do this, SELinux needs to know that this nonstandard behavior has occurred. Otherwise, it denies access to legitimate service access requests.

For example, you decided to keep your HTML files in a different location from the standard `/var/www/html`. You put the files in `/abc/www/html`. You must let SELinux know that you want the `http` service to be able to access the files within `/abc/www/html`. The commands to accomplish this are `semanage` and `restorecon`. In the following code snippet, the commands are used to add the proper security context type on the `/abc/www/html` directory and all it contains:

```
# semanage fcontext -a -t httpd_sys_content_t  "/abc/www/html(/.*)?"
```

To actually set the new security context type to the files within the directory, you need to use the `restorecon -R` command. This is accomplished in the following code:

```
# restorecon -R -v /abc/www/html
# ls -Z /abc/www/html
unconfined_u:object_r:httpd_sys_content_t:s0 abc
```

Now the `httpd` daemon has permission to access your HTML files in their non-standard directory location.

Using a nonstandard port for a service

Similar to the problem just described, you may decide to have a service listening on a nonstandard port. When you make this port change, the service often fails to start.

For example, you decide for security purposes to move `sshd` from port 22 to a nonstandard port, 47347. SELinux does not know about this port, and the service fails to start. To fix this problem, you must first find the security context type for `sshd`. This is accomplished using the code that follows by issuing the `semanage port -l` command and piping the results into `grep` to search for `ssh`.

```
# semanage port -l | grep ssh
ssh_port_t                 tcp              22
```

In the preceding example, you can see that the context type needed is `ssh_port_t`. Now, using the `semanage` command again, you add that type to port 47347, as shown here:

```
# semanage port -a -t ssh_port_t -p tcp 47347
# semanage port -l | grep ssh
ssh_port_t                 tcp              47347, 22
```

At this point, edit the `/etc/ssh/sshd _ config` file to add a `Port 47347` line to the file. Then restart the `sshd` service so that the service listens on the nonstandard port 47347.

Moving files and losing security context labels

You used the `cp` command to move a file from `/etc` temporarily to the `/tmp` directory. Then you used the `mv` command to put it back. Now the file has the security context of the temporary directory instead of its original security context, and your system is getting AVC denial messages when the service using that file tries to start up.

There is an easy fix, thanks to the `restorecon -R` command. Simply type in **`restorecon file`**, and the file has its permanent security context restored.

Booleans set incorrectly

Another common problem is simply setting a Boolean incorrectly. This can give you several AVC denials.

For example, if your system's scripts are no longer able to connect out to the network and you are getting AVC denials in your logs, you need to check the `httpd` Booleans. Use the `getsebool -a` command, and pipe it into `grep` to search for any Booleans that affect `httpd`. The example here shows these commands being used:

```
# getsebool -a | grep http
...
httpd_can_network_connect --> off
...
```

The `getsebool` command shows the Boolean `httpd_can_network_connect` is set to off. To change this Boolean, use the following command: `setsebool -P`

`httpd_can_network_connect on`. Notice the `-P` option was used to make the setting permanent. Now your scripts should be able to connect out to the network.

Putting It All Together

Obviously, SELinux is a rather complicated and rich tool. You now have a good, solid foundation on the SELinux basics. Here are some recommendations as you get started implementing SELinux on your system.

The default `targeted` SELinux mode can be used to secure most basic network services (`httpd`, `vsftpd`, Samba, and so on) without you needing to assign special user roles or otherwise lock down your system. In this case, the main things you need to do are to put files in standard locations (or run commands to assign the proper file contexts to nonstandard locations), make sure that Booleans are turned on for less secure features that you want on anyway, and watch AVC denials for problems.

- Start with the permissive operational mode. This allows requests to succeed that SELinux sees as insecure.
- Run your current system for a significant amount of time in permissive mode. Review the logs and see what problems may occur with the default SELinux settings. You can then change Booleans or file contexts so that features improperly denied can be allowed. After the problems are worked out, turn on enforcing mode.
- Overall, implement SELinux configuration changes one at a time in a test environment or using permissive mode. See what kind of effect each configuration change has before moving on to the next one. You can then use the `audit2allow` command to allow actions that were stopped by AVC denials to be selectively allowed in the policy for a service.

Obtaining More Information on SELinux

Several additional sources of information can help you with SELinux on your Linux system:

Your System's man Pages Issue the command `man -k selinux` to find all of the various man pages that you can review for the SELinux utilities currently installed on your system. If you are debugging SELinux problems for a well-known service (such as `httpd`, `vsftpd`, Samba, and so on), there is probably a man page associated with how specifically to fix SELinux problems with that service.

The Red Hat Enterprise Linux Manuals Located at `http://docs.redhat.com`, this site contains an entire manual on SELinux.

The Fedora Project SELinux Guide Located at `http://docs.fedoraproject.org`, this site has a Security-Enhanced Linux Guide. However, the guide is not updated for every Fedora version, so you may need to look in older versions to find it. Also, the SELinux Guide is not located within the Security manual, but the Security manual is a good manual to review as well.

SELinux Project Wiki This is the official SELinux project page. Several resources are available at this site, which is located at `http://selinuxproject.org`.

Summary

SELinux provides a security enhancement to Linux, and it is installed by default on many Linux distributions. In this chapter, you learned the benefits of SELinux, how it works, how to set it up, how to fix various problems with SELinux, and how to get more information about this important security enhancement.

At first glance, SELinux appears rather complicated. However, after it's broken down into its various components—operational modes, security contexts, policy types, and policy packages—you can see how the various pieces work together. Each component plays an important role for enforcing and testing the chosen security requirements for your organization.

You learned about the various steps available to configure SELinux. Even though SELinux comes preconfigured, you may need to make some modifications to meet your organization's security needs. Each component has its own configuration steps and settings to choose. Though policy rule creation was not covered, you did learn how to modify the supplied policies via Booleans.

SELinux provides another tool for monitoring your Linux system's security. Because SELinux logs all access denials, it can help you determine if an attack has been or is being attempted. Even the best-laid plans can go badly. Therefore, in this chapter, you learned how to fix common SELinux configuration problems.

In the next chapter, you'll learn how to protect your Linux system on a network. You'll learn about controlling access, managing firewalls, and securing remote access.

Exercises

Use these exercises to test your knowledge of using SELinux. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Making no changes to the SELinux primary configuration file, write down the command to set your system into the permissive operating mode for SELinux.
2. Making no changes to the SELinux primary configuration file, write down the command to set your system into the enforcing mode for SELinux.
3. What current and permanent SELinux policy types are set on your system and how did you find them?
4. List the security context for the `/etc/hosts` file and identify its different security context attributes.
5. Create a file called `test.html` in your home directory, and assign its type to `httpd_sys_content_t`. (This is something that you might do to make content available to be shared by your web server outside of the common `/var/www/html` directory.)

6. List the security context for the running `crond` process and identify its security context attributes.
7. Create a file called `/etc/test.txt`, change its file context to `user_tmp_t`, restore it to its proper content (the default context for the `/etc` directory), and remove the file. Use the `ls -Z /etc/test.txt` command to check the file at each point in the process.
8. You have a tftp server on your private network, and you want to allow anonymous writes and access to the tftp service's home directory (while SELinux is in enforcing mode). Determine what Booleans allow anonymous writes and access to the tftp service's home directory and turn those Booleans on.
9. What command would list out all of the SELinux policy modules on your system along with their version number?
10. Tell SELinux to allow access to the sshd service through TCP Port 54903.

CHAPTER 25

Securing Linux on a Network

IN THIS CHAPTER

Managing network services

Controlling access to network services

Implementing firewalls

Setting up your Linux system on a network, especially a public network, creates a whole new set of challenges when it comes to security. The best way to secure your Linux system is to keep it off all networks. However, that is rarely a feasible option.

Entire books have been filled with information on how to secure a computer system on a network. Many organizations hire full-time computer security administrators to watch over their network-attached Linux systems. Therefore, think of this chapter as a brief introduction to securing Linux on a network.

Auditing Network Services

Most Linux systems used for large enterprises are configured as servers that, as the name implies, offer services to remote clients over a network. A *network service* is any task that the computer performs requiring it to send and receive information over the network using some predefined set of rules. Routing email is a network service, as is serving web pages.

A Linux server has the potential to provide thousands of services. Many of them are listed in the `/etc/services` file. Consider the following sections from the `/etc/services` file:

```
$ cat /etc/services
# /etc/services:
# $Id: services,v 1.55 2013/04/14 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2013-04-10
#
```


Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

```
# Note that it is presently the policy of IANA to assign ...
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]
...
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
systat          11/udp          users
daytime         13/tcp
daytime         13/udp
qotd            17/tcp          quote
qotd            17/udp          quote
...
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp-data        20/udp
# 21 is registered to ftp, but also used by fsp
ftp             21/tcp
...
http            80/tcp      www www-http    # WorldWideWeb HTTP
http            80/udp      www www-http    # HyperText Transfer
Protocol
http            80/sctp                     # HyperText Transfer
Protocol
kerberos        88/tcp      kerberos5 krb5  # Kerberos v5
kerberos        88/udp      kerberos5 krb5  # Kerberos v5
...
blp5            48129/udp           # Bloomberg locator
com-bardac-dw   48556/tcp           # com-bardac-dw
com-bardac-dw   48556/udp           # com-bardac-dw
iqobject        48619/tcp               # iqobject
iqobject        48619/udp               # iqobject
```

After the comment lines, notice three columns of information. The left column contains the name of each service. The middle column defines the port number and protocol type used for that service. The right column contains an optional alias or list of aliases for the service.

Many Linux distributions come with unneeded network services running. An unnecessary service exposes your Linux system to malicious attacks. For example, if your Linux server is a print server, then it should only be offering printing services. It should not also offer Apache Web Services. This would unnecessarily expose your print server to any malicious attacks that take advantage of web service vulnerabilities.

Originally, restricting services on Linux systems meant setting up individual physical Linux servers with only a few services running on each. Later, running multiple Linux virtual machines on a physical host let you lock down small sets of services on virtual machines. More recently, containerized applications can allow many more separate and secured services to run on each physical host.

Evaluating access to network services with nmap

A wonderful tool to help you review your network services from a network standpoint is the `nmap` security scanner. The `nmap` utility is available in most Linux distribution repositories and has a web page full of information at `http://nmap.org`.

To install `nmap` on a Fedora or RHEL distribution, use the `yum` or `dnf` command (using root privileges), as shown in the example that follows.

```
# yum install nmap -y
Updating Subscription Management repositories.
Last metadata expiration check: 0:03:41 ago on Sat 12 Oct 2019
11:24:07 PM EDT.
Dependencies resolved.
================================================================================
 Package      Arch    Version       Repository                    Size
================================================================================
Installing:
 nmap         x86_64  2:7.70-4.el8  rhel-8-for-x86_64-appstream-rpms
5.8 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 5.8 M
Installed size: 24 M
...
Installed:
  nmap-2:7.70-4.el8.x86_64

Complete!
```

To install the `nmap` utility on an Ubuntu distribution, type **`sudo apt-get install nmap`** at the command line.

The `nmap` utility's full name is Network Mapper. It has a variety of uses for security audits and network exploration. Using `nmap` to do various port scans allows you to see what services are running on all of the servers on your local network and whether they are advertising their availability.

Note

What is a port? Ports, or more correctly *network ports*, are numeric values used by the TCP and UDP network protocols as access points to services on a system. Standard port numbers are assigned to services so that a service knows to listen on a particular port number and a client knows to request the service on that port number.

For example, port 80 is the standard network port for unencrypted (HTTP) traffic to the Apache web service. So, if you ask for `www.example.com` from your web browser, the browser assumes that you mean to use TCP port 80 on the server that offers that web content. Think of a network port as a door to your Linux server. Each door is numbered. And behind every door is a particular service waiting to help whoever knocks on that door.

To audit your server's ports, the `nmap` utility offers several useful scan types. The `nmap` site has an entire manual on all of the port scanning techniques that you can use at `http://nmap.org/book/man-port-scanning-techniques.html`. Here are two basic port scans to get you started on your service auditing:

TCP Connect port scan For this scan, `nmap` attempts to connect to ports using the Transmission Control Protocol (TCP) on the server. If a port is listening, the connection attempt succeeds.

TCP is a network protocol used in the TCP/IP network protocol suite. TCP is a connection-oriented protocol. Its primary purpose is to negotiate and initiate a connection using what is called a "three-way handshake." TCP sends a synchronize packet (SYN) to a remote server specifying a specific port number in the packet. The remote server receives the SYN and replies with an acknowledgment packet (SYN-ACK) to the originating computer. The original server then acknowledges (ACK) the response, and a TCP connection is officially established. This three-way handshake is often called a SYN-SYN-ACK or SYN, SYN-ACK, ACK.

If you select a TCP Connect port scan, the `nmap` utility uses this three-way handshake to do a little investigative activity on a remote server. Any services that use the TCP protocol will respond to the scan.

UDP port scan For this scan, `nmap` sends a UDP packet to every port on the system being scanned. *UDP* is another popular protocol in the TCP/IP network protocol suite. Unlike TCP, however, UDP is a *connectionless protocol*. If the port is listening and has a service that uses the UDP protocol, it responds to the scan.

Tip

Keep in mind that Free and Open Source Software (FOSS) utilities are also available to those with malicious intent. While you are doing these `nmap` scans, realize that the remote scan results that you see for your Linux server are the same scan results that others will see. This will help you evaluate your system's security settings in terms of how much information is being given out to port scans. Keep in mind that you should use tools like `nmap` only on your own systems, because scanning ports on other people's computers can give the impression that you are trying to break in.

When you run the nmap utility, it provides a handy little report with information on the system you are scanning and the ports it sees. The ports are given a "state" status. nmap reports six possible port states:

- **open**: This is the most dangerous state an nmap scan can report for a port. An open port indicates that a server has a service handling requests on this port. Think of it as a sign on the door, "Come on in! We are here to help you." Of course, if you are offering a public service, you want the port to be open.
- **closed**: A closed port is accessible, but there is no service waiting on the other side of this door. However, the scan status still indicates that there is a Linux server at this particular IP address.
- **filtered**: This is the best state to secure a port that you don't want anyone to access. It cannot be determined if a Linux server is actually at the scanned IP address. It is possible that a service could be listening on a particular port, but the firewall is blocking access to that port, effectively preventing any access to the service through the particular network interface.
- **unfiltered**: The nmap scan sees the port but cannot determine if the port is open or closed.
- **open|filtered**: The nmap scan sees the port but cannot determine if the port is open or filtered.
- **closed|filtered**: The nmap scan sees the port but cannot determine if the port is closed or filtered.

To help you better understand how to use the nmap utility, review the following example. For the purposes of building a network services list, the example nmap scans are conducted on a Fedora system. The first scan is a TCP Connect scan from the command line using the loopback address 127.0.0.1.

```
# nmap -sT 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2020-1-10 11:47 EDT
Nmap scan report for localhost (127.0.0.1)

Host is up (0.016s latency).
Not shown: 998 closed ports

PORT    STATE SERVICE
25/tcp  open  smtp
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 1.34 seconds
```

The TCP Connect nmap scan reports that two TCP ports are open and have services listening on the localhost (127.0.0.1) for requests to these ports:

- Simple Mail Transfer Protocol (SMTP) is listening at TCP port 25.
- Internet Printing Protocol (IPP) is listening at TCP port 631.

The next nmap scan is a UDP scan on the Fedora system's loopback address.

```
# nmap -sU 127.0.0.1

Starting Nmap 7.70 ( https://nmap.org ) at 2020-1-10 11:48 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00048s latency).
Not shown: 997 closed ports

PORT     STATE         SERVICE
68/udp   open|filtered dhcpc
631/udp  open|filtered ipp

Nmap done: 1 IP address (1 host up) scanned in 2.24 seconds
```

The UDP nmap scan reports that two UDP ports are open and have services listening on those ports:

- Dynamic Host Control Protocol client (dhcpc) is listening at port 68.
- Internet Printing Protocol (ipp) is listening at port 631.

Notice that port 631's IPP is listed under both nmap's TCP Connect scan and the UDP scan because the IPP service can communicate over both the TCP and the UDP protocol and thus is listed in both scans.

Using these two simple nmap scans, TCP Connect and UDP on your loopback address, you can build a list of the network services offered by your Linux server. Keep in mind that port numbers are associated with a particular protocol (TCP or UDP) and a particular network interface. For example, if you have one network interface card (NIC) on a computer that faces the Internet and another that faces a private network, you may want to offer a private service (like the CUPS service for printing) to the NIC on your private network. But you may want to filter that port (631) on the NIC that faces the Internet.

Using nmap to audit your network services advertisements

You probably want lots of people to visit your website (httpd service). You probably don't want everyone on the Internet to be capable of accessing your SMB file shares (smb service). To make sure that you are properly separating access to those two types of services, you want to be able to check what a malicious scanner can see of the services available on your public-facing network interfaces.

The idea here is to compare what your Linux server looks like from the inside versus what it looks like from the outside. If you determine that some network services are accessible that you intended to keep private, you can take steps to block access to them from external interfaces.

Tip

You may be tempted to skip the scans from inside your organization's internal network. Don't. Malicious activity often occurs by a company's own employees or by someone who has already penetrated external defenses. Again, the `nmap` utility is a great help here. To get a proper view of how your Linux server's ports are seen, you need to conduct scans from several locations. For example, a simple audit would set up scans in these places:

- On the Linux server itself
- From another server on the organization's same network
- From outside the organization's network

In the following examples, part of a simple audit is conducted. The `nmap` utility is run on a Fedora system, designated as Host-A. Host-A is the Linux server whose network services are to be protected. Host-B is a Linux server using the Linux Mint distribution and is on the same network as Host-A.

Tip

Security settings on various network components, such as the server's firewall and the company's routers, should all be considered when conducting audit scans.

For this audit example, a scan is run from Host-A, using not the loopback address but the actual IP address. First, the IP address for Host-A is determined using the `ip addr show` command. The IP address is 10.140.67.23.

```
# ip addr show
fconfig
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
     group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
     state UP group default qlen 1000
    link/ether 52:54:00:c4:27:4e brd ff:ff:ff:ff:ff:ff
    inet 10.140.67.23/24 brd 10.140.67.255 scope global dynamic
```

```
       noprefixroute ens3
       valid_lft 3277sec preferred_lft 3277sec
    inet6 fe80::5036:9ec3:2ae8:7623/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Now, using the Host-A IP address, an `nmap` TCP Connect scan is issued from Host-A. The `nmap` scan goes out to the network to conduct the scan. All ports are reported as having a status of `closed`.

```
# nmap -sT 10.140.67.23
Starting Nmap 7.80 ( https://nmap.org ) at 2020-1-31 11:53 EDT

Nmap scan report for rhel8 (10.140.67.23)

Host is up (0.010s latency).
All 1000 scanned ports on 10.140.67.23 are closed

Nmap done: 1 IP address (1 host up) scanned in 1.48 seconds
```

The `nmap` scan is moved from originating at Host-A to originating on Host-B. Now the TCP Connect scan is attempted on Host-A's ports from Host-B's command line.

```
$ nmap -sT 10.140.67.23
Starting Nmap 7.80 ( https://nmap.org ) at 2020-1-31 11:57 EDT

Note: Host seems down. If it is really up,
 but blocking our ping probes, try -PN

Nmap done: 1 IP address (0 hosts up) scanned in 0.11 seconds
```

Here, `nmap` gives a helpful hint. Host-A appears to be down, or it could just be blocking the probes. So, another `nmap` scan is attempted from Host-B, using `nmap`'s advice of disabling the scan's `ping` probes via the `-PN` option.

```
$ nmap -sT -PN 10.140.67.23
Starting Nmap 7.80 ( https://nmap.org ) at 2020-1-31 11:58 EDT
Nmap scan report for rhel8 (10.140.67.23)

Host is up (0.0015s latency).
All 1000 scanned ports on 10.140.67.23 are filtered

Nmap done: 1 IP address (1 host up) scanned in 5.54 seconds
```

You can see that Host-A (`10.140.67.23`) is up and running and all of its ports have a status of `filtered`. This means that there is a firewall in place on Host-A. These scans from Host-B give you a better idea of what a malicious scanner may see when scanning your Linux server. In this example, the malicious scanner would not see much.

Note

If you are familiar with `nmap`, you know that the TCP SYN scan is the default scan `nmap` uses. The TCP SYN scan does an excellent job of probing a remote system in a stealth manner. Because you are probing your own system for security auditing purposes, it makes sense to use the more "heavy-duty" `nmap` utility scans. If you still want to use the TCP SYN scan, the command is `nmap -sS ip _ address`.

The services currently running on Host-A are not that "juicy." In the example that follows, another service, `sshd`, is started on Host-A using the `systemctl` command (see Chapter 15, "Starting and Stopping Services"). This should give the `nmap` utility a more interesting target to search for.

```
# systemctl start sshd.service
# systemctl status sshd.service
• sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled;
vendor preset: enabled)
   Active: active (running) since Fri 2020-1-30 15:08:29 EDT; 1 day
20h ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 807 (sshd)
    Tasks: 1 (limit: 12244)
   Memory: 10.9M
   CGroup: /system.slice/sshd.service
           └─807 /usr/sbin/sshd -D -oCiphers=...
```

Also, because Host-A's firewall is blocking the `nmap` scans from Host-B, it would be interesting to see what an `nmap` scan can report when the firewall is down. The example that follows shows the firewall being disabled on Host-A for a Fedora 21 or RHEL 7 system (for other systems, you probably need to disable the iptables service):

```
# systemctl stop firewalld.service
# systemctl status firewalld.service
```

With a new service running and Host-A's firewall lowered, the `nmap` scans should find something. In the following, `nmap` scans are run again from Host-B. This time the `nmap` utility shows the `ssh` service running on open port 22. Notice that with the firewall down on Host-A, both `nmap` scans pick up much more information. This really demonstrates the importance of your Linux server's firewall.

```
# nmap -sT 10.140.67.23
Starting Nmap 7.80 ( http://nmap.org ) at 2020-1-31 11:58 EDT
Nmap scan report for 10.140.67.23
Host is up (0.016s latency).
Not shown: 999 closed ports
```

```
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.40 seconds

# nmap -sU 10.140.67.23
[sudo] password for johndoe: ***************
Starting Nmap 5.21 ( http://nmap.org ) at 2020-1-31 11:59 EDT
Nmap scan report for 10.140.67.23
Host is up (0.00072s latency).
Not shown: 997 closed ports

PORT     STATE         SERVICE
68/udp   open|filtered dhcpc
631/udp  open|filtered ipp
...
Nmap done: 1 IP address (1 host up) scanned in 1081.83 seconds
```

In order to conduct a thorough audit, be sure to include the UDP scan. Also, there are additional `nmap` scans that may be beneficial to your organization. Look at the `nmap` utility's website for additional suggestions.

CAUTION

If you have been following along and lowered your server's firewall to conduct these `nmap` scans, be sure to raise it again. Enter `systemctl start firewalld.service`.

You still need to implement controls for those services that your Linux server should offer. One way to accomplish this is via firewall rules.

Early versions of Linux use TCP wrappers to allow or deny access to Linux services. It did this by offering `/etc/hosts.allow` and `/etc/hosts.deny` files where you could specifically indicate which services are available and which are blocked to particular outside system names and/or IP addresses. As of Fedora 28 and RHEL 8, the TCP wrappers feature was dropped from those distributions. However, some features, such as vsftpd, still honor those configuration files through other means.

Working with Firewalls

A firewall in a building is a fireproof wall that prevents the spread of fire throughout the building. A computer *firewall* blocks the transmission of malicious or unwanted data into and out of a computer system or network. For example, a firewall can block malicious scans from your Linux server ports. A firewall can also change network packets flowing through your system and redirect packets in various ways.

In Linux, *iptables* is the kernel-level firewall feature. It is most commonly used to allow or block access from outside systems to the services running on your local system. iptables works by allowing you to create rules that can be applied to every packet that tries to enter (`INPUT`), leave (`OUTPUT`), or cross through your system (`FORWARD`).

Although allowing or blocking packets trying to enter your system is the primary feature of iptables, you can also create rules for iptables that let you do the following:

- Block packets leaving your system effectively to prevent a process on your system from reaching a remote host, range of addresses, or selected services.
- Forward packets from one network interface on your system to another, effectively allowing your computer to act as a router between two networks.
- Port forward a packet intended for a selected port to be rerouted to another port on your local system, or to a remote system, so that other locations can handle the request from the packet.
- Change information in a packet header (called *mangling*) to redirect the packet or somehow mark it for more processing.
- Allow multiple computers on a private network (such as the computers, televisions, or other devices on your home network) to communicate with the Internet over a single public IP address. (This is referred to as *IP masquerading*.)

In the following sections, I describe many of these features but focus mostly on the rules to block or allow access to the services running on your Linux system.

Understanding firewalls

Although you may tend to think of a firewall as a complete barrier, a Linux firewall is really just a filter that checks each network packet or application request coming into or out of a computer system or network.

Note

What is a network packet? A *network packet* is data that has been broken up into transmittable chunks. The chunks, or packets, have additional data added to them as they traverse down the OSI model. It's like putting a letter inside an envelope at each stage as it moves down the protocol stack. One of the purposes of this additional data is to ensure the packet's safe and intact arrival at its destination. The additional data is stripped off of the packet as it traverses back up the OSI model at its destination (like taking off the outer envelope and handing the letter to the layer above).

Firewalls can be placed into different categories, depending upon their function. Each category has an important place in securing your server and network.

A firewall is either network-based or host-based. A network-based firewall is one that is protecting the entire network or subnet. For example, a network firewall would be used in your workplace, where the network should be protected by a screening router's firewall.

A host-based firewall is one that is running on and protecting an individual host or server. You most likely have a firewall on your PC at home. This is a host-based firewall.

A firewall is either a hardware or a software firewall. Firewalls can be located on network devices, such as routers. Their filters are configured in the router's firmware. In your home, your Internet service provider (ISP) may provide a router to let you gain access to the Internet. The router contains firewall firmware, and it is considered a hardware firewall.

Firewalls can be located on a computer system as an application. The application allows filtering rules to be set that filter the incoming traffic. This is an example of a software firewall. A software firewall is also called a rule-based firewall.

A firewall is either a network-layer filter or an application-layer filter. A firewall that examines individual network packets is also called a *packet filter*. A *network-layer firewall* allows only certain packets into and out of the system. It operates on the lower layers of the OSI reference model.

An *application-layer firewall* filters at the higher layers of the OSI reference model. This firewall allows only certain applications access to and from the system.

You can see how these firewall categories overlap. The best firewall setup is a combination of all of the categories. As with many security practices, the more layers you have, the harder it is for malicious activity to penetrate.

Implementing firewalls

On a Linux system, the firewall is a host-based, network-layer, software firewall managed by the `iptables` utility and related kernel-level components. With `iptables`, you can create a series of rules for every network packet coming through your Linux server. You can fine-tune the rules to allow network traffic from one location but not from another. These rules essentially make up a network access control list for your Linux server.

Fedora, RHEL, and other Linux distributions have added the `firewalld` service to provide a more dynamic way of managing firewall rules than were offered previously. For recent RHEL and Fedora releases, the iptables firewall backend was replaced with nftables. The Firewall Configuration window (`firewall-config` command) provides an easy way to open ports on your firewall and do masquerading (routing private addresses to a public network) or port forwarding. The `firewalld` service can react to changes in conditions, which the static iptables service can't do as well on its own. By enabling access to a service, `firewalld` can also do things like load modules needed to allow access to a service.

Tip

The `iptables` utility manages the Linux firewall, called `netfilter`. Thus, you will often see the Linux firewall referred to as `netfilter/iptables`. The iptables syntax is still supported, but in the latest RHEL and Fedora releases, nftables actually provides the backend for iptables.

Starting with firewalld

The `firewalld` service may already be installed on your Linux system. To check this, type the following:

```
# systemctl status firewalld
• firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; ena>
   Active: active (running) since Sat 2019-10-19 11:43:13 EDT; 5m>
     Docs: man:firewalld(1)
 Main PID: 776 (firewalld)
    Tasks: 2 (limit: 2294)
   Memory: 39.6M
   CGroup: /system.slice/firewalld.service
           └─776 /usr/bin/python3 /usr/sbin/firewalld --nofork -->
```

If it's not, you can still install the service and the associated graphical user interface and then start the `firewalld` service as follows:

```
# yum install firewalld firewall-config
# systemctl start firewalld.service
# systemctl enable firewalld.service
```

To manage the `firewalld` service, you can start the Firewall Configuration window. Do this by entering the following:

```
# firewall-config &
```

Figure 25.1 shows an example of the Firewall Configuration window.

With `firewalld`, you can select from a set of firewall zones, depending on which services you want to share and the level of protection you want for your system. The default set of Fedora Workstation rules selected in this example is appropriate for a Linux workstation operating on a home network. For example, it allows the following:

DHCPv6 Client: To enable automatic assignment of addresses on IPv6 networks.

Multicast DNS (mDNS): To allow domain name system interfaces on small network interfaces, without requiring a regular DNS server.

Network printing client and server (IPP): To allow printer sharing on your local system and network.

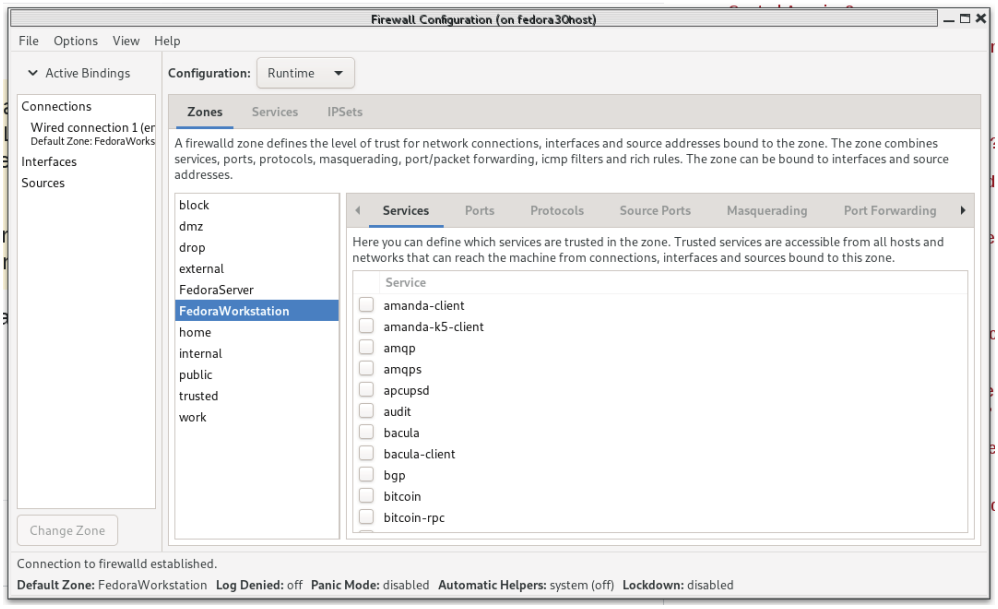
Samba client: To allow file sharing with Windows systems and other systems on your local network.

SSH: To allow others to try to log into your system from the network.

Cockpit: To allow access to Cockpit web-based administration from the network. Cockpit is installed by default in RHEL 8, but it is not installed by default on Fedora 30 Workstation. So, Cockpit won't appear in the Firewall Configuration window until you install the cockpit package.

FIGURE 25.1

Firewall Configuration window



If you connect your computer to networks on which you have different levels of trust (such as a wireless network at an airport), you can adjust your firewall rules by selecting a different zone. For example, to change to the public zone from the Firewall Configuration window, do the following:

1. Under the Active Bindings column, select your active connection (in this example, `Wired connection 1`).
2. Select a new zone (for example, `public`).
3. Select Change Zone.

The `public` zone, while still allowing IPv6 connections, remote login (SSH), and mDNS service, does not allow access to more potentially vulnerable printing, Windows file sharing (Samba), and Cockpit services.

Besides changing zones, another common task that you might want to do is just open some firewall ports to allow access to selected services. From the Firewall Configuration window, with the Fedora Workstation zone set as the current zone, just click each service that you want to open. The port allowing access to each service is opened immediately (when you select the Runtime configuration) and opened permanently (when you select the Permanent configuration).

One nice feature of the Firewall Configuration window is that when you choose to allow access to a service, you might do more than just open a port. For example, enabling the FTP service also causes connection tracking modules to be loaded that allow nonstandard ports to be accessed through the firewall when needed.

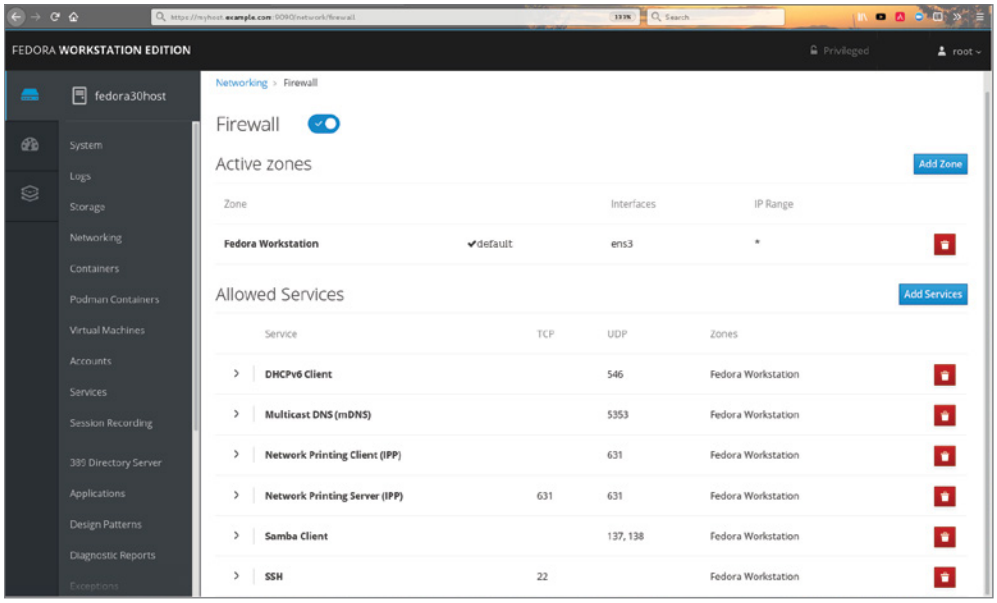
Changing firewall rules with Cockpit

Cockpit offers another intuitive way of working with your system's firewall. To view and modify your firewall with Cockpit, do the following:

1. Open your web browser to the Cockpit interface (`https://yourhost:9090`) and log in with root privilege.
2. Select Networking ➪ Firewall to see the Firewall screen, as shown in Figure 25.2:

FIGURE 25.2

Firewall Configuration



3. Select Add Services. The Add Service pop-up appears.
4. Click the check box next to the service that you want to enable in the current zone and select Add Services.

Access to the selected port is enabled. Assuming that you have a service running on that port, someone requesting the service (such as access to your web server on ports 80 and 443) will be allowed.

As mentioned earlier, underlying the Cockpit and `firewalld` services is the iptables facility. If you have a Linux system without the Cockpit or `firewalld` services (or with `firewalld` disabled), you can still use the iptables service. The next sections describe how you can set iptables firewall rules manually and use the iptables service directly, without the `firewalld` service.

Understanding the iptables utility

Before you start changing the firewall rules via the `iptables` utility, you need to understand `netfilter/iptables` basics, which include the following:

- Tables
- Chains
- Policies
- Rules

Understanding these basics will help you set up and manage your Linux server firewall properly.

netfilter/iptables tables

The `iptables` firewall has the ability to do more than just low-level packet filtering. It defines what type of firewall functionality is taking place. There are four tables in the `iptables` utility, with an additional table added by SELinux. The tables offer the following functionalities:

`filter`: The `filter` table is the packet filtering feature of the firewall. In this table, access control decisions are made for packets traveling to, from, and through your Linux system.

`nat`: The `nat` table is used for Network Address Translation (NAT). NAT table rules let you redirect where a packet goes.

`mangle`: As you would suspect, packets are mangled (modified) according to the rules in the `mangle` table. Using the `mangle` table directly is less common and typically done to change how a packet is managed.

`raw`: The `raw` table is used to exempt certain network packets from something called "connection tracking." This feature is important when you are using Network Address Translation and virtualization on your Linux server.

`security`: This table is available only on Linux distributions that have SELinux. (See Chapter 24, "Enhancing Linux Security with SELinux.") Although typically not used directly, the security table allows SELinux to allow or block a packet based on SELinux policies, adding another layer of filtering on top of standard packet filter rules.

Of all the tables listed, three focus on Network Address Translation. Therefore, the `filter` table is the primary table that this chapter focuses on for basic firewall packet filtering.

netfilter/iptables chains

The `netfilter/iptables` firewall categorizes network packets into categories, called *chains*. There are five chains (categories) to which a network packet can be designated:

`INPUT`: Network packets coming *into* the Linux server

`FORWARD`: Network packets coming into the Linux server that are to be *routed* out through another network interface on the server

`OUTPUT`: Network packets coming *out* of the Linux server

`PREROUTING`: Used by NAT for modifying network packets when they come into the Linux server

`POSTROUTING`: Used by NAT for modifying network packets before they come out of the Linux server

Which `netfilter/iptables` table you choose to work with determines what chains are available for categorizing network packets. Table 25.1 shows what chains are available for each table.

TABLE 25.1 Chains Available for Each netfilter/iptables Table

Table	Chains Available
`filter`	`INPUT, FORWARD, OUTPUT`
`nat`	`PREROUTING, OUTPUT, POSTROUTING`
`mangle`	`INPUT, FORWARD, PREROUTING, OUTPUT, POSTROUTING`
`raw`	`PREROUTING, OUTPUT`
`security`	`INPUT, FORWARD, OUTPUT`

After a network packet is categorized into a specific chain, `iptables` can determine what policies or rules apply to that particular packet.

netfilter/iptables rules, policies, and targets

For each network packet, a rule can be set up defining what to do with that individual packet. Network packets can be identified many ways by the `netfilter/iptables` firewall. These are a few of the ways:

- Source IP address
- Destination IP address
- Network protocol
- Inbound port
- Outbound port
- Network state

If no rule exists for a particular packet, then the overall policy is used. Each packet category or chain has a default policy. After a network packet matches a particular rule or falls to the default policy, then action on the packet can occur. The action taken depends upon what `iptables` target is set. Here are a couple of actions (targets) that can be taken:

ACCEPT: Network packet is accepted into the server.

REJECT: Network packet is dropped and not allowed into the server. A rejection message is sent.

DROP: Network packet is dropped and not allowed into the server. No rejection message is sent.

While `REJECT` gives a rejection message, `DROP` is quiet. You may consider using `REJECT` for internal employees who should be told that you are rejecting their outbound network traffic and why. Consider using `DROP` for inbound traffic so that any malicious personnel are unaware that their traffic is being blocked.

TIP

There are a couple of additional, more sophisticated targets for `iptables`, such as `QUEUE`. You can find out more about these targets via the `iptables` man page.

The `iptables` utility implements a software firewall using the `filter` table via policies and rules. Now that you have a general understanding of the software firewall implementation, you can begin to dig deeper into the specific commands for implementing the firewall via the `iptables` utility.

Using the iptables utility

Your Linux server should come with the firewall up and running. However, it's a good idea to check and see if it really is enabled.

RHEL 7, RHEL 8, and recent Fedora systems netfilter/iptables firewall The firewall interface service running on these distributions is `firewalld`. The `iptables` service is not run directly by default on these systems. To see if this firewall service is running, type **`systemctl status firewalld.service`** at the command line.

- To enable the firewall, enter **`systemctl start firewalld.service`** and **`systemctl enable firewalld.service`** at the command line.
- To disable the firewall, enter **`systemctl stop firewalld.service`** at the command line.

Ubuntu netfilter/iptables firewall The firewall interface service running on this distribution is `ufw`. To see if the firewall service is running, enter **`sudo ufw status`** at the command line. The `ufw` service is an interface to the `iptables` utility that does not run as a service on Ubuntu. You can use `ufw` commands to manipulate

firewall rules. However, all of the `iptables` utility commands are still valid for Ubuntu:

- To enable the firewall, enter **`sudo ufw enable`** at the command line.
- To disable the firewall, enter **`sudo ufw disable`** at the command line.

After you have checked the status and enabled or disabled the `netfilter/iptables` firewall, the differences between the distributions end.

To see what policies and rules are currently in place for the `filter` (default) table, enter **`iptables -vnL`** at the command line:

```
# iptables -vnL
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)...
```

Note that on systems with `firewalld` enabled, there are many more iptables chains and rules listed by default than you might be used to on a system using iptables directly. This is done to offer more flexibility in building your firewalls by allowing your rules to be split into zones for different levels of security.

Only the first line of the iptables output is shown in the preceding example. That line shows that the `INPUT` chain's default policy is applied to all the network packets that don't match another rule. Currently, all of the default `INPUT`, `FORWARD`, and `OUTPUT` policies are set to `ACCEPT`. All network packets are allowed in, through, and out. A firewall in this state is essentially disabled until specific `REJECT` or `DROP` rules are added.

Tip

If your Linux server is dealing with IP v6 network packets, you can use the `ip6tables` utility to manage your firewall for IPv6 addresses. The `ip6tables` utility is nearly identical to the `iptables` utility. For more information, enter **`man ip6tables`** at the command line.

Modifying iptables policies and rules

Before you begin to modify a `netfilter/iptables` firewall directly by using the `iptables` command, you should go on a system that you can use for testing and turn off the `firewalld` service.

To get started, it is helpful to understand a few command options.

A few options for modifying the firewall follow:

`-t` *`table`*

The `iptables` command listed along with this switch is applied to the *`table`*. By default, the `filter` table is used. Example:

```
# iptables -t filter -P OUTPUT DROP
```

`-P chain target`

Sets the overall policy for a particular *chain*. The rules in the *chain* are checked for matches. If no match occurs, then the *chain*'s listed `target` is used. Example:

```
# iptables -P INPUT ACCEPT
```

`-A chain`

Sets a rule called an "appended rule," which is an exception to the overall policy for the *chain* designated. Example:

```
# iptables -A OUTPUT -d 10.140.67.25 -j REJECT
```

`-I rule# chain`

Inserts an appended rule into a specific location, designated by the *rule#*, in the appended rule list for the *chain* designated. Example:

```
# iptables -I 5 INPUT -s 10.140.67.23 -j DROP
```

`-D chain rule#`

Deletes a particular rule, designated by the *rule#*, from the *chain* designated. Example:

```
# iptables -D INPUT 5
```

`-j target`

If the criteria in the rule are met, the firewall should jump to this designated *target* for processing. Example:

```
# iptables -A INPUT -s 10.140.67.25 -j DROP
```

`-d IP address`

Assigns the rule listed to apply to the designated destination *IP address*. Example:

```
# iptables -A OUTPUT -d 10.140.67.25 -j REJECT
```

`-s IP address`

Assigns the rule listed to apply to the designated source *IP address*. Example:

```
# iptables -A INPUT -s 10.140.67.24 -j ACCEPT
```

`-p protocol`

Assigns the rule listed to apply to the *protocol* designated. For example, here incoming ping (`icmp`) requests are dropped:

```
# iptables -A INPUT -p icmp -j DROP
```

`--dport port#`

Assigns the rule listed to apply to certain protocol packets coming into the designated *port#*. Example:

```
# iptables -A INPUT -p tcp --dport 22 -j DROP
```

`--sport` *port#*

Assigns the rule listed to apply to certain protocol packets going out of the designated *port#*. Example:

```
# iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
```

`-m state --state` *network _ state*

Assigns the rule listed to apply to the designated *network state*(s). Example:

```
# iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

To see how the `iptables` options work, consider the following example. You have a Linux server (Host-A) at IP address 10.140.67.23. There are two other Linux servers on your network. One is Host-B at IP address 10.140.67.22 and the other is Host-C at IP address 10.140.67.25. Your goal is to accomplish the following:

- Allow Host-C full access to Host-A.
- Block remote login connections using `ssh` from Host-B to Host-A.

Setting a policy of Drop

The following code shows the default policies of Host-A's firewall. In this example, the firewall is wide open with no restrictions implemented. No rules are set, and the policies are all set to `ACCEPT`.

```
# iptables -vnL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

First, what would happen if the `INPUT` policy was changed from `ACCEPT` to `DROP`? Would that reach the goal? Look at what happens when this is tried. Remember that if no rules are listed for an incoming packet, then the `chain`'s `policy` is followed. This change is made to Host-A's firewall in the example that follows.

```
# iptables -P INPUT DROP
# iptables -vnL

Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
```

```
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Tip

For policies, you cannot set the target to `REJECT`. It fails, and you receive the message "iptables: Bad policy name." Use `DROP` as your policy instead.

Host-B attempts to `ping` Host-A and then attempts an `ssh` connection, as shown in the example that follows. As you can see, both attempts fail. Because `ping` commands are blocked, this does not meet the objective to block only remote login connections using `ssh` from Host-B.

```
$ ping -c 2 10.140.67.23
PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.

--- 10.140.67.23 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms
$ ssh root@10.140.67.23

ssh: connect to host 10.140.67.23 port 22: Connection timed out
```

When Host-C attempts to `ping` Host-A and make an `ssh` connection, both attempts fail. Thus, it is confirmed that the firewall setting, `INPUT` policy equals `DROP`, is not what is needed to reach the goal.

```
$ ping -c 2 10.140.67.23
PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.

--- 10.140.67.23 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
$ ssh root@10.140.67.23

ssh: connect to host 10.140.67.23 port 22: Connection timed out
```

Blocking a source IP address

What if instead only Host-B's IP address were blocked? That would allow Host-C to reach Host-A. Would this setting reach the desired goal?

In the example that follows, the policy of `DROP` must first be changed to `ALLOW` in Host-A's `iptables`. After that, a specific rule must be appended to block network packets from Host-B's IP address, 10.140.67.22, alone.

```
# iptables -P INPUT ACCEPT
# iptables -A INPUT -s 10.140.67.22 -j DROP
# iptables -vnL
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  10.140.67.22             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Host-C can now successfully `ping` and `ssh` into Host-A, meeting one of the set goals.

```
$ ping -c 2 10.140.67.23
PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.
64 bytes from 10.140.67.23: icmp_req=1 ttl=64 time=11.7 ms
64 bytes from 10.140.67.23: icmp_req=2 ttl=64 time=0.000 ms

--- 10.140.67.23 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.000/5.824/11.648/5.824 ms
$ ssh root@10.140.67.23
root@10.140.67.23's password:
```

However, Host-B can neither `ping` nor `ssh` into Host-A. Thus, the appended rule is not quite what is needed to reach the entire goal.

```
$ ping -c 2 10.140.67.23

PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.

--- 10.140.67.23 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms

$ ssh root@10.140.67.23

ssh: connect to host 10.140.67.23 port 22: Connection timed out
```

Blocking a protocol and port

What if, instead of blocking Host-B's IP address entirely, only connections to the `ssh` port (port 22) from Host-B's IP address were blocked? Would that reach the goal of allowing Host-C full access to Host-A and only blocking `ssh` connections from Host-B?

In the example that follows, the `iptables` rules for Host-A are modified to try blocking Host-B's IP address from port 22. Note that the `--dport` option must accompany a particular protocol, such as, for example, `-p tcp`. Before the new rule is added, the rule from the previous example must be deleted using the `-D` option. Otherwise, the rule from the previous example would be used by the `netfilter/iptables` firewall for packets from 10.140.67.22 (Host-B).

```
# iptables -D INPUT 1
# iptables -A INPUT -s 10.140.67.22 -p tcp --dport 22 -j DROP
# iptables -vnL

Chain INPUT (policy ACCEPT)
target     prot opt source      destination
DROP       tcp  --  10.140.67.22    anywhere     tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source      destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source      destination
```

First, the new `iptables` rule is tested from Host-C to ensure that both `ping` attempts and `ssh` connections remain unaffected. It works successfully.

```
$ ping -c 2 10.140.67.23
PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.
64 bytes from 10.140.67.23: icmp_req=1 ttl=64 time=1.04 ms
64 bytes from 10.140.67.23: icmp_req=2 ttl=64 time=0.740 ms

--- 10.140.67.23 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.740/0.892/1.045/0.155 ms

$ ssh root@10.140.67.23
root@10.140.67.23's password:
```

Next, the new `iptables` rule is tested from Host-B to ensure that `ping` works and `ssh` connections are blocked. It also works successfully!

```
$ ping -c 2 10.140.67.23

PING 10.140.67.23 (10.140.67.23) 56(84) bytes of data.
64 bytes from 10.140.67.23: icmp_req=1 ttl=64 time=1.10 ms
64 bytes from 10.140.67.23: icmp_req=2 ttl=64 time=0.781 ms

--- 10.140.67.23 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.781/0.942/1.104/0.164 ms

$ ssh root@10.140.67.23

ssh: connect to host 10.140.67.23 port 22: Connection timed out
```

Again, your organization's Access Control Matrix (see Chapter 22, "Understanding Basic Linux Security") helps you in creating the necessary rules for the `netfilter/iptables`

firewall on your Linux server. Then each modification should be tested in a test or virtual environment before implementing it in your production Linux systems firewall.

Saving an iptables configuration

Because `firewalld` is the recommended service for creating firewalls in RHEL, Fedora, and other Linux systems, manual creation of permanent firewall rules is less common. However, if you like, you can still manually save and restore firewall rules that you create directly with `iptables`.

In the example that follows, the modifications made earlier are still in the firewall. You can save the current set of firewall filter rules using the `iptables-save` command.

```
# iptables -vnL
Chain INPUT (policy ACCEPT 8 packets, 560 bytes)
 pkts bytes target prot opt in  out source        destination
    0     0 DROP   tcp  --  *   *   10.140.67.22  0.0.0.0/0   tcp dpt:22
    0     0 DROP   tcp  --  *   *   0.0.0.0/0     0.0.0.0/0   tcp dpt:33
    0     0 DROP   icmp --  *   *   0.0.0.0/0     0.0.0.0/0
...

# iptables-save > /tmp/myiptables
```

To restore those rules later, you can start by flushing the current rules (`iptables -F`) and restoring them (`iptables-restore`).

```
# iptables -F
# iptables -vnL
Chain INPUT (policy ACCEPT 8 packets, 560 bytes)
 pkts bytes target prot opt in out source       destination
    0     0 DROP   tcp  --  *  *   0.0.0.0/0    0.0.0.0/0  tcp dpt:33
    0     0 DROP   icmp --  *  *   0.0.0.0/0    0.0.0.0/0
...
```

A flush of the rules does not affect the `iptables` configuration file. To restore the firewall to its original condition, use the `iptables-restore` command. In the example that follows, the `iptables` configuration file is redirected into the `restore` command and the original `DROP` rule for `10.140.67.22` is restored.

```
# iptables-restore < /tmp/myiptables
# iptables -vnL
Chain INPUT (policy ACCEPT 16 packets, 1120 bytes)
 pkts bytes target prot opt in out source       destination
    0     0 DROP   tcp  --  *  *   10.140.67.22 0.0.0.0/0      tcp dpt:22
    0     0 DROP   tcp  --  *  *   0.0.0.0/0    0.0.0.0/0      tcp dpt:33
    0     0 DROP   icmp --  *  *   0.0.0.0/0    0.0.0.0/0
```

Note

For an Ubuntu system, the way of saving and restoring your `netfilter/iptables` modifications are very similar to the way it is done in Fedora. You can still use the `iptables-save` command to create an `iptables` configuration file from the current `iptables` setting and use `iptables-restore` to restore it. There are several options for loading a configuration file upon system boot. See the Ubuntu community website at `https://help.ubuntu.com/community/IptablesHowTo` for the various options.

You can also save your `netfilter/iptables` firewall rules to create an audit report. Reviewing these rules periodically should be part of your organization's System Life Cycle Audit/Review phase.

Summary

Securing your Linux server is critical on a network. Inherently, a majority of the malicious attacks originate from a network, especially the Internet. This chapter covered some of the basics that you need in order to get started on this process.

Protecting your network services can be simplified after you determine and remove any unneeded network services. The `nmap` utility helps you here. Also, you can use `nmap` to audit your Linux server's advertising of network services. These audits assist in determining what firewall modifications are needed.

Recent versions of Fedora and RHEL have added the `firewalld` service as a front end to the `iptables` firewall facility that is built into the Linux kernel. By using the `firewalld-config` tool and Cockpit web UI, you can easily open ports in your firewall to allow access to selected services. The `netfilter/iptables` firewall facility is a host-based, network-layer, software firewall. It is managed by the `iptables` and `ip6tables` utilities. With these utilities, a series of policies and rules can be created for every network packet coming through your Linux server.

At this point in this book, you should have a good grasp of what goes into setting up and securing Linux desktop and server systems. In the next two chapters, I'm going to help you extend that knowledge into cloud computing and virtualization.

Exercises

Refer to the material in this chapter to complete the tasks that follow. If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, you can often complete a task in multiple ways). Try each of the exercises before referring to the answers. These tasks assume you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). Please don't use a production system to try out the `iptables` commands in these exercises. Although the commands shown here do

not permanently change your firewall (the old rules will return when the firewall service restarts), improperly modifying your firewall can result in unwanted access.

1. Install the Network Mapper utility on your local Linux system.
2. Run a TCP Connect scan on your local loopback address. What ports have a service running on them?
3. Run a UDP port scan on your Linux system from a remote system.
4. Check to see if your system is running the `firewalld` service. If not, install `firewalld` and `firewall-config` and then start and enable that service.
5. Use the Firewall Configuration window to open access to secure (TCP port 443) and insecure (TCP port 80) ports for a web service.
6. Determine your Linux system's current `netfilter/iptables` firewall policies and rules.
7. Save your Linux system's current firewall rules, flush them, and then restore them.
8. For your Linux system's firewall, set a filter table policy for the input chain to `DROP`.
9. Change your Linux system firewall's filter table policy back to `accept` for the input chain, and then add a rule to drop all network packets from the IP address 10.140.67.23.
10. Without flushing or restoring your Linux system firewall's rules, remove the rule you just added.

Part VI

Engaging with Cloud Computing

IN THIS PART

Chapter 26
Shifting to Clouds and Containers

Chapter 27
Using Linux for Cloud Computing

Chapter 28
Deploying Linux to the Cloud

Chapter 29
Automating Apps and Infrastructure with Ansible

Chapter 30
Deploying Applications as Containers with Kubernetes

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

CHAPTER

26

Shifting to Clouds and Containers

IN THIS CHAPTER

Understanding key technologies for cloud computing

Learning how Linux containers work

Installing and starting container software

Pulling and running container images

Restarting a stopped container

Building a container image

Tagging and pushing container images to a registry

While most of this book focuses on installing and managing individual computers, services, and applications, this part takes you into the technologies needed to bring Linux into large data centers. For a data center to operate efficiently, its computers must become as generic as possible and running components must become more automated. Chapters in this part focus on technologies that make those two things happen.

Computers become more generic by separating the applications from the operating systems. This means not just packaging applications into things you install on an operating system (like RPM or Deb packages), but also putting together sets of software into packages that themselves can run once they are delivered in ways that keep them separate from the operating system. *Virtual machines* (VMs) and *containers* are two ways of packaging sets of software and their dependencies in ways that are ready to run.

From a high level, a *virtual machine* is a complete operating system that runs on another operating system, allowing you to have many VMs active at a time on one physical computer. Everything an application or a service needs to run can be stored within that VM or in attached storage.

A VM has its own kernel, file system, process table, network interfaces, and other operating system features separate from the host, while sharing the CPU and RAM with the host system. You can deploy that VM to a physical system in a way that makes it easy to run the application and then discard the VM when you are done. You can run multiple instances of the VM on the same computer

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

or clone and run the VM across multiple computers. The term *virtual machine* comes from the fact that each VM sees an emulation of computer hardware and not the hardware itself directly.

A *container* is like a VM, with the major difference being that a container doesn't have its own kernel. In most other ways, it is like a VM in that its name spaces are separate from the host operating system and you can move it from host to host to run wherever it is convenient.

The chapters in this part introduce you to the concepts, tools, and technologies that you need to know to engage with cloud computing. You can try out virtual machines on a single Linux host using KVM. You can then deploy virtual machines to cloud technologies such as OpenStack and Amazon Web Services (AWS).

To deploy sets of hosts, either on bare metal or the cloud, you will learn how to use Ansible. With Ansible playbooks, you can also define the software that is installed and run on each host system.

As for containers, the Kubernetes project has grabbed the spotlight as the premier technology for orchestrating massive numbers of containers across large data centers. Products such as Red Hat OpenShift provide supported Kubernetes platforms for large enterprises.

The technology that started the rush to containers a few years ago was the Docker project. The `docker` command and daemon offered simplified ways to build and run containers on Linux systems. Today, standardized container formats (such as the Open Container Initiative) and other container tools, such as `podman`, offer ways of working with containers that align more tightly with the Kubernetes ecosystem.

The remainder of this chapter is devoted to getting started with containers. It covers the `docker` and `podman` commands, along with other popular tools for working with individual containers.

Understanding Linux Containers

Containers make it simple to get and run applications and then discard them when you are done. There are a few things that you should know about containers before you get started.

In working with containers, people refer to the entity that you move around as a *container image* (or simply an *image*). When you run that image, or when it is paused or stopped, it is referred to as a *container*.

A container remains separate from the host system by using its own set of *namespaces*. You typically would build your own container images by getting a secure *base image* and then adding your own layers of software on top of that image to create a new image. To share your images, you push them to shared *container registries* and allow others to pull them.

Namespaces

Linux support for *namespaces* is what allows containers to be contained. With namespaces, the Linux kernel can associate one or more processes with a set of resources. Normal processes, not those run in a container, all use the same host namespaces. By default, processes in a container only see the container's namespaces and not those of the host. Namespaces include the following:

Process table A container has its own set of process IDs and, by default, can only see processes running inside the container. While PID 1 on the host is the `init` process (`systemd`), in a container PID 1 is the first process run inside the container.

Network interfaces By default, a container has a single network interface (`eth0`) and is assigned an IP address when the container runs. By default, a service run inside a container (such as a web server listening on ports 80 and 443) is not exposed outside of the host system. The upside of this is that you could have hundreds of web servers running on the same host without conflict. The downside is that you need to manage how those ports are exposed outside of the host.

Mount table By default, a container can't see the host's root file system, or any other mounted file system listed in the host's mount table. The container brings its own filesystem, consisting of the application and any dependencies it needs to run. Files or directories needed from the host can be selectively bind-mounted inside the container.

User IDs Although containerized processes run as some UID within the host's namespace, another set of UIDs is nested within the container. This can, for example, let a process run as root within a container but not have any special privileges to the host system.

UTS A UTS namespace allows a containerized process to have a different host and domain name from the host.

Control group (cgroup) In some Linux systems (such as Fedora and RHEL), a containerized process runs within a selected control group and cannot see the other cgroups available on the host system. Likewise, it cannot see the identity of its own cgroup.

Interprocess communications (IPC) A containerized process cannot see the IPC namespace from the host.

Although access to any host namespace is restricted by default, privileges to host namespaces can be opened selectively. In that way, you can do things like mount configuration files or data inside the container and map container ports to host ports to expose them outside of the host.

Container registries

Permanent storage for containers is done in what is referred to as a *container registry*. When you create a container image that you want to share, you can *push* that image to a public

registry or a private registry that you maintain yourself (such as a Red Hat Quay registry). Someone who wants to use the image will *pull* it from the registry.

There are large, public container image registries, such as the Docker Hub (docker.io) and Quay Registry (Quay.io). They offer free accounts to get started. If you want access to more features, such as the ability to keep your registry private, premium accounts are available as well.

Base images and layers

Although you can create containers from scratch, most often a container is built by starting with a well-known base image and adding software to it. That base image typically aligns with the operating system from which you are installing software into your container.

You can get official base images from Ubuntu (`https://hub.docker.com/ _ /ubuntu`), CentOS (`https://hub.docker.com/ _ /centos`), Fedora (`https://hub.docker.com/ _ /fedora`), and many other Linux distributions. Those Linux distributions may offer base images in different forms, such as standard and minimal versions. In fact, there are base images that you can build on that offer runtimes for php, Perl, Java, and other development environments.

Although Red Hat offers a subscription model for its software, if you want to use Red Hat software as the foundation for your container images, Red Hat offers freely available Universal Base Images (UBIs) for standard, minimal, and a variety of runtime containers. You can find those images by searching the Red Hat Container Catalog for UBI images (`https://catalog.redhat.com/software/containers/explore`).

You can add software to a base image using commands such as `docker build` or `podman`. By using a Dockerfile to define the build, you can add `yum` or `apt-get` commands to install software from software repositories into your new container.

When you add software to an image, it creates a new layer to become part of the new image. Reusing the same base images for the containers that you build offers several advantages. One advantage is that when you run the container image, only one copy of the base image is needed on the host. So, if you were running 10 different containers based on the same base image, you only need to pull and store the base image once, then possibly only add a few megabytes of extra data for each new image.

If you look at the contents of a base image, it would look like a little Linux filesystem. You see configuration files in `/etc`, executables in `/bin` and `/sbin`, and libraries in `/lib`. In other words, it would have the basic components that an application would need from a Linux host system.

Keep in mind that the container images you run don't necessarily need to match the host Linux system. So, for example, you could run a Fedora base image on an Ubuntu system, as long as there are no specific kernel requirements or libraries built into the container image.

Starting with Linux Containers

Very little preparation is needed to start running containers on your own Linux system. The following procedures describe how to prepare your Linux system to start using containers.

Docker Inc. now makes a free version of its software available via the Moby project, with Docker having become its commercial product. To try an older version of the docker package, you can run the following on a RHEL 7 system to install the docker package and then start and enable the docker service:

```
# yum install docker -y
# systemctl start docker
# systemctl enable docker
```

The `podman` command supports most of the `docker` command line options for working with containers, so you can use it instead of `docker`. Keep in mind that `podman` represents a different code base from `docker`, even though it supports similar management command options. With `podman`, you don't need to have a service running, as you do with `docker`. To install podman on Fedora or RHEL, do the following:

```
# yum install podman -y
```

You can now start using the `podman` or `docker` commands to work with containers and container images for the examples in this chapter.

Pulling and running containers

With the docker or podman packages installed and ready to use, you can try running a container. To start, you can pull a container to your local system and then run it. If you like, you can skip the `pull` command since running the container will pull it if the requested image is not already on your system.

Pulling a container

Choose a reliable container image to try out, as in one that comes from an official project, is up to date, and preferably has been scanned for vulnerabilities. Here is an example of pulling a RHEL 8 UBI base image with the `podman` command (you can replace `podman` with `docker` in these examples):

```
# podman pull registry.access.redhat.com/ubi8/ubi
Trying to pull .../ubi8/ubi...Getting image source signatures
Copying blob fd8daf2668d1 done
Copying blob cb3c77f9bdd8 done
Copying config 096cae65a2 done
Writing manifest to image destination
Storing signatures
096cae65a2078ff26b3a2f82b28685b6091e4e2823809d45aef68aa2316300c7
```

To see that the image is on your system, run the following:

```
# podman images
REPOSITORY   TAG     IMAGE ID      CREATED      SIZE
/ubi8/ubi    latest  096cae65a207  2 weeks ago  239 M
```

Running a shell from a container

Use `podman` or `docker` to run a shell within a container. You can identify the image either by the image ID (`096cae65a207`) or name (`registry.access.redhat.com/ubi8/ubi`). Use the `-i` (interactive) and `-t` (terminal) options so that you can have an interactive session within the container from the bash shell:

```
# podman run -it 096cae65a207 bash
[root@e9086da6ed70 /]#
```

With the shell running, commands that you type will operate within the container. For example, you list the container's filesystem or check out the `os-release` file to see the operating system on which the container is based:

```
[root@e9086da6ed70 /]# ls /
bin   dev  home   lib64       media  opt   root  sbin  sys  usr
boot  etc  lib    lost+found  mnt    proc  run   srv   tmp  var
[root@e9086da6ed70 /]# cat /etc/os-release | grep ^NAME
NAME="Red Hat Enterprise Linux"
```

Because containers are meant to have the minimal amount of content needed to run the intended application, many standard tools may not be inside the container. You can install software inside a running container. However, keep in mind that containers are meant to be discarded. So, if you want to add software permanently, you should build a new image to include the software you want.

Here's an example of adding software to a running container:

```
[root@e9086da6ed70 /]# yum install procps iproute -y
```

Now you can run commands such as `ps` and `ip` inside the container:

```
[root@e9086da6ed70 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 17:44 pts/0    00:00:00 bash
root        40     1  0 17:45 pts/0    00:00:00 ps -ef
[root@e9086da6ed70 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 ...
    inet 127.0.0.1/8 scope host lo
    ...
3: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...
    inet 10.88.0.6/16 brd 10.88.255.255 scope global eth0
    ...
```

Notice that from within the container, you see only two running processes (the shell and the `ps` command). PID 1 is the bash shell. The trimmed output from `ip a` shows that there

is only one external network interface from the container (`eth0@if11`) and that interface is assigned the IP address of 10.88.0.6/16.

When you are done, you can type `exit` to quit the shell and stop the container:

```
[root@e9086da6ed70 /]# exit
```

Although the shell and the container are no longer running, the container is still available on your system in a stopped state. Notice that `podman ps` alone doesn't show the container. You need to add `--all`:

```
[root@e9086da6ed70 /]# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
[root@e9086da6ed70 /]# podman ps --all
CONTAINER ID IMAGE          COMMAND CREATED    STATUS          PORTS
NAMES
437ec53386ca ...ubi:latest bash    1 hour ago Up 1 minute ago
go_ein
```

Later you can see how to delete and restart a stopped container.

Running an FTP server from a container

Because you want to be able to throw away a container when you are done, in general you want any changeable data to be stored outside of the container. The following is a simple example of an FTP server (vsftpd) being run from a container. If you want to try this example yourself, I recommend that you skip to the section "Building a container image" later in this chapter for instructions on how to build the vsftpd container image yourself.

For this procedure, you need a configuration file (`vsftpd.conf`) and an FTP directory that contains a file or two to share (`/var/ftp/pub`) on the host system. When the vsftpd container starts, it bind mounts those items into the container as volumes.

1. **Create a `vsftpd.conf` file:** Create the `vsftpd.conf` file in the default location, `/etc/vsftpd/vsftpd.conf`. See the `vsftpd.conf` man page for details. Here is an example:

   ```
   anonymous_enable=YES
   local_enable=YES
   write_enable=YES
   local_umask=022
   dirmessage_enable=YES
   xferlog_enable=NO
   connect_from_port_20=YES
   listen=NO
   listen_ipv6=YES
   pam_service_name=vsftpd
   userlist_enable=YES
   tcp_wrappers=NO
   vsftpd_log_file=/dev/stdout
   syslog_enable=NO
   ```

```
background=NO
pasv_enable=Yes
pasv_max_port=21100
pasv_min_port=21110
```

2. **Create an ftp directory**: Create an anonymous FTP directory for vsftpd to share in the standard location on the host (`/var/ftp/pub`) and copy a few files to that directory:

```
# mkdir -p /var/ftp/pub
# cp /etc/services /etc/login.defs /var/ftp/pub/
```

3. **Get the vsftpd container image**: Build the vsftpd image as described in "Building a container image" later in this chapter. To check that it is available to run, enter the following:

```
# podman images
REPOSITORY  TAG    IMAGE ID     CREATED        SIZE
vsftpd      latest 487d0db26098 5 seconds ago  208 MB
```

4. **Run the vsftpd container image**: When you run the vsftpd container, you need to expose ports and mount files from the host to the container. Here's an example (you can use `docker` instead of `podman`):

```
# podman run -d -p 20:20 -p 21:21 \
   -p 21100-21110:21100-21110 \
-v /etc/vsftpd/:/etc/vsftpd/ \
-v /var/ftp/pub:/var/ftp/pub \
--name vsftpd vsftpd
# podman ps
CONTAINER ID  IMAGE          COMMAND           CREATED
    STATUS             PORTS                     NAMES
3a5d094dd4b5  vsftpd:latest /usr/local/s2... 9 seconds ago
    Up 10 seconds ago  0.0.0.0:20-21->20-21/tcp  vsftpd
```

This example uses the following options:

-d: Runs the container in detached mode, so the vsftpd service runs in the background.

-p: Standard FTP ports (TCP 20 and TCP 21) are mapped to the same port numbers on the host network interfaces, so the service can be accessed outside of the local host. A range of ports needed for passive FTP are opened to their counterparts on the host as well (21100-21110).

--rm: Although not included in this example, adding `--rm` to the command line will remove the container when it exits.

-v: To use the configuration files (`/etc/vsftpd` directory) and content that you want to share (`/var/ftp` directory) from the host system, those directories are bind-mounted to the same locations on the container with the `-v` option.

--name: Set the name of the container to vsftpd (or any name you like).

Keep in mind that you can bind container content and ports to other locations on the host. In that way, you can have multiple versions of the same software running on the same host without conflicting with each other.

To make your vsftpd service accessible outside of the local system, be sure to open the FTP ports and passive FTP ports that you just assigned as follows:

```
# firewall-cmd --zone=public --add-service=ftp
# firewall-cmd --zone=public \
    --permanent --add-service=ftp
# firewall-cmd --zone=public \
    --add-port=21100-21110/tcp
# firewall-cmd --zone=public \
    --permanent --add-port=21100-21110/tcp
# firewall-cmd reload
```

You can now use any FTP client to access the FTP service via the anonymous user. To configure your vsftpd service further and check that it is working, refer to Chapter 18, "Configuring an FTP Server."

Starting and stopping containers

Unless you specifically set a container to be removed when is stops (`--rm` option), if the container is stopped, paused, or just fails, the container is still on your system. You can see the status of all containers on your system (currently running or not) using the `ps` option:

```
# podman ps
CONTAINER ID  IMAGE                     COMMAND           CREATED
    STATUS                PORTS                      NAMES
4d6be3e63fe3  localhost/vsftpd:latest  /usr/local/s2...  About an
hour ago
    Up About an hour ago  0.0.0.0:20-21->20-21/tcp  vsftpd
# podman ps -a
CONTAINER ID  IMAGE                     COMMAND              CREATED
    STATUS                PORTS                      NAMES
7da88bd62667  ubi8/ubi:latest          bash                 2 minutes
ago
    Exited 7 seconds ago                                   silly_wozniak
4d6be3e63fe3  localhost/vsftpd:latest  /usr/local/s2i/ru... About an
hour ago
    Up About an hour ago  0.0.0.0:20-21->20-21/tcp  vsftpd
```

Only running containers are shown with `podman ps`. By adding `-a`, you can see all containers, including those that are no longer running but have not yet been removed. You can restart an existing container that is no longer running using the `start` option:

```
# podman start -a 7da88bd62667
[root@7da88bd62667 /]#
```

The restarted container was running a bash shell. Because the container's terminal session already existed, I didn't need to start a new one (`-it`). I just needed to attach (`-a`) to the existing session. For a container that was just running in detached mode, I can simply start and stop it as required:

```
# podman stop 4d6be3e63fe3
4d6be3e63fe3...
# podman start 4d6be3e63fe3
4d6be3e63fe3
```

Note that if the container had been started with the `--rm` option, the container would be removed as soon as you stopped it. So, you would have to run a new container instead of just restarting the old one. Because the configuration files and data are stored outside of the container, running a new container is easy and painless. Upgrading the application in the future is as easy as removing the old container and starting up one from the updated container image.

Building a container image

To build a container image, all you need is a Dockerfile describing how to build the image and any other content that you want to include with the image. The following procedures describe how to create a simple container from your own Dockerfile and how to get the software you need to build a vsftpd service into a container from software available on GitHub.

Build a simple container image

This procedure creates a simple container from a Dockerfile. In this process, you create a Dockerfile and a simple script, then build that content into a new container image.

1. Create a directory to hold your container project and then enter that directory:

   ```
   # mkdir myproject
   # cd myproject
   ```

2. Create a script called `cworks.sh` in the current directory that contains the following text:

   ```
   #!/bin/bash
   set -o errexit
   set -o nounset
   set -o pipefail
   echo "This Container Works!"
   ```

3. Create a file named `Dockerfile` in the current directory that contains the following content:

   ```
   FROM registry.access.redhat.com/ubi7/ubi-minimal
   COPY ./cworks.sh /usr/local/bin/
   CMD ["/usr/local/bin/cworks.sh"]
   ```

4. Build a container image called `myproject` from the Dockerfile:

```
# podman build -t myproject .
STEP 1: FROM registry.access.redhat.com/ubi7/ubi-minimal
STEP 2: COPY ./cworks.sh /usr/local/bin/
6382dfd00f7bedf1a64c033515a09eff37cbc6d1244cbeb4f4533ad9f00aa970
STEP 3: CMD ["/usr/local/bin/cworks.sh"]
STEP 4: COMMIT myproject
6837ec3a37a241...
```

5. Run the container to make sure it works. To do that, you can use either the container name (`myproject`) or its image ID (`6837ec3a37a241`):

```
# podman run 6837ec3a37a241
The Container Works!
```

Build an FTP container from GitHub

The following procedure describes how to get the software you need to build a vsftpd service into a container from software available on GitHub. It then shows you how to build and run that container.

1. If you don't already have it, install git on your local system:

```
# yum install git -y
```

2. For this example, we are starting with the vsftpd container-images project on GitHub. Clone a copy of that software to a local directory as follows:

```
# git clone
https://github.com/container-images/vsftpd.git
# cd vsftpd
# ls
default-conf/  Dockerfile  LICENSE  Makefile  README.md
root/          s2i/        tests/
```

3. Modify the files as needed. In particular, go through the Dockerfile and use the latest Fedora image available. Leaving off the *:tag* at the end of the image name says to look for the version of that image that includes the `:latest` tag, which is a special tag that identifies the latest available version of that image. For example, modify the `FROM` line at the beginning so that it appears as follows:

```
FROM registry.fedoraproject.org/fedora
```

4. From the vsftpd directory, use either the `docker` or `podman` commands to build the container image. For example:

```
# podman build -t vsftpd .
STEP 1: FROM registry.fedoraproject.org/fedora:31
Getting image source signatures
Copying blob c0a89efa8873 done
Copying config aaaa3e1d6a done
```

```
Writing manifest to image destination
Storing signatures
STEP 2: ENV SUMMARY="Very Secure Ftp Daemon" ...
STEP 3: LABEL maintainer="Dominika Hodovska <dhodovsk@redhat.com>"
...
STEP 4: RUN dnf install -y vsftpd && dnf clean all
          && mkdir /home/vsftpd
...
Installing:
vsftpd    x86_64   3.0.3-32.fc31   fedora   164 k
...
Complete!
99931652dceacc2e9...
STEP 5: VOLUME /var/log/vsftpd
b79b229d09f726356...
STEP 6: EXPOSE 20 21
b0af5428800140104...
STEP 7: RUN mkdir -p ${APP_DATA}/src
b3652e0d07e35af79...
STEP 8: WORKDIR ${APP_DATA}/src
f9d96dee640c5cedc...
STEP 9: COPY ./s2i/bin/ /usr/local/s2i
ded9b512693ccabaa...
STEP 10: COPY default-conf/vsftpd.conf /etc/vsftpd/vsftpd.conf
0c48af8d4f72b76c7...
STEP 11: CMD ["/usr/local/s2i/run"]
STEP 12: COMMIT vsftpd
aa0274872f23ae94dfee...
```

5. Check that the new image was created:

```
# podman images
REPOSITORY       TAG    IMAGE ID     CREATED       SIZE
localhost/vsftpd latest aa0274872f23 4 minutes ago 607 MB
```

This build process consisted of 12 steps. The `FROM` line in the first step pulls the `fedora` image from `registry.fedoraproject.org` container registry. Each subsequent step runs a command. If content is added during the command, a new layer is created for the image. Step 2 and Step 3 set environment variables and labels that are used during the build as well as to identify attributes of the container image when it is used later.

Step 4 runs the `dnf` command that installs the vsftpd package from the Fedora `yum` repos to the container. Notice that the `RUN` instruction has both `dnf install` and `dnf clean` in the same instruction. This is good practice since it prevents an additional layer of cached dnf data from being included with the image.

Step 5 identifies the volume used to store vsftpd log files. Step 6 exposes TCP port 20 and TCP 21 for the FTP service. Note that even though the ports are exposed (meaning that they can be seen from outside of the container), they will still need to be mapped to host ports

when you run the container later, if you want those ports to be available outside of the local system.

Step 7 and Step 8 create a directory and set that as the working directory for the application. Step 9 copies source-to-image (s2i) scripts into the container to run the vsftpd service. Step 10 copies a default `vsftpd.conf` configuration file into the container.

The CMD instruction in Step 11 sets `/usr/local/s2i/run` as the default command to execute if the container is run without overriding that command. Step 12 commits the final vsftpd image to local storage (which you can see by typing `podman images`).

For more information on creating and using a Dockerfile to build container images, refer to the Dockerfile Reference (`https://docs.docker.com/engine/reference/builder/`). To learn more about options for building container images, refer to the `podman` man pages (`man podman build`).

Tagging and pushing an image to a registry

So far, I have shown an example of building a container image and running it on your local system. To make your image available to other people on other systems, you typically add that image to a container registry. Follow these instructions to tag an image on your local system and push it to a remote container registry.

To try out a simple registry on your local system, install the `docker-distribution` package on a Fedora or RHEL 7 system. For a more permanent solution, you can get accounts on public container registries such as Quay.io and Docker Hub. Both free trials and subscriptions are available from Quay.io (`https://quay.io/plans/`). You can also set up and run your own supported container registry, such as Red Hat Quay (`https://www.openshift.com/products/quay`).

To get you started, the following procedure has you install the `docker-distribution` package on the local system and then tag and push an image to it:

1. **Install docker-distribution**: On a RHEL 7 or recent Fedora system, install and start the docker-distribution:

```
# yum install docker-distribution -y
# systemctl start docker-distribution
# systemctl enable docker-distribution
# systemctl status docker-distribution
• docker-distribution.service-v2 Registry server for Docker
   Loaded: loaded
    (/usr/lib/systemd/system/docker-distribution.service;
    enabled; vendor pres>
    Active: active (running) since Wed 2020-01-01...
```

2. **Open the registry port**: To be able to push and pull container images from other host systems, you need to open TCP port 5000 on the firewall:

```
# firewall-cmd --zone=public
 --add-port=5000/tcp --permanent
```

3. **Tag the image**: By tagging a local image, you identify the location of the registry where the image will be stored. Replace the image id and `host.example.com` with your image ID and hostname or IP address to tag the image:

```
# podman images | grep vsftpd
localhost/vsftpd  latest  aa0274872f23  2 hours ago  607 MB
# podman tag aa0274872f23
host.example.com:5000/myvsftpd:v1.0
```

4. **Push the image**: Push the image to the local registry (substitute your hostname or IP address). Turn off `tls-verify`, because docker-registry uses http protocol:

```
# podman push --tls-verify=false
host.example.com:5000/myvsftpd:v1.0
```

5. **Pull the image**: To make sure that the image is available from your registry, try to pull the image. Either delete the image from your local system or go to another host to try this:

```
# podman pull --tls-verify=false \
               host.example.com:5000/myvsftpd:v1.0
```

At this point, you should be able to share your images with others from your registry.

If you were using a public registry, which is a better solution for sharing images with a wider audience, the procedure for pushing and pulling images would look like the following.

```
# podman login quay.io
Username: myownusername
Password: ***************
# podman tag aa0274872f23 \
    quay.io/myownusername/myvsftpd:v1.0
# podman push quay.io/myownusername/myvsftpd:v1.0
```

Using containers in the enterprise

Although the Docker project made huge strides in simplifying how individual containers can be used, it was the Kubernetes project that helped propel Linux containers into the enterprise. While command-line tools like `docker` and `podman` are good for managing individual containers, Kubernetes offers a platform for deploying large, complex applications across huge data centers. Refer to Chapter 30, "Deploying Applications as Containers with Kubernetes," for information on how to use Kubernetes to deploy and manage containerized applications in the enterprise.

Summary

Containerizing applications has seen widespread adoption over the past few years. The Docker project was a huge contributor to the simplification of containerizing individual

applications and running them on single systems. Tools such as `podman` also became available to deploy and manage individual containers on Linux systems.

This chapter described how to pull, run, build, and otherwise manage containers using command-line tools like `docker` and `podman`. You can use this knowledge as a foundation for understanding how containerization works and for how those concepts are applied later in Chapter 30, as it describes how Kubernetes can manage containerized applications across an entire enterprise.

Exercises

The exercises in this section describe tasks related to working with containers. If you are stuck, solutions to the tasks are shown in Appendix B. Keep in mind that the solutions shown in Appendix B are usually just one of multiple ways to complete a task.

1. Choose either `podman` (for any RHEL or Fedora system) or `docker` (RHEL 7), install the software package containing the tool of your choice, and start any necessary services to use those commands.
2. Using either `docker` or `podman`, pull this image to your host: `registry.access.redhat.com/ubi7/ubi`.
3. Run the `ubi7/ubi` image to open a bash shell.
4. With the bash shell open within a container, run a few commands to see the operating system on which the container is based, install the `proc-ps` package, run a command to see the processes running inside the container, and then exit.
5. Restart the container again and connect to it using an interactive shell. Exit the shell when you are finished.
6. Create a simple Dockerfile from a `ubi7/ubi` base image, include a script named `cworks.sh` that uses `echo` to output the string "The Container Works!" and add that script to the image so that it runs as the default command.
7. Use `docker` or `podman` to build an image named `containerworks` from the Dockerfile you just created.
8. Gain access to a container registry, either by installing the docker-distribution package or by getting an account on Quay.io or Docker Hub.
9. Tag and push your new image to your chosen container registry.

CHAPTER

27

Using Linux for Cloud Computing

IN THIS CHAPTER

How Linux is used in clouds

Understanding basic cloud technology

Setting up a hypervisor

Creating virtual machines

Computer operating systems were originally designed to be installed directly on computer hardware. When it needed memory, storage, processing power, or network interfaces, a computer operating system looked for physical RAM, hard disks, CPUs, and network interface cards. When it needed more of those things than were physically installed, you turned the machine off and physically added them to the computer. Nowadays, virtualizing these items is what makes cloud computing possible.

Virtualization, as is relates to computers, is the act of making computing resources that were originally designed as physical objects to be represented by virtual ones. For example, a virtual operating system (referred to as a *virtual machine*) doesn't communicate directly with the hardware. Instead, a *virtual machine (VM)* interacts with a specially configured host computer referred to as a *hypervisor*. So, instead of being able to run one operating system on a physical computer, you could potentially run dozens or even hundreds of VMs on a single physical computer.

The advantages gained from running VMs are massive. Not only can you have multiple operating systems running on the same computer, but those systems can be different ones—Linux, BSD, Windows, or any other system made to run on the computer's hardware. If you need to shut down the host computer for maintenance, you can migrate the running VMs to another hypervisor with an imperceptible amount of downtime.

To support virtual machines across multiple hypervisors, you can virtualize the features they rely upon as well. For example, virtual networks and virtual storage can span multiple hypervisors, so if a VM needed to move to another hypervisor, the same virtual networks and storage would be available to a newly migrated virtual machine.

You don't need to build a whole data center to begin understanding virtualization and to use some of the underlying technologies that make cloud computing possible. This chapter starts you off by

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

helping you to set up a host computer to run as a hypervisor, start running VMs on that hypervisor, and then learn how to migrate VMs to other hypervisors (in order to prevent downtime or just to grow your capacity).

Overview of Linux and Cloud Computing

Cloud computing moves us into an arena where everything you learned previously in this book is being abstracted and automated. In a cloud environment, when you install a system, you are probably not booting from a physical DVD, erasing the local hard drive, and installing Linux directly on a computer sitting in front of you. You are also probably not logging into the installed system and manually configuring the software and features you want to run on that system.

Instead, you are installing to a VM or running a container that is on some host system in the cloud. The network interfaces that you see may not be represented by a physical switch, rather they may be virtual networks that exist on a single computer or span multiple hypervisors.

Today, every software aspect of cloud computing can be fulfilled using open source technology running on Linux systems. To get a feel for how some of the basic technologies in cloud computing work, this chapter explains some of those technologies and then describes how to set up a hypervisor and start using VMs on that hypervisor.

Cloud hypervisors (aka compute nodes)

In cloud computing, the operating systems serving cloud users are not running directly on computer hardware. Instead, hypervisors are configured to run many operating systems as what are referred to as virtual machines (VMs).

Depending on your cloud environment, you may hear a hypervisor referred to as a compute node, a *worker node*, or simply as a host. Because hypervisors tend to be commodity items (dozens or hundreds of hypervisors may be set up for a location), Linux is the logical choice as the operating system running as hypervisors directly on hardware.

Kernel-based Virtual Machine (KVM) is the basic virtualization technology implemented in most Linux distributions to make a Linux system into a hypervisor. KVM is supported on Ubuntu, Red Hat Enterprise Linux, Fedora, CentOS, and many other Linux systems.

The other major technology that can be used instead of KVM to make a Linux system into a hypervisor is Xen (`www.xenproject.org`). Xen has been around longer than KVM, and it is supported in products from Citrix Systems and Oracle.

Later in this chapter, I describe how to check to see if a computer has the required hardware features to be used as a hypervisor and how to configure it to be used with KVM.

Cloud controllers

Because a cloud configuration can include multiple hypervisors, pools of storage, multiple virtual networks, and many virtual machines, you need centralized tools to manage and monitor those features. You can use both graphical and command-based tools for controlling cloud environments.

Although not considered a full cloud controller, the Virtual Machine Manager (`virt-manager`) GUI and `virsh` command can be used to manage a small cloud-like environment. Using `virt-manager`, you can get a feel for managing multiple virtual machines across several hypervisors, and you can learn how to deal with virtual networks and shared storage pools.

Full-blown cloud platforms have their own controllers for offering much more complex interactions between cloud components. For example, the Red Hat OpenStack platform (`https://access.redhat.com/products/red-hat-openstack-platform`) and its upstream RDO project (`https://www.rdoproject.org`) provide flexible and expandable cloud environments for managing VMs and all associated supporting features. For Red Hat Virtualization (RHV), the RHV Manager provides many of the same features.

If you want to start out more simply, however, you can start by using `virt-manager`, the VM Desktop tool, to manage your first mini cloud-like environment.

Cloud storage

New demands on data storage arise when you move your operating systems and applications into a cloud environment. To be able to move a virtual machine to run on another hypervisor, its storage must be available from that new hypervisor. Storage needs for clouds include back-end storage for your VMs, images for launching VMs, and databases for storing information about the cloud itself.

Shared storage between hypervisors can be done as simply as creating an NFS share (see Chapter 20, "Configuring an NFS File Server") and mounting it on the same mount point between multiple hypervisors. NFS is one of the easiest ways to implement shared storage.

More robust shared storage that can handle disk failures and provide better performance works better for clouds providing critical services. Shared block storage, where you mount a whole disk or disk partition, can be accomplished using technologies such as iSCSI or Fibre Channel.

Ceph (`http://ceph.com`) is an open source project for managing both block and object storage that is popular for managing storage in cloud environments. GlusterFS (`www.gluster.org`) is a scale-out filesystem that is often used in cloud environments.

For the simple mini-cloud example in this chapter, I used NFS to provide shared storage between the hypervisors. Ceph and GlusterFS are more appropriate for enterprise-quality installations.

Cloud authentication

To be able to limit how much cloud resources a user can consume, and possibly track and charge for that use, you need authentication mechanisms. Authentication is necessary for those who are using cloud features as well as for those who are allowed to administer cloud features.

Cloud platform projects sometimes let you connect centralized authentication mechanisms to validate and authorize cloud users. These can include Kerberos, Microsoft Active Directory, and others. In Linux, Identity, Policy, and Audit (IPA) software (see `www.freeipa.org`) offers a full set of authentication features that can be used across an enterprise cloud platform.

Cloud deployment and configuration

If you are managing a large cloud infrastructure, you don't want to have to walk over to each machine and click through a graphical installation every time you want to add a hypervisor or other node on your network. Today, many tools can deploy and configure Linux systems as simply as rebooting the computer and having it boot up to a preconfigured installer.

In Chapter 9, "Installing Linux," I talked about how to use a PXE server (to boot a Linux installer automatically over the network from your network interface card) and Kickstart files (to identify all of the answers you need to complete an installation). With that setup in place, you can simply boot a computer from a network interface and come back a short time later to find a fully installed Linux system.

After a computer is deployed, systems can be configured and possibly monitored and updated using tools such as Puppet (`http://puppetlabs.com`) and Chef (`www.chef.io`). Whole work environments can be deployed in virtual machines using Vagrant (`www.vagrantup.com`). Ansible (`www.ansible.com`) is another tool for automating IT infrastructures and the applications that run on them.

Cloud platforms

If you want to implement your own, private cloud within your organization, the open source OpenStack platform is probably the most popular choice. It offers a huge amount of flexibility and power in how you configure and use it.

Red Hat Virtualization (RHV) is another popular cloud platform. RHV makes it easy to start with a simple RHV Manager and one or two hypervisors to run the VMs it manages and then grow your own cloud platform by adding more hypervisors, storage pools, and other features.

If you want to use public clouds based on open source technology to run the operating systems that you need, you can use any of several different cloud providers. Public cloud providers that you can use to run Linux VMs include Amazon Web Services (`www.amazon.com/aws`), Google Cloud Platform (`https://cloud.google.com`), and Rackspace (`www.rackspace.com`). Chapter 28, "Deploying Linux to the Cloud," covers how to deploy Linux to some of these cloud providers.

Trying Basic Cloud Technology

To help you understand cloud technology from the ground up, this section illustrates some of the basic building blocks of a modern cloud infrastructure. Using three computers, I'll help you create a setup that includes the following:

Hypervisors A *hypervisor* is a software component that allows you to run other multiple computer systems on it. Those other systems are referred to as *virtual machines (VMs)*. A cloud infrastructure may have dozens or hundreds of hypervisors running, possibly running thousands of virtual machines.

Virtual machines The virtual machines that you run on a Linux hypervisor can be the same type of Linux system, a different Linux system, a Windows system, or any other type of system that is compatible with the hardware on which the hypervisor runs. Thus, the virtual machines that run on the hypervisors that we will build here could include Fedora, Ubuntu, RHEL, CentOS, Microsoft Windows, and others.

Shared storage To offer the greatest flexibility, the storage that hypervisors make available to virtual machines is often shared among a pool of hypervisors. This allows a set of hypervisors to share a set of images that they use to install or start virtual machines. It also lets the same set of virtual machines run on any hypervisor in that group and even move to a different hypervisor without shutting down the VM. Moving running VMs can be useful if a hypervisor becomes overloaded or if it needs to be shut down for maintenance.

The setup we build in the following procedure allows you to work with VMs in these ways:

- Installing a new VM on a hypervisor
- Setting features on your VMs
- Logging in to and using a VM running on a hypervisor
- Migrating running VMs to another hypervisor

We will explore the following technologies:

Kernel-based Virtual Machine (KVM)) KVM is the basic kernel technology that allows virtual machines to interact with the Linux kernel.

QEMU Processor Emulator One `qemu` process runs for each active virtual machine on the system. QEMU provides features that make it appear to each virtual machine as though it is running on physical hardware.

Libvirt Service Daemon (`libvirtd`) A single `libvirtd` service runs on each hypervisor. The `libvirtd` daemon listens for requests to start, stop, pause, and otherwise manage virtual machines on a hypervisor. Those requests can come from an application designed to manage virtual machines (such as `virt-manager` or OpenStack Dashboard) or from an application that you create to talk directly to the `libvirt` application programming interface.

Virtual Machine Manager The Virtual Machine Manager (`virt-manager` command) is a GUI tool for managing virtual machines. Besides letting you request to start and stop virtual machines, `virt-manager` lets you install, configure, and manage VMs in different ways. You can use the `virsh` command to pass options to the command line to work with virtual machines instead of clicking in a GUI window.

Virtualization Viewer The `virt-viewer` command launches a virtual machine console window on your Desktop. The window that appears allows you to work from a console window to a Desktop or command-line interface to the selected virtual machine (depending on what that VM has to offer). What this means is that someone consuming your PaaS could bundle together their own operating system, application, configuration files, and data and deploy them. They would rely on your PaaS to provide the compute power, storage, memory, network interfaces, and management features needed to run the virtual machines containing their applications.

The next section gives you your first taste of some of the foundational technologies of Linux clouds. It describes how to set up a small cloud by configuring your own hypervisors, virtual machines, and virtual storage.

Setting Up a Small Cloud

With three physical machines connected together on a network, you can illustrate some of the basic concepts that you need in order to understand how to build your own cloud. The three computers running Fedora 30 and the network connecting them are configured as follows:

Networking A high-speed, wired network was set up to connect the three computers. Fast network connections are critical to successful VM migration. In this example, each hypervisor also has a network bridge configured so that each virtual machine can pick up an IP address directly from a DHCP service on the network.

Hypervisors Two of the computers are configured as hypervisors. A *hypervisor* (sometimes referred to as a *host* or a *compute node*) allows you to run virtual machines. In Fedora 30, the basic hypervisor technology is called *Kernel-based Virtual Machine (KVM)*, while the actual virtual machines are managed by the `libvirtd` service.

Storage One computer is configured to offer shared storage between the two hypervisors. For simplicity, NFS is used to create the shared storage, although in a production environment, iSCSI or Fibre Channel would be better solutions.

Note

For test purposes, you could use one of the two hypervisors to provide the shared storage. However, one of the main purposes of configuring two hypervisors and separate shared storage is that you want to be able to shut down any hypervisor and still have all of your virtual machines operate normally. If you have shared storage available from one of the hypervisors, you could never bring that hypervisor down without shutting down all of the VMs using that storage.

Configuring hypervisors

In the following procedure, I installed Fedora 30 on two physical computers and configured them as KVM hosts running the `libvirtd` service. Follow these steps to accomplish this for yourself.

Step 1: Get Linux software

Go to the Get Fedora page (`https://getfedora.org`) and download Fedora 30. I chose to download the Fedora 30, 64-bit Workstation edition DVD ISO. If a later version of Fedora is available, you could likely use that instead.

Use any available DVD burning application to burn the image to a DVD or otherwise make the image available to install (such as by PXE booting).

Step 2: Check your computers

The computers you use as hypervisors in Fedora 30 need to meet a few requirements. You should check the following on your computer before you start installing:

Supports virtualization. You can check for virtualization support by looking at the flags set in the CPU.

Memory. The computer must have enough RAM not only to run the host operating system, but also for each virtual machine that you expect to run on the system.

Processing power. Keep in mind that each virtual machine consumes processing power for itself and any application running inside the virtual machine.

Storage is another consideration. However, because we intend to configure storage from a separate node on the network, we will address that issue later.

To check that the available features of your computers meet the requirements, boot a Linux Live CD or DVD, open a Terminal window, and type the following commands:

```
# cat /proc/cpuinfo | grep --color -E "vmx|svm|lm"

flags  : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64
monitor ds_cpl vmx smx es...
...
```

Running the above command shows that this computer is a 64-bit computer (`lm`) and that an Intel chip supports virtualization features (`vmx`). If the CPU were an AMD chip that supported virtualization, instead of `vmx`, you would see `svm` highlighted. Those settings show that this computer can be used as a hypervisor.

When you start running VMs on a host, memory is often the bottleneck. For memory requirements, you must add what is needed by the host to whatever you need for each VM.

You can lower memory requirements by not having Desktop software installed, as most hypervisors do. In this case, however, I performed a Fedora Workstation install, which comes with a Desktop. To check the memory and swap on the computer, I entered the following:

```
# free -m
              total        used        free      shared  buff/cache   available
Mem:          15318        4182        6331        1047        4805        9678
Swap:          7743           0        7743
```

This system has about 16Gb of RAM and 8Gb of swap. I estimate that 4Gb is good for a Desktop system. If I allow 1Gb or 2Gb for each VM, this system should be able to run 6–12 VMs along with the Desktop. Check the memory requirements for the operating systems and the applications that you plan to run to determine your particular memory needs.

To check the number and types of processors on your computer, enter the following:

```
# grep processor /proc/cpuinfo

processor : 0
...
processor : 6
processor : 7
# head /proc/cpuinfo
processor  : 0
vendor_id  : GenuineIntel
cpu family : 6
model      : 60
model name : Intel(R) Core(TM) i7-4800MQ CPU @ 2.70GHz
stepping   : 3
cpu MHz    : 2701.000
cache size : 6144 KB
...
```

The first command in the preceding code shows that there are eight (0 through 7) processors on the computer. The second command for the first processor shows that it is GenuineIntel, the model number, model name, CPU speed, and other information.

To do live VM migration between the two hypervisors, the CPUs must be in the same family. If they don't have compatible CPUs, you could migrate a VM by shutting it down on one hypervisor and starting it up from shared storage on the other.

After you have sized up the two hypervisor computers, start installing Fedora on them.

Step 3: Install Linux on hypervisors

Using the Fedora 30 Workstation installation media, begin installing the two hypervisors. Follow descriptions in Chapter 9, "Installing Linux," for installing Fedora. You should know the following things, which are specific to the installation for this procedure:

Name the hypervisors I set the hostnames on the hypervisors to `host1.example.com` and `host2.example.com`.

Partitioning When partitioning, I erase the entire hard disk. Then I create a 500Mb `/boot` partition and a 12Gb swap partition, and I assign the rest of the disk space to the root partition (`/`). The `/var/lib/libvirt/images` directory holds most of the data on this system, but that is a shared directory, available from another system on the network and shared between the two hypervisors. (More on that later.)

Networking If given the option, turn on wired network interfaces for each hypervisor. The hypervisors and storage should all be on the same local network, because the speed of your network connection among those machines is critical to getting good performance.

Software packages During installation, I install only the default Fedora Workstation packages. After installation is complete and the system is rebooted, I install more of the software that's needed for each hypervisor.

Reboot the computer when installation is finished (ejecting the DVD and starting up on the hard drive). After the system is rebooted, update the Fedora software, add new packages, and reboot the system again as follows:

```
# yum update -y
# yum install virt-manager libvirt-daemon-config-network
# reboot
```

The `virt-manager` package contains the GUI tool for managing your virtual machines. The `libvirt-daemon-config-network` package creates the default network interface that lets the virtual machines access external networks (through the host) using Network Address Translation (NAT). The default address range assigned to the virtual machines is 192.168.122.2 through 192.168.122.254.

Other packages that you will need should already be included with the Fedora Workstation install. If you did a different install type, make sure that you have the following packages also added:

- `libvirt-client` (for the `virsh` command)
- `libvirt-daemon` (to get the `libvirtd` service)

Step 4: Start services on the hypervisors

You need to make sure that the `libvirtd` service is running on both hypervisors. Start the `sshd` service as well. They may already be running, but just to make sure, do the following as root on both hypervisors:

```
# systemctl start sshd.service
# systemctl enable sshd.service
# systemctl start libvirtd.service
# systemctl enable libvirtd.service
```

The `sshd` service allows you to log into the hypervisors over the network, if necessary. The `libvirtd` service is the one with which you may be unfamiliar. It is listening for requests to manage your virtual machines on each host.

Step 5: Edit /etc/hosts, or set up DNS

To make it convenient to communicate between the hypervisors and storage system, you should assign hostnames to each system and map those names to IP addresses. Setting up a DNS server to which all the systems point is probably the best way to do that. However, for our simple example, you can just edit the `/etc/hosts` file on each system and add entries for each host.

Here is an example of what additional entries to your `/etc/hosts` file might look like for the three systems used in this procedure:

```
192.168.0.138  host1.example.com host1
192.168.0.139  host2.example.com host2
192.168.0.1    storage.example.com storage
```

Next you need to configure the storage.

Configuring storage

You can provide networked storage to the hypervisors for this procedure in many ways. I chose to set up a separate Fedora system on the same local network as the hypervisors and use NFS to attach the shared storage to both hypervisors.

NFS is not the most efficient method of sharing storage among hypervisors, but it is one of the easiest and most common to set up. In this procedure, I use the Virtualization Manager GUI tool (`virt-manager`) to configure the NFS storage pool.

For consistency's sake, the NFS share set up from the storage system is the `/var/lib/libvirt/images` directory. It is mounted in the same place on each of the hypervisors. (For testing, if you only have two machines available, you can configure storage from one of the hypervisors. Keep in mind, however, that this means that you can't turn off that hypervisor without shutting down all of your VMs.)

Step 1: Install Linux software

To set up your storage on an NFS server, you can use pretty much any Linux system that has an NFS service available. Consider these things when you install Linux:

- **Disk space**. Make sure that you have enough storage space available on the partition that contains the shared directory. For this example, `/var/lib/libvirt/images` is the shared directory.
- **Performance**. For best performance, you want to have a disk that has fast access times and data transfer rates.

For Fedora and RHEL, NFS server software is available from the `nfs-utils` package. For Ubuntu, you need the `nfs-kernel-server` package. After initial installation is finished, check that the NFS server software is installed. If it isn't, you can install it on Fedora or RHEL with this command:

```
# yum install nfs-utils
```

For Ubuntu and similar systems, type this:

```
# apt-get install nfs-kernel-server
```

Step 2: Configure NFS share

To create an NFS share, you need to identify the directory to share and add information about it to the `/etc/exports` file. Follow these steps:

a. **Create a directory**. You can share any directory containing the space that you want to share. Consider making a new directory and mounting a whole disk or partition on it. For this example, I create a directory named `/var/storage` as follows:

```
# mkdir -p /var/storage
```

b. **Allow exporting**. On your storage system, create an entry in the `/etc/exports` file to share the directory with selected systems (by name or IP address). For this example, I allowed read-write access (`rw`) to all systems on the 192.168.0 subnetwork:

```
/var/storage 192.168.0.*(no_root_squash,rw,sync)
```

Step 3: Start the NFS service

Start the NFS service and open the firewall on the storage system to allow access to that service. Here's how:

a. **Start and enable NFS**. On the latest Fedora and RHEL systems, enter the following to start the NFS server:

```
# systemctl start nfs-server.service
# systemctl enable nfs-server.service
```

On RHEL 6, older Fedora, and some Ubuntu systems, use these commands to start and enable the NFS service:

```
# service nfs start
# chkconfig nfs on
```

b. **Open the firewall**. To open the firewall ports so that those outside the local system can use your NFS share, do the following on Fedora 30:

```
# firewall-cmd --permanent --add-service=rpc-bind
# firewall-cmd --permanent --add-service=nfs
# systemctl restart firewalld
```

For systems using `iptables` directly, see Chapter 20 for information on how to open your firewall for the NFS service.

Step 4: Mount the NFS share on the hypervisors

Log in to each hypervisor and follow these steps to make the share available locally. Note that the location of the mount point directory on each hypervisor must be the same. Here's how:

a. **Check the NFS share availability**. From each of the two hypervisors, make sure that you can see the available share by entering the following:

```
# showmount -e storage.example.com
Export list for storage.example.com:

/var/storage 192.168.0.*
```

b. **Mount the NFS share**. Add information about the share to the `/etc/fstab` file. For our example, to allow the directory from the 192.168.0.1 system to be mounted on the same directory locally each time the system boots, the entry in the `/etc/fstab` file could look like this:

```
storage.example.com:/storage /var/lib/libvirt/images nfs defaults 0 0
```

c. **Set SELInux boolean.** If SELinux is in enforcing mode, set the following boolean to allow qemu-kvm to use the NFS share:

```
# setsebool -P virt_use_nfs 1
```

d. **Test the NFS mount**. To check that you got the mount entry correct, run the following command to mount all entries in the `/etc/fstab` that have not already been mounted, and check that the NFS share was mounted:

```
# mount -a
# mount | grep libvirt
storage.example.com:/var/storage on /var/lib/libvirt/images type nfs4
(rw,relatime,vers=4.0,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,
port=0,timeo=600,retrans=2,sec=sys,clientaddr=192.168.0.1,local_lock=none,
addr=192.168.0.138)
```

With your hypervisors and storage now in place, you can begin creating your virtual machines.

Creating virtual machines

The Virtual Machine Manager (`virt-manager`) is a good tool to use to create you first virtual machines. It steps you through the installation and setup of virtual machines, and it provides a way to view and change the status of your existing virtual machines.

Later, when you understand the kinds of features that go into creating virtual machines, you can use the `virt-install` command to create virtual machines instead. The

advantage of `virt-install` is that you can script or easily copy and paste a command line to create a virtual machine instead of having to click through a GUI window.

You downloaded the Fedora 30 Workstation ISO image earlier in this chapter, so I'll use that in the example for creating a virtual machine. However, if you prefer, you can install many different versions of Linux or Windows as your virtual machine.

Step 1: Get images to make virtual machines

You can create a virtual machine in many ways. In general, you start with either a pre-built image (basically a copy of a working virtual machine) or just install from an installation ISO image into a fresh storage area. Here we are going to do the latter and create a VM from the Fedora 30 Workstation installation ISO image.

Assuming that you are logged in to one of the hypervisors as root and the ISO image is in the current directory, copy the ISO to the default directory used by `virt-manager` for storage (`/var/lib/libvirt/images`):

```
# cp Fedora-Workstation-Live-x86_64-30-1.2.iso /var/lib/libvirt/images/
```

Because that directory is shared by both hypervisors, you can go to either hypervisor to use that image.

Step 2: Check the network bridge

On each hypervisor, there should be a default network bridge named `virbr0`. All virtual machines will be added to this network interface and automatically assigned an IP address. This default bridge exists due to libvirtd's default virtual network. By default, the hypervisor uses the address range of 192.168.122.2 through 192.168.122.254 to assign to the virtual machines. Using Network Address Translation (NAT), the host can route packets from the virtual machines using these private addresses to external network interfaces.

Do the following on each hypervisor to check the bridge for each:

```
# brctl show virbr0

bridge name  bridge id            STP enabled  interfaces
virbr0       8000.001aa0d7483e    yes          vnet0
# ip addr show virbr0

5: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
        state UP group default
    link/ether fe:54:00:57:71:67 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1 brd 192.168.122.255 scope global
dynamic virbr0
```

Step 3: Start Virtual Machine Manager (virt-manager)

From the Desktop on either hypervisor, do the following to open Virtual Machine Manager and connect it to the hypervisor:

a. **Start `virt-manager`.** Go to the Activities screen, type **Virtual Machine Manager** into the search box, and press Enter, or type **`virt-manager`** from the shell. Type the root password when prompted. You should see the Virtual Machine Manager window.

b. **Check the connection to the hypervisor.** From the Add Connection pop-up, the hypervisor (QEMU/KVM) should already be set and the Autoconnect check box should be checked. Click Connect to connect to the local hypervisor if it has not already been done.

Step 4: Check connection details

After connecting to the hypervisor, set up some connection details. To do that, from the Virtual Machine Manager window, do the following:

a. **View the connection details.** Select Edit ➪ Connection Details to see the Connection Details window. Select the Overview, Virtual Networks, Storage, and Network Interfaces tabs to familiarize yourself with connection information for your hypervisor. For example, the Storage tab appears in Figure 27.1, showing that there are 438.40Gb of free space in the location used by default for storage by this hypervisor (`/var/lib/libvirt/images` directory).

b. **Check that the network bridge is available.** Select the Virtual Networks tab, and make sure that the bridge (`virbr0`) is in the list of available network interfaces.

Step 5: Create a new virtual machine

To create a new virtual machine from the Virtual Machine Manager window, do the following:

a. **Start the wizard.** To start the Create a New Virtual Machine Wizard, select File ➪ New Virtual Machine. The Create a New Virtual Machine window appears.

b. **Choose the installation method.** Four ways of creating the virtual machine are presented. The first three are ways to identify the location of installation media. The fourth lets you import an existing disk image. For our example, choose the first selection (Local install media) and click Forward.

c. **Choose the ISO.** Select the Use ISO Image button and choose Browse. In the window that appears, select or browse to the Fedora 3021 Workstation ISO, select Choose Volume, and click Forward to continue.

d. **Choose the memory and CPU.** Choose the amount of RAM and number of processors available to the VM and click Forward. I suggest at least 1024Mb of RAM and at least one processor. Using 2048Mb of RAM, if it is available, is better.

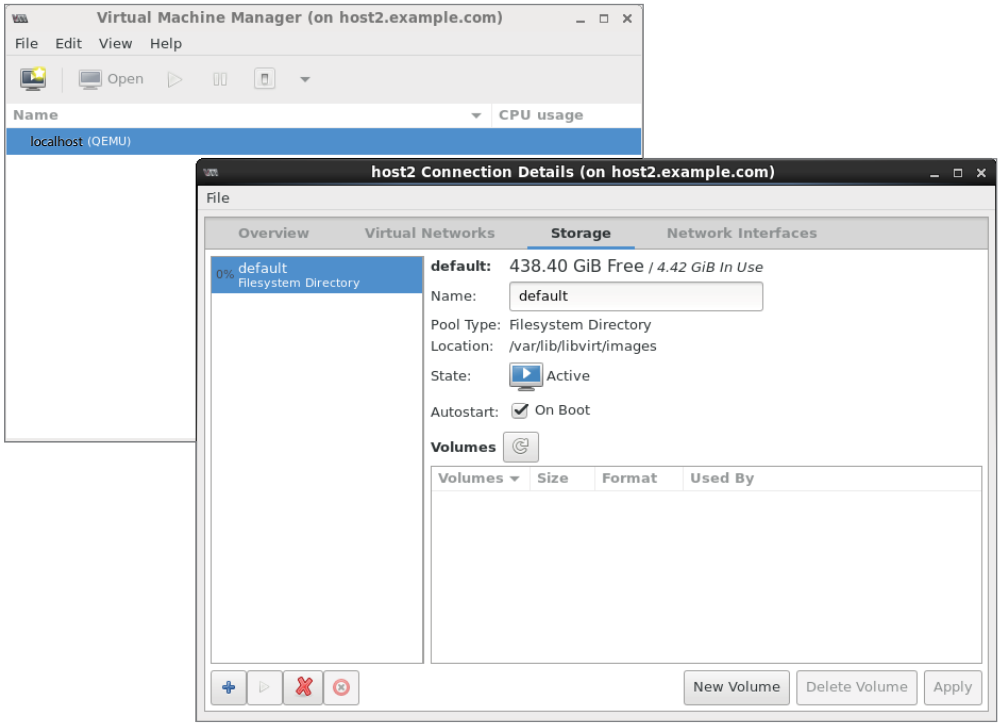
e. **Enable storage.** Choose the amount of disk space that you want the VM to consume. I suggest at least 10Gb for a Fedora Workstation, but you could probably get by with less. The `qcow2` image that is created grows to the size you actually consume (up to the amount allocated), so over-allocating space causes no problem until you actually try to use that space. Set the cache mode to `none` or `directsync` to be able to migrate the VM later. Click Forward.

f. **Review the settings before the installation starts.** Choose the name for the virtual machine, and review the other settings for your installation. Select Customize Configuration Before Install to further review settings. Leave other settings at the default for now, and click Finish.

g. **Review the hardware settings.** If you selected Customize on the previous screen, you can review the settings in more detail. Make sure the cache mode is set to `none` or `directsync`. When you are satisfied, select Begin Installation.

FIGURE 27.1

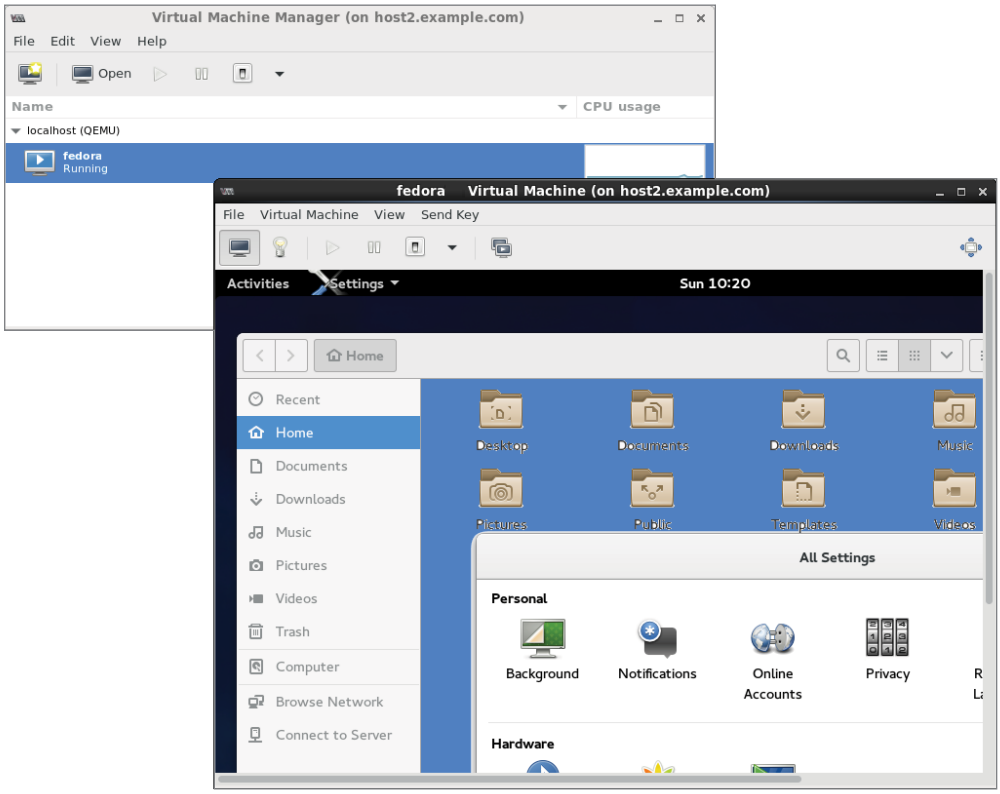
Start Virtual Machine Manager and check connection details.



h. **Install the virtual machine.** You are prompted to install the system just as you would be if you were installing directly to hardware. Complete the installation, and reboot the virtual machine. If the VM window isn't open, double-click the VM entry (in this case, fedora1) in the `virt-manager` window and log in. Figure 27.2 shows an example of the `virt-manager` window with the Fedora Workstation virtual machine displayed.

FIGURE 27.2

Open the virtual machine and begin using it.



Managing virtual machines

After you have one or more virtual machines installed on a hypervisor, you can manage each VM in much the same way that you manage a computer system installed directly on hardware. You can do the following:

- **View the system from a console.** Double-click a running VM in the `virt-manager` window. A console window opens to the VM, allowing you to use the VM

just as you would from a physical console to access an operating system installed directly on the hardware. You can bypass `virt-manager` to display a VM's console directly with `virt-viewer`. For example, for a VM named `rhel8-01`, type **`virt-viewer rhel8-01`**.

- **Shut down the VM**. Right-click the VM entry and select Shut Down. Then select either Shut Down (to shut down properly) or Force Off (effectively pulling the plug). Or, you can select Reboot.
- **Start the VM**. If the VM is currently shut down, right-click the entry and select Run to start the VM running.
- **Delete the VM**. If you are totally finished using the VM, select Delete. You are asked if you want to delete the storage as well. Uncheck the box if you want to keep the storage associated with the VM.

Now that you are comfortable using your virtual machines, you can try migrating a VM to another hypervisor.

Migrating virtual machines

Being able to migrate your virtual machines between different hypervisors gives you tremendous flexibility in managing your computer workloads. Here are some of the advantages:

- *Improve performance* by moving VMs from hypervisors that are overloaded to ones that have more available memory and CPU capacity.
- *Do routine maintenance* on a hypervisor while keeping your VMs running.
- *Move VMs off underutilized hypervisors* so that you can shut them off to save energy until they are needed again.
- *Move VMs off site* if you are expecting to shut down a data center or you are expecting a hurricane or other catastrophe to hit your data center.

Live migration, in particular, is valuable if you need work to continue on the VMs without interruption. The key to getting live VM migration to work is setting up your environment properly. Make sure the following things are in place (keep in mind that these are the kinds of features that something like Red Hat Virtualization does for you):

- Shared networked storage among the hypervisors.
- The same network interfaces configured on each hypervisor.
- Compatible CPUs between hypervisors. (Often, a set of hypervisors have the exact same hardware.)
- A fast network connection between the hypervisors and storage.
- The same or similar versions of virtualization software on the hypervisors. (In our case, we used Fedora 30 on both and installed them similarly.)

With all that in place, live migration requires only a few steps to get going.

Step 1: Identify other hypervisors

Assuming the Virtual Machine Manager window is still up and running on one of your hypervisors, go to that window and do the following to connect to the other hypervisor:

a. **Connect to the hypervisor.** Select File ➪ Add Connection. The Add Connection window should appear.

b. **Add the connection**. Select the Connect to Remote Host check box, choose SSH as the method, use the user name root, and type the hostname of the other hypervisor (for example, `host1.example.com`). When you click Connect, you may be prompted to enter a password for the remote hypervisor's root user and enter other information. Note that you might need to install the `openssh-askpass` package to be prompted for the password.

An entry for the new hypervisor should appear on the Virtual Machine Manager window.

Step 2: Migrate running VM to Other hypervisor

Before you can migrate the VM to another hypervisor, you might need to adjust your firewall rules. With the default firewall rules in place, direct libvirt migration will fail. A random TCP port needs to be opened to allow the migration. The default is 49152, but any available, non-privileged port can be chosen. Tunneled migration requires SSH key authentication.

With the Virtual Machine Manager open, right-click your mouse on any VM that is currently running and select Migrate. The Migrate the Virtual Machine window appears, as shown in Figure 27.3:

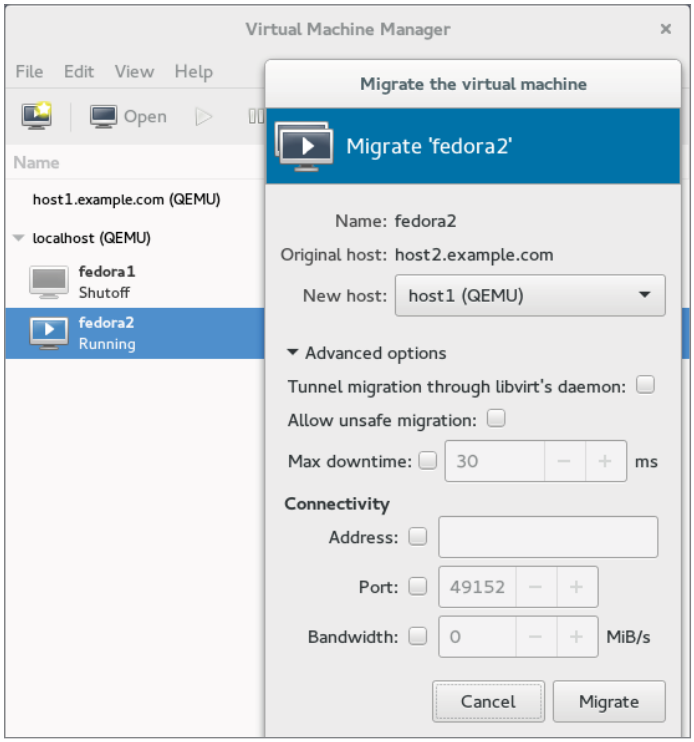
Select the new host. In my example, the VM is currently running on `host2`, so I want to select `host1` as the new host. After a bit of time for the memory image of the VM to copy over to the other host, the VM should appear as running on that host.

If for some reason your migration fails (incompatible CPUs or other problems), you can always shut down the VM on one host and start it again on the other host. Doing that only requires that your shared storage is in place. On the second host, simply run the Create a New Virtual Machine Wizard, but select to run an existing image instead of an installation ISO.

The hypervisor configuration I just demonstrated might suit you well for your home workstation or even for a small business. Although it is beyond the scope of this book to help you develop an entire cloud computing platform, it is within my charter to help you use different cloud platforms to run your Linux systems. The next chapter helps you do just that.

FIGURE 27.3

Choose which hypervisor to migrate the VM to.



Summary

Linux is the foundation on which most of today's emerging cloud technologies are being built. This chapter describes many of the basic components that go into building a cloud based on Linux and other open source technologies. It then helps you to learn about some of those basic technologies by setting up a couple of hypervisors and launching virtual machines.

Exercises

The exercises in this section describe tasks related to setting up a hypervisor (KVM host computer) and using it to run virtual machines. If you are stuck, solutions to the tasks are shown in Appendix B. Keep in mind that the solutions shown in Appendix B are usually just one of many ways to complete a task.

Although the example shown in this chapter for setting up hypervisors uses three physical machines, these exercises can be done on a single physical machine.

1. Check your computer to see if it can support KVM virtualization.
2. Install a Linux system along with the packages needed to use it as a KVM host and to run the Virtual Machine Manager application.
3. Make sure that the `sshd` and `libvirtd` services are running on the system.
4. Get a Linux installation ISO image that is compatible with your hypervisor and copy it to the default directory used by Virtual Machine Manager to store images.
5. Check that the default network bridge (`virbr0`) is currently active.
6. Install a virtual machine using the ISO image you copied earlier.
7. Make sure that you can log into and use the virtual machine.
8. Check that your virtual machine can connect to the Internet or other network outside of the hypervisor.
9. Stop the virtual machine so that it is no longer running.
10. Start the virtual machine again so that it is running and available.

CHAPTER 28

Deploying Linux to the Cloud

IN THIS CHAPTER

Creating Linux cloud images

Deploying a cloud image with `virt-manager` (`libvirtd`)

Deploying a cloud image to OpenStack

Deploying a cloud image to Amazon EC2

To get a new Linux system to use, instead of just running, a standard installation program from a physical DVD, you can get a Linux image and deploy it to a cloud. One way to do that is to take a generic Linux image (one that is bootable but unconfigured) and provide information to configure it to suit your needs. Another way is to go to a cloud provider, choose an image, click through selections to configure it, and launch it.

The point is that cloud computing is offering up new ways to start up and use Linux systems. In Chapter 27, I had you do a standard Linux installation to create a virtual machine that runs on a Linux hypervisor. In this chapter, I will show you how to use cloud images to start up a fresh Linux system.

First, I describe how to use `cloud-init` to combine a Linux cloud image manually with configuration information in order to allow it to run in a variety of environments. Next, I tell you how a similar process is done on an OpenStack cloud or an Amazon Elastic Compute Cloud (EC2) by clicking through easy-to-use cloud controllers to choose images and settings to run the Linux cloud instance that you want.

Getting Linux to Run in a Cloud

Cloud platforms are great for spinning up new virtual machines quickly and efficiently. They can do so because a fresh install is not required each time you want a new instance of an operating system.

Public clouds, such as Amazon EC2 (`http://aws.amazon.com/ec2`), offer instances of different Linux distributions for you to start and use. You choose a Linux instance, such as Ubuntu, Red Hat

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Enterprise Linux (RHEL), or SUSE Linux Enterprise Server (SLES), which is tuned for specific purposes. For example, there are instances that are optimized for high-performance processing or memory-intensive applications.

The content of a cloud instance tends to be generic in nature. It is expected that more information is attached to the image by the cloud user or the cloud provider using a service such as `cloud-init`. This information falls into two general categories: `meta-data` and `user-data`:

meta-data Included with `meta-data` is information that is needed before the image boots. This is data that is outside of the contents of the image and is typically managed by the cloud provider. Some of this data comes from the fact that things such as storage, memory, and processing power are drawn from a pool of resources rather than from the physical machine on which you are installing. So, the `meta-data` tells the cloud provider how many of those resources, and possibly others, to allocate early in the process of starting up the instance.

user-data The `user-data` information is inserted into the operating system that exists on the image. This is data that the person using the virtual machine provides. This might include a user account and password, configuration files, commands to run on first boot, the identities of software repositories, or anything else that you might want to run or change within the operating system itself.

When you go to run a Linux instance in a cloud environment, you typically enter the `meta-data` and `user-data` information by clicking check boxes and filling in forms from a web-based cloud controller (such as the OpenStack Dashboard or Red Hat Virtualization Manager). The information may not be identified as `meta-data` and `user-data` when you configure the instance through the cloud controller.

The cloud you use to run your Linux virtual machines may be a public cloud, a private cloud, or a hybrid cloud. The type of cloud you choose may depend on your needs and your budget:

Public cloud Amazon EC2 and Google Compute Engine are examples of cloud platforms that let you launch and use Linux virtual machines from a web-based interface. You pay for the time that the instance is running. The amount of memory, storage, and virtual CPUs you use to run the service are also figured into the costs. The advantage of public clouds is that you don't have to purchase and maintain your own cloud infrastructure.

Private cloud With a private cloud, you put your own computing infrastructure in place (hypervisors, controllers, storage, network configuration, and so on). Setting up your own private cloud means taking on more up-front costs to own and maintain infrastructure. But it gives you added security and control of your computing resources. Because you control the infrastructure, you can create the images to which users have access in your OpenStack infrastructure and account for user usage of that infrastructure in your own way.

Hybrid cloud Many companies are looking toward hybrid cloud solutions. A hybrid cloud can allow multiple cloud platforms to be managed by a central facility. For example, Red Hat CloudForms can deploy and manage virtual machines on OpenStack, VMware vSphere, and Red Hat Enterprise Virtualization platforms, provisioning different types of workloads to appropriate environments. At times of peak demand, CloudForms can also direct virtual machines to run on Amazon EC2 clouds. These cloud environments have different ways of provisioning and configuring virtual machines. However, the features that clouds need to provide to virtual machine management are similar. Having an understanding of those features can help you when you configure a Linux system to run in a cloud.

To help you get a better feel for configuring Linux cloud instances, in the next sections I'll describe how `cloud-init` works to configure Linux cloud instances. I'll then help you create your own `meta-data` and `user-data` files and apply them to your cloud instance so the information can be used when the cloud image boots.

Creating Linux Images for Clouds

Think about what you did when you installed a Linux system in Chapter 9, "Installing Linux." During the manual installation process, you set a root password, created a regular user account and password, possibly defined your network interfaces, and did other tasks. The information you entered became a permanent part of the operating system that remained each time you booted the system.

When you start with a prebuilt cloud image as your Linux system, you can use `cloud-init` to get a Linux system ready to run. The `cloud-init` facility (`http://launchpad.net/cloud-init`) sets up a generic virtual machine instance to run in the way that you want it to run without going through an install process. The next section describes some ways of using `cloud-init`.

Configuring and running a cloud-init cloud instance

In the next procedure, I show you how to create data manually; the data can be combined with a bootable Linux cloud image so that when that image boots, it is configured based on your data. Combining data with the image at runtime allows you to change the data each time before the image is run instead of installing it permanently in the image.

I suggest that you run this procedure on one of the hypervisors you configured in Chapter 27, "Using Linux for Cloud Computing." This not only allows you to create the customized data for your Linux cloud image, but it also lets you run that image as a virtual machine on that hypervisor.

To add data and run an existing cloud image, this procedure requires you to obtain a cloud image, create data files, and generate a new image that combines those elements. This procedure is meant to be very simple to get a cloud image booted. Later, I will tell you

how to add more features to these data files. To configure and run a cloud image, follow these steps:

1. **Create `cloud-init meta-data` file**. Create a file named `meta-data` to hold data that identifies information about the cloud instance from the outside. For example, you can add a name to identify the instance (`instance-id`), a hostname (`local-hostname`), and other information. To keep your first try simple, I assign only two fields. (You can set them to any names you like.)

   ```
   instance-id: FedoraWS01
   local-hostname: fedora01
   ```

2. **Create `cloud-init user-data` file**. Create a file named `user-data` to hold data that configures inside the operating system on the image itself. For this simple case, I just set a password for the default user (fedora) to `cloudpass` and ensured that `cloud-init` does not expire the password:

   ```
   #cloud-config
   password: cloudpass
   chpasswd: {expire: False}
   ```

3. **Combine data into a separate image.** With the `meta-data` and `user-data` files in the current directory, create an ISO image that contains that data. Later, we present this image as a CD-ROM to the Linux image so `cloud-init` knows how to configure the Linux image. (Install the `genisoimage` and `cloud-init` packages first, if you haven't already. The `cloud-init` package isn't required on the hypervisor.)

   ```
   # yum install genisoimage cloud-init
   # genisoimage -output fedora31-data.iso -volid cidata \
         -joliet-long -rock user-data meta-data
   ```

4. **Get a base cloud image**. Cloud images for Ubuntu, Fedora, and RHEL are configured for use with `cloud-init`. Get an official Fedora cloud image (images for other distributions are described later), and do the following:

   a. **Go to** getfedora.org. Open a web browser, and go to `https://getfedora.org/en/cloud/download/`.

   b. **Click OpenStack**. Click the Download button that appears for the OpenStack image in order to get a `qcow2` image that can be used in an OpenStack environment. The image name is something like `Fedora-Cloud-Base-31-1.9.x86_64.qcow2`.

5. **Snapshot the image**. You probably need to run this procedure a few times before you get the exact image that you want. So, instead of using the downloaded image directly, make a snapshot of it. To keep track of my versions, I added `01` to the new snapshot name:

   ```
   # qemu-img create -f qcow2 \
      -o backing_file=Fedora-Cloud-Base-31-1.9.x86_64.qcow2 \
      Fedora-Cloud-Base-01.qcow2
   ```

6. **Copy files to the images directory.** It's good practice to copy images to the `/var/lib/libvirt/images/` directory when you are using them on a hypervisor (`libvirtd` service). For example, to copy the cloud image and data image to that directory, type the following:

```
# cp Fedora-Cloud-Base-31-1.9.x86_64.qcow2 \
    Fedora-Cloud-Base-01.qcow2 \
    fedora31-data.iso          \
    /var/lib/libvirt/images/
```

7. **Start the cloud instance**. With the files in place, run the following commands to start an instance of your cloud image:

```
# cd /var/lib/libvirt/images
# virt-install --import --name fedora31-01 --ram 4096 --vcpus 2 \
  --disk path=Fedora-Cloud-Base-01.qcow2,format=qcow2,bus=virtio \
  --disk path=fedora21-data.iso,device=cdrom \
  --network network=default &
```

The previous `virt-install` example shows that the virtual machine is assigned to consume 4Gb of RAM (`--ram 4096`) and two virtual CPUs (`--vcpus 2`). The RAM and VCPU values on your system may be different, depending on your computer's resources.

At this point, a virtual machine named `fedora31-01` is running on your hypervisor. As the virtual machine boots up, a console window should open allowing you to log into the new cloud virtual machine.

Investigating the cloud instance

To investigate the cloud image that we created, you can open up the running instance and look inside. One way to do that, if it is not already open, is to open the virtual machine with `virt-viewer`:

```
# virt-viewer fedora31-01
```

From the console window that appears, use the data that we added to the image in order to log in. Use `fedora` as the user and `cloudpass` as the password to log in. The `fedora` user has `sudo` privileges, so you can use that account to investigate the instance by entering some commands.

Here you can see where the `user-data` was copied into the instance:

```
$ sudo cat /var/lib/cloud/instances/FedoraWS01/user-data.txt
#cloud-config
password: cloudpass
chpasswd: {expire: False}
```

The basic cloud configuration is done in the `/etc/cloud/cloud.cfg` file. You can see here that the root user account is disabled by default. At the bottom of the file, you can see that the user named `fedora` is the default user and has `sudo` privileges without requiring a password.

```
$ sudo cat /etc/cloud/cloud.cfg
users:
 - default
disable_root: 1
...
system_info:
  default_user:
    name: fedora0
    lock_paswd: true
    gecos: Fedora Cloud User
    groups: [wheel, adm, systemd-journal]
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    shell: /bin/bash
  distro: fedora
  paths:
    cloud_dir: /var/lib/cloud
    templates_dir: /etc/cloud/templates
  ssh_svcname: sshd

# vim:syntax=yaml
```

You can see other things in the `cloud.cfg` file as well. You can see which `cloud_init_modules` run during initialization (such as those that set the hostname or start `rsyslog` logging). You can see `cloud_config_modules` that set the locale and the time zone and run further configuration tools (such as Chef and Puppet).

Because `yum` repositories are enabled, provided that you have an available network connection (DHCP should have assigned addresses to the virtual machine by default), you can install any packages available from the Fedora repositories.

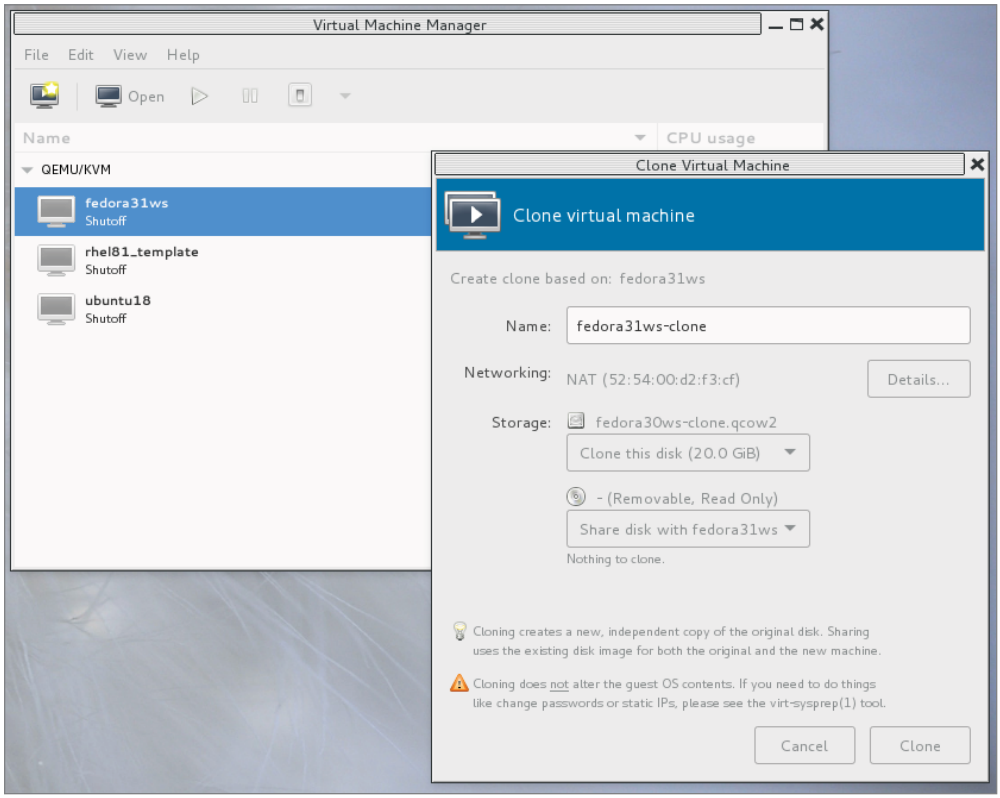
Cloning the cloud instance

If you decide that you like the cloud instance you created, you can save a copy of it to run later by making a clone of the two images (cloud and data image) that make up the cloud instance. To create a clone of the running cloud instance, using `virt-manager`, do the following:

1. **Launch `virt-manager`.** On the host system running the virtual machine, run the `virt-manager` command or start Virtual Machine Manager from the Activities screen on your desktop.
2. **Pause the virtual machine.** Right-click the virtual machine instance entry in the `virt-manager` window and select Pause. This makes the virtual machine inactive for the moment.
3. **Clone the virtual machine.** Right-click the virtual machine instance entry again and select Clone. The Clone Virtual Machine window appears, as shown in Figure 28.1

FIGURE 28.1

Cloning lets you save a permanent copy of a cloud instance.



4. **Choose clone settings**. For the cloud-based image and the data image, you can choose either to make new copies or share them with the existing virtual machine. After you do, select Clone.

The cloned cloud instance is now available to start, stop, and otherwise manage as you like from the Virtual Machine Manager window or the `virsh` command. A great advantage of making clones is that you can make any changes that you like to them, without having to change the originals. Just delete the clone when you are done, and you can quickly generate a new one when you need it.

Expanding your cloud-init configuration

You can add much more information to your `meta-data` and `user-data` files to configure your cloud instances. Examples of `cloud-init` settings can be found on the Cloud-Init

Config Examples page (`http://cloudinit.readthedocs.org/en/latest/topics/examples.html`). The following sections show examples of settings that you can add to your `user-data` files.

NOTE

The `user-data` and `meta-data` files are in yaml format. The yaml format uses indents and well-known delimiters. Items in a list are preceded by a hyphen and a space. Keys and values are separated by a colon and a space. If you are not familiar with YAML, I recommend digging around the Yaml Project site (`https://github.com/yaml`).

Adding ssh keys with cloud-init

Instead of using passwords to log into your cloud instances, you can use key-based authentication along with the `ssh` command to log in over the network. This is commonly used by cloud providers to allow user access to cloud images.

If you have already generated public and private `ssh` keys for the user account that you plan to use to `ssh` into the cloud instance, you can use that public key for this procedure. If you had generated an RSA key pair, the public key is located in the `id_rsa.pub` file by default:

```
# cat $HOME/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDMzdq6hqDUhueWzl7rIUwjxB/rrJY4
oZpoWINzeGVf6m8wXlHmmqd9C7LtnZg2P24/ZBb3S1j7vK2WymOcwEoWekhbZHBAyYeqX
KYQQjUB2E2Mr6qMkmrjQBx6ypxbz+VwADNC
wegY5RCUoNjrN43GVu6nSOxhFf7hv6dtCjvosOvtt0979YS3UcEyrobpNzreGSJ8FMPM
RFMWWg68Jz5hOMCIE1IldhpODvQVbTNsn/STxO7ZwSYV6kfDj0szvdoDDCyh8mPNC1kI
Dhf/qu/Zn1kxQ9xfecQ+SUi+2IwN69o1fNpexJPFr+Bwjkwcrk58C6uowG5eNSgnuu7G
MUkT root@host2.example.com
```

The public key from that file is typically copied to the `$HOME/.ssh/authorized_keys` file for the user on the remote system to which you wish to log in. We can have the key added to that file on our cloud instance using entries in the `user-data` file that looks like the following:

```
users:
  - default
  - name: wsmith
    gecos: William B. Smith
    primary-group: wsmith
    sudo: ALL=(ALL) NOPASSWD:ALL
    lock-passwd: true
    ssh-authorized-keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDMzdq6hqDUhueWzl7rIUwj
xB/rrJY4oZpoWINzeGVf6m8wXlHmmqd9C7LtnZg2P24/
ZBb3S1j7vK2WymOcwEoWekhbZHBAyYeqXKYQQjUB2E2Mr6qMkmrjQBx6ypxbz+V
wADNCwegY5RCUoNjrN43GVu6nSOxhFf7hv6dtCjvosOvtt0979YS3UcEyrobpNz
reGSJ8FMPMRFMWWg68Jz5hOMCIE1IldhpODvQVbTNsn/
STxO7ZwSYV6kfDj0szvdoDDCyh8mPNC1kIDhf/qu/
Zn1kxQ9xfecQ+SUi+2IwN69o1fNpexJPFr+Bwjkwcrk58C6uowG5eNS
gnuu7GMUkT root@host2.example.com
```

From the previous information, you can see that wsmith is the default user. The gecos entry is typically the user's full name, used in the fifth field of the /etc/passwd file. The password is locked for this user. However, because the ssh-rsa entry from my root account on host2.example.com is provided here under ssh-authorized-keys for the user, I can log into the cloud instance as wsmith over ssh without entering a password (provided my private key is associated with that public key).

Adding software with cloud-init

You aren't limited to the software already on your cloud image. Inside your user-data file, you can define YUM repositories (in Fedora and RHEL) or apt repositories (in Ubuntu or Debian) and then identify any packages that you want to have installed when the cloud instance starts.

The following example shows what entries in a user-data file might look like to add a YUM repository (for Fedora or RHEL) to your cloud instance and then install packages from that repository or any other enabled repository:

```
myownrepo:
    baseurl: http://myrepo.example.com/pub/myrepo/
    enabled: true
    gpgcheck: true
    gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-MYREPO
    name: My personal software repository
packages:
 - nmap
 - mycoolcmd
 - [libmystuff, 3.10.1-2.fc21.noarch]
```

In the example just shown, a new yum repository is created in the file /etc/yum.repos.d/myownrepo.repo. A gpgkey is provided to check the validity of installed packages, and GPG checking is turned on. After that, the nmap package is installed (that's in the standard Fedora yum repository), the mycoolcmd package is installed (from my private repository), and a specific version of the libmystuff package is installed.

Configuring apt software repositories for Ubuntu is done a bit differently. Failsafe primary and security apt package mirrors are configured by default (in the cloud.cfg file in the image), along with settings to cause the instance, if run in an Amazon EC2 cloud, to search the closest region for packages. To add more repositories, entries in your user-data file could look like the following:

```
apt_mirror: http://us.archive.ubuntu.com/ubuntu/
apt_mirror_search:
 - http://myownmirror.example.com
 - http://archive.ubuntu.com
packages:
 - nmap
 - mycoolcmd
 - [libmystuff, 3.16.0-25]
```

The myownmirror.example.com entry tells apt to use your own private apt repository to search for packages. Note that packages you want to install can be entered in basically the same format as you did with Fedora, although specific version information (if entered) might look different in some cases.

You can add many other settings to your user-data and meta-data files. Again, refer to the Cloud-Init Cloud Config Examples page (http://cloudinit.readthedocs.org/en/latest/topics/examples.html) for details.

Using cloud-init in enterprise computing

So far, the cloud-init examples in this chapter have focused on taking a cloud image, manually adding configuration data, and running it as a virtual machine temporarily on your local hypervisor. This approach is useful if you want to understand how cloud-init works and the opportunities you have for tuning cloud images to your specifications. This approach doesn't scale well, however, if you are managing large enterprises of virtual machines.

Cloud-init supports the concept of *datasources*. By placing user-data and meta-data in a datasource, you don't have to inject that information manually into a cloud instance, as we did earlier in this chapter. Instead, when the cloud-init service starts running on the instance, it knows to not only look on the local system for data sources, but also outside of it.

For Amazon EC2 clouds, cloud-init queries a particular IP address (http://169.254.169.254/) for data. For example, it may check http://169.254.169.254/2009-04-04/meta-data/ for meta-data and http://169.254.169.254/2009-04-04/user-data/ for user-data. This allows the configuration data to be stored and accessed from a central location.

As for what might be inside the meta-data and user-data, far more complex configuration schemes can be developed for deployment of your cloud instances. Cloud-init supports configuration tools, such as Puppet (http://puppetlabs.com/puppet/puppet-open-source) and Chef (https://www.chef.io/chef/). These tools let you apply scripts of configuration information to your cloud instances, even doing such things as replacing components or restarting services as needed to return the system to a desired state.

At this point, however, my job is not to make you into full-blown cloud administrators (a few hundred pages ago, you could have been a Linux novice). Instead, I want you to understand what you will be dealing with if you eventually land in a cloud data center . . . because many people believe that most data centers will be managed as cloud infrastructures in the not-too-distant future.

So far in this chapter, you have looked at the inside of configuring Linux for cloud computing. Next, let's step back and look at how you can use two of the most popular Linux-based cloud platforms to run your own Linux-based virtual machines: OpenStack and Amazon EC2.

Using OpenStack to Deploy Cloud Images

With *OpenStack*, you get a continually evolving platform for managing your physical cloud computing infrastructure, as well as the virtual systems that run on it. OpenStack lets you deploy your own private cloud or offer it up to the world as a public cloud.

Rather than have you set up your own OpenStack cloud, I'm going to show how you can use OpenStack to deploy virtual machines from an OpenStack Dashboard. If you want to try it yourself, OpenStack is available in the following ways:

Linux distributions Fedora, Ubuntu, and CentOS have free versions of OpenStack that you can deploy yourself. Red Hat Enterprise Linux offers a version of OpenStack that is available by subscription. It's tricky to set up. Some all-in-one setups for OpenStack can run on a single machine, but I think you will have a better experience if you start with three physical machines: one controller node and two hypervisors.

Public OpenStack clouds You can try out public OpenStack clouds for varying costs. A list of public OpenStack clouds is available from the OpenStack project site (`http://www.openstack.org/marketplace/public-clouds/`).

My first point is to help you run a Linux system in a cloud when you lack the capacity to do what you want on your own computers. However, my other point is to show you how a cloud provider's web-based interface (like OpenStack Dashboard) can greatly simplify the cloud configuration we did manually with `cloud-init` earlier in this chapter.

Starting from the OpenStack Dashboard

I'm going to start with an OpenStack setup that is already in place. The OpenStack environment's administrator has created a project for me called `cnegus-test-project` and a user account (`cnegus`) that lets me access that project. Here's what I plan to do:

- **Configure networking**. Just as I would set up a router and physically plug my computers into that router, I'm going to set up a virtual network. That virtual network will include a set of addresses that are distributed to my virtual machines via DHCP.
- **Configure virtual machines**. I'll step through the process of choosing, configuring, and deploying a couple of virtual machines.

The version of OpenStack used for this demonstration is Red Hat OpenStack Platform (RHEL-OSP). However, the experience would be similar on any OpenStack environment. The next section shows you how to start configuring your network.

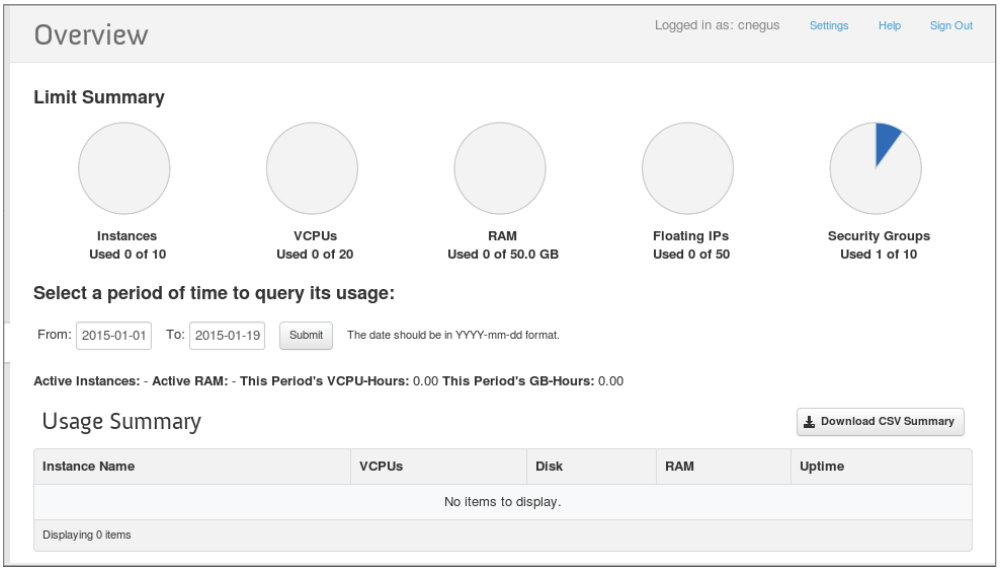
Configuring your OpenStack virtual network

Follow these steps to configure your OpenStack virtual network.

1. **Log in to OpenStack**. Using the username and password assigned to you by the OpenStack administrator, log in to the OpenStack Dashboard from your web browser. You should see an Overview screen, similar to the one shown in Figure 28.2:

FIGURE 28.2

Log in to the OpenStack Dashboard.



2. **Create a network.** To create a network, from the left column on the Overview screen, select Networks. From the Networks screen that appears, create a new network as follows (the examples I used are in parentheses):
   a. **Select the Create Network button.**
   b. **On the Network tab, type a network name (**`mynet`**).**
   c. **On the Subnet tab, type a subnet name (**`mysub01`**), network address (**`192.168.100.0/24`**), IP version (**`IPv4`**), and gateway IP (**`192.168.100.1`**). Leave Disable Gateway unchecked.**
   d. **On the Subnet Detail tab, enter a comma-separated range of IP addresses in the Allocation Pool box.** For my example, I chose `192.168.100.10,192.168.100.50` to hand out a range of IP addresses to clients from 192.168.100.10 to 192.168.100.50. Get a name server suggestion from the administrator of your OpenStack cloud or use a public DNS server (such as Google's 8.8.8.8 or 8.8.4.4)
   e. **Select Create to create the new network.** The new network appears on the Networks screen.
3. **Create a router.** For your virtual machines to be able to access the Internet, you need to identify a router that is attached to your private network on one

interface and a network that can reach the public Internet on the other. Here's how to do that:

a. **From the left column, select Routers**.

b. **Click the Create Router button**.

c. **Type a router name (**`myrouter01`**) and click Create Router**.

d. **Select the Set Gateway button**.

e. **From the Set Gateway screen, click the External Network box and choose from the available external networks**. Leave the Router Name and Router ID fields as they are. Click Set Gateway. The new router appears on the Routers screen.

4. **Connect your network to the external router**. From the Routers screen (you should still be on that screen), select the name of the router that you just created (`myrouter1`):

   a. **From the Router Details screen, select the Add Interface button.**

   b. **From the Add Interface screen, click the Subnet box and choose the subnet you created earlier (**`mynet: 192.168.100.0/24 mysub01`**).** You shouldn't have to change Router Name or Router ID.

   c. **Click Add Interface**.

5. **View network topology**. Click Network Topology from the left column. Then, hover your mouse button over the router name (`myroute01`). Figure 28.3 shows an example of what your network configuration might look like.

With your networking in place, you can create keys to access your virtual machines in OpenStack.

Configuring keys for remote access

The normal way to configure access to your virtual machines in a cloud environment is to create a public/private key pair that provides secure access to your virtual machines using `ssh` and related tools from your desktop system. The private key is stored in your desktop user's home directory, and the public key is injected into the virtual machine so that you can log in remotely (via `ssh`) to the virtual machine without entering a password. Here's how to set up your keys:

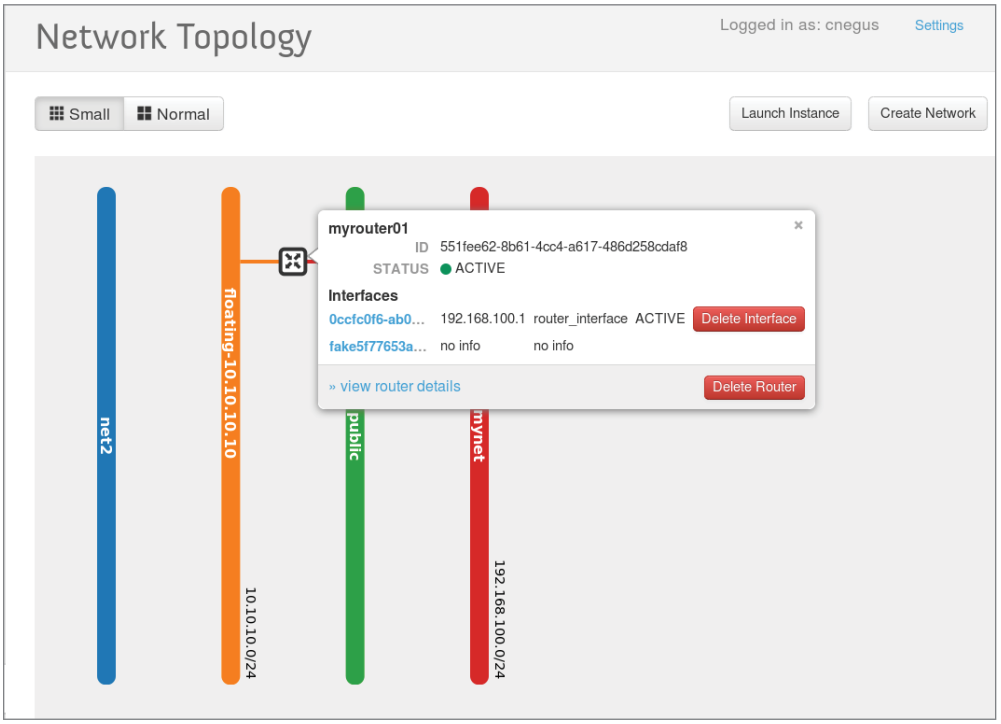
1. **Select Access & Security**. From the left column, select Access & Security.

2. **Create Keypairs**. If you already have a keypair, you can skip to the next step. If not, select the Keypairs tab and click the Create Keypair button. When the Create Keypair window appears, do the following:

   a. **Enter a keypair name (**`mycloudkey`**), and click the Create Keypair button**. A pop-up window asks if you want to open or save the `*.pem` file.

FIGURE 28.3

View your network topology from the OpenStack Dashboard.



b. **Select Save File and click OK**. When prompted where to save it, save it to the `.ssh` directory in your home directory.

You are ready to deploy an OpenStack instance (cloud-based virtual machine).

Launching a virtual machine in OpenStack

To begin launching a new cloud virtual machine instance, go to the left column and select Instances. Then click the Launch instance button. The Launch Instance screen appears. To fill in the information that you need to launch the instance, follow these steps:

1. **Select Details**. From the Details tab, select the following items:
   a. **Availability Zone**. An *availability zone* consists of a group of compute hosts. Separate zones are sometimes created to identify a group of computers that are physically together (such as on the same rack) or that have the same hardware features (so they could be used for the same types of applications). Choose one of the zones from the list.

   b. **Instance Name**. Give the instance any name that helps you remember what it is.
   c. **Flavor**. By choosing a flavor, you allocate a set of resources to your virtual machine instance. The resources include the number of virtual CPU cores, the amount of memory available, the disk space assigned, and ephemeral disk space available. (*Ephemeral space* is space available from the local disk while the instance is running but is not saved when the instance shuts down.) Default flavors include `m1.tiny`, `m1.small`, `m1.medium`, `m1.large`, and `m1.xlarge`. Other flavors can be added by your cloud administrator.
   d. **Instance Count**. By default, this is set to 1, meaning to start one instance. Change the number to start more instances if you like.
   e. **Instance Boot Source**. The instance can be booted from an image, a snapshot, a volume, an image that includes a new volume, or a volume snapshot that includes a new volume.
   f. **Image Name**. Select the image that you want to start. The names typically include the names of the operating systems that you are booting.
   g. **Device size and Device Name (optional)**. If you selected to include a new volume when you chose your instance boot source, you set the size (in GB) and device name for the volume in these fields. For the device name, if you choose `vda` as the device name (for the first disk on a virtual machine), the device representing that device would be `/dev/vda`.

2. **Select Access & Security**. Select the Access & Security tab and choose the keypair that you created earlier.
3. **Select Networking**. Select the Networking tab. From the list of available networks, grab the one that you want with your mouse and drag it into the Selected Networks box.
4. **Add Post-Creation settings**. You can add commands and scripts that configure the system further after it is booted. This is where you can add the kinds of information you added in the `user-data` files described in the sections on `cloud-init` earlier in this chapter.

Select Launch to start up the virtual machine. With the virtual machine running, you can log in to that system by selecting the instance and clicking the Console tab. The virtual machine's console window should present you with a login prompt. If you want to be able to gain access to the virtual machine using `ssh` over the network, go on to the next section.

Accessing the virtual machine via ssh

With your public key injected into your running virtual machine, it is ready for you to log in using `ssh`. However, before you can do that, you must take these steps:

1. **Add floating IP address**. From the OpenStack Dashboard, select the instances, click More on the entry containing the instance, and click Associate Floating IP.

Select the plus sign (+) next to the IP Address box, select a pool that has floating IPs available, and click Allocate IP. The allocated address should appear in the IP Address field. Select the port to be associated and click Associate.

2. **Use `ssh` to access instance**. From a Linux system that has access to the network on which the floating address was assigned, run the `ssh` command to log in. Assuming that your key's `.pem` file was called `mycloud.pem`, the default user on the instance is `cloud-user`, and the IP address is 10.10.10.100, you could enter the following to log in:

```
# ssh -i mycloud.pem cloud-user@10.10.10.100
```

You should be able to log in now without a password. To do administration on the system, you can use the `sudo` command as the default user.

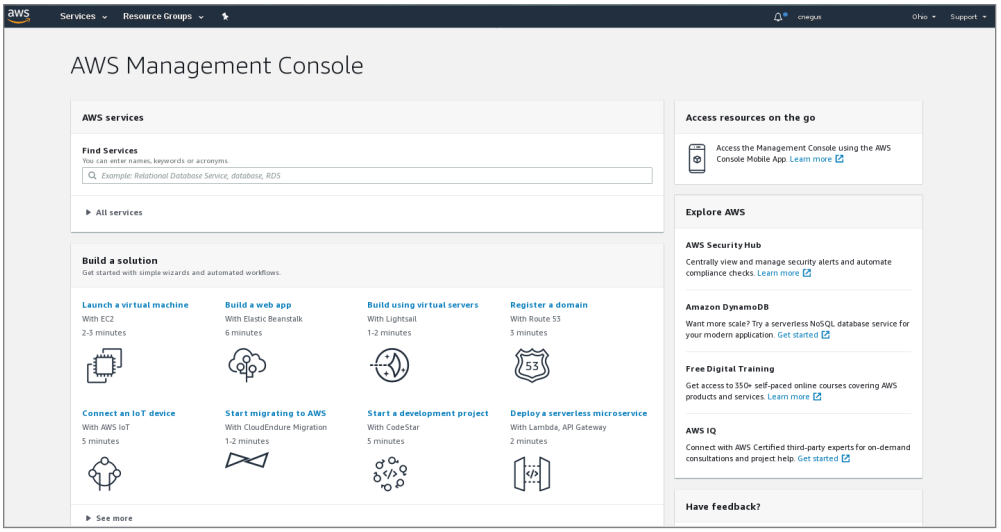
Using Amazon EC2 to Deploy Cloud Images

Amazon Elastic Computer Cloud (Amazon EC2) is a cloud platform that is particularly suited for pay-as-you-go cloud computing. Like OpenStack, it lets you choose from preconfigured virtual machine images and configure them as you need.

To start using Amazon EC2 to launch virtual machines, go to the Getting Started with Amazon Web Services page and follow links to create a new account (`http://aws.amazon.com/getting-started/`). After you log in, the full range of AWS services is displayed. Select Sign In to the Console, and you will see the AWS Management Console, as shown in Figure 28.4:

FIGURE 28.4

Launch cloud instances using the Amazon EC2 Management Console.

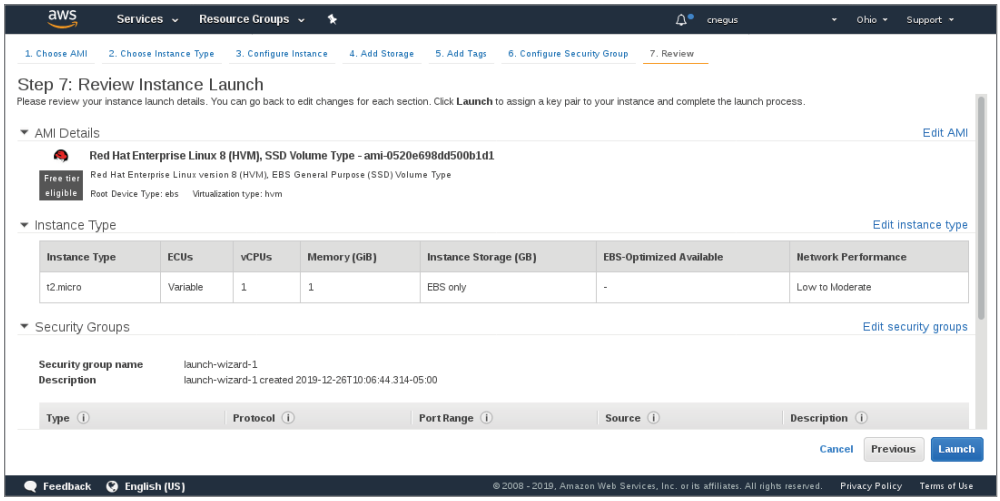


To start your first Linux virtual machine instance, do the following:

1. **Select Launch a Virtual Machine.** You are then given a choice of Linux (Red Hat Enterprise Linux, SUSE Linux, Ubuntu, and so on) and Windows AMIs (Amazon Machine Images) to start up.
2. **Find the image that you want, and click the Select button.**
3. **From the Choose an Instance Type page, select the particular instance type that you want.** Make that selection based on the number of CPUs, amount of memory, type of storage, and networking features.
4. **With the instance type selected, click Next: Configure Instance Details.**
5. **From the Configure Instance Details screen, select an existing VPC, or create a new one.** Then change any other settings. For example, select Enable under Auto-assign Public IP to be able to log in to your instance over the Internet.
6. **Select Review and Launch.** The Review Instance Launch screen appears.
7. **Review the instance settings and select Launch to start the instance.** Figure 28.5 shows an example of a RHEL 8 instance ready to launch.
8. **Select an existing key pair or choose Create a New Key Pair to create a private and public key to use to `ssh` into the instance.**
9. **Select Launch Instances to start the instance.**
10. **Select View Instances to see a list of running instances.** Use the search box to search for a string in the instance name if there is a long list from which to choose.

FIGURE 28.5

Configure and launch a RHEL 8 instance on AWS.



11. **Select your instance and then select the Connect button**. Follow the instructions to use `ssh` to log into the public IP address you created. For example, the command to log in to the AWS instance would look similar to the following:

    ```
    # ssh -i "youraws.pem" ec2-user@ec2-w-xx-yyy-zz.us-east-2.compute
    .amazonaws.com
    ```

12. **When you are done with the instance, you can terminate it by selecting the instance from the Instances page and then selecting Actions ➪ Instance State ➪ Terminate**. When prompted, select Yes, Terminate to remove the instance and its associated storage.

It is important to remember to get rid of the instance when you are done or you will continue to be charged for it.

Summary

Understanding how cloud computing differs from simply installing an operating system directly on computer hardware will help you to adapt as more and more data centers move toward cloud computing. In the beginning of this chapter, I encouraged you to get your hands on some cloud images, combine them with data, and launch them on a local Linux hypervisor to understand how cloud images work.

After that, I demonstrated how you can launch your own virtual images in an OpenStack cloud platform. That included configuring network interfaces, choosing how the virtual instance would run, and launching the virtual image. I also quickly introduced the Amazon Elastic Compute Cloud service, where you can pay to use cloud storage and processing time if you don't have enough computing resources of your own.

The next chapter describes how to use Ansible to automate the deployment of host systems and applications to your data center.

Exercises

The exercises in this section assume that you have already set up a host system as a hypervisor (KVM host computer). You need to use that hypervisor to run the virtual machine created in the exercises. If you are stuck, solutions to the tasks are shown in Appendix B. Keep in mind that the solutions shown in Appendix B are usually just one of multiple ways to complete a task.

1. To be able to create a custom virtual machine image, install the `genisoimage`, `cloud-init`, `qemu-img`, and `virt-viewer` packages.
2. Obtain a cloud image from the Fedora project.
3. Use `qemu-img` to create a snapshot of that image in qcow2 format called `myvm.qcow2`, which you can use later to combine with your own data.

4. Create a `cloud-init` meta data file named `meta-data` that sets the `instance-id` to `myvm` and `local-hostname` to `myvm.example.com`.
5. Create a `cloud-init` user data file called `user-data` that sets the default user's `password` to `test` and sets `chpasswd` never to expire with `{expire: False}`.
6. Run the `genisoimage` command to combine the `meta-data` and `user-data` files to create a `mydata.iso` file that you can combine with a virtual machine image later.
7. Use the `virt-install` command to combine the `myvm.qcow2` virtual machine image with the `mydata.iso` image to create a new virtual machine image named `newvm` that starts running on your hypervisor.
8. Use virt-viewer to open a console to the `newvm` virtual machine.
9. Log into the `newvm` virtual machine, using the `fedora` user and `test` password that you set earlier.

CHAPTER 29

Automating Apps and Infrastructure with Ansible

IN THIS CHAPTER

Understanding Ansible

Installing Ansible

Stepping through a deployment

Running ad-hoc commands

To this point in the book, we have mostly focused on manually configuring individual Linux systems. You have learned how to install software, edit configuration files, and start services directly on the machines where they run. While knowing how to work on individual Linux hosts is foundational to managing Linux systems, by itself it doesn't scale well. That's where Ansible comes in.

Ansible changes the mindset of Linux administration from a focus on single systems to groups of systems. It moves configuration of those nodes from each individual machine to a control node. It replaces the user interface of a shell on each machine with Ansible playbooks that run tasks on other machines over a network.

Although our focus here is on managing Linux systems, Ansible can manage many things around those Linux systems as well. There are Ansible modules for making sure that machines are powered on, that network devices are properly configured, and that remote storage is accessible

In all but the smallest data centers, knowing how to deploy and manage Linux systems and surrounding infrastructure automatically is becoming a requirement for many IT jobs these days. For fully containerized data centers, Kubernetes-based application platforms such as OpenShift are becoming the industry standard for container orchestration and automation (see Chapter 30, "Deploying Applications as Containers with Kubernetes"). For infrastructure and more traditional application deployments, Ansible is becoming a leader.

This chapter takes you through what you should know about Ansible to get started. It then steps you through deploying an application across a set of Linux systems with Ansible and shows you how to work with those systems later by redeploying playbooks and running ad-hoc commands.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Understanding Ansible

Ansible extends, rather than replaces, what you have already learned about Linux. At its most basic level, Ansible comprises the following:

- An automation language that describes the tasks that you want to perform to reach a particular state. These are gathered into *playbooks*.
- The automation engine that is used to run the playbooks.
- Interfaces you can use to manage and secure playbooks and other automation components, implemented with commands and RESTful APIs.

Using inventories (that define sets of hosts) and playbooks (that define sets of actions to take on those hosts), Ansible configures host systems in the following ways:

Simple feature configuration: You create inventories and playbooks as plain-text files, where you identify Linux components that are acted upon by modules. No coding experience is required.

Setting the results that you want: What you describe here are resources that define the state you want a feature to be in on a node. That state can be a `systemd` service running, a network interface with particular addresses set, or a disk partition of a certain size created. If, for some reason, the state changes for a feature, you can run a playbook again to have Ansible return a node to the intended state.

SSH connections: By default, each host node must be running an SSH service that is configured to allow Ansible to communicate to it from the control node. Key-based authentication to regular user accounts allows this to happen, with `sudo` available when root privilege escalation is needed. Because you are using an SSH service that is probably already running on the host, you don't need to run additional agents or configure special firewall rules for this to work.

Once you learn the basics about how Ansible works, you can do a wide range of advanced, complex activities, such as the following:

Provisioning infrastructure: Using Ansible, you can provision the infrastructure that your applications need, whether that is installing operating systems on bare metal or as hypervisors (along with their virtual machines), setting up storage devices, or configuring network devices. In each of those cases, Ansible can leverage your existing provisioning tools so that they can all be managed in one place.

Deploying applications: By describing the desired state of your applications, Ansible can not only use tasks to deploy sets of applications across multiple nodes and devices, but it can also replay those playbooks to return an application to its desired state when a feature may have broken or have been changed unintentionally.

Managing containers and operators: Recent additions to Ansible allow it to deploy containerized applications to a Kubernetes infrastructure such as OpenShift. Operators in OpenShift, which can be managed by an Ansible Operator, can not only define the state of containerized applications, but they can also respond to changes

in real time and make upgrades easier. See the description of the Ansible Operator for details (`https://www.ansible.com/blog/ansible-operator`).

Managing networking and storage: Tasks that are often done manually to configure, test, validate, and enhance your networking infrastructure can be automated with Ansible. Tons of commercial and community playbooks are available that offer the same Ansible intuitive tools that you use to deploy Linux systems, but they are made for specific network (`https://docs.ansible.com/ansible/latest/network/index.html`) and storage (`https://docs.ansible.com/ansible/latest/modules/list_of_storage_modules.html`) devices and environments.

Managing cloud environments: Just as you can deploy infrastructures to bare metal, Ansible offers tools for provisioning infrastructure and applications to cloud environments. For Amazon Web Services (AWS) alone, there are about 200 Ansible modules available for managing infrastructure and applications. Modules for Alibaba, Azure, Google, and a few dozen other cloud environments are also available.

Exploring Ansible Components

When a playbook is run, it acts on one or more target host systems (represented by *inventories*) and executes items referred to as *plays*. Each *play* contains one or more *tasks* that are set to be achieved by that play. To carry out a task, the task calls *modules*, which are executed in the order that they appear. Before you start using Ansible, it helps to understand a little more about these components.

Inventories

By gathering host systems (nodes) that you want to manage in what are referred to as *inventories*, you can manage machines that are similar in some way into groups. Similarities could include the following:

- Located in a similar location
- Provide the same kind of service
- Assigned to a particular stage in a process, such as sets of machines for development, testing, staging, and production

Joining hosts together into more than one group allows them to be acted on based on these different kinds of attributes. For example, `host01` might be both in a group called `newyork` (for its location) and a group called `ftp` (for the application it provides). Tasks run on those inventory groups might allow each host to get network settings based on its location and the applications it runs based on its purpose, respectively.

There are multiple ways of creating inventories. You can set a line of static servers or create a range of systems. You can also use dynamic lists of servers from cloud providers, such as Azure, AWS, and GCP.

Using variables, you can assign attributes to a set of hosts in an inventory. Those variables can configure such things as the port from which a service is available from a host, a

timeout value for a service, or the location of a service used by a host (such as a database for a Network Time Protocol server).

Like playbooks, inventories can be simple text files. They can also be implemented from an inventory script.

Playbooks

Playbooks are created as YAML-formatted files that describe the end state of something. That something can cause software to be installed, applications to be configured, or services to be launched. It can focus on the application alone, or it can include the entire environment (networking, storage, authentication, or other feature) surrounding that application.

Playbooks are meant to be reusable—to deploy the same components later, be adapted for other components, or replayed to reestablish the original intent of a specific instance of the playbook. Because playbooks are intended for reuse, many people keep their playbooks under source control. In that way, you can track changes over time and make the playbooks easily available.

Plays

Inside a playbook is one or more *plays*. Each play has a target, such as a `hosts` identifier that tells the playbook which host systems to act on. That can be followed by a `remote_user` that tells the playbook which user to authenticate to on the host. The play can also indicate that it needs to escalate privileges with `sudo` before it starts executing the tasks. After that, there can be one or more tasks to define the actual activity that is carried out on the hosts.

Tasks

At the most basic level, each task runs one or more modules. A task provides a way to associate the module being run with the parameters and return values associated with that module.

Modules

There are hundreds of Ansible modules available today, with more being created all the time. When run, a *module* makes sure that a requested state is achieved by checking that intended state, as indicated by parameters that are provided, and if the target is not in that state, then doing what needs to be done to get there. The Module Index organizes those modules by category: (`https://docs.ansible.com/ansible/latest/modules/modules_by_category.html`).

Examples of modules include `yum`, `mysql_db` and `ipmi_power`. The `yum` module can install, remove, or otherwise manage software packages and repositories from the YUM facility. A `mysql_db` module lets you add or remove a MySQL database from a host. The `ipmi_power` module lets you check the state of computers with IPMI interfaces and make sure they get to the requested state (on or off).

Conditionals can be applied to each task. For example, with the `yum` module, you can condition whether or not to install a package by its state. You could say that if the state of the package is `installed`, then don't install the package (even if a newer version is available). However, if you use `latest`, then a newer version of the package will be installed if the current package is not the latest.

Parameters let you add information to modify the task. For example, with the `user` module, when you add a user to a system, you can identify the user's name, password, uid, and shell.

Besides setting up modules to be executed from playbooks, you can also run modules directly from the command line. This is useful if you want to act on a host immediately, without running an entire playbook. For example, you can ping a set of hosts to make sure that they are running or check the status of a service. (See the section "Running Ad-Hoc Ansible Commands" later in this chapter for further information.)

To learn more about a particular module, go to the Ansible documentation website (select Modules from the `https://docs.ansible.com` page) or use the `ansible-doc` command. For example, to learn more about how to use the `copy` module to copy files to a remote location, enter the following:

```
# ansible-doc copy
> COPY    (/usr/lib/python3.7/site-packages/ansible/modules/files/
copy.py)

        The 'copy' module copies a file from the local or
        remote machine to a location on the remote machine...
```

Most modules have return values to provide information about the results of that module's action. Common return values include Booleans, indicating if the task was successful (`failed`), whether or not the task was skipped (`skipped`), or if the task had to make changes (`changed`).

Roles, imports, and includes

As your collection of playbooks grows, you may find that you want to break up those playbooks into smaller pieces that you can include in multiple playbooks. You can separate parts of a large playbook into separate, reusable files, then call those files into the main playbook using *includes* and *imports*. *Roles* are similar, but can they encompass more things than tasks, such as modules, variables, and handlers.

For information on using includes, imports, and roles, see "Creating Reusable Playbooks" at `https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse.html`.

Stepping Through an Ansible Deployment

To get you started using Ansible, we are going to step through a procedure to deploy a web service to a set of hosts. After installing Ansible, the procedure shows you how to create

the inventory and playbook which you need to deploy that service. Then it shows how to use `ansible-playbook` to actually deploy the playbook.

Prerequisites

To get started, I created four virtual machines with the following names:

```
ansible     Used as the Ansible control node
host01      First target node
host02      Second target node
host03      Third target node
```

Then I ran the following steps to prepare to use those hosts with Ansible:

1. Installed Fedora on each of the virtual machines (RHEL should work as well).
2. For each of the three target nodes (`host01`, `host02`, and `host03`), I made sure to do the following:
   a. Have the SSH service running and available (opening TCP port 22 if necessary) to the Ansible control node.
   b. Create a non-root user account. Later, when you use the playbook, add the `--ask-become-pass` option to be prompted for the password that you'll need to escalate privileges.
   c. Set a password for that user.

When running Ansible, I use the regular user account to connect to each system, then I escalate to root privilege using `sudo`.

Setting up SSH keys to each node

Log in to the control node (`ansible`) and ensure that it can reach the three other nodes that you are configuring. Either make sure that you can reach the hosts through a DNS server or add them to the `/etc/hosts` file on the control node. Then set up keys to access those nodes. For example:

1. As root user, add the IP address and name for each node to which you want to deploy your Ansible playbooks to the `/etc/hosts` file:

   ```
   192.168.122.154   host01
   192.168.122.94    host02
   192.168.122.189   host03
   ```

2. Still on the `ansible` system, generate `ssh` keys so that you can have passwordless communications with each host. You can run this and the later Ansible commands as a regular user on the `ansible` host system:

   ```
   $ ssh-keygen
   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/joe/.ssh/id_rsa): <ENTER>
   ```

```
Created directory '/home/joe/.ssh'.
Enter passphrase (empty for no passphrase): <ENTER>
Enter same passphrase again:
Your identification has been saved in /home/joe/.ssh/id_rsa.
Your public key has been saved in /home/joe/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Wz63Ax1UdZnX+qKDmefSAZc3zoKS791hfaHy+usRP7g joe@ansible
The key's randomart image is:
+---[RSA 3072]----+
|              ...*|
|             .  o+|
|            . . ..|
|           . + +  |
|        S..= * + |
|        o+o + O.o|
|        .ooB.Bo+o|
|          *+O+o.o|
|         ..=BEo  |
+----[SHA256]-----+
```

3. Using `ssh-copy-id`, copy your public key to the root account on each host. The following `for` loop steps through copying the user's password to all three hosts:

```
$ for i in 1 2 3; do ssh-copy-id joe@host0$i; done
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
 "/home/joe/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the
new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed
-- if you are prompted now it is to install the new keys
joe@host01's password: <password>

Number of key(s) added: 1
Now try logging into the machine, with:   "ssh 'joe@host01'"
and check to make sure that only the key(s) you wanted were added.

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
 "/home/joe/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the
new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed
-- if you are prompted now it is to install the new keys

joe@host02's password: <password> ...
```

The next step is to install the `ansible` package on the control node (`ansible`). From that point on, all that work is done from the control node.

Installing Ansible

Ansible software packages are available for RHEL, Fedora, Ubuntu, and other Linux distributions. Because Ansible playbooks are run from a control node, there is no need to install Ansible software on any of the nodes that it targets.

So, start by installing the `ansible` package on the RHEL, Fedora, Ubuntu, or other Linux system that you want to use as your control node. That control node must simply be able to connect to the SSH service running on the host nodes to which you want to deploy.

Install the `ansible` package in one of the following ways:

RHEL 8:

```
# subscription-manager repos \
    --enable ansible-2.9-for-rhel-8-x86_64-rpms
# dnf install ansible -y
```

Fedora:

```
# dnf install ansible -y
```

Ubuntu:

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

With Ansible installed, you can start to build the inventory that provides the targets for the playbooks that you will run.

Creating an inventory

A simple inventory can consist of the name representing the target for a playbook and the host systems associated with that name. To get started, here is an inventory example that contains three groups of static hosts:

```
[ws]
host01
host02
host03

[newyork]
host01

[houston]
host02
host03
```

Adding these entries to the `/etc/ansible/hosts` file makes them available when you run Ansible commands and playbooks.

Although this procedure just deploys to the set of hosts in the ws group, the other two groups illustrate how you might want to set up playbooks for separate tasks based on the location of the machines (newyork and houston).

Authenticating to the hosts

Just to make sure that you can access each host from the Ansible system, ssh to each host. You should not have to enter a password:

```
$ ssh joe@host01
Last login: Wed Feb  5 19:28:39 2020 from 192.168.122.208
$ exit
```

Repeat for each host.

Creating a playbook

This playbook results in web server software being installed and started on the hosts defined earlier in the ws group. Likewise, the playbook checks that firewall software is installed and running, and that port 80 (http port) is open in the firewall to access the web server. I added the following content to a file called simple_web.yaml:

```
---
- name: Create web server
  hosts: ws
  remote_user: joe
  become_method: sudo
  become: yes
  tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present
  - name: Check that httpd has started
    service:
      name: httpd
      state: started
  - name: Install firewalld
    yum:
      name: firewalld
      state: present
  - name: Firewall access to https
    firewalld:
      service: http
      permanent: yes
      state: enabled
  - name: Restart the firewalld service to load in the firewall
changes
```

```
    service:
      name: firewalld
      state: restarted
```

The three hyphens at the beginning of the `simple_web.yaml` playbook indicate the start of the YAML content in the file. Here's a breakdown of the rest of the file:

- **`name`**: The play is identified as "Create web server."
- **`hosts`**: Apply this inventory to the hosts in the `ws` group.
- **`remote_user`**: The regular user that is used to authenticate to each remote system. This is done because it is a good security practice not to allow direct root login to a remote system.
- **`become`**: Enabling this feature (`yes`) tells Ansible to become a different user than the `remote_user` to run the modules in the task.
- **`become_method`**: What feature to use to escalate privilege (`sudo`).
- **`become_user`**: Which user to authenticate to (root).
- **`tasks`**: Starts the section containing the tasks.
- **`name`**: The name is a title given to the task. In the first case, "Install httpd," then "Check that httpd has started," and so on. The next line starts with the name of a module (`yum`, `service`, `firewalld`, and so on).

For `yum`, it says to check if the `httpd` package is `present`, and if it is not, then install it.

For `service`, it checks whether or not the `httpd` daemon is running (`started`). If `httpd` is not running, Ansible starts it.

For `yum`, it says to check if the `firewalld` package is `present`, and if it is not, then install it.

For `firewalld`, make the port for the `http` service (TCP 80) available immediately (`enabled`) and permanently (`permanent: yes`) through the firewall.

For `service`, restart the `firewalld` service (`restarted`) to enable access to the new `http` service firewall port.

Run the playbook

Use the `ansible-playbook` command to run the playbook. To test the playbook before running it live, use the `-C` option. To see more details (at least until you are sure that it's working), add the `-v` option to see verbose output.

Keep in mind that if you run a playbook with `-C`, it cannot fully test the playbook to make sure that it is correct. The reason is that a later step might require that an earlier step be completed before it can be done. In this example, the `httpd` package would need to be installed before the `httpd` service can be running.

Here's an example of running the Ansible playbook in verbose mode:

```
$ ansible-playbook -v simple_web.yaml
Using /etc/ansible/ansible.cfg as config file

PLAY [Create web server] ************************************

TASK [Gathering Facts] **************************************
ok: [host03]
ok: [host02]
ok: [host01]

TASK [Install httpd]
*************************************************************
changed: [host01] => {"changed": true, "msg": "", "rc": 0,
    "results": ["Installed: httpd", ...
changed: [host02] => {"changed": true, "msg": "", "rc": 0,
    "results": ["Installed: httpd", ...
changed: [host03] => {"changed": true, "msg": "", "rc": 0,
    "results": ["Installed: httpd", ...

TASK [Check that httpd has started]
*************************************************************
changed: [host03] => {"changed": true, "name": "httpd",
    "state": "started", "status":
changed: [host02] => {"changed": true, "name": "httpd",
    "state": "started", "status": ...
changed: [host01] => {"changed": true, "name": "httpd",
    "state": "started", "status": ...
...
TASK [Install firewalld]*************************************
changed: [host03] => {"changed": true, "msg": "", "rc": 0, "results":
    ["Installed: firewalld", "Installed: python3-decorator...
changed: [host02] => {"changed": true, "msg": "", "rc": 0, "results":
    ["Installed: firewalld", "Installed: python3-decorator...
changed: [host01] => {"changed": true, "msg": "", "rc": 0, "results":
  ["Installed: firewalld"...


TASK [Firewall access to https]**************************************
****
ok: [host03] => {"changed": false, "msg": "Permanent operation,
    (offline operation: only on-disk configs were altered)"}
ok: [host02] => {"changed": false, "msg": "Permanent operation,
    (offline operation: only on-disk configs were altered)"}
ok: [host01] => {"changed": false, "msg": "Permanent operation,
    (offline operation: only on-disk configs were altered)"}
```

```
PLAY RECAP *********************************************************
host01: ok=6 changed=4 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
host02: ok=6 changed=4 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
host03: ok=6 changed=4 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

The output from `ansible-playbook` steps through each task. The first task (`Gathering Facts`) shows that all three host systems in the `ws` inventory are accessible. What you can't see is that it is using the credentials to connect to each system and then escalating that user to root privilege before completing each subsequent task.

The "Install httpd" task checks to see if the httpd package is yet installed on each host. If it is not, Ansible asks to install the package, along with any dependent packages. Next, Ansible checks the status of the `httpd` service on each host and, if it is not running, then starts it.

After that, each host is checked to see if the `firewalld` package is installed and installs it if it is not there. Then Ansible adds a firewall rule to each host to allow access to the `http` service (TCP port 80) and makes that setting permanent.

The `PLAY RECAP` then shows you the results of all of the tasks. Here you can see that all six tasks on all hosts were `ok`. If there had been any failed, skipped, rescued, or ignored tasks, they would be listed.

You can rerun this playbook if you think that something may have gotten out of place or if you made a modification to it. You could also use it later to deploy the playbook on different systems.

Although you have seen how Ansible is good at deploying multiple tasks in playbooks, it can also be used for one-off actions. In the next section, I show how to run some ad-hoc Ansible commands to query and further modify the hosts that we just deployed.

Running Ad-Hoc Ansible Commands

There may be times when you want to do one-off tasks on your Ansible-managed nodes. You can do those tasks using *ad-hoc commands*. With an ad-hoc command, you can directly call a module from the Ansible command line and have it act on an inventory. Some of those tasks could include the following:

- Installing RPM software packages
- Managing user accounts
- Copying files to and from nodes
- Changing permissions on a file or directory
- Rebooting a node

Just as when you run playbooks, running ad-hoc commands focuses on reaching a desired state. The ad-hoc command takes a declarative statement, figures out what is being requested, and does what it needs to do to reach the requested state.

To try these examples of ad-hoc Ansible commands, you can use the `ws` inventory created earlier.

Trying ad-hoc commands

When you run an ad-hoc Ansible command, you take some action using an Ansible module. The *command* module is used by default if no other module is indicated. Using the module, you indicate which command and options you want to run on a group of nodes as a one-time activity.

Check that an inventory is up and running. Here, you see that hosts are all running in the `ws` inventory:

```
$ ansible ws -u joe -m ping
host03 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
host02 | SUCCESS => { ...
host01 | SUCCESS => { ...
```

You can find out if the `httpd` service is running on the hosts in the `ws` inventory by checking the state of that service with this `ansible` command as follows:

```
$ ansible ws -u joe -m service \
     -a "name=httpd state=started" --check
host02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "name": "httpd",
    "state": "started",
    "status": { ...
host 01 | SUCCESS => { ...
```

At the moment, there is no content on the web servers. To add an `index.html` file (containing the text "Hello from your web server!") to all of the hosts in the `ws` inventory, you could run this command (type the root password when prompted):

```
$ echo "Hello from your web server!" > index.html
$ ansible ws -m copy -a \
    "src=./index.html dest=/var/www/html/ \
```

```
    owner=apache group=apache mode=0644" \
    -b --user joe --become --ask-become-pass
BECOME password: *********
host01 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "checksum": "213ae4bb07e9b1e96fbc7fe94de372945a202bee",
    "dest": "/var/www/html/index.html",
    "gid": 48,
    "group": "apache",
    "md5sum": "495feb8ad508648cfafcf69681d94f97",
    "mode": "0644",
    "owner": "apache",
    "secontext": "system_u:object_r:httpd_sys_content_t:s0",
    "size": 52,
    "src": "/home/joe/.ansible/tmp/ansible-tmp-1581027374.649223-
29961128730253/source",
    "state": "file",
    "uid": 48
host02 | CHANGED => { ...
host03 | CHANGED => { ...
```

You can see that the `index.html` file is created with the `apache` owner (UID 48) and `apache` group (GID 48) in the `/var/www/html` directory on `host01`. The copy was then repeated to `host02` and `host03`. You can check that everything is working by trying to access that file from the `ansible` host through the web server using the `curl` command:

```
$ curl host01
Hello from your web server!
```

Automating Tasks with Ansible Tower Automation Framework

Although running Ansible playbooks and commands are great for automating and later modifying sets of hosts, for a fully managed enterprise, you can go even further with Ansible. Using Ansible Tower, you can add a larger framework to your Ansible deployments.

Ansible Tower provides a web-based interface for managing your entire IT infrastructure with Ansible playbooks and other components. By centralizing your Ansible assets in one place, you have a single place to receive notifications. You can manage different administrative roles across your enterprise.

The Ansible Tower interface makes it easy to update your provisioned assets continuously. Instead of having to remember command-line options, you can just click to configure and

launch your Ansible tasks. Inventory management is graphical and job scheduling can be done in intuitive, visual ways.

A REST API is available with Ansible Tower that can help you embed your existing infrastructure tools into Ansible. So, you can usually just continue the processes that you already have in place but manage them with Ansible instead.

You can learn more about Ansible Tower from the Ansible Tower site (`https://www.ansible.com/products/tower`).

Summary

Ansible provides a unique formatting language and set of tools to automate many of the tasks that you have learned in other parts of this book. Once you know how to build an Ansible playbook, you can identify the exact configuration that you want on a system and then easily deploy that configuration to one or more host systems.

With Ansible playbooks, you define the exact state of an application and surrounding components and then apply that state to Linux host systems, network devices, or other targets. You can save those playbooks and reuse them to produce similar results on other systems or adapt them to create new and different results.

Ansible can also use ad-hoc commands to update systems. From the `ansible` command line, you can add users, copy files, install software, or do almost anything else you can do with playbooks. With those commands, you can quickly apply a set of changes across multiple hosts or respond to a problem that requires a quick fix that needs to be made immediately to a set of hosts.

In this chapter, you learned about the different components that make up an Ansible toolset. You created your own playbook for deploying a simple web server. Then you ran some ad-hoc commands to modify the systems to which you deployed your playbook.

Exercises

These exercises test your ability to get Ansible installed on your system, create your first Ansible playbook, and run a few ad hoc Ansible commands. These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well).

Although Ansible is meant to deploy tasks to remote systems, the exercises here will just let you try out a playbook and a few commands on a single system. If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, you can often complete a task in multiple ways).

1. Install Ansible on your Fedora or RHEL system.
2. Add `sudo` privilege for the user that you want to use to do these exercises.

3. Create a start to an Ansible playbook (call it `my_playbook.yaml`) that includes the following content.

```
---
- name: Create web server
  hosts: localhost
  tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present
```

4. Run `ansible-playbook` on the `my_playbook.yaml` in check mode to see if there is a problem completing the playbook (*hint*: there is).
5. Modify `my_playbook.yaml` to escalate privileges so that the tasks are run as the root user.
6. Run `ansible-playbook` again until the `httpd` package successfully installs on your system.
7. Modify `my_playbook.yaml` again to start the `httpd` service, and set it so that it will start every time the system boots.
8. Run an `ansible` command that checks whether or not the `httpd` service is up on `localhost`.
9. Create an `index.html` file that contains the text "Web server is up," and use the `ansible` command to copy that file to the `/var/www/html` directory on `localhost`.
10. Use the `curl` command to view the contents of the file that you just copied to the web server.

CHAPTER 30

Deploying Applications as Containers with Kubernetes

IN THIS CHAPTER

Understanding Kubernetes

Trying Kubernetes

Running the Kubernetes Basics Tutorials

Enterprise-quality Kubernetes with OpenShift

Linux containers separate the applications they contain from the operating systems on which they run. Built properly, a container will hold a discrete set of software that can be transported and run efficiently. But the story doesn't end there. Once you have some containers, the next step is to engage them with a platform like Kubernetes that allows you to do the following:

- Group sets of containers together to form a larger application. For example, deploy a web server, a database, and monitoring tools together.
- Scale up your containers as the demand requires. In fact, you want to be able to scale each component of the larger application individually, without having to scale up everything.
- Set the state of your application and not just run it. What this means is that, instead of just saying to run a container, you want to be able to say something like "run three copies of container X, and if one goes down, be sure to start another one to replace it."
- Recover from host computers going down or becoming overloaded. If the host running a container fails, you want the container to recover quickly and start up on another host computer.
- Not be concerned about the infrastructure. You want your application to connect to the services that it needs without having to know the hostnames, IP addresses, or port numbers associated with those services.
- Upgrade your containerized applications without downtime.

Kubernetes offers all of those features and more. While at first there were others competing to be the platform of choice for orchestrating containers, such as Mesos and Docker Swarm, Kubernetes has become the undisputed leader in orchestrating, deploying, and managing containerized applications.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

This chapter introduces you to Kubernetes and the enterprise-quality Kubernetes platform called OpenShift. The best way to learn Kubernetes is to start up a Kubernetes cluster and run commands in order to explore Kubernetes and deploy a containerized application. Before you do that, you should understand a bit about what a Kubernetes cluster is and what components you need to deploy an application to a cluster.

Understanding Kubernetes

A *Kubernetes cluster* is made up of master and worker nodes. You can run all master and worker services on the same system for personal use. For example, with Minikube, as described later in this chapter, you can run a Kubernetes cluster from a virtual machine on your laptop (`https://kubernetes.io/docs/tasks/tools/install-minikube`).

In a production environment, you would spread Kubernetes across multiple physical or virtual systems. Here are the different components you need to consider if you were to set up a production-quality Kubernetes infrastructure:

Masters: A *master node* manages the components running in the Kubernetes cluster. It manages communications between components, schedules applications to run on the workers, scales up the applications as needed, and makes sure that the proper number of containers (distributed in *pods*) are running. You should have at least one master node, but you would typically have three or more available to make sure there is always at least one available master.

Workers: A *worker node* is where the deployed containers actually run. The number of workers that you need depends on your workload. For a production environment, you would certainly want more than one worker in case one failed or needed maintenance.

Storage: Networked storage allows containers to access the same storage, regardless of the node that runs them.

Other services: To integrate a Kubernetes environment into an existing data center, you might want to tap into existing services. For example, you would probably use your company's DNS server for the hostname to address resolution, LDAP or Active Directory service for user authentication, and a Network Time Protocol (NTP) server to synchronize time.

In Kubernetes, the smallest unit with which you can deploy a container is referred to as a *pod*. A pod can hold one or more containers, along with metadata describing its containers. Although a pod can hold only one container, it is sometimes appropriate for a pod to have more than one. For example, a pod might contain a *sidecar container*, which is meant to monitor the service running in the primary container in the pod.

Kubernetes masters

A Kubernetes *master node* directs the activities of a Kubernetes cluster. Master nodes oversee all of the activities of the cluster through a set of services. The centerpiece of a

Kubernetes *master* is the API server (`kube-apiserver`), which receives object requests. Communications between all of the nodes in the cluster pass through the API server.

When a Kubernetes master is presented with an *object*, such as a request that a certain number of pods be running, the Kubernetes scheduler (`kube-scheduler`) finds available nodes to run each pod and schedules them to run on those nodes. To make sure that each object remains in the prescribed state, Kubernetes controllers (`kube-controller-manager`) run continuously to do things such as to make sure that namespaces exist, that defined service accounts are available, that the right number of replicas are running, and that defined endpoints are active.

Kubernetes workers

At the heart of each Kubernetes *worker node* is the kubelet service. A kubelet registers its worker node with the API server. The API server then directs the kubelet to do things like run a container that is requested from the API server through a PodSpec and make sure that it continues to run in a healthy state.

Another service that runs on each node is a *container engine* (often referred to as a *runtime*). Originally, the docker service was by far the most popular container engine used to launch, manage, and delete containers as required by the PodSpec. However, other container engines are now available, such as the CRI-O container engine (`https://cri-o.io/`), which is used with some commercial Kubernetes platforms such as OpenShift.

Worker nodes are meant to be as generic as possible so that you can simply spin up a new node when additional capacity is needed and it will be configured to handle most requests to run containers. There are, however, ways in which a container might not be appropriate to run on a particular node. For example, a pod might request to run on a node that has a minimum amount of memory and CPU available, or it might request to run on a node that is running a related container. Likewise, if a pod requires something special to run, such as a particular computer architecture, hardware, or operating system, there are ways to schedule pods on workers that meet those needs.

Kubernetes applications

In Kubernetes, applications are managed by defining API objects that set the state of resources on the cluster. For example, you can create a `Deployment` object in a YAML file that defines *pods* that each run one or more containers, along with the `namespace` in which it runs and the number of `replicas` of each pod it runs. That object could also define the `ports` that are open and any `volumes` that are mounted for each container. Kubernetes *master* nodes respond to those kinds of requests and make sure that the requests are carried out on the Kubernetes *worker* nodes.

Kubernetes uses the concept of *services* to separate the location of an application from its actual IP address and port number. By assigning a service name to the set of pods that provide that service, the exact location of each pod does not need to be known outside of the cluster. Instead, it is up to Kubernetes to direct a request for that service to an available pod.

IP addresses associated with active pods are not directly addressable from outside the cluster by default. It is up to you to define how you want to expose a service associated with a set of pods outside of the cluster. Using a `Service` object, you can expose services in different ways.

By default, exposing a service via a `ClusterIP` service `type` makes it available only to other components within the cluster. To expose the service outside of the cluster, you can use `NodePort`, which makes the pod providing the service accessible through the same Kubernetes-assigned port on an external IP address from each node on which the pod is running.

A third method is to use `LoadBalancer` to assign an external, fixed IP address that acts as a load balancer to the pods providing the service. With `LoadBalancer`, a cloud's external load balancer directs traffic to the backend pods. Finally, you can expose the service with `ExternalName`, which associates the service with a particular DNS CNAME record.

Regardless of how you expose a Kubernetes service, when there is a request for that service, Kubernetes acts to route communications to the set of pods that provide that service. In that way, pods can come up and down without disrupting the clients using the service.

Kubernetes interfaces

Kubernetes has both command-line and web console interfaces for accessing a Kubernetes cluster. The examples in this chapter focus on command-line tools. Commands include `minikube`, which is used to manage the Kubernetes virtual machine and bring the cluster up and down, and `kubectl`, which is the general-purpose tool for managing the Kubernetes cluster.

Trying Kubernetes

Because setting up your own production-quality Kubernetes cluster requires some forethought, this chapter will focus on a couple of easy ways to get a personal Kubernetes cluster running and accessible quickly. In particular, here are two different ways that you can gain access to a Kubernetes cluster:

Kubernetes Tutorials: The official Kubernetes site offers interactive, web UI tutorials, where you can start up your own cluster and try out Kubernetes. From Kubernetes Tutorials (`https://kubernetes.io/docs/tutorials/`), you can choose from basic, configuration, stateless applications, and other tutorial topics.

Minikube: With Minikube, you can run Kubernetes on your own computer. A Linux, MacOS, or Windows system that can run virtual machines can get the Minikube VM and have a Kubernetes cluster running on a laptop or desktop system within a few minutes.

Docker Desktop: Another option (not detailed here) is Docker Desktop, which lets you enable a pre-configured Kubernetes cluster that runs a master and worker node on your workstation.

To get you started, I'll step you through some of the Kubernetes tutorials and explain the concepts behind what they are doing. You can follow along in the tutorial or run the same commands on your own Minikube setup. I describe how to get Kubernetes in one of these two ways next.

> **NOTE**
> If you have an OpenShift environment up and running, you can follow most of these steps in OpenShift as well. In most case, you can use the `kubectl` command, but typically the same options and arguments can be used by the `oc` command for OpenShift.

Getting Kubernetes

The following descriptions tell you how to access a Kubernetes cluster either through the Kubernetes Basics Tutorial or by installing and starting Minikube.

Starting the Kubernetes Basics Tutorial

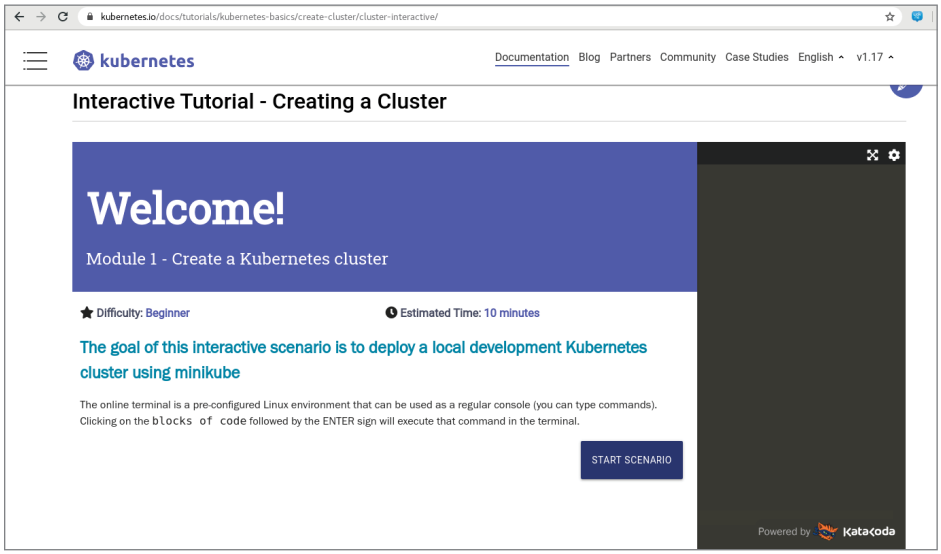
To start up the Kubernetes project basic interactive tutorial, visit the following URL from your web browser:

```
https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-interactive
```

Figure 30.1 shows the start of the Kubernetes Basics Tutorial.

FIGURE 30.1

Step through the Kubernetes project tutorials



At this point, you can follow the prompts through the tutorial. Because the tutorial starts a live cluster, you can use that interface to try other commands as well.

Starting Minikube

Getting Minikube running on your personal computer requires a few things. This includes the following:

- The computer needs to be configured as a hypervisor, so it can run the Minikube virtual machines.
- You need to install the `kubectl` command (used to access and work with the cluster) and the Minikube VM itself.

For Linux, MacOS, and Windows systems, go here to find the latest instructions:

`https://kubernetes.io/docs/tasks/tools/install-minikube/`

You can install Minikube as root user, but you need to run it later from a regular user account. The steps for installing Minikube on a Fedora, RHEL, Ubuntu, or other Linux system are as follows (refer to the `install-minikube` page if something has changed):

1. **Install the `kubectl` command**: Get a version of the `kubectl` command that is within one version of Kubernetes in your Minikube. Installing the latest versions of `kubectl` and `minikube` should take care of that. Enter the following (all on one line):

```
# curl -LO \
 https://storage.googleapis.com/kubernetes-release/release/`curl \
-s https://storage.googleapis.com/kubernetes-release/release/
   stable.txt \
`/bin/linux/amd64/kubectl
```

2. **Copy `kubectl` to a bin directory**: Copy the `kubectl` command to an accessible bin directory and make it executable. For example:

```
# mkdir /usr/local/bin
# cp kubectl /usr/local/bin
# chmod 755 /usr/local/bin/kubectl
```

3. **Configure hypervisor**: Configure your Linux system as a hypervisor. For KVM, use the descriptions found in the section "Configuring hypervisors" in Chapter 27.

4. **Get `minikube`**: Get the `minikube` executable, and enter the following (on one line):

```
# curl -Lo minikube \
https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64 \
&& chmod +x minikube
```

5. **Install Minikube**: Enter the following:

```
# install minikube /usr/local/bin/
```

6. **Run Minikube**: As a regular user, enter the following commands to identify the driver if your hypervisor is KVM (see `https://minikube.sigs.k8s.io/docs/reference/drivers` if you are using a different hypervisor):

```
$ minikube config set vm-driver kvm2
$ minikube start --vm-driver=kvm2
```

7. **Start using Minikube**: You can start using Minikube by running some `minikube` and `kubectl` commands. Examples of how to do that are shown in the next tutorial.

Running the Kubernetes Basics tutorial

The Kubernetes Basics Tutorial take you through a good set of commands to start familiarizing yourself with Kubernetes:

`https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-interactive`

The following text walks you through the first five modules of the Kubernetes Basics Tutorial.

If you are running through this procedure directly from the Kubernetes tutorials page, go ahead and start Minikube (`minikube start`). If you are using Minikube from a VM already running on your laptop, you can still follow this procedure. The steps are the same since both use Minikube.

Get information about your cluster

Run these commands to get basic information about your cluster.

1. **List Minikube version**: To see the version of `minikube` you are using, enter the following:

```
$ minikube version
minikube version: v1.7.2
commit: 50d543b5fcb0e1c0d7c27b1398a9a9790df09dfb
```

2. **List cluster information**: To see the URL from which the Kubernetes master and DNS services are available, enter the following:

```
$ kubectl cluster-info
Kubernetes master is running at https://192.168.39.150:8443
KubeDNS is running at
https://192.168.39.150:8443/api/v1/namespaces/kube-system/
services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use
'kubectl cluster-info dump'.
```

3. **List node information**: To see the number of nodes running (just one master node for Minikube) and their status, enter the following:

```
$ kubectl get nodes
NAME     STATUS   ROLES  AGE   VERSION
minikube Ready    master 23m   v1.17.2
```

4. **List cluster and client versions**: To list the versions of the `kubectl` client and Kubernetes cluster (to make sure that they are within one version of each other), enter the following:

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17",
GitVersion:"v1.17.2",
GitCommit:"59603c6e503c87169aea6106f57b9f242f64df89",
GitTreeState:"clean", BuildDate:"2020-01-18T23:30:10Z",
GoVersion:"go1.13.5", Compiler:"gc", Platform:"linux/amd64"}

Server Version: version.Info{Major:"1", Minor:"17",
GitVersion:"v1.17.2",
GitCommit:"59603c6e503c87169aea6106f57b9f242f64df89",
GitTreeState:"clean", BuildDate:"2020-01-18T23:22:30Z",
GoVersion:"go1.13.5", Compiler:"gc", Platform:"linux/amd64"}
```

Deploy a Kubernetes application

Requests to run and manage containerized applications (in the form of pods) on a Kubernetes cluster is known as a *deployment*. Once a deployment is created, it is up to the Kubernetes cluster to make sure that the requested pods are always running. It does this by doing the following:

- Accepting the deployment creation through the API server
- Asking the scheduler to run the requested containers from each pod on available worker nodes
- Watching the pods to make sure they continue to run as requested
- Starting a new instance of a pod (on the same or different node) if the pod fails (for example, if the container stops running)

The tutorial shows an example of how to create a simple deployment from a container image. In this example, you just give it a name and identify the container image to use. The rest of the deployment settings are filled in from defaults.

1. **Create a deployment**: To start the deployment that pulls the `kubernetes-bootcamp` container with a deployment name of `kubernetes bootcamp`, enter the following:

```
$ kubectl create deployment kubernetes-bootcamp \
      --image=gcr.io/google-samples/kubernetes-bootcamp:v1
deployment.apps/kubernetes-bootcamp created
```

2. **List deployments**: To see that the deployment exists (and has one instance requested and one running), enter the following.

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           4m38s
```

3. **Describe the deployment**: To see details about the deployment, enter the following:

```
$ kubectl describe deployments kubernetes-bootcamp
Name:                   kubernetes-bootcamp
Namespace:              default
...
Replicas:   1 desired | 1 updated | 1 total | 1 available | 0
unavailable
...
Pod Template:
  Labels:  app=kubernetes-bootcamp
  Containers:
   kubernetes-bootcamp:
    Image:        gcr.io/google-samples/kubernetes-bootcamp:v1
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
...
```

In the `kubernetes-bootcamp` deployment, notice that it just set one instance (`replica`) of the pod associated with the deployment to be available. The deployment runs in the current namespace, which happens to be `default`. Notice also that there are no ports open or volumes mounted by default for the pods.

Get information on the deployment's pods

With the deployment created, you can ask for information about the pod created from that deployment and expose the Kubernetes API from the VM to your local system, via a proxy service, to connect to the pod directly.

1. **Expose the Kubernetes API to the local system**: To open a proxy from your system to the Kubernetes API running in Minikube (`kubectl proxy`), enter the following:

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

2. Query the Kubernetes API: Open a second terminal, and query the Kubernetes API running on Minikube by entering the following:

```
$ curl http://localhost:8001/version
{
 "major": "1",
 "minor": "17",
 "gitVersion": "v1.17.2",
 "gitCommit": "59603c6e503c87169aea6106f57b9f242f64df89",
 "gitTreeState": "clean",
 "buildDate": "2020-01-18T23:22:30Z",
```

```
 "goVersion": "go1.13.5",
 "compiler": "gc",
 "platform": "linux/amd64"
```

3. **Get pod information**: The name of the pod used in this deployment is `kubernetes-bootcamp`, followed by a unique string of characters. Enter these commands to output the name of the pod and then list a description of that pod:

```
$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-69fbc6f4cf-njc4b  1/1     Running   0          12m
$ kubectl describe pod kubernetes-bootcamp-69fbc6f4cf-njc4b
Name:         kubernetes-bootcamp-69fbc6f4cf-njc4b
Namespace:    default
Priority:     0
Node:         minikube/192.168.39.150
...
Containers:
  kubernetes-bootcamp:
    Container ID:
docker://dd24fd43ff19d6cf12f5c759036cee74adcf2d0e2c55a42e...
    Image:          gcr.io/google-samples/kubernetes-bootcamp:v1
    Image ID:       docker-pullable://gcr.io/google-samples...
...
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  14m   default-scheduler  Successfully assigned
default/kubernetes-bootcamp-69fbc6f4cf-njc4b to minikube
  Normal  Pulled     14m   kubelet, minikube  Container image
"gcr.io/google-samples/kubernetes-bootcamp:v1"
already present on machine
  Normal  Created    14m   kubelet, minikube  Created container
kubernetes-bootcamp
  Normal  Started    14m   kubelet, minikube  Started container
kubernetes-bootcamp
```

From the trimmed output, you can see the name of the pod, the namespace it is in (`default`), and the node on which it is running (`minikube/192.168.39.150`). Under Containers, you can see the name of the running container (`docker://dd24fd43ff19 . . .`), the image it came from (`. . . kubernetes-bootcamp:v1`), and the image ID for that image. Under Events, starting from the bottom, you can see the `kubelet` on the node `minikube`, starting and creating the container. It goes to pull the image and finds that it is already on the node. Then it assigns the pod to run on that node.

4. **Connect to the pod**: Use the `curl` command to contact the `pod` and get it to respond to your request for information:

```
$ export POD_NAME=$(kubectl get pods -o go-template --template \
 '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}') ; \
echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-69fbc6f4cf-njc4b

$ curl \
 http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/
proxy/
Hello Kubernetes bootcamp!|Running on:kubernetes-bootcamp-
5b48cfdcbd-lf9t2|v=1
```

5. **View the logs**: To see the logs for any container that is running inside the selected pod, run the following command:

```
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2020-02-13T21:29:21.836Z
| Running On:  kubernetes-bootcamp-5b48cfdcbd-lf9t2

Running On: kubernetes-bootcamp-5b48cfdcbd-lf9t2 | Total Requests:
1 | App Uptime: 34.086 seconds | Log Time: 2020-02-13T21:29:55.923Z
```

6. **Execute commands on the pod**: Use `kubectl exec` to run commands inside the pod. The first command runs `env` in order to view shell environment variables from inside of the pod, and the second opens a shell inside the pod so you can run the following commands:

```
$ kubectl exec $POD_NAME env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubernetes-bootcamp-5b48cfdcbd-lf9t2
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
...
$ kubectl exec -ti $POD_NAME bash
root@kubernetes-bootcamp-5b48cfdcbd-lf9t2:/# date
Thu Feb 13 21:57:18 UTC 2020

kubernetes-bootcamp-5b48cfdcbd-lf9t2:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 21:29 ?        00:00:00 /bin/sh -c node
server.js
root         6     1  0 21:29 ?        00:00:00 node server.js
root       115     0  0 21:55 pts/0    00:00:00 bash
root       123   115  0 22:01 pts/0    00:00:00 ps -ef
```

```
root@kubernetes-bootcamp-5b48cfdcbd-lf9t2:/# curl localhost:8080
Hello Kubernetes bootcamp!|Running on:kubernetes-bootcamp-
5b48cfdcbd-lf9t2|v=1

root@kubernetes-bootcamp-5b48cfdcbd-lf9t2:/# exit
```

After starting a shell, you can see output from the `date` and `ps` commands. From `ps`, you can see that the first process run in the container (PID 1) is the `server.js` script. After that, the `curl` command is able to communicate successfully with the container on `localhost` port `8080`.

Expose applications with services

To expose the `kubernetes-bootcamp` pod described in these procedures so that it is accessible from an external IP address from the worker node on which it is running, you can create a `NodePort` object. Here is one way to do that:

1. **Check that the pod is running**: Enter the following to see that the `kubernetes-bootcamp` pod is running.

```
$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-765bf4c7b4-fdl96  1/1     Running   0          26m
```

2. **Check the services**: Enter the following to see the services running in the `default` namespace. Notice that only the `kubernetes` service is available and that there is no service exposing the `kubernetes-bootcamp` pod outside of the cluster:

```
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   31m
```

3. **Create a service**: Create a service that uses NodePort to make the pod available from an IP address on the host at a specific port number (8080). For example, enter the following:

```
$ kubectl expose deployment/kubernetes-bootcamp \
    --type="NodePort" --port 8080
service/kubernetes-bootcamp exposed
```

4. **View the new service**: Type the following to see the IP address (10.96.66.230) and port number (8080) from which the service is made available on the host:

```
$ kubectl get services
NAME              TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes        ClusterIP  10.96.0.1    <none>        443/TCP        33m
```

```
kubernetes-bootcamp NodePort  10.96.66.230 <none>       8080:32374/
TCP 5s

$ kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   app=kubernetes-bootcamp
Annotations:              <none>
Selector:                 app=kubernetes-bootcamp
Type:                     NodePort
IP:                        10.96.66.230
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30000/TCP
Endpoints:                172.17.0.6:8080
Session Affinity:         None
External Traffic Policy:  Cluster
```

5. **Get the assigned node port**: To get the port assigned to the service and set the `$NODE_PORT` variable to that value, enter the following:

```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp \
-o go-template='{{(index .spec.ports 0).nodePort}}')

$ echo NODE_PORT=$NODE_PORT
NODE_PORT=30000
```

6. **Access the service**: To check that the service is available from the NodePort, use the following `curl` command (using the IP address for your Minikube instance):

```
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp!|Running on:kubernetes-bootcamp-
765bf4c7b4-fdl96|v=1
```

Label a service

Use this procedure to add a label to an existing service.

1. **Check the pod's label**: So far, `kubernetes-bootcamp` is the only label assigned to the pod. To make sure, enter the following:

```
$ kubectl describe deployment
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Fri, 14 Feb 2020 05:43:49 +0000
Labels:                 run=kubernetes-bootcamp
Annotations:            deployment.kubernetes.io/revision: 1
...
```

2. **Add another label**: To add an additional label (`v1`) to the pod, get the name of the pod and add the new label as follows:

```
$ export POD_NAME=$(kubectl get pods -o go-template --template \
 '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}') ; \
echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-765bf4c7b4-fdl96


$ kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-765bf4c7b4-fdl96 labeled
```

3. **Check and use the label**: Check that the `v1` label has been assigned to the pod, and then use that label to list information about the pod:

```
$ kubectl describe pods $POD_NAME
Name:         kubernetes-bootcamp-765bf4c7b4-fdl96
Namespace:    default
Priority:     0
Node:         minikube/172.17.0.62
Start Time:   Fri, 14 Feb 2020 05:44:08 +0000
Labels:       app=v1
              pod-template-hash=765bf4c7b4
              run=kubernetes-bootcamp
$ kubectl get pods -l app=v1
NAME                                   READY   STATUS    RESTARTS
AGE
kubernetes-bootcamp-765bf4c7b4-fdl96   1/1     Running   0
60m
```

Delete a service

If you are done using the service, you can delete it. This removes access to the service from the NodePort, but it does not delete the deployment itself.

1. **Check the service**: Make sure that the `kubernetes-bootcamp` service still exists:

```
$ kubectl get services
NAME                TYPE       CLUSTER-IP    EXTERNAL-IP PORT(S)
AGE
kubernetes          ClusterIP 10.96.0.1     <none>      443/TCP
63m
kubernetes-bootcamp NodePort  10.96.66.230 <none>   8  080:32374/
TCP 30m
```

2. **Delete the service**: Using the label name, delete the service:

```
$ kubectl delete service -l run=kubernetes-bootcamp
service "kubernetes-bootcamp" deleted
```

3. **Check the service and deployment**: Make sure the service has been deleted but the deployment still exists:

```
$ kubectl get services
NAME                 TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
AGE
kubernetes           ClusterIP 10.96.0.1     <none>        443/TCP
64m
$ kubectl get deployment
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           65m
```

Scale up an application

One of the most powerful features of Kubernetes is its ability to scale up an application as the demand requires it. This procedure starts with the `kubernetes-bootcamp` deployment, which is running one pod, and scales it up to have additional pods running using the *ReplicaSet* feature and a different means of exposing the application to outside access.

1. **Get the deployment**: List information about the `kubernetes-bootcamp` deployment, and note that it is set to have only one replica set (`rs`) active:

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           107s
$ kubectl get rs
NAME                             DESIRED   CURRENT   READY   AGE
kubernetes-bootcamp-5b48cfdcbd   1         1         1       3m4s
```

2. **Scale up the replicas**: To scale the deployment up to four replica sets, enter the following:

```
$ kubectl scale deployments/kubernetes-bootcamp --replicas=4
deployment.extensions/kubernetes-bootcamp scaled
```

3. **Check the new replicas**: List the deployments to make sure that there are now four replicas ready and available:

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   4/4     4            4           8m44s
```

4. **Check the pods**: There should now also be four `kubernetes-bootcamp` pods running, each with its own IP address inside the cluster. To make sure, enter the following:

```
$ kubectl get pods -o wide
NAME                          READY   STATUS    RESTARTS AGE   IP
   NODE      NOMINATED NODE   READINESS GATES
```

```
kubernetes-bootcamp-5b4... 1/1     Running   0        8m43s
172.18.0.4
   minikube  <none>           <none>
kubernetes-bootcamp-5b4... 1/1     Running   0        12s
172.18.0.8
   minikube  <none>           <none>
kubernetes-bootcamp-5b4... 1/1     Running   0        12s
172.18.0.6
   minikube  <none>           <none>
kubernetes-bootcamp-5b4..  1/1     Running   0        12s
172.18.0.7
   minikube  <none>           <none>
```

5. **View deployment details**: To see details of the increased replicas in the deployment, enter the following:

```
$ kubectl describe deployments/kubernetes-bootcamp
Name:                   kubernetes-bootcamp
Namespace:              default
...
Replicas:    4 desired | 4 updated | 4 total | 4 available | 0
unavailable
...
NewReplicaSet:   kubernetes-bootcamp-5b48cfdcbd (4/4 replicas
created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  17m    deployment-controller  Scaled
up
     replica set kubernetes-bootcamp-5b48cfdcbd to 1
  Normal  ScalingReplicaSet  9m25s  deployment-controller  Scaled
up
replica set kubernetes-bootcamp-5b48cfdcbd to 4
```

Check the load balancer

To check that traffic is being distributed across all four replicated pods, you can get the NodePort and then use the `curl` command to make sure that multiple connections to the NodePort result in different pods being accessed:

1. **List details about the service**: To see details about the `kubernetes-bootcamp` service, enter the following:

```
$ kubectl describe services/kubernetes-bootcamp
Name:            kubernetes-bootcamp
Namespace:       default
Labels:          run=kubernetes-bootcamp
Annotations:     <none>
Selector:        run=kubernetes-bootcamp
Type:            NodePort
IP:              10.99.183.8
```

```
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
NodePort:          <unset>  31915/TCP
Endpoints:         172.18.0.4:8080,172.18.0.6:8080,172.18.0.7:8080 +
1 more...
```

Notice that each pod has its own IP address and port (`172.18.0.4:8080`, `172.18.0.4:8080`, and so on).

2. **Get the NodePort**: Enter the following to set `$NODE_PORT` to the value of the port number assigned to the service:

```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp \
  -o go-template='{{(index .spec.ports 0).nodePort}}')

$ echo NODE_PORT=$NODE_PORT
NODE_PORT=31915
```

3. **Run curl**: Run the `curl` command a few times to query the service. If you run it a few times, you should see that it is accessing different pods. That is how you know that the load balancer is working:

```
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp!|Running on:kubernetes-bootcamp-
5b48cfdcbd-9j4xp|v=1
```

Scale down an application

To scale the number of ReplicaSets defined in your deployment, simply change the number of replicas to a lower number.

1. **Scale down replicas**: Enter the following to change the number of replicas for the deployment to `2`:

```
$ kubectl scale deployments/kubernetes-bootcamp –replicas=2
deployment.extensions/kubernetes-bootcamp scaled
```

2. **Check the deployment**: To see that the deployment is set to `2`, and that only two pods are running, enter the following:

```
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   2/2     2            2           52m
$ kubectl get pods -o wide
NAME                         READY   STATUS    RESTARTS AGE   IP
   NODE      NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-5b4... 1/1     Running   0        8m43s
172.18.0.4
   minikube  <none>           <none>
kubernetes-bootcamp-5b4... 1/1     Running   0        12s
172.18.0.8
```

At this point, you should feel comfortable manually querying your Kubernetes cluster in various ways and starting up and working with deployments, pods, and replicas. To continue with

more advanced Kubernetes tutorials, return to the main Kubernetes Tutorials page (`https://kubernetes.io/docs/tutorials/`). I also recommend the Kubernetes By Example site for more information on using Kubernetes (`https://kubernetesbyexample.com`).

Enterprise-Quality Kubernetes with OpenShift

Red Hat OpenShift Container Platform (`www.openshift.com`) is a product that is designed to deliver an enterprise-quality Kubernetes platform that can be used for mission-critical applications. As a hybrid cloud platform, OpenShift is built to be deployed in both bare metal and cloud environments.

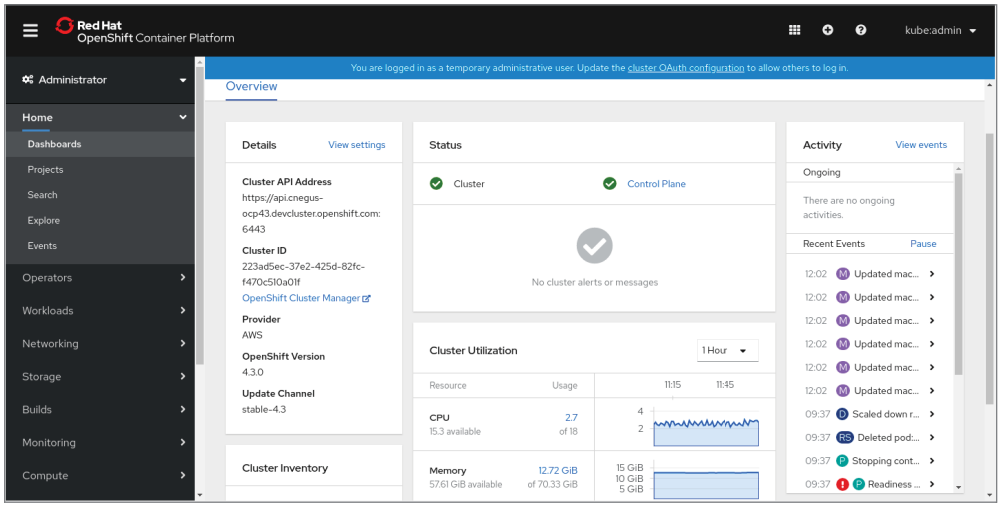
While Kubernetes is an open source project that can be built and run in a tremendous number of ways, Kubernetes-based products, such as OpenShift, are meant to be used when you need a solid, supported platform upon which your business can rely. OpenShift also comes in different variants, which can be installed in your own data center and in cloud environments, such as AWS and Azure, or simply used from a dedicated OpenShift cluster maintained for you by Red Hat.

When you lock down the Kubernetes features that Red Hat builds into OpenShift, those features can be thoroughly tested and supported. Training and documentation can be built around those features. Also, more complex features can be built in, such as advanced government compliance features and tight integrations with various cloud environments.

With an intuitive web console, Red Hat OpenShift is made to be easier to use for people starting with Kubernetes. An example of the OpenShift console is shown in Figure 30.2.

FIGURE 30.2

OpenShift features an intuitive web UI for deploying and managing Kubernetes objects.



There are free trials of OpenShift available from `https://try.openshift.com`. There is also an open source upstream project for OpenShift, called OKD, which you can get for free as well (`www.okd.io`).

Summary

In the past few years, Kubernetes has become the platform of choice for deploying containerized applications across large data centers. A Kubernetes cluster consists of master nodes (that direct the activities of a cluster) and worker nodes (that actually run the containerized payloads).

As someone using Kubernetes to run containerized applications, you can create deployments that define the state of the application you are running. For example, you can deploy an application that is configured to run multiple replicas of the pods representing that application. You can identify the application as a service and set up the application to be available from defined ports on the nodes from which they are run.

Products based on Kubernetes are available when you need to run mission-critical applications in environments that are stable and supported. One such product is the Red Hat OpenShift Container Platform. With OpenShift, you can run supported Kubernetes-based cluster configurations that run in a variety of environments, including bare metal and various cloud environments.

Exercises

The exercises in this section describe tasks related to trying out Kubernetes, either online or by setting up Minikube on a computer. If you are stuck, solutions to the tasks are shown in Appendix B. Keep in mind that the solutions shown in Appendix B are usually just one of many ways to complete a task.

1. Install Minikube on your local system or access a Minikube instance externally (such as through the `Kubernetes.io` Tutorials).
2. View your version of Minikube, as well as the versions of your `kubectl` client and Kubernetes service.
3. Create a deployment that manages a pod running the `hello-node` container image (`gcr.io/hello-minikube-zero-install/hello-node`).
4. Use the appropriate `kubectl` commands to view the `hello-node` deployment and describe the deployment in detail.
5. View the current replica set associated with your `hello-node` deployment.
6. Scale up the `hello-node` deployment to three (3) replicas.
7. Expose the `hello-node` deployment outside of the Kubernetes cluster using `LoadBalancer`.

8. Get the IP address of your Minikube instance and the port number of the exposed `hello-node` service.
9. Use the `curl` command to query the `hello-node` service using the IP address and port number from the previous step.
10. Use the `kubectl` commands to delete the `hello-node` service and deployment, and then use the `minikube` command to stop the Minikube virtual machine.

Part VII

Appendixes

IN THIS PART

Appendix A
Media

Appendix B
Exercise Answers

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

APPENDIX

A

Media

IN THIS APPENDIX

Getting Linux distributions

Creating a bootable CD or DVD

Unless you bought a computer with Linux preinstalled or had someone install it for you, you need to find a way to get a Linux distribution and then either install or run it live on your computer. Fortunately, Linux distributions are widely available and come in a variety of forms.

In this appendix, you learn how to do the following:

- Get a few different Linux distributions
- Create a bootable disk to install your distribution
- Boot Linux from a USB drive

To use this book effectively, you should have a Linux distribution in front of you to work on. It's important to be able to experience Linux as you read. So, try the examples and do the exercises.

Linux distributions are most commonly available from the websites of the organizations that produce them. The following sections describe websites associated with Linux distributions that offer ISO images you can download.

Note

An ISO is a disk image that is formatted in the ISO 9660 filesystem format, a format that is commonly used with CD and DVD images. Because this is a well-known format, it is readable by Windows, Mac, and Linux systems.

An ISO image can be used to create a bootable USB flash drive, CD, or DVD medium, depending on the size of the image. An ISO image in your filesystem can be mounted in Linux in loopback mode, so you can view or copy its contents.

When an ISO image contains a Linux Live CD or installation image, the images are bootable. This means that instead of starting up an operating system, such as Windows or Linux, from the computer's hard drive, you can tell your computer to boot from the CD or DVD instead. This enables you to run a totally different operating system than is installed on your hard drive without changing or damaging the data on that drive.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Getting Fedora

Note

I recommend downloading the Fedora Workstation Live Image to use along with this book because most of the book works with that distribution. You can run it live without committing to overwriting your computer's hard disk until you feel comfortable enough to install it permanently.

To test the examples in this book, I used Fedora 30 and 31, 64-bit Fedora Workstation images, which you can get from `GetFedora.org` (`https://getfedora.org/en/workstation/download`). If you have a 64-bit machine, you must use the 64-bit ISO.

Later versions of Fedora that come with a GNOME desktop should work as well. Here's a link to the exact ISO used for the Fedora 31 Workstation:

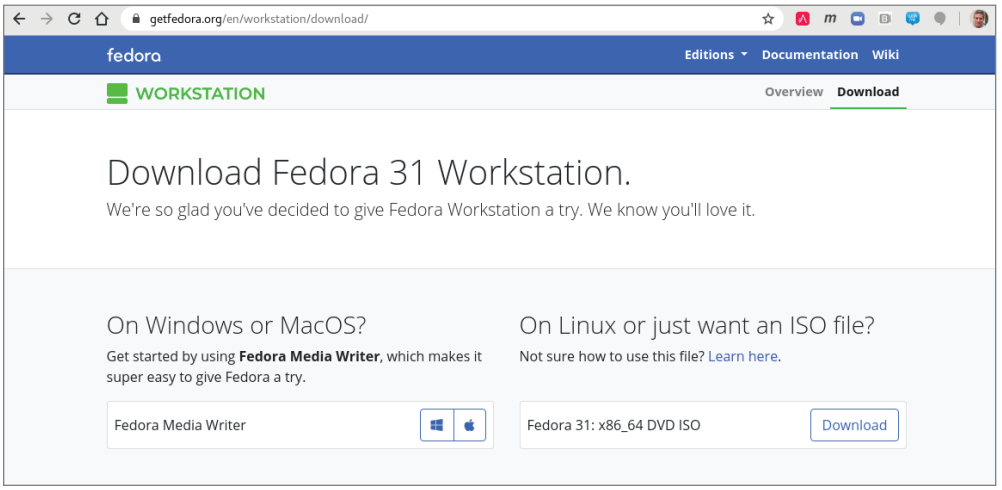
```
https://download.fedoraproject.org/pub/fedora/linux/releases/31/Workstation/
x86_64/iso/Fedora-Workstation-Live-x86_64-31-1.9.iso
```

Keep in mind that the latest Fedora Workstation ISO image does not fit on a CD, so you must burn it to a DVD or USB flash drive. See the descriptions of CD/DVD burning tools available for Windows, MacOS, and Linux later in this appendix.

Figure A.1 shows an example of the Get Fedora page.

FIGURE A.1

Download Fedora ISO images from the Get Fedora page.



Today, the default download is an ISO image of a 64-bit PC-type Fedora Workstation (GNOME) Live DVD. You can boot this image on your computer, and if you choose, you can permanently install it to your computer's hard drive. To download this image, do the following:

1. Select Workstation or Server from `GetFedora.org`. I recommend Workstation to follow along with this book.
2. Select the Download Now button and click the Download button. A pop-up should appear, asking what you want to do with the ISO.
3. Select to save the ISO. Depending on your settings, either you are asked where you want to download it or it simply begins downloading to a default folder (in Linux, it is probably a Downloads folder).
4. If you are prompted for where to put the ISO, select a folder that has enough space to hold it. Remember where this folder is located, because you need to find the ISO when you go to burn it later.

If you need more information about what to do with the downloaded image, there are links to help you on the Fedora page that appears. At the time of this writing, the Learn Here link takes you to descriptions of how to create live installation media. The exact instructions might change as the website is updated.

You have other choices for downloading ISOs from Fedora. From the bottom of the `GetFedora.org` page, you can download specially configured Fedora ISO images called spins (`https://spins.fedoraproject.org`). Here are some special types of Fedora spins that might interest you:

KDE desktop spin: People who prefer the KDE desktop to the GNOME desktop can download the Plasma KDE spin.

Lightweight desktop spin: If you are trying Linux on a computer with less memory or processing power, consider Xfce and LXQt spins (representing lightweight desktops of the same name).

Desktop effects spin: The MATE-Compiz spin offers more of the other extreme to the lightweight desktops, with desktop effects like wobbly windows and desktops that rotate on a cube.

Child-friendly desktop spin: The SOAS desktop is a spin of the Sugar Learning Platform, made to provide a simplified setup and a child-friendly graphical interface. SOAS can be transported on a USB drive and run on any available computer.

Getting Red Hat Enterprise Linux

Many large corporations, government agencies, and universities use Red Hat Enterprise Linux to run their mission-critical applications. While most of the procedures in this book will run well on Fedora, there are many references to how things are done differently in

Red Hat Enterprise Linux because, when you go to get a job as a Linux system administrator, you will, in most cases, be working with Red Hat Enterprise Linux systems.

Although the source code for Red Hat Enterprise Linux is freely available, the ISOs containing the packages you install (often referred to as the *binaries*) are available only to those who have accounts on the Red Hat customer portal (`https://access.redhat.com`) or through evaluation copies.

If you don't have an account, you can try signing up for a 30-day trial. If either you or your company has an account with Red Hat, you can download the ISOs that you need. Go to the following site and follow the instructions to download a Red Hat Enterprise Linux server ISO or sign up to get an evaluation copy:

`https://access.redhat.com/downloads`.

Red Hat does not offer live versions of Red Hat Enterprise Linux. Instead, you can download installation DVDs that you can install as described in Chapter 9, "Installing Linux," of this book.

Note

If you are unable to obtain a Red Hat Enterprise Linux installation DVD, you can get a similar experience using the CentOS installation DVD. CentOS is not exactly the same as RHEL. However, if you download the CentOS installation DVD for CentOS 8.x from links on the CentOS site (`http://www.centos.org/download/`), the installation procedure is similar to the one described for Red Hat Enterprise Linux in Chapter 9.

Getting Ubuntu

Many people new to Linux begin by downloading and installing Ubuntu. Ubuntu has a huge fan base and many active contributors. If you have problems with Ubuntu, there are large, active forums where many people are willing to help you overcome problems.

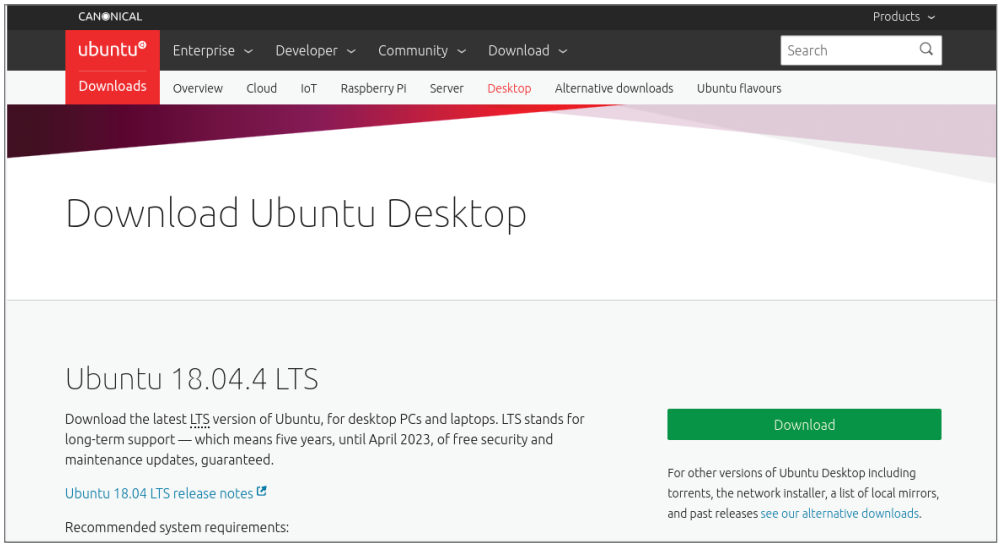
If you already have an Ubuntu system installed, you can follow along with most of this book. You can get Ubuntu with a GNOME desktop, and its default dash shell is similar to bash (or you can switch to bash in Ubuntu to match the shell examples in this book). Although most of the examples of this book focus on Fedora and RHEL, I have added many more references to Ubuntu throughout the book in this edition.

To get Ubuntu, you can download a Live ISO image or installation medium from the Download Ubuntu page: `http://www.ubuntu.com/download/ubuntu`.

Figure A.2 shows an example of the Download Ubuntu Desktop page.

FIGURE A.2

Download Ubuntu Live ISO images, or choose an alternative download.



As with Fedora, the easiest way to download Ubuntu is to select the 64-bit Ubuntu Live image, download it, and burn it. Here's how to do that from the Download Ubuntu page:

1. Click the Download button. By default, this downloads the most recent 64-bit Ubuntu desktop Live ISO image.
2. Either you are asked where you want to download the ISO image, or it simply begins downloading to a default folder.
3. If you are asked where to put the ISO, select a folder that has enough space to hold the ISO. Remember where this folder is located because you need to find the ISO when you go to burn it later.

After the download is complete, burn the ISO image to a DVD using procedures described in the section "Creating Linux CDs and DVDs".

Other types of Ubuntu installation media are also available. To find other Ubuntu media, go to the Alternative Downloads page (`http://www.ubuntu.com/download/alternative-downloads`). From this site, you can get media that contains a variety of desktop and server installs.

Booting Linux from a USB Drive

Instead of burning ISO images to a CD or DVD, you can put your Linux system on a USB drive. USB drives offer the advantage of being writable as well as readable, so you can

save your content between sessions. Most modern computers can boot from a USB drive, although you may have to interrupt the boot process to tell the BIOS to boot from a USB drive instead of hard drive or CD/DVD drive.

You can find procedures for putting Fedora and Ubuntu on a USB drive in the following locations:

Fedora on a USB drive: Using a tool called Live USB Creator, you can install a Fedora ISO image to a USB drive in either Windows or Linux. To run Fedora from that drive, insert it into a USB port on your computer, reboot the computer, interrupt the BIOS as it is booting (possibly F12), and select to boot from a USB drive. The procedure for using Live USB creator is located at

`https://docs.fedoraproject.org/en-US/quick-docs/creating-and-using-a-live-installation-image/index.html`

Ubuntu on a USB drive: Ubuntu has procedures for creating a bootable USB drive with Ubuntu on it that work from Windows, MacOS, or Linux. To find out how to do this, go to the Ubuntu Download page, and under "Easy ways to switch to Ubuntu," look for the appropriate "How to create a bootable USB stick..." procedure for Ubuntu, Windows, or MacOS:

`https://ubuntu.com/tutorials/tutorial-create-a-usb-stick-on-ubuntu#1-overview`

Creating Linux CDs and DVDs

After you have downloaded a Linux CD or DVD image, you can use several tools to create bootable CDs or DVDs for either installing or just running Linux live from those media. Before you begin, you must have the following:

DVD or CD ISO images: Download the ISO images to your computer that represent the physical DVD or CD you will ultimately burn. Today, most Linux ISO images are too big to fit on a CD (including those for RHEL, Fedora, and Ubuntu).

Blank DVDs/CDs: You need blank DVDs or CDs to burn the images to. CDs hold up to about 700MB; DVDs hold up to about 4.7GB (single layer).

CD/DVD burner: You need a drive that is capable of burning CDs or DVDs, depending on which you are burning. Not all CD/DVD drives can burn DVDs (especially older ones). So, you may need to find a computer with a drive that has that capability.

The following sections describe how to burn bootable CDs and DVDs from Windows, MacOS, and Linux systems.

Burning CDs/DVDs in Windows

If you have downloaded your Linux ISO image to a Windows system, you can burn that image to CD or DVD in different ways. Here are some examples:

Windows: In the latest Windows releases, the function of burning ISO images to CD or DVD is built into Windows. After an ISO image is downloaded, simply insert the

appropriate CD or DVD into your computer's drive (assuming the drive is writeable), right-click the ISO image icon from the folder to which you downloaded it, and select Burn Disc Image. When the Windows Disc Image Burner window appears, select Burn to burn the image.

Roxio Creator: This third-party Windows application contains many features for ripping and burning CDs and DVDs. You can read about the product here: `http://www.roxio.com/en/products/creator/`.

Nero CD/DVD Burning ROM: Nero is another popular CD/DVD burning software product for Windows systems. You can find out more about Nero here: `http://www.nero.com`.

Burning CDs/DVDs on a MacOS system

Like Windows, MacOS has CD/DVD burning software built into the operating system. To burn an ISO image to disk on a MacOS system, follow these steps:

1. Download the ISO image you want on your MacOS system. An icon representing the ISO should appear on your desktop.
2. Insert a blank CD or DVD into your CD/DVD burner, as appropriate for the size of the image.
3. Right-click the icon representing the Linux ISO that you just downloaded and select Burn "Linux" to Disk. A pop-up window appears, asking if you are sure you want to burn the image.
4. Fill in the name that you want to give the ISO and the write speed and then select Burn. The image begins burning to disk.
5. After the image has been burned, eject the disk; you are ready to boot the CD or DVD on an appropriate computer.

Burning CDs/DVDs in Linux

Linux has both graphical and command-line tools for burning CD and DVD images to physical media. Examples in this section show how to use `K3b` from the desktop or `cdrecord` (or `wodim`) to burn ISO images to CD or DVD. If they are not installed, you can install either one as follows:

For Fedora or RHEL

```
# yum install k3b
# yum install wodim
```

For Debian or Ubuntu

```
# apt-get install k3b
# apt-get install wodim
```

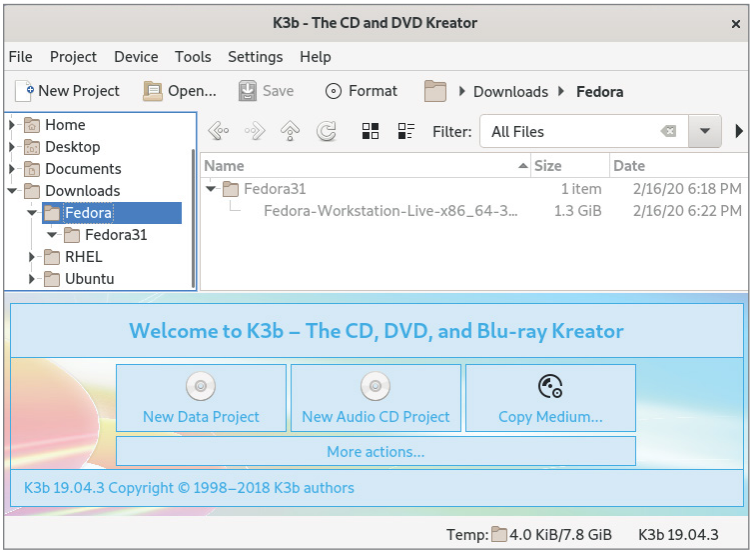
Burning CDs or DVDs from a Linux desktop

Here's how to create bootable Linux CDs or DVDs from a running Linux system (such as Fedora) using `K3b`. `K3b` comes with the KDE desktop but runs on the GNOME desktop as well.

1. Download the ISO images that you want to your computer's hard drive. (A CD image is under about 700MB in size. Single-layer DVD images are under 4.7GB.)
2. Open a CD/DVD burning application. For this procedure, I recommend K3b CD and DVD Kreator (`http://www.k3b.org`). In Fedora, select Activities and type **K3b** (or type **k3b** from a Terminal window). The "K3b – The CD and DVD Kreator" window appears.
3. From the K3b window, select Tools ➪ Burn Image to burn a CD or DVD ISO Image. You are asked to choose an image file.
4. Browse to the image that you just downloaded or copied to hard drive and select it. After you select the image that you want, the Burn Image window appears, as does a checksum on the image. Figure A.3 shows the K3b window ready to select an image of Fedora.

FIGURE A.3

Use K3b to burn your Linux CDs or DVDs.



5. Insert a blank CD or DVD into the CD/DVD drive, which may be a combination CD/DVD drive. (If a CD/DVD Creator window pops up, you can close it.)

6. Check the settings in the Burn Image window (often, the defaults are fine, but you may want to slow down the speed if you get some bad burns). You can also select the Simulate check box to test the burn before actually writing to the CD/DVD. Click Start to continue.
7. When the CD/DVD is finished burning, eject it (or it may eject automatically) and mark it appropriately (information such as the distribution name, version number, date, and name of the ISO image).

Now you're ready to begin installing (or booting) the Linux distribution you just burned.

Burning CDs or DVDs from a Linux command line

If you have no GUI, or you don't mind working from the shell, you can use the `cdrecord` command to burn the ISOs. With a blank CD or DVD inserted and the ISO image you want to burn in the current directory, you can use the following simple command line for burning a CD image to CD or DVD using `cdrecord`:

```
# cdrecord -v whatever.iso
```

See the `cdrecord` man page (`man cdrecord`) for other options available with the `cdrecord` command.

APPENDIX

B

Exercise Answers

This appendix provides answers to each of the chapter exercises. There are many ways to accomplish tasks in Linux. Suggested answers are provided herein.

Some of the exercises require that you modify system files that could change the basic functioning of your system, or even make your system unbootable. Therefore, I recommend that you do the exercises on a Linux system that you are free to modify and erase if something should go wrong. Using virtual machines, that you can discard when you are done, is an excellent option.

Chapter 1: Starting with Linux

There are no exercises in Chapter 1.

Chapter 2: Creating the Perfect Linux Desktop

This section details some ways that these tasks can be completed on both the GNOME 2 and GNOME 3 desktops.

1. To get started, you need a Linux system in front of you to do the procedures in this book. An installed system is preferable, so you don't lose your changes when you reboot. To start out, you can use a Fedora Live CD (or installed system), an Ubuntu installed system, or a Red Hat Enterprise Linux installed system. Here are your choices:
   a. **Fedora Live CD (GNOME 3):** Get a Fedora Live CD as described in Appendix A. Run it live, as described in the section "Starting with the Fedora GNOME Desktop Live image" in Chapter 2, or install it and run it from hard disk as described in Chapter 9, "Installing Linux."
   b. **Ubuntu (GNOME 3):** Install Ubuntu and the GNOME Shell software, as described at the beginning of Chapter 2.
   c. **Red Hat Enterprise Linux 8 (GNOME 3):** Install Red Hat Enterprise Linux 7, as described in Chapter 9.
   d. **Red Hat Enterprise Linux 6 or earlier (GNOME 2):** Install Red Hat Enterprise Linux 6.

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

2. To launch the Firefox web browser and go to the GNOME home page (`http://gnome.org`), there are some easy steps to take. If your network is not working, refer to Chapter 14, "Administering Networking," for help on connecting to wired and wireless networks.

GNOME 3

For GNOME 3, you can press the Windows key to get to the Overview screen. Then type **`Firefox`** to highlight just the Firefox web browser icon. Press Enter to launch it. Type **`http://gnome.org`** in the location box, and press Enter.

GNOME 2

For GNOME 2, select the Firefox icon from the top menu bar. Type **`http://gnome.org`** in the location box, and press Enter.

3. To pick a background that you like from the GNOME art site (`http://gnome-look.org`), download it to your Pictures folder, and select it as your current background. On both GNOME 2 and GNOME 3 systems, do the following:
   a. Type **`http://gnome-look.org/`** in the Firefox location box and press Enter.
   b. Find a background that you like and select it. Then click the Download button and download it to your Pictures folder.
   c. Open your Pictures folder, right-click the image, and select Set as Wallpaper.

      The image is used as your desktop background.
4. To start a Nautilus File Manager window and move it to the second workspace on your desktop, do the following:

For GNOME 3

   a. Press the Windows key.
   b. Select the Files icon from the Dash (left side). A new instance of Nautilus starts in the current workspace.
   c. Right-click the title bar in the Files window and select Move to Monitor Down. The Files window moves to the second workspace.

For GNOME 2

   a. Open the Home folder from the GNOME 2 desktop (double-click).
   b. Right-click in the Nautilus title bar that appears, and select either Move to Workspace Right or Move to Another Workspace. (You can select which workspace you want from the list.)
5. To find the image that you downloaded to use as your desktop background and open it in any image viewer, first go to your Home folder, then open the Pictures folder. Double-click the image to open it in an image viewer.
6. Moving back and forth between the workspace with Firefox on it and the one with the Nautilus file manager is fairly straightforward.

If you did the previous exercises properly, Nautilus and Firefox should be in different workspaces. Here's how you can move between those workspaces in GNOME 3 and GNOME 2:

GNOME 3

Press the Windows key, and select the workspace that you want in the right column. As an alternative, you can go directly to the application that you want by pressing Alt+Tab and pressing Tab again and also arrow keys to highlight the application that you want to open.

GNOME 2

Select the workspace that you want with your mouse by clicking the small representation of the workspace in the right side of the lower panel. If you happen to have Desktop Effects enabled (System ➪ Preferences Desktop Effects ➪ Compiz), try pressing Ctrl+Alt+right arrow (or left arrow) to spin to the next workspace.

7. To open a list of applications installed on your system and select an image viewer to open from that list using as few clicks or keystrokes as possible, do the following:

In GNOME 3

Move the mouse to the upper-left corner of the screen to get to the Overview screen. Select Applications, then select Utilities from the right column, and then select Image Viewer.

In GNOME 2

Select Applications ➪ Graphics ➪ Image Viewer to open an image viewer window on the desktop.

8. To change the view of the windows on your current workspace to smaller views of those windows that you can step through, do the following:

In GNOME 3

With multiple windows open on multiple workspaces, press the Alt+Tab keys. While continuing to hold the Alt key, press Tab until you highlight the application that you want. Release the Alt key to select it.

In GNOME 2

With multiple windows open on multiple workspaces, press and hold the Ctrl+Alt+Tab keys. While continuing to hold the Ctrl+Alt keys, press Tab until you have highlighted the application that you want. Release the Ctrl and Alt keys to select it.

9. To launch a music player from your desktop using only the keyboard, do the following:

In GNOME 3

a. Press the Windows key to go to the Overview screen.

b. Type **Rhyth** (until the icon appears and is highlighted) and press Enter. (In Ubuntu, if you don't have Rhythmbox installed, type **Bansh** to open the Banshee Media Player.)

In GNOME 2

Press Alt+F2. From the Run Application box that appears. Then type **rhythmbox** and press Enter.

10. To take a picture of your desktop using only keystrokes, press the Print Screen key to take a screen shot of your entire desktop in both GNOME 3 and GNOME 2. Press Alt+Print Screen to take a screen shot of just the current window. In both cases, the images are saved to the Pictures folder in your home folder.

Chapter 3: Using the Shell

1. To switch virtual consoles and return to the desktop in Fedora or Ubuntu (this feature is disabled in some RHEL systems), do the following:
   a. Hold Ctrl+Alt and press F2 (Ctrl+Alt+F2). A text-based console should appear.
   b. Type your username (press Enter) and password (press Enter).
   c. Type a few commands, such as **id**, **pwd**, and **ls**.
   d. Type **exit** to exit the shell and return to the login prompt.
   e. Press Ctrl+Alt+F1 to return to the virtual console that holds your desktop. (On different Linux systems, the desktop may be on different virtual consoles. Ctrl+Alt+F7 and Ctrl+Alt+F2 are other common places to find it.)
2. For your Terminal window, make the font red and the background yellow.
   a. From the GNOME desktop, select Applications ➪ System Tools ➪ Terminal to open a Terminal window.
   b. From the Terminal window, select Edit ➪ Profile Preferences.
   c. Select the Colors tab and deselect "Use colors from system theme" box.
   d. Select the box next to Text Color, click the color red that you want from the available selections, and click Select.
   e. Select the box next to Background Color, click the color yellow that you want from the available selections, and click Select.
   f. Click Close on the Profile window to go back to the Terminal window with the new colors.
   g. Go back and reselect "Use colors from system theme" box to go back to the default Terminal colors.
3. Find the `mount` command and `tracepath` man page.
   a. Run `type mount` to see that the `mount` command's location is either `/usr/bin/mount` or `/bin/mount`.

b. Run `locate tracepath` to see that the `tracepath` man page is at `/usr/share/man/man8/tracepath.8.gz`.

4. Run, recall, and change these commands as described:

```
$ cat /etc/passwd
$ ls $HOME
$ date
```

   a. Press the up arrow until you see the `cat /etc/passwd` command. If your cursor is not already at the end of the line, press Ctrl+E to get there. Backspace over the word `passwd`, type the word **`group`**, and press Enter.

   b. Type **`man ls`**, and find the option to list by time (`-t`). Press the up arrow until you see the `ls $HOME` command. Use the left arrow key or Alt+B to position your cursor to the left of `$HOME`. Type **`-t`**, so that the line appears as `ls -t $HOME`. Press Enter to run the command.

   c. Type **`man date`** to view the `date` man page. Use the up arrow to recall the `date` command and add the format indicator that you found. A single `%D` format indicator gets the results you need:

```
$ date +%D
04/27/20
```

5. Use tab completion to type **`basename /usr/share/doc/`**. Type **`basen<Tab> /u<Tab>sh<Tab>do<Tab>`** to get `basename/usr/share/doc/`.

6. Pipe `/etc/services` to the `less` command: `$ cat /etc/services | less`.

7. Make output from the `date` command appear in this format: Today is Thursday, April 23, 2020.

```
$ echo "Today is $(date +'%A, %B %d, %Y')"
```

8. View variables to find your current hostname, username, shell, and home directories.

```
$ echo $HOSTNAME
$ echo $USERNAME
$ echo $SHELL
$ echo $HOME
```

9. Add a permanent `mypass` alias that displays the contents of the `/etc/passwd` file.

   a. Type **`nano $HOME/.bashrc`**.

   b. Move the cursor to an open line at the bottom of the page. (Press Enter to open a new line if needed.)

   c. On its own line, type **`alias m="cat /etc/passwd"`**.

   d. Type Ctrl+O to save and Ctrl+X to exit the file.

   e. Type **`source $HOME/.bashrc`**.

f. Type **alias m** to make sure that the alias was set properly: `alias m='cat /etc/passwd'`.

g. Type **m**. (The `/etc/passwd` file displays on the screen.)

10. To display the man page for the mount system call, use the `man -k` command to find man pages that include the word `mount`. Then use the `mount` command with the correct section number (8) to get the proper `mount` man page:

```
        $ man -k mount | grep ^mount
        mount        (2)  - mount filesystem
        mount        (8)  - mount a filesystem
        ...
        mountpoint   (1)  - see if a directory is a mountpoint
        mountstats   (8)  - Displays various NFS client per-mount
statistics
        $ man 2 mount
        MOUNT(2)      Linux Programmer's Manual
MOUNT(2)
        NAME
              mount - mount file system
        SYNOPSIS
              #include <sys/mount.h>
.
.
.
```

Chapter 4: Moving Around the Filesystem

1. Create the `projects` directory, create nine empty files (`house1` to `house9`), and list just those files.

```
$ mkdir $HOME/projects/
$ touch $HOME/projects/house{1..9}
$ ls $HOME/projects/house{1..9}
```

2. Make the `$HOME/projects/houses/doors/` directory path, and create some empty files in that path.

```
$ cd
$ mkdir $HOME/projects/houses
$ touch $HOME/projects/houses/bungalow.txt
$ mkdir $HOME/projects/houses/doors/
$ touch $HOME/projects/houses/doors/bifold.txt
$ mkdir -p $HOME/projects/outdoors/vegetation/
$ touch $HOME/projects/outdoors/vegetation/landscape.txt
```

3. Copy the files `house1` and `house5` to the `$HOME/projects/houses/` directory.

```
$ cp $HOME/projects/house[15] $HOME/projects/houses
```

4. Recursively copy the /usr/share/doc/initscripts* directory to the $HOME/projects/ directory.

```
$ cp -ra /usr/share/doc/initscripts*/ $HOME/projects/
```

5. Recursively list the contents of the $HOME/projects/ directory. Pipe the output to the less command so that you can page through the output.

```
$ ls -lR $HOME/projects/ | less
```

6. Remove the files house6, house7, and house8 without being prompted.

```
$ rm -f $HOME/projects/house[678]
```

7. Move house3 and house4 to the $HOME/projects/houses/doors directory.

```
$ mv $HOME/projects/house{3,4} $HOME/projects/houses/doors/
```

8. Remove the $HOME/projects/houses/doors directory and its contents.

```
$ rm -rf $HOME/projects/houses/doors/
```

9. Change the permissions on the $HOME/projects/house2 file so that it can be read and written to by the user who owns the file, only read by the group, and have no permission for others.

```
$ chmod 640 $HOME/projects/house2
```

10. Recursively change the permissions of the $HOME/projects/ directory so that nobody has write permission to any files or directories beneath that point in the file system.

```
$ chmod -R a-w $HOME/projects/
$ ls -lR $HOME/projects/
/home/joe/projects/:

total 12

-r--r--r--. 1 joe joe    0 Jan 16 06:49 house1

-r--r-----. 1 joe joe    0 Jan 16 06:49 house2

-r--r--r--. 1 joe joe    0 Jan 16 06:49 house5

-r--r--r--. 1 joe joe    0 Jan 16 06:49 house9

dr-xr-xr-x. 2 joe joe 4096 Jan 16 06:57 houses

dr-xr-xr-x. 2 joe joe 4096 Jul  1  2014 initscripts-9.03.40

dr-xr-xr-x. 3 joe joe 4096 Jan 16 06:53 outdoors
...
```

Chapter 5: Working with Text Files

1. Follow these steps to create the /tmp/services file, and then edit it so that WorldWideWeb appears as World Wide Web.

```
$ cp /etc/services /tmp
$ vi /tmp/services
/WorldWideWeb<Enter>
cwWorld Wide Web<Esc>
```

   The next two lines show the before and after:

```
http            80/tcp     www www-http    # WorldWideWeb HTTP
http            80/tcp     www www-http    # World Wide Web HTTP
```

2. One way to move the paragraph in your /tmp/services file is to search for the first line of the paragraph, delete five lines (5dd), go to the end of the file (G), and put in the text (p):

```
$ vi /tmp/services
/Note that it is<Enter>
5dd
G
p
```

3. To use ex mode to search for every occurrence of the term tcp (case sensitive) in your /tmp/services file, and change it to WHATEVER, you can enter the following:

```
$ vi /tmp/services
:g/tcp/s//WHATEVER/g<Enter>
```

4. To search the /etc directory for every file named passwd and redirect errors from your search to /dev/null, you can enter the following:

```
$ find /etc -name passwd 2> /dev/null
```

5. Create a directory in your home directory called TEST. Create files in that directory named one, two, and three that have full read/write/execute permissions on for everyone (user, group, and other). Construct a find command that would find those files and any other files that have write permission open to "others" from your home directory and below.

```
$ mkdir $HOME/TEST
$ touch $HOME/TEST/{one,two,three}
$ chmod 777 $HOME/TEST/{one,two,three}
$ find $HOME -perm -002 -type f -ls
148120  0 -rwxrwxrwx   1 chris chris 0 Jan  1 08:56 /home/
chris/TEST/two
```

```
        148918  0 -rwxrwxrwx   1 chris chris 0 Jan  1 08:56 home/chris/
TEST/three
        147306  0 -rwxrwxrwx   1 chris chris 0 Jan  1 08:56 /home/chris/
TEST/one
```

6. Find files under the `/usr/share/doc` directory that have not been modified in more than 300 days.

```
$ find /usr/share/doc -mtime +300
```

7. Create a `/tmp/FILES` directory. Find all files under the `/usr/share` directory that are more than 5MB and less than 10MB, and copy them to the `/tmp/FILES` directory.

```
$ mkdir /tmp/FILES
$ find /usr/share -size +5M -size -10M -exec cp {} /tmp/FILES \;
$ du -sh /tmp/FILES/*
6.6M    /tmp/FILES/BidiCharacterTest.txt
7.6M    /tmp/FILES/BidiTest.txt
5.2M    /tmp/FILES/day.jpg
```

8. Find every file in the `/tmp/FILES` directory, and make a backup copy of each file in the same directory. Use each file's existing name and append `.mybackup` to create each backup file.

```
$ find /tmp/FILES/ -type f -exec cp {} {}.mybackup \;
```

9. Install the `kernel-doc` package in Fedora or Red Hat Enterprise Linux. Using `grep`, search inside the files contained in the `/usr/share/doc/kernel-doc*` directory for the term `e1000` (case insensitive), and list the names of the files that contain that term.

```
# yum install kernel-doc
$ cd /usr/share/doc/kernel-doc*
$ grep -rli e1000 .
./Documentation/powerpc/booting-without-of.txt
./Documentation/networking/e100.txt
...
```

10. Search for the `e1000` term again in the same location. However, this time list every line that contains the term and highlight the term in color.

```
$ cd /usr/share/doc/kernel-doc-*
$ grep -ri --color e1000 .
```

Chapter 6: Managing Running Processes

1. To list all processes running on your system with a full set of columns, while piping the output to `less`, enter the following:

```
$ ps -ef | less
```

2. To list all processes running on the system and sort those processes by the name of the user running each process, enter the following:

```
$ ps -ef --sort=user | less
```

3. To list all processes running on the system with the column names process ID, username, group name, nice value, virtual memory size, resident memory size, and command, enter the following:

```
$ ps -eo 'pid,user,group,nice,vsz,rss,comm' | less
  PID USER     GROUP     NI    VSZ   RSS COMMAND
    1 root     root       0  19324  1236 init
    2 root     root       0      0     0 kthreadd
    3 root     root       -      0     0 migration/0
    4 root     root       0      0     0 ksoftirqd/0
```

4. To run the `top` command and then go back and forth between sorting by CPU usage and memory consumption, enter the following:

```
$ top
P
M
P
M
```

5. To start the `gedit` process from your desktop and use the System Monitor window to kill that process, do the following:

```
$ gedit &
```

   Next, in GNOME 2, select Applications ➪ System Tools ➪ System Monitor, or in GNOME 3, from the Activities screen, type **System Monitor** and press Enter. Find the `gedit` process on the Processes tab. (You can sort alphabetically to make it easier by clicking the Process Name heading.) Right-click the `gedit` command, and then select either End Process or Kill Process; the `gedit` window on your screen should disappear.

6. To run the `gedit` process and use the `kill` command to send a signal to pause (stop) that process, enter the following:

```
$ gedit &
[1] 21532

$ kill -SIGSTOP 21532
```

7. To use the `killall` command to tell the `gedit` command (paused in the previous exercise) to continue working, do the following:

```
$ killall -SIGCONT gedit
```

   Make sure that the text you typed after `gedit` was paused now appears in the window.

8. To install the `xeyes` command, run it about 20 times in the background, and run `killall` to kill all 20 `xeyes` processes at once, enter the following:

```
# yum install xorg-x11-apps
$ xeyes &
$ xeyes &
...
$ killall xeyes &
```

   Remember, you need to be the root user to install the package. After that, remember to repeat the `xeyes` command 20 times. Spread the windows around on your screen, and move the mouse for fun to watch the eyes move. All the `xeyes` windows should disappear at once when you type `killall xeyes`.

9. As a regular user, run the `gedit` command so that it starts with a nice value of `5`.

```
# nice -n 5 gedit &
[1] 21578
```

10. To use the `renice` command to change the nice value of the `gedit` command you just started to `7`, enter the following:

```
# renice -n 7 21578
21578: old priority 0, new priority 7
```

    Use any command you like to verify that the current nice value for the `gedit` command is now set to `7`. For example, you could type the following:

```
# ps -eo 'pid,user,nice,comm' | grep gedit
21578 chris     7 gedit
```

Chapter 7: Writing Simple Shell Scripts

1. Here's an example of how to create a script in your `$HOME/bin` directory called `myownscript`. When the script runs, it should output information that appears as follows:

```
Today is Sat Jun 10 15:45:04 EDT 2019.
You are in /home/joe and your host is abc.example.com.
```

   The following steps show one way to create the script named `myownscript`:

   a. If it doesn't already exist, create a bin directory:

```
$ mkdir $HOME/bin
```

   b. Using any text editor, create a script called `$HOME/bin/myownscript` that contains the following:

```
#!/bin/bash
# myownscript
# List some information about your current system
```

```
echo "Today is $(date)."
echo "You are in $(pwd) and your host is $(hostname)."
```

c. Make the script executable:

```
$ chmod 755 $HOME/bin/myownscript
```

2. Create a script that reads in three positional parameters from the command line, assigns those parameters to variables named ONE, TWO, and THREE, respectively. Also, replace X with the number of parameters and Y with all of the parameters entered. Then replace A with the contents of variable ONE, B with variable TWO, and C with variable THREE, as shown below:

   a. To create the script, open a file named $HOME/bin/myposition and add the following contents:

```
#!/bin/bash
# myposition
ONE=$1
TWO=$2
THREE=$3
echo "There are $# parameters that include: $@"
echo "The first is $ONE, the second is $TWO, the third is
$THREE."
```

   b. To make the script called $HOME/bin/myposition executable, enter the following:

```
$ chmod 755 $HOME/bin/myposition
```

   c. To test it, run it with some command-line arguments, as in the following:

```
$ myposition Where Is My Hat Buddy?
There are 5 parameters that include: Where Is My Hat Buddy?
The first is Where, the second is Is, the third is My.
```

3. To create the script described, do the following:

   a. To create a file called $HOME/bin/myhome and make it executable, enter the following:

```
$ touch $HOME/bin/myhome
$ chmod 755 $HOME/bin/myhome
```

   b. Here's what the script myhome might look like:

```
#!/bin/bash
# myhome
read -p "What street did you grow up on? " mystreet
read -p "What town did you grow up in? " mytown
echo "The street I grew up on was $mystreet and the town
was $mytown."
```

c. Run the script to check that it works. The following example shows what the input and output for the script could look like:

```
$ myhome
What street did you grow up on? Harrison
What town did you grow up in? Princeton
The street I grew up on was Harrison and the town was Princeton.
```

4. To create the required script, do the following:

   a. Using any text editor, create a script called `$HOME/bin/myos` and make the script executable:

```
$ touch $HOME/bin/myos
$ chmod 755 $HOME/bin/myos
```

   b. The script could contain the following:

```
#!/bin/bash
# myos
read -p "What is your favorite operating system, Mac,
Windows or Linux? " opsys
if [ $opsys = Mac ] ; then
  echo "Mac is nice, but not tough enough for me."
elif [ $opsys = Windows ] ; then
  echo "I used Windows once. What is that blue screen
for?"
elif [ $opsys = Linux ] ; then
  echo "Great Choice!"
else
  echo "Is $opsys an operating system?"
fi
```

5. To create a script named `$HOME/bin/animals` that runs through the words *moose*, *cow*, *goose*, and *sow* through a `for` loop and have each of those words appended to the end of the line "I have a. . .," do the following:

   a. Make the script executable:

```
$ touch $HOME/bin/animals
$ chmod 755 $HOME/bin/animals
```

   b. The script could contain the following:

```
#!/bin/bash
# animals
for ANIMALS in moose cow goose sow ; do
  echo "I have a $ANIMALS"
done
```

   c. When you run the script, the output should appear as follows:

```
$ animals
I have a moose
```

```
I have a cow
I have a goose
I have a sow
```

Chapter 8: Learning System Administration

1. To enable Cockpit on your system, enter the following:

```
# systemctl enable --now cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/
cockpit.socket
   → /usr/lib/systemd/system/cockpit.socket
```

2. To open the Cockpit interface in your web browser, enter the hostname or IP address of the system holding your Cockpit service, followed by port number 9090. For example, enter this into the location box of your browser:

```
https://host1.example.com:9090/
```

3. To find all of the files under the `/var/spool` directory that are owned by users other than root and do a long listing of them, enter the following. (I recommend becoming root to find files that might be closed off to other users.)

```
$ su -
Password: *********
# find /var/spool -not -user root -ls | less
```

4. To become root user and create an empty or plain-text file named `/mnt/test.txt`, enter the following:

```
$ su -
Password: *********
# touch /mnt/test.txt
# ls -l /mnt/test.txt
-rw-r--r--. 1 root root 0 Jan  9 21:51 /mnt/test.txt
```

5. To become root and edit the `/etc/sudoers` file to allow your regular user account (for example, `bill`) to have full root privilege via the `sudo` command, do the following:

```
$ su -
Password: *********
# visudo
o
bill     ALL=(ALL)      ALL
Esc ZZ
```

   Because `visudo` opens the `/etc/sudoers` file in `vi`, the example types `o` to open a line, and then it types in the line to allow bill to have full root privilege.

After the line is typed, press Esc to return to command mode and type **ZZ** to write and quit.

6. To use the `sudo` command to create a file called `/mnt/test2.txt` and verify that the file is there and owned by the root user, enter the following:

```
[bill]$ sudo touch /mnt/test2.txt
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:
    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
[sudo] password for bill:  *********
[bill]$ ls -l /mnt/text2.txt
-rw-r--r--. 1 root root 0 Jan  9 23:37 /mnt/text2.txt
```

7. Do the following to mount and unmount a USB drive and watch the system journal during this process:

   a. Run the `journalctl -f` command as root in a Terminal window and watch the output from here for the next few steps.

```
# journalctl -f
Jan 25 16:07:59 host2 kernel: usb 1-1.1: new high-speed USB device
    number 16 using ehci-pci
Jan 25 16:07:59 host2 kernel: usb 1-1.1: New USB device found,
    idVendor=0ea0, idProduct=2168
Jan 25 16:07:59 host2 kernel: usb 1-1.1: New USB device strings:
    Mfr=1, Product=2, SerialNumber=3
Jan 25 16:07:59 host2 kernel: usb 1-1.1: Product: Flash Disk
Jan 25 16:07:59 host2 kernel: usb 1-1.1: Manufacturer: USB
...
Jan 25 16:08:01 host2 kernel: sd 18:0:0:0: [sdb] Write Protect is off
Jan 25 16:08:01 host2 kernel: sd 18:0:0:0: [sdb]
    Assuming drive cache: write through
Jan 25 16:08:01 host2 kernel:  sdb: sdb1
Jan 25 16:08:01 host2 kernel: sd 18:0:0:0: [sdb]
    Attached SCSI removable disk
```

   b. Plug in a USB storage drive that mounts a filesystem from that drive automatically. If it does not, run the following commands in a second terminal (as root) to create a mount point directory and mount the device:

```
$ mkdir /mnt/test
$ mount /dev/sdb1 /mnt/test
$ umount /dev/sdb1
```

8. To see what USB devices are connected to your computer, enter the following:

```
$ lsusb
```

9. To load the `bttv` module, list the modules that were loaded, and unload it, enter the following:

```
# modprobe -a bttv
# lsmod | grep bttv
ttv                   167936  0
tea575x                16384  1 bttv
tveeprom               28672  1 bttv
videobuf_dma_sg        24576  1 bttv
videobuf_core          32768  2 videobuf_dma_sg,bttv
v4l2_common            16384  1 bttv
videodev              233472  3 tea575x,v4l2_common,bttv
i2c_algo_bit           16384  1 bttv
```

   Notice that other modules (`v4l2_common`, `videodev`, and others) were loaded when you loaded `bttv` with `modprobe -a`.

10. Enter the following to remove the `bttv` module along with any other modules that were loaded with it. Notice that they were all gone after running `modprobe -r`.

```
# modprobe -r bttv
# lsmod | grep bttv
```

Chapter 9: Installing Linux

1. To install a Fedora system from Fedora Live media, follow the instructions in the section "Installing Fedora from Live Media" in Chapter 9. In general, those steps include the following:
   a. Booting the Live media.
   b. Selecting to install to the hard drive when the system boots up.
   c. Adding information from the summary page needed to configure your system initially.
   d. Rebooting your computer and removing the Live medium so that the newly installed system boots from the hard drive.
2. To update the packages, after the Fedora Live media installation is complete, do the following:
   a. Reboot the computer and fill in the first boot questions as prompted.
   b. Using a wired or wireless connection, make sure that you have a connection to the Internet. Refer to Chapter 14, "Administering Networking," if you have trouble getting your networking connection to work properly. Open a shell as the root user and type **`sudo dnf update`**.
   c. When prompted, type **`y`** to accept the list of packages displayed. The system begins downloading and installing the packages.

3. To run the RHEL installation in text mode, do the following:
   a. Boot the RHEL DVD.
   b. When you see the boot menu, highlight one of the installation boot entries and press Tab. Move the cursor right to the end of the kernel line, and type the literal option **`text`** at the end of that line. Press Enter to start the installer.
   c. Try out the rest of the installation in text mode.
4. To set the disk partitioning as described in question 4 for a Red Hat Enterprise Linux DVD installation, do the following:

NOTE
This procedure ultimately deletes all content on your hard disk. If you just want to use this exercise to practice partitioning, you can reboot your computer before starting the actual installation process without harming your hard disk. After you go forward and partition your disk, assume that all data has been deleted.

   a. On a computer that you can erase with at least 10GB of disk space, insert a RHEL installation DVD, reboot, and begin stepping through the installation screens.
   b. When you get to the Installation Summary screen, select Installation Destination.
   c. From the Installation Destination screen, select the device to use for the installation (probably `sda` if you have a single hard disk that you can completely erase or `vda` for a virtual install).
   d. Select the Custom button.
   e. Select Done to get to the Manual Partitioning screen.
   f. If the existing disk space is already consumed, you need to delete the partitions before proceeding.
   g. Click the plus (+) button at the bottom of the screen. Then add each of the following mount points:

      `/boot - 400M`

      `/ - 3G`

      `/var - 2G`

      `/home -2G`

   h. Select Done. You should see a summary of changes.
   i. If the changes look acceptable, select Accept Changes. If you are just practicing and don't actually want to change your partitions, select Cancel & Return to Custom Partitioning. Then simply exit the installer.

Chapter 10: Getting and Managing Software

1. To search the YUM repository for the package that provides the `mogrify` command, enter the following:

```
# yum provides mogrify
```

2. To display information about the package that provides the `mogrify` command and determine what is that package's home page (URL), enter the following:

```
# yum info ImageMagick
```

   You will see that the URL to the home page for ImageMagick is `http://www.imagemagick.org`.

3. To install the package containing the `mogrify` command, enter the following:

```
# yum install ImageMagick
```

4. To list all of the documentation files contained in the package that provides the `mogrify` command, enter the following:

```
# rpm -qd ImageMagick
...
/usr/share/doc/ImageMagick/README.txt
...
/usr/share/man/man1/identify.1.gz
/usr/share/man/man1/import.1.gz
/usr/share/man/man1/mogrify.1.gz
```

5. To look through the change log of the package that provides the `mogrify` command, enter the following:

```
# rpm -q --changelog ImageMagick | less
```

6. To delete the `mogrify` command from your system and verify its package against the RPM database to see that the command is indeed missing, enter the following:

```
# type mogrify
mogrify is /usr/bin/mogrify
# rm /usr/bin/mogrify
rm remove regular file '/usr/bin/mogrify'? y
# rpm -V ImageMagick
missing   /usr/bin/mogrify
```

7. To reinstall the package that provides the `mogrify` command and make sure that the entire package is intact again, enter the following:

```
# yum reinstall ImageMagick
# rpm -V ImageMagick
```

8. To download the package that provides the `mogrify` command to your current directory, enter the following:

```
# yum download ImageMagick
ImageMagick-6.9.10.28-1.fc30.x86_64.rpm
```

9. To display general information about the package that you just downloaded by querying the package's RPM file in the current directory, enter the following:

```
# rpm -qip ImageMagick-6.9.10.28-1.fc30.x86_64.rpm
Name        : ImageMagick
Epoch       : 1
Version     : 6.9.10.28
Release     : 1.fc30



...
```

10. To remove the package containing the `mogrify` command from your system, enter the following:

```
# yum remove ImageMagick
```

Chapter 11: Managing User Accounts

For questions that involve adding and removing user accounts, you can use the Users window, the User Manager window, or command-line tools such as `useradd` and `usermod`. The point is to make sure that you get the correct results shown in the answers that follow, not necessarily to do it exactly in the same way that I did.

There are multiple ways that you can achieve the same results. The answers here show how to complete the exercises from the command line. (Become root user when you see a # prompt.)

1. To add a local user account to your Linux system that has a username of `jbaxter` and a full name of John Baxter, which uses `/bin/sh` as its default shell and is the next available UID (yours may differ from the one shown here), enter the following. You can use the `grep` command to check the new user account. Then set the password for `jbaxter` to: `My1N1te0ut!`

```
# useradd -c "John Baxter" -s /bin/sh jbaxter
# grep jbaxter /etc/passwd
jbaxter:x:1001:1001:John Baxter:/home/jbaxter:/bin/sh
# passwd jbaxter
Changing password for user jbaxter
New password: My1N1te0ut!
Retype new password: My1N1te0ut!
passwd: all authentication tokens updated successfully
```

2. To create a group account named `testing` that uses group ID 315, enter the following:

```
# groupadd -g 315 testing
# grep testing /etc/group
testing:x:315:
```

3. To add jbaxter to the testing group and the bin group, enter the following:

```
# usermod -aG testing,bin jbaxter
# grep jbaxter /etc/group
bin:x:1:bin,daemon,jbaxter
jbaxter:x:1001:
testing:x:315:jbaxter
```

4. To become jbaxter and temporarily have the testing group be jbaxter's default group, run touch /home/jbaxter/file.txt so that the testing group is assigned as the file's group, and do the following:

```
$ su - jbaxter
Password: My1N1te0ut!
sh-4.2$ newgrp testing
sh-4.2$ touch /home/jbaxter/file.txt
sh-4.2$ ls -l /home/baxter/file.txt
-rw-rw-r--. 1 jbaxter testing 0 Jan 25 06:42 /home/jbaxter/file.
txt
sh-4.2$ exit ; exit
```

5. Note what user ID has been assigned to jbaxter, and then delete the user account without deleting the home directory assigned to jbaxter.

```
$ userdel jbaxter
```

6. Use the following command to find any files in the /home directory (and any subdirectories) that are assigned to the user ID that recently belonged to the user named jbaxter. (When I did it, the UID/GID were both 1001; yours may differ.) Notice that the username jbaxter is no longer assigned on the system, so any files that user created are listed as belonging to UID 1001 and GID 1001, except for a couple of files that were assigned to the testing group because of the newgrp command run earlier:

```
# find /home -uid 1001 -ls
262184  4 drwx------ 4 1001  1001  4096 Jan 25 08:00 /home/
jbaxter
262193  4 -rw-r--r-- 1 1001  1001   176 Jan 27  2011 /home/
jbaxter/.bash_profile
262196  4 -rw------- 1 13602 testing 93 Jan 25 08:00 /home/
jbaxter/.bash_history
262194  0 -rw-rw-r-- 1 13602 testing  0 Jan 25 07:59 /home/
jbaxter/file.txt
...
```

7. Run these commands to copy the /etc/services file to the /etc/skel/ directory; then add a new user to the system named mjones, with a full name of Mary Jones and a home directory of /home/maryjones. List her home directory to make sure that the services file is there.

```
# cp /etc/services /etc/skel/
# useradd -d /home/maryjones -c "Mary Jones" mjones
```

```
# ls -l /home/maryjones
total 628
-rw-r--r--. 1 mjones mjones 640999 Jan 25 06:27 services
```

8. Run the following command to find all files under the `/home` directory that belong to `mjones`. If you did the exercises in order, notice that after you deleted the user with the highest user ID and group ID, those numbers were assigned to `mjones`. As a result, any files left on the system by `jbaxter` now belong to `mjones`. (For this reason, you should remove or change ownership of files left behind when you delete a user.)

```
# find /home -user mjones -ls
262184 4 drwx------ 4 mjones mjones 4096 Jan 25 08:00 /
home/jbaxter
262193 4 -rw-r--r-- 1 mjones mjones 176 Jan 27 2011 /home/
jbaxter/.bash_profile
262189 4 -rw-r--r-- 1 mjones mjones 18 Jan 27 2011 /home/
jbaxter/.bash_logout
262194 0 -rw-rw-r-- 1 mjones testing 0 Jan 25 07:59 /home/
jbaxter/file.txt
262188 4 -rw-r--r-- 1 mjones mjones 124 Jan 27 2011 /home/
jbaxter/.bashrc
262197 4 drwx------ 4 mjones  mjones 4096 Jan 25 08:27 /
home/maryjones
262207 4 -rw-r--r-- 1 mjones mjones 176 Jan 27 2011 /home/
maryjones/.bash_profile
262202 4 -rw-r--r-- 1 mjones mjones 18 Jan 27 2011 /home/
maryjones/.bash_logout
262206 628 -rw-r--r-- 1 mjones mjones 640999 Jan 25 08:27 /
home/maryjones/services
262201 4 -rw-r--r-- 1 mjones mjones 124 Jan 27 2011 /home/
maryjones/.bashrc
```

9. As the user `mjones`, you can use the following to create a file called `/tmp/maryfile.txt`, and use ACLs to assign the `bin` user read/write permission and the `lp` group read/write permission to that file.

```
[mjones]$ touch /tmp/maryfile.txt
[mjones]$ setfacl -m u:bin:rw /tmp/maryfile.txt
[mjones]$ setfacl -m g:lp:rw /tmp/maryfile.txt
[mjones]$ getfacl /tmp/maryfile.txt
# file: tmp/maryfile.txt
# owner: mjones
# group: mjones
user::rw-
user:bin:rw-
group::rw-
group:lp:rw-
mask::rw-
other::r& —
```

10. Run this set of commands (as `mjones`) to create a directory named `/tmp/mydir`, and use ACLs to assign default permissions to it so that the `adm` user has read/write/execute permission to that directory and any files or directories created in it. Test that it worked by creating the `/tmp/mydir/testing/` directory and `/tmp/mydir/newfile.txt`.

```
[mary]$ mkdir /tmp/mydir
[mary]$ setfacl -m d:u:adm:rwx /tmp/mydir
[mjones]$ getfacl /tmp/mydir
# file: tmp/mydir
# owner: mjones
# group: mjones
user::rwx
group::rwx
other::r-x
default:user::rwx
default:user:adm:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
[mjones]$ mkdir /tmp/mydir/testing
[mjones]$ touch /tmp/mydir/newfile.txt
[mjones]$ getfacl /tmp/mydir/testing/
# file: tmp/mydir/testing/
# owner: mjones
# group: mjones
user::rwx
user:adm:rwx
group::rwx
mask::rwx
other::r-x
default:user::rwx
default:user:adm:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
[mjones]$ getfacl /tmp/mydir/newfile.txt
# file: tmp/mydir/newfile.txt
# owner: mjones
# group: mjones
user::rw-
user:adm:rwx     #effective:rw-
group::rwx       #effective:rw-
mask::rw-
other::r--
```

Notice that the adm user effectively has only rw- permission. To remedy that, you need to expand the permissions of the mask. One way to do that is with the chmod command, as follows:

```
[mjones]$ chmod 775 /tmp/mydir/newfile.txt
[mjones]$ getfacl /tmp/mydir/newfile.txt
# file: tmp/mydir/newfile.txt
# owner: mjones
# group: mjones
user::rwx
user:adm:rwx
group::rwx
mask::rwx
other::r-x
```

Chapter 12: Managing Disks and Filesystems

1. To determine the device name of a USB flash drive that you want to insert into your computer, enter the following and insert the USB flash drive. (Press Ctrl+C after you have seen the appropriate messages.)

```
# journalctl -f
kernel: [sdb] 15667200 512-byte logical blocks:
     (8.02 GB/7.47 GiB)
Feb 11 21:55:59 cnegus kernel: sd 7:0:0:0:
     [sdb] Write Protect is off
Feb 11 21:55:59 cnegus kernel: [sdb] Assuming
     drive cache: write through
Feb 11 21:55:59 cnegus kernel: [sdb] Assuming
     drive cache: write through
```

2. To list partitions on the USB flash drive on a RHEL 6 system, enter the following:

```
# fdisk -c -u -l /dev/sdb
```

   To list partitions on a RHEL 7, RHEL 8, or Fedora system, enter the following:

```
# fdisk -l /dev/sdb
```

3. To delete partitions on the USB flash drive, assuming device /dev/sdb, do the following:

```
# fdisk /dev/sdb
Command (m for help): d
Partition number (1-6): 6
Command (m for help): d
Partition number (1-5): 5
Command (m for help): d
```

```
Partition number (1-5): 4
Command (m for help): d
Partition number (1-4): 3
Command (m for help): d
Partition number (1-4): 2
Command (m for help): d
Selected partition 1
Command (m for help): w
# partprobe /dev/sdb
```

4. To add a 100MB Linux partition, 200MB swap partition, and 500MB LVM partition to the USB flash drive, enter the following:

```
# fdisk /dev/sdb

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First sector (2048-15667199, default 2048):  <ENTER>
Last sector, +sectors or +size{K,M,G} (default 15667199): +100M
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First sector (616448-8342527, default 616448):  <ENTER>
Last sector, +sectors or +size{K,M,G} (default 15667199): +200M
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 3
First sector (616448-15667199, default 616448):  <ENTER>
Using default value 616448
Last sector, +sectors or +size{K,M,G} (default 15667199): +500M
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)
Command (m for help): t
Partition number (1-4): 3
Hex code (type L to list codes): 8e
Changed system type of partition 3 to 8e (Linux LVM)
```

```
Command (m for help): w
# partprobe /dev/sdb
# grep sdb /proc/partitions
   8       16    7833600 sdb
   8       17     102400 sdb1
   8       18     204800 sdb2
   8       19     512000 sdb3
```

5. To put an ext4 filesystem on the Linux partition, enter the following:

```
# mkfs -t ext4 /dev/sdb1
```

6. To create a mount point called `/mnt/mypart` and mount the Linux partition on it, do the following:

```
# mkdir /mnt/mypart
# mount -t ext4 /dev/sdb1 /mnt/mypart
```

7. To enable the swap partition and turn it on so that additional swap space is immediately available, enter the following:

```
# mkswap /dev/sdb2
# swapon /dev/sdb2
```

8. To create a volume group called `abc` from the LVM partition, create a 200MB logical volume from that group called `data`, create a VFAT filesystem on it, temporarily mount the logical volume on a new directory named `/mnt/test`, and then check that it was successfully mounted, enter the following:

```
# pvcreate /dev/sdb3
# vgcreate abc /dev/sdb3
# lvcreate -n data -L 200M abc
# mkfs -t vfat /dev/mapper/abc-data
# mkdir /mnt/test
# mount /dev/mapper/abc-data /mnt/test
```

9. To grow the logical volume from 200MB to 300MB, enter the following:

```
# lvextend -L +100M /dev/mapper/abc-data
# resize2fs -p /dev/mapper/abc-data
```

10. To remove the USB flash drive safely from the computer, do the following:

```
# umount /dev/sdb1
# swapoff /dev/sdb2
# umount /mnt/test
# lvremove /dev/mapper/abc-data
# vgremove abc
# pvremove /dev/sdb3
```

You can now safely remove the USB flash drive from the computer.

Chapter 13: Understanding Server Administration

1. To log in to any account on another computer using the `ssh` command, enter the following and then enter the password when prompted:

```
$ ssh joe@localhost
joe@localhost's password:
*********
[joe]$
```

2. To display the contents of a remote `/etc/system-release` file and have its contents displayed on the local system using remote execution with the `ssh` command, do the following:

```
$ ssh joe@localhost "cat /etc/system-release"
joe@localhost's password: *******
Fedora release 30 (Thirty)
```

3. To use X11 forwarding to display a `gedit` window on your local system and then save a file on the remote home directory, do the following:

```
$ ssh -X joe@localhost "gedit newfile"
joe@localhost's password: ********
$ ssh joe@localhost "cat newfile"
joe@localhost's password: ********
This is text from the file I saved in joe's remote home
directory
```

4. To copy all of the files from the `/usr/share/selinux` directory recursively on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files are updated to the time on the local system when they are copied, do the following:

```
$ scp -r joe@localhost:/usr/share/selinux /tmp
joe@localhost's password:
 ********
irc.pp.bz2                          100% 9673     9.5KB/s   00:00
dcc.pp.bz2                          100%   15KB  15.2KB/s   00:01
$ ls -l /tmp/selinux | head
total 20
drwxr-xr-x. 3 root root  4096 Apr 18 05:52 devel
drwxr-xr-x. 2 root root  4096 Apr 18 05:52 packages
drwxr-xr-x. 2 root root 12288 Apr 18 05:52 targeted
```

5. To copy all of the files from the `/usr/share/logwatch` directory recursively on a remote system to the `/tmp` directory on your local system in such a way that all of the modification times on the files from the remote system are maintained on the local system, try the following:

```
$ rsync -av joe@localhost:/usr/share/logwatch /tmp
joe@localhost's password: ********
```

```
receiving incremental file list
logwatch/
logwatch/default.conf/
logwatch/default.conf/logwatch.conf
$ ls -l /tmp/logwatch | head
total 16
drwxr-xr-x. 5 root root 4096 Apr 19  2011 default.conf
drwxr-xr-x. 4 root root 4096 Feb 28  2011 dist.conf
drwxr-xr-x. 2 root root 4096 Apr 19  2011 lib
```

6. To create a public/private key pair to use for SSH communications (no passphrase on the key), copy the public key file to a remote user's account with `ssh-copy-id`, and use key-based authentication to log in to that user account without having to enter a password, use the following code:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/joe/.ssh/id_
rsa): ENTER
/home/joe/.ssh/id_rsa already exists.
Enter passphrase (empty for no passphrase):  ENTER
Enter same passphrase again:  ENTER
Your identification has been saved in /home/joe/.ssh/id_
rsa.
Your public key has been saved in /home/joe/.ssh/id_rsa.
pub.
The key fingerprint is:
58:ab:c1:95:b6:10:7a:aa:7c:c5:ab:bd:f3:4f:89:1e joe@cnegus.
csb
The key's randomart image is:
...
$ ssh-copy-id -i ~/.ssh/id_rsa.pub joe@localhost
joe@localhost's password: ********
Now try logging into the machine, with "ssh 'joe@
localhost'",
and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't
expecting.
$ ssh joe@localhost
$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAyN2Psp5/
LRUC9E8BDCx53yPUa0qoOPd

v6H4sF3vmn04V6E7D1iXpzwPzdo4rpvmR1ZiinHR2xGAEr2uZag7feKgLnww2KPcQ6S
iR7lzrOhQjV+SGb/a1dxrIeZqKMq1Tk07G4EvboIrq//9J47vI4l7iNu0x
RmjI3TTxa
DdCTbpG6J3uSJm1BKzdUtwb413x35W2bRgMI75aIdeBsDgQBBiOdu+zuTM
rXJj2viCA
```

```
        XeJ7gIwRvBaMQdOSvSdlkX353tmIjmJheWdgCccM/1jKdoELpaevg9anCe/
yUP3so31
        tTo4I+qTfzAQD5+66oqW0LgMkWVvfZI7dUz3WUPmcMw== chris@abc.
example.com
```

7. To create an entry in `/etc/rsyslog.conf` that stores all authentication messages at the info level and higher into a file named `/var/log/myauth`, do the following. Watch from one terminal as the data comes in.

```
        # vim /etc/rsyslog.conf
        authpriv.info                                /var/log/myauth
        # service rsyslog restart
             or
        # systemctl restart rsyslog.service
        <Terminal 1>                                 <Terminal 2>
        # tail -f /var/log/myauth                    $ ssh joe@
localhost
        Apr 18 06:19:34 abc unix_chkpwd[30631]       joe@localhost's
password:
        Apr 18 06:19:34 abc sshd[30631]              Permission
denied,try again
         :pam_unix(sshd:auth):
         authentication failure;logname= uid=501
         euid=501 tty=ssh ruser= rhost=localhost
         user=joe
        Apr 18 06:19:34 abc sshd[30631]:
         Failed password for joe from
         127.0.0.1 port 5564 ssh2
```

8. To determine the largest directory structures under `/usr/share`, sort them from largest to smallest, and list the top 10 of those directories in terms of size using the `du` command, enter the following:

```
        $ du -s /usr/share/* | sort -rn | head
        527800 /usr/share/locale
        277108 /usr/share/fonts
        196232 /usr/share/help

        134984 /usr/share/backgrounds
        ...
```

9. To show the space that is used and available from all of the filesystems currently attached to the local system, but exclude any `tmpfs` or `devtmpfs` filesystems by using the `df` command, enter the following:

```
        $ df -h -x tmpfs -x devtmpfs
        Filesystem      Size  Used Avail Use% Mounted on
        /deev/sda4       20G  4.2G 16G    22% /
```

10. To find any files in the `/usr` directory that are more than 10MB in size, do the following:

```
$ find /usr -size +10M
/usr/lib/locale/locale-archive
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.fc30.
x86_64/jre/lib/rt.jar
/usr/libexec/cni/dhcp
/usr/libexec/gdb
/usr/libexec/gcc/x86_64-redhat-linux/9/lto1
/usr/libexec/gcc/x86_64-redhat-linux/9/cc1
```

Chapter 14: Administering Networking

1. To use the desktop to check that NetworkManager has successfully started your network interface (wired or wireless), do the following:
   a. Left-click the upper-right corner of your GNOME desktop to see the drop-down menu. Any active wired or wireless network connections should appear on that menu.
   b. If it has not connected to the network, select from the list of wired or wireless networks available, and then enter the username and password, if prompted, to start an active connection.
2. To run a command to check the active network interfaces available on your computer, enter the following:

```
$ ifconfig
```

   or

```
$ ip addr show
```

3. Try to contact `google.com` from the command line in a way that ensures that DNS is working properly:

```
$ ping google.com
Ctrl-C
```

4. To run a command to check the routes being used to communicate outside of your local network, enter the following:

```
$ route
```

5. To trace the route being taken to connect to `google.com`, use the `traceroute` command:

```
$ traceroute google.com
```

6. To view the network interfaces and related network activities for your Linux system through Cockpit, open a web browser to port 9090 using an IP address or hostname. For example: `https://localhost:9090/network`.

7. To create a host entry that allows you to communicate with your local host system using the name `myownhost`, edit the `/etc/hosts` file (`vi /etc/hosts`), and add `myownhost` to the end of the localhost entry so that it appears as follows (then ping `myownhost` to see if it worked):

```
127.0.0.1        localhost.localdomain localhost myownhost
# ping myownhost
Ctrl+C
```

8. To see the DNS name servers being used to resolve hostnames and IP addresses on your system (yours will be different than those shown below), enter the following:

```
# cat /etc/resolv.conf
        nameserver 10.83.14.9
        nameserver 10.18.2.10
        nameserver 192.168.1.254
        # dig google.com
        ...
        google.com.     91941   IN      NS      ns3.google.com.
        ;; Query time: 0 msec
        ;; SERVER: 10.18.2.9#53(10.18.2.9)
        ;; WHEN: Sat Nov 23 20:18:56 EST 2019
        ;; MSG SIZE  rcvd: 276
```

9. To create a custom route that directs traffic destined for the 192.168.99.0/255.255.255.0 network to some IP address on your local network, such as 192.168.0.5 (first ensuring that the 192.168.99 network is not being used at your location), do the following:

   a. Determine the name of your network interface, for example `enp4s0`. In that case, as root run the following commands:

```
# cd /etc/sysconfig/network-scripts
# vi route-enp4s0
```

   b. Add the following lines to that file:

```
ADDRESS0=192.168.99.0
NETMASK0=255.255.255.0
GATEWAY0=192.168.0.5
```

   c. Restart networking and run `route` to see that the route is active:

```
# systemctl restart NetworkManager
# route -n
Kernel IP routing table
Destination   Gateway       Genmask        Flags Metric
Ref Use Iface
192.168.0.1   0.0.0.0       255.255.255.0  U     600
0     0 enp4s0
```

```
          192.168.99.0   192.168.0.5    255.255.255.0   UG    600
0      0 enp4s0
```

10. To check to see if your system has been configured to allow IPv4 packets to be routed between network interfaces on your system, enter the following:

```
# cat /proc/sys/net/ipv4/ip_forward
0
```

A `0` shows that IPv4 packet forwarding is disabled; a `1` shows that it is enabled.

Chapter 15: Starting and Stopping Services

1. To determine which initialization daemon your server is currently using, consider the following:

   a. In most cases today, PID 1 appears as the `systemd` daemon:

```
# ps -ef | head
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 17:01 ?        00:00:04 /usr/
lib/systemd/systemd --
             switched-root --system --deserialize 18
```

   If you type `ps -ef` and PID 1 is `init`, it still might be the `systemd` daemon. Use the `strings` command to see if `systemd` is in use:

```
# strings /sbin/init | grep -i systemd
systemd.unit=
systemd.log_target=
systemd.log_level=
...
```

   b. Most likely, you have the Upstart, SysVinit, or BSD `init` daemon if your `init` daemon is not `systemd`. But double-check at `http://wikipedia.org/wiki/Init`.

2. The tools you use to manage services depend primarily on which initialization system is in use. Try to run the `systemctl` and `service` commands to determine the type of initialization script in use for the `ssh` service on your system:

   a. For `systemd`, a positive result, shown here, means that the `sshd` has been converted to `systemd`:

```
# systemctl status sshd.service
sshd.service - OpenSSH server daemon
  Loaded: loaded (/lib/systemd/system/sshd.service;
enabled)
  Active: active (running) since Mon, 20 Apr 2020
12:35:20...
```

   b. If you don't see positive results for the preceding test, try the following command for the SysVinit `init` daemon. A positive result here, along with

negative results for the preceding tests, means that `sshd` is still using the `SysVinit` daemon.

```
# service ssh status
sshd (pid 2390) is running...
```

3. To determine your server' previous and current runlevel, use the `runlevel` command. It still works on all `init` daemons:

```
$ runlevel
N 3
```

4. To change the default runlevel or target unit on your Linux server, you can do one of the following (depending upon your server's `init` daemon):
   a. For SysVinit, edit the file `/etc/inittab` and change the # in the line `id:#:initdefault:` to `2`, `3`, `4`, or `5`.
   b. For `systemd`, change the `default.target` to the desired `runlevel#.target`, where # is `2`, `3`, `4`, or `5`. The following shows you how to change the target unit to `runlevel3.target`.

```
# systemctl set-default runlevel3.target
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target →
   /usr/lib/systemd/system/multi-user.target.
```

5. To list out services running (or active) on your server, you need to use different commands, depending upon the initialization daemon you are using.
   a. For SysVinit, use the `service` command as shown in this example:

```
# service --status-all | grep running | sort
anacron (pid 2162) is running...
atd (pid 2172) is running...
```

   b. For `systemd`, use the `systemctl` command, as follows:

```
# systemctl list-unit-files --type=service | grep -v disabled
UNIT FILE                                   STATE
abrt-ccpp.service                           enabled
abrt-oops.service                           enabled
...
```

6. To list out the running (or active) services on your Linux server, use the appropriate command(s) determined in answer 5 for the initialization daemon that your server is using.
7. For each initialization daemon, the following command(s) show a particular service's current status:
   a. For SysVinit, the `service` *`service_name`* `status` command is used.
   b. For `systemd`, the `systemctl status` *`service_name`* command is used.

8. To show the status of the cups daemon on your Linux server, use the following:

   a. For the SysVinit:

   ```
   # service cups status
   cupsd (pid 8236) is running...
   ```

   b. For systemd:

   ```
   # systemctl status cups.service
   cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; enabled)
   Active: active (running) since Tue, 05 May 2020 04:43:5...
   Main PID: 17003 (cupsd)
   CGroup: name=systemd:/system/cups.service
   17003 /usr/sbin/cupsd -f
   ```

9. To attempt to restart the cups daemon on your Linux server, use the following:

   a. For SysVinit:

   ```
   # service cups restart
   Stopping cups:          [  OK  ]
   ```

   b. For systemd:

   ```
   # systemctl restart cups.service
   ```

10. To attempt to reload the cups daemon on your Linux server, use the following:

    a. For SysVinit:

    ```
    # service cups reload
    Reloading cups: [ OK ]
    ```

    b. For systemd, this is a trick question. You cannot reload the cups daemon on a systemd Linux server!

    ```
    # systemctl reload cups.service
    Failed to issue method call: Job type reload is
      not applicable for unit cups.service.
    ```

Chapter 16: Configuring a Print Server

For questions that involve working with printers, you can use either graphical or command-line tools in most cases. The point is to make sure that you get the correct results, shown in the answers that follow. The answers here include a mix of graphical and command-line ways of solving the exercises. (Become root user when you see a # prompt.)

1. To use the Print Settings window to add a new printer called myprinter to your system (generic PostScript printer, connected to a port), do the following from Fedora 30:

   a. Install the system-config-printer package:

   ```
   # dnf install system-config-printer
   ```

b. From the GNOME 3 desktop, select Print Settings from the Activities screen.

c. Unlock the interface and enter the root password.

d. Select the Add button.

e. Select a USB or other port as the device and click Forward.

f. For the driver, choose Generic and click Forward; then choose PostScript and click Forward.

g. Click Forward to skip any installable options, if needed.

h. For the printer name, call it `myprinter`, give it any description and location you like, and click Apply.

i. Click Cancel in order not to print a test page. The printer should appear in the Print Settings window.

2. To use the `lpstat -t` command to see the status of all of your printers, enter the following:

```
# lpstat -t
deskjet-5550 accepting requests since Mon 02 Mar 2020
07:30:03 PM EST
```

3. To use the `lpr` command to print the `/etc/hosts` file, enter the following:

```
$ lp /etc/hosts -P myprinter
```

4. To check the print queue for that printer, enter the following:

```
# lpq -P myprinter
myprinter is not ready
Rank    Owner   Job     File(s)             Total Size
1st     root    655     hosts               1024 bytes
```

5. To remove the print job from the queue (cancel it), enter the following.

```
# lprm -P myprinter
```

6. To use the printing window to set the basic server setting that publishes your printers so that other systems on your local network can print to your printers, do the following:

a. On a GNOME 3 desktop, from the Activities screen, type **Print Settings** and press Enter.

b. Select Server ➪ Settings and type the root password if prompted.

c. Click the check box next to "Publish shared printers connected to this system" and click OK.

7. To allow remote administration of your system from a web browser, follow these steps:
   a. On a GNOME 3 desktop, from the Activities screen, type **Print Settings**, and press Enter.
   b. Select Server ➪ Settings and type the root password if prompted.
   c. Click the check box next to "Allow remote administration" and click OK.
8. To demonstrate that you can do remote administration of your system from a web browser on another system, do the following:
   a. In the location box from a browser window from another computer on your network, enter the following, replacing ***hostname*** with the name or IP address of the system running your print service: **http://*hostname*:631**.
   b. Type **root** as the user and the root password, when prompted. The CUPS home page should appear from that system.
9. To use the `netstat` command to see on which addresses the `cupsd` daemon is listening, enter the following:

```
# netstat -tupln | grep 631
tcp    0    0 0.0.0.0:631      0.0.0.0:*      LISTEN    6492/cupsd
tcp6   0    0 :::631           :::*           LISTEN    6492/cupsd
```

10. To delete the `myprinter` printer entry from your system, do the following:
    a. Click the Unlock button and type the root password when prompted.
    b. From the Print Settings window, right-click the `myprinter` icon and select Delete.
    c. When prompted, select Delete again.

Chapter 17: Configuring a Web Server

1. To install all of the packages associated with the Web Server group on a Fedora system, do the following:

```
# yum groupinstall "Web Server"
```

2. To create a file called `index.html` in the directory assigned to `DocumentRoot` in the main Apache configuration file (with the words "My Own Web Server" inside), do the following:
   a. Determine the location of `DocumentRoot`:

```
# grep ^DocumentRoot /etc/httpd/conf/httpd.conf

DocumentRoot "/var/www/html"
```

b. Echo the words "My Own Web Server" into the `index.html` file located in `DocumentRoot`:

```
# echo "My Own Web Server" > /var/www/html/index.html
```

3. To start the Apache web server and set it to start up automatically at boot time, then check that it is available from a web browser on your local host, do the following. (You should see the words "My Own Web Server" displayed if it is working properly.)

   The `httpd` service is started and enabled differently on different Linux systems. In recent Fedora 30 or RHEL 7 or 8, enter the following:

```
# systemctl start httpd.service
# systemctl enable httpd.service
```

   In RHEL 6 or earlier, enter the following:

```
# service httpd start
# chkconfig httpd on
```

4. To use the `netstat` command to see on which ports the `httpd` server is listening, enter the following:

```
# netstat -tupln | grep httpd
tcp6       0      0 :::80                   :::*                    LISTEN      2496/httpd
tcp6       0      0 :::443                  :::*                    LISTEN      2496/httpd
```

5. Try to connect to your Apache web server from a web browser that is outside of the local system. If it fails, correct any problems that you encounter by investigating the firewall, SELinux, and other security features.

   If you don't have DNS set up yet, use the IP address of the server to view your Apache server from a remote web browser, such as `http://192.168.0.1`. If you are not able to connect, retry connecting to the server from your browser after performing each of the following steps on the system running the Apache server:

```
# iptables -F
# setenforce 0
# chmod 644 /var/www/html/index.html
```

   The `iptables -F` command flushes the firewall rules temporarily. If connecting to the web server succeeds after that, you need to add new firewall rules to open `tcp` ports 80 and 443 on the server. On a system using the `firewalld` service, do this by clicking the check box next to those ports on the Firewall window. For systems running the `iptables` service, add the following rules before the last `DROP` or `REJECT` rule.

```
        -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j
ACCEPT
        -A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j
ACCEPT
```

The `setenforce 0` command puts SELinux in permissive mode temporarily. If connecting to the web server succeeds after that, you need to correct SELinux file context and/or Boolean issues (probably file context in this case). The following should work:

```
# chcon --reference=/var/www/html /var/www/html/index.html
```

If the `chmod` command works, it means that the Apache user and group did not have read permission to the file. You should be able to leave the new permissions as they are.

6. To use the `openssl` or similar command to create your own private RSA key and self-signed SSL certificate, do the following:

```
# yum install openssl
# cd /etc/pki/tls/private
# openssl genrsa -out server.key 1024
# chmod 600 server.key
# cd /etc/pki/tls/certs
# openssl req -new -x509 -nodes -sha1 -days 365 \
   -key /etc/pki/tls/private/server.key \
   -out server.crt
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]: NJ
Locality Name (eg, city) []: Princeton
Organization Name (eg, company) [Internet Widgits Pty
Ltd]:TEST USE ONLY
Organizational Unit Name (eg, section) []:TEST USE ONLY
Common Name (eg, YOUR name) []:secure.example.org
Email Address []:dom@example.org
```

You should now have a `/etc/pki/tls/private/server.key` key file and a `/etc/pki/tls/certs/server.crt` certificate file.

7. To configure your Apache web server to use your key and self-signed certificate to serve secure (HTTPS) content, do the following:

   a. Edit the `/etc/httpd/conf.d/ssl.conf` file to change the key and certificate locations to use the ones that you just created:

```
SSLCertificateFile /etc/pki/tls/certs/server.crt
SSLCertificateKeyFile /etc/pki/tls/private/server.key
```

   b. Restart the httpd service:

```
# systemctl restart httpd.service
```

8. To use a web browser to create an HTTPS connection to your web server and view the contents of the certificate that you created, do the following:

From the system running the Apache server, type **https://localhost** in the browser's location box. You should see a message that reads, "This Connection is Untrusted." To complete the connection, do the following:

a. Click I Understand the Risks.

b. Click Add Exception.

c. Click Get Certificate.

d. Click Confirm Security Exception.

9. To create a file named /etc/httpd/conf.d/example.org.conf, which turns on name-based virtual hosting and creates a virtual host that (1) listens on port 80 on all interfaces, (2) has a server administrator of joe@example.org, (3) has a server name of joe.example.org, (4) has a DocumentRoot of /var/www/html/joe.example.org, and (5) has a DirectoryIndex that includes at least index.html and then create an index.html file in DocumentRoot that contains the words "Welcome to the House of Joe" inside, do the following.

   Create an example.org.conf file that looks like the following:

```
NameVirtualHost *:80
<VirtualHost *:80>
    ServerAdmin     joe@
example.org
    ServerName      joe.
example.org
    ServerAlias     web.example.org
    DocumentRoot    /var/www/html/joe.example.org/
    DirectoryIndex  index.html
</VirtualHost>
```

   This is how you could create the text to go into the index.html file:

```
# echo "Welcome to the House of Joe" > \
     /var/www/html/joe.example.org/index.html
```

10. To add the text joe.example.org to the end of the localhost entry in your /etc/hosts file on the machine that is running the web server, and check it by typing **http://joe.example.org** into the location box of your web browser to see "Welcome to the House of Joe" when the page is displayed, do the following:

    a. Reload the httpd.conf file modified in the previous exercise in one of two ways:

```
# apachectl graceful
# systemctl restart httpd
```

b. Edit the /etc/hosts file with any text editor, so the local host line appears as follows:

```
127.0.0.1       localhost.localdomain localhost joe.
example.org
```

c. From a browser on the local system where httpd is running, you should be able to type **http://joe.example.org** into the location box to access the Apache web server using name-based authentication.

Chapter 18: Configuring an FTP Server

CAUTION

Don't do the tasks described here on a working, public FTP server, because these tasks will interfere with its operations. (You could, however, use these tasks to set up a new FTP server.)

1. To determine which package provides the Very Secure FTP Daemon service, enter the following as root:

```
# yum search "Very Secure FTP"
...
================ N/S Matched: Very Secure FTP ============
vsftpd.i686 : Very Secure Ftp Daemon
```

   The search found the vsftpd package.

2. To install the Very Secure FTP Daemon package on your system and search for the configuration files in the vsftpd package, enter the following:

```
# yum install vsftpd
# rpm -qc vsftpd | less
```

3. To enable anonymous FTP and disable local user login for the Very Secure FTP Daemon service, set the following in the /etc/vsftpd/vsftpd.conf file:

```
anonymous_enable=YES
write_enable=YES
anon_upload_enable=YES
local_enable=NO
```

4. To start the Very Secure FTP Daemon service and set it to start when the system boots, enter the following on a current Fedora or Red Hat Enterprise Linux system:

```
# systemctl start vsftpd.service
# systemctl enable vsftpd.service
```

On a Red Hat Enterprise Linux 6 system, enter the following:

```
# service vsftpd start
# chkconfig vsftpd on
```

5. On the system running your FTP server, enter the following to create a file named `test` in the anonymous FTP directory that contains the words "Welcome to your vsftpd server":

```
# echo "Welcome to your vsftpd server" > /var/ftp/test
```

6. To open the `test` file from the anonymous FTP home directory using a web browser on the system running your FTP server, do the following.

   Open a web browser, enter the following in the location box, and press Enter:

```
ftp://localhost/test
```

   The text "Welcome to your vsftpd server" should appear in the browser window.

7. To access the `test` file in the anonymous FTP home directory, do the following. (If you cannot access the file, check that your firewall, SELinux, and TCP wrappers are configured to allow access to that file, as described here.)

   a. Enter the following into the location box of a browser on a system on your network that can reach the FTP server (replace *host* with your system's fully qualified hostname or IP address):

```
ftp://host/test
```

   If you cannot see the welcome message in your browser window, check what may be preventing access. To turn off your firewall temporarily (flush your `iptables` rules), enter the following command as the root user from a shell on your FTP server system and then try to access the site again:

```
# iptables -F
```

   b. To disable SELinux temporarily, enter the following and then try to access the site again:

```
# setenforce 0
```

   After you have determined what is causing the file on your FTP server to be unavailable, go back to the section "Securing Your FTP Server" in Chapter 18, and go through the steps to determine what might be blocking access to your file. These are the likely possibilities:

   c. For `iptables`, make sure that there is a rule opening TCP port 21 on the server.

   d. For SELinux, make sure that the file context is set to `public_content_t`.

8. To configure your vsftpd server to allow file uploads by anonymous users to a directory named `in`, do the following as root on your FTP server:

   a. Create the `in` directory as follows:

   ```
   # mkdir /var/ftp/in
   # chown ftp:ftp /var/ftp/in
   # chmod 777 /var/ftp/in
   ```

   b. For a recent Fedora or RHEL, open the Firewall Configuration window and check the FTP box under services to open access to your FTP service. For earlier RHEL and Fedora systems, configure your `iptables` firewall to allow new requests on TCP port 21 by adding the following rule at some point before a final `DROP` or `REJECT` rule in your `/etc/sysconfig/iptables` file:

   ```
   -A INPUT -m state --state NEW -m tcp -p tcp --dport 21
   -j ACCEPT
   ```

   c. Configure your `iptables` firewall to do connection tracking by loading the appropriate module to the `/etc/sysconfig/iptables-config` file:

   ```
   IPTABLES_MODULES="nf_conntrack_ftp"
   ```

   d. For SELinux to allow uploading to the directory, first set file contexts properly:

   ```
   # semanage fcontext -a -t public_content_rw_t "/var/ftp/
   in(/.*)?"
   # restorecon -F -R -v /var/ftp/in
   ```

   e. Next, set the SELinux Boolean to allow uploading:

   ```
   # setsebool -P allow_ftpd_anon_write on
   ```

   f. Restart the `vsftpd` service (`service vsftpd restart` or `systemctl restart vsftpd.service`).

9. To install the `lftp` FTP client (if you don't have a second Linux system, install `lftp` on the same host running the FTP server). Optionally, try to upload the `/etc/hosts` file to the `in` directory on the server, to make sure it is accessible. Run the following commands as the root user:

   ```
   # yum install lftp
   # lftp localhost
   lftp localhost:/> cd in
   lftp localhost:/in> put /etc/hosts
   89 bytes transferred
   lftp localhost:/in> quit
   ```

   You won't be able to see that you copied the `hosts` file to the incoming directory. However, enter the following from a shell on the host running the FTP server to make sure that the `hosts` file is there:

   ```
   # ls /var/ftp/in/hosts
   ```

If you cannot upload the file, troubleshoot the problem as described in Exercise 7, recheck your `vsftpd.conf` settings, and review the ownership and permissions on the `/var/ftp/in` directory.

10. Using any FTP client you choose, visit the `/pub/debian-meetings` directory on the `ftp://ftp.gnome.org` site and list the contents of that directory. Here's how to do that with the `lftp` client:

```
# lftp ftp://ftp.gnome.org/pub/debian-meetings/
cd ok, cwd=/pub/debian-meetings
lftp ftp.gnome.org:/pub/debian-meetings>> ls
drwxr-xr-x    3 ftp      ftp             3 Jan 13  2014 2004
drwxr-xr-x    6 ftp      ftp             6 Jan 13  2014 2005
drwxr-xr-x    8 ftp      ftp             8 Dec 20  2006 2006
...
```

Chapter 19: Configuring a Windows File Sharing (Samba) Server

1. To install the `samba` and `samba-client` packages, enter the following as root from a shell on the local system:

```
# yum install samba samba-client
```

2. To start and enable the `smb` and `nmb` services, enter the following as root from a shell on the local system:

```
# systemctl enable smb.service
# systemctl start smb.service
# systemctl enable nmb.service
# systemctl start nmb.service
```

or

```
# chkconfig smb on
# service smb start
# chkconfig nmb on
# service nmb start
```

3. To set the Samba server's workgroup to `TESTGROUP`, the NetBIOS name to `MYTEST`, and the server string to `Samba Test System`, as root user in a text editor, open the `/etc/samba/smb.conf` file, and change three lines so that they appear as follows:

```
workgroup = TESTGROUP
netbios name = MYTEST
server string = Samba Test System
```

4. To add a Linux user named `phil` to your system, and add a Linux password and Samba password for `phil`, enter the following as root user from a shell. (Be sure to remember the passwords you set.)

```
# useradd phil
# passwd phil
New password: *******
Retype new password: *******
# smbpasswd -a phil
New SMB password: *******
Retype new SMB password: *******
Added user phil.
```

5. To set the `[homes]` section so that home directories are browseable (`yes`) and writeable (`yes`), and that `phil` is the only valid user, open the `/etc/samba/smb.conf` file as root, and change the `[homes]` section so that it appears as follows:

```
[homes]
        comment = Home Directories
        browseable = Yes
        read only = No
        valid users = phil
```

6. To set SELinux Booleans that are necessary to make it so that `phil` can access his home directory via a Samba client, enter the following as root from a shell, and restart the `smb` and `nmb` services:

```
# setsebool -P samba_enable_home_dirs on
# systemctl restart smb
# systemctl restart nmb
```

7. From the local system, use the `smbclient` command to list that the `homes` share is available.

```
# smbclient -L localhost
Enter TESTGROUP\root's password: <ENTER>
Anonymous login successful

     Sharename       Type      Comment
     ---------       ----      -------
     homes           Disk      Home Directories
  ...
```

8. To connect to the `homes` share from a Nautilus (file manager) window on the Samba server's local system for the user `phil` in a way that allows you to drag and drop files to that folder, do the following:
   a. Open the Nautilus window (select the files icon).
   b. In the left pane, select Other Locations and then click in the Connect to Server box.
   c. Type the Server address. For example, `smb://localhost/phil/`.

d. When prompted, select Registered User, type **phil** as the username, enter the domain (TESTGROUP), and enter phil's password.

e. Open another Nautilus window and drop a file to phil's homes folder.

9. To open up the firewall so that anyone who has access to the server can access the Samba service (smbd and nmbd daemons), you can simply open the Firewall Configuration window and check the samba and samba-client check boxes (for both Runtime and Permanent). If your system is running basic iptables (and not the firewalld service), change the /etc/sysconfig/iptables file so that the firewall appears like the following (the rules you add being those in bold):

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-I INPUT -m state --state NEW -m udp -p udp --dport 137 -j ACCEPT
-I INPUT -m state --state NEW -m udp -p udp --dport 138 -j ACCEPT
-I INPUT -m state --state NEW -m tcp -p tcp --dport 139 -j ACCEPT
-I INPUT -m state --state NEW -m tcp -p tcp --dport 445 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Then enter the following for the firewall rules to be reloaded:

```
# service iptables restart
```

10. To open the homes share again as the user phil from another system on your network (Windows or Linux), and make sure that you can drag and drop fles to it, do the following:

a. This step is really just repeating the Nautilus example described previously or accessing a Windows File Explorer window and opening the share (by selecting Network, then the Samba server). The trick is to make sure that the service has been made available through the Linux server security features.

b. If you cannot access the Samba share, try disabling your firewall and then disabling SELinux. If the share is accessible when you turn off either of those services, go back and debug the problems with the service that is not working:

```
# setenforce 0
# service iptables stop
```

c. When you have fixed the problem, set SELinux back to Enforcing mode and restart `iptables`:

```
# setenforce 1
# service iptables start
```

Chapter 20: Configuring an NFS File Server

1. To install the packages needed to configure the NFS service on your chosen Linux system, enter the following as root user at a shell (Fedora or RHEL):

```
# yum install nfs-utils
```

2. To list the documentation files that come in the package that provides the NFS server software, enter the following:

```
# rpm -qd nfs-utils
/usr/share/doc/nfs-utils-1.2.5/ChangeLog
...
/usr/share/man/man5/exports.5.gz
/usr/share/man/man5/nfs.5.gz
/usr/share/man/man5/nfsmount.conf.5.gz
/usr/share/man/man7/nfsd.7.gz
/usr/share/man/man8/blkmapd.8.gz
/usr/share/man/man8/exportfs.8.gz
...
```

3. To start and enable the NFS service, enter the following as root user on the NFS server:

```
# systemctl start nfs-server.service
# systemctl enable nfs-server.service
```

4. To check the status of the NFS service that you just started on the NFS server, enter the following as root user:

```
# systemctl status nfs-server.service
```

5. To share a directory `/var/mystuff` from your NFS server as available to everyone, read-only, and with the root user on the client having root access to the share, first create the mount directory as follows:

```
# mkdir /var/mystuff
```

Then create an entry in the `/etc/exports` file that is similar to the following:

```
/var/mystuff    *(ro,no_root_squash,insecure)
```

To make the share available, enter the following:

```
# exportfs -v -a
exporting *:/var/mystuff
```

6. To make sure that the share you created is accessible to all hosts, first check that `rpcbind` is not blocked by TCP wrappers by adding the following entry to the beginning of the `/etc/hosts.allow` file:

```
rpcbind: ALL
```

   a. To open the firewall in systems that use `firewalld` (RHEL 8 and recent Fedora systems), install the firewall-config package. Then run `firewall-config`. From the Firewall Configuration window that appears, make sure that nfs and rpc-bind are checked to On for the Permanent firewall settings.

   b. To open the ports needed to allow clients to reach NFS through the `iptables` firewall (RHEL 6 and earlier Fedora systems without `firewalld`), you need to open at least TCP and UDP ports 111 (`rpcbind`), 20048 (`mountd`), and 2049 (`nfs`) by adding the following rules to the `/etc/sysconfig/iptables` file and starting the `iptables` service:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 111 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 2049 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 2049 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 20048 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 20048 -j ACCEPT
```

   SELinux should be able to share NFS filesystems while in enforcing mode without any changes to file contexts or Booleans. To make sure that the share you created can be shared read-only, run the following command as root user on the NFS server:

```
# setsebool -P nfs_export_all_ro on
```

7. To view the shares available from the NFS server, assuming that the NFS server is named `nfsserver`, enter the following from the NFS client:

```
# showmount -e nfsserver
Export list for nfsserver:
/var/mystuff  *
```

8. To create a directory called `/var/remote` and temporarily mount the `/var/mystuff` directory from the NFS server (named `nfsserver` in this example) on that mount point, enter the following as root user from the NFS client:

```
# mkdir /var/remote
# mount -t nfs nfsserver:/var/mystuff /var/remote
```

9. To add an entry so that the same mount is done automatically when you reboot, first unmount `/var/remote` as follows:

```
# umount /var/remote
```

Then add an entry like the following to the /etc/fstab on the client system:

```
/var/remote   nfsserver:/var/mystuff  nfs bg,ro 0 0
```

To test that the share is configured properly, enter the following on the NFS client as the root user:

```
# mount -a
# mount -t nfs4
nfsserver:/var/mystuff on /var/remote type nfs4
 (ro,vers=4,rsize=524288...
```

10. To copy some files to the /var/mystuff directory, enter the following on the NFS server:

```
# cp /etc/hosts /etc/services /var/mystuff
```

From the NFS client, to make sure that you can see the files just added to that directory, and to make sure that you can't write files to that directory from the client, enter the following:

```
# ls /var/remote
hosts    services
# touch /var/remote/file1
touch: cannot touch '/var/remote/file1': Read-only file
system
```

Chapter 21: Troubleshooting Linux

1. To go into Setup mode from the BIOS screen on your computer, do the following:
   a. Reboot your computer.
   b. Within a few seconds, you should see the BIOS screen, with an indication of which function key to press to go into Setup mode. (On my Dell workstation, it's the F2 function key.)
   c. The BIOS screen should appear. (If the system starts booting Linux, you didn't press the function key fast enough.)
2. From the BIOS setup screen, do the following to determine whether your computer is 32-bit or 64-bit, whether it includes virtualization support, and whether your network interface card is capable of PXE booting.

   Your experience may be a bit different from mine, depending on your computer and Linux system. The BIOS setup screen is different for different computers. In general, however, you can use arrow keys and tab keys to move between different columns, and press Enter to select an entry.

   a. On my Dell workstation, under the System heading, I highlight Processor Info to see that mine is a 64-bit Technology computer. Look in the Processor Info section, or a similar, section on your computer, to see the type of processor that you have.
   b. On my Dell workstation, under the Onboard Devices heading, I highlight Integrated NIC and press Enter. The Integrated NIC screen that appears to the right lets me choose to enable or disable the NIC (On or Off) or enable with PXE or RPL (if I intend to boot the computer over the network).

3. To interrupt the boot process to get to the GRUB boot loader, do the following:
   a. Reboot the computer.
   b. Just after the BIOS screen disappears, when you see the countdown to booting the Linux system, press any key (perhaps the spacebar).
   c. The GRUB boot loader menu should appear, ready to allow you to select which operating system kernel to boot.
4. To boot up your computer to runlevel 1 so that you can do some system maintenance, get to the GRUB boot screen (as described in the previous exercise), and then do the following:
   a. Use the arrow keys to highlight the operating system and kernel that you want to boot.
   b. Type **e** to see the entries needed to boot the operating system.
   c. Move your cursor to the line that included the kernel. (It should include the word `vmlinuz` somewhere on the line.)
   d. Move the cursor to the end of that line, add a space, and then type **init=bash**.
   e. Follow the instructions to boot the new entry. You will probably either press Ctrl+X or press Enter; if there is another screen, type **b**.

      If it worked, your system should bypass the login prompt and boot up directly to a root user shell where you can do administrative tasks without providing a password.
5. To look at the messages that were produced in the kernel ring buffer (which shows the activity of the kernel as it booted up), enter the following from the shell after the system finishes booting:

```
# dmesg | less
```

6. Or, on a system using `systemd`, enter the following:

```
# journalctl -k
```

7. To run a trial `yum update` from Fedora or RHEL and exclude any kernel package that is available, enter the following (when prompted, type **N** to not actually go through with the update, if updates are available):

```
# yum update --exclude='kernel*'
```

8. To check to see what processes are listening for incoming connections on your system, enter the following:

```
# netstat -tupln | less
```

9. To check to see what ports are open on your external network interface, do the following.

   If possible, run the `nmap` command from another Linux system on your network, replacing *yourhost* with the hostname or IP address of your system:

```
# nmap yourhost
```

10. To clear your system's page cache and watch the effect it has on your memory usage, do the following:

    a. Select Terminal from an application menu on your desktop (it is located on different menus for different systems).

    b. Run the `top` command (to watch processes currently running on your system), and then type a capital **M** to sort processes by those consuming the most memory.

    c. From the Terminal window, select File and Open Terminal to open a second Terminal window.

    d. From the second Terminal window, become root user (`su -`).

    e. While watching the `Mem` line (used column) in the first Terminal window, enter the following from the second Terminal window:

```
# echo 3 > /proc/sys/vm/drop_caches
```

    f. The used `RES` memory should go down significantly on the `Mem` line. The numbers in the `RES` column for each process should go down as well.

11. To view memory and swap usage from Cockpit through your web browser, open your browser to Cockpit for your host (`https://hostname:9090`). Then select System ➪ Memory & Swap.

Chapter 22: Understanding Basic Linux Security

1. To check log messages from the `systemd` journal for the `NetworkManager.service`, `sshd.service`, and `auditd.service` services, enter the following:

```
# journalctl -u NetworkManager.service
...
```

```
# journalctl -u sshd.service
...
# journalctl -u auditd.service
...
```

2. User passwords are stored in the `/etc/shadow` file. To see its permissions, type **`ls -l /etc/shadow`** at the command line. (If no shadow file exits, then you need to run `pwconv`.)

   The following are the appropriate settings:

```
 # ls -l /etc/shadow
----------. 1 root root 1049 Feb  10 09:45 /etc/shadow
```

3. To determine your account's password aging and whether it will expire using a single command, type `chage -l` *`user_name`*. For example:

```
# chage -l chris
```

4. To start auditing writes to the `/etc/shadow` with the `auditd` daemon, enter the following at the command line:

```
# auditctl -w /etc/shadow -p w
```

   To check your audit settings, type in `auditctl -l` at the command line.

5. To create a report from the `auditd` daemon on the `/etc/shadow` file, enter `ausearch -f /etc/shadow` at the command line. To turn off the auditing on that file, enter `auditctl -W /etc/shadow -p w` at the command line.

6. To install the lemon package, damage the `/usr/bin/lemon` file, verify that the file has been tampered with, and remove the lemon package, enter the following:

```
# yum install -y lemon
# cp /etc/services /usr/bin/lemon
# rpm -V lemon
S.5....T.    /usr/bin/lemon
# yum erase lemon
```

   From the original lemon file, the file size (S), the md4sum (5), and the modification times (T) all differ. For Ubuntu, install the package with `apt-get install lemon` and enter `debsums lemon` to check it.

7. If you suspect that you have had a malicious attack on your system today and important binary files have been modified, you can find these modified files by entering the following at the command line: `find directory -mtime -1` for the directories, `/bin`, `/sbin`, `/usr/bin`, and `/usr/sbin`.

8. To install and run `chkrootkit` to see if the malicious attack from the exercise above installed a rootkit, choose your distribution and do the following:

   a. To install on a Fedora or RHEL distribution, enter `yum install chkrootkit` at the command line.

b. To install on an Ubuntu or Debian-based distribution, enter `sudo apt-get install chkrootkit` at the command line.

c. To run the check, enter `chkrootkit` at the command line and review the results.

9. To find files anywhere in the system with the `SUID` or `SGID` permission set, enter `find / -perm /6000 -ls` at the command line.

10. To install the `aide` package, run the `aide` command to initialize the `aide` database, copy the database to the correct location, and run the `aide` command to check whether any important files on your system have been modified, enter the following.

```
# yum install aide
# aide -i
# cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
# aide -C
```

To make the output more interesting, you could install the lemon package (described in an earlier exercise) before you run `aide -i`, and modify it before running `aide -C` to see how a modified binary looks from `aide`.

Chapter 23: Understanding Advanced Linux Security

To do the first few exercises, you must have the `gnupg2` package installed. This is not installed by default in Ubuntu, although it is installed for the latest Fedora and RHEL releases.

1. To encrypt a file using the `gpg2` utility and a symmetric key, enter the following command. (The `gpg2` utility asks for a passphrase to protect the symmetric key.)

```
$ gpg2 -c filename
```

2. To generate a key pair using the `gpg2` utility, enter the following:

```
$ gpg2 --gen-key
```

You must provide the following information:

a. Your real name and email address

b. A passphrase for the private key

3. To list out the keys you generated, enter the following:

```
$ gpg2 --list-keys
```

4. To encrypt a file and add your digital signature using the `gpg2` utility, do the following:

a. You must have first generated a key ring (Exercise 2).

b. After you have generated the key ring, enter

```
$ gpg2 --output EncryptedSignedFile --sign
FiletoEncryptSign
```

5. From the getfedora.org page, select one of the Fedora distributions to download. When the download is complete, select Verify your Download to see instructions for verifying your image. For example, download the appropriate CHECKSUM file for your image, then enter the following:

```
$ curl https://getfedora.org/static/fedora.gpg | gpg
--import
$ gpg --verify-files *-CHECKSUM
$ sha256sum -c *-CHECKSUM
```

6. To determine if the su command on your Linux system is PAM-aware, enter the following:

```
$ ldd $(which su) | grep pam
libpam.so.0 => /lib64/libpam.so.0 (0x00007fca14370000)
ibpam_misc.so.0 => /lib64/libpam_misc.so.0
(0x00007fca1416c000
```

   If the su command on your Linux system is PAM-aware, you should see a PAM library name listed when you issue the ldd command.

7. To determine if the su command has a PAM configuration file, type the following:

```
$ ls /etc/pam.d/su
/etc/pam.d/su
```

   If the file exists, type the following at the command line to display its contents. The PAM contexts it uses include any of the following: auth, account, password, or session.

```
$ cat /etc/pam.d/su
```

8. To list out the various PAM modules on your Fedora or RHEL system, enter the following:

```
$ ls /usr/lib64/security/pam*.so
```

   To list out the various PAM modules on your Ubuntu Linux system, enter the following:

```
# find / -name pam*.so
```

9. To find the PAM "other" configuration file on your system, enter ls /etc/pam.d/other at the command line. An "other" configuration file that enforces Implicit Deny should look similar to the following code:

```
$ cat /etc/pam.d/other
#%PAM-1.0
auth     required       pam_deny.so
account  required       pam_deny.so
password required       pam_deny.so
session  required       pam_deny.so
```

10. To find the PAM limits configuration file, enter the following:

```
$ ls /etc/security/limits.conf
```

Display the file's contents by entering the following:

```
$ cat /etc/security/limits.conf
```

Settings in this file to prevent a fork bomb look like the following:

```
@student     hard    nproc           50
@student     -       maxlogins        4
```

Chapter 24: Enhancing Linux Security with SELinux

1. To set your system into the permissive mode for SELinux, enter `setenforce permissive` at the command line. It would also be acceptable to enter `setenforce 0` at the command line.
2. To set your system into the enforcing operating mode for SELinux without changing the SELinux primary configuration file, use caution. It is best not to run this command on your system for an exercise until you are ready for the SELinux to be enforced. Use the following command at the command line: `setenforce enforcing`. It would also be acceptable to enter `setenforce 1` at the command line.
3. To find and view the permanent SELinux policy type (set at boot time), go to the main SELinux configuration file, `/etc/selinux/config`. To view it, enter `cat /etc/selinux/config | grep SELINUX=` at the command line. To be sure how it is currently set, enter the `getenforce` command.
4. To list the `/etc/hosts` file security context and identify the different security context attributes, enter `ls -Z /etc/hosts` at the command line:

```
$ ls -Z /etc/hosts
-rw-r--r--. root root system_u:object_r:net_conf_t:s0  /
etc/hosts
```

   a. The file's user context is `system_u`, indicating a system file.
   b. The file's role is `object_r`, indicating an object in the file system (a text file, in this case).
   c. The file's type is `net_conf_t`, because the file is a network configuration file.
   d. The file's sensitivity level is `s0`, indicating the lowest security level. (This number may be listed in a range of numbers from `s0-s3`.)
   e. The file's category level starts with a `c` and ends with a number. It may be listed in a range of numbers, such as `c0-c102`. This is not required except in highly secure environments and is not set here.

5. To create a file called `test.html` and assign its type as `httpd_sys_content_t`, enter the following:

```
$ touch test.html
$ chcon -t httpd_sys_content_t test.html
$ ls -Z test.html
-rw-rw-r--. chris chris unconfined_u:object_r:httpd_sys_
content_t:s0 test.html
```

6. To list the `crond` process's security context and identify the different security context attributes, enter this at the command line:

```
$ ps -efZ | grep crond
system_u:system_r:crond_t:s0-s0:c0.c1023 root 665  1  0
    Sep18 ?   00:00:00 /usr/sbin/crond -n
```

   a. The process's user context is `system_u`, indicating a system process.

   b. The process's role is `system_r`, indicating a system role.

   c. The process's type or domain is `crond_t`.

   d. The process's sensitivity level starts `s0-s0`, indicating that it is not highly sensitive. (It is secure by normal Linux standards, however, because the process is run as the root user.)

   e. The process's category level is `c0.c1023`, with the `c0`, indicating that the category is also not highly secure from an SELinux standpoint.

7. To create an `/etc/test.txt` file, change its file context to `user_tmp_t`, restore it to its proper content (the default context for the `/etc` directory), and remove the file, enter the following:

```
# touch /etc/test.txt
# ls -Z /etc/test.txt
-rw-r--r--. root root unconfined_u:object_r:etc_t:s0   /etc/
test.txt
# chcon -t user_tmp_t /etc/test.txt
# ls -Z /etc/test.txt
-rw-r--r--. root root unconfined_u:object_r:user_tmp_t:s0 /
etc/test.txt
# restorecon /etc/test.txt
# ls -Z /etc/test.txt
-rw-r--r--. root root unconfined_u:object_r:etc_t:s0   /etc/
test.txt
# rm /etc/test.txt
rm: remove regular empty file `/etc/test.txt'? y
```

8. To determine what Booleans allow anonymous writes and access to the `tftp` service's home directory, then turn those Booleans on permanently, enter the following commands:

```
# getsebool -a | grep tftp
tftp_home_dir --> off
tftpd_anon_write --> off
```

```
...
# setsebool -P tftp_home_dir=on
# setsebool -P tftp_anon_write=on
# getsebool tftp_home_dir tftp_anon_write
tftp_home_dir --> on
tftp_anon_write --> on
```

9. To list all SELinux policy modules on your system, along with their version numbers, enter **`semodule -l`**.

> **Note**
> If you wrote `ls /etc/selinux/targeted/modules/active/modules/*.pp` as your answer, that is okay, but this command doesn't give you the version numbers of the policy modules. Only `semodule -l` gives the version numbers.

10. To tell SELinux to allow access to the `sshd` service through TCP Port 54903, enter the following:

```
# semanage port -a -t ssh_port_t -p tcp 54903
# semanage port -l | grep ssh
ssh_port_t                tcp             54903, 22
```

Chapter 25: Securing Linux on a Network

1. To install the Network Mapper (aka `nmap`) utility on your local Linux system:
   a. On Fedora or RHEL, enter `yum install nmap` at the command line.
   b. On Ubuntu, `nmap` may come pre-installed. If not, enter `sudo apt-get install nmap` at the command line.
2. To run a TCP Connect scan on your local loopback address, enter `nmap -sT 127.0.0.1` at the command line. The ports you have running on your Linux server will vary. However, they may look similar to the following:

```
# nmap -sT 127.0.0.1
...
PORT    STATE SERVICE
25/tcp  open  smtp
631/tcp open  ipp
```

3. To run a UDP Connect scan on your Linux system from a remote system:
   a. Determine your Linux server's IP address by entering **`ifconfig`** at the command line. The output will look similar to the following, and your system's IP address follows `inet addr:` in the `ifconfig` command's output.

```
# ifconfig
...
p2p1  Link encap:Ethernet  HWaddr 08:00:27:E5:89:5A
      inet addr:10.140.67.23
```

b. From a remote Linux system, enter the command `nmap -sU IP address` at the command line, using the *`IP address`* you obtained from above. For example:

```
# nmap -sU 10.140.67.23
```

4. To check to see if your system is running the `firewalld` service, and then install and start it if it is not:
   a. Enter `systemctl status firewalld.service`.
   b. If the `firewalld` service is not running, on a Fedora or RHEL system, enter the following:

```
# yum install firewalld firewall-config -y
# systemctl start firewalld
# systemctl enable firewalld
```

5. To open ports in your firewall to allow remote access to your local web service, do the following:
   a. Start the Firewall Configuration window (`firewalld-config`).
   b. Make sure that Configuration: Runtime is selected.
   c. Select your current zone (for example, FedoraWorkstation).
   d. Under Services, select the http and https check boxes.
   e. Select Configuration: Permanent.
   f. Under Services, select the http and https check boxes.
6. To determine your Linux system's current `netfilter/iptables` firewall policies and rules, enter `iptables -vnL` at the command line.
7. To save, flush, and restore your Linux system's current firewall rules:
   a. To save your current rules:

```
# iptables-save >/tmp/myiptables
```

   b. To flush your current rules:

```
# iptables -F
```

   c. To restore the firewall's rules, enter:

```
# iptables-restore < /tmp/myiptables
```

8. To set your Linux system's firewall filter table for the input chain to a policy of `DROP`, enter `iptables -P INPUT DROP` at the command line.
9. To change your Linux system firewall's filter table policy back to `accept` for the input chain, enter the following:

```
# iptables -P INPUT ACCEPT
```

To add a rule to drop all network packets from the IP address 10.140.67.23, enter the following:

```
# iptables -A INPUT -s 10.140.67.23 -j DROP
```

10. To remove the rule that you just added, without flushing or restoring your Linux system firewall's rules, enter `iptables -D INPUT 1` at the command line. This is assuming that the rule you added above is rule 1. If not, change the `1` to the appropriate rule number in your `iptables` command.

Chapter 26: Shifting to Clouds and Containers

1. To install and start either `podman` (for any RHEL or Fedora system) or `docker` (RHEL 7):

```
# yum install podman -y
     or
# yum install docker -y
# systemctl start docker
# systemctl enable docker
```

2. To use either `docker` or `podman` to pull this image to your host, `registry.access.redhat.com/ubi7/ubi`:

```
# podman pull registry.access.redhat.com/ubi7/ubi
     or
# docker pull registry.access.redhat.com/ubi7/ubi
```

3. To run the `ubi7/ubi` image to open a bash shell:

```
# podman run -it ubi7/ubi bash
     or
# docker run -it ubi7/ubi bash
```

4. To run commands to see the operating system on which the container is based, install the `proc-ps` package, and run a command to see the processes running inside the container:

```
bash-4.4# cat /etc/os-release | grep ^NAME
NAME="Red Hat Enterprise Linux"
bash-4.4# yum install procps -y
bash-4.4# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:37 pts/0    00:00:00 bash
root        20     1  0 03:43 pts/0    00:00:00 ps -ef
bash-4.4# exit
```

5. To restart and connect to the container that you just closed using an interactive shell, enter the following:

```
# podman ps -a
CONTAINER ID  IMAGE              COMMAND  CREATED
```

```
        STATUS                      PORTS  NAMES
  eabf1fb57a3a  ...ubi8/ubi:latest bash     7 minutes ago
        Exited (0) 4 seconds ago         compassionate_hawking
# podman start -a eabf1fb57a3a
bash-4.4# exit
```

6. To create a simple Dockerfile from a `ubi7/ubi` base image, include a script named `cworks.sh` that echoes "The Container Works!", and add that script to the image so that it runs, do the following:

   a. Create and change to a new directory:

   ```
   # mkdir project
   # cd project
   ```

   b. Create a file named `Dockerfile` with the following content:

   ```
   FROM registry.access.redhat.com/ubi7/ubi-minimal
   COPY ./cworks.sh /usr/local/bin/
   CMD ["/usr/local/bin/cworks.sh"]
   ```

   c. Create a file named `cworks.sh` with the following content:

   ```
   #!/bin/bash
   set -o errexit
   set -o nounset
   set -o pipefail
   echo "The Container Works!"
   ```

7. Use `docker` or `podman` to build an image named `containerworks` from the Dockerfile that you just created.

   ```
   # podman build -t myproject .
         or
   # docker build -t myproject .
   ```

8. To gain access to a container registry, either by installing the docker-distribution package or getting an account on Quay.io or Docker Hub:

   ```
   # yum install docker-distribution -y
   # systemctl start docker-distribution
   # systemctl enable docker-distribution
   ```

   or get an account from Quay.io (`https://quay.io/plans/`) or Docker Hub, then:

   ```
   # podman login quay.io
   Username: <username>
   Password: *********
   ```

9. To tag and push a new image to a chosen container registry:

   ```
   # podman tag aa0274872f23 \
   quay.io/<user>/<imagename>:v1.0
   # podman push \
   quay.io/<user>/<imagename>:v1.0
   ```

Chapter 27: Using Linux for Cloud Computing

1. To check your computer to see if it can support KVM virtualization, enter the following:

```
# cat /proc/cpuinfo | grep --color -E "vmx|svm|lm"
flags  : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm
pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64
monitor ds_cpl vmx smx es...
...
```

   The CPU must support either `vmx` or `svm`. The `lm` indicates that it is a 64-bit computer.

2. To install a Linux system along with the packages needed to use it as a KVM host and, to run the Virtual Machine Manager application, do the following:

   a. Get a live or installation image from a Linux site (such as `getfedora.org`), and burn it to a DVD (or otherwise make it available to install).

   b. Boot the installation image, and select to install it to a hard drive.

   c. For a Fedora Workstation, after the install is complete and you have rebooted, install the following package (for different Linux distributions, you might need to install a package that provides `libvirtd` as well):

```
# yum install virt-manager libvirt-daemon-config-network
```

3. To make sure that the `sshd` and `libvirtd` services are running on the system, enter the following:

```
# systemctl start sshd.service
# systemctl enable sshd.service
# systemctl start libvirtd.service
# systemctl enable libvirtd.service
```

4. Get a Linux installation ISO image that is compatible with your hypervisor, and copy it to the default directory used by Virtual Machine Manager to store images. For example, if the Fedora Workstation DVD is in the current directory, you can enter the following:

```
# cp Fedora-Workstation-Live-x86_64-30-1.2.iso /var/lib/
libvirt/images/
```

5. To check the settings on the default network bridge (`virbr0`), enter the following:

```
# ip addr show virbr0
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
     noqueue state UP group default
    link/ether de:21:23:0e:2b:c1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
       valid_lft forever preferred_lft forever6.
```

6. To install a virtual machine using the ISO image you copied earlier, do the following.
   a. Enter this command:

      ```
      # virt-manager &
      ```

   b. Select File and then select New Virtual Machine.
   c. Select Local Install Media and click Forward.
   d. Select Browse, choose the live or install ISO, click Choose Volume, and click Forward.
   e. Select memory and CPUs and click Forward.
   f. Select the size of disk that you want to use and click Forward.
   g. Select "Virtual network default: NAT" (it may already be selected).
   h. If it all looks okay, click Finish.
   i. Follow the installation process indicated by the installation ISO.
7. To make sure that you can log in to and use the virtual machine, do the following:
   a. Double-click the entry for the new virtual machine.
   b. When the viewer window appears, log in as you would normally.
8. To check that your virtual machine can connect to the Internet or other network outside of the hypervisor, do one of the following:
   a. Open a web browser, and try to connect to a website on the Internet.
   b. Open a Terminal window, enter `ping redhat.com`, and then press Ctrl+C to exit.
9. To stop the virtual machine so that it is no longer running:
   a. Right-click the entry for the VM in the virt-manager window.
   b. Select Shut Down, and then select Shut Down again.
   c. If the VM doesn't shut down immediately, you can select Force Off instead, but that is like pulling the plug out and risks data loss.
10. Start the virtual machine again so that it is running and available:
    a. Right-click the entry for the VM in the virt-manager window.
    b. Click Run.

Chapter 28: Deploying Linux to the Cloud

1. To install the `genisoimage`, `cloud-init`, `qemu-img`, and `virt-viewer` packages, enter:

```
# dnf install genisoimage cloud-init qemu-img virt-viewer
```

2. To obtain a Fedora cloud image, go to `https://getfedora.org/en/cloud/download/`, and download a qcow2 image. There is one listed with OpenStack named `Fedora-Cloud-Base-31-1.9.x86_64.qcow2`.

3. To create a snapshot of that image in qcow2 format called `myvm.qcow2`, enter the following:

```
 # qemu-img create -f qcow2 \
-o backing_file=Fedora-Cloud-Base-31-1.9.x86_64.qcow2 \
myvm.qcow2
```

4. Create a `cloud-init` meta data file named `meta-data` that includes the following content:

```
instance-id: myvm
local-hostname: myvm.example.com
```

5. Create a `cloud-init` user data file called `user-data` that includes the following content:

```
#cloud-config
password: test
chpasswd: {expire: False}
```

6. Run the `genisoimage` command to combine the `meta-data` and `user-data` files to create a `mydata.iso` file:

```
# genisoimage -output mydata.iso -volid cidata \
    -joliet-long -rock user-data meta-data
```

7. Use the `virt-install` command to combine the `myvm.qcow2` virtual machine image with the `mydata.iso` image to create a new virtual machine image named `newvm` that runs as a virtual machine on your hypervisor.

```
# virt-install --import --name newvm \
   --ram 4096 --vcpus 2 \
   --disk path=myvm.qcow2,format=qcow2,bus=virtio \
   --disk path=mydata.iso,device=cdrom \
   --network network=default &
```

8. To open the `newvm` virtual machine with `virt-viewer`, enter the following:

```
# virt-viewer newvm
```

9. Log into the `newvm` virtual machine using the `fedora` user and password `test`:

```
Login: fedora
Password: test
```

Chapter 29: Automating Apps and Infrastructure with Ansible

1. To install the ansible package, do the following:

   RHEL 8

   ```
   # subscription-manager repos \
       --enable ansible-2.9-for-rhel-8-x86_64-rpms
   # dnf install ansible -y
   ```

   Fedora

   ```
   # dnf install ansible -y
   ```

   Ubuntu

   ```
   $ sudo apt update
   $ sudo apt install software-properties-common
   $ sudo apt-add-repository --yes --update ppa:ansible/ansible
   $ sudo apt install ansible
   ```

2. To add `sudo` privileges for the user running Ansible commands, run `visudo` and create an entry similar to the following (changing joe to your user name):

   ```
   joe   ALL=(ALL)       NOPASSWD: ALL
   ```

3. Open a file named `my_playbook.yaml`, and add the following content:

   ```
   ---
   - name: Create web server
     hosts: localhost
     tasks:
     - name: Install httpd
       yum:
         name: httpd
         state: present
   ```

4. To run the `my_playbook.yaml` playbook in check mode, do the following. (It should fail because the user does not have privilege to install a package.)

   ```
   $ ansible-playbook -C my_playbook.yaml
   ...

   TASK [Install httpd]
   ************************************************************
   fatal: [localhost]: FAILED! => {"changed": false, "msg":
   "This
        command has to be run under the root user.",
   "results": []}
   ...
   ```

5. Make the following changes to the `my_playbook.yaml` file:

```
---
- name: Create web server
  hosts: localhost
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present
```

6. To run the `my_playbook.yaml` file again to install the `httpd` package, enter the following:

```
$ ansible-playbook my_playbook.yaml
...
TASK [Install httpd] ************************************
changed: [localhost]
PLAY RECAP **********************************************
localhost: ok=2 changed=1 unreachable=0 failed=0 skipped=0
rescued=0 ignored=0
```

7. Modify `my_playbook.yaml` as follows to start the `httpd` service, and set it so that it will start every time the system boots:

```
---
- name: Create web server
  hosts: localhost
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present
  - name: start httpd
    service:
      name: httpd
      state: started
```

8. To run an `ansible` command so that it checks whether or not the `httpd` service is up on `localhost`, enter the following:

```
$ ansible localhost -m service \
      -a "name=httpd state=started" --check
localhost | SUCCESS => {
```

```
    "changed": false,
    "name": "httpd",
    "state": "started",
    "status": { ...
```

9. To create an `index.html` file in the current directory that contains the text "Web server is up" and runs the `ansible` command to copy that file to the `/var/www/html` directory on `localhost`, do the following (changing joe to your user name):

```
$ echo "Web server is up" > index.html
$ ansible localhost
 -m copy -a \
    "src=./index.html dest=/var/www/html/ \
    owner=apache group=apache mode=0644" \
    -b --user joe --become-user root --become-method sudo
host01 | CHANGED => { ...
```

10. To use the `curl` command to view the contents of the file you just copied to the web server, do the following:

```
$ curl localhost
Web server is up
```

Chapter 30: Deploying Applications as Containers with Kubernetes

1. To gain access to a Minikube instance, either:
   a. Install Minikube as described here: `https://kubernetes.io/docs/tasks/tools/install-minikube`, or
   b. Access an available remote Minikube instance, such as through the Kubernetes.io tutorials: `https://kubernetes.io/docs/tutorials/`
2. To view the versions of your Minikube installation, `kubectl` client, and Kubernetes service, enter the following:

```
$ minikube version
$ kubectl version
```

3. To create a deployment that manages a pod running the `hello-node` container image, enter the following:

```
$ kubectl create deployment hello-node \
    --image=gcr.io/hello-minikube-zero-install/hello-node
```

4. To view the `hello-node` deployment and describe the deployment in detail, enter the following:

```
$ kubectl get deployment
$ kubectl describe deployment hello-node
```

5. To view the current replica set associated with your `hello-node` deployment, enter the following:

```
$ kubectl get rs
```

6. To scale up the `hello-node` deployment to three (3) replicas, enter the following:

```
$ kubectl scale deployments/hello-node --replicas=3
```

7. To expose the `hello-node` deployment outside of the Kubernetes cluster using `LoadBalancer`, enter the following:

```
$ kubectl expose deployment hello-node \
     --type=LoadBalancer --port=8080
```

8. To get the IP address of your Minikube instance and port number of the exposed `hello-node` service, enter the following:

```
$ minikube ip
192.168.39.150
$ kubectl describe service hello-node | grep NodePort
NodePort:                 <unset>  31302/TCP
```

9. Use the `curl` command to query the `hello-node` service, using the IP address and port number from the previous step. For example:

```
$ curl 192.168.39.105:31302
Hello World!
```

10. To delete the `hello-node` service and deployment and then stop the Minikube virtual machine, enter the following:

```
$ kubectl delete service hello-node
$ kubectl delete deployment hello-node
$ minikube stop
```

Index

((left parenthesis), 78
< (less than), 78
' (backtick), 80
? (question mark), 99
& (ampersand character), 78
; (semicolon), 78
) (right parenthesis), 78
| (pipe character), 78–79
; (semicolon), 79
{ } (curly braces), 101
~ (tilde) in commands, 97
# shell prompt, 63
$ shell prompt, 63
> (greater than), 78
3D desktop effects, AIGLX, 54–57

A

absolute path, 96
ACLs (Access Control Lists)
  directories, 267
  enabling, 265–266
  restricted deletion directory, 268–269
  set GID directory, 267–268
  setfacl command, 262–264
  setting, 262–264
    default, 264–265
AD (Active Directory), 712
ad-hoc commands, Ansible, 760–762
administrative commands
  /bin directory, 179
  /sbin, 178
  /usr/bin directory, 179
  /usr/sbin, 179
administrative privileges, 168
administrative utilities, 4–5
AIGLX (Accelerated Indirect GLX), 54–57
Alcatel-Lucent, 9
alias command, 81–83
aliases, 71
  completion, 75
  creating, 81–83
  network interfaces, 360–361
  using, 81–83
aliases file, 181
amanda, 566
Amazon EC2, cloud images, 744–746
ampersand character (&), 78
anaconda, Linux installation, 205
Anaconda installer, 17
anonymous FTP server, 456
Ansible, 749
  ad-hoc commands, 760–762
  applications, deployment, 749
  containers, 749–750
  deployment
    prerequisites, 754
    SSH keys, 754–755
  host systems, configuring, 749
  infrastructure, 749
  installation
    authentication, 757
    inventories, 756–757
    playbook creation, 757–758
    playbook running, 758–760
  inventories, 751–752
  operators, 749–750
  playbooks, 749
    imports, 753
    includes, 753
    modules, 752–753
    plays, 752
    roles, 753
    tasks, 752
Ansible Tower, 762–763
antivirus software, 591
Apache HTTPD web server, 427–428, 432–433
  access denied error, 452
  configuration files
    default settings, 438–440
    directives, 435–436, 435–438
  httpd package, 428–431
  index not found error, 452
  installing, 431

Linux® Bible, Tenth Edition. Christopher Negus.
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Apache HTTPD web server *(continued)*
security
file ownership, 433
file permissions, 433
firewalls, 433–434
SELinux and, 434–435
SSL/TLS and, 443–445
certificate signing request, 448–449
self-signed certificates, 447–448
SSL configuration, 445–447
SSL key generation, 447–448
troubleshooting
configuration errors, 449–451
errors, accessing, 451–452
user published content, 442–443
virtual hosts, 440–442
applets, GNOME, 51
application-layer firewalls, 674
applications
Ansible, 749–750
deploying as containers, 765–783
`aptitude` command, 226
arguments, commands, 67–68
arithmetic
expressions, expanding, 80
integer arithmetic, 152–153
shell scripts, 152–153
ASF (Apache Software Foundation), 428
ash shell, 61
AT&T, 8
audits, 595–596
network services, 663–665
`nmap` utility, 665–672
ports, 666
authentication, 4
cloud, 712
key-based, Secure Shell tools, 324–326
PAM
contexts, 619–620
control flags, 620–621
modules, 621–622
public key, 313
author rights, GPL and, 12
`autofs`
home directory automount, 518–520
`/net` directory, 517–518
automation, Ansible Tower and, 762–763
AWS (Amazon Web Services), 694

B

background commands, 79
background processes, 137–140
backtick ('), 80
backup script, 162
backup utilities, 566
base image, 696
container image, 694
bash shell, 61, 148
configuration files, 84
`cut` command, 159
`grep`, 159
prompt, characters, 86
`sed` command, 160
text manipulation, 159–161
`tr` command, 160
variables, untyped, 152–153
`bashrc` file, 181
Bell labs, 7–8
Bell-LaPadula Mandatory Access security model, 639
Berkeley distribution, 9–10
`bigcommand` process, 141
`/bin` directory, 94
BIOS (Basic Input Output System), 526–528
`bond0` interface, 362
bonded network interfaces, 362
Booleans, SELinux, 653–654
`/boot` directory, 94
boot loaders
GRUB (GRand Unified Bootloader), 528–530
installation, 217–218
troubleshooting, 528–530
GRUB 2, 530–531
troubleshooting, 530–531
boot options
feature disable, 210
kickstarts, 211–212
`mediacheck`, 212
rescue mode, 212
special installation, 210–211
video problems, 210
boot order, troubleshooting, 527–528
boot up, 523–524
from firmware, 526–528
GRUB 2 boot loader, 530–531
GRUB boot loader, 528–530

kernel startup, 532–541
startup methods, 524
init facility, 524–525
systemd facility, 525
from USB drive, 791–792
bounties, software, 21
Bourne, Stephen, 61
Bourne shell, 61
brace expansion metacharacters, 101
browser-based admin tools, 173
BSD (Berkeley Software Distribution), 10, 12–13
FreeBSD, 13
init daemon, 371–377
NetBSD, 13
OpenBSD, 13
built-in commands, 71

C

C programming language, 9
Caesar Cipher, 602
case command, 156–157
cd command, 96–97
CDs/DVDs, burning, 792–795
Ceph, 5
certification, 21–22
RHCE (Red Hat Certified Engineer), 21, 22
topics, 23–25
RHCSA (Red Hat Certified System Administrator), 21, 22
topics, 22–23
cgroups, 143–144, 695
chage command, 72
chkconfig command, 385–386
chkrootkit, 592
chmod command, 100, 106–108, 162
chronyd package, 309
CIFS (Common Internet File System), 475–476
CISA (Cybersecurity and Infrastructure Security Agency), 596
classes
implicit, 404
printer classes, 404, 408
classification level, 639
cloning, cloud instances, 734–738
cloud, 3–4
hybrid, 731
hypervisors, 710, 713
configuring, 715–718
setup, 714
images
Amazon EC2, 744–746
OpenStack and, 739–744
instances, 730
cloning, 734–738
investigating, 733–734
networking, setup, 714
platforms, 712
private, 730
public, 730
storage, 711
configuring, 718–720
setup, 714
shared, 713
virtual machines, 713
creating, 720–724
cloud computing, 5
authentication, 712
configuration, 712
controllers, 711
deployment, 712
cloud-based installations, 204–205
cloud-init, 730
configuring, 731–733
enterprise computing, 738
running, 731–733
clustering, 5
cnegus-test-project, 739
Cockpit, 168, 169–171, 249–252
firewall rules, 677–678
storage management and, 301–303
command languages, shell and, 62
command-line
argument, 148, 150–151
completion, 75–76
editing, 73–75
network configuration, nmtui command, 354
NetworkManager TUI, editing connection, 354–355
recall, 76–78
commands, 418
′ (backtick), 80
~ (tilde), 97
ad-hoc (Ansible), 760–762
administrative
/sbin, 178
/usr/sbin, 179

commands *(continued)*
alias, 81–83
aliases, 71
aptitude, 226
arguments, 67–68
background, 79
built-in, 71
case, 156–157
cd, 96–97
chage, 72
chkconfig, 385–386
chmod, 100, 106–108, 162
completion, 75
connecting, 78–81
cp, 110
cryptsetup, 612
cut, 159
date, 66
df, 334
du, 334–335
expanding, 78, 80
exportfs, 507
files, 96–98
filesystem, 71
find, 122–128, 335–336
firewall-config, 674–675
functions, 71
gedit, 113–114
grep, 128–129
groupadd, 260–261
help, 88
here text, 100
history, 72–78
history, 72–78
id, 69
info, 89
information about, 88–90
journalctl, 184
kill, 140
killall, 140, 141–142
lftp, 470–472
locate, 72, 120–122
locating, 70–72
lp, 419
lprm, 419–420
lpstat, 419
ls, 67–68, 101–105
man, 89
mkfs, 300
mount, 297–298
mv, 109–110
nice, 142–143
nmtui, 354
one-command actions, 156
options, 67–68
path, 70
pipe (|)metacharacter, 78–79
podman, 694, 697
ps, 132–134
pwd, 67
renice, 142–143
reserved words, 71
rm, 110
rpm, 241–245
running, 66–72
sar, 332–333
secon, 648–649
sed, 160
sequential, 79
setfacl, 262–264
sftp, 324
ssh, 316
su, 168, 175–176
sudo, 168
syntax, 67–70
systemctl, 381
telinit, 374
text formatting, 79
top, 134–135
touch, 98–99
tr, 160
type, 71
umount, 299
useradd, 252–255
userdel, 258–259
usermod, 257–258
virsh, 711
virt-install, 720–721
virt-manager, 714
virt-viewer, 714
who am i, 65
yum, 229–232, 233–241
Compiz, 55
compliance reviews, 595–596
compute nodes, 710. *See also* cloud hypervisors
configuration
cloud, 712
hypervisors, 715–718
storage, 718–720
files, 310

administrative, 179–185
plain-text files, 179
security, 314
servers, 310–311
configuration files, 310
default configuration, 310–311
connecting commands, 78–81
connectionless protocols, UDP, 666
container registries, 694, 695–696
images
pushing to, 705–706
tagging, 705–706
containers, 693–694
Ansible, 749–750
in enterprise, 706
FTP, GitHub and, 703–705
images, 694
base image, 694, 696
building, 702–703
RPM packages and, 246
namespaces, 694, 695
pulling, 697–698
running FTP servers, 699–701
running shells, 698–699
sidecar, 766
starting/stopping, 701–702
control plane, 336
copying files, 110
interactive copying, 324
`scp` command, 321–324
copyrights, GPL and, 12
`cp` command, 110
`cpio`, 566
CPU (computer processing unit), 273
`cracklib`, 571
`cron`, software updates and, 545
`crontab` file, 182
cryptographic ciphers, 602–603
cryptography, 599–600
asymmetric keys, 604–605
block ciphers, 600
cipher keys, 603–608
ciphers, 600
decryption, 600
digital signatures, 608–610
email message encryption, 607–608
encryption/decryption
cipher keys, 603–608
ciphers, 602–603
digital signatures, 608–610
hashing, 600–601
implementing
directories, 613–615
encryption from desktop, 617–618
file encryption, 616
file integrity, 610–611
filesystem encryption, installation, 611–613
tools, 616–617
key pair generation, 605–606
public key sharing, 607
stream ciphers, 600
symmetric keys, 603–604
tar archive files, 604
tools, 617
`cryptsetup` command, 612
`csh` (C shell), 61, 65
`csh.cshrc` file, 182
CUPS (Common UNIX Printing System), 403–404
configuring
from a browser, 404
manual, 404, 417–418
printer drivers, 404
printers, adding automatically, 405–406
printing to from Windows, 405
remote printers, 413
server
configuring, 415–416
starting, 417
shared printers, 420–422
web-based administration, 406
automatic detection, 407–408
remote administration, 406–407
curly braces (), 101
`cut` command, 159
cutting text, 159
CVE (Common Vulnerabilities and Exposures), 580

D

DAC (Discretionary Access Control), 635–636
daemon processes, 5, 179, 307. *See also* services
`apache`, 185
`avahi`, 185
`bin`, 185
`chrony`, 185
configuration files, 311
`lp`, 185
`news`, 186
permissions and, 311
port numbers, 311

daemon processes *(continued)*
postfix, 185
rpc, 186
services, 369
dash shell, 61, 65
datasources, 738
date command, 66
DEB packaging, 225
Ubuntu Software Center, 225
Debian, 19
debugging, shell scripts, 148
dependencies, 369
dependent software, 224
deployment, automatic, 336
desktop. *See also* GNOME; GNOME 2; GNOME 3; X Window System
3D effects, AIGLX, 54–57
GNOME, 29, 30
GNOME 3, 31
KDE (K Desktop Environment), 29
LXDE (Lightweight X11 Desktop Environment), 29
window manager, 29
Xfce, 29
desktop networking
configuring, NetworkManager, 340–342
NetworkManager, 340–342
/dev directory, 94
df command, 334
DHCP, 340–341
digital signatures, 608–610
directories
/bin, 94
/boot, 94
/dev, 94
encrypting, 613–615
/etc, 94, 180
/etc/cron, 180
/etc/cups, 180, 405
/etc/default, 180
/etc/exports, 504, 505
/etc/httpd, 181
/etc/mail, 181
/etc/postfix, 181
/etc/ppp, 181
/etc/rc?.d, 181
/etc/security, 181
/etc/skel, 181
/etc/sysconfig, 181
/etc/systemd, 181
/etc/X11, 183
/etc/xinetd.d, 181
hard drive partitions, 216
hierarchy, 94
/home, 94
identifying, 104
/lib, 94
listing, 101–105
/media, 94
/misc, 94
/mnt, 94
number of characters, 103
/opt, 94
paths, 70
absolute path, 96
order, 71
/proc, 95
restricted deletion directory, 268–269
root, 93
/root, 95
/sbin, 95
/sys, 95
time and date column, 103
/tmp, 95
/usr, 95
/var, 95
disaster recovery, security, 566
disk images, mounting in loopback, 298–299
disk space, 197
disk storage, 273
distributions
components, 16
DEB packaging, 225
Debian, 19
Fedora, 18–19
GPL and, 12
Red Hat
Anaconda installer, 17
graphical administration, 17
RPM package management, 16–17
Red Hat OpenShift, 18
Red Hat OpenStack Platform, 18
RHEL (Red Hat Enterprise Linux), 17–18
RPM packaging, 225
Ubuntu, 19
DNF (Dandified YUM), 229
Docker, 697
docker command, 694, 697

docker daemon, 694
Docker Desktop, 768
Docker Hub, 696
Docker project, 694
drivers, printer drivers, 404
du command, 334–335
dual booting, 208–209
dumb terminals, 137
dump/restore, 566
DVD, installing Linux, 196
DVD drive, 197

E

echo statement, 148
ecryptfs, 613–615
emacs editor, 114
email, encrypting messages, 607–608
encryption/decryption
  cipher keys
    asymmetric keys, 604–605
    email message encryption, 607–608
    key pair generation, 605–606
    public key sharing, 607
    symmetric keys, 603–604
    tar archive files, 604
  ciphers, 602–603
  digital signatures, 608–610
enterprise
  cloud-init, 738
  containers and, 706
  installing Linux, 196
  network configuration
    Linux as DHCP server, 365
    Linux as DNS server, 365–366
    Linux as proxy server, 366
    Linux as router, 364
  Samba and, 497
  server management, 336
  software management, 245–246
  user management, 261
    ACLs (Access Control Lists), 262–269
    permission setting, 262–269
enterprise networking, 340
environment variables, 81, 82–83
  PATH, 70
  shell, adding, 87
escaping shell characters, 149
/etc directory, 94
  aliases file, 181
  bashrc file, 181
  crontab file, 182
  csh.cshrc file, 182
  exports file, 182
  fstab file, 182
  group file, 182
  gshadow file, 182
  host.conf file, 182
  hostname file, 182
  hosts file, 182
  inittab file, 182
  mtab file, 182
  mtools.conf file, 182
  named.conf file, 182
  nsswitch.conf file, 182
  ntp.conf file, 182
  passwd file, 182
  printcap file, 183
  profile file, 183
  protocols file, 183
  rpc file, 183
  rsyslog.conf file, 183
  services file, 183
  shadow file, 183
  shells file, 183
  sudoers file, 183
  xinetd.conf file, 183
/etc/cron directory, 180
/etc/cups directory, 180, 405
/etc/default directory, 180
/etc/exports file
  configuring, 504
  hostnames, 505–506
  nfsnobody, 506
  options, 506
  root user, 506
  user mapping options, 506
/etc/fstab file, mountable file systems, 295–297
/etc/hostname file, 358
/etc/hosts file, 358
/etc/httpd directory, 181
/etc/mail directory, 181
/etc/nsswitch.conf file, 359–360
/etc/postfix directory, 181
/etc/ppp directory, 181
/etc/rc?.d directory, 181
/etc/rc.d/init.d directory, 375–376
/etc/resolv.conf file, 359

/etc/samba/smb.conf file
[global], 486–487
[homes], 486, 487–489
[printers], 486, 489–493
/etc/security directory, 181
/etc/services file, 663–664
/etc/skel directory, 181
/etc/sysconfig directory, 181
/etc/sysconfig/network file, 358
/etc/systemd directory, 181
/etc/X11 directory, 183
/etc/xinetd.d directory, 181
eth0 interface, 362
Ethernet, 339
channel bonding, 361–362
Execute permissions, 106
exercise answers
Ansible, 858–860
application automation, 858–860
cloud computing, 855–856
deploying to cloud, 857
shifting to, 853–854
desktop creation, 797–800
disk management, 819–821
file management, 819–821
filesystem, 802–803
FTP server configuration, 835–838
infrastructure automation, 858–860
Kubernetes, 860–861
Linux installation, 812–813
network administration, 825–827
NFS file server configuration, 841–843
print server configuration, 829–831
processes, running, 805–807
Samba server configuration, 838–841
security
advanced, 847–849
basic, 845–847
network security, 851–853
SELinux and, 849–851
server administration, 822–825
services, starting/stopping, 827–829
shell, 800–802
shell script writing, 807–810
software acquisition, 814–815
software management, 814–815
system administration, 810–812
text files, 804–805
troubleshooting, 843–845
user account management, 815–819
web server configuration, 831–835
exiting shell, 83–84
expanding commands, 78, 80
expanding parameters, 151–152
expanding variables, 80–81
exportfs command, 507
exporting shared filesystems, 507
exports file, 182
exports man page, 504
expressions
arithmetic, expanding, 80
test expressions, 154
operators, 155–156
extended memory, 4

F

FCoE (Fibre Channel over Ethernet Devices), 213
Fedora, 18–19, 27
downloading, 788–789
installing from Live media
bare metal system, 198
multi-boot, 198
single-boot, 198
virtual system, 198
Terminal window, 64
Fibre Channel, 5
file-matching metacharacters, 98–99
filenames, shell scripts, 148
file-redirection metacharacters, 99–100
files
commands, 96–98
copying, 110
scp command, 321–324
encrypting, 616
listing, 101–105
moving, 109–110
Nautilus, 42, 43
filesystem organization, 42–43
ownership
Apache web server, 433
changing, 109
password files, 574–576
permissions, 105–106
Apache web server, 433
changing, chmod, 106–108
default, 108–109
errors, 452
Execute, 106

Read, 106
vsftpd, 465
Write, 106
removing, 110
searching for
by date and time, 125–126
find command, 122–128
grep command, 128–129
locate command, 120
by name, 123–124
-not, 126–127
-or, 126–127
by permission, 125
by size, 124
by user, 124
filesystems, 4, 93, 273, 275, 500
commands, 71, 96–98
creating, 300
directories, 94
encryption at installation, 611–613
Linux compared to Windows-based, 95
mounting, 291–293
autofs, on demand, 517–520
defining mountable systems, 295–297
/etc/fstab file, 295–297
mount command, 297–298
NFS, 512–520
options, 515–517
swap areas
disabling, 294–295
enabling, 293–294
Nautilus organization, 42–43
noauto, 514–515
partitions, 273
root directory, 93
security
dangerous permissions, 576–577
lockdown, 578–579
securing files, 577–578
shared, exporting, 507
system administrator and, 168
filter table, 678
find command, 122–128, 335–336
Firefox, FTP access, 470
firefox package, 227
Firewall Configuration window, 509, 674–675
firewall-config command, 674–675
firewalld service, 674–675, 675–677
firewalls, 313, 672–674
Apache web server, 433–434
application-layer firewalls, 674
implementing, 674–688
iptables, 673
iptables, 313
network-layer firewalls, 674
rules, Cockpit and, 677–678
Samba, 482–483
troubleshooting and, 552–553
firmware
RAID devices, 213
starting from, 526–528
folders, Nautilus, 42, 43
creating, 43
Home folder, 42
for...do loop, 157–158
foreground processes, 137–138
commands, 139–140
FOSS (Free and Open Source Software), 12, 666
free distribution, GPL and, 12
free software, 12
Free Software Directory, 12
FreeBSD, 13
FSF (Free Software Foundation), 11
fstab file, 182
FTP (File Transfer Protocol), 455–456
active connection, 456
clients, 469–473
command-oriented clients, 456
containers, GitHub and, 703–705
passive connection, 456
server, 309
accessing, 470–472
anonymous, 456
firewall, 461–463
gFTP client, 472–473
graphical tools, 456
SELinux, configuring, 463–465
uploading, allowing, 467–468
user access, setup, 465–466
vsftpd, 457–461
servers, running from container, 699–701
functions, 71
completion, 75

G

gconf-editor, 173
gedit command, 113–114
general regular expression print. *See* grep command

Gentoo, 16, 207
gFTP client, 472–473
Gibson Research Corporation, 596
GitHub, FTP containers, 703–705
GlusterFS, 5
GNOME, 30–31
GNOME 2
  Appearances Preference, 49
  Compiz, 46
  GNOME panels, 47
  Metacity, 46, 48–49
  Nautilus, 46
  panels, 50
    adding, 52
    applets, 51
    application launcher, 52–53
    Applications menu, 51
    drawers, 53
    moving items, 50
    properties, 54
    resizing items, 51
    System menu, 51
    Window list, 51
  Preferences, 47
GNOME 3, 31
  applications, 41
    additional, 34
    launching, 37–38
    opening, 32
  Applications view, 37
  Bluetooth, 39
  bootup, 31
  commands, launching, 37–38
  dash, 36
  devices, 39
  keyboard, Windows key, 36
  Nautilus
    files, 42–43
    folders, 42, 43
    FTP with login, 43
    Public FTP, 43
    remote content, 43
    Rhythmbox, 45–46
    Secure (HTTPS), 43
    software, 43–45
    SSH and, 43
    WebDav (HTTP), 43
    Windows share, 43
  navigation
    keyboard, 36–38
    mouse, 32–35
  networking, 39
  searches, 37
  set up, 38–39
  shell extensions, 39–40
  sound, 39
  stopping, 46
  System Settings window, 38
  toggling, 32
  top bar, 36
  Tweak Tool, 40–41
  views, 36
  window menu, 35
  windows
    active, 37
    minimized, 33
    opening, 32
  Windows view, 36
  workspaces, multiple, 34–35
GNOME Terminal, 64
gnome-disks, 173
gnome-utils, 173
GNU (GNU is Not UNIX), 11–12
  BSD (Berkeley Software Distribution) License, 15
  LGPL (Lesser General Public License), 15
  MIT license, 15
  Mozilla license, 15–16
GNU Project page, 11
GPL (GNU Public License), 12
graphical tools, 172
graphical windows, 168
graphics, Red Hat, 17
greater than (>), 78
grep command, 128–129, 159
group accounts, 259–261
group file, 182
groupadd command, 260–261
groups, 249
GRUB (GRand Unified Bootloader), 217–218, 528–530
  troubleshooting, 528–530
GRUB 2 boot loader, 530–531
  troubleshooting, 530–531
gshadow file, 182
GUID (Globally Unique Identifier), partition tables, 276
GUIs (graphical user interfaces), 61

H

hard drive, partitioning
  assigning to directory, 216
  filesystem types, 214
  Linux partitions, 215

LVM partitions, 215
multiple operating systems, 214
multiple-partition disks, 281–285
partition tables, 275–276
RAID partitions, 215
single-partition disks, 277–281
swap partitions, 215
viewing partitions, 276–277
hardware, 4
checking, 187–189
kernel and, 186–193
modules, loadable, 191–193
removable, 189–191
requirements, 196–197
hashed passwords, 574–576
headers, packet headers, 673
`help` command, 88
here text, 100
hierarchy of directories, 94
`history` command, 72–78
`/home` directory, 94
host systems, generic, 336
`host.conf` file, 182
hostname
completion, 75
`/etc/exports`, 505
`hostname` file, 182
hosts
individual, 505
IP network, 505
TCP/IP domain, 505
`hosts` file, 182
`httpd` daemon, 442–443
`httpd` package, 310
hybrid cloud, 731
hypervisor, 709, 713
configuring, 715–718
DNS, setup, 718
`/etc/hosts`, editing, 718
Linux, installing, 716–717
naming, 717
services, 717–718
setup, 714

I

`id` command, 69
IDS (Intrusion Detection System), 592–595
`if...then` statements, 153–154
images, 694
building, 702–703
cloud
Amazon EC2, 744–746
OpenStack and, 739–744
for clouds, 731–733
container registries and, 705–706
virtual machines, 721
implicit classes, 404
Infiniband, 5
`info` command, 89
information about commands, 88–90
`init`, 369, 370–371, 371–377, 524–525
runlevels, 373–374
`systemd`, 370–371
`SysVinit`, 370
troubleshooting, 533
`inittab` file, 182
input/output redirection, 8
installation
Apache HTTPD web server, 431
boot options
feature disable, 210
kickstarts, 211–212
`mediacheck`, 212
rescue mode, 212
special installation, 210–211
video problems, 210
dual booting
defragmenting, 209
hard disk, adding, 208
Windows partition resize, 208
GRUB (GRand Unified Bootloader), 217–218
NFS server, 502
`nmap` utility, 665–666
Samba, 476–478
from scratch, 207
servers, 308–310
software, 221–222
storage, specialized, 213
installation server, 206
installing Linux
cloud based installations, 204–205
from DVD, 196
Red Hat Enterprise, 201–204
in enterprise, 196, 205–207
GRUB (GRand Unified Bootloader), 217–218
hard drives, partitioning, 214–217
from Live media, 195
Fedora installation, 198–201
from scratch, 207
virtualization and, 209
integer arithmetic, 152–153

interactive copying, 324
interfaces, 4
interpreter, shell script, 148
IP (Internet Protocol), 341
  addresses
    aliases, setting, 350–351
    manually setting, 349–350
    routes, setting, 351–352
    source, blocking, 684–685
IP masquerading, 673
IPC (interprocess communications), 695
IPP (Internet Printing Protocol), 404
`iptables` firewall, 313
`iptables` utility, 673, 674, 680
  chains, 679
  configuration, saving, 687–688
  `DROP`, 683
  `filter` table, 678
  `mangle` table, 678
  `nat` table, 678
  options, 683
  policies, modifying, 680–683
  port blocking, 685–687
  protocol blocking, 685–687
  `raw` table, 678
  rules, modifying, 680–683
  `security` table, 678
  source IP address blocking, 684–685
iSCSI, 5, 213
ISO images, 787

J

Java, JBoss, 18
JBoss, 18
`journalctl` command, 184

K

Kali Linux, 595
KDE (K Desktop Environment), 29
Kerberos, 309, 712
kernel, 13, 16
  hardware and, 186–193
  ring buffer, 532
  starting, 532–541
  startup, 532–541
Kernighan, Brian, 9
key-based authentication, Secure Shell tools, 324–326
keystrokes
  command history, 77–78
  command-line editing, 74
kickstart files, 206
  boot options and, 211–212
  Linux installation, 205
  RPM packages and, 246
`killall` command, 141–142
killing processes
  `kill` command, 140
  `killall` command, 140, 141–142
KNOPPIX, 16
`ksh` (Korn shell), 61, 65
Kubernetes, 5, 765–766
  accessing, 769–771
  applications, 767–768
  clusters, 766
  container engines, 767
  containers and, 694
  Docker Desktop, 768
  interfaces, 768
  Minikube and, 766, 768
    starting, 770–771
  nodes
    master node, 766–767
    worker node, 766, 767
  OpenShift and, 782–783
  pods, 766
  services, 766
  sidecar containers, 766
  storage, 766
  tutorials, 768
Kubernetes Basics Tutorial, 769–770, 771–772
  application deployment, 772–773
  applications
    exposing, 776–777
    scaling down, 781–782
    scaling up, 779–780
  load balancer, 780–781
  pod information, 773–776
  services
    deleting, 778–779
    labeling, 777–778
KVM (Kernel-based Virtual Machine), 5, 209, 710, 713

L

LAMP (Linux, Apache web server, MySQL database, PHP web scripting language) stack, 3
laptops, networking, 340
LDAP (Lightweight Directory Access Protocol), 270, 309

LDP/LPR printers, 413
left parenthesis (( ), 78
less than (<), 78
lftp command, 470–472
/lib directory, 94
Libvirt Service Daemon, 713
libvirtd service, 713, 715–718
libvirt-daemon-config-network package, 717
Linux
  booting from USB drive, 791–792
  compared to other OSs, 6
  features, 4–5
  history, 3
    Bell labs, 7–8
    BSD, 12–13
    commercial UNIX, 9–11
    GNU, 11–12
    OSI, 14–16
    UNIX, 7–8
  kernel, 13
  Minix, 7
  as open source UNIX-like OS, 14
Linux Foundation, 14
Linux partitions, 215
listing
  directories, 101–105
  files, 101–105
Live media, installing Linux from, 195
  Fedora, 198–201
locate command, 72, 120–122
logwatch service, 331–332
loopback, mounting disk image, 298–299
loops
  for...do, 157–158
  until...do, 158–159
  while...do, 158–159
lp command, 419
lprm command, 419–420
lpstat command, 419
ls command, 67–68, 101–105
LUKS (Linux Unified Key Setup), 612
LVM (Logical Volume Manager), 273, 274, 538
  partitions, 215, 285
    creating logical volumes, 289–290
    displaying, 286–288
    volume growth, 290–291
  physical volumes, 273
LXDE (Lightweight X11 Desktop Environment), 29

M

MAC addresses, 341
mail server, 309
man command, 89
man pages, 502
  exports, 504
  sections, 89
Mandrake, 16
Mandriva, 16
mangle table, 678
mangling packet headers, 673
master nodes, 336
MBR (Master Boot Record), 275
MCS (Multi-Category Security), 638
/media directory, 94
mediacheck, 212
memory, 4
  OOM condition, 556
  page caches, 558
  processes, killing, 558
  troubleshooting, 553–559
metacharacters, 78
  brace expansion, 101
  file-matching, 98–99
  file-redirection, 99–100
meta-data, 730
Microsoft Active Directory. *See* AD (Active Directory)
migration, VMs, 725–727
Minikube, 766, 768
  starting, 770–771
Minix, 7
/misc directory, 94
mkfs command, 300
MLS (multi-level security), 638–639
/mnt directory, 94
modules
  loaded, listing, 191–192
  loading, 192
  removing, 192–193
monitoring servers
  Cockpit, 314
  crackers, 315
  logging configuration, 314
  software updates, 315
  system activity reports, 314
mount command, 297–298
mounting, 274
  disk images, in loopback, 298–299
  filesystems, 291–293

mounting *(continued)*
defining mountable systems, 295–297
/etc/fstab file, 295–297
mount command, 297–298
options, 515–517
swap areas, 293–295
unmounting NFS, 520–521
NFS
autofs, on demand mounting, 517–520
at boot time, 513–517
manually, 512–513
umount command, 299
mount-level security, 499
moving files, 109–110
MS-DOS filesystems, 95
MTA (Mail Transport Agent) server, 309
mtab file, 182
mtools.conf file, 182
Multics, 8
multipath devices, 213
multitasking, 167
multiuser features, 167
mv command, 109–110

N

named.conf file, 182
namespaces, containers, 694, 695
nano editor, 114
nat table, 678
Nautilus
files, 42, 43
filesystem organization, 42–43
folders, 42, 43
creating, 43
Home folder, 42
FTP with login, 43
Public FTP, 43
remote content, 43
Rhythmbox, 45–46
Secure (HTTPS), 43
software
installing, 43–45
managing, 43–45
SSH and, 43
WebDav (HTTP), 43
Windows share, 43
NCSA (National Center for Supercomputing Applications), 428
NetBEUI, 475
NetBSD, 13
netfilter/iptables tables, 678–679
policies, 679–680
rules, 679–680
targets, 679–680
network bridge, VMs, 721
network cards, 197
network interfaces
aliases, 360–361
bonded, 362
Cockpit and, 343–345
command line and, 345–349
configuring
IP address aliases, 350–351
IP address manual set, 349–350
route setting, 351–352
domain names, 349
host names, 349
NetworkManager and, 342–343
routing information, 347–348
troubleshooting and, 547–548
viewing, from command line, 345–347
Network Mapper, 665
network services, auditing, 663–665
nmap utility, 665–672
networking
cloud, setup, 714
configuration
command line, 353–364
enterprise, 364–366
files, 355–360
custom routes, 363–364
desktop, 340–353
enterprise, 340
Ethernet channel bonding, 361–362
hostnames, troubleshooting and, 549–550
laptop, 340
physical connections, troubleshooting and, 548
proxy connections, configuring, 352–353
routes
custom, 363–364
troubleshooting and, 548–549
servers, 340
troubleshooting
incoming connections, 550–553
outgoing connections, 547–550
network-layer firewalls, 674
NetworkManager, 340

DHCP
server response, 340
service request, 340
domain name server, 341
gateway, default, 341
IP address, 341
lease time, 341
local settings, 342
network interfaces, activating, 340
network settings, 350
subnet mask, 341
NetworkManager TUI, connection, editing, 354–355
NFS (Network File System) server, 309, 499
filesystems, sharing, 503–507
installing, 502
mounting
`autofs`, on demand mounting, 517–520
at boot time, 513–517
client, 500
manually, 512–513
security, 508
filesystem structure exposure, 508
firewall, 508–510
`root` users, 508
SELinux configuration, 511–512
TCP wrappers, 510–511
unencrypted communications, 508
user mapping, 508
shares, viewing, 512
unmounting, 520–521
`nfs-server` service, starting, 502–503
`nfs-utils` package, 502
`nice` command, 142–143
NIS (Network Information Service), 270
groups, 506
`nmap` utility
installing, 665–666
port scans, 666
port states, 667
`nmbd` service
starting, 480–481
stopping, 481–482
`nmtui` command, 354
`noauto` filesystem, 514–515
Nokia, 9
`nsswitch.conf` file, 182
`ntp.conf` file, 182
`ntpd` package, 309

O

OEM (original equipment manufacture), 10
one-command actions, 156
OOM condition, 556
open source, storage platforms
Ceph, 5
GlusterFS, 5
Open Source Development Labs, 14
open source software, 12
OpenBSD, 13
OpenPGP, 616
OpenShift, 173
Kubernetes and, 782–783
`openssh` package, 316
`openssh-clients` package, 316
`openssh-server` package, 316–318
OpenStack, 5
cloud images, 739–744
remote access keys, 741–742
VM access via ssh, 743–744
VM launch, 742–743
operators
Ansible, 749–750
expressions, test expressions, 155–156
`/opt` directory, 94
options, commands, 67–68
OSI (Open Source Initiative), 14–16
OSs (operating systems), Linux and, 6
oVirt project, 5
ownership of files, 109

P

package collections, 308
packages
`nfs-utils`, 502
`openssh`, 316
`openssh-clients`, 316
`openssh-server`, 316
servers
directory server, 309
DNS server, 309
FTP server, 309
mail server, 309
Network Time Protocol server, 309
NFS file server, 309
print server, 309
`rsyslog` service, 308–309

packages *(continued)*
SQL server, 309
system logging server, 308–309
web server, 309
Windows file server, 309
packet filters, 674
packet headers, mangling, 673
packets, blocking/allowing, 673
PAM (Pluggable Authentication Module), 312
administering
application configuring files, 622–623
password enforcement, 628–632
`sudo` and, 632–633
system event configuration files, 623–626
time restrictions, 626–627
authentication process, 619
contexts, 619–620
control flags, 620–621
modules, 621–622
resources, 633
`PAM` facility, 312
panels (GNOME), 50
adding, 52
applets, 51
application launcher, 52–53
Applications menu, 51
drawers, 53
moving items, 50
properties, 54
resizing items, 51
System menu, 51
Window list, 51
parameters, shell script
expanding, 151–152
reading in, 151
partition tables, GUID, 276
partitioning
filesystems, 273
hard drives
assigning to directory, 216
filesystem types, 214
Linux partitions, 215
LVM partitions, 215
multiple operating systems, 214
RAID partitions, 215
swap partitions, 215
hypervisors and, 717
LVM (Logical Volume Manager), 285
creating volumes, 289–290
displaying, 286–288
volume growth, 290–291
multiple-partition disks, 281–285
partition tables, 275–276
single-partition disks, 277–281
viewing, 276–277
`passwd` file, 182
passwords, 312–313
changing, 571–572
enforcing best practices, 572–574
files, 574–576
hashes, 574–576
PAM, 628–632
public key authentication, 313
selecting, 570–571
setting, 571–572
path, 70
absolute, 96
order, 71
PATH environment variable, 70, 88
PE (Physical Extent), 286
penetration testing, 595
permissions, 105–106
changing, `chmod`, 106–108
daemons and, 311
default, `umask` value, 108–109
errors, 452
Execute, 106
Read, 106
Write, 106
persistent services, enabling, 391–394
physical security, 565–566
PID (process ID), 131, 370
pipe character (|), 78
plain-text files, 179
playbooks (Ansible), 749
creating, 757–758
imports, 753
includes, 753
modules, 752–753
plays, 752
roles, 753
running, 758–760
tasks, 752
`podman` command, 694, 697
port numbers, daemon processes, 311
portability, 8–9
ports, 666

auditing, 666
blocking, 685–687
scans
TCP Connect, 666
UDP, 666
states, 667
positional parameters, 150–151
POSIX (Portable Operating System Interface), 10
PostgreSQL, 309
print server, 309
configuring
shared CUPS printer, 420–422
shared Samba printer, 422–424
Print Settings window, 403
local printers
adding, 409–411
editing, 411–412
remote CUPS printers, 413
remote LDP/LPR printers, 413
remote printers, configuring, 412–413
remote UNIX printers, 413
Windows (SMB) printer, 414–415
`printcap` file, 183
printer browsing, 404
printer classes, 404
printers, adding automatically, 405–406
printing. *See also* CUPS (Common UNIX Printing System)
canceling jobs, 408
commands, 418
`lp`, 419
`lprm`, 419–420
`lpstat`, 419
drivers, 404
`/etc/cups` directory, 405
IPP (Internet Printing Protocol), 404
listing print jobs, 407
moving jobs, 408
printer classes, 408
UNIX print commands, 404
viewing printers, 408
private cloud, 730
private key cryptography, 603
`/proc` directory, 95
processes, 4, 131–132
background, 137–138
commands, 139–140
`bigcommand`, 141
cgroups, 143–144
changing, 134–135
daemon processes, 5, 179
foreground, 137–138
commands, 139–140
`killall` command, 141–142
killing, 135, 558
`kill` command, 140
`killall` command, 140
limiting, 143–144
listing
`ps` command, 132–134
System Monitor, 136–137
`top` command, 134–135
(|)piping, 133
priorities, 142–143
renicing, 135
RSS (resident set size), 133
VSZ (virtual set size), 133
processors, 196
professional opportunities, 19–21
`profile` file, 183
programming, utilities, 5
prompt, setting, 85–87
protocols, blocking, 685–687
`protocols` file, 183
proxy servers, configuring connections, 352–353
`ps` command, 132–134
public cloud, 730
public key authentication, 313
pulling containers, 697–698
`pwd` command, 67
PXE server, 206
RPM packages and, 246

Q

QEMU, 713
`qemu` process, 713
question mark (?), 99
Qwest, 9

R

RAID partitions, 215
RAM (random access memory), 197, 273–274
troubleshooting and, 554–555
`raw` table, 678
RBAC (role-based access control), 635
TE (type enforcement) and, 637–638
`rc.sysinit`, troubleshooting, 533–534

Read permissions, 106
real-time computing, 5
Red Hat
  Anaconda installer, 17
  graphical administration, 17
  RPM package management, 16–17
Red Hat Enterprise
  downloading, 789–790
  installing, from DVD, 201–204
Red Hat OpenShift, 18
Red Hat OpenStack Platform, 18
Red Hat Virtualization, 5
remote access, 307. *See also* Secure Shell tools
  interactive copying, 324
removable hardware, 189–191
Removable Media window, 189–190
`renice` command, 142–143
repositories, software, 223
rescue mode, 212
  troubleshooting in, 559–561
reserved words, 71
REST API, Ansible Tower and, 763
restricted deletion directory, 268–269
reviews
  compliance, 595–596
  security, 596
RHCE (Red Hat Certified Engineer), 21, 22
  network services, 24–25
  system configuration and management, 24
  topics, 23–25
RHCSA (Red Hat Certified System Administrator), 21, 22
  topics, 22–23
RHEL (Red Hat Enterprise Linux), 17–18
  Terminal window, 64
RHELOSP (Red Hat Enterprise Linux OpenStack Platform), 173
RHEV (Red Hat Virtualization), 173
Rhythmbox (Nautilus), 45–46
right parenthesis (`)`), 78
Ritchie, Dennis, 8, 9
`rlogin`, 316
`rm` command, 110
root directory, 93
`/root` directory, 95
root user, 174
  `sudo` facility, 176–178
  via GUI, 176
  via shell, 175–176
rootkits, 590–595
`route-interface` file, 363
`rpc` file, 183
`rpcbind` service, 503
`rpm` command
  package installation, 241–242
  package removal, 241–242
  querying information, 242–244
RPM package management, 16–17
RPM (RPM Package Manager) packaging, 225, 226–228
  container images, 246
  dependencies, 228
  installing, 228
  kickstart files, 246
  location, 228
  origins, 227–228
  PXE boot, 246
  satellite server, 246
  Spacewalk, 246
  verifying packages, 244–245
  YUM and, 229–232
    third-party software repositories, 233
    transition to DNF, 229
    `yum` command, 233–241
RSS (resident set size), 133
`rsync` command, 322–323, 566
`rsyslog` service, 308, 326–331
`rsyslog.conf` file, 183, 327–329
`rsyslogd` daemon, 326–327
`rsyslogd` facility and, 184–185
runlevels, 369, 373–374
  default, 369
    configuring, 394–395
  troubleshooting, 534–538

S

Samba, 475–476
  enterprise, 497
  `/etc/samba/smb.conf` file
    `[global]`, 486–487
    `[homes]`, 486, 487–489
    `[printers]`, 486, 489–493
  firewalls, configuring, 482–483
  folders
    checking share, 490–493
    shared, 489–490
  host/user permissions, 486
  installing, 476–478
  printer configuration, 422–424

SELinux, configuring, 484–486
share access
Linux file manager, 493–495
mounting from command line, 495–496
Windows, 496–497
starting, 478–480
stopping, 481–482
samba package, 309, 476–478
samba-client package, 476
samba-common packages, 476
samba-winbind package, 476
SAN devices, 213
SANS institute, 596
sar command, 332–333
satellite server, RPM packages and, 246
/sbin directory, 95
SCO (Santa Cruz Operation), 10
scp command, file copy and, 321–324
scripts, backup script, 162
secon command, 648–649
secret key cryptography, 603
Secure Shell tools, 316
client tools, 318–324
key-based authentication, 324–326
security, 307
Apache web server
file ownership, 433
file permissions, 433
firewalls, 433–434
SELinux and, 434–435
Bell-LaPadula Mandatory Access model, 639
cryptography, 599–600
block ciphers, 600
ciphers, 600
decryption, 600
encryption/decryption, 602–610
hashing, 600–601
implementing, 610–618
stream ciphers, 600
disaster recovery, 566
filesystem, 576–579
firewalls, 672–674
application-layer firewalls, 674
Cockpit and, 677–678
implementing, 674–688
iptables, 673
network-layer firewalls, 674
MCS (Multi-Category Security), 638
MLS (multi-level security), 638–639
mount-level, 499
network services, auditing, 663–665, 665–672
PAM (Pluggable Authentication Modules), 618
administering, 622–633
authentication process, 619–622
resources, 633
passwords, 570–576
physical, 565–566
servers, 312–314
configuration file settings, 314
firewalls, 313
passwords, 312–313
SELinux, 313
TCP Wrappers, 313
services, 579–580
software, 579–580
system administration and, 169
systems monitoring, 580–581
filesystem, 587–595
log files, 581–584
user accounts, 584–587
user accounts, 566–569
security clearance, 639
security reviews, 596
security table, 678
sed command, 160
SELinux (Security Enhanced Linux), 313
benefits of, 635–636
Booleans and, 653–654
Booleans for Samba, 484–485
configuration, NFS server, 511–512
errors, 452
file contexts for Samba, 485–486
least privilege access, 636
mode, setting, 645–647
monitoring, 654–656
operational modes
disabled mode, 639–640
enforcing mode, 640
permissive mode, 640
policy rules, 644–645
package management, 651–653
policy types, 643
minimum, 644
MLS (Multi-Level Security), 644
setting, 647–648
targeted, 644
process sandboxing, 636
RBAC, 636

packages *(continued)*
resources, 659
security contexts, 640–641
file security, 650–651
files, 642
`level` attribute, 641
processes, 642–643
`role` attribute, 641
`secon` command, 648–649
`type` attribute, 641
user, 649–650
`user` attribute, 641
users, 641–642
security models, implementing, 639–645
TE (type enforcement), 637–638
testing, 636
troubleshooting
Booleans set incorrectly, 658–659
context labels, lost, 658
logging, 656–657
nonstandard directory as service, 657
nonstandard port as service, 658
semicolon character (;), 78, 79
sequential commands, 79
servers, 307
checking, 316
configuring, 310–311
enterprise and, 336
installation server, 206
installing, 308–310
monitoring, 314–315
Cockpit, 314
crackers, 315
logging configuration, 314
software updates, 315
system activity reports, 314
network, 340
packages
directory server, 309
DNS server, 309
FTP server, 309
mail server, 309
Network Time Protocol server, 309
NFS file server, 309
print server, 309
`rsyslog` service, 308–309
SQL server, 309
system logging server, 308–309
web server, 309
Windows file server, 309
PXE, 206
securing, 312–314
configuration file settings, 314
firewalls, 313
passwords, 312–313
SELinux, 313
TCP Wrappers, 313
setting, 316
starting, 311–312
system administration and, 169
services
`firewalld`, 674–675, 675–677
persistent, enabling, 391–394
reloading, 388
security, 579–580
starting, 5
status, checking, 384–387
`systemd`, new, 399–401
`SysVinit`
new, 396–398
starting and stopping, 387–391
`services` file, 183
set GID bit, 267–268
set UID bit, 268
`setfacl` command, 262–264
`sftp` command, 324
`sh` shell, 65
`shadow` file, 183
shared filesystems, exporting, 507
shell, 4, 61
prompt, 63
`$` prompt, 63
accessing, 62–63
ash, 61
bash shell, 61
configuration files, 84
Bourne shell, 61
command languages and, 62
configuring, 84–85
`csh` (C shell), 61, 65
`dash`, 65
dash shell, 61
environment, 84–87
environment variables, 87
exiting, 83–84
`ksh`, 65
metacharacters

brace expansion, 101
file-matching, 98–99
file-redirection, 99–100
prompt, 63
setting, 85–87
reasons to learn, 62
root user, 175–176
running from container, 698–699
selecting, 65–66
sh, 65
tcsh, 61, 65
variables, 81
shell history, 72–73
shell script
arithmetic in, 152–153
backup script, 162
case command, 156–157
chmod command, 162
command-line argument, 148, 150–151
cut command, 159
for...do loop, 157–158
grep, 159
if...then statements, 153–154
interpreter, 148
parameters
expanding, 151–152
reading in, 151
positional parameters, 150–151
programming constructs, 153–159
sed command, 160
special characters, escaping, 149
telephone list example, 161–162
text
cutting, 159–160
deleting, 160
translating, 160
text manipulation, 159–161
tr command, 160
until...do loop, 158–159
variables, 149
while...do loop, 158–159
shell scripts, 147–148
shells file, 183
sidecar containers, 766
signals, 141
single-partition disks, 277–281
Slackware, 16
SLES (SUSE Linux Enterprise Server), 730
SMB (Server Message Block), 475
smb.conf file, 422–423
software
bounties, 21
dependent software, 224
free software, 12
installation, 221–222
system administrator and, 168
Nautilus, 43–45
open source software, 12
repositories, 223
security, 579–580
advisories, 580
package updates, 579–580
subscriptions, 20–21
upstream software providers, 228
Software window, 222–223
source code, UNIX and, 10
Spacewalk, RPM packages and, 246
special characters, escaping, 149
specialized storage, 5
Squid Proxy Server, 366
SSH (Secure Shell), 307
ssh command, 316
remote execution, 320–321
remote login, 318–320
sshd service, 718
starting, 317–318
at boot, 318
status, 317
Stallman, Richard M., 11
startup methods
init facility, 524–525
systemd facility, 525
statements, echo, 148
sticky bits, 103, 268–269
storage
cloud, 711, 713
configuring, 718–720
setup, 714
Cockpit, 301–303
specialized, 5, 213
volume groups, 274
stream editor (sed), 160
su command, 168, 175–176
subscriptions, 20–21
sudo command, 168, 632–633
sudo facility, 176–178
sudoers file, 177, 183
superuser, 167

SVID (System V Interface Definition), 10
swap partitions, 215
swap space, 4, 274
  troubleshooting and, 554–555
SYN (synchronize packet), 666
syntax, commands, 67–68
/sys directory, 95
sysstat package, sar command, 332–333
System Activity Reporter, 332–333. *See also* sar command
system administration, 167
  browser-based tools, 173
  Cockpit, 168, 169–171
  commands, 178–179
  configuration files, 179–185
  daemon processes
    apache, 185
    avahi, 185
    bin, 185
    chrony, 185
    lp, 185
    news, 186
    postfix, 185
    rpc, 186
  filesystems and, 168
  graphical tools, 172
  hardware
    checking, 187–189
    loadable modules, 191–193
    removable, 189–191
  log files, 183–184
    rsyslogd facility and, 184–185
  network interfaces, 169
  root user, 174
    via shell, 175–176
  security, 169
  servers, 169
  software installation and, 168
  system-config* tools, 171–173
  systemd journal, 183–184
  user accounts and, 168
system administrator, 167
system logging
  enabling, 326–331
  loghost, 330–331
  logwatch, 331–332
  message log file, 329
  rsyslogd daemon, 326–327
System Monitor, processes
  ending, 136
  killing, 136
  listing, 136–137
  priorities, 137
  stopping, 136
system space
  disk consumption, 335–336
  disk usage check, 334–335
  displaying, 334
System V, init facility, 524–525
  troubleshooting, 533
system-config* tools, 171
  system-config-authentication, 172
  system-config-bind, 172
  system-config-date, 172
  system-config-firewall, 172
  system-config-httpd, 172
  system-config-kickstart, 173
  system-config-language, 172
  system-config-nfs, 172
  system-config-printer, 172
  system-config-rootpassword, 172
  system-config-samba, 172
  system-config-selinux, 172
  system-config-services, 172
  system-config-users, 172
systemctl command, 381
systemd, 370–371, 525
  initialization, 377–384
  journal, 183–184
  service units, 378–380
  services
    new, 399–401
    reloading, 390–391
    restarting, 389–390
    starting, 389
    stopping, 389
  SysVinit, backward compatibility, 382–384
  target units, 377, 378, 381–382, 395
  troubleshooting, 538–541
  units, 377
systems monitoring, 580–581
  filesystem, 587
    rootkits, 590–595
    scanning, 589–590
    software package verification, 588
    virus detection, 590–595
  log files, 581

special commands, 582–583
/var/log directory, 581–582
user accounts, 584
bad passwords, 586–587
counterfeit accounts and privileges, 584–586
SysVinit, 370, 372
backward compatibility, 382–384
runlevels, default, 394–395
services
checking, 385–387
new, 396–398
persistent, 391–394
starting/stopping, 387–391

T

tar, 566
encryption/decryption, 604
tarballs, 16, 224–225
target units, 377, 395
targeted policy, 638–639
targets, 369
TCP Connect port scan, 666, 668
TCP Wrappers, 313
NFS access, 510
tcsh shell, 61, 65
TE (type enforcement), 637–638
telinit command, 374
telnet, 316
Terminal emulator, 63–64
Terminal window, 63
GNOME Terminal, 64
launching, 64
terminal windows, 62–63
test expressions, 154–156
text
adding, vi editor, 115–116
changing, vi editor, 117
commands, 79
copying, vi editor, 117
cutting, 159
deleting, 160
vi editor, 117
grep, 159
here text, 100
moving within, vi editor, 116–117
pasting, vi editor, 118
searching for, vi editor, 119–120
translating, 160
text editors
emacs, 114
jed, 114
joe, 114
kate, 114
kedit, 114
mcedit, 114
nano, 114
nedit, 114
vi, 113–119
arrow keys, 116–117
command mode, 115
command repeat, 118
cursor, 115
ex mode, 120
exiting, 118–119
input mode, 115
moving around in file, 119
text, 115–120
vim, 113–114
Thompson, Ken, 8
tilde (~), 97
/tmp directory, 95
top command, 134–135
Torvalds, Linus, 7, 13–14
touch command, 98–99
tr command, 160
training, 21
translating text, 160
troubleshooting
BIOS (Basic Input Output System), 526–527
boot order, 527–528
boot up and, 523–524
from firmware, 526–528
kernel startup, 532–541
startup methods, 524–525
GRUB 2 boot loader, 530–531
GRUB boot loader, 528–530
init system, 533
memory, 553–554
uncovering issues, 554–559
networking
incoming connections, 550–553
outgoing connections, 547–550
RAM and, 554–555
rc.sysinit, 533–534
in rescue mode, 559–561
runlevels, 534–538

troubleshooting *(continued)*
software packages, 542–545
RPM databases and caches, 545–546
`systemd` initialization, 538–541
`type` command, 71

U

Ubuntu, 19
downloading, 790–791
Ubuntu Software Center, 225
UDP port scan, 666
UEFI (Unified Extensible Firmware Interface), 526–528
`umask` value, 108–109
`umount` command, 299
units, `systemd`, 377
UNIX, 7–8
assembler, 9
commercial, Berkeley distribution, 9–10
filesystem, 8
input/output redirection, 8
laboratory, 10–11
portability, 8–9
print commands, 404
printers, remote, 413
published interfaces, 10
source code, 10
USL (UNIX System Laboratories), 10
unmounting filesystems, 520–521
untyped variables, 152–153
updates, Gentoo, 207
upgrades from scratch, 207
upstream software providers, 228
USB drive, 197
booting Linux from, 791–792
user accounts
centralizing, 269–270
Cockpit, 249–252
creating, 249–250
defaults, 255–257
deleting, 258–259
modifying, 257–258
security, 566
number of users, 567
root, access, 567
temporary account expiration, 567–568
unused, 568–569
system administrator and, 168
`useradd` command, 252–255
user interfaces, 4
`useradd` command
adding users, 252–255
defaults, setting, 255–257
`userdel` command, 258–259
`usermod` command, 257–258
username, completion, 75
USL (UNIX System Laboratories), 10
`/usr` directory, 95
utilities
administrative, 4–5
backup, 566
`cracklib`, 571
`iptables`, 673–674, 678–688
`nmap`, 665–672
programming, 5
UTS namespace, 695

V

`/var` directory, 95
variables
command output, 149
completion, 75
environment variables, 81, 82–83
adding to shell, 87
PATH, 70
expanding, 80–81
shell, 81
shell scripts, 149
untyped, 152–153
Verizon, 9
Very Secure FTP daemon, 309
VFAT, 275
`vi` editor, 85, 113–114
arrow keys, 116–117
command mode, 115
commands, repeating, 118
cursor, 115
`ex` mode, 120
exiting, 118–119
input mode, 115
moving around in file, 119
text
adding, 115–116
changing, 117
copying, 117
deleting, 117
moving in, 116–117

pasting, 118
searching for, 119–120
video, boot options, 210
vim editor, 113–114
vimtutor, 177
virsh command, 711
virt-install command, 720–721
virt-manager, 711, 717, 720–724, 722
virt-manager command, 714
virtual consoles, 65
Virtual Machine Manager, 711, 714
starting, 722
virtual memory, 556
VirtualBox, 209
virtualization, 5, 709
Linux installation, 209
viewer, 714
virt-viewer command, 714
virus detection, 590
intrusion detection, 592–595
monitoring for rootkits, 591–592
monitoring for viruses, 591
virus signature, 591
VMs (virtual machines), 693–694, 709, 713. *See also* hypervisor
connections, 722
creating, 720–724
images, 721
installing, 724
managing, 724–725
migrating, 725–727
network bridge and, 721
system, viewing, 724–725
VMware, 209
VNC installations, boot options, 212
volume groups, 274
VPN (virtual private network), 339
vsftpd, 309, 458–461
file permissions, 465
installing, 457–458
setup, 468–469
VSZ (virtual set size), 133

W

web server, 309
Apache HTTPD, 427–428
installing, 431
who am i command, 65
wildcards, 505
Winbind, 270
window manager, 29
windows
graphical, 168
terminal windows, 62–63
Windows (SMB) printer, 414–415
Windows-based filesystems, 95
wired networks, 339
wireless networks, 339
worker nodes, 710
Write permissions, 106

X

X Window System, 28
background, 28
clients, 28
servers, 28
window manager, 29
Xen, 5, 209, 710
Xfce, 29
xinetd.conf file, 183

Y–Z

YUM (YellowDog Update Modified)
DNF (Dandified YUM), 229
packages
groups, updating, 239–240
installing, 236–237
maintenance, 240
removing, 236–237
searching for, 234–235
updating, 238
RPM download, 241
third-party software repositories, 233
yum command, syntax, 229–232
yum cache, 546
yum command, 229–241