Covers Red Hat Enterprise Linux 8
Covers ALL Official Exam Objectives
Great for Self-Study and In-Class Training
81 Step-by-Step Exercises
70 Do-It-Yourself Challenge Labs
375 Check Your Understanding Questions & Answers
4 Sample RHCSA Exams (80 Tasks)
21 Chapters

Learn by doing for success

RHCSA

RED HAT ENTERPRISE LINUX 8

TRAINING AND EXAM PREPARATION GUIDE (EX200)

FIRST EDITION

ASGHAR GHORI

RHCSA®
Red Hat® Enterprise Linux® 8

Training
and
Exam Preparation
Guide

Exam Code
EX200

First Edition
January 2020

Asghar Ghori

1246 Heil Quaker Blvd., La Vergne, TN USA 37086
Chapter House, Pitfield, Kiln Farm, Milton Keynes, UK MK11 3LW
Unit A1/A3, 7 Janine Street, Scoresby, Victoria 3179, Australia
www.ingramspark.com

Technical Reviewers: Many of my students and colleagues
Editors: FirstEditing.com and Zainab Ghori
Cover Design: Nid n Nad Graphics Printing Inc.
Printers and Distributors: IngramSpark Inc.

Copyright © 2020 Asghar Ghori
All rights reserved.
No portion of this book may be stored in a retrieval system, transmitted, or reproduced in any form, including but not limited to photocopying or other recording, without the express written consent of the author.

Printed in the USA, Canada, UK, France, Germany, Italy, Spain, and Australia.

ISBN: **978-1-7750621-2-7**
ISBN: **978-1-7750621-3-4 (e-book)**
Library of Congress Control Number: **2019920023**

To order in bulk at special quantity discounts for sales promotions or for use in training programs, please contact the author at *asghar_ghori2002@yahoo.com*

The following are registered trademarks in the U.S. and other countries:

Red Hat® is a registered trademark of Red Hat, Inc.

RHCSA® is a registered trademark of Red Hat, Inc.

Linux® is a registered trademark of Linus Torvalds.

Oracle® is a registered trademark of Oracle Corporation, Inc.

UNIX® is a registered trademark of The Open Group.

Microsoft® and Windows® are US registered trademarks of Microsoft Corporation.

Intel® is the trademark or registered trademark of Intel Corporation or its subsidiaries.

All other trademarks, registered trademarks, or logos used in this book are the property of their respective owners.

The author has made his best efforts to prepare this book. The contents are based on Red Hat® Enterprise Linux® version 8. The author makes no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accepts no liability whatsoever including but not limited to merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or allegedly caused directly or indirectly from this material.

This book is not a replacement for the official RHCSA training courses offered by Red Hat, Inc. However, it may be used to prepare for the Red Hat Certified System Administrator (RHCSA) exam, EX200, based on Red Hat® Enterprise Linux® version 8. Neither author nor publisher warrants that use of this publication will ensure passing the relevant exams or that the information contained herein is endorsed by Red Hat, Inc.

Preface

Red Hat® had revised their EX200 exam for the RHCSA certification soon after the release of Red Hat® Enterprise Linux® version 8 in April 2019. As usual, these exams are performance-based and present scenarios that are to be accomplished on live systems within a stipulated time. This book is written to provide necessary coverage of theoretical and practical information to help you pass the exam. Furthermore, this book may be used for classroom trainings and as a deskside reference.

Keeping in mind the hands-on nature of the exam, I have included a multitude of step-by-step exercises and Do-It-Yourself (DIY) challenge labs throughout the book. Chapter 01 details the infrastructure that is needed to practice the procedures and perform the labs. It also outlines how to obtain a copy of RHEL 8.

I suggest you study the material presented in each chapter thoroughly before proceeding to the relevant hands-on exercise(s). I have provided several review questions with answers at the end of each chapter. Take the quiz and then attempt the DIY challenge labs offered thereafter. I have not furnished solutions to these labs intentionally, as I am confident that the knowledge and skills you will have gained by that time will be sufficient to accomplish the labs on your own; and, in essence, this is what I want you to eventually get at. Once you have read and understood the material, performed exercises, completed review questions, and accomplished DIY challenge labs, take time to attempt the sample RHCSA exams provided in appendices.

While performing exercises and labs, if a command does not produce the published result, I advise that you check the message the command has generated and consult relevant

log files. Minor issues, such as a wrong path or typo, prevent commands from executing correctly. Sometimes, syntax errors in command constructs could result in execution failures. You might have to make modifications to your command in order to run it as expected. RHEL manual pages prove helpful in comprehending commands and their syntaxes.

There are four areas I suggest you focus on in order to develop expertise with RHEL, as well as to prepare for the exams: 1) grasping concepts; 2) mastering implementation procedures, exercises, and labs; 3) learning commands, understanding configuration files, and knowing service processes; and 4) being able to troubleshoot and resolve problems. An excellent knowledge of commands and key options, and the files they update should also be developed along with what processes handle which services, and so on. This will help you obtain a greater overall understanding of what exactly happens in the background when a command is executed. Debugging becomes easier when concepts are clear and working knowledge is solid.

I maintain *www.nixeducation.com* where I add errata, additional certification information, helpful videos on Linux concepts and administration topics, and links to other useful resources. I encourage you to visit this website.

To conclude, I would like to request for your constructive feedback sent to my personal email *asghar_ghori2002@yahoo.com* about any grammatical or technical errors or mistakes in the book, as well as any additional suggestions. Try to be as specific as possible in your description. Improvement is a continuous process, and I believe your feedback will help me continue to deliver quality books to you.

Good luck in your endeavors.

Asghar Ghori / January 2020 / Toronto, Canada

Acknowledgments

I am grateful to God who enabled me to write this book successfully.

I would like to acknowledge the valuable feedback my students, friends, and colleagues provided on my previous RHCSA and RHCE guides. I am thankful to them for making this book better in all respects.

I recognize the constructive and positive feedback I had received from the readers of my previous publications. I have used their comments toward the improvement of this edition.

I would like to express my special thanks to my wife, daughters, and sons, who endured my mental absence while writing this book. I could not have accomplished this project without their continuous support.

Lastly, I would like to offer my very special tributes to my deceased parents and sisters.

Asghar Ghori

About the Author

Asghar Ghori is a seasoned Linux/Cloud/DevOps consultant, trainer, and author. As a consultant, his experience ranges from deployment, support, and administration to solution architecture, design, and consulting; as a trainer, he has designed and delivered numerous training programs; and as a published author, he has eight books on UNIX and Linux (Red Hat Enterprise Linux and CompTIA Linux+) to his credit.

Asghar holds a BS in Engineering. He is RHCE, RHCSA, HP CSA, HP CSE, Oracle SCSA, IBM Certified Specialist for AIX, and CNE, with ITIL and PMP certifications. He is AWS CLP, SA, DA, and SysOps Administrator, as well as Azure Solution Architect Expert.

Asghar Ghori lives in a small town near Toronto, Ontario, Canada with his wife and children, and can be reached via email asghar_ghori2002@yahoo.com or LinkedIn.

Other publications of Asghar are:

1. CompTIA Linux+/LPIC-1: Training and Exam Preparation Guide (Exam Codes: LX0-103/101-400 and LX0-104/102-400) (ISBN: 978-1775062103), published 2017

2. RHCSA & RHCE Red Hat Enterprise Linux 7: Training and Exam Preparation Guide (EX200 and EX300) (ISBN: 978-1495148200) (RHEL version 7), published 2015

3. Red Hat Certified System Administrator & Engineer: Training Guide and a Quick Deskside Reference (ISBN: 978-1467549400) (RHEL version 6), published 2012

4. Red Hat Certified Technician & Engineer (RHCT and RHCE) Training Guide and Administrator's Reference (ISBN: 978-1615844302) (RHEL version 5), published 2009

5. HP-UX: HP Certified Systems Administrator, Exam HP0-A01, Training Guide and Administrator's Reference (ISBN: 978-1606436547) (HP-UX 11iv3), published 2008

6. HP Certified Systems Administrator, Exam HP0-095, Training Guide and Administrator's Reference (ISBN: 978-1424342310) (HP-UX 11iv2 and 11iv3), published 2007

7. Certified System Administrator for HP-UX: Study Guide and Administrator's Reference (ISBN: 978-1419645938) (HP-UX 11iv1), published 2006

Conventions Used in this Book

The following typographic and other conventions are used in this book:

Book Antiqua Italic 10 pt. is used in text paragraphs to introduce new terms. For example:

> “Red Hat renamed the Red Hat Linux operating system series *Red Hat Enterprise Linux* (RHEL) in 2003.”

Times Roman Italic 10 pt. is used in text paragraphs to highlight names of files, directories, commands, daemons, users, groups, hosts, domains, and URLs. This font also highlights file and directory paths. For example:

> “To go directly from */etc* to a subdirectory *dir1* under *user1*’s home directory, create *dir1*, as . . . .”

Times New Roman 9 pt. is used to segregate command output, script/file contents, and information expected to be entered in configuration files from the surrounding text. It is also used in tables, index, and side notes.

Times Roman Bold 10 pt. is used to highlight commands and command line arguments that the user is expected to type and execute at the command prompt. For example:

> [user1@server1 ~]$ **ls -lt**

There are two white spaces between parts of a typed command for the sake of clarity in text.

There are hundreds of screenshots that show commands and output. They are images taken directly from the Linux terminal screen.

All headings and sub-headings are in **California FB** font, and are bolded.

Ctrl+x key sequence implies that you hold down the Ctrl key and then press the other key. Courier New font is used to highlight such combinations. This font is also used to identify keystrokes, such as Enter and Esc.

. . . . . Dotted lines represent truncated command output.

About the RHCSA Exam (EX200) on RHEL 8

The Red Hat Certified System Administrator (RHCSA) certification exam is a performance-based hands-on exam designed for IT professionals. This exam is presented in electronic format on a live desktop computer running Red Hat Enterprise Linux 8. This desktop computer will have two RHEL 8-based virtual machines to accomplish the exam tasks. During the exam, the candidates will not have access to the Internet, or printed or electronic material except for what comes standard with RHEL 8. The official exam objectives (57 in total as of January 8, 2020) are listed at *http://www.redhat.com/training/courses/ex200/examobjective*. Visit the URL for up-to-date and in-depth information. The exam objectives are covered in detail in the chapters throughout the book. An enumerated list of exam objectives is presented below along with the chapter number where the objective is discussed.

Understand and Use Essential Tools

1. Access a shell prompt and issue commands with correct syntax (chapter 2)
2. Use input-output redirection (>, >>, |, 2>, etc) (chapter 7)
3. Use grep and regular expressions to analyze text (chapter 7)
4. Access remote systems using ssh (chapter 19)
5. Log in and switch users in multi-user targets (chapter 6)
6. Archive, compress, unpack, and uncompress files using tar, star, gzip, and bzip2 (chapter 3)
7. Create and edit text files (chapter 3)
8. Create, delete, copy, and move files and directories (chapter 3)
9. Create hard and soft links (chapter 3)
10. List, set, and change standard ugo/rwx permissions (chapter 4)
11. Locate, read, and use system documentation including man, info, and files in /usr/share/doc (chapter 2)

Operate Running Systems

12. Boot, reboot, and shut down a system normally (chapter 12)
13. Boot systems into different targets manually (chapter 12)
14. Interrupt the boot process in order to gain access to a system (chapter 11)
15. Identify CPU/memory intensive processes and kill processes (chapter 8)
16. Adjust process scheduling (chapter 8)
17. Manage tuning profiles (chapter 12)
18. Locate and interpret system log files and journals (chapter 12)
19. Preserve system journals (chapter 12)
20. Start, stop, and check the status of network services (chapter 12)
21. Securely transfer files between systems (chapter 19)

Configure Local Storage

22. List, create, and delete partitions on MBR and GPT disks (chapter 13)
23. Create and remove physical volumes (chapter 14)
24. Assign physical volumes to volume groups (chapter 14)
25. Create and delete logical volumes (chapter 14)
26. Configure systems to mount file systems at boot by Universally Unique ID (UUID) or label (chapter 15)
27. Add new partitions and logical volumes, and swap to a system non-destructively (chapters 14 and 15)

Create and Configure File Systems

28. Create, mount, unmount, and use vfat, ext4, and xfs file systems (chapter 15)
29. Mount and unmount network file systems using NFS (chapter 16)
30. Extend existing logical volumes (chapters 14 and 15)
31. Create and configure set-GID directories for collaboration (chapter 4)
32. Configure disk compression (chapter 13)
33. Manage layered storage (chapter 14)
34. Diagnose and correct file permission problems (chapter 4)

Deploy, Configure, and Maintain Systems

35. Schedule tasks using at and cron (chapter 8)
36. Start and stop services and configure services to start automatically at boot (chapter 12)
37. Configure systems to boot into a specific target automatically (chapter 12)
38. Configure time service clients (chapter 18)

39. Install and update software packages from Red Hat Network, a remote repository, or from the local file system (chapter 9 and 10)
40. Work with package module streams (chapter 10)
41. Modify the system bootloader (chapter 11)

Manage Basic Networking

42. Configure IPv4 and IPv6 addresses (chapter 17)
43. Configure hostname resolution (chapter 18)
44. Configure network services to start automatically at boot (chapter 12)
45. Restrict network access using firewall-cmd/firewall (chapter 20)

Manage Users and Groups

46. Create, delete, and modify local user accounts (chapter 5)
47. Change passwords and adjust password aging for local user accounts (chapter 5 and 6)
48. Create, delete, and modify local groups and group memberships (chapter 6)
49. Configure superuser access (chapter 6)

Manage Security

50. Configure firewall settings using firewall-cmd/firewalld (chapter 20)
51. Create and use file access control lists (chapter 4)
52. Configure key-based authentication for SSH (chapter 19)
53. Set enforcing and permissive modes for SELinux (chapter 21)
54. List and identify SELinux file and process context (chapter 21)
55. Restore default file contexts (chapter 21)
56. Use Boolean settings to modify system SELinux settings (chapter 21)
57. Diagnose and address routine SELinux policy violations (chapter 21)

Taking the Exam

1. Save time wherever possible, as time is of the essence during the exam
2. Make certain that any changes you make must survive system reboots
3. Use any text editor you feel comfortable with to modify text configuration files
4. Exam tasks are split into two groups and each group must be performed in its own assigned virtual machine
5. Inform the proctor right away if you encounter an issue with your exam system or the virtual machines
6. Exam is administered with no access to the Internet, electronic devices, or written material
7. Read each exam task carefully and understand it thoroughly before attempting it

Exam Fees and Registration Procedure

The fee for the RHCSA exam is US$400 (plus any applicable taxes), or equivalent in local currencies. To register, visit *http://www.redhat.com/training/courses/ex200/examobjective*, select your location and an exam format, and click Get Started to continue through the registration process. The RHCSA exam lasts for 2.5 hours.

About this Book

This book covers four major learning objectives: 1) a self-study guide for exam aspirants who intend to pass the RHCSA 8 exam, 2) an in-class training guide for college students, 3) an on-the-job reference for administrators, programmers, and managers, and 4) an easy-to-understand guide for novice and non-RHEL administrators who wish to learn RHEL from scratch.

This book has 21 chapters that are organized logically, keeping in mind the four learning objectives mentioned above. It covers the topics on local RHEL 8 installation; initial interaction with the system and basic commands; compression and archiving; file editing and manipulation; standard and special permissions; file searching and access controls; user monitoring and authentication files; users, groups, and password aging; bash shell features and startup files; processes and task scheduling; basic and advanced software administration techniques; system boot process and bootloader; kernel management and system initialization; logging and system tuning; basic and advanced storage management tools and solutions; local and remote file systems and swap regions; network device and connection configuration; time synchronization and hostname resolution; the secure shell service; and firewall and SELinux controls.

Each chapter highlights the major topics and relevant exam objectives at the beginning, and ends with review questions & answers and Do-It-Yourself challenge labs. Throughout the book, figures, tables, screen shots, examples, and exam tips have been furnished to support explanation and exam

preparation. This book includes four sample exams for RHCSA, which are expected to be done using the knowledge and skills attained from reading the material and practicing the exercises and challenge labs. The labs and the sample exams include references to relevant topics and/or exercises.

TABLE OF CONTENTS

Preface
Acknowledgments
About the Author
Conventions Used in this Book
About the RHCSA Exam (EX200) on RHEL 8
About this Book

01. Local Installation

A Quick Look at Linux Development
- Linux History in a Nutshell
- Linux from Red Hat

Lab Infrastructure for Practice
- What is Needed for the Lab?
- The RHEL Installer Program
- Where Do Installation Logs Go?
- Virtual Console Screens

Exercise 1-1: Download and Install VirtualBox Software, and Create a Virtual Machine
- Downloading and Installing VirtualBox
- Creating a Virtual Machine

Exercise 1-2: Download and Install RHEL 8
- Downloading RHEL 8 ISO Image
- Attaching RHEL 8 ISO Image to the Virtual Machine
- Launching the Installer

Adding Support for Keyboards and Languages
Configuring Time & Date
Choosing an Installation Source
Selecting Software to be Installed
Configuring Installation Destination
Configuring Network and Hostname
Beginning Installation
Setting root Password and Creating a User Account
Concluding Installation
Changing Default Boot Order
Performing Post-Installation Tasks
Logging In and Out
Logging In for the First Time
Logging Out
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 1-1: Build RHEL8-VM2 (server2)

02. Initial Interaction with the System
Linux Graphical Environment
Display/Login Manager
Desktop Environment
Linux Directory Structure and File Systems
Top-Level Directories
File System Categories
The Root File System (/), Disk-Based
The Boot File System (/boot) – Disk-Based
The Home Directory (/home)
The Optional Directory (/opt)
The UNIX System Resources Directory (/usr)
The Variable Directory (/var)

The Temporary Directory (/tmp)
The Devices File System (/dev), Virtual
The Procfs File System (/proc), Virtual
The Runtime File System (/run), Virtual
The System File System (/sys), Virtual
Viewing Directory Hierarchy
Basic System Commands
Starting a GNOME Terminal Session
Understanding the Command Mechanics
Listing Files and Directories
Printing Working Directory
Navigating Directories
Identifying Terminal Device File
Inspecting System’s Uptime and Processor Load
Clearing the Screen
Determining Command Path
Viewing System Information
Viewing CPU Specs
Getting Help
Accessing Manual Pages
Headings in the Manual
Manual Sections
Searching by Keyword
Exposing Short Description
The info and pinfo Commands
Documentation in the /usr/share/doc Directory
Red Hat Enterprise Linux 8 Documentation
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 2-1: Navigate Linux Directory Tree
Lab 2-2: Miscellaneous Tasks
Lab 2-3: Identify System and Kernel Information

Lab 2-4: Use Help

03. Basic File Management

Common File Types

Regular Files
Directory Files
Block and Character Special Device Files
Symbolic Links

Compression and Archiving

Using gzip and gunzip
Using bzip2 and bunzip2
Differences between gzip and bzip2
Using tar
Exercise 3-1: Create Compressed Archives

File Editing

Modes of Operation
Starting vim
Inserting text
Navigating within vim
Deleting Text
Undoing and Repeating
Searching for Text
Replacing Text
Copying, Moving, and Pasting Text
Changing Text
Saving and Quitting vim

File and Directory Operations

Creating Files and Directories
Displaying File Contents
Counting Words, Lines, and Characters in Text Files
Copying Files and Directories
Moving and Renaming Files and Directories
Removing Files and Directories

File Linking

Hard Link
Soft Link
Differences between Copying and Linking
Exercise 3-2: Create and Manage Hard Links
Exercise 3-3: Create and Manage Soft Links
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 3-1: Archive, List, and Restore Files
Lab 3-2: Practice the vim Editor
Lab 3-3: File and Directory Operations

04. Advanced File Management
File and Directory Access Permissions
Determining Access Permissions
Permission Classes
Permission Types
Permission Modes
Modifying Access Permission Bits
Exercise 4-1: Modify Permission Bits Using Symbolic Form
Exercise 4-2: Modify Permission Bits Using Octal Form
Default Permissions
Calculating Default Permissions
Special File Permissions
The setuid Bit on Binary Executable Files
Exercise 4-3: Test the Effect of setuid Bit on Executable Files
The setgid Bit on Binary Executable Files
Exercise 4-4: Test the Effect of setgid Bit on Executable Files
The setgid Bit on Shared Directories

Exercise 4-5: Set up Shared Directory for Group Collaboration
The Sticky Bit on Public and Shared Writable Directories
Exercise 4-6: Test the Effect of Sticky Bit
File Searching
Using the find Command
Using find with -exec and -ok Flags
Using the locate Command
Access Control Lists (ACLs)
ACL Management Commands
The getfacl Command
The setfacl Command
The Role of the mask Value
Exercise 4-7: Identify, Apply, and Erase Access ACLs
Default ACLs
Exercise 4-8: Apply, Identify, and Erase Default ACLs
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 4-1: Manipulate File Permissions
Lab 4-2: Configure Group Collaboration and Prevent File Deletion
Lab 4-3: Find Files
Lab 4-4: Find Files Using Different Criteria
Lab 4-5: Apply ACL Settings

05. Basic User Management
User Login Activity and Information
Listing Logged-In Users
Inspecting History of Successful Login Attempts and System Reboots

Viewing History of Failed User Login Attempts
Reporting Recent User Login Attempts
Examining User and Group Information
Local User Authentication Files
The passwd File
The shadow File
The group File
The gshadow File
The useradd and login.defs Configuration Files
User Account Management
The useradd, usermod, and userdel Commands
Exercise 5-1: Create a User Account with Default Attributes
Exercise 5-2: Create a User Account with Custom Values
Exercise 5-3: Modify and Delete a User Account
No-Login (Non-Interactive) User Account
Exercise 5-4: Create a User Account with No-Login Access
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 5-1: Check User Login Attempts
Lab 5-2: Verify User and Group Identity
Lab 5-3: Create Users
Lab 5-4: Create User with Non-Interactive Shell

06. Advanced User Management
Password Aging and its Management
The chage Command

Exercise 6-1: Set and Confirm Password Aging with chage
The passwd Command
Exercise 6-2: Set and Confirm Password Aging with passwd
The usermod Command
Exercise 6-3: Lock and Unlock a User Account with usermod and passwd
Linux Groups and their Management
The groupadd, groupmod, and groupdel Commands
Exercise 6-4: Create a Group and Add Members
Exercise 6-5: Modify and Delete a Group Account
Substituting Users and Doing as Superuser
Substituting (or Switching) Users
Doing as Superuser (or Doing as Substitute User)
Owning User and Owning Group
Exercise 6-6: Modify File Owner and Owning Group
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 6-1: Create User and Configure Password Aging
Lab 6-2: Lock and Unlock User
Lab 6-3: Modify Group
Lab 6-4: Configure sudo Access
Lab 6-5: Modify Owning User and Group

07. The Bash Shell

The Bourne-Again Shell
Internal and External Shell Commands
Shell and Environment Variables
Setting and Unsetting Variables
Command and Variable Substitutions

Exercise 7-1: Modify Primary Command Prompt
Input, Output, and Error Redirections
History Substitution
Editing at the Command Line
Tab Completion
Tilde Substitution
Alias Substitution
Metacharacters and Wildcard Characters
Piping Output of One Command as Input to Another
Quoting Mechanisms
Regular Expressions
Running and Controlling Jobs in Foreground and Background
Shell Startup Files
System-wide Shell Startup Files
Per-user Shell Startup Files
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 7-1: Customize the Command Prompt
Lab 7-2: Redirect the Standard Input, Output, and Error

08. Linux Processes and Task Scheduling
Processes and Priorities
Process States
Viewing and Monitoring Processes with ps
Viewing and Monitoring Processes with top
Listing a Specific Process
Listing Processes by User and Group Ownership
Understanding Process Niceness
Viewing and Changing Process Niceness
Renicing a Running Process

Controlling Processes with Signals
Job Scheduling
Controlling User Access
Scheduler Log File
Using at
Exercise 8-1: Submit, View, List, and Remove an at Job
Using crontab
Syntax of User Crontables
Exercise 8-2: Add, List, and Remove a Cron Job
Anacron
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 8-1: Nice and Renice a Process
Lab 8-2: Configure a User Crontab File

09. Basic Package Management
Package Overview
Packages and Packaging
Package Naming
Package Dependency
Package Database
Package Management Tools
Package Management with rpm
The rpm Command
Getting Ready to Use rpm
Querying Packages
Installing a Package
Upgrading a Package
Freshening a Package
Overwriting a Package

Removing a Package
Extracting Files from an Installable Package
Validating Package Integrity and Credibility
Viewing GPG Keys
Verifying Package Attributes
Exercise 9-1: Perform Package Management Tasks Using rpm
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 9-1: Install and Verify Packages
Lab 9-2: Query and Remove Packages

10. Advanced Package Management
Advanced Package Management Concepts
Package Groups
Application Streams and Modules
BaseOS Repository
AppStream Repository
Benefits of Segregation
Module Streams
Module Profiles
dnf/yum Repository
Software Management with dnf
dnf Configuration File
The dnf Command
Exercise 10-1: Configure Access to Pre-Built ISO Repositories
Individual Package Management
Listing Available and Installed Packages
Installing and Updating Packages
Exhibiting Package Information

Removing Packages
Exercise 10-2: Manipulate Individual Packages
Determining Provider and Searching Package Metadata
Package Group Management
Listing Available and Installed Package Groups
Installing and Updating Package Groups
Removing Package Groups
Exercise 10-3: Manipulate Package Groups
Module Management
Listing Available and Installed Modules
Installing and Updating Modules
Displaying Module Information
Removing Modules
Exercise 10-4: Manipulate Modules
Switching Module Streams
Exercise 10-5: Install a Module from an Alternative Stream
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 10-1: Configure Access to RHEL 8 Repositories
Lab 10-2: Install and Manage Individual Packages
Lab 10-3: Install and Manage Package Groups
Lab 10-4: Install and Manage Modules
Lab 10-5: Switch Module Streams and Install Software

11. Boot Process, GRUB2, and the Linux Kernel

Linux Boot Process
The Firmware Phase (BIOS and UEFI)
The Bootloader Phase

The Kernel Phase
The Initialization Phase
The GRUB2 Bootloader
Interacting with GRUB2
Understanding GRUB2 Configuration Files
Booting into Specific Targets
Exercise 11-1: Reset the root User Password
The Linux Kernel
Kernel Packages
Analyzing Kernel Version
Understanding Kernel Directory Structure
Installing the Kernel
Exercise 11-2: Download and Install a New Kernel
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 11-1: Modify GRUB2 Timeout
Lab 11-2: Reset root User Password
Lab 11-3: Install New Kernel

12. System Initialization, Message Logging, and System Tuning
System Initialization and Service Management
Units
Targets
The systemctl Command
Listing and Viewing Units
Managing Service Units
Managing Target Units
System Logging
The Syslog Configuration File

Rotating Log Files
The Boot Log File
The System Log File
Logging Custom Messages
The systemd Journal
Retrieving and Viewing Messages
Preserving Journal Information
System Tuning
Tuning Profiles
The tuned-adm Command
Exercise 12-1: Manage Tuning Profiles
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 12-1: Modify Default Boot Target
Lab 12-2: Record Custom Alerts
Lab 12-3: Apply Tuning Profile

13. Basic Storage Partitioning
Storage Management Overview
Master Boot Record (MBR)
GUID Partition Table (GPT)
Disk Partitions
Storage Management Tools
Thin Provisioning
Adding Storage for Practice
Exercise 13-1: Add Required Storage to server2
MBR Storage Management with parted
Exercise 13-2: Create an MBR Partition
Exercise 13-3: Delete an MBR Partition
GPT Storage Management with gdisk
Exercise 13-4: Create a GPT Partition

Exercise 13-5: Delete a GPT Partition
Storage Optimization with Virtual Data Optimizer (VDO)
How VDO Conserves Storage Space
Creating and Managing VDO Volumes
Exercise 13-6: Install Software and Activate VDO
Exercise 13-7: Create a VDO Volume
Exercise 13-8: Delete a VDO Volume
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 13-1: Create and Remove Partitions with parted
Lab 13-2: Create and Remove Partitions with gdisk
Lab 13-3: Create and Delete VDO Volumes
Lab 13-4: Disable and Enable VDO Volume Features

14. Advanced Storage Partitioning
Logical Volume Manager (LVM)
Physical Volume
Volume Group
Physical Extent
Logical Volume
Logical Extent
LVM Operations and Commands
Exercise 14-1: Create a Physical Volume and Volume Group
Exercise 14-2: Create Logical Volumes
Exercise 14-3: Extend a Volume Group and a Logical Volume
Exercise 14-4: Rename, Reduce, Extend, and Remove Logical Volumes
Exercise 14-5: Reduce and Remove a Volume Group

Exercise 14-6: Uninitialize Physical Volumes
Stratis Volume-Managing File System
Stratis Management Operations and Command
Exercise 14-7: Install Software and Activate Stratis
Exercise 14-8: Create and Confirm a Pool and File System
Exercise 14-9: Expand and Rename a Pool and File System
Exercise 14-10: Destroy a File System and Pool
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 14-1: Create Volume Group and Logical Volumes
Lab 14-2: Expand Volume Group and Logical Volume
Lab 14-3: Reduce and Remove Logical Volumes
Lab 14-4: Remove Volume Group and Physical Volumes
Lab 14-5: Create Stratis Pool
Lab 14-6: Expand and Destroy Stratis Pool

15. Local File Systems and Swap
File Systems and File System Types
Extended File Systems
XFS File System
VFAT File System
ISO9660 File System
File System Management
File System Administration Commands
Mounting and Unmounting File Systems
Determining the UUID of a File System
Labeling a File System
Automatically Mounting a File System at Reboots

Monitoring File System Usage
Calculating Disk Usage
Exercise 15-1: Create and Mount Ext4, VFAT, and XFS File Systems in Partitions
Exercise 15-2: Create and Mount XFS File System in VDO Volume
Exercise 15-3: Create and Mount Ext4 and XFS File Systems in LVM Logical Volumes
Exercise 15-4: Resize Ext4 and XFS File Systems in LVM Logical Volumes
Exercise 15-5: Create, Mount, and Expand XFS File System in Stratis Volume
Swap and its Management
Determining Current Swap Usage
Prioritizing Swap Spaces
Swap Administration Commands
Exercise 15-6: Create and Activate Swap in Partition and Logical Volume
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 15-1: Create VFAT, Ext4, and XFS File Systems in Partitions and Mount Persistently
Lab 15-2: Create XFS File System in VDO Volume and Mount Persistently
Lab 15-3: Create Ext4 and XFS File Systems in LVM Logical Volumes and Mount Persistently
Lab 15-4: Extend Ext4 and XFS File Systems in LVM Logical Volumes
Lab 15-5: Create XFS File System in Stratis Volume and Mount Persistently
Lab 15-6: Create Swap in Partition and LVM Logical Volume and Activate Persistently

16. Remote File System

- Network File System
  - Benefits of Using NFS
  - NFS Versions
- NFS Server and Client Configuration
  - Exercise 16-1: Export Share on NFS Server
  - Exercise 16-2: Mount Share on NFS Client
- Auto File System (AutoFS)
  - Benefits of Using AutoFS
  - How AutoFS Works
  - AutoFS Configuration File
  - AutoFS Maps
  - Exercise 16-3: Access NFS Share Using Direct Map
  - Exercise 16-4: Access NFS Share Using Indirect Map
  - Automounting User Home Directories
  - Exercise 16-5: Automount User Home Directories Using Indirect Map
- Chapter Summary
- Check Your Understanding
- Answers to Check Your Understanding
- Do-It-Yourself Challenge Labs
  - Lab 16-1: Configure NFS Share and Automount with Direct Map
  - Lab 16-2: Automount NFS Share with Indirect Map

17. Networking, Network Devices, and Network Connections

- Networking Fundamentals
  - Hostname
  - Exercise 17-1: Change System Hostname
  - IPv4 Address
  - Network Classes
  - Subnetting

Subnet Mask
Classless Inter-Domain Routing (CIDR) Notation
Protocol
TCP and UDP Protocols
Well-Known Ports
ICMP Protocol
Ethernet Address
IPv6 Address
Major Differences between IPv4 and IPv6
Network Devices and Connections
Consistent Network Device Naming
Understanding Interface Connection Profile
Exercise 17-2: Add Network Devices to server10 and server20
Network Device and Connection Administration Tools
Exercise 17-3: Configure New Network Connection Manually
The NetworkManager Service
The nmcli Command
Exercise 17-4: Configure New Network Connection Using nmcli
Understanding Hosts Table
Testing Network Connectivity
Exercise 17-5: Update Hosts Table and Test Connectivity
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 17-1: Add New Interface and Configure Connection Profile with nmcli
Lab 17-2: Add New Interface and Configure Connection Profile Manually

18. Time Synchronization and Hostname Resolution

Time Synchronization

Time Sources
NTP Roles
Stratum Levels
Chrony Configuration File
Chrony Daemon and Command
Exercise 18-1: Configure NTP Client
Displaying and Setting System Date and Time

DNS and Name Resolution

DNS Name Space and Domains
DNS Roles
Understanding Resolver Configuration File
Performing Name Resolution with dig
Performing Name Resolution with host
Performing Name Resolution with nslookup
Performing Name Resolution with getent

Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs

Lab 18-1: Modify System Date and Time
Lab 18-2: Configure Chrony

19. The Secure Shell Service

The OpenSSH Service

Common Encryption Techniques
Authentication Methods
OpenSSH Protocol Version and Algorithms
OpenSSH Packages
OpenSSH Server Daemon and Client Commands
Server Configuration File

Client Configuration File
System Access and File Transfer
Exercise 19-1: Access RHEL System from Another RHEL System
Exercise 19-2: Access RHEL System from Windows
Exercise 19-3: Generate, Distribute, and Use SSH Keys
Executing Commands Remotely Using ssh
Copying Files Remotely Using scp
Transferring Files Remotely Using sftp
Synchronizing Files Remotely Using rsync
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 19-1: Establish Key-Based Authentication
Lab 19-2: Test the Effect of PermitRootLogin Directive

20. The Linux Firewall
Firewall Overview
Overview of firewalld
firewalld Zones
Zone Configuration Files
firewalld Services
Service Configuration Files
Firewall Management
The firewall-cmd Command
Querying the Operational Status of firewalld
Exercise 20-1: Add Services and Ports, and Manage Zones
Exercise 20-2: Remove Services and Ports, and Manage Zones
Exercise 20-3: Test the Effect of Firewall Rule

Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs
Lab 20-1: Add Service to Firewall
Lab 20-2: Add Port Range to Firewall

21. Security Enhanced Linux
Security Enhanced Linux
Terminology
SELinux Contexts for Users
SELinux Contexts for Processes
SELinux Contexts for Files
Copying, Moving, and Archiving Files with SELinux Contexts
SELinux Contexts for Ports
Domain Transitioning
SELinux Booleans
SELinux Administration
Management Commands
Viewing and Controlling SELinux Operational State
Querying Status
Exercise 21-1: Modify SELinux File Context
Exercise 21-2: Add and Apply File Context
Exercise 21-3: Add and Delete Network Ports
Exercise 21-4: Copy Files with and without Context
Exercise 21-5: View and Toggle SELinux Boolean Values
Monitoring and Analyzing SELinux Violations
Chapter Summary
Check Your Understanding
Answers to Check Your Understanding
Do-It-Yourself Challenge Labs

Lab 21-1: Disable and Enable the SELinux Operating Mode
Lab 21-2: Modify Context on Files
Lab 21-3: Add Network Port to Policy Database
Lab 21-4: Copy Files with and without Context
Lab 21-5: Flip SELinux Booleans

Appendix A: Sample RHCSA Exam 1

Appendix B: Sample RHCSA Exam 2

Appendix C: Sample RHCSA Exam 3

Appendix D: Sample RHCSA Exam 4

Glossary

Index

List of Figures

Figure 1-1 Lab Setup
Figure 1-2 VirtualBox Website
Figure 1-3 VirtualBox Download
Figure 1-4 VirtualBox Installation 1
Figure 1-5 VirtualBox Installation 2
Figure 1-6 VirtualBox Installation 3
Figure 1-7 VirtualBox Installation 4
Figure 1-8 VirtualBox Installation 5
Figure 1-9 VirtualBox Installation 6
Figure 1-10 Virtual Machine Creation 1
Figure 1-11 Virtual Machine Creation 2
Figure 1-12 Virtual Machine Creation 3
Figure 1-13 Virtual Machine Creation 4
Figure 1-14 Virtual Machine Creation 5
Figure 1-15 Virtual Machine Creation 6
Figure 1-16 Virtual Machine Creation 7
Figure 1-17 Virtual Machine Creation 8
Figure 1-18 Red Hat User Account Creation 1
Figure 1-19 Red Hat User Account Creation 2
Figure 1-20 Red Hat Developer Login
Figure 1-21 Attach ISO Image to VM
Figure 1-22 Boot Menu
Figure 1-23 Language Selection
Figure 1-24 Installation Summary | Main
Figure 1-25 Installation Summary | Time & Date
Figure 1-26 Installation Summary | Installation Source
Figure 1-27 Installation Summary | Software Selection
Figure 1-28 Installation Summary | Installation Destination
Figure 1-29 Installation Summary | Network & Hostname
Figure 1-30 Installation Summary | Network & Hostname | Configure
Figure 1-31 Installation Summary | Network & Hostname
Figure 1-32 Installation Summary | Begin Installation
Figure 1-33 Configuration | User Settings
Figure 1-34 Configuration | User Settings | Create User
Figure 1-35 Configuration | Finishing Installation
Figure 1-36 VirtualBox Manager | System | Boot Order
Figure 1-37 VirtualBox Manager | System | Boot Order | Alter
Figure 1-38 Initial Setup
Figure 1-39 Graphical Desktop | Sign-in Screen

Figure 1-40 First Time User Login
Figure 1-41 First Time User Login | Getting Started
Figure 1-42 GNOME Desktop Environment
Figure 1-43 GNOME Desktop Environment| Log Out
Figure 2-1 GNOME Display Manager
Figure 2-2 GNOME Desktop Environment
Figure 2-3 GNOME Desktop Environment | Activities
Figure 2-4 Linux Directory Structure
Figure 2-5 GNOME Desktop Environment | Activities | Terminal
Figure 2-6 Red Hat’s Webpage for RHEL 8 Documentation
Figure 3-1 Hard Link
Figure 3-2 Soft Link
Figure 4-1 Permission Weights
Figure 4-2 find Command Syntax
Figure 5-1 The passwd File
Figure 5-2 The shadow File
Figure 5-3 The group File
Figure 5-4 The gshadow File
Figure 8-1 Process State Transition
Figure 8-2 Syntax of Crontables
Figure 11-1 GRUB2 Menu
Figure 11-2 GRUB2 Kernel Edit
Figure 11-3 GRUB2 Commands
Figure 11-4 Anatomy of a Kernel Version
Figure 13-1 VirtualBox Interface
Figure 13-2 VirtualBox – Add Storage
Figure 13-3 VirtualBox – Adjust Disk Name and Size
Figure 13-4 VirtualBox – 7 New Disks Added
Figure 14-1 LVM Structure
Figure 14-2 Stratis Structure
Figure 16-1 NFS Server/Client
Figure 17-1 VirtualBox – Add Network Interface
Figure 18-1 NTP Stratum Levels
Figure 18-2 Sample DNS Hierarchy

List of Tables

Table 1-1 Installation Logs
Table 1-2 Installation Summary | Software Selection | Base Environments
Table 2-1 tree Command Options
Table 2-2 ls Command Options
Table 2-3 Navigating within Manual Pages
Table 2-4 Navigating within info Documentation
Table 3-1 tar Command Options
Table 3-2 tar with Compression Options
Table 3-3 Inserting Text
Table 3-4 Navigating within vim
Table 3-5 Deleting Text
Table 3-6 Undoing and Repeating
Table 3-7 Searching for Text
Table 3-8 Replacing Text
Table 3-9 Copying, Moving, and Pasting Text
Table 3-10 Changing Text
Table 3-11 Saving and Quitting vim
Table 3-12 Navigating with less and more
Table 3-13 wc Command Options
Table 3-14 Copying vs. Linking
Table 4-1 Octal Permission Notation
Table 4-2 setfacl Command Format for Access ACLs
Table 4-3 setfacl Command Switches
Table 5-1 login.defs File Directives
Table 5-2 useradd Command Options
Table 5-3 usermod Command Options
Table 6-1 chage Command Options
Table 6-2 passwd Command Options
Table 6-3 usermod Command Options for User Lock/Unlock
Table 6-4 groupadd Command Options
Table 7-1 Common Predefined Environment Variables
Table 7-2 Helpful Command Line Editing Shortcuts
Table 7-3 Predefined Aliases
Table 7-4 Job Control Commands and Control Sequences
Table 7-5 System-wide Startup Files
Table 7-6 Per-user Startup Files
Table 8-1 ps Command Output Description
Table 8-2 Control Signals
Table 8-3 User Access Restrictions to Scheduling Tools

Table 8-4 Crontable Syntax Explained
Table 9-1 rpm Command Query Options
Table 9-2 rpm Command Install/Remove/Verify Options
Table 9-3 Red Hat GPG Key Files
Table 9-4 Package Verification Codes
Table 9-5 File Type Codes
Table 10-1 Directive Settings in dnf.conf File
Table 10-2 dnf Subcommands for Packages and Repositories
Table 10-3 dnf Subcommands for Package Groups and Modules
Table 11-1 GRUB2 Default Settings
Table 11-2 Kernel Packages
Table 12-1 systemd Unit Types
Table 12-2 systemd Targets
Table 12-3 systemctl Subcommands
Table 12-4 Journal Data Storage Options
Table 12-5 Tuning Profiles
Table 13-1 Common parted Subcommands
Table 13-2 vdo Subcommands
Table 14-1 Common LVM Operations and Commands
Table 14-2 Common Stratis Subcommands
Table 15-1 File System Types
Table 15-2 File System Management Commands
Table 15-3 Common mount Command Options
Table 16-1 AutoFS Directives
Table 17-1 IPv4 vs IPv6
Table 17-2 Network Connection Configuration Directives
Table 17-3 Basic Network Management Tools
Table 17-4 Network Connection and Device Administration Tools
Table 18-1 Chrony Directives
Table 18-2 The Resolver Configuration File
Table 18-3 Name Service Source and Order Determination
Table 19-1 OpenSSH Client Tools
Table 19-2 OpenSSH Server Configuration File
Table 19-3 OpenSSH Client Configuration File
Table 20-1 firewalld Default Zones
Table 20-2 Common firewall-cmd Options
Table 21-1 SELinux Management Commands

Chapter 01

Local Installation

This chapter describes the following major topics:

- A quick look at Linux and Open Source
- Linux distribution from Red Hat
- Recommended lab setup for RHCSA exam preparation
- Overview of the installer program
- Where are installation messages stored?
- What are virtual console screens?
- Download and install VirtualBox
- Create virtual machine
- Download and install Red Hat Enterprise Linux 8 in virtual machine
- Execute post-installation configuration tasks
- Log in and out at the graphical console

RHCSA Objectives:

None, but this chapter sets up the foundation for learning and practicing the exam objectives for RHCSA

Linux is a free operating system and it has been in existence for almost three decades. Its source code is available to developers, amateurs, and general public for enhancements and customization. Red Hat Inc. modifies a copy of a selected version of Linux source code and introduces features, adds improvements, and fixes bugs. The company packages the updated version as a Linux distribution of their own for commercial purposes. This distribution is thoroughly tested to run smoothly and perform well on a wide range of computer hardware platforms. It is stable, robust, feature-rich, and is ready to host workload of any size.

Red Hat Enterprise Linux may be downloaded for learning, practicing, and preparing for the RHCSA exam. It is available as a single installable image file. A lab environment is necessary to practice the procedures to solidify the understanding of the concepts and tools learned. The install process requires careful planning to identify critical system configuration pieces prior to launching the installer program. Once the operating system is installed, users can log in at the console or over the network.

A Quick Look at Linux Development

Linux is a free computer operating system (OS) that is similar to the UNIX OS in terms of concepts, features, functionality, and stability. It is referred to as a UNIX-like operating system.

Linux powers an extensive range of computer hardware platforms, from laptop and desktop computers to massive mainframes and supercomputers. Linux also runs as the base OS on networking, storage, gaming, smart television, and mobile devices. Numerous vendors, including Red Hat, IBM, Canonical, Oracle, DXC Technology, Novell, and Dell, offer commercial support to Linux users worldwide.

Linux is the main alternative to proprietary UNIX and Windows operating systems because of its functionality, adaptability, portability, and cost-effectiveness. At present, over one hundred different Linux distributions are circulating from various vendors, organizations, non-profit groups, and individuals, though only a few of them are popular and widely recognized.

Linux is largely used in government agencies, corporate businesses, academic institutions, scientific organizations, as well as in home computers. Linux deployment and usage are constantly on the rise.

Linux History in a Nutshell

In 1984, Richard Stallman, an American software engineer, had a goal to create a completely free UNIX-compatible open source (non-proprietary) operating system. The initiative was called the GNU Project (*GNU's Not Unix*) and by 1991, significant software had been developed. The only critical piece missing was a core software component called *kernel* to drive and control the GNU software and to regulate its communication with the hardware.

Around the same time, Finnish computer science student Linus Torvalds developed a kernel and proclaimed its availability. The new kernel was named *Linux*, and it was gradually integrated with the GNU software to form what is now referred to as *GNU/Linux*, *Linux operating system*, or simply *Linux*.

Linux was released under the GNU *General Public License* (GPL). Initially written to run on Intel x86-based computers, the first version (0.01) was released in September 1991 with little more than 10,000 lines of code. In 1994, the first major release (1.0.0) was introduced, followed by a series of successive major and minor versions until the version 4.0 in 2015. Development and enhancements continued, and version 4.0 was followed by several stable versions. At the time of this writing, version 4.19, with its millions of lines of code, is the latest stable kernel.

The Linux kernel, and the operating system in general, has been enhanced with contributions from tens of thousands of software

programmers, amateurs, and organizations around the world into a large and complex system under GNU GPL, which provides public access to its source code free of charge and with full consent to amend, package, and redistribute.

Linux from Red Hat

Red Hat, Inc., founded in 1993, used the available Linux source code and created one of the first commercial Linux operating system distribution called *Red Hat Linux* (RHL). The company released the first version 1.0 in November 1994. Several versions followed until the last version in the series, Red Hat Linux 9 (later referred to as RHEL 3), based on kernel 2.4.20, was released in March 2003. Red Hat renamed their Red Hat Linux brand as *Red Hat Enterprise Linux* (RHEL) commencing 2003.

RHL was originally assembled and enhanced within the Red Hat company. In 2003, Red Hat sponsored and facilitated the *Fedora Project* and invited the user community to join hands in enhancing and updating the source code. This project served as the test bed for developing and testing new features and enabled Red Hat to include the improved code in successive versions of RHEL.

The Fedora distribution is completely free, while RHEL is commercial. RHEL 4 (based on kernel 2.6.9 and released in February 2005), RHEL 5 (based on kernel 2.6.18 and released in March 2007), RHEL 6 (based on kernel 2.6.32 and released in November 2010), RHEL 7 (based on kernel 3.10 and released in June 2014), and RHEL 8 (based on kernel 4.18 and released in May 2019) have been built using Fedora distributions 3, 6, 12, 13, 19, and 28, respectively.

RHEL 8 has been tested to run on bare-metal computer hardware, virtualized platforms, high-end graphics workstations, IBM Power little endian, IBM System Z, and in the cloud.

Lab Infrastructure for Practice

RHEL 8 is available as a free download from Red Hat for Intel and AMD processor machines. You will need to create a free Red Hat user account in order to download it. The downloaded image file can then be attached to a *Virtual Machine* (VM) as an ISO image, burned to a DVD to support installation on a physical computer, or placed on a remote server for network-based installations via HTTP, FTP, or NFS protocols.

An ISO image is a single file that represents the content of an entire DVD or CD.

Burning the image to a DVD and configuring a server for network-based installation are beyond the scope of this book. This chapter will focus on installing the operating system with an ISO image.

What is Needed for the Lab?

Throughout this book, there will be several discussions about system, network, and security, along with examples on how to implement and administer them. Each chapter will contain a number of exercises that will help you perform certain tasks and execute commands.

You'll need a laptop or a desktop computer with at least a dual-core processor, 8GB of physical memory, and 27GB of free storage space to run two virtual machines with required storage. If you want to use the static IP addresses on your home router, make sure that you keep a map between them, and the ones provided below to avoid any confusion. The computer must have hardware virtualization support enabled in the BIOS to allow for 64-bit OS installation. Here is a snapshot of what is needed and how it will be configured:

Base Operating System:	Windows 10 or MacOS 10.12 or higher
Hypervisor Software:	Oracle VirtualBox (VB) 5.2.24 or higher
Number of VMs:	2
vCPUs in each VM:	1
OS in each VM:	RHEL 8.0
VM1 (RHEL8-VM1):	*server1.example.com* with static IP 192.168.0.110,

	1024MB memory, 1x10GB virtual disk for OS, and one virtual network device. This VM will be built using the RHEL 8 ISO image. Exercises 1-1 and 1-2 will walk through the process of installation. In Chapter 17, “Networking, Network Devices, and Network Connections”, you will add another virtual network device and will rename this server to *server10.example.com*.
VM2 (RHEL8-VM2):	*server2.example.com* with static IP 192.168.0.120, 2048MB memory, 1x10GB virtual disk for OS, and one virtual network device. In Chapter 13, “Basic Storage Partitioning”, you will add 4x250MB data disks for parted and LVM exercises, 1x4GB data disk for VDO exercises, and 2x1GB data disks for Stratis exercises that are presented in chapters 13, 14, and 15. You wil build this VM using the RHEL 8 ISO image by referencing the steps outlined in Exercises 1-1 and 1- In Chapter 17, “Networking, Network Devices, and Network Connections”, you will add another virtual network device and will rename this server to *server20.example.com*.

The entire setup for the lab is shown in Figure 1-1.

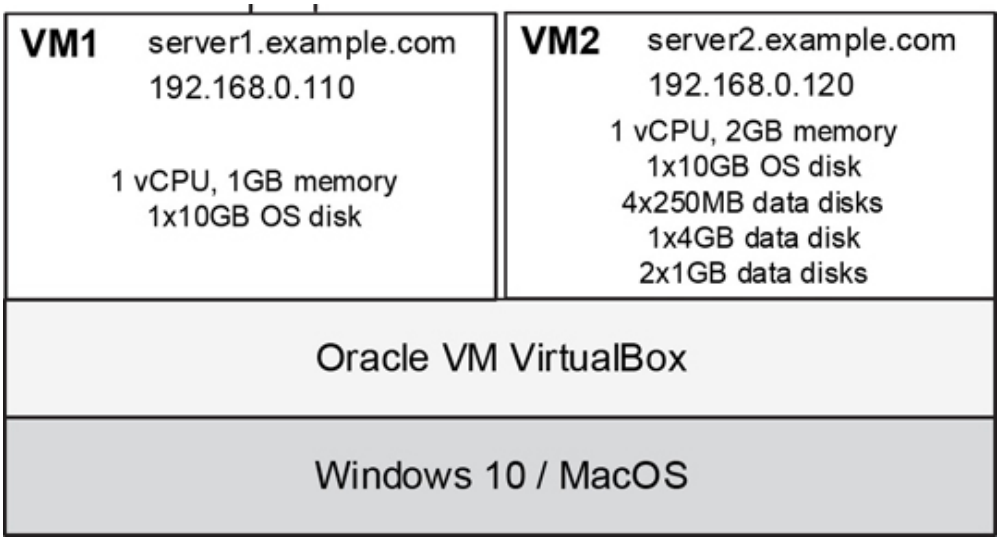


Figure 1-1 Lab Setup

You will install VirtualBox 5.2.24 (or higher) on Windows 10 or MacOS 10.12 (or higher). You may use VMware or other virtualization software as an alternative; however, all of the exercises and examples in this book reference VirtualBox (VB).

The RHEL Installer Program

The RHEL installer program is called *Anaconda*. There are several configuration options on the main screen that require modification before the installation process begins. Some of the questions are compulsory and must be answered appropriately while others are optional and may be skipped for post-installation setup.

The configuration can be done in any sequence that you prefer. You should have the minimum mandatory configuration data handy and be ready to enter it when prompted. Some of the key configuration items are language, keyboard type, time zone, disk partitioning, hostname/IP, software selection, root password, user information, and kdump.

Where Do Installation Logs Go?

There are plentiful log files created and updated as the installation progresses. These files record configuration and status information. You can view their contents after the installation has been completed to check how the installation proceeded. These files are described in Table 1-1.

File	Description
/root/anaconda-ks.cfg	Records the configuration entered
/root/install.log	Lists the packages being installed
/root/install.log.syslog	Stores general messages
/var/log/anaconda/anaconda.log	Contains informational, debug, and other general messages
/var/log/anaconda/ifcfg.log	Captures messages related to network interfaces and connections
/var/log/anaconda/journal.log	Stores messages generated by many services and components during system installation
/var/log/anaconda/packaging.log	Records messages generated by the yum and rpm commands during software installation
/var/log/anaconda/program.log	Captures messages generated by external programs
/var/log/anaconda/storage.log	Records messages generated by storage modules
/var/log/anaconda/syslog	Records messages related to the kernel
/var/log/anaconda/X.log	Stores X Window System information
/tmp/yum.log	Contains messages related to yum packages

Table 1-1 Installation Logs

Files in the */var/log/anaconda* directory are actually created and updated in the */tmp* directory during the installation; however, they are moved over once the installation is complete.

Virtual Console Screens

During the installation, there are six text-based virtual console screens available to monitor the process, view diagnostic messages, and discover and fix any issues encountered. The information displayed on the console screens is captured in the installation log files (Table 1-1). You can switch between screens by pressing a combination of keys as described below.

Console 1 (Ctrl+Alt+F1): This is the main screen. Before Anaconda begins, you will select a language to use during installation, and then it will switch the default console to the sixth screen (Console 6).

Console 2 (Ctrl+Alt+F2): The screen displays the shell interface to run commands as the *root* user.

Console 3 (Ctrl+Alt+F3): This screen displays installation messages and stores them in */tmp/anaconda.log* file. This file also captures information on detected hardware, in addition to other data.

Console 4 (Ctrl+Alt+F4): This screen shows storage messages and records them in */tmp/storage.log* file.

Console 5 (Ctrl+Alt+F5): This screen displays program messages and logs them to */tmp/program.log* file.

Console 6 (Ctrl+Alt+F6): This is the default graphical configuration and installation console screen.

Exercise 1-1: Download and Install VirtualBox Software, and Create a Virtual Machine

In this exercise, you will download and install VirtualBox software. You will create a virtual machine to set up the foundation to install RHEL 8 for the next exercise.

> **EXAM TIP:** Downloading and installing VirtualBox software and creating a virtual machine are not part of the exam objectives. These tasks have been included here only to support readers with building their own lab environment for practice.

Downloading and Installing VirtualBox

VirtualBox is available for free download and use. At the time of this writing, the latest version is 6.0; however, you can use any previous 5.x or a future version. Here is a quick guide on how to download and install the current version of VirtualBox on a Windows 10 computer.

1. Go to *www.virtualbox.org* (Figure 1-2) and click “Download VirtualBox 6.0”.

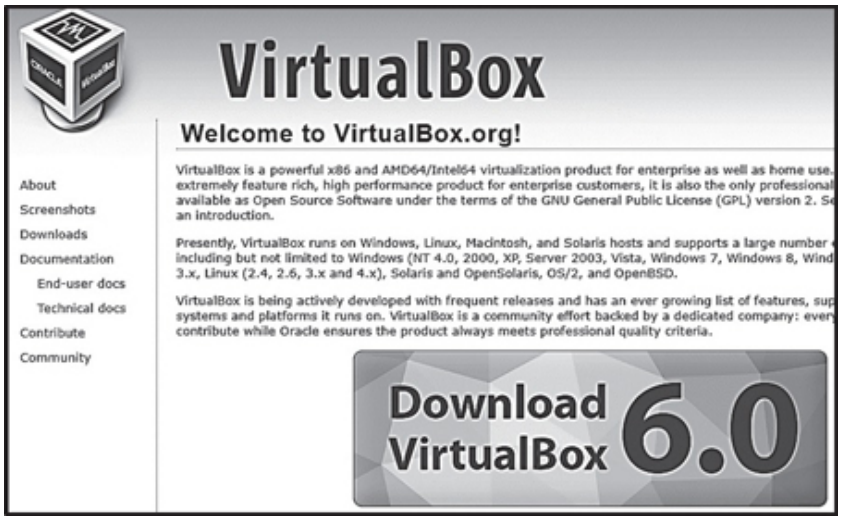


Figure 1-2 VirtualBox Website

2. On the next screen, click on “Windows hosts”. This will start the download to your computer.

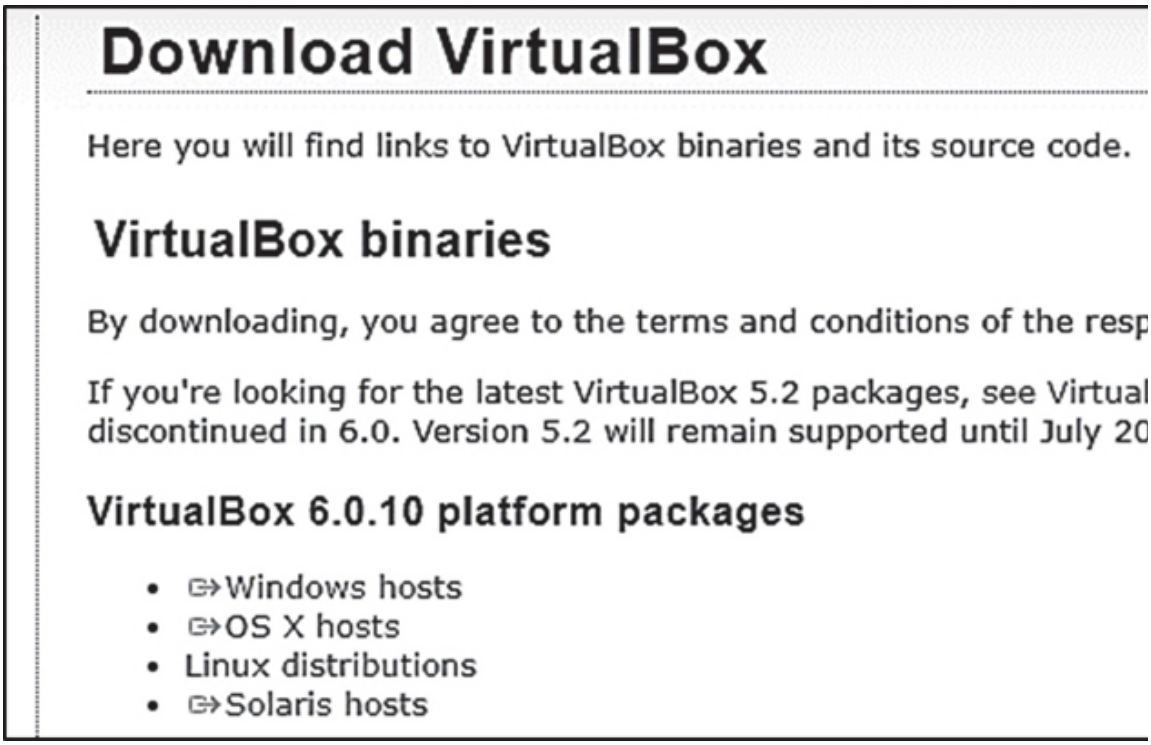


Figure 1-3 VirtualBox Download

The software is now available on your computer.

3. Double-click on the VirtualBox binary to start the installation. Click Next to proceed on the first screen that appears.

Figure 1-4 VirtualBox Installation 1

4. If needed, choose a different location on the disk to install VirtualBox. Click Next to continue.

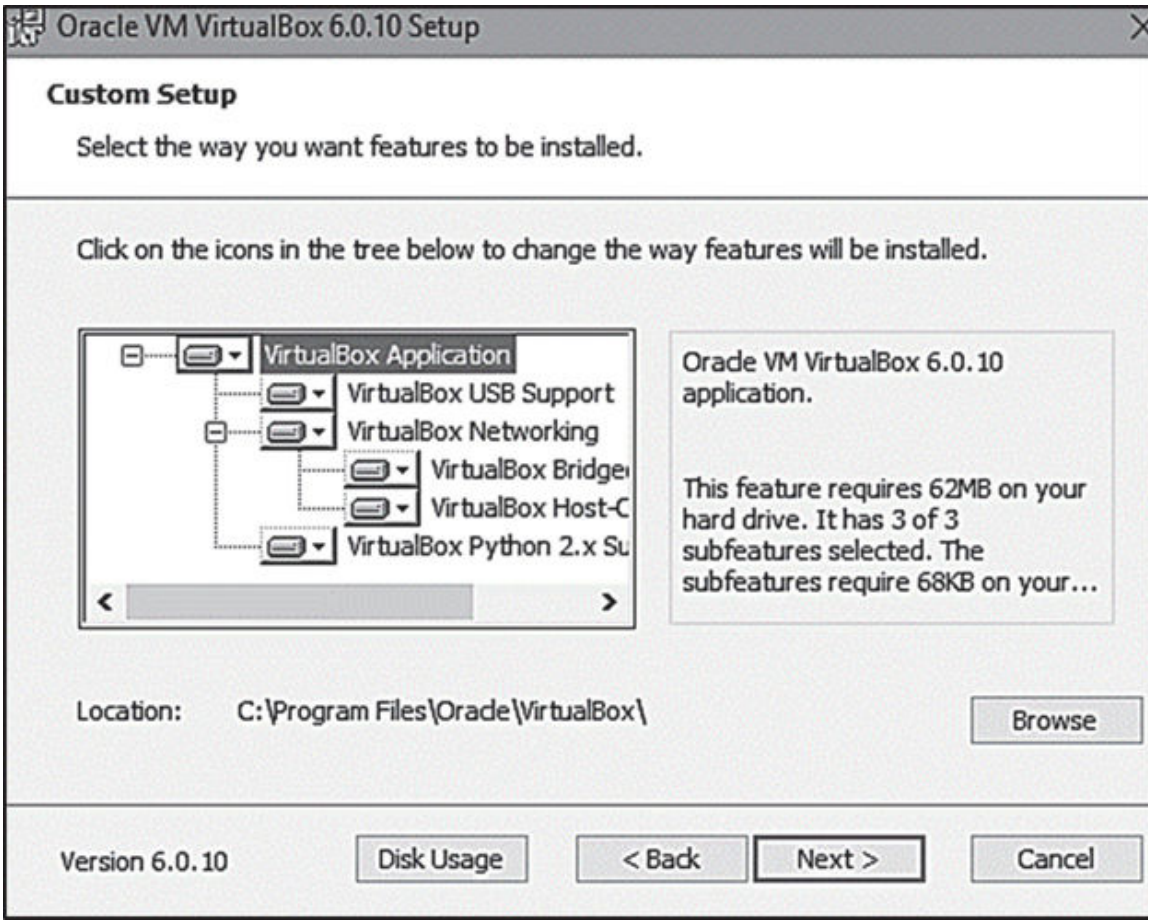


Figure 1-5 VirtualBox Installation 2

5. Untick any of the checked items if required and click Next to continue.

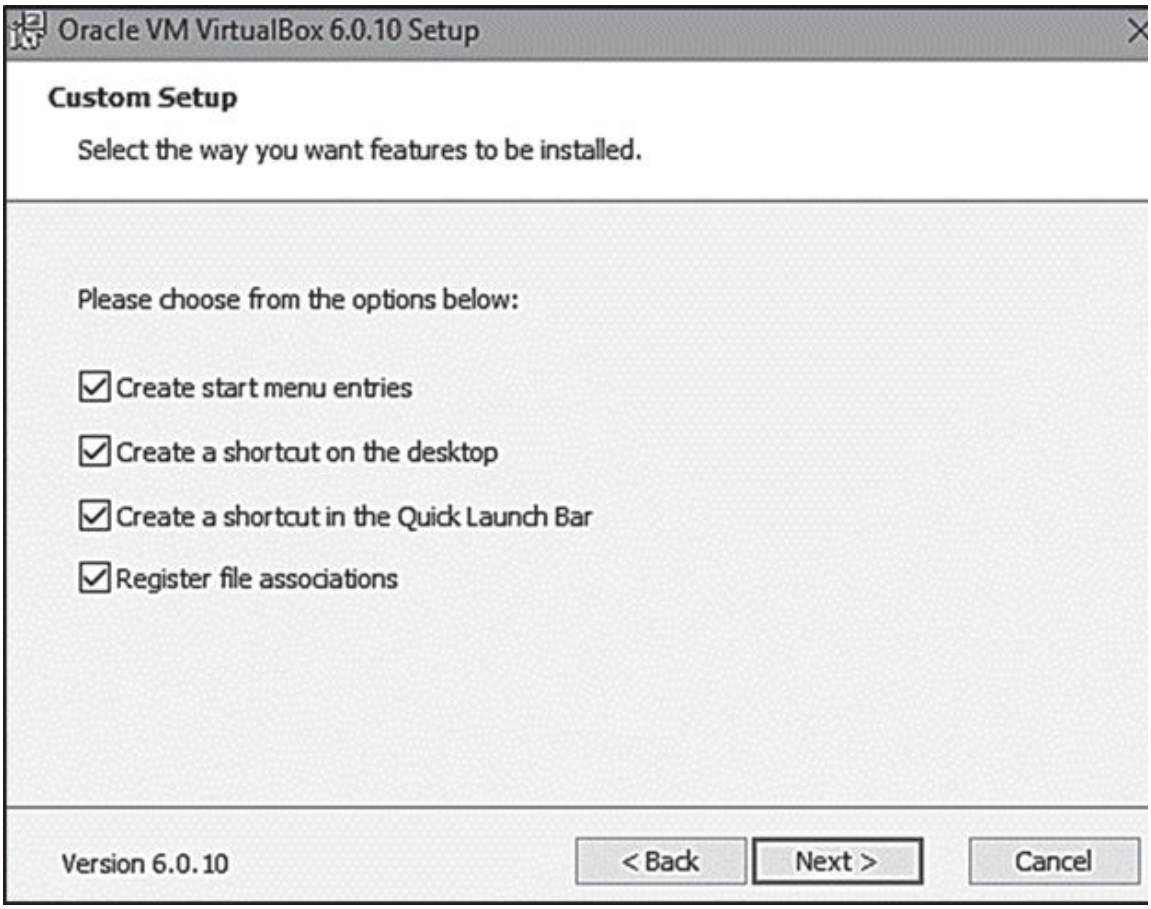


Figure 1-6 VirtualBox Installation 3

6. Accept the warning and continue by pressing Yes.

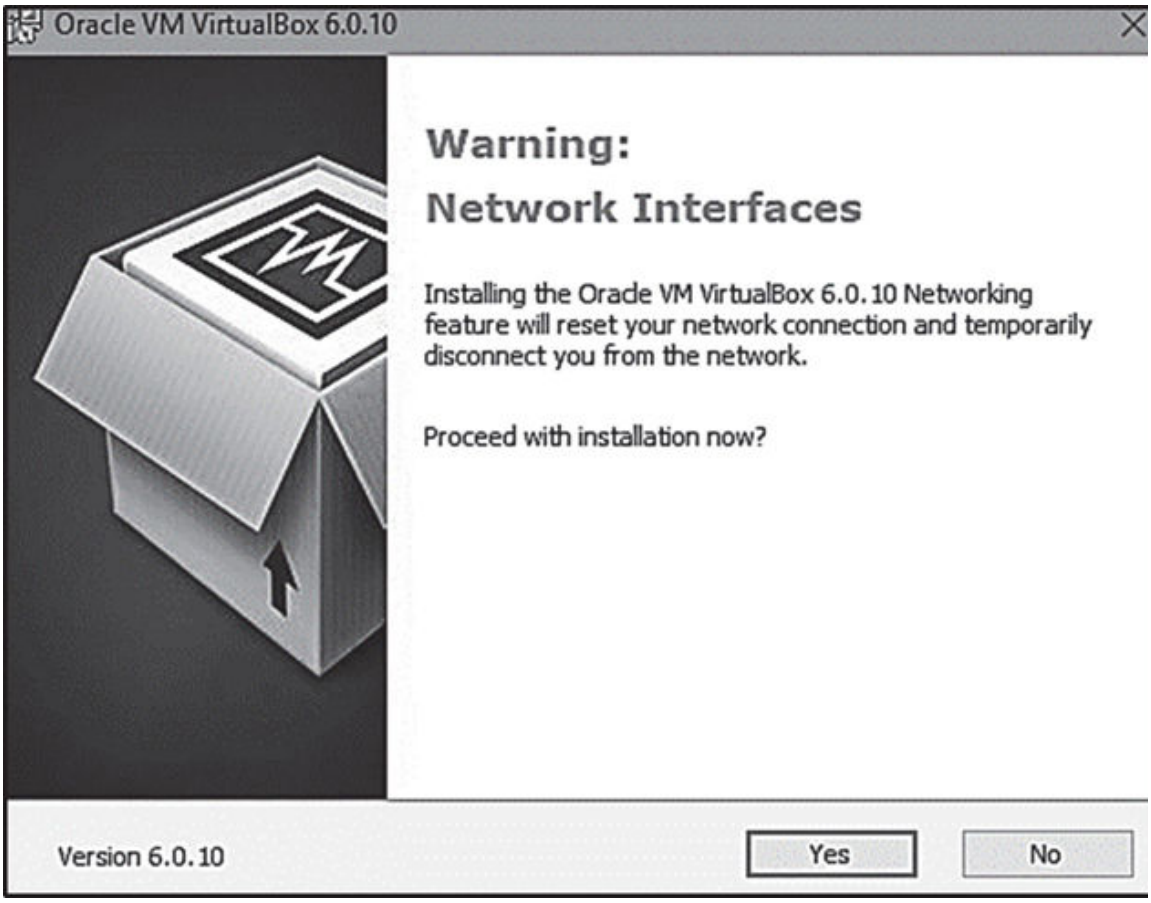


Figure 1-7 VirtualBox Installation 4

7. The setup wizard is now ready to begin the installation. Click Install to proceed.

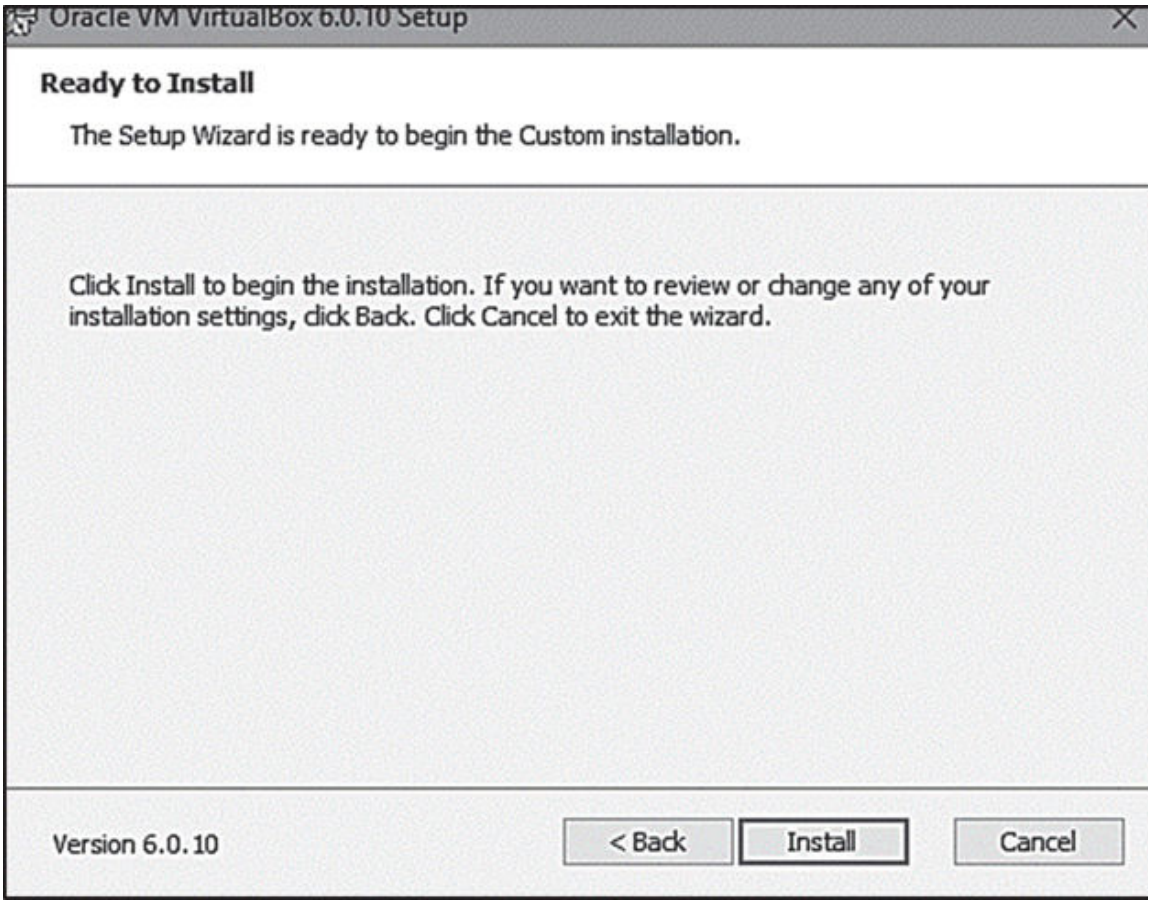


Figure 1-8 VirtualBox Installation 5

8. Click Finish to continue.

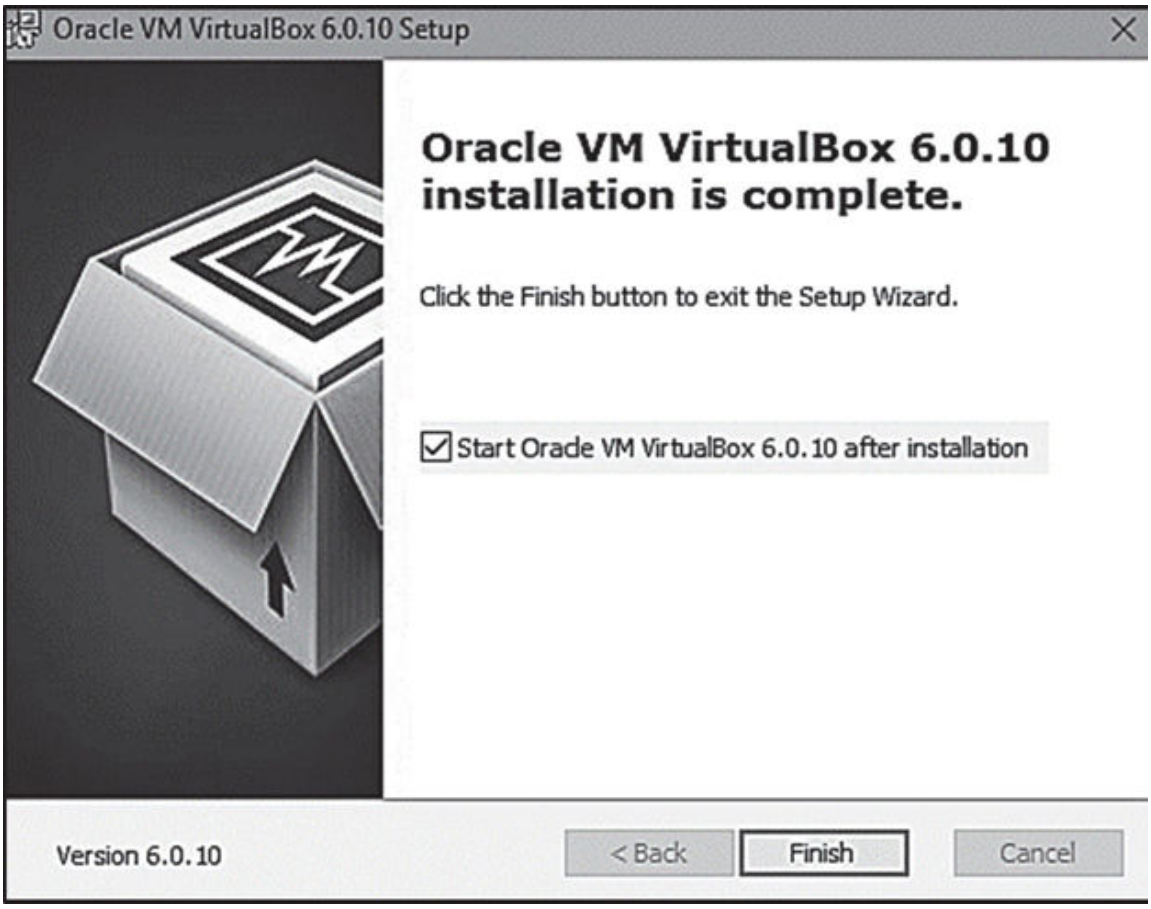


Figure 1-9 VirtualBox Installation 6

This brings the installation of VirtualBox to a successful completion. It will also launch the application.

Creating a Virtual Machine

Use VirtualBox to create the first virtual machine called *RHEL8-VM1* with specifications described earlier in this chapter. Here are the steps for the creation.

9. Launch VirtualBox if it is not already running. The interface looks similar to what is shown in Figure 1-10.

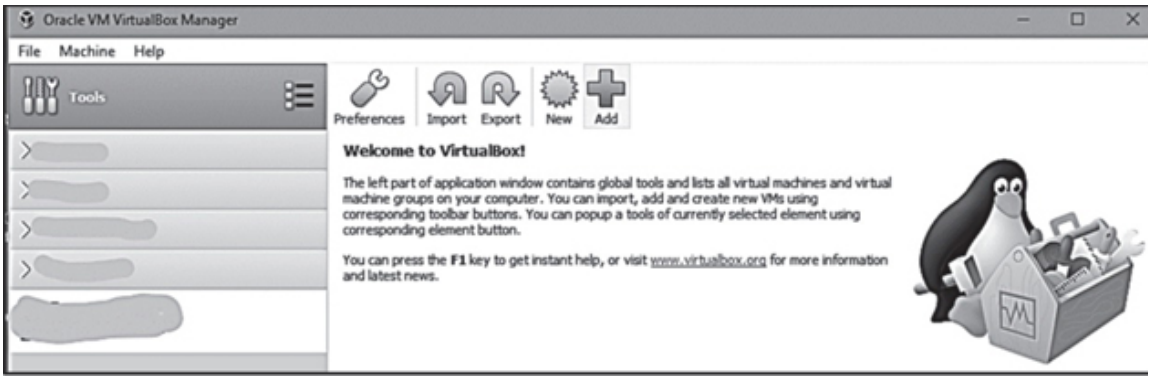


Figure 1-10 Virtual Machine Creation 1

10. Click on New on the top menu bar to start the virtual machine creation wizard (see Figure 1-11). Enter the name *RHEL8-VM1*, select Linux as the operating system type, and Red Hat (64-bit) as the version. For this demonstration, accept the default location to store the VM files on the C drive. Click Next to continue.

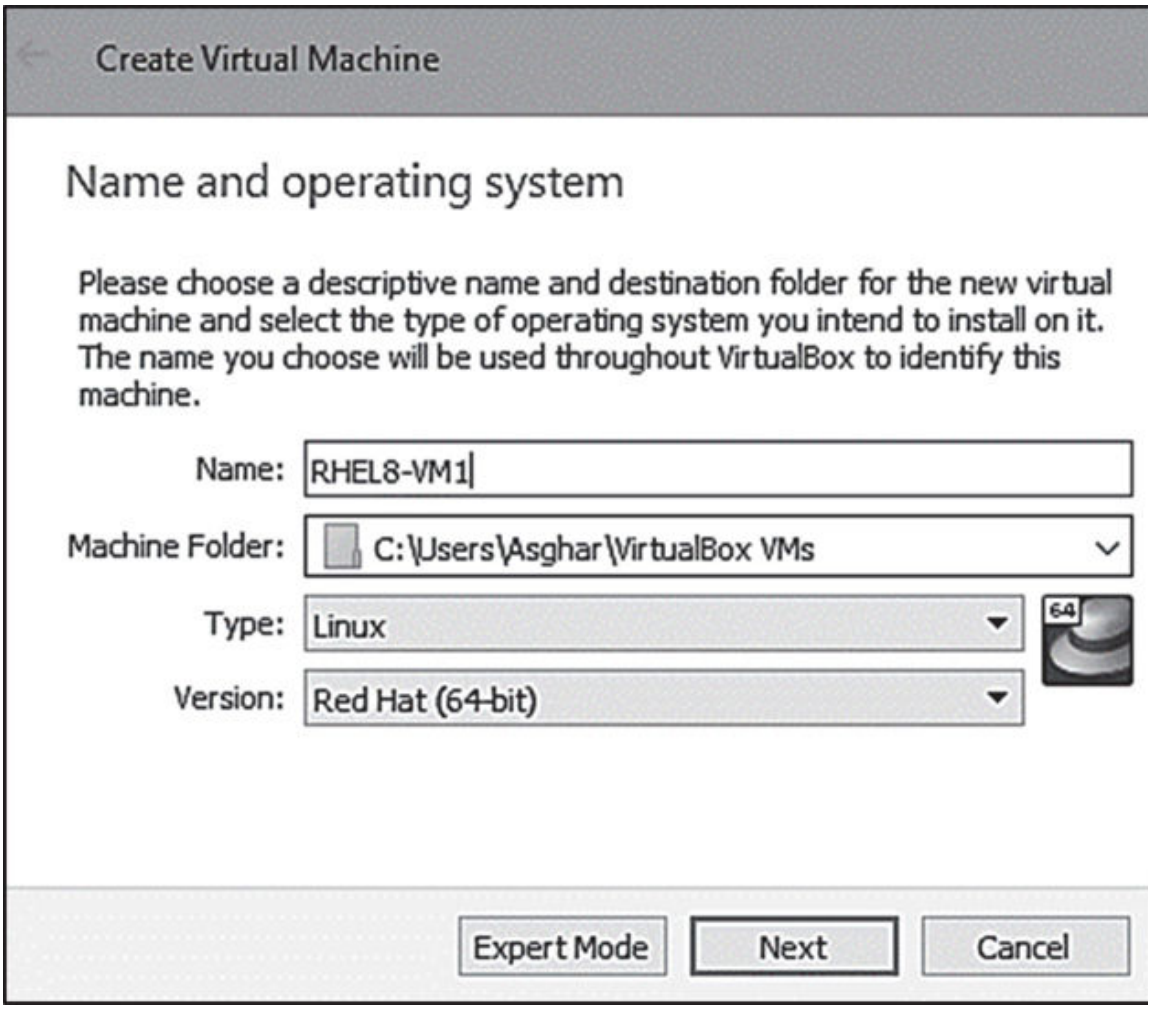


Figure 1-11 Virtual Machine Creation 2

11. In the next window, specify the memory size you want allocated to the VM. Accept the recommended 1GB for this VM and click Next.

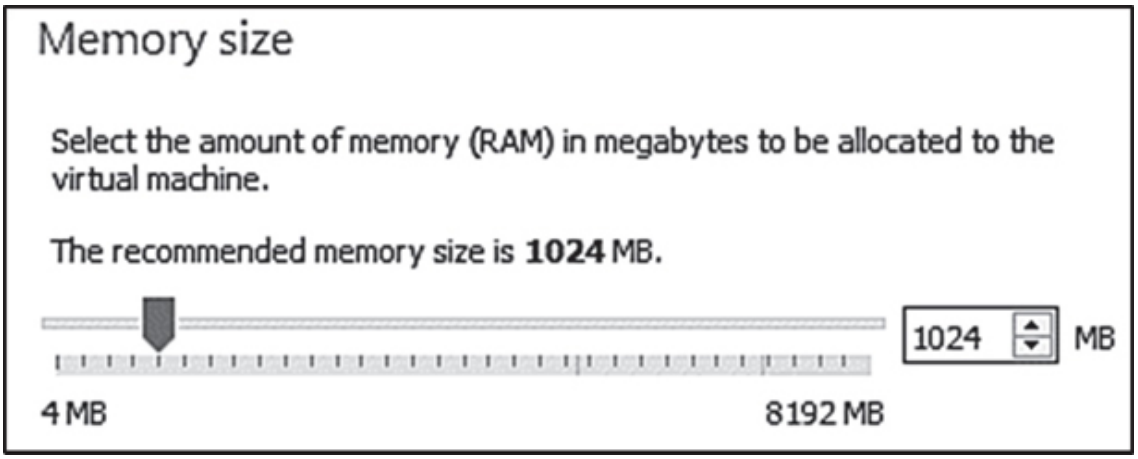


Figure 1-12 Virtual Machine Creation 3

12. The VM will need a hard disk to store the RHEL 8 operating system. For this demonstration, choose the creation of a virtual hard disk option, which is also the default selection. The other two options are irrelevant for this exercise. You will adjust the recommended hard disk size of 8GB in a later step.

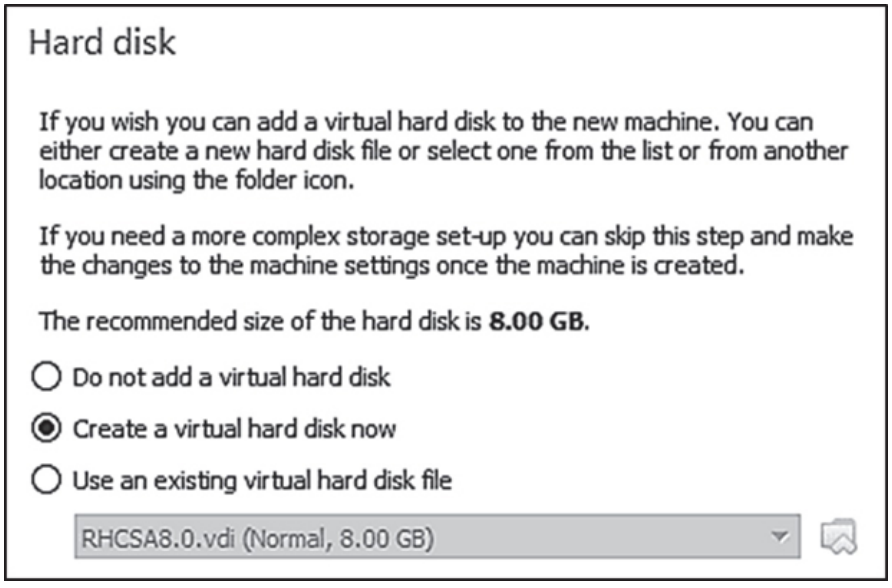


Figure 1-13 Virtual Machine Creation 4

13. The virtual hard disk is in essence a file. VirtualBox supports three different virtual hard disk file types: VDI (*VirtualBox Disk Image*), VHD (*Virtual Hard Disk*), and VMDK (*Virtual Machine Disk*). These file types represent

VirtualBox, Microsoft, and VMware disk image formats. Select the default, VDI, for this demonstration.

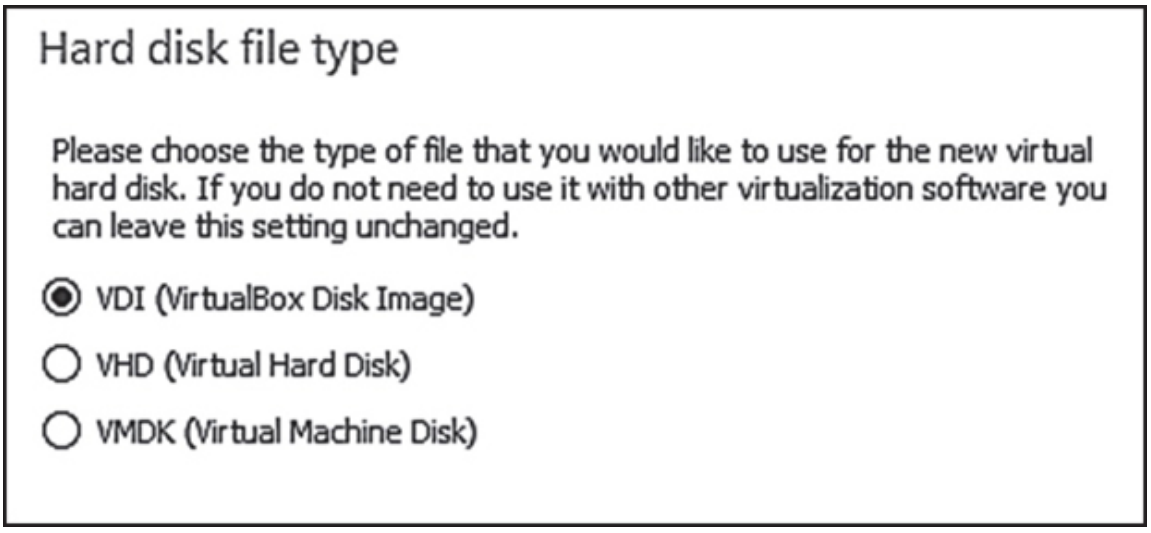


Figure 1-14 Virtual Machine Creation 5

14. There are a couple of methods to allocate storage for the OS. A fixed allocation method reserves the entire specified capacity for the disk right away and takes a while to create the disk. In contrast, the dynamically allocated option only uses the amount of disk space that is needed for the storage without reserving the entire disk capacity. This option is preferred over the former. Click Next to continue with the default option.

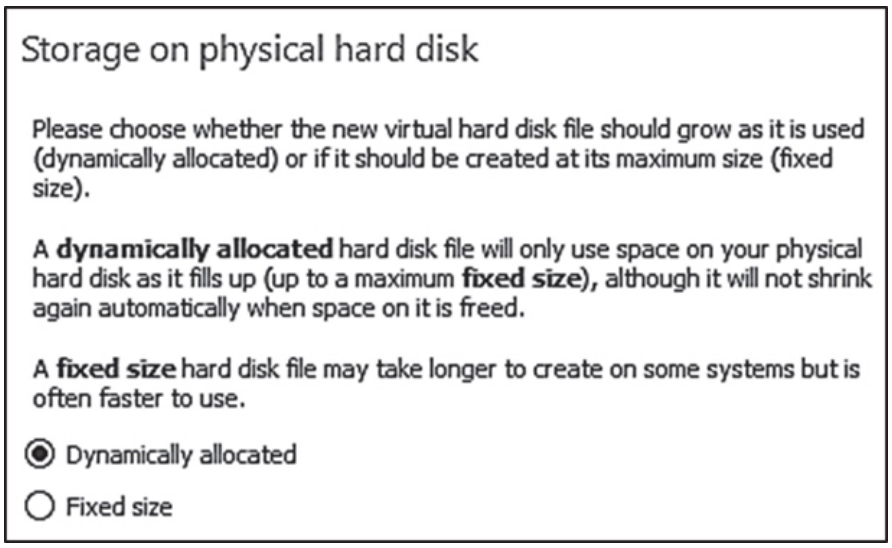


Figure 1-15 Virtual Machine Creation 6

15. The RHEL 8 virtual disk image will be stored on a computer disk. The name of the VDI file will match the name of the virtual machine name. You can also specify the size for the disk. Enter 10GB for the OS disk and leave the VDI file storage location to the default.

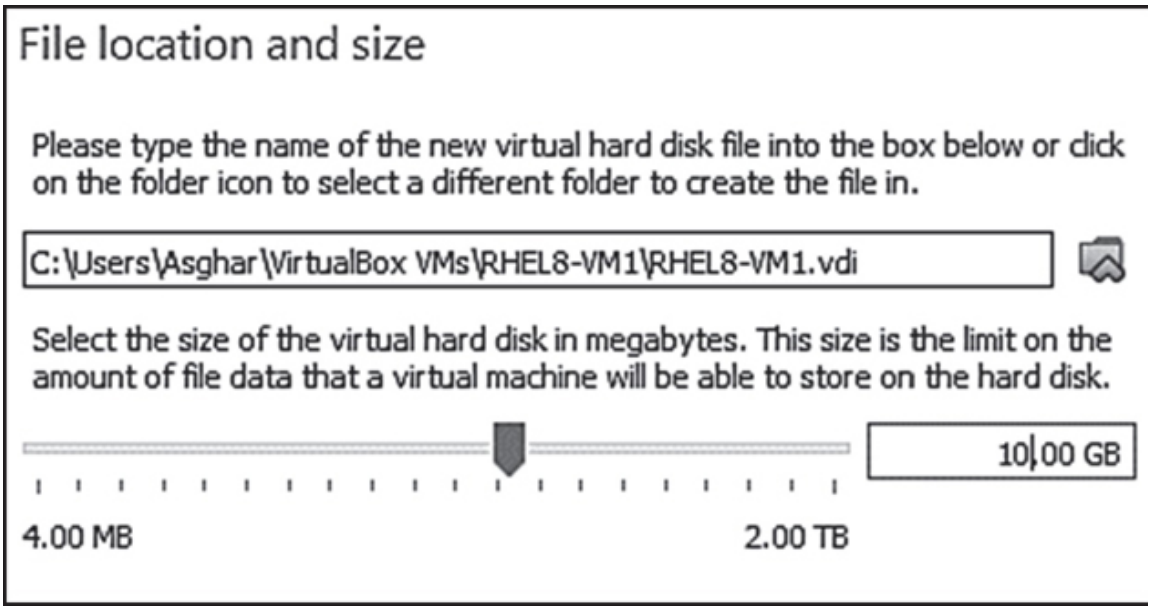


Figure 1-16 Virtual Machine Creation 7

16. Clicking Next on the previous window completes the VM creation process and ends the wizard. VirtualBox will have the VM listed along with its configuration (see Figure 1-17).

Figure 1-17 Virtual Machine Creation 8

There are a number of other configurable items as depicted in Figure 1-17. You may have to adjust the display controller setting to VMSVGA under Display to view full screen content on the console and attach the network adapter to Bridged Adapter under Network for bi-directional communication with the Windows host and the Internet.

Exercise 1-2: Download and Install RHEL 8

This exercise will build *server1* in *RHEL8-VM1*.

In this exercise, you will download RHEL 8 and install it in *RHEL8-VM1* that you created in Exercise 1-1. You will attach the RHEL 8 ISO image to the VM, name the Linux system *server1.example.com* and IP 192.168.0.110. Additional configuration will be supplied as the installation advances.

> **EXAM TIP:** Downloading and installing RHEL 8 are beyond the scope of the exam objectives. They have been included here only to support the readers with building their own lab environment for practice.

The user creation, base environments, storage partitioning, network device and connection configuration, time synchronization, and other topics are not explained as part of this exercise; however, they will be discussed in later chapters.

Downloading RHEL 8 ISO Image

RHEL 8 image is available for free download from Red Hat Developer's webpage. You need to create a user account in order to log in and obtain a copy for yourself. Alternatively, you can use your credentials on Facebook, Google, LinkedIn, Twitter, etc. for login. For this demonstration, you will find instructions on how to open a new account and download the software.

1. Visit *https://developers.redhat.com/login* and click "Create one now".

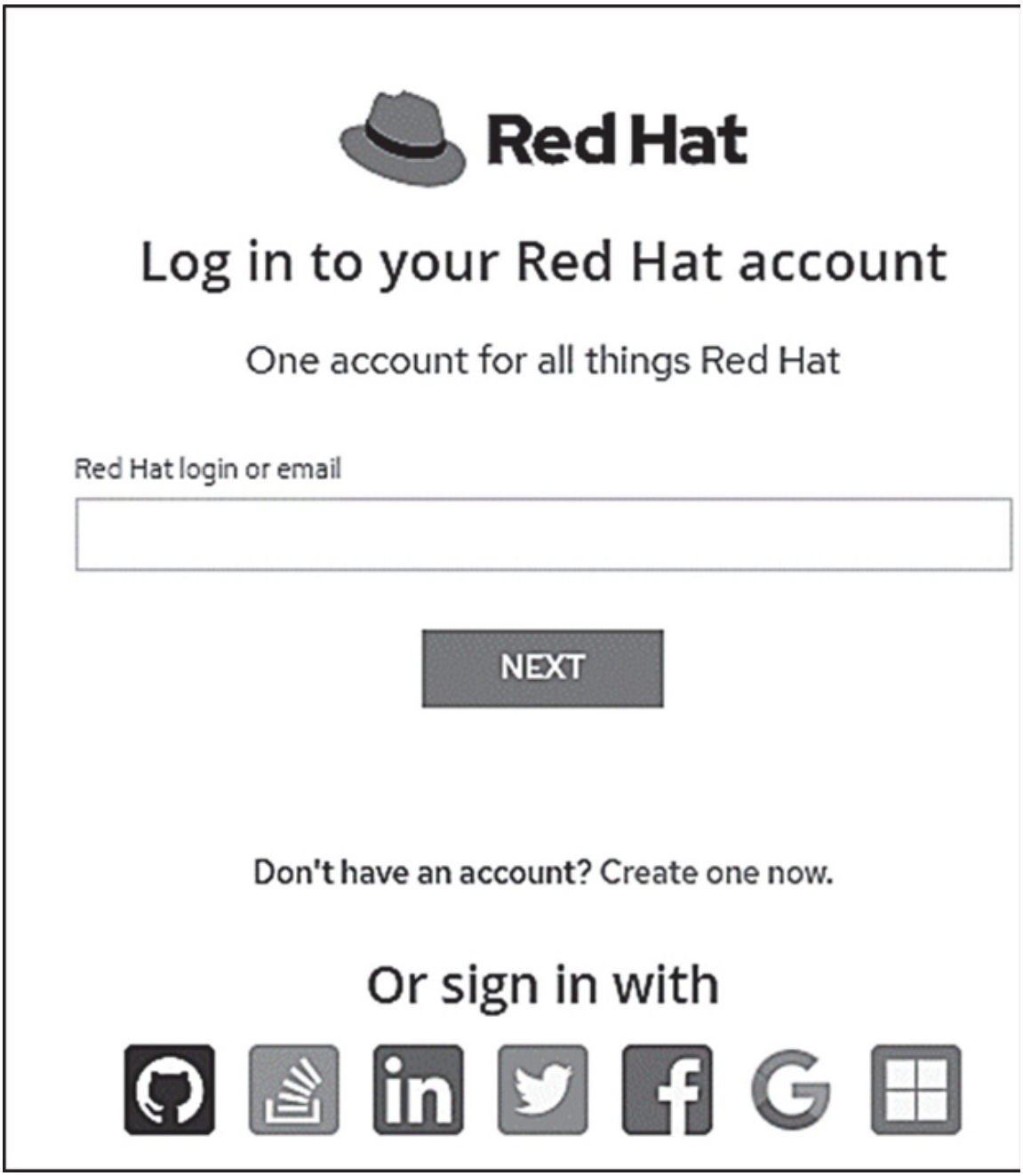


Figure 1-18 Red Hat User Account Creation 1

2. Fill out the form by providing a unique username, email address, and password. Make sure to checkmark the boxes to accept terms and conditions.

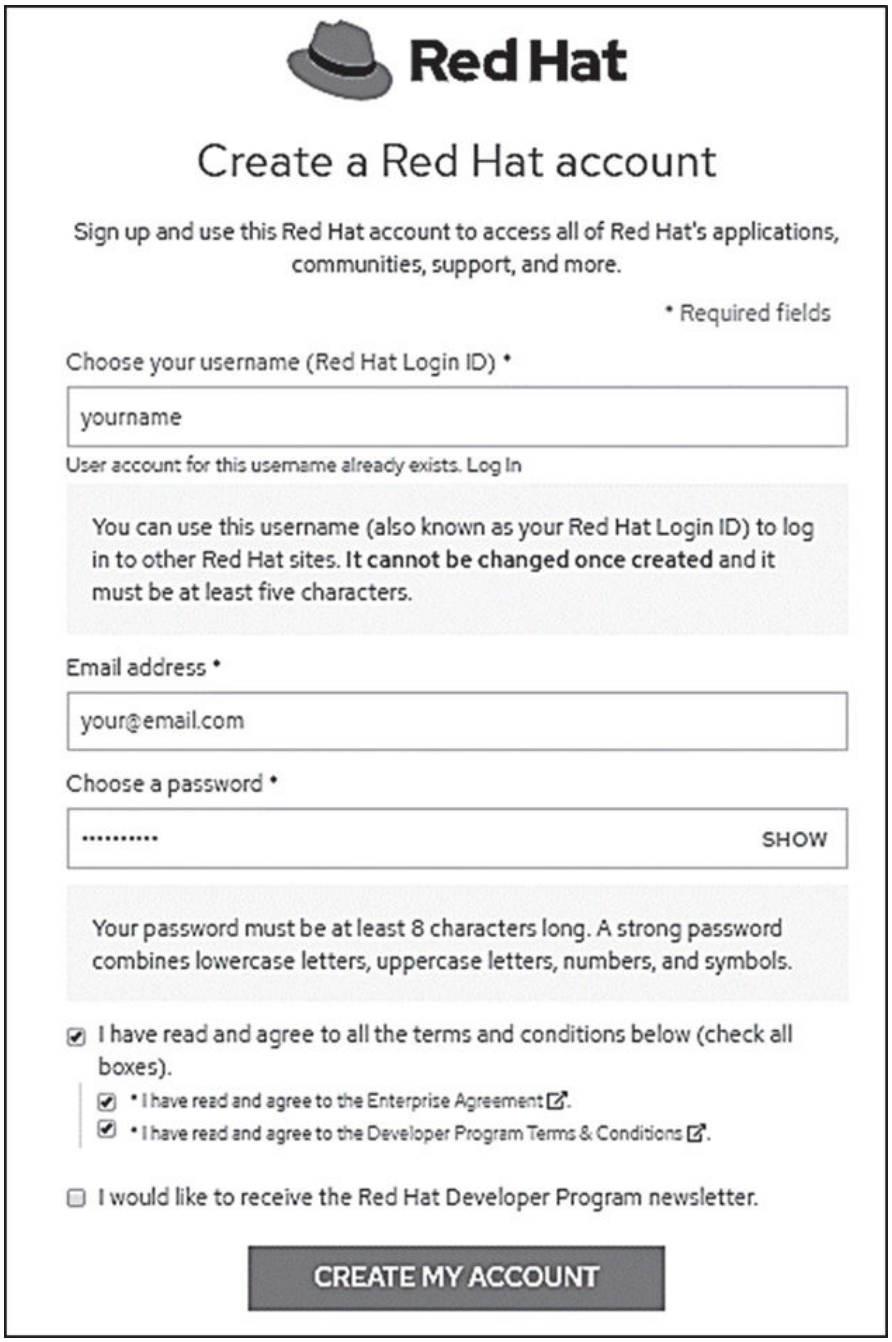


Figure 1-19 Red Hat User Account Creation 2

3. After an account has been created, go back to the login page *https://developers.redhat.com/login* and submit the credentials to log in.
4. Click on Linux at the top of the page under the Red Hat Developer logo.

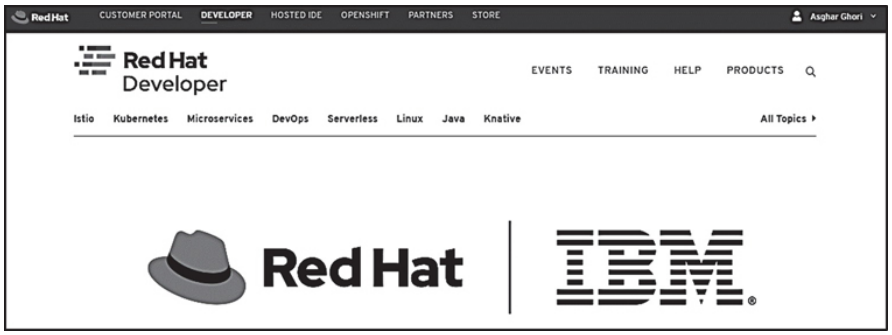


Figure 1-20 Red Hat Developer Login

5. Click DOWNLOAD on the following page.
6. Click DOWNLOAD INTEL ISO on the subsequent page to initiate a download.

The filename of the downloaded image for RHEL version 8.0 will be rhel-8.0-x86_64-dvd.iso and it will be about 6.9GB in size. You can move this file to a disk location on your computer where you want it stored.

Attaching RHEL 8 ISO Image to the Virtual Machine

We now attach the RHEL 8 ISO image to *RHEL8-VM1* in order to boot and install the OS in the VM. Click on "[Optical Drive] Empty" under Storage in VirtualBox for this VM and select Choose Disk Image. Navigate to where you have the ISO image stored. Highlight the image and click Open to attach it to the VM. After the image has been attached, the VirtualBox Storage configuration will look like Figure 1-21.

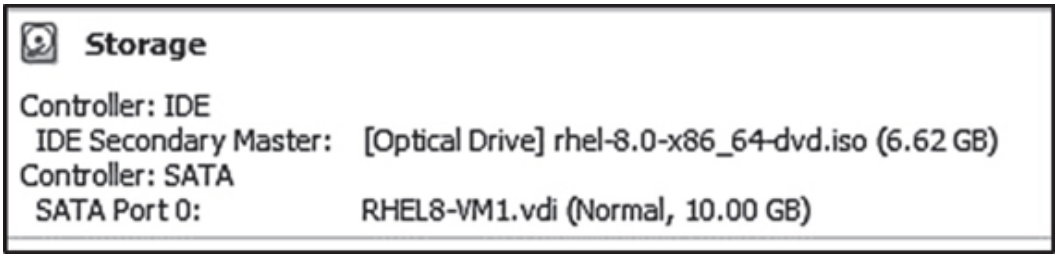


Figure 1-21 Attach ISO Image to VM

Leave the rest of the settings to their default values.

Launching the Installer

7. While the VM is highlighted in VirtualBox, click the Start button at the top to power up the VM.
8. A console screen pops up displaying the boot menu (Figure 1-22) with three options. Press the Spacebar key to halt the autoboot process.

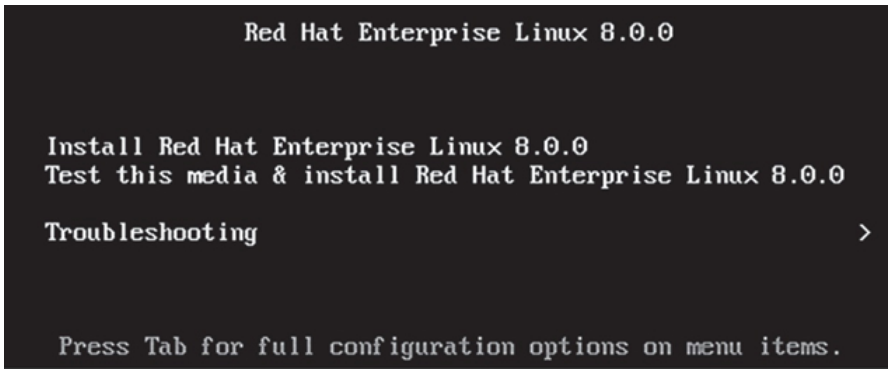


Figure 1-22 Boot Menu

The first option, “Install Red Hat Enterprise Linux 8.0.0”, is usually used for installing RHEL 8 unless you want the installation media tested for integrity before continuing, in which case you will select the second option. Anaconda waits 60 seconds for you to alter the selection, or it proceeds and autoboots using the second option on the list, which is also the default. The third option, “Troubleshooting”, allows you to address some boot-related issues that might occur during installation.

Use the Up or Down arrow key to select the “Install Red Hat Enterprise Linux 8.0.0” entry and press Enter. The installer is launched in graphical mode.

9. The installer program shows a welcome screen with a long list of supported languages that you could use during the installation. The default is set to English. Click Continue to accept the default and move on.

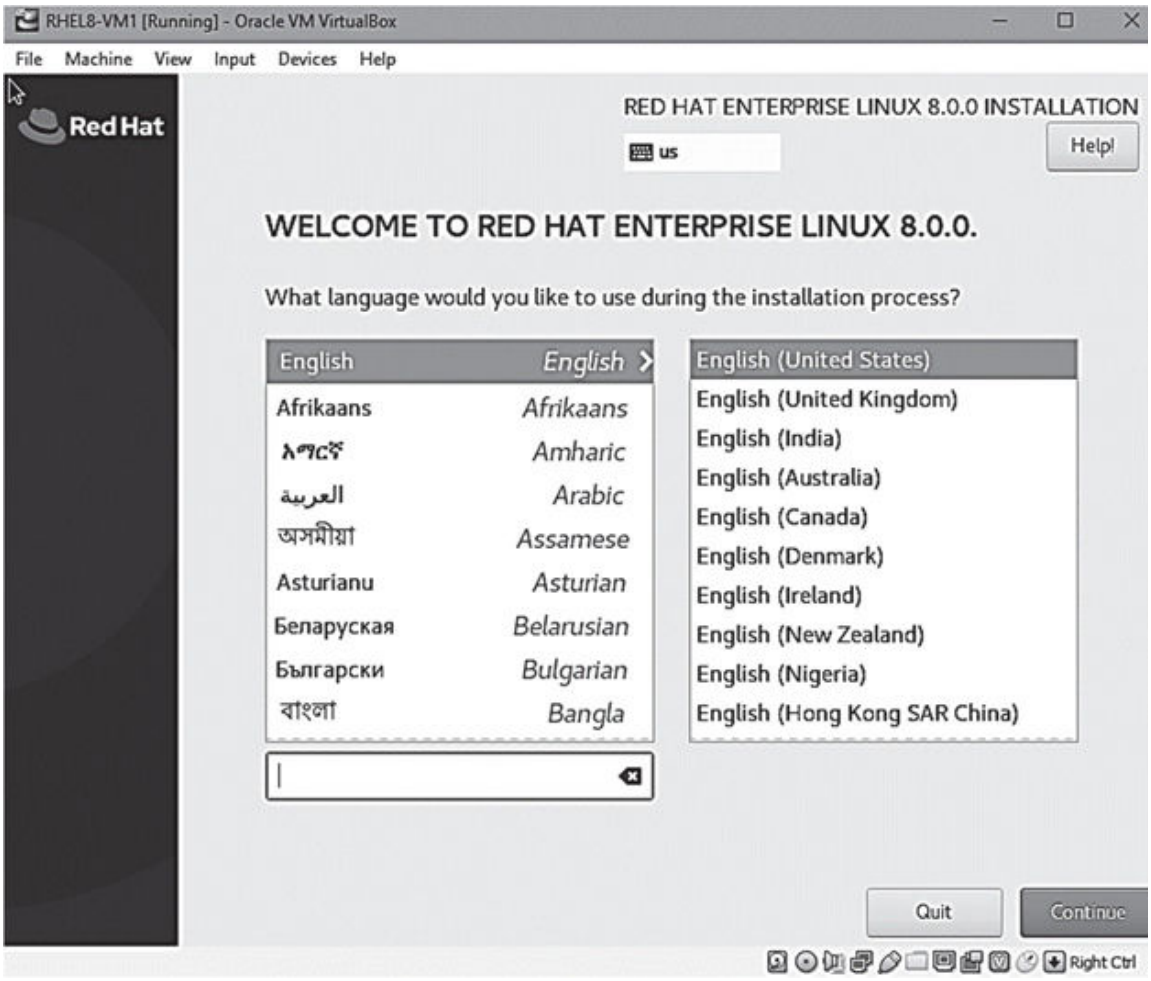


Figure 1-23 Language Selection

If all the content does not fit on the console screen, try changing the Graphics Controller to VMSVGA under Settings | Display in the VirtualBox Manager for the VM.

10. The “Installation Summary” screen appears next, as shown in Figure 1-24. You have the opportunity to make all necessary configuration changes prior to starting the installation. This screen presents a single interface to configure localization (keyboard, language, date, time, and time zone), software (installation source and software selection), and system (disk selection and partitioning, network and hostname assignments, etc.).

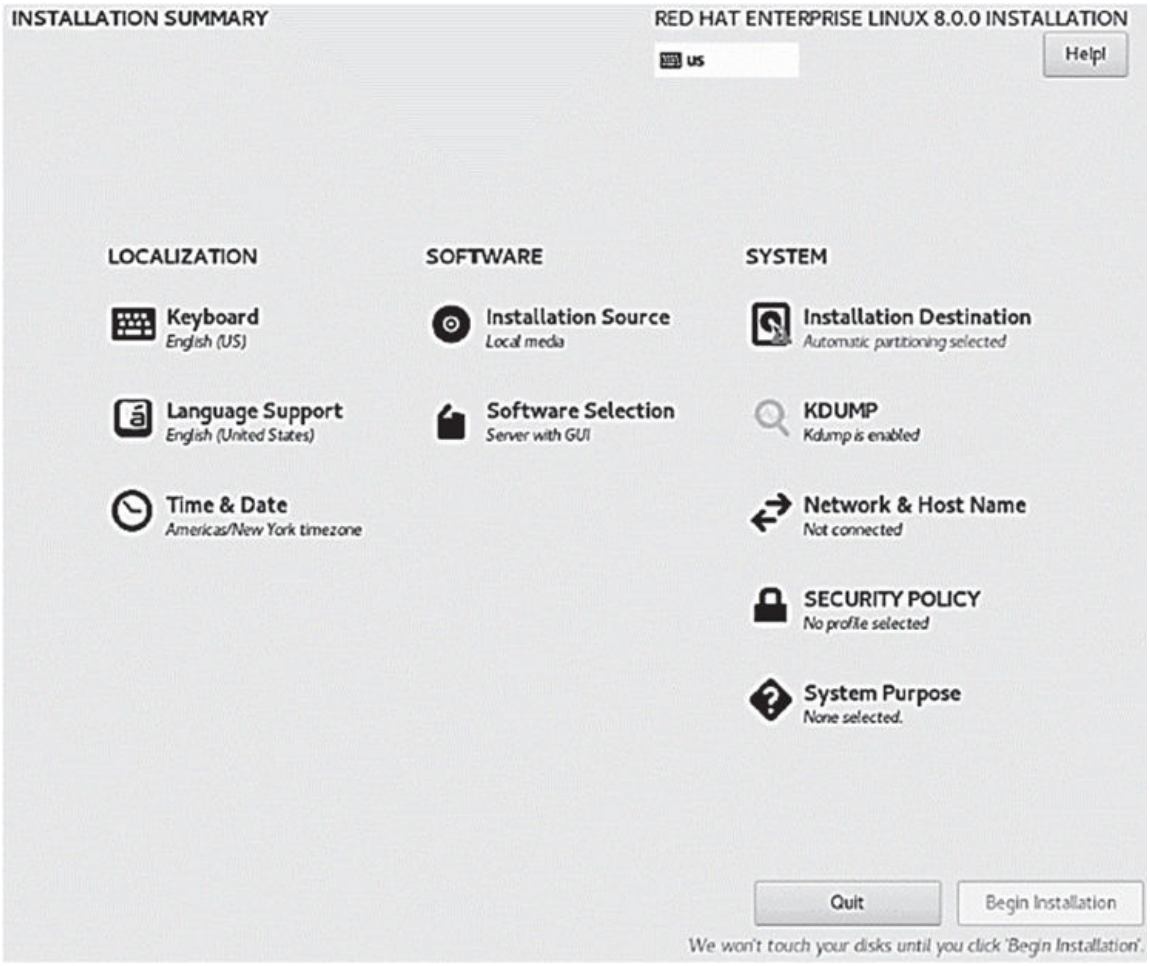


Figure 1-24 Installation Summary | Main

Any items highlighted in red and with a warning sign must be configured before the Begin Installation button at the bottom right of the screen is enabled.

There is no particular sequence to configure these items. If you do not wish to change a non-highlighted item, simply leave it intact and the installation program will apply the default settings for it.

Adding Support for Keyboards and Languages

11. Anaconda presents additional choices for keyboard layouts and languages for use during and after the installation. This

should only be done on systems where support for multiple keyboard layouts and languages is required. The default is the US English for both.

Configuring Time & Date

12. Click Time & Date to set the time zone (region and city), date, and time for the system. See Figure 1-25. Click Done in the upper left corner to save the changes and return to the Installation Summary screen.

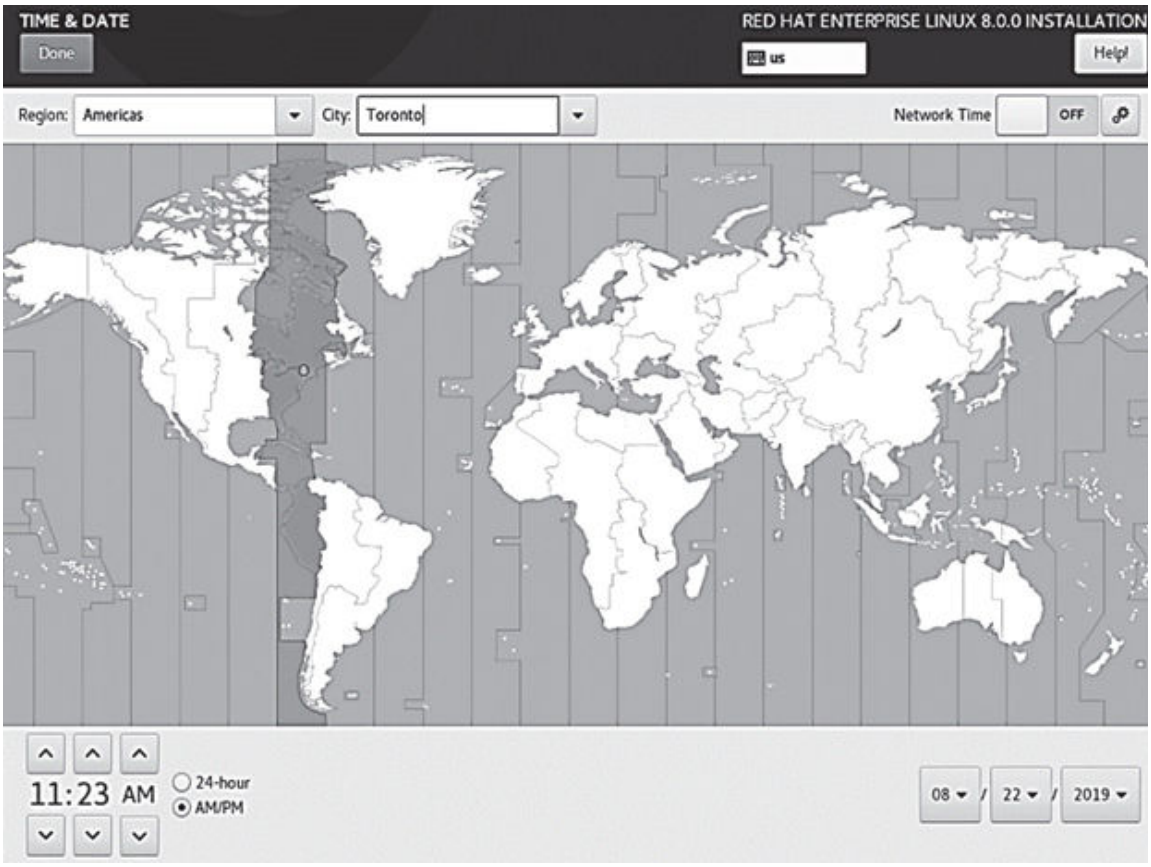


Figure 1-25 Installation Summary | Time & Date

Figure 1-25 reflects two adjustments from the default. The city is changed to Toronto, and the clock format is switched to AM/PM.

Choosing an Installation Source

13. You can set the installation source for RHEL 8. By default, Anaconda chooses the auto-detected local media (DVD, USB flash drive, or ISO image) that was used to start this

installation. For this demonstration, leave the installation source to the default. Click Done to return to the Installation Summary page.

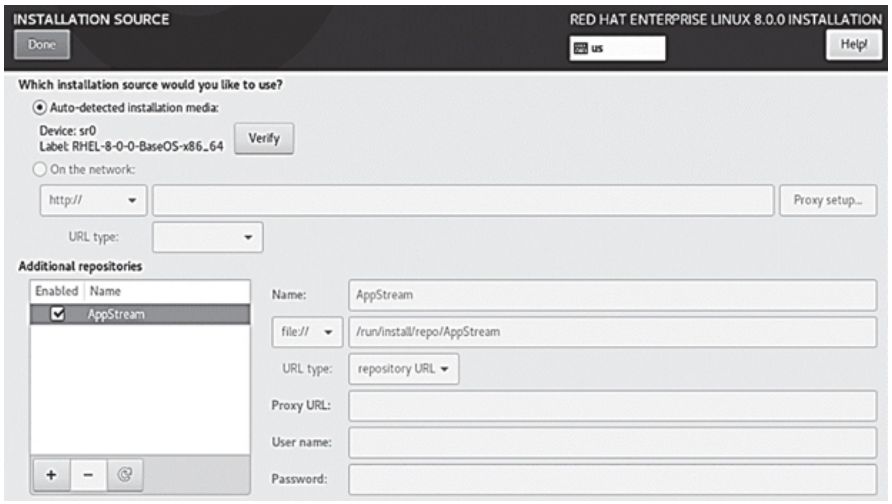


Figure 1-26 Installation Summary | Installation Source

If you have access to a configured network location hosting the installation files, you can choose “On the network” and specify the HTTP, HTTPS, FTP, or NFS protocol, hostname or IP address of the network server, and the path to the files. You can also specify the locations of additional software repositories, besides the default AppStream, if you have access to them.

Selecting Software to be Installed

14. You can choose the base operating environment that you want installed. Base environments are predefined groups of software packages designed for specific use cases. The six base environments are described in Table 1-2.

Base Environment	Description
Server with GUI	Infrastructure server with graphics support
Server	Infrastructure server without graphics support
Minimal Install	Installs a minimum number of packages for basic system use
Workstation	Ideal for desktop and laptop users who require graphical support and with a minimal set of services
Custom Operating System	Gives you a set of basic building blocks for custom installations
Virtualization Host	Infrastructure plus virtualization support to host virtual machines

Table 1-2 Installation Summary | Software Selection | Base Environments

Choosing a base environment in the left pane reveals additional components on the right that may be ticked for installation along with the selected base environment. See Figure 1-27.

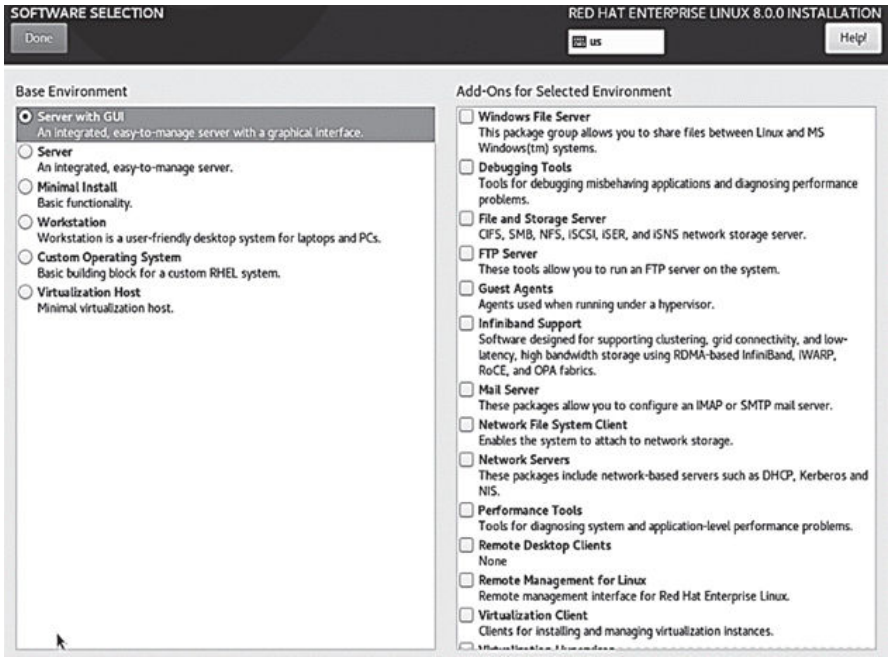


Figure 1-27 Installation Summary | Software Selection

The installer automatically picks and installs prerequisite software components to fulfill dependency requirements for a

successful installation. The default base environment is “Server with GUI” for this demonstration. Leave add-ons to the default as well. Click Done to return to the Installation Summary page.

Configuring Installation Destination

15. The Installation Destination allows you to choose an available local or remote disk for partitioning and installing the OS on. Anaconda selects “Automatic partitioning selected” (highlighted in red) on the Installation Summary page (Figure 1-24), which you can change on Installation Destination (Figure 1-28). By default, the 10GB virtual disk you assigned to the VM initially is automatically picked up by the installer as the target and it is represented as sda. The “Encrypt my data” checkbox under Encryption encrypts all partitions on the disk. If you choose this option, you will be prompted to enter a passphrase to access the partitions later. The “Full disk summary and bootloader” link at the bottom left allows you to choose a disk to place the bootloader program on. This does not need to be modified on a single disk system. The default and the only bootloader program available in RHEL 8 is called GRUB2, and it is explained at length in Chapter 11, “Boot and Initialization”.

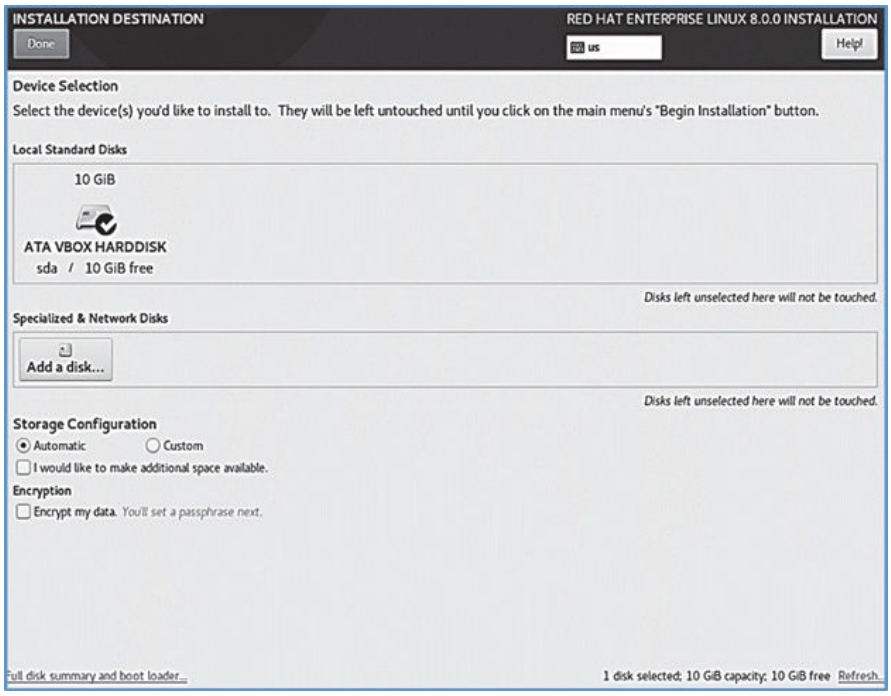


Figure 1-28 Installation Summary | Installation Destination

For this demonstration, stick to the default automatic partitioning scheme. Simply click Done to return to the previous screen. This scheme will create three partitions—*/boot*, */*, and swap, and together they will consume the entire selected disk.

Configuring Network and Hostname

16. Assigning appropriate IP and hostname are essential for system functionality in a network environment. Click Network & Hostname on the Installation Summary page and a window similar to the one shown in Figure 1-29 will appear. Anaconda detects all attached network interfaces, but it does not automatically assign them IPs. Also, the default hostname is set to *localhost.localdomain*. You need to modify these assignments so that your system is able to communicate with other systems on the network. Currently, there is one network device assigned to the system, which is represented as *enp0s3*.

The terms “network interface” and “network device” refer to the same network hardware component. These terms are used interchangeably throughout this book. These terms are different from the term “network connection”, which is the software configuration applied to a network interface/device.

The default naming convention for network devices vary based on the underlying virtualization software being used.

Change the hostname to *server1.example.com* in the Hostname field. For IP assignments, there are a couple of options. You can obtain them automatically from an available DHCP server by simply sliding the ON/OFF switch located in the top right-hand corner. However, for this demonstration, click Configure at the bottom right and enter IP information manually. You also need to ensure that the network connection is set to autostart.

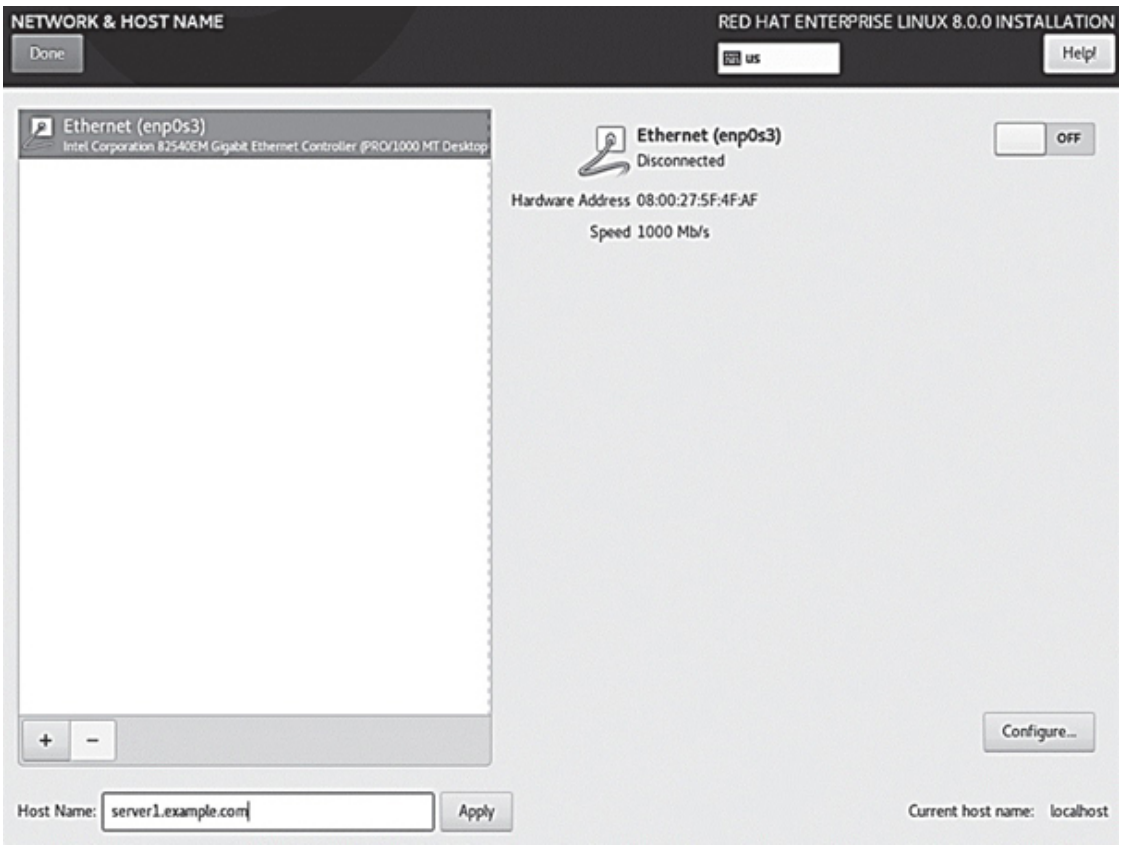


Figure 1-29 Installation Summary | Network & Hostname

There are multiple tabs available on the network connection configuration screen, as depicted in Figure 1-30. Go to IPv4 Settings and choose Manual from the drop-down list against Method. Click Add and enter address 192.168.0.110, netmask 24, and gateway 192.168.0.1. Click Save to save the configuration and return to the Network & Hostname window.

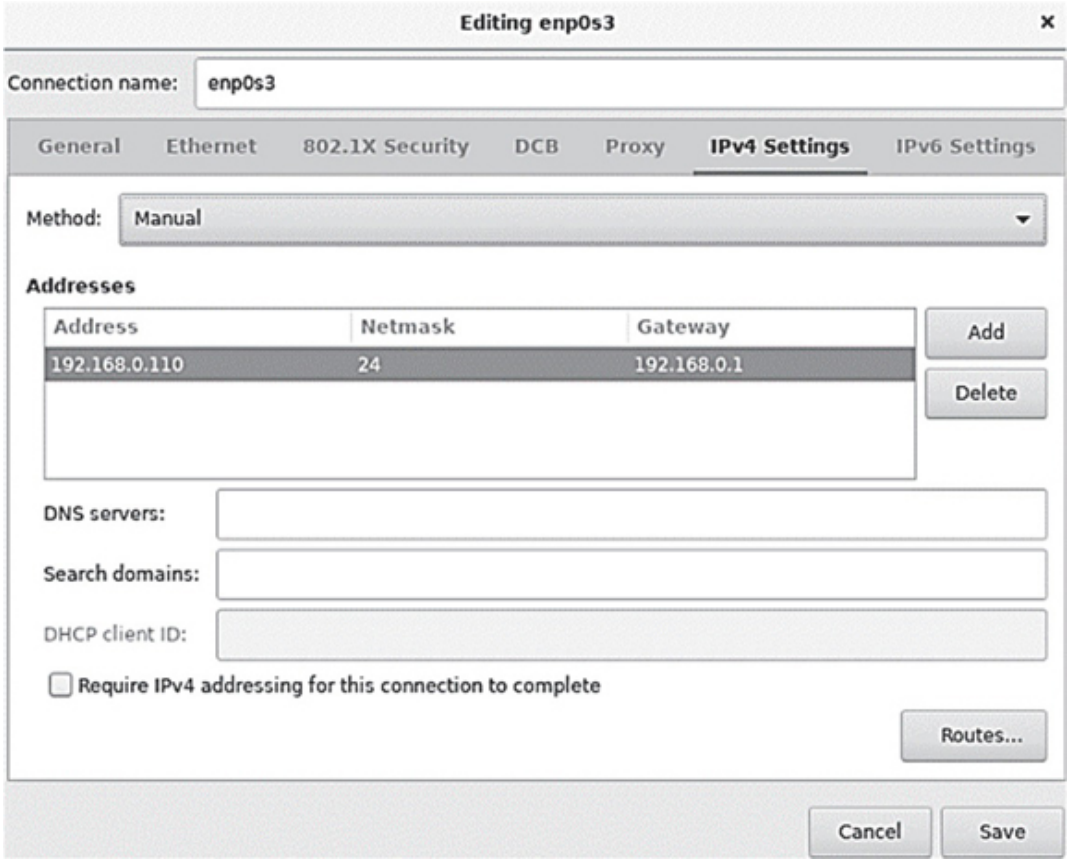


Figure 1-30 Installation Summary | Network & Hostname | Configure

On the Network & Hostname window, slide the ON/OFF switch to the ON position so that the new assignments take effect right away. This will also ensure that the assignments are applied automatically on subsequent system reboots.

Figure 1-31 Installation Summary | Network & Hostname

Now click Done to return to the Installation Summary page. Chapter 17 “Networking, Network Devices, and Network Connections” discusses configuring hostnames, network interfaces, and network connections in detail.

Beginning Installation

17. You’re now on the Installation Summary page (Figure 1-32). You still have the opportunity to go back and configure or reconfigure any items you’ve missed. Once you are satisfied, click Begin Installation at the bottom right to initiate the installation based on the configuration entered in the previous steps. Anaconda will now partition the selected disk and install the software. Any data previously stored on the disk will be erased and unrecoverable.

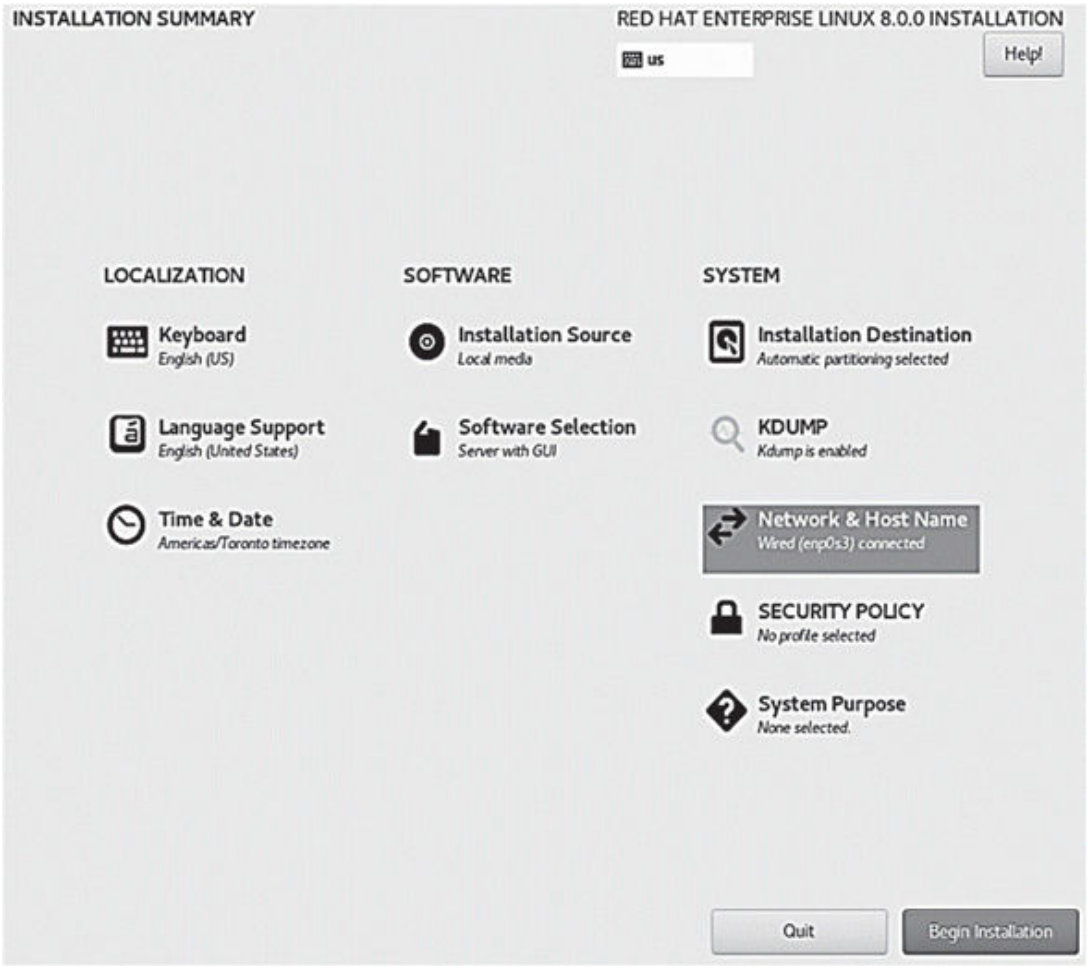


Figure 1-32 Installation Summary | Begin Installation

The Begin Installation button remains inactive until all the items highlighted in red and with a warning sign are configured.

The configuration and software copy will take some time to complete. The progress will depend on the system performance and resources allocated to the VM.

Setting root Password and Creating a User Account

18. Once the installation has begun, a new screen, called Configuration, pops up. See Figure 1-33. This is where you can monitor the progress, and it also allows you to assign a password to the *root* user and create a user account.

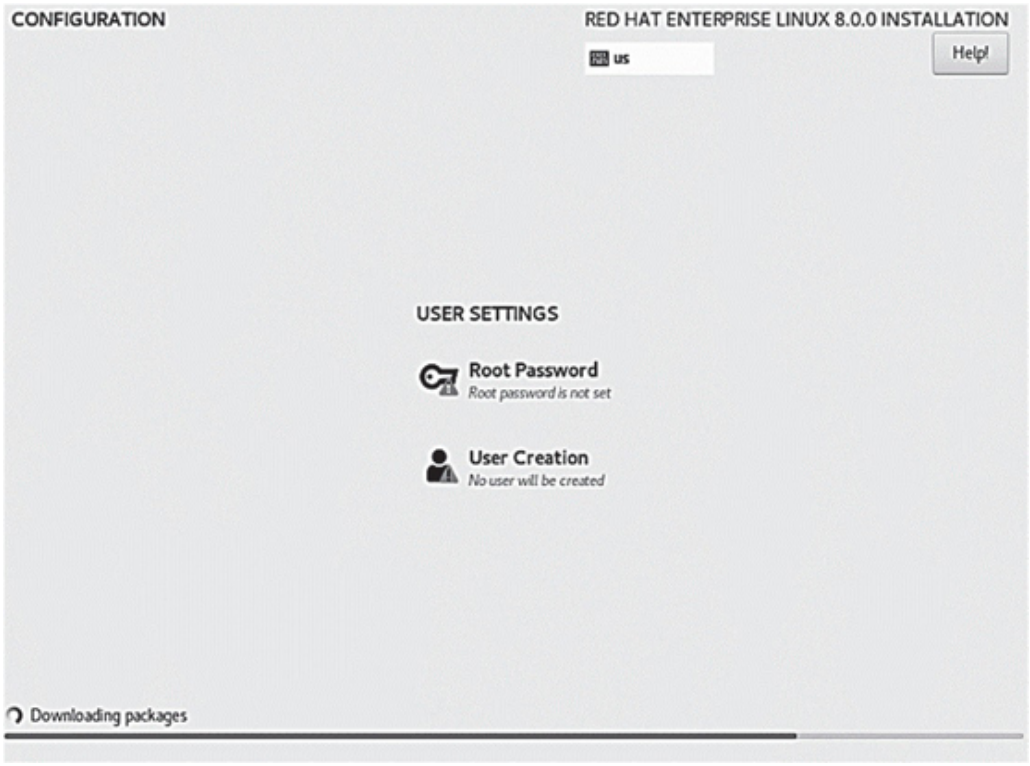


Figure 1-33 Configuration | User Settings

19. While the installer continues to run, click Root Password and set a password for the *root* user. Click Done (two clicks if the password entered is too short or simple) to return to the Configuration screen.
20. Next, click User Creation to create a user account called *user1* and assign it a password. Leave the “Make this user administrator” option unticked. See Figure 1-34. Click Done (two clicks if the password entered is too short or simple) to return to the Configuration screen.

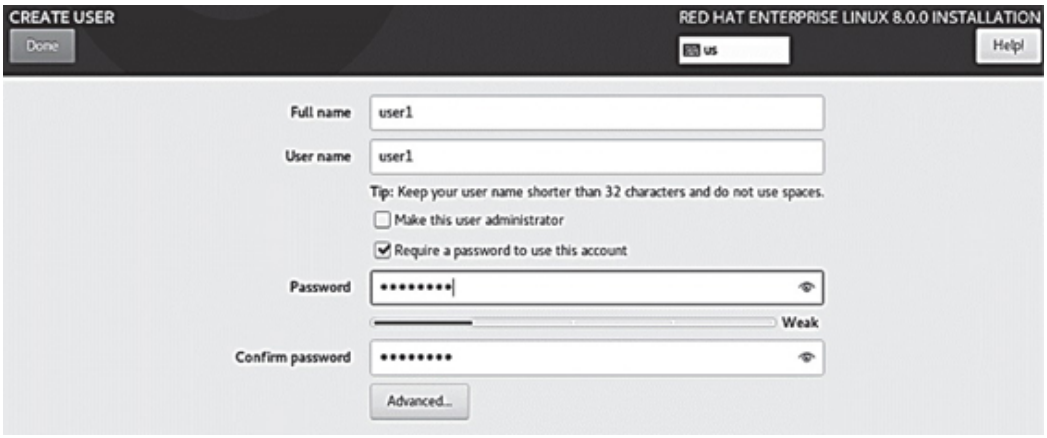


Figure 1-34 Configuration | User Settings | Create User

Anaconda will set the *root* user password and create the user account during the configuration part of the installation.

Concluding Installation

21. When the required setup is complete and all software packages are installed, a Reboot button will appear at the bottom right on the Configuration screen (Figure 1-35). Click this button to reboot the new system.

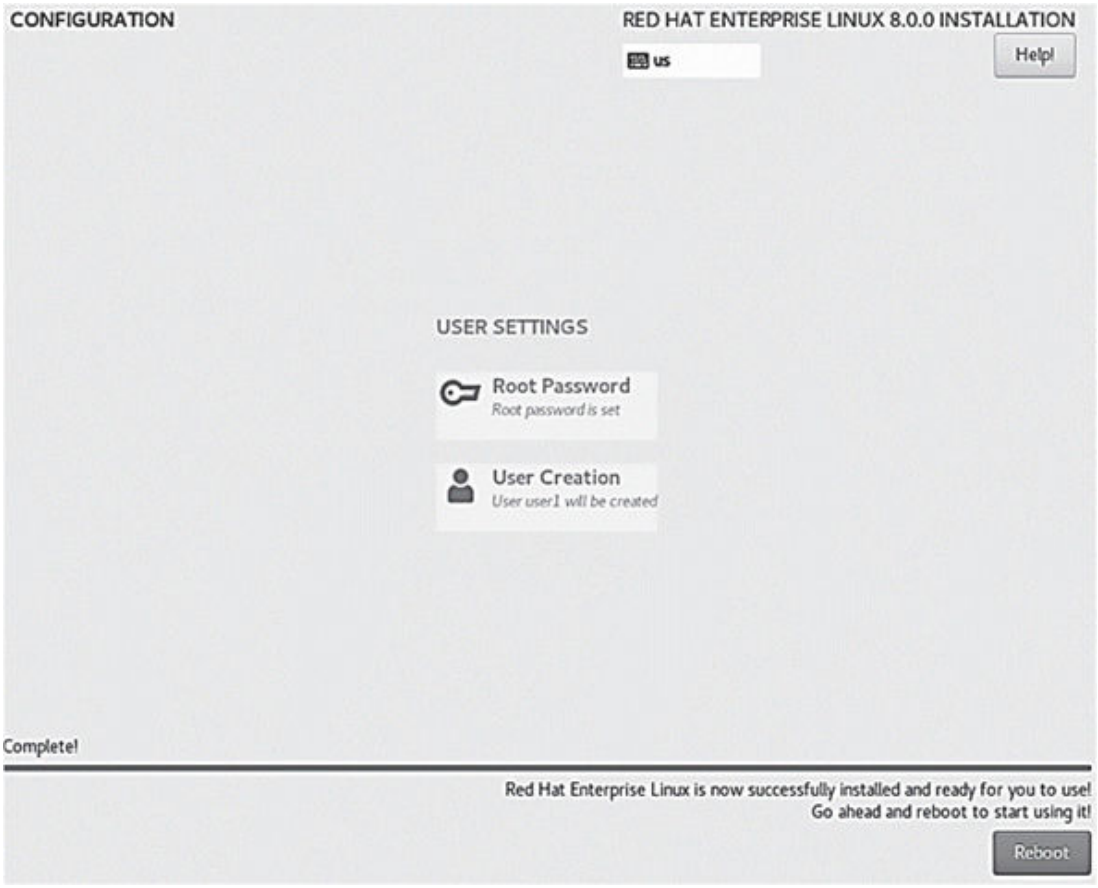


Figure 1-35 Configuration | Finishing Installation

By default, VirtualBox does not automatically change the default boot order. This results in rebooting the VM from the ISO image again and restarting the installation. To avoid this situation, power off the virtual machine from VirtualBox and alter the boot sequence.

Changing Default Boot Order

22. Power off the VM from VirtualBox.
23. The current boot sequence, as shown in Figure 1-36, is set to boot with floppy first and then optical (DVD/CD) followed by hard disk.

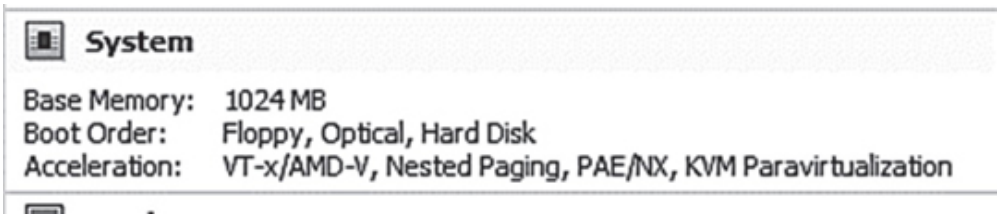


Figure 1-36 VirtualBox Manager | System | Boot Order

24. Change this sequence to hard disk first and optical next. See Figure 1-37. Untick Floppy, as it is not required.

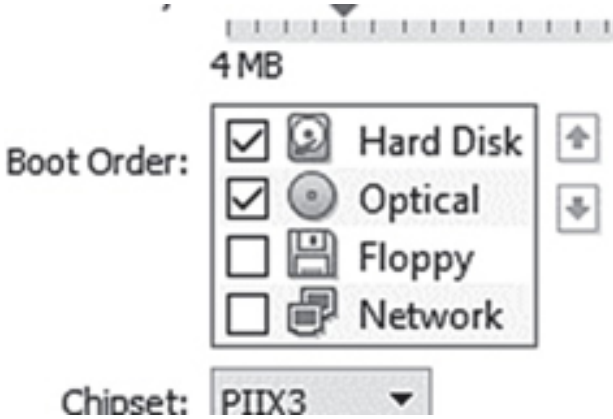


Figure 1-37 VirtualBox Manager | System | Boot Order | Alter

Now power up the VM from the VirtualBox Manager. It will boot the installed OS.

Performing Post-Installation Tasks

25. The system initiates the Initial Setup application upon restart so that you can complete certain post-installation tasks. Figure 1-38 shows the Initial Setup screen with license already accepted.

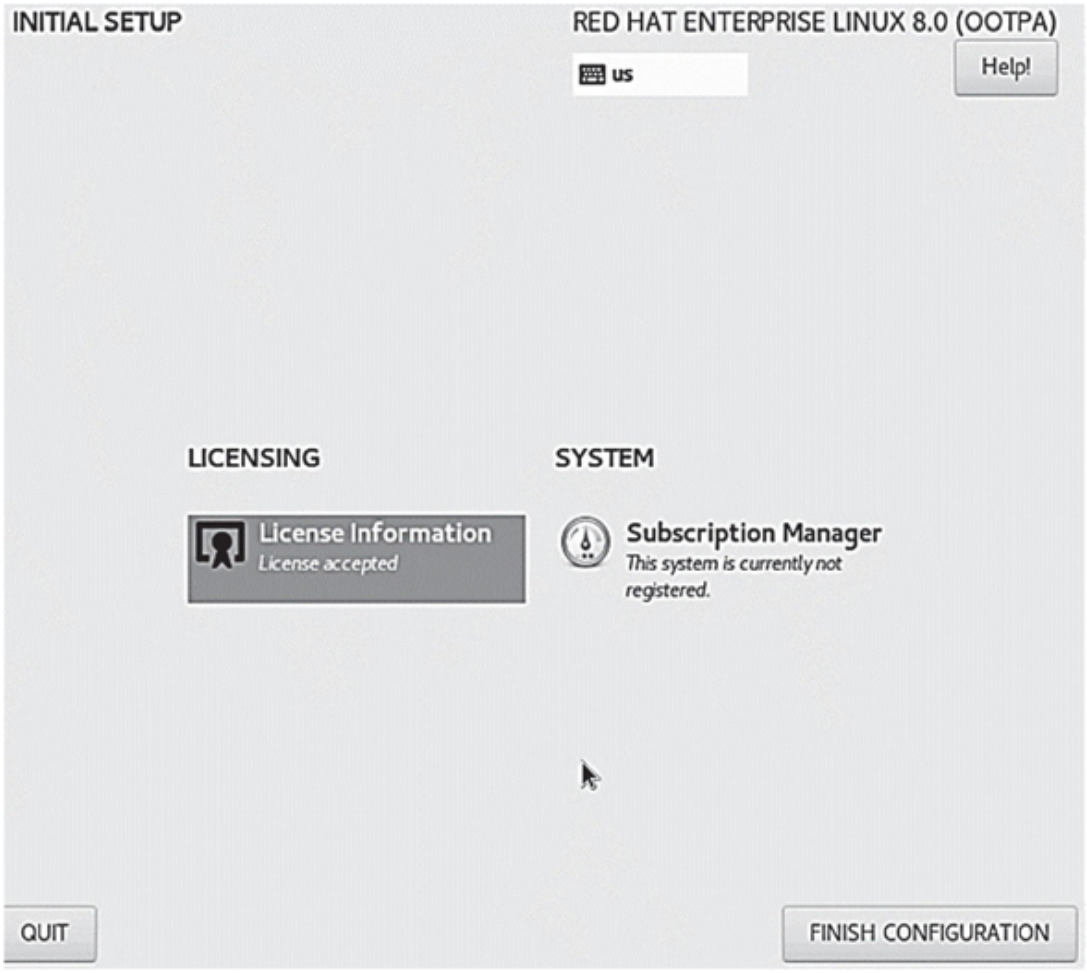


Figure 1-38 Initial Setup

26. On this screen you must click the License Information icon and accept the license terms and conditions by ticking the box beside “I accept the license agreement”. Click Done to return to the Initial Setup screen.
27. The second item on the Initial Setup screen helps you register this system with Red Hat’s subscription management service to enable it to receive automatic software updates and perform certain other management tasks. Leave this item intact for this demonstration and click Finish Configuration.

This brings the installation and initial configuration of RHEL 8 to a successful completion.

Logging In and Out

Now that the installation is complete, you can log on to the system. You selected the Server with GUI base environment, which includes graphical desktop support to interact with the system. You also entered credentials for a user account, *user1*, during installation. You can now use this account to log in.

Logging In for the First Time

When you sign in with a new user account for the first time, several screens appear in succession to allow you to configure a few basic items for the user. Follow the steps below to go through this process. You will not be prompted again for this configuration upon next logon.

1. On the graphical logon screen, click *user1* and enter the password when prompted.

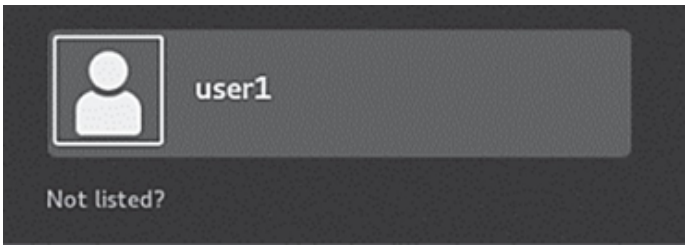


Figure 1-39 Graphical Desktop | Sign-in Screen

The login process continues, and a Welcome screen pops up that shows the language that you had selected at the beginning of the installation. You can change it to a different language if you so desire. Click Next.

2. Add an Input Source to be used on the next screen. The default is the US English keyboard type that you had selected earlier. Click Next.
3. Enable the geographical location of this system to be automatically determined for applications to use. The default is set to on. Click Next.
4. You can connect one of your online user accounts such as Google, Nextcloud, Microsoft, or Facebook in order to

access your email, contact, and other information and services. Click Skip for this demonstration.

5. Click “Start using Red Hat Enterprise Linux” on the final “Ready to Go” screen, Figure 1-40.

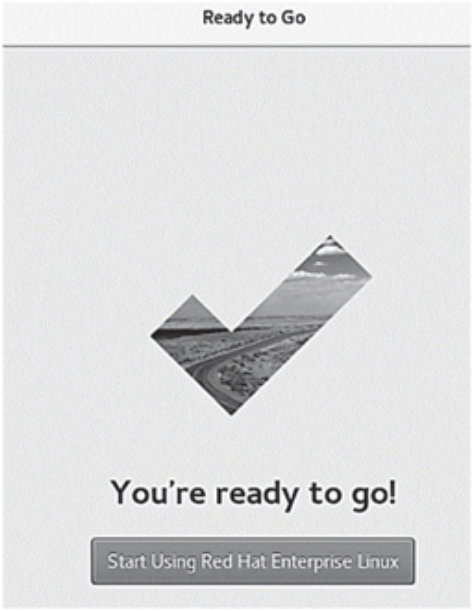


Figure 1-40 First Time User Login

6. A Getting Started help screen pops up (Figure 1-41). Here, you can watch videos on how to launch applications, switch tasks, and use windows and workspaces in the *Graphical User Interface* (GUI). The help is available in text format as well. You can close this window by clicking the x (exit) button at the top right.

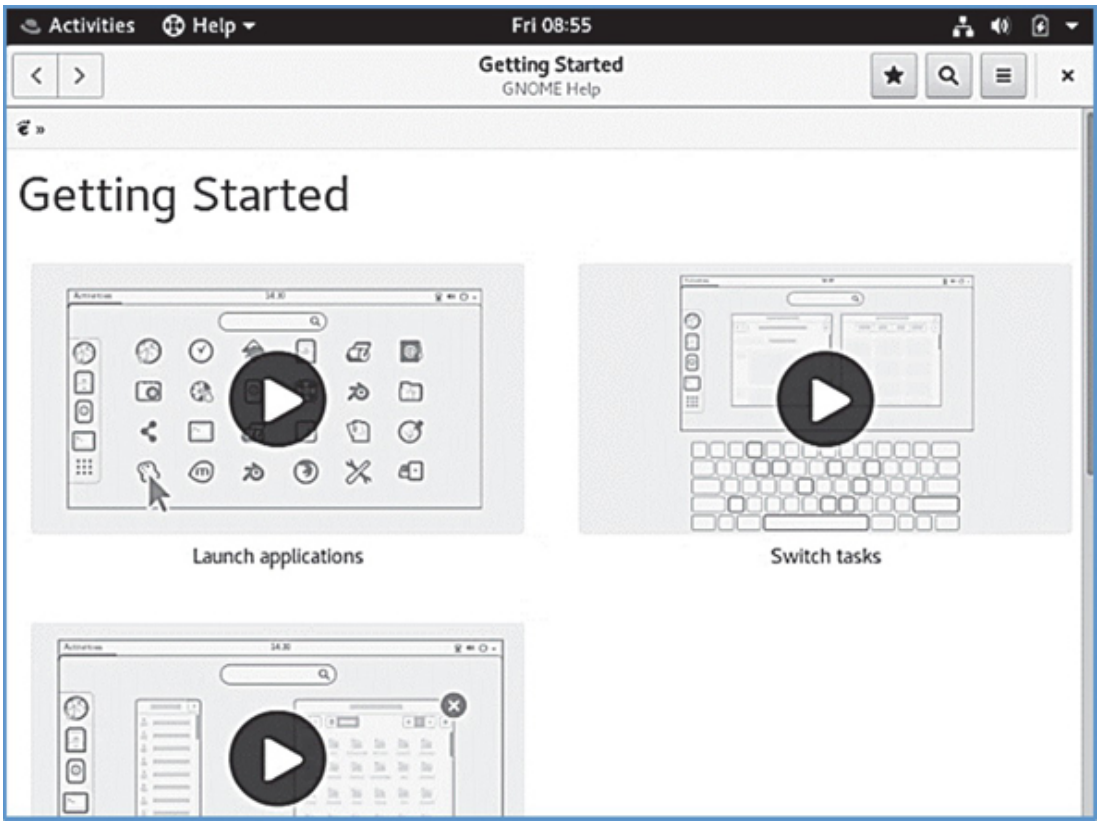


Figure 1-41 First Time User Login | Getting Started

7. The default graphical desktop included in RHEL 8 is the GNOME desktop environment (Figure 1-42). You should now be able to start using the system as *user1*.

Figure 1-42 GNOME Desktop Environment

GNOME stands for *GNU Network Object Model Environment*. It is the default graphical display manager and desktop environment for users in RHEL 8. Chapter 02 “Initial Interaction with the System” provides more details on this topic.

Logging Out

8. Logging out of the system is easy. Click on the down arrow (top right), expand *user1* (the name of the logged-in user), and click Log Out. See Figure 1-43. The user will be signed out and the main login screen will reappear.

Figure 1-43 GNOME Desktop Environment| Log Out

Chapter 02 will explore how to navigate within the GNOME desktop environment, execute basic Linux commands at the command prompt, and obtain necessary help.

Chapter Summary

In this chapter, we started by looking at Linux history and exploring available versions of Linux from Red Hat. We examined various pre-installation items for our lab environment to prepare for a smooth installation in order to practice the exercises and labs presented in this book. We demonstrated downloading the images for VirtualBox Manager software and RHEL 8. We built a virtual machine and installed RHEL 8 in it. We completed post-installation tasks to conclude the demonstration. Finally, we logged in to the new system at the console to verify the installation.

Check Your Understanding

1. The minimal Install base environment includes the graphical support. True or False?
2. Can you install RHEL 8 in text mode?
3. You can use the */boot* partition within LVM to boot RHEL. True or False?
4. Which kernel version is the initial release of RHEL 8 based on?
5. Several log files are created and updated in the */tmp* directory during the installation process. Where are these files moved to after the completion of installation?
6. Name the RHEL installer program.
7. How many console screens do you have access to during the installation process?
8. RHEL 8 may be downloaded from Red Hat's developer site. True or False?
9. RHEL 8 cannot be installed over the network. True or False?
10. What is the name of the default graphical user desktop if Server with GUI is installed?

Answers to Check Your Understanding

1. False. Minimal Install base environment does not include graphics support.
2. Yes, RHEL 8 can be installed in text mode.
3. False. */boot* cannot reside within LVM.
4. The initial release of RHEL 8 is based on kernel version 4.18.
5. These files are moved to the */var/log* directory.
6. The name of the RHEL installer program is Anaconda.
7. There are six console screens available to you during the installation process.
8. True. RHEL 8 may be downloaded from *developers.redhat.com*. You need to open a new account or use an existing before you can download it.

9. False. RHEL 8 can be installed with installation files located on a network server.
10. The default graphical desktop is called GNOME desktop environment.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 1-1: Build RHEL8-VM2 (server2)

Create another virtual machine called *RHEL8-VM2* in VirtualBox, attach the ISO image to it, and install RHEL 8. Use the configuration provided in “What is Needed for the Lab?” and follow the procedures outlined in Exercises 1-1 and 1-2.

Chapter 02

Initial Interaction with the System

This chapter describes the following major topics:

- Interact with display manager and understand graphical interface
- Overview of Linux directory structure
- Recognize top-level directories
- Understand command construct
- Describe and run basic Linux commands
- Obtain help using multiple native tools and RHEL documentation

RHCSA Objectives:

01. Access a shell prompt and issue commands with correct syntax
11. Locate, read, and use system documentation including man, info, and files in /usr/share/doc

Wayland is an advanced display protocol that sets up the foundation for running graphical applications, which includes system administration tools, user applications, as well as Linux graphical display and desktop manager programs. Working in a graphical environment to interact with the system is convenient for users with limited command line knowledge or specific requirements.

Linux files are organized logically for ease of administration. This file organization is maintained in hundreds of directories located in larger containers (file systems). Red Hat Enterprise Linux follows the File system Hierarchy Standard for file organization, which describes names, locations, and permissions for many file types and directories.

Linux offers a variety of commands for users and system managers. User commands are general purpose that are intended for execution by any user on the system. However, system management commands require elevated privileges of the superuser. Knowledge of these tools is essential for productive usage and efficient administration of the system. This chapter provides an analysis of command components and how to construct a command. Following that, it introduces a few basic user-level commands.

The availability of native help on the system simplifies task execution for Linux users and system administrators alike. This assistance is available on commands and configuration files via locally installed searchable manual pages and documentation for installed packages. In addition, Red Hat documentation website provides a wealth of information on various topics, procedures, and command usage.

Linux Graphical Environment

RHEL allows users to work in both text and graphical environments. Text interface might be cumbersome, but many administrators and programmers prefer to work in a text-mode setting without needing graphics capabilities. Nevertheless, a graphical environment provides easier and convenient interaction with the OS by hiding the challenges that users may otherwise experience when working in text-mode.

Wayland is a client/server display protocol that sets up the foundation for running graphical programs and applications in RHEL 8. It is available alongside the legacy X Window System, which has been around in RHEL for decades. Wayland provides superior graphics capabilities, features, and performance than X. There are two components that are critical to the functionality of a graphical environment: the *display manager* (a.k.a. *login manager*) and the *desktop environment*. Both are launched following the completion of the groundwork established by Wayland.

Display/Login Manager

A display/login manager handles the presentation of graphical login screen. It allows users to enter credentials to log on to the system. A preconfigured graphical *desktop manager* appears after the credentials are verified. In RHEL 8, the default display manager is called *GNOME Display Manager* (GDM). Figure 2-1 provides an image of GDM.

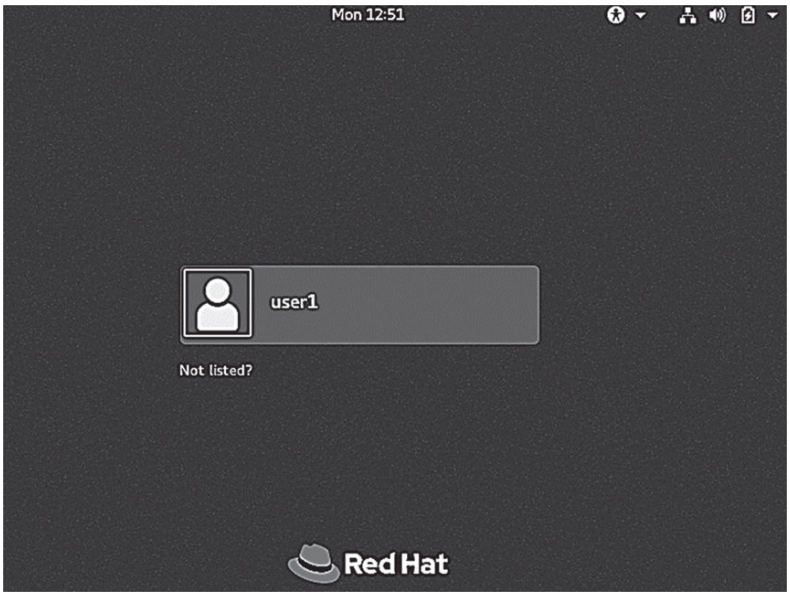


Figure 2-1 GNOME Display Manager

The login screen presents a list of all normal user accounts that exist on the system. You can log in as any one of them by selecting the desired account. If you wish to sign in as an unlisted user or the *root* user, click “Not Listed?” and enter the username and password for the desired account. The current system day and time also appear at the top of the login screen.

There are two downward arrowheads at the top right of the login screen. The arrowhead on the left is to enable or disable an accessibility feature. The arrowhead on the right allows you to power off or reboot the system and change the system volume. More controls become available after you have logged in. There are three additional icons at the top right that show the network connectivity, sound level, and battery/power status.

Desktop Environment

Once the credentials are validated for a user, the display/login manager establishes a *Desktop Environment* (DE) to work in. RHEL 8 comes with several graphical desktop software with GNOME desktop environment set as the default. It provides an easy and point-and-click GUI for users to run programs and

operating system tools. Figure 2-2 is an image of the default GNOME desktop environment for *root*.

Figure 2-2 GNOME Desktop Environment

If you have worked with Microsoft Windows, you should have no difficulty using this desktop environment. The default screen has an Activities icon at the top left, which allows you to search and access programs. Figure 2-3 depicts a list of application icons when you click on Activities.

Figure 2-3 GNOME Desktop Environment | Activities

These application icons represent Firefox web browser, file manager, software updates, GNOME help, and shell terminal. The icon with nine dots displays all available programs, including Settings. The Settings application includes administrative and user-level controls to view or modify configuration items such as Wi-Fi, Bluetooth, desktop background, notifications, regional settings, privacy, sound, power, screensaver, network, and more.

Linux Directory Structure and File Systems

Linux files are organized logically in a hierarchy for ease of administration and recognition. This organization is maintained in hundreds of directories located in larger containers called *file systems*. Red Hat Enterprise Linux follows the *Filesystem Hierarchy Standard* (FHS) for file organization, which describes names, locations, and permissions for many file types and directories.

Linux file systems contain files and subdirectories. A subdirectory, also referred to as a *child* directory, is located under a *parent* directory. The parent directory is a subdirectory of a higher-level directory. The Linux directory structure is analogous to an inverted tree, where the top of the tree is the root of the directory, tree branches are subdirectories, and leaves are files. The root of the directory is represented by the forward slash character (*/*), and this is where the entire directory structure is ultimately connected. The forward slash is also used as a directory separator in a path, such as */etc/rc.d/init.d/functions*.

In this example, the *etc* subdirectory is located under */*, making *root* the parent of *etc* (which is a child), *rc.d* (child) is located under *etc* (parent), *init.d* (child) is located under *rc.d* (parent), and *functions* (leaf) is located under *init.d* (parent) at the bottom.

The term *subdirectory* is used for a directory that has a parent directory.

Each directory has a parent directory and a child directory, with the exception of the root and the lowest level subdirectories. The root directory has no parent, and the lowest level subdirectory has no child.

Top-Level Directories

The key top-level directories under the / are shown in Figure 2-4. Some of these directories hold *static* data, while others contain *dynamic* (or *variable*) information. Static data refers to file content that remains unchanged unless modified explicitly. Dynamic or variable data, in contrast, refers to file content that is

modified and updated as required by system processes. Static directories normally contain commands, configuration files, library routines, kernel files, device files, etc., and dynamic directories contain log files, status files, temporary files, etc.

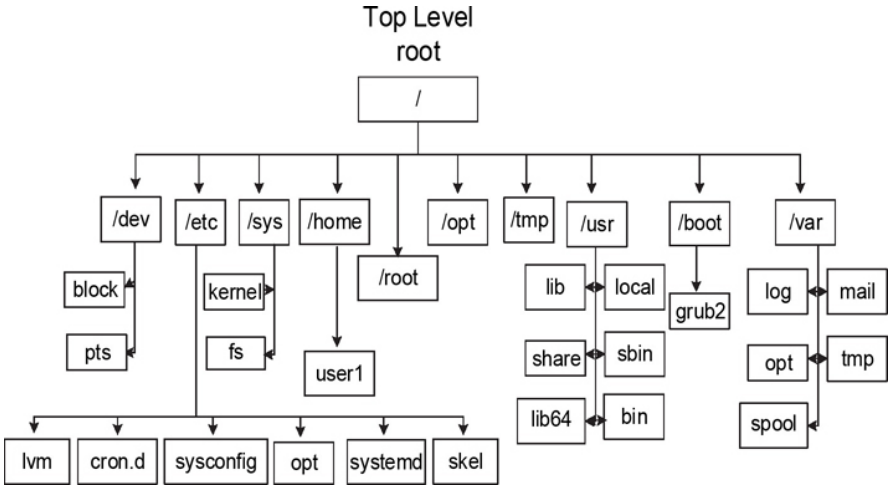


Figure 2-4 Linux Directory Structure

The hierarchical directory structure keeps related information together in a logical fashion. Compare this concept with a file cabinet containing several drawers, with each drawer storing multiple file folders.

File System Categories

There are a variety of file system types supported in RHEL that can be categorized in three basic groups: *disk-based*, *network-based*, and *memory-based*. Disk-based file systems are typically created on physical media such as a hard drive or a USB flash drive. Network-based file systems are essentially disk-based file systems that are shared over the network for remote access. Memory-based file systems are virtual; they are created automatically at system startup and destroyed when the system goes down. The first two types of file systems store information persistently, while any data saved in virtual file systems is lost at system reboots.

During RHEL installation, two disk-based file systems are created when you select the default partitioning. These file systems are

referred to as the root and boot file systems. Furthermore, several memory-based file systems are vital to the operation of a RHEL system.

The Root File System (/), Disk-Based

The root directory is the top-level file system in the FHS and contains many higher-level directories that store specific information. Some of the key directories are:

/etc: The *etcetera* (or *extended text configuration*) directory holds system configuration files. Some common subdirectories are *systemd*, *sysconfig*, *lvm*, and *skel*, which comprise configuration files for systemd, most system services, the Logical Volume Manager, and per-user shell startup template files, respectively.

/root: This is the default home directory location for the *root* user.

/mnt: This directory is used to mount a file system temporarily.

The size of the root file system is automatically determined by the installer program based on the available disk space when you select the default partitioning; however, it may be altered if required.

The Boot File System (/boot) – Disk-Based

The */boot* file system contains the Linux kernel, boot support files, and boot configuration files. Just like the *root* file system, the size of this file system is also automatically determined by the installer program based on the available disk space when you select the default partitioning; however, it may be set to a different size during or after the installation if required.

The Home Directory (/home)

The */home* directory is designed to store user *home* directories and other user contents. Each user is assigned a home directory

to save personal files, and the user can block access to other users.

The Optional Directory (/opt)

This directory can be used to hold additional software that may need to be installed on the system. A subdirectory is created for each installed software.

The UNIX System Resources Directory (/usr)

This directory contains most of the system files. Some of the important subdirectories are:

/usr/bin: The *binary* directory contains crucial user executable commands.

/usr/sbin: Most commands are required at system boot, and those that require the *root* user privileges in order to run are located in this *system binary* directory. In other words, this directory contains crucial system administration commands that are not intended for execution by normal users (although they can still run a few of them). This directory is not included in the default search path for normal users because of the nature of data it holds.

/usr/lib and **/usr/lib64:** The *library* directories contain shared library routines required by many commands and programs located in the */usr/bin* and */usr/sbin* directories, as well as by the kernel and other applications and programs for their successful installation and operation. The */usr/lib* directory also stores system initialization and service management programs. The subdirectory */usr/lib64* contains 64-bit shared library routines.

/usr/include: This directory contains header files for *C* language.

/usr/local: This directory serves as a system administrator repository for storing commands and tools downloaded from the web, developed in-house, or obtained elsewhere. These commands and tools are not generally included with the original Linux distribution. In particular, */usr/local/bin* holds executables, */usr/local/etc* holds configuration files, and */usr/local/lib* and */usr/local/lib64* holds library routines.

/usr/share: This is the directory location for manual pages, documentation, sample templates, configuration files, etc., that may be shared with other Linux platforms.

/usr/src: This directory is used to store source code.

The Variable Directory (/var)

The */var* directory contains data that frequently changes while the system is operational. Files in this directory contain log, status, spool, lock, and other dynamic data. Some common subdirectories under */var* are:

/var/log: This is the storage for most system log files, such as system logs, boot logs, user logs, failed user logs, installation logs, cron logs, mail logs, etc.

/var/opt: This directory stores log, status, and other variable data files for additional software installed in */opt*, t.

/var/spool: Directories that hold print jobs, cron jobs, mail messages, and other queued items before being sent out to their intended destinations are located here.

/var/tmp: Large temporary files or temporary files that need to exist for longer periods of time than what is typically allowed in another temporary directory, */tmp*, are stored here. These files survive system reboots and are automatically deleted if they are not accessed or modified for a period of 30 days.

The Temporary Directory (/tmp)

This directory is a repository for temporary files. Many programs create temporary files here during runtime or installation. These files survive system reboots and are automatically removed if they are not accessed or modified for a period of 10 days.

The Devices File System (/dev), Virtual

The Devices (*dev file system*) file system is accessible via the */dev* directory, and it is used to store device nodes for physical hardware and virtual devices. The Linux kernel communicates with these devices through corresponding device nodes located here. These device nodes are automatically created and deleted by the *udevd* service (a Linux service for dynamic device management) as necessary.

There are two types of device files: *character* (or *raw*) device files, and *block* device files. The kernel accesses devices using one of these files or both.

Character devices are accessed serially with streams of bits transferred during kernel and device communication. Examples of such devices are console, serial printers, mice, keyboards, terminals, etc.

Block devices are accessed in a parallel fashion with data exchanged in blocks (parallel) during kernel and device communication. Data on block devices is accessed randomly. Examples of block devices are hard disk drives, optical drives, parallel printers, etc.

The Procfs File System (/proc), Virtual

The Procfs (*process file system*) file system is accessible via the */proc* directory, and it is used to maintain information about the current state of the running kernel. This includes the details for current hardware configuration and status information on CPU, memory, disks, partitioning, file systems, networking, running processes, and so on. This information is stored in a hierarchy of subdirectories that contain thousands of zero-length pseudo files.

These files point to relevant data maintained by the kernel in the memory. This virtual directory structure simply provides an easy interface to interact with kernel-maintained information. The Procfs file system is dynamically managed by the system.

The contents in */proc* are created in memory at system boot time, updated during runtime, and destroyed at system shutdown.

The Runtime File System (/run), Virtual

This virtual file system is a repository of data for processes running on the system. One of its subdirectories, */run/media*, is also used to automatically mount external file systems such as those that are on optical (CD and DVD) and flash USB.

The contents of this file system are automatically deleted at system shutdown.

The System File System (/sys), Virtual

Information about hardware devices, drivers, and some kernel features is stored and maintained in the */sys* file system. This information is used by the kernel to load necessary support for the devices, create device nodes in */dev*, and configure the devices. This file system is auto-maintained as well.

Viewing Directory Hierarchy

The *tree* command lists a hierarchy of directories and files. There are a number of options with this command that can be specified to include additional information. Table 2-1 describes some common options.

Option	Description
-a	Includes hidden files in the output
-d	Excludes files from the output
-h	Displays file sizes in human-friendly format
-f	Prints the full path for each file
-p	Includes file permissions in the output

Table 2-1 tree Command Options

Let’s try to understand the usage of the *tree* command with the help of some examples:

To list only the directories (-d) in the *root* user’s home directory (*/root*):

```
[root@server1 ~]# tree -d
.
├── Desktop
├── Documents
├── Downloads
├── Music
├── Pictures
├── Public
├── Templates
└── Videos

8 directories
```

The output above indicates that there are eight directories under */root*.

To list files in the */etc/sysconfig* directory along with their permissions (-p), sizes in human-readable format (-h), and full path (-f):

```
[root@server1 ~]# tree -hpf /etc/sysconfig
/etc/sysconfig
├── [-rw-r--r--   309]  /etc/sysconfig/anaconda
├── [-rw-r--r--   403]  /etc/sysconfig/atd
├── [drwxr-xr-x    43]  /etc/sysconfig/cbq
│   ├── [-rw-r--r--    11]  /etc/sysconfig/cbq/avpkt
│   └── [-rw-r--r--    79]  /etc/sysconfig/cbq/cbq-0000.example
├── [-rw-r--r--    46]  /etc/sysconfig/chronyd
├── [drwxr-xr-x     6]  /etc/sysconfig/console
├── [-rw-r--r--   150]  /etc/sysconfig/cpupower
├── [-rw-r--r--   110]  /etc/sysconfig/crond
├── [-rw-------   417]  /etc/sysconfig/ebtables-config
├── [-rw-r--r--    73]  /etc/sysconfig/firewalld
├── [lrwxrwxrwx    15]  /etc/sysconfig/grub -> ../default/grub
├── [-rw-------  2.1K]  /etc/sysconfig/ip6tables-config
├── [-rw-------  2.1K]  /etc/sysconfig/iptables-config
├── [-rw-r--r--   903]  /etc/sysconfig/irqbalance
├── [-rw-r--r--  1.7K]  /etc/sysconfig/kdump
├
.........
```

The output shows permissions in column 1, sizes in column 2, and full path of the files in column 3.

Run **man tree** at the command prompt to view the manual pages of the *tree* command for additional options and their usage.

Basic System Commands

There are hundreds of commands available in RHEL that range from simple to complex in terms of their construct and usage. Commands can be combined to build complex structures for innovative use cases. Some commands offer a few options, while others have as many as 70 or more. This section furnishes an understanding of how commands are formed and demonstrates the use of some of the basic, common Linux commands. You will learn more commands and their advanced usages throughout this book.

Starting a GNOME Terminal Session

In order to issue commands and run programs at the command prompt, you need access to a terminal session. It you are logged in as the *root* user on the GNOME desktop, you can launch a terminal session by clicking the terminal icon (second icon from the bottom under Activities). Figure 2-5 illustrates a typical terminal window.

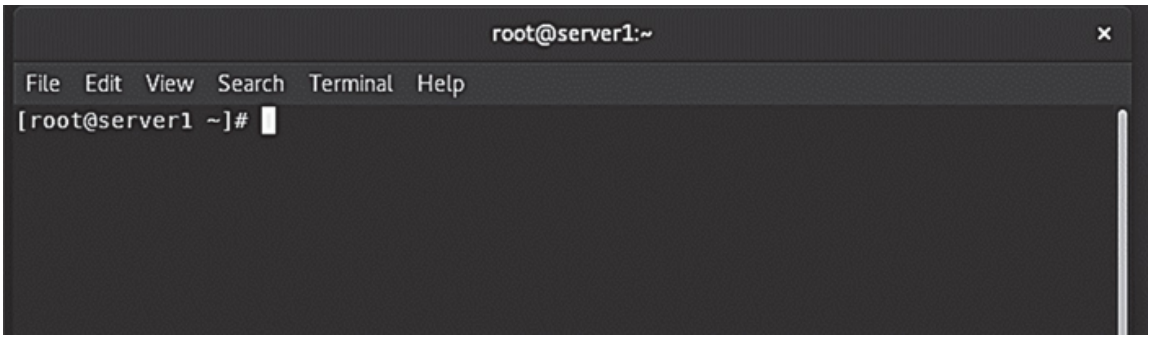


Figure 2-5 GNOME Desktop Environment | Activities | Terminal

The header bar as depicted in Figure 2-5 displays the logged-in username, hostname, and your current directory location. Underneath the header are several menu items that contain sub-items for viewing and customizing the terminal window, copying/pasting text, searching for text, and so on. For instance, you can click Edit | Preferences and change the background color, cursor style/type, screen size, font size, etc.

The "[root@server1 ~]#" is the default representation of the command prompt. It reflects the logged-in username (*root*, *user1*, etc.), hostname of the system (*server1*, *server2*, etc.), and your current directory location, all enclosed within the square brackets []. The command prompt ends with the hash sign (#) for the *root* user or the dollar sign ($) for normal users (*user1*, etc.) outside of the closing square bracket. Commands are typed at the cursor position and executed by pressing the Enter key.

Understanding the Command Mechanics

To practice the commands provided in this chapter, you can log in as *user1*, run the commands, and observe their outputs. However, as you are learning Linux system administration, it's important to feel comfortable working as *root* in the beginning. If something breaks, *server1* and *server2* are built for testing so they can be rebuilt.

The basic syntax of a Linux command is:

command option(s) argument(s)

Options (a.k.a. a *switch* of *flag*) are optional. You can specify zero or more options with a command. Arguments, in contrast, may be optional or mandatory depending on the command and its usage. Many commands have preconfigured default options and arguments. You are not required to specify them. Other commands do require at least one option or argument in order to work. An option modifies the behavior of the command. An argument supplies a target on which to perform the command action.

An option may start with a single hyphen character (-la, for instance), and it is referred to as the short-option format. Each individual letter in this depiction represents a separate option (l and a are two options in -la). This is a frequent format throughout this book.

An option may also begin with two hyphen characters (--all, for instance), and it is referred to as the long-option format. All letters in this representation are collectively identified as a single option (-all is one option).

The following examples express some command structures with a description on the right that states the number of options and arguments supplied:

# **ls**	No option, no explicit argument; the default argument is the current directory name
# **ls -l**	One option, no explicit argument; the default argument is the current directory name
# **ls -al**	Two options, no explicit argument; the default argument is the current directory name
# **ls --all**	One option, no explicit argument; the default argument is the current directory name
# **ls -l directory_name**	One option, one explicit argument

EXAM TIP: Use online help on the usage of a command if needed. Refer to “Getting Help” later in this chapter for more on how to access and use help.

Now let’s take a look at some essential Linux commands and understand their usage.

Listing Files and Directories

One of the most rudimentary commands in Linux is the *ls* (*list*) command. It is used to show the list of files and directories. This command supports a multitude of options, some of which are listed in Table 2-2 along with a short explanation.

Option	Description
-a	Includes hidden files and directories in the output. A file or directory name that begins with the period character (.) is considered hidden.
-l	Displays long listing with detailed file information including the file type, permissions, link count, owner, group, size, date and time of last modification, and name of the file
-ld	Displays long listing of the specified directory but hides its contents
-lh	Displays long listing with file sizes shown in human-friendly format
-lt	Lists all files sorted by date and time with the newest file first
-ltr	Lists all files sorted by date and time with the oldest file first (reverse)
-R	Lists contents of the specified directory and all its subdirectories (recursive listing)

Table 2-2 ls Command Options

A grasp of the usage of this command and the output it produces is important. The following examples will illustrate the impact of options used with the *ls* command. Again, log in as the *root* user, if you are not already, to run these examples.

To list files in the current directory with the assumption that you are in the */root* directory:

```
[root@server1 ~]# ls
anaconda-ks.cfg  Documents  initial-setup-ks.cfg  Pictures  Templates
Desktop          Downloads  Music                 Public    Videos
```

To list files in the current directory with detailed information:

```
[root@server1 ~]# ls -l
total 8
-rw-------. 1 root root 1447 Aug 22 15:27 anaconda-ks.cfg
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Desktop
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Documents
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Downloads
-rw-r--r--. 1 root root 1602 Aug 23 06:23 initial-setup-ks.cfg
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Music
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Pictures
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Public
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Templates
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Videos
```

The long listing in the output above furnishes a unique piece of information about the file or directory in nine discrete columns:

Column 1: The first character (hyphen or d) divulges the file type, and the next nine characters (rw-rw-r--) indicate permissions.
Column 2: Displays the number of links (links are explained later in this chapter)
Column 3: Shows the owner name
Column 4: Exhibits the owning group name
Column 5: Identifies the file size in bytes. For directories, this number reflects the number of blocks being used by the directory to hold information about its contents.
Columns 6, 7, and 8: Displays the month, day, and time of creation or last modification
Column 9: Indicates the name of the file or directory

As an alternative to **ls -l**, you may use its shortcut **ll** for brevity and convenience unless there is a specific need to use the former.

To show the long listing of only the specified directory without showing its contents:

```
[root@server1 ~]# ls -ld /usr
drwxr-xr-x. 12 root root 144 Aug 22 15:06 /usr
```

To display all files in the current directory with their sizes in human-friendly format:

```
[root@server1 ~]# ls -lh
total 8.0K
-rw-------. 1 root root 1.5K Aug 22 15:27 anaconda-ks.cfg
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Desktop
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Documents
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Downloads
-rw-r--r--. 1 root root 1.6K Aug 23 06:23 initial-setup-ks.cfg
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Music
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Pictures
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Public
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Templates
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Videos
```

To list all files, including the hidden files, in the current directory with detailed information:

```
[root@server1 ~]# ls -la
total 44
dr-xr-x---. 15 root root 4096 Aug 30 23:17 .
dr-xr-xr-x. 17 root root  224 Aug 22 15:06 ..
-rw-------.  1 root root 1447 Aug 22 15:27 anaconda-ks.cfg
-rw-------.  1 root root 1044 Aug 30 23:36 .bash_history
-rw-r--r--.  1 root root   18 Aug 12  2018 .bash_logout
-rw-r--r--.  1 root root  176 Aug 12  2018 .bash_profile
-rw-r--r--.  1 root root  176 Aug 12  2018 .bashrc
drwx------. 10 root root  230 Aug 26 12:59 .cache
drwx------. 11 root root  215 Aug 26 12:59 .config
-rw-r--r--.  1 root root  100 Aug 12  2018 .cshrc
drwx------.  3 root root   25 Aug 22 20:46 .dbus
........
```

To repeat the previous example with the output sorted by date and time with the newest file first:

```
[root@server1 ~]# ls -lt
total 8
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Videos
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Desktop
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Documents
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Downloads
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Music
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Pictures
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Public
drwxr-xr-x. 2 root root    6 Aug 26 12:56 Templates
-rw-r--r--. 1 root root 1602 Aug 23 06:23 initial-setup-ks.cfg
-rw-------. 1 root root 1447 Aug 22 15:27 anaconda-ks.cfg
```

To list contents of the */etc* directory recursively:

```
[root@server1 ~]# ls -R /etc
/etc:
adjtime                     hostname                  profile.d
aliases                     hosts                     protocols
alsa                        hosts.allow               pulse
alternatives                hosts.deny                qemu-ga
anacrontab                  hp                        qemu-kvm
asound.conf                 idmapd.conf               radvd.conf
at.deny                     init.d                    ras
audit                       inittab                   rc0.d

.........
```

Run **man ls** at the command prompt to view the manual pages of the *ls* command with all the options it supports and how to use them.

Printing Working Directory

The *pwd* (*print working directory* or *present working directory*) command displays a user's current location in the directory tree. The following example shows that *root* is currently in the */root* directory:

```
[root@server1 ~]# pwd
/root
```

/root is the home directory for the *root* user. The *pwd* command always returns the absolute path to a file or directory.

Navigating Directories

Files are placed in various directories in Linux, and there are tens of thousands of them. Each file and directory is recognized by a unique path in the directory tree. A *path* (or *pathname*) is like a road map that shows you how to get from one place in the directory hierarchy to another. It uniquely identifies a particular file or directory by its absolute or relative location in the directory structure.

An *absolute path* (a.k.a. a *full path* or a *fully qualified pathname*) points to a file or directory in relation to the top of the directory tree. It always starts with the forward slash (*/*). You can use the *pwd* command to view your current absolute path in the tree:

```
[root@server1 ~]# pwd
/root
```

The output indicates that */root* is the current location for the *root* user in the directory hierarchy, and the leading / identifies this location as a full path.

A *relative path*, on the other hand, points to a file or directory in relation to your current location. This file path never begins with the forward slash (*/*). It may begin with two period characters (..) or with a subdirectory name without a leading */*, such as *etc/sysconfig*.

It's easy to navigate in the directory hierarchy if you have a good understanding of the absolute and relative paths and the key difference between them. Let's run a couple of examples using the *cd* (*change directory*) command, which is used to switch between directories, and verify the result with the *pwd* command.

To determine the current location and then go one level up into the parent directory using the relative path:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# cd ..
[root@server1 /]# pwd
/
```

The above sequence of commands display the current location (*/root* output of *pwd*), then move one level up (*cd ..* is relative to the current location), and finally verify the new location (*/* output of *pwd*). You may want to use the absolute path (**cd /**) instead of (**cd ..**) to go to the top of the directory tree (parent directory of */root*).

Now, let's switch to the directory *sysconfig*, which is located under */etc*. There are two options: the absolute path (*/etc/sysconfig*), or the relative path (*etc/sysconfig*):

```
# cd /etc/sysconfig
```
or
```
# cd etc/sysconfig
```

To change into the */usr/bin* directory, for instance, you can run either of the following:

```
# cd /usr/bin
```
or
```
# cd ../../usr/bin
```

The above example indicates that the absolute path can be used to switch into a target directory regardless of your current location in the hierarchy. However, a relative path must be entered based on your current location in order to move to a target directory.

At this point, if you want to return to your home directory, you can simply run the *cd* command without inputting a path or by supplying the *tilde* character (~) with it. Either of the two will produce the desired result.

```
# cd        or        # cd ~
```

In the above example, you could also use the absolute path (**cd /root**) or a relative path (**cd ../../root**) instead.

To switch between the current and previous directories, issue the *cd* command with the hyphen character (-). See the following example and observe the output:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# cd -
/usr/bin
[root@server1 bin]# cd -
/root
[root@server1 ~]# pwd
/root
```

The example shows the use of the hyphen character (-) with the *cd* command to switch between the current and previous directories.

Identifying Terminal Device File

Linux allocates unique *pseudo* (or *virtual*) numbered device files to represent terminal sessions opened by users on the system. It uses these files to communicate with individual sessions. By default, these files are stored in the */dev/pts* (*pseudo terminal session*) directory. These files are created by the system when a user opens a new terminal session and they are removed on its closure. The destroyed files are recreated and reused for new terminal sessions.

Linux provides a command called *tty* (*teletype*) to identify your current active terminal session. Here is an example:

```
[root@server1 ~]# tty
/dev/pts/0
```

The output discloses the filename “0” and its location (*/dev/pts* directory).

Inspecting System’s Uptime and Processor Load

The *uptime* command is used to display the system’s current time, length of time it has been up for, number of users currently logged in, and the average CPU (processing) load over the past 1, 5, and 15 minutes. See the following output:

```
[root@server1 ~]# uptime
 08:22:25 up 1 day, 23:58,  2 users,  load average: 0.00, 0.00, 0.00
```

The output shows the current system time (08:22:25), up duration (1 day, 23 hours, and 58 minutes), number of logged-in users (2), and the CPU load averages over the past 1, 5, and 15 minutes (0.00, 0.00, and 0.00), respectively.

The load average numbers correspond to the percentage of CPU load with 0.00 and 1.00 represent no load and full load, and a number greater than 1.00 signifies excess load (over 100%).

Clearing the Screen

The *clear* command clears the terminal screen and places the cursor at the top left of the screen. This command is useful to clear the screen of any distractive content and run new commands on a clean slate.

```
# clear
```

You can also use Ctrl+l for this command.

Determining Command Path

RHEL provides a set of tools that can be used to identify the absolute path of the command that will be executed when you

run it without specifying its full path. These tools are the *which*, *whereis*, and *type* commands. The following examples show the full location of the *ls* command:

```
[root@server1 ~]# which ls
/usr/bin/ls
[root@server1 ~]# whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
[root@server1 ~]# type ls
ls is hashed (/usr/bin/ls)
```

As shown above, all three commands responded with an identical path location for the *ls* command, which is */usr/bin/ls*. This implies that there is no need to type the entire path */usr/bin/ls* to run the *ls* command, as the system will automatically determine the location of *ls* based on some predefined settings. Refer to Chapter 07 “The Bash Shell” for more guidance.

Viewing System Information

There are several elements in the RHEL system that identify various information regarding the operating system, hardware, kernel, storage, networking, and so on. The *uname* command identifies elementary information about the system including its hostname. Without any options, the output of this command is restricted to displaying the operating system name only; however, it reports other details by adding the -a option.

```
[user1@server1 ~]$ uname
Linux
[user1@server1 ~]$ uname -a
Linux server1.example.com 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 20
19 x86_64 x86_64 x86_64 GNU/Linux
```

The data returned by the second command above is elaborated below:

Linux	Kernel name
server1.example.com	Hostname of the system
4.18.0-80.el8.x86_64	Kernel release
#1 SMP Wed Mar 13 12:02:46 UTC 2019	Date and time of the kernel built
x86_64	Machine hardware name
x86_64	Processor type
x86_64	Hardware platform

GNU/Linux	Operating system name

Try running the *uname* command with the -s (kernel name), -n (node name), -r (kernel release), -v (kernel build date), -m (hardware name), -p (processor type), -i (hardware platform), and -o (OS name) options separately to view specific information.

Viewing CPU Specs

A CPU has many architectural pieces that can be looked at using the *lscpu* command. These pieces include the CPU architecture, its operating modes, vendor, family, model, speed, cache memory, and whether it supports virtualization. The following example shows the CPU information from *server1*:

```
[root@server1 ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 58
Model name:            Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz
Stepping:              9
CPU MHz:               2294.784
BogoMIPS:              4589.56
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cm
ov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_
good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq monitor ssse3
 cx16 pcid sse4_1 sse4_2 x2apic popcnt aes xsave avx rdrand hypervisor lahf_lm p
ti fsgsbase flush_l1d
```

The output indicates the architecture of the CPU (x86_64), supported modes of operation (32-bit and 64-bit), sequence number of the CPU on this system (1), threads per core (1), cores per socket (1), number of sockets (1), vendor ID (GenuineIntel), CPU model (58) model name (Intel …), speed (2294.784 MHz), the amount and levels of cache memory (L1d, L1i, L2, and L3), and other information.

Getting Help

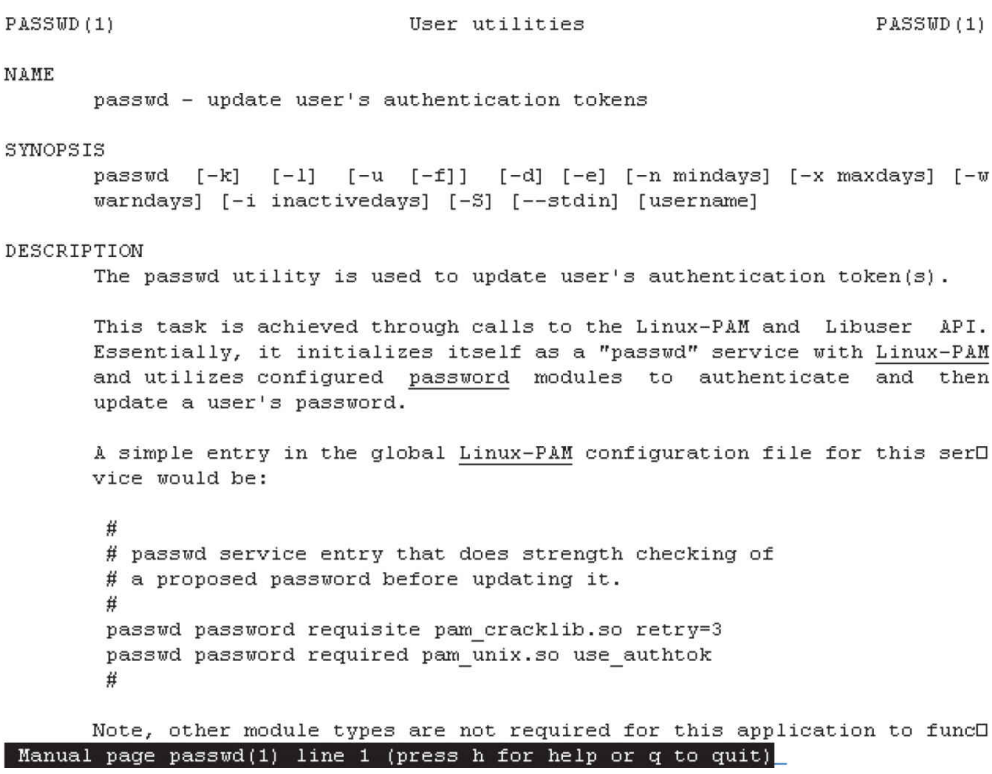
While working on the system, you may require help to obtain information about a command or a configuration file. RHEL offers online help via *manual* pages. Manual pages are online documentation that provides details on commands, configuration files, etc. They are installed under the */usr/share/man* directory when associated software packages are installed.

In addition to the manual pages, *apropos*, *whatis*, *info*, and *pinfo* commands as well as documentation located in the */usr/share/doc* directory are also available on the system.

> **EXAM TIP:** If you need help with a command or configuration file, do not hesitate to use the man pages, refer to the documentation available in the /usr/share/doc directory, or employ one of the other help tools.

Accessing Manual Pages

Use the *man* command to view manual pages. The following example shows how to check manual pages for the *passwd* command:



The output returns the name of the command, the section of the manual pages it is documented in within the parentheses, and the type (User utilities) of the command on line 1. It then shows a short description (NAME), the command's usage (SYNOPSIS), and a long description (DESCRIPTION), followed by a detailed explanation of each option that the command supports and other relevant data. The highlighted line at the bottom indicates the line number of the manual page. Press h to get help on navigation, press q to quit and return to the command prompt, use the Up and Down arrow keys to scroll up and down, and the PgUp and PgDn keys to scroll one page at a time.

Table 2-3 summarizes the six keys described above and introduces a few more to assist in navigation.

Key	Action
Enter / Down arrow	Moves forward one line
Up arrow	Moves backward one line
f / Spacebar / Page down	Moves forward one page
b / Page up	Moves backward one page
d / u	Moves down / up half a page
g / G	Moves to the beginning / end of the man pages
:f	Displays line number and bytes being viewed
q	Quits the man pages
/pattern	Searches forward for the specified pattern
?pattern	Searches backward for the specified pattern
n / N	Finds the next / previous occurrence of a pattern
h	Gives help on navigational keys

Table 2-3 Navigating within Manual Pages

Run *man* on any of the commands that you have reviewed so far and navigate using the keys provided in Table 2-3 for practice.

Headings in the Manual

Each page in the manual organizes information under several headings. Some common headings are NAME (of the command or

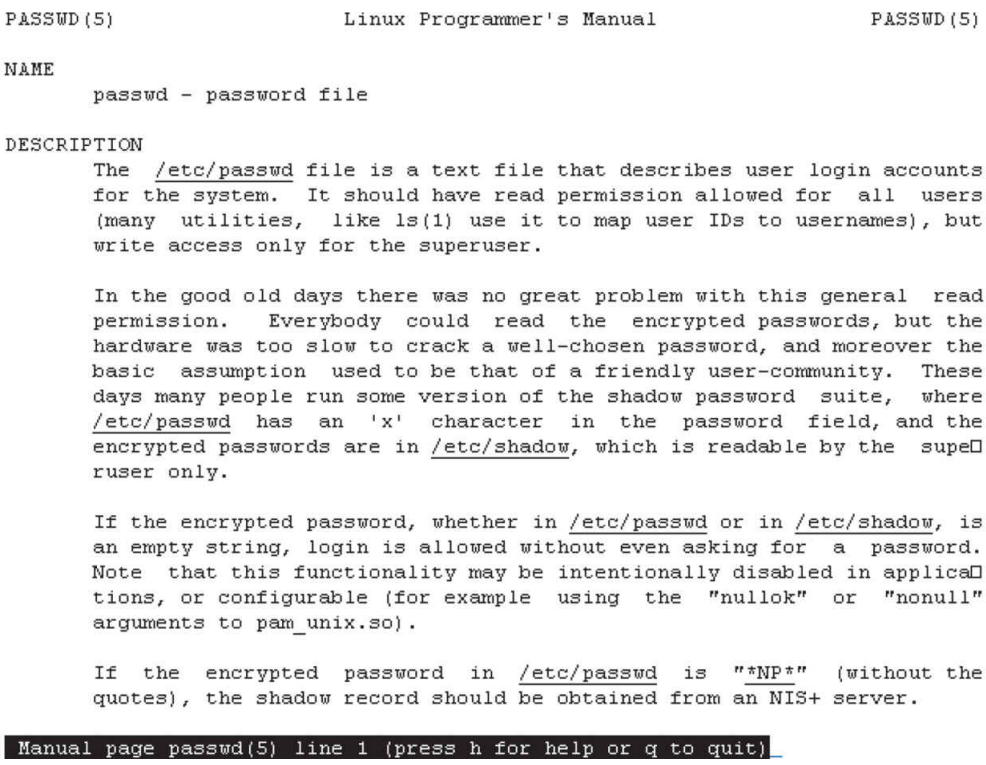
file with a short description), SYNOPSIS (syntax summary), DESCRIPTION (an overview of the command or file), OPTIONS (options available for use), EXAMPLES (some examples to explain the usage), FILES (a list of related files), SEE ALSO (reference to other manual pages or topics), BUGS (any reported bugs or issues), and AUTHOR (contributor information). You may find a subset of these headings or additional headings depending on what information is documented for that command or file.

Manual Sections

Depending on the type of information, the manual information is split into nine sections for organization and clarity. For instance, section 1 refers to user commands (see “NAME” for an example of the *passwd* command), section 4 contains special files, section 5 describes file formats for many system configuration files, and section 8 documents system administration and privileged commands that are designed for execution by the *root* user.

The default behavior of the *man* command is to search through section 1 and each successive section until it finds a match. There are a few commands in Linux that also have a configuration file with an identical name. One instance is the *passwd* command and the *passwd* file. The former is located in the */usr/bin* directory and the latter in the */etc* directory.

When you run **man passwd**, the *man* command scans through the manual pages and the first occurrence it finds is of the *passwd* command that’s stored in section 1. If you want to consult the manual pages of the *passwd* configuration file, you would have to specify the section number with the command (**man 5 passwd**) to instruct it to scan that one particular section only.



The header at the top in the above output is an indication that this help is for the *passwd* file and it is documented in section 5.

Searching by Keyword

Sometimes you need to use a command, but you can't recall its name. Linux allows you to perform a keyword search on manual pages using the *man* command with the *-k* (lowercase) flag, or the *apropos* command. These commands search all sections of the manual pages and show a list of all entries matching the specified keyword in their names or descriptions.

Before you can perform keyword searches on a new Linux installation, you'll need to run the *mandb* command in order to build an indexed database of the manual pages. This activity depends on the speed and the number of RHEL packages installed on the system, and it should not take long to perform. Simply type the command at the prompt and press the Enter key as follows:

```
# mandb
Updating index cache for path `/usr/share/man/ja/man7'. Wait...done.
Checking for stray cats under /usr/share/man/ja...
Checking for stray cats under /var/cache/man/ja...
Processing manual pages under /usr/share/man/fr...
Updating index cache for path `/usr/share/man/fr/man5'. Wait...done.
Checking for stray cats under /usr/share/man/fr...
Checking for stray cats under /var/cache/man/fr...
Processing manual pages under /usr/share/man/it...
Updating index cache for path `/usr/share/man/it/man8'. Wait...done.
........

Checking for stray cats under /usr/local/share/man...
Checking for stray cats under /var/cache/man/local...
107 man subdirectories contained newer manual pages.
7318 manual pages were added.
0 stray cats were added.
0 old database entries were purged.
```

Now you are ready to run keyword lookups. As an example, to find a forgotten XFS administration command, search for a string "xfs" by running either **man -k xfs** or **apropos xfs**. Both will produce an identical result.

```
[root@server1 ~]# man -k xfs
attr (1)             - extended attributes on XFS filesystem objects
filesystems (5)      - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i...
fs (5)               - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i...
fsck.xfs (8)         - do nothing, successfully
fsfreeze (8)         - suspend access to a filesystem (Ext3/4, ReiserFS, JFS,...
mkfs.xfs (8)         - construct an XFS filesystem
xfs (5)              - layout, mount options, and supported file attributes f...
xfs_admin (8)        - change parameters of an XFS filesystem
xfs_bmap (8)         - print block mapping for an XFS file
........
```

Once you have identified the command you were looking for, you can check that command's manual pages for usage.

Some commands also support the use of the --help and -? parameters. These parameters provide a brief list of options and a description without going through the manual pages. For example, to get quick help on the *passwd* command, run either **passwd --help** or **passwd -?**:

```
[root@server1 ~]# passwd --help
Usage: passwd [OPTION...] <accountName>
  -k, --keep-tokens       keep non-expired authentication tokens
  -d, --delete            delete the password for the named account (root
                          only); also removes password lock if any
  -l, --lock              lock the password for the named account (root only)
  -u, --unlock            unlock the password for the named account (root only)
  -e, --expire            expire the password for the named account (root only)
  -f, --force             force operation

.........
```

Not all commands support the --help and -? parameters.

Exposing Short Description

The *whatis* command searches for a short description of the specified command or file in the manual database. It quickly scans through the installed manual pages for the specified string and displays all matching entries. For instance, the following shows outputs of the command when run on *yum.conf* and *passwd* files:

```
[root@server1 ~]# whatis yum.conf
yum.conf (5)          - redirecting to DNF Configuration Reference
[root@server1 ~]# whatis passwd
passwd (5)            - password file
passwd (1)            - update user's authentication tokens
openssl-passwd (1ssl) - compute password hashes
```

The first output indicates that the specified file is a configuration file associated with the *yum* command, and the second output points to three entries for the *passwd* file (one configuration file and two commands).

You may alternatively run **man -f yum.conf** and **yum -f passwd** for the exact same results. Both *man -f* and *whatis* produce identical output.

The info and pinfo Commands

The *info* and *pinfo* commands display command documentation in a document-like format in great detail. Documentation is divided into sections called *nodes*. The header across the top of the screen shows the name of the file being displayed, names of the current, next, and previous nodes, and the name of the node prior to the current node. The operation of the *info* and *pinfo*

commands is almost identical. The following example provides a look at the first screen exhibited when either command is executed on the *ls*:

```
Next: dir invocation,  Up: Directory listing

10.1 'ls': List directory contents
==================================

The 'ls' program lists information about files (of any type, including
directories).  Options and file arguments can be intermixed arbitrarily,
as usual.

   For non-option command-line arguments that are directories, by
default 'ls' lists the contents of directories, not recursively, and
omitting files with names beginning with '.'.  For other non-option
arguments, by default 'ls' lists just the file name.  If no non-option
argument is specified, 'ls' operates on the current directory, acting as
if it had been invoked with a single argument of '.'.
```

While viewing help with *info* or *pinfo*, some common keys listed in Table 2-4 will help you navigate efficiently.

Key	Action
Down / Up arrows	Moves forward / backward one line
Spacebar / Del	Moves forward / backward one page
q	Quits the tutorial
[/]	Goes to the previous / next node in the document
t	Goes to the top node of the document
s	Searches forward for the specified string
{ / }	Searches for the previous / next occurrence of the string

Table 2-4 Navigating within info Documentation

Run *info*/*pinfo* on any of the commands you have reviewed so far and navigate using the keys provided in Table 2-4 for practice.

Documentation in the /usr/share/doc Directory

The */usr/share/doc* directory stores general documentation for installed packages under subdirectories that match their names. For example, the documentation for the gzip package includes the following files:

```
[root@server1 ~]# ls -l /usr/share/doc/gzip
total 144
-rw-r--r--. 1 root root    98 Oct  5  2014 AUTHORS
-rw-r--r--. 1 root root 88326 Aug 12  2018 ChangeLog
-rw-r--r--. 1 root root 22553 Jan  7  2018 NEWS
-rw-r--r--. 1 root root  6134 Jan  7  2018 README
-rw-r--r--. 1 root root 13262 Oct  5  2014 THANKS
-rw-r--r--. 1 root root  3467 Jan  7  2018 TODO
```

These text files contain general information about the gzip package. However, the manual pages and the info documentation prove to be more resourceful to Linux users and administrators.

Red Hat Enterprise Linux 8 Documentation

The website at *docs.redhat.com* hosts documentation for Red Hat's various products including RHEL 8 in HTML, PDF, and EPUB formats (see Figure 2-6).

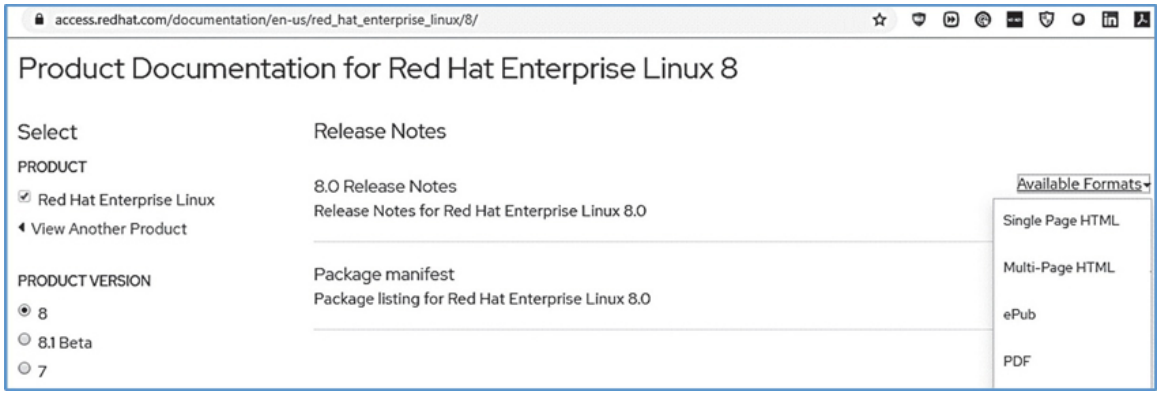


Figure 2-6 Red Hat's Webpage for RHEL 8 Documentation

This set of documentation includes release notes, as well as guides on planning, installation, administration, security, storage management, virtualization, and so on. For reference, you can download any of these guides for free in a format of your choice.

Chapter Summary

This chapter touched upon a few basic topics. We started by looking at a new display protocol that has replaced the legacy X Window System in RHEL 8. We interacted with the operating

system at the graphical console screen and examined various settings.

Next, we reviewed the Linux file system structure standard and significant higher-level subdirectories that consisted of static and variable files, and were grouped logically into lower-level subdirectories.

We analyzed command construct, and learned and run selected user level commands for familiarity. These tools included listing, viewing, and identifying basic information, and navigating in the directory hierarchy.

Finally, we learned how to access online help for commands and configuration files. We saw how to search through manual pages for desired text. Explanations regarding what commands to use were offered for additional help.

Check Your Understanding

1. What is the use of the *apropos* command?
2. Which Linux service creates device nodes at system startup?
3. What is the function of the *pwd* command?
4. Which three formats RHEL 8 documentation is available in at *docs.redhat.com*?
5. What type of information does section 5 in manual pages contain?
6. Which protocol has replaced X Window System as the default display protocol in RHEL 8?
7. Consider a Linux system with six processor cores. How many times do you have to run the lscpu command to pull information for all the cores?
8. Which two commands can you use to display the tutorial of a command?
9. Which three categories can Linux file systems be divided into?
10. Which three commands may be used to identify the full path of a command?

11. Which Linux command displays the hierarchy of a directory?
12. Linux commands may be categorized into two groups: privileged and non-privileged. True or False?
13. What is the name of the default display manager in RHEL 8?
14. What are the -R and -a options used for with the *ls* command?
15. What is the default number of days files in */tmp* are kept before they are automatically deleted if not accessed or modified.
16. Which command can be used to determine the time the kernel was build?
17. RHEL follows the Filesystem Hierarchy Standard. True or False?

Answers to Check Your Understanding

1. The *apropos* command can be used to perform a keyword search in the manual pages.
2. The *udevd* service.
3. The *pwd* command shows the absolute path of the current working directory.
4. The RHEL 8 documentation is available in PDF, EPUB, and HTML formats.
5. Section 5 of the manual pages contain information on configuration files.
6. Wayland is the new default display protocol in RHEL 8.
7. Only one time.
8. You can use the *info* or the *pinfo* command to display the tutorial of a command.
9. Linux file systems may be divided into disk-based, network-based, and memory-based file systems.
10. The *which*, *whereis*, and *type* commands may be used to determine the full path for a specified command.
11. The *tree* command shows the hierarchy of a directory.
12. True.

13. GNOME Display Manager is the default display manager in RHEL 8.
14. The -R option is used for recursive directory listing and the -a option for listing hidden files.
15. Ten days.
16. The *uname* command shows the kernel build time.
17. True.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 2-1: Navigate Linux Directory Tree

As *user1* on *server1*, execute the *pwd* command to check your location in the directory tree. Run the *ls* command with appropriate switches to show files in the current directory including the hidden files. Change directory into */etc* and run *pwd* again to confirm the directory change. Switch back to the directory where you were before, and run *pwd* again to verify. (Hint: Basic System Commands).

Lab 2-2: Miscellaneous Tasks

As *user1* on *server1*, execute the **tty** command to identify the terminal device file. Observe the terminal number reported. Open a couple of more terminal sessions, and run the **tty** command and compare the terminal numbers. Execute the **uptime** command and analyze the system uptime and processor load information. Use the three commands—**which**, **whereis**, and **type**—and identify the location of the *vgs* command. (Hint: Basic System Commands).

Lab 2-3: Identify System and Kernel Information

As *user1* on *server1*, issue **uname -a**. Analyze the basic information about the system and kernel reported. Run the **lscpu** command and examine the key items relevant to the processor. (Hint: Basic System Commands).

Lab 2-4: Use Help

As *user1* on *server1*, run **man uname** and **man 5 shadow**, and browse various headings and understand what they contain. Try **apropos ext4** and **man -k ext4**, **whatis group**, and observe their outputs. (Hint: Getting Help).

Chapter 03

Basic File Management

This chapter describes the following major topics:

- Common types of file in Linux
- Compress and uncompress files
- Archive and compress files and directories
- Edit files with the vim editor
- Create, list, display, copy, move, rename, and remove files and directories
- Create file and directory links
- Identify differences between copying and linking

RHCSA Objectives:

06. Archive, compress, unpack, and uncompress files using tar, star, gzip, and bzip2
07. Create and edit text files

08. Create, delete, copy, and move files and directories
09. Create hard and soft links

Linux supports different file types that are identified based on the kind of data they store. There are files that save information in plain text or binary format. This file type is very common. There are other files that store device information or simply point to the same data on the disk. A good comprehension of Linux file types is important for both Linux users and administrators.

Compressing and archiving one or more large files or an entire directory hierarchy allows users to conserve disk space or remote copy them at a faster pace. The resulting compressed archive can be easily uncompressed and unarchived whenever and wherever needed. RHEL offers native tools to support both user needs.

Normal and application users and database and system administrators all need to edit text files on a regular basis as part of their job. Linux delivers several text editors for this purpose, including the vim editor, which is popular within the Linux community. A sound, working knowledge of this tool is essential for all these roles.

There are a number of common operations that can be performed on files and directories in addition to viewing their contents. These operations include creating, listing, copying, moving, renaming, and removing both files and directories. Normal users will require higher privilege in order to perform these operations outside of their realm.

There is a tool available in the operating system that helps in linking files and directories for various use cases. An understanding of when to link files or directories versus when to copy them is important.

Common File Types

RHEL supports seven types of files: regular, directory, block special device, character special device, symbolic link, named pipe, and socket. The first two are the most common in Linux. The two types of device files are used by the operating system to communicate with peripheral devices. There are many instances of symbolic links as well. The last two types—named pipes and sockets—are used in inter-process communication.

Linux does not require an extension to a file to identify its type. It provides two elementary commands called *file* and *stat*, in addition to the *ls* command, to ascertain the type of data that a file may contain. This chapter discusses the first five file types and skips the last two (named pipes and sockets), as they are beyond the scope of this book.

Regular Files

Regular files may contain text or binary data. These files may be shell scripts or commands in the binary form. When you list a directory, all line entries for files in the output that begin with the hyphen character (-) represent regular files. The following truncated output is of the */root* directory:

```
[root@server1 ~]# ls -l
total 8
-rw-------. 1 root root 1447 Aug 22 15:27 anaconda-ks.cfg
```

Notice the hyphen in field 1 of column 1 before rw. This character indicates that the listed file is a regular file. Now, let’s run the *file* and *stat* commands on this file and see what they report:

```
[root@server1 ~]# file anaconda-ks.cfg
anaconda-ks.cfg: ASCII text
[root@server1 ~]#
[root@server1 ~]# stat anaconda-ks.cfg
  File: anaconda-ks.cfg
  Size: 1447            Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 10176129    Links: 1
Access: (0600/-rw-------)  Uid: (    0/    root)   Gid: (    0/    root)
Context: system_u:object_r:admin_home_t:s0
Access: 2019-09-02 22:18:40.881000000 -0400
Modify: 2019-08-22 15:27:48.030000000 -0400
Change: 2019-08-22 15:27:48.030000000 -0400
 Birth: -
```

The two commands report the file type differently. The first command returns the specific type of data that the file contains (ASCII text), and the latter simply states that it is a regular file.

Directory Files

Directories are logical containers that hold files and subdirectories. The following *ls* command output shows a few directories from */usr/bin*:

```
[root@server1 ~]# ls -l /usr
total 224
dr-xr-xr-x.   2 root root 40960 Aug 22 15:16 bin
drwxr-xr-x.   2 root root     6 Aug 12  2018 games
drwxr-xr-x.   3 root root    24 Aug 22 15:08 include
dr-xr-xr-x.  37 root root  4096 Aug 22 15:16 lib
dr-xr-xr-x. 119 root root 65536 Aug 22 15:22 lib64
drwxr-xr-x.  49 root root 12288 Aug 22 15:16 libexec
drwxr-xr-x.  12 root root   131 Aug 22 15:06 local
```

The letter “d” at the beginning of each line entry identifies the file as a directory. Try running the **file** and **stat** commands on */usr* and see what they report.

Block and Character Special Device Files

Each piece of hardware in the system has an associated file in the */dev* directory that is used by the system to communicate with that device. This type of file is called a *device file*. There are two types of device files: *character* (or *raw*) and *block*. In the example below, the *ls* command in

field 1 of column 1 distinguishes between the two with a “c” for character and a “b” for block:

```
[root@server1 ~]# ls -l /dev/console
crw-------. 1 root root 5, 1 Aug 31 08:25 /dev/console
[root@server1 ~]#
[root@server1 ~]# ls -l /dev/sd*
brw-rw----. 1 root disk 8, 0 Aug 31 08:24 /dev/sda
brw-rw----. 1 root disk 8, 1 Aug 31 08:24 /dev/sda1
brw-rw----. 1 root disk 8, 2 Aug 31 08:24 /dev/sda2
```

Use the **file** and **stat** commands on */dev/console* and */dev/sda* files for additional verification.

Every hardware device such as a disk, CD/DVD, printer, and terminal has an associated device driver loaded in the kernel. The kernel communicates with hardware devices through their respective device drivers. Each device driver is assigned a unique number called the *major* number, which the kernel uses to recognize its type.

Furthermore, there may be more than one instance of the same device type in the system. In that case, the same driver is used to control all those instances. For example, SATA device driver controls all SATA hard disks and CD/DVD drives. The kernel in this situation allots a *minor* number to each individual device within that device driver category to identify it as a unique device. This scheme applies to disk partitions as well. In short, a major number points to the device driver, and a minor number points to a unique device or partition that the device driver controls.

In the above long listing, columns 5 and 6 depict the major and minor number association for each device instance. Major number 8 represents the block device driver for SATA disks. Similarly (not shown in the above output), major number 253 denotes the driver for device mapper (VDO volumes, LVM logical volumes, etc.), and 11 signifies optical devices. VDO volumes are discussed in Chapter 13 “Basic

Storage Partitioning” and LVM logical volumes are covered in Chapter 14 “Advanced Storage Partitioning”.

Symbolic Links

A *symbolic link* (a.k.a. a *soft link* or a *symlink*) may be considered a shortcut to another file or directory. If you issue *ls -l* on a symbolically linked file or directory, the line entry will begin with the letter “l”, and an arrow will be pointing to the target link. For example:

```
[root@server1 ~]# ls -l /usr/sbin/vigr
lrwxrwxrwx. 1 root root 4 Dec 18  2018 /usr/sbin/vigr -> vipw
```

Run the **file** and **stat** commands on */usr/sbin/vigr* for additional confirmation.

Compression and Archiving

Compression tools are used to compress one or more files to conserve space. They may be used with archive commands, such as *tar* or *star*, to create a single compressed archive of hundreds of files and directories. A compressed archive can then be copied to a remote system faster than a non-compressed archive or stored at a backup location. RHEL offers a multitude of compression tools such as *gzip* (*gunzip*) and *bzip2* (*bunzip2*).

The *tar* and *star* commands have the ability to preserve general file attributes such as ownership, owning group, and timestamp as well as extended attributes such as ACLs and SELinux contexts.

Chapter 04 “Advanced File Management” expounds on ACLs, and Chapter 21 “Security Enhanced Linux” on SELinux.

The syntax and usage of *tar* and *star* commands are similar, and most of the common options are identical. We discuss only the *tar* command.

Using gzip and gunzip

The *gzip/gunzip* compression utility pair has been available in Linux for over two decades. The *gzip* command is used to create a compressed file of each of the specified files and it adds the *.gz* extension to each file for identification. This tool can be used with the -r option to compress an entire directory tree, and with the -l option to display compression information about a gzipped file. The -l option also instructs the command to display the filename that will be given to the file when it is uncompressed.

To compress the file *fstab* located in the */etc* directory, copy this file in the *root* user's home directory */root* using the *cp* command and confirm with *ls*:

```
[root@server1 ~]# pwd
/root
[root@server1 ~]# cp /etc/fstab .
[root@server1 ~]# ls -l fstab
-rw-r--r--. 1 root root 579 Sep  2 23:01 fstab
```

Now use the *gzip* command to compress this file and *ls* to confirm:

```
[root@server1 ~]# gzip fstab
[root@server1 ~]# ls -l fstab.gz
-rw-r--r--. 1 root root 349 Sep  2 23:01 fstab.gz
```

Notice that the original file is compressed, and it now has the *.gz* extension added to it. If you wish to view compression information for the file, run the *gzip* command again with the -l option:

```
[root@server1 ~]# gzip -l fstab.gz
         compressed        uncompressed  ratio uncompressed_name
                349                 579  43.9% fstab
```

To decompress this file, use the *gunzip* command:

```
[root@server1 ~]# gunzip fstab.gz
[root@server1 ~]# ls -l fstab
-rw-r--r--. 1 root root 579 Sep  2 23:01 fstab
```

Check the file after the decompression with the *ls* command. It will be the same file with the same size, timestamp, and other attributes.

Using bzip2 and bunzip2

The *bzip2*/*bunzip2* is another compression pair that has been available in Linux for a long time. The *bzip2* command creates a compressed file of each of the specified files and it adds the *.bz2* extension to each file for identification.

Let's compress the *fstab* file again, but this time with *bzip2* and confirm with *ls*:

```
[root@server1 ~]# bzip2 fstab
[root@server1 ~]# ls -l fstab.bz2
-rw-r--r--. 1 root root 386 Sep  2 23:01 fstab.bz2
```

Notice that the original file is compressed, and it now has the *.bz2* extension. To decompress this file, use the *bunzip2* command:

```
[root@server1 ~]# bunzip2 fstab.bz2
[root@server1 ~]# ls -l fstab
-rw-r--r--. 1 root root 579 Sep  2 23:01 fstab
```

Check the file after the decompression with the *ls* command. It will be the same file with the same size, timestamp, and other attributes.

Differences between gzip and bzip2

The function of both *gzip* and *bzip2* is the same: to compress and decompress files. However, in terms of the compression and decompression rate, *bzip2* has a better compression ratio (smaller target file size), but it is slower. These differences are evident on fairly large files. On small files,

you can use either of the two. Both commands support several identical options.

Using tar

The *tar* (*tape archive*) command is used to create, append, update, list, and extract files or an entire directory tree to and from a single file, which is called a *tarball* or *tarfile*. This command can be instructed to also compress the tarball after it has been created.

tar supports a multitude of options such as those described in Table 3-1.

Option	Definition
-c	Creates a tarball.
-f	Specifies a tarball name.
-p	Preserve file permissions. Default for the root user. Specify this option if you create an archive as a normal user.
-r	Appends files to the end of an extant uncompressed tarball.
-t	Lists contents of a tarball.
-u	Appends files to the end of an extant uncompressed tarball provided the specified files being added are newer.
-v	Verbose mode.
-x	Extracts or restores from a tarball.

Table 3-1 tar Command Options

The -r and -u options do not support adding files to an existing compressed tarball.

A few examples are provided below to elucidate the use of *tar*. Pay special attention to the syntax and options used in each command and observe the output.

To create a tarball called */tmp/home.tar* of the entire */home* directory, use the -v option for verbosity and the -f option to specify the name of the archive file with the command. The following is a truncated output of the command:

```
[root@server1 ~]# tar -cvf /tmp/home.tar /home
tar: Removing leading `/' from member names
/home/
/home/user1/
/home/user1/.mozilla/
/home/user1/.mozilla/extensions/
/home/user1/.mozilla/plugins/
/home/user1/.mozilla/firefox/
/home/user1/.mozilla/firefox/3cbvln5q.default/
```

The resulting tarball will not include the leading forward slash (*/*) in the file paths as indicated on line 1 of the output even though the full path of */home* is supplied. This is the default behavior of the *tar* command, which gives you the flexibility to restore the files at any location of your choice without having to worry about the full pathnames. Use the -P option at the creation time to override this behavior.

To create a tarball called */tmp/files.tar* containing only a select few files (two files in this example) from the */etc* directory:

```
[root@server1 ~]# tar -cvf /tmp/files.tar /etc/passwd /etc/yum.conf
tar: Removing leading `/' from member names
/etc/passwd
tar: Removing leading `/' from hard link targets
/etc/yum.conf
```

To append files located in the */etc/yum.repos.d* directory to the existing tarball */tmp/home.tar*:

```
[root@server1 ~]# tar -rvf /tmp/files.tar /etc/yum.repos.d
tar: Removing leading `/' from member names
/etc/yum.repos.d/
/etc/yum.repos.d/redhat.repo
tar: Removing leading `/' from hard link targets
```

To list what files are included in the *home.tar* tarball:

```
[root@server1 ~]# tar -tvf /tmp/files.tar
-rw-r--r-- root/root      2484 2019-08-26 23:20 etc/passwd
lrwxrwxrwx root/root         0 2019-02-14 07:02 etc/yum.conf -> dnf/dnf.conf
drwxr-xr-x root/root         0 2019-08-22 15:46 etc/yum.repos.d/
-rw-r--r-- root/root       358 2019-08-22 15:46 etc/yum.repos.d/redhat.repo
```

To restore a single file, *etc/yum.conf*, from */tmp/files.tar* under */root* and confirm the output with *ls*:

```
[root@server1 ~]# cd
[root@server1 ~]# tar -xf /tmp/files.tar etc/yum.conf
[root@server1 ~]# ls -l etc/yum.conf
lrwxrwxrwx. 1 root root 12 Feb 14  2019 etc/yum.conf -> dnf/dnf.conf
```

To restore all files from */tmp/files.tar* under */root* and confirm the output with *ls*:

```
[root@server1 ~]# tar -xf /tmp/files.tar
[root@server1 ~]# ls -l etc
total 4
-rw-r--r--. 1 root root 2484 Aug 26 23:20 passwd
lrwxrwxrwx. 1 root root   12 Feb 14  2019 yum.conf -> dnf/dnf.conf
drwxr-xr-x. 2 root root   25 Aug 22 15:46 yum.repos.d
```

tar also supports options to directly compress the target file while being archived using the *gzip* or *bzip2* command. These options are described in Table 3-2.

Option	Description
-j	Compresses a tarball with bzip2
-z	Compresses a tarball with gzip

Table 3-2 tar with Compression Options

You will use the options in Table 3-2 to create compressed archives in Exercise 3-1.

EXAM TIP: Archiving and compression are tasks usually done together to produce smaller archive files.

Exercise 3-1: Create Compressed Archives

This exercise should be done on *server1* as *root*.

In this exercise, you will create a tarball called *home.tar.gz* of the */home* directory under */tmp* and compress it with *gzip*. You will create another tarball called *home.tar.bz2* of the */home* directory under */tmp* and compress it with *bzip2*. You will list the content of *home.tar.gz* without uncompressing it and then extract all the files in the current directory. Finally, you will extract the *bzip2*-compressed archive in the */tmp* directory.

1. Create (-c) a *gzip*-compressed (-z) tarball under */tmp* (-f) for */home*:

```
[root@server1 ~]# tar -czf /tmp/home.tar.gz /home
```

2. Create (-c) a *bzip2*-compressed (-j) tarball under */tmp* (-f) for */home*:

```
[root@server1 ~]# tar -cjf /tmp/home.tar.bz2 /home
```

3. List (-t) the content of the *gzip*-compressed archive (-f) without uncompressing it:

```
[root@server1 ~]# tar -tf /tmp/home.tar.gz
home/
home/user1/
home/user1/.mozilla/
home/user1/.mozilla/extensions/
. . . . . .
```

4. Extract (-x) files from the *gzip*-compressed tarball (-f) in the current directory:

```
[root@server1 ~]# tar -xf /tmp/home.tar.gz
```

5. Extract (-x) files from the *bzip2*-compressed (-f) tarball under */tmp* (a different directory location than the current directory) (-C):

```
[root@server1 ~]# tar -xf /tmp/home.tar.bz2 -C /tmp
```

Run **ls /tmp** to view the list of extracted files.

File Editing

The *vim editor* is an interactive, full-screen *visual* text-editing tool that allows you to create and modify text files. This tool is available as a standard editor in all vendor UNIX versions and Linux distributions. It does not require the graphical capability and it is not heavy on compute resources. All text editing within *vim* takes place in a *buffer* (a small chunk of memory used to hold file updates). Changes can either be written to the disk or discarded.

It is essential for you as a system administrator to master the *vim* editor skills. The best way to learn *vim* is to practice by opening or creating a file and run the *vim* commands. See the manual pages of *vim* for details. Alternatively, you can run the *vimtutor* command to view the tutorial.

Modes of Operation

The *vim* editor has three modes of operation: the command mode, the input mode, and the last line mode. The fourth mode is referred to as the visual mode, but it is not discussed in the book.

The *command* mode is the default mode of vim. The *vim* editor places you into this mode when you start it. While in the command mode, you can carry out tasks such as copy, cut, paste, move, remove, replace, change, and search on text, in addition to performing navigational operations. This mode is also known as the *escape* mode because the Esc key is used to enter the mode.

In the *input* mode, anything that is typed on the keyboard is entered into the file as text. Commands cannot be run in this mode. The input mode is also called the *edit* mode or the *insert* mode. You need to press the Esc key to return to the command mode.

While in the command mode, you may carry out advanced editing tasks on text by pressing the colon character (:), which places the cursor at the beginning of the last line of the screen, and hence it is referred to as the *last line* or *extended* mode. This mode is considered a special type of command mode.

Starting vim

The *vim* editor may be started by typing the command *vim* at the command prompt, and it may follow an existing or a new filename as an argument. Without a specified filename, it simply opens an empty screen where you can enter text. You can save the text in a file or discard using commands provided in subsequent subsections.

```
[root@server1 ~]# vim
```

Alternatively, you can supply a filename as an argument. This way, *vim* will open the specified file for editing if the file exists, or it will create a file by that name if it does not exist.

```
[root@server1 ~]# vim <filename>
```

There are options available that you may specify at the time of starting this editing tool. Refer to the manual pages for more information.

Inserting text

Once *vim* is started, there are six commands that can switch into the edit mode. These commands are simple lower and uppercase i, a, and o, and are described in Table 3-3.

Command	Action
i	Inserts text before the current cursor position
I	Inserts text at the beginning of the current line
a	Appends text after the current cursor position
A	Appends text to the end of the current line
o	Opens a new line below the current line
O	Opens a new line above the current line

Table 3-3 Inserting Text

Press the Esc key when you've finished entering text in the edit mode to return to the command mode.

Navigating within vim

Navigation keys are helpful in editing small and large files. They allow you to make rapid moves in the file. There are multiple key sequences available within *vim* to control the cursor movement. Some of the elementary keystrokes are elaborated in Table 3-4.

Command	Action
h	Moves backward one character
j	Moves downward one line
k	Moves upward one line
l	Moves forward one character
w	Moves to the start of the next word
b	Moves backward to the start of the preceding word
e	Moves to the ending character of the next word
$	Moves to the end of the current line
Enter	Moves to the beginning of the next line
Ctrl+f	Scrolls down to the next page
Ctrl+b	Scrolls up to the previous page

Table 3-4 Navigating within vim

You can precede any of the commands listed in Table 3-4 by a numeral to repeat the command action that many times. For instance, 3h would move the cursor three places to the left, 5Enter would move the cursor five lines below, and 2Ctrl+f would move the cursor two screens down.

In addition, you can use 0 (zero) to move to the beginning of the current line, [[to move to the first line of the file, and]] to move to the last line of the file.

Deleting Text

vim provides several commands to carry out delete operations. Some of the commands are described in Table 3-5.

Command	Action
x	Deletes the character at the cursor position
X	Deletes the character before the cursor location
dw	Deletes the word or part of the word to the right of the cursor location
dd	Deletes the current line
D	Deletes at the cursor position to the end of the current line
:6,12d	Deletes lines 6 through 12

Table 3-5 Deleting Text

You can precede any of the commands listed in Table 3-5, except for the last line mode command, by a numeral to repeat the command action that many times. For instance, 2X would delete two characters before the cursor position, and 3dd would delete the current line and the two lines below it.

Undoing and Repeating

Table 3-6 explicates the commands that undo the last change made and repeat the last command run.

Command	Action
u	Undoes the previous command.
U	Undoes all the changes done on the current line.
:u	Undoes the previous last line mode command.
. (dot)	Repeats the last command run.

Table 3-6 Undoing and Repeating

You can precede any of the commands listed in Table 3-6, except for the U and :u commands, by a numeral to repeat the command action that many times. For instance, 2u would undo the previous two changes, and 2U would undo all the changes done on the current and the previous lines.

Searching for Text

You can perform forward and reverse searches while in the command mode by using the / and ? characters followed by the string to be searched. For instance, in a file with numerous occurrences of the string “profile,” you can run /profile or ?profile for a forward or reverse search.

Table 3-7 summarizes these actions.

Command	Action
/string	Searches forward for a string
?string	Searches backward for a string
n	Finds the next occurrence of a string
N	Finds the previous occurrence of a string

Table 3-7 Searching for Text

For forward searches, repeating “n” takes the cursor to the previous occurrences of the searched string, and repeating

“N” moves the cursor to the next occurrences.

The behavior is reversed for backward searches. Repeating “n” takes the cursor to the next occurrences of the searched string, and repeating “N” moves the cursor to the previous occurrences.

Replacing Text

Table 3-8 describes two last line mode commands that are used to perform a search and replace operation.

Command	Action
:%s/old/new	Replaces the first occurrence of *old* with *new* in a file. For example, to replace the first occurrence of profile with Profile, use *:%s/profile/Profile*.
:%s/old/new/g	Replaces all occurrences of *old* with *new* in a file. For example, to replace all the occurrences of profile with Profile in a file, use *:%s/profile/Profile/g*.

Table 3-8 Replacing Text

If you have used either of these and would like to undo it, use the last line mode command :u.

Copying, Moving, and Pasting Text

vim allows you to copy some text and paste it to the desired location within the file. You can copy (yank) a single character, a single word, or an entire line, and then paste it wherever you need it. The copy function can be performed on multiple characters, words, or lines simultaneously. Table 3-9 describes the copy, move, and paste commands.

Command	Action
yl	Yanks the current letter into buffer
yw	Yanks the current word into buffer
yy	Yanks the current line into buffer
p	Pastes yanked data below the current line
P	Pastes yanked data above the current line
:1,3co6	Copies lines 1 through 3 and pastes them after line 6
:4,6m9	Moves lines 4 through 6 after line 9

Table 3-9 Copying, Moving, and Pasting Text

You can precede any of the commands listed in Table 3-9, except for the last line mode commands, by a numeral to repeat the command action that many times. For instance, 2yw would yank two words, 2yy would yank two lines, and 2p would paste two times.

Changing Text

There are numerous commands available within *vim* to change and modify text as summarized in Table 3-10. Most of these commands switch into the edit mode, so you will have to press the Esc key to return to the command mode.

Command	Action
cl	Changes the letter at the cursor location
cw	Changes the word (or part of the word) at the cursor location to the end of the word
cc	Changes the entire line
C	Changes text at the cursor position to the end of the line
r	Replaces the character at the cursor location with the character entered following this command
R	Overwrites or replaces the text on the current line
J	Joins the next line with the current line
xp	Switches the position of the character at the cursor position with the character to the right of it
~	Changes the letter case (uppercase to lowercase, and vice versa) at the cursor location

Table 3-10 Changing Text

You can precede any of the commands listed in Table 3-10 by a numeral to repeat the command action that many times. For instance, 2cc would change the entire current and the next line, and 2r would replace the current character and the next character.

Saving and Quitting vim

When you are done with modifications, you can save or discard them. Use one of the commands listed in Table 3-11 as required.

Command	Action
:w	Writes changes into the file without quitting vim
:w file2	Writes changes into a new file called *file2* without quitting vim
:w!	Writes changes to the file even if the file owner does not have write permission on the file
:wq	Writes changes to the file and quits vim
:wq!	Writes changes to the file and quits vim even if the file owner does not have write permission on the file
:q	Quits vim if no modifications were made
:q!	Quits vim if modifications were made, but we do not wish to save them

Table 3-11 Saving and Quitting vim

The exclamation mark (!) can be used to override the write protection placed on the file for the owner.

File and Directory Operations

This section elaborates on various management operations that can be performed on files and directories. These operations include creating, displaying contents, copying, moving, renaming, and deleting files and directories. These common operations can be performed by normal users who own or have appropriate permissions. The *root* user can accomplish these tasks on any file or directory on the system, regardless of who owns it. In case there's a lack of user permissions, an error message is generated.

Creating Files and Directories

Files can be created in multiple ways using different commands; however, there is only one command to create directories.

Creating Empty Files Using touch

The *touch* command creates an empty file. If the file already exists, it simply updates the timestamp on it to match the current system date and time. Execute the following as *root* in the *root* user's home directory to create *file1* and then run *ls* to verify:

```
[root@server1 ~]# cd
[root@server1 ~]# touch file1
[root@server1 ~]# ls -l file1
-rw-r--r--. 1 root root 0 Sep  5 08:59 file1
```

As expected, column 5 (the size column) in the output is 0, meaning that *file1* is created with zero bytes in size. If you rerun the *touch* command on this file after a minute or so, a new timestamp is placed on it:

```
[root@server1 ~]# touch file1
[root@server1 ~]# ls -l file1
-rw-r--r--. 1 root root 0 Sep  5 09:03 file1
```

The *touch* command has a few interesting options. The -d and -t options set a specific date and time on a file; the -a and -m options enable you to change only the access or the modification time on a file to the current system time; and the -r option sets the modification time on a file to that of a reference file's. Let's use a couple of these options in the examples below:

To set the date on *file1* to September 20, 2019:

```
[root@server1 ~]# touch -d 2019-09-20 file1
[root@server1 ~]# ls -l file1
-rw-r--r--. 1 root root 0 Sep 20  2019 file1
```

To change the modification time on *file1* to the current system time:

```
[root@server1 ~]# touch -m file1
[root@server1 ~]# ls -l file1
-rw-r--r--. 1 root root 0 Sep  5 09:13 file1
```

Try the rest of the options for practice.

Creating Short Files Using cat

The *cat* command allows you to create short text files. The ending angle bracket “>” must be used to redirect the output to the specified file (*catfile1* in this example):

```
[root@server1 ~]# cat > catfile1
```

Nothing is displayed when you execute the above, as the system is waiting for you to input something. Type some text. Press the Enter key to open a new line and continue typing. When you are done, press Ctrl+d to save the text in *catfile1* and return to the command prompt. You can verify the file creation with the *ls* command.

Creating Files Using vim

You can use the *vim* editor to create and modify text files of any size. Refer to the previous section in this chapter on how to use vim.

Making Directories Using mkdir

The *mkdir* command is used to create directories. This command shows an output if you run it with the -v option. The following example demonstrates the creation of a directory called *dir1* in the *root* user’s home directory:

```
[root@server1 ~]# mkdir dir1 -v
mkdir: created directory 'dir1'
[root@server1 ~]# ls -ld dir1
drwxr-xr-x. 2 root root 6 Sep  5 09:23 dir1
```

You can create a hierarchy of subdirectories by specifying the -p (parent) option with *mkdir*. In the following example, *mkdir* is used to create the hierarchy *dir2/perl/perl5*:

```
[root@server1 ~]# mkdir -vp dir2/perl/perl5
mkdir: created directory 'dir2'
mkdir: created directory 'dir2/perl'
mkdir: created directory 'dir2/perl/perl5'
```

Notice the placement of options in the two examples. Many commands in Linux accept either format.

Displaying File Contents

RHEL offers a variety of tools for showing file contents. Directory contents are simply the files and subdirectories that it contains. Use the *ls* command as explained earlier to view directory contents.

For viewing files, you can use the *cat*, *more*, *less*, *head*, and *tail* commands. These tools are explained below.

Using cat

cat displays the contents of a text file. It is typically used to view short files. It shows the entire file on the screen. The following example shows the *.bash_profile* file in the *root* user's home directory with the *cat* command:

```
[root@server1 ~]# cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
```

You can add the -n option to the *cat* command to view the output in numbered format.

Using tac

tac displays the contents of a text file in reverse. In the example below, the *.bash_profile* file in the *root* user's home directory is displayed with the *tac* command:

```
[root@server1 ~]# tac .bash_profile
export PATH

PATH=$PATH:$HOME/bin

# User specific environment and startup programs

fi
        . ~/.bashrc
if [ -f ~/.bashrc ]; then
# Get the aliases and functions

# .bash_profile
```

Compare this output with that of the *cat* command's. Both are reversed.

Using less and more

Both *less* and *more* are text filters that are used for viewing long text files one page at a time, starting at the beginning. The *less* command is more capable than the *more* command. *less* does not need to read the entire file before it starts to display its contents, thus making it faster. The *more* command is limited to forward text searching only, whereas *less* is able to perform both forward and backward searches. Run the *less* and *more* commands one at a time and observe the visual difference in the outputs:

```
[root@server1 ~]# less /usr/bin/znew
[root@server1 ~]# more /usr/bin/znew
```

You can navigate with the keys described in Table 3-12 while viewing the files with either tool.

Key	Purpose
Spacebar / f	Scrolls forward one screen
Enter	Scrolls forward one line
b	Scrolls backward one screen
d	Scrolls forward half a screen
h	Displays help
q	Quits and returns to the command prompt
/string	Searches forward for a string
?string	Searches backward for a string; only applies to the less command
n	Finds the next occurrence of a string
N	Finds the previous occurrence of a string; only applies to the less command

Table 3-12 Navigating with less and more

If the */usr/bin/znew* file is unavailable, use */etc/profile* instead.

Using head and tail

head displays the starting few lines of the specified text file. By default, it returns the first ten lines. See the example below:

```
[root@server1 ~]# head /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

# It's NOT a good idea to change this file unless you know what you
# are doing. It's much better to create a custom.sh shell script in
# /etc/profile.d/ to make custom changes to your environment, as this
# will prevent the need for merging in future updates.
```

The above output includes three empty lines as well. You can pass a numeral to the command as an argument to limit the number of lines in the output. For example, run the following to view only the top three lines from */etc/profile*:

```
[root@server1 ~]# head -3 /etc/profile
```

On the other hand, the *tail* command displays the ending ten lines from the specified file by default unless a numeral is passed as an argument to alter the behavior. Issue the following two commands on your terminal to witness the difference:

```
[root@server1 ~]# tail /etc/services
[root@server1 ~]# tail -3 /etc/services
```

The *tail* command is particularly useful when watching a log file while it is being updated. The -f (follow) option enables this function. The following example enables us to view the updates to the system log file */var/log/messages* in real time:

```
[root@server1 ~]# tail -f /var/log/messages
```

You may have to wait for some time before you see an update. Press Ctrl+c to quit when you are done.

Counting Words, Lines, and Characters in Text Files

The *wc* (*word count*) command displays the number of lines, words, and characters (or bytes) contained in a text file or input supplied. For example, when you run this command on the */etc/profile* file, you will see output similar to the following:

```
[root@server1 ~]# wc /etc/profile
  85  294 2078 /etc/profile
```

Column 1 in the output discloses the number of lines (85) in the file followed by the number of words (294), the number of characters (or bytes) (2078), and the filename (*/etc/profile*).

You can use the options listed in Table 3-13 to restrict the output as desired.

Option	Action
-l	Prints a count of lines
-w	Prints a count of words
-c	Prints a count of bytes
-m	Prints a count of characters

Table 3-13 wc Command Options

The following example displays only the count of characters in */etc/profile*:

```
[root@server1 ~]# wc -m /etc/profile
2078 /etc/profile
```

Try running *wc* with the other options and observe the outcomes.

Copying Files and Directories

The copy operation duplicates a file or directory. RHEL provides the *cp* command for this purpose and it has a variety of options.

Copying Files

The *cp* command copies one or more files within a directory or to another directory. To duplicate a file in the same directory, you must give a different name to the target file. However, if the copy is being made to a different directory, you can use either the same filename or assign a different one. Consider the following examples:

To copy *file1* as *newfile1* within the same directory:

```
[root@server1 ~]# cp file1 newfile1
```

To copy *file1* by the same name to another existing directory *dir1*:

```
[root@server1 ~]# cp file1 dir1
```

By default, the copy operation overwrites the destination file if it exists without presenting a warning. To alter this behavior, use the -i (interactive) option to instruct *cp* to prompt for confirmation before overwriting:

```
[root@server1 ~]# cp file1 dir1 -i
cp: overwrite 'dir1/file1'? y
```

Press Enter after keying in a “y” for yes or an “n” for no to proceed.

By default, you do not need to specify the -i option for yes/no confirmation if you attempt to copy a file to overwrite the destination file as *root*. The predefined alias—“alias cp=’cp -i’”—in the *.bashrc* file in the *root* user’s home directory takes care of that.

Copying Directories

The *cp* command with the -r (recursive) option copies an entire directory tree to another location. In the following example, *dir1* is copied to *dir2* and then the directory contents of *dir2* are listed for validation:

```
[root@server1 ~]# cp -r dir1 dir2
[root@server1 ~]# ls -l dir2
total 0
drwxr-xr-x. 2 root root 19 Sep  5 12:18 dir1
drwxr-xr-x. 3 root root 19 Sep  5 09:25 perl
```

You may use the -i option for overwrite confirmation if the destination already has a matching file or directory.

Try running **ls -l dir2 -R** to view the entire *dir2* hierarchy.

The *cp* command can also use -p, which can provide the ability to preserve the attributes (timestamp, permissions, ownership, etc.) of a file or directory being copied. Try running **cp -p file1 /tmp** and then use *ls -l* to compare the attributes for both files.

Moving and Renaming Files and Directories

A file or directory can be moved within the same file system or to another. Within the file system move, an entry is added to the target directory and the source entry is removed, which leaves the actual data intact. On the other hand, a move to a different file system physically moves the file or directory content to the new location and deletes the source.

A rename simply changes the name of a file or directory; data is not touched.

Moving and Renaming Files

The *mv* command is used to move or rename files. The -i option can be specified for user confirmation if a file by that name already exists. The following example moves *file1* to *dir1* and prompts for confirmation:

```
[root@server1 ~]# mv -i file1 dir1
mv: overwrite 'dir1/file1'? y
```

By default, you do not need to specify the -i option for yes/no confirmation if you attempt to move a file to overwrite the destination file as *root*. The predefined alias—“alias mv=’mv -i’”—in the *.bashrc* file in the *root* user’s home directory takes care of that.

To rename *newfile1* as *newfile2*:

```
[root@server1 ~]# mv newfile1 newfile2
```

Verify the above operations with **ls -l**.

Moving and Renaming Directories

Use the *mv* command to move a directory and its contents to somewhere else or to change the name of the directory. For

example, you can move *dir1* into *dir2* (*dir2* must exist, otherwise it will be a simple rename operation):

```
[root@server1 ~]# mv dir1 dir2
```

To rename *dir2* as *dir20*:

```
[root@server1 ~]# mv dir2 dir20
```

Verify the above operations with **ls -l**.

Removing Files and Directories

The remove operation deletes a file entry from the directory structure and marks its data space as free. For a directory, the remove operation weeds corresponding entries out from the file system structure.

Removing Files

You can remove a file using the *rm* command, which deletes one or more specified files. For example, issue the following command to erase *newfile2*:

```
[root@server1 ~]# rm -i newfile2
rm: remove regular empty file 'newfile2'? y
```

By default, you do not need to specify the -i option for yes/no confirmation if you attempt to remove a file as *root*. The predefined alias—“alias rm=’rm -i’”—in the *.bashrc* file in the *root* user’s home directory takes care of that.

The *rm* command can also be used to remove a file that has a wildcard character, such as an asterisk (*) or a question mark (?), embedded in its name. These characters have special meaning to the shell, and filenames containing them must be prepended with the backslash character (\) to instruct the shell to treat them as regular characters.

A careful use of the *rm* command is particularly important when you have administrative rights on the system.

For example, if a file exists by the name * under the */tmp* directory (use **touch /tmp/*** to create it), you can remove it by executing **rm /tmp/***. If you mistakenly run **rm /tmp/*** instead, all files under */tmp* will be deleted.

Wildcard characters are used in filename globbing and in commands where an action needs to occur on multiple files matching certain criteria. They are discussed in Chapter 07 “The Bash Shell”.

Removing Directories

The *rmdir* and *rm* commands remove directories. The *rmdir* command is used to delete empty directories, while *rm* requires the -d option to accomplish the same. In addition, the -r or -R (recursive) flag with *rm* will remove a directory and all of its contents. Both commands support the -v switch for reporting what they are doing. Let’s look at a few examples.

To erase an empty directory called *emptydir* (assuming *emptydir* exists), use either of the following:

```
[root@server1 ~]# rmdir emptydir -v
[root@server1 ~]# rm -dv emptydir
```

or

To remove *dir20* and all its contents recursively, use either -r or -R with the command:

```
[root@server1 ~]# rm -r dir20
```

The *rm* command supports the -i flag for interactive deletions.

The same rules that apply on filenames with wildcard characters in their names, apply on directory names as well.

See the previous topic for details.

File Linking

Each file within a file system has a multitude of attributes assigned to it at the time of its creation. These attributes are collectively referred to as the file's *metadata*, and they change when the file is accessed or modified. A file's metadata includes several pieces of information, such as the file type, size, permissions, owner's name, owning group name, last access/modification times, link count, number of allocated blocks, and pointers to the data storage location. This metadata takes 128 bytes of space for each file. This tiny storage space is referred to as the file's *inode* (*index node*).

An inode is assigned a unique numeric identifier that is used by the kernel for accessing, tracking, and managing the file. In order to access the inode and the data it points to, a filename is assigned to recognize it and access it. This mapping between an inode and a filename is referred to as a *link*. It is important to note that the inode does not store the filename in its metadata; the filename and corresponding inode number mapping is maintained in the directory's metadata where the file resides.

Linking files or directories creates additional instances of them, but all of them eventually point to the same physical data location in the directory tree. Linked files may or may not have identical inode numbers and metadata depending on how they are linked.

There are two ways to create file and directory links in RHEL, and they are referred to as hard links and soft links. Links are created between files or between directories, but not between a file and a directory.

Hard Link

A *hard* link is a mapping between one or more filenames and an inode number, making all hard-linked files indistinguishable from one another. This implies that all hard-linked files will have identical metadata. Changes to the file metadata and content can be made by accessing any of the filenames.

Figure 3-1 shows two filenames—*file10* and *file20*—both sharing the same inode number 10176147. Here, each filename is essentially a hard link pointing to the same inode.

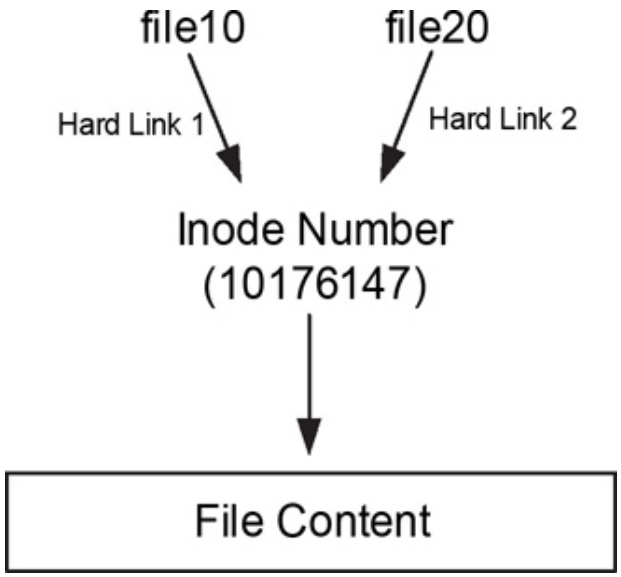


Figure 3-1 Hard Link

A hard link cannot cross a file system boundary, and it cannot be used to link directories because of the restrictions placed within Linux designed to avoid potential issues with some commands.

The following example creates an empty file called *file10* and then uses the *ln* command to create a hard link called *file20* in the same directory:

```
[root@server1 ~]# touch file10
[root@server1 ~]# ln file10 file20
```

After creating the link, run *ls* with the -li flags as follows:

```
[root@server1 ~]# ls -li file*
10176147 -rw-r--r--. 2 root root 0 Sep  6 14:29 file10
10176147 -rw-r--r--. 2 root root 0 Sep  6 14:29 file20
```

Look at columns 1 and 3. Column 1 shows the shared inode number (10176147), and column 3 provides a *link count* of the hard links that each file has (*file10* points to *file20*, and vice versa). If you remove the original file (*file10*), you will still have access to the data through *file20*. Each time you add a hard link to an extant file, the link count will increase by 1. Similarly, if you delete a hard link, the link count will go down by 1. When all the hard links (files) are erased, the link count will set to 0. The increase and decrease in the number of links is reflected on all hard-linked files.

Soft Link

A *soft* link (a.k.a. a *symbolic* link or a *symlink*) makes it possible to associate one file with another. The concept is analogous to that of a shortcut in Microsoft Windows where the actual file is resident somewhere in the directory structure, but there can be one or more shortcuts with different names pointing to it. With a soft link, you can access the file directly via the actual filename as well as any of the shortcuts. Each soft link has a unique inode number that stores the pathname to the file it is linked with. For a symlink, the link count does not increase or decrease, rather each symlinked file receives a new inode number. The pathname can be absolute or relative depending on what was specified at the time of its creation. The size of the soft link is the number of characters in the pathname to the target.

Figure 3-2 shows the file *file10* with a soft link called *softfile10* pointing to it.

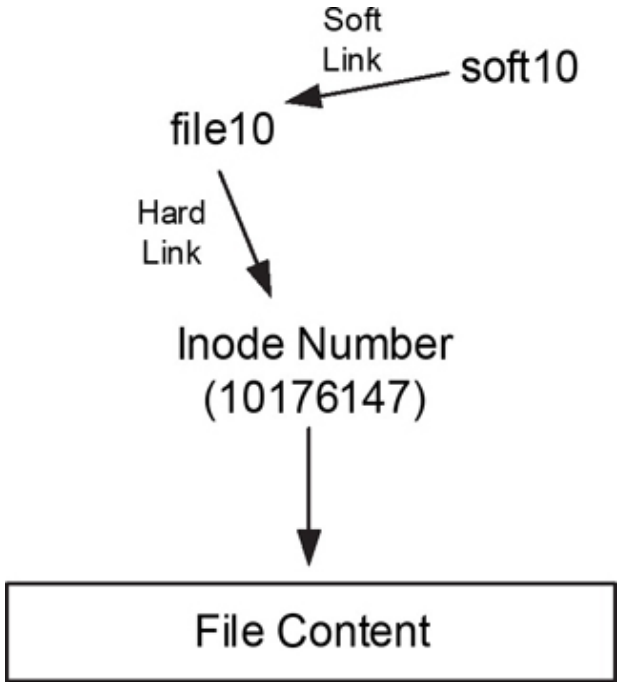


Figure 3-2 Soft Link

A soft link can cross a file system boundary and it can be used to link directories, as it simply uses the pathname of the destination object.

To create a soft link for *file10* as *soft10* in the same directory, use the *ln* command with the -s switch:

```
[root@server1 ~]# ln -s file10 soft10
```

After you have created the link, issue *ls -l* and notice the letter “l” as the first character in column 2 of the output. Also notice the arrow that is pointing from the linked file to the original file. Both of these indicate that *soft10* is merely a pointer to *file10*. The -i option displays the associated inode numbers in the first column. See the output of **ls -il** below:

```
[root@server1 ~]# ls -il file10 soft10
10176147 -rw-r--r--. 2 root root 0 Sep  6 14:29 file10
10185731 lrwxrwxrwx. 1 root root 6 Sep  6 18:22 soft10 -> file10
```

If you remove the original file (*file10* in this case), the link *soft10* will stay but points to something that does not exist.

RHEL 8 has four soft-linked directories under */*. They are:

```
lrwxrwxrwx.   1 root root    7 Aug 12  2018 bin -> usr/bin
lrwxrwxrwx.   1 root root    7 Aug 12  2018 lib -> usr/lib
lrwxrwxrwx.   1 root root    9 Aug 12  2018 lib64 -> usr/lib64
lrwxrwxrwx.   1 root root    8 Aug 12  2018 sbin -> usr/sbin
```

The syntax for creating soft-linked directories is exactly the same as that for soft-linked files.

Differences between Copying and Linking

There are key differences between copying and linking operations. This subsection will discuss when to use copy and when to opt for a soft or hard link. Table 3-14 highlights the main differences between the two:

Copying	Linking
Creates a duplicate of the source file. If either file is modified, the other file will remain intact.	Creates a shortcut that points to the source file. The source can be accessed or modified using either the source file or the link.
Each copied file stores its own data at a unique location.	All linked files point to the same data.
Each copied file has a unique inode number with its unique metadata.	**Hard Link:** All hard-linked files share the same inode number, and hence the metadata. **Symlink:** Each symlinked file has a unique inode number, but the inode number stores only the pathname to the source.
If a copy is moved, erased, or renamed, the source file will have no impact, and vice versa.	**Hard Link:** If the hard link is weeded out, the other file and the data will remain untouched. **Symlink:** If the source is deleted, the soft link will be broken and become meaningless. If the soft link is removed, the source will have no impact.
Copy is used when the data needs to be edited independent of the other.	Links are used when access to the same source is required from multiple locations.
Permissions on the source and the copy are managed independent of each other.	Permissions are managed on the source file.

Table 3-14 Copying vs. Linking

Keep these differences in mind when you need to decide whether to use copy or a link.

Exercise 3-2: Create and Manage Hard Links

This exercise should be done on *server1* as *root*.

In this exercise, you will create an empty file *hard1* under */tmp* and display its attributes (the inode number,

permissions, number of links, owning user, owning group, size, and timestamp). You will create two hard links *hard2* and *hard3* for it, and list the attributes for all three files. You will edit *hard2* and add some text. You will list the attributes for all three files again, and observe identicalness in all attributes except for the names of files. You will remove *hard1* and *hard3* and list the attributes again for the remaining file. You will notice a decrease in the link count by 2.

1. Create an empty file */tmp/hard1*, and display the long file listing including the inode number:

```
[root@server1 ~]# touch /tmp/hard1
[root@server1 ~]# ls -li /tmp/hard1
103198 -rw-r--r--. 1 root root 0 Jan  8 09:11 /tmp/hard1
```

The file listing indicates the inode number in column 1, followed by permissions (column 2), number of links (column 3), owning user and group (columns 4 and 5), size (column 6), timestamp (columns 7, 8, and 9), and filename (column 10).

2. Create two hard links called *hard2* and *hard3* under */tmp*, and display the long listing:

```
[root@server1 ~]# ln /tmp/hard1 /tmp/hard2
[root@server1 ~]# ln /tmp/hard1 /tmp/hard3
[root@server1 ~]# ls -li /tmp/hard*
103198 -rw-r--r--. 3 root root 0 Jan  8 09:11 /tmp/hard1
103198 -rw-r--r--. 3 root root 0 Jan  8 09:11 /tmp/hard2
103198 -rw-r--r--. 3 root root 0 Jan  8 09:11 /tmp/hard3
```

Observe the file listing. All attributes are identical.

3. Edit file *hard2* and display the long listing for all three files again:

```
[root@server1 ~]# vim /tmp/hard2
[root@server1 ~]# ls -li /tmp/hard*
103198 -rw-r--r--. 3 root root 37 Jan  8 09:21 /tmp/hard1
103198 -rw-r--r--. 3 root root 37 Jan  8 09:21 /tmp/hard2
103198 -rw-r--r--. 3 root root 37 Jan  8 09:21 /tmp/hard3
```

Observe the size and timestamp columns. They are identical.

4. Remove file *hard1* and display the long listing for the rest of the files:

```
[root@server1 ~]# rm -f /tmp/hard1 /tmp/hard3
[root@server1 ~]# ls -li /tmp/hard*
103198 -rw-r--r--. 1 root root 37 Jan  8 09:21 /tmp/hard2
```

The number of links reduced to 1, all other attributes are the same. You can still access the same data through this last file.

Exercise 3-3: Create and Manage Soft Links

This exercise should be done on *server1* as *root*.

In this exercise, you will create a soft link *soft1* under */root* pointing to */tmp/hard2*. You will display the attributes (the inode number, permissions, number of links, owning user, owning group, size, and timestamp) for both files. You will open *soft1* for edit and list the attributes after editing. You will remove *hard2* and then list *soft1*. You will notice that *soft1* becomes invalid, pointing to something that does not exist. Remove *soft1* to complete the exercise.

1. Create soft link */root/soft1* pointing to */tmp/hard2*, and display the long file listing for both:

```
[root@server1 ~]# ln -s /tmp/hard2 /root/soft1
[root@server1 ~]# ls -li /tmp/hard2 /root/soft1
9373383 lrwxrwxrwx. 1 root root 10 Jan  8 09:49 /root/soft1 -> /tmp/hard2
 103198 -rw-r--r--. 1 root root 37 Jan  8 09:21 /tmp/hard2
```

The file listing indicates the inode number in column 1, followed by permissions (column 2), number of links (column 3), owning user and group (columns 4 and 5), size (column 6), timestamp (columns 7, 8, and 9), and filename (column 10). The soft link file has an “l” prefixed to column 2 and an arrow pointing to the actual file after column 10. Both are indications of a soft link. Notice the file size (10 bytes for the full path */tmp/hard2*) for *soft1*. Observe similarities and other differences.

2. Edit *soft1* and display the long listing again:

```
[root@server1 ~]# vim /root/soft1
[root@server1 ~]# ls -li /tmp/hard2 /root/soft1
9373383 lrwxrwxrwx. 1 root root 10 Jan  8 09:49 /root/soft1 -> /tmp/hard2
 103198 -rw-r--r--. 1 root root 76 Jan  8 09:56 /tmp/hard2
```

The number of bytes for *hard1* and the timestamp reflects the editing. The rest of the attributes are the same.

3. Remove *hard2* and display the long listing:

```
[root@server1 ~]# rm -f /tmp/hard2
[root@server1 ~]# ls -li /tmp/hard2 /root/soft1
ls: cannot access '/tmp/hard2': No such file or directory
9373383 lrwxrwxrwx. 1 root root 10 Jan  8 09:49 /root/soft1 -> /tmp/hard2
```

The actual file, *hard2*, is gone and the link is now invalid. You can remove it with **rm -f /root/soft1**.

Chapter Summary

This chapter started with an introduction of common file types that are available in RHEL. A file’s type is determined by the type of data it stores. Regular is the most common type of file that stores plain text or binary information. Directories are also very common and there are thousands of them on a typical RHEL system. Other file types include device files and linked files.

We looked at creating and manipulating compressed files and compressed archives. This is a common practice among Linux users for storing old files and transferring a large amount of data to remote systems.

We learned about the vim editor, which is a favorite text file creation and editing tool. We looked at its various modes of operations and switching between them. The basics of vim were discussed, including how to start and insert text, navigate and search for text, copy and paste text, modify and delete text, save edits, and quit with or without saving the changes.

Next, we described file and directory manipulation tools for operations such as creating, listing, displaying, copying, moving, renaming, and removing them. Normal and super users perform these tasks on Linux systems very often.

We examined soft and hard links, and their advantages and limitations. Based on the knowledge gained, we can identify and create the type of link we need for a particular use case.

Finally, we explored the differences between file copying and file linking.

Check Your Understanding

1. Which three Linux utilities can be used to determine a file's type?
2. What is the function of the *tac* command?
3. There are two hard linked files in a directory. How would you identify them?
4. Which *vim* mode allows to execute advanced copy and move functions?
5. A file compressed with *bzip2* can be uncompressed with the *gunzip* command. True or False?

6. Which numeric identifier does the kernel use to determine the uniqueness of a device within a device driver type?
7. What are the two indications in the output of *ls -l* that tells us if the file is a symlink?
8. The *rmdir* command without any switches can be used to remove an entire directory structure. True or False?
9. What would the command *tar pczf output.file /usr/local* do if it is executed by a normal user?
10. The *ls -l* command produces 9 columns in the output by default. True or False?
11. What would the command *wc -c file1* show?
12. What does the kernel use the major number for?
13. The *tail* command can be used to view a file while it is being updated. True or False?
14. Soft linked directories cannot cross file system boundaries, but hard linked directories can. True or False?
15. A file must have the *.exe* extension in order to run. True or False?
16. The *tar* command can be used to archive files with their SELinux contexts. True or False?
17. What would the command *touch file1* do on an existing *file1* file?

Answers to Check Your Understanding

1. The *stat*, *file*, and *ls* commands can be used to determine a file's type.
2. The *tac* command is used to display a text file in reverse.
3. You can identify them by running *ls -li*.
4. The last line mode (extended mode) allows users to copy or move lines.

5. False. The file will have to be uncompressed with either *bzip2* or *bunzip2*.
6. The kernel uses the minor number to identify the uniqueness of a device within a particular device category.
7. A symlink file line entry in the *ls -l* command output begins with the letter l and has an arrow pointing to the source file.
8. False. The *rmdir* command is used to remove empty directories.
9. This command will create a gzip'ed tar archive called *output.file* of the */usr/local* directory with file permissions preserved.
10. True.
11. This command will show the number of bytes in *file1*.
12. The kernel employs the major number to identify the device type.
13. True. You need to include the -f switch in the command.
14. False. Soft linked directories can and hard linked directories cannot cross file system boundaries.
15. False.
16. True. The *tar* command has the --selinux switch that provides this support.
17. This command will update the access time on *file1*.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 3-1: Archive, List, and Restore Files

As *user1* on *server1*, execute the *tar* command to create a *gzip*-compressed archive of the */etc* directory. Run the *tar* command again to create a *bzip2*-compressed archive of the */etc* directory. Compare the file sizes of the two archives. Run the *tar* command and uncompress and restore both archives without specifying the compression tool used. (Hint: Compression and Archiving).

Lab 3-2: Practice the vim Editor

As *user1* on *server1*, create a file called *vipractice* in the home directory using *vim*. Type (do not copy and paste) each sentence from Lab 3-1 on a separate line (do not worry about line wrapping). Save the file and quit the editor. Open *vipractice* in *vim* again and reveal line numbering. Copy lines 2 and 3 to the end of the file to make the total number of lines in the file to 6. Move line 3 to make it line 1. Go to the last line and append the contents of the *.bash_profile*. Substitute all occurrences of the string “Profile” with “Pro File”, and all occurrences of the string “profile” with “pro file”. Remove lines 5 to 8. Save the file and quit *vim*. Provide a count of lines, words, and characters in the *vipractice* file using the *wc* command. (Hint: File Editing).

Lab 3-3: File and Directory Operations

As *user1* on *server1*, create one file and one directory in the home directory. List the file and directory and observe the permissions, ownership, and owning group. Try to move the file and the directory to the */var/log* directory and notice what happens. Try again to move them to the */tmp* directory. Duplicate the file with the *cp* command, and then rename the duplicated file using any name. Remove the file and directory created for this lab. (Hint: File and Directory Operations).

Chapter 04

Advanced File Management

This chapter describes the following major topics:

- Understand ugo/rwx access permissions on files and directories
- Know symbolic and octal notations of permission allocation
- Modify permissions for file owner, owning group, and others
- Calculate and set default permissions on new files and directories
- Comprehend and configure special permission bits: setuid, setgid, and sticky
- Use setgid bit for group collaboration
- Apply sticky bit on public and shared writable directories
- Search for files in a variety of different ways
- Grasp and manage extended access controls for named users and named groups
- Set default extended access controls on directories

RHCSA Objectives:

10. List, set, and change standard ugo/rwx permissions
31. Create and configure set-GID directories for collaboration
34. Diagnose and correct file permission problems
51. Create and use file access control lists

Permissions are set on files and directories to prevent access from unauthorized users. Users are grouped into three distinct categories. Each user category is then assigned required permissions. Permissions may be modified using one of two available methods. The user mask may be defined for individual users so new files and directories they create always get preset permissions. Every file in Linux has an owner and a group.

RHEL offers three additional permission bits to control user access to certain executable files and shared directories. A directory with one of these bits can be used for group collaboration. A public or group writable directory may also be configured with one of these bits to prevent file deletion by non-owners.

There is a tool available in RHEL that proves to be very helpful in searching for files at the specified location using a range of options to specify the search criteria. This tool may be set to execute an action on the output files as they are found. There is another tool available that users may use for simple searches.

Access Control Lists allows the administrator to enforce extended security attributes on files and directories for specific users or specific user groups. These attributes are on top of the standard Linux access permissions for owning users and owning group members. A directory can have default ACL settings applied to it to allow content sharing among users without having to change permissions on each new file and subdirectory created within it.

File and Directory Access Permissions

Linux is a multi-user operating system that allows hundreds of users the ability to log in and work concurrently. In addition, the

OS has hundreds of thousands of files and directories that it must maintain securely in order to warrant a successful system and application operation from a security standpoint. Given these factors, it is imperative to regulate user access to files and directories and grant them appropriate rights to carry out their designated functions without jeopardizing system security. This control of permissions on files and directories may also be referred to as user *access rights*.

Determining Access Permissions

Access permissions on files and directories allow administrative control over which users (permission classes) can access them and to what level (permission types). File and directory permissions discussed in this section are referred to as *standard ugo/rwx permissions*.

Permission Classes

Users are categorized into three unique classes for maintaining file security through access rights. These classes are *user* (u), *group* (g), and *other* (o, also referred to as *public*). These permission classes represent the owner, the set of users with identical access requirements, and everyone else on the system, respectively. There is another special user class called *all* (a) that represents the three user classes combined.

Permission Types

Permissions control what actions can be performed on a file or directory and by whom. There are three types of permissions bits —read (r), write (w), and execute (x)—and they behave differently for files and directories. For files, the permissions allow viewing and copying (read), modifying (write), and running (execute). And in the case of directories, they allow listing contents with *ls* (read); creating, erasing, and renaming files and subdirectories (write); and enter (with the *cd* command) into it (execute).

If a read, write, or execute permission bit is not desired, the hyphen character (-) is used to represent its absence.

Permission Modes

A permission mode is used to add (+), revoke (-), or assign (=) a permission type to a permission class. You can view the permission settings on files and directories in the long listing of the *ls* command. This information is encapsulated in column 1 of the output, a sample of which is shown below:

- rwx rw- r--

The first character indicates the type of file: - for regular file, d for directory, l for symbolic link, c for character device file, b for block device file, p for named pipe, s for socket, and so on.

The next nine characters—three groups of three characters—show the read (r), write (w), and execute (x) permissions for the three user classes: user (owner), group, and other (public), respectively. The hyphen character (-) represents a permission denial for that level.

Modifying Access Permission Bits

The *chmod* command modifies access rights. It works identically on files and directories. *chmod* can be used by *root* or the file owner, and can modify permissions specified in one of two ways: *symbolic* or *octal*. Symbolic notation uses a combination of letters (ugo/rwx) and symbols (+, -, =) to add, revoke, or assign permission bits. The octal notation (a.k.a. the *absolute* representation) uses a three-digit numbering system ranging from 0 to 7 to express permissions for the three user classes. Octal values are given in Table 4-1.

Octal Value	Binary Notation	Symbolic Notation	Explanation
0	000	---	No permissions
1	001	--x	Execute permission only
2	010	-w-	Write permission only
3	011	-wx	Write and execute permissions
4	100	r--	Read permission only
5	101	r-x	Read and execute permissions
6	110	rw-	Read and write permissions
7	111	rwx	Read, write, and execute permissions

Table 4-1 Octal Permission Notation

In Table 4-1, each “1” corresponds to an r, w, or x, and each “0” corresponds to the hyphen character (-) for no permission at that level. Figure 4-1 shows weights associated with each digit position in the 3-digit octal numbering model.

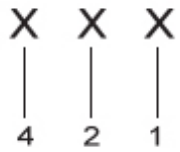


Figure 4-1 Permission Weights

The position to the right is weight 1, the middle position is weight 2, and the left position is weight 4. If we assign a permission of 6, for example, it would correspond to the left and middle positions. Similarly, a permission of 2 would point to the middle position only.

Exercise 4-1: Modify Permission Bits Using Symbolic Form

This exercise should be done on *server1* as *user1*.

For this exercise, presume that a file called *permfile1* exists with read permission for the owner (*user1*), owning group (*user1*), and other, as shown below. If the permissions vary, bring them to the desired state by executing **chmod 444 permfile1** prior to starting the exercise.

```
-r--r--r--. 1 user1 user1 0 Sep  6 20:02 permfile1
```

In this exercise, you will add an execute bit for the owner and a write bit for group and public. You will then revoke the write bit from public and assign read, write, and execute bits to the three user categories at the same time. Finally, you will revoke write from the owning group and write and execute bits from public. The *chmod* command accepts the -v switch to display what it has changed. You may alternatively view the long listing after each command execution for verification.

1. Add an execute bit for the owner:

```
[user1@server1 ~]$ chmod u+x permfile1 -v
mode of 'permfile1' changed from 0444 (r--r--r--) to 0544 (r-xr--r--)
```

2. Add a write bit for group members and public:

```
[user1@server1 ~]$ chmod -v go+w permfile1
mode of 'permfile1' changed from 0544 (r-xr--r--) to 0566 (r-xrw-rw-)
```

3. Remove the write permission for public:

```
[user1@server1 ~]$ chmod -v o-w permfile1
mode of 'permfile1' changed from 0566 (r-xrw-rw-) to 0564 (r-xrw-r--)
```

4. Assign read, write, and execute permission bits to all three user categories:

```
[user1@server1 ~]$ chmod a=rwx -v permfile1
mode of 'permfile1' changed from 0564 (r-xrw-r--) to 0777 (rwxrwxrwx)
```

5. Revoke write bit from the group members and write and execute bits from public:

```
[user1@server1 ~]$ chmod g-w,o-wx permfile1 -v
mode of 'permfile1' changed from 0777 (rwxrwxrwx) to 0754 (rwxr-xr--)
```

Exercise 4-2: Modify Permission Bits Using Octal Form

This exercise should be done on *server1* as *user1*.

For this exercise, a file called *permfile2* exists with read permission for the owner (*user1*), owning group (*user1*), and other, as shown below. If the permissions vary, bring them to the desired state by executing **chmod 444 permfile2** prior to starting the exercise.

```
-r--r--r--. 1 user1 user1 0 Sep  6 21:12 permfile2
```

In this exercise, you will add an execute bit for the owner and a write permission bit for group and public. You will then revoke the write bit from public and assign read, write, and execute permissions to the three user categories at the same time. The *chmod* command accepts the -v flag to display what it has changed. You may alternatively view the long listing after each command execution for verification.

1. Add an execute bit for the owner:

```
[user1@server1 ~]$ chmod -v 544 permfile2
mode of 'permfile2' changed from 0444 (r--r--r--) to 0544 (r-xr--r--)
```

2. Add a write permission bit for group and public:

```
[user1@server1 ~]$ chmod -v 566 permfile2
mode of 'permfile2' changed from 0544 (r-xr--r--) to 0566 (r-xrw-rw-)
```

3. Remove the write bit for public:

```
[user1@server1 ~]$ chmod -v 564 permfile2
mode of 'permfile2' changed from 0566 (r-xrw-rw-) to 0564 (r-xrw-r--)
```

4. Assign read, write, and execute permission bits to all three user categories:

```
[user1@server1 ~]$ chmod 777 -v permfile2
mode of 'permfile2' changed from 0564 (r-xrw-r--) to 0777 (rwxrwxrwx)
```

Default Permissions

Linux assigns *default permissions* to a file or directory at the time of its creation. Default permissions are calculated based on the

umask (*user mask*) permission value subtracted from a preset *initial* permissions value.

The umask is a three-digit octal value (also represented in symbolic notation) that refers to read, write, and execute permissions for owner, group, and public. Its purpose is to set default permissions on new files and directories without touching the permissions on existing files and directories. The default umask value is set to 0022 for the *root* user and 0002 for all normal users. Note that the left-most 0 has no significance. Run the *umask* command without any options and it will display the current umask value in octal notation:

```
[user1@server1 ~]$ umask
0002
```

Run the command again but with the -S option to display the umask in symbolic form:

```
[user1@server1 ~]$ umask -S
u=rwx,g=rwx,o=rx
```

The predefined initial permission values are 666 (rw-rw-rw-) for files and 777 (rwxrwxrwx) for directories. Even if the umask is set to 000, the new files will always get a maximum of 666 permissions; however, you can add the executable bits explicitly with the *chmod* command if desired.

Calculating Default Permissions

Consider the following example to calculate the default permission values on files for normal users:

Initial Permissions	666	
umask	– 002	(subtract)
==========	==========	
Default Permissions	664	

This is an indication that every new file will have read and write permissions assigned to the owner and the owning group, and a read-only permission to other.

To calculate the default permission values on directories for normal users:

Initial Permissions	777	
umask	– 002	(subtract)
==========	==========	
Default Permissions	775	

This indicates that every new directory created will have read, write, and execute permissions assigned to the owner and the owning group, and read and execute permissions to everyone else.

If you want different default permissions set on new files and directories, you will need to modify the umask. You first need to ascertain the desired default values. For instance, if you want all new files and directories to get 640 and 750 permissions, you can set umask to 027 by running either of the following:

```
[user1@server1 ~]$ umask 027
[user1@server1 ~]$ umask u=rwx,g=rx,o=
```

or

The new value becomes effective right away, and it will only be applied to files and directories created thereafter. The existing files and directories will remain intact. Now create *tempfile1* and *tempdir1* as *user1* under */home/user1* to test the effect of the new umask:

```
[user1@server1 ~]$ touch tempfile1
[user1@server1 ~]$ ls -l tempfile1
-rw-r-----. 1 user1 user1 0 Sep  6 21:42 tempfile1
[user1@server1 ~]$ mkdir tempdir1
[user1@server1 ~]$ ls -ld tempdir1
drwxr-x---. 2 user1 user1 6 Sep  6 21:42 tempdir1
```

The above examples show that the new file and directory were created with different permissions. The file got (666 – 027 = 640) and the directory got (777 – 027 = 750) permissions.

The umask value set at the command line will be lost as soon as you log off. In order to retain the new setting, place it in an appropriate shell startup files discussed in Chapter 06 “Advanced User Management”.

Special File Permissions

Linux offers three types of special permission bits that may be set on binary executable files or directories that respond differently to non-*root* users for certain operations. These permission bits are *set user identifier* bit (commonly referred to as *setuid* or *suid*), *set group identifier* bit (a.k.a. *setgid* or *sgid*), and *sticky* bit.

The setuid and setgid bits may be defined on binary executable files to provide non-owners and non-group members the ability to run them with the privileges of the owner or the owning group, respectively. The setgid bit may also be set on shared directories for group collaboration. The sticky bit may be set on public directories for inhibiting file erasures by non-owners.

The setuid and sticky bits may be set on directories and files; however, they will have no effect.

The use of the special bits should be regulated and monitored to evade potential security issues to system operation and applications.

The setuid Bit on Binary Executable Files

The setuid flag is set on binary executable files at the file owner level. With this bit set, the file is executed by non-owners with the same privileges as that of the file owner. A common example is the *su* command that is owned by the *root* user. This command has the setuid bit enabled on it by default. See the underlined “s” in the owner’s permission class below:

```
[root@server1 ~]# ls -l /usr/bin/su
-rwsr-xr-x. 1 root root 52784 Dec 11  2018 /usr/bin/su
```

The *su* (*switch user*) command allows a user to switch to a different user account with the password for the target user. However, the *root* user can switch into any other user account without being prompted for a password. When a normal user executes this command, it will run as if *root* (the owner) is

running it and, therefore, the user is able to run it successfully and gets the desired result.

Exercise 4-3: Test the Effect of setuid Bit on Executable Files

This exercise should be done on *server1* as *root* and *user1*.

In this exercise, you will need two terminal windows, one with a *root* session running and another with *user1* on it. As *user1*, you will switch into *root* and observe what happens. As *root*, you will then revoke the setuid bit from the */usr/bin/su* file and retry switching into *root* again. After the completion of the exercise, you will restore the setuid bit on */usr/bin/su*.

1. Log in as *root* and have a terminal window open (let's call it Terminal 1). Open another terminal (let's name it Terminal 2) and run the following to switch into *user1*:

```
[root@server1 ~]# su - user1
```

2. On Terminal 2, run the *su* command to switch into *root*:

```
[user1@server1 ~]$ su - root
Password:
[root@server1 ~]#
```

The output confirms the switch.

3. On Terminal 1, revoke the setuid bit from */usr/bin/su*:

```
[root@server1 ~]# chmod -v u-s /usr/bin/su
mode of '/usr/bin/su' changed from 4755 (rwsr-xr-x) to 0755 (rwxr-xr-x)
```

The file is still executable by everyone as indicated by the execute flag; however, it will prevent regular non-owning users from switching accounts, as they have lost that special elevated privilege.

4. On Terminal 2, press Ctrl+d to log off as *root*.
5. On Terminal 2, switch back into *root* and see what happens:

```
[user1@server1 ~]$ su -
Password:
su: Authentication failure
```

user1 gets an “authentication failure” message even though they entered the correct password.

6. On Terminal 1, restore the setuid bit on */usr/bin/su*:

```
[root@server1 ~]# chmod -v +4000 /usr/bin/su
mode of '/usr/bin/su' changed from 0755 (rwxr-xr-x) to 4755 (rwsr-xr-x)
```

With the argument +4000, the *chmod* command enables setuid on the specified file without altering any existing underlying permissions. Alternatively, you can use the symbolic notation as follows:

```
[root@server1 ~]# chmod u+s /usr/bin/su
```

If the file already has the “x” bit set for the group, the long listing will show a lowercase “s”, otherwise it will list it with an uppercase “S”.

The setuid bit has no effect on directories.

The setgid Bit on Binary Executable Files

The setgid attribute is set on binary executable files at the group level. With this bit set, the file is executed by non-owners with the exact same privileges as that of the group members. A common example is the *write* command that is owned by the *root* user with *tty* as the owning group. This command has the setgid bit enabled on it by default. See the “s” in the group’s permission class below:

```
[root@server1 ~]# ls -l /usr/bin/write
-rwxr-sr-x. 1 root tty 22544 Dec 11  2018 /usr/bin/write
```

The *write* command allows users to write a message on another logged-in user’s terminal. By default, normal users are allowed this special elevated privilege because of the presence of the setgid flag on the file. When a normal user executes this command to write to the terminal of another user, the command will run as if a member of the *tty* group is running it, and the user is able to execute it successfully.

Exercise 4-4: Test the Effect of setgid Bit on Executable Files

This exercise should be done on *server1* as *root* and *user1*.

In this exercise, you will need two terminal windows, one with a *root* session running and the other with *user1* on it. As *user1*, you will produce a list of logged-in users; try to send the *root* user a message and observe what happens. As *root*, you will then revoke the setgid bit from the */usr/bin/write* file and retry sending another message to *root* as *user1*. After the completion of the exercise, you will restore the setgid bit on */usr/bin/write*.

1. Log in as *root* and have a terminal window open (let's call it Terminal 1). Open another terminal (let's name it Terminal 2) and run the following to switch into *user1*:

```
[root@server1 ~]# su - user1
```

The *su* (switch user) command allows a user to switch to a different user account provided the user knows the password of the target user account. However, the *root* user can switch into any other user account without being prompted for a password.

2. On Terminal 2, run the *who* command to list users who are currently logged on:

```
[user1@server1 ~]$ who
root     pts/0        2019-09-06 07:58 (192.168.0.219)
user1    pts/1        2019-09-06 19:36 (192.168.0.219)
```

The output discloses that there are two users—*root* and *user1*—currently signed in.

3. On Terminal 2, execute the *write* command as follows to send a message to *root*:

```
[user1@server1 ~]$ write root
```

4. On Terminal 1, you will see the following message from *user1*:

```
[root@server1 ~]#
Message from user1@server1.example.com on pts/1 at 13:56 ...
_
```

Any text you type on Terminal 2 will appear on Terminal 1.

5. On Terminal 1, revoke the setgid bit from */usr/bin/write*:

```
[root@server1 ~]# chmod g-s /usr/bin/write -v
mode of '/usr/bin/write' changed from 2755 (rwxr-sr-x) to 0755 (rwxr-xr-x)
```

The file is still executable by everyone as indicated by the execute flag; however, it will prevent them from writing to the terminals of other users, as they have lost that special elevated privilege.

6. On Terminal 2, press Ctrl+c to terminate the current write session.
7. On Terminal 2, rewrite to *root* and see what happens:

```
[user1@server1 ~]$ write root
write: effective gid does not match group of /dev/pts/1
```

user1 gets an error as indicated in the above output.

8. On Terminal 1, restore the setgid bit on */usr/bin/write*:

```
[root@server1 ~]# chmod -v +2000 /usr/bin/write
mode of '/usr/bin/write' changed from 0755 (rwxr-xr-x) to 2755 (rwxr-sr-x)
```

With the argument +2000, the *chmod* command enables setgid on the specified file without altering any existing underlying permissions. Alternatively, you can use the symbolic form as follows:

```
[root@server1 ~]# chmod g+s /usr/bin/write
```

If the file already has the “x” bit set for the group, the long listing will show a lowercase “s”, otherwise it will list it with an uppercase “S”.

The setgid bit has an impact on shared (and public) directories (next subsection).

The setgid Bit on Shared Directories

The setgid bit can also be set on group-shared directories to allow files and subdirectories created underneath to automatically inherit the directory’s owning group. This saves

group members who are sharing the directory contents from changing the group ID for every new file and subdirectory that they add. The standard behavior for new files and subdirectories is to always receive the creator's group.

Exercise 4-5: Set up Shared Directory for Group Collaboration

This exercise should be done on *server1* as *root* and two test users *user100* and *user200*. Create the user accounts by running **useradd user100** and **useradd user200** (if they don't already exist) as *root*.

In this exercise, you will create a group called *sgrp* with GID 9999, and add *user100* and *user200* to this group as members with shared data needs. You will create a directory called */sdir* with ownership and owning group belonging to *root* and *sgrp*, then set the setgid bit on */sdir* and test. For details on managing users and groups, consult Chapter 05 "Basic User Management" and Chapter 06 "Advanced User Management".

1. Add group *sgrp* with GID 9999 with the *groupadd* command:

```
[root@server1 ~]# groupadd -g 9999 sgrp
```

2. Add *user100* and *user200* as members to *sgrp* using the *usermod* command:

```
[root@server1 ~]# usermod -aG sgrp user100
[root@server1 ~]# usermod -aG sgrp user200
```

3. Create */sdir* directory:

```
[root@server1 ~]# mkdir /sdir
```

4. Set ownership and owning group on */sdir* to *root* and *sgrp*, using the *chown* command:

```
[root@server1 ~]# chown root:sgrp /sdir
```

5. Set the setgid bit on */sdir* using the *chmod* command:

```
[root@server1 ~]# chmod g+s /sdir
```

6. Add write permission to the group members on */sdir* and revoke all permissions from public:

```
[root@server1 ~]# chmod g+w,o-rx /sdir
```

7. Verify the attributes set in the previous three steps using the *ls* command on */sdir*:

```
[root@server1 ~]# ls -ld /sdir
drwxrws---. 2 root sgrp 6 Sep  7 11:57 /sdir
```

8. Switch or log in as *user100* and change to the */sdir* directory:

```
[root@server1 ~]# su - user100
[user100@server1 ~]$ cd /sdir
[user100@server1 sdir]$
```

9. Create a file and check the owner and owning group on it:

```
[user100@server1 sdir]$ touch file100
[user100@server1 sdir]$ ls -l file100
-rw-rw-r--. 1 user100 sgrp 0 Sep  7 12:07 file100
```

10. Log out as *user100*, and switch or log in as *user200* and change to the */sdir* directory:

```
[user100@server1 sdir]$ exit
logout
[root@server1 ~]# su - user200
[user200@server1 ~]$ cd /sdir
[user200@server1 sdir]$
```

11. Create a file and check the owner and owning group on it:

```
[user200@server1 sdir]$ touch file200
[user200@server1 sdir]$ ls -l file200
-rw-rw-r--. 1 user200 sgrp 0 Sep  7 12:11 file200
```

As shown above, the owning group for each file is the same, *sgrp*, and the group members have identical rights (read and write). Both can modify or delete each other's file. The group members own the files, but the owning group will always be *sgrp* to which they both belong.

The Sticky Bit on Public and Shared Writable Directories

The sticky bit is set on public and shared writable directories to protect files and subdirectories owned by normal users from being deleted or moved by other normal users. This attribute is set on the */tmp* and */var/tmp* directories by default as depicted below; however, it can be applied to any writable directory:

```
[root@server1 ~]# ls -l /tmp /var/tmp -d
drwxrwxrwt. 15 root root 4096 Sep  7 12:17 /tmp
drwxrwxrwt.  7 root root 4096 Sep  6 22:03 /var/tmp
```

Notice the underlined letter “t” in other’s permission fields. This indicates the presence of this attribute on the two directories.

Exercise 4-6: Test the Effect of Sticky Bit

This exercise should be done on *server1* as *root* and two test users *user100* and *user200*. Create the user accounts by running **useradd user100** and **useradd user200** (if they don’t already exist) as *root*.

In this exercise, you will create a file under */tmp* as *user100* and then try to delete it as *user200*. You will unset the sticky bit on */tmp* and try to erase the file again. After the completion of the exercise, you will restore the sticky bit on */tmp*. For details on managing users and groups, consult Chapter 05 “Basic User Management” and Chapter 06 “Advanced User Management”.

1. Switch or log in as *user100* and change to the */tmp* directory:

```
[root@server1 ~]# su - user100
[user100@server1 ~]$ cd /tmp
```

2. Create a file called *stickyfile*:

```
[user100@server1 tmp]$ touch stickyfile
```

3. Log out as *user100* and switch or log in as *user200* and change to the */tmp* directory:

```
[user100@server1 tmp]$ exit
logout
[root@server1 ~]# su - user200
[user200@server1 ~]$ cd /tmp
```

4. Try to erase the file and observe the system reaction:

```
[user200@server1 tmp]$ rm stickyfile
rm: remove write-protected regular empty file 'stickyfile'? y
rm: cannot remove 'stickyfile': Operation not permitted
```

It says, “Operation not permitted”. The user cannot remove the file owned by another user.

5. Log out as *user100* and revoke the sticky bit from */tmp* as *root* and confirm:

```
[user100@server1 tmp]$ exit
logout
[root@server1 ~]# chmod o-t /tmp
[root@server1 ~]# ls -ld /tmp
drwxrwxrwx. 15 root root 4096 Sep  7 13:15 /tmp
```

6. Switch or log back in as *user200* and retry the removal:

```
[root@server1 ~]# su - user200
[user200@server1 ~]$ cd /tmp
[user200@server1 tmp]$ rm stickyfile
rm: remove write-protected regular empty file 'stickyfile'? y
[user200@server1 tmp]$
```

The file is gone. A normal user, *user200*, was able to successfully delete a file, *stickyfile*, that was owned by a different normal user, *user100*, in a public writable directory, */tmp*.

7. Log out as *user200* and restore the sticky bit back on */tmp*:

```
[user200@server1 tmp]$ exit
logout
[root@server1 ~]# chmod -v +1000 /tmp
mode of '/tmp' changed from 0777 (rwxrwxrwx) to 1777 (rwxrwxrwt)
```

With the argument +1000, the *chmod* command sets the sticky bit on the specified directory without altering any existing underlying permissions. Alternatively, you can use the symbolic notation as follows:

```
[root@server1 ~]# chmod o+t /tmp
```

If the directory already has the “x” bit set for public, the long listing will show a lowercase “t”, otherwise it will list it with an uppercase “T”.

The sticky bit can also be set on group writable directories such as */sdir* that you created in the previous exercise.

File Searching

A typical running RHEL system has a few hundred thousand files distributed across several file systems. At times, it is imperative to look for one or more files based on certain criteria. One example would be to find all files owned by employees who left the company over a year ago. Another example would be to search for all the files that have been modified in the past 20 days by a specific user. For such situations, RHEL offers a command called *find*. You supply your search criteria and this command gets you the result. You can also instruct this utility to execute a command on the files as they are found.

Using the find Command

The *find* command recursively searches the directory tree, finds files that match the specified criteria, and optionally performs an action on the files as they are discovered. This powerful tool can be tailored to look for files in a number of ways. The search criteria may include tracking files by name or part of the name, ownership, owning group, permissions, inode number, last access or modification time in days or minutes, size, and file type. Figure 4-2 shows the command syntax.

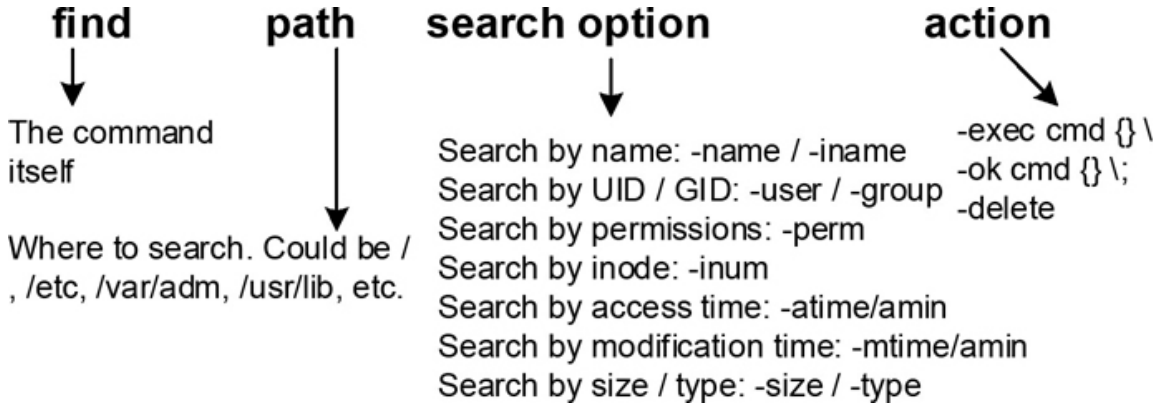


Figure 4-2 find Command Syntax

With *find*, files that match the criteria are located and their full paths are displayed.

To search for a file called *file10* (execute **touch file10** if it does not already exist) by its name (name) in *root*'s home directory, run the *find* command as follows. The period character (.) represents the current directory, which is */root* in this example.

```
[root@server1 ~]# find . -name file10 -print
./file10
```

-print is optional. The *find* command, by default, displays the results on the screen. You do not need to specify this option.

To perform a case-insensitive (-iname) search for files and directories in */dev* that begin with the string "usb" followed by any characters:

```
[root@server1 ~]# find /dev -iname usb*
/dev/bus/usb
/dev/usbmon1
/dev/usbmon0
```

To find files smaller than 1MB (-1M) in size (-size) in the *root* user's home directory (~). You do not need to issue the command from this user's home directory. In fact, you can be anywhere in the directory tree.

```
[root@server1 ~]# find ~ -size -1M
```

The tilde character (~) represents a user's home directory.

To search for files larger than 40MB (+40M) in size (-size) in the */usr* directory:

```
[root@server1 ~]# find /usr -size +40M
```

To find files in the entire root file system (*/*) with ownership (-user) set to user *daemon* and owning group (-group) set to any group other than (-not or ! for negation) *user1*:

```
[root@server1 ~]# find / -user daemon -not -group user1
```

To search for directories (-type) by the name “src” (-name) in */usr* at a maximum of two subdirectory levels below (-maxdepth):

```
[root@server1 ~]# find /usr -maxdepth 2 -type d -name src
```

To run the above search but at least three subdirectory levels beneath */usr*, substitute -maxdepth 2 with -mindepth 3.

To find files in the */etc* directory that were modified (-mtime) more than (the + sign) 2000 days ago:

```
[root@server1 ~]# find /etc -mtime +2000
```

To run the above search for files that were modified exactly 12 days ago, replace “+2000” with “12”.

To find files in the */var/log* directory that have been modified (-mmin) in the past (the - sign) 100 minutes:

```
[root@server1 ~]# find /var/log -mmin -100
```

To run the above search for files that have been modified exactly 25 minutes ago, replace “-100” with “25”.

To search for block device files (-type) in the */dev* directory with permissions (-perm) set to exactly 660:

```
[root@server1 ~]# find /dev -type b -perm 660
```

To search for character device files (-type) in the */dev* directory with at least (-222) world writable permissions (this example would ignore checking the write and execute permissions):

```
[root@server1 ~]# find /dev -type c -perm -222
```

To find files in the */etc/systemd* directory that are executable by at least their owner or group members:

```
[root@server1 ~]# find /etc/systemd -perm /110
```

To search for symlinked files (-type) in */usr* with permissions (-perm) set to read and write for the owner and owning group:

```
[root@server1 ~]# find /usr -type l -perm -ug=rw
```

find is a very useful and powerful file-searching tool with numerous other options available to use. Refer to its manual pages, as there are a ton of examples there. Try some of them out for additional practice.

Using find with -exec and -ok Flags

An advanced use of the *find* command is to perform an action on the files as they are found based on any criteria outlined in the previous subsection and in the command's manual pages. The action may include performing basic file management operations such as copying, erasing, renaming, changing ownership, or modifying permissions on each file found. This is done with the -exec switch. An equivalent option -ok may be used instead, which requires user confirmation before taking an action.

To search for directories in the entire directory tree (/) by the name "core" (-name) and list them (ls -ld) as they are discovered without prompting for user confirmation (-exec):

```
[root@server1 ~]# find / -name core -type d -exec ls -ld {} \;
```

The *find* command replaces {} for each filename as it is found. The semicolon character (;) marks the termination of the command and it is escaped with the backslash character (\).

In the next example, the *find* command uses the -ok switch to prompt for confirmation before it copies each matched file (-name) in */etc/sysconfig* to */tmp*:

```
[root@server1 ~]# find /etc/sysconfig -name *.conf -ok cp {} \;
```

The destination directory (*/tmp*) is specified between {} and \;. There are many advanced examples provided in the *find* command's manual pages. I suggest to try a few of them.

Using the locate Command

The *locate* command is used to discover all occurrences of the specified string as they appear in file pathnames. It can also be used to locate files with certain extensions. Unlike the *find*

command that performs a new search every time you run it, *locate* searches the */var/lib/mlocate/mlocate.db* database, finds matches, and displays them. This database is usually auto-updated daily. Alternatively, it can be created and updated manually with the *updatedb* command.

The output of the *locate* command is the absolute path of files and directories for which the user has access permissions. Here are a few examples that explain the working of this command.

To locate all occurrences of the string “passwd”:

```
[root@server1 ~]# locate passwd
```

Use -n and specify the number of occurrences you wish to see:

```
[root@server1 ~]# locate -n 3 passwd
```

To locate all files with .sh extension and list only the first two of them:

```
[root@server1 ~]# locate .sh -n2
```

To view the number of directories, files, and size information retained in *mlocate.db*:

```
[root@server1 ~]# locate -S
Database /var/lib/mlocate/mlocate.db:
        10,750 directories
        123,760 files
        6,207,398 bytes in file names
        2,977,959 bytes used to store database
```

As with other Linux commands, there are many other options available with the *locate* command. Refer to its manual pages for details and usage.

Access Control Lists (ACLs)

The *Access Control Lists* (ACLs) provide an extended set of permissions that can be applied on files and directories. These permissions are in addition to the standard ugo/rwx permissions and the setuid, setgid, and sticky bit settings. The ACLs define permissions for *named users* and *named groups* using either

octal or symbolic representation of permission allocation. The named users may or may not be part of the same group. ACLs are configured and treated the same way on both files and directories.

ACLs are categorized into two groups based on their type and are referred to as *access ACLs* and *default ACLs*. Access ACLs are set on individual files and directories, whereas default ACLs can only be applied at the directory level with files and subdirectories inheriting them automatically. The directory to be applied the default ACLs needs to have the execute bit set at the public level.

ACL Management Commands

RHEL offers two commands—*getfacl* and *setfacl*—to view and manage ACLs on files and directories. The *getfacl* command is used to display ACL settings, and the *setfacl* command can set, modify, substitute, or delete ACL settings.

The getfacl Command

The *getfacl* command has several options to see the output as desired; however, it reveals all necessary information without furnishing any flags with it. The example below creates an empty file *aclfile1* in */tmp* and then displays the ACLs on it:

```
[root@server1 ~]# cd /tmp
[root@server1 tmp]# touch aclfile1
[root@server1 tmp]# getfacl aclfile1
# file: aclfile1
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

The output returns the names of the file, owner, and owning group. It then exhibits the existing permissions placed on the file: read and write for the owner, and read-only for everyone else. Notice the pair of colon character (:) in the permission rows. There is a space between them where a named user, a named group, or their corresponding UID or GID is inserted when extended permissions are set (see Chapter 05 “Basic User Management” for a description of UID and GID). For instance, the ACL setting “user:1000:r--” would imply that the named user with

UID 1000, who is neither the file owner nor a member of the owning group, is allowed read-only access to this file. Likewise, the ACL “group:dba:rw-” would give the named group (*dba*) read and write access to the file.

In addition, the file’s long listing is also indicative of the presence of ACLs by exhibiting the plus sign (+) right beside the permissions column. An example column from *ls -l* would be -rw-rw-r--+.

The setfacl Command

The *setfacl* command is used to apply, modify, or remove ACL settings. The format for supplying ACL permissions with the command is explained in Table 4-2.

Format	Description
u[ser]:UID:perms	Permissions assigned to a named user (username or UID). The named user must exist in the /etc/passwd file. If this field is left blank, the permission will be applied to the owner of the file or directory, which is equivalent to using the chmod command. See Chapter 05 “Basic User Management” to understand the content and format of the /etc/passwd file.
g[roup]:GID:perms	Permissions assigned to a named group (group name or GID). The named group must exist in the /etc/group file. If this field is left blank, the permission will be applied to the owning group of the file or directory, which is equivalent to using the chmod command. See Chapter 05 “Basic User Management” to understand the content and format of the /etc/group file.
o[ther]:perms	Permissions assigned to users that are neither the owner nor part of the owning group.
m[ask]:perms	Maximum permissions for a named user or a named group. If this is set to rw-, for example, then no named user or group will have permissions beyond read and write.

Table 4-2 setfacl Command Format for Access ACLs

The *setfacl* command provides a multitude of switches to use depending on what you want to achieve. Table 4-3 describes some of them.

Switch	Description
-b	Removes all access ACLs
-d	Applies to default ACLs
-k	Removes all default ACLs
-m	Sets or modifies ACLs
-n	Prevents an automatic recalculation of the mask
-R	Applies recursively to a directory
-x	Removes an access ACL

Table 4-3 setfacl Command Switches

We will explore some of these options shortly.

The Role of the mask Value

The value of the ACL mask determines the maximum allowable permissions placed for a named user or group on a file or directory. If it is set to rw, for instance, no named user or group will exceed those permissions. The mask value is displayed on a separate line in the *getfacl* output. Each time ACLs are modified for a file or directory, the mask is recalculated automatically and applied unless an explicit value is input with the *setfacl* command or the -n option is employed to override this behavior. On *aclfile1*, there are currently no ACLs set, as it is a new file. The *getfacl* command with the -c flag displays the output without the header, as shown below:

```
[root@server1 tmp]# getfacl -c aclfile1
user::rw-
group::r--
other::r--
```

If you want to give read and write permissions to a specific user (*user1*) and change the mask to read-only at the same time, the *setfacl* command will allocate the permissions as mentioned; however, the effective permissions for the named user will only be read-only.

```
[root@server1 tmp]# setfacl -m u:user1:rw,m:r aclfile1
[root@server1 tmp]# getfacl -c aclfile1
user::rw-
user:user1:rw-                  #effective:r--
group::r--
mask::r--
other::r--
```

In the example, *user1* will not be able to modify this file even though it appears they have the write permission. The actual permissions for *user1* include both read and write, but they are curtailed to read-only due to the limitation placed by the mask. Now, let's promote the mask value to include the write bit as well:

```
[root@server1 tmp]# setfacl -m m:rw aclfile1
[root@server1 tmp]# getfacl -c aclfile1
user::rw-
user:user1:rw-
group::r--
mask::rw-
other::r--
```

The actual permissions for *user1* are now boosted to include the write bit to reflect the new higher value.

Exercise 4-7: Identify, Apply, and Erase Access ACLs

This exercise should be done on *server1* as *user1*.

In this exercise, you will create a file *acluser1* as *user1* in */tmp* and check to see if there are any ACL settings on the file. You will apply access ACLs on the file for a named user, *user100*, for read and write access. You will observe the change in the mask value. You will then add another named user, *user200*, to the file for full permissions. You will observe the update in the mask value. You will delete the settings for *user200* and then the rest of the access ACLs from the file.

1. Switch or log in as *user1* and create a file *acluser1* in */tmp*:

```
[root@server1 ~]# su - user1
[user1@server1 ~]$ cd /tmp
[user1@server1 tmp]$ touch acluser1
```

2. Use the *ls* and *getfacl* commands and check for the existence of any ACL entries:

```
[user1@server1 tmp]$ ls -l acluser1
-rw-rw-r--. 1 user1 user1 0 Sep  8 14:32 acluser1
[user1@server1 tmp]$ getfacl acluser1 -c
user::rw-
group::rw-
other::r--
```

The output discloses an absence of ACLs on the file. The owner and group members have read and write permissions and everyone else has the read-only permission.

3. Allocate read and write permissions to *user100* with the *setfacl* command using the octal form:

```
[user1@server1 tmp]$ setfacl -m u:user100:6 acluser1
```

4. Run the *ls* command to check if the plus sign (+) has appeared next to the first column, then run the *getfacl* command to check the new access ACLs and the mask:

```
[user1@server1 tmp]$ ls -l acluser1
-rw-rw-r--+ 1 user1 user1 0 Sep  8 14:52 acluser1
[user1@server1 tmp]$ getfacl -c acluser1
user::rw-
user:user100:rw-
group::rw-
mask::rw-
other::r--
```

A row is added for the named user showing read/write permissions. Another row with the mask is also added and it is set to read/write as well. The mask value is auto-calculated based on the current maximum permissions that a named user or group has. This is reflected in the above output.

5. At this point, you can open another terminal session, switch into *user100*, change directory into */tmp*, and open the file *acluser1* in the *vim* editor. You should be able to modify the file and save it.
6. Add *user200* with full rwx permissions to *acluser1* using the symbolic notation and then show the updated ACL settings:

```
[user1@server1 tmp]$ setfacl -m u:user200:rwx acluser1
[user1@server1 tmp]$ getfacl -c acluser1
user::rw-
user:user100:rw-
user:user200:rwx
group::rw-
mask::rwx
other::r--
```

Notice the updated value for the mask, which is increased to rwx to reflect the maximum permission the named user, *user200*, has. If this file were to be an executable command, you would have been able to run it as *user200* based on the assigned ACLs.

7. Delete the ACL entries set for *user200* and validate:

```
[user1@server1 tmp]$ setfacl -x u:user200 acluser1
[user1@server1 tmp]$ getfacl acluser1 -c
user::rw-
user:user100:rw-
group::rw-
mask::rw-
other::r--
```

Notice the reduction in the mask value to rw-, which now reflects the new current maximum permissions placed on the named user, *user100*.

8. Delete the rest of the ACLs:

```
[user1@server1 tmp]$ setfacl -b acluser1
```

9. Use the *ls* and *getfacl* commands and confirm for the ACLs removal:

```
[user1@server1 tmp]$ ls -l acluser1
-rw-rw-r--. 1 user1 user1 0 Sep  8 14:52 acluser1
[user1@server1 tmp]$ getfacl acluser1 -c
user::rw-
group::rw-
other::r--
```

This concludes the exercise.

You can also create a group such as *aclgroup1* by running **groupadd -g 8000 aclgroup1** as the *root* user and repeat this exercise by adding this group as a named group along with the two named users (*user100* and *user200*).

Default ACLs

A group collaboration on a shared directory gives members of the group identical access on files and subdirectories in the directory. Access ACLs may be applied to the shared directory to give non-group members certain rights. Furthermore, default ACLs can also be set on the shared directory to ensure new files and subdirectories created under the shared directory always have a consistent set of access rights for group and non-group members. This way the users do not have to adjust permissions on each new file and subdirectory they will create. The inheritance works slightly different for files and subdirectories, as indicated below:

- Files receive the shared directory's default ACLs as their access ACLs
- Subdirectories receive both default ACLs and access ACLs as they are

The default ACLs can be described as the maximum discretionary permissions that can be allocated on a directory. Let's perform the following exercise and see how default ACLs are applied, viewed, and erased.

Exercise 4-8: Apply, Identify, and Erase Default ACLs

This exercise should be done on *server1* as *user1*.

In this exercise, you will create a directory *projects* as *user1* under */tmp*. You will set default ACLs on the directory for named users, *user100* and *user200*, to give them full permissions. You will create a subdirectory *prjdir1* and a file *prjfile1* under *projects* and observe the effects of default ACLs on them. You will delete all the default entries at the end of the exercise.

1. Switch or log in as *user1* and create a directory *projects* in */tmp*:

```
[root@server1 ~]# su - user1
[user1@server1 ~]$ cd /tmp
[user1@server1 tmp]$ mkdir projects
```

2. Use the *getfacl* command for an initial look at the permissions on the directory:

```
[user1@server1 tmp]$ getfacl -c projects
user::rwx
group::rwx
other::r-x
```

The output discloses that the owner and group members have full permissions on the directory and everyone else has read and execute permission.

3. Allocate default read, write, and execute permissions to *user100* and *user200* on the directory. Use both octal and symbolic notations and the -d (default) option with the *setfacl* command.

```
[user1@server1 tmp]$ setfacl -dm u:user100:7,u:user200:rwx projects/
[user1@server1 tmp]$ getfacl -c projects/
user::rwx
group::rwx
other::r-x
default:user::rwx
default:user:user100:rwx
default:user:user200:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
```

The named users have the default ACLs. The rest of the default ACL entries are for the directory owner, owning group, and public with the mask reflecting the maximum permissions.

4. Create a subdirectory *prjdir1* under *projects* and observe the ACL inheritance:

```
[user1@server1 projects]$ mkdir prjdir1
[user1@server1 projects]$ getfacl -c prjdir1,
user::rwx
user:user100:rwx
user:user200:rwx
group::rwx
mask::rwx
other::r-x
default:user::rwx
default:user:user100:rwx
default:user:user200:rwx
default:group::rwx
default:mask::rwx
default:other::r-x
```

As stated earlier, the subdirectory *prjdir1* inherited both default and access ACLs. This is reflected in the above output.

5. Create a file *prjfile1* under *projects* and observe the ACL inheritance:

```
[user1@server1 projects]$ touch prjfile1
[user1@server1 projects]$ getfacl -c prjfile1
user::rw-
user:user100:rwx                #effective:rw-
user:user200:rwx                #effective:rw-
group::rwx                      #effective:rw-
mask::rw-
other::r--
```

As stated earlier, the file *prjfile1* inherited the parent directory's default ACLs as its access ACLs minus what the mask limits. The maximum effective access for the named users is rw.

6. At this point, you can log in as one of the named users, change directory into */tmp/projects*, and edit *prjfile1* (add some random text). Then change into the *prjdir1* and create file *file100*. You should be able to perform both tasks successfully.

```
[root@server1 ~]# su - user100
[user100@server1 ~]$ cd /tmp/projects
[user100@server1 projects]$ vim prjfile1
[user100@server1 projects]$ ls -l prjfile1
-rw-rw-r--+ 1 user1 user1 9 Sep  9 09:34 prjfile1
[user100@server1 projects]$ cd prjdir1
[user100@server1 prjdir1]$ touch file100
[user100@server1 prjdir1]$ pwd
/tmp/projects/prjdir1
```

7. Delete all the default ACLs from the *projects* directory as *user1* and confirm:

```
[user100@server1 ~]$ exit
logout
[root@server1 ~]# su - user1
[user1@server1 ~]$ cd /tmp
[user1@server1 tmp]$ setfacl -k projects
[user1@server1 tmp]$ getfacl -c projects
user::rwx
group::rwx
other::r-x
```

This concludes the exercise.

You can create a group such as *aclgroup2* by running **groupadd -g 9000 aclgroup2** as the *root* user and repeat this exercise by adding this group as a named group along with the two named users (*user100* and *user200*).

Chapter Summary

This chapter covered four topics: file and directory permissions, special permissions, file searching, and access control.

We learned classes, types, and modes of permissions, looked at octal and symbolic notations of changing permissions, and applied the knowledge to modify user access on files and directories. We examined the concept of default permissions and how they could be employed on parent directories to enable files and subdirectories created underneath to automatically get the desired permissions. We analyzed the role of the umask value in determining the new default permissions.

We looked at special permission bits that could be set on executable files to gain privileged access, applied on shared directories for content sharing, and enabled on shared and public writable directories to prevent file deletion by non-owning users.

Next, we explored the criteria and the tool to search for files at the specified directory location. We also employed an extended flag for optional execution of an action on the outcome of the search command.

Finally, we discussed access control on files and directories using ACLs. This feature let named users and named groups to have extended access on given files and directories without the need to modify ownership or owning group on them.

Check Your Understanding

1. The output generated by the *umask* command shows the current user mask in four digits. What is the significance of the left-most digit?

2. What would the command *find /dev -type c -perm 660* do?
3. Default permissions are calculated by subtracting the initial permissions from the umask value. True or False?
4. What would be the effect of the sticky bit on an executable file?
5. Which command references the *mlocate.db* file?
6. Name the permission classes, types, and modes.
7. Default ACLs are meant for directories only. True or False?
8. The default umask for a normal user in bash shell is 0027. True or False?
9. What digit represents the setuid bit in the *chmod* command?
10. What would the command *chmod g-s file1* do?
11. Sticky bit is recommended for every system directory. True or False?
12. What would the command *setfacl -m g:dba:rw file1* do?
13. The setgid bit enables group members to run a command at a higher priority. True or False?
14. What is the equivalent symbolic value for permissions 751?
15. What permissions would the owner of the file get if the *chmod* command is executed with 555?
16. What would the *find / -name core -ok rm {} \;* command do?
17. Which special permission bit is set on a directory for team sharing?

Answers to Check Your Understanding

1. The left-most digit has no significance in the umask value.
2. The *find* command provided will find all character device files under */dev* with exact permissions of 660.
3. False. Default permissions are calculated by subtracting the umask value from the initial permission values.
4. Nothing. The sticky bit is meant for directories only.
5. The *locate* command references the *mlocate.db* database to determine the absolute paths of files and directories.

6. Permission classes are user, group, and public; permission types are read, write, and execute; and permission modes are add, revoke, and assign.
7. True. Default ACLs are meant for directories only.
8. False. The default umask for bash shell users is 0002.
9. The digit 4 represents the setuid bit.
10. It would remove the setgid bit from *file1*.
11. False.
12. This command will give the users of the *dba* group read and write permissions on *file1*.
13. False.
14. The equivalent for octal 751 is rwxr-x--x.
15. The owner will get read and execute permissions.
16. The *find* command provided will display all files by the name core in the entire directory hierarchy and ask for removal confirmation as it finds them.
17. The setgid bit is set for team sharing.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 4-1: Manipulate File Permissions

As *user1* on *server1*, create file *file11* and directory *dir11* in the home directory. Make a note of the permissions on them. Run the **umask** command to determine the current umask. Change the umask value to 0035 using symbolic notation. Create *file22* and directory *dir22* in the home directory. Observe the permissions on *file22* and *dir22,* and compare them with the permissions on *file11* and *dir11*. Use the *chmod* command and modify the permissions on *file11* to match those on *file22*. Use the *chmod* command and modify the permissions on *dir22* to match those on

dir11. Do not remove *file11*, *file22*, *dir11*, and *dir22* yet. (Hint: File and Directory Access Permissions).

Lab 4-2: Configure Group Collaboration and Prevent File Deletion

As *root* on *server1*, create directory */sdir*. Create group *sgrp* and add *user1000* and *user2000* (create the users). Set up appropriate ownership (*root*), owning group (*sgrp*), and permissions (rwx for group, --- for public, s for group, and t for public) on the directory to support group collaboration and ensure non-owners cannot delete files. Log on as *user1000* and create a file under */sdir*. Log on as *user2000* and try to edit that file. You should be able to edit the file successfully. As *user2000* try to delete the file. You should not be able to. (Hint: Special File Permissions).

Lab 4-3: Find Files

As *root* on *server1*, execute the *find* command to search for all files in the entire directory structure that have been modified in the last 300 minutes and display their type. Use the *find* command again and search for named pipe and socket files. (Hint: File Searching).

Lab 4-4: Find Files Using Different Criteria

As *root* on *server1*, issue the *find* command to search for regular files under */usr* that were accessed more than 100 days ago, are not bigger than 5MB in size, and are owned by the user *root*. (Hint: File Searching).

Lab 4-5: Apply ACL Settings

As *root* on *server1*, create file *testfile* under */tmp*. Apply ACL settings on the file so that *user2000* gets 7, *user3000* gets 6, and *user4000* gets 4 permissions. Create users. Remove the ACLs for *user2000*, and verify. Remove all remaining ACLs at once, and confirm. (Hint: Access Control Lists).

Chapter 05

Basic User Management

This chapter describes the following major topics:

- Show who is currently logged in
- Review history of successful user login attempts and system reboots
- Report history of failed user log in attempts
- View recent user login attempts
- Examine user and group information
- Understand the content and syntax of local user authentication files
- Analyze user configuration files
- Add, modify, and delete local user accounts with default and custom values
- Set and modify user passwords
- Add user account with nologin access

RHCSA Objectives:

46. Create, delete, and modify local user accounts

47. Change passwords and adjust password aging for local user accounts (only the first part of this objective “change passwords for local user accounts” is covered in this chapter; the second part is in Chapter 06)

User login activities are monitored and recorded in various files. RHEL offers a set of tools that read the activity data from these files and display the results. In addition to user activities, a history of system reboots is also maintained and it may be viewed with one of the tools. This information may be useful in debugging, testing, or auditing purposes.

In order for an authorized person to gain access to the system, a unique username must be designated and a user account must be created for them. This user is assigned a password and is allowed to change it themselves. User account information is recorded in several files. These files may be edited manually if necessary; however, this practice is discouraged. A good knowledge and grasp of the syntax and the type of information these files store is paramount for Linux administrators. User attributes may be modified later or the account may be removed from the system altogether if not required anymore.

Though service user accounts are added to the system when a corresponding service is installed, there may be situations when they need to be added manually. These accounts do not require login access; their presence is needed to support an installed application.

User Login Activity and Information

On a busy RHEL system, many users sign in and run jobs as themselves, or they switch into the *root* or another user account to run tasks that only those users have the privilege to execute. As an administrator, it is one of your responsibilities to ensure that only authorized users are able to log in to the system. You can keep track of user logins, such as who is currently logged in

and their previous and recent successful and unsuccessful login attempts. This information can be of immense help in determining any suspicious login activity. For instance, multiple failed attempts of logging in by an authorized user could be due to a forgotten or lost password, or it might be a result of an unauthorized individual trying to hack in.

There are many log files in RHEL that various services running on the system update automatically and instantly as activities occur. These logs capture user login activities among many others. This section will focus on the log files and tools that are relevant to user logins only. Others will be discussed in later chapters.

Listing Logged-In Users

A list of the users who have successfully signed on to the system with valid credentials can be printed using one of the two basic Linux tools: *who* and *w*. These commands show various pieces of information separated in multiple columns.

The *who* command references the */run/utmp* file and displays the information. Here is a sample from *server1*:

```
[root@server1 ~]# who
root     pts/0        2019-09-06 07:58 (192.168.0.219)
user1    tty2         2019-09-10 12:54 (tty2)
```

Column 1 displays the login name of the user. Column 2 shows the terminal session device filename (pts stands for *pseudo terminal session*, and tty identifies a terminal window on the console).

Columns 3 and 4 show the date and time of the user login, and column 5 indicates if the terminal session is graphical (:0), remote (IP address), or textual on the console.

The *who* command can only print information about the user who executes it with the arguments “am i”:

```
[root@server1 ~]# who am i
root     pts/0        2019-09-06 07:58 (192.168.0.219)
```

The *w* (*what*) command displays information in a similar format as the *who* command, but it also tells the length of time the user has been idle for (IDLE), along with the CPU time used by all processes including any existing background jobs attached to this terminal (JCPU), the CPU time used by the current process (PCPU), and current activity (WHAT). In the following example, line 1 displays the current system time (12:57:51), the system up duration (4 days, 5 hours, and 3 minutes), number of users currently logged in (2), and the CPU load averages over the past 1, 5, and 15 minutes (0.40, 0.63, and 0.9), respectively. This is exactly what the *uptime* command shows, which was also discussed in Chapter 02 “Initial Interaction with the System”.

```
[root@server1 ~]# w
 12:57:51 up 4 days,  5:03,  2 users,  load average: 0.40, 0.63, 0.29
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    192.168.0.219    Fri07    1.00s  1.20s  0.01s w
user1    tty2     tty2             12:54    4days 25.23s  0.35s /usr/libexec/tr
```

The load average numbers represent the percentage of CPU load with 0.00 and 1.00 correspond to no load and full load, and a number greater than 1.00 signifies excess load (over 100%).

Inspecting History of Successful Login Attempts and System Reboots

The *last* command reports the history of successful user login attempts and system reboots by consulting the *wtmp* file located in the */var/log* directory. This file keeps a record of all login and logout activities, including the login time, duration a user stayed logged in, and tty (where the user session took place). Consider the following two examples.

To list all user login, logout, and system reboot occurrences, issue the *last* command without any arguments:

```
[root@server1 ~]# last
user1    tty2         tty2             Tue Sep 10 12:54   still logged in
user1    pts/1        192.168.0.219    Fri Sep  6 19:36 - 15:21  (19:44)
root     pts/0        192.168.0.219    Fri Sep  6 07:58   still logged in
reboot   system boot  4.18.0-80.el8.x8 Fri Sep  6 07:54   still running
root     pts/0        192.168.0.219    Wed Sep  4 22:20 - 20:41  (22:21)
reboot   system boot  4.18.0-80.el8.x8 Wed Sep  4 22:20 - 20:42  (22:21)
root     pts/0        192.168.0.219    Wed Sep  4 07:31 - 16:26  (08:55)
root     pts/1        192.168.0.219    Mon Sep  2 13:20 - 13:22  (00:01)
user1    pts/3        192.168.0.219    Sat Aug 31 11:33 - 11:34  (00:00)

........

 reboot   system boot  4.18.0-80.el8.x8 Thu Aug 22 19:55 - 08:57  (13:01)
 reboot   system boot  4.18.0-80.el8.x8 Thu Aug 22 15:36 - 08:57  (17:20)

 wtmp begins Thu Aug 22 15:36:41 2019
```

The output has information that is distributed across eight to nine columns. Here is the description for each column for user history:

Column 1: Login name of the user
Column 2: Terminal name assigned upon logging in
Column 3: Terminal name or IP address from where the connection was established
Column 4 to 7: Day, month, date, and time when the connection was established
Column 8: Log out time. If the user is still logged on, it will say “still logged in”
Column 9: Duration of the login session

For system reboots, this is what it shows:

Column 1: Action name (reboot)
Column 2: Activity name (system boot)
Column 3: Linux kernel version
Column 4 to 7: Day, month, date, and time when the *reboot* command was issued
Column 8: System restart time
Column 9: Duration the system remained down. If the system is running, it will say “still running”.

The last line in the output indicates the log filename (*wtmp*) being used to record this information and the time when it started to log events.

To list system reboot details only, you can issue the last command and specify reboot as an argument:

```
[root@server1 ~]# last reboot
reboot   system boot  4.18.0-80.el8.x8 Fri Sep  6 07:54   still running
reboot   system boot  4.18.0-80.el8.x8 Wed Sep  4 22:20 - 20:42  (22:21)
reboot   system boot  4.18.0-80.el8.x8 Sat Aug 31 08:24 - 16:26 (4+08:02)
reboot   system boot  4.18.0-80.el8.x8 Fri Aug 30 20:41 - 07:48  (11:07)
reboot   system boot  4.18.0-80.el8.x8 Mon Aug 26 22:52 - 07:10 (2+08:17)
reboot   system boot  4.18.0-80.el8.x8 Fri Aug 23 14:13 - 15:21 (3+01:07)
```

The output includes the same information that it depicts with the *last* command executed without any argument.

Viewing History of Failed User Login Attempts

The *lastb* command reports the history of unsuccessful user login attempts by reading the *btmp* file located in the */var/log* directory. This file keeps a record of all unsuccessful login attempts, including the login name, time, and tty (where the attempt was made). Consider the following example.

To list all unsuccessful login attempts, type the *lastb* command without any arguments. You must be *root* in order to run this command.

```
[root@server1 ~]# lastb
root     ssh:notty    192.168.0.219    Fri Sep  6 07:58 - 07:58  (00:00)
root     ssh:notty    192.168.0.219    Mon Sep  2 13:20 - 13:20  (00:00)

btmp begins Mon Sep  2 13:20:35 2019
```

The output has information that is presented in nine columns. Here is the description for each column:

Column 1: Name of the user who made the login attempt
Column 2: Name of the protocol used. No tty was assigned as the attempt failed

Column 3: Terminal name or IP address from where the connection attempt was launched
Column 4 to 7: Day, month, date, and time of the attempt
Column 8: Duration the login attempt was tried
Column 9: Duration the login attempt lasted for

The last line in the output discloses the log filename (*btmp*) being used to record this information and the time when it started to log events.

Reporting Recent User Login Attempts

The *lastlog* command reports the most recent login evidence information for every user account that exists on the system. This information is captured in the *lastlog* file located in the */var/log* directory. This file keeps a record of the most recent user login attempts, including the login name, time, and port (or tty). Consider the following example.

```
[root@server1 ~]# lastlog
Username         Port     From             Latest
root             pts/1                     Sat Sep  7 14:29:54 -0400 2019
bin                                        **Never logged in**
daemon                                     **Never logged in**
adm                                        **Never logged in**
lp                                         **Never logged in**
sync                                       **Never logged in**
........
sshd                                       **Never logged in**
insights                                   **Never logged in**
avahi                                      **Never logged in**
tcpdump                                    **Never logged in**
user1            tty2                      Tue Sep 10 12:54:50 -0400 2019
user2            pts/0                     Fri Sep  6 19:38:35 -0400 2019
user100          pts/0                     Mon Sep  9 09:38:41 -0400 2019
user200          pts/0                     Sat Sep  7 13:28:14 -0400 2019
```

The output displays the information across four columns. Here is the description for each column:

Column 1: Login name of the user
Column 2: Terminal name assigned upon logging in
Column 3: Terminal name or IP address from where the session was initiated

Column 4: Timestamp for the latest login or “Never logged in” if the user never signed in

Note that service accounts are used by their respective services, and they are not meant for logging. More information on service accounts is discussed in the next section.

Examining User and Group Information

The *id* (*identifier*) command displays the calling user’s UID (*User IDentifier*), username, GID (*Group IDentifier*), group name, all secondary groups the user is a member of, and SELinux security context. Here is a sample output for the *root* user when this command is executed without an option or argument:

```
[root@server1 ~]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Each user and group has a corresponding number (called UID and GID) for identification purposes. These will be discussed in subsequent sections of this chapter. For SELinux, see Chapter 21 “Security Enhanced Linux”.

The *id* command can be executed by a user to view other users’ identification information. The following example shows an instance with the *root* user viewing another user’s id:

```
[root@server1 ~]# id user1
uid=1000(user1) gid=1000(user1) groups=1000(user1)
```

The *groups* command, in contrast, lists all groups the calling user is a member of:

```
[root@server1 ~]# groups
root
```

The first group listed is the primary group for the user who executed this command; all other groups are secondary (or supplementary). The *groups* command can also be used to view group membership information for a different user. Try running it as **groups user1** and observe the outcome.

Local User Authentication Files

RHEL supports three fundamental user account types: *root*, *normal*, and *service*. The *root* user (a.k.a. the *superuser* or the *administrator*), has full access to all services and administrative functions on the system. This user is created by default during installation. Normal users have user-level privileges; they cannot perform any administrative functions but can run applications and programs that have been authorized. Service accounts take care of their respective services, which include apache, ftp, mail, and chrony.

User account information for local users is stored in four files that are located in the */etc* directory. These files—*passwd, shadow*, *group*, and *gshadow*—are updated when a user or group account is created, modified, or deleted. The same files are referenced to check and validate the credentials for a user at the time of their login attempt, and hence the files are referred to as user authentication files. These files are so critical to the operation of the system that the system creates their automatic backups by default as *passwd-*, *shadow-*, *group-*, and *gshadow-* in the */etc* directory.

Here is the list of the four files and their backups from the */etc* directory:

```
[root@server1 ~]# ls -l /etc/passwd* /etc/group* /etc/shadow* /etc/gshadow*
-rw-r--r--. 1 root root 1024 Sep  7 11:57 /etc/group
-rw-r--r--. 1 root root 1016 Sep  7 11:57 /etc/group-
----------. 1 root root  821 Sep  7 11:57 /etc/gshadow
----------. 1 root root  813 Sep  7 11:57 /etc/gshadow-
-rw-r--r--. 1 root root 2615 Sep  7 11:57 /etc/passwd
-rw-r--r--. 1 root root 2570 Sep  7 11:57 /etc/passwd-
----------. 1 root root 1395 Sep  7 11:57 /etc/shadow
----------. 1 root root 1365 Sep  7 11:57 /etc/shadow-
```

All files are short in size, but they grow bigger as new users are added. Two of the files—*gshadow* and *shadow*—along with their backups have no access permissions for any user, not even for *root*. Let’s analyze these files and see what information they store and how.

The passwd File

The *passwd* file is a simple plaintext file but it contains vital user login data. Each row in the file holds information for one user account. There are seven colon-separated fields per line entry. A sample row from the file is displayed in Figure 5-1.

Figure 5-1 The passwd File

Here is a description for each field:

Field 1 (Login Name): Contains the login name for signing in. Login names up to 255 characters, including the underscore (_) and hyphen (-) characters, are supported. It is not recommended to include special characters and uppercase letters in login names. **Field 2 (Password):** Can contain an “x” (points to the */etc/shadow* file for the actual password), an asterisk * to identify a disabled account, or a hashed password.

A hashed password—a combination of random letters, numbers, and special characters—is an irreversible, unique, and scrambled string of characters to safeguard a clear text password. It is generated as a result of a conversion process of a password using one of the available hashing algorithms. By default, RHEL uses the SHA-512 algorithm for this purpose.

An algorithm is a set of well-defined but complex mathematical instructions used in data encryption and decryption techniques.

Field 3 (UID): Comprises a numeric UID between 0 and approximately 4.2 billion. UID 0 is reserved for the *root* account, UIDs between 1 and 200 are used by Red Hat to statically assign them to core service accounts, UIDs between 201 and 999 are reserved for non-core service accounts, and UIDs 1000 and beyond are employed for normal user accounts. By default, RHEL begins assigning UIDs to new users at 1000.
Field 4 (GID): Holds a GID that corresponds with a group entry in the */etc/group* file. By default, RHEL creates a group for every new user matching their login name and the same GID as their UID. The GID defined in this field represents the user's primary group.
Field 5 (Comments): Called GECOS (*General Electric Comprehensive Operating System* and later changed to GCOS), optionally stores general comments about the user that may include the user's name, phone number, location, or other useful information to help identify the person for whom the account is set up.
Field 6 (Home Directory): Defines the absolute path to the user home directory. A *home* directory is the location where a user is placed after signing in and it is used for personal storage. The default location for user home directories is */home*.
Field 7 (Shell): Consists of the absolute path of the shell file that the user will be using as their primary shell after logging in. The default shell used in RHEL is the Bash shell (*/bin/bash*). Consult Chapter 07 "The Bash Shell" for details on the Bash shell.

A *head* and *tail* from the *passwd* file for the first and last three lines is shown below:

```
[root@server1 ~]# head -3 /etc/passwd ; tail -3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
user1:x:1000:1000:user1:/home/user1:/bin/bash
user100:x:1002:1002::/home/user100:/bin/bash
user200:x:1003:1003::/home/user200:/bin/bash
```

The output indicates the *root* user with UID 0 followed by two service accounts (*bin* and *daemon*). The last three lines display the three user accounts that were created earlier as part of some of the exercises.

Let's verify the permissions and ownership on the *passwd* file:

```
[root@server1 ~]# ls -l /etc/passwd
-rw-r--r--. 1 root root 2574 Sep 11 11:49 /etc/passwd
```

The access permissions on the file are 644 (world-readable and owner-writable), and it is owned by the *root* user.

The shadow File

RHEL has a secure password control mechanism in place that provides an advanced level of password security for local users. This control is referred to as the *shadow password*. With this control mechanism in place, user passwords are hashed and stored in a more secure file */etc/shadow*, but there are certain limits on user passwords in terms of expiration, warning period, etc., that can also be applied on a per-user basis. These limits and other settings are defined in the */etc/login.defs*file, which the shadow password mechanism enforces on user accounts. This is called *password aging*. Unlike the *passwd* file, which is world-readable and owner-writable, the *shadow* file has no access permissions at all. This is done to safeguard the file's content.

With the shadow password mechanism active, a user is initially checked in the *passwd* file for existence and then in the *shadow* file for authenticity.

The *shadow* file contains user authentication and password aging information. Each row in the file corresponds to one entry in the *passwd* file. There are nine colon-separated fields per line entry. A sample row from this file is showcased in Figure 5-2.

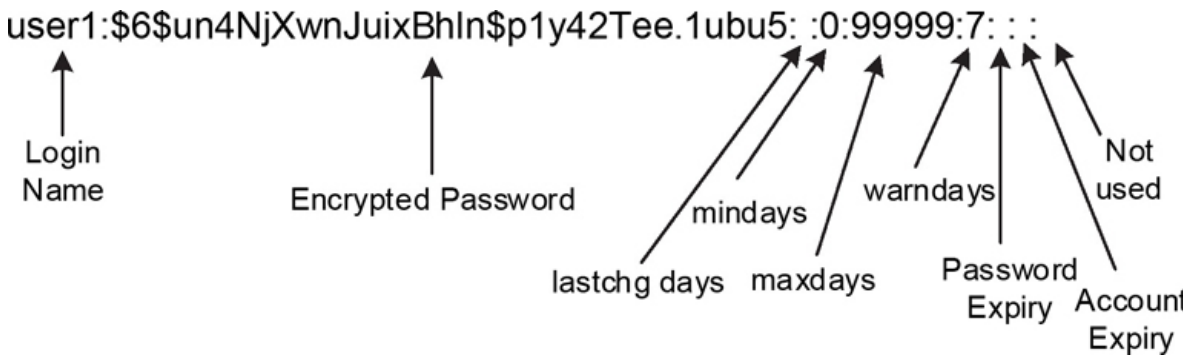


Figure 5-2 The shadow File

Here is a description for each field:

Field 1 (Login Name): Contains the login name as appears in the *passwd* file.
Field 2 (Encrypted Password): Consists of a hashed password. A single exclamation mark (!) at the beginning of this field implies that the user account is locked. If this field is empty, the user will have password-less entry into the system.
Field 3 (Last Change): Sets the number of days (lastchg) since the UNIX epoch, a.k.a. UNIX time (January 01, 1970 00:00:00 UTC) when the password was last modified. An empty field represents the passiveness of password aging features, and a 0 forces the user to change their password upon next login.
Field 4 (Minimum): Expresses the minimum number of days (mindays) that must elapse before the user is allowed to change their password. This field can be altered using the *chage* command with the -m option or the *passwd* command with the -n option. A 0 or null in this field disables this feature.
Field 5 (Maximum): Defines the maximum number of days (maxdays) of password validity before the user password expires and it must be changed. This field may be altered using the *chage* command with the -M option or the *passwd* command with the -x option. A null value here disables this

feature along with other features such as the maximum password age, warning alerts, and the user inactivity period.
Field 6 (Warning): Denotes the number of days (warndays) for which the user gets warnings for changing their password before it actually expires. This field may be altered using the *chage* command with the -W option or the *passwd* command with the -w option. A 0 or null in this field disables this feature.
Field 7 (Password Expiry): Contains the maximum allowable number of days for the user to be able to log in with the expired password. This period is referred to as the inactivity period. This field may be altered using the *chage* command with the -I option or the *passwd* command with the -i option. An empty field disables this feature.
Field 8 (Account Expiry): Determines the number of days since the UNIX time when the user account will expire and no longer be available. This field may be altered using the *chage* command with the -E option. An empty field disables this feature.
Field 9 (Reserved): Reserved for future use.

A *head* and *tail* from the *shadow* file for the first and last three lines is shown below:

```
[root@server1 ~]# head -3 /etc/shadow ; tail -3 /etc/shadow
root:$6$wBS453YXp85IO/Gf$xjPHvemxqXwFgAc3nMyb4tMKB4FQscvHWKVd5boUwaqz3rVoPOZJHBN
PHxAO8DBuz2H8OsYFKqhhsF2XoSh.PO::0:99999:7:::
bin:*:17784:0:99999:7:::
daemon:*:17784:0:99999:7:::
user1:$6$qlSIof2tOZDXzTaH$p59YmX12xI8QozHoWWdKNFZxyt..2YMVXCPObTf7Ok5ufU.9qxTXAD
EfR1PcXcMVDlz9GttbS.peyUreCWXOb1::0:99999:7:::
user100:!!:18146:0:99999:7:::
user200:!!:18146:0:99999:7:::
```

The output indicates the *root* user with UID 0 followed by two service accounts (*bin* and *daemon*). The last three lines display the three user accounts that were created earlier as part of some of the exercises. Notice that login names are used as a common key between the *shadow* and *passwd* files.

Let’s verify the permissions and ownership on the *shadow* file:

```
[root@server1 ~]# ls -l /etc/shadow
----------. 1 root root 1367 Sep 11 11:49 /etc/shadow
```

The access permissions on the file are 000 (no permissions at all), and it is owned by the *root* user. There is a special mechanism in place that is employed in the background to update this file when a user account is added, modified, deleted, or the password changed. This will be discussed in Chapter 21 “Security Enhanced Linux”.

The group File

The *group* file is a simple plaintext file and contains critical group information. Each row in the file stores information for one group entry. Every user on the system must be a member of at least one group, which is referred to as the *User Private Group* (UPG). By default, a group name matches the username it is associated with. Additional groups may be set up, and users with common file access requirements can be added to them. There are four colon-separated fields per line entry. A sample row from the file is exhibited in Figure 5-3.

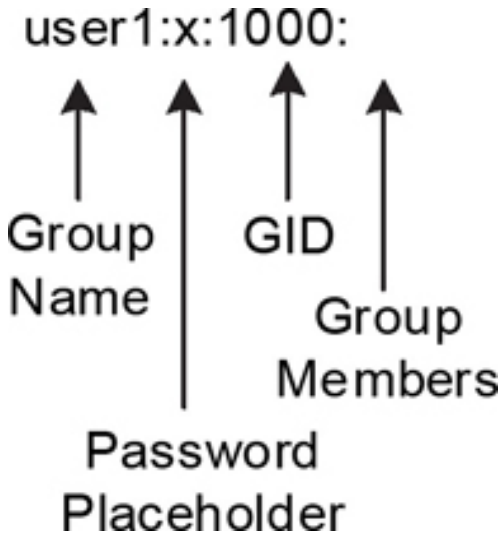


Figure 5-3 The group File

Here is a description for each field:

Field 1 (Group Name): Holds a group name that must begin with a letter. Group names up to 255 characters, including the underscore (_) and hyphen (-) characters, are supported. It is not recommended to include special characters and uppercase letters in group names.
Field 2 (Encrypted Password): Can be empty or contain an “x” (points to the */etc/gshadow* file for the actual password), or a hashed group-level password. You can set a password on a group if you want non-members to be able to change their group identity temporarily using the *newgrp* command. The non-members must enter the correct password in order to do so.
Field 3 (GID): Holds a GID, which is also placed in the GID field of the *passwd* file. By default, groups are created with GIDs starting at 1000 and with the same name as the username. The system allows several users to belong to a single group; it also allows a single user to be a member of multiple groups at the same time.
Field 4 (Group Members): Lists the membership for the group. Note that a user’s primary group is always defined in the GID field of the *passwd* file.

A *head* and *tail* from the *group* file for the first and last three lines is shown below:

```
[root@server1 ~]# head -3 /etc/group ; tail -3 /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sgrp:x:9999:user100,user200
user100:x:1002:
user200:x:1003:
```

The output discloses the *root* user with GID 0 followed by two group service accounts (*bin* and *daemon*). The last three lines showcase the three groups that were created earlier as part of some of the exercises.

Let’s verify the permissions and ownership on the *group* file:

```
[root@server1 ~]# ls -l /etc/group
-rw-r--r--. 1 root root 1010 Sep 12 07:48 /etc/group
```

The access permissions on the file are 644 (world-readable and owner-writable), and it is owned by the *root* user.

The gshadow File

The shadow password implementation also provides an added layer of protection at the group level. With this mechanism in place, the group passwords are hashed and stored in a more secure file */etc/gshadow*. Unlike the *group* file, which is world-readable and owner-writable, the *gshadow* file has no access permissions at all. This is done to safeguard the file's content.

The *gshadow* file stores hashed group-level passwords. Each row in the file corresponds to one entry in the *group* file. There are four colon-separated fields per line entry. A sample row from this file is exhibited in Figure 5-4.

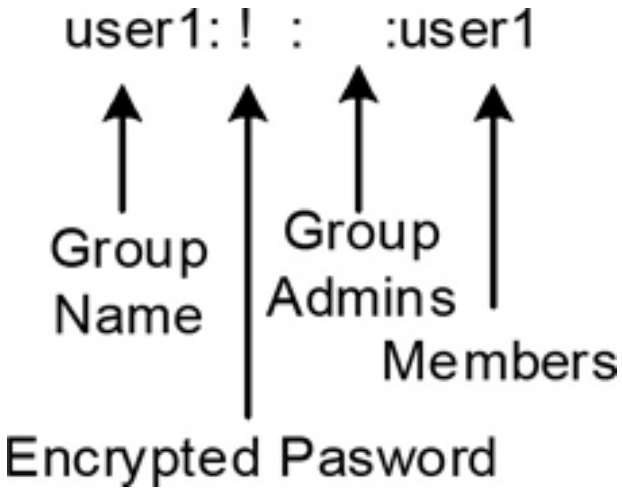


Figure 5-4 The gshadow File

Here is a description for each field:

Field 1 (Group Name): Consists of a group name as appeared in the *group* file.
Field 2 (Encrypted Password): Can contain a hashed password, which may be set with the *gpasswd* command for non-group members to access the group temporarily using the *newgrp* command. A single exclamation mark (!) or a null

value in this field allows group members password-less access and restricts non-members from switching into this group.
Field 3 (Group Administrators): Lists usernames of group administrators that are authorized to add or remove members with the *gpasswd* command.
Field 4 (Members): Holds a comma-separated list of members.

The gpasswd command is used to add group administrators, add or delete group members, assign or revoke a group-level password, and disable the ability of the newgrp command to access a group. This command picks up the default values from the /etc/login.defs file. Additional discussion on this command is beyond the scope; however, you can view the manual pages of this command for details on its usage.

A *head* and *tail* from the *gshadow* file for the first and last three lines is shown below:

```
[root@server1 ~]# head -3 /etc/gshadow ; tail -3 /etc/gshadow
root:::
bin:::
daemon:::
sgrp:!::user100,user200
user100:!::
user200:!::
```

The output indicates the *root* user with GID 0 followed by two group service accounts (*bin* and *daemon*). The last three lines exhibit the three groups that were created earlier as part of some of the exercises. Notice that group names are used as a common key between the *gshadow* and *group* files.

Let’s verify the permissions and ownership on the *gshadow* file:

```
[root@server1 ~]# ls -l /etc/gshadow
----------. 1 root root 811 Sep 12 10:54 /etc/gshadow
```

The access permissions on the file are 000 (no permissions at all) and it is owned by the *root* user. There is a special mechanism in place that is employed in the background to update this file when a group account is added, modified,

deleted, or the group password changed. This will be discussed further in Chapter 21 “Security Enhanced Linux”.

The useradd and login.defs Configuration Files

The *useradd* command (discussed in the next section) picks up the default values from the */etc/default/useradd* and */etc/login.defs* files for any options that are not specified at the command line when executing it. Moreover, the *login.defs* file is also consulted by the *usermod*, *userdel*, *chage*, and *passwd* commands (also discussed in the next section) as needed. Both files store several defaults including those that affect the password length and password lifecycle. You can use the *cat* or *less* command to view the *useradd* file content or display the settings with the *useradd* command as follows:

```
[root@server1 ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

There are a multitude of defaults defined with the directives. These include the starting GID (GROUP) provided the USERGROUPS_ENAB directive in the *login.defs* file is set to no, home directory location (HOME), number of inactivity days between password expiry and permanent account disablement (INACTIVE), account expiry date (EXPIRE), login shell (SHELL), skeleton directory location to copy user initialization files from (SKEL), and whether to create mail spool directory (CREATE_MAIL_SPOOL). You will find a description for some of these in the next section.

The other file *login.defs* comprises of additional directives that set several defaults. User and group management commands consult this file to obtain information that is not supplied at the

command line. A *grep* on the file with uncommented and non-empty lines is shown below:

```
[root@server1 ~]# grep -v ^# /etc/login.defs | grep -v ^$
MAIL_DIR        /var/spool/mail
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               201
SYS_UID_MAX               999
GID_MIN                  1000
GID_MAX                 60000
SYS_GID_MIN               201
SYS_GID_MAX               999
CREATE_HOME     yes
UMASK           077
USERGROUPS_ENAB yes
ENCRYPT_METHOD SHA512
```

These directives are elaborated in Table 5-1.

Option	Description
MAIL_DIR	Specifies the mail directory location
PASS_MAX_DAYS, PASS_MIN_DAYS, PASS_MIN_LEN, and PASS_WARN_AGE	Define password aging attributes. See Chapter 06 “Advanced User Management” for details.
UID_MIN, UID_MAX, GID_MIN, and GID_MAX	Identify the ranges of UIDs and GIDs to be allocated to new users and groups
SYS_UID_MIN, SYS_UID_MAX, SYS_GID_MIN, and SYS_GID_MAX	Identify the ranges of UIDs and GIDs to be allocated to new service users and groups
CREATE_HOME	Defines whether to create a home directory
UMASK	Sets a umask value
USERGROUPS_ENAB	Defines whether to delete a user’s group (at the time of user deletion) if it contains no more members
ENCRYPT_METHOD	Specifies the encryption method for user passwords

Table 5-1 login.defs File Directives

There are many more directives available that can be added to this file for more control and flexibility. Check the manual pages of the file for details.

User Account Management

Managing user accounts involves creating, assigning passwords to, modifying, and deleting them. RHEL provides a set of tools for performing these operations. These tools are *useradd* to add a new user to the system, *usermod* to modify the attributes of an existing user, and *userdel* to remove a user from the system. In addition, the *passwd* command is available to set or modify a user's password.

The useradd, usermod, and userdel Commands

This set of commands is used to add, modify, and delete a user account from the system. The *useradd* command adds entries to the four user authentication files for each account added to the system. It creates a home directory for the user and copies the default user startup files from the *skeleton* directory */etc/skel* into the user's home directory. It can also be used to update the default settings that are used at the time of new user creation for unspecified settings. The *useradd* command supports a variety of flags; Table 5-2 lists some common options in both short and long versions.

Option	Description
-b (--base-dir)	Defines the absolute path to the base directory for placing user home directories. The default is /home.
-c (--comment)	Describes useful information about the user.
-d (--home-dir)	Defines the absolute path to the user home directory.
-D (--defaults)	Displays the default settings from the /etc/default/useradd file and modifies them.
-e (--expiredate)	Specifies a date on which a user account is automatically disabled. The format for the date specification is YYYY-MM-DD.
-f (--inactive)	Denotes maximum days of inactivity between password expiry and permanent account disablement.
-g (--gid)	Specifies the primary GID. Without this option, a group account matching the username is created with the GID matching the UID.
-G (--groups)	Specifies the membership to supplementary groups.
-k (--skel)	Specifies the location of the skeleton directory (default is /etc/skel), which stores default user startup files. These files are copied to the user's home directory at the time of account creation. Three hidden bash shell files—.bash_profile, .bashrc, and .bash_logout—are available in this directory by default. You can customize these files or add your own to be used for accounts created thereafter.
-m (--create-home)	Creates a home directory if it does not already exist.
-o (--non-unique)	Creates a user account sharing the UID of an existing user. When two users share a UID, both get identical rights on each other's files. This should only be done in specific situations.
-r (--system)	Creates a service account with a UID below 1000 and a never-expiring password.
-s (--shell)	Defines the absolute path to the shell file. The default is /bin/bash.
-u (--uid)	Indicates a unique UID. Without this option, the next available UID from the /etc/passwd file is used.
login	Specifies a login name to be assigned to the user account.

Table 5-2 useradd Command Options

You can modify the attributes of a user account with the *usermod* command. The syntax of this command is very similar to that of the *useradd*'s, with most switches identical. Table 5-3 describes the options that are specific to *usermod* only, and shows them in both short and long versions. There are two additional flags of interest that are discussed in Chapter 06 "Advanced User Management".

Option	Description
-a (--append)	Adds a user to one or more supplementary groups
-l (--login)	Specifies a new login name
-m (--move-home)	Creates a home directory and moves the content over from the old location

Table 5-3 usermod Command Options

The *userdel* command is straightforward. It removes entries for the specified user from the authentication files, and deletes the user's home directory if the -r flag is also specified. The -f option may be used to force the removal even if the user is still logged in.

Exercise 5-1: Create a User Account with Default Attributes

This exercise should be done on *server1* as *root*.

In this exercise, you will create a user account *user2* using the defaults defined in the *useradd* and *login.defs* files. You will assign this user a password and show the new line entries from all four authentication files.

1. Create *user2* with all the default values:

```
[root@server1 ~]# useradd user2
```

2. Assign this user a password and enter it twice when prompted:

```
[root@server1 ~]# passwd user2
Changing password for user user2.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```

3. *grep* for *user2:* on the authentication files to examine what the *useradd* command has added:

```
[root@server1 ~]# cd /etc ; grep user2: passwd shadow group gshadow
passwd:user2:x:1004:1004::/home/user2:/bin/bash
shadow:user2:$6$HjGW61CCZRJWifCP$HTJuDdHrpM79.KAbD7bM3DVVCV.Ij9sc6Sgv3MNHqk/azmu
eXmiPEMQYDANGEaO6J6FbhvIiNShfgOHff9q9GO:18152:0:99999:7:::
group:user2:x:1004:
gshadow:user2:!::
```

The command used the next available UID (1004) and GID (1004), and the default settings for the home directory (*/home/user2*) and shell file (*/bin/bash*).

4. Test this new account by logging in as *user2* and then run the *id* and *groups* commands to verify the UID, GID, and group membership information:

```
[root@server1 etc]# su - user2
[user2@server1 ~]$ id
uid=1004(user2) gid=1004(user2) groups=1004(user2) context=unconfined_u:unconfin
ed_r:unconfined_t:s0-s0:c0.c1023
[user2@server1 ~]$ groups
user2
```

The above outputs confirmed the UID and username, GID and group name, and primary group name.

Exercise 5-2: Create a User Account with Custom Values

This exercise should be done on *server1* as *root*.

In this exercise, you will create an account *user3* with UID 1010, home directory */usr/user3a*, and shell */bin/sh*. You will assign this user a password and exhibit the new line entries from all four authentication files.

1. Create *user3* with UID 1010 (-u), home directory */usr/user3a* (-d), and shell */bin/sh* (-s):

```
[root@server1 ~]# useradd -u 1010 -d /usr/user3a -s /bin/sh user3
```

2. Assign user1234 as password (passwords assigned in the following way is not recommended; however, it is okay in a lab environment):

```
[root@server1 ~]# echo user1234 | passwd --stdin user3
Changing password for user user3.
passwd: all authentication tokens updated successfully.
```

3. *grep* for *user3:* on the four authentication files to see what was added for this user:

```
[root@server1 ~]# cd /etc ; grep user3: passwd shadow group gshadow
passwd:user3:x:1010:1010::/usr/user3a:/bin/sh
shadow:user3:$6$R3jWTWYymYIeF55h$r6qkL8Tk3vhW8jw1BdvOFuJRJGRhFUT2P5pUpdIFkyR4OWf
cNATNv9RcjYKyuexB1GWYcA/Ivf5uiwHQ/ujPP/:18152:0:99999:7:::
group:user3:x:1010:
gshadow:user3:!::
```

4. Test this account by switching to or logging in as *user3* and entering user1234 as the password. Run the *id* and *groups* commands for further verification.

Exercise 5-3: Modify and Delete a User Account

This exercise should be done on *server1* as *root*.

In this exercise, you will modify certain attributes for *user2* and then delete it. You will change the login name to *user2new*, UID to 2000, home directory to */home/user2new*, and login shell to */sbin/nologin*. You will display the line entry for *user2new* from the *passwd* file for validation. Finally, you will remove this user and confirm the deletion.

1. Modify the login name for *user2* to *user2new* (-l), UID to 2000 (-u), home directory to */home/user2new* (-m and -d), and login shell to */sbin/nologin* (-s). Notice that the options are specified in a different sequence.

```
[root@server1 ~]# usermod -l user2new -m -d /home/user2new -s /sbin/nologin -u 2
000 user2
```

2. Obtain the information for *user2new* from the *passwd* file for confirmation:

```
[root@server1 ~]# grep user2new /etc/passwd
user2new:x:2000:1004::/home/user2new:/sbin/nologin
```

3. Remove *user2new* along with their home and mail spool directories (-r):

```
[root@server1 ~]# userdel -r user2new
```

4. Confirm the user deletion:

```
[root@server1 ~]# grep user2new /etc/passwd
```

Nothing is returned in the output, which means this user does not exist in the *passwd* file any longer. This confirms the removal.

No-Login (Non-Interactive) User Account

The *nologin* shell is a special purpose program that can be employed for user accounts that do not require login access to the system. It is located in the */usr/sbin* (or */sbin*) directory. With this shell assigned, the user is refused with the message, “This account is currently not available.” displayed on the screen. If a custom message is required, you can create a file called *nologin.txt* in the */etc* directory and add the desired text to it. The content of this file is printed on the screen upon user access denial instead of the default message.

Typical examples of user accounts that do not require login access are the service accounts such as *ftp*, *apache*, and *sshd*. Let’s take a look at the *passwd* file and list such users:

```
[root@server1 ~]# grep nologin /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
```

This output returns a truncated list of non-interactive user accounts. There are tens of them, and they grow as you install more services to the system.

Exercise 5-4: Create a User Account with No-Login Access This exercise should be done on *server1* as *root*.

In this exercise, you will create an account *user4* with all the default attributes but with a non-interactive shell. You will assign this user the nologin shell to prevent them from signing in. You will display the new line entry from the *passwd* file and test the account.

1. Create *user4* with non-interactive shell file */sbin/nologin*:

```
[root@server1 ~]# useradd -s /sbin/nologin user4
```

2. Assign user1234 as password:

```
[root@server1 ~]# echo user1234 | passwd --stdin user4
```

3. *grep* for *user4* on the *passwd* file and verify the shell field containing the nologin shell:

```
[root@server1 ~]# grep user4 /etc/passwd
user4:x:1011:1011::/home/user4:/sbin/nologin
```

4. Test this account by attempting to log in or switch:

```
[root@server1 ~]# su - user4
This account is currently not available.
```

As mentioned earlier, you may create the */etc/nologin.txt* file and add a custom message to it. The new message will appear on your next login attempt.

Chapter Summary

We started this chapter by discovering various tools that are employed to monitor user login and system reboot activities. This information is stored in several system files and may be

used for testing, auditing, or troubleshooting reasons. Most of these tools can be executed by normal users to obtain desired results; however, others may require privileged access of the root user in order to be viewed.

Next, we looked closely at the four local user authentication files: *passwd, shadow*, *group*, and *gshadow*. We examined their contents and syntax to comprehend what they store and how. These files are critical to the user login process as well as the functioning of applications and services.

Finally, we explored user management tools and put them into action in exercises to create, modify, and delete user accounts. These tools offer a number of switches to add accounts with default and custom attributes, as well as user accounts that do not require login access.

Check Your Understanding

1. What would the command *useradd -D* do?
2. What does the *lastlog* command do?
3. What would the command *useradd user500* do?
4. The *id* and *groups* commands are useful for listing a user identification. True or False.
5. Name the four local user authentication files.
6. The *passwd* file contains secondary user group information. True or False?
7. What is the first UID assigned to a normal user?
8. Name the three fundamental user account classes in RHEL.
9. Every user in RHEL gets a private group by default. True or False?
10. What does the “x” in the password field in the *passwd* file imply?
11. Name the file that the *who* command consults to display logged-in users.
12. What information does the *lastb* command provide?

13. The *who* command may be used to view logged out users. True or False?
14. What would the *userdel* command do if it is run with the -r option?
15. What is the first GID assigned to a group?
16. What is the name of the default backup file for *shadow*?
17. Which file does the *last* command consult to display reports?
18. UID 999 is reserved for normal users. True or False?
19. Which file is assigned to users to deny them login access?
20. You execute the *lastb* command as a normal user. The output reports an error. What do you need to do to run this command successfully?

Answers to Check Your Understanding

1. This command displays the defaults settings used at the time of user creation or modification.
2. The *lastlog* command provides information about recent user logins.
3. The command provided will add *user500* with all predefined default values.
4. False. Only the *id* command is used for this purpose.
5. The *passwd*, *shadow*, *group*, and *gshadow* files.
6. False. The *passwd* file contains primary user group information.
7. The first UID assigned to a normal user is 1000.
8. The three fundamental user account categories are root, normal, and system.
9. True.
10. The “x” in the password field implies that the hashed password is stored in the *shadow* file.
11. The *who* command consults the */run/utmp* file to list logged-in users.

12. The *lastb* command reports the history of unsuccessful user login attempts.
13. False. The *who* command shows currently logged-in users only.
14. The *userdel* command with the -r switch will delete the specified user along with their home directory.
15. The first GID assigned to a normal user is 1000.
16. The name of the default backup file for *shadow* is *shadow-*.
17. The *last* command consults the */var/log/wtmp* file to display reports.
18. False. UID 999 is reserved for system accounts.
19. The */sbin/nologin* file is assigned to users to deny them login.
20. You need to run the *lastb* command as *root* or with the *root* privilege.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 5-1: Check User Login Attempts

As *root* on *server1*, execute the *last*, *lastb*, and *lastlog* commands, and observe the outputs. Check which users have recently logged in and out of the system successfully (*last*) and unsuccessfully (*lastb*). List the timestamps when the system was last rebooted (*last*). Check the last login status for each user (*lastlog*). Use the *vim* editor to record your results. (Hint: User Login Activity and Information).

Lab 5-2: Verify User and Group Identity

As *user1* on *server1*, run the **who** and **w** commands one at a time, and compare the outputs. Execute the **id** and **groups** commands, and compare the outcomes. Examine the extra information that the *id* command shows, but not the *groups* command. (Hint: User Login Activity and Information).

Lab 5-3: Create Users

As *root* on *server1*, create user account *user4100* with UID 4100 and home directory under */usr*. Create another user account *user4200* with default attributes. Assign both users a password. View the contents of the *passwd*, *shadow*, *group*, and *gshadow* files, and observe what has been added for the two new users. (Hint: Local User Authentication Files, and User Account Management).

Lab 5-4: Create User with Non-Interactive Shell

As *root* on *server1*, create user account *user4300* with the disability of logging in. Assign this user a password. Try to log on with this user and see is displayed on the screen. View the content of the *passwd* file, and see what is there that prevents this user from logging in. (Hint: Local User Authentication Files, and User Account Management).

Chapter 06

Advanced User Management

This chapter describes the following major topics:

- Configure password aging attributes on local user accounts
- Lock and unlock user account
- Understand, create, modify, and delete local groups and group memberships
- Switch into another user account
- Configure who can execute which privileged commands
- Identify and manage file owners and owning groups

RHCSA Objectives:

05. Log in and switch users in multi-user targets
47. Change passwords and adjust password aging for local user accounts (only the

second part of this objective “adjust password aging for local user accounts” is covered in this chapter; the first part is in Chapter 05)

48. Create, delete, and modify local groups and group memberships
49. Configure superuser access

Password aging attributes may be set on user accounts for increased control on their logins and passwords. This can be done for an individual user or applied to all users. Password aging information for users is stored in one of the authentication files that was discussed at length in the previous chapter. Individual user accounts may be prevented from logging in to the system by locking their access for a period of time or permanently. This lock may be lifted when required. Setting password aging and locking/unlocking accounts are administrative functions and must be performed by a user with elevated privileges of the root user.

Users are apportioned membership to a single group at the time of their addition to the system. Later, they may be assigned membership to additional groups. Members of the same group possess the same access rights on files and directories. Other users and members of other groups may optionally be given access to those files. Group membership information is stored in user and group authentication files that were examined in the previous chapter as well.

Users may switch into other user accounts, including the root user, provided they know the target user’s password. Normal users may be allowed access to privileged commands by defining them appropriately in a configuration file. Each file that exists on the system regardless of its type has an owning user and an owning group. Similarly, every file that a user creates is in the ownership of that user. The ownership may be changed and given to another user by a super user.

Password Aging and its Management

As mentioned, password aging is a secure mechanism to control user passwords in Linux. The key advantages include setting restrictions on password expiry, account disablement, locking and unlocking users, and password change frequency. These controls are applied to all user accounts at the time of their creation and can be set explicitly on a per-user basis later. You can even choose to inactivate it completely for an individual user.

Password aging information is stored in the */etc/shadow* file (fields 4 to 8) and its default policies in the */etc/login.defs* configuration file. These files were thoroughly examined in Chapter 05 “Basic User Management”. In this section, we explore the aging management tools—*chage* and *passwd*—and look at how to employ them to apply password controls on user accounts, *user100* and *user200*. Alongside *chage* and *passwd*, the *usermod* command can also be used to implement two aging attributes (user expiry and password expiry); however, this section focuses on this command’s ability to lock and unlock user accounts.

The chage Command

The *chage* command is used to set or alter password aging parameters on a user account. This command changes various fields in the *shadow* file depending on which option(s) you pass to it. There are plenty of switches available with the command in both short and long formats. Table 6-1 describes most of them.

Option	Description
-d (--lastday)	Specifies an explicit date in the YYYY-MM-DD format, or the number of days since the UNIX time when the password was last modified. With -d 0, the user is forced to change the password at next login. It corresponds to field 3 in the shadow file.
-E (--expiredate)	Sets an explicit date in the YYYY-MM-DD format, or the number of days since the UNIX time on which the user account is deactivated. This feature can be disabled with -E -1. It corresponds to the eighth field in the shadow file.
-I (--inactive)	Defines the number of days of inactivity after the password expiry and before the account is locked. The user may be able to log in during this period with their expired password. This feature can be disabled with -I -1. It corresponds to field 7 in the shadow file.
-l	Lists password aging attributes set on a user account.
-m (--mindays)	Indicates the minimum number of days that must elapse before the password can be changed. A value of 0 in this field allows the user to change their password at any time. It corresponds to field 4 in the shadow file.
-M (--maxdays)	Denotes the maximum number of days of password validity before the user password expires and it must be changed. This feature can be disabled with -M -1. It corresponds to field 5 in the shadow file.
-W (--warndays)	Designates the number of days for which the user gets alerts to change their password before it actually expires. It corresponds to field 6 in the shadow file.

Table 6-1 chage Command Options

You will use most of these flags in Exercise 6-1 as well as later in this chapter.

Exercise 6-1: Set and Confirm Password Aging with chage

This exercise should be done on *server1* as *root*.

In this exercise, you will configure password aging for *user100* using the *chage* command. You will set mindays to 7, maxdays to 28, and warndays to 5, and verify the new settings. You will then rerun the command and set account expiry to January 31, 2020. You will complete the exercise with another confirmation.

1. Set password aging parameters for *user100* to mindays (-m) 7, maxdays (-M) 28, and warndays (-W) 5:

```
[root@server1 ~]# chage -m 7 -M 28 -W 5 user100
```

2. Confirm the new settings:

```
[root@server1 ~]# chage -l user100
Last password change                                    : Sep 13, 2019
Password expires                                        : Oct 11, 2019
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 7
Maximum number of days between password change          : 28
Number of days of warning before password expires       : 5
```

The bottom three rows in the output confirm the new settings. The current password expiry (October 11, 2019) reflects the 28-day duration from September 13, 2019.

3. Set the account expiry to January 31, 2020:

```
[root@server1 ~]# chage -E 2020-01-31 user100
```

4. Verify the new account expiry setting:

```
[root@server1 ~]# chage -l user100
Last password change                                    : Sep 13, 2019
Password expires                                        : Oct 11, 2019
Password inactive                                       : never
Account expires                                         : Jan 31, 2020
Minimum number of days between password change          : 7
Maximum number of days between password change          : 28
Number of days of warning before password expires       : 5
```

The middle row reflects the new account expiry for *user100*. Similarly, you can use the -d and -I options with the *chage* command as required.

The passwd Command

The common use of the *passwd* command is to set or modify a user's password; however, you can also use this command to modify the password aging attributes and lock or unlock their account. Table 6-2 lists some key options in both short and long formats.

Option	Description
-d (--delete)	Deletes a user password without expiring the user account.
-e (--expire)	Forces a user to change their password upon next logon.
-i (--inactive)	Defines the number of days of inactivity after the password expiry and before the account is locked. It corresponds to field 7 in the shadow file.
-l (--lock)	Locks a user account.
-n (--minimum)	Specifies the number of days that must elapse before the password can be changed. It corresponds to field 4 in the shadow file.
-S (--status)	Displays the status information for a user.
-u (--unlock)	Unlocks a locked user account.
-w (--warning)	Designates the number of days for which the user gets alerts to change their password before it actually expires. It corresponds to field 6 in the shadow file.
-x (--maximum)	Denotes the maximum number of days of password validity before the user password expires and it must be changed. It corresponds to field 5 in the shadow file.

Table 6-2 passwd Command Options

Most of these flags will be used in the next few exercises.

Exercise 6-2: Set and Confirm Password Aging with passwd

This exercise should be done on *server1* as *root*.

In this exercise, you will configure password aging for *user200* using the *passwd* command. You will set mindays to 10, maxdays to 90, and warndays to 14, and verify the new settings.

Next, you will set the number of inactivity days to 5 and ensure that the user is forced to change their password upon next login. You will complete the exercise with another verification.

1. Set password aging attributes for *user200* to mindays (-n) 10, maxdays (-x) 90, and warndays (-w) 14:

```
[root@server1 ~]# passwd -n 10 -x 90 -w 14 user200
```

2. Confirm the new settings:

```
[root@server1 ~]# passwd -S user200
user200 PS 2019-09-14 10 90 14 -1 (Password set, SHA512 crypt.)
```

The output confirms the three new settings (10, 90, and 14).

3. Set the number of inactivity days to 5:

```
[root@server1 ~]# passwd -i5 user200
```

4. Confirm the new setting:

```
[root@server1 ~]# passwd -S user200
user200 PS 2019-09-14 10 90 14 5 (Password set, SHA512 crypt.)
```

The output verifies the new setting (5).

5. Ensure that the user is forced to change their password at next login:

```
[root@server1 ~]# passwd -e user200
```

6. Display the new setting for confirmation:

```
[root@server1 ~]# passwd -S user200
user200 PS 1969-12-31 10 90 14 5 (Password set, SHA512 crypt.)
```

The output shows a date prior to the UNIX time. This user will be prompted to change their password when they attempt to log in the next time.

The usermod Command

The common use of the *usermod* command is to modify a user's attribute, but it can also lock or unlock their account. Table 6-3 explains the two options in both short and long formats.

Option	Description
-L (--lock)	Locks a user account by placing a single exclamation mark (!) at the beginning of the password field and before the hashed password string.
-U (--unlock)	Unlocks a user's account by removing the exclamation mark (!) from the beginning of the password field.

Table 6-3 usermod Command Options for User Lock/Unlock

Let's apply these options in the next exercise.

Exercise 6-3: Lock and Unlock a User Account with usermod and passwd

This exercise should be done on *server1* as *root*.

In this exercise, you will disable the ability of *user200* to log in using the *usermod* and *passwd* commands. You will verify the change and then reverse it.

1. Obtain the current password information for *user200* from the *shadow* file:

```
[root@server1 ~]# grep user200 /etc/shadow
user200:$6$gnvX93VTOeWU94s.$6SPEdtB31glIULSxbbEAKPMw8H8PXnyston.3n1F/kc15Aw8Kt2e
To1IHiOoouq/OrNBSa9Bow1rssW3ZzVmp1:0:10:90:14:5::
```

An unlocked user account never has its password field begin with an exclamation mark (!). The above output is indicative of the fact that the account is currently not locked.

2. Lock the account for *user200*:

```
[root@server1 ~]# usermod -L user200
```
or
```
[root@server1 ~]# passwd -l user200
```

3. Confirm the change:

```
[root@server1 ~]# grep user200 /etc/shadow
user200:!$6$gnvX93VTOeWU94s.$6SPEdtB31glIULSxbbEAKPMw8H8PXnyston.3n1F/kc15Aw8Kt2
eTo1IHiOoouq/OrNBSa9Bow1rssW3ZzVmp1:0:10:90:14:5::
```

Notice that an exclamation mark (!) is prepended to the encrypted password, which indicates a locked account.

4. Unlock the account with either of the following:

```
[root@server1 ~]# usermod -U user200
```
or
```
[root@server1 ~]# passwd -u user200
```

Verify the reversal with the *grep* command as demonstrated in step 3. You can also try to log in as *user200* for an additional validation.

Linux Groups and their Management

Linux groups are collections of one or more users with identical permission requirements on files and directories. They allow group members to collaborate on files of common interest.

Group information is stored in the */etc/group* file and the default policies in the */etc/login.defs*configuration file. Furthermore, the */etc/gshadow* file stores group administrator information and group-level passwords. These files were

thoroughly examined in Chapter 05 “Basic User Management”.

This section explores group management tools—*groupadd*, *groupmod*, and *groupdel*—and looks at how to utilize them to create, alter, and remove groups. Additional group administration operations, such as adding and deleting group administrators, and setting and revoking group-level passwords, are beyond the scope.

The groupadd, groupmod, and groupdel Commands

This set of management commands is used to add, modify, and delete a group from the system. The *groupadd* command adds entries to the *group* and *gshadow* files for each group added to the system. Table 6-4 lists and explains some of the *groupadd* command options in both short and long formats.

Option	Description
-g (--gid)	Specifies the GID to be assigned to the group
-o (--non-unique)	Creates a group with a matching GID of an existing group. When two groups have an identical GID, members of both groups get identical rights on each other’s files. This should only be done in specific situations.
-r	Creates a system group with a GID below 1000
groupname	Specifies a group name

Table 6-4 groupadd Command Options

The *groupadd* command picks up the default values from the *login.defs* file.

You can modify the attributes of a group with the *groupmod* command. The syntax of this command is very similar to the *groupadd* with most options identical. The only flag that is

additional with this command is -n, which can change the name of an existing group.

The *groupdel* command is straightforward. It removes entries for the specified group from both *group* and *gshadow* files.

Exercise 6-4: Create a Group and Add Members

This exercise should be done on *server1* as *root*.

In this exercise, you will create a group called *linuxadm* with GID 5000 and another group called *dba* sharing the GID 5000. You will add *user1* as a secondary member to group *linuxadm*.

1. Create the group *linuxadm* with GID 5000:

```
[root@server1 ~]# groupadd -g 5000 linuxadm
```

2. Create a group called *dba* with the same GID as that of group *linuxadm*:

```
[root@server1 ~]# groupadd -o -g 5000 dba
```

3. Confirm the creation of both groups:

```
[root@server1 ~]# grep linuxadm /etc/group
linuxadm:x:5000:
[root@server1 ~]# grep dba /etc/group
dba:x:5000:
```

The GID for both groups is identical.

4. Add *user1* as a secondary member of group *dba* using the *usermod* command. The existing membership for the user must remain intact.

```
[root@server1 ~]# usermod -aG dba user1
```

5. Verify the updated group membership information for *user1* by extracting the relevant entry from the *group* file, and running the *id* and *groups* command for *user1*:

```
[root@server1 ~]# grep dba /etc/group
dba:x:5000:user1
[root@server1 ~]#
[root@server1 ~]# id user1
uid=1000(user1) gid=1000(user1) groups=1000(user1),5000(dba)
[root@server1 ~]# groups user1
user1 : user1 dba
```

The output confirms the primary (*user1*) and secondary (*dba*) group memberships for *user1*.

Exercise 6-5: Modify and Delete a Group Account

This exercise should be done on *server1* as *root*.

In this exercise, you will change the *linuxadm* group name to *sysadm* and the GID to 6000. You will modify the primary group for *user200* to *sysadm*. Finally, you will remove the *sysadm* group and verify the actions.

1. Alter the name of *linuxadm* to *sysadm*:

```
[root@server1 ~]# groupmod -n sysadm linuxadm
```

2. Change the GID of *sysadm* to 6000:

```
[root@server1 ~]# groupmod -g 6000 sysadm
```

3. Confirm the above actions:

```
[root@server1 ~]# grep sysadm /etc/group
sysadm:x:6000:user1,user200
[root@server1 ~]# grep linuxadm /etc/group
[root@server1 ~]# _
```

The outputs confirm both actions. The new group name is *sysadm* and the new GID is 6000. The membership remains the same. Also notice that the *linuxadm* group does not exist anymore. The second command returned nothing for it.

4. Delete the *sysadm* group and confirm:

```
[root@server1 ~]# groupdel sysadm
[root@server1 ~]# grep sysadm /etc/group
[root@server1 ~]#
```

The second command returns nothing, which confirms a successful deletion of the group.

Substituting Users and Doing as Superuser

In real world Linux environments, you'll always sign in as a normal user. Normal users have limited rights on the system. Their profiles are set to allow them to accomplish routine jobs efficiently and securely. This is done to secure the overall system functionality and to prevent unexpected issues. There are times though when a normal user needs to assume the identity of a different user to carry out a task as that user or execute a command successfully that requires elevated privileges. Linux offers two fundamental tools to support users in both situations.

Substituting (or Switching) Users

Even though you can log in to the system directly as *root*, it is not a recommended practice. Instead, log in with a normal user account and then switch to the *root* account if necessary. This is safer and ensures system security and protection. In addition to becoming *root*, you can switch into another user account. In either case, you'll need to know the password for the target user in order for a successful switch. The *su* command has the ability to switch into other user accounts.

To switch from *user1* (assuming you are logged in as *user1*) into *root* without executing the startup scripts (startup scripts are explained later in this chapter) for the target user, run the *su* command and enter the *root* user password when prompted:

```
[user1@server1 ~]$ su
```

Press Ctrl+d key combination to return to the *user1* command prompt. Then rerun the *su* command with the hyphen character (-) specified to ensure that startup scripts for the target user are also executed to provide an environment similar to that of a real login:

```
[user1@server1 ~]$ su -
```

To switch into a different user account, such as *user100*, specify the name of the target user with the command. You must enter the password of the target user when prompted.

```
[user1@server1 ~]$ su - user100
```

RHEL offers two tools—*whoami* (*who am i*) and *logname* (*login name*)—that show a user's current identity (after *su*'ing into the target user) and the identity of the user who originally logged in. Let's see what they report after switching into *user100*:

```
[user1@server1 ~]$ su - user100
Password:
[user100@server1 ~]$ whoami
user100
[user100@server1 ~]$ logname
user1
```

The *whoami* command returns the effective (current) username (*user100*), and the *logname* command reports the user's real (original) username (*user1*).

To issue a command as a different user without switching into that user, use the -c option with *su*. For example, the *firewall-cmd* command with the --list-services option requires superuser privileges. *user1* can use *su* as follows and execute this privileged command to obtain desired results:

```
[user1@server1 ~]$ su -c 'firewalld-cmd --list-services'
```

The *root* user can switch into any user account that exists on the system without being prompted for that user's password.

Although the *su* command does provide the flexibility and convenience, switching into the *root* account to execute privileged actions is not recommended. There is a preferred method available in Linux that should be used instead. The following section sheds light on it.

Doing as Superuser (or Doing as Substitute User)

On production Linux servers, there are necessary steps to ensure that users have the ability to carry out their assigned job without hassle. In most cases, this requires privileged access to certain tools and functions, which the *root* user is normally allowed to run.

RHEL provides normal users the ability to run a set of privileged commands or to access non-owning files without the knowledge of the *root* password. This allows the flexibility of assigning a specific command or a set of commands to an individual user or a group of users based on their needs. These users can then precede one of those commands with a utility called *sudo* (*superuser do*, a.k.a. *substitute user do*) at the time of executing that command. The users are prompted to enter their own password, and if correct, the command is executed successfully for them. The *sudo* utility is designed to provide protected access to administrative functions as defined in the */etc/sudoers* file or files in the drop-in directory */etc/sudoers.d*. It can also be used to allow a user or a group of users to run scripts and applications owned by a different user.

Any normal user that requires privileged access to administrative commands or non-owning files is defined in the *sudoers* file. This file may be edited with a command called *visudo*, which creates a copy of the file as *sudoers.tmp* and applies the changes there. After the *visudo* session is over, the updated file overwrites the original *sudoers* file and

sudoers.tmp is deleted. This is done to prevent multiple users editing this file simultaneously.

The syntax for user and group entries in the file is similar to the following example entries for user *user1* and group *dba*:

```
user1    ALL=(ALL)     ALL
%dba     ALL=(ALL)     ALL
```

These entries may be added to the beginning of the file, and they are intended to provide full access to every administrative function to *user1* and members of the *dba* group (group is prefixed by the percentage sign (%)). In other words, *user1* and *dba* group members will have full *root* user authority on the system.

When *user1* or a *dba* group member attempts to access a privileged function, they will be required to enter their own password. For instance:

```
[user1@server1 ~]$ sudo head /etc/sudoers
[sudo] password for user1:
user1 ALL=(ALL) ALL
%dba ALL=(ALL) ALL
## Sudoers allows particular users to run various commands as
## the root user, without needing the root password.
##
## Examples are provided at the bottom of the file for collections
## of related commands, which can then be delegated out to particular
## users or groups.
##
## This file must be edited with the 'visudo' command.
```

If you want *user1* and *dba* group members not to be prompted for their passwords, modify the entries in the *sudoers* file to look like:

```
user1    ALL=(ALL)     NOPASSWD:ALL
%dba     ALL=(ALL)     NOPASSWD:ALL
```

Rather than allowing them full access to the system, you can restrict their access to the functions that they need access to.

For example, to limit their access to a single command */usr/bin/cat*, modify the directives as follows:

```
user1   ALL=/usr/bin/cat
%dba    ALL=/usr/bin/cat
```

These users should now be able to use the *cat* command to view the content of the */etc/sudoers* or any other file that requires the *root* privilege. Try **cat /etc/sudoers** as *user1* and then again as **sudo cat /etc/sudoers**. You will see the difference.

Configuring sudo to work the way it has just been explained may result in a cluttered *sudoers* file with too many entries to fulfill diversified needs of users and groups. A preferred method is to use predefined aliases—User_Alias and Cmnd_Alias—to configure groups of users and commands, and assign access as required. For instance, you can define a Cmnd_Alias called PKGCMD containing *yum* and *rpm* package management commands, and a User_Alias called PKGADM for *user1*, *user100*, and *user200*. These users may or may not belong to the same Linux group.

Next, assign PKGCMD to PKGADM. This way one rule is set that allows a group of users access to a group of commands. You can add or remove commands and users anytime as needed, and the change will take effect right away. Here is what this configuration will look like:

```
Cmnd_Alias      PKGCMD = /usr/bin/yum, /usr/bin/rpm
User_Alias      PKGADM = user1, user100, user200
PKGADM          ALL = PKGCMD
```

Append the above to the bottom of the *sudoers* file and then run the *yum* or *rpm* command preceded by *sudo* as one of the users listed. You will be able to perform software management tasks just like the *root* user.

The *sudo* command logs successful authentication and command data to the */var/log/secure* file under the name of the actual user executing the command (and not *root*).

The *sudoers* file contains several examples with a brief explanation. It is a good idea to look at those examples for additional understanding.

Now that the normal user *user1* is added to the *sudoers* configuration file, you will be using this user account with *sudo* where appropriate in the examples and exercises in the remainder of the book.

Owning User and Owning Group

In Linux, every file and directory has an owner. By default, the creator assumes the ownership, but this may be altered and allocated to a different user if required.

Similarly, every user is a member of one or more groups. A group is a collection of users with common permission requirements. By default, the owner's group is assigned to a file or directory.

Let's create a file *file1* as *user1* in their home directory and exhibit the file's long listing:

```
[user1@server1 ~]$ touch file1
[user1@server1 ~]$ ls -l file1
-rw-rw-r--. 1 user1 user1 0 Sep 17 15:00 file1
```

The output indicates that the owner of *file1* is *user1* who belongs to group *user1*. If you wish to view the corresponding UID and GID instead, you can specify the -n option with the command:

```
[user1@server1 ~]$ ls -ln file1
-rw-rw-r--. 1 1000 1000 0 Sep 17 15:00 file1
```

Linux provides the *chown* and *chgrp* commands that can alter the ownership and owning group for files and directories; however, you must have the *root* user privilege to make these modifications.

Exercise 6-6: Modify File Owner and Owning Group

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will first create a file *file10* and a directory *dir10* as *user1* under */tmp*, and then change the ownership for *file10* to *user100* and the owning group to *dba* in two separate transactions. Then you'll apply ownership on *file10* to *user200* and owning group to *user100* at the same time. Finally, you will change the two attributes on the directory to *user200:dba* recursively. Make sure to use *sudo* where necessary.

1. Change into the */tmp* directory and create *file10* and *dir10*:

```
[user1@server1 ~]$ cd /tmp
[user1@server1 tmp]$ touch file10
[user1@server1 tmp]$ mkdir dir10
```

2. Check and validate that both attributes are set to *user1*:

```
[user1@server1 tmp]$ ls -l file10
-rw-rw-r--. 1 user1 user1 0 Sep 17 15:15 file10
[user1@server1 tmp]$ ls -ld dir10
drwxrwxr-x. 2 user1 user1 6 Sep 17 15:15 dir10
```

3. Set the ownership to *user100* and confirm:

```
[user1@server1 tmp]$ sudo chown user100 file10
[user1@server1 tmp]$ ls -l file10
-rw-rw-r--. 1 user100 user1 0 Sep 17 15:15 file10
```

4. Alter the owning group to *dba* and verify:

```
[user1@server1 tmp]$ sudo chgrp dba file10
[user1@server1 tmp]$ ls -l file10
-rw-rw-r--. 1 user100 dba 0 Sep 17 15:15 file10
```

5. Change the ownership to *user200* and owning group to *user100* and confirm:

```
[user1@server1 tmp]$ sudo chown user200:user100 file10
[user1@server1 tmp]$ ls -l file10
-rw-rw-r--. 1 user200 user100 0 Sep 17 15:15 file10
```

6. Modify the ownership to *user200* and owning group to *dba* recursively on *dir10* and validate:

```
[user1@server1 tmp]$ sudo chown -R user200:dba dir10
[user1@server1 tmp]$ ls -ld dir10
drwxrwxr-x. 2 user200 dba 6 Sep 17 15:15 dir10
```

You can also use **ls -lR dir10** to view file and directory information under *dir10*; however, it will show nothing as the directory is currently empty.

Chapter Summary

At the outset, we described various aging attributes for user login and password controls that are available to us. We also looked at files that store default attributes that are applied to user accounts at the time of their creation. These defaults may be modified if necessary.

Next, we modified aging attributes for certain user accounts with the help of the tools we learned. We also employed a tool with a pair of flags to deny and restore user access to the system.

We examined group management commands and used them in exercises to create, modify, and delete group accounts, and add users to supplementary groups. The set of the management commands is straightforward and easy to use, but they require execution by a privileged user.

We looked at a couple of tools towards the end of the chapter to switch into other user accounts, including the root user, and to run privileged commands as a normal user. The use of the latter tool is recommended.

Finally, we discussed ownership and owning groups on files and directories, and how they are assigned. We performed exercises to understand who can modify them and how.

Check Your Understanding

1. Which command can be used to display the effective (current) username?
2. What is the recommended location to store custom *sudo* rules?
3. What would the command *passwd -l user10* do?
4. The *chown* command may be used to modify both ownership and group membership on a file. True or False?
5. What would the command *passwd -n 7 -x 15 -w 3 user5* do?
6. Write the two command names for managing all of the password aging attributes.
7. Which file has the default password aging settings defined?
8. What would the command *chage -E 2020-10-22 user10* do?
9. What would the command *chage -l user5* do?
10. Which two commands can be used to lock and unlock a user account?
11. When using *sudo*, log files record activities under the *root* user account. True or False?
12. What is the difference between running the *su* command with and without the hyphen sign?
13. What is the significance of the -o option with the *groupadd* and *groupmod* commands?

14. What would the entry user10 ALL=(ALL) NOPASSWD:ALL in the sudoers file imply?
15. The *chgrp* command may be used to modify both ownership and group membership on a file. True or False?
16. What would the command *chage -d 0 user60* do?

Answers to Check Your Understanding

1. The *whoami* command reports the effective username of the user running this command.
2. The custom sudo rules should be stored in files under the */etc/sudoers.d* directory.
3. This command will lock *user10*.
4. True.
5. The command will set mindays to 7, maxdays to 15, and warndays to 3 for *user5*.
6. The *passwd* and *chage* commands.
7. The */etc/login.defs* file has the defaults for password aging defined.
8. The command provided will set October 22, 2020 as the expiry date for *user10*.
9. The command provided will display password aging attributes for *user5*.
10. The *usermod* and *passwd* commands can be used to lock and unlock a user account.
11. False. The activities are registered under the username who invokes the *sudo* command.
12. With the dash sign the *su* command processes the specified user's startup files, and it won't without this sign.
13. The -o option lets the commands share a GID between two or more groups.
14. It will give *user10* password-less access to all privileged commands.

15. False.
16. The command provided will force *user60* to change their password at next login.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 6-1: Create User and Configure Password Aging

As *root* on *server1*, create group *lnxgrp* with GID 6000. Create user *user5000* with UID 5000 and GID 6000. Assign this user a password, and establish password aging attributes so that this user cannot change their password within 4 days after setting it and with a password validity of 30 days. This user should start getting warning messages for changing password 10 days prior to account lock down. This user account needs to expire on the 20th of December, 2021. (Hint1: Chapter 05: Local User Authentication Files, and User Account Management). (Hint2: Password Aging and its Management, and Linux Groups and their Management).

Lab 6-2: Lock and Unlock User

As *root* on *server1*, lock the user account for *user5000* using the *passwd* command, and confirm by examining the change in the */etc/shadow* file. Try to log in with *user5000* and observe what happens. Use the *usermod* command and unlock this account. Verify the unlocking by checking the

entry for the user in the */etc/shadow* file. (Hint: Password Aging and its Management).

Lab 6-3: Modify Group

As *root* on *server1*, modify the GID from 6000 to 7000 for the *lnxgrp* group. Add users *user1000* and *user2000* as supplementary members (create users if needed). Change the group's name to *dbagrp*, and verify. (Hint: Linux Groups and their Management).

Lab 6-4: Configure sudo Access

As *root* on *server1*, add a rule for user5000 to the */etc/sudoers* file to allow this user full *root* access on the system. Make sure that this user is not prompted for a password when they use *sudo* to execute a command. Now switch into this user account and try running **sudo vgs**, and see if that works. (Hint: Substituting Users and Doing as Superuser).

Lab 6-5: Modify Owning User and Group

As *user1* on *server1*, create file *f6* and directory *d6* under */tmp*. Change owning user for *f6* to *user90* (create user) using **sudo chown**, and owning group to *dba* with **sudo chgrp**. Change owning user and group on *d6* to *user90:g1* (create group) recursively using **sudo chown**. (Hint: Owing User and Owning Group).

Chapter 07

The Bash Shell

This chapter describes the following major topics:

- Introduction to the bash shell
- Internal and external commands
- Understand, set, and unset shell and environment variables
- Comprehend and use command and variable expansions
- Grasp and utilize input, output, and error redirections
- Know and use history substitution, command line editing, tab completion, tilde substitution, and command aliasing
- Identify and use metacharacters, wildcard characters, pipes, and pipelines
- Discover the use of quoting mechanisms and regular expressions

- Run and control jobs in background and foreground
- Identify and examine shell startup files

RHCSA Objectives:

02. Use input-output redirection (>, >>, |, 2>, etc.)
03. Use grep and regular expressions to analyze text

Shells allow users to interact with the operating system for execution of their instructions through programs, commands, and applications. RHEL supports a variety of shells of which the bash shell is the most common. It is also the default shell for users in RHEL 8. This shell offers a variety of features that help users and administrators perform their job with ease and flexibility. Some of these features include recalling a command from history and editing it at the command line before running it, sending output and error messages to non-default target locations, creating command shortcuts, sending output of one command as input to the next, assigning the output of a command to a variable, using special characters to match a string of characters, treating a special character as a literal character, and so on.

Regular expressions are text patterns for matching against an input provided in a search operation. Text patterns may include any sequenced or arbitrary characters or character range. RHEL offers a powerful command to work with pattern matching.

At login, plenty of system and user startup scripts are executed to set up the user environment. The system startup scripts may be customized for all users by the Linux administrator and the user startup scripts may be modified by individual users as per their need.

The Bourne-Again Shell

A *shell* is referred to as the command interpreter, and it is the interface between a user and the Linux kernel. The shell accepts instructions (commands) from users (or programs),

interprets them, and passes them on to the kernel for processing. The kernel utilizes all hardware and software components required for a successful processing of the instructions. When concluded, it returns the results to the shell, which then exhibits them on the screen. The shell also shows appropriate error messages, if generated. In addition, the shell delivers a customizable environment to users.

A widely used shell by Linux users and administrators is the *bash* (*bourne-again shell*) shell. Bash is a replacement for the older *Bourne* shell with numerous enhancements and plenty of new features incorporated from other shells. It is the default shell in most popular Linux distributions including RHEL 8 and offers several features such as variable manipulation, variable substitution, command substitution, input and output redirections, history substitution, tab completion, tilde substitution, alias substitution, metacharacters, pattern matching, filename globbing, quoting mechanisms, conditional execution, flow control, and shell scripting. This section discusses all of these features except for the last three.

The bash shell is identified by the dollar sign ($) for normal users and the hash sign (#) for the *root* user. The bash shell is resident in the */usr/bin/bash* file.

Internal and External Shell Commands

There is a rich collection of commands built in to the bash shell known as the *internal* commands. These include *cd*, *pwd*, *umask*, *alias*/*unalias*, *history*, *command*, *.* (dot), *export*, *exit*, *test*, *shift*, *set*/*unset*, *source*, *exec*, and *break*. Upon invocation, these commands are executed directly by the shell without creating a new process for them. Other commands located in various directories, such as */usr/bin* and */usr/sbin*, are *external* to the shell, and the shell spawns a temporary sub-shell (child shell) to run them.

Shell and Environment Variables

A *variable* is a transient storage for data in memory. It retains information that is used for customizing the shell environment and referenced by many programs to function properly. The shell stores a value in a variable, and one or more white space characters must be enclosed within quotation marks (“”).

There are two types of variables: *local* (or *shell*) and *environment*. A local variable is private to the shell in which it is created, and its value cannot be used by programs that are not started in that shell. This introduces the concept of *current* shell and *sub*-shell (or *child* shell). The current shell is where a program is executed, whereas a sub-shell (or child shell) is created within a shell to run a program. The value of a local variable is only available in the current shell.

The value of an environment variable is inherited from the current shell to the sub-shell during the execution of a program. In other words, the value stored in an environment variable is accessible to the program, as well as any sub-programs that it spawns during its lifecycle. Any environment variable set in a sub-shell is lost when the sub-shell terminates.

There are a multitude of predefined environment variables that are set for each user upon logging in. Use the **env** or the **printenv** command to view their values. Run these commands on *server1* as *user1* and observe the output. There should be around 25 of them. Some of the common predefined environment variables are described in Table 7-1.

Variable	Description
DISPLAY	Stores the hostname or IP address for graphical terminal sessions
HISTFILE	Defines the file for storing the history of executed commands
HISTSIZE	Defines the maximum size for the HISTFILE
HOME	Sets the home directory path
LOGNAME	Retains the login name
MAIL	Contains the path to the user mail directory
PATH	Defines a colon-separated list of directories to be searched when executing a command. A correct setting of this variable eliminates the need to specify the absolute path of a command to run it.
PPID	Holds the identifier number for the parent program
PS1	Defines the primary command prompt
PS2	Defines the secondary command prompt
PWD	Stores the current directory location
SHELL	Holds the absolute path to the primary shell file
TERM	Holds the terminal type value
UID	Holds the logged-in user's UID
USER	Retains the name of the logged-in user

Table 7-1 Common Predefined Environment Variables

RHEL provides the *echo* command to view the values stored in variables. For instance, to view the value for the PATH variable, run the *echo* command and ensure to prepend the variable name (PATH) with the dollar sign ($):

```
[user1@server1 ~]$ echo $PATH
/home/user1/.local/bin:/home/user1/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/
usr/sbin
```

Try running **echo $HOME**, **echo $SHELL**, **echo $TERM**, **echo $PPID**, **echo $PS1**, and **echo $USER** and see what values they store.

Setting and Unsetting Variables

Shell and environment variables may be set or unset at the command prompt or via programs, and their values may be viewed and used as necessary. We define and undefine variables and view their values using built-in shell commands such as *export*, *unset*, and *echo*. It is recommended to use uppercase letters to name variables so as to circumvent any possible conflicts with a command, program, file, or directory name that exist somewhere on the system. To understand how variables are defined, viewed, made environment, and undefined, a few examples are presented below. We run the commands as *user1*.

To define a local variable called VR1:

```
[user1@server1 ~]$ VR1=RHEL8
```

To view the value stored in VR1:

```
[user1@server1 ~]$ echo $VR1
```

Now type **bash** at the command prompt to enter a sub-shell and then run **echo $VR1** to check whether the variable is visible in the sub-shell. You will not find it there, as it was not an environment variable. Exit out of the sub-shell by typing **exit**.

To make this variable an environment variable, use the *export* command:

```
[user1@server1 ~]$ export VR1
```

Repeat the previous test by running the **bash** command and then the **echo $VR1**. You should be able to see the variable in the sub-shell, as it is now an environment variable. Exit out of the sub-shell with the **exit** command.

To undefine this variable and remove it from the shell environment:

```
[user1@server1 ~]$ unset VR1
```

To define a local variable that contains a value with one or more white spaces:

```
[user1@server1 ~]$ VR2="I love RHEL 8"
```

To define and make the variable an environment variable at the same time:

```
[user1@server1 ~]$ export VR3="I love RHEL 8"
```

The *echo* command is restricted to showing the value of a specific variable and the *env* and *printenv* commands display the environment variables only. If you want to view both local and environment variables, use the *set* command. Try running **set** and observe the output.

Command and Variable Substitutions

The primary command prompt ends in the hash sign (#) for the *root* user and in the dollar sign ($) for normal users. Customizing this prompt to exhibit useful information such as who you are, the system you are logged on to, and your current location in the directory tree is a good practice. By default, this setting is already in place through the PS1 environment variable, which defines the primary command prompt. You can validate this information by looking at the command prompt **[user1@server1 ~]$**.

You can also view the value stored in PS1 by issuing **echo $PS1**. The value is \u@\h \W\$. The \u translates into the logged-in username, \h represents the hostname of the system, \W shows your current working directory, and \$ indicates the end of the command prompt.

Exercise 7-1 demonstrates how to employ the command and variable substitution features to customize the primary command prompt for *user1*. You will use the *hostname* command and assign its output to the PS1 variable. This is an example of *command substitution*. Note that the

command whose output you want assigned to a variable must be encapsulated within either backticks `hostname` or parentheses $(hostname).

There are two instances of the *variable substitution* feature employed in Exercise 7-1. The LOGNAME and the PWD environment variables to display the username and to reflect the current directory location.

Exercise 7-1: Modify Primary Command Prompt

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will customize the primary shell prompt to display the information enclosed within the quotes "< username on hostname in pwd >:" using the variable and command substitution features. You will do this at the command prompt. You will then edit the *~/.profile* file for *user1* and define the new value in there for permanence (see Shell Startup Files later in this chapter to understand the purpose of *.profile*), otherwise, the value will be lost when *user1* closes the terminal window or logs off.

1. Change the value of the variable PS1 to reflect the desired information:

```
[user1@server1 ~]$ export PS1="< $LOGNAME on $(hostname) in \$PWD > "
< user1 on server1.example.com in /home/user1 >
```

The prompt has changed to display the desired information.

2. Edit the *.profile* file for *user1* and define the value exactly as it was run in Step 1.
3. Test by logging off as *user1* and logging back in. The new command prompt will be displayed.

Input, Output, and Error Redirections

Programs read input from the keyboard and write output to the terminal window where they are initiated. Any errors, if encountered, are printed on the terminal window too. This is the default behavior. The bash handles input, output, and errors as character streams. If you do not want input to come from the keyboard or output and error to go to the terminal screen, the shell gives you the flexibility to redirect input, output, and error messages to allow programs and commands to read input from a non-default source, and forward output and errors to one or more non-default destinations.

> **EXAM TIP:** You may be asked to run a command or report and redirect its output and/or error messages to a file.

The default (or the standard) locations for the three streams are referred to as *standard input* (or *stdin*), *standard output* (or *stdout*), and *standard error* (or *stderr*). These locations may also be epitomized using the opening angle bracket symbol (<) for stdin, and the closing angle bracket symbol (>) for stdout and stderr. Alternatively, you may use the *file descriptors* (the digits 0, 1, and 2) to represent the three locations.

Redirecting Standard Input

Input redirection instructs a command to read input from an alternative source, such as a file, instead of the keyboard. The opening angle bracket (<) is used for input redirection. For example, run the following to have the *cat* command read the */etc/redhat-release* file and display its content on the standard output (terminal screen):

```
[user1@server1 ~]$ cat < /etc/redhat-release
Red Hat Enterprise Linux release 8.0 (Ootpa)
```

Redirecting Standard Output

Output redirection sends the output generated by a command to an alternative destination, such as a file, instead of to the terminal window. The closing angle bracket (>) is used for this purpose. For instance, the following directs the *ls* command output to a file called *ls.out*. This will overwrite any existing *ls.out* file if there is one; otherwise, a new file will be created.

```
[user1@server1 ~]$ ls > ls.out
```

The above can also be run as **ls 1> ls.out** where the digit "1" represents the standard output location.

If you want to prevent an inadvertent overwriting of the output file, you can enable the shell's noclobber feature with the *set* command and confirm its activation by re-issuing the above redirection example:

```
[user1@server1 ~]$ set -o noclobber
[user1@server1 ~]$ ls > ls.out
-bash: ls.out: cannot overwrite existing file
```

You are denied the action.

You can disable the noclobber option by running **set +o noclobber** at the command prompt.

To direct the *ls* command to append the output to the *ls.out* file instead of overwriting it, use the two closing angle brackets (>>):

```
[user1@server1 ~]$ ls >> ls.out
```

Again, the equivalent for the above is **ls 1>> ls.out**.

Redirecting Standard Error

Error redirection forwards any error messages generated to an alternative destination rather than to the terminal

window. An alternative destination could be a file. For example, the following directs the *find* command issued as a normal user to search for all occurrences of files by the name *core* in the entire directory tree and sends any error messages produced to */dev/null* (*/dev/null* is a special file that is used to discard data). This way only the useful output is exhibited on the screen and errors are thrown away.

```
[user1@server1 ~]$ find / -name core -print 2> /dev/null
```

Redirecting both Standard Output and Error

You may redirect both output and error to alternative locations as well. For instance, issue the following to forward them both to a file called *outerr.out*:

```
[user1@server1 ~]$ ls /usr /cdr &> outerr.out
```

This example will produce a listing of the */usr* directory and save the result in *outerr.out*. At the same time, it will generate an error message complaining about the non-existence of */cdr*, and it will send it to the same file as well.

Another method to run the above command is by typing **ls /usr /cdr 1> outerr.out 2>&1**, which essentially means to redirect file descriptor 1 to file *outerr.out* as well as to file descriptor 2.

You can exchange &> with &>> in the above example to append the information in the file rather than overwriting it.

History Substitution

History substitution (a.k.a. *command history* or *history expansion*) is a time-saver bash shell feature that keeps a log of all commands or commandsets that you run at the command prompt in chronological order with one command or commandset per line. The history feature is enabled by

default; however, you can disable and re-enable it if required. The bash shell stores command history in a file located in the user's home directory and in system memory. You may retrieve the commands from history, modify them at the command prompt, and rerun them.

There are three variables—HISTFILE, HISTSIZE, and HISTFILESIZE—that control the location and history storage. HISTFILE defines the name and location of the history file to be used to store command history, and the default is *.bash_history* in the user's home directory. HISTSIZE dictates the maximum number of commands to be held in memory for the current session. HISTFILESIZE sets the maximum number of commands allowed for storage in the history file at the beginning of the current session and are written to the HISTFILE from memory at the end of the current terminal session. Usually, HISTSIZE and HISTFILESIZE are set to a common value. These variables and their values can be viewed with the *echo* command. The following shows the settings for *user1*:

```
[user1@server1 ~]$ echo $HISTFILE
/home/user1/.bash_history
[user1@server1 ~]$ echo $HISTSIZE
1000
[user1@server1 ~]$ echo $HISTFILESIZE
1000
```

The values of any of these variables may be altered for individual users by editing the *.bashrc* or *.bash_profile* file in the user's home directory. A discussion on these files is provided later in this chapter.

In RHEL, the *history* command displays or reruns previously executed commands. This command gets the history data from the system memory as well as from the *.bash_history* file. By default, it shows all entries. Run this command at the prompt without any options and it will dump everything on the screen:

```
[user1@server1 ~]$ history
```

The *history* command has some useful options. Let's use them and observe the impact on the output.

To display this command and the ten preceding entries:

```
[user1@server1 ~]$ history 10
```

To re-execute a command by its line number (line 15 for example):

```
[user1@server1 ~]$ !15
```

To re-execute the most recent occurrence of a command that started with a particular letter or series of letters (ch for example):

```
[user1@server1 ~]$ !ch
```

To issue the most recent command that contained "grep":

```
[user1@server1 ~]$ !?grep?
```

To remove entry 24 from history:

```
[user1@server1 ~]$ history -d 24
```

To repeat the last command executed:

```
[user1@server1 ~]$ !!
```

The second exclamation mark (!) in the above example is used to retrieve the last executed command.

You may disable the shell's history expansion feature by issuing **set +o history** at the command prompt and re-enable it with **set -o history**. The *set* command is used in this way to enable or disable a bash shell feature.

Editing at the Command Line

While typing a command with a multitude of options and arguments at the command prompt, you often need to move the cursor backward to add or modify something. For instance, if you are typing a long command as a normal user and then realize the command needs to have *sudo* added at the beginning, move the cursor quickly to the start of the command. This is a shortcut to save time. There are plenty of key combinations for rapid movement on the command line. Table 7-2 lists and explains several of these.

Key Combinations	Action
Ctrl+a / Home	Moves the cursor to the beginning of the command line
Ctrl+e / End	Moves the cursor to the end of the command line
Ctrl+u	Erase the entire line
Ctrl+k	Erase from the cursor to the end of the command line
Alt+f	Moves the cursor to the right one word at a time
Alt+b	Moves the cursor to the left one word at a time
Ctrl+f / Right arrow	Moves the cursor to the right one character at a time
Ctrl+b / Left arrow	Moves the cursor to the left one character at a time

Table 7-2 Helpful Command Line Editing Shortcuts

For normal users and administrators who often work at the command line, these key combinations will be very helpful in saving time.

Tab Completion

Tab completion (a.k.a. *command line completion*) is a bash shell feature whereby typing one or more initial characters of

a file, directory, or command name at the command line and then hitting the Tab key twice automatically completes the entire name. In case of multiple possibilities matching the entered characters, it completes up to the point they have in common and prints the rest of the possibilities on the screen. You can then type one or more following characters and press Tab again to further narrow down the possibilities. When the desired name appears, press Enter to accept it and perform the action. One of the major benefits of using this feature is the time saved on typing long file, directory, or command names.

Try this feature out on your Linux terminal window and get yourself familiarized with it.

Tilde Substitution

Tilde substitution (or *tilde expansion*) is performed on words that begin with the tilde character (~). The rules to keep in mind when using the ~ are:

1. If ~ is used as a standalone character, the shell refers to the $HOME directory of the user running the command. The following example displays the $HOME directory of *user1*:

```
[user1@server1 ~]$ echo ~
/home/user1
```

2. If the plus sign (+) follows the ~, the shell refers to the current directory. For example, if *user1* is in the */usr/bin* directory and does ~+, the output exhibits the user's current directory location:

```
[user1@server1 bin]$ echo ~+
/usr/bin
```

3. If the hyphen character (-) follows the ~, the shell refers to the previous working directory. For example, if

user1 switches into the */usr/share/man* directory from */usr/bin* and does ~-, the output displays the user's last working directory:

```
[user1@server1 bin]$ cd /usr/share/man
[user1@server1 man]$ echo ~-
/usr/bin
```

4. If a username has the ~ prepended, the shell refers to the $HOME directory of that user:

```
[user1@server1 man]$ echo ~user200
/home/user200
```

Tilde substitution can also be used with commands such as *cd* and *ls*. For instance, to *cd* into the home directory of *user1* (or any other user for that matter) from anywhere in the directory structure, specify the login name with the ~:

```
[user1@server1 man]$ cd ~user1
[user1@server1 ~]$ pwd
/home/user1
```

This command is only successful if *user1* has the right permissions to navigate into the target user's home directory. Let's try this example as the *root* user:

```
[root@server1 ~]# cd ~user100
[root@server1 user100]# pwd
/home/user100
```

The above example demonstrates the navigation into the home directory of *user100* as the *root* user. By default, *root* has full rights to *cd* into any users' home directory.

Another example below exhibits how a user can *cd* into one of their subdirectories (such as */home/user1/dir1*) directly from anywhere (*/etc/systemd/system*) in the directory structure (create *dir1* for this test):

```
[user1@server1 ~]$ cd
[user1@server1 ~]$ mkdir dir1
[user1@server1 ~]$ cd /etc/systemd/system/
[user1@server1 system]$ pwd
/etc/systemd/system
[user1@server1 system]$ cd ~/dir1
[user1@server1 dir1]$ pwd
/home/user1/dir1
```

Try using the ~ with the *ls* command. For example, run **sudo ls -ld ~root/Desktop** as *user1*.

Alias Substitution

Alias substitution (a.k.a. *command aliasing* or *alias*) allows you to define a shortcut for a lengthy and complex command or a set of commands. Defining and using aliases saves time and saves you from typing. The shell executes the corresponding command or commandset when an alias is run.

The bash shell includes several predefined aliases that are set during user login. These aliases may be viewed with the *alias* command. The following shows all aliases that are currently set for *user1*:

```
[user1@server1 ~]$ alias
alias vi='vim'
alias which='(alias; declare -f) | /usr/bin/which --tty-only --read-alias --read
-functions --show-tilde --show-dot'
```

There are more default aliases set for the *root* user. Run *alias* as *root*:

```
[root@server1 ~]# alias
alias cp='cp -i'
alias ll='ls -l'
alias mv='mv -i'
alias rm='rm -i'
alias which='(alias; declare -f) | /usr/bin/which --tty-only --read-alias --read
-functions --show-tilde --show-dot'
```

Table 7-3 describes aliases from the previous two outputs.

Alias	Value	Definition
vi	vim	Runs the vim command instead of the old vi
cp	cp -i	Issues the cp command in interactive mode
ll	ls -l	Shows the ls command output in long format
mv	mv -i	Executes the mv command in interactive mode
rm	rm -i	Runs the rm command in interactive mode
which	(alias; declare -f) | /usr/bin/which -- tty-only --read-alias --read-functions --show-tilde --show-dot	Runs the which command with the specified options

Table 7-3 Predefined Aliases

In addition to listing the set aliases, the *alias* command can also be used to define new aliases. The opposite function of unsetting an alias is performed with the *unalias* command. Both *alias* and *unalias* are internal shell commands. Let's look at a few examples.

Create an alias "search" to abbreviate the *find* command with several switches and arguments. Enclose the entire command within single quotation marks (‘’) to ensure white spaces are taken care of. Do not leave any spaces before and after the equal sign (=).

```
[user1@server1 ~]$ alias search='find / -name core -exec ls -l {} \;'
```

Now, when you type the string "search" at the command prompt and press the Enter key, the shell will trade the alias "search" with what is stored in it and will run it. Essentially, you have created a shortcut to that lengthy command.

Sometimes you define an alias by a name that matches the name of some system command or program. In this situation, the shell gives the execution precedence to the alias. This means the shell will run the alias and not the command or the program. For example, the *rm* command deletes a file without giving any warning if run as a normal user. To prevent accidental deletion of files with *rm*, you may create an alias by the same name as the command but with the interactive option added, as shown below:

```
[user1@server1 ~]$ alias rm='rm -i'
```

When you execute *rm* now to remove a file, the shell will run what is stored in the *rm* alias and not the command *rm*. If you wish to run the *rm* command instead, run it by preceding a backslash (\) with it:

```
[user1@server1 ~]$ \rm file1
```

You can use the *unalias* command to unset one or more specified aliases if they are no longer in need. The following will undefine the two aliases that were defined for our examples:

```
[user1@server1 ~]$ unalias search rm
```

Metacharacters and Wildcard Characters

Metacharacters are special characters that possess special meaning to the shell. Some of them are the dollar sign ($), caret (^), period (.), asterisk (*), question mark (?), pipe (|), angle brackets (< >), curly brackets ({}), square brackets ([]), parentheses (()), plus (+), exclamation mark (!), semicolon (;), and backslash (\) characters. They are used in pattern matching (a.k.a. *filename expansion* or *file globbing*) and regular expressions. This sub-section discusses the metacharacters (* ? [] !) that are used in pattern matching. The *, ?, and [] characters are also referred to as *wildcard characters*.

The * Character

The asterisk (*) matches zero to an unlimited number of characters except for the leading period (.) in a hidden filename. See the following examples on usage.

To list all files in the */etc* directory that begin with letters “ma” and followed by any characters:

```
[user1@server1 ~]$ ls /etc/ma*
/etc/machine-id  /etc/mailcap                    /etc/man_db.conf
/etc/magic       /etc/makedumpfile.conf.sample
```

To list all hidden files and directories in */home/user1*:

```
[user1@server1 ~]$ ls -d .*
.  ..  .bash_history  .bash_logout  .bash_profile  .bashrc  .lesshst  .mozilla
```

To list all files in the */var/log* directory that end in “.log”:

```
[user1@server1 ~]$ ls /var/log/*.log
/var/log/boot.log         /var/log/dnf.rpm.log       /var/log/Xorg.0.log
/var/log/dnf.librepo.log  /var/log/hawkey.log        /var/log/Xorg.9.log
/var/log/dnf.log          /var/log/vmware-vmusr.log
```

The * is probably the most common metacharacter that is used in pattern matching.

The ? Character

The question mark (?) matches exactly one character except for the leading period in a hidden filename. See the following example to understand its usage.

To list all files and directories under */var/log* with exactly four characters in their names:

```
[user1@server1 ~]$ ls -d /var/log/????
/var/log/btmp  /var/log/cups  /var/log/sssd
/var/log/cron  /var/log/rhsm  /var/log/wtmp
```

The ? is another metacharacter that is used widely in pattern matching.

The Square Brackets [] and the Exclamation Mark !

The square brackets ([]) can be used to match either a set of characters or a range of characters for a single character position.

For a set of characters specified in this enclosure, the order in which they are listed has no importance. This means the shell will interpret [xyz], [yxz], [xzy], and [zyx] alike during pattern matching. In the following example, two characters are enclosed within the square brackets. The output will include all files and directories that begin with either of the two characters and followed by any number of characters.

```
[user1@server1 ~]$ ls /usr/bin/[yw]*
/usr/bin/w            /usr/bin/whereis               /usr/bin/wvunpack
/usr/bin/wait         /usr/bin/which                 /usr/bin/yelp
/usr/bin/wall         /usr/bin/whiptail              /usr/bin/yelp-build
/usr/bin/watch        /usr/bin/who                   /usr/bin/yelp-check
/usr/bin/watchgnupg   /usr/bin/whoami                /usr/bin/yelp-new
/usr/bin/wavpack      /usr/bin/wnck-urgency-monitor  /usr/bin/yes
/usr/bin/wc           /usr/bin/word-list-compress    /usr/bin/ypdomainname
/usr/bin/wdctl        /usr/bin/write                 /usr/bin/yum
/usr/bin/wget         /usr/bin/wvgain
/usr/bin/whatis       /usr/bin/wvtag
```

A range of characters must be specified in a proper sequence such as [a-z] or [0-9]. The following example matches all directory names that begin with any letter between “m” and “o” in the */etc/systemd/system* directory:

```
[user1@server1 ~]$ ls -d /etc/systemd/system/[m-o]*
/etc/systemd/system/multi-user.target.wants
/etc/systemd/system/network-online.target.wants
/etc/systemd/system/nfs-blkmap.service.requires
/etc/systemd/system/nfs-idmapd.service.requires
/etc/systemd/system/nfs-mountd.service.requires
/etc/systemd/system/nfs-server.service.requires
```

The shell enables the exclamation mark (!) to inverse the matches. For instance, [!a-d]* would exclude all filenames that begin with any of the first four alphabets. The following

example will produce the reverse of what the previous example did:

```
[user1@server1 ~]$ ls -d /etc/systemd/system/[!m-o]*
/etc/systemd/system/basic.target.wants
/etc/systemd/system/bluetooth.target.wants
/etc/systemd/system/dbus-org.bluez.service
/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service
/etc/systemd/system/dbus-org.freedesktop.Avahi.service
/etc/systemd/system/dbus-org.freedesktop.ModemManager1.service
/etc/systemd/system/dbus-org.freedesktop.NetworkManager.service
/etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service
```

The output will have a multitude of filenames printed.

Piping Output of One Command as Input to Another

The *pipe*, represented by the vertical bar (|) and normally resides with the backslash (\) on most keyboards, is used to send the output of one command as input to the next. This character is also used to define alternations in regular expressions. You can use the pipe operator as many times in a command as you require.

The */etc* directory contains plenty of files, but they all do not fit on one terminal screen when you want to see its long listing. You can use the pipe operator to pipe the output to the *less* command in order to view the directory listing one screenful at a time:

```
[user1@server1 ~]$ ls -l /etc | less
total 1352
-rw-r--r--.  1 root root        16 Aug 22 15:26 adjtime
-rw-r--r--.  1 root root      1518 Sep 10  2018 aliases
drwxr-xr-x.  3 root root        65 Aug 22 15:16 alsa
drwxr-xr-x.  2 root root      4096 Aug 22 15:22 alternatives
-rw-r--r--.  1 root root       541 Oct  2  2018 anacrontab
........
```

In another example, the *last* command is run and its output is piped to the *nl* command to number each output line:

```
[user1@server1 ~]$ last | nl
     1  user1    pts/0        192.168.0.219    Wed Sep 18 11:05   still logged in
     2  user1    pts/1        192.168.0.219    Wed Sep 18 10:40 - 11:05  (00:24)
     3  user1    pts/0        192.168.0.219    Wed Sep 18 10:06 - 10:40  (00:34)
     4  user1    pts/0        192.168.0.219    Wed Sep 18 10:00 - 10:02  (00:01)
     5  root     pts/0        192.168.0.219    Wed Sep 18 10:00 - 10:00  (00:00)
     6  user1    pts/0        192.168.0.219    Tue Sep 17 15:00 - 10:00  (18:59)
     7  user1    pts/1        192.168.0.219    Tue Sep 17 10:03 - 12:51  (02:47)
.........
```

The following example sends the output of *ls* to *grep* for the lines that do not contain the pattern “root”. The new output is further piped for a case-insensitive selection of all lines that exclude the pattern “dec”. The filtered output is numbered, and the final result shows the last four lines on the display.

```
[user1@server1 ~]$ ls -l /proc | grep -v root | grep -iv dec | nl | tail -4
    49  dr-xr-xr-x.  9 gdm            gdm                            0 Sep 14 20:23 960
    50  dr-xr-xr-x.  9 gdm            gdm                            0 Sep 14 20:23 979
    51  dr-xr-xr-x.  9 gdm            gdm                            0 Sep 14 20:23 987
    52  dr-xr-xr-x.  9 gdm            gdm                            0 Sep 14 20:23 995
```

A construct like the above with multiple pipes is referred to as a *pipeline*.

Quoting Mechanisms

As you know, metacharacters have special meaning to the shell. In order to use them as regular characters, the bash shell offers three *quoting mechanisms* to disable their special meaning and allow the shell to treat them as literal characters. These mechanisms are available through the use of the backslash (\), single quotation (‘’), and double quotation (“”) characters, and work by prepending a special character to the backslash, or enclosing it within single or double quotation marks.

Prefixing with a Backslash \

The backslash character (\), also referred to as the *escape* character in shell terminology, instructs the shell to mask the meaning of any special character that follows it. For example, if a file exists by the name * and you want to

remove it with the *rm* command, you will have to escape the * so that it is treated as a regular character, not as a wildcard character.

```
[user1@server1 ~]$ rm \*
```

In the above example, if you forget to escape the *, the *rm* command will remove all files from the directory.

Enclosing within Single Quotes ‘’

The single quotation marks (‘’) instructs the shell to mask the meaning of all encapsulated special characters. For example, LOGNAME is a variable, and its value can be viewed with the *echo* command:

```
[user1@server1 ~]$ echo $LOGNAME
user1
```

If you encapsulate $LOGNAME within single quotes, the *echo* command will exhibit the entire string as is:

```
[user1@server1 ~]$ echo '$LOGNAME'
$LOGNAME
```

In the above example, the shell interprets the dollar sign ($) as a literal character, and that’s why the *echo* command displays what is inside the enclosure, including the $ rather than the value of the variable.

Enclosing within Double Quotes “”

The double quotation marks (“”) commands the shell to mask the meaning of all but the backslash (\), dollar sign ($), and single quotes (‘’). These three special characters retain their special meaning when they are enclosed within double quotes. Look at the following examples to understand its usage.

```
[user1@server1 ~]$ echo "$SHELL"
/bin/bash
[user1@server1 ~]$ echo "\$PWD"
$PWD
[user1@server1 ~]$ echo "'\'"
'\'
```

The above three instances confirm the use of the double quotes as a special character pair, as depicted in the outputs.

Regular Expressions

A *regular expression*, also referred to as a *regexp* or simply *regex*, is a text pattern or an expression that is matched against a string of characters in a file or supplied input in a search operation. The pattern may include a single character, multiple random characters, a range of characters, word, phrase, or an entire sentence. Any pattern containing one or more white spaces must be surrounded by quotation marks.

RHEL provides a powerful tool called *grep* (*global regular expression print* or *get regular expression and print*) to work with pattern matching in regular expressions. This tool searches the contents of one or more text files or input supplied for a match. If the expression is matched, *grep* prints every line containing that expression on the screen without changing the source content. *grep* has plenty of options and it accepts expressions in various forms.

> **EXAM TIP:** The grep command is a handy tool to extract needed information from a file or command output. The extracted information can then be redirected to a file. The sequence of the grep'ed data remains unchanged.

Let's consider the following examples to comprehend the usage of the *grep* command.

To search for the pattern "operator" in the */etc/passwd* file:

```
[user1@server1 ~]$ grep operator /etc/passwd
operator:x:11:0:operator:/root:/sbin/nologin
```

To search for the space-separated pattern “aliases and functions” in the $HOME/.bashrc file:

```
[user1@server1 ~]$ grep 'aliases and functions' .bashrc
# User specific aliases and functions
```

To search for the pattern “nologin” in the *passwd* file and exclude (-v) the lines in the output that contain this pattern. Add the -n switch to show the line numbers associated with the matched lines.

```
[user1@server1 ~]$ grep -nv nologin /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
6:sync:x:5:0:sync:/sbin:/bin/sync
7:shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8:halt:x:7:0:halt:/sbin:/sbin/halt
45:user1:x:1000:1000:user1:/home/user1:/bin/bash
46:user100:x:1002:1002::/home/user100:/bin/bash
47:user200:x:1003:1003::/home/user200:/bin/bash
48:user3:x:1010:1010::/usr/user3a:/bin/sh
```

To find any duplicate entries for the *root* user in the *passwd* file. Prepend the caret sign (^) to the pattern “root” to mark the beginning of a line.

```
[user1@server1 ~]$ grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

To identify all users in the *passwd* file with bash as their primary shell. Append the dollar sign ($) to the pattern “bash” to mark the end of a line.

```
[user1@server1 ~]$ grep bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
user1:x:1000:1000:user1:/home/user1:/bin/bash
user100:x:1002:1002::/home/user100:/bin/bash
user200:x:1003:1003::/home/user200:/bin/bash
```

To show the entire *login.defs* file but exclude all the empty lines:

```
[user1@server1 ~]$ grep -v ^$ /etc/login.defs
```

To perform a case-insensitive search (-i) for all the lines in the */etc/bashrc* file that match the pattern “path.”

```
[user1@server1 ~]$ grep -i path /etc/bashrc
    # Need to redefine pathmunge, it gets undefined at the end of /etc/profile
    pathmunge () {
        case ":${PATH}:" in
                    PATH=$PATH:$1
                    PATH=$1:$PATH
    unset -f pathmunge
```

To print all the lines from the */etc/lvm/lvm.conf* file that contain an exact match for a word (-w). You can use the period character (.) in the search string to match a single position. The following example searches for words in the *lvm.conf* file that begin with letters “acce” followed by exactly two characters:

```
[user1@server1 ~]$ grep -w acce.. /etc/lvm/lvm.conf
        # Commands also accept this as a prefix on volume group names.
        # This is a list of regular expressions used to accept or reject block
        # (or any character) and is preceded by 'a' to accept the path, or
        # device is rejected. Unmatching path names do not affect the accept
        # The minimum IO size for VDO volume to accept, in bytes.
        # Allow quicker VG write access during high volume read access.
        # When there are competing read-only and read-write access requests for
        # be serviced. Without this setting, write access may be stalled by a
        # errors on access. Using 'zero' will return success (and zero) on I/O
```

In addition to the caret (^), dollar ($), and period (.), the asterisk (*), question mark (?), square brackets ([]), and curly brackets ({}) are also used in regular expressions.

To print all the lines from the *ls* command output that include either (-E) the pattern “cron” or “ly”. The pipe | character is used as an OR operator in this example. This is referred to as *alternation*. Regex allows you to add more patterns to this set if desired. For instance, if you search for three patterns, use ‘pattern1|pattern2|pattern3’. The patterns must be enclosed within single or double quotes. Here is what you will issue when you want to look for the patterns “cron” and “ly” in the */etc* directory listing:

```
[user1@server1 ~]$ ls -l /etc | grep -E 'cron|ly'
-rw-r--r--.  1 root root       541 Oct  2  2018 anacrontab
drwxr-xr-x.  2 root root        39 Aug 22 15:14 cron.d
drwxr-xr-x.  2 root root        36 Aug 22 15:14 cron.daily
-rw-r--r--.  1 root root         0 Oct  2  2018 cron.deny
drwxr-xr-x.  2 root root        22 Aug 22 15:09 cron.hourly
drwxr-xr-x.  2 root root         6 Aug 12  2018 cron.monthly
-rw-r--r--.  1 root root       451 Aug 12  2018 crontab
drwxr-xr-x.  2 root root         6 Aug 12  2018 cron.weekly
drwxr-xr-x.  2 root root        28 Aug 22 15:14 plymouth
```

To show all the lines from the */etc/ssh/sshd_config* file but exclude (-v) the empty lines and the rows that begin with the hash (#) character (commented lines). Using the -e flag multiple times as shown below is equivalent to how "-E 'pattern1|pattern2'" was used in the above example.

```
[user1@server1 ~]$ sudo grep -ve ^$ -ve ^# /etc/ssh/sshd_config
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
SyslogFacility AUTHPRIV
```

You can combine options and employ diverse regular expression criteria to perform complex matches on input. Issue **man 7 regex** in a terminal window to learn more about regex. Also consult the manual pages of the *grep* command to view available options and other details.

Running and Controlling Jobs in Foreground and Background

By default, any program or command you issue runs in the foreground and ties itself to the terminal window where it is initiated. It can be moved to continue to run in the background so that the associated terminal session is released for performing other tasks. A *job* is a process that is started in the background and controlled by the terminal where it is spawned. Just like any other process that is started on the system, a job is also assigned a PID (*process identifier*) by the kernel and, additionally, a job ID by the shell. Unlike a normal process, a job does not hold the terminal window where it is initiated. This enables you to run

other programs from the same terminal window. See Chapter 08 “Linux Processes and Task Scheduling” for details on processes and PIDs.

The shell allows running multiple jobs simultaneously, including transferring large amounts of data and running application programs in the background. Background jobs can be brought to foreground, returned to the background, suspended, or stopped. The management of multiple jobs within a shell environment is called *job control*.

The shell offers certain commands and control sequences for administering the jobs. See Table 7-4 for their description.

Command	Description
jobs	Shell built-in command to display jobs
bg	Shell built-in command to move a job to the background or restart a job in the background that was suspended with Ctrl+z
fg	Shell built-in command to move a job to the foreground
Ctrl+z	Suspends a foreground job and allows the terminal window to be used for other purposes

Table 7-4 Job Control Commands and Control Sequences

To run a job in the background, type the command at the command prompt followed by the ampersand (&) operator.

The examples below start the *top* and *vim* programs in the background. The shell displays their assigned job IDs (enclosed within square brackets) and PIDs. The job IDs allow us to control the jobs, and the PIDs are used by the kernel to manage the processes.

```
[user1@server1 ~]$ top &
[1] 31726
[user1@server1 ~]$ vim testfile1 &
[2] 31727

[1]+  Stopped                 top
```

Issue the *jobs* command with the -l switch to view all the jobs running in the background:

```
[user1@server1 ~]$ jobs -l
[1]- 31726 Stopped (signal)         top
[2]+ 31727 Stopped (tty output)     vim testfile1
```

The plus sign (+) indicates the current background job and the minus sign (-) signifies the previous job. “Stopped” implies that the jobs are currently suspended and can be signaled to continue their execution in the background with the *bg* command or to bring them back to the foreground with the *fg* command.

To bring job ID 1 to the foreground and start running it:

```
[user1@server1 ~]$ fg %1
```

To suspend job ID 1, press **^z** followed by **bg %1** to let it run in the background.

To terminate job ID 1, supply its PID (31726) to the *kill* command:

```
[user1@server1 ~]$ kill 31726
```

A message similar to the following appears when a background job ends its execution:

[2] + Done vi testfile1 &

See Chapter 08 “Linux Processes and Task Scheduling” for more information on the *kill* command.

Shell Startup Files

Earlier in this chapter, you used local and environment variables and learned how to modify the primary command prompt to add useful information to it. In other words, you modified the default shell environment to suit your needs. You stored the PS1 value in a shell startup file to ensure the changes are always available after you log off and log back in.

Modifications to the default shell environment can be stored in *startup* (or *initialization*) files. These files are sourced by the shell following user authentication at the time of logging in and before the command prompt appears. In addition, aliases, functions, and scripts can be added to these files as well. There are two types of startup files: *system-wide* and *per-user*.

System-wide Shell Startup Files

System-wide startup files set the general environment for all users at the time of their login to the system. These files are located in the */etc* directory and are maintained by the Linux administrator. System-wide files can be modified to include general environment settings and customizations.

Table 7-5 lists and describes system-wide startup files for bash shell users.

File	Comments
/etc/bashrc	Defines functions and aliases, sets umask for user accounts with a non-login shell, establishes the command prompt, etc. It may include settings from the shell scripts located in the /etc/profile.d directory.
/etc/profile	Sets common environment variables such as PATH, USER, LOGNAME, MAIL, HOSTNAME, HISTSIZE, and HISTCONTROL for all users, establishes umask for user accounts with a login shell, processes the shell scripts located in the /etc/profile.d directory, and so on.
/etc/profile.d	Contains scripts for bash shell users that are executed by the /etc/profile file.

Table 7-5 System-wide Startup Files

Excerpts from the *bashrc* and *profile* files and a list of files in the *profile.d* directory are displayed below:

```
[user1@server1 ~]$ head /etc/bashrc
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# It's NOT a good idea to change this file unless you know what you
# are doing. It's much better to create a custom.sh shell script in
# /etc/profile.d/ to make custom changes to your environment, as this
# will prevent the need for merging in future updates.


[user1@server1 ~]$ head /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

# It's NOT a good idea to change this file unless you know what you
# are doing. It's much better to create a custom.sh shell script in
# /etc/profile.d/ to make custom changes to your environment, as this
# will prevent the need for merging in future updates.

[user1@server1 ~]$ ls -l /etc/profile.d/
total 64
-rw-r--r--. 1 root root  664 Aug 13  2018 bash_completion.sh
-rw-r--r--. 1 root root   80 Sep 10  2018 csh.local
-rw-r--r--. 1 root root  294 Feb 14  2019 flatpak.sh
-rw-r--r--. 1 root root 1107 Dec 14  2017 gawk.csh
-rw-r--r--. 1 root root  757 Dec 14  2017 gawk.sh
-rw-r--r--. 1 root root 2291 Sep 10  2018 lang.csh
-rw-r--r--. 1 root root 2276 Sep 10  2018 lang.sh
-rw-r--r--. 1 root root  500 Aug 12  2018 less.csh
```

Any file in the *profile.d* directory can be edited and updated. Alternatively, you can create your own file and define any customization that you want.

Per-user Shell Startup Files

Per-user shell startup files override or modify system default definitions set by the system-wide startup files. These files may be customized by individual users to suit their needs. By default, two such files, in addition to the *.bash_logout* file, are located in the skeleton directory */etc/skel* and are copied into user home directories at the time of user creation.

You may create additional files in your home directories to set more environment variables or shell properties if required.

Table 7-6 lists and describes per-user startup files for bash shell users.

File	Comments
.bashrc	Defines functions and aliases. This file sources global definitions from the /etc/bashrc file.
.bash_profile	Sets environment variables and sources the .bashrc file to set functions and aliases.
.gnome2/	Directory that holds environment settings when GNOME desktop is started. Only available if GNOME is installed.

Table 7-6 Per-user Startup Files

Excerpts from the *.bashrc* and *.bash_profile* files are exhibited below:

```
[user1@server1 ~]$ cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

[user1@server1 ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
```

The order in which the system-wide and per-user startup files are executed is important to grasp. The system runs the */etc/profile* file first, followed by *.bash_profile*, *.bashrc*, and finally the */etc/bashrc* file.

There is also a per-user file *.bash_logout* in the user's home directory. This file is executed when the user leaves the shell or logs off. This file may be customized as well.

Chapter Summary

In this chapter, we explored the bash shell, which has numerous features that are essential for users and administrators alike. We touched upon a few that are more prevalent. These features included variable settings, command prompt customization using substitution techniques, input/output/error redirections, history expansion, command line editing, filename completion, tilde substitution, and command aliasing. Moreover, we looked at the difference between the shell's internal and external commands.

We continued to examine additional bash shell features and expanded on metacharacters, wildcard characters, pipe symbol, and quoting mechanisms. We demonstrated an example of building a pipeline by incorporating multiple pipe characters between commands.

Finally, we analyzed various system-wide and per-user shell initialization scripts that are executed upon logging in to set a user environment. These scripts may be customized for all users or individual users to suit specific needs.

Check Your Understanding

1. You want to change the command prompt for yourself. Where (the bash shell file) would you define it so that you get the custom command prompt whenever you log in?
2. What is the other name for the command line completion feature?

3. What is a common use of the pipe symbol?
4. Which directory location stores most, if not all, privileged commands?
5. You have a file called “?” in your home directory along with other one-letter files. What would you run to delete the “?” file only?
6. What would the command *ls /cdr /usr > output* do? Assume */cdr* does not exist.
7. What is the name of the command that allows a user to define shortcuts to lengthy commands?
8. Which file typically defines the history variables for all users?
9. You have a command running that is tied to your terminal window. You want to get the command prompt back without terminating the running command. Which key combination would you press to return to the command prompt?
10. Name the three quoting mechanisms?
11. What would the command export *VAR1=”I passed RHCSA”* do?
12. Name the default filename and location where user command history is stored?
13. What is the primary function of the shell in Linux?
14. What would the command *cd ~user20* do if executed as *user10*?
15. Which command would you use to display all matching lines from a text file?
16. What would the command *ls > ls.out* do?

Answers to Check Your Understanding

1. You can define it in the *~/.bash_profile* file.
2. The other name for the command line completion feature is tab completion.

3. A common use of the pipe character is to receive the output of one command and pass it along to the next command as an input.
4. Privileged commands are stored in */usr/sbin* directory.
5. You can issue *rm \?* to remove the file.
6. The command provided will redirect *ls /usr* output to the specified file and *ls /cdr* (error) to the terminal.
7. The *alias* command.
8. */etc/profile* is the file where the history variables are defined for all users.
9. The Ctrl+z key combination will suspend the running program and give you access to the command prompt.
10. The three quoting mechanisms are backslash, single quotation mark, and double quotation mark.
11. The command provided will set an environment variable with the quoted string as its value.
12. The user command history is stored in *~/.bash_history* file.
13. The shell acts as an interface between a user and the system.
14. The command provided will allow *user10* to *cd* into *user20*'s home directory provided *user10* has proper access permissions.
15. The *grep* command.
16. The command provided will redirect the *ls* command output to the specified file.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 7-1: Customize the Command Prompt

As *user1* on *server1*, customize the primary shell prompt to display the information enclosed within the quotes “<user1@server1 in /etc >:” when this user switches into the */etc* directory. The prompt should always reflect the current directory path. Add this to the appropriate per-user startup file for permanence. (Hint: Command and Variable Substitutions).

Lab 7-2: Redirect the Standard Input, Output, and Error

As *user1* on *server1*, run the *ls* command on */etc*, */dvd*, and */var*. Have the output printed on the screen as well as redirected to file */tmp/ioutput*, and the errors forwarded to file */tmp/ioerror*. Check both files after the execution of the command and analyze the results. (Hint: Input, Output, and Error Redirections).

Chapter 08

Linux Processes and Task Scheduling

This chapter describes the following major topics:

- Identify and display system and user executed processes
- View process states and priorities
- Examine and change process niceness and priority
- Understand signals and their use in controlling processes
- Review job scheduling
- Control who can schedule jobs
- Schedule and manage jobs using at
- Comprehend crontab and understand the syntax of crontables
- Schedule and manage jobs using cron
- Understand anacron and its function

RHCSA Objectives:

15. Identify CPU/memory intensive processes, adjust process priority with renice, and kill processes
16. Adjust process scheduling
35. Schedule tasks using at and cron

A process is any program, command, or application running on the system. Every process has a unique numeric identifier and it is managed by the kernel through its entire lifespan. It may be viewed, listed, and monitored, and can be launched at a non-default priority based on the requirement and available computing resources. A process may also be re-prioritized while it is running. A process is in one of several states at any given time during its lifecycle. A process may be tied to the terminal window where it is initiated, or it may run on the system as a service.

There are plenty of signals that may be passed to a process to accomplish various actions. These actions include hard killing a process, soft terminating it, running it as a background job, bringing the background job back to the foreground, and forcing it to restart with the same identifier.

Job scheduling allows a user to schedule a command for a one-time or recurring execution in the future. A job is submitted and managed by authorized users only. All executed jobs are logged. Anacron is a service that automatically runs jobs that were missed while the system was down.

Processes and Priorities

A *process* is a unit for provisioning system resources. It is any program, application, or command that runs on the system. A process is created in memory when a program, application, or command is initiated. Processes are organized in a hierarchical fashion. Each process has a *parent process* (a.k.a. a *calling process*) that spawns it. A single parent process may have one or many *child processes* and passes many of its

attributes to them at the time of their creation. Each process is assigned an exclusive identification number known as the *Process IDentifier* (PID), which is used by the kernel to manage and control the process through its lifecycle. When a process completes its lifespan or is terminated, this event is reported back to its parent process, and all the resources provisioned to it (cpu cycles, memory, etc.) are then freed and the PID is removed from the system.

Plenty of processes are spawned at system boot, many of which sit in the memory and wait for an event to trigger a request to use their services. These background system processes are called *daemons* and are critical to system operation.

Process States

A process changes its operating state multiple times during its lifecycle. Many factors, such as load on the processor, availability of free memory, priority of the process, and response from other applications, affect how often a process jumps from one operating state to another. It may be in a non-running condition for a while or waiting for other process to feed it information so that it can continue to run.

There are five basic process states: *running*, *sleeping*, *waiting*, *stopped*, and *zombie*. Each process is in one state at any given time. See Figure 8-1.

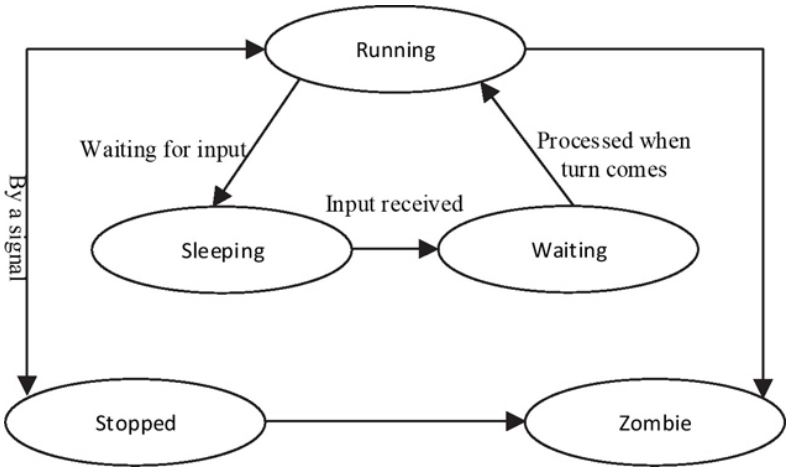


Figure 8-1 Process State Transition

Running: The process is being executed by the system CPU.
Sleeping: The process is waiting for input from a user or another process.
Waiting: The process has received the input it was waiting for and is now ready to run as soon as its turn comes.
Stopped: The process is currently halted and will not run even when its turn comes unless a signal is sent to change its behavior. (Signals are explained later in this chapter.)
Zombie: The process is dead. A zombie process exists in the process table alongside other process entries, but it takes up no resources. Its entry is retained until its parent process permits it to die. A zombie process is also called a *defunct* process.

Viewing and Monitoring Processes with ps

A system may have hundreds or thousands of processes running concurrently depending on the purpose of the system. These processes may be viewed and monitored using various native tools such as *ps* (*process status*) and *top* (*table of processes*). The *ps* command offers plentiful switches that influence its output, whereas *top* is used for real-time viewing and monitoring of processes and system resources.

Without any options or arguments, *ps* lists processes specific to the terminal where this command is issued:

```
[user1@server1 ~]$ ps
  PID TTY          TIME CMD
 1253 pts/0    00:00:00 ps
18359 pts/0    00:00:00 bash
```

The above output returns the elementary information about processes in four columns. These processes are tied to the current terminal window. It exhibits the PID, the terminal (TTY) the process spawned in, the cumulative time (TIME) the system CPU has given to the process, and the name of the actual command or program (CMD) being executed.

Some common options that can be used with the *ps* command to generate detailed reports include -e (every), -f (full-format), -F (extra full-format), and -l (long format). A combination of -e, -F, and -l (**ps -eFl**) produces a very thorough process report, however, that much detail may not be needed in most situations. Other common options such as --forest and -x will report the output in tree-like hierarchy and include the daemon processes as well. Check the manual pages of the command for additional options and their usage.

Here are a few sample lines from the beginning and end of the output when *ps* is executed with -e and -f flags on the system.

```
[user1@server1 ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Sep14 ?        00:00:06 /usr/lib/systemd/systemd --switched-root
root         2     0  0 Sep14 ?        00:00:00 [kthreadd]
root         3     2  0 Sep14 ?        00:00:00 [rcu_gp]
root         4     2  0 Sep14 ?        00:00:00 [rcu_par_gp]
root         6     2  0 Sep14 ?        00:00:00 [kworker/0:0H-kblockd]
root         8     2  0 Sep14 ?        00:00:00 [mm_percpu_wq]
root         9     2  0 Sep14 ?        00:00:01 [ksoftirqd/0]
```

This output is disseminated across eight columns showing details about every process running on the system. Table 8-1 describes the content type of each column.

Column	Description
UID	User ID or name of the process owner
PID	Process ID of the process
PPID	Process ID of the parent process
C	CPU utilization for the process
STIME	Process start date or time
TTY	The controlling terminal the process was started on. “Console” represents the system console and “?” represents a daemon process.
TIME	Aggregated execution time for a process
CMD	The command or program name

Table 8-1 ps Command Output Description

The *ps* output above pinpoints several daemon processes running in the background. These processes are not associated with any terminal, which is why there is a ? in the TTY column. Notice the PID and PPID numbers. The smaller the number, the earlier it is started. The process with PID 0 is started first at system boot, followed by the process with PID 1, and so on. Each PID has an associated PPID in column 3. The owner of each process is exposed in the UID column along with the name of the command or program under CMD.

Information for each running process is recorded and maintained in the */proc* file system, which *ps* and many other commands reference to acquire desired data for viewing.

The *ps* command output may be customized to view only the desired columns. For instance, if you want to produce an output with the command name in column 1, PID in column 2, PPID in column 3, and owner name in column 4, run it as follows:

```
[user1@server1 ~]$ ps -o comm,pid,ppid,user
COMMAND           PID  PPID USER
ps               1327 18359 user1
bash            18359 18354 user1
```

Make sure the -o option is specified for a user-defined format and there is no white space before or after the column names. You can add or remove columns and switch their positions as needed.

Another switch to look at with the *ps* command is -C (command list). This option is used to list only those processes that match the specified command name. For example, run it to check how many *sshd* processes are currently running on the system:

```
[user1@server1 ~]$ ps -C sshd
  PID TTY          TIME CMD
  877 ?        00:00:00 sshd
17771 ?        00:00:00 sshd
17790 ?        00:00:00 sshd
18101 ?        00:00:00 sshd
18105 ?        00:00:00 sshd
18350 ?        00:00:01 sshd
18354 ?        00:00:02 sshd
```

The output exhibits multiple background *sshd* processes.

Viewing and Monitoring Processes with top

The other popular tool for viewing process information is the *top* command. This command displays statistics in real time and continuously, and may be helpful in identifying possible performance issues on the system. A sample of a running *top* session is shown below:

```
[user1@server1 ~]$ top
top - 15:50:11 up 4 days, 19:27,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 153 total,   2 running, 151 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  0.3 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    821.4 total,     87.6 free,    433.7 used,    300.1 buff/cache
MiB Swap:   1024.0 total,   1014.2 free,      9.8 used.    247.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1029 gdm       20   0 2381764 170824  81688 S   1.0  20.3   4:56.46 gnome-shell
 1387 user1     20   0   63876   4380   3752 R   0.7   0.5   0:00.03 top
18354 user1     20   0  167968   5940   4640 S   0.3   0.7   0:02.08 sshd
    1 root      20   0  244596  13076   8924 S   0.0   1.6   0:06.31 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.05 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblo+
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0   0.0   0:01.20 ksoftirqd/0
   10 root      20   0       0      0      0 R   0.0   0.0   0:01.55 rcu_sched
   11 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/0
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.20 watchdog/0
   13 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
```

Press q or Ctrl+c to quit.

The *top* output may be divided into two major portions: the summary portion and the tasks portion. The summary area spreads over the first five lines of the output, and it shows the information as follows:

Line 1: Indicates the system uptime, number of users logged in, and system load averages over the period of 1, 5, and 15 minutes. See the description for the *uptime* command output in Chapter 02 “Initial Interaction with the System”.
Line 2: Displays the task (or process) information, which includes the total number of tasks running on the system and how many of them are in running, sleeping, stopped, and zombie states.
Line 3: Shows the processor usage that includes the CPU time in percentage spent in running user and system processes, in idling and waiting, and so on.
Line 4: Depicts memory utilization that includes the total amount of memory allocated to the system, and how much of it is free, in use, and allocated for use in buffering and caching.

Line 5: Exhibits swap (virtual memory) usage that includes the total amount of swap allocated to the system, and how much of it is free and in use. The “avail Mem” shows an estimate of the amount of memory available for starting new processes without using the swap.

The second major portion in the *top* command output showcases the details for each process in 12 columns as described below:

Columns 1 and 2: Pinpoint the process identifier (PID) and owner (USER)
Columns 3 and 4: Display the process priority (PR) and nice value (NI)
Columns 5 and 6: Depict amounts of virtual memory (VIRT) and non-swapped resident memory (RES) in use
Column 7: Shows the amount of shareable memory available to the process (SHR)
Column 8: Represents the process status (S)
Columns 9 and 10: Express the CPU (%CPU) and memory (%MEM) utilization
Column 11: Exhibits the CPU time in hundredths of a second (TIME+)
Column 12: Identifies the process name (COMMAND)

While in *top*, you can press “o” to re-sequence the process list, “f” to add or remove fields, “F” to select the field to sort on, and “h” to obtain help. *top* is highly customizable. See the command’s manual pages for details.

Listing a Specific Process

Though the tools discussed so far provide a lot of information about processes including their PIDs, Linux also offers the *pidof* and *pgrep* commands to list only the PID of a specific process. These commands have a few switches available to modify their behavior; however, their most elementary use is to pass a process name as an argument to view its PID. For

instance, to list the PID of the *rsyslogd* daemon, use either of the following:

```
[user1@server1 ~]$ pidof rsyslogd
1271
[user1@server1 ~]$ pgrep rsyslogd
1271
```

Both commands produce an identical result if used without an option.

Listing Processes by User and Group Ownership

A process can be listed by its ownership or owning group. You can use the *ps* command for this purpose. For example, to list all processes owned by *user1*, specify the -U (or -u) option with the command and then the username:

```
[user1@server1 ~]$ ps -U user1
  PID TTY          TIME CMD
 1733 pts/0    00:00:00 ps
17777 ?        00:00:00 systemd
17784 ?        00:00:00 (sd-pam)
17790 ?        00:00:00 sshd
18105 ?        00:00:00 sshd
18354 ?        00:00:03 sshd
18359 pts/0    00:00:00 bash
```

The command lists the PID, TTY, TIME, and CMD name for all the processes owned by *user1*. You can specify the -G (or -g) option instead and the name of an owning group to print processes associated with that group only:

```
[user1@server1 ~]$ ps -G root
  PID TTY          TIME CMD
    1 ?        00:00:06 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    6 ?        00:00:00 kworker/0:0H-kblockd
```

The above output reveals all the running processes with *root* as their owning group.

Understanding Process Niceness

Linux is a multitasking operating system. It runs numerous processes on a single processor core by giving each process a slice of time. The process scheduler on the system performs rapid switching of processes, giving the notion of concurrent execution of multiple processes.

A process is spawned at a certain priority, which is established at initiation based on a numeric value called *niceness* (a.k.a. a *nice* value). There are 40 niceness values, with -20 being the highest or the most favorable to the process, and +19 being the lowest or the least favorable to the process. Most system-started processes run at the default niceness of 0. A higher niceness lowers the execution priority of a process, and a lower niceness increases it. In other words, a process running at a higher priority gets more CPU attention. A child process inherits the niceness of its calling process in its priority calculation. Though programs are normally run at the default niceness, you can choose to initiate them at a different niceness to adjust their priority based on urgency, importance, or system load. As a normal user, you can only make your processes nicer, but the *root* user can raise or lower the niceness of any process.

Viewing and Changing Process Niceness

The current process priorities and niceness values can be viewed with the *ps* or the *top* command. With *ps*, add the -l option to -ef and look for the priority (PRI, column 7), which is calculated based on the current niceness value (NI, column 8):

```
[user1@server1 ~]$ ps -efl
[user1@server1 ~]$ ps -efl
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     0  0  80   0 - 61149 -      Sep14 ?        00:00:06 /usr/lib/syst
1 S root         2     0  0  80   0 -     0 -      Sep14 ?        00:00:00 [kthreadd]
1 I root         3     2  0  60 -20 -     0 -      Sep14 ?        00:00:00 [rcu_gp]
1 I root         4     2  0  60 -20 -     0 -      Sep14 ?        00:00:00 [rcu_par_gp]
1 I root         6     2  0  60 -20 -     0 -      Sep14 ?        00:00:00 [kworker/0:0H
```

The above example indicates the use of the default niceness (0) for the first two processes and the highest niceness of -20

for the next three. These values are used by the process scheduler to adjust the execution time of the processes on the CPU. The *ps* command maintains an internal mapping between niceness levels and priorities. A niceness of 0 (NI column) corresponds to priority 80 (PR column), and a niceness of -20 (NI column) maps to priority 60 (PR column).

In contrast to the niceness-priority mapping that the ps command uses, the top command displays it differently. For a 0-80 ps mapping, the top session will report it 0-20. Likewise, ps' (-20)-60 will be the same as the top's (-20)-0.

Check the default niceness using the *nice* command:

```
[user1@server1 ~]$ nice
0
```

As reported, the default niceness is 0 for the bash shell process running for *user1*. This niceness level was inherited from the parent process.

A different niceness may be assigned to a program or command at its startup. To run the *top* command at a lower priority with a nice value of +2:

```
[user1@server1 ~]$ nice -n 2 top
```

Open another terminal window and run the *ps* command to validate. Look for the priority and nice values in columns 7 and 8.

```
[user1@server1 ~]$ ps -el | grep top
0 S  1000  5297 18359  0  82   2 - 15969 core_s pts/0    00:00:00 top
```

Based on the internal mapping that the *ps* command uses, the priority and niceness are 82 and 2, respectively.

Let's terminate the *top* session in the first terminal window by pressing the letter q and relaunch it at a higher priority with a

niceness of -10. You must be *root* or have the *root* privilege to start a program at a higher priority.

```
[user1@server1 ~]$ sudo nice -n -10 top
```

Validate in the other window:

```
[user1@server1 ~]$ ps -el | grep top
4 S     0  5572  5570  0  70 -10 - 15969 -      pts/0    00:00:00 top
```

As you can see, the process is running at a higher priority (70) with a nice value of -10.

Renicing a Running Process

The niceness of a running process may be altered using the *renice* command. This adjustment affects the priority at which the process is currently running. For example, to change the niceness of the running *top* session from -10 to +5, specify the PID (5572 from column 4 above) with the *renice* command. You need to prepend *sudo* with the command.

```
[user1@server1 ~]$ sudo renice 5 5572
5572 (process ID) old priority -10, new priority 5
```

Validate the above change with *ps*. See columns 7 and 8.

```
[user1@server1 ~]$ ps -el | grep top
4 S     0  5572  5570  0  85   5 - 15969 -      pts/0    00:00:00 top
```

The *renice* command can also alter the nice values of all the processes owned by a specific user or members of a particular group by specifying the -u or -g option. There are a few additional options available with this command. Refer to the command’s manual pages for usage help.

You can also renice a process from within the *top* command window by pressing the letter “r” and entering the PID of the process you want reniced and the desired nice level.

Controlling Processes with Signals

A system may have hundreds or thousands of processes running on it. Sometimes it becomes necessary to alert a process of an event. This is done by sending a control signal to the process. Processes may use signals to alert each other as well. The receiving process halts its execution as soon as it gets the signal and takes an appropriate action as per the instructions enclosed in the signal. The instructions may include terminating the process gracefully, killing it abruptly, or forcing it to re-read its configuration.

There are plentiful signals available for use, but only a few are common. Each signal is associated with a unique numeric identifier, a name, and an action. A list of available signals can be viewed with the *kill* command using the -l option:

```
[user1@server1 ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

The output returns 64 signals available for process-to-process and user-to-process communication. Table 8-2 describes the control signals that are most often used.

Signal Number	Signal Name	Action
1	SIGHUP	Hang up signal causes a process to disconnect itself from a closed terminal that it was tied to. Also used to instruct a running daemon to re-read its configuration without a restart.
2	SIGINT	The ^c (Ctrl+c) signal issued on the controlling terminal to interrupt the execution of a process.
9	SIGKILL	Terminates a process abruptly
15	SIGTERM	Sends a soft termination signal to stop a process in an orderly fashion. This is the default signal if none is specified with the command.
18	SIGCONT	Same as using the bg command to resume
19	SIGSTOP	Same as using Ctrl+z to suspend a job
20	SIGTSTP	Same as using the fg command

Table 8-2 Control Signals

The commands used to pass a signal to a process are *kill* and *pkill*. These commands are usually used to terminate a process. Ordinary users can kill processes that they own, while the *root* user privilege is needed to kill any process on the system.

The *kill* command requires one or more PIDs, and the *pkill* command requires one or more process names to send a signal to. You can specify a non-default signal name or number with either utility.

Let’s look at a few examples to understand the usage of these tools.

To pass the soft termination signal to the *crond* daemon, use either of the following:

```
[user1@server1 ~]$ sudo pkill crond
[user1@server1 ~]$ sudo kill $(pidof crond)
```

The *pidof* command in the above example is used to discover the PID of the *crond* process using command substitution and it is then passed to the *kill* command for termination. You may also use the *pgrep* command to determine the PID of a process, as demonstrated in the next example. Use **ps -ef | grep crond** to confirm the termination.

Using the *pkill* or *kill* command without specifying a signal name or number sends the default signal of 15 to the process. This signal may or not terminate the process. Some processes ignore the soft termination signal as they might be in a waiting state. These processes may be ended forcefully using signal 9 in any of the following ways:

```
[user1@server1 ~]$ sudo pkill -9 crond
[user1@server1 ~]$ sudo pkill -s SIGKILL crond
[user1@server1 ~]$ sudo kill -9 $(pgrep crond)
```

You may run the *killall* command to terminate all processes that match a criterion. Here is how you can use this command to kill all *crond* processes (assuming there are many of them running):

```
[user1@server1 ~]$ sudo killall crond
```

There are plenty of options available with the *kill*, *killall*, *pkill*, *pgrep*, and *pidof* commands. Consult respective manual pages for more details.

Job Scheduling

Job scheduling allows a user to submit a command for execution at a specified time in the future. The execution of the command could be one time or periodic based on a pre-determined time schedule. A one-time execution may be scheduled for an activity that needs to be performed at a time of low system usage. One example of such an activity would

be the execution of a lengthy shell program. In contrast, a recurring activity could include creating a compressed archive, trimming log files, monitoring the system, running a custom script, or removing unwanted files from the system.

Job scheduling and execution is taken care of by two service daemons: *atd* and *crond*. While *atd* manages the jobs scheduled to run one time in the future, *crond* is responsible for running jobs repetitively at pre-specified times. At startup, this daemon reads the schedules in files located in the */var/spool/cron* and */etc/cron.d* directories, and loads them in the memory for on-time execution. It scans these files at short intervals and updates the in-memory schedules to reflect any modifications. This daemon runs a job at its scheduled time only and does not entertain any missed jobs. In contrast, the *atd* daemon retries a missed job at the same time next day. For any additions or changes, neither daemon needs a restart.

Controlling User Access

By default, all users are allowed to schedule jobs using the at and cron services. However, this access may be controlled and restricted to specific users only. This can be done by listing users in the allow or deny file located in the */etc* directory for either service. These files are named *at.allow* and *at.deny* for the at service, and *cron.allow* and *cron.deny* for the cron service.

The syntax for the four files is identical. You only need to list usernames that are to be allowed or denied access to these scheduling tools. Each file takes one username per line. The *root* user is always permitted; it is affected neither by the existence or non-existence of these files, nor by the inclusion or exclusion of its entry in these files.

Table 8-3 shows various combinations and their impact on user access.

at.allow / cron.allow	at.deny / cron.deny	Impact
Exists, and contains user entries	Existence does not matter	All users listed in allow files are permitted
Exists, but is empty	Existence does not matter	No users are permitted
Does not exist	Exists, and contains user entries	All users, other than those listed in deny files, are permitted
Does not exist	Exists, but is empty	All users are permitted
Does not exist	Does not exist	No users are permitted

Table 8-3 User Access Restrictions to Scheduling Tools

By default, the *deny* files exist and are empty, and the *allow* files are non-existent. This opens up full access to using both tools for all users.

> **EXAM TIP:** One username is entered per line entry in an appropriate allow or deny file.

The following message appears if an unauthorized user attempts to execute *at*:

```
You do not have permission to use at.
```

And the following warning is displayed for unauthorized access attempt to the cron service:

```
You (user1) are not allowed to use this program (crontab)
See crontab(1) for more information
```

To generate the denial messages, you need to place entries for *user1* in the *deny* files.

Scheduler Log File

All activities for *atd* and *crond* services are logged to the */var/log/cron* file. Information such as the time of activity,

hostname, process name and PID, owner, and a message for each invocation is captured. The file also keeps track of other events for the *crond* service such as the service start time and any delays. A few sample entries from the log file are shown below:

```
[user1@server1 ~]$ sudo cat /var/log/cron
Sep 15 03:29:01 server1 run-parts[8206]: (/etc/cron.daily) finished logrotate
Sep 15 03:29:01 server1 run-parts[8206]: (/etc/cron.daily) starting rhsmd
Sep 15 03:29:02 server1 run-parts[8206]: (/etc/cron.daily) finished rhsmd
Sep 15 03:29:02 server1 anacron[7971]: Job `cron.daily' terminated
Sep 15 03:29:02 server1 anacron[7971]: Normal exit (1 job run)
Sep 15 04:01:01 server1 CROND[8501]: (root) CMD (run-parts /etc/cron.hourly)
Sep 15 04:01:01 server1 run-parts[8501]: (/etc/cron.hourly) starting 0anacron
Sep 15 04:01:01 server1 run-parts[8501]: (/etc/cron.hourly) finished 0anacron
```

The truncated output shows some past entries from the file. You need the *root* user privilege to be able to read the file content.

Using at

The *at* command is used to schedule a one-time execution of a program in the future. All submitted jobs are spooled in the */var/spool/at* directory and executed by the *atd* daemon at the specified time. Each submitted job will have a file created containing the settings for establishing the user's shell environment to ensure a successful execution. This file also includes the name of the command or program to be run. There is no need to restart the daemon after a job submission.

There are multiple ways and formats for expressing the time with *at*. Some examples are:

at 1:15am	(executes the task at the next 1:15 a.m.)
at noon	(executes the task at 12:00 p.m.)
at 23:45	(executes the task at 11:45 p.m.)
at midnight	(executes the task at 12:00 a.m.)
at 17:05 tomorrow	(executes the task at 5:05 p.m. on the next da
at now + 5 hours	(executes the task 5 hours from now. We can specify minutes, days, or weeks in place of hours)
at 3:00 10/15/20	(executes the task at 3:00 a.m. on October 15,

2020)

at assumes the current year and today's date if the year and date are not mentioned.

You may supply a filename with the *at* command using the -f option. The command will execute that file at the specified time. For instance, the following will run */home/user1/.bash_profile* file for *user1* 2 hours from now:

```
[user1@server1 ~]$ at -f ~/.bash_profile now + 2 hours
warning: commands will be executed using /bin/sh
job 4 at Fri Sep 20 10:31:00 2019
```

The above will be executed as scheduled and will have an entry placed for it in the log file.

Exercise 8-1: Submit, View, List, and Remove an at Job

This exercise should be done on *server1* as *user1*.

In this exercise, you will submit an at job as *user1* to run the *date* command at 11:30 p.m. on March 31, 2020, and have the output and any error messages generated redirected to the */tmp/date.out* file. You will list the submitted job, exhibit its contents for verification, and then remove the job.

1. Run the *at* command and specify the correct execution time and date for the job. Type the entire command at the first at> prompt and press Enter. Press Ctrl+d at the second at> prompt to complete the job submission and return to the shell prompt.

```
[user1@server1 ~]$ at 11:30pm 3/31/20
warning: commands will be executed using /bin/sh
at> date &> /tmp/date.out
at> <EOT>
job 5 at Tue Mar 31 23:30:00 2020
```

The system assigned job ID 5 to it, and the output also pinpoints the job's execution time.

2. List the job file created in the */var/spool/at* directory:

```
[user1@server1 ~]$ sudo ls -l /var/spool/at/
total 4
-rwx------. 1 user1 user1 1449 Sep 20 08:39 a0000501934472
```

3. List the spooled job with the *at* command. You may alternatively use **atq** to list it.

```
[user1@server1 ~]$ at -l
5       Tue Mar 31 23:30:00 2020 a user1
```

4. Display the contents of this file with the *at* command and specify the job ID:

```
[user1@server1 ~]$ at -c 5
#!/bin/sh
# atrun uid=1000 gid=1000
# mail user1 0
umask 2
SSH_CONNECTION=192.168.0.219\ 53248\ 192.168.0.110\ 22; export SSH_CONNECTION
LANG=en_US.UTF-8; export LANG
HISTCONTROL=ignoredups; export HISTCONTROL
HOSTNAME=server1.example.com; export HOSTNAME
OLDPWD=/usr/share/dict; export OLDPWD
XDG_SESSION_ID=15; export XDG_SESSION_ID
USER=user1; export USER
```

5. Remove the spooled job with the *at* command by specifying its job ID. You may alternatively run **atrm 5** to delete it.

```
[user1@server1 ~]$ at -d 5
```

This should remove the job file from the */var/spool/at* directory. You can confirm the deletion by running **atq** or **at -l**.

Using crontab

Using the *crontab* command is the other method for scheduling tasks for running in the future. Unlike *atd*, *crond* executes cron jobs on a regular basis if they comply with the format defined in the */etc/crontab* file. Crontables (another name for crontab files) are located in the */var/spool/cron* directory. Each authorized user with a scheduled job has a file matching their login name in this directory.

For example, the crontable for *user1* would be */var/spool/cron/user1*. The other two locations where system crontables can be stored are the */etc/crontab* file and the */etc/cron.d* directory; however, only the *root* user is allowed to create, modify, and delete them. The *crond* daemon scans entries in the files at the three locations to determine job execution schedules. The daemon runs the commands or programs at the specified time and adds a log entry to the */var/log/cron* file for each invocation. There is no need to restart the daemon after submitting a new or modifying an existing cron job.

The *crontab* command is used to edit (-e), list (-l), and remove (-r) crontables. The -u option is also available for users who wish to modify a different user's crontable, provided they are allowed to do so and the other user is listed in the *cron.allow* file. The *root* user can also use the -u flag to alter other users' crontables even if the affected users are not listed in the *allow* file. By default, crontab files are opened in the *vim* editor when the *crontab* command is issued to edit them.

Syntax of User Crontables

The */etc/crontab* file specifies the syntax that each user cron job must comply with in order for *crond* to interpret and execute it successfully. Based on this structure, each line in a user crontable with an entry for a scheduled job is comprised of six fields. Fields 1 to 5 are for the schedule, field 6 may contain the login name of the executing user, and the rest for the command or program to be executed. See Figure 8-2 for the syntax.

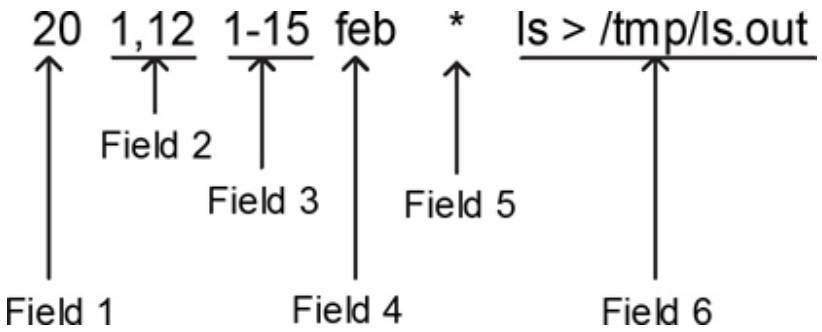


Figure 8-2 Syntax of Crontables

A description of each field is provided in Table 8-4.

Field	Field Content	Description
1	Minute of the hour	Valid values are 0 (the exact hour) to 59. This field can have one specific value as in field 1, multiple comma-separated values as in field 2, a range of values as in field 3, a mix of fields 2 and 3 (1-5,6-19), or an * representing every minute of the hour as in field 5.
2	Hour of the day	Valid values are 0 (midnight) to 23. Same usage applies as described for field 1.
3	Day of the month	Valid values are 1 to 31. Same usage applies as described for field 1.
4	Month of the year	Valid values are 1 to 12 or jan to dec. Same usage applies as described for field 1.
5	Day of the week	Valid values are 0 to 7 or sun to sat, with 0 and 7 representing Sunday, 1 representing Monday, and so on. Same usage applies as described for field 1.
6	Command or program to execute	Specifies the full path name of the command or program to be executed, along with any options or arguments that it requires.

Table 8-4 Crontable Syntax Explained

Furthermore, step values may be used with * and ranges in the crontables using the forward slash character (/). Step

values allow the number of skips for a given value. For example, */2 in the minute field would mean every second minute, */3 would mean every third minute, 0-59/4 would mean every fourth minute, and so on. Step values are also supported in the same format in fields 2 to 5.

> **EXAM TIP:** Make sure you understand and memorize the order of the fields defined in crontables.

Consult the manual pages of the *crontab* configuration file (**man 5 crontab**) for more details on the syntax.

Exercise 8-2: Add, List, and Remove a Cron Job

This exercise should be done on *server1* as *user1* with *sudo* where required.

For this exercise, assume that all users are currently denied access to cron.

In this exercise, you will submit a cron job as *user1* to echo “Hello, this is a cron test.”. You will schedule this command to execute at every fifth minute past the hour between 10:00 a.m. and 11:00 a.m. on the fifth and twentieth of every month. You will have the output redirected to the */tmp/hello.out* file, list the cron entry, and then remove it.

1. Edit the */etc/cron.allow* file and add *user1* to it:

```
[user1@server1 ~]$ sudo vim /etc/cron.allow
user1
```

2. Open the crontable and append the following schedule to it. Save the file when done and exit out of the editor.

```
[user1@server1 ~]$ crontab -e
*/5 10-11 5,20 * * echo "Hello, this is a cron test." > /tmp/hello.out
```

3. Check for the presence of a new file by the name *user1* under the */var/spool/cron* directory:

```
[user1@server1 ~]$ sudo ls -l /var/spool/cron/
total 4
-rw-------. 1 user1 user1 71 Sep 20 10:16 user1
```

4. List the contents of the crontable:

```
[user1@server1 ~]$ crontab -l
*/5 10-11 5,20 * * echo "Hello, this is a cron test." > /tmp/hello.out
```

5. Remove the crontable and confirm the deletion:

```
[user1@server1 ~]$ crontab -r
[user1@server1 ~]$ crontab -l
no crontab for user1
```

Do not run **crontab -r** if you do not wish to remove the crontab file. Instead, edit the file with **crontab -e** and just remove the entry.

Anacron

Anacron is a service that runs after every system reboot. It checks for any cron and at jobs that were scheduled for execution during the time the system was down and were missed as a result. Anacron proves to be useful on laptop, desktop, and similar purpose systems with extended periods of frequent downtimes and are not intended for 24/7 operations. Anacron scans the */etc/cron.hourly/0anacron* file for three factors to learn whether to run missed jobs. The three factors it examines are: (1) the presence of the */var/spool/anacron/cron.daily* file, (2) the elapsed time of 24 hours since it was last run, and (3) if the system is plugged in to an AC source. If all three conditions are true, Anacron automatically executes the scripts located in the */etc/cron.daily*, */etc/cron.weekly*, and */etc/cron.monthly* directories based on the settings and conditions defined in Anacron's own configuration file */etc/anacrontab*. The default content of the */etc/anacrontab* file are displayed below. It excludes the commented and empty lines.

```
[user1@server1 ~]$ cat /etc/anacrontab | grep -ve ^# -ve ^$
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
RANDOM_DELAY=45
START_HOURS_RANGE=3-22
1       5       cron.daily              nice run-parts /etc/cron.daily
7       25      cron.weekly             nice run-parts /etc/cron.weekly
@monthly 45     cron.monthly            nice run-parts /etc/cron.monthly
```

This file has five variables defined as depicted at the beginning of the above output. SHELL and PATH set the shell and path to be used for executing the programs. MAILTO defines the login name or an email of the user who is to be sent any output and error messages. RANDOM_DELAY expresses the maximum arbitrary delay in minutes added to the base delay of the jobs as defined in column 2 of the last three lines. START_HOURS_RANGE states the hour duration within which the missed jobs could be run.

The bottom three lines (each line with six columns) define the schedule and the programs to be executed:

Column 1: Denotes the period in days (or @daily, @weekly, @monthly, or @yearly), which Anacron uses to determine how often to run the specified job
Column 2: Specifies the delay in minutes after the system has been booted up for Anacron to wait before executing the job
Column 3: Contains a unique job identifier
Columns 4 to 6: Represent the command to be used to execute the scripts located under the */etc/cron.daily*, */etc/cron.weekly*, and */etc/cron.monthly* directories. Here the *run-parts* command is invoked for execution at the default niceness.

For each job, Anacron examines whether the job was already run during the specified period (column 1). It executes it after waiting for the number of minutes (column 2) plus the RANDOM_DELAY value if it wasn’t. When all missed jobs have been carried out and there is none pending, Anacron exits.

Anacron may be run manually at the command prompt. To run all the jobs that are scheduled in the */etc/anacrontab* file but were missed, simply type the **anacron** command and press the Enter key.

Anacron stores job execution dates in files located in the */var/spool/anacron* directory for each defined entry in the */etc/anacrontab* file. There are several options available with the *anacron* command. Check the manual pages for further information.

Chapter Summary

This chapter discussed two major topics: process management and job scheduling.

It is vital for users and administrators alike to have a strong grasp on running processes, resources they are consuming, process owners, process execution priorities, etc. They should learn how to list processes in a variety of ways. We looked at the five states a process is in at any given time during its lifecycle. We examined the concepts of niceness and reniceness for increasing or decreasing a process's priority. We analyzed some of the many available signals and looked at how they could be passed to running processes to perform an action on them.

The second and the last topic talked about submitting and managing tasks to run in the future one time or on a recurring basis. We learned about the service daemons that handle the task execution and the control files where we list users who can or cannot submit jobs. We looked at the log file that stores information for all executed jobs. We reviewed the syntax of the crontable and examined a variety of date/time formats for use with both at and cron job submission. We performed two exercises to get a grasp on their usage. Finally, we studied the anacron service that RHEL uses to execute any

scheduled jobs that were missed from running due to reasons such as system shutdown.

Check Your Understanding

1. What are the two commands to list the PID of a specific process?
2. By default the **.allow* files exist. True or False?
3. Where do the scheduling daemons store log information of executed jobs?
4. You must restart the *crond* service after modifying the */etc/crontab* file. True or False?
5. What are the background service processes normally referred to in Linux?
6. What is the default nice value?
7. Which service runs missed scheduled tasks?
8. The parent process gets the nice value of its child process. True or False?
9. When would the *cron* daemon execute a job that is submitted as **/10 * 2-6 6 * /home/user1/script1.sh*?
10. What is the other command besides *ps* to view running processes?
11. Every process that runs on the system has a unique identifier called UID. True or False?
12. Why would you use the *renice* command?
13. Which user does not have to be explicitly defined in either **.allow* or **.deny* file to run the at and cron jobs?
14. When would the *at* command execute a job that is submitted as at 01:00 12/12/2020?
15. What are the two commands that you can use to terminate a process?
16. What is the directory location where user crontab files are stored?
17. What would the *nice* command display without any options or arguments?
18. Which command can be used to edit crontables?

19. The default location to send application error messages is the system log file. True or False?
20. What are the five process states?
21. Signal 9 is used for a hard termination of a process. True or False?

Answers to Check Your Understanding

1. The *pidof* and *pgrep* commands.
2. False. By default, the **.deny* files exist.
3. The scheduling daemons store log information of executed jobs in the */var/log/cron* file.
4. False. The *crond* daemon does not need a restart after a crontable is modified.
5. The background service processes are referred to as daemons.
6. The default nice value is zero.
7. The *anacron* service executes any missed at and cron jobs.
8. False. The child process inherits its parent's niceness.
9. The *cron* daemon will run the script every tenth minute past the hour on the 2nd, 3rd, 4th, 5th, and 6th day of every sixth month.
10. The *top* command.
11. False. It is called the PID.
12. The *renice* command is used to change the niceness of a running process.
13. The *root* user.
14. The *at* command will run it at 1am on December 12, 2020.
15. The *kill* and *pkill* commands.
16. The user crontab files are stored in the */var/spool/cron* directory.
17. The *nice* command displays the default nice value when executed without any options.

18. You can use the *crontab* command with the -e option to edit crontables.
19. False. The default location is the user screen where the program is initiated.
20. The five process states are running, sleeping, waiting, stopped, and zombie.
21. True.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 8-1: Nice and Renice a Process

As *user1* with *sudo* on *server1*, open two terminal sessions. Run the *top* command in terminal 1. Run the *pgrep* or *ps* command in terminal 2 to determine the PID and the nice value of *top*. Stop *top* on terminal 1 and relaunch at a lower priority (+8). Confirm the new nice value of the process in terminal 2. Issue the *renice* command in terminal 2 and increase the priority of *top* to -10, and validate. (Hint: Processes and Priorities).

Lab 8-2: Configure a User Crontab File

As *user1* on *server1*, run the **tty** and **date** commands to determine the terminal file (assume */dev/pts/1*) and current system time. Create a cron entry to display “Hello World” on the terminal. Schedule **echo “Hello World” > /dev/tty/1** to run 3 minutes from the current system time. As *root*, ensure *user1* can schedule cron jobs. (Hint: Job Scheduling).

Chapter 09

Basic Package Management

This chapter describes the following major topics:

- Overview of Red Hat packages, naming, and management tools
- Package dependency and database
- Query, install, upgrade, freshen, overwrite, and remove packages
- Extract package files from installable package
- Validate package integrity and authenticity
- View GPG keys and verify package attributes
- Manage packages using the rpm command

RHCSA Objectives:

39. Install and update software packages from Red Hat Network, a remote repository, or

from the local file system (part of this objective is also covered in Chapter 10)

The Red Hat software management system is known as RPM Package Manager (RPM). RPM also refers to one or more files that are packaged together in a special format and stored in files with the .rpm extension. These rpm files (also called rpms, rpm packages, or packages) are manipulated by the RPM package management system. Each package included in and available for RHEL is in this file format. Packages have meaningful names and contain necessary files, as well as metadata structures such as ownership, permissions, and directory location for each included file. Packages may be downloaded and saved locally or on a network share for quick access, and they may have dependencies over files or other packages. In other words, a package may require the presence of additional files, another package, or a group of packages in order to be installed successfully and operate properly. Once a package has been installed and its metadata information stored in a package database, each attempt to update the package updates its metadata information as well.

RHEL provides a powerful tool for the installation and administration of RPM packages. The rpm command is flexible and it offers multitude of options and subcommands to perform functions such as querying, installing, upgrading, freshening, removing, and decompressing packages, and validating package integrity and authenticity.

Package Overview

RHEL is essentially a set of packages grouped together to create an operating system. They are prepackaged for

installation and assembled for various intended use cases. They are built around the Linux kernel and include thousands of packages that are digitally signed, tested, and certified. There are several basic and advanced concepts associated with packages, packaging, and their management that are touched upon in this chapter and Chapter 10 “Advanced Package Management”.

Packages and Packaging

A software *package* is a group of files organized in a directory structure along with metadata and intelligence that make up a software application. They are available in two types: *binary* (or *installable*) and *source*. Binary packages are installation-ready and are bundled for distribution. They have .rpm extension and contain install scripts, pre- and post-installation scripts, executables, configuration files, library files, dependency information, where to install files, and documentation. The documentation includes detailed instructions on how to install and uninstall the package, manual pages for the configuration files and commands, and other necessary information pertaining to the installation and usage of the package.

All metadata related to packages is stored at a central location and includes information such as package version, installation location, checksum values, and a list of included files with their attributes. This allows the package management toolset to handle package administration tasks efficiently by referencing this metadata.

The package intelligence is used by the package administration toolset for a successful completion of the package installation process. It may include information on prerequisites, user account setup (if required), and any directories and soft links that need to be created. The

intelligence also includes the reverse of this process for uninstallation.

Source packages come with the original unmodified version of the software that may be unpacked, modified as desired, and repackaged in the binary format for installation or redistribution. They are identified with the .src extension.

Package Naming

Red Hat software packages follow a standard naming convention. Typically, there are five parts to a package name: (1) the package name, (2) the package version, (3) the package release (revision or build), (4) the Enterprise Linux the package is created for, and (5) the processor architecture the package is built for. An installable package name always has the .rpm extension; however, this extension is removed from the installed package name.

For example, if the name of an installable package is openssl-1.1.1-8.el8.x86_64.rpm, its installed name would be openssl-1.1.1-8.el8.x86_64. Here is a description of each part of the package name:

openssl: package name
1.1.1: version
8: release
el8: stands for *Enterprise Linux* 8 (not all packages have it)
x86_64: processor architecture the package is created for. You may see “noarch” for platform-independent packages that can be installed on any hardware architecture, or “src” for source code packages.
.rpm: the extension

Package Dependency

An installable package may require the presence of one or more additional packages in order to be installed successfully. Likewise, a software package component may require the functionality provided by one or more packages to exist in order to operate as expected. This is referred to as *package dependency*, where one package depends on one or more other packages for installation or execution. Package dependency information is recorded in each package's metadata from where it is read by package handling utilities.

Package Database

Metadata for installed packages and package files is stored and maintained in the */var/lib/rpm* directory. This directory location is referred to as the *package database*, and it is referenced by package manipulation utilities to obtain package name and version data and information about ownerships, permissions, timestamps, and sizes for each and every file that is part of the package. The package database also contains information on dependencies. All this data aids management commands in listing and querying packages, verifying dependencies and file attributes, installing new packages, upgrading and uninstalling existing packages, and carrying out other package handling tasks.

The package database does not update existing package information by simply adding available enhancements. It removes the metadata of the package being replaced and then adds the information of the replacement package. In RHEL 8, it can maintain multiple versions of a single package alongside their metadata.

Package Management Tools

The primary tool for package management on Red Hat Enterprise Linux is called *rpm* (*redhat package manager*). This tool offers abundant options for easy package handling;

however, a major caveat is that it does not automatically resolve package dependencies. To overcome this gap, a more innovative tool called *yum* (*yellowdog update, modified*) was introduced, which offered an easier method for package management that can find, get, and install all required dependent packages automatically.

In RHEL 8, a major upcoming enhancement to *yum* has been introduced known as *dnf*. *dnf* is not an official acronym, but some documentation refers to it as *dandified* yum. You may still use the *yum* command; however, it is simply a soft link to *dnf*.

This chapter focuses on the use of the *rpm* command. Chapter 10 “Advanced Package Management” details the *dnf* command.

Package Management with rpm

The *rpm* command handles package management tasks including querying, installing, upgrading, freshening, overwriting, removing, extracting, validating, and verifying packages. As mentioned, this command has a major drawback as it does not have the ability to automatically satisfy package dependencies, which can be frustrating during software installation and upgrade. The *rpm* command works with both installed and installable packages.

The rpm Command

Before getting into the details, let’s look at some common *rpm* command options. Table 9-1 describes query options in both short and long option formats. You may use either format.

Query Options	Description
-q (--query)	Queries and displays packages
-qa (--query --all)	Lists all installed packages
-qc (--query --configfiles)	Lists configuration files in a package
-qd (--query --docfiles)	Lists documentation files in a package
-qf (--query --file)	Exhibits what package a file comes from
-qi (--query --info)	Shows installed package information including version, size, installation status and date, signature, and description
-qip (--query --info --package)	Shows installable package information including version, size, installation status and date, signature, and description
-ql (--query --list)	Lists all files in a package
-qR (--query --requires)	Lists files and packages a package depends on (requires)
-q --whatprovides	Lists packages that provide the specified package or file
-q --whatrequires	Lists packages that require the specified package or file

Table 9-1 rpm Command Query Options

Table 9-2 describes options related to package installation, removal, and verification. These options are also available in both short and long formats. You may use either format.

Install/Remove Options	Description
-e (--erase)	Removes a package
--force	Installs and replaces a package or files of the same version
-F (--freshen)	Upgrades an installed package
-h (--hash)	Shows installation progress with hash marks
-i (--install)	Installs a package
--import	Imports a public key
-K	Validates the signature and package integrity
-U (--upgrade)	Upgrades an installed package or loads it if it is not already installed
-v (--verbose) or -vv	Displays detailed information
-V (--verify)	Verifies the integrity of a package or package files

Table 9-2 rpm Command Install/Remove/Verify Options

The examples in this chapter employ most of these options in short format to understand how they are used to achieve desired results.

Getting Ready to Use rpm

In order to run some of the package management examples, attach the RHEL 8 ISO image to the *RHEL8-VM1* in VirtualBox. This will allow you to mount the image on *server1* and access the software packages in it. The following mounts the image on the empty */mnt* directory using the *mount* command. The */dev/sr0* represents the device file for the image and “ro” mounts the image in read-only mode. See Chapter 15 “Local File Systems and Swap” for details on mount and mount point concepts.

```
[user1@server1 ~]$ sudo mount -o ro /dev/sr0 /mnt
```

sr0 represents the first instance of the read-only optical device.

Verify the mount using the *df* command:

```
[user1@server1 ~]$ df -h | grep mnt
/dev/sr0               6.7G  6.7G     0 100% /mnt
```

Add the information for this optical device image to a file called *fstab* in the */etc* directory to ensure that the image is automatically mounted at each subsequent system reboot. Use the *vim* editor to open the file and append the following entry to the end of the file:

```
/dev/sr0  /mnt  iso9660  ro  0 0
```

Confirm that the new entry placed in the file has no typing or syntax errors. The command should return to the command prompt without displaying anything in the output:

```
[user1@server1 ~]$ sudo mount -a
[user1@server1 ~]$
```

The image and the packages therein can now be accessed via the */mnt* directory just like any other directory on the system. Here is a quick look at the subdirectories —*/mnt/BaseOS/Packages* and */mnt/AppStream/Packages*— where all the packages on this image are located:

```
[user1@server1 ~]$ ls -l /mnt/BaseOS/Packages |more
total 922884
-r--r--r--.  868 root root   151876 Dec 14  2018 aajohan-comfortaa-fonts-3.001-2
.el8.noarch.rpm
-r--r--r--.  303 root root    82976 Dec 14  2018 acl-2.2.53-1.el8.x86_64.rpm
-r--r--r--.  222 root root  1022540 Dec 14  2018 acpica-tools-20180629-3.el8.x86
_64.rpm
-r--r--r--.  302 root root   104672 Oct 12  2018 adcli-0.8.2-2.el8.x86_64.rpm

........
[user1@server1 ~]$ ls -l /mnt/AppStream/Packages |more
total 5450689
-r--r--r--.  165 root root   1977828 Feb  1  2019 389-ds-base-1.4.0.20-7.module+
el8+2750+1f4079fb.x86_64.rpm
-r--r--r--.  165 root root    357236 Feb  1  2019 389-ds-base-devel-1.4.0.20-7.m
odule+el8+2750+1f4079fb.x86_64.rpm
-r--r--r--.  165 root root    469808 Feb  1  2019 389-ds-base-legacy-tools-1.4.0
.20-7.module+el8+2750+1f4079fb.x86_64.rpm

........
```

You will find thousands of files in the two directory locations, each representing a single package.

Querying Packages

You can query for packages in the package database or at the specified location. The following are some examples.

To query all installed packages:

```
[user1@server1 ~]$ rpm -qa
qemu-kvm-block-gluster-2.12.0-63.module+el8+2833+c7d6d092.x86_64
boost-atomic-1.66.0-6.el8.x86_64
gnome-session-wayland-session-3.28.1-6.el8.x86_64
grub2-tools-2.02-66.el8.x86_64
lohit-gurmukhi-fonts-2.91.2-3.el8.noarch
liberation-fonts-common-2.00.3-4.el8.noarch
........
```

To query whether the specified package is installed:

```
[user1@server1 ~]$ rpm -q perl
package perl is not installed
```

To list all files in a package:

```
[user1@server1 ~]$ rpm -ql iproute
/etc/iproute2
/etc/iproute2/bpf_pinning
/etc/iproute2/ematch_map
/etc/iproute2/group
........
```

To list only the documentation files in a package:

```
[user1@server1 ~]$ rpm -qd audit
/usr/share/doc/audit/ChangeLog
/usr/share/doc/audit/README
/usr/share/doc/audit/auditd.cron
/usr/share/doc/audit/rules/10-base-config.rules
/usr/share/doc/audit/rules/10-no-audit.rules
........
```

To list only the configuration files in a package:

```
[user1@server1 ~]$ rpm -qc cups
/etc/cups/classes.conf
/etc/cups/client.conf
/etc/cups/cups-files.conf
/etc/cups/cupsd.conf
/etc/cups/lpoptions
........
```

To identify which package owns the specified file:

```
[user1@server1 ~]$ rpm -qf /etc/passwd
setup-2.12.2-1.el8.noarch
```

To display information about an installed package including version, release, installation status, installation date, size, signatures, description, and so on:

```
[user1@server1 ~]$ rpm -qi setup
Name        : setup
Version     : 2.12.2
Release     : 1.el8
Architecture: noarch
Install Date: Fri 23 Aug 2019 06:06:31 AM EDT
Group       : System Environment/Base
Size        : 725081
License     : Public Domain
Signature   : RSA/SHA256, Mon 10 Sep 2018 08:25:55 AM EDT, Key ID 199e2f91fd431d
51
Source RPM  : setup-2.12.2-1.el8.src.rpm
Build Date  : Mon 10 Sep 2018 08:17:59 AM EDT
Build Host  : s390-001.fedora-internal.build.eng.bos.redhat.com
Relocations : (not relocatable)
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Vendor      : Red Hat, Inc.
URL         : https://pagure.io/setup/
Summary     : A set of system configuration and setup files
Description :
The setup package contains a set of important system configuration and
setup files, such as passwd, group, and profile.
```

To list all file and package dependencies for a given package:

```
[user1@server1 ~]$ rpm -qR chrony
/bin/bash
/bin/sh
/bin/sh
/bin/sh
/bin/sh
/bin/sh
config(chrony) = 3.3-3.el8
libc.so.6()(64bit)
libc.so.6(GLIBC_2.12)(64bit)
libc.so.6(GLIBC_2.14)(64bit)
libc.so.6(GLIBC_2.15)(64bit)
libc.so.6(GLIBC_2.17)(64bit)
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.25)(64bit)
........
```

To query an installable package for metadata information (version, release, architecture, description, size, signatures, etc.):

```
[user1@server1 ~]$ rpm -qip /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm
warning: /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm: Header V3 RSA/SHA256 S
ignature, key ID fd431d51: NOKEY
Name        : zsh
Version     : 5.5.1
Release     : 6.el8
Architecture: x86_64
Install Date: (not installed)
Group       : Unspecified
Size        : 7270070
License     : MIT
Signature   : RSA/SHA256, Mon 17 Dec 2018 12:38:12 PM EST, Key ID 199e2f91fd431d
51
Source RPM  : zsh-5.5.1-6.el8.src.rpm
Build Date  : Mon 17 Dec 2018 11:08:47 AM EST
Build Host  : x86-vm-05.build.eng.bos.redhat.com
Relocations : (not relocatable)
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Vendor      : Red Hat, Inc.
........
```

To determine what packages require the specified package in order to operate properly:

```
[user1@server1 ~]$ rpm -q --whatrequires lvm2
libblockdev-lvm-2.19-7.el8.x86_64
udisks2-lvm2-2.8.0-2.el8.x86_64
libvirt-daemon-driver-storage-logical-4.5.0-23.module+el8+2800+2d311f65.x86_64
vdo-6.2.0.293-10.el8.x86_64
```

The above output lists all the packages that will require the specified package “lvm2” in order to work fully and properly.

Installing a Package

Installing a package creates the necessary directory structure for the package, installs the required files, and runs any post-installation steps. The following command installs a package called *zsh-5.5.1-6.el8.x86_64.rpm* on the system:

```
[user1@server1 ~]$ sudo rpm -ivh /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm

warning: /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm: Header V3 RSA/SHA256 S
ignature, key ID fd431d51: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:zsh-5.5.1-6.el8                  ################################# [100%]
```

If this package required the presence of any missing packages, you would see an error message related to failed dependencies. In that case, you would have to first install the missing packages for this package to be loaded successfully.

Upgrading a Package

Upgrading a package upgrades an installed version of the package. In the absence of an existing version, the upgrade simply installs the package.

To upgrade a package called *sushi*, use the -U option with the *rpm* command. Notice that the *sushi* package is located in a different directory than the *zsh* package in the previous example.

```
[user1@server1 ~]$ sudo rpm -Uvh /mnt/AppStream/Packages/sushi-3.28.3-1.el8.x86_
64.rpm
warning: /mnt/AppStream/Packages/sushi-3.28.3-1.el8.x86_64.rpm: Header V3 RSA/SH
A256 Signature, key ID fd431d51: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
        package sushi-3.28.3-1.el8.x86_64 is already installed
```

The command makes a backup of all the affected configuration files during the upgrade process and adds the extension *.rpmsave* to them. In the above example, the *sushi* package was installed, as it was not already on the system.

Freshening a Package

Freshening a package requires that an older version of the package must already exist on the system.

To freshen the *sushi* package, use the -F option:

```
[user1@server1 ~]$ sudo rpm -Fvh /mnt/AppStream/Packages/sushi-3.28.3-1.el8.x86_
64.rpm
warning: /mnt/AppStream/Packages/sushi-3.28.3-1.el8.x86_64.rpm: Header V3 RSA/SH
A256 Signature, key ID fd431d51: NOKEY
```

The above command did nothing because the same package version specified with the command is already installed on the system. It will only work if a newer version of the installed package is available.

Overwriting a Package

Overwriting a package replaces the existing files associated with the package of the same version.

To overwrite the package *zsh-5.5.1-6.el8.x86_64* that was installed earlier, use the --replacepkgs option:

```
[user1@server1 ~]$ sudo rpm -ivh --replacepkgs /mnt/BaseOS/Packages/zsh-5.5.1-6.
el8.x86_64.rpm
warning: /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm: Header V3 RSA/SHA256 S
ignature, key ID fd431d51: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:zsh-5.5.1-6.el8                  ################################# [100%]
```

The installation progress indicates that the *zsh* package was replaced successfully. This action is particularly useful when you suspect corruption in one or more installed package files and you want to start fresh.

Removing a Package

Removing a package uninstalls the package and all its associated files and the directory structure.

To remove the package *sushi*, use the -e option and specify -v for verbosity:

```
[user1@server1 ~]$ sudo rpm sushi -ve
Preparing packages...
sushi-3.28.3-1.el8.x86_64
```

This command performs a dependency inspection to check whether there are any packages that require the existence of the package being weeded out, and fails the removal if it detects a dependency.

Extracting Files from an Installable Package

Files in an installable package can be extracted using the *rpm2cpio* command for reasons such as examining the contents of the package, replacing a corrupted or lost command, or replacing a critical configuration file of an installed package to its original state.

Assuming you have lost the */etc/chrony.conf* configuration file and want to retrieve it from its installable package and put it back, you'll first need to determine what package this file comes from:

```
[user1@server1 ~]$ rpm -qf /etc/chrony.conf
chrony-3.3-3.el8.x86_64
```

Now use the *rpm2cpio* command to extract (-i) all files from the "chrony" package and create (-d) the necessary directory structure during the retrieval. Extract the files in a temporary location such as the */tmp* directory before you proceed with overwriting the destination under */etc*.

```
[user1@server1 ~]$ cd /tmp
[user1@server1 tmp]$ sudo rpm2cpio /mnt/BaseOS/Packages/chrony-3.3-3.el8.x86_64.rpm |
cpio -imd
1066 blocks
```

Run the *find* command to locate the *chrony.conf* file:

```
[user1@server1 tmp]$ sudo find . -name chrony.conf
./etc/chrony.conf
```

The above output shows that the file is in the */tmp/etc* directory. You can copy it to the */etc* directory now, and you're back in business.

Validating Package Integrity and Credibility

Before it is installed, a package may be checked for integrity (completeness and error-free state) and credibility

(authenticity) after it has been copied to another location, downloaded from the web, or obtained elsewhere. Use the MD5 checksum for verifying its integrity and the *GNU Privacy Guard* (GnuPG or GPG) public key signature for ensuring the credibility of its developer or publisher. This will ensure an uncorrupted and genuine piece of software.

The commercial version of GPG is referred to as PGP (Pretty Good Privacy).

To check the integrity of a package such as *zsh-5.5.1-6.el8.x86_64.rpm* located in */mnt/BaseOS/Packages*:

```
[user1@server1 ~]$ rpm -K /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm --nosignature
/mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm: digests OK
```

The OK in the output confirms that the package is free of corruption.

Red Hat signs their products and updates with a GPG key, and includes necessary public keys in the products for verification. For RHEL, the keys are in files on the installation media and are copied to the */etc/pki/rpm-gpg/* directory during the OS installation. Refer to Table 9-3 for a list of files in that directory and a short explanation.

GPG File	Description
RPM-GPG-KEY-redhat-release	Used for packages shipped after November 2009 and their updates
RPM-GPG-KEY-redhat-beta	Used for beta test products shipped after November 2009

Table 9-3 Red Hat GPG Key Files

To check the credibility of a package, import the relevant GPG key and then verify the package. The table above shows that the GPG file for the recent packages is *RPM-GPG-KEY-redhat-release*. In the following example, run the *rpm*

command to import the GPG key from this file and verify the signature for the *zsh-5.5.1-6.el8.x86_64.rpm* package using the -K option with the command:

```
[user1@server1 ~]$ sudo rpmkeys --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
[user1@server1 ~]$ sudo rpmkeys -K /mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm
/mnt/BaseOS/Packages/zsh-5.5.1-6.el8.x86_64.rpm: digests signatures OK
```

The OK validates the package signature and certifies the authenticity and integrity of the package.

Viewing GPG Keys

The GPG key imported in the previous subsection can be viewed with the *rpm* command. You can list the key and display its details as well. Run the command as follows to list the imported key:

```
[user1@server1 ~]$ rpm -q gpg-pubkey
gpg-pubkey-fd431d51-4ae0493b
gpg-pubkey-d4082792-5b32db75
```

The output suggests that there are two GPG public keys currently imported on the system. Let’s view the details for the first one:

```
[user1@server1 ~]$ rpm -qi gpg-pubkey-fd431d51-4ae0493b
Name        : gpg-pubkey
Version     : fd431d51
Release     : 4ae0493b
Architecture: (none)
Install Date: Mon 23 Sep 2019 03:23:26 PM EDT
Group       : Public Keys
Size        : 0
License     : pubkey
Signature   : (none)
Source RPM  : (none)
Build Date  : Thu 22 Oct 2009 07:59:55 AM EDT
Build Host  : localhost
Relocations : (not relocatable)
Packager    : Red Hat, Inc. (release key 2) <security@redhat.com>
Summary     : gpg(Red Hat, Inc. (release key 2) <security@redhat.com>)
Description :
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: rpm-4.14.2 (NSS-3)

mQINBErgSTsBEACh2A4b009t+vzC9VrVtL1AKvUWi90PCjkvR7Xd8DtJxeeMZ5eF
0HtzIG58qDRybwUe89FZprB1ffuUKzdE+HcL3FbNWSS0XVjZIersdXyH3NvnLLLF
........
JUfNh3DVRGhg8cMIti21njiRh7gyFI20ccATY7bBSr79JhuNwelHuxLrCFpY7V25
0Fktll5jZJaMxuQBqYdBgSay2G0U6D1+7VsWufpzd/Abx1/c3oi9ZaJvW22kAggq
dzdA27UUYjWvx42w9menJwh/0jeQcTecIUd0d0rFcw/clpvgMMl/Q73yzKgKYw==
=zbHE
-----END PGP PUBLIC KEY BLOCK-----
```

The output returns both the metadata and the key data for the specified GPG public key.

Verifying Package Attributes

Verifying the integrity of an installed package compares the attributes of files in the package with the original file attributes saved and stored in the package database at the time of package installation. The verification process uses the *rpm* command with the -V option to compare the owner, group, permission mode, size, modification time, digest, and type among other attributes.

The command returns to the prompt without exhibiting anything if it detects no changes in the attributes. You can use the -v or -vv option with the command for increased verbosity.

Run this check on the *at* package:

```
[user1@server1 ~]$ sudo rpm -V at
```

The command returned nothing, which implies that the file attributes are intact. Now change the permissions on one of the files, */etc/sysconfig/atd*, in this package to 770 from the current value of 644, and then re-execute the verification test:

```
[userl@serverl ~]$ ls -l /etc/sysconfig/atd
-rw-r--r--. 1 root root 403 Aug 12  2018 /etc/sysconfig/atd
[userl@serverl ~]$ sudo chmod -v 770 /etc/sysconfig/atd
mode of '/etc/sysconfig/atd' changed from 0644 (rw-r--r--) to 0770 (rwxrwx---)
[userl@serverl ~]$ sudo rpm -V at
.M.......  c /etc/sysconfig/atd
```

The output is indicative of a change in the permission mode on the *atd* file in the *at* package. You may alternatively run the verification check directly on the file by adding the -f option to the command and passing the filename as an argument:

```
[userl@serverl ~]$ sudo rpm -Vf /etc/sysconfig/atd
.M.......  c /etc/sysconfig/atd
```

The output returns three columns: column 1 contains nine fields, column 2 shows the file type, and column 3 expresses the full path of the file. The command performs a total of nine checks, as illustrated by the codes in column 1 of the output, and displays any changes that have occurred since the package that contains the file was installed. Each of these codes has a meaning. Table 9-4 lists the codes with description as they appear from left to right. The period character (.) appears for an attribute that is not in an altered state.

Code	Description
S	Appears if the file size is different
M	Appears if the (mode) permission or file type is altered
5	Appears if MD5 checksum does not match
D	Appears if the file is a device file and its major or minor number has changed
L	Appears if the file is a symlink and its path has altered
U	Appears if the ownership has modified
G	Appears if the group membership has modified
T	Appears if timestamp has changed
P	Appears if capabilities have altered
.	Appears if no modification is detected

Table 9-4 Package Verification Codes

Column 2 in the output above exposes a code that represents the type of file. Table 9-5 lists them.

File Type	Description
c	Configuration file
d	Documentation file
g	Ghost file
l	License file
r	Readme file

Table 9-5 File Type Codes

Based on the information in the tables, the */etc/sysconfig/atd* is a configuration file with a modified permission mode. Reset the attribute to its previous value and rerun the check to ensure the file is back to its original state.

```
[user1@server1 ~]$ sudo chmod -v 644 /etc/sysconfig/atd
mode of '/etc/sysconfig/atd' changed from 0770 (rwxrwx---) to 0644 (rw-r--r--)
[user1@server1 ~]$ sudo rpm -V at
[user1@server1 ~]$
```

The command produced no output, which confirms the integrity of the file as well as the package.

Exercise 9-1: Perform Package Management Tasks Using rpm

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will verify the integrity and authenticity of a package called *dcraw* located in the */mnt/AppStream/Packages* directory on the installation image and then install it. You will display basic information about the package, show files it contains, list documentation files, verify the package attributes, and remove the package.

1. Run the *ls* command on the */mnt/AppStream/Packages* directory to confirm that the *dcraw* package is available:

```
[user1@server1 ~]$ ls -l /mnt/AppStream/Packages/dcraw*
-r--r--r--. 221 root root 273600 Dec 14  2018 /mnt/AppStream/Packages/dcraw-9.27.0-9.el8.x
86_64.rpm
```

2. Run the *rpm* command and verify the integrity and credibility of the package:

```
[user1@server1 ~]$ rpmkeys -K /mnt/AppStream/Packages/dcraw-9.27.0-9.el8.x86_64.rpm
/mnt/AppStream/Packages/dcraw-9.27.0-9.el8.x86_64.rpm: digests signatures OK
```

3. Install the package:

```
[user1@server1 ~]$ sudo rpm -ivh /mnt/AppStream/Packages/dcraw-9.27.0-9.el8.x86_64.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:dcraw-9.27.0-9.el8               ################################# [100%]
```

4. Show basic information about the package:

```
[user1@server1 ~]$ rpm -qi dcraw
Name        : dcraw
Version     : 9.27.0
Release     : 9.el8
Architecture: x86_64
Install Date: Mon 23 Sep 2019 10:31:43 PM EDT
Group       : Unspecified
Size        : 560675
License     : GPLv2+
Signature   : RSA/SHA256, Fri 14 Dec 2018 04:59:21 PM EST, Key ID 199e2f91fd431d51
Source RPM  : dcraw-9.27.0-9.el8.src.rpm
Build Date  : Sun 12 Aug 2018 05:19:30 AM EDT
Build Host  : x86-vm-01.build.eng.bos.redhat.com
Relocations : (not relocatable)
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Vendor      : Red Hat, Inc.
URL         : http://cybercom.net/~dcoffin/dcraw
Summary     : Tool for decoding raw image data from digital cameras
Description :
This package contains dcraw, a command line tool to decode raw image data
downloaded from digital cameras.
```

5. Show all the files the package contains:

```
[user1@server1 ~]$ rpm -ql dcraw
/usr/bin/dcraw
/usr/lib/.build-id
/usr/lib/.build-id/af
/usr/lib/.build-id/af/3c0f26d5d7cbb4d1210c45454d30c3b08d4d16
/usr/share/locale/ca/LC_MESSAGES/dcraw.mo
/usr/share/locale/cs/LC_MESSAGES/dcraw.mo
/usr/share/locale/da/LC_MESSAGES/dcraw.mo
/usr/share/locale/de/LC_MESSAGES/dcraw.mo
.........
```

6. List the documentation files the package has:

```
[user1@server1 ~]$ rpm -qd dcraw
/usr/share/man/ca/man1/dcraw.1.gz
/usr/share/man/cs/man1/dcraw.1.gz
/usr/share/man/da/man1/dcraw.1.gz
/usr/share/man/de/man1/dcraw.1.gz
/usr/share/man/eo/man1/dcraw.1.gz
/usr/share/man/es/man1/dcraw.1.gz
/usr/share/man/fr/man1/dcraw.1.gz
/usr/share/man/hu/man1/dcraw.1.gz
/usr/share/man/it/man1/dcraw.1.gz
.........
```

7. Verify the attributes of each file in the package. Use verbose mode.

```
[user1@server1 ~]$ rpm -Vv dcraw
.........    /usr/bin/dcraw
.........  a /usr/lib/.build-id
.........  a /usr/lib/.build-id/af
.........  a /usr/lib/.build-id/af/3c0f26d5d7cbb4d12l0c4
.........    /usr/share/locale/ca/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/cs/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/da/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/de/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/eo/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/es/LC_MESSAGES/dcraw.mo
.........    /usr/share/locale/fr/LC_MESSAGES/dcraw.mo
.........
```

8. Remove the package:

```
[user1@server1 ~]$ sudo rpm -ve dcraw
Preparing packages...
dcraw-9.27.0-9.el8.x86_64
```

This concludes the exercise.

Chapter Summary

This chapter is the first of the two chapters (second one being Chapter 10) with coverage on software management. It covered the foundational topics and set the groundwork for more advanced features and functions.

We learned the concepts around packages, packaging, naming convention, dependency, and patch database. We looked at the variety of options available with the rpm utility to perform package administration tasks and put many of them into action to demonstrate their usage. Moreover, we employed appropriate options to view and validate package metadata information, GPG keys, and package attributes.

Check Your Understanding

1. What would the *rpm -ql zsh* command do?
2. State the purpose of the *rpm2cpio* command.
3. What is the difference between freshening and upgrading a package?

4. The *rpm* command automatically takes care of package dependencies. True or False?
5. What is the difference between installing and upgrading a package?
6. Package database is located in the */var/lib/rpm* directory. True or False?
7. What would the *rpm -qf /bin/bash* command do?
8. Name the directory where RHEL 8 stores GPG signatures.
9. What would the options ivh cause the *rpm* command to do?
10. What would the *rpm -qa* command do?

Answers to Check Your Understanding

1. The command provided will list all the files that are included in the installed *zsh* package.
2. The *rpm2cpio* command is used to extract files from the specified package.
3. Both are used to upgrade an existing package, but freshening requires an older version of the package to exist.
4. False.
5. Installing will install a new package whereas upgrading will upgrade an existing package or install it if it does not already exist.
6. True.
7. The command provided will display information about the */bin/bash* file.
8. RHEL 8 stores GPG signatures in the */etc/pki/rpm-gpg* directory.
9. The *rpm* command will install the specified package and show installation details and hash signs for progress.

10. The command provided will list all installed packages.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 9-1: Install and Verify Packages

As *user1* with *sudo* on *server1*, make sure the RHEL 8 ISO image is attached to the VM and mounted. Use the *rpm* command and install the *zsh* package by specifying its full path. Run the *rpm* command again and perform the following on the *zsh* package: (1) show information, (2) validate integrity, and (3) display attributes. (Hint: Package Management with rpm).

Lab 9-2: Query and Remove Packages

As *user1* with *sudo* on *server1*, make sure the RHEL 8 ISO image is attached to the VM and mounted. Use the *rpm* command to perform the following: (1) check whether the *setup* package is installed, (2) display the list of configuration files in the *setup* package, (3) show information for the *zlib-devel* package on the ISO image, (4) reinstall the *zsh* package (--reinstall -vh), and (5) remove the *zsh* package. (Hint: Package Management with rpm).

Chapter 10

Advanced Package Management

This chapter describes the following major topics:

- Describe package groups
- Understand application streams and modules
- Software repositories and how to access them
- Review module streams and module profiles
- Perform software management operations using dnf
- List, install, update, and delete individual packages, package groups, and modules
- Show package information, determine provider, and search metadata

- Exhibit package group and module information
- Install module from alternative stream

RHCSA Objectives:

39. Install and update software packages from Red Hat Network, a remote repository, or from the local file system (part of this objective is covered in Chapter 09)
40. Work with package module streams

This is the second of the two chapters that discusses software administration in RHEL. While the first chapter (Chapter 09) expounds upon the basic concepts of rpm package management and demonstrates a basic command to manipulate individual packages, this chapter elaborates on the concepts and handling of package groups, package modules, and package streams. It also presents a coverage on the concept of software repositories and how to configure them.

The dnf command (the yum command) is superior to the rpm command in the sense that it performs automatic dependency checks and marks any identified extra packages for installation. There are numerous options and subcommands available with this advanced tool and it is capable of interacting with software repositories as well.

Advanced Package Management Concepts

We discussed and grasped the basic package management concepts and employed the *rpm* tool in a variety of ways to manipulate individual packages. The focus of this chapter will be on understanding advanced software packaging and distribution techniques, and using a tool that is capable of handling single, discrete packages as well as collections of packages at a time, efficiently.

Package Groups

A *package group* is a collection of correlated packages designed to serve a common purpose. It provides the

convenience of querying, installing, and deleting as a single unit rather than dealing with packages individually. There are two types of package groups: *environment groups* and *package groups*. The environment groups available in RHEL 8 are server, server with GUI, minimal install, workstation, virtualization host, and custom operating system. These are listed on the software selection window during RHEL 8 installation. The package groups include container management, smart card support, security tools, system tools, network servers, etc.

Application Streams and Modules

Application Streams is a new concept introduced in RHEL 8. It employs a modular approach to organize multiple versions of a software application alongside its dependencies to be available for installation from a single repository. A *module* can be thought of as a logical set of application packages that includes everything required to install it, including the executables, libraries, documentation, tools, and utilities as well as dependent components. Modularity gives the flexibility to choose the version of software based on need.

In older RHEL releases, each version of a package would have to come from a separate repository. This has changed in RHEL 8. Now modules of a single application with different versions can be stored and made available for installation from a common repository. The package management tool has also been enhanced to manipulate modules.

RHEL 8 is shipped with two core repositories called *BaseOS* and *Application Stream* (AppStream).

BaseOS Repository

The BaseOS repository includes the core set of RHEL 8 components including the kernel, modules, bootloader, and other foundational software packages. These components lay

the foundation to install and run software applications and programs. BaseOS repository components are available in the traditional rpm format.

AppStream Repository

The AppStream repository comes standard with core applications, as well as several add-on applications many of them in the traditional rpm format and some in the new modular format. These add-ons include web server software, development languages, database software, etc. and are shipped to support a variety of use cases and deployments.

Benefits of Segregation

There are two fundamental benefits to a segregation of the BaseOS components from other applications: (1) it separates the application components from the core operating system elements, and (2) it allows publishers to deliver and administrators to apply application updates more frequently. In previous RHEL versions, an OS update would update all installed components including the kernel, service, and application components to the latest versions by default. This could result in an unstable system or a misbehaving application due to an unwanted upgrade of one or more packages. By detaching the base OS components from the applications, either of the two can be updated independent of the other. This provides enhanced flexibility in tailoring the system components and application workloads without impacting the underlying stability of the system.

Module Streams

A *module stream* is a collection of packages organized by version. Each module can have multiple streams, and each stream receives updates independent of the other streams. A stream can be enabled or disabled. An enabled stream allows the packages it contains to be queried or installed.

Only one stream of a specific module can be enabled at a time. Each module has a default stream, which provides the latest or the recommended version. For example, the *perl* module provides two independent streams for the same software, one for version 5.24 and the other for 5.26. Stream 5.26 is the default, as it is a later version. The system will automatically attempt to install it if asked for *perl*.

Module Profiles

A *module profile* is a list of recommended packages organized for purpose-built, convenient deployments to support a variety of use cases such as minimal, development, common, client, server, etc. A profile may also include packages from the BaseOS repository or the dependencies of the stream. Each module stream can have zero, one, or more profiles associated with it with only one of them marked as the *default*.

dnf/yum Repository

A *dnf repository* (*yum repository* or a *repo*) is a digital library for storing software packages. A repository is accessed for package retrieval, query, update, and installation, and it may be free or for a fee. The two repositories—BaseOS and AppStream—come preconfigured with the RHEL 8 ISO image. There are a number of other repositories available on the Internet that are maintained by software publishers such as Red Hat and CentOS. Furthermore, you can build private custom repositories for internal IT use for stocking and delivering software. This may prove to be a good practice for an organization with a large Linux server base, as it manages dependencies automatically and aids in maintaining software consistency across the board. These repositories can also be used to store in-house developed packages.

It is important to obtain software packages from authentic and reliable sources such as Red Hat to prevent potential damage to your system and to circumvent possible software corruption.

There is a process to create repositories and to access preconfigured repositories. Creating repositories is beyond the scope of this book, but there are two pre-set repositories available on the RHEL 8 image. You will configure access to them via a definition file to support the exercises and lab environment.

A sample repo definition file is shown below with some key directives:

```
[BaseOS_RHEL_8.0]
name= RHEL 8.0 base operating system components
baseurl=file:///mnt/BaseOS
enabled=1
gpgcheck=0
```

EXAM TIP: Knowing how to configure a dnf/yum repository using a URL plays an important role in completing some of the RHCSA exam tasks successfully. Use two forward slash characters (//) with the baseurl directive for an FTP, HTTP, or HTTPS source.

The above example shows five lines from a sample repo file. Line 1 defines an exclusive ID within the square brackets. Line 2 is a brief description of the repo with the “name” directive. Line 3 is the location of the repodata directory with the “baseurl” directive. Line 4 shows whether this repository is active. Line 5 shows if packages are to be GPG-checked for authenticity.

Each repository definition file must have a unique ID, a description, and a baseurl directive defined at a minimum; other directives are set as required. The baseurl directive for a local directory path is defined as file:///local_path (the first

two forward slash characters represent the URL convention, and the third forward slash is for the absolute path to the destination directory), and for FTP and HTTP(S) sources as ftp://hostname/network_path and http(s)://hostname/network_path, respectively. The network path must include a resolvable hostname or an IP address.

Software Management with dnf

Software for enterprise Linux distributions such as Red Hat and CentOS is available in the rpm format. These distributions offer tools to work with individual packages as well as package groups and modules. The *rpm* command was used in the previous chapter to query, list, install, and erase packages, in addition to a few other tasks that it can perform. This command is limited to managing one package at a time.

A more capable tool available in RHEL for managing a single package, a group of packages, and a module is referred to as *dnf* (or *yum*). This tool has an associated configuration file that can define settings to control its behavior.

dnf Configuration File

The key configuration file for *dnf* is *dnf.conf* that resides in the */etc/dnf* directory. The “main” section in the file sets directives that have a global effect on dnf operations. You can define separate sections for each custom repository that you plan to set up on the system. However, the preferred location to store configuration for each custom repository in their own definition files is in the */etc/yum.repos.d* directory, which is the default location created for this purpose. The default content of this configuration file is listed below:

```
[user1@server1 ~]$ cat /etc/dnf/dnf.conf
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
```

Table 10-1 explains the above and a few other directives that you may define in the file. The directives in Table 10-1 are listed in an alphabetical order.

Directive	Description
best	Specifies whether to install (or upgrade to) the latest available version
clean_requirements_on_remove	Defines whether to remove dependencies during a package removal process that are no longer in use
debuglevel	Sets the level between 1 (minimum) and 10 (maximum) at which the debug is to be recorded in the logfile. Default is 2. A value of 0 disables this feature.
gpgcheck	Indicates whether to check the GPG signature for package authenticity. Default is 1 (enabled).
installonly_limit	Specifies a count of packages that can be installed concurrently. Default is 3.
keepcache	Defines whether to store the package and header cache following a successful installation. Default is 0 (disabled).
logdir	Sets the directory location to store the log files. Default is */var/log.*
obsoletes	Checks and removes any obsolete dependent packages during installs and updates. Default is 1 (enabled).

Table 10-1 Directive Settings in dnf.conf File

There are a multitude of additional directives available that you may want to set in the main section of this file or in custom repository definition files. Run **man 5 dnf.conf** for details.

The dnf Command

The *dnf* package management tool introduced in RHEL 8 is the next upcoming major version of the *yum* package manager that had been around in RHEL for years. You can use either of the two interchangeably in RHEL 8. In fact, *yum* is simply a soft link to the *dnf* utility. The *dnf* command is used in the examples and exercises throughout this book. The *dnf* utility requires the system to have access to a local or remote software repository or to a local installable package file. The Red Hat Subscription Management service offers access to official Red Hat software repositories. There are other web-based repositories that host packages that you may want to install and use. Alternatively, you can set up a local, custom repository on your system and add packages of your choice to it.

The primary benefit of using *dnf* over *rpm* is the command's ability to resolve dependencies automatically by identifying and installing any additional required packages for a successful installation of the specified software. With multiple repositories set up, *dnf* extracts the software from wherever it finds it.

dnf may be used to perform abundant software administration tasks. It invokes the *rpm* utility in the background. *dnf* can perform a number of operations on individual packages, package groups, and modules such as listing, querying, installing, and removing them, as well as enabling and disabling specific module streams.

Table 10-2 summarizes the software handling tasks that *dnf* can perform on packages. It also lists two subcommands (clean and repolist) that are specific to repositories. The subcommands in Table 10-2 are sequenced alphabetically. Refer to the manual pages of *dnf* for additional

subcommands, operators, options, examples, and other details.

Subcommand	Description
check-update	Checks if updates are available for installed packages
clean	Removes cached data
history	Displays previous dnf activities as recorded in the /var/lib/dnf/history directory
info	Shows details for a package
install	Installs or updates a package
list	Lists installed and available packages
provides	Searches for packages that contain the specified file or feature
reinstall	Reinstalls the exact version of an installed package
remove	Removes a package and its dependencies
repolist	Lists enabled repositories
repoquery	Runs queries on available packages
search	Searches package metadata for the specified string
upgrade	Updates each installed package to the latest version

Table 10-2 dnf Subcommands for Packages and Repositories

Table 10-3 lists and describes *dnf* subcommands that are intended for operations on package groups and modules.

Subcommand	Description
group install	Installs or updates a package group
group info	Returns details for a package group
group list	Lists available package groups
group remove	Removes a package group
module disable	Disables a module along with all the streams it contains
module enable	Enables a module along with all the streams it contains
module install	Installs a module profile including its packages
module info	Shows details for a module
module list	Lists all available module streams along with their profiles and status
module remove	Removes a module profile including its packages
module reset	Resets a module so that it is neither in enable nor in disable state
module update	Updates packages in a module profile

Table 10-3 dnf Subcommands for Package Groups and Modules

You will use most of the subcommands from Table 10-2 and Table 10-3 in the examples and exercises that follow, but first you'll need to create a definition file and configure access to the two repositories available on the RHEL 8 ISO image.

Exercise 10-1: Configure Access to Pre-Built ISO Repositories

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will set up access to the two dnf repositories that are available on RHEL 8 image. You've already configured an automatic mounting of RHEL 8 image on */mnt* in Chapter 09 "Basic Package Management". You will create a definition file for the repositories and confirm.

1. Verify that the image is currently mounted:

```
[user1@server1 ~]$ df -h | grep mnt
/dev/sr0              6.7G  6.7G     0 100% /mnt
```

2. Create a definition file called *local.repo* in the */etc/yum.repos.d* directory using the *vim* editor and define the following data for both repositories in it:

```
[BaseOS]
name=BaseOS
baseurl=file:///mnt/BaseOS
gpgcheck=0

[AppStream]
name=AppStream
baseurl=file:///mnt/AppStream
gpgcheck=0
```

3. Confirm access to the repositories:

```
[user1@server1 ~]$ sudo dnf repolist
Updating Subscription Management repositories.
Unable to read consumer identity
This system is not registered to Red Hat Subscription Management. You can use su
bscription-manager to register.
AppStream                                       3.1 MB/s | 3.2 kB     00:00
BaseOS                                          2.7 MB/s | 2.7 kB     00:00
repo id                               repo name                           status
AppStream                             AppStream                           4,672
BaseOS                                BaseOS                              1,658
```

Ignore lines 1-4 in the output that are related to subscription and system registration. Lines 5 and 6 show the rate at which the command read the repo data. Line 7 displays the timestamp of the last metadata check. The last two lines show the repo IDs, repo names, and a count of packages they hold. As depicted, the AppStream repo consists of 4,672 packages, and the BaseOS repo contains 1,658 packages. Both repos are enabled by default and are ready for use.

We have divided software administration into three sections to focus on individual packages, package groups, and modules separately.

Individual Package Management

The *dnf* command can be used to perform a variety of operations on individual packages, just like the *rpm* command. The following subsections elaborate with examples on using this tool to list, install, query, and remove packages.

Listing Available and Installed Packages

Listing the packages that are available for installation from one or more enabled repositories helps you understand what is in the current software inventory and what is needed. Likewise, listing the packages that are already installed on the system enables you to make important decisions as to whether they should be retained, upgraded, downgraded, or erased. The *dnf* command lists available packages as well as installed packages.

To list all packages available for installation from all enabled repos, run a query against the two repositories that we configured earlier, as those are the only ones you currently have access to. The following command will result in a long output.

```
[user1@server1 ~]$ sudo dnf repoquery
Last metadata expiration check: 0:04:03 ago on Tue 24 Sep 2019 11:17:10 PM EDT.
CUnit-0:2.1.3-17.el8.i686
CUnit-0:2.1.3-17.el8.x86_64
GConf2-0:3.2.6-22.el8.i686
GConf2-0:3.2.6-22.el8.x86_64
Judy-0:1.0.5-18.module+el8+2765+cfa4f87b.x86_64
LibRaw-0:0.19.1-1.el8.i686
LibRaw-0:0.19.1-1.el8.x86_64
ModemManager-0:1.8.0-1.el8.x86_64
ModemManager-glib-0:1.8.0-1.el8.i686
.........
```

To limit the above to the list of packages that are available only from a specific repo:

```
[user1@server1 ~]$ sudo dnf repoquery --repo "BaseOS"
Last metadata expiration check: 0:06:17 ago on Tue 24 Sep 2019 11:17:10 PM EDT.
ModemManager-0:1.8.0-1.el8.x86_64
ModemManager-glib-0:1.8.0-1.el8.i686
ModemManager-glib-0:1.8.0-1.el8.x86_64
NetworkManager-1:1.14.0-14.el8.x86_64
NetworkManager-adsl-1:1.14.0-14.el8.x86_64
NetworkManager-bluetooth-1:1.14.0-14.el8.x86_64
NetworkManager-config-connectivity-redhat-1:1.14.0-14.el8.noarch
NetworkManager-config-server-1:1.14.0-14.el8.noarch
NetworkManager-dispatcher-routing-rules-1:1.14.0-14.el8.noarch
.........
```

You can *grep* for an expression to narrow down your search. For example, to find whether the BaseOS repo includes the *zsh* package, run the following:

```
[user1@server1 ~]$ sudo dnf repoquery --repo BaseOS |grep zsh
Last metadata expiration check: 0:22:59 ago on Tue 24 Sep 2019 04:46:59 PM EDT.
zsh-0:5.5.1-6.el8.x86_64
```

To list all installed packages on the system:

```
[user1@server1 ~]$ sudo dnf list installed
Installed Packages
GConf2.x86_64                           3.2.6-22.el8                          @AppStream
ModemManager.x86_64                     1.8.0-1.el8                           @anaconda
ModemManager-glib.x86_64                1.8.0-1.el8                           @anaconda
NetworkManager.x86_64                   1:1.14.0-14.el8                       @anaconda
NetworkManager-adsl.x86_64              1:1.14.0-14.el8                       @anaconda
NetworkManager-bluetooth.x86_64         1:1.14.0-14.el8                       @anaconda
NetworkManager-config-server.noarch     1:1.14.0-14.el8                       @anaconda
NetworkManager-libnm.x86_64             1:1.14.0-14.el8                       @anaconda
NetworkManager-team.x86_64              1:1.14.0-14.el8                       @anaconda
.........
```

The graphic above shows the output in three columns: package name, package version, and the repo it was installed from. @anaconda means the package was installed at the time of RHEL installation.

To list all installed packages and all packages available for installation from all enabled repositories:

```
[user1@server1 ~]$ sudo dnf list
Installed Packages
GConf2.x86_64                                3.2.6-22.el8                                @AppStream
ModemManager.x86_64                          1.8.0-1.el8                                 @anaconda
ModemManager-glib.x86_64                     1.8.0-1.el8                                 @anaconda
NetworkManager.x86_64                        1:1.14.0-14.el8                             @anaconda
NetworkManager-adsl.x86_64                   1:1.14.0-14.el8                             @anaconda
NetworkManager-bluetooth.x86_64              1:1.14.0-14.el8                             @anaconda
.........
Available Packages
CUnit.i686                                   2.1.3-17.el8                                AppStream
CUnit.x86_64                                 2.1.3-17.el8                                AppStream
GConf2.i686                                  3.2.6-22.el8                                AppStream
Judy.x86_64                                  1.0.5-18.module+el8+2765+cfa4f87b           AppStream
LibRaw.i686                                  0.19.1-1.el8                                AppStream
LibRaw.x86_64                                0.19.1-1.el8                                AppStream
.........
```

The @ sign that precedes a repository name in column 3 identifies the package as installed.

To list all packages available from all enabled repositories that should be able to update:

```
[user1@server1 ~]$ sudo dnf list updates
```

To list whether a package (*bc*, for instance) is installed or available for installation from any enabled repository:

```
[user1@server1 ~]$ sudo dnf list bc
```

To list all installed packages whose names begin with the string “gnome” followed by any number of characters:

```
[user1@server1 ~]$ sudo dnf list installed gnome*
Installed Packages
gnome-autoar.x86_64                                  0.2.3-1.el8              @AppStream
gnome-bluetooth.x86_64                               1:3.28.2-2.el8           @AppStream
gnome-bluetooth-libs.x86_64                          1:3.28.2-2.el8           @AppStream
gnome-boxes.x86_64                                   3.28.5-3.el8             @AppStream
gnome-calculator.x86_64                              3.28.2-1.el8             @AppStream
.........
```

To list recently added packages:

```
[user1@server1 ~]$ sudo dnf list recent
```

Refer to the *repoquery* and *list* subsections of the *dnf* command manual pages for more options and examples.

Installing and Updating Packages

Installing a package creates the necessary directory tree for the specified and dependent packages, installs the required files, and runs any post-installation steps. If the package being loaded is already present, the *dnf* command updates it to the latest available version. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is entered with the command.

The following attempts to install a package called *ypbind*, but proceeds with an update if it detects the presence of an older version:

```
[user1@server1 ~]$ sudo dnf install ypbind
Dependencies resolved.
================================================================================
 Package             Arch              Version                 Repository            Size
================================================================================
Installing:
 ypbind              x86_64            3:2.5-2.el8             AppStream             69 k
Installing dependencies:
 yp-tools            x86_64            4.2.3-1.el8             AppStream             90 k
 nss_nis             x86_64            3.0-8.el8               BaseOS                41 k

Transaction Summary
================================================================================
Install  3 Packages

Total size: 200 k
Installed size: 424 k
Is this ok [y/N]: y
Downloading Packages:

Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : nss_nis-3.0-8.el8.x86_64                               1/3
  Running scriptlet: nss_nis-3.0-8.el8.x86_64                               1/3
  Installing       : ypbind-3:2.5-2.el8.x86_64                              2/3
  Running scriptlet: ypbind-3:2.5-2.el8.x86_64                              2/3
  Installing       : yp-tools-4.2.3-1.el8.x86_64                            3/3
  Running scriptlet: yp-tools-4.2.3-1.el8.x86_64                            3/3
  Verifying        : yp-tools-4.2.3-1.el8.x86_64                            1/3
  Verifying        : ypbind-3:2.5-2.el8.x86_64                              2/3
  Verifying        : nss_nis-3.0-8.el8.x86_64                               3/3
Installed products updated.

Installed:
  ypbind-3:2.5-2.el8.x86_64    yp-tools-4.2.3-1.el8.x86_64    nss_nis-3.0-8.el8.x86_64

Complete!
```

The above *dnf* command example resolved dependencies and showed a list of the packages that it would install. It exhibited the size of the packages and the amount of disk space that the installation would consume. It downloaded the packages after confirmation to proceed and installed them. It completed the installation after every package was verified. A list of the installed packages was displayed at the bottom of the output.

To install or update a package called *dcraw* located locally at */mnt/AppStream/Packages*:

```
[user1@server1 ~]$ sudo dnf localinstall /mnt/AppStream/Packages/dcraw-9.27.0-9.el8.x86_64
.rpm
```

To update an installed package (*autofs*, for example) to the latest available version. Note that *dnf* will fail if the specified package is not already installed.

```
[user1@server1 ~]$ sudo dnf update autofs
Package autofs available, but not installed.
No match for argument: autofs
Error: No packages marked for upgrade.
```

To update all installed packages to the latest available versions:

```
[user1@server1 ~]$ sudo dnf -y update
```

Refer to the *install* and *update* subsections of the *dnf* command manual pages for more options and examples.

Exhibiting Package Information

Displaying information for a package shows its name, architecture it is built for, version, release, size, whether it is installed or available for installation, repo name it was installed or is available from, short and long descriptions, license, and so on. This information can be viewed by supplying the *info* subcommand to *dnf*.

To view information about a package called *autofs*:

```
[user1@server1 ~]$ dnf info autofs
Available Packages
Name         : autofs
Epoch        : 1
Version      : 5.1.4
Release      : 29.el8
Arch         : x86_64
Size         : 755 k
Source       : autofs-5.1.4-29.el8.src.rpm
Repo         : BaseOS
Summary      : A tool for automatically mounting and unmounting filesystems
License      : GPLv2+
Description  : autofs is a daemon which automatically mounts filesystems when you use
             : them, and unmounts them later when you are not using them.  This can
             : include network filesystems, CD-ROMs, floppies, and so forth.
```

The output shows that *autofs* is not currently installed, but it is available for installation from the BaseOS repo. The *info* subcommand automatically determines whether the specified package is installed or not.

Refer to the *info* subsection of the *dnf* command manual pages for more options available for viewing package information.

Removing Packages

Removing a package uninstalls it and removes all associated files and directory structure. It also erases any dependencies as part of the deletion process. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is specified at the command line.

To remove a package called *ypbind*:

```
[user1@server1 ~]$ sudo dnf remove ypbind
Dependencies resolved.
================================================================================
 Package             Arch             Version                Repository        Size
================================================================================
Removing:
 ypbind              x86_64           3:2.5-2.el8            @AppStream       100 k
Removing unused dependencies:
 nss_nis             x86_64           3.0-8.el8              @BaseOS           88 k
 yp-tools            x86_64           4.2.3-1.el8            @AppStream       236 k

Transaction Summary
================================================================================
Remove  3 Packages

Freed space: 424 k
Is this ok [y/N]: y
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Running scriptlet: yp-tools-4.2.3-1.el8.x86_64                            1/1
  Erasing          : yp-tools-4.2.3-1.el8.x86_64                            1/3
  Running scriptlet: ypbind-3:2.5-2.el8.x86_64                              2/3
  Erasing          : ypbind-3:2.5-2.el8.x86_64                              2/3
  Running scriptlet: ypbind-3:2.5-2.el8.x86_64                              2/3
  Erasing          : nss_nis-3.0-8.el8.x86_64                               3/3

  Running scriptlet: nss_nis-3.0-8.el8.x86_64                               3/3
  Verifying        : nss_nis-3.0-8.el8.x86_64                               1/3
  Verifying        : yp-tools-4.2.3-1.el8.x86_64                            2/3
  Verifying        : ypbind-3:2.5-2.el8.x86_64                              3/3
Installed products updated.

Removed:
  ypbind-3:2.5-2.el8.x86_64    nss_nis-3.0-8.el8.x86_64    yp-tools-4.2.3-1.el8.x86_64

Complete!
```

The above output resolved dependencies and showed a list of the packages that it would remove. It displayed the amount of disk space that their removal would free up. After confirmation to proceed, it erased the identified packages and verified their removal. A list of the removed packages was exhibited at the bottom of the output.

Refer to the *remove* subsection of the *dnf* command manual pages for more options and examples available for removing packages.

Exercise 10-2: Manipulate Individual Packages

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will perform management operations on a package called *cifs-utils*. You will determine if this package is already installed and if it is available for installation. You will display its information before installing it. You will install the package and exhibit its information. Finally, you will remove the package along with its dependencies and confirm the removal.

1. Check whether the *cifs-utils* package is already installed:

```
[userl@serverl ~]$ dnf list installed |grep cifs-utils
[userl@serverl ~]$
```

The command returned to the prompt without any output, which implies that the specified package is not installed.

2. Determine if the *cifs-utils* package is available for installation:

```
[userl@serverl ~]$ dnf repoquery cifs-utils
cifs-utils-0:6.8-2.el8.x86_64
```

The package is available for installation. The output shows its version as well.

3. Display detailed information about the package:

```
[userl@serverl ~]$ dnf info cifs-utils
Available Packages
Name         : cifs-utils
Version      : 6.8
Release      : 2.el8
Arch         : x86_64
Size         : 93 k
Source       : cifs-utils-6.8-2.el8.src.rpm
Repo         : BaseOS
Summary      : Utilities for mounting and managing CIFS mounts
URL          : http://linux-cifs.samba.org/cifs-utils/
License      : GPLv3
Description  : The SMB/CIFS protocol is a standard file sharing protocol widely
             : deployed on Microsoft Windows machines. This package contains
             : tools for mounting shares on Linux using the SMB/CIFS protocol.
             : The tools in this package work in conjunction with support in the
             : kernel to allow one to mount a SMB/CIFS share onto a client and
             : use it as if it were a standard Linux file system.
```

The output is indicative of the fact that the package is available for installation from the BaseOS repo.

4. Install the package:

```
[user1@server1 ~]$ sudo dnf install -y cifs-utils
Dependencies resolved.
================================================================================
 Package             Arch           Version              Repository       Size
================================================================================
Installing:
 cifs-utils          x86_64         6.8-2.el8            BaseOS           93 k

Transaction Summary
================================================================================
Install  1 Package

Total size: 93 k
Installed size: 189 k
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : cifs-utils-6.8-2.el8.x86_64                            1/1
  Running scriptlet: cifs-utils-6.8-2.el8.x86_64                            1/1
  Verifying        : cifs-utils-6.8-2.el8.x86_64                            1/1
Installed products updated.

Installed:
  cifs-utils-6.8-2.el8.x86_64

Complete!
```

5. Display the package information again:

```
[user1@server1 ~]$ dnf info cifs-utils
Installed Packages
Name         : cifs-utils
Version      : 6.8
Release      : 2.el8
Arch         : x86_64
Size         : 189 k
Source       : cifs-utils-6.8-2.el8.src.rpm
Repo         : @System
From repo    : BaseOS
```

The output reveals that the package is now installed.

6. Remove the package:

```
[user1@server1 ~]$ sudo dnf remove -y cifs-utils
Dependencies resolved.
================================================================================
 Package             Arch            Version             Repository        Size
================================================================================
Removing:
 cifs-utils          x86_64          6.8-2.el8           @BaseOS          189 k

Transaction Summary
================================================================================
Remove  1 Package

Freed space: 189 k
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Running scriptlet: cifs-utils-6.8-2.el8.x86_64                            1/1
  Erasing          : cifs-utils-6.8-2.el8.x86_64                            1/1
  Running scriptlet: cifs-utils-6.8-2.el8.x86_64                            1/1
  Verifying        : cifs-utils-6.8-2.el8.x86_64                            1/1
Installed products updated.

Removed:
  cifs-utils-6.8-2.el8.x86_64

Complete!
```

7. Confirm the removal:

```
[user1@server1 ~]$ dnf list installed | grep cifs-utils
[user1@server1 ~]$
```

The no output above confirms that the package is not loaded on the system.

Determining Provider and Searching Package Metadata

Determining package contents includes search operations on installed and available packages. For instance, you can determine what package a specific file belongs to or which package comprises a certain string. The following examples show how to carry out these tasks.

To search for packages that contain a specific file such as */etc/passwd*, use the *provides* or the *whatprovides* subcommand with *dnf*:

```
[user1@server1 ~]$ dnf provides /etc/passwd
setup-2.12.2-1.el8.noarch : A set of system configuration and setup files
Repo        : @System
Matched from:
Filename    : /etc/passwd

setup-2.12.2-1.el8.noarch : A set of system configuration and setup files
Repo        : BaseOS
Matched from:
Filename    : /etc/passwd
```

The output returns two instances of the file. The first one indicates that the *passwd* file is part of a package called *setup*, which was installed during RHEL installation, and the second instance states that the *setup* package is part of the BaseOS repository.

With the *provides* (*whatprovides*) subcommand, you can also use a wildcard character for filename expansion. For example, the following command will list all packages that contain filenames beginning with “system-config” followed by any number of characters:

```
[user1@server1 ~]$ dnf whatprovides /usr/bin/system-config*
abrt-gui-2.10.9-10.el8.x86_64 : abrt's gui
Repo        : AppStream
Matched from:
Filename    : /usr/bin/system-config-abrt

policycoreutils-gui-2.8-16.1.el8.noarch : SELinux configuration GUI
Repo        : AppStream
Matched from:
Filename    : /usr/bin/system-config-selinux
```

To search for all the packages that match the specified string in their name or summary:

```
[user1@server1 ~]$ dnf search system-config
============================== Name Matched: system-config ==============================
system-config-printer-libs.noarch : Libraries and shared code for printer administration
                                  : tool
system-config-printer-udev.x86_64 : Rules for udev for automatic configuration of USB
                                  : printers
system-config-printer-libs.noarch : Libraries and shared code for printer administration
                                  : tool
system-config-printer-udev.x86_64 : Rules for udev for automatic configuration of USB
                                  : printers
============================ Summary Matched: system-config ============================
cups-pk-helper.x86_64 : A helper that makes system-config-printer use PolicyKit
cups-pk-helper.x86_64 : A helper that makes system-config-printer use PolicyKit
```

The above outcome depicts six matches, four in package names and two in summaries.

Package Group Management

The *dnf* command can be used to perform ample operations on package groups by specifying the *group* subcommand with it. The following subsections elaborate with examples on using this tool to list, install, query, and remove groups of packages.

Listing Available and Installed Package Groups

The *group list* subcommand can be used with *dnf* to list the package groups available for installation from either or both repos, as well as to list the package groups that are already installed on the system.

To list all available and installed package groups from all repositories:

```
[user1@server1 ~]$ dnf group list
Available Environment Groups:
   Server
   Minimal Install
   Workstation
   Virtualization Host
   Custom Operating System
Installed Environment Groups:
   Server with GUI
Installed Groups:
   Container Management
   Headless Management
Available Groups:
   .NET Core Development
   RPM Development Tools
   Smart Card Support
   Development Tools
   Graphical Administration Tools
   Legacy UNIX Compatibility
   Network Servers
   Scientific Support
   Security Tools
   System Tools
```

The output reveals two categories of package groups: an *environment group* and a *group*. An environment group is a larger collection of RHEL packages that provides all necessary software to build the operating system foundation for a desired purpose. The Server with GUI environment group was selected at the time of installing *server1*, which installed a multitude of packages as well as the GNOME desktop. See Table 1-2 in Chapter 01 “Local Installation” for a description of other environment groups.

A group, on the other hand, is a small bunch of RHEL packages that serve a common purpose. It also saves time on the deployment of individual and dependent packages. The above output shows two installed and several available package groups.

To display the number of installed and available package groups:

```
[user1@server1 ~]$ sudo dnf group summary
Installed Groups: 2
Available Groups: 10
```

To list all installed and available package groups including those that are hidden:

```
[user1@server1 ~]$ sudo dnf group list hidden
Available Environment Groups:
   Server
   Minimal Install
   Workstation
   Virtualization Host
   Custom Operating System
Installed Environment Groups:
   Server with GUI
Installed Groups:
   Container Management
   Guest Desktop Agents
   Internet Browser
   Multimedia
   Printing Client
   Core
   Fonts
   GNOME
........
```

Try *group list* with --installed and --available options to narrow down the output list.

To list all packages that a specific package group such as *Base* contains:

```
[user1@server1 ~]$ sudo dnf group info Base
Group: Base
 Description: The standard installation of Red Hat Enterprise Linux.
 Mandatory Packages:
   acl
   at
   attr
   bc
   cpio
   crontabs
   cyrus-sasl-plain
   dbus
   ed
   file
   iptstate
   irqbalance
   kpatch
   logrotate
   lsof
.........
```

You may use the -v option with the *group info* subcommand for more information. Refer to the *group list* and *group info* subsections of the *dnf* command manual pages for more details.

Installing and Updating Package Groups

Installing a package group creates the necessary directory structure for all the packages included in the group and all dependent packages, installs the required files, and runs any post-installation steps. If the package group is being loaded or part of it is already present, the command attempts to update all the packages included in the group to the latest available versions. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is entered with the command.

The following example attempts to install a package group called *smart card support*, but proceeds with an update if it detects the presence of an older version. You may enclose

the group name within either single or double quotation marks.

```
[user1@server1 ~]$ sudo dnf -y groupinstall "smart card support"
Dependencies resolved.
================================================================================
 Package               Arch          Version             Repository        Size
================================================================================
Installing group/module packages:
 esc                   x86_64        1.1.2-7.el8         AppStream        175 k
Installing dependencies:
 opensc                x86_64        0.19.0-4.el8        BaseOS           1.2 M
 pcsc-lite             x86_64        1.8.23-3.el8        BaseOS           108 k
 pcsc-lite-ccid        x86_64        1.4.29-3.el8        BaseOS           315 k
 pcsc-lite-libs        x86_64        1.8.23-3.el8        BaseOS            44 k
Installing Groups:
 Smart Card Support

Transaction Summary
================================================================================
Install  5 Packages

Total size: 1.8 M
Installed size: 5.8 M
.........
Installed products updated.

Installed:
  esc-1.1.2-7.el8.x86_64                opensc-0.19.0-4.el8.x86_64
  pcsc-lite-1.8.23-3.el8.x86_64         pcsc-lite-ccid-1.4.29-3.el8.x86_64
  pcsc-lite-libs-1.8.23-3.el8.x86_64

Complete!
```

The output discloses that the command installed five packages located across the two repositories to complete the installation of the package group.

To update the *smart card support* package group to the latest version:

```
[user1@server1 ~]$ sudo dnf groupupdate "smart card support" -y
```

The above command will update all the packages within the package group to their latest versions if it has access to them. Refer to the *group install* and *group update* subsections of the *dnf* command manual pages for more details.

Removing Package Groups

Removing a package group uninstalls all the included packages and deletes all associated files and directory structure. It also erases any dependencies as part of the deletion process. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is specified at the command line.

To erase the *smart card support* package group that was installed:

```
[user1@server1 ~]$ sudo dnf -y groupremove 'smart card support'
Dependencies resolved.
================================================================================
 Package              Arch         Version               Repository        Size
================================================================================
Removing:
 esc                  x86_64       1.1.2-7.el8           @AppStream       507 k
Removing unused dependencies:
 opensc               x86_64       0.19.0-4.el8          @BaseOS          3.6 M
 pcsc-lite            x86_64       1.8.23-3.el8          @BaseOS          224 k
 pcsc-lite-ccid       x86_64       1.4.29-3.el8          @BaseOS          1.4 M
 pcsc-lite-libs       x86_64       1.8.23-3.el8          @BaseOS           53 k
Removing Groups:
 Smart Card Support

Transaction Summary
================================================================================
Remove  5 Packages

Freed space: 5.8 M
Running transaction check
Transaction check succeeded.
........
Installed products updated.

Removed:
  esc-1.1.2-7.el8.x86_64                opensc-0.19.0-4.el8.x86_64
  pcsc-lite-1.8.23-3.el8.x86_64         pcsc-lite-ccid-1.4.29-3.el8.x86_64
  pcsc-lite-libs-1.8.23-3.el8.x86_64

Complete!
```

The above output resolved dependencies and showed a list of the packages that it would remove. It displayed the amount of disk space that their removal would free up. After confirmation to proceed, it erased the identified packages

and verified their removal. A list of the removed packages was exposed at the bottom of the output.

Refer to the *remove* subsection of the *dnf* command manual pages for more details.

Exercise 10-3: Manipulate Package Groups

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will perform management operations on a package group called *system tools*. You will determine if this group is already installed and if it is available for installation. You will list the packages it contains and install it. Finally, you will remove the group along with its dependencies and confirm the removal.

1. Check whether the *system tools* package group is already installed:

```
[user1@server1 ~]$ dnf group list installed
Installed Environment Groups:
   Server with GUI
Installed Groups:
   Container Management
   Headless Management
```

The output does not show the *system tools* group installed.

2. Determine if the *system tools* group is available for installation:

```
[user1@server1 ~]$ dnf group list available
Available Environment Groups:
   Server
   Minimal Install
   Workstation
   Virtualization Host
   Custom Operating System
Available Groups:
   .NET Core Development
   RPM Development Tools
   Smart Card Support
   Development Tools
   Graphical Administration Tools
   Legacy UNIX Compatibility
   Network Servers
   Scientific Support
   Security Tools
   System Tools
```

The group name is exhibited at the bottom of the list under the available groups.

3. Display the list of packages this group contains:

```
[user1@server1 ~]$ dnf group info 'system tools'
Group: System Tools
 Description: This group is a collection of various tools for the system, such a
s the client for connecting to SMB shares and tools to monitor network traffic.
 Default Packages:
   NetworkManager-libreswan
   chrony
   cifs-utils
   libreswan
   nmap
   openldap-clients
   samba-client
   setserial
   tigervnc
   tmux
   xdelta
   zsh
 Optional Packages:
   PackageKit-command-not-found
   aide
........
```

The output returns a long list of packages that are included in this group. All of the packages will be installed as part of the group installation.

4. Install the group:

```
[user1@server1 ~]$ sudo dnf group install 'system tools'
Dependencies resolved.
================================================================================
 Package                      Arch       Version            Repository     Size
================================================================================
Installing group/module packages:
 NetworkManager-libreswan     x86_64     1.2.10-3.el8       AppStream     120 k
 libreswan                    x86_64     3.27-9.el8         AppStream     1.3 M
 nmap                         x86_64     2:7.70-4.el8       AppStream     5.8 M
 tigervnc                     x86_64     1.9.0-9.el8        AppStream     276 k
 cifs-utils                   x86_64     6.8-2.el8          BaseOS         93 k
 openldap-clients             x86_64     2.4.46-9.el8       BaseOS        202 k
 samba-client                 x86_64     4.9.1-8.el8        BaseOS        636 k
 setserial                    x86_64     2.17-45.el8        BaseOS         32 k
 tmux                         x86_64     2.7-1.el8          BaseOS        317 k
 xdelta                       x86_64     3.1.0-4.el8        BaseOS         96 k
Installing dependencies:
 fltk                         x86_64     1.3.4-5.el8        AppStream     575 k
 ldns                         x86_64     1.7.0-20.el8       AppStream     165 k
 mesa-libGLU                  x86_64     9.0.0-15.el8       AppStream     185 k
 nss-tools                    x86_64     3.41.0-5.el8       AppStream     567 k
 tigervnc-icons               noarch     1.9.0-9.el8        AppStream      46 k
Installing Groups:
 System Tools

Transaction Summary
================================================================================
Install  15 Packages

Total size: 10 M
Installed size: 39 M
.........
Installed products updated.

Installed:
  NetworkManager-libreswan-1.2.10-3.el8.x86_64
  libreswan-3.27-9.el8.x86_64
  nmap-2:7.70-4.el8.x86_64
  tigervnc-1.9.0-9.el8.x86_64
  cifs-utils-6.8-2.el8.x86_64
  openldap-clients-2.4.46-9.el8.x86_64
  samba-client-4.9.1-8.el8.x86_64
  setserial-2.17-45.el8.x86_64
  tmux-2.7-1.el8.x86_64
  xdelta-3.1.0-4.el8.x86_64
  fltk-1.3.4-5.el8.x86_64
  ldns-1.7.0-20.el8.x86_64
  mesa-libGLU-9.0.0-15.el8.x86_64
  nss-tools-3.41.0-5.el8.x86_64
  tigervnc-icons-1.9.0-9.el8.noarch

Complete!
```

5. Remove the group:

```
[user1@server1 ~]$ sudo dnf group remove 'system tools' -y
Dependencies resolved.
================================================================================
 Package                      Arch       Version           Repository      Size
================================================================================
Removing:
 NetworkManager-libreswan     x86_64     1.2.10-3.el8      @AppStream     426 k
 cifs-utils                   x86_64     6.8-2.el8         @BaseOS        189 k
 libreswan                    x86_64     3.27-9.el8        @AppStream     4.4 M
 nmap                         x86_64     2:7.70-4.el8      @AppStream      24 M
 openldap-clients             x86_64     2.4.46-9.el8      @BaseOS        644 k
 samba-client                 x86_64     4.9.1-8.el8       @BaseOS        2.1 M
 setserial                    x86_64     2.17-45.el8       @BaseOS         37 k
 tigervnc                     x86_64     1.9.0-9.el8       @AppStream     858 k
 tmux                         x86_64     2.7-1.el8         @BaseOS        770 k
 xdelta                       x86_64     3.1.0-4.el8       @BaseOS        183 k
Removing unused dependencies:
 fltk                         x86_64     1.3.4-5.el8       @AppStream     1.6 M
 ldns                         x86_64     1.7.0-20.el8      @AppStream     428 k
 mesa-libGLU                  x86_64     9.0.0-15.el8      @AppStream     477 k
 nss-tools                    x86_64     3.41.0-5.el8      @AppStream     3.6 M
 tigervnc-icons               noarch     1.9.0-9.el8       @AppStream      33 k
Removing Groups:
 System Tools

Transaction Summary
================================================================================
Remove  15 Packages

Freed space: 39 M

........
Installed products updated.

Removed:
  NetworkManager-libreswan-1.2.10-3.el8.x86_64  cifs-utils-6.8-2.el8.x86_64
  libreswan-3.27-9.el8.x86_64                   nmap-2:7.70-4.el8.x86_64
  openldap-clients-2.4.46-9.el8.x86_64          samba-client-4.9.1-8.el8.x86_64
  setserial-2.17-45.el8.x86_64                  tigervnc-1.9.0-9.el8.x86_64
  tmux-2.7-1.el8.x86_64                         xdelta-3.1.0-4.el8.x86_64
  fltk-1.3.4-5.el8.x86_64                       ldns-1.7.0-20.el8.x86_64
  mesa-libGLU-9.0.0-15.el8.x86_64               nss-tools-3.41.0-5.el8.x86_64
  tigervnc-icons-1.9.0-9.el8.noarch

Complete!
```

6. Confirm the removal:

```
[user1@server1 ~]$ dnf group list installed
Installed Environment Groups:
   Server with GUI
Installed Groups:
   Container Management
   Headless Management
```

The package group is not listed in the above output, which confirms its removal.

Module Management

Modules are a new concept introduced in RHEL 8, and the *yum*/*dnf* command has also been modified to handle them. The command can be used to perform a number of operations on modules by specifying the *module* subcommand with it. The following subsections elaborate with examples on using this tool to list, enable, install, query, remove, and disable modules.

Listing Available and Installed Modules

The *dnf* command can list the modules available for installation from either or both repos, as well as the modules that are already installed on the system.

To list all modules along with their stream, profile, and summary information available from all configured repos:

```
[user1@server1 ~]$ dnf module list
Last metadata expiration check: 0:14:50 ago on Tue 24 Sep 2019 11:17:09 PM EDT.
AppStream
Name                 Stream       Profiles       Summary
389-ds               1.4                         389 Directory Server (base)
ant                  1.10 [d]     common [d]     Java build tool
container-tools      1.0          common [d]     Common tools and dependencies for contain
                                                 er runtimes
container-tools      rhel8 [d][e] common [d]     Common tools and dependencies for contain
                                                 er runtimes
freeradius           3.0 [d]      server [d]     High-performance and highly configurable
                                                 free RADIUS server

.........
Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

You may limit the output to a list of modules available from a specific repo such as AppStream by adding *--repo AppStream* to the above. The letters *d*, *e*, *x*, and *i* under the Stream and Profiles columns and at the last line of the output indicate their status as default (d), enabled (e), disabled (x), or installed (i).

To list all the streams for a specific module such as *perl* and display their status:

```
[user1@server1 ~]$ dnf module list perl
AppStream
Name  Stream       Profiles             Summary
perl  5.24         common [d], minimal  Practical Extraction and Report Language
perl  5.26 [d][e]  common [d], minimal  Practical Extraction and Report Language

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

The above output reveals that there are two streams (5.24 and 5.26) available for the *perl* module from the AppStream repo, and the current default and enabled version is the latter of the two.

To modify the above and list only the specified stream (5.24) for the module *perl*:

```
[user1@server1 ~]$ dnf module list perl:5.24
AppStream
Name   Stream   Profiles              Summary
perl   5.24     common [d], minimal   Practical Extraction and Report Language

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

To list all enabled module streams:

```
[user1@server1 ~]$ dnf module list --enabled
AppStream
Name               Stream       Profiles  Summary
container-tools    rhel8 [d][e] common [d Common tools and dependencies for cont
                                ] [i]     ainer runtimes
llvm-toolset       rhel8 [d][e] common [d LLVM
                                ]
perl               5.26 [d][e]  common [d Practical Extraction and Report Langua
                                ] [i], mi ge
                                nimal
perl-DBD-SQLite    1.58 [d][e]  common [d SQLite DBI driver
                                ]
perl-DBI           1.641 [d][e] common [d A database access API for Perl
                                ]
python36           3.6 [d][e]   common [d Python programming language, version 3
                                ], build  .6
satellite-5-client 1.0 [d][e]   common [d Red Hat Satellite 5 client packages
                                ], gui
virt               rhel [d][e]  common [d Virtualization module
                                ]

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

Similarly, you can use the --installed and --disabled options with **dnf module list** to output only the installed or the disabled streams.

Refer to the *module list* subsection of the *dnf* command manual pages for more details.

Installing and Updating Modules

Installing a module creates the necessary directory tree for all the packages included in the module and all dependent packages, installs the required files for the selected profile, and runs any post-installation steps. If the module being loaded or a part of it is already present, the command attempts to update all the packages included in the profile to the latest available versions. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is entered with the command.

To install the *perl* module using its default stream and default profile:

```
[user1@server1 ~]$ sudo dnf -y module install perl

 Dependencies resolved.
 ================================================================================
  Package                       Arch     Version              Repository   Size
 ================================================================================
 Installing group/module packages:
  perl                          x86_64   4:5.26.3-416.el8     AppStream    72 k
 Installing dependencies:
  dwz                           x86_64   0.12-9.el8           AppStream   109 k
  efi-srpm-macros               noarch   3-2.el8              AppStream    22 k
  ghc-srpm-macros               noarch   1.4.2-7.el8          AppStream   9.4 k
  go-srpm-macros                noarch   2-16.el8             AppStream    14 k
  ocaml-srpm-macros             noarch   5-4.el8              AppStream   9.5 k
  openblas-srpm-macros          noarch   2-2.el8              AppStream   8.0 k
  perl-Algorithm-Diff           noarch   1.1903-9.el8         AppStream    52 k
  perl-Archive-Tar              noarch   2.30-1.el8           AppStream    79 k
  perl-Archive-Zip              noarch   1.60-3.el8           AppStream   108 k

........
  redhat-rpm-config-116-1.el8.noarch
  rust-srpm-macros-5-2.el8.noarch
  systemtap-sdt-devel-4.0-7.el8.x86_64
  make-1:4.2.1-9.el8.x86_64
  python3-pyparsing-2.1.10-7.el8.noarch

Complete!
```

The default stream and the default profile are the same, which is 5.26, and the above command installed it.

To update a module called *squid* to the latest version:

```
[user1@server1 ~]$ sudo dnf module update squid -y
```

To install the profile “common” with stream “rhel8” for the *container-tools* module (run **dnf module list** to view available *container-tools* modules):

```
[user1@server1 ~]$ sudo dnf module install container-tools:rhel8/common
Dependencies resolved.
================================================================================
 Package  Arch   Version                                        Repository
                                                                          Size
================================================================================
Installing group/module packages:
 oci-umount
          x86_64 2:2.3.4-2.git87f9237.module+el8+2769+577ad176  AppStream  38 k
 skopeo   x86_64 1:0.1.32-3.git1715c90.module+el8+2769+577ad176 AppStream 4.8 M
Installing module profiles:
 container-tools/common


Transaction Summary
================================================================================
Install  2 Packages

Total size: 4.8 M
Installed size: 19 M
Is this ok [y/N]: y
Downloading Packages:
........
```

The syntax of entering module, stream, and profile together (module:stream/profile) is illustrated on the above command line.

Displaying Module Information

Exhibiting information about a module shows its name, stream, version, list of profiles, default profile, repo name it was installed or is available from, summary, description, and artifacts. This information can be viewed by supplying *module info* with *dnf*. Let’s take a look at the following examples.

To list all profiles available for the module *perl*:

```
[user1@server1 ~]$ dnf module info --profile perl
Name    : perl:5.24:820190207164249:ee766497:x86_64
common  : perl-core
minimal : perl

Name    : perl:5.26:820181219174508:9edba152:x86_64
common  : perl
minimal : perl-interpreter
```

To limit the output to a particular stream such as 5.26:

```
[user1@server1 ~]$ dnf module info --profile perl:5.26
Name    : perl:5.26:820181219174508:9edba152:x86_64
common  : perl
minimal : perl-interpreter
```

To display details for a specific module stream (a non-default stream), specify the module stream with the module name:

```
[user1@server1 ~]$ dnf module info --profile perl:5.24
Name    : perl:5.24:820190207164249:ee766497:x86_64
common  : perl-core
minimal : perl
```

Refer to the *module info* subsection of the *dnf* command manual pages for more details.

Removing Modules

Removing a module will uninstall all the included packages and delete all associated files and directory structure. It also erases any dependencies as part of the deletion process. By default, *dnf* prompts for a yes or no confirmation unless the -y flag is specified at the command line.

To remove the *container-tools* module with “rhel8” stream:

```
[user1@server1 ~]$ sudo dnf module remove container-tools:rhel8
```

```
Dependencies resolved.
================================================================================
 Package Arch   Version                                    Repository  Size
================================================================================
Removing:
 oci-umount
        x86_64 2:2.3.4-2.git87f9237.module+el8+2769+577ad176  @AppStream  67 k
 skopeo  x86_64 1:0.1.32-3.git1715c90.module+el8+2769+577ad176 @AppStream  19 M
Removing module profiles:
 container-tools/common


Transaction Summary
================================================================================
Remove  2 Packages

Freed space: 19 M
Is this ok [y/N]: y
........
Removed:
  oci-umount-2:2.3.4-2.git87f9237.module+el8+2769+577ad176.x86_64
  skopeo-1:0.1.32-3.git1715c90.module+el8+2769+577ad176.x86_64

Complete!
```

The above output resolved dependencies and showed a list of the packages that it would remove. It displayed the amount of disk space that their removal would free up. After confirmation to proceed, it erased the identified packages and verified their removal. A list of the removed packages was exposed at the bottom of the output.

Refer to the *module remove* subsection of the *dnf* command manual pages for more details.

Exercise 10-4: Manipulate Modules

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will perform management operations on a module called *postgresql*. You will determine if this module is already installed and if it is available for installation. You will show its information and install the default profile for stream “10”. Finally, you will remove the module profile along with any dependencies and confirm the removal.

1. Check whether the *postgresql* module is already installed:

```
[user1@server1 ~]$ dnf module list postgresql
AppStream
Name         Stream   Profiles             Summary
postgresql   10 [d]   client, server [d]   PostgreSQL server and client module
postgresql   9.6      client, server [d]   PostgreSQL server and client module

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

The *postgresql* module is not currently installed as there is no indication (i). The default stream is “10” and the default profile is “server”. The module with all its streams and profiles is available from the AppStream repository.

2. Display detailed information about the default stream of the module:

```
[user1@server1 ~]$ dnf module info postgresql:10
Name             : postgresql
Stream           : 10 [d][a]
Version          : 820190104140132
Context          : 9edba152
Profiles         : client, server [d]
Default profiles : server
Repo             : AppStream
Summary          : PostgreSQL server and client module
Description      : PostgreSQL is an advanced Object-Relational database manageme
nt system (DBMS). The postgresql-server package contains the programs needed to
create and run a PostgreSQL server, which will in turn allow you to create and m
aintain PostgreSQL databases. The base postgresql package contains the client pr
ograms that you'll need to access a PostgreSQL DBMS server.
Artifacts        : postgresql-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-contrib-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-docs-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-plperl-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-plpython3-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-pltcl-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-server-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-server-devel-0:10.6-1.module+el8+2469+5ecd5aae.x86
_64
                 : postgresql-static-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-test-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-test-rpm-macros-0:10.6-1.module+el8+2469+5ecd5aae.
x86_64
                 : postgresql-upgrade-0:10.6-1.module+el8+2469+5ecd5aae.x86_64
                 : postgresql-upgrade-devel-0:10.6-1.module+el8+2469+5ecd5aae.x8
6_64

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled, [a]ctive]
```

3. Install the module with default profile for stream “10”:

```
[user1@server1 ~]$ sudo dnf module install --profile postgresql:10 -y
Dependencies resolved.
================================================================================
 Package            Arch    Version                              Repository  Size
================================================================================
Installing group/module packages:
 postgresql-server  x86_64  10.6-1.module+el8+2469+5ecd5aae    AppStream  5.1 M
Installing dependencies:
 libpq              x86_64  10.5-1.el8                         AppStream  188 k
 postgresql         x86_64  10.6-1.module+el8+2469+5ecd5aae    AppStream  1.5 M
Installing module profiles:
 postgresql/server
Enabling module streams:
 postgresql                 10

........

Installed:
  postgresql-server-10.6-1.module+el8+2469+5ecd5aae.x86_64
  libpq-10.5-1.el8.x86_64
  postgresql-10.6-1.module+el8+2469+5ecd5aae.x86_64

Complete!
```

4. Display the module information again:

```
[user1@server1 ~]$ dnf module info postgresql:10
Name             : postgresql
Stream           : 10 [d][e][a]
Version          : 8201901041401 32
Context          : 9edba152
Profiles         : client, server [d] [i]
Default profiles : server
```

Line 2 from the bottom signifies that the default (d) profile “server” is now installed (i).

5. Remove the module profile for the stream:

```
[user1@server1 ~]$ sudo dnf module remove -y postgresql:10
Dependencies resolved.
================================================================================
 Package            Arch    Version                              Repository   Size
================================================================================
Removing:
 postgresql-server  x86_64  10.6-1.module+el8+2469+5ecd5aae   @AppStream   20 M
Removing unused dependencies:
 libpq              x86_64  10.5-1.el8                       @AppStream  702 k
 postgresql         x86_64  10.6-1.module+el8+2469+5ecd5aae   @AppStream  5.6 M
Removing module profiles:
 postgresql/server

Transaction Summary
================================================================================
Remove  3 Packages

........
Removed:
  postgresql-server-10.6-1.module+el8+2469+5ecd5aae.x86_64
  libpq-10.5-1.el8.x86_64
  postgresql-10.6-1.module+el8+2469+5ecd5aae.x86_64

Complete!
```

6. Confirm the removal:

```
[user1@server1 ~]$ dnf module info postgresql:10

 Name             : postgresql
 Stream           : 10 [d][e][a]
 Version          : 820190104140132
 Context          : 9edba152
 Profiles         : client, server [d]
 Default profiles : server
```

Line 2 from the bottom indicates the absence of (i) from the “server” profile, which means the profile is no longer installed.

Switching Module Streams

Switching module streams is typically performed to upgrade or downgrade the version of an installed module. The correct process for either operation is to uninstall the existing version provided by a stream alongside any dependencies that it has, switch to the other stream, and install the desired version. By default, installing a module from a stream automatically enables the stream if it was previously disabled, otherwise you can manually enable or disable it with the *dnf* command. You can have only one stream of a

given module enabled at a time. Attempting to enable another one for the same module automatically disables the current enabled stream.

Earlier in this section, you executed **dnf module list** and **dnf module info** against multiple modules to list and view information about them. One of the properties they expose is the enable/disable status of the module stream. Exercise 10-5 will show you how to switch from one stream to another of a module and install a different version from there.

Exercise 10-5: Install a Module from an Alternative Stream

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will downgrade a module to a lower version. You will remove the stream *perl* 5.26 and confirm its removal. You will manually enable the stream *perl* 5.24 and confirm its new status. You will install the new version of the module and display its information.

1. Check the current state of all *perl* streams:

```
[user1@server1 ~]$ dnf module list perl
AppStream
Name Stream      Profiles              Summary
perl 5.24        common [d], minimal   Practical Extraction and Report Language
perl 5.26 [d][e] common [d] [i], minima Practical Extraction and Report Language
                 l

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

The above output reveals several interesting facts about the *perl* module and its available and installed streams and profiles. In columns 1 and 2, perl 5.26 is the default (d) stream and it is currently enabled (e). In column 3, the default (d) profile for this stream is “common” and it is currently installed (i). The other stream for this module is

5.24, which is also available for installation from the AppStream repository.

2. Let’s remove perl 5.26:

```
[user1@server1 ~]$ sudo dnf module remove perl -y
Dependencies resolved.
================================================================================
 Package                      Arch    Version               Repository   Size
================================================================================
Removing:
 perl                         x86_64  4:5.26.3-416.el8      @AppStream    0
Removing unused dependencies:
 dwz                          x86_64  0.12-9.el8            @AppStream  226 k
 efi-srpm-macros              noarch  3-2.el8               @AppStream   38 k
.....
Removing module profiles:
 perl/common

Transaction Summary
================================================================================
Remove  112 Packages

Freed space: 27 M
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Erasing          : perl-4:5.26.3-416.el8.x86_64                         1/112
.....
```

The module profile for perl stream 5.26 was removed successfully.

3. Confirm the removal:

```
[user1@server1 ~]$ dnf module list perl
AppStream
Name  Stream       Profiles             Summary
perl  5.24         common [d], minimal  Practical Extraction and Report Language
perl  5.26 [d][e]  common [d], minimal  Practical Extraction and Report Language
```

The install (i) indication in column 3 is now gone, which confirms the profile deletion.

4. Reset the module so that neither stream is enabled or disabled. This will remove the enabled (e) indication from perl 5.26.

```
[userl@serverl ~]$ sudo dnf module reset perl

Dependencies resolved.
================================================================================
 Package           Arch             Version             Repository         Size
================================================================================
Resetting module streams:
 perl                               5.26

Transaction Summary
================================================================================

Is this ok [y/N]: y
Complete!
```

5. Install the non-default profile “minimal” for perl stream 5.24. This will auto-enable the stream. The --allowerasing option will instruct the command to remove installed packages for dependency resolution.

```
[userl@serverl ~]$ sudo dnf module install perl:5.24/minimal --allowerasing
Modular dependency problems with Defaults:

 Problem: conflicting requests
  - module freeradius:3.0:820190131191847:fbe42456-0.x86_64 requires module(perl
:5.26), but none of the providers can be installed
  - module perl:5.26:820181219174508:9edba152-0.x86_64 conflicts with module(per
l:5.24) provided by perl:5.24:820190207164249:ee766497-0.x86_64
  - module perl:5.24:820190207164249:ee766497-0.x86_64 conflicts with module(per
l:5.26) provided by perl:5.26:820181219174508:9edba152-0.x86_64
Dependencies resolved.
================================================================================
 Package          Arch   Version                                 Repository  Size
================================================================================
Upgrading:
 perl-DBD-SQLite  x86_64 1.58-1.module+el8+2519+e351b2a7         AppStream  186 k
 perl-DBI         x86_64 1.641-2.module+el8+2701+78cee6b5        AppStream  739 k
 perl-Digest      noarch 1.17-395.module+el8+2464+d274aedl       AppStream   27 k
........                _
Installing group/module packages:
 perl             x86_64 4:5.24.4-403.module+el8+2770+c759b41a AppStream  6.1 M
Removing dependent packages:
 perl-IO-Socket-SSL
                  noarch 2.060-2.el8                             @AppStream 580 k
 perl-Mozilla-CA  noarch 20160104-7.el8                          @AppStream 5.6 k
 perl-Net-SSLeay  x86_64 1.85-6.el8                              @AppStream 1.2 M
Downgrading:
 perl-Carp        noarch 1.40-366.module+el8+2464+d274aedl       AppStream   30 k
 perl-Data-Dumper x86_64 2.161-4.module+el8+2464+d274aedl        AppStream   57 k
 perl-Encode      x86_64 4:2.88-6.module+el8+2464+d274aedl       AppStream  1.5 M
........
```

```
Installing module profiles:
 perl/minimal
Enabling module streams:
 perl                    5.24

Transaction Summary
================================================================================
Install     1 Package
Upgrade    26 Packages
Remove      3 Packages
Downgrade  16 Packages

Total size: 12 M
Is this ok [y/N]: y
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Downgrading      : perl-libs-4:5.24.4-403.module+el8+2770+c759b4la.x8    1/88
  Upgrading        : perl-Exporter-5.72-1000.module+el8+2464+d274aedl.n    2/88
........
```

The above graphics are excerpts from a long output that the *dnf* command produced. The sequence of activities that occurred are: (1) dependency resolution, (2) a list of packages to be upgraded, installed, removed, and downgraded, along with their versions and sizes, (3) the module and profile to be installed, (4) the stream (5.24) to be enabled, (5) a summary of the entire transaction with a count of the packages to be manipulated, and (6) execution of the entire transaction.

6. Now that the downgrade is complete and a lower version of the software is installed, you can check the status of the module:

```
[userl@serverl ~]$ dnf module list perl
AppStream
Name  Stream    Profiles                Summary
perl  5.24 [e]  common [d], minimal [i] Practical Extraction and Report Language
perl  5.26 [d]  common [d], minimal     Practical Extraction and Report Language
```

The profile “minimal” is installed (i under Profiles) and the stream 5.24 is enabled (e under Stream).

Chapter Summary

This chapter is the second of the two chapters (the first one being Chapter 09) with coverage on software management. This chapter covers advanced topics: yum repositories, package groups, and package modules.

We looked at the concept of package repositories and demonstrated setting up access to a local repo. Employing repositories for package installation and automatic dependency selection make software installation much easier than to use the rpm command.

We learned about package groups and the benefits of using them in contrast to handling individual software packages. We explored package modules, streams, and profiles, and how they are organized and manipulated.

Finally, we examined the dnf command and discovered its benefits over the standard rpm command. dnf offers a variety of options and subcommands that we employed in this chapter to demonstrate installing, listing, querying, updating, and deleting individual packages, package groups, package streams, and package profiles. Other management operations such as exhibiting package data, ascertaining provider, searching metadata for information, and switching module streams were also covered.

Check Your Understanding

1. A module profile is a list of recommended packages organized for purpose-built deployments. True or False?
2. What would the *dnf group info Base* command do?
3. What would you run to list all available profiles for the module postgresql?
4. Which *dnf* subcommand can be used to list all available repositories?

5. What is the use of the -y option with the *dnf group install* and *dnf module remove* commands?
6. What is the main advantage of using *dnf* over *rpm*?
7. What would be the command to install *perl* 5.26?
8. What *dnf* subcommand would we use to check available updates for installed packages?
9. What is the concept of module in RHEL 8?
10. What must be the extension of a yum repository file?
11. A module stream is a group of packages organized by version. True or False?
12. What would the *dnf list zsh* command do?
13. What would the *dnf list installed* **gnome** command do?
14. How many package names can be specified at a time with the *dnf install* command?
15. Name the two default yum repositories that contain all the packages for RHEL 8?
16. We can update all the packages within a package group using the *groupupdate* subcommand with *dnf*. True or False?
17. What would you run to reset a module so that it is neither enabled nor disabled?
18. What would the *dnf info zsh* command do?

Answers to Check Your Understanding

1. True.
2. The command provided will list all packages in the specified package group.
3. To view *postgresql* module profiles, you would run *dnf module info --profile postgresql*.
4. The *repolist* subcommand.
5. *dnf* will not prompt for user confirmation if the -y option is used with it.

6. *dnf* resolves and installs dependent packages automatically.
7. It would be *dnf module install perl:5.26*.
8. The *check-update* subcommand.
9. A module is a collection of packages, including the dependent packages, that are required to install an application.
10. The extension of a yum repository configuration file must be *.repo* in order to be recognized as a valid repo file.
11. True.
12. The command provided will display installed and available *zsh* package.
13. The command provided will display all installed packages that contain gnome in their names.
14. There is no limit.
15. The two yum repositories the BaseOS and AppStream.
16. True.
17. You would run *dnf module reset* to reset a module.
18. The command provided will display the header information for the *zsh* package.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 10-1: Configure Access to RHEL 8 Repositories

As *user1* with *sudo* on *server1*, make sure the RHEL 8 ISO image is attached to the VM and mounted. Create a definition file under */etc/yum.repos.d*, and define two blocks (one for BaseOS and another for AppStream). Verify the configuration with **dnf repolist**. You should see numbers in thousands under the Status column for both repositories. (Hint1: Chapter 09: Package Management with rpm). (Hint2: Software Management with dnf).

Lab 10-2: Install and Manage Individual Packages

As *user1* with *sudo* on *server1* and using the *dnf* command, list all installed and available packages separately. Show which package contains the */etc/group* file. Install the package *policycoreutils*. Review */var/log/yum.log* for confirmation. Perform the following on the *policycoreutils* package: (1) show information, (2) list dependencies, and (3) remove it. (Hint: Individual Package Management).

Lab 10-3: Install and Manage Package Groups

As *user1* with *sudo* on *server1* and using the *dnf* command, list all installed and available package groups separately. Install package groups *Security Tools* and *Scientific Support*. Review */var/log/yum.log* for confirmation. Show the packages included in the *Scientific Support* package group, and delete this group. (Hint: Package Group Management).

Lab 10-4: Install and Manage Modules

As *user1* with *sudo* on *server1* and using the *dnf* command, list all modules. Identify which modules, streams and profiles are installed, default, disabled, and enabled from the output. Install the default stream of the development profile for

module *php*, and verify. Remove the module. (Hint: Module Management).

Lab 10-5: Switch Module Streams and Install Software

As *user1* with *sudo* on *server1* and using the *dnf* command, list *postgresql* module. This will display the streams and profiles, and their status. Reset both streams, enable the stream for the older version, and install its client profile. (Hint: Module Management).

Chapter 11

Boot Process, GRUB2, and the Linux Kernel

This chapter describes the following major topics:

- Linux boot process: firmware, bootloader, kernel, and initialization
- Understand and interact with GRUB2 to boot into different targets
- Modify GRUB2 configuration
- Boot system into specific targets
- Reset lost or forgotten root user password
- Linux kernel, packages, version anatomy, and key directories
- Download and install a newer kernel version

RHCSA Objectives:

14. Interrupt the boot process in order to gain access to a system
41. Modify the system bootloader

RHEL goes through multiple phases during the boot process. It starts selective services during its transition from one phase into another. It presents the administrator an opportunity to interact with a preboot program to boot the system into a non-default target, pass an option to the kernel, or reset the lost or forgotten root user password. It launches a number of services during its transition to the default or specified target.

The kernel controls everything on the system. It controls the system hardware, enforces security and access controls, and runs, schedules, and manages processes and service daemons. The kernel is comprised of several modules. A new kernel must be installed or an existing kernel must be upgraded when the need arises from an application or functionality standpoint.

Linux Boot Process

RHEL goes through a *boot* process after the system has been powered up or restarted. The boot process lasts until all enabled services are started. A login prompt will appear on the screen, which allows users to log in to the system. The boot process is automatic, but you may need to interact with it to take a non-default action, such as booting an alternative kernel, booting into a non-default operational state, repairing the system, recovering from an unbootable state, and so on. The boot process on an x86 computer may be split into four major phases: (1) the firmware phase, (2) the bootloader phase, (3) the kernel phase, and (4) the initialization phase. The system accomplishes these phases one after the other while performing and attempting to complete the tasks identified in each phase.

The Firmware Phase (BIOS and UEFI)

The *firmware* is the BIOS (*Basic Input/Output System*) or the UEFI (*Unified Extensible Firmware Interface*) code that is stored in

flash memory on the x86-based system board. It runs the *Power-On-Self-Test* (POST) to detect, test, and initialize the system hardware components. While doing so, it installs appropriate drivers for the video hardware and exhibit system messages on the screen. The firmware scans the available storage devices to locate a boot device, starting with a 512-byte image that contains 446 bytes of the bootloader program, 64 bytes for the partition table, and the last two bytes with the boot signature. This 512-byte tiny area is referred to as the *Master Boot Record* (MBR) and it is located on the first sector of the boot disk. As soon as it discovers a usable boot device, it loads the bootloader into memory and passes control over to it.

The BIOS is a small memory chip in the computer that stores system date and time, list and sequence of boot devices, I/O configuration, etc. This configuration is customizable. Depending on the computer hardware, you need to press a key to enter the BIOS setup or display a menu to choose a source to boot the system. The computer goes through the hardware initialization phase that involves detecting and diagnosing peripheral devices. It runs the POST on the devices as it finds them, installs drivers for the graphics card and the attached monitor, and begins exhibiting system messages on the video hardware. It discovers a usable boot device, loads the bootloader program into memory, and passes control over to it. Boot devices on most computers support booting from optical and USB flash devices, hard drives, network, and other media.

The UEFI is a new 32/64-bit architecture-independent specification that computer manufacturers have widely adopted in their latest hardware offerings replacing BIOS. This mechanism delivers enhanced boot and runtime services, and superior features such as speed over the legacy 16-bit BIOS. It has its own device drivers, is able to mount and read extended file systems, includes UEFI-compliant application tools, and supports one or more bootloader programs. It comes with a boot manager that allows you to choose an alternative boot source. Most computer manufacturers have customized the features for their hardware

platform. You may find varying menu interfaces among other differences.

The Bootloader Phase

Once the firmware phase is over and a boot device is detected, the system loads a piece of software called *bootloader* that is located in the boot sector of the boot device. RHEL uses GRUB2 (*GRand Unified Bootloader*) version 2 as the bootloader program. GRUB2 supports both BIOS and UEFI firmware.

The primary job of the bootloader program is to spot the Linux kernel code in the */boot* file system, decompress it, load it into memory based on the configuration defined in the */boot/grub2/grub.cfg* file, and transfer control over to it to further the boot process. For UEFI-based systems, GRUB2 looks for the EFI system partition */boot/efi* instead, and runs the kernel based on the configuration defined in the */boot/efi/EFI/redhat/grub.efi* file. The next section details the interaction with the bootloader.

The Kernel Phase

The *kernel* is the central program of the operating system, providing access to hardware and system services. After getting control from the bootloader, the kernel extracts the *initial RAM disk* (initrd) file system image found in the */boot* file system into memory, decompresses it, and mounts it as read-only on */sysroot* to serve as the temporary root file system. The kernel loads necessary modules from the initrd image to allow access to the physical disks and the partitions and file systems therein. It also loads any required drivers to support the boot process. Later, it unmounts the initrd image and mounts the actual physical root file system on */* in read/write mode.

At this point, the necessary foundation has been built for the boot process to carry on and to start loading the enabled services. The kernel executes the *systemd* process with PID 1 and passes the control over to it.

The Initialization Phase

This is the fourth and the last phase in the boot process. *systemd* takes control from the kernel and continues the boot process. It is the default system initialization scheme used in RHEL 8. It starts all enabled userspace system and network services and brings the system up to the preset boot target.

A boot target is an operational level that is achieved after a series of services have been started to get to that state. More on targets later in this chapter.

The system boot process is considered complete when all enabled services are operational for the boot target and users are able to log in to the system. A detailed discussion on *systemd* is available in Chapter 12 “System Initialization, Message Logging, and System Tuning”.

The GRUB2 Bootloader

After the firmware phase has concluded, the bootloader presents a menu with a list of bootable kernels available on the system and waits for a predefined amount of time before it times out and boots the default kernel. You may want to interact with GRUB2 before the autoboot times out to boot with a non-default kernel, boot to a different target, or customize the kernel boot string.

Pressing a key before the timeout expires allows you to interrupt the autoboot process and interact with GRUB2. If you wish to boot the system using the default boot device with all the configured default settings, do not press any key, as shown in Figure 11-1, and let the system go through the autoboot process.

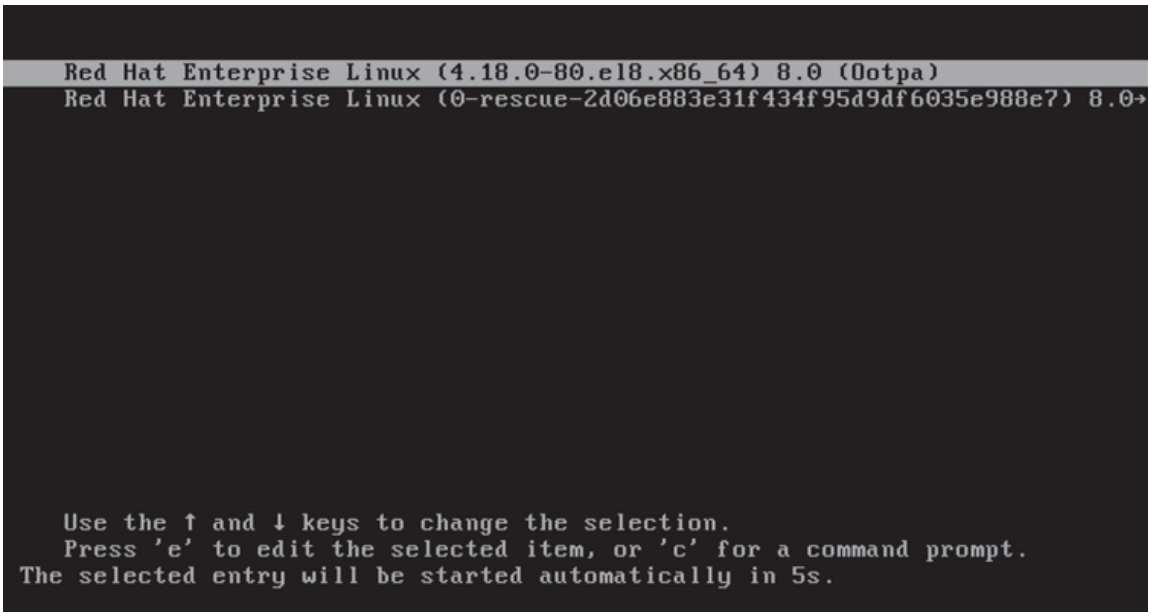


Figure 11-1 GRUB2 Menu

The line at the bottom in Figure 11-1 shows the autoboot countdown in seconds. The default value is 5 seconds. If you press no keys within the 5 seconds, the highlighted kernel will boot automatically.

Interacting with GRUB2

The GRUB2 main menu shows a list of bootable kernels at the top. You can change the selection using the Up or Down arrow key. It lets you edit a selected kernel menu entry by pressing an *e* or go to the grub> command prompt by pressing a *c*.

In the edit mode, GRUB2 loads the configuration for the selected kernel entry from the */boot/grub2/grub.cfg* file in an editor, enabling you to make a desired modification before booting the system. For instance, you can boot the system into a less capable operating target by adding “rescue”, “emergency”, or “3” to the end of the line that begins with the keyword “linux”, as depicted in Figure 11-2. Press Ctrl+x when done to boot. Remember that this is a one-time temporary change and it won’t touch the *grub.cfg* file.

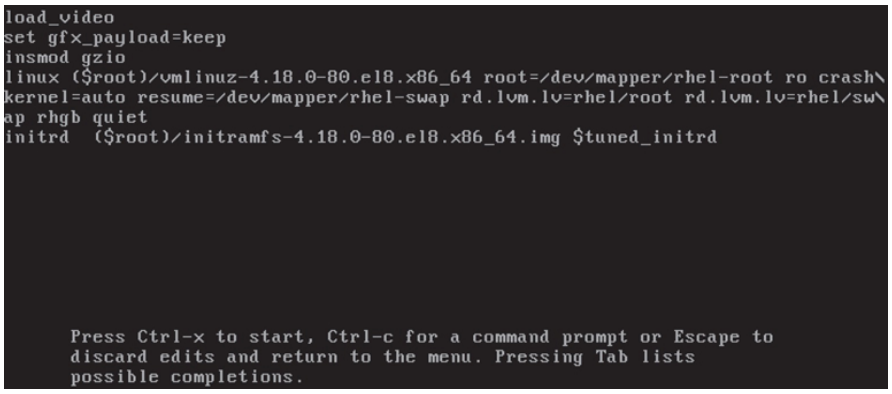


Figure 11-2 GRUB2 Kernel Edit

If you do not wish to boot the system at this time, you can press ESC to discard the changes and return to the main menu.

The grub> command prompt appears when you press Ctrl+c while in the edit window or a *c* from the main menu. The command mode provides you with the opportunity to execute debugging, recovery, and many other tasks. You can view available commands by pressing the TAB key. See Figure 11-3.

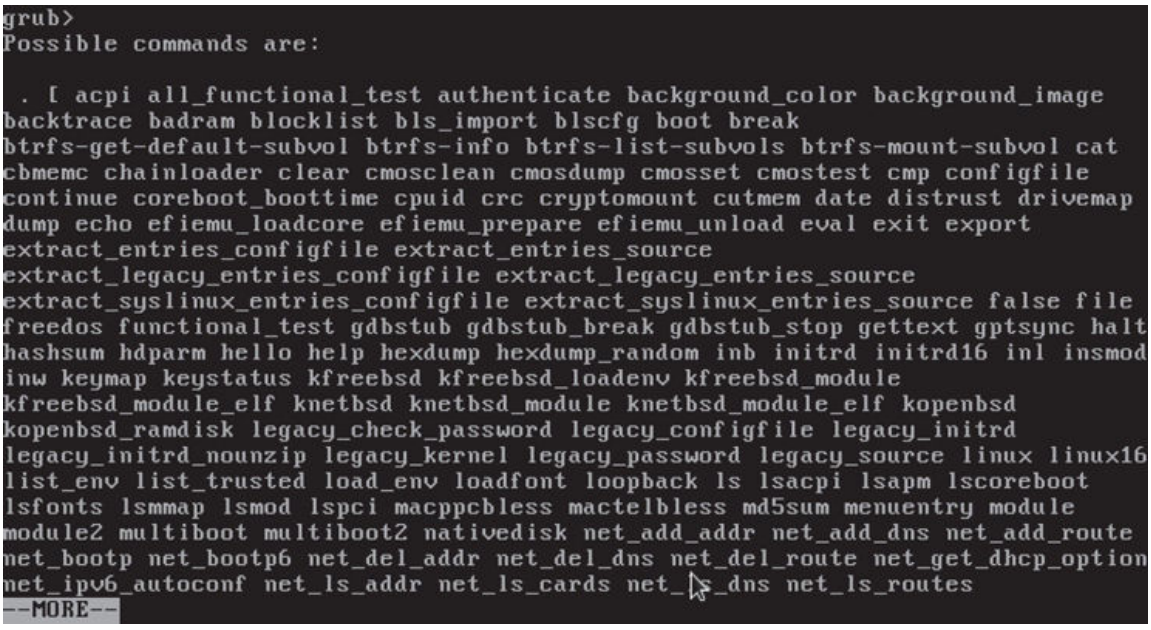


Figure 11-3 GRUB2 Commands

There are over one hundred commands available to perform a variety of tasks at the GRUB2 level.

Understanding GRUB2 Configuration Files

The GRUB2 configuration file, *grub.cfg*, is located in the */boot/grub2* directory. This file is referenced at boot time. This file is generated automatically when a new kernel is installed or upgraded, so it is not advisable to modify it directly, as your changes will be overwritten. The primary source file that is used to regenerate *grub.cfg* is called *grub*, and it is located in the */etc/default* directory. This file defines the directives that govern how GRUB2 should behave at boot time. Any changes made to the *grub* file will only take effect after the *grub2-mkconfig* utility has been executed.

Let's analyze the two files to understand their syntax and contents.

The /etc/default/grub File

The *grub* file defines the directives that control the behavior of GRUB2 at boot time. Any changes in this file must be followed by the execution of the *grub2-mkconfig* command in order to be reflected in *grub.cfg*.

Here is an enumerated list of the default settings from the *grub* file, followed by an explanation in Table 11-1:

```
[user1@server1 ~]$ nl /etc/default/grub
     1  GRUB_TIMEOUT=5
     2  GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
     3  GRUB_DEFAULT=saved
     4  GRUB_DISABLE_SUBMENU=true
     5  GRUB_TERMINAL_OUTPUT="console"
     6  GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm
.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet"
     7  GRUB_DISABLE_RECOVERY="true"
     8  GRUB_ENABLE_BLSCFG=true
```

Directive	Description
GRUB_TIMEOUT	Defines the wait time, in seconds, before booting off the default kernel. Default value is 5.
GRUB_DISTRIBUTOR	Sets the name of the Linux distribution
GRUB_DEFAULT	Boots the selected option from the previous system boot
GRUB_DISABLE_SUBMENU	Enables/disables the appearance of GRUB2 submenu
GRUB_TERMINAL_OUTPUT	Sets the default terminal
GRUB_CMDLINE_LINUX	Specifies the command line options to pass to the kernel at boot time
GRUB_DISABLE_RECOVERY	Lists/hides system recovery entries in the GRUB2 menu
GRUB_ENABLE_BLSCFG	Defines whether to use the new bootloader specification to manage bootloader configuration

Table 11-1 GRUB2 Default Settings

Generally, you do not need to make any changes to this file, as the default settings are good enough for normal system operation.

The grub.cfg File

The *grub.cfg* is the main GRUB2 configuration file that supplies boot-time configuration information. This file is located in the */boot/grub2* directory on BIOS-based systems and in the */boot/efi/EFI/redhat* directory on UEFI-based systems. This file can be regenerated manually with the *grub2-mkconfig* utility, or it is automatically regenerated when a new kernel is installed or upgraded. In either case, this file will lose any previous manual changes made to it.

Here is how you would run this utility after making a change such as adjusting the GRUB_TIMEOUT value to 10 in the */etc/default/grub* file to reproduce *grub.cfg*:

```
[user1@server1 ~]$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
done
```

Use **grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg** on UEFI systems.

Restart the system with the *reboot* command and confirm the new timeout value when GRUB2 menu appears.

grub2-mkconfig also uses the settings defined in helper scripts located in the */etc/grub.d* directory during the regeneration process. There are plenty of files located here, as shown below:

```
[user1@server1 ~]$ sudo ls -l /etc/grub.d
total 84
-rwxr-xr-x. 1 root root  8958 Dec 19  2018 00_header
-rwxr-xr-x. 1 root root  1043 Jul  4  2018 00_tuned
-rwxr-xr-x. 1 root root  1240 Dec 19  2018 01_menu_auto_hide
-rwxr-xr-x. 1 root root   232 Dec 19  2018 01_users
-rwxr-xr-x. 1 root root 13406 Dec 19  2018 10_linux
-rwxr-xr-x. 1 root root 11696 Dec 19  2018 20_linux_xen
-rwxr-xr-x. 1 root root  2559 Dec 19  2018 20_ppc_terminfo
-rwxr-xr-x. 1 root root 10670 Dec 19  2018 30_os-prober
-rwxr-xr-x. 1 root root  1412 Dec 19  2018 30_uefi-firmware
-rwxr-xr-x. 1 root root   214 Dec 19  2018 40_custom
-rwxr-xr-x. 1 root root   216 Dec 19  2018 41_custom
-rw-r--r--. 1 root root   483 Dec 19  2018 README
```

The first script, *00_header*, sets the GRUB2 environment; the *10_linux* script searches for all installed kernels on the same disk partition; the *30_os-prober* searches for the presence of other operating systems; and *40_custom* and *41_custom* are to introduce any customization. An example would be to add custom entries to the boot menu.

The *grub.cfg* file also sources the *grubenv* file located in the */boot/grub2* directory for kernel options and other settings. Here is what this file contains on *server1*:

```
[root@server1 grub2]# cat grubenv
# GRUB Environment Block
saved_entry=0
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhe
l-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
boot_success=1
boot_indeterminate=0
###############################################################################
###############################################################################
```

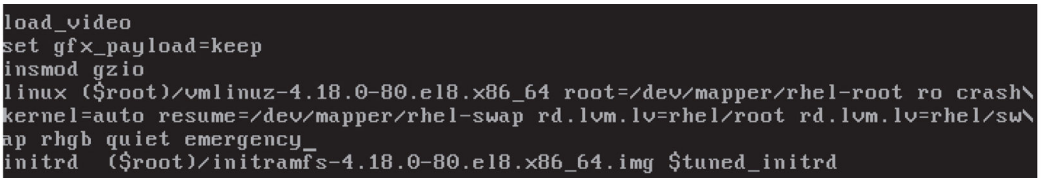
If a new kernel is installed on the system, the existing kernel entries remain intact. All bootable kernels are listed in the GRUB2 menu, and any of the kernel entries can be selected to boot.

Booting into Specific Targets

RHEL boots into graphical target state by default if the Server with GUI software selection is made during installation. It can also be directed to boot into non-default but less capable operating targets from the GRUB2 menu. Additionally, in situations when it becomes mandatory to boot the system into an administrative state for implementing a function that cannot be otherwise performed in other target states or for system recovery, RHEL offers emergency and rescue boot targets. These special target levels can be launched from the GRUB2 interface by selecting a kernel, pressing *e* to enter the edit mode, and appending the desired target name to the line that begins with the keyword “linux”.

> **EXAM TIP:** You need to know how to boot a RHEL 8 system into a specific target from the GRUB2 menu to modify the fstab file or reset an unknown root user password.

Here is how you would append “emergency” to the kernel line entry:



Press Ctrl+x after making the modification to boot the system into the supplied target. You will be required to enter the *root* user password to log on. Run *reboot* after you are done to reboot the system.

Similarly, you can append “rescue” (or simply “1”, “s”, or “single”) to the “linux” line and press Ctrl+x to boot into the rescue target.

Exercise 11-1: Reset the root User Password

This exercise should be done on *server1*.

For this exercise, assume that the *root* user password is lost or forgotten, and it needs to be reset.

In this exercise, you will terminate the boot process at an early stage to be placed in a special debug shell in order to reset the *root* password.

1. Reboot or reset *server1*, and interact with GRUB2 by pressing a key before the autoboot times out. Highlight the default kernel entry in the GRUB2 menu and press *e* to enter the edit mode. Scroll down to the line entry that begins with the keyword “linux” and press the End key to go to the end of that line:

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-80.el8.x86_64 root=/dev/mapper/rhel-root ro crash\
kernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/sw\
ap rhgb quiet
initrd  ($root)/initramfs-4.18.0-80.el8.x86_64.img $tuned_initrd
```

2. Modify this kernel string and append “rd.break” to the end of the line. It should look like:

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-80.el8.x86_64 root=/dev/mapper/rhel-root ro crash\
kernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/sw\
ap rhgb quiet rd.break
initrd  ($root)/initramfs-4.18.0-80.el8.x86_64.img $tuned_initrd
```

3. Press Ctrl+x when done to boot to the special shell. The system mounts the root file system read-only on the */sysroot* directory. Make */sysroot* appear as mounted on */* using the *chroot* command:

```
switch_root:/# chroot /sysroot
sh-4.4# pwd
/
sh-4.4#
```

4. Remount the root file system in read/write mode for the *passwd* command to be able to modify the *shadow* file with a new password:

```
sh-4.4# mount -o remount,rw /
```

5. Enter a new password for *root* by invoking the *passwd* command:

```
sh-4.4# passwd
Changing password for user root.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
```

6. Create a hidden file called *.autorelabel* to instruct the operating system to run SELinux relabeling on all files, including the *shadow* file that was updated with the new *root* password, on the next reboot:

```
sh-4.4# touch .autorelabel
```

7. Issue the *exit* command to quit the chroot shell and then the *reboot* command to restart the system and boot it to the default target.

```
sh-4.4# exit
exit
switch_root:/# reboot_
```

Depending on the storage speed, the system might take a few minutes to complete the relabeling and return to a fully functional state.

The Linux Kernel

The *kernel* is the core of the Linux system. It manages hardware, enforces security, regulates access to the system, as well as handles processes, services, and application workloads. It is a collection of software components called *modules* that work in tandem to provide a stable and controlled platform. Modules are *device drivers* that control hardware devices—the processor, memory, storage, controller cards, and peripheral equipment,

and interact with software subsystems, such as storage partitioning, file systems, networking, and virtualization.

Some of these modules are static to the kernel and are integral to system functionality, while others are loaded dynamically as needed, making the kernel speedier and more efficient in terms of overall performance and less vulnerable to crashes. RHEL 8.0 is shipped with kernel version 4.18.0 for the 64-bit Intel/AMD processor architecture computers with single, multi-core, and multi-processor configurations. On a Linux system, **uname -m** reveals the architecture of the system.

The default kernel installed during the installation is usually adequate for most system needs; however, it requires a rebuild when a new functionality is added or removed. The new functionality may be introduced by installing a new kernel, upgrading an existing one, installing a new hardware device, or changing a critical system component. Likewise, an existing functionality that is no longer needed may be removed to make the overall footprint of the kernel smaller, resulting in improved performance and reduced memory utilization.

To control the behavior of the modules, and the kernel in general, a variety of tunable parameters are set that define a baseline for kernel functionality. Some of these parameters must be tuned to allow certain applications and database software to be installed smoothly and operate properly.

RHEL allows you to generate and store several custom kernels with varied configuration and required modules, but only one of them can be active at a time. A different kernel may be loaded by interacting with GRUB2.

Kernel Packages

Just like any other software that comes in the rpm format for RHEL, the software comprising the kernel is no different. There is a set of core kernel packages that must be installed on the system at a minimum to make it work. Additional packages providing supplementary kernel support are also available. Table

11-2 lists and describes the core and some add-on kernel packages.

Kernel Package	Description
kernel	Contains no files, but ensures other kernel packages are accurately installed
kernel-core	Includes a minimal number of modules to provide core functionality
kernel-devel	Includes support for building kernel modules
kernel-modules	Contains modules for common hardware devices
kernel-modules-extra	Contains modules for not-so-common hardware devices
kernel-headers	Includes files to support the interface between the kernel and userspace libraries and programs
kernel-tools	Includes tools to manipulate the kernel
kernel-tools-libs	Includes the libraries to support the kernel tools

Table 11-2 Kernel Packages

Moreover, packages containing the source code for RHEL 8 are also available for those who wish to customize and recompile the code for their precise needs.

Currently, the following kernel packages are installed on *server1*:

```
[user1@server1 ~]$ dnf list installed kernel*
Not root, Subscription Management repositories not updated
Installed Packages
kernel.x86_64                                4.18.0-80.el8                          @anaconda
kernel-core.x86_64                           4.18.0-80.el8                          @anaconda
kernel-headers.x86_64                        4.18.0-80.el8                          @BaseOS
kernel-modules.x86_64                        4.18.0-80.el8                          @anaconda
kernel-tools.x86_64                          4.18.0-80.el8                          @anaconda
kernel-tools-libs.x86_64                     4.18.0-80.el8                          @anaconda
```

The output returns six kernel packages that were loaded during the OS installation.

Analyzing Kernel Version

Sometimes you need to ascertain the version of the kernel running on the system to check for compatibility with an application or database that you need to deploy. RHEL has a

basic tool available to extract the version. The *uname* command with the -r switch depicts this information, as follows:

```
[user1@server1 ~]$ uname -r
4.18.0-80.el8.x86_64
```

The output shows the current kernel version in use is 4.18.0-80.el8.x86_64. An anatomy of the version is illustrated in Figure 11-4 and explained subsequently.

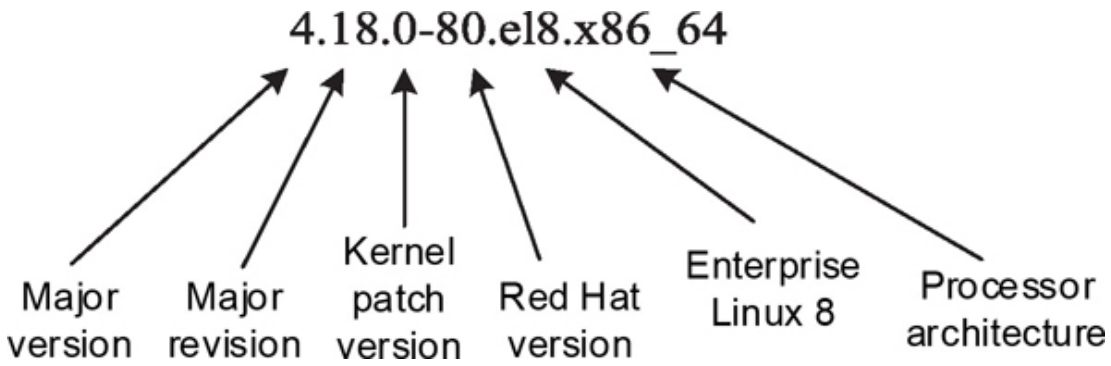


Figure 11-4 Anatomy of a Kernel Version

From left to right:

- (4) major version of the Linux kernel. It changes when significant alterations, enhancements, and updates to the previous major version are made.
- (18) major revision of the 4th major version
- (0) represents patched version of 4.18 with minor bug and security hole fixes, minor enhancements, and so on.
- (80) Red Hat customized version of 4.18.0
- (el8) Enterprise Linux this kernel is built for
- (x86_64) architecture this kernel is built for

A further analysis designates that 4.18.0 holds the general Linux kernel version information, the numbers and letters (80.el8) represent the Red Hat specific information, and x86_64 identifies the hardware architecture type.

Understanding Kernel Directory Structure

Kernel and its support files are stored at different locations in the directory hierarchy, of which three locations—*/boot*, */proc*, and */usr/lib/modules*—are noteworthy.

The /boot Location

/boot is essentially a file system that is created at system installation. It houses the Linux kernel, GRUB2 configuration, and other kernel and boot support files. A long listing produces the following output for this file system:

```
[userl@serverl ~]$ ls -l /boot
total 126992
-rw-r--r--. 1 root root   180942 Mar 13  2019 config-4.18.0-80.el8.x86_64
drwxr-xr-x. 3 root root       17 Aug 23 06:06 efi
drwx------. 4 root root       83 Oct  6 20:00 grub2
-rw-------. 1 root root 65698603 Aug 23 06:22 initramfs-0-rescue-2d06e883e31f434
f95d9df6035e988e7.img
-rw-------. 1 root root 26614276 Aug 23 06:27 initramfs-4.18.0-80.el8.x86_64.img
-rw-------. 1 root root 18030660 Oct 10 15:48 initramfs-4.18.0-80.el8.x86_64kdum
p.img
drwxr-xr-x. 3 root root       21 Aug 23 06:15 loader
-rw-------. 1 root root  3751920 Mar 13  2019 System.map-4.18.0-80.el8.x86_64
-rwxr-xr-x. 1 root root  7872864 Aug 23 06:20 vmlinuz-0-rescue-2d06e883e31f434f9
5d9df6035e988e7
-rwxr-xr-x. 1 root root  7872864 Mar 13  2019 vmlinuz-4.18.0-80.el8.x86_64
```

The output displays several files, four of which are for the kernel and two for its rescue version. The kernel files are *vmlinuz*, *initramfs*, *config*, and *System.map*, and they all have the current kernel version appended to their names. The *vmlinuz* is the main kernel file with *initramfs*, *config*, and *System.map* storing the main kernel's boot image, configuration, and mapping, respectively.

The files for the rescue version have the string "rescue" embedded within their names, as indicated in the above output.

The *efi* and *grub2* subdirectories under */boot* hold bootloader information specific to firmware type used on the system: UEFI or BIOS. For *server1*, *grub2* contains GRUB2 information as shown below:

```
[userl@serverl ~]$ sudo ls -l /boot/grub2
total 28
-rw-r--r--. 1 root root   64 Aug 23 06:26 device.map
drwxr-xr-x. 2 root root   25 Aug 23 06:26 fonts
-rw-r--r--. 1 root root 5032 Oct  6 20:00 grub.cfg
-rw-r--r--. 1 root root 1024 Oct 11 10:09 grubenv
drwxr-xr-x. 2 root root 8192 Aug 23 06:26 i386-pc
```

The files *grub.cfg* and *grubenv* contain critical data with the former holding bootable kernel information and the latter stores the environment information that the kernel uses.

The subdirectory *loader* under */boot* is the storage location for configuration of the running and rescue kernels. The configuration is stored in files under the *entries* subdirectory, as shown below:

```
[userl@serverl ~]$ sudo ls -l /boot/loader/entries/
total 8
-rw-r--r--. 1 root root 408 Aug 23 06:22 2d06e883e31f434f95d9df6035e988e7-0-resc
ue.conf
-rw-r--r--. 1 root root 331 Aug 23 06:22 2d06e883e31f434f95d9df6035e988e7-4.18.0
-80.el8.x86_64.conf
```

The files are named using the machine id of the system as stored in the */etc/machine-id* file and the kernel version they are for. The content of the kernel file is presented below:

```
[userl@serverl ~]$ sudo cat /boot/loader/entries/2d06e883e31f434f95d9df6035e988e
7-4.18.0-80.el8.x86_64.conf
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 “title” is displayed on the bootloader screen where the countdown to autoboot a selected kernel entry runs. Other than the kernel version, and the default kernel and boot image filenames, the environment variables “kernelopts” and “tuned_params” supply various values to the booting kernel to control its behavior.

The /proc Location

/proc is a virtual, memory-based file system. Its contents are created and updated in memory at system boot and during runtime, and they are destroyed at system shutdown. It maintains information about the current state of the kernel, which includes hardware configuration and status information about processor, memory, storage, file systems, swap, processes, network interfaces and connections, routing, etc. This data is kept in tens of thousands of zero-byte files organized in a hierarchy of hundreds of subdirectories. A long directory listing of */proc* is provided below:

```
[user1@server1 ~]$ ls -l /proc
total 0
dr-xr-xr-x.  9 root          root                        0 Oct 11 10:06 1
dr-xr-xr-x.  9 root          root                        0 Oct 11 10:06 10
dr-xr-xr-x.  9 avahi         avahi                       0 Oct 11 10:07 1010
dr-xr-xr-x.  9 root          root                        0 Oct 11 10:06 1012
dr-xr-xr-x.  9 root          root                        0 Oct 11 10:06 1016
........
```

Some subdirectory names are numerical and contain information about a specific process, and the process ID matches the subdirectory name. Within each subdirectory are other files and subdirectories containing a plethora of information, such as memory segments for processes and configuration data for system components. You can view the configuration of a particular item using any text file viewing tool. The following show selections from the *cpuinfo* and *meminfo* files that hold processor and memory information:

```
[user1@server1 ~]$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2294.786
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
........
[user1@server1 ~]$ cat /proc/meminfo
MemTotal:         841124 kB
MemFree:          167684 kB
MemAvailable:     466216 kB
Buffers:            3488 kB
Cached:           395344 kB
SwapCached:            0 kB
Active:           304880 kB
Inactive:         200116 kB
........
```

The data stored under */proc* is referenced by many system utilities, including *top*, *ps*, *uname*, *free*, *uptime* and *w*, to display information.

The /usr/lib/modules Location

This directory holds information about kernel modules. Underneath it are subdirectories specific to the kernels installed on the system. For example, the long listing of */usr/lib/modules* below shows that there is only one kernel installed. The name of the subdirectory corresponds with the installed kernel version.

```
[user1@server1 ~]$ ls -l /usr/lib/modules
total 4
drwxr-xr-x. 6 root root 4096 Oct  3 23:22 4.18.0-80.el8.x86_64
```

And this subdirectory contains:

```
[user1@server1 ~]$ ls -l /usr/lib/modules/4.18.0-80.el8.x86_64/
total 15100
-rw-r--r--.  1 root root     303 Mar 13  2019 bls.conf
lrwxrwxrwx.  1 root root      37 Mar 13  2019 build -> /usr/src/kernels/4.18.0-8
0.el8.x86_64
-rw-r--r--.  1 root root  180942 Mar 13  2019 config
drwxr-xr-x. 12 root root     128 Aug 23 06:09 kernel
-rw-r--r--.  1 root root  840950 Oct  3 23:22 modules.alias
-rw-r--r--.  1 root root  806340 Oct  3 23:22 modules.alias.bin
-rw-r--r--.  1 root root     481 Mar 13  2019 modules.block
-rw-r--r--.  1 root root    7443 Mar 13  2019 modules.builtin
........
```

There are several files and a few subdirectories here; they hold module-specific information for the kernel version.

One of the key subdirectories is */usr/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers*, which stores modules for a variety of hardware and software components in various subdirectories, as shown in the listing below:

```
[user1@server1 ~]$ ls -l /usr/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/
total 48
drwxr-xr-x.  5 root root  209 Aug 23 06:09 acpi
drwxr-xr-x.  2 root root  169 Aug 23 06:09 ata
drwxr-xr-x.  2 root root   24 Aug 23 06:10 bcma
drwxr-xr-x.  3 root root  163 Aug 23 06:09 block
drwxr-xr-x.  2 root root  272 Aug 23 06:10 bluetooth
drwxr-xr-x.  2 root root   25 Aug 23 06:09 cdrom
drwxr-xr-x.  5 root root  181 Aug 23 06:09 char
drwxr-xr-x.  2 root root  143 Aug 23 06:09 cpufreq
drwxr-xr-x.  7 root root  121 Aug 23 06:09 crypto
drwxr-xr-x.  2 root root   52 Aug 23 06:09 dax
drwxr-xr-x.  2 root root   23 Aug 23 06:09 dca
drwxr-xr-x.  4 root root   48 Aug 23 06:09 dma
drwxr-xr-x.  2 root root 4096 Aug 23 06:09 edac
drwxr-xr-x.  2 root root  113 Aug 23 06:10 firewire
drwxr-xr-x.  2 root root   89 Aug 23 06:10 firmware
........
```

Additional modules may be installed on the system to support more components.

Installing the Kernel

Installing kernel packages requires extra care because it could leave your system in an unbootable or undesirable state. It is advisable to have the bootable medium handy prior to starting the kernel install process. By default, the *dnf* command adds a new kernel to the system, leaving the existing kernel(s) intact. It does not replace or overwrite existing kernel files.

EXAM TIP: Always install a new version of the kernel instead of upgrading it. The upgrade process removes any existing kernel and replaces it with a new one. In case of a post-installation issue, you will not be able to revert to the old working kernel.

A newer version of the kernel is typically required if an application needs to be deployed on the system that requires a different kernel to operate. When deficiencies or bugs are identified in the existing kernel, it can hamper the kernel's smooth operation. In either case, the new kernel addresses existing issues as well as adds bug fixes, security updates, new features, and improved support for hardware devices.

The *dnf* command is the preferred tool to install a kernel, as it resolves and installs any required dependencies automatically. The *rpm* command may alternatively be used; however, you must install any reported dependencies manually.

The kernel packages for RHEL 8 are available to subscribers for download on Red Hat's Customer Portal. You need a user account in order to log in and download.

Exercise 11-2: Download and Install a New Kernel

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will download the latest available kernel packages from the Red Hat Customer Portal and install them using the *dnf* command. You will ensure that the existing kernel and its configuration remain intact.

1. Check the version of the running kernel:

```
[user1@server1 ~]$ uname -r
4.18.0-80.el8.x86_64
```

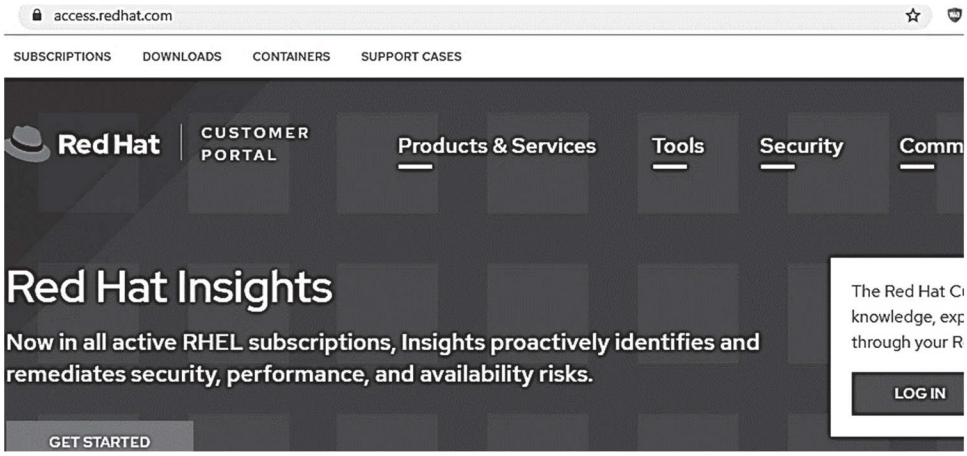
The output discloses the current active kernel version as 4.18.0-80.el8.x86_64.

2. List the kernel packages currently installed:

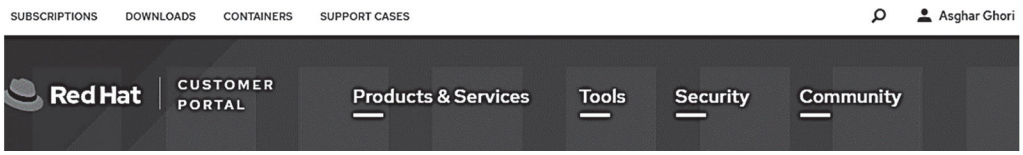
```
[user1@server1 ~]$ rpm -qa |grep kernel
kernel-modules-4.18.0-80.el8.x86_64
kernel-core-4.18.0-80.el8.x86_64
kernel-headers-4.18.0-80.el8.x86_64
kernel-4.18.0-80.el8.x86_64
kernel-tools-4.18.0-80.el8.x86_64
kernel-tools-libs-4.18.0-80.el8.x86_64
```

There are six kernel packages on the system as reported.

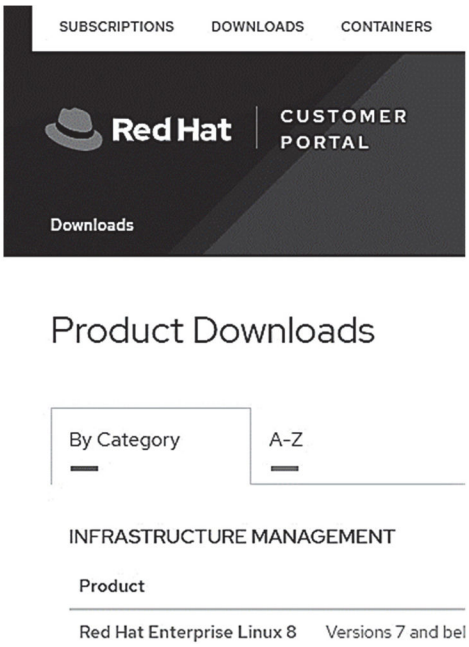
3. Access the Red Hat Customer Portal webpage at *access.redhat.com* and click LOG IN to sign in to the portal using the credentials you used/created in Chapter 01 “Local Installation” for the developer account:



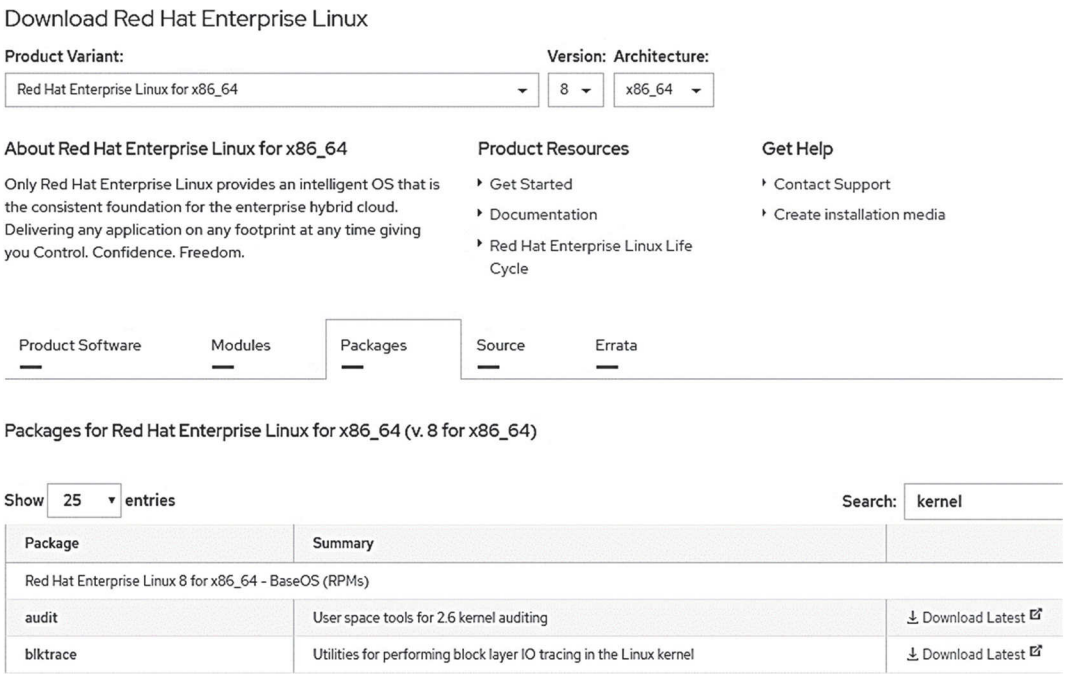
4. Click DOWNLOADS at the top:



5. Click “Red Hat Enterprise Linux 8” under “By Category”:



6. Click Packages and enter “kernel” in the Search bar to narrow the list of available packages:



7. Click “Download Latest” against the packages *kernel*, *kernel-core*, *kernel-headers*, *kernel-modules*, *kernel-tools*, and *kernel-tools-libs* to download them. These are the newer packages for the version you currently have on the system.

8. Once downloaded, move the packages to the */tmp* directory using the *mv* command.
9. List the packages after moving them:

```
[user1@server1 ~]$ ls /tmp/kernel*
/tmp/kernel-4.18.0-80.11.2.el8_0.x86_64.rpm          /tmp/kernel-modules-4.18.0-80.11.2.el8_0.x86_64.rpm
/tmp/kernel-core-4.18.0-80.11.2.el8_0.x86_64.rpm     /tmp/kernel-tools-4.18.0-80.11.2.el8_0.x86_64.rpm
/tmp/kernel-headers-4.18.0-80.11.2.el8_0.x86_64.rpm  /tmp/kernel-tools-libs-4.18.0-80.11.2.el8_0.x86_64.rpm
```

10. Install all the six packages at once using the *dnf* command:

```
[user1@server1 ~]$ sudo dnf install /tmp/kernel* -y

Dependencies resolved.
================================================================================================================
 Package                  Arch          Version                          Repository              Size
================================================================================================================
Installing:
 kernel                   x86_64        4.18.0-80.11.2.el8_0             @commandline           424 k
 kernel-core              x86_64        4.18.0-80.11.2.el8_0             @commandline            24 M
 kernel-modules           x86_64        4.18.0-80.11.2.el8_0             @commandline            20 M
Upgrading:
 kernel-headers           x86_64        4.18.0-80.11.2.el8_0             @commandline           1.6 M
 kernel-tools             x86_64        4.18.0-80.11.2.el8_0             @commandline           574 k
 kernel-tools-libs        x86_64        4.18.0-80.11.2.el8_0             @commandline           433 k

Transaction Summary
================================================================================================================
Install  3 Packages
Upgrade  3 Packages

Total size: 47 M

........
Installed products updated.

Upgraded:
  kernel-headers-4.18.0-80.11.2.el8_0.x86_64              kernel-tools-4.18.0-80.11.2.el8_0.x86_64
  kernel-tools-libs-4.18.0-80.11.2.el8_0.x86_64

Installed:
  kernel-4.18.0-80.11.2.el8_0.x86_64                     kernel-core-4.18.0-80.11.2.el8_0.x86_64
  kernel-modules-4.18.0-80.11.2.el8_0.x86_64

Complete!
```

11. Confirm the installation alongside the previous version:

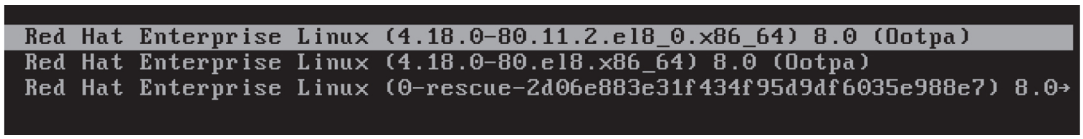
```
[user1@server1 ~]$ sudo dnf list installed kernel*
Installed Packages
kernel.x86_64                                4.18.0-80.el8                          @anaconda
kernel.x86_64                                4.18.0-80.11.2.el8_0                   @@commandline
kernel-core.x86_64                           4.18.0-80.el8                          @anaconda
kernel-core.x86_64                           4.18.0-80.11.2.el8_0                   @@commandline
kernel-headers.x86_64                        4.18.0-80.11.2.el8_0                   @@commandline
kernel-modules.x86_64                        4.18.0-80.el8                          @anaconda
kernel-modules.x86_64                        4.18.0-80.11.2.el8_0                   @@commandline
kernel-tools.x86_64                          4.18.0-80.11.2.el8_0                   @@commandline
kernel-tools-libs.x86_64                     4.18.0-80.11.2.el8_0                   @@commandline
```

The output indicates that packages for a higher kernel version 4.18.0-80.11.2.el8 have been installed. It also shows the presence of the previous kernel packages.

12. The */boot/grub2/grubenv* file now has the directive "saved_entry" set to the new kernel, which implies that this new kernel will boot up on the next system restart:

```
[userl@serverl ~]$ sudo cat /boot/grub2/grubenv
# GRUB Environment Block
saved_entry=2d06e883e31f434f95d9df6035e988e7-4.18.0-80.11.2.el8_0.x86_64
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root r
d.lvm.lv=rhel/swap rhgb quiet
boot_success=1
boot_indeterminate=0
```

13. Reboot the system. You will see the new kernel entry in the GRUB2 boot list at the top. The system will autoboot this new default kernel.



14. Run the *uname* command once the system has been booted up to confirm the loading of the new kernel:

```
[userl@serverl ~]$ uname -r
4.18.0-80.11.2.el8_0.x86_64
```

The new active kernel is the one that was just installed.

15. You can also view the contents of the *version* and *cmdline* files under */proc* to verify the active kernel:

```
[userl@serverl ~]$ cat /proc/version
Linux version 4.18.0-80.11.2.el8_0.x86_64 (mockbuild@x86-vm-07.build.eng.bos.redhat.com) (gcc version 8.2.1
20180905 (Red Hat 8.2.1-3) (GCC)) #1 SMP Sun Sep 15 11:24:21 UTC 2019
[userl@serverl ~]$
[userl@serverl ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-80.11.2.el8_0.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto r
esume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
```

This concludes the process of downloading and installing a newer kernel version on RHEL 8.

Chapter Summary

This chapter presented a high-level look at the Linux boot process, highlighting the four phases the system passes through and the complexities involved. We reviewed firmware and looked at preboot administration tasks, which included interacting with

the bootloader program, booting into specific targets, and an analysis of the bootloader configuration files. We performed a critical exercise that focused on resetting a forgotten or lost root user password.

The next major topic discussed the Linux kernel, its key components and management. We explored various packages that made up the core kernel software, performed an anatomy of its versioning, and examined key directories and file systems that are employed to hold kernel-specific information. Before the chapter was concluded, we downloaded and installed a new kernel without impacting the old one.

Check Your Understanding

1. UEFI is replacing BIOS in newer computers. True or False?
2. You have changed the timeout value in the *grub* configuration file located in the */etc/default* directory. Which command would you run now to ensure the change takes effect on next system reboots?
3. You want to view the parameters passed to the kernel at boot time. Which virtual file would you look at?
4. Name the location of the *grub.efi* file in the UEFI-based systems.
5. What is the name of the default bootloader program in RHEL 8?
6. The *systemd* command may be used to rebuild a new kernel. True or False?
7. What does the *chroot* command do?
8. At what stage should you interrupt the boot sequence to boot the system into a non-default target?
9. Which two files would you view to obtain processor and memory information?
10. By default, GRUB2 is stored in the MBR on a BIOS-based system. True or False?
11. Which file stores the location of the boot partition on the BIOS systems?
12. You have lost the *root* user password and you need to reset it. What would you add to the default kernel boot

string to break the boot process at an early stage?

13. You have installed a newer version of the kernel. What would you now have to do to make the new kernel the default boot kernel?
14. What is the system initialization and service management scheme called?

Answers to Check Your Understanding

1. True.
2. You will need to run the *grub2-mkconfig* command.
3. The *cmdline* file in the */proc* file system.
4. The *grub.efi* file is located in the */boot/efi/EFI/redhat* directory.
5. GRUB2.
6. False.
7. The *chroot* command changes the specified directory path to /.
8. At the GRUB2 stage.
9. The *cpuinfo* and *meminfo* files in the */proc* file system.
10. True.
11. The *grub.cfg* file stores the location information of the boot partition.
12. You would add rd.break to the kernel boot string.
13. Nothing. The install process takes care of that.
14. It is called *systemd*.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 11-1: Modify GRUB2 Timeout

As *user1* with *sudo* on *server1*, modify the default timeout value from 5 seconds to 10 seconds in the */etc/default/grub* file and run *grub2-mkconfig* to apply the new value. Reboot the system and observe the new wait time in the bootloader menu. (Hint: The GRUB2 Bootloader).

Lab 11-2: Reset root User Password

As *root* on *server2*, reset the *root* user password by booting the system into emergency mode with SELinux disabled. Try to log in with *root* and enter the new password after the reboot. (Hint: The GRUB2 Bootloader).

Lab 11-3: Install New Kernel

As *user1* with *sudo* on *server1*, check the current version of the kernel using the *uname* or *rpm* command. Download a higher version from the Red Hat Customer Portal or *rpmfind.net*, and install it. Reboot the system and ensure the new kernel is listed on the bootloader menu. (Hint: The Linux Kernel).

Chapter 12

System Initialization, Message Logging, and System Tuning

This chapter describes the following major topics:

- Understand systemd, units, and targets
- Analyze service and target unit configuration files
- List and view status of running units
- Manage service units and target units
- Display and configure default system boot target
- Switch into non-default targets
- Analyze system log configuration file
- Examine log file rotation settings
- Review boot and system log files
- Record custom messages in system log file

- Describe systemd journal service
- Retrieve and scrutinize messages from journal
- Store journal information persistently
- Know system tuning and apply tuning profile

RHCSA Objectives:

12. Boot, reboot, and shut down a system normally
13. Boot systems into different targets manually
20. Start, stop, and check the status of network services
36. Start and stop services and configure services to start automatically at boot
37. Configure systems to boot into a specific target automatically
44. Configure network services to start automatically at boot
17. Manage tuning profiles
18. Locate and interpret system log files and journals
19. Preserve system journals

Systemd is the default system initialization and service management scheme. It boots the system into one of several predefined targets and it is used to handle operational states of services. It employs the concepts of units and targets for initialization, service administration, and state changes. A good grasp of what unit and target configuration files store is key to understanding how systemd operates.

Alerts and messages generated by system services and user activities are forwarded to predefined locations for storage. These alerts and messages include those that are produced during system boot time. The log data may be analyzed for debugging or auditing purposes. Log files grow over time and need to be rotated periodically to prevent the file system space from filling up. There are configuration files that define the default and custom locations to direct the log messages to and to configure rotation settings. The system log file records custom messages sent to it. systemd includes a service for viewing and managing system logs in addition to the traditional logging service. This service maintains a log of runtime activities for faster retrieval and can be configured to store the information permanently.

System tuning service is employed to monitor connected devices and to tweak their parameters to improve performance or conserve power. A recommended tuning profile may be identified and activated for optimal performance and power saving.

System Initialization and Service Management

systemd (short for *system daemon*) is the system initialization and service management mechanism. It has fast-tracked system initialization and state transitioning by introducing features such as parallel processing of startup scripts, improved handling of service dependencies, and on-demand activation of services. Moreover, it supports snapshotting of system states, tracks processes using control groups, and automatically maintains mount points. systemd is the first process with PID 1 that spawns at boot and it is the last process that terminates at shutdown.

systemd spawns several processes during a service startup. It places the processes in a private hierarchy composed of *control groups* (or *cgroups* for short) to organize processes for the purposes of monitoring and controlling system resources such as processor, memory, network bandwidth, and disk I/O. This includes limiting, isolating, and prioritizing their usage of resources. This way resources can be distributed among users, databases, and applications based on their needs and priorities, resulting in overall improved system performance.

In order to benefit from parallelism, systemd initiates distinct services concurrently, taking advantage of multiple CPU cores and other compute resources. To achieve this, systemd creates sockets for all enabled services that support socket-based activation instantaneously at the very beginning of the initialization process, and passes them on to service daemon processes as they attempt to start in parallel. This approach lets systemd handle inter-service order dependencies and allows services to start without any delays. With systemd, dependent daemons need not be running; they only need the correct socket to be available. systemd creates all sockets first, all the daemons next, and any client requests to daemons not yet running are cached

in the socket buffer and filled when the daemons come online.

Socket is a communication method that allows a single process to talk to another process on the same or remote system.

During the operational state, systemd maintains the sockets and uses them to reconnect other daemons and services that were interacting with an old instance of a daemon before that daemon was terminated or restarted. Likewise, services that use activation based on D-Bus (*Desktop Bus*) are started when a client application attempts to communicate with them for the first time. Additional methods used by systemd for activation are device-based and path-based, with the former starting the service when a specific hardware type such as USB is plugged in, and the latter starting the service when a particular file or directory alters its state.

D-Bus is another communication method that allows multiple services running in parallel on a system to talk to one another on the same or remote system.

With on-demand activation, systemd defers the startup of services—Bluetooth and printing—until they are actually needed during the boot process or during runtime. Together, parallelization and on-demand activation save time and compute resources, and contribute to expediting the boot process considerably.

Another major benefit of parallelism witnessed at system boot is when systemd uses the autofs service to temporarily mount the configured file systems. During the boot process, the file systems are checked that may result in unnecessary delays. With autofs, the file systems are temporarily mounted on their normal mount points, and as soon as the

checks on the file systems are finished, systemd remounts them using their standard devices. Parallelism in file system mounts does not affect the root and virtual file systems.

Units

Units are systemd objects used for organizing boot and maintenance tasks, such as hardware initialization, socket creation, file system mounts, and service startups. Unit configuration is stored in their respective configuration files, which are auto-generated from other configurations, created dynamically from the system state, produced at runtime, or user-developed. Units are in one of several operational states, including active, inactive, in the process of being activated or deactivated, and failed. Units can be enabled or disabled. An enabled unit can be started to an active state; a disabled unit cannot be started.

Units have a name and a type, and they are encoded in files with names in the form unitname.type. Some examples are tmp.mount, sshd.service, syslog.socket, and umount.target. There are two types of unit configuration files: (1) system unit files that are distributed with installed packages and located in the */usr/lib/systemd/system* directory, and (2) user unit files that are user-defined and stored in the */etc/systemd/user* directory (run **ls -l** on both directories to list their contents). This information can be vetted with the *pkg-config* command:

```
[user1@server1 ~]$ pkg-config systemd --variable=systemdsystemunitdir
/usr/lib/systemd/system
[user1@server1 ~]$ pkg-config systemd --variable=systemduserconfdir
/etc/systemd/user
```

Furthermore, there are additional system units that are created at runtime and destroyed when they are no longer needed. They are located in the */run/systemd/system* directory. These runtime unit files take precedence over the

system unit files, and the user unit files take priority over the runtime files.

Unit configuration files are a direct replacement of the initialization scripts found in the */etc/rc.d/init.d* directory in older RHEL releases.

systemd includes 11 unit types, which are described in Table 12-1.

Unit Type	Description
Automount	Offers automount capabilities for on-demand mounting of file systems
Device	Exposes kernel devices in systemd and may be used to implement device-based activation
Mount	Controls when and how to mount or unmount file systems.
Path	Activates a service when monitored files or directories are accessed
Scope	Manages foreign processes instead of starting them
Service	Starts, stops, restarts, or reloads service daemons and the processes they are made up of
Slice	May be used to group units, which manage system processes in a tree-like structure for resource management
Socket	Encapsulates local inter-process communication or network sockets for use by matching service units
Swap	Encapsulates swap partitions
Target	Defines logical grouping of units
Timer	Useful for triggering activation of other units based on timers

Table 12-1 systemd Unit Types

Unit files contain common and specific configuration elements. Common elements fall under the [Unit] and [Install] sections, and comprise the description,

documentation location, dependency information, conflict information, and other options that are independent of the type of unit. The unit-specific configuration data is located under the unit type section: [Service] for the service unit type, [Socket] for the socket unit type, and so forth. A sample unit file for *sshd.service* is shown below from the */usr/lib/systemd/system* directory:

```
[user1@server1 ~]$ cat /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=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
```

Units can have dependency relationships among themselves based on a sequence (ordering) or a requirement. A sequence outlines one or more actions that need to be taken before or after the activation of a unit (the Before and After directives). A requirement specifies what must already be running (the Requires directive) or not running (the Conflicts directive) in order for the successful launch of a unit. For instance, the *graphical.target* unit file tells us that the system must already be operating in the multi-user mode and not in rescue mode in order for it to boot successfully into the graphical mode. Another option, Wants, may be used instead of Requires in the [Unit] or [Install] section so that the unit is not forced to fail activation if a required unit fails to start. Run **man systemd.unit** for details on systemd unit files.

There are a few other types of dependencies that you may see in other unit configuration files. systemd generally sets and maintains inter-service dependencies automatically; however, this can be implemented manually if needed.

Targets

Targets are simply logical collections of units. They are a special systemd unit type with the .target file extension. They are also stored in the same directory locations as the other unit configuration files. Targets are used to execute a series of units. This is typically true for booting the system to a desired operational run level with all the required services up and running. Some targets inherit services from other targets and add their own to them. systemd includes several predefined targets that are described in Table 12-2.

Target	Description
halt	Shuts down and halts the system
poweroff	Shuts down and powers off the system
shutdown	Shuts down the system
rescue	Single-user target for running administrative and recovery functions. All local file systems are mounted. Some essential services are started, but networking remains disabled.
emergency	Runs an emergency shell. The root file system is mounted in read-only mode; other file systems are not mounted. Networking and other services remain disabled.
multi-user	Multi-user target with full network support, but without GUI
graphical	Multi-user target with full network support and GUI
reboot	Shuts down and reboots the system
default	A special soft link that points to the default system boot target (multi-user.target or graphical.target)
hibernate	Puts the system into hibernation by saving the running state of the system on the hard disk and powering it off. When powered up, the system restores from its saved state rather than booting up.

Table 12-2 systemd Targets

Target unit files contain all information under the [Unit] section, and it comprises the description, documentation location, and dependency and conflict information. A sample file for the *graphical.target* target is shown below from the */usr/lib/systemd/system* directory:

```
[user1@server1 ~]$ cat /usr/lib/systemd/system/graphical.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 file shows four dependencies: Requires, Wants, Conflicts, and After. It suggests that the system must have already accomplished the rescue.service, rescue.target, multi-user.target, and display-manager.service levels in order to be declared running in the graphical target. Run **man systemd.target** for details on systemd targets.

The systemctl Command

systemd comes with a set of management tools for querying and controlling operations. The primary tool for interaction in this command suite is *systemctl*. The *systemctl* command performs administrative functions and supports plentiful subcommands and flags. Table 12-3 lists and describes some common operations.

Subcommand	Description
daemon-reload	Re-reads and reloads all unit configuration files and recreates the entire user dependency tree.
enable (disable)	Activates (deactivates) a unit for autostart at system boot
get-default (set-default)	Shows (sets) the default boot target
get-property (set-property)	Returns (sets) the value of a property
is-active	Checks whether a unit is running
is-enabled	Displays whether a unit is set to autostart at system boot
is-failed	Checks whether a unit is in the failed state
isolate	Changes the running state of a system
kill	Terminates all processes for a unit
list-dependencies	Lists dependency tree for a unit
list-sockets	Lists units of type socket
list-unit-files	Lists installed unit files
list-units	Lists known units. This is the default behavior when systemctl is executed without any arguments.
mask (unmask)	Prohibits (permits) auto and manual activation of a unit to avoid potential conflict
reload	Forces a running unit to re-read its configuration file. This action does not change the PID of the running unit.
restart	Stops a running unit and restarts it
show	Shows unit properties
start (stop)	Starts (stops) a unit
status	Presents the unit status information

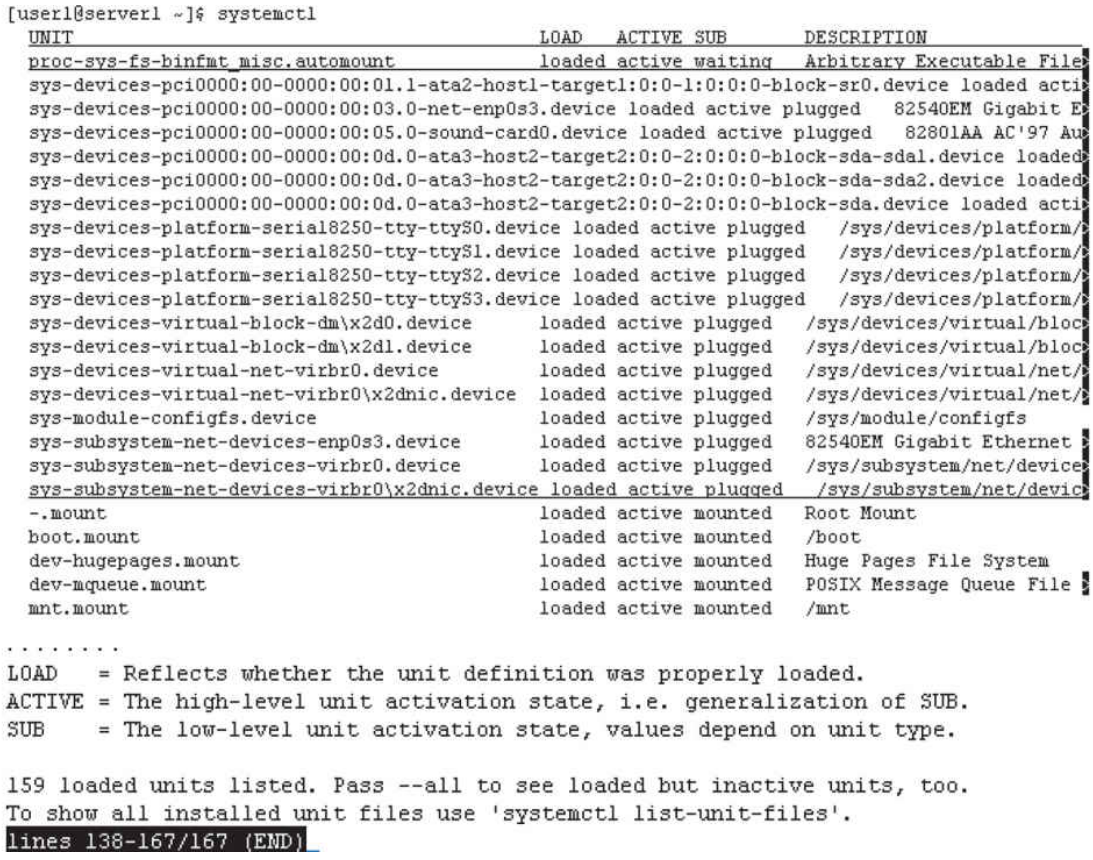
Table 12-3 systemctl Subcommands

You will use a majority of these subcommands with *systemctl* going forward. Refer to the manual pages of the command for more details.

Listing and Viewing Units

The *systemctl* command is used to view and manage all types of units. The following examples demonstrate common operations pertaining to viewing and querying units.

To list all units that are currently loaded in memory along with their status and description, run the *systemctl* command without any options or subcommands. A long output is generated. The graphic below shows a few starting and concluding lines followed by a brief explanation.



Here is a breakdown of the graphic above: the UNIT column shows the name of the unit and its location in the tree, the LOAD column reflects whether the unit configuration file was properly loaded (other possibilities are not found, bad setting, error, and masked), the ACTIVE column returns the high-level activation state (other possible states are active,

reloading, inactive, failed, activating, and deactivating), the SUB column depicts the low-level unit activation state (reports unit-specific information), and the DESCRIPTION column illustrates the unit's content and functionality.

By default, the *systemctl* command lists only the active units. You can use the --all option to include the inactive units too.

To list all (--all) active and inactive units of type (-t) socket:

```
[user1@server1 ~]$ systemctl -t socket --all
UNIT                          LOAD   ACTIVE   SUB       DESCRIPTION
avahi-daemon.socket           loaded active   running   Avahi mDNS/DNS-SD Stack Activation Sock>
cups.socket                   loaded active   running   CUPS Scheduler
dbus.socket                   loaded active   running   D-Bus System Message Bus Socket
dm-event.socket               loaded active   listening Device-mapper event daemon FIFOs
iscsid.socket                 loaded active   listening Open-iSCSI iscsid Socket
iscsiuio.socket               loaded active   listening Open-iSCSI iscsiuio Socket
lvm2-lvmpolld.socket          loaded active   listening LVM2 poll daemon socket
multipathd.socket             loaded active   listening multipathd control socket
rpcbind.socket                loaded active   running   RPCbind Server Activation Socket
spice-vdagentd.socket         loaded active   listening Activation socket for spice guest agent>
sssd-kcm.socket               loaded active   listening SSSD Kerberos Cache Manager responder s>
syslog.socket                 loaded inactive dead      Syslog Socket
```

To list all units of type socket (column 2) currently loaded in memory and the service they activate (column 3), sorted by the listening address (column 1):

```
[user1@server1 ~]$ systemctl list-sockets
LISTEN                                  UNIT                        ACTIVATES
/run/avahi-daemon/socket                avahi-daemon.socket         avahi-daemon.service
/run/dbus/system_bus_socket             dbus.socket                 dbus.service
/run/dmeventd-client                    dm-event.socket             dm-event.service
/run/dmeventd-server                    dm-event.socket             dm-event.service
/run/initctl                            systemd-initctl.socket      systemd-initctl.service
/run/lvm/lvmpolld.socket                lvm2-lvmpolld.socket        lvm2-lvmpolld.service
```

To list all unit files (column 1) installed on the system and their current state (column 2). This will generate a long list of units in the output. The following only shows a selection.

```
[user1@server1 ~]$ systemctl list-unit-files
UNIT FILE                                   STATE
proc-sys-fs-binfmt_misc.automount           static
-.mount                                     generated
boot.mount                                  generated
dev-hugepages.mount                         static
dev-mqueue.mount                            static
mnt.mount                                   generated
proc-fs-nfsd.mount                          static
```

To list all units that failed (--failed) to start at the last system boot:

```
[user1@server1 ~]$ systemctl --failed
  UNIT                      LOAD   ACTIVE SUB    DESCRIPTION
• vboxadd-service.service   loaded failed failed vboxadd-service.service
• vboxadd.service           loaded failed failed vboxadd.service

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

2 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

To list the hierarchy of all dependencies (required and wanted units) for the current default target:

```
[user1@server1 ~]$ systemctl list-dependencies
default.target
• ├─accounts-daemon.service
• ├─gdm.service
• ├─rtkit-daemon.service
• ├─systemd-update-utmp-runlevel.service
• ├─udisks2.service
• └─multi-user.target
•   ├─atd.service
•   ├─auditd.service
```

To list the hierarchy of all dependencies (required and wanted units) for a specific unit such as *atd.service*:

```
[user1@server1 ~]$ systemctl list-dependencies atd.service
atd.service
• ├─system.slice
• └─sysinit.target
•   ├─dev-hugepages.mount
•   ├─dev-mqueue.mount
•   ├─dracut-shutdown.service
•   ├─import-state.service
•   ├─iscsi.service
```

There are other listing subcommands and additional flags available that can be used to produce a variety of reports.

Managing Service Units

The *systemctl* command offers several subcommands to manage service units, including starting, stopping, restarting, and checking their status. These and other

management operations are summarized in Table 12-3. The following examples demonstrate their use on a service unit called *atd*.

To check the current operational status and other details for the *atd* service:

```
[user1@server1 ~]$ systemctl status atd
● atd.service - Job spooling tools
   Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-10-09 07:42:00 EDT; 1 day 4h ago
 Main PID: 942 (atd)
    Tasks: 1 (limit: 5064)
   Memory: 496.0K
   CGroup: /system.slice/atd.service
           └─942 /usr/sbin/atd -f
```

The above output reveals a lot of information about the *atd* service. On line 1, it shows the service description (read from the */usr/lib/systemd/system/atd.service* file). Line 2 illustrates the load status, which reveals the current load status of the unit configuration file in memory. Other possibilities for “Loaded” include “error” (if there was a problem loading the file), "not-found" (if no file associated with this unit was found), "bad-setting" (if a key setting was missing), and "masked" (if the unit configuration file is masked). Line 2 also tells us whether the service is set (enable or disable) for autostart at system boot.

Line 3 exhibits the current activation status and the time the service was started. An activation status designates the current state of the service. Possible states include:

Active (running): The service is running with one or more processes
Active (exited): Completed a one-time configuration
Active (waiting): Running but waiting for an event
Inactive: Not running
Activating: In the process of being activated
Deactivating: In the process of being deactivated
Failed: If the service crashed or could not be started

The output also depicts the PID of the service process and other information.

To disable the *atd* service from autostarting at the next system reboot:

```
[user1@server1 ~]$ sudo systemctl disable atd
Removed /etc/systemd/system/multi-user.target.wants/atd.service.
```

To re-enable *atd* to autostart at the next system reboot:

```
[user1@server1 ~]$ sudo systemctl enable atd
Created symlink /etc/systemd/system/multi-user.target.wants/atd.service → /usr/lib/systemd/system/
atd.service.
```

To check whether *atd* is set to autostart at the next system reboot:

```
[user1@server1 ~]$ systemctl is-enabled atd
enabled
```

To check whether the *atd* service is running:

```
[user1@server1 ~]$ systemctl is-active atd
active
```

To stop and restart *atd*, run either of the following:

```
[user1@server1 ~]$ sudo systemctl stop atd ; sudo systemctl start atd
[user1@server1 ~]$ sudo systemctl restart atd
```

To show the details of the *atd* service:

```
[user1@server1 ~]$ systemctl show atd
Type=simple
Restart=no
NotifyAccess=none
RestartUSec=100ms
```

. . . . . .

To prohibit *atd* from being enabled or disabled:

```
[user1@server1 ~]$ sudo systemctl mask atd
Created symlink /etc/systemd/system/atd.service → /dev/null.
```

Try disabling or enabling *atd* and observe the effect of the previous command:

```
[user1@server1 ~]$ sudo systemctl disable atd
Unit /etc/systemd/system/atd.service is masked, ignoring.
[user1@server1 ~]$ sudo systemctl enable atd
Failed to enable unit: Unit file /etc/systemd/system/atd.service is masked.
```

Reverse the effect of the *mask* subcommand and try disable and enable operations:

```
[user1@server1 ~]$ sudo systemctl unmask atd
Removed /etc/systemd/system/atd.service.
[user1@server1 ~]$
[user1@server1 ~]$ sudo systemctl disable atd
Removed /etc/systemd/system/multi-user.target.wants/atd.service.
[user1@server1 ~]$ sudo systemctl enable atd
Created symlink /etc/systemd/system/multi-user.target.wants/atd.service → /usr/lib/systemd/system/
atd.service.
```

Notice that the *unmask* subcommand has removed the restriction that was placed on the *atd* service.

Managing Target Units

The *systemctl* command is also used to manage the target units. It can be used to view or change the default boot target, switch from one running target into another, and so on. These operations are briefed in Table 12-3. Let’s look at some examples.

To view what units of type (-t) target are currently loaded and active:

```
[userl@serverl ~]$ systemctl -t target
UNIT                   LOAD   ACTIVE SUB    DESCRIPTION
basic.target           loaded active active Basic System
cryptsetup.target      loaded active active Local 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 System
network-online.target  loaded active active Network is Online
network-pre.target     loaded active active Network (Pre)
network.target         loaded active active Network
nfs-client.target      loaded active active NFS client services
nss-user-lookup.target loaded active active User and Group Name Lookups
paths.target           loaded active active Paths
remote-fs-pre.target   loaded active active Remote File Systems (Pre)
remote-fs.target       loaded active active Remote File Systems
rpc_pipefs.target      loaded active active rpc_pipefs.target
rpcbind.target         loaded active active RPC Port Mapper
slices.target          loaded active active Slices
sockets.target         loaded active active Sockets
sound.target           loaded active active Sound Card
sshd-keygen.target     loaded active active sshd-keygen.target
swap.target            loaded active active Swap
sysinit.target         loaded active active System Initialization
timers.target          loaded active active Timers

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

24 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

For each target unit, the above output returns the target unit’s name, load state, high-level and low-level activation states, and a short description. Add the --all option to the above to see all loaded targets in either active or inactive state.

Viewing and Setting Default Boot Target

The *systemctl* command is used to view the current default boot target and to set it. Let’s use the *get-default* and *set-default* subcommands with *systemctl* to perform these operations.

To check the current default boot target:

```
[userl@serverl ~]$ systemctl get-default
graphical.target
```

EXAM TIP: You may have to modify the default boot target persistently.

To change the current default boot target from graphical.target to multi-user.target:

```
[user1@server1 ~]$ sudo systemctl set-default multi-user
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/multi-user.target.
```

The command simply removes the existing symlink (*default.target*) pointing to the old boot target and replaces it with the new target file path.

Execute **sudo systemctl set-default graphical** to revert the default boot target to graphical.

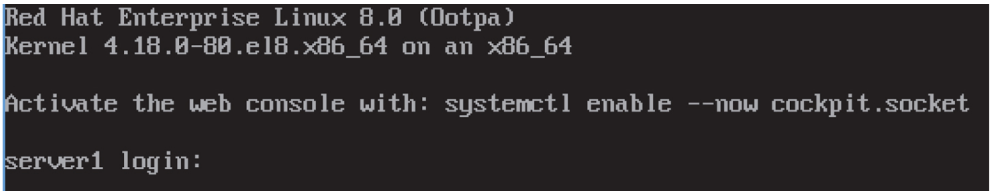
Switching into Specific Targets

The *systemctl* command can be used to transition the running system from one target state into another. There are a variety of potential targets available to switch into as listed in Table 12-2; however, only a few of them—graphical, multi-user, reboot, shutdown—are typically used. The rescue and emergency targets are for troubleshooting and system recovery purposes, poweroff and halt are similar to shutdown, and hibernate is suitable for mobile devices. Consider the following examples that demonstrate switching targets.

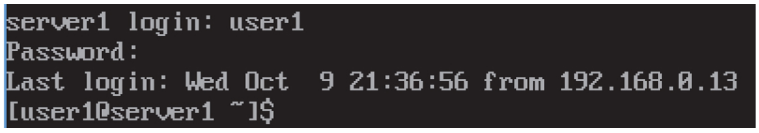
The current default target on *server1* is graphical. To switch into multi-user, use the *isolate* subcommand with *systemctl*:

```
[user1@server1 ~]$ sudo systemctl isolate multi-user
```

This should stop the graphical service on the system and display the text-based console login screen, as shown below:



Type in a username such as *user1* and enter the password to log in:



To return to the graphical target:

The graphical login screen should appear shortly and you should be able to log back in.

To shut down the system and power it off, use the following or simply run the *poweroff* command:

```
[user1@server1 ~]$ sudo systemctl poweroff
```

To shut down and reboot the system, use the following or simply run the *reboot* command:

```
[user1@server1 ~]$ sudo systemctl reboot
```

The *halt*, *poweroff*, and *reboot* commands are mere symbolic links to the *systemctl* command, as the following long listing suggests:

```
[user1@server1 ~]$ ls -l /usr/sbin/halt /usr/sbin/poweroff /usr/sbin/reboot
lrwxrwxrwx. 1 root root 16 Feb 26  2019 /usr/sbin/halt -> ../bin/systemctl
lrwxrwxrwx. 1 root root 16 Feb 26  2019 /usr/sbin/poweroff -> ../bin/systemctl
lrwxrwxrwx. 1 root root 16 Feb 26  2019 /usr/sbin/reboot -> ../bin/systemctl
```

The *halt*, *poweroff*, and *reboot* commands are available in RHEL 8 for compatibility reasons only. It is recommended to

use the *systemctl* command instead when switching system states.

The three commands, without any arguments, perform the same action that the *shutdown* command would with the “-H now”, “-P now”, and “-r now” arguments, respectively. In addition, it also broadcasts a warning message to all logged-in users, blocks new user login attempts, waits for the specified amount of time for users to save their work and log off, stops the services, and eventually shut the system down to the specified target state.

System Logging

System logging (*syslog* for short) is one of the most rudimentary elements of an operating system. Its purpose is to capture messages generated by the kernel, daemons, commands, user activities, applications, and other events, and forwarded them to various log files, which store them for security auditing, service malfunctioning, system troubleshooting, or informational purposes.

The daemon that is responsible for system logging is called *rsyslogd* (*rocket-fast system for log processing*). This service daemon is multi-threaded, with support for enhanced filtering, encryption-protected message relaying, and a variety of configuration options. The *rsyslogd* daemon reads its configuration file */etc/rsyslog.conf* and the configuration files located in the */etc/rsyslog.d* directory at startup. The default depository for most system log files is the */var/log* directory. Other services such as audit, Apache, and GNOME desktop manager have subdirectories under */var/log* for storing their respective log files.

The *rsyslog* service is modular, allowing the modules listed in its configuration file to be dynamically loaded in the kernel

as and when needed. Each module brings a new functionality to the system upon loading.

The *rsyslogd* daemon can be stopped manually using **systemctl stop rsyslog**. Replace stop with start, restart, reload, and status as appropriate.

A PID is assigned to the daemon at startup and a file by the name *rsyslogd.pid* is created in the */run* directory to save the PID. The reason this file is created and stores the PID is to prevent the initiation of multiple instances of this daemon.

The Syslog Configuration File

The *rsyslog.conf* is the primary syslog configuration file located in the */etc* directory . The default uncommented line entries from the file are shown below and explained thereafter. Section headings have been added to separate the directives in each section.

```
[user1@server1 ~]$ sudo grep '^#|^$' /etc/rsyslog.conf
#### MODULES ####
module(load="imuxsock"    # provides support for local system logging (e.g. via l
       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
global(workDirectory="/var/lib/rsyslog")
#### GLOBAL DIRECTIVES ####
global(workDirectory="/var/lib/rsyslog")
module(load="builtin:omfile" Template="RSYSLOG_TraditionalFileFormat")
include(file="/etc/rsyslog.d/*.conf" mode="optional")
#### RULES ####
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
authpriv.*                                              /var/log/secure
mail.*                                                  -/var/log/maillog
cron.*                                                  /var/log/cron
*.emerg                                                 :omusrmsg:*
uucp,news.crit                                          /var/log/spooler
local7.*                                                /var/log/boot.log
```

As depicted, the syslog configuration file contains three sections: Modules, Global Directives, and Rules. The Modules section defines two modules—*imuxsock* and *imjournal*—and they are loaded on demand. The *imuxsock* module furnishes

support for local system logging via the *logger* command, and the *imjournal* module allows access to the systemd journal.

The Global Directives section contains three active directives. The definitions in this section influence the overall functionality of the *rsyslog* service. The first directive sets the location for the storage of auxiliary files (*/var/lib/rsyslog*). The second directive instructs the *rsyslog* service to save captured messages using traditional file formatting. The third directive directs the service to load additional configuration from files located in the */etc/rsyslogd.d/* directory.

The Rules section has many two-field line entries. The left field is called *selector*, and the right field is referred to as *action*. The selector field is further divided into two period-separated sub-fields called *facility* (left) and *priority* (right), with the former representing one or more system process categories that generate messages, and the latter identifying the severity associated with the messages. The semicolon (;) character is used as a distinction mark if multiple facility.priority groups are present. The action field determines the destination to send the messages to.

There are numerous supported facilities such as auth, authpriv, cron, daemon, kern, lpr, mail, news, syslog, user, uucp, and local0 through local7. The asterisk (*) character represents all of them.

Similarly, there are several supported priorities, and they include emerg, alert, crit, error, warning, notice, info, debug, and none. This sequence is in the descending criticality order. The asterisk (*) represents all of them. If a lower priority is selected, the daemon logs all messages of the service at that and higher levels.

Line 1 under the Rules section instructs the syslog daemon to catch and store informational messages from all services to the */var/log/messages* file and ignore all messages generated by mail, authentication, and cron services. Lines 2, 3, and 4 command the daemon to collect and log all messages produced by authentication, mail, and cron to the *secure*, *maillog*, and *cron* files, respectively. Line 5 orders the daemon to display emergency messages (omusrmsg stands for *user message output module*) on the terminals of all logged-in users. Line 6 shows two comma-separated facilities that are set at a common priority. These facilities tell the daemon to gather critical messages from uucp and news facilities and log them to the */var/log/spooler* file. Line 7 (the last line) is for recording the boot-time service startup status to the */var/log/boot.log* file.

If you have made any modifications to the syslog configuration file, you need to run the *rsyslogd* command with the -N switch and specify a numeric verbosity level to inspect the file for any syntax or typing issues:

```
[user1@server1 ~]$ sudo rsyslogd -N 1
rsyslogd: version 8.37.0-9.el8, config validation run (level 1), master config /etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

The validation returns the version of the command, verbosity level used, and the configuration file path. With no issues reported, the *rsyslog* service can be restarted (or reloaded) in order for the changes to take effect.

Rotating Log Files

RHEL records all system activities in log files that are stored in a central location under the */var/log* directory, as defined in the rsyslog configuration file. A long listing of this directory reveals the files along with subdirectories that may have multiple service-specific logs. Here is a listing from *server1*:

```
[user1@server1 ~]$ ls -l /var/log
total 6852
drwxr-xr-x. 2 root   root       250 Aug 23 06:27 anaconda
drwx------. 2 root   root        23 Aug 23 06:30 audit
-rw-------. 1 root   root         0 Oct 15 03:33 boot.log
-rw-rw----. 1 root   utmp         0 Oct  1 03:23 btmp
-rw-rw----. 1 root   utmp         0 Aug 23 06:08 btmp-20191001
drwxr-xr-x. 2 chrony chrony       6 Aug 13  2018 chrony
-rw-------. 1 root   root     15578 Oct 15 07:01 cron
drwxr-xr-x. 2 lp     sys        135 Sep 29 13:39 cups
-rw-------. 1 root   root    656807 Oct 15 07:23 dnf.log
-rw-------. 1 root   root    534386 Oct  6 03:05 dnf.log-20191006
-rw-------. 1 root   root    212004 Oct 13 02:35 dnf.log-20191013
-rw-------. 1 root   root      6424 Oct 15 07:23 dnf.rpm.log
-rw-r-----. 1 root   root      2100 Oct  6 23:33 firewalld
drwx--x--x. 2 root   gdm          6 Feb 11  2019 gdm
drwxr-xr-x. 2 root   root         6 Mar  8  2019 glusterfs
drwxr-xr-x. 2 root   root         6 Aug 23 06:16 insights-client
-rw-rw-r--. 1 root   utmp    295504 Oct 14 17:49 lastlog
-rw-------. 1 root   root         0 Oct 13 03:17 maillog
-rw-------. 1 root   root    782565 Oct 15 07:23 messages
-rw-------. 1 root   root   1430871 Oct  6 03:05 messages-20191006
-rw-------. 1 root   root   3133990 Oct 13 02:35 messages-20191013
drwx------. 2 root   root         6 Aug 23 06:08 private
-rw-------. 1 root   root     46185 Oct 15 07:56 secure
-rw-------. 1 root   root         0 Oct 13 03:17 spooler
drwxr-xr-x. 3 root   root        21 Aug 23 06:15 swtpm
-rw-rw-r--. 1 root   utmp     77568 Oct 14 15:12 wtmp
-rw-r--r--. 1 root   root     18990 Aug 23 06:31 Xorg.9.log
```

The output shows log files for various services. Depending on the usage and the number of events generated and captured, log files may quickly fill up the */var* file system, resulting in unpredictable system behavior. Also, they may grow to an extent that would make it difficult to load, read, send, or analyze them. To avoid getting into any unwanted situation, it’s important to ensure that they’re rotated on a regular basis and their archives are removed automatically.

To that end, a script called *logrotate* under */etc/cron.daily/* invokes the *logrotate* command on a daily basis. Via the Anacron service, the command runs a rotation as per the schedule defined in the */etc/logrotate.conf* file and the configuration files for various services located in the */etc/logrotate.d* directory. The configuration files may be modified to alter the schedule or include additional tasks such as removing, compressing, and emailing selected log files.

Here is what the */etc/cron.daily/logrotate* script contains:

```
[user1@server1 ~]$ cat /etc/cron.daily/logrotate
#!/bin/sh

/usr/sbin/logrotate /etc/logrotate.conf
EXITVALUE=$?
if [ $EXITVALUE != 0 ]; then
    /usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
fi
exit $EXITVALUE
```

The following returns the default content of the */etc/logrotate.conf* file:

```
[user1@server1 ~]$ cat /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d
```

The file content shows the default log rotation frequency (weekly). It indicates the period of time (4 weeks) to retain the rotated logs before deleting them. Each time a log file is rotated, an empty replacement file is created with the date as a suffix to its name, and the *rsyslog* service is restarted. The script presents the option of compressing the rotated files using the *gzip* utility. During the script execution, the *logrotate* command checks for the presence of additional log configuration files in the */etc/logrotate.d* directory and includes them as necessary. The directives defined in the *logrotate.conf* file have a global effect on all log files. You can define custom settings for a specific log file in *logrotate.conf* or create a separate file in the */etc/logrotate.d* directory. Any

settings defined in user-defined files overrides the global settings.

The */etc/logrotate.d* directory includes additional configuration files for other service logs, as shown below:

```
[user1@server1 ~]$ ls -l /etc/logrotate.d/
total 64
-rw-r--r--. 1 root root  91 Feb  8  2019 bootlog
-rw-r--r--. 1 root root 130 Feb 19  2018 btmp
-rw-r--r--. 1 root root 160 Apr  4  2018 chrony
-rw-r--r--. 1 root root  71 Dec 14  2018 cups
-rw-r--r--. 1 root root 526 Jan  4  2019 dnf
-rw-r--r--. 1 root root 172 Jul  9  2018 iscsiuiolog
-rw-r--r--. 1 root root 165 Feb 15  2019 libvirtd
-rw-r--r--. 1 root root 142 Feb 15  2019 libvirtd.qemu
-rw-r--r--. 1 root root 106 Jun  2  2015 numad
-rw-r--r--. 1 root root 408 Aug 12  2018 psacct
-rw-r--r--. 1 root root 115 Jan  7  2019 samba
-rw-r--r--. 1 root root 237 Feb 11  2019 sssd
-rw-r--r--. 1 root root 226 Dec 17  2018 syslog
-rw-r--r--. 1 root root  32 Feb 19  2018 up2date
-rw-r--r--. 1 root root 100 Feb  8  2019 wpa_supplicant
-rw-r--r--. 1 root root 145 Feb 19  2018 wtmp
```

Here there are log configuration files for a number of services—*chrony*, *cups*, *dnf*, and *samba*—all with their own rules defined. The following shows the file content for *btmp* (records of failed user login attempts) that is used to control the rotation behavior for the */var/log/btmp* file:

```
[user1@server1 ~]$ cat /etc/logrotate.d/btmp
# no packages own btmp -- we'll rotate it here
/var/log/btmp {
    missingok
    monthly
    create 0600 root utmp
    rotate 1
}
```

The rotation is once a month. The replacement file created will get read/write permission bits for the owner (*root*), the owning group will be set to *utmp*, and the *rsyslog* service will maintain one rotated copy of the *btmp* log file.

The Boot Log File

Logs generated during the system startup display the service startup sequence with a status showing whether the service was started successfully. This information may help in any post-boot troubleshooting if required. Boot logs are stored in the *boot.log* file under */var/log*. Here are a few lines from the beginning of the file:

```
[user1@server1 ~]$ sudo head /var/log/boot.log
------------ Tue Oct 15 09:16:18 EDT 2019 ------------
[  OK  ] Stopped Journal Service.
         Starting Journal Service...
[  OK  ] Listening on initctl Compatibility Named Pipe.
[  OK  ] Listening on Process Core Dump Socket.
[  OK  ] Created slice system-getty.slice.
         Starting Read and set NIS domainname from /etc/sysconfig/network...
[  OK  ] Listening on udev Kernel Socket.
[  OK  ] Set up automount Arbitrary Executable File Formats File System Automount Point.
[  OK  ] Listening on Device-mapper event daemon FIFOs.
```

OK or FAILED within the square brackets indicates if the service was started successfully or not.

The System Log File

The default location for storing most system activities, as defined in the *rsyslog.conf* file, is the */var/log/messages* file. This file saves log information in plain text format and may be viewed with any file display utility, such as *cat*, *more*, *pg*, *less*, *head*, or *tail*. This file may be observed in real time using the *tail* command with the -f switch. The *messages* file captures the date and time of the activity, hostname of the system, name and PID of the service, and a short description of the event being logged.

> **EXAM TIP:** It is helpful to “tail” the messages file when starting or restarting a system service or during testing to identify any issues encountered.

The following illustrates some recent entries from this file:

```
[userl@serverl ~]$ sudo tail /var/log/messages
Oct 15 09:36:24 serverl dnf[2303]: Metadata cache refreshed recently.
Oct 15 09:36:25 serverl systemd[1]: Started dnf makecache.
Oct 15 09:41:23 serverl systemd[1]: Starting Cleanup of Temporary Directories...
Oct 15 09:41:23 serverl systemd-tmpfiles[2349]: [/usr/lib/tmpfiles.d/libstoragemgmt.conf:1] Line references
path below legacy directory /var/run/, updating /var/run/lsm → /run/lsm; please update the tmpfiles.d/ drop-
in file accordingly.
Oct 15 09:41:23 serverl systemd-tmpfiles[2349]: [/usr/lib/tmpfiles.d/libstoragemgmt.conf:2] Line references
path below legacy directory /var/run/, updating /var/run/lsm/ipc → /run/lsm/ipc; please update the tmpfiles.
d/ drop-in file accordingly.
Oct 15 09:41:23 serverl systemd-tmpfiles[2349]: [/usr/lib/tmpfiles.d/mdadm.conf:1] Line references path belo
w legacy directory /var/run/, updating /var/run/mdadm → /run/mdadm; please update the tmpfiles.d/ drop-in fi
le accordingly.
Oct 15 09:41:23 serverl systemd-tmpfiles[2349]: [/usr/lib/tmpfiles.d/radvd.conf:1] Line references path belo
w legacy directory /var/run/, updating /var/run/radvd → /run/radvd; please update the tmpfiles.d/ drop-in fi
le accordingly.
```

Each line entry represents the detail for one event.

Logging Custom Messages

Many times it is worthwhile to add a manual note to the system log file to mark the start or end of an activity for future reference. This is especially important when you run a script to carry out certain tasks and you want to record the status or add comments at various stages throughout its execution. This is also beneficial in debugging the startup of an application to know where exactly it is failing.

The Modules section in the *rsyslog.conf* file provides the support via the imuxsock module to record custom messages to the *messages* file using the *logger* command. This command may be run by normal users or the *root* user. The following example shows how to add a note indicating the calling user has rebooted the system:

```
[userl@serverl ~]$ logger -i "System rebooted by $USER"
```

tail the last line from the *messages* file and you'll observe the message recorded along with the timestamp, hostname, and PID:

```
[userl@serverl ~]$ sudo tail -1 /var/log/messages
Oct 15 09:59:22 serverl userl[2533]: System rebooted by userl
```

You may add the -p option and specify a priority level either as a numerical value or in the facility.priority format. The

default priority at which the events are recorded is user.notice. See the manual pages for the *logger* command for more details.

The systemd Journal

In addition to the *rsyslog* service, RHEL offers a systemd-based logging service for the collection and storage of logging data. This service is implemented via the *systemd-journald* daemon. The function of this service is to gather, store, and display logging events from a variety of sources such as the kernel, *rsyslog* and other services, initial RAM disk, and alerts generated during the early boot stage. It stores these messages in the binary format in files called *journals* that are located in the */run/log/journal* directory. These files are structured and indexed for faster and easier searches, and may be viewed and managed using the *journalctl* command. As you know, */run* is a virtual file system that is created in memory at system boot, maintained during system runtime, and destroyed at shutdown. Therefore, the data stored therein is non-persistent, but you can enable persistent storage for the logs if desired.

RHEL runs both *rsyslogd* and *systemd-journald* concurrently. In fact, the data gathered by *systemd-journald* may be forwarded to *rsyslogd* for further processing and persistent storage in text format.

The main configuration file for this service is */etc/systemd/journald.conf*, which contains numerous default settings that affect the overall functionality of the service. These settings may be modified as required.

Retrieving and Viewing Messages

RHEL provides the *journalctl* command to retrieve messages from the journal for viewing in a variety of ways using different options. One common usage is to run the command without any options to see all the messages generated since the last system reboot. The following shows a few initial entries from the journal:

```
[user1@server1 ~]$ sudo journalctl
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:15:01 EDT. --
Oct 15 09:25:41 server1.example.com kernel: Linux version 4.18.0-80.11.2.el8_0.x86_64 (mockbuild@x86-vm-07.>
Oct 15 09:25:41 server1.example.com kernel: Command line: BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-80.11.2.el>
Oct 15 09:25:41 server1.example.com kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point re>
Oct 15 09:25:41 server1.example.com kernel: x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
Oct 15 09:25:41 server1.example.com kernel: x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
Oct 15 09:25:41 server1.example.com kernel: x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
Oct 15 09:25:41 server1.example.com kernel: x86/fpu: Enabled xstate features 0x7, context size is 832 bytes>
Oct 15 09:25:41 server1.example.com kernel: BIOS-provided physical RAM map:
```

Notice that the format of the messages is similar to that of the events logged to the */var/log/messages* file that you saw earlier. Each line begins with a timestamp followed by the system hostname, process name with or without a PID, and the actual message.

Let’s run the *journalctl* command with different options to produce various reports.

To display detailed output for each entry, use the -o verbose option:

```
[user1@server1 ~]$ sudo journalctl -o verbose
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:19:09 EDT. --
Tue 2019-10-15 09:25:41.923932 EDT [s=ffd2564870b64db7a6a4ca448f53266b;i=1;b=e75da4e8e6fc4ebb867dd01caa6ef5>
    _SOURCE_MONOTONIC_TIMESTAMP=0
    _TRANSPORT=kernel
    PRIORITY=5
    SYSLOG_FACILITY=0
    SYSLOG_IDENTIFIER=kernel
    MESSAGE=Linux version 4.18.0-80.11.2.el8_0.x86_64 (mockbuild@x86-vm-07.build.eng.bos.redhat.com) (gcc v>
    _BOOT_ID=e75da4e8e6fc4ebb867dd01caa6ef544
    _MACHINE_ID=2d06e883e31f434f95d9df6035e988e7
    _HOSTNAME=server1.example.com
Tue 2019-10-15 09:25:41.924033 EDT [s=ffd2564870b64db7a6a4ca448f53266b;i=2;b=e75da4e8e6fc4ebb867dd01caa6ef5>
    _SOURCE_MONOTONIC_TIMESTAMP=0

  ........
```

To view all events since the last system reboot, use the -b options:

```
[user1@server1 ~]$ sudo journalctl -b
```

You may specify -0 (default, since the last system reboot), -1 (the previous system reboot), -2 (two reboots before), and so on to view messages from previous system reboots.

To view only kernel-generated alerts since the last system reboot:

```
[userl@serverl ~]$ sudo journalctl -kb0
```

To limit the output to view a specific number of entries only (3 in the example below), use the -n option:

```
[userl@serverl ~]$ sudo journalctl -n3
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:29:36 EDT. --
Oct 15 12:29:36 serverl.example.com sudo[4160]: userl : TTY=pts/0 ; PWD=/home/userl ; USER=root ; COMMAND=/>
Oct 15 12:29:36 serverl.example.com sudo[4160]: pam_systemd(sudo:session): Cannot create session: Already r>
Oct 15 12:29:36 serverl.example.com sudo[4160]: pam_unix(sudo:session): session opened for user root by use>
```

To show all alerts generated by a particular service, such as *crond*:

```
[userl@serverl ~]$ sudo journalctl /usr/sbin/crond
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:31:15 EDT. --
Oct 15 09:25:56 serverl.example.com crond[912]: (CRON) STARTUP (1.5.2)
Oct 15 09:25:56 serverl.example.com crond[912]: (CRON) INFO (Syslog will be used instead of sendmail.)
Oct 15 09:25:56 serverl.example.com crond[912]: (CRON) INFO (RANDOM_DELAY will be scaled with factor 9% if >
Oct 15 09:25:59 serverl.example.com crond[912]: (CRON) INFO (running with inotify support)
```

To retrieve all messages logged for a certain process, such as the PID associated with the *chronyd* service:

```
[userl@serverl ~]$ sudo journalctl _PID=$(pgrep chronyd)
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:35:39 EDT. --
Oct 15 09:25:51 serverl.example.com chronyd[784]: chronyd version 3.3 starting (+CMDMON +NTP +REFCLOCK +RTC>
Oct 15 09:25:51 serverl.example.com chronyd[784]: Frequency -33.628 +/- 4.368 ppm read from /var/lib/chrony>
Oct 15 09:25:51 serverl.example.com chronyd[784]: Using right/UTC timezone to obtain leap second data
Oct 15 09:26:10 serverl.example.com chronyd[784]: Selected source 207.34.49.172
Oct 15 09:26:10 serverl.example.com chronyd[784]: System clock TAI offset set to 37 seconds
```

To reveal all messages for a particular system unit, such as sshd.service:

```
[userl@serverl ~]$ sudo journalctl _SYSTEMD_UNIT=sshd.service
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:39:53 EDT. --
Oct 15 09:25:56 serverl.example.com sshd[894]: Server listening on 0.0.0.0 port 22.
Oct 15 09:25:56 serverl.example.com sshd[894]: Server listening on :: port 22.
Oct 15 09:27:40 serverl.example.com sshd[2150]: Accepted password for userl from 192.168.0.219 port 49876 s>
Oct 15 09:27:41 serverl.example.com sshd[2150]: pam_unix(sshd:session): session opened for user userl by (u>
```

To view all error messages logged between a date range, such as October 10, 2019 and October 16, 2019:

```
[user1@server1 ~]$ sudo journalctl --since 2019-10-10 --until 2019-10-16 -p err
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:44:26 EDT. --
Oct 15 09:25:43 server1.example.com kernel: [drm:vmw_host_log [vmwgfx]] *ERROR* Failed to send log
Oct 15 09:25:43 server1.example.com kernel: [drm:vmw_host_log [vmwgfx]] *ERROR* Failed to send log
Oct 15 09:25:52 server1.example.com sssd[788]: Could not open file [/var/log/sssd/sssd.log]. Error: [2][No
```

To get all warning messages that have appeared today and display them in reverse chronological order:

```
[user1@server1 ~]$ sudo journalctl --since today -p warning -r
-- Logs begin at Tue 2019-10-15 09:25:41 EDT, end at Tue 2019-10-15 12:46:28 EDT. --
Oct 15 12:29:31 server1.example.com packagekitd[4154]: g_path_get_basename: assertion 'file_name != NULL' f
Oct 15 11:55:19 server1.example.com packagekitd[3672]: g_path_get_basename: assertion 'file_name != NULL' f
Oct 15 09:26:32 server1.example.com kernel: [drm:vmw_sou_crtc_page_flip [vmwgfx]] *ERROR* Page flip error -
Oct 15 09:26:32 server1.example.com gnome-shell[1076]: Failed to flip: Device or resource busy
Oct 15 09:26:30 server1.example.com gsd-color[1891]: unable to get EDID for xrandr-Virtual-1: unable to get
```

You can specify the time range in hh:mm:ss format, or yesterday, today, or tomorrow instead.

Similar to the -f (follow) option that is used with the *tail* command for real-time viewing of a log file, you can use the same switch with *journalctl* as well. Press Ctrl+c to terminate.

```
[user1@server1 ~]$ sudo journalctl -f
```

Check the manual pages of the *journalctl* command and the *systemd-journald* service for more details.

Preserving Journal Information

By default, journals are stored in the */run/log/journal* directory for the duration of system runtime. This data is transient and it does not survive across reboots. The *journalctl* command examples demonstrated earlier retrieve journal information from this temporary location. The *rsyslogd* daemon, by default, reads the temporary journals and stores messages in the */var/log/messages* file. You can enable a separate storage location for the journal to save all its messages there persistently. The default is under the */var/log/journal* directory. This will make the journal information available for future reference.

The *systemd-journald* service supports four options with the Storage directive in its configuration file *journald.conf* to control how the logging data is handled. These options are described in Table 12-4.

Option	Description
volatile	Stores data in memory only
persistent	Stores data permanently under /var/log/journal and falls back to memory-only option if this directory does not exist or has a permission or other issue. The service creates /var/log/journal in case of its non-existence.
auto	Similar to “persistent” but does not create /var/log/journal if it does not exist. This is the default option.
none	Disables both volatile and persistent storage options. Not recommended.

Table 12-4 Journal Data Storage Options

The default (auto) option appears more suitable as it stores data in both volatile and on-disk storage; however, you need to create the */var/log/journal* directory manually. This option provides two fundamental benefits: faster query responses from in-memory storage and access to historical log data from on-disk storage.

To enable persistent storage without changing the auto option, simply create the target directory. The service will automatically start writing the log data there.

```
[user1@server1 ~]$ sudo mkdir /var/log/journal
[user1@server1 ~]$ ls -l /var/log/journal
total 0
drwxr-xr-x. 2 root root 28 Oct 17 11:40 2d06e883e31f434f95d9df6035e988e7
[user1@server1 ~]$ ls -l /var/log/journal/2d06e883e31f434f95d9df6035e988e7/
total 8192
-rw-r-----. 1 root root 8388608 Oct 17 11:40 system.journal
```

As soon as you created the */var/log/journal* directory, a subdirectory matching the system’s machine ID was created

underneath. The file *system.journal* stores the log data in binary format.

The */etc/machine-id* stores the system's machine ID, which the *systemd-journald* service uses to create the subdirectory under */var/log/journal*. You can run **cat /etc/machine-id** to view the file content.

This log file is rotated automatically once a month based on the settings in the *journald.conf* file. Check the manual pages of the configuration file for details and relevant directives.

System Tuning

RHEL uses a system tuning service called *tuned* to monitor storage, networking, processor, audio, video, and a variety of other connected devices, and adjusts their parameters for better performance or power saving based on a chosen profile. There are several predefined tuning profiles for common use cases shipped with RHEL that may be activated either statically or dynamically.

The *tuned* service activates a selected profile at service startup and continues to use it until it is switched to a different profile. This is the static behavior and it is enabled by default.

The dynamic alternative adjusts the system settings based on the live activity data received from monitored system components to ensure optimal performance. In most cases, the utilization of system components vary throughout the day. For example, a disk and processor's utilization increases during a program startup and the network connection use goes up during a large file transfer. A surge in a system

component activity results in heightened power consumption.

Tuning Profiles

tuned includes nine predefined profiles to support a variety of use cases. In addition, you can create custom profiles from nothing or by using one of the existing profiles as a template. In either case, you need to store the custom profile under the */etc/tuned* directory in order to be recognized by the *tuned* service.

Tuning profiles may be separated into three groups: (1) optimized for better performance, (2) geared more towards power consumption, and (3) that offers a balance between the other two and the maximum performance/power combination. Table 12-5 lists and describes these profiles.

Profile	Description
Profiles Optimized for Better Performance	
Desktop	Based on the balanced profile for desktop systems. Offers improved throughput for interactive applications.
Latency-performance	For low-latency requirements
Network-latency	Based on the latency-performance for faster network throughput
Network-throughput	Based on the throughput-performance profile for maximum network throughput
Virtual-guest	Optimized for virtual machines
Virtual-host	Optimized for virtualized hosts
Profiles Optimized for Power Saving	
Powersave	Saves maximum power at the cost of performance
Balanced/Max Profiles	
Balanced	Preferred choice for systems that require a balance between performance and power saving
Throughput-performance	Provides maximum performance and consumes maximum power

Table 12-5 Tuning Profiles

Predefined profiles are located in the */usr/lib/tuned* directory in subdirectories matching their names. The following shows a long listing of the directory:

```
[user1@server1 ~]$ ls -l /usr/lib/tuned
total 16
drwxr-xr-x. 2 root root    24 Aug 23 06:16 balanced
drwxr-xr-x. 2 root root    24 Aug 23 06:16 desktop
-rw-r--r--. 1 root root 14413 Feb 22  2019 functions
drwxr-xr-x. 2 root root    24 Aug 23 06:16 latency-performance
drwxr-xr-x. 2 root root    24 Aug 23 06:16 network-latency
drwxr-xr-x. 2 root root    24 Aug 23 06:16 network-throughput
drwxr-xr-x. 2 root root    41 Aug 23 06:16 powersave
drwxr-xr-x. 2 root root    27 Aug 23 06:16 recommend.d
drwxr-xr-x. 2 root root    24 Aug 23 06:16 throughput-performance
drwxr-xr-x. 2 root root    24 Aug 23 06:16 virtual-guest
drwxr-xr-x. 2 root root    24 Aug 23 06:16 virtual-host
```

The default active profile set on *server1* and *server2* is the *virtual-guest* profile, as the two systems are hosted in a VirtualBox virtualized environment.

The tuned-adm Command

tuned comes with a single profile management command called *tuned-adm*. This tool can list active and available profiles, query current settings, switch between profiles, and turn the tuning off. This command can also recommend the best profile for the system based on many system attributes. Refer to the manual pages of the command for more details.

The following exercise demonstrates the use of most of the management operations listed above.

Exercise 12-1: Manage Tuning Profiles

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will install the *tuned* service, start it now, and enable it for auto-restart upon future system reboots. You will display all available profiles and the current active profile. You will switch to one of the available profiles and confirm. You will determine the recommended profile for the system and switch to it. Finally, you will deactivate tuning and reactivate it. You will confirm the activation to conclude the exercise.

1. Install the *tuned* package if it is not already installed:

```
[user1@server1 ~]$ sudo dnf install -y tuned
Package tuned-2.10.0-15.el8.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

The output indicates that the software is already installed.

2. Start the *tuned* service and set it to autostart at reboots:

```
[user1@server1 ~]$ sudo systemctl --now enable tuned
```

3. Confirm the startup:

```
[user1@server1 ~]$ sudo systemctl status tuned
● tuned.service - Dynamic System Tuning Daemon
   Loaded: loaded (/usr/lib/systemd/system/tuned.service; enabled; vendor prese>
   Active: active (running) since Tue 2019-12-10 23:16:19 EST; 19min ago
     Docs: man:tuned(8)
           man:tuned.conf(5)
           man:tuned-adm(8)
 Main PID: 881 (tuned)
    Tasks: 4 (limit: 5076)
   Memory: 2.9M
   CGroup: /system.slice/tuned.service
           └─881 /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P

Dec 10 23:16:16 server1.example.com systemd[1]: Starting Dynamic System Tuning >
Dec 10 23:16:19 server1.example.com systemd[1]: Started Dynamic System Tuning D>
```

4. Display the list of available tuning profiles:

```
[user1@server1 ~]$ sudo tuned-adm list
Available profiles:
- balanced                    - General non-specialized tuned profile
- desktop                     - Optimize for the desktop use-case
- latency-performance         - Optimize for deterministic performance at the co
st of increased power consumption
- network-latency             - Optimize for deterministic performance at the co
st of increased power consumption, focused on low latency network performance
- network-throughput          - Optimize for streaming network throughput, gener
ally only necessary on older CPUs or 40G+ networks
- powersave                   - Optimize for low power consumption
- throughput-performance      - Broadly applicable tuning that provides excellen
t performance across a variety of common server workloads
- virtual-guest               - Optimize for running inside a virtual guest
- virtual-host                - Optimize for running KVM guests
Current active profile: virtual-guest
```

The output shows the nine predefined profiles. It also exhibits the current active profile.

5. List only the current active profile:

```
[user1@server1 ~]$ sudo tuned-adm active
Current active profile: virtual-guest
```

6. Switch to the *powersave* profile and confirm:

```
[user1@server1 ~]$ sudo tuned-adm profile powersave
[user1@server1 ~]$ sudo tuned-adm active
Current active profile: powersave
```

The active profile is now *powersave*.

7. Determine the recommended profile for *server1* and switch to it:

```
[user1@server1 ~]$ sudo tuned-adm recommend
virtual-guest
[user1@server1 ~]$ sudo tuned-adm profile virtual-guest
[user1@server1 ~]$ sudo tuned-adm active
Current active profile: virtual-guest
```

The first instance of the command shows the best recommended profile for *server1* based on its characteristics, the second command instance switched to the recommended profile, and the third instance confirmed the switching.

8. Turn off tuning:

```
[user1@server1 ~]$ sudo tuned-adm off
[user1@server1 ~]$ sudo tuned-adm active
No current active profile.
```

The service will not perform any tuning until it is reactivated.

9. Reactivate tuning and confirm:

```
[user1@server1 ~]$ sudo tuned-adm profile virtual-guest
[user1@server1 ~]$ sudo tuned-adm active
Current active profile: virtual-guest
```

The tuning is re-enabled and the *virtual-guest* profile is in effect. This concludes the exercise.

Chapter Summary

This chapter started with a discussion of systemd, the default service management and system initialization scheme used in RHEL. We explored key components of

systemd, its key directories, and examined unit and target configuration files. We utilized the lone systemd administration command to switch system operational states, identify and set default boot targets, and manage service start, stop, and status checking.

Next, we looked at the traditional system logging and newer systemd journaling services. Both mechanisms have similarities and differences in how they capture log data and where they direct it for storage and retrieval. We examined the system log configuration file and the configuration file that controls the log file rotation settings. The log subsystem proves valuable when records are needed for monitoring, troubleshooting, auditing, or reporting purpose.

Finally, we explored preconfigured tuning profiles and analyzed pros and cons associated with each one of them. We demonstrated how to determine a recommended profile for the system and how to set and activate it.

Check Your Understanding

1. The *systemd* command may be used to rebuild a new kernel. True or False?
2. Which command is used to manage system services?
3. Which configuration file must be modified to ensure journal log entries are stored persistently?
4. What is the PID of the *systemd* process?
5. What is a target in *systemd*?
6. You need to append a text string “Hello world” to the system log file. What would be the command to achieve this?
7. What is the recommended location to store custom log configuration files?
8. What would the command *systemctl list-dependencies crond* do?

9. Name the two directory paths where *systemd* unit files are stored.
10. What would you run to identify the recommended tuning profile for the system?
11. What would the command *systemctl get-default* do?
12. *systemd* starts multiple services concurrently during system boot. True or False?
13. What is the name of the boot log file?
14. Which *systemctl* subcommand is executed after a unit configuration file has been modified to apply the changes?
15. Which other logging service complements the *rsyslog* service?
16. A RHEL system is booted up. You want to view all messages that were generated during the boot process. Which log file would you look at?
17. What would the command *systemctl restart rsyslog* do?
18. What are the two common *systemd* targets production RHEL servers are typically configured to run at?
19. By default, log files are rotated automatically every week. True or False?

Answers to Check Your Understanding

1. False.
2. The *systemctl* command.
3. The *journald.conf* file under the */etc/systemd* directory.
4. The PID of the *systemd* process is 1.
5. A target is a collection of units.
6. The command to accomplish the desired result would be *logger -i “Hello world”*.
7. The recommended location to store custom log configuration files is */etc/rsyslog.d* directory.

8. The command provided will display all dependent units associated with the specified service.
9. The directory locations are */etc/systemd/system* and */usr/lib/systemd/system*.
10. The *tuned-adm recommend* command.
11. The command provided will reveal the current default boot target.
12. True.
13. The *boot.log* file in the */var/log* directory.
14. The *daemon-reload* subcommand.
15. The *systemd-journald* service.
16. The */var/log/boot.log* file.
17. The command provided will restart the *rsyslog* service.
18. The two common *systemd* boot targets are multi-user and graphical.
19. True.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 12-1: Modify Default Boot Target

As *user1* with *sudo* on *server1*, modify the default boot target from graphical to multi-user, and reboot the system to test it. Run the *systemctl* and *who* commands after the reboot for validation. Restore the default boot target back to graphical and reboot to verify. (Hint: System Initialization and Service Management).

Lab 12-2: Record Custom Alerts

As *user1* with *sudo* on *server1*, write the message “This is $LOGNAME adding this marker on $(date)” to */var/log/messages* file. Ensure that variable and command expansions work. Verify the entry in the file. (Hint: System Logging).

Lab 12-3: Apply Tuning Profile

As *user1* with *sudo* on *server1*, identify the current system tuning profile with the *tuned-adm* command. List all available profiles. List the recommended profile for *server1*. Apply the “balanced” profile and verify with *tuned-adm*. (Hint: System Tuning).

Chapter 13

Basic Storage Partitioning

This chapter describes the following major topics:

- Master Boot Record vs. GUID Partition Table
- Identify and understand disk partitions
- The concept of thin provisioning, and its benefits
- Create and delete partition on MBR disk
- Create and delete partition on GPT disk
- Overview of Virtual Data Optimizer and how it conserves storage
- Create and delete a Virtual Data Optimizer volume

RHCSA Objectives:

22. List, create, and delete partitions on MBR and GPT disks
32. Configure disk compression

Data is stored on disks that are logically divided into partitions. A partition can exist on a portion of a disk, on an entire disk, or it may span multiple disks. Each partition is accessed and managed independent of other partitions and may contain a file system or swap space. Partitioning information is stored at special disk locations that the system references at boot time. RHEL offers a number of tools for partition management. Partitions created with a combination of most of these tools can coexist on a single disk.

Thin provisioning is a powerful feature that guarantees an efficient use of storage space by allocating only what is needed and by storing data at adjacent locations. Many storage management solutions such as those we discuss later in this chapter and in the next incorporate thin provisioning technology in their core configuration.

Virtual Disk Optimizer is one of the newer storage management solutions incorporated in RHEL. It capitalizes on thin provisioning, de-duplication, and compression technologies to conserve storage space, improve data throughput, and save money.

This is the first of the three chapters that shed light on storage solutions. The next two chapters (Chapters 14 and 15) discuss advanced concepts and management tools.

Storage Management Overview

A disk in RHEL can be carved up into several partitions. This partition information is stored on the disk in a small region, which is read by the operating system at boot time. This region is referred to as the *Master Boot Record* (MBR) on the BIOS-based systems, and *GUID Partition Table* (GPT) on the UEFI-based systems. At system boot, the BIOS/UEFI scans all storage

devices, detects the presence of MBR/GPT areas, identifies the boot disks, loads the bootloader program in memory from the default boot disk, executes the boot code to read the partition table and identify the */boot* partition, loads the kernel in memory, and passes control over to it. Though MBR and GPT are designed for different PC firmware types, their job is essentially the same: to store disk partition information and the boot code.

Master Boot Record (MBR)

The MBR resides on the first sector of the boot disk. MBR was the preferred choice for saving partition table information on x86-based computers. However, with the arrival of bigger and larger hard drives, a newer firmware specification (UEFI) was introduced. MBR is still widely used, but its use is diminishing in favor of UEFI.

MBR allows the creation of three types of partition—*primary*, *extended*, and *logical*—on a single disk. Of these, only primary and logical can be used for data storage; the extended is a mere enclosure for holding the logical partitions and it is not meant for data storage. MBR supports the creation of up to four primary partitions numbered 1 through 4 at a time. In case additional partitions are required, one of the primary partitions must be deleted and replaced with an extended partition to be able to add logical partitions (up to 11) within that extended partition. Numbering for logical partitions begins at 5. MBR supports a maximum of 14 usable partitions (3 primary and 11 logical) on a single disk.

MBR cannot address storage space beyond 2TB. This is due to its 32-bit nature and its 512-byte disk sector size. The MBR is non-redundant; the record it contains is not replicated, resulting in an unbootable system in the event of corruption. If your disk is smaller than 2TB and you don’t intend

to build more than 14 usable partitions, you can use MBR without issues. For more information on MBR, refer to Chapter

11 “Boot Process, GRUB2, and the Linux Kernel”.

GUID Partition Table (GPT)

With the increasing use of disks larger than 2TB on x86 computers, a new 64-bit partitioning standard called *Globally Unique Identifiers* (GUID) *Partition Table* (GPT) was developed and integrated into the UEFI firmware. This new standard introduced plenty of enhancements, including the ability to construct up to 128 partitions (no concept of extended or logical partitions), utilize disks larger than 2TB, use 4KB sector size, and store a copy of the partition information before the end of the disk for redundancy.

Moreover, this standard allows a BIOS-based system to boot from a GPT disk using the bootloader program stored in a protective MBR at the first disk sector. In addition, the UEFI firmware also supports the secure boot feature, which only allows signed binaries to boot. For more information on UEFI and GPT, refer to Chapter 11 “Boot Process, GRUB2, and the Linux Kernel”.

Disk Partitions

The space on a storage device can be sliced into partitions. Care must be taken when adding a new partition to elude data corruption with overlapping an extant partition or wasting storage by leaving unused space between adjacent partitions. On *server1*, the disk that was allocated at the time of installation is recognized as *sda* (**s** for **S**ATA, SAS, or SCSI device) **d**isk **a**, with the first partition identified as *sda1* and the second partition as *sda2*. Any subsequent disks added to the system will be known as *sdb*, *sdc*, *sdd*, and so on, and will use 1, 2, 3, etc. for partition numbering.

RHEL offers a command called *lsblk* to list disk and partition information. The following graphic illustrates the current storage status on *server1*:

```
[user1@server1 ~]$ lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda             8:0    0   10G  0 disk
├─sda1          8:1    0    1G  0 part /boot
└─sda2          8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sr0            11:0    1  6.6G  0 rom  /mnt
```

It reveals the presence of one 10GB disk, *sda*, with two partitions: *sda1* and *sda2*. The first partition holds */boot*, and the second one is an LVM object encapsulating *root* and *swap* logical volumes within it. Both *sda1* and *sda2* partitions occupy the entire disk capacity. The *sr0* represents the ISO image mounted as an optical medium.

LVM is discussed at length in Chapter 14 “Advanced Storage Partitioning”.

There are additional tools such as *fdisk* and *parted* available that can be used to expose disk and partitioning information. Let’s run *fdisk* with -l and see what it reveals:

```
[user1@server1 ~]$ sudo fdisk -l
Disk /dev/sda: 10 GiB, 10737418240 bytes, 20971520 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
Disklabel type: dos
Disk identifier: 0xae0aa336

Device     Boot   Start      End  Sectors Size Id Type
/dev/sda1  *       2048  2099199  2097152   1G 83 Linux
/dev/sda2       2099200 20971519 18872320   9G 8e Linux LVM


Disk /dev/mapper/rhel-root: 8 GiB, 8585740288 bytes, 16769024 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 /dev/mapper/rhel-swap: 1 GiB, 1073741824 bytes, 2097152 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
```

The output depicts the size of *sda* in GBs, bytes, and sectors, the type of disk label (dos) the disk has, and the disk’s

geometry in the top block. The second block shows the two disk partitions: *sda1* as the bootable partition marked with an asterisk (*) and *sda2* as an LVM partition. It also exposes the starting and ending sector numbers, size in 1KB blocks, and type of each partition. The identifiers 83 and 8e are hexadecimal values for the partition types. The last two blocks are specific to the LVM logical volumes that exist within the *sda2* partition. A detailed coverage on LVM is provided in Chapter 14 “Advanced Storage Partitioning”.

Storage Management Tools

RHEL offers numerous tools and toolsets for storage management, and they include *parted*, *gdisk*, VDO, LVM, and Stratis. There are other native tools available in the OS, but their discussion is beyond the scope of this book. Partitions created with a combination of most of these tools and toolsets can coexist on the same disk. We look at *parted*, *gdisk*, and VDO in this chapter and LVM and Stratis in Chapter 14 “Advanced Storage Partitioning”.

parted is a simple tool that understands both MBR and GPT formats. *gdisk* is designed to support the GPT format only, and it may be used as a replacement of *parted*. VDO is a disk optimizer software that takes advantage of certain technologies to minimize the overall data footprint on storage devices. LVM is a feature-rich logical volume management solution that gives flexibility in storage management. Stratis capitalizes on thin provisioning to create volumes much larger in size than the underlying storage devices they are built upon.

Thin Provisioning

Thin provisioning technology allows for an economical allocation and utilization of storage space by moving arbitrary data blocks to contiguous locations, which results in empty block elimination. With thin provisioning support in VDO, LVM, and Stratis, you can create a *thin pool* of storage space and assign

volumes much larger storage space than the physical capacity of the pool.

Workloads begin consuming the actual allocated space for data writing. When a preset custom threshold (80%, for instance) on the actual consumption of the physical storage in the pool is reached, expand the pool dynamically by adding more physical storage to it. The volumes will automatically start exploiting the new space right away. The thin provisioning technique helps prevent spending more money upfront.

Adding Storage for Practice

This and the next two chapters have a considerable number of exercises that require block storage devices for practice. In Chapter 01 “Local Installation” under “Lab Infrastructure for Practice”, we mentioned that *server2* will have 4x250MB, 1x4GB, and 2x1GB virtual disks for storage exercises. We presume that *server2* was built as part of Lab 1-1 and it is now available for use.

Exercise 13-1: Add Required Storage to server2

This exercise will add the required storage disks to *server2* (*RHEL8-VM2*) using VirtualBox.

In this exercise, you will start VirtualBox and add 4x250MB, 1x4GB, and 2x1GB disks to *server2* in preparation for exercises in this chapter and Chapters 14 and 15.

1. Start VirtualBox on your Windows/Mac computer and highlight the *RHEL8-VM2* virtual machine that you created in Lab 1-1. See Figure 13-1.

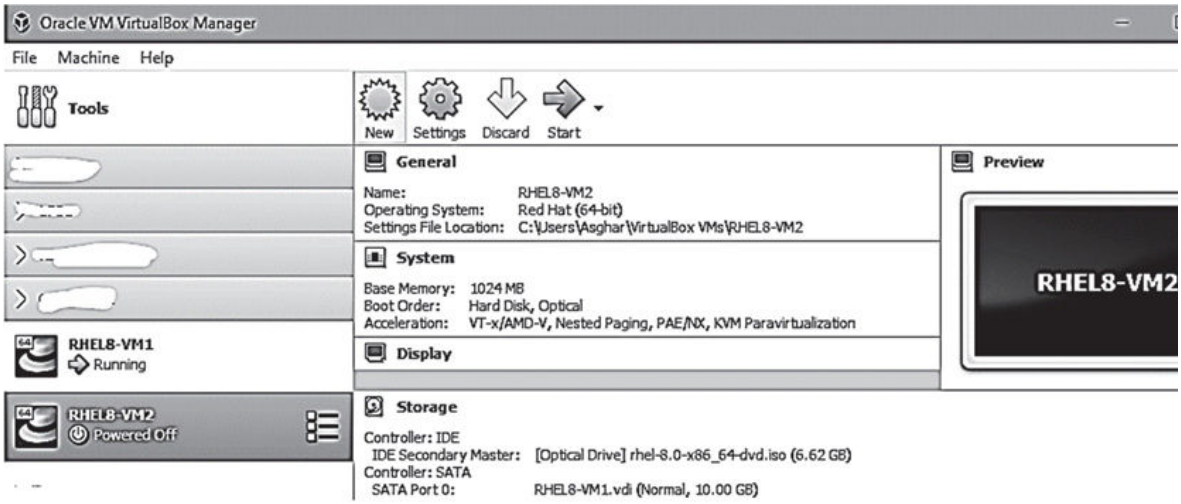


Figure 13-1 VirtualBox Interface

2. Click Settings at the top and then Storage on the window that pops up. Click on “Controller: SATA” to select it. Figure 13-2.

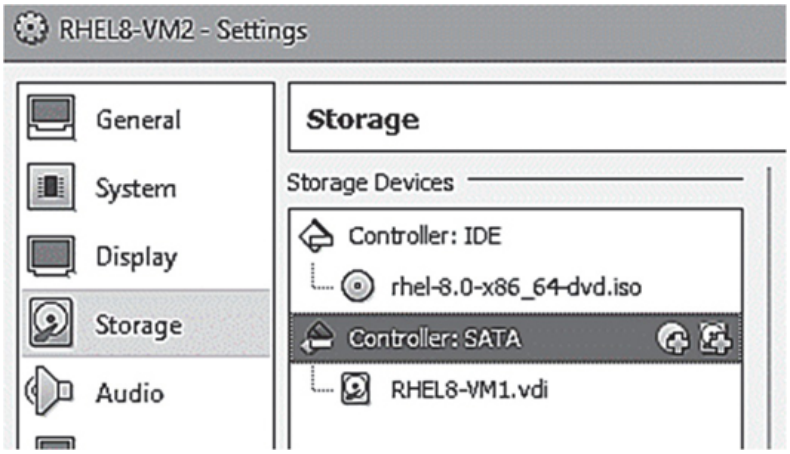


Figure 13-2 VirtualBox – Add Storage

3. Click on the right-side icon besides “Controller: SATA” to add a hard disk.
4. Follow this sequence to add a 250MB disk: Click “Create new disk”, “VDI (Virtualization Disk Image)”, “Dynamically allocated”, and adjust the size to 250MB. Assign the disk a unique name. Figure 13-3.

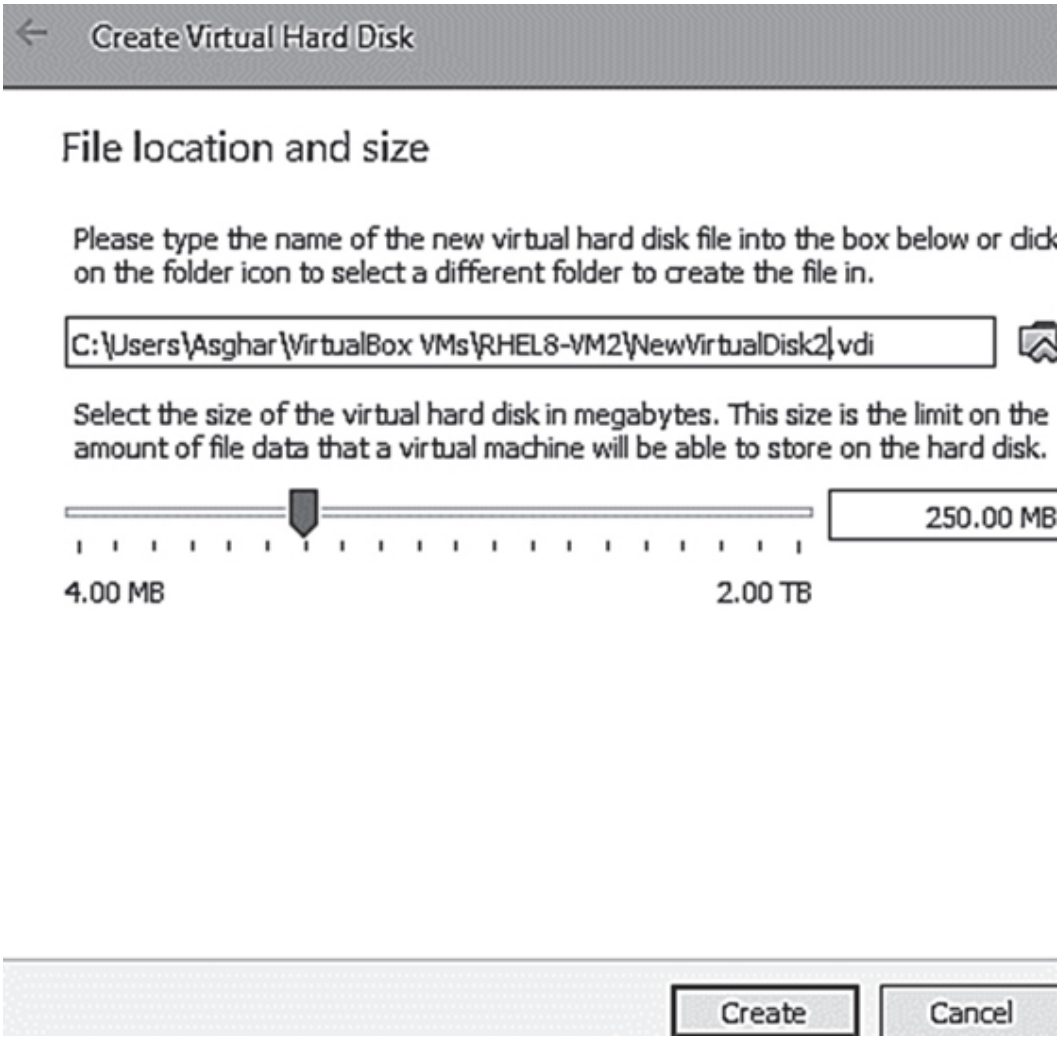


Figure 13-3 VirtualBox – Adjust Disk Name and Size

5. Click Create to create and attach the disk to the VM.
6. Repeat steps 3 through 5 three more times to add disks of the same size to the VM.
7. Repeat steps 3 through 5 one time to add a disk of size 4GB to the VM.
8. Repeat steps 3 through 5 two times to add two disks of size 1GB to the VM.
9. The final list of disks should look similar to what is shown in Figure 13-4 after the addition of all seven drives. Disk names may vary.

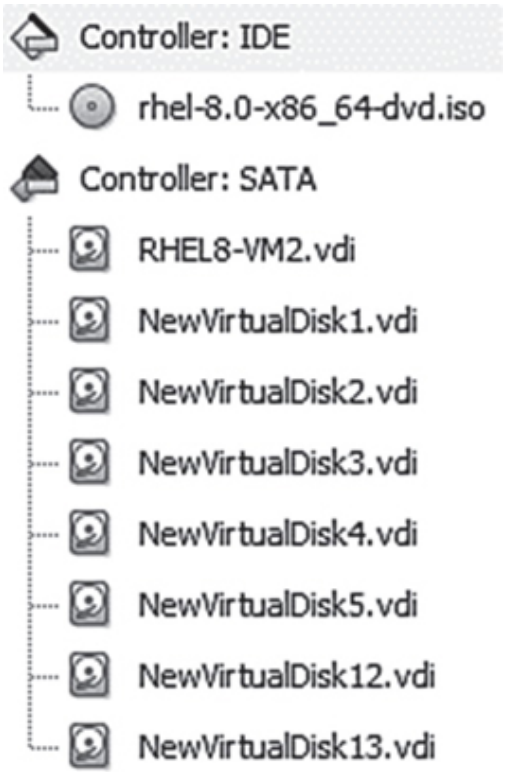


Figure 13-4 VirtualBox – 7 New Disks Added

10. Click OK to return to the main VirtualBox interface.
11. Power on *RHEL8-VM2* to boot RHEL 8 in it.
12. When the server is booted up, log on as *user1* and run the *lsblk* command to verify the new storage:

```
[user1@server2 ~]$ lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda             8:0    0   10G  0 disk
├─sda1          8:1    0    1G  0 part /boot
└─sda2          8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sdb             8:16   0  250M  0 disk
sdc             8:32   0  250M  0 disk
sdd             8:48   0  250M  0 disk
sde             8:64   0  250M  0 disk
sdf             8:80   0    4G  0 disk
sdg             8:96   0    1G  0 disk
sdh             8:112  0    1G  0 disk
sr0            11:0    1  6.6G  0 rom  /mnt
```

13. The seven new disks added to *server2* are 250MB (*sdb*, *sdc*, *sdd*, and *sde*), 4GB (*sdf*), and 1GB (*sdg* and *sdh*).

This concludes the exercise for storage addition to *server2*.

MBR Storage Management with parted

parted (*partition editor*) is a popular tool in RHEL that can be used to partition disks. This program may be run interactively or directly from the command prompt. It understands and supports both MBR and GPT schemes, and can be used to create up to 128 partitions on a single GPT disk. *parted* provides an abundance of subcommands to perform disk management operations such as viewing, labeling, adding, naming, and deleting partitions. Table 13-1 describes these subcommands in that sequence.

Subcommand	Description
print	Displays the partition table that includes disk geometry and partition number, start and end, size, type, file system type, and relevant flags.
mklabel	Applies a label to the disk. Common labels are gpt and msdos.
mkpart	Makes a new partition
name	Assigns a name to a partition
rm	Removes the specified partition

Table 13-1 Common parted Subcommands

For the basic partition creation and deletion operations, Exercises 13-2 and 13-3 will show the use of this tool by directly invoking it from the command prompt. You will use the */dev/sdb* disk for these exercises. After making a partition, use the *print* subcommand to ensure you created what you wanted. The */proc/partitions* file is also updated to reflect the results of partition management operations.

Exercise 13-2: Create an MBR Partition

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will assign partition type “msdos” to */dev/sdb* for using it as an MBR disk. You will create and confirm a 100MB primary partition on the disk.

1. Execute *parted* on */dev/sdb* to view the current partition information:

```
[user1@server2 ~]$ sudo parted /dev/sdb print
Error: /dev/sdb: unrecognised disk label
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 262MB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
```

There is an error on line 1 of the output, indicating an unrecognized label. This disk must be labeled before it can be partitioned.

2. Assign disk label “msdos” to the disk with *mklabel*. This operation is performed only once on a disk.

```
[user1@server2 ~]$ sudo parted /dev/sdb mklabel msdos
Information: You may need to update /etc/fstab.

[user1@server2 ~]$ sudo parted /dev/sdb print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 262MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End  Size  Type  File system  Flags
```

The *print* subcommand confirms the successful application of the label.

To use the GPT partition table type, run “**sudo parted /dev/sdb mklabel gpt**” instead.

3. Create a 100MB primary partition starting at 1MB (beginning of the disk) using *mkpart*:

```
[user1@server2 ~]$ sudo parted /dev/sdb mkpart primary 1 101m
```

4. Verify the new partition with *print*:

```
[user1@server2 ~]$ sudo parted /dev/sdb print
Number  Start   End    Size    Type     File system  Flags
 1      1049kB  101MB  99.6MB  primary
```

Partition numbering begins at 1 by default.

5. Confirm the new partition with the *lsblk* command:

```
[user1@server2 ~]$ lsblk /dev/sdb
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb      8:16   0  250M  0 disk
└─sdb1   8:17   0   95M  0 part
```

The device file for the first partition on the *sdb* disk is *sdb1* as identified on the bottom line. The partition size is 95MB.

Different tools will have variance in reporting partition sizes. You should ignore minor differences.

6. Check the */proc/partitions* file also:

```
[user1@server2 ~]$ cat /proc/partitions | grep sdb
   8       16     256000 sdb
   8       17      97280 sdb1
```

The virtual file is also updated with the new partition information. This completes the steps for creating and verifying an MBR partition using the *parted* command.

Exercise 13-3: Delete an MBR Partition

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will delete the *sdb1* partition that was created in Exercise 13-2 and confirm the deletion.

1. Execute *parted* on */dev/sdb* with the *rm* subcommand to remove partition number 1:

```
[user1@server2 ~]$ sudo parted /dev/sdb rm 1
```

2. Confirm the partition deletion with *print*:

```
........
Number  Start  End  Size  Type  File system  Flags
```

The partition no longer exists.

3. Check the */proc/partitions* file:

```
[user1@server2 ~]$ cat /proc/partitions | grep sdb
   8       16     256000 sdb
```

The virtual file has the partition entry deleted as well. You can also run the *lsblk* command for further verification. The partition has been removed successfully.

We will recreate partitions for use in LVM in Chapter 14 and then again in Chapter 15 to construct file system and swap structures.

GPT Storage Management with gdisk

The *gdisk* (*GPT disk*) utility partitions disks using the GPT format. This text-based, menu-driven program can show, add, verify, modify, and delete partitions among other operations. *gdisk* can create up to 128 partitions on a single disk on systems with UEFI firmware.

The main interface of *gdisk* can be invoked by specifying a disk device name such as */dev/sdc* with the command. Type *help* or *?* (question mark) at the prompt to view available subcommands.

```
[user1@server2 ~]$ sudo gdisk /dev/sdc
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.

Command (? for help): help
b       back up GPT data to a file
c       change a partition's name
d       delete a partition
i       show detailed information on a partition
l       list known partition types
n       add a new partition
o       create a new empty GUID partition table (GPT)
p       print the partition table
q       quit without saving changes
r       recovery and transformation options (experts only)
s       sort partitions
t       change a partition's type code
v       verify disk
w       write table to disk and exit
x       extra functionality (experts only)
?       print this menu

Command (? for help): _
```

The output illustrates that there is no partition table defined on the disk at the moment. There are several subcommands in the main menu followed by a short description. Refer to the screenshot above for a list of subcommands.

Exercise 13-4: Create a GPT Partition

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will assign partition type “gpt” to */dev/sdc* for using it as a GPT disk. You will create and confirm a 200MB partition on the disk.

1. Execute *gdisk* on */dev/sdc* to view the current partition information:

```
[user1@server2 ~]$ sudo gdisk /dev/sdc
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.

Command (? for help): _
```

The disk currently does not have any partition table on it.

2. Assign “gpt” as the partition table type to the disk using the *o* subcommand. Enter “y” for confirmation to proceed. This operation is performed only once on a disk.

```
Command (? for help): o
This option deletes all partitions and creates a new protective MBR.
Proceed? (Y/N): Y
```

3. Run the *p* subcommand to view disk information and confirm the GUID partition table creation:

```
Command (? for help): p
Disk /dev/sdc: 512000 sectors, 250.0 MiB
Model: VBOX HARDDISK
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): CD1525A6-F747-4495-85F5-3C242603FB9A
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 511966
Partitions will be aligned on 2048-sector boundaries
Total free space is 511933 sectors (250.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
```

The output returns the assigned GUID and states that the partition table can hold up to 128 partition entries.

4. Create the first partition of size 200MB starting at the default sector with default type “Linux filesystem” using the *n* subcommand:

```
Command (? for help): n
Partition number (1-128, default 1):
First sector (34-511966, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-511966, default = 511966) or {+-}size{KMGTP}: +200M
Current type is 'Linux filesystem'
Hex code or GUID (L to show codes, Enter = 8300):
Changed type of partition to 'Linux filesystem'
```

5. Verify the new partition with *p*:

   Command (? For help): **p**

```
.......
Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048          411647   200.0 MiB   8300  Linux filesystem
```

6. Run *w* to write the partition information to the partition table and exit out of the interface. Enter “y” to confirm when prompted.

```
Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to /dev/sdc.
The operation has completed successfully.
```

You may need to run the partprobe command after exiting the gdisk utility to update the kernel of the changes.

7. Verify the new partition by issuing either of the following at the command prompt:

```
[user1@server2 ~]$ grep sdc /proc/partitions
   8       32     256000 sdc
   8       33     204800 sdc1
[user1@server2 ~]$
[user1@server2 ~]$ lsblk /dev/sdc
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdc      8:32   0  250M  0 disk
└─sdc1   8:33   0  200M  0 part
```

The device file for the first partition on the *sdc* disk is *sdc1* and it is 200MB in size as reported in the above outputs. This completes the steps for creating and verifying a GPT partition using the *gdisk* command.

Exercise 13-5: Delete a GPT Partition

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will delete the *sdc1* partition that was created in Exercise 13-4 and confirm the removal.

1. Execute *gdisk* on */dev/sdc* and run *d1* at the utility's prompt to delete partition number 1:

```
Command (? for help): d1
Using 1
```

2. Confirm the partition deletion with *p*:

```
........
Number  Start (sector)    End (sector)  Size       Code  Name
```

The partition no longer exists.

3. Write the updated partition information to the disk with *w* and quit *gdisk*:

```
Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to /dev/sdc.
The operation has completed successfully.
```

4. Verify the partition deletion by issuing either of the following at the command prompt:

```
[user1@server2 ~]$ grep sdc /proc/partitions
   8       32     256000 sdc
[user1@server2 ~]$
[user1@server2 ~]$ lsblk /dev/sdc
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdc    8:32   0  250M  0 disk
```

Both commands confirm the successful partition removal.

Storage Optimization with Virtual Data Optimizer (VDO)

One of the new features recently introduced in RHEL is a device driver layer that sits between the operating system kernel and the physical storage devices. The goals are to conserve disk space, improve data throughput, and save on storage cost. This feature is referred to as *Virtual Data Optimizer* (VDO). VDO employs thin provisioning, de-duplication, and compression technologies to help realize the goals.

How VDO Conserves Storage Space

VDO makes use of the thin provisioning technology to identify and eliminate empty (zero-byte) data blocks. This is referred to as *zero-block elimination*. VDO removes randomization of data blocks by moving in-use data blocks to contiguous locations on the storage device. This is the initial stage in the process.

Next, VDO keeps an eye on data being written to the disk. If it detects that the new data is an identical copy of some existing data, it makes an internal note of it but does not actually write the redundant data to the disk. VDO uses the technique called *de-duplication* to this end. This technique is implemented in RHEL with the inclusion of a kernel module called *UDS* (*Universal De-duplication Service*). This is the second stage in the process.

In the third and final stage, VDO calls upon another kernel module called *kvdo*, which compresses the residual data blocks and consolidates them on a lower number of blocks. This results in a further drop in storage space utilization.

VDO runs in the background and processes inbound data through the three stages on VDO-enabled volumes. VDO is not a CPU- or memory-intensive process; it consumes a low amount of system resources.

Creating and Managing VDO Volumes

The concept of VDO volumes is similar to that of disk partitions, which you created in Exercises 13-1 and 13-3 using *parted* and

gdisk. VDO volumes can be initialized for use just like disk partitions, or they can be used as LVM physical volumes.

VDO offers a set of commands to create, manage, and monitor volumes. Of these *vdo* and *vdostats* commands are discussed and used in this section. The *vdo* command is used to create and perform essential operations on VDO volumes, and the *vdostats* command is employed to monitor usage statistics of the underlying physical storage device.

Table 13-2 summarizes the subcommands available with *vdo*.

Subcommand	Description
create	Adds a new VDO volume on the specified block device
status	Returns the status and attributes of VDO volumes
list	Lists the names of all started VDO volumes
start	Starts a VDO volume
stop	Stops a VDO volume

Table 13-2 vdo Subcommands

The *vdostats* command has a couple of interesting options that you will use shortly.

Exercise 13-6: Install Software and Activate VDO

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will install the VDO software packages, start the VDO service, and mark it for autostart on subsequent system reboots.

1. Install packages *vdo* and *kmod-kvdo*:

```
[user1@server2 ~]$ sudo dnf install vdo kmod-kvdo -y
```

2. Start the service and enable it to start automatically on future system reboots:

```
[user1@server2 ~]$ sudo systemctl --now enable vdo
```

3. Check the operational status of the service:

```
[user1@server2 ~]$ sudo systemctl status vdo
● vdo.service - VDO volume services
   Loaded: loaded (/usr/lib/systemd/system/vdo.service; enabled; vendor preset:
   Active: active (exited) since Tue 2019-11-05 11:35:38 EST; 39s ago
  Process: 8898 ExecStart=/usr/bin/vdo start --all --confFile /etc/vdoconf.yml
 Main PID: 8898 (code=exited, status=0/SUCCESS)

Nov 05 11:35:37 server2.example.com systemd[1]: Starting VDO volume services...
```

The relevant packages for VDO are installed, and the VDO service is started and activated. This concludes the exercise.

Exercise 13-7: Create a VDO Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a volume called *vdo-vol1* of logical size 16GB on */dev/sdf* disk (the actual size of */dev/sdf* is 4GB). You will list the volume and display its status information. You will also show the activation status of the compression and de-duplication features.

1. Create volume *vdo-vol1* (--name) on */dev/sdf* (--device) of logical size 16GB (--vdoLogicalSize) with a slab size of 128MB (--vdoSlabSize):

```
[user1@server2 ~]$ sudo vdo create --name vdo-vol1 --device /dev/sdf --vdoLogica
lSize 16G --vdoSlabSize 128
Creating VDO vdo-vol1
Starting VDO vdo-vol1
Starting compression on VDO vdo-vol1
VDO instance 1 volume is ready at /dev/mapper/vdo-vol1
```

If the logical size is not specified, the VDO volume will have the same size as the underlying disk (/dev/sdf in this case).

The slab size is the size of the increment by which VDO volumes grow. This value must be a power of two between 128MB and 32GB; the default is 2GB. The default unit of size specification is MB.

2. List the new volume using the *vdo* and *lsblk* commands:

```
[user1@server2 ~]$ sudo vdo list
vdo-vol1
[user1@server2 ~]$ lsblk /dev/sdf
NAME       MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdf          8:80   0   4G  0 disk
└─vdo-vol1 253:2    0  16G  0 vdo
```

As indicated, the major number for the VDO volume is 253, which is associated with the device mapper kernel driver. The output also shows the logical volume size (16GB) and type (vdo). It also depicts the disk (*sdf*) that houses the volume, along with its actual size (4GB).

3. Display the usage status of the volume:

```
[user1@server2 ~]$ sudo vdostats --hu
Device                    Size      Used Available Use% Space saving%
/dev/mapper/vdo-vol1      4.0G      2.8G      1.2G  68%            0%
```

The size of the actual disk is 4GB. Due to thin provisioning, the system allowed you to create the VDO volume much larger in size (4 times) than the physical disk capacity.

4. Show detailed statistics for the volume including configuration information:

```
[user1@server2 ~]$ sudo vdostats --verbose
/dev/mapper/vdo-vol1 :
  version                             : 30
  release version                     : 133524
  data blocks used                    : 0
  overhead blocks used                : 723226
  logical blocks used                 : 0
  physical blocks                     : 1048576
  logical blocks                      : 4194304
  1K-blocks                           : 4194304
  1K-blocks used                      : 2892904
  1K-blocks available                 : 1301400
  used percent                        : 68
........
```

The output will expose over one hundred different settings for the volume.

5. Display detailed statistics for the volume including configuration information:

```
[user1@server2 ~]$ sudo vdo status --name vdo-vol1
VDO status:
  Date: '2019-10-24 14:07:35-04:00'
  Node: server2.example.com
Kernel module:
  Loaded: true
  Name: kvdo
  Version information:
    kvdo version: 6.2.0.293
Configuration:
  File: /etc/vdoconf.yml
  Last modified: '2019-10-24 14:03:13'
VDOs:
  vdo-vol1:
    Acknowledgement threads: 1
........
```

The status includes volume, kernel module, and configuration information. It also provides a detailed look at volume-specific elements.

6. Show the activation status of the compression and de-duplication features:

```
[user1@server2 ~]$ sudo vdo status --name vdo-vol1 | grep -i compression
    Compression: enabled
[user1@server2 ~]$ sudo vdo status --name vdo-vol1 | grep -i deduplication
    Deduplication: enabled
```

Both compression and de-duplication features are activated by default on new VDO volumes. This concludes the exercise.

Exercise 13-8: Delete a VDO Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will delete the *vdo-vol1* volume that was created in Exercise 13-7 and confirm the removal.

1. Verify that the volume still exists with the *vdo* and *lsblk* commands:

```
[user1@server2 ~]$ sudo vdo list
vdo-vol1
[user1@server2 ~]$ lsblk /dev/sdf
NAME       MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdf          8:80   0   4G  0 disk
└─vdo-vol1 253:2    0  16G  0 vdo
```

2. Specify the name (--name) of the volume with the *vdo* command to delete:

```
[user1@server2 ~]$ sudo vdo remove --name vdo-vol1
Removing VDO vdo-vol1
Stopping VDO vdo-vol1
```

3. Confirm the removal with the *vdo* and *lsblk* commands:

```
[user1@server2 ~]$ sudo vdo list

[user1@server2 ~]$ lsblk /dev/sdf
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdf    8:80   0   4G  0 disk
```

The volume has been deleted successfully as reported in the above outputs.

You will recreate VDO volumes in Chapter 15 “Local File Systems and Swap” for use as file systems and swap areas.

Chapter Summary

This chapter started with an overview of how and where disk partitioning information is stored. It presented a comparison between the two common schemes and explained which one to use and in which situation. A little later, we touched briefly on the common storage management solutions available in RHEL.

We examined the concept of thin provisioning and realized the benefits associated with this technology.

Next, we carved up available disk devices using both MBR and GPT partitioning schemes. We demonstrated the partition creation, display, and delete operations by running one command directly at the command prompt and launching the other in interactive mode.

The last topic of the chapter focused on a storage management solution that was introduced in RHEL not too long ago. This solution exploits the underlying thin provisioning, de-duplication, and compression technologies to save cost, ensure efficient use of storage space, and improve data throughput. We performed a couple of exercises in the end to demonstrate the creation and deletion of volumes using this solution.

Check Your Understanding

1. What is missing in the command *vdo create --name vdo1 --vdoLogicalSize 16GB --vdoSlabSize 128MB*?
2. What is the maximum number of usable partitions that can be created on a GPT disk?
3. Which kernel module is responsible for data block compression in VDO volumes?
4. What are the three techniques VDO volumes employ for storage conservation?
5. What would the command *parted /dev/sdc mkpart pri 1 200m* do?
6. Where is the partition table information stored on BIOS-based systems?
7. What is the name of the technology that VDO employs to remove randomization of data blocks?
8. What would *sdd3* represent?
9. Which command can be used to view the usage statistics of VDO volumes?
10. Thin provisioning technology allows us to create logical volumes of sizes larger than the actual physical storage size. True or False?
11. You have an unused VDO volume called *vdo1* on */dev/sdf* disk and you try to delete it. You run *vdo remove --device /dev/sdf*, but the removal fails. What are you doing wrong?
12. What is the maximum number of usable partitions that can be created on an MBR disk?
13. What would the command *systemctl --now enable vdo* do?

14. The *gdisk* utility can be used to store partition information in MBR format. True or False?
15. What would the command *parted /dev/sdd mklabel msdos* do?
16. VDO is a memory-intensive storage optimization solution and should not be used. True or False?
17. Which file in the */proc* file system stores the in-memory partitioning information?
18. You have created a VDO volume and you want to check whether compression is enabled. Which command would you use?
19. De-duplication is the process of zero-block elimination. True or False?

Answers to Check Your Understanding

1. The storage device (--device) name is missing from the command provided.
2. 128.
3. The *kvdo* module is responsible for compressing data blocks in VDO volumes.
4. VDO volumes use thin provisioning, de-duplication, and compression techniques for storage conservation.
5. The command provided will create a primary partition of size 200MB on the *sdc* disk starting at the beginning of the disk.
6. The partition table information is stored on the Master Boot Record.
7. VDO employs thin provisioning technology to remove data block randomization.
8. *sdd3* represents the third partition on the fourth disk.
9. The *vdostats* command can be used to view VDO volume usage statistics.
10. True.
11. The correct command would be *vdo remove --name vdo1*.

12. 14.
13. The command provided will start the VDO service and enable it to autostart on system reboots.
14. False. The *gdisk* tool is only for GPT type tables.
15. The command provided will apply msdos label to the *sdd* disk.
16. False. VDO uses low memory and other compute resources.
17. The *partitions* file.
18. You can issue the *vdo status* command and pipe the output to *grep* for the pattern “compression”.
19. False. De-duplication is the process of removing blocks of identical data.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Add more storage to *server2* if required.

Lab 13-1: Create and Remove Partitions with parted

As *user1* with *sudo* on *server2*, create a 100MB primary partition on one of the available 250MB disks (*lsblk*) by invoking the *parted* utility directly at the command prompt. Apply label “msdos” if the disk is new. Create another 100MB partition by running *parted* interactively while ensuring that the second partition won’t overlap the first. Verify the label and the partitions. Remove both partitions from the command prompt. (Hint: MBR Storage Management with parted).

Lab 13-2: Create and Remove Partitions with gdisk

As *user1* with *sudo* on *server2*, create two 80MB partitions on one of the 250MB disks (*lsblk*) using the *gdisk* utility. Make sure the partitions won't overlap. Verify the partitions. You may delete the partitions if you want. (Hint: GPT Storage Management with gdisk).

Lab 13-3: Create and Delete VDO Volumes

As *user1* with *sudo* on *server2*, check to see if VDO software is installed and the VDO service is enabled and started. Identify the 4GB disk with the *lsblk* command, and make sure that it is not in use. Create a volume *testvdo* with a logical size 16GB on the 4GB disk using the *vdo* command. Select an appropriate slab size for the volume. Verify the volume creation with the *vdo*, *lsblk*, and *vdostats* commands. (Hint: Storage Optimization with Virtual Data Optimizer).

Lab 13-4: Disable and Enable VDO Volume Features

As *user1* with *sudo* on *server2*, use the *vdostats* command to check whether compression and de-duplication are enabled for the volume created in Lab 13-3. Use the *vdo* command with disableCompression and disableDeduplication subcommands to disable compression and de-duplication, and verify with *vdostats*. Reactivate both features and confirm activation. You may delete the volume if you want. (Hint: Storage Optimization with Virtual Data Optimizer).

Chapter 14

Advanced Storage Partitioning

This chapter describes the following major topics:

- Describe Logical Volume Manager and its components
- Understand various Logical Volume Manager management operations
- Know Logical Volume Manager administration commands
- Create and confirm physical volumes, volume groups, and logical volumes
- Rename, reduce, extend, and remove logical volumes
- Extend, reduce, and remove volume groups
- Remove physical volumes
- Overview of Stratis (a volume-managing file system solution in RHEL) service and how it

works

- Understand various Stratis management operations and the command
- Create, confirm, expand, rename, and destroy pools and file systems

RHCSA Objectives:

23. Create and remove physical volumes
24. Assign physical volumes to volume groups
25. Create and delete logical volumes
27. Add new partitions and logical volumes, and swap to a system non-destructively (the swap portion of this objective is covered in Chapter 15)
30. Extend existing logical volumes (additional coverage on this objective is available in Chapter 15)
33. Manage layered volumes

This chapter is the second of the three chapters (the other two being Chapter 13 (previous) and Chapter 15 (next)) that expounds upon storage management concepts and solutions available in RHEL. We discussed partitioning and thinly-provisioned volumes in the previous chapter. This chapter presents a detailed coverage on Logical Volume Manager solution. LVM sets up an abstraction layer between the operating system and the storage hardware. It utilizes virtual objects for storage pooling and allocation, and offers a whole slew of management commands, each of which carries out a particular operation.

The other advanced storage management solution discussed in this chapter is a volume-managing file system that capitalizes on the proven features of LVM and the kernel device driver software. This solution dynamically adjusts the size of the underlying volume, eliminating the need for manual expansion.

Logical Volume Manager (LVM)

The *Logical Volume Manager* (LVM) solution is widely used for managing block storage in Linux. LVM provides an abstraction layer between the physical storage and the file system, enabling the file system to be resized, span across multiple physical disks, use arbitrary disk space, etc. LVM accumulates spaces taken from partitions or entire disks (called *Physical Volumes*) to form a logical container (called *Volume Group*), which is then divided into logical partitions (called *Logical Volumes*). The other key benefits of LVM include online resizing of volume groups and logical volumes, online data migration between logical volumes and between physical volumes, user-defined naming for volume groups and logical volumes, mirroring and striping across multiple physical disks, and

snapshotting of logical volumes. Figure 14-1 depicts the LVM components.

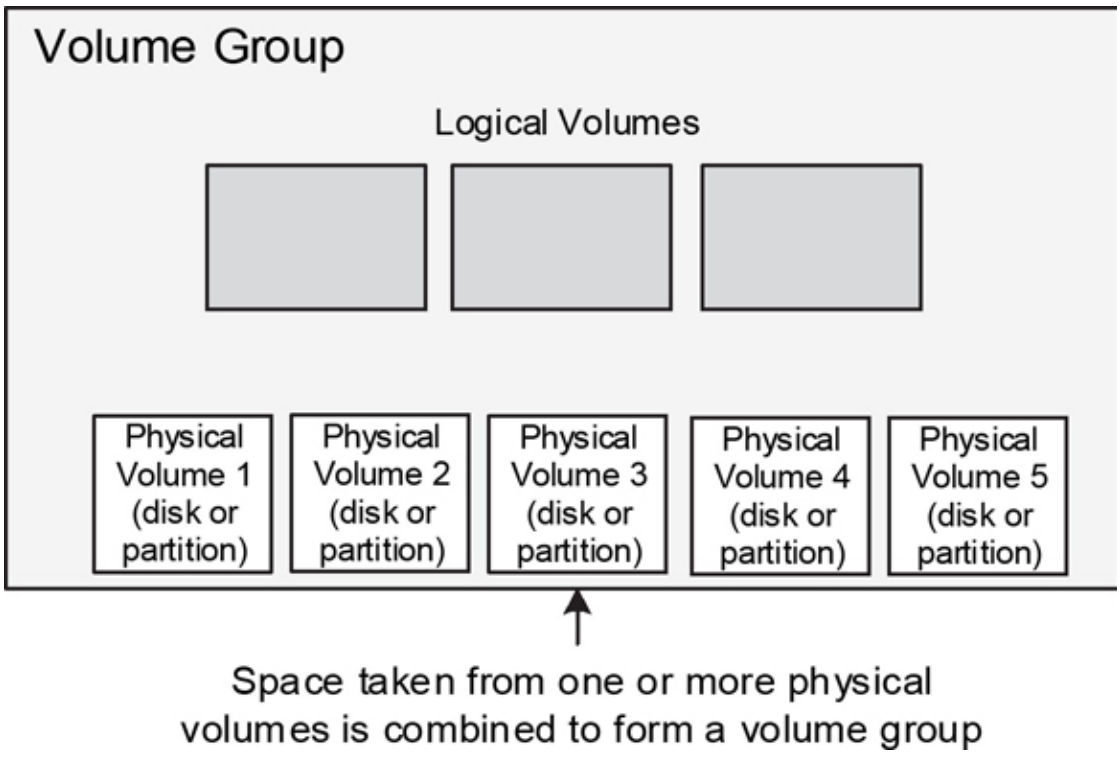


Figure 14-1 LVM Structure

As noted above, the LVM structure is made up of three key objects called physical volume, volume group, and logical volume. These objects are further virtually broken down into *Physical Extents* (PEs) and *Logical Extents* (LEs). The LVM components are explained in the following subsections.

Physical Volume

A *Physical Volume* (PV) is created when a block storage device such as a partition or an entire disk is initialized and brought under LVM control. This process constructs LVM data structures on the device, including a label on the second sector and metadata shortly thereafter. The label includes the UUID, size, and pointers to the locations of data and metadata areas. Given the criticality of metadata, LVM stores a copy of it at the

end of the physical volume as well. The rest of the device space is available for use.

You can use an LVM command called *pvs* (*physical volume scan* or *summary*) to scan and list available physical volumes on *server2*:

```
[user1@server2 ~]$ sudo pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sda2  rhel lvm2 a--  <9.00g    0
```

The output shows one physical volume (PV) */dev/sda2* of size 9GB in *rhel* volume group (VG). Additional information displays the metadata format (Fmt) used, status of the physical volume under the Attr column (a for allocatable), and the amount of free space available on the physical volume (PFree).

Try running this command again with the -v flag to view more information about the physical volume.

Volume Group

A *Volume Group* (VG) is created when at least one physical volume is added to it. The space from all physical volumes in a volume group is aggregated to form one large pool of storage, which is then used to build logical volumes. The physical volumes added to a volume group may be of varying sizes. LVM writes volume group metadata on each physical volume that is added to it. The volume group metadata contains its name, date and time of creation, how it was created, the extent size used, a list of physical and logical volumes, a mapping of physical and logical extents, etc. A volume group can have a custom name assigned to it at the time of its creation. For example, it may be called *vg01*, *vgora*, or *vgweb* that identifies the type of information it is constructed to store. A copy of the volume group metadata is stored and maintained at two distinct locations on each physical volume within the volume group.

You can use an LVM command called *vgs* (*volume group scan* or *summary*) to scan and list available volume groups on *server2*:

```
[user1@server2 ~]$ sudo vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  rhel   1   2   0 wz--n- <9.00g    0
```

The output shows one volume group (VG) *rhel* on *server2* containing one physical volume (#PV). Additional information displays the number of logical volumes (#LV) and snapshots (#SN) in the volume group, status of the volume group under the Attr column (w for writeable, z for resizable, and n for normal), size of the volume group (VSize), and the amount of free space available in the volume group (VFree).

Try running this command again with the -v flag to view more information about the volume group.

Physical Extent

A physical volume is divided into several smaller logical pieces when it is added to a volume group. These logical pieces are known as *Physical Extents* (PE). An extent is the smallest allocatable unit of space in LVM. At the time of volume group creation, you can either define the size of the PE or leave it to the default value of 4MB. This implies that a 20GB physical volume would have approximately 5,000 PEs. Any physical volumes added to this volume group thereafter will use the same PE size.

You can use an LVM command called *vgdisplay* (*volume group display*) on *server2* and *grep* for ‘PE Size’ to view the PE size used in the *rhel* volume group:

```
[user1@server2 ~]$ sudo vgdisplay rhel | grep 'PE Size'
  PE Size               4.00 MiB
```

The output reveals the PE size used for the *rhel* VG.

Logical Volume

A volume group consists of a pool of storage taken from one or more physical volumes. This volume group space is used to create one or more *Logical Volumes* (LVs). A logical volume can be created or weeded out online, expanded or shrunk online, and can use space taken from one or multiple physical volumes inside the volume group.

The default naming convention used for logical volumes is *lvol0, lvol1, lvol2*, and so on; however, you may assign custom names to them. For example, a logical volume may be called *system*, *undo*, or *webdata1* so as to establish the type of information it is constructed to store.

You can use an LVM command called *lvs* (*logical volume scan* or *summary*) to scan and list available logical volumes on *server2*:

```
[user1@server2 ~]$ sudo lvs
  LV   VG   Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  root rhel -wi-ao---- <8.00g
  swap rhel -wi-ao----  1.00g
```

The output shows two logical volumes *root* and *swap* in *rhel* volume group. Additional information displays the status of the logical volumes under the Attr column (w for writeable, i for inherited allocation policy, a for active, and o for open) and their sizes.

Try running this command again with the -v flag to view more information about the logical volumes.

Logical Extent

A logical volume is made up of *Logical Extents* (LE). Logical extents point to physical extents, and they may be random or contiguous. The larger a logical volume is, the more logical extents it will have. Logical extents are a set of physical extents allocated to the logical volume.

The PE and LE sizes are normally kept the same within a volume group; however, a logical extent can be smaller or larger than a physical extent. The default LE size is 4MB, which corresponds to the default PE size.

You can use an LVM command called *lvdisplay* (*logical volume display*) on *server2* to view information about the *root* logical volume in the *rhel* volume group.

```
[user1@server2 ~]$ sudo lvdisplay /dev/rhel/root
  --- Logical volume ---
  LV Path                /dev/rhel/root
  LV Name                root
  VG Name                rhel
  LV UUID                M6eGU8-wc5j-iJIC-rdUX-Uitm-NjjO-ZpyOIR
  LV Write Access        read/write
  LV Creation host, time server1.example.com, 2019-08-22 15:03:12 -0400
  LV Status              available
  # open                 1
  LV Size                <8.00 GiB
  Current LE             2047
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

The output does not disclose the LE size; however, you can convert the LV size in MBs (8,000) and then divide the result by the Current LE count (2,047) to get the LE size (which comes close to 4MB).

LVM Operations and Commands

The LVM toolset offers a multitude of administrative commands to carry out various disk and volume management operations. These operations include creating and removing a physical volume, volume group, and logical volume; extending and reducing a volume group and logical volume; renaming a volume group and logical volume; and listing and displaying physical volume, volume group, and logical volume information.

Table 14-1 summarizes the common LVM tasks and the commands that are employed to accomplish them.

Command	Description
Create and Remove Operations	
pvcreate/pvremove	Initializes/uninitializes a disk or partition for LVM use
vgcreate/vgremove	Creates/removes a volume group
lvcreate/lvremove	Creates/removes a logical volume
Extend and Reduce Operations	
vgextend/vgreduce	Adds/removes a physical volume to/from a volume group
lvextend/lvreduce	Extends/reduces the size of a logical volume
lvresize	Resizes a logical volume. With the -r option, this command calls the resize2fs command and resizes the underlying file system as well. Applies to Ext2/Ext3/Ext4 file system types only.
Rename Operations	
vgrename	Renames a volume group
lvrename	Renames a logical volume
List and Display Operations	
pvs/pvdisplay	Lists/displays physical volume information
vgs/vgdisplay	Lists/displays volume group information
lvs/lvdisplay	Lists/displays logical volume information

Table 14-1 Common LVM Operations and Commands

All the tools accept the -v switch to support verbosity. Refer to the manual pages of the commands for usage and additional details.

As noted earlier, there are seven disks available on *server2* for practice. Issue the *lsblk* command to confirm:

```
sdb             8:16   0  250M  0 disk
sdc             8:32   0  250M  0 disk
sdd             8:48   0  250M  0 disk
sde             8:64   0  250M  0 disk
sdf             8:80   0    4G  0 disk
sdg             8:96   0    1G  0 disk
sdh             8:112  0    1G  0 disk
```

You will use the *sdd* and *sde* disks for LVM activities in the following exercises.

Exercise 14-1: Create a Physical Volume and Volume Group

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will initialize one partition *sdd1* (90MB) and one disk *sde* (250MB) for use in LVM. You will create a volume group called *vgbook* and add both physical volumes to it. You will use the PE size of 16MB and list and display the volume group and the physical volumes.

1. Create a partition of size 90MB on *sdd* using the *parted* command and confirm. You need to label the disk first, as it is a new disk.

```
[user1@server2 ~]$ sudo parted /dev/sdd mklabel msdos
[user1@server2 ~]$ sudo parted /dev/sdd mkpart primary 1 91m
[user1@server2 ~]$ sudo parted /dev/sdd print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdd: 262MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  91.2MB  90.2MB  primary
```

The *print* subcommand confirms the creation of the partition. It is the first partition on the disk.

2. Set (*set*) the flag on the partition (1) to “lvm” using the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdd set 1 lvm on
```

3. Verify flag activation using the *print* subcommand with *parted*:

```
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  91.2MB  90.2MB  primary               lvm
```

The flag is applied and enabled on the partition as indicated under the Flags column.

4. Initialize the *sdd1* partition and the *sde* disk using the *pvcreate* command. Note that there is no need to apply a disk label on *sde* with *parted* as LVM does not require it.

```
[user1@server2 ~]$ sudo pvcreate /dev/sdd1 /dev/sde -v
    Wiping signatures on new PV /dev/sdd1.
    Wiping signatures on new PV /dev/sde.
    Set up physical volume for "/dev/sdd1" with 176128 available sectors.
    Zeroing start of device /dev/sdd1.
    Writing physical volume data to disk "/dev/sdd1".
  Physical volume "/dev/sdd1" successfully created.
    Set up physical volume for "/dev/sde" with 512000 available sectors.
    Zeroing start of device /dev/sde.
    Writing physical volume data to disk "/dev/sde".
  Physical volume "/dev/sde" successfully created.
```

The command generated a verbose output. You now have two physical volumes available for use.

5. Create *vgbook* volume group using the *vgcreate* command and add the two physical volumes to it. Use the -s option to specify the PE size in MBs.

```
[user1@server2 ~]$ sudo vgcreate -vs 16 vgbook /dev/sdd1 /dev/sde
    Wiping signatures on new PV /dev/sdd1.
    Wiping signatures on new PV /dev/sde.
    Adding physical volume '/dev/sdd1' to volume group 'vgbook'
    Adding physical volume '/dev/sde' to volume group 'vgbook'
    Archiving volume group "vgbook" metadata (seqno 0).
    Creating volume group backup "/etc/lvm/backup/vgbook" (seqno 1).
  Volume group "vgbook" successfully created
```

The above command combines the two options with a single hyphen.

6. List the volume group information:

```
[user1@server2 ~]$ sudo vgs vgbook
  VG     #PV #LV #SN Attr   VSize   VFree
  vgbook   2   0   0 wz--n- 320.00m 320.00m
```

The total capacity available in the *vgbook* volume group is 320MB.

7. Display detailed information about the volume group and the physical volumes it contains:

```
[user1@server2 ~]$ sudo vgdisplay -v vgbook
  --- Volume group ---
  VG Name               vgbook
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               320.00 MiB
  PE Size               16.00 MiB
  Total PE              20
  Alloc PE / Size       0 / 0
  Free  PE / Size       20 / 320.00 MiB
  VG UUID               CQVzWQ-myrd-OM2o-URWP-jHzE-sUBd-V9ZAOj

  --- Physical volumes ---
  PV Name               /dev/sdd1
  PV UUID               OcxCYf-yGYp-pUzK-MlSg-ZZ56-6doL-d1ZJwV
  PV Status             allocatable
  Total PE / Free PE    5 / 5

  PV Name               /dev/sde
  PV UUID               GnnUDd-RrV2-4Ylq-KPJO-tgt3-cPyy-bqfVOD
  PV Status             allocatable
  Total PE / Free PE    15 / 15
```

The verbose output includes the physical volume attributes as well. There are a total of 20 PEs in the volume group (5 in *sdd1* and 15 in *sde*), and each PE is 16MB in size. The collective size of all the physical volumes represents the total size of the volume group, which is 20x16 = 320MB.

8. List the physical volume information:

```
[user1@server2 ~]$ sudo pvs
  PV         VG     Fmt  Attr PSize   PFree
  /dev/sda2  rhel   lvm2 a--   <9.00g      0
  /dev/sdc          lvm2 ---  250.00m 250.00m
  /dev/sdd1  vgbook lvm2 a--   80.00m  80.00m
  /dev/sde   vgbook lvm2 a--  240.00m 240.00m
```

The output shows the physical volumes in *vgbook*, along with their utilization status.

9. Display detailed information about the physical volumes:

```
[user1@server2 ~]$ sudo pvdisplay /dev/sdd1
  --- Physical volume ---
  PV Name               /dev/sdd1
  VG Name               vgbook
  PV Size               86.00 MiB / not usable 6.00 MiB
  Allocatable           yes
  PE Size               16.00 MiB
  Total PE              5
  Free PE               5
  Allocated PE          0
  PV UUID               OcxCYf-yGYp-pUzK-MlSg-ZZ56-6doL-d1ZJwV

[user1@server2 ~]$ sudo pvdisplay /dev/sde
  --- Physical volume ---
  PV Name               /dev/sde
  VG Name               vgbook
  PV Size               250.00 MiB / not usable 10.00 MiB
  Allocatable           yes
  PE Size               16.00 MiB
  Total PE              15
  Free PE               15
  Allocated PE          0
  PV UUID               GnnUDd-RrV2-4Ylq-KPJO-tgt3-cPyy-bqfVOD
```

Once a partition or disk is initialized and added to a volume group, they are treated identically within the volume group. LVM does not prefer one over the other.

Exercise 14-2: Create Logical Volumes

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create two logical volumes, *lvol0* and *lvbook1*, in the *vgbook* volume group. You will use 120MB for *lvol0* and 192MB for *lvbook1* from the available pool of space. You will display the details of the volume group and the logical volumes.

1. Create a logical volume with the default name *lvol0* using the *lvcreate* command. Use the -L option to specify the logical volume size, 120MB. You may use the -v, -vv, or -vvv option with the command for verbosity.

```
[user1@server2 ~]$ sudo lvcreate -vL 120 vgbook
  Rounding up size to full physical extent 128.00 MiB
    Archiving volume group "vgbook" metadata (seqno 1).
    Creating logical volume lvol0
    Creating volume group backup "/etc/lvm/backup/vgbook" (seqno 2).
    Activating logical volume vgbook/lvol0.
    activation/volume_list configuration setting not defined: Checking only host
 tags for vgbook/lvol0.
    Creating vgbook-lvol0
    Loading table for vgbook-lvol0 (253:2).
    Resuming vgbook-lvol0 (253:2).
    Wiping known signatures on logical volume "vgbook/lvol0"
    Initializing 4.00 KiB of logical volume "vgbook/lvol0" with value 0.
  Logical volume "lvol0" created.
```

The size for the logical volume may be specified in units such as MBs, GBs, TBs, or as a count of LEs; however, MB is the default if no unit is specified (see the previous command). The size of a logical volume is always in multiples of the PE size. For instance, logical volumes created in *vgbook* with the PE size set at 16MB can be 16MB, 32MB, 48MB, 64MB, and so on. The output above indicates that the logical volume is 128MB (16x8), and not 120MB as specified.

2. Create *lvbook1* of size 192MB (16x12) using the *lvcreate* command. Use the -l switch to specify the size in logical extents and -n for the custom name. You may use -v for verbose information.

```
[user1@server2 ~]$ sudo lvcreate -l 12 -n lvbook1 vgbook
  Logical volume "lvbook1" created.
```

3. List the logical volume information:

```
[user1@server2 ~]$ sudo lvs
  LV      VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  root    rhel   -wi-ao----  <8.00g
  swap    rhel   -wi-ao----   1.00g
  lvbook1 vgbook -wi-a----- 192.00m
  lvol0   vgbook -wi-a----- 128.00m
```

Both logical volumes are listed in the output with their attributes and sizes.

4. Display detailed information about the volume group including the logical volumes and the physical volumes:

```
[user1@server2 ~]$ sudo vgdisplay -v vgbook
........
--- Logical volume ---
LV Path                /dev/vgbook/lvol0
LV Name                lvol0
VG Name                vgbook
LV UUID                J9O2So-7x5x-2rhb-vaSM-vB2P-YSfu-cpCpFu
LV Write Access        read/write
LV Creation host, time server2.example.com, 2019-10-27 20:07:14 -0400
LV Status              available
# open                 0
LV Size                128.00 MiB
Current LE             8
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     8192
Block device           253:2

--- Logical volume ---
LV Path                /dev/vgbook/lvbook1
LV Name                lvbook1
VG Name                vgbook
LV UUID                ls46dQ-L3AN-UcS5-Ok1D-ljs8-GwJO-wOw7qu
LV Write Access        read/write
LV Creation host, time server2.example.com, 2019-10-27 20:19:11 -0400
LV Status              available
# open                 0
LV Size                192.00 MiB
Current LE             12
Segments               2
Allocation             inherit
Read ahead sectors     auto
- currently set to     8192
Block device           253:3

........
```

Alternatively, you can run the following to view only the logical volume details:

```
[user1@server2 ~]$ sudo lvdisplay /dev/vgbook/lvol0
[user1@server2 ~]$ sudo lvdisplay /dev/vgbook/lvbook1
```

Review the attributes of the logical volumes as detailed above.

Exercise 14-3: Extend a Volume Group and a Logical Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will add another partition *sdd2* of size 158MB to *vgbook* to increase the pool of allocatable space. You will initialize the new partition prior to adding it to the volume group. You will increase the size of *lvbook1* to 336MB. You will

display basic information for the physical volumes, volume group, and logical volume.

1. Create a partition of size 158MB on *sdd* and set the flag to “lvm” using the *parted* command. Display the new partition to confirm the partition number, size, and flag.

```
[user1@server2 ~]$ sudo parted /dev/sdd mkpart pri 92 250
[user1@server2 ~]$ sudo parted /dev/sdd set 2 lvm on
[user1@server2 ~]$ sudo parted /dev/sdd print

........
 2       92.3MB  250MB   157MB   primary               lvm
```

2. Initialize *sdd2* using the *pvcreate* command:

```
[user1@server2 ~]$ sudo pvcreate /dev/sdd2
  Physical volume "/dev/sdd2" successfully created.
```

3. Extend *vgbook* by adding the new physical volume to it:

```
[user1@server2 ~]$ sudo vgextend vgbook /dev/sdd2
  Volume group "vgbook" successfully extended
```

4. List the volume group:

```
[user1@server2 ~]$ sudo vgs
  VG     #PV #LV #SN Attr   VSize   VFree
  rhel     1   2   0 wz--n-  <9.00g      0
  vgbook   3   2   0 wz--n- 464.00m 144.00m
```

The output reflects the addition of a third physical volume to *vgbook*. The total capacity of the volume group has now increased to 464MB with 144MB free.

5. Extend the size of *lvbook1* to 340MB by adding 144MB using the *lvextend* command:

```
[user1@server2 ~]$ sudo lvextend -L +144 /dev/vgbook/lvbook1
  Size of logical volume vgbook/lvbook1 changed from 192.00 MiB (12 extents) to 336.00 MiB (2
1 extents).
  Logical volume vgbook/lvbook1 successfully resized.
```

EXAM TIP: Make sure the expansion of a logical volume does not affect the file system and the data it contains. More details in Chapter 15.

6. Issue *vgdisplay* on *vgbook* with the -v switch for the updated details:

```
[user1@server2 ~]$ sudo vgdisplay -v vgbook
<output truncated>
```

The output will show a lot of information about the volume group and the logical and physical volumes it contains. It will reflect the updates made in this exercise. In fact, each time a volume group or a logical volume is resized, *vgdisplay* will reflect those changes. The above output will display three physical volumes with the combined allocatable space grown to 464MB. The number of PEs will have increased to 29, with all of them allocated to logical volumes and 0 unused. The Logical Volume sections will display the updated information for the logical volumes. And at the very bottom, the three physical volumes will show with their device names, and total and available PEs in each.

7. View a summary of the physical volumes:

```
[user1@server2 ~]$ sudo pvs
  PV         VG     Fmt  Attr PSize   PFree
  /dev/sda2  rhel   lvm2 a--   <9.00g      0
  /dev/sdc          lvm2 ---  250.00m 250.00m
  /dev/sdd1  vgbook lvm2 a--   80.00m      0
  /dev/sdd2  vgbook lvm2 a--  144.00m      0
  /dev/sde   vgbook lvm2 a--  240.00m      0
```

8. View a summary of the logical volumes:

```
[user1@server2 ~]$ sudo lvs
  LV      VG     Attr       LSize   Po
  root    rhel   -wi-ao----  <8.00g
  swap    rhel   -wi-ao----   1.00g
  lvbook1 vgbook -wi-a----- 336.00m
  lvol0   vgbook -wi-a----- 128.00m
```

This brings the exercise to an end.

Exercise 14-4: Rename, Reduce, Extend, and Remove Logical Volumes

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will rename *lvol0* to *lvbook2*. You will decrease the size of *lvbook2* to 50MB using the *lvreduce* command and then add 32MB with the *lvresize* command. You will then remove both logical volumes. You will display the summary for the volume groups, logical volumes, and physical volumes.

1. Rename *lvol0* to *lvbook2* using the *lvrename* command and confirm with *lvs*:

```
[user1@server2 ~]$ sudo lvrename vgbook lvol0 lvbook2
  Renamed "lvol0" to "lvbook2" in volume group "vgbook"
[user1@server2 ~]$ sudo lvs
  LV      VG     Attr       LSize   Pool Origin Data%
  root    rhel   -wi-ao----  <8.00g
  swap    rhel   -wi-ao----   1.00g
  lvbook1 vgbook -wi-a----- 336.00m
  lvbook2 vgbook -wi-a----- 128.00m
```

2. Reduce the size of *lvbook2* to 50MB with the *lvreduce* command. Specify the absolute desired size for the logical volume. Answer “Do you really want to reduce vgbook/lvbook2?” in the affirmative.

```
[user1@server2 ~]$ sudo lvreduce -L 50 /dev/vgbook/lvbook2
  Rounding size to boundary between physical extents: 64.00 MiB.
  WARNING: Reducing active logical volume to 64.00 MiB.
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce vgbook/lvbook2? [y/n]: y
  Size of logical volume vgbook/lvbook2 changed from 128.00 MiB (8 extents) to 64.00 MiB (4 e
xtents).
  Logical volume vgbook/lvbook2 successfully resized.
```

3. Add 32MB to *lvbook2* with the *lvresize* command:

```
[user1@server2 ~]$ sudo lvresize -L +32 /dev/vgbook/lvbook2
  Size of logical volume vgbook/lvbook2 changed from 64.00 MiB (4 extents) to 96.00 MiB (6 ex
tents).
  Logical volume vgbook/lvbook2 successfully resized.
```

4. Use the *pvs*, *lvs*, *vgs*, and *vgdisplay* commands to view the updated allocation.
5. Remove both *lvbook1* and *lvbook2* logical volumes using the *lvremove* command. Use the -f option to suppress the “Do you really want to remove active logical volume” message.

```
[user1@server2 ~]$ sudo lvremove /dev/vgbook/lvbook1 -f
  Logical volume "lvbook1" successfully removed
[user1@server2 ~]$ sudo lvremove /dev/vgbook/lvbook2 -f
  Logical volume "lvbook2" successfully removed
```

Removing a logical volume is a destructive task. You need to ensure that you perform a backup of any data in the target logical volume prior to deleting it. You will need to unmount the file system or disable swap in the logical volume. See Chapter 15 on how to unmount a file system and disable swap.

6. Execute the *vgdisplay* command and *grep* for “Cur LV” to see the number of logical volumes currently available in *vgbook*. It should show 0, as you have removed both logical volumes.

```
[user1@server2 ~]$ sudo vgdisplay vgbook | grep 'Cur LV'
  Cur LV                0
```

This concludes the exercise.

Exercise 14-5: Reduce and Remove a Volume Group

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will reduce *vgbook* by removing the *sdd1* and *sde* physical volumes from it, and then remove the volume group. Confirm the deletion of the volume group and the logical volumes at the end.

1. Remove *sdd1* and *sde* physical volumes from *vgbook* by issuing the *vgreduce* command:

```
[user1@server2 ~]$ sudo vgreduce vgbook /dev/sdd1 /dev/sde
  Removed "/dev/sdd1" from volume group "vgbook"
  Removed "/dev/sde" from volume group "vgbook"
```

2. Remove the volume group using the *vgremove* command. This will also remove the last physical volume, *sdd2*, from it.

```
[user1@server2 ~]$ sudo vgremove vgbook
  Volume group "vgbook" successfully removed
```

You can also use the -f option with the *vgremove* command to force the volume group removal even if it contains any number of logical and physical volumes in it.

Remember to proceed with caution whenever you perform reduce and remove operations.

3. Execute the *vgs* and *lvs* commands for confirmation:

```
[user1@server2 ~]$ sudo vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  rhel   1   2   0 wz--n- <9.00g    0
[user1@server2 ~]$
[user1@server2 ~]$ sudo lvs
  LV   VG   Attr       LSize  Pool Orig
  root rhel -wi-ao---- <8.00g
  swap rhel -wi-ao----  1.00g
```

This concludes the exercise.

Exercise 14-6: Uninitialize Physical Volumes

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will uninitialize all three physical volumes—*sdd1*, *sdd2*, and *sde*—by deleting the LVM structural information from them. Use the *pvs* command for confirmation. Remove the partitions from the *sdd* disk and verify that all disks used in Exercises 14-1 to 14-5 are now in their original raw state.

1. Remove the LVM structures from *sdd1*, *sdd2*, and *sde* using the *pvremove* command:

```
[user1@server2 ~]$ sudo pvremove /dev/sdd1 /dev/sdd2 /dev/sde
  Labels on physical volume "/dev/sdd1" successfully wiped.
  Labels on physical volume "/dev/sdd2" successfully wiped.
  Labels on physical volume "/dev/sde" successfully wiped.
```

2. Confirm the removal using the *pvs* command:

```
[user1@server2 ~]$ sudo pvs
  PV         VG   Fmt  Attr PSize   PFree
  /dev/sda2  rhel lvm2 a--   <9.00g      0
  /dev/sdc        lvm2 ---  250.00m 250.00m
```

The partitions and the disk are now back to their raw state and can be repurposed.

3. Remove the partitions from *sdd* using the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdd rm 1 ; sudo parted /dev/sdd rm 2
```

4. Verify that all disks used in previous exercises have returned to their original raw state using the *lsblk* command:

```
[user1@server2 ~]$ lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda              8:0    0   10G  0 disk
├─sda1           8:1    0    1G  0 part /boot
└─sda2           8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sdb              8:16   0  250M  0 disk
sdc              8:32   0  250M  0 disk
sdd              8:48   0  250M  0 disk
sde              8:64   0  250M  0 disk
sdf              8:80   0    4G  0 disk
sdg              8:96   0    1G  0 disk
sdh              8:112  0    1G  0 disk
```

This brings the exercise to an end.

We will recreate logical volumes in Chapter 15 and construct file system and swap structures in them.

Stratis Volume-Managing File System

RHEL 8 introduces a new simplified storage management solution called *Stratis*. Stratis capitalizes on three existing matured storage components: the *device mapper* (dm) kernel

driver, the LVM solution, and the XFS file system. It implements the advanced features of these components to deliver file systems that are encapsulated within logical volumes. These logical volumes are created and expanded dynamically and transparently, hiding the underlying complexity. The Stratis solution delivers file systems that are referred to as *volume-managing file systems*. Stratis uses the thin provisioning technology as part of the solution.

File systems are explained in Chapter 15 “Local File Systems and Swap”.

The central idea surrounding the Stratis solution is a storage *pool*. A storage pool is created using at least one disk or partition, which is referred to as a *blockdev*. There can be a combination of the two in a single pool and more can be added as the space requirement grows. The total capacity of the pool is the aggregate of the spaces taken from all the block devices that are part of the pool. Within a pool, *file systems* can be created, all sharing the entire pool capacity.

LVM logical volumes can also be used as block devices in a Stratis pool.

Figure 14-2 provides a bird’s-eye view of the three major objects—pool, blockdev, and file system—used in Stratis.

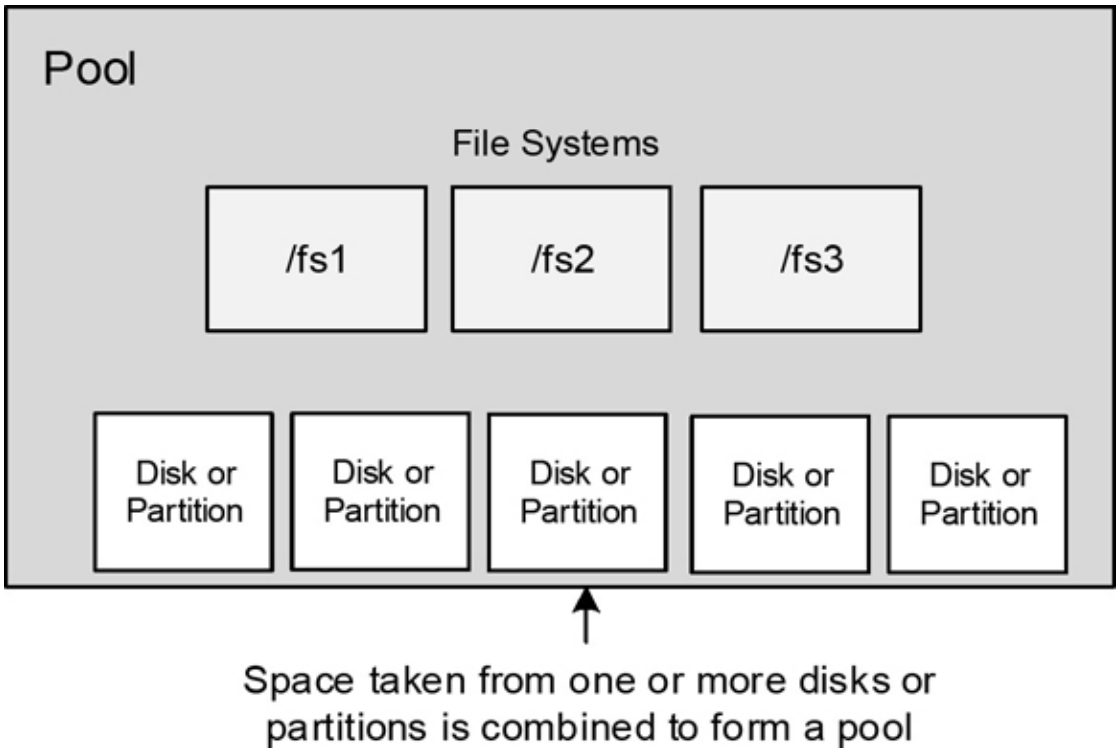


Figure 14-2 Stratis Structure

The diagram in Figure 14-2 shows a pool containing multiple disks and partitions. The pool capacity is the aggregate of all the block storage devices included in the pool, and this capacity is shared among all the file systems. Each Stratis file system appears to occupy the entire pool capacity exclusively; however, they are thinly provisioned. Their actual size grows as the amount of data stored in them increases. Stratis takes care of the dynamic expansion of the file systems and the underlying volumes as needed. As Stratis handles the creation, formatting, and expansion of the file systems, they must not be manually initialized or reconfigured.

Stratis Management Operations and Command

The primary command to manage Stratis is called *stratis*. This command has a set of subcommands available to perform

management operations such as creating, viewing, renaming, and destroying pools and file systems, and expanding pools.

Table 14-2 summarizes the common Stratis subcommands that are employed to accomplish various management tasks.

Command	Description
pool	Administers storage pools. Subcommands are available to list, create, rename, expand, and destroy a pool.
blockdev	Lists block devices
filesystem	Administers file systems within storage pools. Subcommands are available to list, create, rename, and destroy a file system.

Table 14-2 Common Stratis Subcommands

Stratis runs as a service, so its operational state can be managed with the *systemctl* command. The *stratis* command interacts with the Stratis service to manage the pool and file systems dynamically.

For each pool added, Stratis creates a subdirectory under the */stratis* directory matching the pool name. It then creates a symbolic link for each file system under that subdirectory to the actual device file located in the */dev* directory.

Stratis file systems are not fixed-sized; however, pool space can be reserved to assure availability if multiple file systems share a pool.

Exercise 14-7: Install Software and Activate Stratis

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will install the Stratis software packages, start the Stratis service, and mark it for autostart on

subsequent system reboots.

1. Install the packages *stratisd* and *stratis-cli*:

```
[user1@server2 ~]$ sudo dnf install stratisd stratis-cli -y
........
Installed:
  stratis-cli-1.0.2-1.el8.noarch
  stratisd-1.0.3-1.el8.x86_64
  python3-dbus-client-gen-0.4-1.el8.noarch
  python3-dbus-python-client-gen-0.6-2.el8.noarch
  python3-dbus-signature-pyparsing-0.03-2.el8.noarch
  python3-into-dbus-python-0.06-2.el8.noarch
  python3-justbases-0.9-6.el8.noarch
  python3-justbytes-0.11-2.el8.noarch
  python3-pyparsing-2.1.10-7.el8.noarch

Complete!
```

2. Start the service and enable it to start automatically on future system reboots:

```
[user1@server2 ~]$ sudo systemctl --now enable stratisd
```

3. Check the operational status of the service:

```
[user1@server2 ~]$ sudo systemctl status stratisd
● stratisd.service - A daemon that manages a pool of block devices to create flexible file s>
   Loaded: loaded (/usr/lib/systemd/system/stratisd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-10-30 22:58:16 EDT; 10s ago
     Docs: man:stratisd(8)
 Main PID: 24310 (stratisd)
    Tasks: 1 (limit: 11516)
   Memory: 980.0K
   CGroup: /system.slice/stratisd.service
           └─24310 /usr/libexec/stratisd --debug
```

The relevant packages for the Stratis storage management solution are installed, and the Stratis service is started and activated. This concludes the exercise.

Exercise 14-8: Create and Confirm a Pool and File System

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a Stratis pool and a file system in it. You will display information about the pool, file system, and device used.

1. You allocated 2x1GB disks: *sdg* and *sdh* for Stratis exercises. Use the *lsblk* command to confirm their availability:

```
[user1@server2 ~]$ lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda             8:0    0   10G  0 disk
├─sda1          8:1    0    1G  0 part /boot
└─sda2          8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sdb             8:16   0  250M  0 disk
sdc             8:32   0  250M  0 disk
sdd             8:48   0  250M  0 disk
sde             8:64   0  250M  0 disk
sdf             8:80   0    4G  0 disk
sdg             8:96   0    1G  0 disk
sdh             8:112  0    1G  0 disk
```

The 2x1GB disks are available and unused.

2. Create a pool called *bookpool* using the *sdg* disk and verify the creation:

```
[user1@server2 ~]$ sudo stratis pool create bookpool /dev/sdg
[user1@server2 ~]$ sudo stratis pool list
Name        Total Physical Size  Total Physical Used
bookpool                 1 GiB               52 MiB
```

The specified pool *bookpool* is created. The second command output returns the basic information about the pool, including the number of physical devices used in the pool and their sizes, and the amount of in-use space in MiBs.

3. Display the block device that is used to form the pool:

```
[user1@server2 ~]$ sudo stratis blockdev list bookpool
Pool Name  Device Node    Physical Size   State  Tier
bookpool   /dev/sdg               1 GiB  In-use  Data
```

The pool *bookpool* has a single disk of size 1GiB and it is currently in use.

4. Create a file system called *bookfs* in the *bookpool* and verify the creation:

```
[user1@server2 ~]$ sudo stratis filesystem create bookpool bookfs
[user1@server2 ~]$ sudo stratis filesystem list
Pool Name  Name    Used     Created            Device                   UUID

bookpool   bookfs  546 MiB  Oct 30 2019 23:33  /stratis/bookpool/bookfs  516e345bc9fa4a90b1ec
dcc7a1d4d12f
```

The pool *bookpool* has a single file system *bookfs* of used size 546MiB. Its device file is */stratis/bookpool/bookfs*. The file system's UUID is also displayed.

5. Create a directory called */bookfs1* and mount the new file system on it:

```
[user1@server2 ~]$ sudo mkdir /bookfs1
[user1@server2 ~]$ sudo mount /stratis/bookpool/bookfs /bookfs1
```

6. Check the pool usage:

```
[user1@server2 ~]$ sudo stratis pool list
Name        Total Physical Size  Total Physical Used
bookpool                  1 GiB              598 MiB
```

Observe how the usage grew from 52MiB to 598MiB. This brings this exercise to an end.

Exercise 14-9: Expand and Rename a Pool and File System

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will expand the Stratis pool *bookpool* (created in Exercise 14-8) using the *sdh* disk. You will rename the pool and the file system it contains.

1. Expand the *bookpool* pool by adding the *sdh* disk to it:

```
[user1@server2 ~]$ sudo stratis pool add-data bookpool /dev/sdh
```

2. Verify the new capacity of the pool:

```
[user1@server2 ~]$ sudo stratis pool list
Name        Total Physical Size  Total Physical Used
bookpool                  2 GiB              602 MiB
```

The addition of another 1GB disk brings the total physical size to 2GiB.

3. Change the name of the pool from *bookpool* to *rhcsapool* and verify:

```
[user1@server2 ~]$ sudo stratis pool rename bookpool rhcsapool
[user1@server2 ~]$
[user1@server2 ~]$ sudo stratis pool list
Name         Total Physical Size  Total Physical Used
rhcsapool                  2 GiB              602 MiB
```

The new name is depicted in the first column.

4. Change the name of the file system from *bookfs* to *rhcsafs*, and verify:

```
[user1@server2 ~]$ sudo stratis filesystem rename rhcsapool bookfs rhcsafs
[user1@server2 ~]$ sudo stratis filesystem list
Pool Name  Name     Used     Created            Device                      UUID

rhcsapool  rhcsafs  546 MiB  Oct 30 2019 23:33  /stratis/rhcsapool/rhcsafs  516e345bc9fa4a90b
1ecdcc7a1d4d12f
```

The file system name is changed and it’s reflected in the output of the second command. The exercise is completed successfully.

Exercise 14-10: Destroy a File System and Pool

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will destroy the Stratis file system and the pool that was created, expanded, and renamed in Exercises 14-8 and 14-9. You will verify the deletion with appropriate commands.

1. The first step in the process is to unmount the file system *rhcsafs1* from its mount point */bookfs1*:

```
[user1@server2 ~]$ sudo umount /bookfs1
```

Unmounting a file system ensures that the file system is not busy.

2. Remove the file system *rhcsafs* from the pool:

```
[user1@server2 ~]$ sudo stratis filesystem destroy rhcsapool rhcsafs
```

3. Remove the pool *rhcsapool* from the system:

```
[user1@server2 ~]$ sudo stratis pool destroy rhcsapool
```

4. Confirm the removal of the file system and the pool:

```
[user1@server2 ~]$ sudo stratis filesystem list
Pool Name  Name  Used  Created  Device  UUID
[user1@server2 ~]$
[user1@server2 ~]$ sudo stratis pool list
Name    Total Physical Size  Total Physical Used
```

The above outputs confirm the removal of the file system and the pool.

5. Verify that both *sdg* and *sdh* disks used in the previous Stratis exercises have returned to their original raw state using the *lsblk* command:

```
[user1@server2 ~]$ lsblk /dev/sdg /dev/sdh
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdg    8:96   0   1G  0 disk
sdh    8:112  0   1G  0 disk
```

This concludes the exercise.

We will recreate Stratis file systems in Chapter 15 and add them for persistent mounting.

Chapter Summary

This chapter explicated two advanced storage management solutions: Logical Volume Manager and Stratis. The LVM solution has been around in RHEL for decades. The Stratis solution, on the other hand, is recently added.

We discovered how LVM works. We looked at various LVM objects and their relationship with one another. We explored LVM management commands and common options available with them. We performed a series of exercises to demonstrate the creation, expansion, renaming, reduction, and deletion of physical volumes, storage pools, and logical volumes.

The next advanced storage solution we looked at is called Stratis, which is a volume-managing file system. Stratis takes advantage of the underlying LVM functionality and the device mapper kernel driver to create and expand the XFS file system that it holds. Stratis dynamically expands the underlying LVM volume without the need for manual administrative intervention. We executed a couple of step-by-step exercises to explain the creation, growth, renaming, and removal of Stratis pools and file systems.

Check Your Understanding

1. The *parted* utility may be used to create LVM logical volumes. True or False?
2. What are the two commands that you can use to reduce the number of logical extents from a logical volume?
3. Stratis uses LVM as the underlying logical volume management solution. True or False?
4. Provide the command to add physical volumes */dev/sdd1* and */dev/sdc* to *vg20* volume group.
5. What are the two commands that you can use to add logical extents to a logical volume?
6. Provide the command to create a volume group called *vg20* on */dev/sdd* disk with physical extent size 64MB.
7. Name the three Stratis objects?

8. Provide the command to remove *vg20* volume group along with any logical and physical volumes that it contains.
9. What is the default size of a physical extent in LVM?
10. What is the default name for the first logical volume in a volume group?
11. What is one difference between the *pvs* and *pvdisplay* commands?
12. When can a disk or partition be referred to as a physical volume?
13. Provide the command to remove *webvol* logical volume from *vg20* volume group.
14. It is necessary to create file system structures in a logical volume before it can be used to store files in it. True or False?
15. What would the command *stratis pool create pool1 /dev/sdg* do?
16. Physical and logical extents are typically of the same size. True or False?
17. What is the purpose of the *pvremove* command?
18. What would the command *pvcreate /dev/sdd* do?
19. A disk or partition can be added to a volume group without being initialized. True or False?
20. What is the file system type that is created in a Stratis file system?
21. Provide the command to create a logical volume called *webvol* of size equal to 100 logical extents in *vg20* volume group.
22. A volume group can be created without any physical volume in it. True or False?
23. A single disk can be used by both *parted* and LVM solutions at the same time. True or False?
24. Provide the command to add */dev/sdh* to an existing pool called *pool1*.
25. Provide the command to remove */dev/sdd1* physical volume from *vg20* volume group.
26. A partition can be used as an LVM object. True or False?

27. Which command would we use to view the details of a volume group and logical and physical volumes it contains?

Answers to Check Your Understanding

1. False.
2. The *lvreduce* and *lvresize* commands.
3. True.
4. *vgextend vg20 /dev/sdd1 /dev/sdc*
5. The *lvextend* and *lvresize* commands.
6. *vgcreate -s 64 vg20 /dev/sdd*
7. The three Stratis objects are pool, block device, and file system.
8. *vgremove -f vg20*
9. The default PE size is 4MB.
10. *lvol0* is the default name for the first logical volume created in a volume group.
11. The *pvs* command lists basic information about physical volumes whereas the *pvdisplay* command shows the details.
12. After the *pvcreate* command has been executed on it successfully.
13. *lvremove /dev/vg20/webvol*
14. True.
15. The command provided will create a Stratis pool called *pool1* containing */dev/sdg* device.
16. True.
17. The *pvremove* command is used to remove LVM information from a physical volume.
18. The command provided will prepare the */dev/sdd* disk for use in a volume group.
19. False. A disk or partition must be initialized before it can be added to a volume group.
20. XFS.

21. *lvcreate -l 100 -n webvol vg20*
22. False.
23. True. A single disk can be shared between *parted*-created partitions and LVM.
24. *stratis pool add-data pool1 /dev/sdh*
25. *vgreduce vg20 /dev/sdd1*
26. True.
27. The *vgdisplay* command with the -v option.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Add more storage to *server2* if required.

Lab 14-1: Create Volume Group and Logical Volumes

As *user1* with *sudo* on *server2*, initialize 1x250MB disk for use in LVM (use *lsblk* to identify available disks). Create volume group *vg100* with PE size 16MB and add the physical volume. Create two logical volumes *lvol0* and *swapvol* of sizes 100MB and 120MB. Use the *vgs*, *pvs*, *lvs*, and *vgdisplay* commands for verification. (Hint: Logical Volume Manager).

Lab 14-2: Expand Volume Group and Logical Volume

As *user1* with *sudo* on *server2*, create a partition on an available 250MB disk and initialize it for use in LVM (use *lsblk* to identify available disks). Add the new physical volume to *vg100*. Expand the *lvol0* logical volume to size 300MB. Use the

vgs, *pvs*, *lvs*, and *vgdisplay* commands for verification. (Hint: Logical Volume Manager).

Lab 14-3: Reduce and Remove Logical Volumes

As *user1* with *sudo* on *server2*, reduce the size of *lvol0* logical volume to 80MB. Then remove both logical volumes *swapvol* and *lvol0*. Confirm the deletion with *vgs*, *pvs*, *lvs*, and *vgdisplay* commands. (Hint: Logical Volume Manager).

Lab 14-4: Remove Volume Group and Physical Volumes

As *user1* with *sudo* on *server2*, remove the volume group and uninitialized the physical volumes. Confirm the deletion with *vgs*, *pvs*, *lvs*, and *vgdisplay* commands. Use the *lsblk* command and verify that the disks used for the LVM labs no longer show LVM information. (Hint: Logical Volume Manager).

Lab 14-5: Create Stratis Pool

As *user1* with *sudo* on *server2*, check to see if Stratis software is installed and the Stratis service is enabled and started. Identify 2x1GB disks with the *lsblk* command and ensure they are not in use. Create pool *strpool* on one of the 1GB disks and verify the pool and the block device with the *stratis* command. (Hint: Stratis Volume-Managing File System).

Lab 14-6: Expand and Destroy Stratis Pool

As *user1* with *sudo* on *server2*, use the other 1GB disk and expand *strpool* using the *stratis* command. Verify the expansion. Finally, remove the entire pool and confirm the deletion. Use the *lsblk* command and verify that the disks used for the Stratis labs no longer show Stratis information. (Hint: Stratis Volume-Managing File System).

Chapter 15

Local File Systems and Swap

This chapter describes the following major topics:

- Understand file systems and their benefits, categories, and types
- Review file system types: Ext3/Ext4, XFS, VFAT, and ISO9660
- Know file system administration commandset
- Mount and unmount file systems manually and persistently
- Determine and use UUID
- Apply and use file system label
- Monitor file system and directory usage
- Create and mount different types of local file systems in partitions
- Create and mount XFS file system in VDO volume

- Create, mount, and resize Ext4 and XFS file systems in LVM
- Create, mount, and expand Stratis file system
- Understand, create, and activate swap in partitions and LVM

RHCSA Objectives:

26. Configure systems to mount file systems at boot by Universally Unique ID (UUID) or label
27. Add new partitions and logical volumes, and swap to a system non-destructively (the first part of this objective is covered in more detail in Chapter 14)
28. Create, mount, unmount, and use vfat, ext4, and xfs file systems
30. Extend existing logical volumes (more details is also available in Chapter 14)

File systems are the most common structures created in partitions and volumes regardless of the underlying storage management solution employed. They are logical containers employed for file storage and can be optimized, resized, mounted, and unmounted independently. They must be connected to the root of the directory hierarchy in order to be accessed by users and applications. This may be accomplished automatically at system boot or manually when required. File systems can be mounted or unmounted using their unique identifiers, labels, or device files. There is a whole slew of commands available for file system creation and administration; some of them are file system type specific while others are general.

The other common structure created in partitions and logical volumes is the swap space. Swapping provides a mechanism to move out and in pages of idle data between the physical memory and the swap. Swap areas act as extensions to the physical memory, and they may be activated or deactivated independent of swap spaces located in other partitions and volumes.

This chapter is the last one in the three chapter series (the other two being Chapters 13 and 14) on storage management. It elaborates on file systems and swap, and demonstrates their creation and management in several exercises. It also highlights the tools to monitor their usage.

File Systems and File System Types

A *file system* is a logical container that stores files and directories. Each file system is created in a discrete partition, VDO volume, logical volume, or Stratis pool. A typical production RHEL system usually has numerous file systems. During OS installation, only two file systems—*/* and */boot*—are created in the default disk layout, but you can design a custom disk layout and construct separate containers to store dissimilar information. Typical additional file systems created during an installation are */home*, */opt*, */tmp*, */usr*, and */var*. The two mandatory file systems—*/* and */boot*—are required for installation and booting.

Storing disparate data in distinct file systems versus storing all data in a single file system offers the following advantages:

- ✓ Make any file system accessible (mount) or inaccessible (unmount) to users independent of other file systems. This hides or reveals information contained in that file system.
- ✓ Perform file system repair activities on individual file systems
- ✓ Keep dissimilar data in separate file systems
- ✓ Optimize or tune each file system independently
- ✓ Grow or shrink a file system independent of other file systems

RHEL supports several types of file systems that may be categorized in three basic groups: *disk-based*, *network-based*, and *memory-based*. Disk-based file systems are typically created on physical drives using SATA, USB, Fibre Channel, and other technologies. Network-based file systems are essentially disk-based file systems shared over the network for remote access. Memory-based file systems are virtual; they are created at system startup and destroyed when the system goes down. Disk-based and network-based

file systems store information persistently, while any data saved in virtual file systems does not survive across system reboots.

Table 15-1 lists and explains various common disk- and network-based file system types supported in RHEL 8.

File System Type	Category	Description
Ext3	Disk	The third generation of the extended file system. It supports metadata journaling for faster recovery, offers superior reliability, allows the creation of up to 32,000 subdirectories, and supports larger file systems and bigger files than its predecessor.
Ext4	Disk	The fourth generation of the extended file system developed as the successor to Ext3. It supports all features of Ext3 in addition to a larger file system size, bigger file size, an unlimited number of subdirectories, metadata and quota journaling, and extended user attributes.
XFS	Disk	XFS is a highly scalable and high-performing 64-bit file system. It supports metadata journaling for faster crash recovery, and online defragmentation, expansion, quota journaling, and extended user attributes. XFS is the default file system type in RHEL 8.
VFAT	Disk	This file system is used for post-Windows 95 file system formats on hard disks, USB drives, and floppy disks.
ISO9660	Disk	This is used for optical file systems such as CD and DVD.
NFS	Network	Network File System. A shared directory or file system for remote access by other Linux systems.
AutoFS	Network	Auto File System. An NFS file system set to mount and unmount automatically on remote client systems.

Table 15-1 File System Types

This chapter covers Ext3, Ext4, XFS, and VFAT file systems at length. It also touches upon mounting and unmounting ISO9660. For a brief discussion on memory-based file systems, see Chapter 02 “Initial Interaction with the

System”. NFS and AutoFS are discussed in Chapter 16 “Remote File System”.

Extended File Systems

Extended file systems have been part of RHEL for many years. The first generation is obsolete and is no longer supported. The second, third, and fourth generations are currently available and supported. The fourth generation is the latest in the series and is superior in features and enhancements to its predecessors.

The structure of an extended file system is built on a partition or logical volume at the time of file system creation. This structure is divided into two sets. The first set holds the file system’s metadata and it is very tiny. The second set stores the actual data, and it occupies almost the entire partition or the logical volume (VDO, LVM, and Stratis) space.

The metadata includes the *superblock*, which keeps vital file system structural information, such as the type, size, and status of the file system, and the number of data blocks it contains. Since the superblock holds such critical information, it is automatically replicated and maintained at various known locations throughout the file system. The superblock at the beginning of the file system is referred to as the *primary superblock*, and all of its copies as *backup superblocks*. If the primary superblock is corrupted or lost, it renders the file system inaccessible. One of the backup superblocks is then used to supplant the corrupted or lost primary superblock to bring the file system back to its normal state.

The metadata also contains the *inode table*, which maintains a list of *index node* (*inode*) numbers. Each file is assigned an inode number at the time of its creation, and the inode number holds the file’s attributes such as its type,

permissions, ownership, owning group, size, and last access/modification time. The inode also holds and keeps track of the pointers to the actual data blocks where the file contents are located.

The Ext3 and Ext4 file systems support a journaling mechanism that provides them with the ability to recover swiftly after a system crash. Both Ext3 and Ext4 file systems keep track of recent changes in their metadata in a *journal* (or log). Each metadata update is written in its entirety to the journal after completion. The system peruses the journal of each extended file system following the reboot after a crash to determine if there are any errors, and it recovers the file system rapidly using the latest metadata information stored in its journal. The ext2 file system does not support journaling, but the support for journaling may be added to it if required.

In contrast to Ext3 that supports file systems up to 16TiB and files up to 2TiB, Ext4 supports very large file systems up to 1EiB (ExbiByte) and files up to 16TiB (TebiByte). Additionally, Ext4 uses a series of contiguous physical blocks on the hard disk called *extents*, resulting in improved read and write performance with reduced fragmentation. Ext4 supports extended user attributes, acl mount options (to support file permission allocation to specific users and groups), as well as metadata and quota journaling.

XFS File System

The *X File System* (XFS) is a high-performing 64-bit extent-based journaling file system type. XFS allows the creation of file systems and files up to 8EiB (ExbiByte). It does not run file system checks at system boot; rather, it relies on you to use the *xfs_repair* utility to manually fix any issues. XFS sets the extended user attributes and acl mount options by default on new file systems. It enables defragmentation on

mounted and active file systems to keep as much data in contiguous blocks as possible for faster access. The only major caveat with using XFS is its inability to shrink.

Like Ext3 and Ext4, XFS also uses journaling for metadata operations, guaranteeing the consistency of the file system against abnormal or forced unmounting. The journal information is read and any pending metadata transactions are replayed when the XFS file system is remounted.

XFS uses sophisticated techniques in its architecture for speedy input/output performance. It can be snapshot in a mounted, active state. The snapshot can then be used for backup or other purposes.

VFAT File System

VFAT (*Virtual File Allocation Table*) is an extension to the legacy *FAT* file system type, also called *FAT16*, that was introduced in early versions of MS-DOS. The support for FAT16 was later added to Microsoft Windows, MacOS, and some UNIX versions, enabling them to read and write files written in that format. FAT16 had limitations; it was designed to use no more than 8.3 characters in filenames, limiting filenames to a maximum of eight characters plus three characters as an extension. Moreover, it only allowed filenames to begin with a letter or number and to not contain spaces. FAT16 treated lowercase and uppercase letters alike.

VFAT was introduced with Microsoft Windows 95 and it has since been available. It supports 255 characters in filenames including spaces and periods; however, it still does not differentiate between lowercase and uppercase letters. VFAT support was added to Linux several years ago. A VFAT file system may be created on hard drives, but it is primarily used on removable media, such as floppy and USB flash drives, for exchanging data between Linux and Windows.

ISO9660 File System

This file system type conforms to the ISO 9660 standard, hence the name. It is used for removable optical disc media such as CD/DVD drives for transporting software and patches, and operating system images in ISO format between computers. The ISO9660 format originated from the *High-Sierra File System* (HSFS) format, and it has now been enhanced to include innovative features.

File System Management

Managing file systems involves such operations as creating, mounting, labeling, viewing, growing, shrinking, unmounting, and removing them. These management tasks are common to both Extended and XFS types. Most of these functions are also applicable to VFAT and a few to optical file systems.

In Chapters 13 and 14, you created several partitions, VDO volumes, LVM logical volumes, and Stratis volumes. However, you did not initialize them with a file system type (except for Stratis), and therefore you could not mount or use them. Later, you destroyed all the partitions and the volumes that were created. You also deleted the LVM volume group and the Stratis pool. All the disks were returned to their unused state after the completion of the exercises.

Here is a listing of the block devices to confirm the current state of the disks:

```
[user1@server2 ~]$ lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda              8:0    0   10G  0 disk
├─sda1           8:1    0    1G  0 part /boot
└─sda2           8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sdb              8:16   0  250M  0 disk
sdc              8:32   0  250M  0 disk
sdd              8:48   0  250M  0 disk
sde              8:64   0  250M  0 disk
sdf              8:80   0    4G  0 disk
sdg              8:96   0    1G  0 disk
sdh              8:112  0    1G  0 disk
```

The output verifies the unused state and availability status for all the disks—*sdb* through *sdh*. You should be able to reuse them in the exercises in this chapter.

File System Administration Commands

In order to create and manage file systems, RHEL offers a number of commands of which some are limited to their operations on the Extended, XFS, or VFAT file system type, while others are general and applicable to all file system types. Table 15-2 describes common file system administration commands.

Command	Description
Extended File System	
e2label	Modifies the label of a file system
mke2fs	Creates a file system. Can also be invoked as mkfs.ext3, mkfs.ext4, mkfs -t ext3, and mkfs -t ext4.
resize2fs	Resizes a file system. This command is automatically invoked when the lvresize command is run with the -r switch.
tune2fs	Tunes or displays file system attributes
XFS	
mkfs.xfs	Creates a file system. Can also be invoked as mkfs -t xfs.
xfs_admin	Tunes file system attributes
xfs_growfs	Extends the size of a file system
xfs_info	Exhibits information about a file system
VFAT	
mkfs.vfat	Creates a file system. It is equivalent to using mkfs -t vfat.
General File System Commands	
blkid	Displays block device attributes including their UUIDs and labels
df	Reports file system utilization
du	Calculates disk usage of directories and file systems
lsblk	Lists block devices and file systems and their attributes including their UUIDs and labels
mount	Mounts a file system for user access. Displays currently mounted file systems.
umount	Unmounts a file system

Table 15-2 File System Management Commands

Most of these commands are used in this chapter.

Mounting and Unmounting File Systems

In order to enable users to access files and application programs in a file system, the file system must be connected to the directory structure at a desired attachment point, which is referred to as the *mount point*. A mount point in essence is any empty directory that is created and used for this purpose.

There are many file systems already mounted on your system, such as the root file system mounted on */* and the boot file system mounted on */boot*. Both of them are empty directories and are reserved to connect the two file systems to the directory hierarchy. You can use the *mount* command to view information about mounted file systems. The following shows the XFS file systems only:

```
[user1@server2 ~]$ mount -t xfs
/dev/mapper/rhel-root on / type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
/dev/sda1 on /boot type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

The “-t xfs” option makes the command to only show the file systems initialized with the XFS type.

The *mount* command is also used for mounting a file system to a mount point, and this action is performed with the *root* user privileges. The command requires the absolute pathnames of the file system block device and the mount point name. It also accepts the UUID or label of the file system in lieu of the block device name. Options are available with this command to mount all or a specific type of file system. The *mount* command is also used to mount other types of file systems such as those located in removable media. Upon successful mount, the kernel places an entry for the file system in the */proc/self/mounts* file.

A mount point should be empty when an attempt is made to mount a file system on it, otherwise the content of the mount point will hide. As well, the mount point must not be in use or the mount attempt will fail.

The *mount* command supports numerous options that may be used as required to override its default behavior. We can also specify multiple comma-separated options. Table 15-3 describes some common options.

Option	Description
acl (noacl)	Enables (disables) the support for ACLs
auto (noauto)	Mounts (does not mount) the file system when the -a option is specified
defaults	Mounts a file system with all the default values (async, auto, rw, etc.)
_netdev	Used for a file system that requires network connectivity in place before it can be mounted. VDO and NFS are examples.
remount	Remounts an already mounted file system to enable or disable an option
ro (rw)	Mounts a file system read-only (read/write)

Table 15-3 Common mount Command Options

The opposite of the *mount* command is *umount*, which is used to detach a file system from the directory hierarchy and make it inaccessible to users and applications. This command expects the absolute pathname to the block device containing the file system or its mount point name in order to detach it. Options are available with *umount* to unmount all or a specific type of file system. The kernel removes the corresponding file system entry from the */proc/self/mounts* file after it has been successfully disconnected.

Determining the UUID of a File System

Every Extended and XFS file system has a 128-bit (32 hexadecimal characters) UUID (*Universally Unique IDentifier*) assigned to it at the time of its creation. In contrast, UUIDs assigned to vfat file systems are 32-bit (8 hexadecimal characters) in length. Assigning a UUID makes the file

system unique among many other file systems that potentially exist on the system. The primary benefit of using a UUID is the fact that it always stays persistent across system reboots. A UUID is used by default in RHEL 8 in the */etc/fstab* file for any file system that is created by the system in a standard partition.

RHEL attempts to mount all file systems listed in the */etc/fstab* file at reboots. Each file system has an associated device file and UUID, but may or may not have a corresponding label. The system checks for the presence of each file system's device file, UUID, or label, and then attempts to mount it.

The */boot* file system, for instance, is located in a partition and the device file associated with it is on *server2* is */dev/sda1*. You can use the *xfs_admin* command, the *blkid* command, or the *lsblk* command as follows to determine its UUID:

```
[user1@server2 ~]$ sudo xfs_admin -u /dev/sda1
UUID = c1ff315e-4320-442c-a3c5-36db403b53f2
[user1@server2 ~]$ sudo blkid /dev/sda1
/dev/sda1: UUID="c1ff315e-4320-442c-a3c5-36db403b53f2" TYPE="xfs" PARTUUID="7685
6860-01"
[user1@server2 ~]$ sudo lsblk -f /dev/sda1
NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs          c1ff315e-4320-442c-a3c5-36db403b53f2 /boot
```

The UUID reported by the above commands for the */boot* file system is "c1ff315e-4320-442c-a3c5-36db403b53f2". If you *grep* for the string "boot" on the */etc/fstab* file, you will see that the system uses this UUID to mount */boot*. A discussion on the */etc/fstab* file is provided later in this chapter.

For extended file systems, you can use the *tune2fs* command in addition to the *blkid* and *lsblk* commands to determine the UUID.

EXAM TIP: Knowing how to find the UUID of a file system created in a standard partition or with Stratis is important.

A UUID is also assigned to a file system that is created in a VDO or LVM volume; however, it need not be used in the *fstab* file, as the device files associated with the logical volumes are always unique and persistent.

Labeling a File System

A unique label may be used instead of a UUID to keep the file system association with its device file exclusive and persistent across system reboots. A label is limited to a maximum of 12 characters on the XFS file system and 16 characters on the Extended file system. By default, no labels are assigned to a file system at the time of its creation.

The */boot* file system is located in the */dev/sda1* partition and its type is XFS. You can use the *xfs_admin* command, the *blkid* command, or the *lsblk* command as follows to determine its label:

```
[user1@server2 ~]$ sudo xfs_admin -l /dev/sda1
label = ""
[user1@server2 ~]$ sudo blkid /dev/sda1
/dev/sda1: UUID="c1ff315e-4320-442c-a3c5-36db403b53f2" TYPE="xfs" PARTUUID="7685
686O-01"
[user1@server2 ~]$ sudo lsblk -f /dev/sda1
NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sda1 xfs          c1ff315e-4320-442c-a3c5-36db403b53f2 /boot
```

The output discloses that there is currently no label assigned to the */boot* file system.

A label is not needed on a file system if you intend to use its UUID or if it is created in a VDO or LVM logical volume; however, you can still apply one using the *xfs_admin* command with the -L option. Labeling an XFS file system requires that the target file system be unmounted.

The following example demonstrates the steps to unmount */boot*, set the label “bootfs” on its device file, and remount it:

```
[user1@server2 ~]$ sudo umount /boot
[user1@server2 ~]$ sudo xfs_admin -L bootfs /dev/sda1
writing all SBs
new label = "bootfs"
[user1@server2 ~]$ sudo mount /boot
```

You can confirm the new label by executing **sudo xfs_admin -l /dev/sda1**, **sudo blkid /dev/sda1**, or **sudo lsblk -f /dev/sda1**.

For extended file systems, you can use the *e2label* command to apply a label and the *tune2fs*, *blkid*, and *lsblk* commands to view and verify.

Now you can replace the UUID="c1ff315e-4320-442c-a3c5-36db403b53f2" for */boot* in the *fstab* file with LABEL=bootfs, and unmount and remount */boot* as demonstrated above for confirmation.

A label may also be applied to a file system created in a VDO or LVM volume; however, it is not recommended for use in the *fstab* file, as the device files for these logical volumes are always unique and remain persistent across system reboots.

Automatically Mounting a File System at Reboots

File systems defined in the */etc/fstab* file are mounted automatically at reboots. This file must contain proper and complete information for each listed file system. An incomplete or inaccurate entry might leave the system in an undesirable or unbootable state. Another benefit of adding entries to this file is that you only need to specify one of the four attributes—block device name, UUID, label, or mount point—of the file system that you wish to mount manually with the *mount* command. The *mount* command obtains the rest of the information from this file. Similarly, you only need

to specify one of these attributes with the *umount* command to detach it from the directory hierarchy.

The default *fstab* file contains entries for file systems that are created at the time of installation. On *server2*, for instance, this file currently has the following three entries:

```
/dev/mapper/rhel-root                     /       xfs    defaults    0 0
UUID=c1ff315e-4320-442c-a3c5-36db403b53f2 /boot   xfs    defaults    0 0
/dev/mapper/rhel-swap                     swap    swap   defaults    0 0
```

EXAM TIP: Any missing or invalid entry in this file may render the system unbootable. You will have to boot the system in emergency mode to fix this file. Ensure that you understand each field in the file for both file system and swap entries.

The format of this file is such that each row is broken out into six columns to identify the required attributes for each file system to be successfully mounted. Here is what the columns contain:

Column 1: Defines the physical or virtual device path where the file system is resident, or its associated UUID or label. There can be entries for network file systems here as well.
Column 2: Identifies the mount point for the file system. For swap partitions, use either “none” or “swap”.
Column 3: Specifies the type of file system such as Ext3, Ext4, XFS, VFAT, or ISO9660. For swap, the type “swap” is used. You may use “auto” instead to leave it up to the *mount* command to determine the type of the file system.
Column 4: Identifies one or more comma-separated options to be used when mounting the file system. See Table 15-3 for a description of some of the options, consult the manual pages of the *mount* command or the *fstab* file for additional options and details.
Column 5: Is used by the *dump* utility to ascertain the file systems that need to be dumped. A value of 0 (or the

absence of this column) disables this check. This field is applicable only on Extended file systems; XFS does not use it.

Column 6: Expresses the sequence number in which to run the *e2fsck* (file system check and repair utility for Extended file system types) utility on the file system at system boot. By default, 0 is used for memory-based, remote, and removable file systems, 1 for */*, and 2 for */boot* and other physical file systems. 0 can also be used for */*, */boot*, and other physical file systems you don't want to be checked or repaired. This field is applicable only on Extended file systems; XFS does not use it.

A 0 in columns 5 and 6 for XFS, virtual, remote, and removable file system types has no meaning. You do not need to add them for these file system types.

This file is edited manually, so care must be observed to circumvent syntax and typing errors.

Monitoring File System Usage

On a live system, you'll often need to check file system usage to know if a mounted file system requires an expansion for growth or a clean up to generate free space. This involves examining the used and available spaces for a file system. The *df* (*disk free*) command has been used for this purpose. It reports usage details for mounted file systems. By default, this command reports the numbers in KBs unless the -m or -h option is specified to view the sizes in MBs or human-readable format.

This command may not produce correct information for VDO and Stratis file systems. Use their own tools for viewing usage.

Let's run this command with the -h option on *server2*:

```
[user1@server2 ~]$ df -h
Filesystem              Size  Used Avail Use% Mounted on
devtmpfs                900M     0  900M   0% /dev
tmpfs                   915M     0  915M   0% /dev/shm
tmpfs                   915M  9.2M  906M   1% /run
tmpfs                   915M     0  915M   0% /sys/fs/cgroup
/dev/mapper/rhel-root   8.0G  3.9G  4.2G  48% /
/dev/sr0                6.7G  6.7G     0 100% /mnt
/dev/sda1              1014M  169M  846M  17% /boot
tmpfs                   183M   28K  183M   1% /run/user/42
tmpfs                   183M  4.0K  183M   1% /run/user/1000
```

The output shows the file system device file or type in column 1, followed by the total, used, and available spaces in columns 2, 3, and 4, and then the usage percentage and mount point in columns 5 and 6.

There are a few other useful flags available with the *df* command that can produce the desired output. These flags include:

-T to add the file system type to the output (example: **df -hT**)
-x to exclude the specified file system type from the output (example: **df -hx tmpfs**)
-t to limit the output to a specific file system type (example: **df -t xfs**)
-i to show inode information (example: **df -hi**)

You may use -h with any of these examples to print information in human-readable format.

Calculating Disk Usage

In contrast to the *df* command that returns usage information for an entire file system, the *du* command reports the amount of space a file or directory occupies. By default, it shows the output in KBs; however, you can use the -m or -h option to view the output in MBs or human-readable format. In addition, you can view a usage summary with the -s switch and a grand total with -c.

Let’s run this command on the */usr/bin* directory to view the usage summary:

```
[user1@server2 ~]$ du -sh /usr/bin
206M    /usr/bin
```

To add a “total” row to the output and with numbers displayed in KBs:

```
[user1@server2 ~]$ du -sc /usr/bin
210540  /usr/bin
210540  total
```

Try this command with different options on the */usr/sbin/lvm* file and observe the results.

Exercise 15-1: Create and Mount Ext4, VFAT, and XFS File Systems in Partitions

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create 2 x 100MB partitions on the */dev/sdb* disk, initialize them separately with the Ext4 and VFAT file system types, define them for persistence using their UUIDs, create mount points called */ext4fs1* and */vfatfs1*, attach them to the directory structure, and verify their availability and usage. Moreover, you will use the disk */dev/sdc* and repeat the above procedure to establish an XFS file system in it and mount it on */xfsfs1*.

1. Apply the label “msdos” to the *sdb* disk using the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdb mklabel msdos
```

2. Create 2 x 100MB primary partitions on *sdb* with the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdb mkpart primary 1 101m
[user1@server2 ~]$ sudo parted /dev/sdb mkpart primary 102 201m
```

3. Initialize the first partition (*sdb1*) with Ext4 file system type using the *mkfs* command:

```
[user1@server2 ~]$ sudo mkfs -t ext4 /dev/sdb1
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 97280 1k blocks and 24384 inodes
Filesystem UUID: 681c0535-4e44-4043-8bef-17413c897f9b
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

4. Initialize the second partition (*sdb2*) with VFAT file system type using the *mkfs* command:

```
[user1@server2 ~]$ sudo mkfs -t vfat /dev/sdb2
mkfs.fat 4.1 (2017-01-24)
```

5. Initialize the whole disk (*sdc*) with the XFS file system type using the *mkfs.xfs* command. Add the -f flag to force the removal of any old partitioning or labeling information from the disk.

```
[user1@server2 ~]$ sudo mkfs.xfs /dev/sdc -f
meta-data=/dev/sdc               isize=512    agcount=4, agsize=16000 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=64000, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

6. Determine the UUIDs for all three file systems using the *lsblk* command:

```
[user1@server2 ~]$ sudo lsblk -f /dev/sdb /dev/sdc
NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sdb
├─sdb1 ext4         681c0535-4e44-4043-8bef-17413c897f9b
└─sdb2 vfat         E4BB-3A5E
sdc    xfs          1ebce60e-d73d-4737-adeb-c1410b2b8c5f
```

7. Open the */etc/fstab* file, go to the end of the file, and append entries for the file systems for persistence using their UUIDs:

```
UUID="681c0535-4e44-4043-8bef-17413c897f9b"  /ext4fs1 ext4 defaults 0 0
UUID="E4BB-3A5E"                             /vfatfs1 vfat defaults 0 0
UUID="1ebce60e-d73d-4737-adeb-c1410b2b8c5f"  /xfsfs1  xfs  defaults 0 0
```

8. Create mount points */ext4fs1*, */vfatfs1*, and */xfsfs1* for the three file systems using the *mkdir* command:

```
[user1@server2 ~]$ sudo mkdir /ext4fs1 /vfatfs1 /xfsfs1
```

9. Mount the new file systems using the *mount* command. This command will fail if there are any invalid or missing information in the file.

```
[user1@server2 ~]$ sudo mount -a
```

10. View the mount and availability status as well as the types of all three file systems using the *df* command:

```
[user1@server2 ~]$ df -hT
........
/dev/sdb1              ext4       88M  1.6M   80M   2% /ext4fs1
/dev/sdb2              vfat       95M     0   95M   0% /vfatfs1
/dev/sdc               xfs       245M   15M  231M   6% /xfsfs1
```

The output verifies the creation and availability status of the three file systems. They are added to the *fstab* file for persistence. A system reboot at this point will remount them automatically. These file systems may now be used to store files.

Exercise 15-2: Create and Mount XFS File System in VDO Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a VDO volume called *vdo1* of logical size 16GB on the *sdf* disk (the actual size of this disks

is 4GB). You will initialize the volume with the XFS file system type, define it for persistence using its device files, create a mount point called */xfsvdo1*, attach it to the directory structure, and verify its availability and usage.

Prior to proceeding, ensure that the steps outlined in Exercise 13-6 for VDO software installation and service startup have been accomplished.

1. Create a VDO volume *vdo1* on the *sdf* disk with a logical size of 16GB and a slab size of 128MB:

```
[user1@server2 ~]$ sudo vdo create --device /dev/sdf --name vdo1 --vdoSlabSize 128
--vdoLogicalSize 16G
Creating VDO vdo1
Starting VDO vdo1
Starting compression on VDO vdo1
VDO instance 2 volume is ready at /dev/mapper/vdo1
```

2. List the new VDO volume using the *vdo* and *lsblk* commands:

```
[user1@server2 ~]$ sudo vdo list
vdo1
[user1@server2 ~]$ lsblk /dev/sdf
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdf      8:80   0   4G  0 disk
└─vdo1 253:2    0  16G  0 vdo
```

The output shows the logical volume size (16GB), type (vdo), and the actual size (4GB) of the underlying disk.

3. Initialize the VDO volume with the XFS file system type with the *mkfs.xfs* command. The VDO volume device file is */dev/mapper/vdo1* as indicated in the output in step 1. Add the -f flag to force the removal of any old partitioning or labeling information from the disk.

```
[user1@server2 ~]$ sudo mkfs.xfs /dev/mapper/vdo1
meta-data=/dev/mapper/vdo1       isize=512    agcount=4, agsize=1048576 blks
         =                       sectsz=4096  attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=4194304, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=4096  sunit=1 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

4. Open the */etc/fstab* file, go to the end of the file, and append the following entry for the file system for persistence using its device file:

```
/dev/mapper/vdo1            /xfsvdo1 xfs  x-systemd.requires=vdo.service 0 0
```

Make sure to include the option **x-systemd.requires=vdo.service** (or try _netdev instead) with the file system entry in the *fstab* file or your system will land into the emergency target on the next reboot. This option holds the *mount* command from mounting this file system until the *vdo.service* service has been operational.

5. Create the mount point */xfsvdo1* using the *mkdir* command:

```
[user1@server2 ~]$ sudo mkdir /xfsvdo1
```

6. Mount the new file system using the *mount* command. This command will fail if there are any invalid or missing information in the file.

```
[user1@server2 ~]$ sudo mount -a
```

The *mount* command with the -a flag is a validation test for the *fstab* file. It should always be executed after updating this file and before rebooting the server to avoid landing the system in an unbootable state.

7. View the mount and availability status as well as the types of the VDO file system using the *lsblk* and *df*

commands:

```
[user1@server2 ~]$ lsblk /dev/sdf
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdf      8:80   0   4G  0 disk
└─vdo1 253:2    0  16G  0 vdo  /xfsvdo1
[user1@server2 ~]$
[user1@server2 ~]$ df -hT /xfsvdo1
Filesystem       Type  Size  Used Avail Use% Mounted on
/dev/mapper/vdo1 xfs    16G  147M   16G   1% /xfsvdo1
```

The *lsblk* command output illustrates the VDO volume name (*vdo1*) , the disk it is located on (*sdf*), the actual (4GB) and logical (16GB) sizes, and the mount point (*/xfsvdo1*) where the file system is connected to the directory structure.

The *df* command shows the logical size of the file system and its usage status, but it does not reveal the underlying disk information. This file system is added to the *fstab* file for persistence, meaning a future system reboot will remount it automatically. This file system may now be used to store files.

Refer to Chapter 13 “Basic Storage Partitioning” for details on VDO.

Exercise 15-3: Create and Mount Ext4 and XFS File Systems in LVM Logical Volumes

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a volume group called *vgfs* comprised of a 160MB physical volume created in a partition on the */dev/sdd* disk. The PE size for the volume group should be set at 16MB. You will create two logical volumes called *ext4vol* and *xfsvol* of sizes 80MB each and initialize them with the Ext4 and XFS file system types. You will ensure that both file systems are persistently defined using their logical volume device filenames. You will create mount points

called */ext4fs2* and */xfsfs2*, mount the file systems, and verify their availability and usage.

1. Create a 150MB partition on the *sdd* disk using the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdd mkpart pri 1 181m
```

2. Initialize the *sdd1* partition for use in LVM using the *pvcreate* command:

```
[user1@server2 ~]$ sudo pvcreate /dev/sdd1
  Physical volume "/dev/sdd1" successfully created.
```

3. Create the volume group *vgfs* with a PE size of 16MB using the physical volume *sdd1*:

```
[user1@server2 ~]$ sudo vgcreate -s 16 vgfs /dev/sdd1
  Volume group "vgfs" successfully created
```

The PE size is not easy to alter after a volume group creation, so ensure it is defined as required at creation.

4. Create two logical volumes *ext4vol* and *xfsvol* of size 80MB each in *vgfs* using the *lvcreate* command:

```
[user1@server2 ~]$ sudo lvcreate -n ext4vol -L 80 vgfs
  Logical volume "ext4vol" created.
[user1@server2 ~]$ sudo lvcreate -n xfsvol -L 80 vgfs
  Logical volume "xfsvol" created.
```

5. Format the *ext4vol* logical volume with the Ext4 file system type using the *mkfs.ext4* command:

```
[user1@server2 ~]$ sudo mkfs.ext4 /dev/vgfs/ext4vol
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 81920 1k blocks and 20480 inodes
Filesystem UUID: 6a30d72f-003b-4797-a7ba-c6f6d36f7a58
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

You may alternatively use **sudo mkfs -t ext4 /dev/vgfs/ext4vol**.

6. Format the *xfsvol* logical volume with the XFS file system type using the *mkfs.xfs* command:

```
[user1@server2 ~]$ sudo mkfs.xfs /dev/vgfs/xfsvol
meta-data=/dev/vgfs/xfsvol       isize=512    agcount=4, agsize=5120 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=20480, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

You may use **sudo mkfs -t xfs /dev/vgfs/xfsvol** instead.

7. Open the */etc/fstab* file, go to the end of the file, and append entries for the file systems for persistence using their device files:

```
/dev/vgfs/ext4vol                          /ext4fs2 ext4 defaults 0 0
/dev/vgfs/xfsvol                           /xfsfs2  xfs  defaults 0 0
```

8. Create mount points */ext4fs2* and */xfsfs2* using the *mkdir* command:

```
[user1@server2 ~]$ sudo mkdir /ext4fs2 /xfsfs2
```

9. Mount the new file systems using the *mount* command. This command will fail if there is any invalid or missing information in the file.

```
[user1@server2 ~]$ sudo mount -a
```

Fix any issues in the file if reported.

10. View the mount and availability status as well as the types of the new LVM file systems using the *lsblk* and *df* commands:

```
[user1@server2 ~]$ lsblk /dev/sdd
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdd                8:48   0  250M  0 disk
└─sdd1             8:49   0  172M  0 part
  ├─vgfs-ext4vol 253:3    0   80M  0 lvm  /ext4fs2
  └─vgfs-xfsvol  253:4    0   80M  0 lvm  /xfsfs2
[user1@server2 ~]$
[user1@server2 ~]$ df -hT | grep fs2
/dev/mapper/vgfs-ext4vol ext4       74M  1.6M   67M   3% /ext4fs2
/dev/mapper/vgfs-xfsvol  xfs        75M  4.9M   70M   7% /xfsfs2
```

The *lsblk* command output illustrates the LVM logical volumes (*ext4vol* and *xfsvol*), the disk they are located on (*sdd*), the sizes (80MB), and the mount points (*/ext4fs2* and */xfsfs2*) where the file system are connected to the directory structure.

The *df* command shows the size and usage information. Both file systems are added to the *fstab* file for persistence, meaning future system reboots will remount them automatically. They may now be used to store files.

Exercise 15-4: Resize Ext4 and XFS File Systems in LVM Logical Volumes

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will grow the size of the *vgfs* volume group that was created in Exercise 15-3 by adding the whole *sde* disk to it. You will extend the *ext4vol* logical volume along with the file system it contains by 40MB using two separate commands. You will extend the *xfsvol* logical volume along with the file system it contains by 40MB using a single command. You will verify the new extensions.

1. Initialize the *sde* disk and add it to the *vgfs* volume group:

```
[user1@server2 ~]$ sudo pvcreate /dev/sde
  Physical volume "/dev/sde" successfully created.
[user1@server2 ~]$ sudo vgextend vgfs /dev/sde
  Volume group "vgfs" successfully extended
```

2. Confirm the new size of *vgfs* using the *vgs* and *vgdisplay* commands:

```
[user1@server2 ~]$ sudo vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  rhel   1   2   0 wz--n-  <9.00g      0
  vgfs   2   2   0 wz--n- 400.00m 240.00m
[user1@server2 ~]$ sudo vgdisplay vgfs
  --- Volume group ---
  VG Name               vgfs
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                2
  Open LV               2
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               400.00 MiB
  PE Size               16.00 MiB
  Total PE              25
  Alloc PE / Size       10 / 160.00 MiB
  Free  PE / Size       15 / 240.00 MiB
  VG UUID               OeTIVf-IByc-gVbN-oW9q-5k8L-sK5A-ixxotw
```

There are now two physical volumes in the volume group and the total size increased to 400MiB.

3. Grow the logical volume *ext4vol* and the file system it holds by 40MB using the *lvextend* and *fsadm* command pair. Make sure to use an uppercase L to specify the size. The default unit is MiB. The plus sign (+) signifies an addition to the current size.

```
[user1@server2 ~]$ sudo lvextend -L +40 /dev/vgfs/ext4vol
  Rounding size to boundary between physical extents: 48.00 MiB.
  Size of logical volume vgfs/ext4vol changed from 80.00 MiB (5 extents) to 128.
00 MiB (8 extents).
  Logical volume vgfs/ext4vol successfully resized.

[user1@server2 ~]$ sudo fsadm resize /dev/vgfs/ext4vol
resize2fs 1.44.3 (10-July-2018)
Filesystem at /dev/mapper/vgfs-ext4vol is mounted on /ext4fs2; on-line resizing
required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mapper/vgfs-ext4vol is now 131072 (1k) blocks long.
```

The *resize* subcommand instructs the *fsadm* command to grow the file system to the full length of the specified logical volume.

4. Grow the logical volume *xfsvol* and the file system (-r) it holds by (+) 40MB using the *lvresize* command:

```
[user1@server2 ~]$ sudo lvresize -r -L +40 /dev/vgfs/xfsvol
  Rounding size to boundary between physical extents: 48.00 MiB.
  Size of logical volume vgfs/xfsvol changed from 80.00 MiB (5 extents) to 128.0
0 MiB (8 extents).
  Logical volume vgfs/xfsvol successfully resized.
meta-data=/dev/mapper/vgfs-xfsvol isize=512    agcount=4, agsize=5120 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=20480, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=1368, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
data blocks changed from 20480 to 32768
```

5. Verify the new extensions to both logical volumes using the *lvs* command. You may also issue the *lvdisplay* or *vgdisplay* command instead.

```
[user1@server2 ~]$ sudo lvs | grep vol
  ext4vol vgfs -wi-ao---- 128.00m

  xfsvol  vgfs -wi-ao---- 128.00m
```

6. Check the new sizes and the current mount status for both file systems using the *df* and *lsblk* commands:

```
[user1@server2 ~]$ df -hT | grep -E 'ext4vol|xfsvol'
/dev/mapper/vgfs-xfsvol  xfs       123M  5.4M  118M   5% /xfsfs2
/dev/mapper/vgfs-ext4vol ext4      120M  1.6M  110M   2% /ext4fs2
[user1@server2 ~]$
[user1@server2 ~]$ lsblk /dev/sdd /dev/sde
NAME              MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdd                 8:48   0  250M  0 disk
└─sdd1              8:49   0  172M  0 part
  ├─vgfs-ext4vol 253:2    0  128M  0 lvm  /ext4fs2
  └─vgfs-xfsvol  253:3    0  128M  0 lvm  /xfsfs2
sde                 8:64   0  250M  0 disk
├─vgfs-ext4vol   253:2    0  128M  0 lvm  /ext4fs2
└─vgfs-xfsvol    253:3    0  128M  0 lvm  /xfsfs2
```

The outputs reflect the new sizes (128MB) for both file systems. They also indicate their mount status.

This concludes the exercise.

Exercise 15-5: Create, Mount, and Expand XFS File System in Stratis Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a Stratis pool called *strpool* and a file system (*strfs2*) in it (the file system type will be XFS) by reusing the 1GB *sdg* disk from Chapter 14. You will display information about the pool, file system, and device used. You will expand the pool to include another 1GB disk *sdh* and confirm.

For further details on Stratis storage management solution, refer to Chapter 14 “Advanced Storage Partitioning”. Prior to proceeding, ensure that the steps outlined in Exercise 14-7 for Stratis software installation and service startup have been accomplished.

1. Create a pool called *strpool* using the *sdg* disk and verify the creation:

```
[user1@server2 ~]$ sudo stratis pool create strpool /dev/sdg
[user1@server2 ~]$ sudo stratis pool list
Name       Total Physical Size  Total Physical Used
strpool                  1 GiB               52 MiB
```

The pool is created with a single disk included, and confirmed.

2. Show the block device used in the pool:

```
[user1@server2 ~]$ sudo stratis blockdev list strpool
Pool Name  Device Node    Physical Size   State  Tier
strpool    /dev/sdg               1 GiB  In-use  Data
```

3. Create a file system called *strfs2* in the pool and verify the creation:

```
[user1@server2 ~]$ sudo stratis filesystem create strpool strfs2
[user1@server2 ~]$ sudo stratis filesystem list
Pool Name  Name    Used     Created            Device                    UUID

strpool    strfs2  546 MiB  Nov 05 2019 08:08  /stratis/strpool/strfs2  439eeef8
45f543788c2bf3f9d6923d92
```

4. Determine the UUID of the new file system to be added to the *fstab* file:

```
[user1@server2 ~]$ sudo lsblk /stratis/strpool/strfs2 -o UUID
UUID
439eeef8-45f5-4378-8c2b-f3f9d6923d92
```

5. Open the */etc/fstab* file, go to the end of the file, and append the following entry for the file system for persistence using the UUID:

```
UUID="439eeef8-45f5-4378-8c2b-f3f9d6923d92"   /strfs2  xfs  x-systemd.requires=s
tratisd.service 0 0
```

Make sure to include the option **x-systemd.requires=stratisd.service** (or try _netdev instead) with the file system entry in the *fstab* file or your system will land into the emergency target on the next reboot. This option holds the *mount* command from mounting this file system until the *stratisd.service* service has been started successfully.

6. Create the mount point */strfs2*:

```
[user1@server2 ~]$ sudo mkdir /strfs2
```

7. Mount the new file system using the *mount* command as follows. This command will fail if there are any invalid or missing information in the file.

```
[user1@server2 ~]$ sudo mount -a
```

8. Check the pool usage:

```
[user1@server2 ~]$ sudo stratis pool list
Name       Total Physical Size  Total Physical Used
strpool                  1 GiB              598 MiB
```

9. Check the file system usage:

```
[user1@server2 ~]$ sudo stratis filesystem list
Pool Name  Name    Used     Created            Device                    UUID

strpool    strfs2  546 MiB  Nov 05 2019 08:08  /stratis/strpool/strfs2  439eeef8
45f543788c2bf3f9d6923d92
```

10. Grow the pool by adding the *sdh* disk to it and confirm the growth:

```
[user1@server2 ~]$ sudo stratis pool add-data strpool /dev/sdh
[user1@server2 ~]$
[user1@server2 ~]$ sudo stratis pool list
Name       Total Physical Size  Total Physical Used
strpool                  2 GiB              602 MiB
```

The file system *strfs2* will automatically expand to take advantage of the additional pool capacity when required.

This concludes the exercise.

Swap and its Management

Physical memory (or main memory) in the system is a finite temporary storage resource employed for loading kernel and running user programs and applications. *Swap space* is an independent region on the physical disk used for holding idle

data until it is needed. The system splits the physical memory into small logical chunks called *pages* and maps their physical locations to virtual locations on the swap to facilitate access by system processors. This physical-to-virtual mapping of pages is stored in a data structure called *page table*, and it is maintained by the kernel.

When a program or process is spawned, it requires space in the physical memory to run and be processed. Although many programs can run concurrently, the physical memory cannot hold all of them at once. The kernel monitors the memory usage. As long as the free memory remains below a high threshold, nothing happens. However, when the free memory falls below that threshold, the system starts moving selected idle pages of data from physical memory to the swap space in an effort to make room to accommodate other programs. This piece in the process is referred to as *page out*. Since the system CPU performs the process execution in a round-robin fashion, when the system needs this paged-out data for execution, the CPU looks for that data in the physical memory and a *page fault* occurs, resulting in moving the pages back to the physical memory from the swap. This return of data to the physical memory is referred to as *page in*. The entire process of paging data out and in is known as *demand paging*.

RHEL systems with less physical memory but high memory requirements can become over busy with paging out and in. When this happens, they do not have enough cycles to carry out other useful tasks, resulting in degraded system performance. The excessive amount of paging that affects the system performance is called *thrashing*.

When thrashing begins, or when the free physical memory falls below a low threshold, the system deactivates idle processes and prevents new processes from being launched.

The idle processes are only reactivated and new processes are only allowed to be started when the system discovers that the available physical memory has climbed above the threshold level and thrashing has ceased.

Determining Current Swap Usage

The size of a swap area should not be less than the amount of physical memory; however, depending on workload requirements, it may be twice the size or larger. It is also not uncommon to see systems with less swap than the actual amount of physical memory. This is especially witnessed on systems with a huge physical memory size.

RHEL offers the *free* command to view memory and swap space utilization. Use this command to view how much physical memory is installed (total), used (used), available (free), used by shared library routines (shared), holding data before it is written to disk (buffers), and used to store frequently accessed data (cached) on the system. The -h flag may be specified with the command to list the values in human-readable format, otherwise -k for KB, -m for MB, -g for GB, and so on are also supported. Add -t with the command to display a line with the “total” at the bottom of the output. Here is a sample output from *server2*:

```
[user1@server2 ~]$ free -ht
               total        used        free      shared  buff/cache   available
Mem:           1.8Gi       895Mi       447Mi        20Mi       486Mi       749Mi
Swap:          1.0Gi          0B       1.0Gi
Total:         2.8Gi       895Mi       1.4Gi
```

The output indicates that the system has 1.8GiB of total memory of which 895MiB is in use and 447MiB is free. It also shows on the same line the current memory usages by temporary (tmpfs) file systems (20MiB) and kernel buffers and page cache (486MiB). It also illustrates an estimate of free memory available to start new processes (749MiB).

On the subsequent row, it reports the total swap space (1.0GiB) configured on the system with a look at used (0 Bytes) and free (1.0GiB) space. The last line prints the combined usage summary of the main memory and swap.

Try **free -hts 3** and **free -htc 2** to refresh the output every three seconds (-s) and to display the output twice (-c).

The *free* command reads memory and swap information from the */proc/meminfo* file to produce the report. The values are shown in KBs by default, and they are slightly off from what is shown in the above screenshot with *free*. Here are the relevant fields from this file:

```
[user1@server2 ~]$ cat /proc/meminfo |grep -E 'Mem|Swap'
MemTotal:        1873316 kB
MemFree:          457848 kB
MemAvailable:     767288 kB
SwapCached:            0 kB
SwapTotal:       1048572 kB
SwapFree:        1048572 kB
```

This data depicts the system's runtime memory and swap usage, as it is located in a virtual file.

Prioritizing Swap Spaces

On many production RHEL servers, you may find multiple swap areas configured and activated to meet the workload demand. The default behavior of RHEL is to use the first activated swap area and move on to the next when the first one is exhausted. The system allows us to prioritize one area over the other by adding the option "pri" to the swap entries in the *fstab* file. This flag supports a value between -2 and 32767 with -2 being the default. A higher value of "pri" sets a higher priority for the corresponding swap region. For swap areas with an identical priority, the system alternates between them.

Swap Administration Commands

In order to create and manage swap spaces on the system, the *mkswap*, *swapon*, and *swapoff* commands are available. Use *mkswap* to initialize a partition for use as a swap space. Once the swap area is ready, you can activate or deactivate it from the command line with the help of the other two commands, or set it up for automatic activation by placing an entry in the *fstab* file. The *fstab* file accepts the swap area's device file, UUID, or label.

Exercise 15-6: Create and Activate Swap in Partition and Logical Volume

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create one swap area in a new 40MB partition called *sdb3* using the *mkswap* command. You will create another swap area in a 140MB logical volume called *swapvol* in *vgfs*. You will add their entries to the */etc/fstab* file for persistence. You will use the UUID and priority 1 for the partition swap and the device file and priority 2 for the logical volume swap. You will activate them and use appropriate tools to validate the activation.

> **EXAM TIP:** Use the lsblk command to determine available disk space.

1. Use the *parted*'s *print* subcommand on the *sdb* disk and the *vgs* command on the *vgfs* volume group to determine available space for a new 40MB partition and a 144MB logical volume:

```
[user1@server2 ~]$ sudo parted /dev/sdb print
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  101MB   99.6MB  primary  ext4
 2      102MB   201MB   99.6MB  primary  fat16
[user1@server2 ~]$ sudo vgs vgfs
  VG   #PV #LV #SN Attr   VSize   VFree
  vgfs   2   2   0 wz--n- 400.00m 144.00m
```

The outputs show 49MB (250MB minus 201MB) free space on the *sdb* disk and 144MB free space in the volume group.

2. Create partition called *sdb3* of size 40MB using the *parted* command:

```
[user1@server2 ~]$ sudo parted /dev/sdb mkpart primary 202 242
```

3. Create logical volume *swapvol* of size 144MB in *vgs* using the *lvcreate* command:

```
[user1@server2 ~]$ sudo lvcreate -L 144 -n swapvol vgfs
  Logical volume "swapvol" created.
```

4. Construct swap structures in *sdb3* and *swapvol* using the *mkswap* command:

```
[user1@server2 ~]$ sudo mkswap /dev/sdb3 ; sudo mkswap /dev/vgfs/swapvol
Setting up swapspace version 1, size = 38 MiB (39841792 bytes)
no label, UUID=94d86394-0eb3-4c52-a643-969ae193e1a9
Setting up swapspace version 1, size = 144 MiB (150990848 bytes)
no label, UUID=7ff47d80-a86c-467c-836d-5835977db11e
```

5. Edit the *fstab* file and add entries for both swap areas for auto-activation on reboots. Obtain the UUID for partition swap with **lsblk -f /dev/sdb3** and use the device file for logical volume. Specify their priorities.

```
UUID="94d86394-0eb3-4c52-a643-969ae193e1a9"  swap  swap  pri=1 0 0
/dev/vgfs/swapvol                           swap  swap  pri=2 0 0
```

EXAM TIP: You will not be given any credit for this work if you forget to add entries to the fstab file.

6. Determine the current amount of swap space on the system using the *swapon* command:

```
[user1@server2 ~]$ sudo swapon
NAME      TYPE       SIZE USED PRIO
/dev/dm-1 partition 1024M   0B   -2
```

There is one 1GB swap area on the system configured at the default priority of -2.

7. Activate the new swap regions using the *swapon* command:

```
[user1@server2 ~]$ sudo swapon -a
```

The command would display errors if there were any issues with swap entries in the */fstab* file.

8. Confirm the activation using the *swapon* command or by viewing the */proc/swaps* file:

```
[user1@server2 ~]$ sudo swapon
NAME       TYPE       SIZE USED PRIO
/dev/dm-1  partition 1024M   0B   -2
/dev/sdb3  partition   38M   0B    1
/dev/dm-11 partition  144M   0B    2
[user1@server2 ~]$
[user1@server2 ~]$ cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       1048572 0       -2
/dev/sdb3                               partition       38908   0       1
/dev/dm-11                              partition       147452  0       2
```

The two new swap regions are activated and listed in the above outputs. Their sizes and priorities are also visible. The device mapper device files for the logical volumes and the device file for the partition swap are also exhibited.

9. Issue the *free* command to view the reflection of swap numbers on the Swap and Total lines:

```
[user1@server2 ~]$ free -ht
              total        used        free      shared  buff/cache   available
Mem:          1.8Gi       889Mi       457Mi        20Mi       483Mi       757Mi
Swap:         1.2Gi          0B       1.2Gi
Total:        3.0Gi       889Mi       1.6Gi
```

The total swap is now 1.2GiB. This concludes the exercise.

Chapter Summary

This chapter covered two major storage topics: file systems and swap. These structures are created in partitions or logical volumes irrespective of the underlying storage management solution used to build them.

The chapter began with a detailed look at the concepts, categories, benefits, and types of file systems. We reviewed file system administration and monitoring utilities. We discussed the concepts around mounting and unmounting file systems. We examined the UUID associated with file systems and applied labels to file systems. We analyzed the file system table and added entries for auto-activating file systems at reboots. We explored tools for reporting file system usage and calculating disk usage. We performed a number of exercises on file system creation and administration in partitions and VDO, LVM, and Stratis volumes to reinforce the concepts and theory learned in this and the last two chapters.

We touched upon the concepts of swapping and paging, and studied how they work. We performed exercises on creating, activating, viewing, deactivating, and removing swap spaces, as well as configuring them for auto-activation at system reboots.

Check Your Understanding

1. Which two file systems are created in a default RHEL 8 installation?
2. What would the command *lvresize -r -L +30 /dev/vg02/lvol2* do?
3. XFS is the default file system type in RHEL 8. True or False?
4. What type of information does the *blkid* command display?

5. What would the command *xfs_admin -L bootfs /dev/sda1* do?
6. The *lsblk* command cannot be used to view file system UUIDs. True or False?
7. What is the process of paging out and paging in known as?
8. What would the command *mkswap /dev/sdc2* do?
9. What would happen if you mount a file system on a directory that already contains files in it?
10. A UUID is always assigned to a file system at its creation time. True or False?
11. Arrange the activities to create and activate a swap space while ensuring persistence: (a) swapon, (b) update fstab, (c) mkswap, and (d) reboot?
12. The difference between the primary and backup superblocks is that the primary superblock includes pointers to the data blocks where the actual file contents are stored whereas the backup superblocks don't. True or False?
13. What would the command *mkfs.ext4 /dev/vgtest/lvoltest* do?
14. Arrange the tasks in correct sequence: umount file system, mount file system, create file system, remove file system.
15. Which of these statements is wrong with respect to file systems: (a) optimize each file system independently, (b) keep dissimilar data in separate file systems, (c) grow and shrink a file system independent of other file systems, and (d) file systems cannot be expanded independent of other file systems.
16. Which command can be used to create a label for an XFS file system?
17. What would the *mount* command do with the -a switch?
18. What would the command *df -t xfs* do?

19. What is the difference between the *mkfs.ext4* and *mke2fs* commands?
20. Which command can be used to determine the total and used physical memory and swap in the system?
21. Which virtual file contains information about the current swap?
22. The */etc/fstab* file can be used to activate swap spaces automatically at system reboots. True or False?
23. What is the default file system type used for optical media?
24. The *xfs_repair* command must be run on a mounted file system. True or False?
25. Provide two commands that can be used to activate and deactivate swap spaces manually.
26. Provide the *fstab* file entry for an Ext4 file system located in device */dev/mapper/vg20-lv1* and mounted with default options on the */ora1* directory.
27. What is the name of the virtual file that holds currently mounted file system information?
28. Which option must be specified with persistent VDO file system mounting to avoid landing the system in an unbootable state?
29. Both Ext3 and Ext4 file system types support journaling. True or False?
30. Name three commands that can be employed to view the UUID of an XFS file system?

Answers to Check Your Understanding

1. */* and */boot*.
2. The command provided will expand the logical volume *lvol2* in volume group *vg02* along with the file system it contains by 30MB.
3. True.

4. The *blkid* command displays attributes for block devices.
5. The command provided will apply the specified label to the XFS file system in */dev/sda1*.
6. False.
7. The process of paging out and in is known as demand paging.
8. The command provided will create swap structures in the */dev/vdc2* partition.
9. The files in the directory will hide.
10. True.
11. c/a/b or c/b/a.
12. False.
13. The command provided will format */dev/vgtest/lvoltest* logical volume with Ext4 file system type.
14. Create, mount, unmount, and remove.
15. d is incorrect.
16. The *xfs_admin* command can be used to create a label for an XFS file system.
17. The command provided will mount all file systems listed in the */etc/fstab* file but are not currently mounted.
18. The command provided will display all mounted file systems of type XFS.
19. No difference.
20. The *free* command.
21. The */proc/swaps* file contains information about the current swap.
22. True.
23. The default file system type for optical devices is ISO9660.
24. False.
25. The *swapon* and *swapoff* commands.
26. /dev/mapper/vg20-lv1 /ora1 swap defaults 0 0
27. The *mounts* file under */proc/self* directory.
28. The _netdev option.

29. True.
30. You can use the *xfs_admin*, *lsblk*, and *blkid* commands to view the UUID of an XFS file system.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

To use the same storage that was used in the labs for Chapters 13 and 14, destroy all traces of the partitions, volume groups, and pools that were created before proceeding. Alternatively, you can add more storage to *server2* for the following labs.

Lab 15-1: Create VFAT, Ext4, and XFS File Systems in Partitions and Mount Persistently

As *user1* with *sudo* on *server2*, create three 70MB primary partitions on one of the available 250MB disks (*lsblk*) by invoking the *parted* utility directly at the command prompt. Apply label “msdos” if the disk is new. Initialize partition 1 with VFAT, partition 2 with Ext4, and partition 3 with XFS file system types. Create mount points */vfatfs5*, */ext4fs5*, and */xfsfs5*, and mount all three manually. Determine the UUIDs for the three file systems, and add them to the *fstab* file. Unmount all three file systems manually, and execute **mount -a** to mount them all. Run **df -h** for verification. (Hint: File System Management).

Lab 15-2: Create XFS File System in VDO Volume and Mount Persistently

As *user1* with *sudo* on *server2*, ensure that VDO software is installed and the VDO service is enabled and started. Create a volume *vdo5* with a logical size 16GB on an available 4GB disk (*lsblk*) using the *vdo* command. Select an appropriate slab size for the volume. Verify the volume creation with the *vdo*, *lsblk*, and *vdostats* commands. Initialize the volume with XFS file system type. Create mount point */vdofs5*, and mount it manually. Add the file system information to the *fstab* file, and use "_netdev" as a mount option. Unmount the file system manually, and execute **mount -a** to mount it back. Run **df -h** to confirm. (Hint: File System Management).

Lab 15-3: Create Ext4 and XFS File Systems in LVM Logical Volumes and Mount Persistently

As *user1* with *sudo* on *server2*, initialize an available 250MB disk for use in LVM (*lsblk*). Create volume group *vg200* with PE size 8MB and add the physical volume. Create two logical volumes *lv200* and *lv300* of sizes 120MB and 100MB. Use the *vgs*, *pvs*, *lvs*, and *vgdisplay* commands for verification. Initialize the volumes with Ext4 and XFS file system types. Create mount points */lvmfs5* and */lvmfs6*, and mount them manually. Add the file system information to the *fstab* file using their device files. Unmount the file systems manually, and execute **mount -a** to mount them back. Run **df -h** to confirm. (Hint: File System Management).

Lab 15-4: Extend Ext4 and XFS File Systems in LVM Logical Volumes

As *user1* with *sudo* on *server2*, initialize an available 250MB disk for use in LVM (*lsblk*). Add the new physical volume to volume group *vg200*. Expand logical volumes *lv200* and

lv300 along with the underlying file systems to 200MB and 250MB. Use the *vgs*, *pvs*, *lvs*, *vgdisplay*, and *df* commands for verification. (Hint: File System Management).

Lab 15-5: Create XFS File System in Stratis Volume and Mount Persistently

As *user1* with *sudo* on *server2*, confirm Stratis software is installed and the Stratis service is enabled and started. Create pool *strpool5* on an available 1GB disk and confirm. Create file system *strfs5* in the pool and verify. Create mount point */strfs5*. Determine the UUID of the Stratis file system and add it to the *fstab* file for persistence. Use the x-systemd.requires=stratisd.service as a mount option. Reboot the system and confirm mounting with **df -h**. (Hint: File System Management).

Lab 15-6: Create Swap in Partition and LVM Logical Volume and Activate Persistently

As *user1* with *sudo* on *server2*, create two 100MB partitions on an available 250MB disk (*lsblk*) by invoking the *parted* utility directly at the command prompt. Apply label “msdos” if the disk is new. Initialize one of the partitions with swap structures. Apply label *swappart* to the swap partition, and add it to the *fstab* file. Execute **swapon -a** to activate it. Run **swapon -s** to confirm activation.

Initialize the other partition for use in LVM. Expand volume group *vg200* (Lab 15-4) by adding this physical volume. Create logical volume *swapvol* of size 180MB. Use the *vgs*, *pvs*, *lvs*, and *vgdisplay* commands for verification. Initialize the logical volume for swap. Add an entry to the *fstab* file for the new swap area using its device file. Execute **swapon -a** to activate it. Run **swapon -s** to confirm activation. (Hint: Swap and its Management).

Chapter 16

Remote File System

This chapter describes the following major topics:

- Overview of Network File System service and key components
- Network File System benefits and versions
- Export a share on NFS server
- Mount the share on NFS client using standard mount method
- Understand the AutoFS service, and its benefits and functioning
- Analyze AutoFS configuration maps
- Mount the exported share on NFS client using AutoFS
- Configure NFS and AutoFS to share and mount user home directories

RHCSA Objectives:

29. Mount and unmount network file systems using NFS

Remote shares may be mounted on RHEL and accessed the same way as local file systems. This can be done manually using the same tools that are employed for mounting and unmounting local file systems. An alternative solution is to implement the AutoFS service to automatically mount and unmount them without the need to execute any commands explicitly. AutoFS monitors activities in mount points based on which it triggers a mount or unmount action.

RHEL exports shares using the Network File System service for remote mounting on clients. The combination of NFS and AutoFS/standard mount is prevalent in real world scenarios. This chapter elaborates on the benefits of the file sharing solution and expounds upon AutoFS maps. It demonstrates a series of exercises to detail the configuration of the NFS server, NFS client, and AutoFS service. It also explores the automatic mounting of user home directories on clients.

Network File System

Network File System (NFS) is a networking protocol that allows file sharing over the network. The Network File System service is based upon the client/server architecture whereby users on one system access files, directories, and file systems (collectively called “shares”) that reside on a remote system as if they are mounted locally on their system. The remote system that makes its shares available for network access is referred to as an NFS server, and the process of making the shares accessible is referred to as *exporting*. The shares the NFS server exports may be accessed by one or more systems. These systems are called NFS clients, and the process of making the shares accessible on clients is referred to as

mounting. See Figure 16-1 for a simple NFS client/server arrangement that shows two shares—*/export1* and */export2*—exported on the network to a remote system, which has them mounted there.

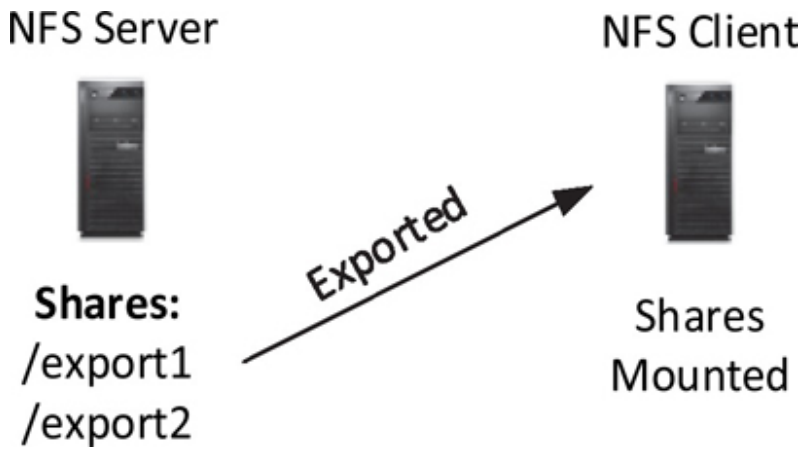


Figure 16-1 NFS Server/Client

A system can provide both server and client functionality concurrently. When a directory or file system share is exported, the entire directory structure beneath it becomes available for mounting on the client. A subdirectory or the parent directory of a share cannot be re-exported if it exists in the same file system. Similarly, a mounted share cannot be exported further. A single exported file share is mounted on a directory mount point.

Benefits of Using NFS

The use of NFS provides several benefits, some of which are highlighted below:

- ✓ Supports a variety of operating system platforms including Linux, UNIX, and Microsoft Windows.
- ✓ Multiple NFS clients can access a single share simultaneously.
- ✓ Enables the sharing of common application binaries and other read-only information, resulting in reduced

administration overhead and storage cost.

- ✓ Gives users access to uniform data.
- ✓ Allows the consolidation of scattered user home directories on the NFS server and then exporting them to the clients. This way users will have only one home directory to maintain.

NFS Versions

RHEL 8 provides the support for NFS versions 3, 4.0, 4.1, and 4.2, with version 4.2 being the default. NFSv3 supports both TCP and UDP transport protocols, asynchronous writes, and 64-bit file sizes that gives clients the ability to access files of sizes larger than 2GB.

NFSv4.x are *Internet Engineering Task Force* (IETF) series of protocols that provide all the features of NFSv3, plus the ability to transit firewalls and work on the Internet. They provide enhanced security and support for encrypted transfers and ACLs, as well as greater scalability, better cross-platform interoperability, and better system crash handling. They use usernames and group names rather than UIDs and GIDs for files located on network shares. NFSv4.0 and NFSv4.1 use the TCP protocol by default, but can work with UDP for backward compatibility. In contrast, NFSv4.2 only supports TCP.

NFS Server and Client Configuration

This section presents two exercises, one demonstrating how to export a share on a server (NFS server) and the other outlines the steps on mounting and accessing a share on a remote system (NFS client). The basic setup of the NFS service is straightforward. It requires adding an entry of the share to a file called */etc/exports* and using a command called *exportfs* to make it available on the network. It also requires

the addition of a firewall rule to allow access to the share by NFS clients.

The *mount* command employed on an NFS client is the same command that was used in Chapter 15 to mount local file systems. Moreover, the *fstab* file requires an entry for the NFS share on the client for persistent mounting.

The exercises in this section illustrate the usage of both commands and the syntax of both files.

Exercise 16-1: Export Share on NFS Server

This exercise should be done on *server2* as *user1* with *sudo* where required.

In this exercise, you will create a directory called */common* and export it to *server1* in read/write mode. You will ensure that NFS traffic is allowed through the firewall. You will confirm the export.

1. Install the NFS software called *nfs-utils*:

```
[user1@server2 ~]$ sudo dnf -y install nfs-utils
Package nfs-utils-1:2.3.3-14.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

2. Create */common* directory to be exported as a share:

```
[user1@server2 ~]$ sudo mkdir /common
```

3. Add full permissions on */common*:

```
[user1@server2 ~]$ sudo chmod 755 /common
```

4. Add the NFS service persistently to the firewalld configuration to allow the NFS traffic to pass through, and load the new rule:

```
[user1@server2 ~]$ sudo firewall-cmd --permanent --add-service nfs
success
[user1@server2 ~]$ sudo firewall-cmd --reload
success
```

Refer to Chapter 20 for details around the firewall service.

5. Start the NFS service and enable it to autostart at system reboots:

```
[user1@server2 ~]$ sudo systemctl --now enable nfs-server
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-server.service
 /usr/lib/systemd/system/nfs-server.service.
```

6. Verify the operational status of the NFS services:

```
[user1@server2 ~]$ sudo systemctl status nfs-server
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor
   Active: active (exited) since Sun 2019-11-10 08:32:21 EST; 1min 26s ago
  Process: 15800 ExecStart=/bin/sh -c if systemctl -q is-active gssproxy; then
  Process: 15786 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
  Process: 15785 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCC
 Main PID: 15800 (code=exited, status=0/SUCCESS)

Nov 10 08:32:21 server2.example.com systemd[1]: Starting NFS server and service
Nov 10 08:32:21 server2.example.com systemd[1]: Started NFS server and services.
```

7. Open the */etc/exports* file in a text editor and add an entry for */common* to export it to *server1* with read/write option:

```
/common 192.168.0.110(rw)
```

8. Export the entry defined in the */etc/exports* file. The -a option exports all the entries listed in the file and -v displays verbose output.

```
[user1@server2 ~]$ sudo exportfs -av
exporting 192.168.0.110:/common
```

The NFS service is now set up on *server1* with the */common* share available for mounting on *server1* (NFS client in this case).

For practice, you can unexport the share by issuing the *exportfs* command with the -u flag as follows:

```
[user1@server2 ~]$ sudo exportfs -u 192.168.0.110:/common
```

Before proceeding, re-export the share using **sudo exportfs -av**.

Exercise 16-2: Mount Share on NFS Client

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will mount the */common* share exported in Exercise 16-1. You will create a mount point called */local*, mount the remote share manually, and confirm the mount. You will add the remote share to the file system table for persistence. You will remount the share and confirm the mount. You will create a test file in the mount point and confirm the file creation on the NFS server (*server2*).

1. Install the NFS software called *nfs-utils*:

```
[user1@server2 ~]$ sudo dnf -y install nfs-utils
Package nfs-utils-1:2.3.3-14.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

2. Create */local* mount point:

```
[user1@server1 ~]$ sudo mkdir /local
```

3. Mount the share manually using the *mount* command:

```
[user1@server1 ~]$ sudo mount 192.168.0.120:/common /local
```

The remote share is successfully mounted on *server1*, and it can be accessed as any other local file system.

4. Confirm the mount using either the *mount* or the *df* command:

```
[user1@server1 ~]$ mount | grep local
192.168.0.120:/common on /local type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsi
ze=262144,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.1
68.0.110,local_lock=none,addr=192.168.0.120)
[user1@server1 ~]$
[user1@server1 ~]$ df -h | grep local
192.168.0.120:/common  8.0G  3.9G  4.2G  49% /local
```

The *mount* command output returns the NFS protocol version (NFS4) and all the options used in mounting the share.

5. Open the */etc/fstab* file and append the following entry for persistence:

```
192.168.0.120:/common  /local  nfs  _netdev  0  0
```

The _netdev option will cause the system to wait for networking to establish before attempting to mount this share.

A mount point should be empty and must not be in use when an attempt is made to mount a share on it.

6. Unmount the share manually using the *umount* command and remount it via the *fstab* file to validate the accuracy of the entry placed in the file:

```
[user1@server1 ~]$ sudo umount /local
[user1@server1 ~]$ sudo mount -a
```

7. Run **mount** and **df -h** again to verify the remounting.
8. Create a file called *nfsfile* under */local* and verify:

```
[user1@server1 ~]$ touch /local/nfsfile
[user1@server1 ~]$ ls -l /local
total 0
-rw-rw-r--. 1 user1 user1 0 Nov 10 10:10 nfsfile
```

9. Confirm the file creation on the NFS server (*server2*):

```
[user1@server2 ~]$ ls -l /common
total 0
-rw-rw-r--. 1 user1 user1 0 Nov 10 10:10 nfsfile
```

EXAM TIP: Do not forget to update the /etc/fstab file on the client.

This completes the setup and testing of mounting, unmounting, and remounting of an NFS share on the client.

Auto File System (AutoFS)

In the previous section, you learned how to attach (mount) an NFS share to the Linux directory tree manually for access by users and applications on an NFS client. Once attached, the share was treated just like any other local file system. You also learned how to detach (unmount) an NFS share manually from the directory tree to make it inaccessible to users and applications. You placed an entry for the share in the *fstab* file to guarantee a remount during system reboots.

RHEL offers an alternative way of mounting and unmounting a share on the clients during runtime as well as system reboots. This feature is delivered by a service called the *Auto File System* (AutoFS). AutoFS is a client-side service, which is employed to mount an NFS share on-demand. With a proper entry placed in AutoFS configuration files, the AutoFS service automatically mounts a share upon detecting an activity in its mount point with a command such as *ls* or *cd*. In the same manner, AutoFS unmounts the share automatically if it has not been accessed for a predefined period of time.

To avoid inconsistencies, mounts managed with AutoFS should not be mounted or unmounted manually or via the /etc/fstab file.

The use of AutoFS saves the kernel from dedicating system resources to maintain unused NFS shares, ultimately contributing to the overall system performance.

Benefits of Using AutoFS

There are several benefits associated with using the AutoFS service over placing entries in the */etc/fstab* file. Some of the

key benefits are described below:

- AutoFS requires that NFS shares be defined in text configuration files called *maps*, which are located in the */etc* or */etc/auto.master.d* directory. AutoFS does not make use of the */etc/fstab* file.
- AutoFS does not require *root* privileges to mount an NFS share; manual mounting and mounting via *fstab* do require that privilege.
- AutoFS prevents an NFS client from hanging if an NFS server is down or inaccessible. With the other method, the unavailability of the NFS server may cause the NFS client to hang.
- With AutoFS, a share is unmounted automatically if it is not accessed for five minutes by default. With the *fstab* method, the share stays mounted until it is either manually unmounted or the client shuts down.
- AutoFS supports wildcard characters and environment variables, which the other method does not support.

How AutoFS Works

The AutoFS service consists of a daemon called *automount* in the userland that mounts configured shares automatically upon access. This daemon is invoked at system boot. It reads the AutoFS master map and creates initial mount point entries, but it does not mount any shares yet. When the service detects a user activity under a mount point during runtime, it mounts the requested file system at that time. If a share remains idle for a certain time period, *automount* unmounts it by itself.

AutoFS Configuration File

The configuration file for the AutoFS service is */etc/autofs.conf*, which AutoFS consults at service startup. Some key directives from this file are shown below along with preset values:

```
master_map_name = auto.master
timeout = 300
negative_timeout = 60
mount_nfs_default_protocol = 4
logging = none
```

There are additional directives available in this file and more can be added to modify the default behavior of the AutoFS service. Table 16-1 describes the above directives.

Directive	Description
master_map_name	Defines the name of the master map. The default is auto.master located in the /etc directory.
timeout	Specifies, in seconds, the maximum idle time after which a share is automatically unmounted. The default is five minutes.
negative_timeout	Expresses, in seconds, a timeout value for failed mount attempts. The default is one minute.
mount_nfs_default_protocol	Sets the default NFS version to be used to mount shares.
logging	Configures a logging level. Options are none, verbose, and debug. The default is none (disabled).

Table 16-1 AutoFS Directives

The directives in the *autofs.conf* file are normally left to their default values, but you can alter them if required.

AutoFS Maps

The AutoFS service needs to know the NFS shares to be mounted and their locations. It also needs to know any specific options to use with mounting them. This information is defined in AutoFS files called *maps*. There are three common AutoFS map types: *master*, *direct*, and *indirect*.

The Master Map

The *auto.master* file located in the */etc* directory is the default master map, as defined in the */etc/autofs.conf* configuration file with the master_map_name directive. This map may be used to define entries for indirect and direct maps. However, it is recommended to store user-defined map files in the */etc/auto.master.d* directory, which the AutoFS service automatically parses at startup. The following presents two samples to explain the format of the map entries:

```
/-          /etc/auto.master.d/auto.direct
/misc       /etc/auto.misc
```

Line 1 defines a direct map and points to the */etc/auto.master.d/auto.direct* file for mount details.

The second one is for an indirect map, notifying AutoFS to refer to the */etc/auto.misc* file for mount details. The umbrella mount point */misc* will precede all mount point entries listed in the */etc/auto.misc* file. This indirect map entry is normally used to automount removable file systems, such as CD, DVD, external USB disks, and so on. Any custom indirect map file should be located in the */etc/auto.master.d* directory.

You may append an option to either entry in the *auto.master* file; however, that option will apply globally to all subentries in the specified map file.

The Direct Map

The direct map is used to mount shares automatically on any number of unrelated mount points. Some key points to note when working with direct maps are:

- ✓ Direct mounted shares are always visible to users.
- ✓ Local and direct mounted shares can coexist under one parent directory.

- ✔ Accessing a directory containing many direct mount points mounts all shares.

Each direct map entry places a separate share entry to the */etc/mtab* file, which maintains a list of all mounted file systems whether they are local or remote. This file is updated whenever a local file system, removable file system, or a remote share is mounted or unmounted.

Exercise 16-3: Access NFS Share Using Direct Map

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will configure a direct map to automount the NFS share */common* that is available from *server2*. You will install the relevant software, create a local mount point */autodir*, and set up AutoFS maps to support the automatic mounting. Note that */common* is already mounted on the */local* mount point on *server1* (NFS client) via the *fstab* file. There should'nt be any conflict in configuration or functionality between the two.

1. Install the AutoFS software package called *autofs*:

```
[user1@server1 ~]$ sudo dnf install -y autofs
  ........
Running transaction
  Preparing        :                                                        1/1
  Installing       : autofs-1:5.1.4-29.el8.x86_64                           1/1
  Running scriptlet: autofs-1:5.1.4-29.el8.x86_64                           1/1
  Verifying        : autofs-1:5.1.4-29.el8.x86_64                           1/1
2019-11-10 12:25:30,057 [WARNING] dnf:7057:MainThread @logutil.py:142 - logging already initialize
Installed products updated.

Installed:
  autofs-1:5.1.4-29.el8.x86_64

Complete!
```

2. Create a mount point */autodir* using the *mkdir* command:

```
[user1@server1 ~]$ sudo mkdir /autodir
```

3. Edit the */etc/auto.master* file and add the following entry at the beginning of the file. This entry will point the AutoFS service to the *auto.dir* file for additional information.

```
/-  /etc/auto.master.d/auto.dir
```

4. Create */etc/auto.master.d/auto.dir* file and add the mount point, NFS server, and share information to it:

```
/autodir  192.168.0.120:/common
```

5. Start the AutoFS service now and set it to autostart at system reboots:

```
[user1@server1 ~]$ sudo systemctl enable --now autofs
Created symlink /etc/systemd/system/multi-user.target.wants/autofs.service → /usr/lib/systemd/syste
m/autofs.service.
```

6. Verify the operational status of the AutoFS service. Use the -l and --no-pager options to show full details without piping the output to a pager program (the *pg* command in this case).

```
[user1@server1 ~]$ sudo systemctl status autofs -l --no-pager
● autofs.service - Automounts filesystems on demand
   Loaded: loaded (/usr/lib/systemd/system/autofs.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2019-11-10 19:08:43 EST; 1min 55s ago
 Main PID: 7790 (automount)
    Tasks: 6 (limit: 5076)
   Memory: 4.2M
   CGroup: /system.slice/autofs.service
           └─7790 /usr/sbin/automount --foreground --dont-check-daemon

Nov 10 19:08:42 server1.example.com systemd[1]: Starting Automounts filesystems on demand...
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com systemd[1]: Started Automounts filesystems on demand.
```

7. Run the *ls* command on the mount point */autodir* and then issue the *mount* command to verify that the share is automounted and accessible:

```
[user1@server1 ~]$ ls /autodir
nfsfile
[user1@server1 ~]$ mount | grep autodir
/etc/auto.master.d/auto.dir on /autodir type autofs (rw,relatime,fd=5,pgrp=7790,timeout=300,minprot
o=5,maxproto=5,direct,pipe_ino=48296)
192.168.0.120:/common on /autodir type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsize=262144,namlen=
255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.168.0.110,local_lock=none,addr=192.16
8.0.120)
```

Observe the above outcomes. The *mount* command output depicts the path of the AutoFS map (*/etc/auto.master.d/auto.dir*), the file system type (autofs), and the options used during the mount process. An activity in the mount point (the *ls* command) caused AutoFS to mount the share */common* on */autodir*. Wait five minutes and run the *mount* command again. You'll notice that the auto file system has disappeared. A *cd*, *ls*, or some other activity in the mount point will bring it back.

This completes the AutoFS setup for an NFS share on the client using a direct map.

The Indirect Map

The indirect map is preferred over the direct map if you want to mount all of the shares under one common parent directory. Some key points to note when working with indirect maps are:

- ✓ Indirect mounted shares become visible only after they have been accessed.
- ✓ Local and indirect mounted shares cannot coexist under the same parent directory.
- ✓ Each indirect map puts only one entry in the */etc/mtab* mount table.
- ✓ Accessing a directory containing many indirect mount points shows only the shares that are already mounted.

Both direct and indirect maps have their own merits and demerits. By comparing their features, it seems more prudent to use the indirect map for automounting NFS shares. However, this statement may not be true for every

environment, as there could be specifics that would dictate which option to go with.

Exercise 16-4: Access NFS Share Using Indirect Map

This exercise should be done on *server1* as *user1* with *sudo* where required.

In this exercise, you will configure an indirect map to automount the NFS share */common* that is available from *server2*. You will install the relevant software and set up AutoFS maps to support the automatic mounting. You will observe that the specified mount point "autoindir" is created automatically under */misc*.

Note that */common* is already mounted on the */local* mount point via the *fstab* file and it is also configured via a direct map for automounting on */autodir*. There should occur no conflict in configuration or functionality among the three.

1. Install the AutoFS software package called *autofs*:

```
[user1@server1 ~]$ sudo dnf install -y autofs
........
Package autofs-1:5.1.4-29.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

2. Confirm the entry for the indirect map */misc* in the */etc/auto.master* file exists:

```
[user1@server1 ~]$ grep ^/misc /etc/auto.master
/misc   /etc/auto.misc
```

3. Edit the */etc/auto.misc* file and add the mount point, NFS server, and share information to it:

```
autoindir  192.168.0.120:/common
```

4. Start the AutoFS service now and set it to autostart at system reboots:

```
[user1@server1 ~]$ sudo systemctl enable --now autofs
Created symlink /etc/systemd/system/multi-user.target.wants/autofs.service → /usr/lib/systemd/syste
m/autofs.service.
```

5. Verify the operational status of the AutoFS service. Use the -l and --no-pager options to show full details without piping the output to a pager program (the *pg* command in this case):

```
[user1@server1 ~]$ sudo systemctl status autofs -l --no-pager
● autofs.service - Automounts filesystems on demand
   Loaded: loaded (/usr/lib/systemd/system/autofs.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2019-11-10 19:08:43 EST; 1min 55s ago
 Main PID: 7790 (automount)
    Tasks: 6 (limit: 5076)
   Memory: 4.2M
   CGroup: /system.slice/autofs.service
           └─7790 /usr/sbin/automount --foreground --dont-check-daemon

Nov 10 19:08:42 server1.example.com systemd[1]: Starting Automounts filesystems on demand...
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com automount[7790]: setautomntent: lookup(sss): setautomntent: No
such file or directory
Nov 10 19:08:43 server1.example.com systemd[1]: Started Automounts filesystems on demand.
```

6. Run the *ls* command on the mount point */misc/autoindir* and then *grep* for *auto.misc* or *autoindir* on the *mount* command output to verify that the share is automounted and accessible:

```
[user1@server1 ~]$ ls /misc/autoindir
nfsfile
[user1@server1 ~]$ mount | egrep 'auto.misc|autoindir'
/etc/auto.misc on /misc type autofs (rw,relatime,fd=10,pgrp=7790,timeout=300,minproto=5,maxproto=5,
indirect,pipe_ino=48333)
192.168.0.120:/common on /misc/autoindir type nfs4 (rw,relatime,vers=4.2,rsize=262144,wsize=262144,
namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=192.168.0.110,local_lock=none,addr
=192.168.0.120)
```

Observe the above outcomes. The *mount* command output illustrates the path of the AutoFS map (*/etc/auto.misc*), the auto-generated mount point (*/misc/autoindir*), file system type (autofs), and the options used during the mount process. An activity in the mount point (*ls* command in this case) caused AutoFS to mount the share */common* on */misc/autoindir*. You can use the same umbrella mount point */misc* to mount additional auto-generated mount points.

This mount point will automatically disappear after five minutes of idling. You can verify that by issuing the *mount* command again. A *cd*, *ls*, or some other activity in the mount point will bring it back.

This completes the AutoFS setup for an NFS share on the client using an indirect map.

Automounting User Home Directories

AutoFS allows us to automount user home directories by exploiting two special characters in indirect maps. The asterisk (*) replaces the references to specific mount points and the ampersand (&) substitutes the references to NFS servers and shared subdirectories. With user home directories located under */home*, on one or more NFS servers, the AutoFS service will connect with all of them simultaneously when a user attempts to log on to a client. The service will mount only that specific user's home directory rather than the entire */home*. The indirect map entry for this type of substitution is defined in an indirect map, such as */etc/auto.master.d/auto.home*, and will look like:

```
*  -rw  &:/home/&
```

With this entry in place, there is no need to update any AutoFS configuration files if NFS servers with */home* shared are added or removed. Similarly, if user home directories are added or deleted, there will be no impact on the functionality of AutoFS. If there is only one NFS server sharing the home directories, you can simply specify its name in lieu of the first & symbol in the above entry.

Exercise 16-5: Automount User Home Directories Using Indirect Map

There are two portions for this exercise. The first portion should be done on *server2* (NFS server) and the second

portion on *server1* (NFS client) as *user1* with *sudo* where required.

In the first portion, you will create a user account called *user30* with UID 3000. You will add the */home* directory to the list of NFS shares so that it becomes available for remote mount.

In the second portion, you will create a user account called *user30* with UID 3000, base directory */nfshome*, and no user home directory. You will create an umbrella mount point called */nfshome* for mounting the user home directory from the NFS server. You will install the relevant software and establish an indirect map to automount the remote home directory of *user30* under */nfshome*. You will observe that the home directory of *user30* is automounted under */nfshome* when you sign in as *user30*.

On NFS server *server2*:

1. Create a user account called *user30* with UID 3000 (-u) and assign password “password1”:

```
[user1@server2 ~]$ sudo useradd -u 3000 user30
[user1@server2 ~]$ echo password1 | sudo passwd --stdin user30
Changing password for user user30.
passwd: all authentication tokens updated successfully.
```

2. Edit the */etc/exports* file and add an entry for */home*:

```
/home 192.168.0.110(rw)
```

3. Export all the shares listed in the */etc/exports* file:

```
[user1@server2 ~]$ sudo exportfs -avr
exporting 192.168.0.110:/home
exporting 192.168.0.110:/common
```

On NFS client *server1*:

1. Install the AutoFS software package called *autofs*:

```
[user1@server1 ~]$ sudo dnf install -y autofs
........
Package autofs-1:5.1.4-29.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

2. Create a user account called *user30* with UID 3000 (-u), base home directory location */nfshome* (-b), no home directory (-M), and password “password1”:

```
[user1@server1 ~]$ sudo useradd -u 3000 -Mb /nfshome user30
[user1@server1 ~]$ echo password1 | sudo passwd --stdin user30
Changing password for user user30.
passwd: all authentication tokens updated successfully.
```

This is to ensure that the UID for the user is consistent on the server and the client to avoid access issues.

3. Create the umbrella mount point */nfshome* to automount the user’s home directory:

```
[user1@server1 ~]$ sudo mkdir /nfshome
```

4. Edit the */etc/auto.master* file and add the mount point and indirect map location to it:

```
 /nfshome  /etc/auto.master.d/auto.home
```

5. Create the */etc/auto.master.d/auto.home* file and add the following information to it:

```
 *  -rw  192.168.0.120:/home/user30
```

For multiple user setup, you can replace “user30” with the & character, but ensure that those users exist on both the server and the client with consistent UIDs.

6. Start the AutoFS service now and set it to autostart at system reboots. This step is not required if AutoFS is already running and enabled.

```
[user1@server1 ~]$ sudo systemctl --now enable autofs
Created symlink /etc/systemd/system/multi-user.target.wants/autofs.service → /usr/lib/systemd/syste
m/autofs.service.
```

7. Verify the operational status of the AutoFS service. Use the -l and --no-pager options to show full details without piping the output to a pager program (the *pg* command):

```
[user1@server1 ~]$ sudo systemctl status autofs -l --no-pager
● autofs.service - Automounts filesystems on demand
   Loaded: loaded (/usr/lib/systemd/system/autofs.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2019-11-11 07:37:25 EST; 13s ago
 Main PID: 2440 (automount)
    Tasks: 7 (limit: 5076)
   Memory: 6.1M
   CGroup: /system.slice/autofs.service
           └─2440 /usr/sbin/automount --foreground --dont-check-daemon

Nov 11 07:37:25 server1.example.com systemd[1]: Starting Automounts filesystems on demand...
```

8. Log in as *user30* and run the *pwd*, *ls*, and *df* commands for verification:

```
[user1@server1 ~]$ sudo su - user30
[user30@server1 ~]$ pwd
/nfshome/user30
[user30@server1 ~]$ ls -l
total 0
[user30@server1 ~]$ df -h .
Filesystem                  Size  Used Avail Use% Mounted on
192.168.0.120:/home/user30  8.0G  3.9G  4.1G  49% /nfshome/user30
```

The user is successfully logged in with their home directory automounted from the NFS server. The *pwd* command confirms the path. The *df* command verifies the NFS server IP and the source home directory location for *user30*, as well as the mount location. You can also use the *mount* command and pipe the output to *grep* for *user30* to view mount details (**mount | grep user30**).

> **EXAM TIP:** You may need to configure AutoFS for mounting a user home directory.

This completes the setup for an automounted home directory share for a user.

Chapter Summary

This chapter discussed the sharing and mounting of remote file systems using the Network File System protocol. It elucidated the concepts, benefits, and versions of the NFS service, and described the commands and configuration files involved in NFS management on the server and the client.

Next, we performed an exercise to demonstrate the configuration and sharing of a directory on one of the lab servers (NFS server) and another exercise on the second lab system (NFS client) to mount that share manually and persistently using the standard NFS mount method.

We explored the client-side service called AutoFS for automounting NFS shares. We discussed the concepts, benefits, and components associated with AutoFS, and analyzed its maps. We performed exercises to mount, confirm, and unmount the remote NFS share using both direct and indirect methods.

Finally, we described the AutoFS setting to automount user home directories from the NFS server.

Check Your Understanding

1. What would the entry * *server10:/home/&* in an AutoFS indirect map imply?
2. Which command is used to export a share?
3. An NFS server exports a share and an NFS server mounts it. True or False?
4. Which command would you use to unexport a share?
5. What is the name of the NFS server configuration file?
6. What would the line entry */dir1 *(rw)* in the */etc/exports* file mean?
7. What type of AutoFS map would have the */- /etc/auto.media* entry in the *auto.master* file?
8. AutoFS requires *root* privileges to automatically mount a network file system. True or False?

9. What is the default timeout value for a file system before AutoFS unmounts it automatically?
10. Name the three common types of maps that AutoFS support.
11. Arrange the tasks in three different correct sequences to export a share using NFS: (a) update */etc/exports*, (b) add service to firewalld, (c) run *exportfs*, (d) install *nfs-utils*, and (e) start nfs service.
12. What is the name of the AutoFS configuration file and where is it located?
13. The name of the AutoFS service daemon is *autofs*. True or False?

Answers to Check Your Understanding

1. This indirect map entry would mount individual user home directories from *server10*.
2. The *exportfs* command.
3. True.
4. The *exportfs* command with the -u switch.
5. The */etc/nfs.conf* file.
6. The line entry would export */dir1* in read/write mode to all systems.
7. A direct map.
8. False.
9. Five minutes.
10. The three common AutoFS maps are master, direct, and indirect.
11. d/e/b/a/c, d/e/a/c/b, or d/e/a/b/c.
12. The name of the AutoFS configuration file is *autofs.conf* and it is located in the */etc* directory.
13. False. The name of the AutoFS service daemon is *automount*.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 16-1: Configure NFS Share and Automount with Direct Map

As *user1* with *sudo* on *server2*, share directory */common* (create it) in read/write mode using NFS. On *server1* as *user1* with *sudo*, install the AutoFS software and start the service. Configure the master and a direct map to automount the share on */mntauto* (create it). Run **exportfs -avr** to export the share. Run *ls* on */mntauto* to trigger the mount. Use **df -h** to confirm. (Hint: NFS Server and Client Configuration, and Auto File System).

Lab 16-2: Automount NFS Share with Indirect Map

As *user1* with *sudo* on *server1*, configure the master and an indirect map to automount the share under */autoindir* (create it). Run *ls* on */autoindir/common* to trigger the mount. Use **df -h** to confirm. (Hint: Auto File System).

Chapter 17

Networking, Network Devices, and Network Connections

This chapter describes the following major topics:

- Overview of basic networking concepts: hostname, IPv4, network classes, subnetting, subnet mask, CIDR, protocol, TCP/UDP, well-known ports, ICMP, Ethernet address, IPv6, IPv4/IPv6 differences, consistent device naming, etc.
- Change hostname of the system
- Understand the concepts of network device and connection
- Anatomy of a network connection profile
- Know network device and connection management tools and techniques

- Configure network connections by hand and using commands
- Describe the hosts table
- Test network connectivity using hostname and IP address

RHCSA Objectives:

42. Configure IPv4 and IPv6 addresses

A computer network is formed when two or more physical or virtual computers are connected together for sharing resources and data. The computers may be linked via wired or wireless means, and a device such as a switch is used to interconnect several computers to allow them to communicate with one another on the network. There are numerous concepts and terms that need to be grasped in order to work effectively and efficiently with network device and network connection configuration and troubleshooting, and several other network services. This chapter provides a wealth of that information.

For a system to be able to talk to other systems, one of its network devices must have a connection profile attached containing a unique IP address, hostname, and other essential network parameters. The network assignments may be configured statically or obtained automatically from a DHCP server. Few files are involved in the configuration, which may be modified by hand or using commands. Testing follows the configuration to confirm the system’s ability to communicate.

Networking Fundamentals

The primary purpose of computer networks is to allow users to share data and resources. A simple network is formed when two computers are interconnected. Using a networking device such as a *switch*, this network can be expanded to include additional computers, as well as printers, scanners, storage, and other devices (collectively referred to as *nodes* or *entities*). A computer on the network can be configured to act as a file server, storage server, or as a gateway to the Internet for the rest of the networked computers. Nodes may be interconnected using wired or wireless means. A corporate

network may have thousands of nodes linked via a variety of data transmission media. The Internet is the largest network of networks with millions of nodes interconnected.

There are many elementary concepts and terms that you need to grasp before being able to configure network interfaces, connection profiles, and client/server setups that are elaborated in this and other chapters. As well, there are many configuration files and commands related to various network services that you need to understand thoroughly in order to manage a RHEL-based environment effectively. Some of the concepts, terms, configuration files, and commands are explained in this chapter.

Hostname

A *hostname* is a unique alphanumeric label (the hyphen (-), underscore (_), and period (.) characters are also allowed) that is assigned to a node to identify it on the network. It can consist of up to 253 characters. It is normally allotted based on the purpose and principal use of the system. In RHEL, the hostname is stored in the */etc/hostname* file.

The hostname can be viewed with several different commands, such as *hostname*, *hostnamectl*, *uname*, and *nmcli*, as well as by displaying the content of the */etc/hostname* file. Let's run these commands on *server1*:

```
[user1@server1 ~]$ hostname
server1.example.com
[user1@server1 ~]$ hostnamectl --static
server1.example.com
[user1@server1 ~]$ uname -n
server1.example.com
[user1@server1 ~]$ nmcli general hostname
server1.example.com
[user1@server1 ~]$ cat /etc/hostname
server1.example.com
```

All of the above reported exactly the same output.

Exercise 17-1: Change System Hostname

This exercise should be done on both *server1* and *server2* as *user1* with *sudo* where required.

In this exercise, you will change the hostnames of both lab servers persistently. You will rename *server1.example.com* to *server10.example.com* by editing a file and restarting the corresponding service daemon. You will rename *server2.example.com* to *server20.example.com* using a command. You will validate the change on both systems.

On *server1*:

1. Open the */etc/hostname* file in a text editor and change the current entry to the following:

```
server10.example.com
```

2. Execute the *systemctl* command to restart the *systemd-hostnamed* service daemon and verify the new hostname with the *hostname* command:

```
[user1@server1 ~]$ sudo systemctl restart systemd-hostnamed
[user1@server1 ~]$ hostname
server10.example.com
```

3. To view the reflection of the new hostname in the command prompt, log off and log back in as *user1*. The new prompt will look like:

```
[user1@server10 ~]$
```

On *server2*:

1. Execute the *hostnamectl* command to change the hostname to *server20.example.com*:

```
[user1@server2 ~]$ sudo hostnamectl set-hostname server20.example.com
```

2. To view the reflection of the new hostname in the command prompt, log off and log back in as *user1*. You

can also use the *hostname* command to view the new name.

```
[user1@server20 ~]$
[user1@server20 ~]$ hostname
server20.example.com
```

You can also change the system hostname using the *nmcli* command. For instance, you could have used **nmcli general hostname server20.example.com** to rename *server2.example.com*. The *nmcli* command is explained in detail later in this chapter.

Going forward, you will be using the new hostnames *server10* and *server20*.

IPv4 Address

IPv4 stands for *Internet Protocol version 4* and represents a unique 32-bit software address that every single entity on the network must have in order to communicate with other entities. It was the first version of IP that was released for public use. IPv4 addresses are also referred to as *dotted-quad* addresses, and they can be assigned on a temporary or permanent basis. Temporary addresses are referred to as *dynamic* addresses and are typically leased from a DHCP server for a specific period of time. Permanent addresses, on the other hand, are called *static* addresses and they are manually set.

You can use the *ip* command with the addr argument to view the current IP assignments on the system. Let’s run this command on *server10* and see what it returns:

```
[user1@server10 ~]$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:bf:55:8e brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.110/24 brd 192.168.0.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:febf:558e/64 scope link
       valid_lft forever preferred_lft forever
```

The output indicates one configured network connection (number 2 above) called *enp0s3* with IPv4 address 192.168.0.110 assigned to it. The other connection (number 1 above), represented as *lo*, is a special purpose software device reserved for use on every Linux system. Its IPv4 address is always 127.0.0.1, and it is referred to as the system's *loopback* (or *localhost*) address. Network programs and applications that communicate with the local system employ this hostname.

Network Classes

An IPv4 address is comprised of four period-separated octets (4 x 8 = 32 bit address) that are divided into a *network* portion (or network ID/bits) comprising of the *Most Significant Bits* (MSBs) and a *node* portion (or node/host ID/bits) containing the *Least Significant Bits* (LSBs). The network portion identifies the correct destination network, and the node portion represents the correct destination node on that network. Public network addresses are classified into three categories: class A, class B, and class C. Private network addresses are classified into two categories: class D and class E. Class D addresses are multicast and they are employed in special use cases only. Class E addresses are experimental and are reserved for future use.

Class A

Class A addresses are used for large networks with up to 16 million nodes. This class uses the first octet as the network portion and the rest of the octets as the node portion. The total number of usable network and node addresses can be up to 126 and 16,777,214, respectively. The network address range for class A networks is between 0 and 127. See an example below of a random class A IP address, which also shows two reserved addresses:

10.121.51.209	(class A IP address)
10.121.51.**0**	(network address)
10.121.51.**255**	(broadcast address)

The 0 and 255 (highlighted) are network and broadcast addresses, and they are always reserved.

Class B

Class B addresses are used for mid-sized networks with up to 65 thousand nodes. This class employs the first two octets as the network portion and the other two as the node portion. The total number of usable network and node addresses can be up to 16,384 and 65,534, respectively. The network address range for class B networks is between 128 and 191. See an example below of a random class B IP address, which also shows two reserved addresses:

161**.**121**.**51**.**209	(class B IP address)
161.121.51.**0**	(network address)
161.121.51.**255**	(broadcast address)

The 0 and 255 (highlighted) are network and broadcast addresses, and they are always reserved.

Class C

Class C addresses are employed for small networks with up to 254 nodes. This class uses the first three octets as the network portion and the last octet as the node portion. The total number of usable network and node addresses can be

up to 2,097,152 and 254, respectively. The network address range for class C networks is between 192 and 223. See an example below of an arbitrary class C IP address, which also shows two reserved addresses:

215**.**121**.**51**.**209	(class C IP address)
215.121.51.**0**	(network address)
215.121.51.**255**	(broadcast address)

The 0 and 255 (highlighted) are network and broadcast addresses, and they are always reserved.

Class D

Class D addresses range from 224 to 239.

Class E

Class E addresses range from 240 to 255.

Subnetting

Subnetting is a technique by which a large network address space is divided into several smaller and more manageable logical subnetworks, referred to as *subnets*. Subnetting results in reduced network traffic, improved network performance, and de-centralized and easier administration, among other benefits. Subnetting does not touch the network bits; it uses the node bits only.

The following should be kept in mind when dealing with subnetting:

- ✓ Subnetting does not increase the number of IP addresses in a network. In fact, it reduces the number of usable addresses.
- ✓ All nodes in a given subnet have the same subnet mask.
- ✓ Each subnet acts as an isolated network and requires a router to talk to other subnets.

- The first and the last IP address in a subnet are reserved. The first address points to the subnet itself, and the last address is the broadcast address.

Subnet Mask

A *subnet mask* or *netmask* is the network portion plus the subnet bits. It segregates the network bits from the node bits. It is used by routers to pinpoint the start and end of the network/subnet portion and the start and end of the node portion for a given IP address.

The subnet mask, like an IP address, can be represented in either decimal or binary notation. The 1s in the subnet mask isolate the subnet bits from the node bits that contain 0s. The default subnet masks for class A, B, and C networks are 255.0.0.0, 255.255.0.0, and 255.255.255.0, respectively.

To determine the subnet address for an arbitrary IP address, such as 192.168.12.72 with netmask 255.255.255.224, write the IP address in binary format. Then write the subnet mask in binary format with all network and subnet bits set to 1 and all node bits set to 0. Then perform a logical AND operation. For each matching 1 you get a 1, otherwise you get a 0. The following highlights the ANDed bits:

11000000.**10101**000.0000**11**00.0**1**001000	(IP address 192.168.12.72)
11111111.**111111**11.1111**11**11.**11**100000	(subnet mask 255.255.255.224)
================================	
11000000.**10101**000.0000**11**00.0**1**000000	(subnet IP 192.168.12.64 in binar format)
192 . 168 . 12 . 64	(subnet IP in decimal format)

This calculation enables you to ascertain the subnet address from a given IP and subnet mask.

Classless Inter-Domain Routing (CIDR) Notation

Classless Inter-Domain Routing (CIDR) is a technique designed to control the quick depletion of IPv4 addresses and the rapid surge in the number of routing tables required to route IPv4 traffic on the network and the Internet. This technique was introduced as a substitute for the *classful* scheme, which was not scalable and had other limitations. Using CIDR, IPv4 addresses can be allocated in custom blocks suitable for networks of all sizes. This technique has resulted in smaller and less cluttered routing tables. CIDR was originally designed to address IPv4 needs; however, it has been extended to support IPv6 as well.

An IPv4 address written in CIDR notation has a leading forward slash (/) character followed by the number of routing bits. A sample class C IP address of 192.168.0.20 with the default class C subnet mask of 255.255.255.0 will be written as 192.168.0.20/24. This notation presents a compact method of denoting an IP address along with its subnet mask.

Protocol

A *protocol* is a set of rules governing the exchange of data between two network entities. These rules include how data is formatted, coded, and controlled. The rules also provide error handling, speed matching, and data packet sequencing. In other words, a protocol is a common language that all nodes on the network speak and understand. Protocols are defined in the */etc/protocols* file. An excerpt from this file is provided below:

```
[user1@server10 ~]$ cat /etc/protocols
ip      0       IP              # internet protocol, pseudo protocol number
hopopt  0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipv4    4       IPv4            # IPv4 encapsulation
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
cbt     7       CBT             # CBT, Tony Ballardie <A.Ballardie@cs.ucl.ac.uk>
egp     8       EGP             # exterior gateway protocol
........
```

Column 1 in the output lists the name of a protocol, followed by the associated port number, alias, and a short description in columns 2, 3, and 4.

Some common protocols are TCP, UDP, IP, and ICMP.

TCP and UDP Protocols

TCP (*Transmission Control Protocol*) and UDP (*User Datagram Protocol*) protocols are responsible for transporting data packets between network entities. TCP is reliable, connection-oriented, and point-to-point. It inspects for errors and sequencing upon a packet's arrival on the destination node, and returns an acknowledgement to the source node, establishing a point-to-point connection with the peer TCP layer on the source node. If the packet is received with an error or if it is lost in transit, the destination node requests a resend of the packet. This ensures guaranteed data delivery and makes TCP reliable. Due to its reliability and connection-oriented nature, TCP is widely implemented in network applications.

UDP, in contrast, is unreliable, connectionless, and multi-point. If a packet is lost or contains errors upon arrival at the destination, the source node is unaware of it. The destination node does not send an acknowledgment back to the source node. A common use of this protocol is in broadcast-only applications where reliability is not sought.

Well-Known Ports

Both TCP and UDP use ports for data transmission between a client and its server program. Ports are either well-known or private. A well-known port is reserved for an application's exclusive use, and it is standardized across all network operating systems. Well-known ports are defined in the */etc/services* file, an excerpt of which is presented below:

```
[user1@server10 ~]$ cat /etc/services
........
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                          # The Secure Shell (SSH) Protocol
ssh             22/udp                          # The Secure Shell (SSH) Protocol
telnet          23/tcp
telnet          23/udp
# 24 - private mail system
lmtp            24/tcp                          # LMTP Mail Delivery
lmtp            24/udp                          # LMTP Mail Delivery
smtp            25/tcp          mail
smtp            25/udp          mail
time            37/tcp          timserver
........
```

Column 1 lists the official name of a network service, followed by the port number and transport layer protocol the service uses, optional aliases, and comments in successive columns.

Some common services and the ports they listen on are FTP (*File Transfer Protocol*) 21, SSH (*Secure Shell*) 22, SMTP (*Simple Mail Transfer Protocol*) 25, DNS (*Domain Name System*) 53, HTTP (*HyperText Transfer Protocol*) 80, NTP (*Network Time Protocol*) 123, secure HTTP (*HyperText Transfer Protocol Secure*) 443, and rsyslog 514.

A private port, on the other hand, is an arbitrary number generated when a client application attempts to establish a communication session with its server process. This port number no longer exists after the session has ended.

ICMP Protocol

The *Internet Control Message Protocol* (ICMP) is a key protocol. It is primarily used for testing and diagnosing

network connections. Commands such as *ping* uses this protocol to send a stream of messages to remote network devices to examine their health and report statistical and diagnostic information. The report includes the number of packets transmitted, received, and lost; a round-trip time for individual packets with an overall average; a percentage of packets lost during the communication; and so on. See a sample below that shows two packets (-c2) sent from *server10* to the IP address of *server20*:

```
[user1@server10 ~]$ ping -c2 192.168.0.120
PING 192.168.0.120 (192.168.0.120) 56(84) bytes of data.
64 bytes from 192.168.0.120: icmp_seq=1 ttl=64 time=0.325 ms
64 bytes from 192.168.0.120: icmp_seq=2 ttl=64 time=0.402 ms

--- 192.168.0.120 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 45ms
rtt min/avg/max/mdev = 0.325/0.363/0.402/0.042 ms
```

Other commands, such as *traceroute*, also employ this protocol for route determination and debugging between network entities. The IPv6 version of ICMP is referred to as *ICMPv6* and it is used by tools such as *ping6* and *tracepath6*.

Ethernet Address

An *Ethernet* address represents an exclusive 48-bit address that is used to identify the correct destination node for data packets transmitted from the source node. The data packets include hardware addresses for the source and the destination node. The Ethernet address is also referred to as the *hardware*, *physical*, *link layer*, or *MAC* address.

You can use the *ip* command to list all network interfaces available on the system along with their Ethernet addresses:

```
[user1@server10 ~]$ ip addr | grep ether
    link/ether 08:00:27:bf:55:8e brd ff:ff:ff:ff:ff:ff
```

IP and hardware addresses work hand in hand, and a combination of both is critical to identifying the correct destination node on the network. A network protocol called

Address Resolution Protocol (ARP) is used to enable IP and hardware addresses to work in tandem. ARP determines the hardware address of the destination node when its IP address is known.

IPv6 Address

With the explosive growth of the Internet, the presence of an extremely large number of network nodes requiring an IP, and an ever-increasing demand for additional addresses—the conventional IPv4 address space, which provides approximately 4.3 billion addresses—has been exhausted. To meet the future demand, a new version of IP is now available and its use is on the rise. This new version is referred to as IPv6 (*IP version 6*) or IPng (*IP next generation*). By default, IPv6 is enabled in RHEL 8 on all configured network connections.

IPv6 is a 128-bit software address, providing access to approximately 340 undecillion (340 followed by 36 zeros) addresses. This is an extremely large space, and it is expected to fulfill the IP requirements for several decades to come.

IPv6 uses a messaging protocol called *Neighbor Discovery Protocol* (NDP) to probe the network to discover neighboring IPv6 devices, determine their reachability, and map their associations. This protocol also includes enhanced functionalities (provided by ICMP and ARP on IPv4 networks) for troubleshooting issues pertaining to connectivity, address duplication, and routing.

Unlike IPv4 addresses, which are represented as four dot-separated octets, IPv6 addresses contain eight colon-separated groups of four hexadecimal numbers. A sample v6 IP would be 1204:bab1:21d1:bb43:23a1:9bde:87df:bac9. It looks a bit daunting at first sight, but there are methods that will simplify their representation.

The *ip addr* command also shows IPv6 addresses for the interfaces:

```
[user1@server10 ~]$ ip addr | grep inet6
    inet6 ::1/128 scope host
    inet6 fe80::a00:27ff:febf:558e/64 scope link
```

It returns two IPv6 addresses. The first one belongs to the loopback interface, and the second one is assigned to the *enp0s3* connection.

Major Differences between IPv4 and IPv6

There are a number of differences between IPv4 and IPv6 protocols. Some of the major ones are highlighted in Table 17-1.

IPv4	IPv6
Uses 4x8-bit, period-separated decimal number format for address representation. Example: 192.168.0.100	Uses 8x16-bit, colon-separated hexadecimal number format for address representation. Example: fe80::a00:27ff:feae:f35b
Number of address bits: 32	Number of address bits: 128
Maximum number of addresses: ~4.3 billion.	Maximum number of addresses: virtually unlimited
Common testing and troubleshooting tools: ping, traceroute, tracepath, etc.	Common testing and troubleshooting tools: ping6, traceroute6, tracepath6, etc.
Support for IP autoconfiguration: no	Support for IP autoconfiguration: yes
Packet size: 576 bytes	Packet size: 1280 bytes

Table 17-1 IPv4 vs IPv6

These and other differences not listed here are due to enhancements and new features added to IPv6.

Network Devices and Connections

Network Interface Cards (NICs) are hardware adapters that provide one or more Ethernet ports for network connectivity. NICs may also be referred to as *network adapters* and individual ports as *network interfaces* or *network devices*. NICs may be built-in to the system board or are add-on adapters. They are available in one, two, and four port designs on a single adapter.

Individual interfaces (devices) can have one or more connection profiles attached to them with different configuration settings. Each connection profile has a unique name and includes settings such as the device name, hardware address, activating the connection on reboot, and so on. A connection profile can be configured by editing files or using commands. A device can have multiple connection profiles attached, but only one of them can be active at a time.

Consistent Network Device Naming

In RHEL versions earlier than 7, network interfaces were named *eth* (Ethernet), *em* (embedded), and *wlan* (wireless lan), and were numbered 0 and onwards as the interfaces were discovered during a system boot. This was the default scheme that had been in place for network device naming for years. Given a large number of interfaces located onboard and on add-on NICs, the number assignments could possibly change on the next boot due to failures or errors in their detection, which will result in connectivity and operational issues.

As of RHEL 7, the default naming scheme has been augmented to base on several rules governed by the *udevd* service. The default ruleset is to assign names using the

device’s location and topology, and the setting in firmware. The underlying virtualization layer (VMware, VirtualBox, KVM) also plays a role in the naming. Some sample device names are *enp0s3*, *ens160*, etc.

This advanced ruleset has resulted in consistent and predictable naming, eliminating the odds of reenumeration during a hardware rescan. Moreover, the designated names are not affected by the addition or removal of interface cards. This naming scheme helps in identifying, configuring, troubleshooting, and replacing the right adapter without hassle.

Understanding Interface Connection Profile

Each network connection has a configuration file that defines IP assignments and other relevant parameters for it. The networking subsystem reads this file and applies the settings at the time the connection is activated. Connection configuration files (or *connection profiles*) are stored in a central location under the */etc/sysconfig/network-scripts* directory. The filenames begin with *ifcfg*-and are followed by the name of the connection. Some instances of connection filenames are *ifcfg-enp0s3*, *ifcfg-ens160*, and *ifcfg-em1*.

On *server10* and *server20*, the device name for the first interface is *enp0s3* with connection name *enp0s3* and relevant connection information stored in the *ifcfg-enp0s3* file. This connection was established at the time of installation. The current content of the *ifcfg-enp0s3* file from *server10* are presented below with IPv6 directives excluded:

```
[user1@server10 ~]$ cat /etc/sysconfig/network-scripts/ifcfg-enp0s3 | grep -vi ipv6
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=enp0s3
UUID=5ece88f2-2b9c-420f-8e1d-a66d648ed171
DEVICE=enp0s3
ONBOOT=yes
IPADDR=192.168.0.110
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
```

These directives and a few others that can be defined in this file are listed in alphabetical order in Table 17-2.

Directive	Description
BOOTPROTO	Defines the boot protocol to be used. Common values include dhcp to obtain IP assignments from a DHCP server and none or static to use a static IP as set with the IPADDR directive.
BROWSER_ONLY	Works if PROXY_METHOD is set to auto. Default is no.
DEFROUTE	Whether to use this connection as the default route
DEVICE	Specifies the device name for the network interface
DNS1	Defines the IP address or the hostname of the first DNS server. This address/hostname is placed in the /etc/resolv.conf file if the PEERDNS directive is set to no in this file.
GATEWAY	Specifies the gateway address for the connection if the BOOTPROTO directive is set to none or static
HWADDR	Describes the hardware address for the device
IPADDR	Specifies the static IP for the connection if the BOOTPROTO directive is set to none or static
IPV4_FAILURE_FATAL	Whether to disable the device if IPv4 configuration fails. Default is no.
IPV6INIT	Whether to enable IPv6 support for this connection
NAME	Any description given to this connection. The default matches the device name.
NETMASK	Sets the netmask address for the connection if the BOOTPROTO directive is set to none or static
NM_CONTROLLED	Whether the NetworkManager service is to be allowed to modify the configuration for this connection. It should be turned off on computers that use static IP addresses. Default is yes.
ONBOOT	Whether to auto-activate this connection at system boot
PEERDNS	Whether to modify the DNS client resolver

	file /etc/resolv.conf.Default is yes if BOOTPROTO is set to dhcp.
PREFIX	Defines the number of subnet bits. This directive may be used in lieu of NETMASK.
PROXY_METHOD	Method to be used for proxy setting. Default is no.
UUID	The UUID associated with this connection
TYPE	Specifies the type of this connection

Table 17-2 Network Connection Configuration Directives

There are numerous other directives, including those for IPv6, that may be defined in connection profiles. Run **man nm-settings** for a description of additional directives.

Exercise 17-2: Add Network Devices to server10 and server20

This exercise will add one network interface to *server10* and one to *server20* using VirtualBox.

In this exercise, you will shut down *server10* and *server20*. You will launch VirtualBox and add one network interface to *server10* and one to *server20* in preparation for Exercises 17-3 and 17-4.

1. Execute **sudo shutdown now** on *server10*.
2. Start VirtualBox on your Windows/Mac computer and highlight the *RHEL8-VM1* virtual machine that you created in Exercise 1-2. Make sure it is powered off.
3. Click Settings at the top and then Network on the window that pops up. Click on “Adapter 2” and check the “Enable Network Adapter” box. Select “Internal Network” from the drop down list besides “Attached to”. See Figure 17-1.

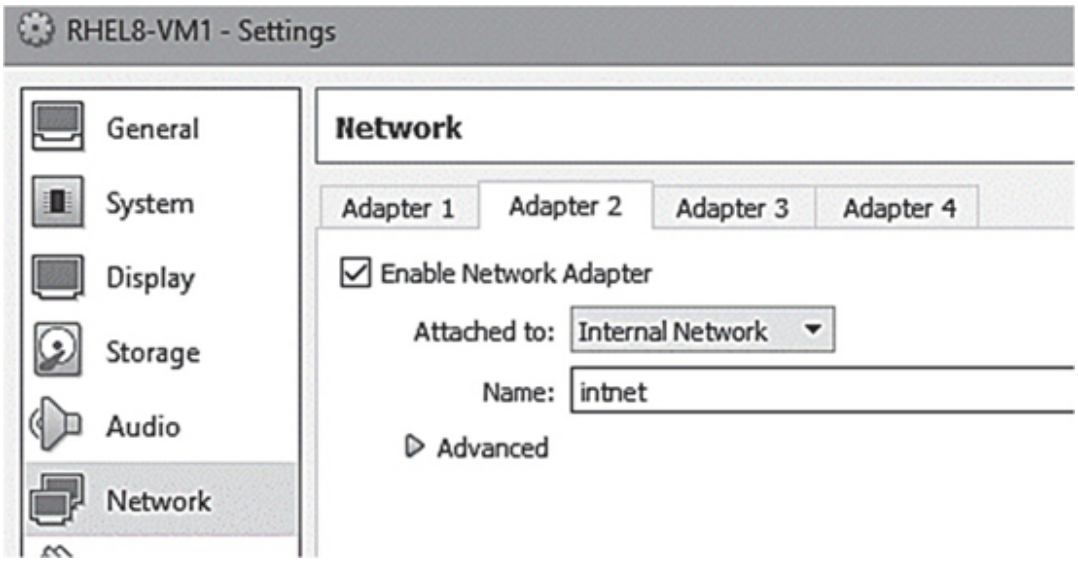


Figure 17-1 VirtualBox – Add Network Interface

4. Click OK to return to the main VirtualBox interface.
5. Power on *RHEL8-VM1* to boot RHEL 8 in it.
6. When the server is up, log on as *user1* and run the *ip* command as follows to verify the new interface:

```
[user1@server10 ~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:bf:55:8e brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.110/24 brd 192.168.0.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:febf:558e/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:42:1e:1d brd ff:ff:ff:ff:ff:ff
```

The output reveals a new network device called *enp0s8*. This is the new interface you just added to the VM.

7. Repeat steps 1 through 6 to add a network interface to *server20* and verify.

This completes the addition of new network interfaces to *server10* and *server20*. You are now ready to use them in the upcoming exercises.

Network Device and Connection Administration Tools

There are a few tools and methods available for configuring and administering network interfaces, connections, and connection profiles. The *NetworkManager* service includes a toolset for this purpose as well. Let's take a quick look at the basic management tools in Table 17-3.

Command	Description
ip	A powerful and versatile tool for displaying, monitoring, and managing interfaces, connections, routing, traffic, etc.
ifup	Activates a connection
ifdown	Deactivates a connection

Table 17-3 Basic Network Management Tools

You can manually create a connection profile and attach it to a network device. Many Linux administrators prefer this approach. RHEL also offers an alternative method for this purpose, which is discussed later in this chapter.

In Exercise 17-3, you'll use the manual method to configure a connection profile for a new network device that was added in Exercise 17-2 and employ the tools listed in Table 17-3.

Exercise 17-3: Configure New Network Connection Manually

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will create a connection profile for the new network interface *enp0s8* using a text editing tool. You will assign the IP 172.10.10.110/24 with gateway 172.10.10.1 and set it to autoactivate at system reboots. You will deactivate and reactivate this interface at the command prompt.

1. Create a file called *ifcfg-enp0s8* in the */etc/sysconfig/network-scripts* directory and enter the following information to establish a connection profile:

```
[user1@server10 ~]$ sudo cat /etc/sysconfig/network-scripts/ifcfg-enp0s8
TYPE=Ethernet
BOOTPROTO=static
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
NAME=enp0s8
DEVICE=enp0s8
ONBOOT=yes
IPADDR=172.10.10.110
PREFIX=24
GATEWAY=172.10.10.1
```

2. Deactivate and reactivate this interface using the *ifdown* and *ifup* commands:

```
[user1@server10 ~]$ sudo ifdown enp0s8
Connection 'enp0s8' successfully deactivated (D-Bus active path: /org/freedeskto
p/NetworkManager/ActiveConnection/2)
[user1@server10 ~]$
[user1@server10 ~]$ sudo ifup enp0s8
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkMa
nager/ActiveConnection/3)
```

3. Verify the activation of *enp0s8* connection:

```
[user1@server10 ~]$ ip a
........
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:42:1e:1d brd ff:ff:ff:ff:ff:ff
    inet 172.10.10.110/24 brd 172.10.10.255 scope global noprefixroute enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe42:1e1d/64 scope link
       valid_lft forever preferred_lft forever
```

The new connection profile called *enp0s8* has been applied to the new network device called *enp0s8*. The connection is now

ready for use. The connectivity to *server10* over this new connection is tested later in this chapter.

The NetworkManager Service

NetworkManager is the default interface and connection configuration, administration, and monitoring service used in RHEL 8. This service has a daemon program called *NetworkManager*, which is responsible for keeping available interfaces and connections up and active. It offers a powerful command line tool called *nmcli* to manage interfaces and connections, and to control the service. This utility offers many options for their effective management. The *NetworkManager* service also furnishes a text-based interface called *nmtui* and a graphical equivalent called *nm-connection-editor* that you may use in lieu of *nmcli*.

The nmcli Command

nmcli is a NetworkManager command line tool that is employed to create, view, modify, remove, activate, and deactivate network connections, and to control and report network device status. It operates on seven different object categories, with each category supporting several options to form a complete command. The seven categories are general, networking, connection, device, radio, monitor, and agent. This discussion only focuses on the connection and device object categories. They are described in Table 17-4 along with management operations that they can perform.

Object	Description
Connection: activates, deactivates, and administers network connections	
show	Lists connections
up / down	Activates/deactivates a connection
add	Adds a connection
edit	Edits an existing connection or adds a new one
modify	Modifies one or more properties of a connection
delete	Deletes a connection
reload	Instructs NetworkManager to re-read all connection profiles
load	Instructs NetworkManager to re-read a connection profile
Device: displays and administers network interfaces	
status	Exhibits device status
show	Displays detailed information about all or the specified interface

Table 17-4 Network Connection and Device Administration Tools

Object categories and the objects within them may be written in an abridged form to save typing. For instance, the connection category may be abbreviated as a “c” or “con” and the device category as a “d” or “dev”. The same rule applies to object names as well. For instance, add may be specified as an “a”, delete as a “d”, and so on. Check the manual pages for nmcli-examples.

The *nmcli* command supports tab completion to make its use easier. Let’s run a few examples on *server10* to understand the command’s usage.

To show (s) all available connections (c) including both active and inactive:

```
[user1@server10 ~]$ nmcli c s
NAME    UUID                                  TYPE      DEVICE
enp0s3  5ece88f2-2b9c-420f-8e1d-a66d648ed171  ethernet  enp0s3
enp0s8  00cb8299-feb9-55b6-a378-3fdc720e0bc6  ethernet  enp0s8
```

The output lists two connection profiles (NAME) and the devices (DEVICE) they are attached to. It also shows their UUID and type.

To deactivate (down) the connection (c) *enp0s8*:

```
[user1@server10 ~]$ sudo nmcli c down enp0s8
Connection 'enp0s8' successfully deactivated (D-Bus active path: /org/freedeskto
p/NetworkManager/ActiveConnection/3)
```

The connection profile is detached from the device, disabling the connection. You can check with **nmcli c s**.

To activate (up) the connection (c) enp0s8:

```
[user1@server10 ~]$ sudo nmcli c up enp0s8
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkMa
nager/ActiveConnection/4)
```

The connection profile is reattached to the device, enabling the connection. You can check with **nmcli c s**.

To display the status (s) of all available network devices (d):

```
[user1@server10 ~]$ nmcli d s
DEVICE  TYPE      STATE      CONNECTION
enp0s3  ethernet  connected  enp0s3
enp0s8  ethernet  connected  enp0s8
lo      loopback  unmanaged  --
```

The output shows three devices and their types, states, and the connection profiles attached to them. The loopback interface is not managed by the NetworkManager service.

Exercise 17-4: Configure New Network Connection Using nmcli

This exercise should be done on *server20* as *user1* with *sudo* where required.

In this exercise, you will create a connection profile using the *nmcli* command for the new network interface *enp0s8* that was added to *server20* in Exercise 17-2. You will assign the IP 172.10.10.120/24 with gateway 172.10.10.1, and set it to autoactivate at system reboots. You will deactivate and reactivate this interface at the command prompt.

1. Check the running status of the *NetworkManager* service:

```
[user1@server20 ~]$ sudo systemctl status NetworkManager
● NetworkManager.service - Network Manager
   Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; ven
   Active: active (running) since Sat 2019-11-16 23:31:47 EST; 13h ago
     Docs: man:NetworkManager(8)
 Main PID: 1183 (NetworkManager)
    Tasks: 3 (limit: 11516)
   Memory: 11.3M
   ........
```

The service is up and active on the server.

2. Check the presence of the new interface:

```
[user1@server20 ~]$ sudo nmcli d status | grep enp
enp0s3      ethernet  connected     enp0s3
enp0s8      ethernet  disconnected  --
```

The output signifies the presence of a new network device called *enp0s8*. It does not have a connection profile attached to it.

3. Add (add) a connection profile (con) and attach it to the new interface. Use the type Ethernet, device name (ifname) *enp0s8* with a matching connection name (con-name), CIDR (ip4) 172.10.10.120/24, and gateway (gw4) 172.10.10.1:

```
[user1@server20 ~]$ sudo nmcli con add type Ethernet ifname enp0s8 con-name enp0s8
ip4 172.10.10.120/24 gw4 172.10.10.1
Connection 'enp0s8' (c9f81ca1-4418-4a8d-9f9a-b6e4368bba46) successfully added.
```

A new connection has been added, attached to the new interface, and activated. In addition, the command has saved

the connection information in a new file called *ifcfg-enp0s8* and stored it in the */etc/sysconfig/network-scripts* directory.

4. Confirm the new connection status:

```
[user1@server20 ~]$ nmcli c s
NAME    UUID                                  TYPE      DEVICE
enp0s3  7426fe5c-9bd4-42ad-bb95-3d15491082b9  ethernet  enp0s3
enp0s8  c9f81ca1-4418-4a8d-9f9a-b6e4368bba46  ethernet  enp0s8
```

The output indicates the association of the new connection with the network device.

5. Check the content of the *ifcfg-enp0s8* connection profile:

```
[user1@server20 ~]$ cat /etc/sysconfig/network-scripts/ifcfg-enp0s8
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=172.10.10.120
PREFIX=24
GATEWAY=172.10.10.1
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp0s8
UUID=c9f81ca1-4418-4a8d-9f9a-b6e4368bba46
DEVICE=enp0s8
ONBOOT=yes
```

There are a number of default directives added to the connection profile in addition to the configuration items you entered with the *nmcli* command above. The ONBOOT directive is also set to yes automatically. This setting is an indicative of the fact that the connection will be auto-enabled at system reboots.

6. Check the IP assignments for the new connection:

```
[user1@server20 ~]$ ip a
.......
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 default qlen 1000
    link/ether 08:00:27:a1:15:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.10.10.120/24 brd 172.10.10.255 scope global noprefixroute enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::9d:bc7a:b38f:5bb3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

The IP is assigned to the interface. The connection is tested later in this chapter.

7. Deactivate this connection to detach it from the interface:

```
[user1@server20 ~]$ sudo nmcli c down enp0s8
Connection 'enp0s8' successfully deactivated (D-Bus active path: /org/freedesktop/N
etworkManager/ActiveConnection/4)
```

The connection profile is now detached from the interface, deactivating the connection. You can check with **nmcli c s**.

8. Reactivate the connection to attach it to the interface:

```
[user1@server20 ~]$ sudo nmcli c up enp0s8
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManag
er/ActiveConnection/5)
```

The connection profile is now reattached to the interface, activating the connection. You can check with **nmcli c s**.

This brings the exercise to a conclusion.

Understanding Hosts Table

Each IP address used on the system should have a hostname assigned to it. In an environment with multiple systems on the network, it is prudent to have some sort of a hostname to IP address resolution method in place to avoid typing the destination system IP repeatedly to access it. DNS is one such method. It is designed for large networks such as corporate networks and the Internet. For small, internal networks, the use of a local hosts table (the */etc/hosts* file) is also common. This table is used to maintain hostname to IP mapping for

systems on the local network, allowing us to access a system by simply employing its hostname. In this book, there are two systems in place: *server10.example.com* with IP 192.168.0.110 and alias *server10*, and *server20.example.com* with IP 192.168.0.120 and alias *server20*. You can append this information to the */etc/hosts* file on both *server10* and *server20* as shown below:

```
192.168.0.110 server10.example.com server10
192.168.0.120 server20.example.com server20
```

Each row in the file contains an IP address in column 1 followed by the *official* (or *canonical*) hostname in column 2, and one or more optional aliases thereafter. The official hostname and one or more aliases give users the flexibility of accessing the system using any of these names.

> **EXAM TIP:** In the presence of an active DNS with all hostnames resolvable, there is no need to worry about updating the hosts file.

As expressed above, the use of the *hosts* file is common on small networks and it should be updated on each individual system to reflect any changes for best inter-system connectivity experience.

Testing Network Connectivity

RHEL includes the *ping* command to examine network connectivity between two systems. It uses the IP address of the destination system to send a series of 64-byte *Internet Control Message Protocol* (ICMP) test packets to it. A response from the remote system validates connectivity and health. With the -c option, you can specify the number of packets that you want transmit.

The following sends two packets from *server10* to 192.168.0.120 (IP for *server20*):

```
[user1@server10 ~]$ ping 192.168.0.120 -c 2
PING 192.168.0.120 (192.168.0.120) 56(84) bytes of data.
64 bytes from 192.168.0.120: icmp_seq=1 ttl=64 time=0.406 ms
64 bytes from 192.168.0.120: icmp_seq=2 ttl=64 time=0.418 ms

--- 192.168.0.120 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 38ms
rtt min/avg/max/mdev = 0.406/0.412/0.418/0.006 ms
```

Under “192.168.0.120 ping statistics,” the output depicts the number of packets transmitted, received, and lost. The packet loss should be 0%, and the round trip time should not be too high for a healthy connection. In general, you can use this command to test connectivity with the system’s own IP, the loopback IP (127.0.0.1), a static route, the default gateway, and any other address on the local or remote network.

If a *ping* response fails, you need to check if the NIC is seated properly, its driver is installed, network cable is secured appropriately, IP and netmask values are set correctly, and the default or static route is accurate.

Exercise 17-5: Update Hosts Table and Test Connectivity

This exercise should be done on *server10* and *server20* as *user1* with *sudo* where required.

In this exercise, you will update the */etc/hosts* file on both *server10* and *server20*. You will add the IP addresses assigned to both connections and map them to hostnames *server10*, *server10s8*, *server20*, and *server20s8* appropriately. You will test connectivity from *server10* to *server20* and from *server10s8* to *server20s8* using their IP addresses and then their hostnames.

On *server20*:

1. Open the */etc/hosts* file and add the following entries:

```
192.168.0.110 server10.example.com server10
192.168.0.120 server20.example.com server20
172.10.10.110 server10s8.example.com server10s8
172.10.10.120 server20s8.example.com server20s8
```

The IP addresses for both connections are added for both servers.

On *server10*:

2. Open the */etc/hosts* file and add the following entries:

```
192.168.0.110 server10.example.com server10
192.168.0.120 server20.example.com server20
172.10.10.110 server10s8.example.com server10s8
172.10.10.120 server20s8.example.com server20s8
```

The IP addresses for both connections are added for both servers.

3. Send two packets from *server10* to the IP address of *server20*:

```
[user1@server10 ~]$ ping -c2 192.168.0.120
PING 192.168.0.120 (192.168.0.120) 56(84) bytes of data.
64 bytes from 192.168.0.120: icmp_seq=1 ttl=64 time=0.369 ms
64 bytes from 192.168.0.120: icmp_seq=2 ttl=64 time=0.446 ms

--- 192.168.0.120 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 45ms
rtt min/avg/max/mdev = 0.369/0.407/0.446/0.043 ms
```

4. Issue two ping packets on *server10* to the hostname of *server20*:

```
[user1@server10 ~]$ ping -c2 server20
PING server20.example.com (192.168.0.120) 56(84) bytes of data.
64 bytes from server20.example.com (192.168.0.120): icmp_seq=1 ttl=64 time=0.351 ms
64 bytes from server20.example.com (192.168.0.120): icmp_seq=2 ttl=64 time=0.422 ms

--- server20.example.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 26ms
rtt min/avg/max/mdev = 0.351/0.386/0.422/0.040 ms
```

5. Send one packet from *server10* to the IP address of *server20s8*:

```
[user1@server10 ~]$ ping -c1 172.10.10.120
PING 172.10.10.120 (172.10.10.120) 56(84) bytes of data.
64 bytes from 172.10.10.120: icmp_seq=1 ttl=64 time=0.430 ms

--- 172.10.10.120 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.430/0.430/0.430/0.000 ms
```

6. Issue one ping packet on *server10* to the hostname of *server20s8*:

```
[user1@server10 ~]$ ping -c1 server20s8
PING server20s8.example.com (172.10.10.120) 56(84) bytes of data.
64 bytes from server20s8.example.com (172.10.10.120): icmp_seq=1 ttl=64 time=0.341 ms

--- server20s8.example.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.341/0.341/0.341/0.000 ms
```

Steps 3 through 6 verified the connectivity to the remote server over both connections. Each server has two IP addresses, and each IP address has a unique hostname assigned to it.

This concludes the exercise.

Chapter Summary

This chapter discussed the rudiments of networking. It began by providing an understanding of various essential networking terms and concepts to build the foundation for networking topics going forward. Topics such as hostname, IPv4, IPv6, network classes, subnetting, subnet mask, CIDR notation, protocol, port, Ethernet address, and consistent device naming were covered in sufficient detail.

We modified hostnames on both lab servers by modifying a configuration file and restarting the hostname service on one server and using a single command on the other. We employed two different methods to demonstrate multiple ways of doing the same thing. A third method was also mentioned to rename the hostname.

Next, we described the terms network devices and network connections, and realized the difference between the two. We examined a connection profile and looked at a number of directives that may be defined for a network connection. We added a new network device to each lab server and configured them by employing two different methods. We activated the new connections and performed a ping test for functional validation. Lastly, we populated the hosts table with the IP and hostname mapping on both lab servers.

Check Your Understanding

1. What is the use of *ifup* and *ifdown* commands?
2. Which service is responsible for maintaining consistent device naming?
3. List three key differences between TCP and UDP protocols.
4. What is the significance of the NAME and DEVICE directives in a connection profile?
5. Which class of IP addresses has the least number of node addresses?
6. Which command can you use to display the hardware address of a network device?
7. Define protocol.
8. Which directory stores the network connection profiles?
9. True or False. A network device is a physical or virtual network port and a network connection is a configuration file attached to it.
10. IPv4 is a 32-bit software address. How many bits does an IPv6 address have?
11. Which file defines the port and protocol mapping?
12. What would the command *hostnamectl set-hostname host20* do?
13. Name the file that stores the hostname of the system.
14. What would the command *nmcli cs* do?

15. What is the purpose of the ONBOOT directive in the network connection profile?
16. The */etc/hosts* file maintains hostname to hardware address mappings. True or False?
17. Which file contains service, port, and protocol mappings?
18. What would the *ip addr* command produce?
19. Which file would you consult to identify the port number and protocol associated with a network service?
20. Adding a connection profile with the *nmcli* command creates a connection profile in the */etc/sysconfig/network-scripts* directory. True or False?
21. Name four commands that can be used to display the system hostname?
22. List any two benefits of subnetting.

Answers to Check Your Understanding

1. The *ifup* and *ifdown* commands are used to enable and disable a network connection, respectively.
2. The *udevd* service handles consistent naming of network devices.
3. TCP is connection-oriented, reliable, and point-to-point; UDP is connectionless, unreliable, and multi-point.
4. The NAME directive sets the name for the network connection and the DEVICE directive defines the network device the connection is associated with.
5. The C class supports the least number of node addresses.
6. The *ip* command.
7. A set of rules that govern the exchange of information between two network entities.
8. The */etc/sysconfig/network-scripts* directory.
9. True.
10. 128.

11. The */etc/protocols* file.
12. The command provided will update the */etc/hostname* file with the specified hostname and restart the *systemd-hostnamed* daemon for the change to take effect.
13. The */etc/hostname* file.
14. The command provided will display the status information for all network connections.
15. The purpose of the ONBOOT directive is to direct the boot scripts whether to activate this connection.
16. False. This file maintains hostname to IP address mapping.
17. The */etc/services* file.
18. This command provided will display information about network connections including IP assignments and hardware address.
19. The */etc/services* file.
20. True.
21. The *hostname*, *uname*, *hostnamectl*, and *nmcli* commands can be used to view the system hostname.
22. Better manageability and less traffic.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 17-1: Add New Interface and Configure Connection Profile with nmcli

Add a new network interface to *RHEL8-VM1* in VirtualBox.

As *user1* with *sudo* on *server10*, run **ip a** and verify the presence of the new interface (*enp0s8*). Use the *nmcli* command and assign IP 192.168.0.210/24 and gateway 192.168.0.1. Set the network connection to auto-activate on system reboots. Deactivate and reactivate this connection manually. (Hint: Network Devices and Connections).

Lab 17-2: Add New Interface and Configure Connection Profile Manually

Add a new network interface to *RHEL8-VM2* in VirtualBox.

As *user1* with *sudo* on *server20*, run **ip a** and verify the presence of the new interface (*enp0s8*). Make a copy of the *ifcfg-enp0s3* file as *ifcfg-enp0s8* under the */etc/sysconfig/network-scripts* directory. Remove the HWADDR and UUID directives, and set the values for IPADDR, NETMASK, and GATEWAY directives appropriately. Set the network connection to auto-activate on system reboots. Deactivate and reactivate this connection manually. (Hint: Network Devices and Connections).

Chapter 18

Time Synchronization and Hostname Resolution

This chapter describes the following major topics:

- Describe time synchronization and the role of Network Time Protocol
- Comprehend the terms: time source, NTP roles, and stratum levels
- Anatomy of the Chrony service configuration file
- Configure and verify NTP/Chrony client service
- View and set system date and time
- Overview of Domain Name System and hostname resolution
- Understand various DNS roles
- Analyze entries in resolver configuration files

- Perform name resolution using a variety of lookup tools

RHCSA Objectives:

38. Configure time service clients
43. Configure hostname resolution

The Chrony service, a new implementation of the Network Time Protocol, maintains the clock on the system and keeps it synchronized with a more accurate and reliable source of time. Providing accurate and uniform time for systems on the network allows time-sensitive applications such as monitoring software, backup tools, scheduling utilities, billing systems, file sharing protocols, and authentication programs to perform correctly and precisely. It also aids logging and auditing services to capture and record messages and alerts in log files with accurate timestamps.

Domain Name System is an OS- and hardware-independent network service employed for determining the IP address of a system when its hostname is known, and vice versa. This mechanism is implemented to map human-friendly hostnames to their assigned numeric IP addresses by consulting one or more servers offering the hostname resolution service. This service has been used on the Internet and corporate networks as the de facto standard for this purpose. DNS clients use this service to communicate with remote systems. There are several lookup programs that use DNS to obtain information.

Time Synchronization

Network Time Protocol (NTP) is a networking protocol for synchronizing the system clock with remote time servers for accuracy and reliability. This protocol has been in use with tens of millions of computing devices employing it to synchronize their clocks with tens of thousands of time servers deployed across the globe. Having steady and exact time on networked systems allows time-sensitive applications, such as authentication and email applications, backup and

scheduling tools, financial and billing systems, logging and monitoring software, and file and storage sharing protocols, to function with precision.

NTP sends a stream of messages to configured time servers and binds itself to the one with least amount of delay in its responses, the most accurate, and may or may not be the closest distance-wise. The client system maintains a drift in time in a file and references this file for gradual drop in inaccuracy.

RHEL version 8 introduces a new implementation of NTP called *Chrony*. Chrony uses the UDP protocol over the well-known port 123. If enabled, it starts at system boot and continuously operates to keep the system clock in sync with a more accurate source of time.

Chrony performs well on computers that are occasionally connected to the network, attached to busy networks, do not run all the time, or have variations in temperature.

Time Sources

A *time source* is any reference device that acts as a provider of time to other devices. The most precise sources of time are the atomic clocks. They use *Universal Time Coordinated* (UTC) for time accuracy. They produce radio signals that radio clocks use for time propagation to computer servers and other devices that require correctness in time. When choosing a time source for a network, preference should be given to the one that takes the least amount of time to respond. This server may or may not be closest physically.

The common sources of time employed on computer networks are the local system clock, an Internet-based public time server, and a radio clock.

The local system clock can be used as a provider of time. This requires the maintenance of correct time on the server either manually or automatically via *cron*. Keep in mind that this server has no way of synchronizing itself with a more reliable and precise external time source. Therefore, using the local system clock as a time server is the least recommended option.

Several public time servers are available over the Internet for general use (visit *www.ntp.org* for a list). These servers are typically operated by government agencies, research and scientific organizations, large software vendors, and universities around the world. One of the systems on the local network is identified and configured to receive time from one or more public time servers. This option is preferred over the use of the local system clock.

The official *ntp.org* site also provides a common pool called *pool.ntp.org* for vendors and organizations to register their own NTP servers voluntarily for general public use. Examples include *rhel.pool.ntp.org* and *ubuntu.pool.ntp.org* for distribution-specific pools, and *ca.pool.ntp.org* and *oceania.pool.ntp.org* for country and continent/region-specific pools. Under these sub-pools, the owners maintain multiple time servers with enumerated hostnames such as *0.rhel.pool.ntp.org*, *1.rhel.pool.ntp.org*, *2.rhel.pool.ntp.org*, and so on.

A radio clock is regarded as the perfect provider of time, as it receives time updates straight from an atomic clock. *Global Positioning System* (GPS), WWVB, and DCF77 are some popular radio clock methods. A direct use of signals from these sources requires connectivity of some hardware to the computer identified to act as an organizational or site-wide time server.

NTP Roles

From an NTP standpoint, a system can be configured to operate as a primary server, secondary server, peer, or client.

A *primary* server gets time from a time source and provides time to secondary servers or directly to clients.

A *secondary* server receives time from a primary server and can be configured to furnish time to a set of clients to offload the primary or for redundancy. The presence of a secondary server on the network is optional but highly recommended.

A *peer* reciprocates time with an NTP server. All peers work at the same stratum level, and all of them are considered equally reliable. Both primary and secondary servers can be peers of each other.

A *client* receives time from a primary or a secondary server and adjusts its clock accordingly.

Stratum Levels

As mentioned, there are different types of time sources available so you can synchronize the system clock. These time sources are categorized hierarchically into several levels that are referred to as *stratum levels* based on their distance from the reference clocks (atomic, radio, and GPS). The reference clocks operate at stratum level 0 and are the most accurate provider of time with little to no delay. Besides stratum 0, there are fifteen additional levels that range from 1 to 15. Of these, servers operating at stratum 1 are considered perfect, as they get time updates directly from a stratum 0 device. See Figure 18-1 for a sample hierarchy.

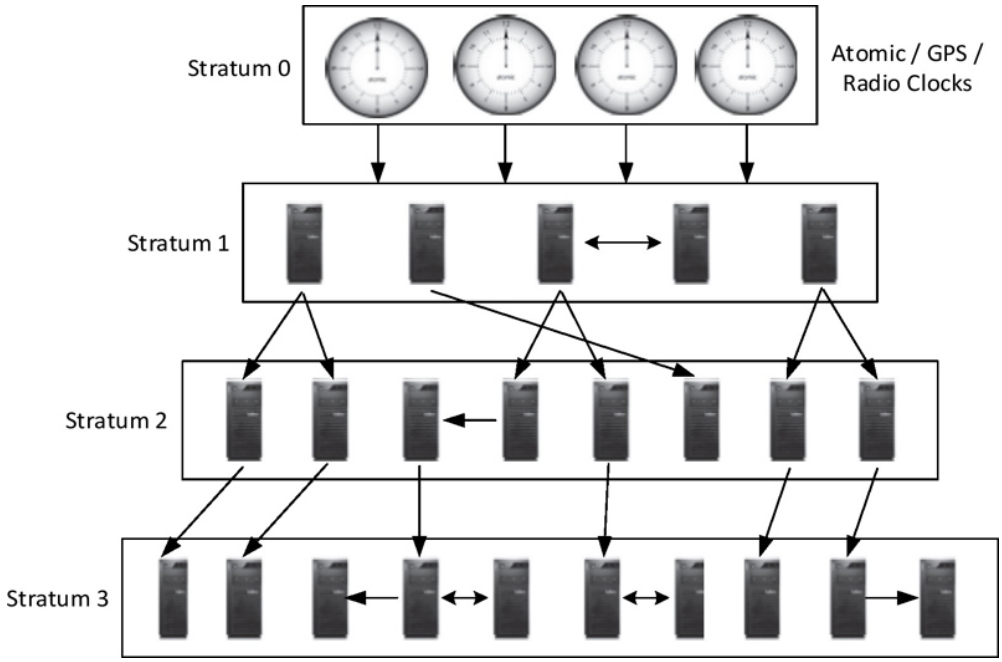


Figure 18-1 NTP Stratum Levels

A stratum 0 device cannot be used on the network directly. It is attached to a computer, which is then configured to operate at stratum 1. Servers functioning at stratum 1 are called *time servers* and they can be set up to deliver time to stratum 2 servers. Similarly, a stratum 3 server can be configured to synchronize its time with a stratum 2 server and deliver time to the next lower level servers, and so on. Servers sharing the same stratum can be configured as peers to exchange time updates with one another.

If a secondary server also gets time updates directly from a stratum 1 server, it will act as a peer to the primary server.

There are a number of public NTP servers available for free that synchronize time. They normally operate at higher stratum levels such as 2 and 3.

Chrony Configuration File

The key configuration file for the Chrony service is *chrony.conf* located in the */etc* directory. This file is referenced by the Chrony daemon at startup to determine the sources to synchronize the clock, the log file location, and other details. This file can be modified by hand to set or alter directives as required. Some common directives used in this file along with real or mock values are presented below with an explanation in Table 18-1:

driftfile	/var/lib/chrony/drift
logdir	/var/log/chrony
pool	0.rhel.pool.ntp.org iburst
server	server20s8.example.com iburst
server	127.127.1.0
peer	prodntp1.abc.net

Table 18-1 describes these directives.

Directive	Description
driftfile	Indicates the location and name of the drift file to be used to record the rate at which the system clock gains or losses time. This data is used by Chrony to maintain local system clock accuracy.
logdir	Sets the directory location to store the log files in
pool	Defines the hostname that represents a pool of time servers. Chrony binds itself with one of the servers to get updates. In case of a failure of that server, it automatically switches the binding to another server within the pool. The iburst option dictates the Chrony service to send the first four update requests to the time server every 2 seconds. This allows the daemon to quickly bring the local clock closer to the time server at startup.
server	Defines the hostname or IP address of a single time server. The IP 127.127.1.0 is a special address that epitomizes the local system clock.
peer	Identifies the hostname or IP address of a time server running at the same stratum level. A peer provides time to a server as well as receives time from the same server.

Table 18-1 Chrony Directives

There are plenty of other directives and options available with Chrony that may be defined in this file. Use **man chrony.conf** for details.

Chrony Daemon and Command

The Chrony service runs as a daemon program called *chronyd* that handles time synchronization in the background. It uses the configuration defined in the */etc/chrony.conf* file at startup and sets its behavior accordingly. If the local clock requires a time adjustment, Chrony takes multiple small steps toward minimizing the gap rather than doing it abruptly in a single step. There are a number of additional options available that may be passed to the service daemon if required.

The Chrony service has a command line program called *chronyc* available that can be employed to monitor the performance of the service and control its runtime behavior. There are a few subcommands available with *chronyc*; the *sources* and *tracking* subcommands list current sources of time and view performance statistics, respectively.

Exercise 18-1: Configure NTP Client

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will install the Chrony software package and activate the service without making any changes to the default configuration. You will validate the binding and operation.

1. Install the Chrony package using the *dnf* command:

```
[user1@server10 ~]$ sudo dnf -y install chrony

 Package chrony-3.3-3.el8.x86_64 is already installed.
 Dependencies resolved.
 Nothing to do.
 Complete!
```

The software is already installed on the system.

2. Ensure that preconfigured public time server entries are present in the */etc/chrony.conf* file:

```
[user1@server10 ~]$ grep -E 'pool|server' /etc/chrony.conf | grep -v ^#
pool 2.rhel.pool.ntp.org iburst
```

There is a single pool entry set in the file by default. This pool name is backed by multiple NTP servers behind the scene.

3. Start the Chrony service and set it to autostart at reboots:

```
[user1@server10 ~]$ sudo systemctl --now enable chronyd
Created symlink /etc/systemd/system/multi-user.target.wants/chronyd.service → /usr/li
b/systemd/system/chronyd.service.
```

4. Examine the operational status of Chrony:

```
[user1@server10 ~]$ sudo systemctl status chronyd --no-pager
● chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor preset: e
nabled)
   Active: active (running) since Mon 2019-11-18 10:33:29 EST; 2min 2s ago
     Docs: man:chronyd(8)
           man:chrony.conf(5)
  Process: 11145 ExecStartPost=/usr/libexec/chrony-helper update-daemon (code=exited,
 status=0/SUCCESS)
  Process: 11141 ExecStart=/usr/sbin/chronyd $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 11143 (chronyd)
    Tasks: 1 (limit: 5076)
   Memory: 2.9M
   CGroup: /system.slice/chronyd.service
           └─11143 /usr/sbin/chronyd

Nov 18 10:33:28 server10.example.com systemd[1]: Starting NTP client/server...
Nov 18 10:33:29 server10.example.com chronyd[11143]: chronyd version 3.3 starting…UG)
Nov 18 10:33:29 server10.example.com chronyd[11143]: Frequency -39.266 +/- 1.653 …ift
Nov 18 10:33:29 server10.example.com chronyd[11143]: Using right/UTC timezone to …ata
Nov 18 10:33:29 server10.example.com systemd[1]: Started NTP client/server.
Nov 18 10:33:34 server10.example.com chronyd[11143]: Selected source 50.101.251.61
Nov 18 10:33:34 server10.example.com chronyd[11143]: System clock TAI offset set …nds
```

The service has started successfully and it is set for autostart.

5. Inspect the binding status using the *sources* subcommand with *chronyc*:

```
[user1@server10 ~]$ chronyc sources
210 Number of sources = 4
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^+ 64.ip-54-39-23.net            3   6   377    17   +147us[ +147us] +/-   12ms
^- up2.com                       2   6   377    17    +59us[  +59us] +/-   40ms
^+ bras-vprn-toroon2638w-lp>     2   6   377    82    -88us[ -107us] +/-   19ms
^* gpg.n1zyy.com                 2   6   377    18   -311us[ -344us] +/-   12ms
```

The output shows the number of available time sources in row 1 and rest of the information in eight columns. Columns 1 to 4—M, S, Name/IP, and Stratum—illustrate the mode, state, name/IP, and stratum level of the source. The ^ means server and the * implies current association.

Columns 4 to 8—Poll, Reach, LastRx, and Last Sample—display the polling rate (6 means 64 seconds), reachability register (377 indicates a valid response was received), how long ago the last sample was received, and the offset between the local clock and the source at the last

measurement. Check out the manual pages of the *chronyc* command and search for the section ‘sources’ for additional details.

The last line in the output depicts the *server10* binding with time server *gpg.n1zyy.com*. This association is identified with the asterisk character (*) beside the time server.

6. Display the clock performance using the *tracking* subcommand with *chronyc*:

```
[user1@server10 ~]$ chronyc tracking
Reference ID    : C0630208 (gpg.n1zyy.com)
Stratum         : 3
Ref time (UTC)  : Mon Nov 18 18:06:47 2019
System time     : 0.000128811 seconds fast of NTP time
Last offset     : +0.000033660 seconds
RMS offset      : 0.000096103 seconds
Frequency       : 39.044 ppm slow
Residual freq   : +0.001 ppm
Skew            : 0.051 ppm
Root delay      : 0.017188223 seconds
Root dispersion : 0.003284285 seconds
Update interval : 1041.1 seconds
Leap status     : Normal
```

Lines 1 and 2 in the above output identify the current source of time (Reference ID) and the stratum level it is configured at (Stratum). Line 3 shows the reference time at which the last measurement from the time source was processed (Ref time). Line 4 displays the local time offset from NTP time (System time). Line 5 depicts the last reported offset from the NTP server (Last offset). Line 6 identifies the frequency at which time adjustments are occurring (Frequency). The rest of the lines in the output show additional information. Check out the manual pages of the *chronyc* command and search for the section “tracking” for additional details.

EXAM TIP: You will not have access to the outside network during the exam, so you will need to point your system to an NTP server available on the exam network. Simply comment the default server/pool directive(s) and add a single directive “server

<hostname>” to the file. Replace <hostname> with the NTP server name or its IP address as provided.

The concludes the exercise.

Displaying and Setting System Date and Time

System date and time can be viewed and manually adjusted with native Linux tools such as the *timedatectl* command. This command can modify the date, time, and time zone. When executed without any option, as shown below, it outputs the local time, Universal time, RTC time (*real-time clock*, a battery-backed hardware clock located on the system board), time zone, and the status of NTP:

```
[user1@server10 ~]$ timedatectl
               Local time: Mon 2019-11-18 17:50:53 EST
           Universal time: Mon 2019-11-18 22:50:53 UTC
                 RTC time: Mon 2019-11-18 22:50:46
                Time zone: America/New_York (EST, -0500)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

This command requires that the NTP/Chrony service is deactivated in order to make time adjustments. Run the *timedatectl* command as follows to turn off NTP and verify:

```
[user1@server10 ~]$ sudo timedatectl set-ntp false
[user1@server10 ~]$ timedatectl |grep NTP
              NTP service: inactive
```

To modify the current date to January 1, 2020, and confirm:

```
[user1@server10 ~]$ sudo timedatectl set-time 2020-1-1
[user1@server10 ~]$ timedatectl
               Local time: Wed 2020-01-01 00:00:05 EST
           Universal time: Wed 2020-01-01 05:00:05 UTC
                 RTC time: Wed 2020-01-01 05:00:05
                Time zone: America/New_York (EST, -0500)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
```

To change the time to 11:20 p.m. and date to November 18, 2019:

```
[user1@server10 ~]$ sudo timedatectl set-time "2019-11-18 23:00"
[user1@server10 ~]$ timedatectl
               Local time: Mon 2019-11-18 23:00:06 EST
           Universal time: Tue 2019-11-19 04:00:06 UTC
                 RTC time: Tue 2019-11-19 04:00:06
                Time zone: America/New_York (EST, -0500)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
```

To reactivate NTP:

```
[user1@server10 ~]$ sudo timedatectl set-ntp true
[user1@server10 ~]$ timedatectl | grep NTP
              NTP service: active
```

Check out the manual pages of the *timedatectl* command for more subcommands and usage examples.

Alternatively, you can use the *date* command to view or modify the system date and time.

To view current date and time:

```
[user1@server10 ~]$ date
Tue Nov 19 11:01:08 EST 2019
```

To change the date and time to November 22, 2019 1:00 p.m.:

```
[user1@server10 ~]$ sudo date --set "2019-11-22 13:00"
Fri Nov 22 13:00:00 EST 2019
```

There are many options available with the *date* command. Consult its manual pages for details.

DNS and Name Resolution

Domain Name System (DNS) is an inverted tree-like structure employed on the Internet and private networks (including home and corporate networks) as the de facto standard for resolving hostnames to their numeric IP addresses. DNS is platform-independent with support integrated in every

operating system. DNS is also referred to as BIND, *Berkeley Internet Name Domain*, which is an implementation of DNS, and it has been the most popular DNS application in use. *Name resolution* is the technique that uses DNS/BIND for hostname lookups.

In order to understand DNS, a brief discussion of its components and roles is imperative. The following subsections provide a look at the client-side configuration files and commands, along with examples on how to use the tools for resolving hostnames.

DNS Name Space and Domains

The DNS *name space* is a hierarchical organization of all the domains on the Internet. The root of the name space is represented by a period (.). The hierarchy below the root (.) denotes the *top-level domains* (TLDs) with names such as .com, .net, .edu, .org, .gov, .ca, and .de. A DNS *domain* is a collection of one or more systems. Subdomains fall under their parent domains and are separated by a period (.). For example, *redhat.com* is a second-level subdomain that falls under .com, and *bugzilla.redhat.com* is a third-level subdomain that falls under *redhat.com*.

Figure 18-2 exhibits a sample hierarchy of the name space, showing the top three domain levels.

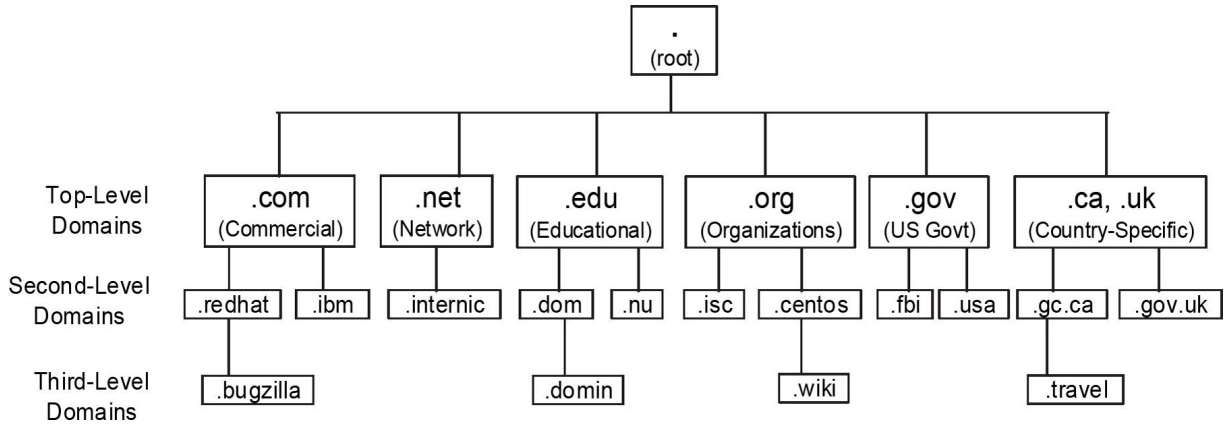


Figure 18-2 Sample DNS Hierarchy

At the deepest level of the hierarchy are the *leaves* (systems, nodes, or any device with an IP address) of the name space. For example, a network switch *net01* in *.travel.gc.ca* subdomain will be known as *net01.travel.gc.ca*. If a period (.) is added to the end of this name to look like *net01.travel.gc.ca.*, it will be referred to as the *Fully Qualified Domain Name* (FQDN) for *net01*.

DNS Roles

From a DNS perspective, a system can be configured to operate as a primary server, secondary server, or client. A DNS server is also referred to as a *nameserver*.

A *primary* (a.k.a. *master*) *server* is responsible for its domain (or subdomain). It maintains a master database of all the hostnames and their associated IP addresses that are included in that domain. Any changes in the database is done on this server. Each domain must have one primary server with one or more optional *secondary* (a.k.a. *slave*) servers for load balancing and redundancy. A secondary server also stores an updated copy of the master database and it continues to provide name resolution service in the event the primary server becomes unavailable or inaccessible.

A *DNS client* queries nameservers for name lookups. Every system with access to the Internet or other external networks will have the DNS client functionality configured and operational. Setting up DNS client on Linux involves two text files that are discussed in the next two subsections.

Understanding Resolver Configuration File

The *resolv.conf* file under */etc* is the DNS resolver configuration file where information to support hostname lookups is defined. This file may be edited manually with a text editor. It is referenced by resolver utilities to obtain necessary information to construct and transmit queries.

There are three key directives set in this file—domain, nameserver, and search—and they are described in Table 18-2.

Directive	Description
domain	Identifies the default domain name to be searched for queries
nameserver	Declares up to three DNS server IP addresses to be queried one at a time in the order in which they are listed. Nameserver entries may be defined as separate line items with the directive or on a single line.
search	Specifies up to six domain names, of which the first must be the local domain. No need to define the domain directive if the search directive is used.

Table 18-2 The Resolver Configuration File

A sample entry showing the syntax is provided below for reference:

```
domain      example.com
search      example.net example.org example.edu example.gov
nameserver  192.168.0.1 8.8.8.8 8.8.4.4
```

A variation of the above would be:

```
domain      example.com
search      example.net example.org example.edu example.gov
nameserver  192.168.0.1
nameserver  8.8.8.8
nameserver  8.8.4.4
```

Currently, there are two entries “search example.com” and “nameserver 192.168.0.1” defined in the *resolv.conf* file on *server10* and *server20*.

```
[user1@server10 ~]$ cat /etc/resolv.conf
# Generated by NetworkManager
search example.com
nameserver 192.168.0.1
```

On a system with this file absent, the resolver utilities only query the nameserver configured on the localhost, determine the domain name from the hostname of the system, and construct the search path based on the domain name.

Viewing and Adjusting Name Resolution Sources and Order

The *nsswitch.conf* file under */etc* directs the lookup utilities to the correct source to get hostname information. In the presence of multiple sources, this file also identifies the order in which to consult them and an action to be taken next. There are four keywords—success, notfound, unavail, and tryagain—that oversee this behavior, and are described along with default actions in Table 18-3.

Keyword	Meaning	Default Action
success	Information found in source and provided to the requester	return (do not try the next source)
notfound	Information not found in source	continue (try the next source)
unavail	Source down or not responding; service disabled or not configured	continue (try the next source)
tryagain	Source busy, retry later	continue (try the next source)

Table 18-3 Name Service Source and Order Determination

The following example entry shows the syntax of a relevant entry from the *nsswitch.conf* file. It shows two sources for name resolution: files (*/etc/hosts*) and DNS (*/etc/resolv.conf*).

```
hosts:      files      dns
```

Based on the default behavior, the search will terminate if the requested information is found in the *hosts* table. However, you can alter this behavior and instruct the lookup programs to return if the requested information is not found there. The modified entry will look like:

```
hosts:      files [notfound=return]     dns
```

This altered entry will ignore the DNS.

See Chapter 17 for details on the /etc/hosts file.

Once the *resolv.conf* and *nsswitch.conf* files are configured appropriately, you can use any of the native client resolver tools for lookups. Common query tools available in RHEL 8 include *dig*, *host*, *nslookup*, and *getent*.

Performing Name Resolution with dig

dig (*domain information groper*) is a DNS lookup utility. It queries the nameserver specified at the command line or consults the *resolv.conf* file to determine the nameservers to be queried. This tool may be used to troubleshoot DNS issues due to its flexibility and verbosity. The following shows a few usage examples.

To get the IP for *redhat.com* using the nameserver listed in the *resolv.conf* file:

```
[user1@server10 ~]$ dig redhat.com

........
;; ANSWER SECTION:
redhat.com.             3152    IN      A       209.132.183.105

;; Query time: 13 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Tue Nov 19 09:38:50 EST 2019
;; MSG SIZE  rcvd: 55
```

The output shows the total time (13 milliseconds) it took to get the result, the IP address (209.132.183.105) of

redhat.com, the nameserver IP (192.168.0.1) used for the query, the DNS port number (53), query timestamp, the size of the received message, and other information.

To perform a reverse lookup on the redhat.com IP (209.132.183.105), use the -x option with the command:

```
[user1@server10 ~]$ dig -x 209.132.183.105
........
;; ANSWER SECTION:
105.183.132.209.in-addr.arpa. 600 IN    PTR     redirect.redhat.com.

;; AUTHORITY SECTION:
183.132.209.in-addr.arpa. 600   IN      NS      ns2.redhat.com.
183.132.209.in-addr.arpa. 600   IN      NS      ns3.redhat.com.
183.132.209.in-addr.arpa. 600   IN      NS      ns4.redhat.com.
183.132.209.in-addr.arpa. 600   IN      NS      ns1.redhat.com.

;; Query time: 67 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Tue Nov 19 09:46:38 EST 2019
;; MSG SIZE  rcvd: 162
```

Reference the command's manual pages for details and options.

Performing Name Resolution with host

host is an elementary DNS lookup utility that works on the same principles as the *dig* command in terms of nameserver determination. This tool produces lesser data in the output by default; however, you can add the -v option for verbosity.

To perform a lookup on *redhat.com*:

```
[user1@server10 ~]$ host redhat.com
redhat.com has address 209.132.183.105
redhat.com mail is handled by 10 us-smtp-inbound-2.mimecast.com.
redhat.com mail is handled by 10 us-smtp-inbound-1.mimecast.com.
```

Rerun the above with -v added. The output will be similar to that of the *dig* command.

To perform a reverse lookup on the IP of *redhat.com* using the -v flag to add details:

```
[user1@server10 ~]$ host -v 209.132.183.105
Trying "105.183.132.209.in-addr.arpa"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31347
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;105.183.132.209.in-addr.arpa.  IN      PTR

;; ANSWER SECTION:
105.183.132.209.in-addr.arpa. 595 IN    PTR     redirect.redhat.com.

Received 79 bytes from 192.168.0.1#53 in 4 ms
```

Refer to the command's manual pages for options and more information.

Performing Name Resolution with nslookup

nslookup queries the nameservers listed in the *resolv.conf* file or specified at the command line. The following shows a few usage examples.

To get the IP for *redhat.com* using nameserver 8.8.8.8 instead of the nameserver defined in *resolv.conf*:

```
[user1@server10 ~]$ nslookup redhat.com 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   redhat.com
Address: 209.132.183.105
```

To perform a reverse lookup on the IP of *redhat.com* using the nameserver from the resolver configuration file:

```
[user1@server10 ~]$ nslookup 209.132.183.105
105.183.132.209.in-addr.arpa    name = redirect.redhat.com.
```

Consult the command's manual pages on how to use it in interactive mode.

Performing Name Resolution with getent

The *getent* (*get entries*) command is a rudimentary tool that can fetch matching entries from the databases defined in the

nsswitch.conf file. This command reads the corresponding database and displays the information if found. For name resolution, use the *hosts* database and *getent* will attempt to resolve the specified hostname or IP address. For instance, run the following for forward and reverse lookups:

```
[user1@server10 ~]$ getent hosts redhat.com
209.132.183.105 redhat.com
[user1@server10 ~]$ getent hosts 209.132.183.105
209.132.183.105 redirect.redhat.com
```

Check the command's manual pages for available flags and additional information.

Chapter Summary

The focus of this chapter was on two topics: network time synchronization and hostname resolution.

The chapter began with a discussion of Network Time Protocol, what role it plays in keeping the clocks synchronized, and what is its relationship with the Chrony service. We explored various sources for obtaining time, different roles that systems could play, and the strata paradigm. We analyzed the configuration file to understand some key directives and their possible settings. We performed an exercise to configure the service, display clock association, and analyze the results. We also employed a couple of other RHEL tools to display the system time and set it instantly.

We concluded the chapter with a deliberation of DNS and name resolution. We discussed the concepts and roles, analyzed the resolver configuration file, and examined the source/order determination file. We added required entries to the resolver configuration file and tested the functionality by employing client tools for hostname lookup.

Check Your Understanding

1. Chrony is an implementation of the Network Time Protocol. True or False?
2. What is the name and location of the DNS resolver file?
3. What stratum level do two peer systems operate on a network?
4. Provide the maximum number of nameservers that can be defined in the resolver configuration file.
5. What is the purpose of a drift file in Chrony/NTP?
6. What is a relative distinguished name?
7. What would you add to the Chrony configuration file if you want to use the local system clock as the provider of time?
8. BIND is an implementation of Domain Name System. True or False?
9. Define time source.
10. Name the three DNS roles that a RHEL system can play.
11. What would you run to check the NTP bind status with time servers?
12. Define DNS name space.
13. Name the four Chrony/NTP roles that a RHEL system can play.
14. Which file defines the name resolution sources and controls the order in which they are consulted?
15. What is the filename and directory location for the Chrony configuration file?
16. The Chrony client is preconfigured when the *chrony* software package is installed. You just need to start the service to synchronize the clock. True or False?
17. List any three DNS lookup tools.
18. List two utilities that you can use to change system time.

Answers to Check Your Understanding

1. True.
2. The DNS resolver file is called *resolv.conf* and it is located in the */etc* directory.
3. Two peers on a network operate at the same stratum level.
4. Three.
5. The purpose of a drift file is to keep track of the rate at which the system clock gains or losses time.
6. A relative distinguished name represents individual components of a distinguished name.
7. You will add “server 127.127.1.0” to the Chrony configuration file and restart the service.
8. True.
9. A time source is a reference device that provides time to other devices.
10. From a DNS perspective, a RHEL machine can be a primary server, a secondary server, or a client.
11. You will run *chronyc sources* to check the binding status.
12. DNS name space is a hierarchical organization of all the domains on the Internet.
13. From a Chrony/NTP standpoint, a RHEL machine can be a primary server, a secondary server, a peer, or a client.
14. The */etc/nsswitch.conf* file.
15. The name of the Chrony configuration file is *chrony.conf* and it is located in the */etc* directory.
16. True.
17. Name resolution tools are *dig*, *host*, *getent*, and *nslookup*.
18. The *timedatectl* and *date* commands can be used to modify the system time.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 18-1: Modify System Date and Time

As *user1* with *sudo* on *server10*, execute the *date* and *timedatectl* commands to check the current system date and time. Identify the distinctions between the two outputs. Use *timedatectl* and change the system date to a future date. Issue the *date* command and change the system time to one hour ahead of the current time. Observe the new date and time with both commands. Reset the date and time to the current actual time using either the *date* or the *timedatectl* command. (Hint: Time Synchronization).

Lab 18-2: Configure Chrony

As *user1* with *sudo* on *server10*, install Chrony and mark the service for autostart on reboots. Edit the Chrony configuration file and comment all line entries that begin with “pool” or “server”. Go to the end of the file, and add a new line “server 127.127.1.0”. Start the Chrony service and run **chronyc sources** to confirm the binding. (Hint: Time Synchronization).

Chapter 19

The Secure Shell Service

This chapter describes the following major topics:

- Understand the OpenSSH service, versions, and algorithms
- Overview of encryption techniques and authentication methods
- Describe OpenSSH administration commands and configuration files
- Configure private/public key-based authentication
- Access OpenSSH server from Windows and other Linux systems
- Use OpenSSH client tools to transfer files
- Synchronize files remotely over OpenSSH

RHCSA Objectives:

04. Access remote systems using ssh

21. Securely transfer files between systems
52. Configure key-based authentication for SSH

Secure Shell is a network service that delivers a secure mechanism for data transmission between source and destination systems over insecure network paths. It provides a set of utilities that allows users to generate key pairs and use them to set up trusted logins between systems for themselves. Additional utilities in the set gives remote users the ability to log in, execute commands, and transfer files securely over encrypted network channels. These tools have predominantly supplanted their insecure counterparts in the corporate world.

The OpenSSH Service

Secure Shell (SSH) delivers a secure mechanism for data transmission between source and destination systems over IP networks. It was designed to replace the old remote login programs that transmitted user passwords in clear text and data unencrypted. SSH employs digital signatures for user authentication with encryption to secure a communication channel. As a result, this makes it extremely hard for unauthorized people to gain access to passwords or the data in transit. It also monitors the data being transferred throughout a session to ensure integrity. SSH includes a set of utilities for remote users to log in, transfer files, and execute commands securely. Due to strong security features, SSH utilities have supplanted their conventional, unsecure login and file transfer counterpart programs.

OpenSSH is a free, open source implementation of the proprietary SSH. Once applied successfully on the system, the unsecure services—*telnet*, *rlogin*, *rcp*, and *ftp*—can be disabled after a careful examination to eliminate potential impact. The secure command that has substituted *telnet* and

rlogin remote login services is called *ssh*, and those for *rcp* and *ftp* are called *scp* and *sftp*, respectively.

Common Encryption Techniques

Encryption is a way of scrambling information with the intent to conceal the real information from unauthorized access. OpenSSH can utilize various encryption techniques during an end-to-end communication session between two entities (client and server). The two common techniques are *symmetric* and *asymmetric*. They are also referred to as *secret key encryption* and *public key encryption* techniques.

Symmetric Technique

This technique uses a single key called a *secret* key that is generated as a result of a negotiation process between two entities at the time of their initial contact. Both sides use the same secret key during subsequent communication for data encryption and decryption.

Asymmetric Technique

This technique uses a combination of *private* and *public* keys, which are randomly generated and mathematically related strings of alphanumeric characters attached to messages being exchanged. The client transmutes the information with a *public* key and the server decrypts it with the paired *private* key. The private key must be kept secure since it is private to a single sender; the public key is disseminated to clients. This technique is used for channel encryption as well as user authentication.

Authentication Methods

Once an encrypted channel is established between the client and server, additional negotiations take place between the two to authenticate the user trying to access the server. OpenSSH offers several methods for this purpose; they are

listed below in the order in which they are attempted during the authentication process:

- GSSAPI-based (*Generic Security Service Application Program Interface*) authentication
- Host-based authentication
- Public key-based authentication
- Challenge-response authentication
- Password-based authentication

Let’s review each one in detail.

GSSAPI-Based Authentication

GSSAPI provides a standard interface that allows security mechanisms, such as Kerberos, to be plugged in. OpenSSH uses this interface and the underlying Kerberos for authentication. With this method, an exchange of tokens takes place between the client and server to validate user identity.

Host-Based Authentication

This type of authentication allows a single user, a group of users, or all users on the client to be authenticated on the server. A user may be configured to log in with a matching username on the server or as a different user that already exists there. For each user that requires an automatic entry on the server, a *~/.shosts* file is set up containing the client name or IP address, and, optionally, a different username.

The same rule applies to a group of users or all users on the client that require access to the server. In that case, the setup is done in the */etc/ssh/shosts.equiv* file on the server.

Private/Public Key-Based Authentication

This method uses a private/public key combination for user authentication. The user on the client has a public key and the

server stores the corresponding private key. At the login attempt, the server prompts the user to enter the passphrase associated with the key and logs the user in if the passphrase and key are validated.

Challenge-Response Authentication

This method is based on the response(s) to one or more arbitrary challenge questions that the user has to answer correctly in order to be allowed to log in to the server.

Password-Based Authentication

This is the last fall back option. The server prompts the user to enter their password. It checks the password against the stored entry in the *shadow* file and allows the user in if the password is confirmed.

Of the five authentication methods, the password-based method is common and requires no further explanation. The GSSAPI-based, host-based, and challenge-response methods are beyond the scope of this book. The public/private authentication and encryption methods will be the focus in the remainder of this chapter.

OpenSSH Protocol Version and Algorithms

OpenSSH has evolved over the years. Its latest and the default version in RHEL 8, version 2, has numerous enhancements, improvements, and sophisticated configuration options. It supports various algorithms for data encryption and user authentication (digital signatures) such as RSA, DSA, and ECDSA. RSA is more common and it is widely employed partly because it supports both encryption and authentication. In contrast, DSA and ECDSA are restricted to authentication only. These algorithms are used to generate public and private key pairs for the asymmetric technique.

RSA stands for *Rivest-Shamir-Adleman*, who first published this algorithm, DSA for *Digital Signature Algorithm*, and ECDSA is an acronym for *Elliptic Curve Digital Signature Algorithm*.

OpenSSH Packages

OpenSSH has three software packages that are of interest. These are *openssh*, *openssh-clients*, and *openssh-server*. The *openssh* package provides the *ssh-keygen* command and some library routines; the *openssh-clients* package includes commands, such as *scp*, *sftp*, *ssh*, and *ssh-copy-id*, and a client configuration file */etc/ssh/ssh_config*; and the *openssh-server* package contains the *sshd* service daemon, server configuration file */etc/ssh/sshd_config*, and library routines. By default, all three packages are installed during OS installation.

OpenSSH Server Daemon and Client Commands

The OpenSSH server program *sshd* is preconfigured and operational on new RHEL installations, allowing remote users to log in to the system using an ssh client program such as PuTTY or the *ssh* command. This daemon listens on well-known TCP port 22 as documented in the */etc/ssh/sshd_config* file with the Port directive.

The client software includes plenty of utilities such as those listed and described in Table 19-1.

Command	Description
scp	The secure remote copy command that replaced the non-secure rcp command
sftp	The secure remote copy command that replaced the non-secure ftp command
ssh	The secure remote login command that replaced non-secure telnet and rlogin commands
ssh-copy-id	Copies public key to remote systems
ssh-keygen	Generates and manages private and public key pairs

Table 19-1 OpenSSH Client Tools

The use of these commands is demonstrated in the following subsections.

Server Configuration File

The OpenSSH server daemon *sshd* has a configuration file that defines default global settings on how it should operate. This file is located in the */etc/ssh* directory and called *sshd_config*. There are a number of directives preset in this file that affect all inbound ssh communication and are tuned to work as-is for most use cases. In addition, the */var/log/secure* log file is used to capture authentication messages.

A few directives with their default values from the *sshd_config* file are displayed below:

```
[user1@server10 ~]$ sudo cat /etc/ssh/sshd_config

#Port                          22
#Protocol                      2
ListenAddress                  0.0.0.0
SyslogFacility                 AUTHPRIV
#LogLevel                      INFO
PermitRootLogin                yes
#PubkeyAuthentication          yes
AuthorizedKeysFile             .ssh/authorized_keys
PasswordAuthentication         yes
```

```
#PermitEmptyPasswords                   no
ChallengeResponseAuthentication         no
UsePAM                                  yes
X11Forwarding                           yes
```

The above directives are elaborated in Table 19-2.

Directive	Description
Port	Specifies the port number to listen on. Default is 22.
Protocol	Specifies the default protocol version to use.
ListenAddress	Sets the local addresses the sshd service should listen on. Default is to listen on all local addresses.
SyslogFacility	Defines the facility code to be used when logging messages to the /var/log/secure file. This is based on the configuration in the /etc/rsyslog.conf file. Default is AUTHPRIV.
LogLevel	Identifies the level of criticality for the messages to be logged. Default is INFO.
PermitRootLogin	Allows or disallows the root user to log in directly to the system. Default is yes.
PubKeyAuthentication	Enables or disables public key-based authentication. Default is yes.
AuthorizedKeysFile	Sets the name and location of the file containing a user's authorized keys. Default is ~/.ssh/authorized_keys.
PasswordAuthentication	Enables or disables local password authentication. Default is yes.
PermitEmptyPasswords	Allows or disallows the use of null passwords. Default is no.
ChallengeResponseAuthentication	Enables or disables challenge-response authentication mechanism. Default is yes.
UsePAM	Enables or disables user authentication via PAM. If enabled, only root will be able to run the sshd daemon. Default is yes.

X11Forwarding	Allows or disallows remote access to graphical applications. Default is yes.

Table 19-2 OpenSSH Server Configuration File

There are many more settings available that may be added to the file for additional control. Check out the manual pages of the *sshd_config* file (**man 5 sshd_config**) for details.

Client Configuration File

Each RHEL client machine that uses ssh to access a remote OpenSSH server has a local configuration file that directs how the client should behave. This file, *ssh_config*, is located in the */etc/ssh* directory. There are a number of directives preset in this file that affect all outbound ssh communication and are tuned to work as-is for most use cases.

A few directives with their default values from the *ssh_config* file are displayed below:

```
[user1@server10 ~]$ sudo more /etc/ssh/ssh_config

# Host *
# ForwardX11                    no
# PasswordAuthentication        yes
# StrictHostKeyChecking         ask
# IdentityFile                  ~/.ssh/id_rsa
# IdentityFile                  ~/.ssh/id_dsa
# Port                          22
# Protocol                      2
```

The above directives are described in Table 19-3.

Directive	Description
Host	Container that declares directives applicable to one host, a group of hosts, or all hosts. It ends when another occurrence of Host or Match is encountered. Default is *, which sets global defaults for all hosts.
ForwardX11	Enables or disables automatic redirection of X11 traffic over SSH connections. Default is no.
PasswordAuthentication	Allows or disallows password authentication. Default is yes.
StrictHostKeyChecking	Controls (1) whether to add host keys (host fingerprints) to ~/.ssh/known_hosts when accessing a host for the first time, and (2) what to do when the keys of a previously-accessed host mismatch with what is stored in ~/.ssh/known_hosts. Options are: **no:** adds new host keys and ignores changes to existing keys. **yes:** adds new host keys and disallows connections to hosts with non-matching keys. **accept-new:** adds new host keys and disallows connections to hosts with non-matching keys. **ask (default):** prompts whether to add new host keys and disallows connections to hosts with non-matching keys.
IdentityFile	Defines the name and location of a file that stores a user’s private key for their identity validation. Defaults are id_rsa, id_dsa, and id_ecdsa based on the type of algorithm used. Their corresponding public key files with .pub extension are also stored at the same directory location.
Port	Sets the port number to listen on. Default is 22.
Protocol	Specifies the default protocol version to use

Table 19-3 OpenSSH Client Configuration File

The *~/.ssh* directory does not exist by default; it is created when a user executes the *ssh-keygen* command for the first time to generate a key pair or connects to a remote ssh server and accepts its host key for the first time. In the latter case, the client stores the server's host key locally in a file called *known_hosts* along with its hostname or IP address. On subsequent access attempts, the client will use this information to verify the server's authenticity.

There are a lot more settings available that may be added to the file for additional control. Check out the manual pages of the *ssh_config* file (**man 5 ssh_config**) for details.

System Access and File Transfer

A user must log in to the Linux system in order to use it or transfer files. The login process identifies the user to the system. For accessing a RHEL system remotely, use the *ssh* command, and the *scp* or the *sftp* command for copying files. These commands require either a resolvable hostname of the target system or its IP address in order to try to establish a connection. All these commands are secure and may be used over secure and unsecure network channels to exchange data.

The following subsections and exercises describe multiple access scenarios including accessing a RHEL system (*server20*) from another RHEL system (*server10*) and a Windows computer, accessing a RHEL system (*server20*) using ssh keys, and transferring files using *scp* and *sftp*.

Exercise 19-1: Access RHEL System from Another RHEL System

This exercise should be done on *server10* and *server20* as *user1* with *sudo* where required.

This exercise works under two assumptions: (1) *user1* exists on both *server10* and *server20*, and (2) hostname and IP mapping is in place in the */etc/hosts* file (Chapter 17). Use the IP address in lieu of the hostname if the mapping is unavailable for *server20*.

In this exercise, you will issue the *ssh* command as *user1* on *server10* to log in to *server20*. You will run appropriate commands on *server20* for validation. You will log off and return to the originating system.

1. Issue the *ssh* command while logged in as *user1* on *server10*:

```
[user1@server10 ~]$ ssh server20
The authenticity of host 'server20 (192.168.0.120)' can't be established.
ECDSA key fingerprint is SHA256:fDOVHpHQYEOMrB25UGKNEkWONABCPBHfSnb6y8nnbhO.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'server20,192.168.0.120' (ECDSA) to the list of known hosts.
user1@server20's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Fri Nov 22 11:02:49 2019 from 192.168.0.110
[user1@server20 ~]$
```

Answer ‘yes’ to the question presented and press Enter to continue. This step adds the hostname of *server20* to a file called *known_hosts* under */home/user1/.ssh* directory on the originating computer (*server10*). This message will not reappear on subsequent login attempts to *server20* for this user. Enter the correct password for *user1* to be allowed in. You will be placed in the home directory of *user1* on *server20*. The command prompt will reflect that information.

2. Issue the basic Linux commands *whoami*, *hostname*, and *pwd* to confirm that you are logged in as *user1* on *server20* and placed in the correct home directory:

```
[user1@server20 ~]$ whoami
user1
[user1@server20 ~]$ hostname
server20.example.com
[user1@server20 ~]$ pwd
/home/user1
[user1@server20 ~]$
```

3. Run the *logout* or the *exit* command or simply press the key combination Ctrl+d to log off *server20* and return to *server10*:

```
[user1@server20 ~]$ exit
logout
Connection to server20 closed.
[user1@server10 ~]$
```

This concludes the exercise.

If you wish to log on as a different user such as *user2* (assuming *user2* exists on the target server *server20*), you may run the *ssh* command in either of the following ways:

```
[user1@server10 ~]$ ssh -l user2 server20
[user1@server10 ~]$ ssh user2@server20
```

The above will allow you to log in if the password entered for *user2* is valid.

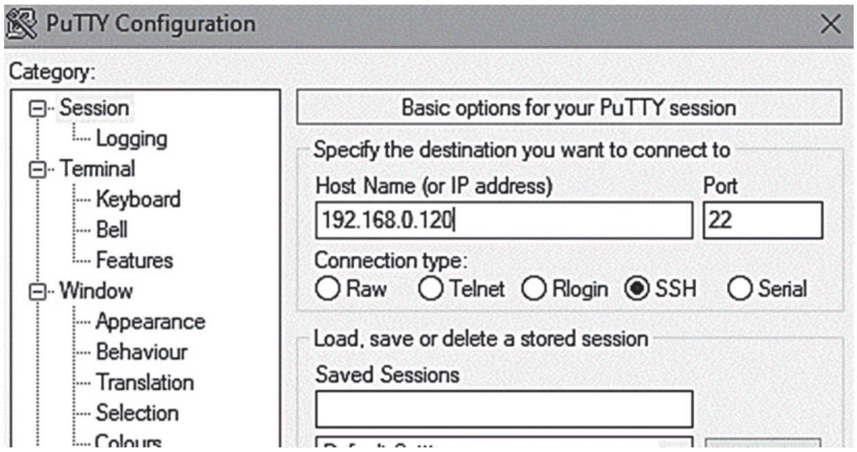
Exercise 19-2: Access RHEL System from Windows

This exercise should be done on the Windows computer hosting the virtual machine for *server20*.

For this exercise, it's assumed that *user1* exists on *server20*.

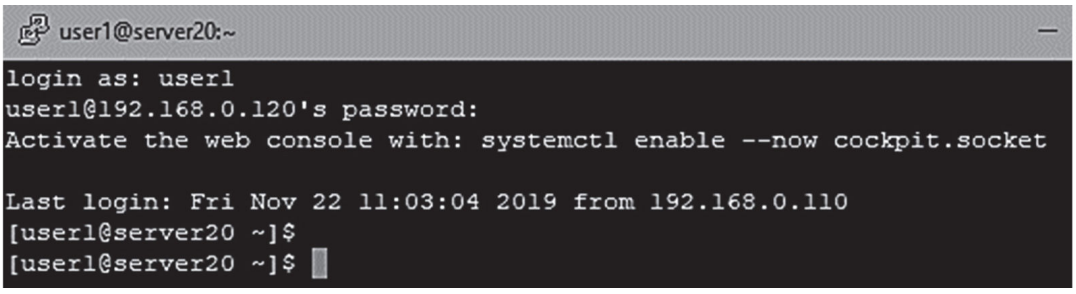
In this exercise, you will use a program called PuTTY to access *server20* using its IP address and as *user1*. You will run appropriate commands on *server20* for validation. You will log off to terminate the session.

1. On Windows, download PuTTY free of charge from the Internet. Launch this program and enter the target host's IP address. Leave the rest of the settings to their defaults.



You may assign a name to this session (typically the hostname is used as the session name) in the Saved Sessions field and click Save to store this information so as to avoid retyping in the future.

2. Click on the "Open" button at the bottom of the screen to try a connection.
3. Enter *user1* and password at the "login as" prompt to log in:



4. Issue the basic Linux commands *whoami*, *hostname*, and *pwd* to confirm that you are logged in as *user1* on *server20* and placed in the correct home directory:

```
[user1@server20 ~]$ whoami
user1
[user1@server20 ~]$ hostname
server20.example.com
[user1@server20 ~]$ pwd
/home/user1
[user1@server20 ~]$
```

5. Run the *logout* or the *exit* command or press the key combination Ctrl+d to log off *server20* and terminate the login session:

```
[user1@server20 ~]$ logout
```

This concludes the exercise.

Exercise 19-3: Generate, Distribute, and Use SSH Keys

This exercise should be done on *server1* and *server2* as *user1* and *sudo* where required.

In this exercise, you will generate a password-less ssh key pair using RSA algorithm for *user1* on *server10*. You will display the private and public file contents. You will distribute the public key to *server20* and attempt to log on to *server20* from *server10*. You will show the log file message for the login attempt.

1. Log on to *server10* as *user1*.
2. Generate RSA keys without a password (-N) and without detailed output (-q). Press Enter when prompted to provide the filename to store the private key.

```
[user1@server10 ~]$ ssh-keygen -N "" -q
Enter file in which to save the key (/home/user1/.ssh/id_rsa):
[user1@server10 ~]$
```

The content of the *id_rsa* (private key) file is shown below:

```
[user1@server10 ~]$ cat .ssh/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABFwAAAAdzc2gtcn
NhAAAAAwEAAQAAAQEApGlHp95XPXCoXfs8qQWDsYnQYiTfAb8VqP46mXSZtKoYNHsC+FQa
n+6XNh5wOxG7O+l1ZN15y/RBIRwOT5WxrgKkm9u6EREOkP2F36vFWWJT12Cm3e5LFeuBwt
.......
MmF3yBgsOjwF2oquUIX8hI/NJCrc1qut/oP31Om4CiO/GrFn9FJgS8A1/KFgYn+H/pY9u6
xEV5ZHZ86RvrcGISyyQr6q63+P+ojzR4n2gO26NHnOFpE8ZDeA+125l6rdaS9iqkieQhrk
uTr4PeOgXgFfGLQxAAAAGnVzZXIxQHNlcnZlcjEwLmV4YW1wbGUuY29t
-----END OPENSSH PRIVATE KEY-----
```

The content of the *id_rsa.pub* (public key) file is displayed below:

```
[user1@server10 ~]$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCkaUen3lc9cKhd+zypBYOxidBiJN8BvxWo/jqZdJmO
qhgOewL4VBqf7pc2HnA7EbvT6XVk2XnL9EEhHA5PlbGuAqSb27oREQ6Q/YXfq8VZY1PXYKbd7ksV64HC
OFavtOJYlOkXWj21sBFuqFvPnfqmynzuZ6O+Hb66K4fvmgTZcfjfDksU7czWhXQ36rt25Ih/7klWHO92
8p8naQ8tZvEB7hxHqeArHHJcbSOd4SdrhXNImGgYe9deyWSRT4eG6KcIcN8+vo267UGpLTngFDy3kUMI
xzd5R1Q/fmmeK3MDF5AtpNjAPoGcFuns3drQbwM+gTrncemqBWHq2XSD7KM3 user1@server10.exam
ple.com
```

3. Copy the public key file to *server20* under */home/user1/.ssh* directory. Accept the fingerprints for *server20* when prompted (only presented on the first login attempt). Enter the password for *user1* set on *server20* to continue with the file copy. The public key will be copied as *authorized_keys*.

```
[user1@server10 ~]$ ssh-copy-id server20
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/user1/.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 prompt
ed now it is to install the new keys
user1@server20's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'server20'"
and check to make sure that only the key(s) you wanted were added.

[user1@server10 ~]$ _
```

At the same time, this command also creates or updates the *known_hosts* file on *server10* and stores the fingerprints for *server20* in it. Here is what is currently stored in it:

```
[user1@server10 ~]$ cat .ssh/known_hosts
server20,192.168.0.120 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIb
mlzdHAyNTYAAABBBJwhgvOsYtOShMjVKWyqS7HJQb54i5eASbt9kBJE9DK6gOMYrII4mIR/s5NVdhu9o
zrPrV7AIAxx8jSGOsmvRjk=
```

4. On *server10*, run the *ssh* command as *user1* to connect to *server20*. You will not be prompted for a password because there was none assigned to the ssh keys.

```
[user1@server10 ~]$ ssh server20
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sun Nov 24 17:11:37 2019 from 192.168.0.13
[user1@server20 ~]$
```

You can view this login attempt in the */var/log/secure* file on *server20*:

```
[user1@server20 ~]$ sudo tail /var/log/secure
........
Nov 24 17:41:31 server20 sshd[19228]: Accepted publickey for user1 from 192.168.
0.110 port 56374 ssh2: RSA SHA256:OLtCqjQPiXk2fNecJOEbR8KqP5w59kOIOIopluZ5OIO
Nov 24 17:41:31 server20 sshd[19228]: pam_unix(sshd:session): session opened for
 user user1 by (uid=0)
```

The log entry shows the timestamp, hostname, process name and PID, username and source IP, and other relevant information. This file will log all future login attempts for this user.

Executing Commands Remotely Using ssh

The *ssh* command is a secure replacement for the legacy unsecure tools *telnet*, *rlogin*, and *rsh*. It allows you to securely sign in to a remote system or execute a command without actually logging on to it. Exercise 19-3 demonstrated how a user can log in using this command. The following shows a few basic examples on how to use *ssh* to execute a command on a remote system.

Invoke the *ssh* command on *server10* to execute the *hostname* command on *server20*:

```
[user1@server10 ~]$ ssh server20 hostname
server20.example.com
```

Run the *nmcli* command on *server20* to show (s) active network connections (c):

```
[user1@server10 ~]$ ssh server20 nmcli c s
NAME    UUID                                  TYPE      DEVICE
enp0s3  7426fe5c-9bd4-42ad-bb95-3d15491082b9  ethernet  enp0s3
enp0s8  c9f81ca1-4418-4a8d-9f9a-b6e4368bba46  ethernet  enp0s8
virbr0  58c12da2-fbba-488b-bccd-280b67ef176b  bridge    virbr0
```

You can run any command on *server20* this way without having to log in to it.

Copying Files Remotely Using scp

Similar to *ssh*, a user can execute the *scp* command to transfer files from *server10* to *server20*, and vice versa. This program can be run by a normal user as long as the user has the required read and write permissions on the source and destination, or by the *root* user. Here are a few examples to understand the program's syntax and usage.

To transfer the */etc/chrony.conf* file from *server20* to */tmp* on *server10* and confirm:

```
[user1@server10 ~]$ scp server20:/etc/chrony.conf /tmp
chrony.conf                                   100% 1103   830.3KB/s   00:00
[user1@server10 ~]$ ls -l /tmp/chrony.conf
-rw-r--r--. 1 user1 user1 1103 Nov 24 18:33 /tmp/chrony.conf
```

The program took not even a second to transfer the file. The file size is 1103 bytes. The *ls* command confirms the pull.

Now let's transfer the entire */etc/sysconfig* directory (-r) from *server10* into */tmp* on *server20* and confirm. Ignore any permission errors reported in the output.

```
[user1@server10 ~]$ scp -r /etc/sysconfig server20:/tmp
/etc/sysconfig/ip6tables-config: Permission denied
/etc/sysconfig/iptables-config: Permission denied
/etc/sysconfig/ebtables-config: Permission denied
nftables.conf                                 100%  507   377.2KB/s   00:00
ifcfg-enp0s3                                  100%  344   264.3KB/s   00:00
ifcfg-enp0s8                                  100%  166   123.2KB/s   00:00
samba                                         100%  428   360.1KB/s   00:00
selinux                                       100%  548   599.5KB/s   00:00
run-parts                                     100%    0     0.0KB/s   00:00
crond                                         100%  110    94.4KB/s   00:00
chronyd                                       100%   46    41.0KB/s   00:00
cpupower                                      100%  150   133.1KB/s   00:00
raid-check                                    100% 2915     2.1MB/s   00:00
wpa_supplicant                                100%  258   177.8KB/s   00:00
grub                                          100%  329   218.5KB/s   00:00
up2date                                       100% 1796     1.8MB/s   00:00
........
```

Run the *ls* command on *server20* for verification:

```
[user1@server10 ~]$ ssh server20 ls -l /tmp/sysconfig
total 88
-rw-r--r--. 1 user1 user1  162 Jan  8 23:33 anaconda
-rw-r--r--. 1 user1 user1  403 Jan  8 23:33 atd
-rw-r--r--. 1 user1 user1  339 Jan  8 23:33 autofs
drwxr-xr-x. 2 user1 user1   43 Jan  8 23:33 cbq
-rw-r--r--. 1 user1 user1   46 Jan  8 23:33 chronyd
drwxr-xr-x. 2 user1 user1    6 Jan  8 23:33 console
-rw-r--r--. 1 user1 user1  150 Jan  8 23:33 cpupower
-rw-r--r--. 1 user1 user1  110 Jan  8 23:33 crond
-rw-r--r--. 1 user1 user1   73 Jan  8 23:33 firewalld
-rw-r--r--. 1 user1 user1  329 Jan  8 23:33 grub
-rw-r--r--. 1 user1 user1  903 Jan  8 23:33 irqbalance
-rw-r--r--. 1 user1 user1 1722 Jan  8 23:33 kdump
-rw-r--r--. 1 user1 user1  185 Jan  8 23:33 kernel
-rw-r--r--. 1 user1 user1  310 Jan  8 23:33 man-db
drwxr-xr-x. 2 user1 user1    6 Jan  8 23:33 modules
-rw-r--r--. 1 user1 user1   22 Jan  8 23:33 network
........
```

The above output verifies the directory copy.

In the above examples, the user account that was used on the source and target servers is the same user, *user1*. To transfer a file or directory using a different user account on the target server, you need to include that user's name with the command. You must know the password for the user on the target server. Here is the syntax:

```
[user1@server10 ~]$ scp /etc/hosts user2@server20:/tmp
```

Check the manual pages of the *scp* command for more details and usage examples.

Transferring Files Remotely Using sftp

The *sftp* command is an interactive file transfer tool that can be used instead of *scp*. This tool can be launched as follows on *server10* to connect to *server20*:

```
[user1@server10 ~]$ sftp server20
Connected to server20.
sftp> ?
```

Type ? at the prompt to list available commands along with a short description:

```
sftp> ?
Available commands:
bye                                Quit sftp
cd path                            Change remote directory to 'path'
chgrp grp path                     Change group of file 'path' to 'grp'
chmod mode path                    Change permissions of file 'path' to 'mode'
chown own path                     Change owner of file 'path' to 'own'
df [-hi] [path]                    Display statistics for current directory or
                                   filesystem containing 'path'
exit                               Quit sftp
get [-afPpRr] remote [local]       Download file
reget [-fPpRr] remote [local]      Resume download file
reput [-fPpRr] [local] remote      Resume upload file
help                               Display this help text
lcd path                           Change local directory to 'path'
lls [ls-options [path]]            Display local directory listing
lmkdir path                        Create local directory
ln [-s] oldpath newpath            Link remote file (-s for symlink)
lpwd                               Print local working directory
ls [-1afhlnrSt] [path]             Display remote directory listing
lumask umask                       Set local umask to 'umask'
mkdir path                         Create remote directory
progress                           Toggle display of progress meter
put [-afPpRr] local [remote]       Upload file
pwd                                Display remote working directory
quit                               Quit sftp
rename oldpath newpath             Rename remote file
rm path                            Delete remote file
rmdir path                         Remove remote directory
symlink oldpath newpath            Symlink remote file
version                            Show SFTP version
!command                           Execute 'command' in local shell
!                                  Escape to local shell
?                                  Synonym for help
```

As shown in the above screenshot, there are many common commands available at the sftp> prompt. These include *cd* to change directory, *get*/*put* to download/upload a file, *ls* to list files, *pwd* to print working directory, *mkdir* to create a directory, *rename* to rename a file, *rm* to remove a file, and *bye*/*quit*/*exit* to exit the program and return to the command prompt. These commands will run on the remote server (*server20*). The following screenshot shows how these commands are used:

```
sftp> mkdir /tmp/dir10-20
sftp>
sftp> cd /tmp/dir10-20
sftp> pwd
Remote working directory: /tmp/dir10-20
sftp> put /etc/group
Uploading /etc/group to /tmp/dir10-20/group
/etc/group                                    100% 1105     1.4MB/s   00:00
sftp>
sftp> ls -l
-rw-r--r--    1 user1    user1        1105 Nov 24 19:03 group
sftp>
sftp> cd ..
sftp> pwd
Remote working directory: /tmp
sftp> cd /home/user1
sftp> get /usr/bin/gzip
Fetching /usr/bin/gzip to gzip
/usr/bin/gzip                                 100%  106KB  17.9MB/s   00:00
```

Furthermore, there are four commands beginning with an ‘l’—*lcd*, *lls*, *lpwd*, and *lmkdir*—at the sftp> prompt. These commands are intended to be run on the source server (*server10*). Other Linux commands are also available at the sftp> prompt that you may use for basic file management operations on the remote server.

Type *quit* at the sftp> prompt to exit the program when you’re done.

You may use either *sftp* or *scp* for transferring files depending on your comfort level. Consult the manual pages of the commands for options and additional details.

Synchronizing Files Remotely Using rsync

The *rsync* (*remote synchronization*) program works in a manner similar to the *cp*, *rcp*, and *scp* commands to copy files between the source and destination. With *rsync*, the source and destination could be on the same system or different systems. The first initiation of *rsync* copies all files from the source to the destination with subsequent executions copy only the updated files. The *rsync* command uses the ssh protocol by default.

The following examples explain the usage of the program and introduce some common flags.

To copy a single file such as *grub.conf* to */tmp* on the same system:

```
[user1@server10 ~]$ sudo rsync -av /boot/grub2/grub.cfg /tmp
sending incremental file list
grub.cfg

sent 5,126 bytes  received 35 bytes  10,322.00 bytes/sec
total size is 5,032  speedup is 0.98
```

The -a option in the above example instructs the command to perform an archive operation and preserve all file attributes such as permissions, ownership, symlinks, and timestamps. The -v switch is used for verbosity.

The actual size of the *grub.cfg* file is 5,032 bytes. The additional bytes sent (5,126 minus 5,032 = 94 bytes) contain the metadata and other overhead, and the received bytes signify the metadata received. The output displays the files being copied and the file transfer rate as well.

Subsequent invocations of the above would produce an output similar to the following if the file has not been modified:

```
[user1@server10 ~]$ sudo rsync -avz /boot/grub2/grub.cfg /tmp
sending incremental file list

sent 47 bytes  received 12 bytes  118.00 bytes/sec
total size is 5,032  speedup is 85.29
```

It shows no filenames under the file list, as there was no transfer occurred.

To copy */etc/rsyslog.conf* to */tmp* to *server20* using in-transit compression (-z) and displaying the transfer progress (-P):

```
[user1@server10 ~]$ rsync -avPz /etc/rsyslog.conf server20:/tmp
sending incremental file list
rsyslog.conf
          3,185 100%    0.00kB/s    0:00:00 (xfr#1, to-chk=0/1)

sent 1,478 bytes  received 35 bytes  3,026.00 bytes/sec
total size is 3,185  speedup is 2.11
```

To copy the entire */home/user1* directory recursively (-r) from *server20* to */tmp/trans* directory on *server10* (create */tmp/trans* before running the *rsync* command):

```
[user1@server10 ~]$ mkdir /tmp/trans
[user1@server10 ~]$ rsync -aPvzr server20:/home/user1 /tmp/trans
receiving incremental file list
user1/
user1/.ICEauthority
          1,244 100%    1.19MB/s    0:00:00 (xfr#1, to-chk=150/152)
user1/.bash_history
         19,437 100%   18.54MB/s    0:00:00 (xfr#2, to-chk=149/152)
user1/.bash_logout
             18 100%    2.20kB/s    0:00:00 (xfr#3, to-chk=148/152)
user1/.bash_profile
            141 100%   10.59kB/s    0:00:00 (xfr#4, to-chk=147/152)
........
sent 1,394 bytes  received 897,769 bytes  599,442.00 bytes/sec
total size is 8,639,186  speedup is 9.61
```

The *rsync* command is fast and versatile, and has numerous other options available. Refer to the command's manual pages for a description of options and usage examples.

Chapter Summary

This chapter discussed the open source version of the secure shell service. It started with an overview of the service, and described what it is, how it works, available versions, and algorithms employed. We skimmed through various encryption techniques and authentication methods. We touched upon the service daemon, client and server configuration files, and commands. We demonstrated accessing a lab server from another lab server and from Windows. We generated and distributed password-less private/public key pair and employed ssh utilities to remote execute commands and transfer files.

Lastly, we examined a program that may be put into action to keep files synchronized between two systems over an ssh channel.

Check Your Understanding

1. What is the secure equivalent for the *rcp* command?
2. What would the command *ssh-keygen -N “”* do?
3. The primary secure shell server configuration file is *ssh_config*. True or False?
4. Which three common algorithms are used with SSH version 2 for encryption and/or authentication?
5. What is the secure shell equivalent for the *telnet* command?
6. True or False? By default, the *root* user can directly log on to a RHEL system.
7. What is the default location to store user SSH keys?
8. What would the command *ssh-copy-id* do?
9. What is the *rsync* command used for?
10. Which two of the five authentication methods mentioned in this chapter are more prevalent?
11. What is the use of the *ssh-keygen* command?
12. Name the default algorithm used with SSH.
13. What kind of information does the *~/.ssh/known_hosts* file store?

14. List the two encryption techniques described in this chapter.
15. What is the default port used by the secure shell service?
16. Which log file stores authentication messages?
17. The *ssh* tool provides a non-secure tunnel over a network for accessing a RHEL system. True or False?
18. Name the SSH client-side configuration file.
19. What would the command *ssh server10 ls* do?

Answers to Check Your Understanding

1. The *scp* command is the secure equivalent for the *rcp* command.
2. The command provided will generate a password-less ssh key pair using the default RSA algorithm.
3. False. The primary secure shell configuration file is *sshd_config*.
4. The SSH version 2 uses RSA, DSA, and ECDSA algorithms.
5. The secure equivalent for *telnet* is the *ssh* command.
6. True.
7. Under the *~/.ssh* directory.
8. The *ssh-copy-id* command is used to distribute the public key to remote systems.
9. The *rsync* command is used to maintain a copy of source files at remote location.
10. The public key-based and password-based authentication methods are more prevalent.
11. The *ssh-keygen* command is used to generate public/private key combination for use with ssh.
12. The default algorithm used with ssh is RSA.
13. The *~/.ssh/known_hosts* file stores fingerprints of remote servers.

14. The two encryption techniques are symmetric (secret key) and asymmetric (public key).
15. The default port used by the secure shell service is 22.
16. The */var/log/secure* file stores authentication messages.
17. False. The *ssh* command provides a secure tunnel over a network.
18. The client-side SSH configuration file is *ssh_config* and it is located in the */etc/ssh* directory.
19. The command provided will run the *ls* command on the specified remote ssh server without the need for the user to log in.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 19-1: Establish Key-Based Authentication

As *user1* with *sudo* on *server10* and *server20*, create user account *user20* and assign a password. As *user20* on *server10*, generate a private/public key pair without a passphrase using the *ssh-keygen* command. Distribute the public key to *server20* with the *ssh-copy-id* command. Log on to *server20* as *user20* and accept the fingerprints for the server if presented. On subsequent log in attempts from *server10* to *server20*, *user20* should not be prompted for their password. (Hint: System Access and File Transfer).

Lab 19-2: Test the Effect of PermitRootLogin Directive

As *user1* with *sudo* on *server20*, edit the */etc/ssh/sshd_config* file and change the value of the directive PermitRootLogin to “no”. Use the *systemctl* command to activate the change. As *root* on *server10*, run **ssh server20** (or its IP address). You’ll get permission denied message. Reverse the change on *server20* and retry **ssh server20**. You should be able to log in. (Hint: The OpenSSH Service).

Chapter 20

The Linux Firewall

This chapter describes the following major topics:

- Describe Linux firewall for host-based security control
- Overview of the firewalld service
- Understand the concepts of firewalld zones and services
- Analyze zone and service configuration files
- Control access to network services and ports through firewalld
- Use firewall-cmd command to manage firewall rules

RHCSA Objectives:

45. Restrict network access using firewall-cmd/firewall

50. Configure firewall settings using firewall-cmd/firewalld

Running a system in a networked or an Internet-facing environment requires that some security measures be taken to tighten access to the system in general and individual services in particular. This can be accomplished by implementing a firewall and restricting inbound traffic to allowed ports from valid source IP addresses only. We discuss a host-based firewall solution in this chapter.

Firewall Overview

A *firewall* is a protective layer implemented at the network or server level to secure, monitor, and control inbound and outbound traffic flow. Firewalls employed at the network level use either dedicated hardware or sophisticated software appliances to form a shield around the network. Server level firewalls are referred to as *host-based firewalls* and they run in a computer operating system to monitor and manage traffic in and out. Firewalls defend a network or an individual server from undesired traffic.

RHEL is shipped with a host-based firewall solution that works by filtering data packets. A data packet is formed as a result of a process called *encapsulation* whereby the header information is attached to a message (called *payload*) during packet formation. The header includes information such as source and destination IP addresses, port, and type of data. Based on predefined *rules*, a firewall intercepts each inbound and outbound data packet, inspects its header, and decides whether to allow the packet to pass through.

Ports are defined in the */etc/services* file for common network services that are standardized across all network

operating systems, including RHEL. Some common services and the ports they listen on are FTP (*File Transfer Protocol*) on port 21, SSH (*Secure Shell*) 22, Postfix (an email service) 25, HTTP (*HyperText Transfer Protocol*) 80, and NTP (*Network Time Protocol*) on port 123.

The host-based firewall solution employed in RHEL uses a kernel module called *netfilter* together with a filtering and packet classification framework called *nftables* for policing the traffic movement. It also supports other advanced features such as *Network Address Translation* (NAT) and *port forwarding*. This firewall solution inspects, modifies, drops, or routes incoming, outgoing, and forwarded network packets based on defined rulesets.

Overview of firewalld

firewalld is the default host-based firewall management service in RHEL 8. One of the major advantages is its ability to add, modify, or delete firewall rules immediately without disrupting current network connections or restarting the service process. This is especially useful in testing and troubleshooting scenarios. *firewalld* also allows to save rules persistently so that they are activated automatically at system reboots.

The *firewalld* service lets you perform management operations at the command line using the *firewall-cmd* command, graphically using the web console, or manually by editing rules files. *firewalld* stores the default rules in files located in the */usr/lib/firewalld* directory, and those that contain custom rules in the */etc/firewalld* directory. The default rules files may be copied to the custom rules directory and modified.

firewalld Zones

firewalld uses the concept of *zones* for easier and transparent traffic management. Zones define policies based on the trust level of network connections and source IP addresses. A network connection can be part of only one zone at a time; however, a zone can have multiple network connections assigned to it. Zone configuration may include services, ports, and protocols that may be open or closed. It may also include rules for advanced configuration items such as masquerading, port forwarding, NATting, ICMP filters, and rich language. Rules for each zone are defined and manipulated independent of other zones.

firewalld inspects each incoming packet to determine the source IP address and applies the rules of the zone that has a match for the address. In the event no zone configuration matches the address, it associates the packet with the zone that has the network connection defined, and applies the rules of that zone. If neither works, *firewalld* associates the packet with the default zone, and enforces the rules of the default zone on the packet.

The *firewalld* software installs several predefined zone files that may be selected or customized. These files include templates for traffic that must be blocked or dropped, and for traffic that is public-facing, internal, external, home, public, trusted, and work-related. Of these, the *public* zone is the default zone, and it is activated by default when the *firewalld* service is started. Table 20-1 lists and describes the predefined zones sorted based on the trust level from trusted to untrusted.

Zone	Description
trusted	Allow all incoming
internal	Reject all incoming traffic except for what is allowed. Intended for use on internal networks.
home	Reject all incoming traffic except for what is allowed. Intended for use in homes.
work	Reject all incoming traffic except for what is allowed. Intended for use at workplaces.
dmz	Reject all incoming traffic except for what is allowed. Intended for use in publicly-accessible demilitarized zones.
external	Reject all incoming traffic except for what is allowed. Outgoing IPv4 traffic forwarded through this zone is masqueraded to look like it originated from the IPv4 address of an outgoing network interface. Intended for use on external networks with masquerading enabled.
public	Reject all incoming traffic except for what is allowed. It is the default zone for any newly added network interfaces. Intended for us in public places.
block	Reject all incoming traffic with icmp-host-prohibited message returned. Intended for use in secure places.
drop	Drop all incoming traffic without responding with ICMP errors. Intended for use in highly secure places.

Table 20-1 firewalld Default Zones

For all the predefined zones, outgoing traffic is allowed by default.

Zone Configuration Files

firewalld stores zone rules in XML format at two locations: the system-defined rules in the */usr/lib/firewalld/zones* directory, and the user-defined rules in the */etc/firewalld/zones* directory. The files at the former location can be used as templates for adding new rules, or applied instantly to any available network connection. A system zone

configuration file is automatically copied to the */etc/firewalld/zones* directory if it is modified with a management tool. Alternatively, you can copy the required zone file to the */etc/firewalld/zones* directory manually, and make the necessary changes. The *firewalld* service reads the files saved in this location, and applies the rules defined in them. A listing of the system zone files is presented below:

```
[user1@server10 ~]$ ls -l /usr/lib/firewalld/zones/
total 36
-rw-r--r--. 1 root root 299 Jan 14  2019 block.xml
-rw-r--r--. 1 root root 293 Jan 14  2019 dmz.xml
-rw-r--r--. 1 root root 291 Jan 14  2019 drop.xml
-rw-r--r--. 1 root root 304 Jan 14  2019 external.xml
-rw-r--r--. 1 root root 397 Jan 14  2019 home.xml
-rw-r--r--. 1 root root 412 Jan 14  2019 internal.xml
-rw-r--r--. 1 root root 343 Jan 14  2019 public.xml
-rw-r--r--. 1 root root 162 Jan 14  2019 trusted.xml
-rw-r--r--. 1 root root 339 Jan 14  2019 work.xml
```

The default *public* zone file is displayed below:

```
[user1@server10 ~]$ cat /usr/lib/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on
networks to not harm your computer. Only selected incoming connections are accep
ted.</description>
  <service name="ssh"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
</zone>
```

As depicted in the screenshot, the zone has a name and description, and it contains a list of all the allowed services—*ssh*, *dhcpv6-client*, and *cockpit*. See the manual pages for *firewalld.zone* for details on zone configuration files.

firewalld Services

In addition to the concept of zones, *firewalld* also uses the idea of *services* for easier activation and deactivation of specific rules. *firewalld* services are preconfigured firewall rules delineated for various services and stored in different files. The rules consist of necessary settings, such as the port

number, protocol, and possibly helper modules, to support the loading of the service. *firewalld* services can be added to a zone. By default, *firewalld* blocks all traffic unless a service or port is explicitly opened.

Service Configuration Files

firewalld stores service rules in XML format at two locations: the system-defined rules in the */usr/lib/firewalld/services* directory, and the user-defined rules in the */etc/firewalld/services* directory. The files at the former location can be used as templates for adding new service rules, or activated instantly. A system service configuration file is automatically copied to the */etc/firewalld/services* directory if it is modified with a management tool. Alternatively, you can copy the required service file to the */etc/firewalld/services* directory manually, and make the necessary changes. The *firewalld* service reads the files saved in this location, and applies the rules defined in them. A listing of the system service files is presented below:

```
[user1@server10 ~]$ ls -l /usr/lib/firewalld/services/
total 616
-rw-r--r--. 1 root root  412 Jan 14  2019 amanda-client.xml
-rw-r--r--. 1 root root  447 Jan 14  2019 amanda-k5-client.xml
-rw-r--r--. 1 root root  283 Jan 14  2019 amqps.xml
-rw-r--r--. 1 root root  273 Jan 14  2019 amqp.xml
-rw-r--r--. 1 root root  285 Jan 14  2019 apcupsd.xml
-rw-r--r--. 1 root root  301 Jan 14  2019 audit.xml
-rw-r--r--. 1 root root  320 Jan 14  2019 bacula-client.xml
-rw-r--r--. 1 root root  346 Jan 14  2019 bacula.xml
-rw-r--r--. 1 root root  339 Jan 14  2019 bgp.xml
-rw-r--r--. 1 root root  275 Jan 14  2019 bitcoin-rpc.xml
-rw-r--r--. 1 root root  307 Jan 14  2019 bitcoin-testnet-rpc.xml
-rw-r--r--. 1 root root  281 Jan 14  2019 bitcoin-testnet.xml
-rw-r--r--. 1 root root  244 Jan 14  2019 bitcoin.xml
-rw-r--r--. 1 root root  294 Jan 14  2019 ceph-mon.xml
-rw-r--r--. 1 root root  329 Jan 14  2019 ceph.xml
-rw-r--r--. 1 root root  168 Jan 14  2019 cfengine.xml
-rw-r--r--. 1 root root  211 Jan 14  2019 cockpit.xml
........
```

The following shows the content of the *ssh* service file:

```
[user1@server10 ~]$ cat /usr/lib/firewalld/services/ssh.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>SSH</short>
  <description>Secure Shell (SSH) is a protocol for logging into and executing c
ommands on remote machines. It provides secure encrypted communications. If you
plan on accessing your machine remotely via SSH over a firewalled interface, ena
ble this option. You need the openssh-server package installed for this option t
o be useful.</description>
  <port protocol="tcp" port="22"/>
</service>
```

As depicted in the screenshot, the service has a name and description, and it defines the port and protocol for the service. See the manual pages for *firewalld.service* for details on service configuration files.

Firewall Management

Managing the *firewalld* service involves a number of operations, such as listing, querying, adding, changing, and removing zones, services, ports, IP sources, and network connections. There are three methods available in RHEL 8 to perform the management tasks. They include the *firewall-cmd* command for those who prefer to work at the command line and the web interface for graphical administration. The third management option is to make use of the zone and service templates, and edit them as desired. We use the command line method in this book.

The firewall-cmd Command

The *firewall-cmd* command is a powerful tool to manage the *firewalld* service at the command prompt. This tool can be used to add or remove rules from the runtime configuration, or save any modifications to service configuration for persistence. It supports numerous options for the management of zones, services, ports, connections, and so on; Table 20-2 lists and describes the common options only.

Option	Description
General	
--state	Displays the running status of firewalld
--reload	Reloads firewall rules from zone files. All runtime changes are lost.
--permanent	Stores a change persistently. The change only becomes active after a service reload or restart.
Zones	
--get-default-zone	Shows the name of the default/active zone
--set-default-zone	Changes the default zone for both runtime and permanent configuration
--get-zones	Prints a list of available zones
--get-active-zones	Displays the active zone and the assigned interfaces
--list-all	Lists all settings for a zone
--list-all-zones	Lists the settings for all available zones
--zone	Specifies the name of the zone to work on. Without this option, the default zone is used.
Services	
--get-services	Prints predefined services
--list-services	Lists services for a zone
--add-service	Adds a service to a zone
--remove-service	Removes a service from a zone
--query-service	Queries for the presence of a service
Ports	
--list-ports	Lists network ports
--add-port	Adds a port or a range of ports to a zone
--remove-port	Removes a port from a zone
--query-port	Queries for the presence of a port
Network Connections	
--list-interfaces	Lists network connections assigned to a zone
--add-interface	Binds a network connection to a zone
--change-	Changes the binding of a network connection to

interface	a different zone
--remove-interface	Unbinds a network connection from a zone
IP Sources	
--list-sources	Lists IP sources assigned to a zone
--add-source	Adds an IP source to a zone
--change-source	Changes an IP source
--remove-source	Removes an IP source from a zone

Table 20-2 Common firewall-cmd Options

With all the --add and --remove options, the --permanent switch may be specified to ensure the rule is stored in the zone configuration file under the */etc/firewalld/zones* directory for persistence. Some of the options from Table 20-2 are used in the upcoming exercises; the rest are beyond the scope of this book. Consult the manual pages of the command for details on the usage of these and other options.

Querying the Operational Status of firewalld

You can check the running status of the *firewalld* service using either the *systemctl* or the *firewall-cmd* command. Both commands will produce different outputs, but the intent here is to ensure the service is in the running state.

```
[user1@server10 ~]$ sudo firewall-cmd --state
running
[user1@server10 ~]$ sudo systemctl status firewalld -l --no-pager
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor pr
eset: enabled)
   Active: active (running) since Mon 2019-11-25 22:47:22 EST; 1min 0s ago
     Docs: man:firewalld(1)
 Main PID: 15403 (firewalld)
    Tasks: 2 (limit: 5076)
   Memory: 25.1M
   CGroup: /system.slice/firewalld.service
           └─15403 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork
--nopid

Nov 25 22:47:21 server10.example.com systemd[1]: Starting firewalld - dynamic fi
rewall daemon...
Nov 25 22:47:22 server10.example.com systemd[1]: Started firewalld - dynamic fir
ewall daemon.
```

The output indicates that the *firewalld* service is in the running state on *server10*. The other command outcome also reports that the service is marked for autostart at system reboots. In case *firewalld* is not enabled or is inactive, issue **sudo systemctl --now enable firewalld** to start it immediately, and mark it for autostart on reboots.

You are ready to perform the exercises presented next.

Exercise 20-1: Add Services and Ports, and Manage Zones

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will determine the current active zone. You will add and activate a permanent rule to allow HTTP traffic on port 80, and then add a runtime rule for traffic intended for TCP port 443 (the HTTPS service). You will add a permanent rule to the *internal* zone for TCP port range 5901 to 5910. You will confirm the changes and display the contents of the affected zone files. Lastly, you will switch the default zone to the *internal* zone and activate it.

1. Determine the name of the current default zone:

```
[user1@server10 ~]$ sudo firewall-cmd --get-default-zone
public
```

2. Add a permanent rule to allow HTTP traffic on its default port:

```
[user1@server10 ~]$ sudo firewall-cmd --permanent --add-service http
success
```

The command made a copy of the *public.xml* file from */usr/lib/firewalld/zones* directory into the */etc/firewalld/zones* directory, and added the rule for the HTTP service.

3. Activate the new rule:

```
[user1@server10 ~]$ sudo firewall-cmd --reload
success
```

4. Confirm the activation of the new rule:

```
[user1@server10 ~]$ sudo firewall-cmd --list-services
cockpit dhcpv6-client http ssh
```

5. Display the content of the default zone file to confirm the addition of the permanent rule:

```
[user1@server10 ~]$ sudo cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on networks to
not harm your computer. Only selected incoming connections are accepted.</description>
  <service name="ssh"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
  <service name="http"/>
</zone>
```

6. Add a runtime rule to allow traffic on TCP port 443 and verify:

```
[user1@server10 ~]$ sudo firewall-cmd --add-port 443/tcp
success
[user1@server10 ~]$ sudo firewall-cmd --list-ports
443/tcp
```

7. Add a permanent rule to the *internal* zone for TCP port range 5901 to 5910:

```
[user1@server10 ~]$ sudo firewall-cmd --add-port 5901-5910/tcp --permanent --zone internal
success
```

8. Display the content of the *internal* zone file to confirm the addition of the permanent rule:

```
[user1@server10 ~]$ sudo cat /etc/firewalld/zones/internal.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Internal</short>
  <description>For use on internal networks. You mostly trust the other computers on the net
works to not harm your computer. Only selected incoming connections are accepted.</descripti
on>
  <service name="ssh"/>
  <service name="mdns"/>
  <service name="samba-client"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
  <port port="5901-5910" protocol="tcp"/>
</zone>
```

The *firewall-cmd* command makes a backup of the affected zone file with a *.old* extension whenever an update is made to a zone.

9. Switch the default zone to internal and confirm:

```
[user1@server10 ~]$ sudo firewall-cmd --set-default-zone internal
success
[user1@server10 ~]$ sudo firewall-cmd --get-default-zone
internal
```

10. Activate the rules defined in the *internal* zone and list the port range added earlier:

```
[user1@server10 ~]$ sudo firewall-cmd --reload
success
[user1@server10 ~]$ sudo firewall-cmd --list-ports
5901-5910/tcp
```

This completes the exercise.

Exercise 20-2: Remove Services and Ports, and Manage Zones

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will remove the two permanent rules that were added in Exercise 20-1. You will switch the *public* zone back as the default zone, and confirm the changes.

1. Remove the permanent rule for HTTP from the *public* zone:

```
[user1@server10 ~]$ sudo firewall-cmd --remove-service=http --zone public --permanent
success
```

Notice the equal sign (=) used with the --remove-service option. The *firewall-cmd* command supports the specification of add, remove, change, and zone options with and without an equal sign (=). The --zone option is used to specify the *public* zone as it is currently the non-default.

2. Remove the permanent rule for ports 5901 to 5910 from the *internal* zone:

```
[user1@server10 ~]$ sudo firewall-cmd --remove-port 5901-5910/tcp --permanent
success
```

The --zone option is not used, as ‘internal’ is currently the default zone.

3. Switch the default zone to *public* and validate:

```
[user1@server10 ~]$ sudo firewall-cmd --set-default-zone=public
success
[user1@server10 ~]$ sudo firewall-cmd --get-default-zone
public
```

4. Activate the *public* zone rules, and list the current services:

```
[user1@server10 ~]$ sudo firewall-cmd --reload
success
[user1@server10 ~]$
[user1@server10 ~]$ sudo firewall-cmd --list-services
cockpit dhcpv6-client ssh
```

The *public* zone reflects the removal of the *http* service. This concludes the exercise.

Exercise 20-3: Test the Effect of Firewall Rule

This exercise should be done on *server10* and *server20* as *user1* with *sudo* where required.

In this exercise, you will remove the *sshd* service rule from the runtime configuration on *server10*, and try to access the server from *server20* using the *ssh* command.

1. Remove the rule for the *sshd* service on *server10*:

```
[user1@server10 ~]$ sudo firewall-cmd --remove-service ssh
success
```

2. Issue the *ssh* command on *server20* to access *server10*:

```
[user1@server20 ~]$ ssh server10
ssh: connect to host server10 port 22: No route to host
```

The error displayed is because the firewall on *server10* blocked the access. Put the rule back on *server10* and try to access it from *server20* again:

3. Add the rule back for *sshd* on *server10*:

```
[user1@server10 ~]$ sudo firewall-cmd --add-service=ssh
success
```

4. Issue the *ssh* command on *server20* to access *server10*. Enter “yes” if prompted and the password for *user1*.

```
[user1@server20 ~]$ ssh server10
The authenticity of host 'server10 (192.168.0.110)' can't be established.
ECDSA key fingerprint is SHA256:UavWZeF2HXWcfDrPWsDnb8uHTO4/uQ2pWJBI6LVGDOQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'server10,192.168.0.110' (ECDSA) to the list of kno
 hosts.
user1@server10's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sat Nov 30 20:21:41 2019 from 192.168.0.13
[user1@server10 ~]$
```

This brings the exercise to an end.

Chapter Summary

We discussed a native host-based firewall solution for system protection in this chapter. We explored the concepts around firewall and described how it works. We looked at the firewalld service and examined the concepts of zones and services. We reviewed predefined zones and services, and analyzed their configuration files. We studied the lone firewall management command and reviewed options for listing and administering zones, services, ports, network connections, and source IP addresses.

We learned how to change and check the operational state of the firewalld service. We performed exercises to add and remove services and ports persistently and non-persistently, and manage zones. Finally, we tested the effect of deleting a port from the firewall configuration and adding it back.

Check Your Understanding

1. A firewall can be configured between two networks but not between two hosts. True or False?
2. Which directory stores the configuration file for modified firewalld zones?
3. After changing the default firewalld zone to internal, what would you run to verify?
4. What is the process of data packet formation called?
5. What would the command *firewall-cmd --permanent --add-service=nfs --zone=external* do?
6. If you have a set of firewall rules defined for a service stored under both */etc/firewalld* and */usr/lib/firewalld* directories, which of the two sets will take precedence?
7. What is the kernel module in RHEL 8 that implements the host-level protection called?

8. firewalld is the firewall management solution in RHEL 8. True or False?
9. Name the default firewalld zone.
10. What is the purpose of firewalld service configuration files?
11. What would the command *firewall-cmd --remove-port=5000/tcp* do?
12. What is the primary command line tool for managing firewalld called?

Answers to Check Your Understanding

1. False. A firewall can also be configured between two host computers.
2. The modified *firewalld* zone files are stored under */etc/firewalld/zones* directory.
3. You run *firewall-cmd --get-default-zone* for validation.
4. The process of data packet formation is called encapsulation.
5. The command provided will add the *nfs* service to *external* firewalld zone persistently.
6. The ruleset located in the */etc/firewalld* directory will have precedence.
7. The kernel module that implements the host-level protection is called netfilter.
8. True.
9. The default firewalld zone is the *public* zone.
10. firewalld service configuration files store service-specific port, protocol, and other details, which makes it easy to activate and deactivate them.
11. The command provided will remove the runtime firewall rule for TCP port 5000.
12. The primary command line tool for managing *firewalld* is called *firewall-cmd*.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 20-1: Add Service to Firewall

As *user1* with *sudo* on *server10*, add and activate a permanent rule for HTTPs traffic to the default zone. Confirm the change by viewing the zone's XML file and running the *firewall-cmd* command. (Hint: Firewall Management).

Lab 20-2: Add Port Range to Firewall

As *user1* with *sudo* on *server10*, add and activate a permanent rule for the UDP port range 8000 to 8005 to the *trusted* zone. Confirm the change by viewing the zone's XML file and running the *firewall-cmd* command. (Hint: Firewall Management).

Chapter 21

Security Enhanced Linux

This chapter describes the following major topics:

- Describe Security Enhanced Linux and its terminology
- Understand SELinux contexts for users, processes, files, and ports
- Copy, move, and archive files with and without SELinux context
- How domain transitioning works
- Overview of SELinux Booleans
- Query and manage SELinux via management tools
- Modify SELinux contexts for files and ports
- Add SELinux rules to policy database
- View and analyze SELinux alerts

RHCSA Objectives:

53. Set enforcing and permissive modes for SELinux
54. List and identify SELinux file and process context
55. Restore default file contexts
56. Use Boolean settings to modify system SELinux settings
57. Diagnose and address routine SELinux policy violations

Security Enhanced Linux is a mechanism that controls who can access and do what on the system. It is part and parcel of the Linux kernel. It handles access beyond what the traditional access control system delivers including file and directory permissions, user and group-level permissions, shadow password and password aging mechanisms, and ACLs. The goal of SELinux is to limit the possible damage that could occur to the system due to unauthorized user or program access. This chapter covers SELinux in reasonable detail.

Security Enhanced Linux

Security Enhanced Linux (SELinux) is an implementation of the *Mandatory Access Control* (MAC) architecture developed by the U.S. *National Security Agency* (NSA) in collaboration with other organizations and the Linux community for flexible, enriched, and granular security controls in Linux. MAC is integrated into the Linux kernel as a set of patches using the *Linux Security Modules* (LSM) framework that allows the kernel to support various security implementations, including SELinux.

MAC provides an added layer of protection above and beyond the standard Linux *Discretionary Access Control* (DAC) security architecture. DAC includes the traditional file and directory permissions, ACLs, extended attribute settings, setuid/setgid bits, su/sudo privileges, and other controls. MAC limits the ability of a *subject* (Linux user or process) to access an *object* (file, directory, file system, device, network interface/connection, port, pipe, socket, etc.)to reduce or eliminate the potential damage the subject may be able to

cause to the system if compromised due to the exploitation of vulnerabilities in services, programs, or applications.

MAC controls are fine-grained; they protect other services in the event one service is negotiated. For instance, if the HTTP service process is compromised, the attacker can only damage the files the hacked process will have access to, and not the other processes running on the system, or the objects the other processes will have access to. To ensure this coarse-grained control, MAC uses a set of defined authorization rules called *policy* to examine security attributes associated with subjects and objects when a subject tries to access an object, and decides whether to permit the access attempt. These attributes are stored in *contexts* (a.k.a. *labels*), and are applied to both subjects and objects.

SELinux decisions are stored in a special cache area called *Access Vector Cache* (AVC). This cache area is checked for each access attempt by a process to determine whether the access attempt was previously allowed. With this mechanism in place, SELinux does not have to check the policy ruleset repeatedly, thus improving performance.

By default, SELinux controls are enabled at the time of RHEL installation with the default configuration, which confines the processes to the bare minimum privileges that they need to function.

Terminology

In order to comprehend SELinux, an understanding of some key terms is essential. These terms are useful in explaining the concepts and SELinux functionality in the remainder of this chapter.

Subject

A *subject* is any user or process that accesses an object. Examples include system_u for the SELinux system user, and unconfined_u for subjects that are not bound by the SELinux policy. The subject is stored in field 1 of the context.

Object

An *object* is a resource, such as a file, directory, hardware device, network interface/connection, port, pipe, or socket, that a subject accesses. Examples include object_r for general objects, system_r for system-owned objects, and unconfined_r for objects that are not bound by the SELinux policy.

Access

An *access* is an action performed by the subject on an object. Examples include creating, reading, or updating a file, creating or navigating a directory, and accessing a network port or socket.

Policy

A *policy* is a defined ruleset that is enforced system-wide, and is used to analyze security attributes assigned to subjects and objects. This ruleset is referenced to decide whether to permit a subject's access attempt to an object, or a subject's attempt to interact with another subject. The default behavior of SELinux in the absence of a rule is to deny the access. Two standard preconfigured policies are *targeted* and *mls* with targeted being the default.

The targeted policy dictates that any process that is targeted runs in a confined domain, and any process that is not targeted runs in an unconfined domain. For instance, SELinux runs logged-in users in the unconfined domain, and the *httpd* process in a confined domain by default. Any subject running unconfined is more vulnerable than the one running confined.

The mls policy places tight security controls at deeper levels.

A third preconfigured policy called *minimum* is a light version of the targeted policy, and it is designed to protect only selected processes.

Context

A *context* (a.k.a. *label*) is a tag to store security attributes for subjects and objects. In SELinux, every subject and object has a context assigned, which consists of a SELinux user, role, type (or domain), and sensitivity level. SELinux uses this information to make access control decisions.

Labeling

Labeling is the mapping of files with their stored contexts.

SELinux User

SELinux policy has several predefined SELinux user identities that are authorized for a particular set of roles. SELinux policy maintains Linux user to SELinux user identity mapping to place SELinux user restrictions on Linux users. This controls what roles and levels a process (with a particular SELinux user identity) can enter. A Linux user, for instance, cannot run the *su* and *sudo* commands or the programs located in their home directories if they are mapped to the SELinux user *user_u*.

Role

A *role* is an attribute of the *Role-Based Access Control* (RBAC) security model that is part of SELinux. It classifies who (subject) is allowed to access what (domains or types). SELinux users are authorized for roles, and roles are authorized for domains and types. Each subject has an associated role to ensure that the system and user processes are separated. A subject can transition into a new role to gain access to other domains and types. Examples roles include user_r for ordinary users, sysadm_r for administrators, and

system_r for processes that initiate under the system_r role. The role is stored in field 2 of the context.

Type Enforcement

Type enforcement (TE) identifies and limits a subject's ability to access domains for processes, and types for files. It references the contexts of the subjects and objects for this enforcement.

Type and Domain

A *type* is an attribute of type enforcement. It is a group of objects based on uniformity in their security requirements. Objects such as files and directories with common security requirements, are grouped within a specific type. Examples of types include user_home_dir_t for objects located in user home directories, and usr_t for most objects stored in the */usr* directory. The type is stored in field 3 of a file context.

A *domain* determines the type of access that a process has. Processes with common security requirements are grouped within a specific domain type, and they run confined within that domain. Examples of domains include init_t for the *systemd* process, firewalld_t for the *firewalld* process, and unconfined_t for all processes that are not bound by SELinux policy. The domain is stored in field 3 of a process context.

SELinux policy rules outline how types can access each other, domains can access types, and domains can access each other.

Level

A *level* is an attribute of *Multi-Level Security* (MLS) and *Multi-Category Security* (MCS). It is a pair of sensitivity:category values that defines the level of security in the context. A category may be defined as a single value or a range of values, such as c0.c4 to represent c0 through c4. In RHEL 8,

the targeted policy is used as the default, which enforces MCS (MCS supports only one sensitivity level (s0) with 0 to 1023 different categories).

SELinux Contexts for Users

SELinux contexts define security attributes placed on subjects and objects. Each context contains a type and a security level with subject and object information. Use the *id* command with the -Z option to view the context set on Linux users. The following example shows the context for *user1*:

```
[user1@server10 ~]$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

The output indicates that *user1* is mapped to the SELinux unconfined_u user, and that there are no SELinux restrictions placed on this user. You'll get the same result if you run this command for other users. This entails that all Linux users, including *root*, run unconfined by default, which gives them full access to the system.

In addition to the unconfined user with unlimited privileges, SELinux includes seven confined user identities with restricted access to objects. These accounts are mapped to Linux users via SELinux policy. This regulated access helps safeguard the system from potential damage that Linux users might inflict on the system.

You can use the *seinfo* query command to list the SELinux users; however, the *setools-console* software package must be installed before doing so.

```
[user1@server10 ~]$ sudo seinfo -u

Users: 8
   guest_u
   root
   staff_u
   sysadm_u
   system_u
   unconfined_u
   user_u
   xguest_u
```

The output shows the eight predefined SELinux users. You can use the *semanage* command to view the mapping between Linux and SELinux users:

```
[user1@server10 ~]$ sudo 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       *
```

The output displays Linux users in column 1 (Login Name) and SELinux users they are mapped to in column 2 (SELinux User). Columns 3 and 4 show the associated security level (MLS/MCS Range), and the context for the Linux user (the * represents all services). By default, all non-*root* Linux users are represented as __default__, which is mapped to the unconfined_u user in the policy.

SELinux Contexts for Processes

You can determine the context for processes using the *ps* command with the -Z flag. The following example shows only the first two lines from the command output:

```
[user1@server10 ~]$ ps -eZ
LABEL                             PID TTY          TIME CMD
system_u:system_r:init_t:s0         1 ?        00:00:04 systemd
```

In the output, the subject (system_u) is a SELinux username (mapped to Linux user *root*), object is system_r, domain (init_t) reveals the type of protection applied to the process,

and level of security (s0). Any process that is unprotected will run in the unconfined_t domain.

SELinux Contexts for Files

You can spot the context for files and directories. To this end, use the *ls* command with the -Z switch. The following shows the four attributes set on the */etc/passwd* file:

```
[user1@server10 ~]$ ls -lZ /etc/passwd
-rw-r--r--. 1 root root system_u:object_r:passwd_file_t:s0 2809 Nov 11 07:24 /etc/passwd
```

The outcome indicates the subject (system_u), object (object_r), type (passwd_file_t), and security level (s0) for the *passwd* file. Contexts for system-installed and user-created files are stored in the *file_contexts* and *file_contexts.local* files located in the */etc/selinux/targeted/contexts/files* directory. These policy files can be updated using the *semanage* command.

Copying, Moving, and Archiving Files with SELinux Contexts

As mentioned, all files in RHEL are labeled with an SELINUX security context by default. New files inherit the parent directory's context at the time of creation. However, three common file management operations—copy, move, and archive—require special attention. There are certain rules to be kept in mind during their use to ensure correct contexts on affected files. These rules are:

1. If a file is copied to a different directory, the destination file will receive the destination directory's context, unless the --preserve=context switch is specified with the *cp* command to retain the source file's original context.
2. If a copy operation overwrites the destination file in the same or different directory, the file being copied will receive the context of the overwritten file, unless the --

preserve=context switch is specified with the *cp* command to preserve the source file's original context.

3. If a file is moved to the same or different directory, the SELinux context will remain intact, which may differ from the destination directory's context.
4. If a file is archived with the *tar* command, use the --selinux option to preserve the context.

Later in the chapter, we will perform an exercise to confirm the behavior of the three operations.

SELinux Contexts for Ports

SELinux contexts define security attributes for network ports, which can be viewed with the *semanage* command . The following illustrates a few entries from the output of this command:

```
[user1@server10 ~]$ sudo semanage port -l
SELinux Port Type              Proto    Port Number

chronyd_port_t                 udp      323
dns_port_t                     tcp      53, 853
ftp_port_t                     tcp      21, 989, 990
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009,
                                        8443, 9000
ntp_port_t                     udp      123
syslogd_port_t                 udp      514, 601, 20514
```

The output is displayed in three columns. Column 1 shows the SELinux type, column 2 depicts the protocol, and column 3 indicates the port number(s). By default, SELinux allows services to listen on a restricted set of network ports only. This is evident from the above output.

Domain Transitioning

SELinux allows a process running in one domain to enter another domain to execute an application that is restricted to run in that domain only, provided a rule exists in the policy to

support such transition. SELinux defines a permission setting called *entrypoint* in its policy to control processes that can transition into another domain. To understand how this works, a basic example is provided below that shows what happens when a Linux user attempts to change their password using the */usr/bin/passwd* command.

The *passwd* command is labeled with the passwd_exec_t type, which can be confirmed as follows:

```
[user1@server10 ~]$ ls -lZ /usr/bin/passwd
-rwsr-xr-x. 1 root root system_u:object_r:passwd_exec_t:s0 34512 Aug 12  2018 /usr/bin/pass
wd
```

The *passwd* command requires access to the */etc/shadow* file in order to modify a user password. The *shadow* file has a different type set on it (shadow_t):

```
[user1@server10 ~]$ ls -lZ /etc/shadow
----------. 1 root root system_u:object_r:shadow_t:s0 1613 Nov 11 07:25 /etc/shadow
```

The SELinux policy has rules that specifically allow processes running in domain passwd_t to read and modify the files with type shadow_t, and allow them entrypoint permission into domain passwd_exec_t. This rule enables the user's shell process executing the *passwd* command to switch into the passwd_t domain and update the *shadow* file.

Open two terminal windows. In window 1, issue the *passwd* command as *user1* and wait at the prompt:

```
[user1@server10 ~]$ passwd
Changing password for user user1.
Current password: _
```

In window 2, run the *ps* command:

```
[user1@server10 ~]$ ps -eZ | grep passwd
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023 14188 pts/1 00:00:00 passwd
```

As you can see, the *passwd* command (process) transitioned into the passwd_t domain to change the user password. A

process running in this domain is allowed to modify the content of the */etc/shadow* file.

SELinux Booleans

Booleans are on/off switches that SELinux uses to determine whether to permit an action. Booleans activate or deactivate certain rule in the SELinux policy immediately and without the need to recompile or reload the policy. For instance, the ftpd_anon_write Boolean can be turned on to enable anonymous users to upload files. This privilege can be revoked by turning this Boolean off. Boolean values are stored in virtual files located in the */sys/fs/selinux/booleans* directory. The filenames match the Boolean names. A sample listing of this directory is provided below:

```
[user1@server10 ~]$ ls -l /sys/fs/selinux/booleans
total 0
-rw-r--r--. 1 root root 0 Nov 30 20:20 abrt_anon_write
-rw-r--r--. 1 root root 0 Nov 30 20:20 abrt_handle_event
-rw-r--r--. 1 root root 0 Nov 30 20:20 abrt_upload_watch_anon_write
-rw-r--r--. 1 root root 0 Nov 30 20:20 antivirus_can_scan_system
-rw-r--r--. 1 root root 0 Nov 30 20:20 antivirus_use_jit
-rw-r--r--. 1 root root 0 Nov 30 20:20 auditadm_exec_content
-rw-r--r--. 1 root root 0 Nov 30 20:20 authlogin_nsswitch_use_ldap
.........
```

On a typical server, you'll see hundreds of Boolean files in the output.

The manual pages of the Booleans are available through the *selinux-policy-doc* package. Once installed, use the -K option with the *man* command to bring the pages up for a specific Boolean. For instance, issue **man -K abrt_anon_write** to view the manual pages for the abrt_anon_write Boolean.

Boolean values can be viewed, and flipped temporarily or for permanence. The new value takes effect right away. Temporary changes are stored as a "1" or "0" in the corresponding Boolean file in the */sys/fs/selinux/booleans*

directory and permanent changes are saved in the policy database.

One of the exercises in the next section demonstrates how to display and change Boolean values.

SELinux Administration

Managing SELinux involves plentiful tasks, including controlling the activation mode, checking operational status, setting security contexts on subjects and objects, and switching Boolean values. RHEL provides a set of commands to perform these operations. These commands are available through multiple packages, such as *libselinux-utils* provides *getenforce*, *getenforce*, and *getsebool* commands, *policycoreutils* contains *sestatus*, *setsebool*, and *restorecon* commands, *policycoreutils-python-utils* provides the *semanage* command, and *setools-console* includes the *seinfo* and *sesearch* commands.

For viewing alerts and debugging SELinux issues, a graphical tool called *SELinux Alert Browser* is available, which is part of the *setroubleshoot-server* package. In order to fully manage SELinux, you need to ensure that all these packages are installed on the system. Besides this toolset, there are additional utilities available to accomplish specific SELinux administration tasks, but their use is not as frequent.

Management Commands

SELinux delivers a variety of commands for effective administration. Table 21-1 lists and describes the commands mentioned above plus a few more under various management categories.

Command	Description
Mode Management	
getenforce	Displays the current mode of operation
sestatus	Shows SELinux runtime status and Boolean values
setenforce	Switches the operating mode between enforcing and permissive temporarily
Context Management	
chcon	Changes context on files (changes do not survive file system relabeling)
restorecon	Restores default contexts on files by referencing the files in the /etc/selinux/targeted/contexts/files directory
semanage	Changes context on files with the fcontext subcommand (changes survive file system relabeling)
Policy Management	
seinfo	Provides information on policy components
semanage	Manages policy database
sesearch	Searches rules in the policy database
Boolean Management	
getsebool	Displays Booleans and their current settings
setsebool	Modifies Boolean values temporarily, or in the policy database
semanage	Modifies Boolean values in the policy database with the boolean subcommand
Troubleshooting	
sealert	The graphical troubleshooting tool

Table 21-1 SELinux Management Commands

Most of these commands are examined in this chapter.

Viewing and Controlling SELinux Operational State

One of the key configuration files that controls the SELinux operational state, and sets its default type is the *config* file

located in the */etc/selinux* directory. The default content of the file are displayed below:

```
[user1@server10 ~]$ 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 - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protecte
d.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

The SELINUX directive in the file sets the activation mode for SELinux. *Enforcing* activates it and allows or denies actions based on the policy rules. *Permissive* activates SELinux, but permits all actions. It records all security violations. This mode is useful for troubleshooting and in developing or tuning the policy. The third option is to completely turn SELinux off. When running in enforcing mode, the SELINUXTYPE directive dictates the type of policy to be enforced. The default SELinux type is targeted.

Issue the *getenforce* command to determine the current operating mode:

```
[user1@server10 ~]$ getenforce
Enforcing
```

The output returns enforcing as the current active policy. You may flip the state to permissive using the *setenforce* command, and verify the change with *getenforce*:

```
[user1@server10 ~]$ sudo setenforce permissive
[user1@server10 ~]$ getenforce
Permissive
```

You may alternatively use a “0” for permissive and a “1” for enforcing.

The change takes effect at once; however, it will be lost at the next system reboot. To make it persistent, edit the */etc/selinux/config* file and set the SELINUX directive to the desired mode.

> **EXAM TIP:** You may switch SELinux to permissive for troubleshooting a non-functioning service. Change it back to enforcing when the issue is resolved.

To disable SELinux completely, the SELINUX directive needs to be set to disabled in the *config* file, and the system must be rebooted. Reactivation in the future to either enforcing or permissive will require the mode resetting in the *config* file followed by a system reboot. The reboot will take longer than normal, as SELinux will go through the process of relabeling all the files.

Querying Status

The current runtime status of SELinux can be viewed with the *sestatus* command. This command also displays the location of principal directories, the policy in effect, and the activation mode.

```
[user1@server10 ~]$ 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
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      31
```

The output reveals that SELinux is enabled (SELinux status), and it is running in permissive mode (Current mode) with the targeted policy in effect (Loaded policy name). It also indicates the current mode setting in the *config* file (Mode from config file) along with other information.

The *sestatus* command may be invoked with the -v switch to report on security contexts set on files and processes, as listed in the */etc/sestatus.conf* file. The default content of this file is shown below:

```
[user1@server10 ~]$ cat /etc/sestatus.conf
[files]
/etc/passwd
/etc/shadow
/bin/bash
/bin/login
/bin/sh
/sbin/agetty
/sbin/init
/sbin/mingetty
/usr/sbin/sshd
/lib/libc.so.6
/lib/ld-linux.so.2
/lib/ld.so.1

[process]
/sbin/mingetty
/sbin/agetty
/usr/sbin/sshd
```

Run the *sestatus* command with -v:

```
[user1@server10 ~]$ sestatus -v
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

Process contexts:
Current context:                unconfined_u:unconfined_r:unconfined_t:s0-s0:
c1023
Init context:                   system_u:system_r:init_t:s0

File contexts:
Controlling terminal:           unconfined_u:object_r:user_devpts_t:s0
/etc/passwd                     system_u:object_r:passwd_file_t:s0
/etc/shadow                     system_u:object_r:shadow_t:s0

........
```

With -v included, the command reports the contexts for the current process (Current context) and the *init* (*systemd*)

process (Init context) under Process Contexts. It also reveals the file contexts for the controlling terminal and associated files under File Contexts.

Exercise 21-1: Modify SELinux File Context

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will create a directory *sedir1* under */tmp* and a file *sefile1* under *sedir1*. You will check the context on the directory and file. You will change the SELinux user and type to user_u and public_content_t on both and verify.

1. Create the hierarchy *sedir1/sefile1* under */tmp*:

```
[user1@server10 ~]$ cd /tmp
[user1@server10 tmp]$ mkdir sedir1
[user1@server10 tmp]$ touch sedir1/sefile1
```

2. Determine the context on the new directory and file:

```
[user1@server10 tmp]$ ls -ldZ sedir1
drwxrwxr-x. 2 user1 user1 unconfined_u:object_r:user_tmp_t:s0 21 Dec  3 11:38 sedir1
[user1@server10 tmp]$ ls -lZ sedir1/sefile1
-rw-rw-r--. 1 user1 user1 unconfined_u:object_r:user_tmp_t:s0 0 Dec  3 11:38 sedir1/sefile1
```

The directory and the file get unconfined_u and user_tmp_t as the SELinux user and type.

3. Modify the SELinux user (-u) on the directory to user_u and type (-t) to public_content_t recursively (-R) with the *chcon* command:

```
[user1@server10 tmp]$ sudo chcon -vu user_u -t public_content_t sedir1 -R
changing security context of 'sedir1/sefile1'
changing security context of 'sedir1'
```

4. Validate the new context:

```
[user1@server10 tmp]$ ls -ldZ sedir1
drwxrwxr-x. 2 user1 user1 user_u:object_r:public_content_t:s0 21 Dec  3 11:38 sedir1
[user1@server10 tmp]$ ls -lZ sedir1/sefile1
-rw-rw-r--. 1 user1 user1 user_u:object_r:public_content_t:s0 0 Dec  3 11:38 sedir1/sefile1
```

This concludes the exercise.

Exercise 21-2: Add and Apply File Context

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will add the current context on *sedir1* to the SELinux policy database to ensure a relabeling will not reset it to its previous value (see Exercise 21-1). Next, you will change the context on the directory to some random values. You will restore the default context from the policy database back to the directory recursively.

1. Determine the current context:

```
[user1@server10 tmp]$ ls -ldZ sedir1
drwxrwxr-x. 2 user1 user1 user_u:object_r:public_content_t:s0 21 Dec  3 11:38 sedir1
[user1@server10 tmp]$ ls -lZ sedir1/sefile1
-rw-rw-r--. 1 user1 user1 user_u:object_r:public_content_t:s0 0 Dec  3 11:38 sedir1/sefile1
```

The output indicates the current SELinux user (user_u) and type (public_content_t) set on the directory and the file.

2. Add (-a) the directory recursively to the policy database using the *semanage* command with the *fcontext* subcommand:

```
[user1@server10 tmp]$ sudo semanage fcontext -a -s user_u -t public_content_t '/tmp/sedir1(/.*)?'
```

The regular expression (/.*)? instructs the command to include all files and subdirectories under */tmp/sedir1*. This expression is needed only if recursion is required.

The above command added the context to the */etc/selinux/targeted/contexts/files/file_contexts.local* file. You can use the *cat* command to view the content.

3. Validate the addition by listing (-l) the recent changes (-C) in the policy database:

```
[user1@server10 tmp]$ sudo semanage fcontext -Cl | grep sedir
/tmp/sedir1(/.*)?        all files        user_u:object_r:public_content_t:s0
```

4. Change the current context on *sedir1* to something random (staff_u/etc_t) with the *chcon* command:

```
[user1@server10 tmp]$ sudo chcon -vu staff_u -t etc_t sedir1 -R
changing security context of 'sedir1/sefile1'
changing security context of 'sedir1'
```

5. The security context is changed successfully. Confirm with the *ls* command:

```
[user1@server10 tmp]$ ls -ldZ sedir1 ; ls -lZ sedir1/sefile1
drwxrwxr-x. 2 user1 user1 staff_u:object_r:etc_t:s0 21 Dec  3 11:38 sedir1
-rw-rw-r--. 1 user1 user1 staff_u:object_r:etc_t:s0 0 Dec  3 11:38 sedir1/sefile1
```

6. Reinstate the context on the *sedir1* directory recursively (-R) as stored in the policy database using the *restorecon* command:

```
[user1@server10 tmp]$ sudo restorecon -Rv sedir1
Relabeled /tmp/sedir1 from staff_u:object_r:etc_t:s0 to staff_u:object_r:public_content_t:s0
Relabeled /tmp/sedir1/sefile1 from staff_u:object_r:etc_t:s0 to staff_u:object_r:public_content_t:s0
```

The output confirms the restoration of the default context on the directory and the file.

EXAM TIP: Use the combination of semanage and restorecon commands to add a file context to the SELinux policy and then apply it. This will prevent the context on file to reset to the original value in the event of SELinux relabeling (disabled to enforcing/permissive).

Exercise 21-3: Add and Delete Network Ports

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will add a non-standard network port 8010 to the SELinux policy database for the *httpd* service and confirm the addition. You will then remove the port from the policy and verify the deletion.

1. List (-l) the ports for the *httpd* service as defined in the SELinux policy database:

```
[user1@server10 ~]$ sudo semanage port -l | grep ^http_port
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
```

The output reveals eight network ports the *httpd* process is currently allowed to listen on.

2. Add (-a) port 8010 with type (-t) http_port_t and protocol (-p) tcp to the policy:

```
[user1@server10 ~]$ sudo semanage port -at http_port_t -p tcp 8010
```

3. Confirm the addition:

```
[user1@server10 ~]$ sudo semanage port -l | grep ^http_port
http_port_t                    tcp      8010, 80, 81, 443, 488, 8008, 8009, 8443, 9000
```

The new network port is visible in the outcome.

4. Delete (-d) port 8010 from the policy and confirm:

```
[user1@server10 ~]$ sudo semanage port -d -p tcp 8010
[user1@server10 ~]$ sudo semanage port -l | grep ^http_port
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
```

The port is removed from the policy database.

> **EXAM TIP:** Any non-standard port you want to use for any service, make certain to add it to the SELinux policy database with the correct type.

Exercise 21-4: Copy Files with and without Context

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will create a file called *sefile2* under */tmp* and display its context. You will copy this file to the */etc/default* directory, and observe the change in the context.

You will remove se*file2* from */etc/default*, and copy it again to the same destination, ensuring that the target file receives the source file's context.

1. Create file *sefile2* under */tmp* and show context:

```
[user1@server10 ~]$ touch /tmp/sefile2
[user1@server10 ~]$ ls -lZ /tmp/sefile2
-rw-rw-r--. 1 user1 user1 unconfined_u:object_r:user_tmp_t:s0 0 Dec  3 17:54 /tmp/sefile2
```

The context on the file is unconfined_u, object_r, and user_tmp_t.

2. Copy this file to the */etc/default* directory, and check the context again:

```
[user1@server10 ~]$ sudo cp /tmp/sefile2 /etc/default
[user1@server10 ~]$ ls -lZ /etc/default/sefile2
-rw-r--r--. 1 root root unconfined_u:object_r:etc_t:s0 0 Dec  3 17:57 /etc/default/sefile2
```

The target file (*/etc/default/sefile2*) received the default context of the destination directory (*/etc/default*).

3. Remove the */etc/default/sefile2* file, and copy it again with the --preserve=context option:

```
[user1@server10 ~]$ sudo rm /etc/default/sefile2
[user1@server10 ~]$ sudo cp /tmp/sefile2 /etc/default --preserve=context
```

4. List the file to view the context:

```
[user1@server10 ~]$ ls -lZ /etc/default/sefile2
-rw-r--r--. 1 root root unconfined_u:object_r:user_tmp_t:s0 0 Dec  3 18:02 /etc/default/sefile2
```

The original context (user_tmp_t) is preserved on the target file after the copy operation has finished.

Exercise 21-5: View and Toggle SELinux Boolean Values

This exercise should be done on *server10* as *user1* with *sudo* where required.

In this exercise, you will display the current state of the Boolean nfs_export_all_rw. You will toggle its value temporarily, and reboot the system. You will flip its value persistently after the system has been back up.

1. Display the current setting of the Boolean nfs_export_all_rw using three different commands—*getsebool*, *sestatus*, and *semanage*:

```
[user1@server10 ~]$ sudo getsebool -a | grep nfs_export_all_rw
nfs_export_all_rw --> on
[user1@server10 ~]$ sudo sestatus -b | grep nfs_export_all_rw
nfs_export_all_rw                           on
[user1@server10 ~]$ sudo semanage boolean -l | grep nfs_export_all_rw
nfs_export_all_rw              (on   ,   on)  Allow nfs to export all rw
```

2. Turn off the value of nfs_export_all_rw using the *setsebool* command by simply furnishing “off” or “0” with it and confirm:

```
[user1@server10 ~]$ sudo setsebool nfs_export_all_rw 0
[user1@server10 ~]$ sudo getsebool -a | grep nfs_export_all_rw
nfs_export_all_rw --> off
```

3. Reboot the system and rerun the *getsebool* command to check the Boolean state:

```
[user1@server10 ~]$ sudo getsebool -a | grep nfs_export_all_rw
nfs_export_all_rw --> on
```

The value reverted to its previous state.

4. Set the value of the Boolean persistently (-P or -m as needed) using either of the following:

```
[user1@server10 ~]$ sudo setsebool -P nfs_export_all_rw off
[user1@server10 ~]$ sudo semanage boolean -m -0 nfs_export_all_rw
```

5. Validate the new value using the *getsebool*, *sestatus*, or *semanage* command:

```
[user1@server10 ~]$ sudo getsebool nfs_export_all_rw
nfs_export_all_rw --> off
[user1@server10 ~]$ sudo sestatus -b | grep nfs_export_all_rw
nfs_export_all_rw                           off
[user1@server10 ~]$ sudo semanage boolean -l | grep nfs_export_all_rw
nfs_export_all_rw              (off  ,  off)  Allow nfs to export all rw
```

The command outputs confirm the permanent change.

Monitoring and Analyzing SELinux Violations

SELinux generates alerts for system activities when it runs in enforcing or permissive mode. It writes the alerts to the */var/log/audit/audit.log* file if the *auditd* daemon is running, or to the */var/log/messages* file via the *rsyslog* daemon in the absence of *auditd*. SELinux also logs the alerts that are generated due to denial of an action, and identifies them with a type tag AVC (*Access Vector Cache*) in the *audit.log* file. It also writes the rejection in the *messages* file with a message ID, and how to view the message details.

If it works with SELinux in permissive mode and not in enforcing, something needs to be adjusted in SELinux.

SELinux denial messages are analyzed and the audit data is examined to identify the potential cause of the rejection. The results of the analysis are recorded with recommendations on how to fix it. These results can be reviewed to aid in troubleshooting, and recommended actions taken to address the issue. SELinux runs a service daemon called *setroubleshootd* that performs this analysis and examination in the background. This service also has a client interface called *SELinux Troubleshooter* (the *sealert* command) that reads the data and displays it for assessment. The client tool has both text and graphical interfaces. The server and client components are part of the *setroubleshoot-server* software package that must be installed on the system prior to using this service.

The following shows a sample allowed record in raw format from the */var/log/audit/audit.log* file:

```
type=USER_ACCT msg=audit(1575462873.869:744): pid=7496 uid=0 auid=1000 ses=2 subj
=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=PAM:accounting gra
ntors=pam_succeed_if acct="root" exe="/usr/bin/su" hostname=server10.example.com
addr=? terminal=pts/0 res=success'UID="root" AUID="user1"
```

The record shows a successful *user1* attempt to *su* into the *root* user account on *server10*.

The following is a sample denial record from the same file in raw format:

```
type=AVC msg=audit(1575464056.692:782): avc:  denied  { create } for  pid=7677 co
mm="passwd" name="nshadow" scontext=unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c
1023 tcontext=system u:object r:etc t:s0 tclass=file permissive=0
```

The message has the AVC type, and it is related to the *passwd* command (comm) with source context (scontext) “unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023”, and the *nshadow* file (name) with file type (tclass) “file”, and target context (tcontext) “system_u:object_r:etc_t:s0”. It also indicates the SELinux operating mode, which is enforcing (permissive=0). This message indicates that the */etc/shadow* file does not have the correct context set on it, and that’s why SELinux prevented the *passwd* command from updating the user’s password.

You can also use the *sealert* command to analyze (-a) all AVC records in the *audit.log* file. This command produces a formatted report with all relevant details:

```
[user1@server10 ~]$ sudo sealert -a /var/log/audit/audit.log
100% done
found 1 alerts in /var/log/audit/audit.log
--------------------------------------------------------------------------------

SELinux is preventing /usr/bin/passwd from create access on the file nshadow.

.....
 Additional Information:
 Source Context                unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023
 Target Context                system_u:object_r:etc_t:s0
 Target Objects                nshadow [ file ]
 Source                        passwd
 Source Path                   /usr/bin/passwd
 Port                          <Unknown>
 Host                          <Unknown>
 Source RPM Packages           passwd-0.80-2.el8.x86_64
 Target RPM Packages
 Policy RPM                    selinux-policy-3.14.1-61.el8.noarch
 Selinux Enabled               True
 Policy Type                   targeted
 Enforcing Mode                Enforcing
 Host Name                     server10.example.com
 Platform                      Linux server10.example.com
                               4.18.0-80.11.2.el8_0.x86_64 #1 SMP Sun Sep 15
                               11:24:21 UTC 2019 x86_64 x86_64
 Alert Count                   1
 First Seen                    2019-12-04 07:54:16 EST
 Last Seen                     2019-12-04 07:54:16 EST
 Local ID                      2cbcdcb9-c14d-460f-a012-cc56888c5d27

.....
```

The above SELinux denial was due to the fact that I produced the scenario by changing the SELinux type on the *shadow* file to something random (etc_t). I then issued the *passwd* command as *user1* to modify the password. As expected, SELinux disallowed the *passwd* command to write the new password to the *shadow* file, and it logged the password rejection attempt to the audit log. I then restored the type on the *shadow* file with **restorecon /etc/shadow**. I re-tried the password change and it worked.

Chapter Summary

In this last chapter of the book, we discussed security enhanced Linux in fair detail. We looked at the concepts, features, and terminology at length. We examined how security contexts are associated with users, processes, files, and ports, and viewed and modified contexts for them. We analyzed the configuration file that controls its state and

defines the policy to be enforced. We examined how domain transitioning works. We learned several SELinux administrative commands and performed tasks such as checking and switching activation mode and operational status. We studied the concept of Booleans and learned how to modify certain parts of the SELinux policy temporarily and persistently. Finally, we reviewed the SELinux Troubleshooter program and used it to view and analyze SELinux related messages.

Check Your Understanding

1. What is the name and location of the SELinux configuration file?
2. What would the command *semanage fcontext -Cl* do?
3. Which command can be used to ensure modified contexts will survive file system relabeling?
4. What would the command *semanage login -a -s user_u user10* do?
5. Name the two commands that can be used to modify a Boolean value.
6. Which option is used with the *ps* command to view the security contexts for processes?
7. What would the command *restorecon -F /etc/sysconfig* do?
8. What is the name of the default SELinux policy used in RHEL 8?
9. What would the command *sestatus -b | grep nfs_export_all_rw* do?
10. Name the directory that stores SELinux Boolean files.
11. What are the two commands to display and modify the SELinux mode?
12. Name the two SELinux subjects.
13. SELinux is an implementation of discretionary access control. True or False?

14. Where are SELinux denial messages logged in the absence of the *auditd* daemon?
15. Name the four parts of a process context.
16. What one task must be done to change the SELinux mode from enforcing to disabled?
17. Which option with the *cp* command must be specified to preserve SELinux contexts?
18. What is the purpose of the command *sestatus*?
19. With SELinux running in enforcing mode and a service is compromised, all other services are affected. True or False?
20. Name the command that starts the SELinux Troubleshooter program.

Answers to Check Your Understanding

1. The SELinux configuration filename is *config* and it is located in the */etc/selinux* directory.
2. The command provided will show recent changes made to the SELinux policy database.
3. The *semanage* command.
4. This command provided will map Linux user *user10* with SELinux user user_u.
5. The *semanage* and *setsebool* commands.
6. The -Z option.
7. The command provided will restore the default SELinux contexts on the specified directory.
8. The default SELinux policy used in RHEL 8 is targeted.
9. The command provided will display the current value of the specified Boolean.
10. The */sys/fs/selinux/booleans* directory.
11. The *getenforce* and *setenforce* commands.
12. User and process are two SELinux subjects.
13. False. SELinux is an implementation of mandatory access control.

14. The SELinux denial messages are logged to the */var/log/messages* file.
15. The four parts of a process context are user, role, type/domain, and sensitivity level.
16. The system must be rebooted.
17. The --preserve=context option.
18. The *sestatus* command displays SELinux status information.
19. False.
20. The command name is *sealert*.

Do-It-Yourself Challenge Labs

The following labs are useful to strengthen most of the concepts and topics learned in this chapter. It is expected that you perform the labs without external help. A step-by-step guide is not supplied, as the knowledge and skill required to implement the lab has already been disseminated in the chapter; however, hints to the relevant major topic(s) are included.

Lab 21-1: Disable and Enable the SELinux Operating Mode

As *user1* with *sudo* on *server10*, check and make a note of the current SELinux operating mode. Modify the configuration file and set the mode to disabled. Reboot the system to apply the change. Run **sudo getenforce** to confirm the change when the system is up. Restore the directive's value to enforcing in the configuration file, and reboot to apply the new mode. Run **sudo getenforce** to confirm the mode when the system is up. (Hint: SELinux Administration).

Lab 21-2: Modify Context on Files

As *user1* with *sudo* on *server10*, create directory hierarchy */tmp/d1/d2*. Check the contexts on */tmp/d1* and */tmp/d1/d2*. Change the SELinux type on */tmp/d1* to etc_t recursively with

the *chcon* command and confirm. Add */tmp/d1* to the policy database with the *semanage* command to ensure the new context is persistent on the directory hierarchy. (Hint: SELinux Administration).

Lab 21-3: Add Network Port to Policy Database

As *user1* with *sudo* on *server10*, add network port 9001 to the SELinux policy database for the secure HTTP service using the *semanage* command. Verify the addition. (Hint: SELinux Administration).

Lab 21-4: Copy Files with and without Context

As *user1* with *sudo* on *server10*, create file *sef1* under */tmp*. Copy the file to the */usr/local* directory. Check and compare the contexts on both source and destination files. Create another file *sef2* under */tmp* and copy it to the */var/local* directory using the --preserve=context option with the *cp* command. Check and compare the contexts on both source and destination files. (Hint: SELinux Administration).

Lab 21-5: Flip SELinux Booleans

As *user1* with *sudo* on *server10*, check the current value of Boolean *ssh_use_tcpd* using the *getsebool* and *sestatus* commands. Use the *setsebool* command and toggle the value of the directive. Confirm the new value with the *getsebool*, *semanage*, or *sestatus* command. (Hint: SELinux Administration).

Appendix A: Sample RHCSA Exam 1

Time Duration: 2.5 hours
Passing Score: 70% (210 out of 300)
Instructions: The RHCSA exam, EX200, is offered electronically on a desktop system running RHEL 8. The exam presents a list of tasks that are to be completed within the stipulated time. Firewall and SELinux need to be considered. All settings performed on the systems must survive system reboots, or you will not be given credits. Access to the Internet, printed material, and electronic devices is prohibited during the exam.

Setup for the Sample Exam 1:
Build a virtual machine with RHEL 8 Server for GUI (Exercises 1-1 and 1-2). Add a 10GB disk for the OS and use the default storage partitioning. Add 2x300MB disks. Add a network interface, but do not configure the hostname and network connection. Do not

create a normal user account during installation.

Instructions:

Instruction 01: The following tasks are in addition to the exercises and labs presented in the book. No solutions are furnished, but hints to applicable exercises, chapters, or topics are provided in parentheses for reference.

Instruction 02: Do not browse the Internet or other material for help. This rule does not apply to the kernel download task if included. You can refer to the manual pages, and the documentation under the /usr/share/doc directory.

Instruction 03: This exam should be done in a terminal window using only the command line interface (no GUI).

Instruction 04: You can reboot the VM whenever you want during this exam, but retest the configuration after the reboot for verification.

Instruction 05: Use your knowledge and judgement for any missing configuration in task description.

Tasks:

Task 01: Assuming the root user password is lost, and your system is running in multi-user target with no current root session open. Reboot the system into an appropriate target, and reset the root user password to root1234. (Exercise 11-1). Log in with root and perform the remaining tasks.

Task 02: Using a manual method (create/modify files by hand), configure a network connection on the primary network device with IP address 192.168.0.241/24, gateway

192.168.0.1, and nameserver 192.168.0.1 (you may use different IP assignments based on your lab environment). (Exercise 17-3).

Task 03: Using a manual method (modify file by hand), set the system hostname to rhcsa1.example.com and alias rhcsa1. Make sure that the new hostname is reflected in the command prompt. (Exercises 17-1 and 17-5).

Task 04: Set the default boot target to multi-user. (Chapter 12, topic: Managing Target Units).

Task 05: Set SELinux to permissive mode. (Chapter 21, topic: Viewing and Controlling SELinux Operational State).

Task 06: Perform a case-insensitive search for all lines in the /usr/share/dict/linux.words file that begin with the pattern “essential”. Redirect the output to /tmp/pattern.txt file. Make sure that empty lines are omitted. (Chapter 07, topic: Regular Expressions).

Task 07: Change the primary command prompt for the root user to display the hostname, username, and current working directory information in that order. Update the per-user initialization file for permanence. (Exercise 7-1).

Task 08: Create user accounts called user10, user20, and user30. Set their passwords to Temp1234. Make accounts for user10 and user30 to expire on December 31, 2021. (Exercises 5-1 and 6-1).

Task 09: Create a group called group10 and add users user20 and user30 as secondary members. (Exercise 6-4).

Task 10: Create a user account called user40 with UID 2929. Set the password to user1234. (Exercise 5-2).

Task 11: Create a directory called dir1 under /tmp with ownership and owning group set to root. Configure default ACLs on the directory and give user user10 read, write, and execute permissions. (Exercise 4-8).

Task 12: Attach the RHEL 8 ISO image to the VM and mount it persistently to /mnt/cdrom. Define access to both repositories and confirm. (Chapter 09, topic: Getting Ready to Use rpm, and Exercise 10-1).

Task 13: Create a logical volume called lvol1 of size 300MB in vgtest volume group. Mount the Ext4 file system persistently to /mnt/mnt1. (Exercises 14-1, 14-2, and 15-3).

Task 14: Change group membership on /mnt/mnt1 to group10. Set read/write/execute permissions on /mnt/mnt1 for group members, and revoke all permissions for public. (Exercises 6-4, 6-6, and either 4-1 or 4-2).

Task 15: Create a logical volume called lvswap of size 300MB in vgtest volume group. Initialize the logical volume for swap use. Use the UUID and place an entry for persistence. (Exercise 15-6).

Task 16: Use tar and bzip2 to create a compressed archive of the /etc/sysconfig directory. Store the archive under /tmp as etc.tar.bz2. (Exercise 3-1).

Task 17: Create a directory hierarchy /dir1/dir2/dir3/dir3, and apply SELinux contexts for /etc on it recursively. (Chapter 03, topic: Creating Files and Directories, and Exercise 21-2).

Task 18: Enable access to the atd service for user20 and deny for user30. (Chapter 08, topic: Controlling User Access).

Task 19: Add a custom message “This is RHCSA sample exam on $(date) by $LOGNAME” to the /var/log/messages file as the root user. Use regular expression to confirm the message entry to the log file. (Chapter 07, topic: Regular Expressions, and Chapter 12, topic: Logging Custom Messages).

Task 20: Allow user20 to use sudo without being prompted for their password. (Chapter 06, topic: Doing as Superuser or Doing as Substitute User).

Reboot the system and validate the configuration.

Appendix B: Sample RHCSA Exam 2

Time Duration: 2.5 hours
Passing Score: 70% (210 out of 300)
Instructions: The RHCSA exam, EX200, is offered electronically on a desktop system running RHEL 8. The exam presents a list of tasks that are to be completed within the stipulated time. Firewall and SELinux need to be considered. All settings performed on the systems must survive system reboots, or you will not be given credits. Access to the Internet, printed material, and electronic devices is prohibited during the exam.

Setup for the Sample Exam 2:
Build a virtual machine with RHEL 8 Server for GUI (Exercises 1-1 and 1-2). Add a 10GB disk for the OS and use the default storage partitioning. Add one 400MB disk. Add a network interface, but do not configure the hostname and network connection. Do not

create a normal user account during installation.

Instructions:

Instruction 01: The following tasks are in addition to the exercises and labs presented in the book. No solutions are furnished, but hints to applicable exercises, chapters, or topics are provided in parentheses for reference.

Instruction 02: Do not browse the Internet or other material for help. This rule does not apply to the kernel download task if included. You can refer to the manual pages, and the documentation under the /usr/share/doc directory.

Instruction 03: This exam should be done in a terminal window using only the command line interface (no GUI).

Instruction 04: You can reboot the VM whenever you want during this exam, but retest the configuration after the reboot for verification.

Instruction 05: Use your knowledge and judgement for any missing configuration in task description.

Tasks:

Task 01: Using the nmcli command, configure a network connection on the primary network device with IP address 192.168.0.242/24, gateway 192.168.0.1, and nameserver 192.168.0.1 (you may use different IP assignments based on your lab environment). (Exercise 17-4).

Task 02: Using the hostnamectl command, set the system hostname to rhcsa2.example.com and alias rhcsa2. Make

sure that the new hostname is reflected in the command prompt. (Exercises 17-1 and 17-5).

Task 03: Create a user account called user70 with UID 7000 and comments “I am user70”. Set the maximum allowable inactivity for this user to 30 days. (Exercises 5-2, and 6-1 or 6-2).

Task 04: Create a user account called user50 with a non-interactive shell. (Exercise 5-4).

Task 05: Create a file called testfile1 under /tmp with ownership and owning group set to root. Configure access ACLs on the file and give user10 read and write access. Test access by logging in as user10 and editing the file. (Chapter 03, topic: Creating Files and Directories, and Exercise 4-7).

Task 06: Attach the RHEL 8 ISO image to the VM and mount it persistently to /mnt/dvdrom. Define access to both repositories and confirm. (Chapter 09, topic: Getting Ready to Use rpm, and Exercise 10-1).

Task 07: Create a logical volume called lv1 of size equal to 10 LEs in vg1 volume group (create vg1 with PE size 8MB in a partition on the 400MB disk) Initialize the logical volume with XFS file system type and mount it on /mnt/lvfs1. Create a file called lv1file1 in the mount point. Set the file system to automatically mount at each system reboot. (Exercises 14-1, 14-2, and 15-3).

Task 08: Add a group called group20 and change group membership on /mnt/lvfs1 to group20. Set read/write/execute permissions on /mnt/lvfs1 for the owner and group members, and no permissions for others. (Exercises 6-4, 6-6, and either 4-1 or 4-2).

Task 09: Extend the file system in the logical volume lv1 by 64MB without unmounting it and without losing any data. (Exercise 15-4).

Task 10: Create a swap partition of size 85MB on the 400MB disk. Use its UUID and ensure it is activated after every system reboot. (Exercise 15-6).

Task 11: Create a disk partition of size 100MB on the 400MB disk and format it with Ext4 file system structures. Assign label stdlabel to the file system. Mount the file system on /mnt/stdfs1 persistently using the label. Create file stdfile1 in the mount point. (Exercise 13-2 or Exercise 13-4, Chapter 15, topic: Labeling a File System, and Exercise 15-1).

Task 12: Use tar and gzip to create a compressed archive of the /usr/local directory. Store the archive under /tmp using a filename of your choice. (Exercise 3-1).

Task 13: Create a directory /direct01 and apply SELinux contexts for /root. (Exercise 21-2).

Task 14: Set up a cron job for user70 to search for core files in the /var directory and copy them to the directory /tmp/coredir1. This job should run every Monday at 1:20 a.m. (Chapter 04, topics: Using the find Command, and Using find with -exec and -ok Flags, and Exercise 8-2).

Task 15: Search for all files in the entire directory structure that have been modified in the past 30 days and save the file listing in the /var/tmp/modfiles.txt file. (Chapter 04, topics: Using the find Command and Using find with -exec and -ok Flags).

Task 16: Modify the bootloader program and set the default autoboot timer value to 2 seconds. (Chapter 11, topic:

Understanding GRUB2 Configuration Files).

Task 17: Determine the recommended tuning profile for the system and apply it. (Exercise 12-1).

Task 18: Configure Chrony to synchronize system time with the hardware clock. (Exercise 18-1).

Task 19: Install package group called “Development Tools”, and capture its information in /tmp/systemtools.out file. (Chapter 03, topic: Regular Expressions, and Exercise 10-3).

Task 20: Lock user account user70. Use regular expressions to capture the line that shows the lock and store the output in file /tmp/user70.lock. (Chapter 03, topic: Regular Expressions, and Exercise 6-3).

Reboot the system and validate the configuration.

Appendix C: Sample RHCSA Exam 3

Time Duration: 2.5 hours
Passing Score: 70% (210 out of 300)
Instructions: The RHCSA exam, EX200, is offered electronically on a desktop system running RHEL 8. The exam presents a list of tasks that are to be completed within the stipulated time. Firewall and SELinux need to be considered. All settings performed on the systems must survive system reboots, or you will not be given credits. Access to the Internet, printed material, and electronic devices is prohibited during the exam.

Setup for the Sample Exam 3:

Build two virtual machines with RHEL 8 Server for GUI (Exercises 1-1 and 1-2). Add a 10GB disk for the OS and use the default storage partitioning. Add 1x4GB disk to VM1 and 2x1GB to VM2. Add a network interface, but do not configure the hostname and network connection. Do not create a normal user account during installation.

Instructions:

Instruction 01: The following tasks are in addition to the exercises and labs presented in the book. No solutions are furnished, but hints to applicable exercises, chapters, or topics are provided in parentheses for reference.

Instruction 02: Do not browse the Internet or other material for help. This rule does not apply to the kernel download task if included. You can refer to the manual pages, and the documentation under the /usr/share/doc directory.

Instruction 03: This exam should be done in a terminal window using only the command line interface (no GUI).

Instruction 04: You can reboot the VM whenever you want during this exam, but retest the configuration after the reboot for verification.

Instruction 05: Use your knowledge and judgement for any missing configuration in task description.

Tasks:

Task 01: On VM1, set the system hostname to rhcsa3.example.com and alias rhcsa3 using the hostnamectl command. Make sure that the new hostname is reflected in the command prompt. (Exercises 17-1 and 17-5).

Task 02: On rhcsa3, configure a network connection on the primary network device with IP address 192.168.0.243/24, gateway 192.168.0.1, and nameserver 192.168.0.1 using the nmcli command. You may use different IP assignments based on your lab environment. (Exercise 17-4).

Task 03: On VM2, set the system hostname to rhcsa4.example.com and alias rhcsa4 using a manual

method (modify file by hand). Make sure that the new hostname is reflected in the command prompt. (Exercises 7-1 and 17-5).

Task 04: On rhcsa4, configure a network connection on the primary network device with IP address 192.168.0.244/24, gateway 192.168.0.1, and nameserver 192.168.0.1 using a manual method (create/modify files by hand). You may use different IP assignments based on your lab environment. (Exercise 17-3).

Task 05: Run “ping -c2 rhcsa4” on rhcsa3. Run “ping -c2 rhcsa3” on rhcsa4. You should see 0% loss in both outputs. (Exercise 17-5).

Task 06: On rhcsa3 and rhcsa4, attach the RHEL 8 ISO image to the VM and mount it persistently to /mnt/cdrom. Define access to both repositories and confirm. (Chapter 09, topic: Getting Ready to use rpm, and Exercise 10-1).

Task 07: On rhcsa3, add HTTP port 8300/tcp to the SELinux policy database. (Exercise 21-3).

Task 08: On rhcsa3, create VDO volume vdo1 on the 4GB disk with logical size 16GB and mounted with Ext4 structures on /mnt/vdo1. (Exercises 13-6 and 13-7).

Task 09: Configure NFS service on rhcsa3 and share /rh_share3 with rhcsa4. Configure AutoFS direct map on rhcsa4 to mount /rh_share3 on /mnt/rh_share4. User user80 (create on both systems) should be able to create files under the share on the NFS server and under the mount point on the NFS client. (Exercises 16-1 and 16-3).

Task 10: Configure NFS service on rhcsa4 and share the home directory for user user60 (create on both systems) with rhcsa3. Configure AutoFS indirect map on rhcsa3 to

automatically mount the home directory under /nfsdir when user60 logs on to rhcsa3. (Exercises 16-1 and 16-4, and topic: Automounting User Home Directories).

Task 11: On rhcsa4, create Stratis pool pool1 and volume str1 on a 1GB disk, and mount it to /mnt/str1. (Exercise 15-5).

Task 12: On rhcsa4, expand Stratis pool pool1 using the other 1GB disk. Confirm that /mnt/str1 sees the storage expansion. (Exercise 15-5).

Task 13: On rhcsa3, create a group called group30 with GID 3000, and add user60 and user80 to this group. Create a directory called /sdata, enable setgid bit on it, and add write permission bit for the group. Set ownership and owning group to root and group30. Create a file called file1 under /sdata as user user60 and modify the file as user80 successfully. (Exercises 4-5, 6-4, and 6-6).

Task 14: On rhcsa3, create directory /dir1 with full permissions for everyone. Disallow non-owners to remove files. Test by creating file /tmp/dir1/stkfile1 as user60 and removing it as user user80. (Exercise 4-6).

Task 15: On rhcsa3, search for all manual pages for the description containing the keyword “password” and redirect the output to file /tmp/man.out. (Chapter 02, topic Searching by Keyword, and Chapter 07, topic: Regular Expressions).

Task 16: On rhcsa3, create file lnfile1 under /tmp and create one hard link /tmp/lnfile2 and one soft link /boot/file1. Edit lnfile1 using the links and confirm. (Exercises 3-2 and 3-3).

Task 17: On rhcsa3, install module postgresql version 9.6 (select a different non-default version if 9.6 is not available). (Exercise 10-5).

Task 18: On rhcsa3, add the http service to the “external” firewalld zone persistently. (Exercise 20-1).

Task 19: On rhcsa3, set SELinux type shadow_t on a new file testfile1 in /usr and ensure that the context is not affected by a SELinux relabeling. (Exercises 21-1 and 21-2).

Task 20: Configure password-less ssh access for user60 from rhcsa3 to rhcsa4. (Exercise 19-3).

Reboot the system and validate the configuration.

Appendix D: Sample RHCSA Exam 4

Time Duration: 2.5 hours
Passing Score: 70% (210 out of 300)
Instructions: The RHCSA exam, EX200, is offered electronically on a desktop system running RHEL 8. The exam presents a list of tasks that are to be completed within the stipulated time. Firewall and SELinux need to be considered. All settings performed on the systems must survive system reboots, or you will not be given credits. Access to the Internet, printed material, and electronic devices is prohibited during the exam.

Setup for the Sample Exam 4:
Build two virtual machines with RHEL 8 Server for GUI (Exercises 1-1 and 1-2). Add a 10GB disk for the OS and use the default storage partitioning. Add 1x4GB to VM2. Add a network interface, but do not configure the hostname and network connection. Do not

create a normal user account during installation.

Instructions:

Instruction 01: The following tasks are in addition to the exercises and labs presented in the book. No solutions are furnished, but hints to applicable exercises, chapters, or topics are provided in parentheses for reference.

Instruction 02: Do not browse the Internet or other material for help. This rule does not apply to the kernel download task if included. You can refer to the manual pages, and the documentation under the /usr/share/doc directory.

Instruction 03: This exam should be done in a terminal window using only the command line interface (no GUI).

Instruction 04: You can reboot the VM whenever you want during this exam, but retest the configuration after the reboot for verification.

Instruction 05: Use your knowledge and judgement for any missing configuration in task description.

Tasks:

Task 01: On VM1, set the system hostname to rhcsa5.example.com and alias rhcsa5 using the hostnamectl command. Make sure that the new hostname is reflected in the command prompt. (Exercises 17-1 and 17-5).

Task 02: On rhcsa5, configure a network connection on the primary network device with IP address 192.168.0.245/24, gateway 192.168.0.1, and nameserver 192.168.0.1 using

the nmcli command. You may use different IP assignments based on your lab environment. (Exercise 17-4).

Task 03: On VM2, set the system hostname to rhcsa6.example.com and alias rhcsa6 using a manual method (modify file by hand). Make sure that the new hostname is reflected in the command prompt. (Exercises 7-1 and 17-5).

Task 04: On rhcsa6, configure a network connection on the primary network device with IP address 192.168.0.246/24, gateway 192.168.0.1, and nameserver 192.168.0.1 using a manual method (create/modify files by hand). You may use different IP assignments based on your lab environment. (Exercise 17-3).

Task 05: Run “ping -c2 rhcsa6” on rhcsa5. Run “ping -c2 rhcsa5” on rhcsa6. You should see 0% loss in both outputs. (Exercise 17-5).

Task 06: On rhcsa5 and rhcsa6, attach the RHEL 8 ISO image to the VM and mount it persistently to /mnt/cdrom. Define access to both repositories and confirm. (Chapter 09, topic: Getting Ready to use rpm, and Exercise 10-1).

Task 07: Export /share5 on rhcsa5 and mount it to /share6 persistently on rhcsa6. (Exercises 16-1 and 16-2).

Task 08: Use NFS to export home directories for all users (u1, u2, and u3) on rhcsa6 so that their home directories become available under /home1 when they log on to rhcsa5. Create user accounts as necessary. (Exercises 16-1 and 16-4, and topic: Automounting User Home Directories).

Task 09: On rhcsa5, add HTTP port 8300/tcp to the public zone persistently. (Exercise 21-3).

Task 10: Configure password-less ssh access for u1 from rhcsa5 to rhcsa6. Copy /etc/sysconfig from rhcsa5 to rhcsa6 under /tmp/remote directory securely. (Exercise 19-3, and Chapter 19, topic: Copying Files Remotely Using scp).

Task 11: On rhcsa6, create VDO volume vdo2 on the 4GB disk with logical size 16GB and mounted with XFS structures on /mnt/vdo2. (Exercises 13-6 and 13-7).

Task 12: On rhcsa6, install perl module stream 5.24. (Exercise 10-5).

Task 13: On rhcsa6, flip the value of the Boolean nfs_export_all_rw persistently. (Exercise 21-5).

Task 14: On rhcsa5 and rhcsa6, set the tuning profile to powersave. (Exercise 12-1).

Task 15: On rhcsa5, create file lnfile1 under /tmp and create three hard links called hard1, hard2, and hard3 for it. Identify the inode number associated with all four files. Edit any of the files and observe the metadata for all the files for confirmation. (Exercise 3-2).

Task 16: On rhcsa5, members (user100 and user200) of group100 should be able to collaborate on files under /shared but cannot delete each other's files. (Exercises 4-5 and 4-6).

Task 17: Synchronize the entire /etc directory on rhcsa5 to /tmp/etc on rhcsa6. Use in-transit compression. Capture the output and any errors in the /tmp/etc.transfer file on rhcsa5 during the synchronization process. (Chapter 19, topic: Synchronizing Files Remotely Using rsync, and Chapter 07, topic: Regular Expressions).

Task 18: On rhcsa6, list all files that are part of the “setup” package, and use regular expressions and redirection to send the output lines containing “hosts” to /tmp/setup.pkg. (Exercise 9-1, and Chapter 07, topics: Regular Expressions, and Input, Output, and Error Redirections).

Task 19: On rhcsa5, check the current version of the Linux kernel. Download a higher version from the Red Hat Customer Portal (or from rpmfind.net) and install it. Ensure that the existing kernel and its configuration remain intact. (Exercise 11-2, and Chapter 02, topic: Viewing System Information).

Task 20: On rhcsa5, configure journald to store messages permanently under /var/log/journal and fall back to memory-only option if /var/log/journal directory does not exist or has permission/access issues. (Chapter 12, topic: Preserving Journal Information).

Reboot the system and validate the configuration.

Bibliography

The following websites, forums, and guides were referenced in writing this book:

1. www.virtualbox.org
2. docs.redhat.com/docs/en-US
3. developers.redhat.com
4. www.redhat.com
5. www.opensource.org
6. www.systemd.io
7. www.tldp.org
8. wiki.archlinux.org
9. www.ibm.com
10. www.centos.org
11. www.wikipedia.org
12. www.linux.org
13. www.firewalld.org
14. www.apache.org
15. www.gnome.org
16. www.ietf.org
17. www.isc.org
18. www.netfilter.org
19. www.nftables.org
20. www.nsa.gov/research/selinux
21. www.ntp.org
22. www.chrony.tuxfamily.org
23. www.openssh.org
24. www.pathname.com/fhs

25. Red Hat Certified System Administrator & Engineer for RHEL 7 book by Asghar Ghori
26. Red Hat Certified System Administrator & Engineer for RHEL 6 book by Asghar Ghori

Glossary

. (single dot)	Represents current directory.
.. (double dots)	Represents parent directory of the current directory.
Absolute mode	A method of permission allocation to a file or directory.
Absolute path	A pathname that begins with a /.
Access ACLs	ACL settings applied to files.
Access Control List	A method of allocating file permissions to a specific user or group. See Named user and Named group.
Access mode	See Permission mode.
Access permission	See File permission.
Access right	See File permission.
Access Cache Vector	A special cache area that SELinux uses to store its decisions.
ACL	See Access Control List.
ACL mask	Controls the maximum permissions a named user or named group can have.
Address Resolution Protocol	A protocol used to determine a system’s Ethernet address when its IP address is known.
Address space	Memory location that a process can refer.
Administrator	See Superuser.
Algorithm	A set of well-defined but complex mathematical instructions used for data encryption and decryption.
Alias	A short name to refer to a lengthy command.
Alias substitution	See Alias.
Anaconda	RHEL’s installation program.
Anacron	A service that runs missing cron and at jobs after a system reboot.
Apache	A popular HTTP web server software.

Application module	A complete set of packages to install a software application.
Application stream	A method of making multiple versions of a software application available for installation from the same repository.
AppStream	One of the yum repositories in RHEL 8 that provides a number of add-on software applications along with some core operating system components.
Archive	A file that contains one or more files.
Argument	A value passed to a command or program.
ARP	See Address Resolution Protocol.
ASCII	An acronym for American Standard Code for Information Interchange.
Asymmetric encryption technique	A technique that uses a combination of public/private keys to allow two network entities to communicate privately.
Auditing	System and user activity record and analysis.
Authentication	The process of identifying a user to a system.
AutoFS	The NFS client-side service that automatically mounts and unmounts an NFS share on an as-needed basis.
AutoFS maps	Configuration files to define the directory location to automount a remote share.
Automounter	See AutoFS.
AVC	See Access Vector Cache.
Background process	A process that runs in the background.
Backup	Process of saving data on an alternative media such as a tape or another disk.
BaseOS	One of the yum repositories in RHEL 8 that includes the foundational RHEL components.
Bash shell	A feature-rich default shell available in Red Hat Enterprise Linux.
Berkeley Internet Name Domain	A University of California at Berkeley implementation of DNS for Linux and UNIX platforms. See also Domain Name System.
Binary package	A software package available in a format that yum/dnf/rpm can recognize and install.
BIND	See Berkeley Internet Name Domain.
BIOS	Basic I/O System. Software code that sits in the computer’s non-volatile memory and is executed

	when the system is booted. Also see Firmware.
Block	A collection of bytes of data transmitted as a single unit.
Block device file	A file associated with devices that transfer data randomly in blocks. Common examples are disk, CD, and DVD.
Bluetooth	A wireless technology for communication.
Boolean	The on/off switch to permit or deny an SELinux rule for a service.
Boot	See Boot process.
Bootloader	A small program that loads the operating system in memory.
Boot order	The sequence in which to try devices to boot the system.
Boot process	The process of starting up a system to a usable state.
Bourne Again Shell	See Bash shell.
Bus	Data communication path among devices in a computer system.
Cache	A temporary storage area on the system where frequently accessed information is duplicated for quick future access.
Calling process	See Parent process.
CentOS	Community Enterprise Operating System. A 100% unsponsored rebuild of Red Hat Enterprise Linux OS available for free.
Cgroup	See Control group.
Challenge-response authentication	An authentication method that presents one or more arbitrary challenge questions to the user.
Character special file	A file associated with devices that transfer data serially, one character at a time. Common examples are disk, tape, and mouse.
Child directory	A directory one level below the current directory.
Child process	A sub-process started by a process.
Child shell	A child shell is spawned by the current shell as needed.
Chrony	An implementation of Network Time Protocol for time synchronization on network devices.
CIDR	See Classless Inter-Domain Routing.
Classless Inter-	A technique for better use of IP addresses. It also

Domain Routing	results in smaller and less cluttered routing tables.
Command	An instruction given to the system to perform a task.
Command aliasing	See Alias.
Command history	See History substitution.
Command interpreter	See Shell.
Command completion line	See Tab completion.
Command editing line	Allows editing at the command line.
Command prompt	The OS prompt where you type commands.
Command substitution	A shell feature that allows the assignment of the output of an executed command to a variable.
Compression	The process of compressing data.
Context (SELinux)	A set of SELinux attributes applied to SELinux subjects and objects.
Contiguous blocks data	A series of data blocks.
Control group	A process management technique.
Core	A core is a processor that shares the chip with other cores. Multi-core processor chips are common.
CPU-intensive	A program or application that heavily uses system processors.
Crash	An abnormal system shutdown caused by electrical outage or kernel malfunction, etc.
Crontable	A table of cron jobs scheduled for a user. Commonly abbreviated as crontab.
Current directory	The present working directory.
Current shell	The shell where a program is launched. Compare with Child shell.
DAC (SELinux)	See Discretionary Access Control.
Daemon	A server process that runs in the background and responds to client requests.
Database	A collection of data.
D-bus	Desktop Bus. Another communication method that allows multiple services running in parallel on a system to talk to one another on the same or remote system. Compare with Socket.
De-duplication	A technique to remove redundant data blocks

	from storage to conserve space and improve performance.
De-encapsulation	The reverse of encapsulation. See Encapsulation.
Default	Predefined values or settings that are automatically accepted by commands or programs.
Default ACLs	ACL settings applied to directories.
Default permissions	Permissions assigned to a file and directory at creation.
Defunct process	See Zombie process.
Desktop bus	See D-bus.
Desktop environment	Software such as GNOME that provides graphical environment for users to interact with the system.
Device	A peripheral such as a printer, disk drive, or a CD/DVD device.
Device driver	The software that controls a device.
Device file	See Special file.
DHCP	See Dynamic Host Configuration Protocol.
Directory structure	Inverted tree-like Linux/UNIX directory structure.
Discretionary Access Control	A rich set of traditional access controls in Linux.
Disk-based file system	A file system created on a non-volatile storage device.
Disk partitioning	Creation of partitions on a given storage device so as to access them as distinct, independent logical containers for data storage.
Display manager	Application that is responsible for the presentation of graphical login screen.
Dnf	An upcoming major enhancement to yum.
DNS	See Domain Name System.
DNS name space	See Name space.
Domain	A group of computers configured to use a service such as DNS or NIS.
Domain Name System	The de facto hostname resolution service used on the Internet and corporate networks.
Domain (SELinux)	It ascertains the type of access that a process has.
Domain transitioning	The ability of a process running in one SELinux domain to enter another domain to execute a task in that domain.
Driver	See Device driver.

Dynamic Host Configuration Protocol	A networking service that provides IP assignments to devices.
Encapsulation	The process of forming a packet through the seven OSI layers.
Encryption	A method of scrambling information for privacy. See asymmetric encryption technique and symmetric encryption technique.
Encryption keys	A single secret key or a pair of private/public keys that is used to encrypt and decrypt data for private communication between two network entities.
Environment variable	A variable whose value is inherited by programs in sub-shells.
EOF	Marks the End OF File.
Error redirection	A shell feature that allows forwarding error messages generated during a command execution to an alternative destination (file, printer, etc.).
Ethernet	A family of networking technologies designed for LANs.
Ethernet address	See MAC address.
Export	See Share.
Exporting	The process of making a directory or file system available over the network for sharing.
Extended file system	A type of file system that has been around in Linux for decades and currently has the fourth generation included and widely used in recent Linux distributions.
Extent	The smallest unit of space allocation in LVM. It is always contiguous. See Logical extent and Physical extent.
External command	A command external to the shell.
Fedora	Red Hat sponsored community project for collaborative enhancement of Red Hat Enterprise Linux OS.
Fibre channel	A family of networking technologies designed for storage networking.
File descriptor	A unique, per-process integer value used to refer to an open file.
File globbing	See Filename expansion.
Filename expansion	A series of characters used in matching filenames.

	Also see Metacharacters and Wildcard characters.
File permission	Read, write, execute or no permission assigned to a file or directory at the user, group, or public level.
File system	A grouping of files stored in special data structures.
File Transfer Protocol	A widely used protocol for file exchange.
Filter	A command that performs data transformation on the given input.
Firewall	A software or hardware appliance used for blocking inbound unauthorized access.
Firewalld	A dynamic firewall manager.
Firewalld zone	A method of segregating incoming network traffic.
Firmware	The BIOS or the UEFI code in x86-based systems.
FTP	See File Transfer Protocol.
Full path	See Absolute path.
Gateway	A device that connects two networks.
Gateway address	An IP address that allows a system to communicate with computers on a different network.
GECOS	General Electric Comprehensive Operating System. The comments field in the /etc/passwd file.
GID	See Group ID.
Globally Unique IDentifier	See Universally Unique IDentifier.
Globbing	See Regular expression.
GNOME	GNU Object Model Environment. An intuitive graphical user environment.
GNU	GNU Not Unix. A project initiated to develop a completely free Unix-like operating system.
GPG	Gnu Privacy Guard. An open source implementation of PGP. See PGP.
GPL	General Public License that allows the use of software developed under GNU project to be available for free to the general public.
GPT	See GUID Partition Table.
Graphical User Interface	An interface that allows users to interact with the operating system or application graphically.
Group	A collection of users that requires same

	permissions on files and directories.
Group collaboration	A collection of users from different groups with identical rights on files for the purpose of sharing.
Group ID	A numeric identifier assigned to a group.
GRUB2	Grand Unified Bootloader version 2. The second generation of the GRUB bootloader program that loads the operating system in memory.
GSSAPI-based authentication	An authentication method that provides a standard interface for security mechanisms to be plugged in.
Guest	An operating system instance that runs in a virtual machine.
GUI	See Graphical User Interface.
GUID	See Universally Unique IDentifier.
GUID Table Partition	A small disk partition on a UEFI system that stores disk partition information.
Hard link	A mapping between a filename and its inode number.
Hardware address	See MAC address.
Hardware clock	See Real-Time Clock.
Hashing	See Password hashing.
History expansion	See History substitution.
History substitution	A shell feature that enables the storage of previously executed commands.
Home directory	A directory where a user lands when he logs into the system.
Host-based firewall	A firewall service that runs on the Linux system.
Host-based authentication	An authentication method that allows a single user, a group of users, or all users on the client to be authenticated on the server.
Hostname	A unique name assigned to a network node.
Hostname resolution	See Name resolution.
Host table	A file that maintains IP and hostname mappings.
HTTP	See HyperText Transfer Protocol.
HTTPS	See HyperText Transfer Protocol Secure.
HyperText Transfer Protocol	HyperText Transfer Protocol. Allows access to web pages.
HyperText Protocol Secure Transfer	Secure cousin of HTTP. Allows access to secure web pages.
Hypervisor	Software loaded on a computer to virtualize its

	hardware.
ICMP	See Internet Control Message Protocol.
Index node	An index node number holds a file’s properties including permissions, size and creation/modification time as well as contains a pointer to the data blocks that actually store the file data.
Init	An older method of system initialization. It has been replaced by systemd in newer Linux versions.
Initialization files	See Shell startup files.
Initial permissions	Predefined permission settings that are used to calculate default permissions for new files and directories.
Initial Setup	Program that starts at first system reboot after a system has been installed to customize authentication, firewall, network, time zone and other services.
Inode	See Index node.
Inode table	A table in a file system that keeps a record of inode numbers.
Input redirection	A shell feature that allows supplying input to a command from an alternative source (file, etc.).
Installable package	See Binary package.
Installer program	A program that is launched to install an operating system or application.
Interface card	See Network device.
Internal command	A command built-in to the shell.
Internet	A complex network of computers and routers.
Internet Control Message Protocol	A well-known networking protocol that is primarily used for testing and debugging.
Internet Protocol	A protocol that is responsible for relaying traffic between network entities.
Inter-Process Communication	Allows processes to communicate directly with each other by sharing parts of their virtual memory address space, and then reading and writing data stored in that shared virtual memory.
I/O redirection	A shell feature that allows getting input from a non-default location and sending output and error messages to non-default locations.
IP	See Internet Protocol.

IP address	A unique 32- or 128-bit software address assigned to a network node.
IPC	See Inter-Process Communication.
ISO9660	A file system type used to mount optical devices.
Job	A process started in the background.
Job control	The management of jobs running in the background and foreground.
Job scheduling	Execution of commands, programs, or scripts in future.
Journald	A systemd-based logging service for collecting and storing logging data.
Journaled file system	A file system that uses the journaling mechanism for swift recovery after a system crash.
Journaling	A file system feature that allows it to maintain a journal (log) of its metadata changes to be used to fix any potential anomalies that may arise due to an abnormal system shutdown.
Kerberos	A networking protocol used for user authentication over unsecure networks.
Kernel	Software that controls the entire system including all hardware and software.
Kernel-based Virtual Machine	An open source hypervisor software used for host virtualization.
Kvdo	A kernel module to support the Virtual Data Optimizer feature.
KVM	See Kernel-based Virtual Machine.
Label (storage)	A unique partition identifier that may be used instead of a UUID or device file.
Label (SELinux)	See Context.
Labeling	The process of mapping files with their stored SELinux contexts.
Latency	The time it takes for a data packet to travel between two network entities.
Link	An object that associates a filename to any type of file.
Link count	Number of links that refers to a file.
Link layer address	See MAC address.
Linux	A UNIX-like, open source operating system.
Load balancing	A technique whereby more than one server serve client requests to share the load.

Localhost	A reserved, non-networked hostname assigned to every device. It represents the device itself.
Local variable	A variable whose value is private to the shell (current shell) it is defined in.
Logical extent	A unit of space allocation for logical volumes in LVM.
Logical volume	A logical container in LVM that holds a file system or swap.
Login Manager	See Display manager.
Logging	A process of capturing desired alerts and forwarding them to preconfigured locations.
Logical Volume Manager	A widely used disk partitioning solution.
Login	A process that begins when a user enters a username and password at the login prompt.
Login directory	See Home directory.
Loopback	A reserved IP address assigned to a device for testing and troubleshooting local issues.
LVM	See Logical Volume Manager.
MAC address	A unique 48-bit hardware address of a network interface. Also called physical address, Ethernet address, and hardware address.
MAC (SELinux)	See Mandatory Access Control.
Machine	A computer, system, workstation, desktop, or server.
Major number	A number that points to a device driver.
Mandatory Access Control	A rich set of policies for granular access controls.
Map	See AutoFS map.
Masquerading	A variant of NAT.
Master Boot Record	A small region on the disk that stores disk partition information.
MBR	See Master Boot Record.
Memory-based file system	A kernel-managed virtual file system created in memory at system boot and destroyed at system shutdown.
Memory-intensive	A program or application that heavily uses memory.
Metacharacters	A series of characters that have special meaning to the shell and are used in pattern matching and

	filename globbing. Also see Wildcard characters.
Minor number	A unique number that points to an individual device controlled by a specific device driver.
MLS	See Multi-Level Security.
Module (kernel)	Device drivers used to control hardware devices and software components.
Module (package)	See Application module.
Mounting	Attaching a device (a file system, a CD/DVD) to the directory structure.
Multi-Level Security	One of the two standard SELinux policies that controls access at deeper levels.
Named group	A specific group that receives ACLs.
Named pipe	Allows two unrelated processes running on the same system or on two different systems to communicate with each other and exchange data.
Named user	A specific user that receives ACLs.
Name resolution	A technique to determine IP address by providing hostname.
Name space	A hierarchical organization of DNS domains on the Internet.
NAT	See Network Address Translation.
NDP	See Neighbor Discovery Protocol.
Neighbor Discovery Protocol	A networking protocol that is used to discover IPv6 devices and troubleshoot networking issues.
Netfilter	A framework that provides a set of hooks within the kernel to enable it to intercept and manipulate data packets.
Netmask	See Subnet mask.
Network	Two or more computers joined together to share resources.
Network Address Translation	Allows systems on an internal network to access external networks using a single IP address.
Network classes	Ranges of IP addresses classified into five distinct categories.
Network connection	A connection profile attached to a network device (interface).
Network device	A physical or virtual network interface assigned to a system for network connectivity.
Network File System	A networking protocol that allows Linux systems to share resources (files, directories, and file

	systems) on the network.
Network interface	See Network device.
Network interface card	See Network device.
NetworkManager	A Linux service that is used to configure, administer, and monitor network devices and connections.
Network mask	See Subnet mask.
Network Time Protocol	A networking protocol that is used to synchronize the system clock with a reliable time source.
NIC	See Network device.
NFS	See Network File System.
NFS client	A system that mounts an exported Linux resource.
NFS server	A system that exports (shares) a resource for mounting by an NFS client.
Nftables	A packet classification framework to monitor network traffic.
Niceness	It determines the priority of a process.
Nice value	See Niceness.
Node	A network device with a hostname and IP address.
Node name	A unique name assigned to a node.
Nologin (user) account	A user without the ability to log in to the system.
Normal (user) account	A user account with limited privileges on the system.
NTP	See Network Time Protocol.
NTP client	A system that receives time from a primary or secondary NTP server for its clock adjustments.
NTP peer	Two or more time servers that operate at the same stratum level.
NTP pool	A pool of time servers.
NTP server	See Primary NTP server and Secondary NTP server.
Object (SELinux)	A file, directory, file system, device, network connection, network interface, network socket, network port, etc.
Octal mode	A method for setting permissions on a file or directory using octal numbering system.
Octal numbering system	A 3 digit numbering system that represents values from 0 to 7.

On-demand activation	A systemd way of activating a service when needed.
Open source	Any software whose source code is published and is accessible at no cost to the public under GNU GPL for copy, modification and redistribution.
OpenSSH	A free implementation of secure shell services and utilities.
Orphan process	An alive child process of a terminated parent process.
Output redirection	A shell feature that allows forwarding a command output to an alternative destination (file, printer, etc.).
Owner	A user who has ownership rights on a file, directory, or process.
Owning user	The owner of a file or directory.
Owning group	The group of a file or directory.
Package	A set of necessary files and metadata that makes up a software application.
Package credibility	The authenticity or originality of a package.
Package database	A directory location that stores metadata for installed packages.
Package dependency	Additional required packages for a successful installation or functioning of another package.
Package group	A group of similar applications that can be managed as a single entity.
Package integrity	A state of being complete and error-free.
Package module	See Application module.
Paging	The process of transferring data between memory and swap space.
PAM	See Pluggable Authentication Module.
Parent directory	A directory one level above the current directory.
Parent process	A process with one or more child processes spawned.
Parent process ID	The ID of a process that starts a child process.
Parallelism	A systemd way of starting multiple services concurrently at system boot.
Partition	A partition created on a storage device.
Password aging	A mechanism that provides enhanced control on user passwords.
Password-based	An authentication method that prompts users to

authentication	enter their passwords to be signed in.
Password hashing	A one-way process of converting a legible text string into a random but unique string of characters using one of the several available password hashing algorithms.
Pattern matching	See Regular expression.
Peer	See NTP peer.
Per-user startup files	A set of initialization files that defines custom settings for an individual user upon logging in.
Performance-based	Hands-on implementation.
Performance monitoring	The process of acquiring data from system components for analysis and decision-making purposes.
Permission	Right to read, write, or execute.
Permission class	Access rights on files and directories based on an individual user, a group of users, or everyone else on the system.
Permission type	Read, write, or execute permission bits set on files or directories.
Permission mode	Add, revoke, or assign a permission type to a permission class.
PGP	Pretty Good Privacy. An encryption program to ensure data privacy and secrecy.
Physical address	See MAC address.
Physical extent	A unit of space allocation on physical volumes in LVM.
Physical volume	A disk or a partition logically brought under LVM control.
PID	See Process ID.
Pipe	Sends output of one command as input to the second command.
Pipeline	A command construction with the pipe character used multiple times.
Pluggable Authentication Module	A set of library routines that allows using any authentication service available on a system for user authentication, password modification and user account validation purposes.
Policy (SELinux)	A set of rules enforced system-wide for analysis of security attributes on subjects and objects.
Pool	See Storage pool and Thin pool.
Pool (NTP)	See NTP pool.

Port	A number appended to an IP address. This number could be associated with a well-known service or is randomly generated.
Port forwarding	A method of directing incoming network traffic to an alternative network port.
POST	Power-On-Self-Test that runs at system boot to test hardware. See BIOS, Firmware, and UEFI.
Postfix	A mail transfer application used for sending and receiving mail.
PPID	See Parent process ID.
Primary DNS	A system that acts as the primary provider of DNS zones.
Primary NTP server	A system that gets time from a more reliable source and provides time to secondary servers or clients.
Primary prompt	The symbol where commands and programs are typed for execution.
Priority	See Process priority.
Private key	A randomly generated portion of the private/public key combination that is used to decode the messages encrypted with the paired public key.
Privilege	An extra right to accomplish something.
Process	Any command, program, or daemon that runs on a system.
Process ID	A numeric identifier assigned by kernel to each process spawned.
Process niceness	See Niceness.
Process priority	The value at which a process is running. This value is determined based on the current niceness setting.
Process state	One of multiple states in which a process is held during its lifecycle.
Processor	A CPU. It may contain more than one cores.
Profile (module)	A list of recommended packages that are organized for purpose-built convenient installations.
Prompt	See Primary prompt and Secondary prompt.
Protocol	A common language that communicating nodes understand.
Proxy	A system that acts on behalf of other systems to

	access network services.
Public key	A randomly generated portion of the private/public key combination that is used to encode messages destined for a specific user.
Public key-based authentication	An authentication method that uses a public/private key pair for user authentication.
Public key encryption	See Asymmetric encryption technique.
Quoting	Treats the specified special character as a regular character by disabling their special meaning.
Real-Time Clock	A battery-backed hardware clock on the system.
Recovery	A function that recovers a crashed system to its previous normal state. It may require restoring lost data files.
Redhat Package Manager	A file format used for packaging software for RHEL and its clones.
Red Hat Subscription Management	A comprehensive management service provided by Red Hat to its clients.
Redirection	Getting input from and sending output to non-default destinations.
Regex	See Regular expression.
Regexp	See Regular expression.
Regular expression	A string of characters commonly used for pattern matching and filename globbing.
Relative path	A path to a file relative to the current user location in the file system hierarchy.
Renicing	Changing the niceness of a running process.
Repository	A URL location that provides access to software packages for installation.
Rescue mode	A special boot mode for fixing and recovering an unbootable system.
Resolver	The client-side of DNS.
RHCE	Red Hat Certified Engineer. A designation that may be earned by passing a performance based RHCE exam.
RHCSA	Red Hat Certified System Administrator. A designation that may be earned by passing a performance based RHCSA exam.
RHEL	Red Hat Enterprise Linux.
RHSM	See Red Hat Subscription Management.
Role (SELinux)	It controls who (SELinux subject) is allowed to

	access what (SELinux domains or types).
Root (user) account	See Superuser.
Router	A device that routes data packets from one network to another.
Routing	The process of choosing a path over which to send a data packet.
Root servers	The thirteen most accurate root DNS servers.
RPM	See RedHat Package Manager.
Rsyslog	Essential Linux service for capturing system messages and forwarded them to various destinations for storage.
RTC	See Real-Time Clock.
Runtime	The operational state of an operating system.
SAS	Serial Attached SCSI. See Small Computer System Interface.
SATA	Serial Advanced Technology Attachment. This disk technology is a successor to the PATA drives.
Script	A text program written to perform a series of tasks.
SCSI	See Small Computer System Interface.
Search path	A list of directories where the system looks for the specified command.
Secondary DNS	A system that acts as an alternate provider of DNS zones.
Secondary NTP server	A system that gets time from a primary NTP server and provides time to NTP clients.
Secondary prompt	A prompt indicating that the entered command needs more input.
Secret key encryption	See Symmetric encryption technique.
Secure shell	A set of tools that gives secure access to a system.
Security context	SELinux security attributes set on files, processes, users, ports, etc.
Security Enhanced Linux	An implementation of Mandatory Access Control architecture for enhanced and granular control on files, processes, users, ports, etc.
SELinux	See Security Enhanced Linux.
Server (hardware)	Typically, a larger and more powerful system that offers services to network users.

Server (software)	A process or daemon that runs on the system to serve client requests.
Service (user) account	A user account that is used to control an installed application or service.
Set Group ID	Sets effective group ID.
Set User ID	Sets effective user ID.
Setgid	See Set group ID.
Setuid	See Set user ID.
Shadow password	A mechanism to store passwords and password aging data in a secure file.
Share	A directory or file system shared over the network.
Shared memory	A portion in physical memory created by a process to share it with other processes that communicate with that process.
Sharing	See Exporting.
Shell	The Linux command interpreter that sits between a user and kernel.
Shell program	See Script.
Shell script	See Script.
Shell scripting	Programming in a Linux shell to automate one or a series of tasks.
Shell startup files	A set of files that are used to define the environment for a user upon logging in.
Shell variable	See Local variable.
Signal	A software interrupt sent to a process.
Simple Mail Transfer Protocol	A networking protocol used for email transfer over the Internet.
Single user mode	An operating system state in which the system cannot be accessed over the network.
Skeleton directory	A directory location where user default configuration templates are stored.
Small Computer System Interface	A parallel interface used to connect peripheral devices to the system.
SMTP	See Simple Mail Transfer Protocol.
Snapshot	The state of a system at a certain point in time.
Socket	A communication method that allows a process to talk to another process on the same or remote system.
Soft link	See Symbolic link.

Source package	A software package that can be modified and repackaged for a specific purpose.
Special characters	See Metacharacters.
Special file	A file that points to a specific device.
Special file permissions	Additional access permission bits that may be set on files and directories, where applicable, to give extra rights to (or limit rights for) normal users on executable files and shared directories. Also see Set user ID, Set group ID, and Sticky bit.
SSH	See Secure Shell.
Standard error	A standard location to forward error messages to. Also see Error redirection.
Standard input	A standard location to receive input from. Also see Input redirection.
Standard output	A standard location to forward output to. Also see Output redirection.
Startup files	See Shell startup files.
Stderr	See Standard error.
Stdin	See Standard input.
Stdout	See Standard output.
Sticky bit	Disallows non-owners to delete files located in a directory.
Storage pool	A logical storage space created with one or more disks or partitions.
Stratis	A simplified storage management solution.
Stratum level	The categorization of NTP time sources based on reliability and accuracy.
Stream (module)	Represents a collection of packages that are organized by version.
String	A series of characters.
Subject (SELinux)	A process or user.
Subnet	One of the smaller networks formed using the process of subnetting. See Subnetting.
Subnet mask	Segregates the network bits from the node bits in an IP address.
Subnetting	The process of dividing an IP address into several smaller subnetworks.
Sub-shell	See Child shell. Compare with Current shell.
Substituting users	See Switching users.
Sudo	A method of delegating a portion of superuser

	privileges to normal users.
Superblock	A small portion in a file system that holds the file system’s critical information.
Superuser	A user with full powers on the system.
Swap	Alternative disk or file system location for paging.
Switch	A network device that looks at the MAC address and switches the packet to the correct destination port based on the MAC address.
Switching users	The ability to switch into a different user account provided the target user’s password is known.
Symbolic link	A shortcut that points to a file or directory located somewhere in the directory hierarchy. Compare with hard link.
Symbolic mode	A method of setting permissions on a file using non-decimal values.
Symlink	See Symbolic link.
Symmetric encryption technique	A technique that employs a secret key for private communication between two network entities.
Syslog	See rsyslog.
System	A computer or a logical partition in a computer that runs an operating system.
System Administrator	Person responsible for installing, configuring and managing a RHEL system.
System call	A mechanism that applications use to request service from the kernel.
System console	A display terminal that acts as the system console.
Systemd	System daemon. The default method of system initialization and service management in newer Linux distributions including RHEL 7 and RHEL 8.
System recovery	The process of recovering an unbootable system.
System tuning	A service in RHEL 8 to monitor connected devices and dynamically adjust their parameters for performance improvement.
System-wide startup files	A set of initialization files that defines common settings for all users upon logging in.
Tab completion	A shell feature that allows completing a file or command name by typing a partial name at the command line and then hitting the Tab key twice for additional matching possibilities.
Target	A logical collection of systemd units. All units

	within a target are treated as a single entity.
Targeted policy	An SELinux policy.
TCP	See Transmission Control Protocol.
TCP/IP	Transmission Control Protocol / Internet Protocol. A stacked, standard suite of protocols for computer communication.
Terminal	A window where commands are executed.
Thin pool	A pool of storage that uses the thin provisioning technology to allow the creation of volumes much larger than their actual physical size.
Thin provisioning	An economical technique of storage allocation and utilization.
Thrashing	Excessive amount of paging.
Throughput	The amount of data transferred between two network entities within a specified period of time.
Tilde expansion	See Tilde substitution.
Tilde substitution	A shell feature that uses the tilde character as a shortcut to navigate within the directory tree.
Time source	A reference device that provides time to other devices.
Transmission Control Protocol	A stateful and reliable transport protocol. Compare with UDP.
Tty	Refers to a terminal.
Tuning profile	A set of attributes that can be applied to a system for improving performance of certain components.
Type enforcement	It controls the ability of an SELinux subject to access domains and types.
Udevd	Dynamic device management service.
UDP	See User Datagram Protocol.
UDS	See Universal De-duplication Service.
UEFI	See Unified Extensible Firmware Interface.
UID	See User ID.
Umask	See User mask.
Unified Extensible Firmware Interface	Software code used in computers for pre-boot system management. Also see Firmware.
Universal De-duplication Service	A kernel module to support data de-duplication.
Universally Unique IDentifier	A unique alphanumeric software identifier used to identify an object, such as a disk or disk partition.
Unmounting	Detaching a mounted file system or a CD/DVD

	from the directory structure.
Unit	A systemd object used to organize service startups, socket creation, etc.
Universal Time Coordinated	The reference time used around the world to determine the local time and time zone.
USB	Universal Serial Bus. A bus standard to connect peripheral devices.
User Datagram Protocol	A stateless and unreliable transport protocol. Compare with TCP.
User ID	A numeric identifier assigned to a user.
User mask	A value used in calculating default access rights on new files and directories.
User Private Group	Referred to the GID that matches with the user's UID for safeguarding the user's private data from other users.
UTC	See Universal Time Coordinated.
UUID	See Universally Unique IDentifier.
Variable	A temporary storage of data in memory.
Variable substitution	A shell feature that allows the value of a variable to be used in a command.
VDO	See Virtual Data Optimizer.
VFAT	See Virtual File Allocation Table.
VirtualBox	A type II hypervisor to virtualize an operating system.
VirtualBox Manager	The management interface for VirtualBox.
Virtual console	One of several console screens available for system access.
Virtual Data Optimizer	A feature to conserve disk space, improve data throughput, and save cost.
Virtual Allocation File Table	An MSDOS-compatible file system type.
Virtual file system	See memory-based file system.
Virtual host	An approach to host more than one website on a single system using unique or shared IP addresses.
Virtualization	A technology that allows a single physical computer to run several independent logical computers (called virtual machines) with complete isolation from one another.
Virtual machine	A logical computer running within a virtualized environment.

Volume group	A logical container in LVM that holds physical volumes, logical volumes, file systems, and swap.
Volume-managing file system	A storage management solution that dynamically and transparently manages the underlying logical volume layer for file systems.
Wayland	An innovative, superior, faster networking protocol that has replaced the X Window System protocol in RHEL 8. See X Window System protocol.
Web	A system of interlinked hypertext documents accessed over a network or the Internet via a web browser.
Web server	A system or service that provides web clients access to website pages.
Wildcard characters	A subset of metacharacters used for character matching in strings. See also Metacharacters.
Workload	Any application, database, program, or a combination that runs on the system.
XFS	eXtended File System. A high-performance journaling file system type.
X Window System protocol	A networking protocol that lays the foundation to run graphical applications. See Wayland.
Yum repository	See Repository.
Zero-block elimination	A technique to remove empty (zero-byte) data blocks from storage.
Zombie process	A child process that terminated abnormally and whose parent process still waits for it.
Zone (DNS)	A delegated portion of a DNS name space.
Zone (Firewalld)	A firewalld zone for traffic management.

Index

.

.bash_history file, 157
.bash_profile file, 172
.bashrc file, 172

A

Absolute path, 47
Access Control List, 103
    Access ACLs, 103
    Default ACLs, 108
    Defined, 103
    mask, 105
    Named groups, 103
    Named users, 103
Access permissions (See File permissions)
Address Resolution Protocol (See Networking)
alias command, 161
Alias substitution (See Shell)
anacron command, 191
anacrontab file, 191
Application stream (See Package)
apropos command, 53
Archiving, 62
Archiving tools, 62
ARP (See Networking)
at command, 187
at.allow file, 185
at.deny file, 185
atd service, 185, 187
audit.log file, 476
Auto File System (See AutoFS)
AutoFS, 380

Automounting user home directories, 386
Benefits, 381
Configuration file, 381
Defined, 380
How AutoFS works, 381
Maps, 382
Direct, 382
Indirect, 384
Master, 382
autofs service, 381
autofs.conf file, 381

B

bashrc file, 170
Berkeley Internet Name Domain (See Domain Name System)
bg command, 170
BIND (See Domain Name System)
blkid command, 350
Block device file (See File type)
Boot process, 250
Bootloader phase, 251
Firmware phase, 250
BIOS, 250
UEFI, 250
Initialization phase, 251
Kernel phase, 251
boot.log file, 289
btmp file, 117
bunzip2 command, 63
bzip2 command, 63
bzip2 vs gzip, 64

C

Calling process (See Process)
cat command, 72, 73
cd command, 47
chage command, 136
Character device file (See File type)
chcon command, 469
chgrp command, 146

Child process (See Process)
chmod command, 89
chown command, 146
Chrony (See Network Time Protocol)
chrony.conf file, 418
chronyc command, 419
chronyd service, 419
CIDR (See Networking)
Classless Inter-Domain Routing (See Networking)
clear command, 49
Command aliasing (See Shell)
Command construction, 43
Command history (See Shell)
Command interpreter (See Shell)
Command line completion (See Shell)
Command line editing (See Shell)
Compression (file), 62
Compression tools, 62
config file, 469
Counting words, lines, and characters, 76
cp command, 76
cron log file, 186, 188
cron.allow file, 185
crond service, 185, 188
crontab command, 188
crontab file, 188
Crontable, 189

D

DAC (See SELinux
Daemon (See Process)
date command, 423
D-bus (See Initialization)
De-duplication, 314
Desktop environment, 35
Desktop manager, 34
Device driver (See Kernel)
df command, 350, 354
dig command, 426
Display/Login manager, 34
dnf command, 217
dnf.conf file, 216
DNS (See Domain Name System)

Documentation /usr/share/doc, 56
Domain Name System, 423
    BIND, 423
    Domain, 423
    FQDN, 424
    Managing
        Lookup tools, 426
    Name resolution, 423
    Name space, 423
    Resolver configuration file, 424
    Resolver sources and order, 425
    Roles, 424
        Client, 424
        Primary (Master), 424
        Secondary (Slave), 424
du command, 350, 355

E

e2label command, 350
echo command, 153
Editing files, 67
Encapsulation, 450
env command, 153
Ethernet address (See Networking)
export command, 154
exportfs command, 377

F

fdisk command, 303
fg command, 169
FHS (See Filesystem Hierarchy Standard)
File and directory
    Copying directory, 77
    Copying file, 76
    Creating directory, 73
    Creating file, 72
    Displaying content, 73
    Moving directory, 78
    Moving file, 77
    Removing file, 78
    Renaming directory, 78

Renaming file, 77
file command, 60
File permissions, 88
Calculating default, 92
Classes, 88
Default, 91
Initial, 91
Modes, 89
Modifying, 89
Using octal notation, 89
Using symbolic notation, 89
Special, 93
setgid on directories, 96
setgid on files, 94
setuid (suid), 93
Sticky bit, 98
Types, 88
umask, 91
File system
/, 38
/boot, 39
/dev, 40
/home, 39
/opt, 39
/proc, 40
/run, 41
/sys, 41
/tmp, 40
/usr, 39
/var, 39
Benefits, 346
Categories, 38, 346
Disk-based, 38
Memory-based, 38
Network-based, 38
Defined, 346
Extended file system with journal, 348
fstab file, 353
Managing, 349
Commands, 349
Determining UUID, 351
Labeling, 352
Mount options, 351
Mounting, 350
Mounting automatically, 353

Unmounting, 351
Monitoring, 354
Top-level, 37
Types, 346
Extended, 347
ISO9660, 349
VFAT, 348
XFS, 348
UUID, 351
File type
Block device, 40, 61
Character device, 61
Directory, 61
Raw device, 40
Regular, 60
Symbolic link, 62
Symlink (See Symbolic link)
Filesystem Hierarchy Standard, 37
find command, 100
Finding files, 100
Firewall
Defined, 450
Firewalld, 450
Managing, 453
Service, 452
Service files, 452
Zone, 450
Zone files, 451
Host-based, 450
firewall-cmd command, 454
Firewalld (See Firewall)
firewalld service, 450
FQDN (See Domain Name System)
free command, 367
fstab file, 351, 353, 377
Fully Qualified Domain Name (See Domain Name System)

G

gdisk command, 310
getenforce command, 469, 470
getent command, 428
getfacl command, 104
getsebool command, 469

Getting help, 50
GNU, 2
gpasswd command, 124
GPL, 2
grep command, 166
Group
    Creating, 141
    Deleting, 141
    Identifying, 118
    Modifying, 141
    Owning group, 146
group file, 122
groupadd command, 141
groupdel command, 141
groupmod command, 141
groups command, 118
grub file, 254
grub.cfg file, 253, 254
GRUB2, 251
    Configuration files
        grub, 254
        grub.cfg, 253
        grubenv, 255
    Managing, 251
    Specific targets, 256
grub2-mkconfig command, 254, 255
grubenv file, 255
gshadow file, 124
gunzip command, 63
gzip command, 62
gzip vs bzip2, 64

H

halt command, 284
Hardware address (See Networking)
Hardware clock, 422
head command, 75
history command, 158
History expansion (See Shell)
History substitution (See Shell)
host command, 427
Hostname, 392
hostname command, 392

hostname file, 392
hostnamectl command, 392
hosts file, 409, 426

I

ICMP (See Networking)
id command, 118
ifdown command, 403
ifup command, 403
Index node number (See Inode), 79
info command, 55
Initialization
    systemd
        Control groups, 272
        D-bus, 273
        Listing previous system reboots, 115
        Managing, 276
            Setting default target, 283
            Switching target, 283
            Target, 282
            Unit, 277
            Viewing default target, 283
        Overview, 272
        Parallelism, 272
        Socket, 272
        Target
            Analyzing file, 276
            Defined, 275
            Types, 275
        Unit
            Analyzing file, 274
            Defined, 273
            State, 273
            Types, 274
Inode, 79
Installation
    LAB Setup, 4
    RHEL
        Adding keyboard and languages, 20
        Attaching image, 17
        Configuring install destination, 22
        Configuring network and hostname, 23
        Creating user account, 26

Downloading, 14
Finishing, 27
Launching installer, 17
Logs, 5
Planning, 5
Post-Installation, 28
Selecting software, 21
Selecting source, 20
Setting root password, 26
Virtual consoles, 5
VirtualBox
Creating virtual machine, 10
Downloading, 6
Internet Control Message Protocol (See Networking)
Internet Protocol (See Networking)
IP address (See Networking)
ip command, 394, 399
IPv6 address (See Networking)

J

Job control, 169
Job scheduling
Anacron, 190
Controlling access, 185
crontab file syntax, 189
Daemon
atd, 187
crond, 188
Logging, 186
Overview, 185
Step values, 189
Using at, 187
Using crontab, 188
jobs command, 169
journalctl command, 291
journald.conf file, 291

K

Kernel
Analyzing version, 259

Device driver, 258
Directory structure, 260
Installing, 263
Module, 262
Modules, 258
Overview, 258
Packages, 258
kill command, 183
killall command, 185
kvdo module, 314

L

last command, 115
lastb command, 116
lastlog command, 117
less command, 74
Linking files, 79
Copying vs. linking, 82
Hard link, 80
Link, 79
Symbolic link, 80
Linux
A quick look, 2
Defined, 2
Fedora Project, 3
RHEL history, 3
Linux directory structure, 37
ln command, 81
locate command, 102
Locating files, 102
Log rotation, 286
logger command, 290
Logging in, 29
Logging out, 31
Logical Volume Manager (See Storage)
login.defs file, 125
logrotate command, 287
logrotate.conf file, 287
Loopback address (See Networking)
ls command, 44
lsblk command, 303, 326, 349
lscpu command, 50
lvcreate command, 325

lvdisplay command, 325, 326
lvextend command, 325
LVM (See Storage)
lvreduce command, 325
lvremove command, 325
lvrename command, 325
lvresize command, 325
lvs command, 324, 326

M

MAC address (See Networking)
machine-id file, 294
Major number, 62
man command, 51
mandb command, 53
Manual page headings, 52
Manual page sections, 52
meminfo virtual file, 368
messages log file, 289
Metacharacters (See Shell)
Metadata (file), 79
Minor number, 62
mkdir command, 73
mke2fs command, 350
mkfs command, 350
mkfs.ext3 command, 350
mkfs.ext4 command, 350
mkfs.vfat command, 350
mkfs.xfs command, 350
mkswap command, 368
Module (See Package)
more command, 74
mount command, 350, 377
mounts virtual file, 351
mv command, 77

N

Name resolution (See Domain Name System)
NDP (See Networking)
Neighbor Discovery Protocol (See Networking)
Netfilter, 450

netfilter kernel module, 450
Netmask (See Networking)
Network adapter (See Networking)
Network classes (See Networking)
Network connection (See Networking)
Network connection profile (See Networking)
Network device (See Networking)
Network device naming (See Networking)
Network entity (See Networking)
Network File System
    Benefits, 376
    Configuring, 377
        Exporting, 376
        Mounting, 376
    Defined, 376
    Share, 376
    Versions, 377
Network Interface Card (See Networking)
Network Time Protocol
    Chrony, 416
    Configuration file, 418
    Displaying time, 422
    Modifying time, 422
    Overview, 416
    Roles, 417
        Client, 417
        Peer, 417
        Primary server, 417
        Secondary server, 417
    Stratum levels, 417
    Time server, 418
    Time source
        Internet-based, 417
        Overview, 416
        Radio/Atomic clock, 417
Networking
    Changing hostname, 393
    Configuring
        Commands, 403
    Fundamentals
        ARP, 399
        CIDR notation, 396
    Classes, 394
        Class A, 395
        Class B, 395

Class C, 395
Class D, 395
Class E, 395
Common protocols, 398
Connection profile (anatomy), 401
Consistent naming, 400
Entity, 392
Hostname, 392
Hosts table, 409
ICMP, 398
IP address, 394
IPv4 vs. IPv6, 400
IPv6, 399
Localhost, 394
Loopback address, 394
MAC address, 399
NDP, 399
Network connection, 401
Network connection profile, 401
Network Interface Card, 400
NetworkManager, 405
Node, 392
Port, 397
Protocol, 397
Subnet mask, 396
Subnetting, 396
TCP, 397
UDP, 397
NetworkManager service, 405
network-scripts directory, 401
NFS (See Network File System)
Nftables, 450
NIC (See Networking)
nice command, 182
Nice value (See Process)
Niceness (See Process)
nmcli command, 392, 405
nm-connection-editor command, 405
nmtui command, 405
Node (See Networking)
nologin.txt file, 130
nslookup command, 428
nsswitch.conf file, 425
NTP (See Network Time Protocol)

O

Online help tools, 50
OpenSSH
    Algorithms, 434
    Authentication methods, 433
        Challenge-Response, 433
        GSSAPI-based, 433
        Host-based, 433
        Password-based, 433
        Public/private key, 433
    Copying files, 442
    Defined, 432
    Encryption techniques, 432
        Asymmetric, 432
        Symmetric, 432
    Legacy unsecure commands, 432
    Managing
        Client configuration file, 436
        Server configuration file, 434
    Packages, 434
    Synchronizing files, 445
    Transferring files, 444
    Version, 434

P

Package
    Application Stream, 214
    Binary, 196
    Database, 197
    Dependency, 197
    Groups, 214
    Managing
        dnf/yum, 216
            Configuration file, 216
            Group
                Installing, 230
                Listing, 229
                Removing, 231
                Updating, 230
            Individual package
                Displaying, 223

Installing, 222
Listing, 220
Removing, 224
Searching metadata, 228
Searching provider, 228
Updating, 222
Module
Displaying, 239
Installing, 237
Listing, 236
Removing, 239
Switching stream, 243
Updating, 237
rpm
Extracting, 204
Freshening, 203
Installing, 203
Overwriting, 204
Querying, 200
Removing, 204
Upgrading, 203
Verifying attributes, 206
Verifying signatures, 205
Viewing GPG keys, 206
Metadata, 196
Module, 214
Profile, 215
Stream, 215
Naming convention, 197
Overview, 196
Repository
AppStream, 215
BaseOS, 214
Overview, 215
Source, 196
Parent process (See Process)
parted command, 304, 308
partitions virtual file, 308
passwd command, 138, 140
passwd file, 119
Path, 46
Pathname, 46
Pattern matching, 166
Payload, 450
pgrep command, 180

Physical address (See Networking)
pidof command, 180
pinfo command, 55
ping command, 398, 409
Pipe (See Shell)
pkg-config command, 273
pkill command, 184
Port (See Networking)
poweroff command, 284
printenv command, 153
Process
    Background job, 169
    Calling, 176
    Child, 176
    Daemon, 176
    Foreground job, 169
    Job control, 169
    Listing, 180
    Listing by ownership, 181
    Nice value, 181
    Niceness, 181
    Parent, 176
    Priority, 181
    Process ID, 176
    Renicing, 183
    Signals, 183
    States, 176
    Viewing with ps, 177
    Viewing with top, 179
Process file system (See File system)
Process ID (See Process)
Process priority (See Process)
Process states (See Process)
profile file, 170
profile.d directory, 171
Protocol (See Networking)
protocols file, 397
ps command, 177
Pseudo terminal, 48
pvcreate command, 325
pvdisplay command, 326
pvremove command, 325
pvs command, 323, 326
pwd command, 46

Q

Quoting mechanisms, 165

R

Real-time clock (See Hardware clock)
reboot command, 284
redhat-release file, 156
Redirection
    Defined, 155
    Error, 157
    Input, 156
    Output, 156
Regex (See Pattern matching)
Regexp (See Pattern matching)
Regular expressions (See Pattern matching)
Relative path, 47
renice command, 183
Renicing (See Process)
Repository (See Package)
resize2fs command, 350
resolv.conf file, 401, 424
restorecon command, 469
rm command, 78
rmdir command, 79
rpm command, 198
rpm2cpio command, 204
rsync command, 445
rsyslog.conf file, 285
rsyslogd service, 284
rsyslogd.pid file, 285
RTC (See Hardware clock)

S

scp command, 432, 434, 442
sealert command, 469
secure log file, 146
Secure shell (See OpenSSH)
seinfo command, 465, 469
SELinux

Activation modes, 469
Booleans, 467
Contexts for files, 466
Contexts for ports, 466
Contexts for processes, 465
Contexts for users, 464
Defined, 462
Discretionary Access Control, 462
File operations with SELinux context, 466
Managing, 468
Analyzing alerts, 476
Commands, 468
Operational state, 469
Querying, 470
Mandatory Access Control, 462
Terms
Access, 463
Context, 463
Domain, 464
Domain transitioning, 467
Labeling, 463
Level, 464
Object, 463
Policy, 463
Role, 464
SELinux user, 463
Subject, 463
Type, 464
Type enforcement, 464
semanage command, 465, 466, 469
sesearch command, 469
sestatus command, 469, 470
sestatus.conf file, 471
setenforce command, 469
setfacl command, 104
Setgid bit, 94, 96
setroubleshootd service, 476
setsebool command, 469
Setuid bit, 93
sftp command, 432, 434, 444
shadow file, 120
Shadow mechanism, 120
Shell
Alias substitution, 161
Bash, 152

Child shell, 153
Command history, 157
Command line completion, 159
Command line editing, 159
Command substitution, 155
Current shell, 153
External commands, 152
Features, 152
History expansion, 157
History substitution, 157
Internal commands, 152
Metacharacters, 162
Modifying command prompt, 155
Overview, 152
Pipe, 164
Quoting mechanisms, 165
Tab completion, 159
Tilde substitution, 159
Variable substitution, 155
Variables
Displaying, 153, 154
Environment, 153
Local, 153
Setting, 154
Shell, 153
Unsetting, 154
Wildcards
Asterisk, 162
Exclamation point, 164
Question mark, 163
Square brackets, 163
shutdown command, 284
Signal (See Process)
Socket, 272
ssh command, 432, 434, 441
ssh_config file, 436
ssh-copy-id command, 434
sshd service, 434
sshd_config file, 434
ssh-keygen command, 434
stat command, 60
Sticky bit, 98
Storage
Benefits, 322
LVM, 322

Concept, 322
Logical extent, 324
Logical volume, 324
Managing
Commands, 325
Physical extent, 324
Physical volume, 323
Volume group, 323
Managing
Tools, 304
Using gdisk, 310
Using parted, 308
Partition table
GPT, 303
MBR, 302
UEFI, 302
Stratis
Defined, 335
Dynamic expansion, 336
Managing, 337
Pool, 336
Thin provisioning
Defined, 304
pool, 304
VDO
Compression, 314
De-duplication, 314
Defined, 314
How it works, 314
Managing, 315
Zero-block elimination, 314
Stratis (See Storage)
stratis command, 337
stratisd service, 337
Stream (See Package)
su command, 143
Subnet mask (See Networking)
Subnetting (See Networking)
Substituting users, 143
sudo command, 144
sudoers file, 144
Swap
Commands, 368
Defined, 366
Demand paging, 367

Determing usage, 367
Prioritizing, 368
Thrashing, 367
swapoff command, 368
swapon command, 368
System logging
Configuration file, 285
Journal, 290
journald
Configuration file, 291
Preserving, 293
Viewing, 291
Logging, 289
Logging custom messages, 290
Rotating log files, 286
rsyslogd, 284
systemd-journald daemon, 291
System tuning (See Tuning)
systemctl command, 276
systemd (See Initialization)
systemd-hostnamed service, 393
systemd-journald service, 290

T

tac command, 74
tail command, 75
Tape archive (See Archiving)
tar command, 64
TCP (See Networking)
Thin provisioning (See Storage)
Tilde expansion (See Shell)
Tilde substitution (See Shell)
Time synchronization (See Network Time Protocol)
timedatectl command, 422
top command, 179
touch command, 72
traceroute command, 398
Transmission Control Protocol (See Networking)
tree command, 41
tty command, 48
tune2fs command, 350
tuned service, 294

tuned-adm command, 295
Tuning, 294
    Daemon, 294
    Defined, 294
    Managing, 295
    Profile location, 295
    Profiles, 294
type command, 49

U

udevd service, 40, 400
UDP (See Networking)
UDS (See Universal De-duplication Service)
UDS module, 314
umask command, 91
umount command, 350, 351
unalias command, 162
uname command, 49, 259
Universal De-duplication Service, 314
unset command, 154
UPG (See User Private Group)
uptime command, 48
User
    Authentication file
        group, 122
        gshadow, 124
        passwd, 119
        shadow, 121
    Configuring password aging, 136
    Creating, 127
    Deleting, 128
    Doing as superuser, 144
    Identifying, 118
    Initialization file
        Per-user, 171
        Sourcing sequence, 172
        System-wide, 170
    Locking and unlocking, 140
    login.defs file, 125
    Managing
        Listing logged-in users, 114
        Listing previous failed logins, 116
        Listing previous successful logins, 115

Listing recent logins, 117
Modifying, 127
Nologin account, 130
Owning user, 146
Password aging, 136
Skeleton directory, 127
Substituting, 143
Type
Normal, 118
Service, 118
Superuser, 118
User Private Group, 122
useradd file, 125
User Datagram Protocol (See Networking)
User Private Group, 122
useradd command, 127
useradd file, 125
usermod command, 127
UUID (See File system)

V

vdo command, 315
vdostats command, 315
vgcreate command, 325
vgdisplay command, 324
vgextend command, 325
vgreduce command, 325
vgremove command, 325
vgrename command, 325
vgs command, 323, 326
vi editor (See vim editor)
vim editor, 67
Changing, 70
Copying, 70
Deleting, 68
Inserting, 67
Modes, 67
Command mode, 67
Extended mode, 67
Input mode, 67
Last line mode, 67
Moving, 70
Navigating, 68

Pasting, 70
Replacing, 70
Saving and quitting, 71
Searching, 69
Starting, 67
Undoing and repeating, 69
VirtualBox
Changing boot order, 27
Creating virtual machine, 10
Downloading, 6
Volume-Managing file system, 336

W

w command, 115
Wayland, 34
Desktop environment, 35
Desktop manager, 34
Display/Login manager, 34
wc command, 76
whatis command, 54
whereis command, 49
which command, 49
who command, 114
Wildcard characters (See Wildcards under Shell)
wtmp file, 116

X

xfs_admin command, 350
xfs_growfs command, 350
xfs_info command, 350
xfs_repair command, 348

Z

Zero-block elimination, 314