LINUX

WITH OPERATING SYSTEM CONCEPTS

SECOND EDITION

RICHARD FOX

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Linux with Operating System Concepts

Linux with Operating System Concepts

Second Edition

Richard Fox

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Second Edition published 2022
by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

and by CRC Press
2 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

First edition published by CRC Press 2015

CRC Press is an imprint of Taylor & Francis Group, LLC

Library of Congress Cataloging-in-Publication Data
Names: Fox, Richard, 1964- author.
Title: Linux with operating system concepts / Richard Fox.
Description: Second edition. | Boca Raton : CRC Press, 2022. | Includes bibliographical references and index.
Identifiers: LCCN 2021031731 | ISBN 9781032063454 (paperback) | ISBN 9781032066707 (hardback) | ISBN 9781003203322 (ebook)
Subjects: LCSH: Linux. | Operating systems (Computers)
Classification: LCC QA76.774.L46 F69 2021 | DDC 005.4/32—dc23
LC record available at https://lccn.loc.gov/2021031731

ISBN: 978-1-032-06670-7 (hbk)
ISBN: 978-1-032-06345-4 (pbk)
ISBN: 978-1-003-20332-2 (ebk)

DOI: 10.1201/9781003203322

Typeset in Times
by codeMantra

Access the Support Material: https://www.routledge.com/9781032063454

With all my love to Cheri Klink, Sherre Kozloff and Laura Smith.

Contents

Preface

The Unix/Linux textbook market is full of books for people who are looking to acquire hands-on knowledge of Unix/Linux, whether as a user or a system administrator. There are almost no books that serve as textbooks for an academic class. Why not? There are plenty of college courses that cover or include Unix/Linux. We tend to see conceptual operating system texts which include perhaps a chapter or two on Unix/Linux or Unix/Linux books that cover almost no operating system concepts. Most Unix/Linux books either focus on how to use Unix/Linux or how to administer Unix/Linux.

This book is unique. It explores operating system concepts while introducing Linux-specific content. It is divided roughly into two parts: an introduction to Linux for users and an introduction to Linux for system administrators. It covers Linux and operating system concepts not as a how-to, hands-on guide but as a true textbook, complete with definitions, concepts, chapter reviews and a collection of ancillary material to help support the instructor and student if used in a class.

If you are reading this book, I hope you find it helpful and learn a great deal. My intention is not to cover all things Linux but to explore Linux and offer background for why things are set up the way they are.

HOW TO USE THIS TEXTBOOK

In this edition of the textbook, Chapters 1–6 cover Linux from a user perspective, while Chapters 7–12 cover Linux from an administrator's perspective. Throughout the book are operating system concepts related to topics of the chapter. This textbook is envisioned to be used in a 1- or 2-semester course on Linux. A 2-semester course should be able to cover all 12 chapters, dividing the semesters between user and administrator topics. Such a course could be split between lectures (roughly 50% of the time) and labs. See the lab manual which accompanies this text (available online).

A 1-semester junior or senior-level IT or computer science course should be able to cover a majority of the book. An IT course might cut out some of the operating system concepts and use the lab manual, while a computer science course might include portions of the lab manual and eliminate some of the less needed topics like `sed` (Chapter 5), `systemd` initialization (Chapter 9), open-source software installation (Chapter 11) and system troubleshooting (Chapter 12).

A lower-level 1-semester course that is using this text as an introductory to Linux should cover Chapters 1–4 in detail and then select relevant topics to fill out the course, for instance including some portions of user accounts from Chapter 7, file system topics from Chapter 8, network configuration topics from Chapter 10 and software installation from Chapter 11.

Below is a list of topics found in the chapters that can be considered optional. Omitting some of these topics will not reduce the readability of the remainder of the text.

- Chapter 1: All content outside of Section 1.5, on Linux installation, can be omitted as it mostly explores a history of operating systems and Linux development.
- Chapter 2: Removing Sections 2.7.1 and 2.7.2 should not be an issue; Section 2.6 on `vi` can be skipped if students will not be using `vi`.
- Chapter 3: Sections 3.2 and 3.8 on storage terminology and devices are not essential; Section 3.9 on file compression (although 3.9.4 might be useful) can be skipped; Section 3.7.1 on File Browser searching is only needed if students are not to learn the `find` program.
- Chapter 4: Forms of process management (Section 4.2) and memory and virtual memory (Section 4.7.1) can be skimmed over if the course is an IT course.
- Chapter 5: Sections 5.5 and 5.6 (`sed` and `awk`) can be omitted if these two programs are not going to be used, although some parts of `awk` should still be covered as it is referenced later in the textbook.

- Chapter 6: Section 6.3.5 on the `expr` instruction can be skipped; depending on how much scripting the student is to learn, Sections 6.7–6.9 on arrays, string pattern matching and functions may be omitted.
- Chapter 7: Section 7.5 on PAM and strong passwords may not be needed; Section 7.8 on SELinux could be reduced to just an introduction; Section 7.9 is not necessary.
- Chapter 8: Sections 8.2.1 and 8.2.2 provide background for better understanding inodes but 8.2.2 can be eliminated and 8.2.1 reduced; if the course will not cover topics like partitioning, then Section 8.5 can be skipped.
- Chapter 9: If the course is an IT course, coverage of booting (Section 9.2) may not be warranted; if the course is a computer science course, detailed information about `systemd's` initialization process may not be desired (Section 9.3); some content from Section 9.6 can be eliminated unless the specific service is going to be examined in detail.
- Chapter 10: Depending on the student's background, Section 10.2 may not be needed; Section 10.4.2 is only needed if the course covers DHCP; based on the version of Linux used, either cover Sections 10.6.1–10.6.3 or Section 10.6.4.
- Chapter 11: Sections 11.2 and 11.8 provide background on software but can be skipped; depending on whether the course covers Red Hat or Debian-based Linux, subsections within Sections 11.3–11.5 can be skipped; Section 11.7 can be omitted if the students will not use `gcc`.
- Chapter 12: Coverage of RAID (12.2.2) and encryption (12.2.3) may be skipped if the topics are beyond the scope of the course and similarly the content on operating system issues (12.4.1) may be omitted.
- Chapter 13: This chapter, available online, covers Apache and Squid installation and configuration; it can be skipped entirely if the course will not cover either of these software titles.

NEW TO THIS EDITION

The first edition of this book was written specifically for Red Hat 6 and correspondingly, CentOS 6. At the time of its publication, Red Hat 7 had come out, but Red Hat 7 was viewed as an in-between step as Red Hat was moving from `init` to `systemd`. With both Red Hat 6 and Red Hat 7 reaching the end of their lives, this edition has updated all content to move on to `systemd` Linux distributions, concentrating on Red Hat 8. While preparing this manuscript, CentOS announced that they would not support CentOS 8 beyond 2021 and would instead concentrate on CentOS Stream. At the time of this writing, CentOS 8 and Stream are nearly identical.

So, the first major change between the first edition of this textbook and this new edition is updating to Red Hat 8/CentOS Stream. In an attempt to broaden the appeal of this text, this edition has a good deal of content on Debian/Ubuntu versions of Linux when those versions differ from Red Hat. The information on Debian/Ubuntu is not complete, but attempts have been made to note the differences when space permits.

Chapters 1–7 in the first edition are now Chapters 1–6. Material from Chapter 5 has been moved (`vi` is now in Chapter 2, network software is now in Chapter 10, compression is now covered in Chapter 3, and encryption topics are now in Chapter 12) or removed. Most of these chapters have been rewritten to improve their clarity with improved examples and more tables and figures. Some of the removed content has been uploaded to the textbook's companion website as supplemental reading material.

Chapter 8 from the first edition has been removed. Content on the Linux kernel is now described in Chapters 1, 4 and 9. Installation of Linux has been moved to Chapter 1. Other content, such as virtual memory, has also been moved. With this chapter removed, Chapters 7–12 are primarily what Chapters 9–14 had been. All content in this part of the book has been updated to `systemd`-versions of Linux (primarily Red Hat) and include such new topics as the new top-level directory layout, the

xfs file system (briefly), the NetworkManager service, `firewalld` and `ufw` as new front-ends to `iptables`, the `journald` service and Cockpit (covered briefly). These chapters have also been substantially rewritten and new examples, figures and tables added.

Some material from the former Chapters 9–14 are being moved online via supplemental readings, including for instance the Red Hat 6/init initialization process. The former appendix (binary numbers) is also being removed and made available online.

AVAILABLE ONLINE SUPPLEMENTS

There is a zipped file available for instructors who adopt this book and a zipped file available to everyone. This latter file is available via the textbook's companion website at https://www.routledge.com/9781032063454. If you are an instructor, contact your CRC Press/Taylor & Francis book rep. The contents of these files are listed below.

Instructor-Only Resources

- Instructor's manual complete with answers to chapter review questions
- Testbank

Other Available Material

- PowerPoint notes
- Glossary of terms from chapter reviews (consolidated into one file)
- Select answers to some chapter review questions
- Complete lab manual (assuming CentOS Stream can be adapted for other Linux distributions)
- Supplementary readings (reference to first edition chapters)
  - The fetch-execute cycle and details on the CPU (previously from Chapter 1)
  - Comparing Bash to Csh (previously from Chapters 2 and 7)
  - A brief introduction to emacs (previously from Chapter 5)
  - `vi` and `emacs` cheatsheets
  - System V/Upstart initialization process (previously from Chapter 11)
  - Some example network scripts (previously from Chapter 12)
  - A look at disaster planning and recovery (previously from Chapter 14)
  - Review of binary numbers (previously from the Appendix)
  - Apache/Squid installation (previously Chapter 15, available only online)
  - Perl scripting

Acknowledgments and Contributions

First, I am indebted to Randi Cohen for her encouragement and patience in my writing and completing this text. I would also like to thank Stan Wakefield for connecting me with Randi and CRC Press/Taylor & Francis Group. I would like to thank Olivia Snowden, a recent graduate from Northern Kentucky University (NKU), for so kindly volunteering to proofread the entire manuscript. She caught many errors and typos and helped me improve the book considerably. I would also like to express my thanks to colleague Dr. Wei Hao (NKU) and to Dr. Jim Furstenberg (Ferris State University) for proofreading select chapters and providing both corrections and insightful comments. Dr. Yi Hu (NKU) has maintained a list of errata from the first edition and has given me some great ideas for this new edition. I would also like to thank Micah Sidebottom, a student who, while taking our Linux course in fall 2020, pointed out several inconsistencies in the first edition that I have hopefully corrected. I am also indebted to several former colleagues and associates for their insight into networking (Professor Scot Cunningham) and Linux (Dr. Xiannong Meng, Bucknell University and Professor Peter Bartol, San Diego State University).

I would also like to thank everyone in the open-source community who contribute their time and expertise to better all of our computing lives. Without all of the efforts put into Linux, this book would obviously not exist!

On a personal note, I would like to thank Cheri Klink for all of her love and support.

Author

Richard Fox is a professor of computer science at Northern Kentucky University (NKU). He primarily teaches artificial intelligence, computer architecture, computer systems, concepts of programming languages, object-oriented programming and Unix systems. He has also taught data structures, IT fundamentals, web development and web server administration, among other courses. Dr. Fox, who has been at NKU since 2001, is currently the undergraduate program director of Computer Science and chair of NKU's University Curriculum Committee. Prior to NKU, Dr. Fox taught for 9 years at the University of Texas – Pan American. He has received two Teaching Excellence awards, from the University of Texas – Pan American in 2000 and from NKU in 2012, and an award for Outstanding Service from NKU in 2016.

Dr. Fox received a Ph.D. in Computer and Information Sciences from The Ohio State University in 1992. He also has an M.S. in Computer and Information Sciences from Ohio State (1988) and a B.S. in Computer Science from the University of Missouri Rolla (now Missouri University of Science and Technology) from 1986.

Dr. Fox has published two other books with CRC Press/Taylor & Francis Group: an introduction to information technology text (in its second edition) and a book on Internet infrastructure (coauthored by colleague Dr. Wei Hao). He has also authored or coauthored over 45 peer-reviewed research articles primarily in the area of artificial intelligence.

Richard Fox grew up in St. Louis, Missouri, and now lives in Cincinnati, Ohio. He is a big science fiction fan and progressive rock fan. As you will see in reading this text, his favorite composer is Frank Zappa.

1 Linux
What, Why, Who and When, and How

This chapter's learning objectives are to be able to

- Describe what operating systems are and how and why we use them
- Compare the more popular versions of Linux
- Explain the term open source software and the role the open-source community has had in the development of Linux
- Enumerate reasons why IT personnel should learn Linux
- Identify the roles of the most significant individuals in the development of Linux
- Install Linux
- Use Linux from the GUI to start applications and open terminal windows

1.1 INTRODUCTION

As you are reading this book, you must want to learn about Linux. Why should you learn about Linux? From a user's point of view, Linux offers a different approach than other operating systems. It is an open-source product meaning that you can obtain it for free. Similarly, most software for Linux is open source. With open-source software, you can also enhance the code (if you have the capability of doing so). From a system administrator's perspective, Linux, like Unix, lets you dive deeply into the operating system (OS) and have more control than you can in Windows or MacOS. Learning Linux teaches you not only how to use it but more about operating systems and computers.

The intention of *open-source software* is to make the software available in its source code format. Those who are skilled at coding can then enhance or alter the software or develop new software that uses some portions of the existing software. Although open-source software was originally synonymous with the development of Unix, Linux and related operating systems, the open-source community has produced a number of highly useful applications software. You may have used open-source software yourself. Table 1.1 lists the top ten open-source products as rated by TechRadar as of April 2021. All of the software listed in Table 1.1 can run in Linux, Windows and MacOS unless noted.

One drawback of open-source software is that it does not come with *commercial support*. When we buy software, we are not purchasing the software itself but purchasing the right to use the software. With that purchase almost always comes a guarantee that the software developers will provide timely feedback on issues identified with the software. Bugs will be fixed for us. Security holes will be resolved and patches released quickly. Manuals (whether printed or online) show users how to use the software.

Most commercial software comes with a guarantee of this support; open-source software products are supported by those who developed the titles but that support is not guaranteed. This is an issue that organizations might have when considering the adoption of open-source software. Even with support available, open source might lag behind commercial software with respect to bug and security fixes and improved features.

DOI: 10.1201/9781003203322-1

TABLE 1.1
Some of the Best Open-Source Products

Title	Type of Software	Website
LibreOffice	Productivity software/office suite	libreoffice.org
VLC media player	Plays movies, music, live streams (also available for Android, iOS)	videoland.org/vlc
GIMP	Photo and image editor	gimp.org
Shotcut	Video editor (Windows only).	shotcut.org
Brave	Web browser which supports privacy in browsing (also Android, iOS)	brave.com
Audacity	Audio editor for music and spoken word	audacityteam.org
KeePass	Password generator, credential storage tool (Windows although has been unofficially ported to other platforms)	keepass.info
Thunderbird	Email manager	thunderbird.net
FileZilla	FTP client	filezilla-project.org
Linux	Operating system (can be dual booted from Windows; can be installed in a virtual machine in Windows and MacOS)	Varies by distribution

But while this is a concern, we can also see that the open-source community can operate at a speed greater than that of commercial software companies. One interesting example demonstrating the support from the open-source community comes from the OpenSSL project. OpenSSL is a popular software product used to support secure communication over computer networks, with the capability of creating and utilizing public- and private- key encryption.

A security flaw was discovered in OpenSSL on April 1, 2014. The bug, called *HeartBleed*, would allow clever hackers to break the encryption generated by OpenSSL making websites protected by OpenSSL insecure. Six days after discovery, OpenSSL announced the security flaw and released a patch to solve the problem. One of the biggest security flaws discovered in open source software was patched in *less than a week*. Such agility is found among the open-source community. Many security flaws found in commercial software, including Windows, have taken far longer to fix. Note that while the security flaw was patched, it often took weeks, months or even years for administrators of the various websites that used OpenSSL to apply the patch and thus remove the security flaw.

Linux is free and supported. Is that the only reason to use Linux? Actually, no. It is perhaps the most visible reason but for anyone in IT, it is not the most important. If you are a Windows user, you may have used any number of Windows GUI-based tools to administer your operating system. With these tools, you can create new accounts, change permissions on files and directories, schedule tasks, view events (recorded in logs), modify disk partitions and modify any number of hardware settings, to name only a few of the tasks that these tools support. Nearly every one of these tasks uses a GUI (although some of these operations can be handled through PowerShell commands).

In Linux, every type of administrative operation is handled from the command line. This sounds like a detriment; who wants to type cryptic commands when those operations can be input through a GUI? But, in fact, using the command line gives the user greater flexibility in issuing commands which in turn provides the administrator *greater control* over the operating system. There are some OS settings that cannot be modified in Windows. Just about everything in Linux is modifiable, whether via the command line or, when necessary, by modifying Linux source code; fortunately, we won't have to resort to that step in this textbook!

Greater control through the command line also leads to, in many cases, *easier control*. This seems unlikely in that using a GUI should always be easier. And yet if you have experience with Windows, you know that some settings are hidden behind numerous operations. Click here, select

this, open that, select yet another link, change tabs, fill in this form, select OK and then confirm. Making multiple changes may require repeating the same operations with minor modifications, over and over. We will learn that Linux offers very convenient ways to recall a previous command line instruction, make minor modifications to it and execute the revised version. Alternatively, we can write shell scripts to perform such operations.

If you are a student studying Linux or hold an IT position, another aspect of Linux that you might not gain in Windows or MacOS is learning about operating systems. Windows and MacOS tend to shelter many OS concepts from you. You may learn to control processes using a tool like the Task Manager, learn about disk partitions and file systems, or learn about virtual memory because of heavy disk usage. But in Linux, you may need to delve into these details. You may want to change process priorities, launch some processes in the background and change effective user ID of other processes. You may need to manually mount and unmount partitions. You may need to examine memory usage statistics to see how frequently your computer is swapping between main and virtual memory. Learning Linux from the command line forces you to understand, at least to some extent, what the OS does and how.

In learning Linux, you will expand your knowledge of computers in general. And, well, using Linux is cool (at least that's what many people think!).

In this textbook, we look at Linux from two different perspectives. Early on, we learn Linux as a user. We explore how to enter commands from the command line prompt and navigate through the Linux file space. We learn how to launch and manage processes. We learn some of the powerful tools available in Linux including regular expressions and shell scripting. Over the course of these early chapters, we learn dozens of Linux commands and useful features that make command line entry easier. Midway through the text, we shift focus to learning Linux as a system administrator. In this set of material, we look at creating and managing user accounts, managing files and file systems, the Linux boot and initialization process, controlling services, configuring network access and installing software. Throughout the book, we introduce OS concepts.

To get us started in this chapter, we focus on five questions. *What is Linux*, and more generally what are operating systems? We define operating systems and examine the components that make up operating systems before we turn to Linux specifically and its components. *Why should we use Linux*? We gave a brief answer to this question but delve more into the significance of Linux as an OS of choice. *Who developed Linux and when*? We look at the history of Linux and the important players, starting with some earlier operating systems that led to or factored into the development of Linux. Finally, *how do we use Linux*? Although that is a topic for the entire book, in which chapter we explore how to install several different versions of Linux. We also introduce different forms of Linux interfaces, concentrating on how to open a terminal window for command line input, which we will then use throughout most of the book.

SECTION ACTIVITIES

1. How many operating systems do you have experience with? Count not only desktop/laptop computers but any servers, mainframes, supercomputers, tablets and smartphones. If you have experience with more than one, what similarities do you find between those that you know? What differences?
2. Make a list of your own reasons for learning Linux. How do they compare to the reasons covered in this section?
3. Read about Heartbleed at https://heartbleed.com/. Were you aware of it when the problem was first announced in 2014?

1.2 WHAT IS LINUX?

We start with something more general, what is an *operating system*? A computer's OS is a collection of programs that, as a whole, support our use of the computer through managing hardware resources and running processes. Operating systems usually comprise many different programs. The heart of the OS is a single program called the kernel. The *kernel* is loaded into memory when a computer is booted, and it remains resident in memory until the computer is shut down. The kernel's role is to handle process and resource management, among other tasks. The kernel calls upon other OS components to accomplish some of its tasks. Some of these components are loaded and run as needed. Users may also call upon some of these other OS components, again loaded and run as needed. Although we asked specifically what Linux is, we will concentrate first on operating systems in general. We then shift to Linux specifically later in this section.

1.2.1 Early Operating Systems

The earliest electronic computers, developed in the mid-1940s through the early to the mid-1950s were one-of-a-kind devices, created as much to explore how to build computers as to be useful computational devices. These computers had no operating systems at all. For a "user" to use a computer, that user would write their program code and submit it. Code might have been entered into the computer by making connections of various components through cables, setting switches on the computer's console and pressing the start button. Such programs used no external resources. If a program required a resource like access to input from magnetic tape or punch cards, the instructions to perform such input had to be included in the program itself. Otherwise, the computer would not know how to access the tape drive or punch card reader.

Around 1958, programmers began shifting from low-level machine languages to more sophisticated, high-level languages. Among the first were FORTRAN and COBOL. A program written in one of these languages could not be directly executed. Instead, the program had to be translated from the high-level language into machine language using a separate program called a *compiler*.

The programmer had several distinct steps to run their program. First, they would mount and load the compiler. Next, they would input their FORTRAN or COBOL program into the running compiler. The compiler would execute, outputting the executable version of the program onto magnetic tape. Now, the programmer would unmount the compiler and mount the tape containing their executable program. The programmer would next load and run the executable program. The program's input would likely come from another tape or punch cards. Remember, every one of these steps requires that the programmer implement the steps by additional program code. That's a lot of work to run a program!

To simplify this process, some of the tasks were captured into a program called the *resident monitor*. This program would be loaded into memory after the computer booted. The program would stay in memory until the computer was shut down, thus the word *resident*. The term *monitor* described the role of this program: to monitor a running program's requests for access to the system resources available. The system resources were generally limited to magnetic tape and punch card reader (printers usually were separate devices that would print data from magnetic tape).

The first resident monitor predates FORTRAN and was released in 1955. By the early 1960s, the tasks of the monitor had grown to the point that people were referring to these programs as operating systems. Over the course of decades, operating systems have grown in complexity and size. Table 1.2 examines some of the earliest operating systems/resident monitors.

Today, the OS contains the kernel (what had been the resident monitor) and supporting software including device drivers, utilities, shells, services and servers. Nearly all of our modern computers have and require operating systems. Without an operating system, most users would be unable to use their computer.

TABLE 1.2
Early Resident Monitor/Operating Systems of Note

System	Platform	Notable Features
MIT's Tape Director (1955)	UNIVAC 1103	Mounting and access to files on tape.
General Motors Operating System (unnamed, 1955)	IBM 701	First batch operating system.
GM-NAA I/O (1956) SHARE OS (1959)	IBM 704	Next-generation follow-ups to the General Motors OS, executing scheduled jobs in a batch mode but sharing routines that were common across processes.
Atlas Supervisor (1957)	Atlas Computer	Managed resources including virtual memory.
BESYS (Bell Labs Systems, 1957)	IBM 704 (and later 7090 and 7094)	Batch processing and tape management, use of punch cards, program libraries, core dumping.
IBSYS (1960)	IBM 7090, 7094	Batch processing, job control cards to control operations.
Compatible Time-Sharing System (CTSS, from MIT, 1961)	IBM 7094	Time sharing (multitasking).
Master Control Program MCP, 1961)	Burroughs mainframes	Multiprocessors and virtual memory; the first OS written in a high-level language.
Bolt, Beranek and Newman (BBN) Time-Sharing System (1962)	PDP-1	Time sharing, supported by both the operating system and specialized hardware.
OS/360 (announced 1964, released 1966)	IBM 360 mainframes	Batch processing with multiprogramming and a separate process scheduler.

Computers without an OS are known as *bare machines*. You might find a bare machine if you either construct your own computer hardware and run it without installing an OS or delete the existing operating system. The only reason to use a bare machine is to experiment with hardware and to gauge hardware efficiency irrespective of OS or application software load.

1.2.2 The Operating System Kernel

We identified several types of software that make up the OS in the last subsection. Let's take a closer look at each. The kernel, as already noted, is an expanded version of the resident monitor. It is the kernel that is responsible for most of the important tasks of the OS. We highlight several of these tasks in Table 1.3. The tasks are listed in alphabetical order rather than in order of importance. Process management is likely the most important role of any OS.

There are three different types of kernel: monolithic, microkernel and hybrid. A *monolithic* kernel is one in which the kernel is a single program which operates solely within the computer's *privileged mode* and in its own address (memory) space. Communication between the user side (running applications) and the kernel side is handled through *system calls*. A system call invokes a specific portion of the OS kernel. Upon receiving a system call, the kernel changes from *user mode* to privileged mode. The kernel then ensures that the user program's request is legitimate, meaning that the user program (or user) has an appropriate level of access for the call to be carried out.

Early operating systems used a monolithic kernel. But, as these operating systems governed computers with limited amounts of memory and few system resources, the kernel was not asked to do a great deal and so the kernel was fairly small (in comparison to later OS kernels). As computer capabilities grew, operating systems became larger and monolithic kernels became more and more complex.

In response to the complexity of the monolithic kernel, the *microkernel* was developed starting in the 1980s. The microkernel is based on a smaller kernel that operates within privileged mode and in its own address space. In order to support the expected range of functions, the OS

TABLE 1.3
Roles of the OS Kernel

Role	Meaning	Example
Auditing and accounting	Keep track of users logged in and the resources allocated to them; log events.	Auditing and logging programs.
Device management	Ability to add, remove and interface with peripheral devices.	Mounting disk drives, adding devices via USB ports.
File management	Ability to open, close, create, save, rename, move and copy files and directories.	Windows File Explorer or similar commands available in MS-DOS and PowerShell.
Interprocess communication	Information sharing between running processes.	Shared data in memory or message passing.
Interrupt handling	Dealing with device interrupts and error situations.	I/O interrupts, interprocess interrupts, timer interrupts.
Memory management	Manage memory allocation and deallocation; protect areas of memory from memory violation; move code and data into and out of memory as needed.	Virtual memory (demand paging, demand segmentation), contiguous memory allocation and compaction, memory overlays.
Process management	Start new processes; monitor processes as they run to initiate the handling of requests; detect and initiate the process of handling errors; switch between processes; remove processes from the computer when they terminate.	Single tasking, batch processing, cooperative multitasking, pre-emptive multitasking; multithreading.
Protection	Ensure a process can only access resources available to that process or that process' owner.	Access control lists, user accounts and a login process, user mode versus privileged mode.
Resource management	Grant a process access to a resource (e.g., a file); maintain mutually exclusive access; maintain process liveness.	Synchronization mechanisms, deadlock handling mechanisms.
Scheduling	Determine the order with which processes will run.	First-come first-served, priority, round-robin.
Security	Extend protection across a computer network.	User accounts and a login process, encryption.
User interface	Provide a means for the user to interface with the OS, running applications and hardware.	Graphical-User interface (GUI), command line interface (CLI), menu-driven interface.

is extended by a number of subsystems referred to as *servers* (not to be confused with the use of the word server as used in webserver, file server, print server and so forth). Servers operate based on requests from user applications in the computer's user mode and the user section of memory's address space. Communication between application software and kernel is much like with the monolithic kernel, but the kernel then calls upon servers to handle many of the operations. Thus, the microkernel involves a far greater amount of system calls as the servers are separate from the kernel.

Among the server components are file system servers, network servers, display servers, user interface servers and servers that can directly communicate with device drivers. Remaining in the kernel are the process scheduler, memory manager (including virtual memory) and interprocess communication between OS components. One of the most prominent operating systems to use the microkernel is Mach, Carnegie Mellon University's implementation of Unix.

The *hybrid* kernel compromises between the two extremes where the kernel is kept small but server-like components are added on. Here, the server components run in kernel mode but often in the user's address space. In this way, processes can call upon the servers more easily than with the

microkernel approach and thus bypass some of the time-consuming system calls. The smaller kernel might handle such tasks as interrupt handling and process and thread scheduling. The servers might handle virtual memory, interprocess communication, the user interface, I/O (device drivers) and process management.

Windows NT and later versions of Windows, including Windows 7, 8 and 10, use forms of hybrid kernels. Mac OS X and iOS (used on Apple mobile devices) combine the microkernel of Mach with components from FreeBSD and NetBSD. The resulting kernel for Mac OS and iOS is known as XNU (X is Not Unix) and is another hybrid kernel.

Those who work with either microkernels or hybrid kernels cite that the monolithic kernel is too complicated and large. As the monolithic kernel is essentially one large piece of code, making a minor modification to one portion of the code might have completely unexpected impacts on other portions. Errors may arise that are very hard to identify and locate. This could lead to an unstable OS which yields errors and system crashes for reasons that have little to do with the modified code. There are concerns that the monolithic kernel will be inefficient because of its size.

Given the problems with a monolithic kernel, why would anyone want to produce one? In order to improve on the efficiency and reduce the impact of errors of the monolithic kernel, the modern monolithic kernel includes numerous *modules*. Each module implements one or more of the core kernel's responsibilities. Thus, the kernel can itself be partially modularized. Modules are loaded either at kernel initialization time or on demand. The efficiency of a monolithic kernel's execution is in part based on the number of modules loaded. Loading fewer modules leads to a more efficiently executing kernel. Through modules, the Linux kernel can be kept relatively small.

Figure 1.1 illustrates the difference, at a rudimentary level, between the monolithic kernel and microkernel. On the top, the monolithic kernel is large and handles all kernel operations. System calls are limited to just the applications software invoking the kernel. On the bottom, the microkernel is smaller and simpler, but system calls occur between the kernel and servers as well as between the application software and servers.

As seen in Figure 1.1, a running application invokes a part of the kernel through a system call. The *system call* is a function invocation where the function is not part of the application that invoked it but part of the kernel. In this way, a programmer writing an application has a ready-made interface by which to call upon the kernel.

In Linux, a system call is not directly intercepted by the kernel but instead handled by a *wrapper function*. Wrapper functions are part of one or more Linux libraries such as `glibc`. The role of the wrapper function is to place the arguments of the function call into appropriate hardware

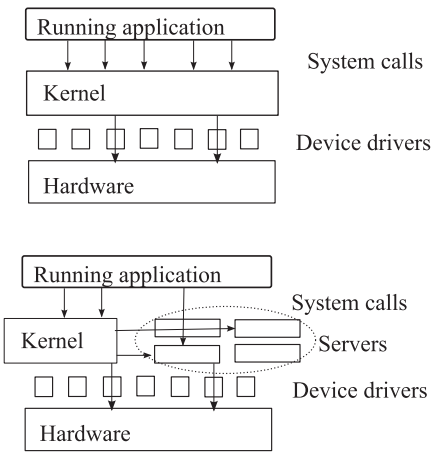

FIGURE 1.1 Monolithic kernel (top) compared to microkernel (bottom).

registers where the kernel expects them and to switch processor mode from user mode to privileged mode. The kernel operates under privileged mode while the user's process operates under user mode. If an error arises during the execution of the system call, it is the wrapper's responsibility to intercept the error from the kernel and modify this into an error number, stored in the special variable `errno`. This variable can then be referenced by the application or the user from the command line. Some wrappers are more complicated and perform various forms of preprocessing and/or postprocessing.

Table 1.4 provides a list of some of the many system calls available in Linux. Note that system calls will vary by both distribution and between versions. The system calls in the table, listed in alphabetical order, are those found in most versions of Linux although the name might vary slightly. The table describes the listed system calls and provides a basic categorization of the type of activity it oversees. Keep in mind that any implementation of Linux will likely have several hundred system calls available.

If you look over the list of system calls you might recognize some of the names by their type of operation (e.g., `open`, `close`, `read` and `write`). Others have names similar to that of Linux instructions like `chmod`, `kill`, `stat` and `mount/umount`. If you are already familiar with Linux, the system call `pipe` should sound familiar.

TABLE 1.4
Some Linux System Calls

Name	Category	Use
`access`	File management	Test user permissions on a file.
`chdir`	File management	Change the current working directory.
`chmod`	File management	Change file permissions.
`close`	File management	Close a file descriptor.
`creat`	File management	Create a file or device.
`exec`, `execv`, `execve` (and others)	Process management	Create a child process.
`exit`	Process management	Terminate current process.
`fork`, `vfork`	Process management	Create a child process, create a child process while blocking the parent process.
`fstat`, `stat`	File management	Get file status.
`getpid`, `getppid`	Process management	Get the ID of a process, process' parent.
`getpriority`, `setpriority`, `nice`	Process management	Get or set the process' priority, change process priority.
`ioctl`	Device management	Send control signal to I/O device.
`lseek`	File management	Move pointer within file to new location.
`kill`	Interprocess communication	Send signal to a process (may result in exiting the process).
`mlock`, `munlock`	Memory management	Lock/unlock page in memory.
`mount`, `umount`	Device management	Mount or unmount file system.
`open`	File management	Open a file/device, assign to a file descriptor.
`pipe`	Interprocess communication	Create interprocess channel.
`poll`	Process management	Cause process to wait for event from file descriptor.
`read`	File management	Read from a file descriptor.
`stat`	File management	Get file status information.
`truncate`	File management	Set a file to a specified length.
`write`	File management	Write to a file descriptor.

Before we leave this topic, we have two additional comments. First, the *file descriptor* is a designator (number) that the kernel assigns to an open file or resource. It is through the designator that the kernel will control access to that open file or resource. Thus, when we want to read data from an open file, the operation to the kernel becomes "read from this file descriptor". Second, the terms `exec` and `fork` are used to express what happens when a process is created. In Linux, all processes are generated, or spawned, by an existing process (with the exception of the first process that runs during system initialization). The system calls of `execve` and `fork`/`vfork` are used so that the current process can generate a new process. We explore the exec and fork system calls in more detail in Chapter 4.

1.2.3 Other Operating System Components

We wrap up our introduction to the OS with a look at other OS components, first with the *device driver*. Referring back to Figure 1.1, you can see that device drivers are positioned between the kernel and the hardware devices. Each driver is a program written specifically to facilitate communication between the kernel and the device. The reason for device drivers is that I/O devices vary dramatically in terms of type of operation, quantity of data transmitted and speed of transmission. It is too much to expect the CPU to be able to communicate directly with the great variety of I/O devices. Instead, the CPU sends generic instructions to the device which the kernel intercepts and converts into actions by calling upon the corresponding driver.

Many device drivers are already preinstalled in most operating systems. The device driver merely needs to be enabled to be used. For less common devices, or devices that were not available when the OS was implemented, the drivers must be installed. These drivers might come from the manufacturer on optical disc provided with the device but more commonly are downloaded over the Internet and installed.

Some OSs, most notably Unix/Linux, Windows and Mac OS, also employ operating system *services*. Services run in the *background* meaning that they run without user interaction. A service runs on demand when it is needed. When unneeded, it uses few system resources. There are a great many services in Linux supporting such tasks as process scheduling, management of hardware and the network, logging and file system handling. We explore services in Chapter 9.

Another component of many OSs is, like the device drivers and services, a collection of programs, referred to as OS *utilities*; also possibly referred to as *tools*. Each utility offers some functionality in improving system performance. One class of utility pertains to files and file systems. There are disk defragmenters to make disk access more efficient, disk recoverers to scavenge deleted files from disk, file system cleaners to remove unused or outdated files, backup and archive utilities and encryption/decryption utilities. Another form of utility is an anti-malware program which not only searches your hard disk for computer viruses but also examines website URLs to see if the website should be avoided as untrustworthy.

Most utilities run at the request of the user. This makes the utility different from other OS components which run as needed or are invoked by application software or requests from other OS components. And unlike the kernel and OS services, utilities run in the foreground i.e., they present the user with an interface. Windows has a great number of utilities, some of which are part of the Windows OS and some are third-party add-ons. Linux has fewer utilities because some of the same tasks are handled by Linux services. Some utilities are not part of Linux at all and must be installed as third-party products.

The last "add-on" component to the OS is tailorable user interface. In Windows and Mac OS, the user can tailor the desktop with files and shortcut icons. In Linux, we more commonly use the shell to personalize our interface. The *shell* is an environment in which the user enters commands. The shell permits definitions and instructions to be recalled later. Definitions include shell scripts and aliases. We explore the Linux shell in detail in Chapter 2.

We noted earlier that the system call wrapper functions, among other tasks, cause the computer to switch modes. The Linux OS has a clear differentiation of roles between user mode and

TABLE 1.5
Uses and Platforms Where We Find Linux

Uses/Platforms	Explanation
Arts	Preferred choice for animation by many movie studios.
Cloud	Used to support cloud computing and other cloud services.
Gaming	Often used in gaming development.
Governments	Used extensively by dozens of governments including the US, China and India.
Mobile device	Android and Raspberry Pi, among others.
Networking	Found in firewall and router hardware devices.
Smart devices	Found in many smart TVs and home automation devices.
Webserver	Linux is used to run more webservers than any other OS.

privileged mode. User mode is applied when running all applications software, Linux services, the GUI, common software libraries and system calls. Privileged mode consists of the OS kernel, including the execution of system calls and device drivers.

1.2.4 So, What Is Linux?

We posed this question to start this section but then dedicated most of the space to describing OSs in general. What is Linux? Succinctly, Linux is a family of OSs, loosely based on Unix, and largely developed by and supported by the open-source community. Linux is an OS that uses a monolithic kernel supported by modules that can be loaded and unloaded. Linux is an OS whose interface with applications software is made through system calls.

Linux is also one of the few portable operating systems available. The term *portable* means that the OS is not written for one or a few forms of hardware but could be run on many different platforms of hardware. Linux is also one of the more efficient OSs. It is small enough to run on nearly any type of computer device but powerful enough to run large-scale applications such as webservers. It is modern enough to use features found in most other OSs while also having its own cutting-edge facilities such as its security handling (SELinux). It is similar enough to Unix so that users of Unix will have no problem learning Linux, and its GUI is relatable enough for a common user to be able to use it. As Linux is available in open source, a programmer can modify how Linux runs by updating and recompiling Linux source code.

There are many uses for Linux. We share some in Table 1.5. Move on to Section 1.3 for more reasons for using Linux.

SECTION ACTIVITIES

1. How important is it for a user to understand the operating system concepts explored in this section? For instance, do you need to know what a kernel is? What services are? Think about your own knowledge of the OS. Does it make it easier for you to use computers?
2. In this section, we saw, briefly, the development of early OSs until 1966. You might wonder about Windows and MacOS. As personal computers were not introduced until the 1970s, it's no surprise we didn't see them included in the list. IBM PC and compatible computers started off using MS-DOS, with Windows being introduced in 1985, 1 year after the first Macintosh came out. Research the development of both MacOS and Windows. You can find a great many websites that describe their individual histories or the history of all operating systems combined.

1.3 WHY USE LINUX?

What makes Linux different from other operating systems? We noted earlier in this chapter that it is open source and provides the user more control. There are other open-source operating systems, albeit not that many, and none have been as successful as Linux (Android by Google is perhaps the most popular alternative but is itself based on a modified version of the Linux kernel). There are other OSs that provide the user with the degree of control as Linux although the one that equals Linux is Unix, on which Linux is based.

It's worth reiterating that open source does not *just* mean free. With Linux, open source comes with an entire community of programmers. These programmers work, usually in large groups, on related projects. Some are involved in enhancing existing Linux versions. Others work on applications that run within one or more versions of Linux. And those same people or others are involved in providing the support that comes with commercial products.

In addition to the open-source nature of Linux, it was initially built to require few resources. We might refer to Linux as a *lightweight* OS compared to that of Mac OS and Windows because of the approach taken in implementing Linux. Linux should boot and initialize faster than other operating systems. A computer running Linux should require less memory and should be able to function with a slower processor. Although these statements are generally true, some versions of Linux require more resources than others.

Another advantage of Linux is portability. The Unix OS was developed to be run on any platform of mainframe computer, and later versions were released for workstations, personal computers and servers. Linux is even more portable. Versions of Linux can run on just about every classification of computer: supercomputer, mainframe, server, personal computer and laptop, tablet, smartphone and embedded devices. This level of portability does not exist with other OSs.

Another reason to choose Linux is that it is less attack-prone than Windows. Hackers more commonly target Windows-based computers when attempting to breach computers, inflict viruses or perform other forms of attacks because there are far more Windows computers than Linux computers (excluding servers and supercomputers). You might think that Linux would be open to attacks of all kinds because its source code is readily available. In fact, the open-source community takes great efforts to ensure a lack of security flaws in the OS before a Linux version is released. That is not to say that they are 100% successful. But because of the efforts taken combined with the lower incidents of attack, some companies and even whole governments are moving to Linux as the preferred OS because it is, at least currently, more secure.

Linux has become the OS of choice for many computer programmers and system administrators. It is also an OS of choice for many computer users, hobbyists and companies. But in spite of its increased popularity, the number of desktop/laptop computers running Linux remains in the minority when compared to Windows and Mac OS. It is estimated that no more than 1.93% of the world's desktop computers are currently running Linux. Figure 1.2 highlights some of the more interesting statistics about Linux usage, as of spring 2021.

SECTION ACTIVITIES

1. What open-source software have you used? Make a list. It's probably more than ten titles!
2. The statistics in Figure 1.2 provide an interesting dichotomy: Linux is the operating system of choice for supercomputers and most servers but few desktops. Why do you suppose this is the case? Write down three reasons. Now research the topic and see if your reasons are those cited by others.

*Linux runs on **all** of the world's fastest supercomputers.*

***Every** major space program in the world uses Linux-based computers to run their software.*

***96.3%** of the world's top 1 million servers run on Linux computers (less than 2% use Windows).*

*Of the world's top 25 most visited websites, 23 (**92%**) run on Linux computers.*

***95%** of the world's top 1 million Internet domains run on Linux computers.*

***90%** of all public cloud facilities run on Linux computers, and all of Amazon Web Services runs on Linux computers.*

***90%** of all Hollywood special effects are created on Linux computers; special effects found in such movies as Titanic and Lord of the Rings were generated on Linux computers.*

***85%** of all smartphones are Linux-based.*

***83.1%** of all software developers prefer to work in Linux while **54.1%** actual use Linux (as of 2019).*

Countries whose governments prefer to use Linux computers include Austria, India, Cuba, Russia, Turkey, French and Dutch police use Linux, and both the White House and the US military have migrated their computers to Linux

Linux is most popular in the states of California and Utah.

FIGURE 1.2 Interesting statistics about Linux usage.

1.4 WHO DEVELOPED LINUX AND WHEN?

We can start the history of Linux with one person and one event: Linus Torvalds, a Finnish computer science student, who in 1991 was dissatisfied with the operating system that came with his operating system textbook. He set about creating his own OS which he eventually named Linux. But rather than starting with Torvalds in 1991, we start 27 years earlier.

1.4.1 The Birth and Development of Unix

In 1964, a group of software engineers from AT&T Bell Labs, General Electric and MIT began developing a new and novel operating system called MULTICS (Multiplexed Information and Computer Service). MULTICS was a multiuser, time-sharing (multitasking) OS that pioneered a number of innovative features. While the development of MULTICS seemed promising, AT&T withdrew from the project in 1969, shortly before its release.

Ken Thompson and Dennis Ritchie were two of the AT&T members involved in the early development of MULTICS. They did not want to see their efforts end at that time. AT&T was using a batch processing OS that they felt was unsatisfactory. They worked on their own version of a MULTICS-like OS with help from other AT&T employees, including Rudd Canaday. The result was a new OS, written for the PDP-7 mainframe and written in that computer's assembly language. They called it UNICS, short for Un-multiplexed Information and Computing Service, likely a joke name spoofing the name MULTICS.

Further development targeted the PDP-11 computer but rather than reimplement the OS in PDP-11 assembly language, Thompson and Ritchie wanted to implement their OS in a neutral

language so that the OS could be ported more easily to other computers. Ritchie, along with another AT&T employee, Brian Kernighan, created a new programming language called C. It was in C that UNICS was rewritten and eventually released in 1973. During this time, the OS was renamed from UNICS to Unix. Other employees contributed significantly to Unix as it was developed.

A look at Unix features shows that it was at least partially derived out of the experience that Thompson and Ritchie gained in their work on MULTICS. Table 1.6 shows some of the features either introduced in MULTICS and used or enhanced in Unix, or introduced in Unix. Compare these features to the brief descriptions of other early 1960s OS from Table 1.2, and you will see that first MULTICS and then Unix popularized many features that had yet to be developed.

In spite of Unix being referred to as platform-independent, the 1973 version still contained some code specific to the PDP-11. It was not until 1978 that Unix was successfully ported to another platform. When distributed by AT&T, Unix was provided in source code format (for a small fee). Although AT&T released many versions of Unix, they were all referred to as Research Unix. But because AT&T offered no support in the versions they distributed, it was up to other organizations to modify the Unix source code as needed to run on different platforms, fix errors and add desired features. This led to a splintering of Unix into many versions.

One such version was provided to the University of California Berkeley's computer science department. At Berkeley, programmers developed their own OS, using some of the Unix code as components. In 1975, Ken Thompson became a visiting professor at UC Berkeley. Under his

TABLE 1.6
Unix Features (From Richard Fox, 2021, *Information Technology: An Introduction for Today's Digital World,* 2nd edition)

Feature	Explanation	Comments
Command line interpreter	Beyond a CLI, Unix's CLI provides flexibility with operations to redirect input and output, define constructs like aliases, and the ability to write shell scripts.	Popularized from Multics
Commands as programs	Rather than having the kernel handle all operations, many basic commands are provided as their own programs such as file system commands, text editing and processing commands, and process management commands.	First
Hierarchical file system	Directories can have their own subdirectories and administrators to better organize files.	Popularized from Multics
Interprocess communication	Allows synchronized access to shared data among multitasking processes.	
Multitasking/multiuser OS	Runs multiple processes of multiple logged-in users at a time.	
Portability	Unix could be installed on a large variety of computer types.	First
Regular expressions	Ability to search files for strings through command-line programs like `grep`, `awk` and `sed` and the programming language `Perl`.	Popularized from Multics
Shells	The CLI was one component of a shell; shells introduced convenient shortcut features to assist the user.	First
System calls	Written in C, Unix made many kernel operations available as C functions known as system calls, which could be invoked by user programs.	First
Top-level directory structure	Separate directories for OS components and user files simplify the process of searching for a file.	First
Treating devices as files	Nearly everything in Unix is either a process or a "file"; this allows file commands to be executed on devices either from the command line or from a script.	First

guidance, a new version of Unix was produced. This became known as the Berkeley Standard Distribution (BSD) Unix. Unlike AT&T's Unix, which was proprietary, BSD Unix was freely distributed under the BSD License (which imposed only minimal restrictions on the use and distribution of this version of Unix). BSD versions 4.2 and 4.3 became leading versions of Unix, distributed to other organizations. BSD led to other versions such as SunOS, Darwin (which would become the basis for the Mac OS), FreeBSD and NetBSD. Other versions splintered from Unix to become IBM's AIX, HP's HP-UX, Microsoft's Xenix and Sun's Solaris. By 1990, there were dozens of versions of Unix, many of which were proprietary.

1.4.2 GNU

In the early 1980s, Richard Stallman, a researcher at MIT's artificial intelligence laboratory, wanted to create his own Unix-like operating system. He wanted to make this OS available in source code format for free and with no licensing restrictions so that other programmers could modify the OS as they desired just as long as any modifications would similarly be made available in source code for the community to use. He called upon many members of the Unix user and programming community for help. He dubbed the project GNU for GNU Not Unix, a recursive definition (recursion is a programming tool commonly used in artificial intelligence).

Stallman's group began work on the GNU OS at the same time BSD was becoming popular. But while BSD became successful, a GNU kernel was never completed. Stallman succeeded, though, in his version of a free software movement. His vision was that software should be free. He would explain free as in "free speech" not "free beer". He opposed proprietary software and saw the software field as one where all programmers could or should contribute to the development of free software. Software, he felt, were based on ideas and ideas could not be owned.

Stallman created the GNU General Public License (GNU GPL). The GPL requires that software published under the GPL must be free for anyone to use for any purpose: free to be studied, free to be changed, free to be redistributed and free to be improved. The proviso is that anything created by GPL software would also be published under the GPL so that further distribution of such software would also be available as source code allowing others the same freedoms.

GPL version 1 was released in February 1989. Its main points are that any software distributed under the GPL can be copied and distributed in the source code format (as long as it is distributed exactly as provided) and distributed in any medium, and as long as the redistribution maintains the exact same notices as the original. The same stipulations apply to modified code: such code must contain the original notices as well as notices that this version has been modified, including the files changed and the dates of those changes, and that the new distribution be carried under the GPL. The GPL also permits distribution of the code in its compiled, executable form (known as object code). There are ten clauses in GPLv1 including two that clearly indicate that the software comes with no warranty because it is free.

GPLv2 was released in June 1991 adding a restriction that works covered by the GPL can only be redistributed if they can satisfy all of the license's obligations. The inclusion of this clause is to discourage any form of patent infringement lawsuit being claimed over a GPL-distributed product. GPLv3 was released in June 2007, which added more detail and language to protect GPL-covered items (which at this point went beyond software).

Although we will not examine the GPL itself, Figure 1.3 shows the first version's Preamble, which is worth reading. It states, right from the start, that the GPL is there to express our freedom rather than the restrictions found in most software licenses. It is for this reason that Stallman called the GPL a *copyleft* instead of a copyright. The entire GPLv1 license, along with all newer versions, is available at https://www.gnu.org/licenses/ with version 1 at https://www.gnu.org/licenses/old-licenses/gpl-1.0.en.html.

```
                            Preamble

  The license agreements of most software companies try to keep users
at the mercy of those companies.  By contrast, our General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  The
General Public License applies to the Free Software Foundation's
software and to any other program whose authors commit to using it.
You can use it for your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Specifically, the General Public License is designed to make
sure that you have the freedom to give away or sell copies of free
software, that you receive source code or can get it if you want it,
that you can change the software or use pieces of it in new free
programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of a such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must tell them their rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  The precise terms and conditions for copying, distribution and
modification follow.
```

FIGURE 1.3 The GPLv1 preamble.

1.4.3 Enter Linus Torvalds

In 1991, Linus Torvalds, a computer science student at the University of Helsinki, was unhappy with the toy OS that came with his operating system textbook. Torvalds began to develop his own small OS on his Intel 386-based computer, written in Intel 80386 assembly language. He developed only a few components: a small multithreading kernel, a `bash` interpreter and the `gcc` compiler. He posted on the USENET newsgroup, `comp.os.minix`, to attract other programmers who wanted to help develop an OS that was different from and better than MINIX. He posted the source code for his fledgling OS on an FTP server. Over the next few months, members of the GNU community contributed GNU components including the facility to compile code using `make` as well as `compress` and `sed`, among other programs.

As the OS took shape, he published it under a version of the GPL (GPLv2). Over the next several months, the community of programmers working on Linux grew. A new USENET newsgroup was established for the community, `alt.os.linux` (which would be renamed `comp.os.linux` a couple of months later). Enhancements to Linux during this time included a number of device drivers, a more fleshed-out kernel, the adoption of the POSIX standard and an implementation of the X Window System. By 1994, Linux had grown to over 175,000 lines of code, written in C. This version was distributed as Linux version 1.0.0. Aside from versions released for Intel processors, Linux was successfully ported to the DEC Alpha, Sun SPARC and MIPS processors.

In 1992, the first well-known splintering of Linux took place when Canadian software engineer Peter MacDonald developed Softlanding Linux System (SLS). Whereas Torvald's Linux was a kernel, device drivers and full utilities like `gcc`, SLS included a full windowing system (based on X Windows) and a number of applications software. Computer science student Patrick Volkerding (at Moorhead State University in Minnesota) downloaded SLS and modified it to fix some bugs. He distributed his version to fellow students who urged him to publish his version. His version became known as Slackware.

A German software company, Software und System Entwicklung (Software and System Development, or SUSE for short) adopted Slackware as a platform, translating it into German and enhancing it. This led to three different versions from the same branch of the Linux "tree". All three versions continued to be updated although SLS was largely ignored in favor of Slackware. But it was SUSE that found the most popularity in the end and has had several spinoff versions under various names including SUSE, SuSE, SuSELinux and openSUSE. In comparing SUSE to other versions of Linux, we find it comes closest to Red Hat Linux, and as SUSE/openSUSE is of limited popularity, we will not examine it any further.

The SLS/Slackware/SUSE version of Linux leads us to the term *distribution*, or *distro* for short. More and more distributions took place in the 1990s and into the 2000s resulting in a large number of Linux versions. There are far too many distros to track them all here. The remainder of this subsection looks at some of the more significant developments of the distros.

In 1994, software developer Marc Ewing developed his own version of Linux that he named Red Hat Linux. Released late in 1994, his business was bought by entrepreneur Bob Young. The new company was called Red Hat Software and largely revolved around a commercial version of Linux (both the OS and application software). Unlike the versions of Linux being developed by the open-source community, the intention of Red Hat was to include support so that companies could trust that bugs would be fixed quickly. Red Hat Software was later renamed Red Hat, Inc., which was purchased by IBM in 2018.

Two spinoffs of Red Hat are supported by Red Hat, Inc., Fedora and CentOS. Both of these versions are open-source versions of what is now called Red Hat Enterprise Linux (RHEL). There are a few differences between these three distributions but mostly they are built on the same kernel with most of the same services, applications and features. Similarities and differences between the three distributions are described in Table 1.7. Note that the most recent version is of spring 2021.

Debian was first announced in 1993 with the intent to be the first release of a Linux outside of Torvald's. The project was headed by programmer Ian Murdock, and the name was a combination of his girlfriend (Debra) and his first names. Although the first release of Debian (version 0.01) was in 1993, the first major release did not take place until 1996, thus it trailed both SLS/Slackware and

TABLE 1.7
Comparisons between RHEL, Fedora and CentOS

	RHEL	Fedora	CentOS
Initial release	2000	2003	2004
License type	Commercial	GPL	GPL
Maintained by	Red Hat, Inc/IBM	Open-source community	Open-source community
Processor targets	x86, ARM, IBM Z, IBM power systems	x86, ARM, aarch64	x86, ARM64, Power8
Computer types	Servers, mainframes, workstations, supercomputers	Servers, desktops, cloud	Servers, desktops, workstations, supercomputers
User interface	GNOME	GNOME	GNOME
Release targets	2–4 minor releases per year, major releases every 5 or so years	Every 6 months	Same as RHEL
Most recent version	RHEL version 8 (Oct 2020)	Version 33 (Oct 2020)	Stream (Dec 2020)

Red Hat. Murdock handed off control of Debian to Bruce Perens in 1996 and then leadership was established by elections every year (starting in 1999).

Unlike Red Hat which had both commercial and open-source versions, Debian was intended to be open source from day one. The Debian development community took their time with each version, planning to release a new version only every 2 years. Three different versions of Debian are available at any time: the current stable version, a testing version of the intended next release and an unstable version which includes libraries and packages that may not install correctly or at all (that is, these items are untested).

The deliberative and slow nature of new releases under Debian was a point of contention with some developers and so Ubuntu spun off from Debian with new releases targeted for every 6 months. The first version of Ubuntu came out in 2004. Ubuntu attempts to be a more secure version of Linux and as such, pioneered the idea that rather than users having access to the system administrator account (*root*), the first user account would automatically receive access to administrator commands through the `sudo` command. We explore this later in the text. Other Linux distributions later adopted this practice.

Two other notable Debian descendants are Knoppix and Linux Mint. Knoppix is noteworthy as a form of Linux that can be booted and run entirely off of an optical disc. This feature, known as a *live boot*, allows the user to use Linux without installing it on their computer. The disadvantage is that you start with a new version of the OS every time you launch it. Any previously created accounts or changes made to the OS or environment are lost. Files must be saved externally, for instance via a USB-based drive. Linux Mint is based on a Ubuntu-derivative called Kubuntu, which was first released in 2006. As Linux Mint was further developed, it became more based on Debian than Ubuntu and is now called Linux Mint Debian Edition (LMDE). Two differences between Linux Mint and Ubuntu are that Linux Mint supposedly uses less memory when running and it has more preinstalled applications than Ubuntu.

In reading through the distributions listed above you might question why there are so many. What are their differences? All of the spinoff distributions are based around the same Linux kernel. Each version's kernel shares many of the same modules with the other versions. The versions share many of the same C libraries, system calls, shells and X Windows interface (although the specific look of the desktop will differ). Many of the same applications run on all of the Linux distributions.

There are perhaps four significant differences between these main distributions. The most significant difference in terms of usage is the package manager program used to perform software installation and upgrades. A second difference is the quantity of available applications software as some distributions have fewer titles available or support fewer third-party applications. There is also a philosophical difference behind the developer's plans for major version releases. Finally, there is a different philosophy taken regarding whether the root password is made available.

There are two primary package managers used in today's Linux distributions: `dnf` (or the slightly older `yum`) and `apt`. These are built on top of other package manager programs: `rpm` (Red Hat Package Manager which `dnf`/`yum` uses) and `dpkg` (Debian Package Manager which `apt` calls upon). Amazingly, other than the specific desktop design, this is the single most visible difference between the various distributions.

In terms of available software, we find most distributions come with a `Firefox` web browser (usually built in) and can run the `emacs` editor (not built in). They also all have the ability to run an office suite (built in or not), usually `LibreOffice`. However, some have far more applications than others. For instance, Ubuntu is a popular choice for gaming enthusiasts.

The development philosophy is less visible but no less impactful. The Debian community takes a slow but thorough approach when creating and vetting a new version. The target of every 2 years seems extreme to some Linux users and so they favor the more rapid development and deployment of Ubuntu. With Red Hat distributions, Red Hat, Inc. drives most of the development but as new versions are made available, members of the Red Hat community take the source code and use it to develop a new release of CentOS. So, in fact, CentOS will have mostly the same code and features as RHEL.

TABLE 1.8
Comparing Linux Distributions

	CentOS	Debian	Fedora	Linux Mint	RHEL	Ubuntu
Initial release	2004	1996	2003	2006	1995	2004
Parent (if any)	Red Hat	N/A	Red Hat	Ubuntu	N/A	Debian
Package manager	rpm, dnf/yum	apt, dpkg	rpm, dnf/yum	apt, dpkg	rpm, dnf/yum	apt, dpkg
Approximate amount of software packages	6000–7000	90,000	65,000	30,000	10,000–12,000	90,000+
Target release frequency	Coupled with RHEL	Every 2–3 years	Yearly	Every 2–3 years	Major releases every 5–10 years	Every 6 months
Degree of open source	Completely	Completely	Completely	Mostly	None	Mostly
Most recent version	8 and Stream	10.8	33	20	8	20.04
Most recent release date	2020	2021	2020	2021	2020	2020

In previous releases of RHEL, a related version of CentOS was released shortly afterward. Starting in 2020 however, a new approach is being taken. CentOS is now releasing versions under the name Stream. The idea is that rather than waiting for a new, completed version of RHEL, CentOS will release versions earlier that contain features being proposed for the next RHEL release.

Finally, there is a distinction between RHEL, and the rest of the Linux distributions described here in that RHEL is not a free OS. Instead, a commercial purchase must be made in order to receive RHEL support. The other versions of Linux are free and can be found in source code format. Within the different distributions, some software may be proprietary. Early versions of Linux Mint had proprietary software (which has supposedly been removed).

We review the most significant differences between these most notable distributions in Table 1.8. In the table, we cite which of the original distributions it evolved from, the choice of package manager, the quantity of available software (estimated from various website reports) and the development philosophy. We also note the most recent version and its distribution date. The Linux distributions are listed in alphabetical order (left-to-right) rather than by year of initial or most recent release.

There is no official count of the total number of Linux distributions available or in use today, but the estimate is between 600 and 1000. Some of these distributions vary more dramatically than what was explained above. Nearly all of the distributions are available for free (RHEL being one of the notable exceptions), and these can come in either source code or as an executable installation program (usually as an ISO image). The history is so complicated that mapping the distros requires a very large chart (see the Linux distro tree at https://5thlobe.com/reference/gnulinux-distro-chart/).

We should point out that Linux is not the only open-source OS. Three notable other open-source OSs, all of which are Unix-like, are FreeBSD, Dragonfly BSD and GhostBSD. Another open-source OS is XNU (which stands for X is Not Unix), which is the basis for the iOS operating systems found on such devices as the Apple iPad and Apple TV. ReactOS was developed in 1996 as a clone for Microsoft Windows 95 and has the look and feel of more recent Windows OSs. And then there's FreeDOS which is primarily used to run old MS-DOS games and other legacy software. There are others, but none of these operating systems has achieved anything close to the popularity or name recognition of Linux.

1.4.4 The Open-Source Community

The earliest open-source projects were the further development of Unix, GNU and then Linux. The first application software that the open-source community contributed to was the Netscape web browser, which the original developers offered to the community for input. It was at this point that the term *open source* was first used.

As work continued on both Linux and Netscape, the open-source software community fell into a dispute. Torvalds did not require that modified versions of software be made freely available. He felt that if a person modified a piece of open-source code, the modified code could continue to be freely distributed as source code, or it could be freely distributed but as executable code, or it could even be sold for profit. But some members of the open-source community eschewed commercial software and felt that anything developed out of the open-source community must remain open source and freely available.

This rift caused the community to splinter into two groups: the Open-Source Initiative (OSI) and the Free Software Foundation (FSF). OSI was founded in 1998 by members of the open-source community developing versions of Linux including Bruce Perens from Debian. FSF was founded by Richard Stallman in 1985. The difference between the two groups is that OSI is willing to accept copyrights on some software restricting freedoms that were established under the GPL while the FSF generally feels that anything created from a GPL-produced product must also be made available in open source under the GPL. Although this rift has driven some projects to splinter, it has not impacted the productivity of the open-source community and the number of titles being produced.

The shift in perspective from commercial software development to open source not only has impacted software developers' perspective of open source but the corporate world. Early on, open-source software was primarily used by hobbyist and people in the open-source community. Open-source software was largely shunned by companies. There was a stigma attached to open source products that they would have errors that could potentially harm an organization because the software would not work when it was needed. This could result in a loss of productivity and revenue and cause the organization to receive a bad reputation. Companies preferred the guarantee that proprietary software would be fully supported such that errors would be fixed in a timely manner.

Today, open-source software competes regularly with proprietary software in part because the open-source community has been found to be as or more responsive than many of the companies producing proprietary software. A comparison between open-source software and proprietary software products often shows that open-source software has an equal level of security, is of equal (or even higher) quality, has a greater degree of interoperability (available on multiple platforms), and provides a reasonable amount of support. The support might be available for free, or for a fee but the fee would in most cases still be less than the expense of the proprietary software and its support.

We also find today that the open-source community exists well beyond individual developers wanting to make contributions to Linux or other projects. For instance, Microsoft, who would have opposed the open-source initiative years earlier, is now a steady contributor with such products as .net development tools, the Visual Studio Code editor, PowerShell, and tools for building and training machine learning systems. Their open-source contributions are not limited to Windows; they also offer versions of software for MacOS, Linux, Unix and in the cloud via Microsoft Azure. Other companies that contribute to the open-source community include Adobe, Facebook, Google, IBM, Intel, LinkedIn, Netflix, Oracle and Twitter.

Why are people willing to contribute their time to the open-source community? It is especially puzzling because contributions are often made by programmers who develop software for a livelihood. If they are willing to freely contribute, then they are in essence producing for free something that they could earn a salary for creating instead. It is possible that the software they are helping to produce might compete against software that they are paid to produce or maintain. And yet being involved in the open-source community could help their careers. Table 1.9 lists several reasons for why developers freely contribute to open source projects as a means to further their career.

Participation in open-source development also gives the programmer an opportunity to give back to the open-source community, or even a larger audience, all computer users. Mozilla's Firefox, for instance, is such a success that a contribution to its development can be thought of as a contribution to much of humanity.

Contributions are not limited to just software developers. There are a lot of open-source programmers who program for a hobby and not as part of their job. Their motivations may be similar

TABLE 1.9
Some Reasons to Contribute to Open-Source Software Development

Improve coding skills.	Demonstrate a wider range of skills than what the developer uses at their current position.
Learn new technologies.	Learn different software approaches.
Gain experience with developers from other regions of the world.	Learn programming concepts not already known.
Improve on coding habits.	Make contacts with other developers.
Participate in a large group project.	Learn about or develop tools that can be applied in the development career.

to those who want to further their career: learning, working with other developers and producing something useful. But in their case, the impact may be more of a personal nature over improving their career.

Contributions can also be made by non-programmers through software testing. Most open-source software consists of very complex programs. Testing is a crucial aspect of software development, and it is typical that testers are not the same group as the developers. Testing can be performed at different levels. Software testers commonly run the software on a number of test cases, that is, groups of input. They then compare the output to the expectations and write up reports where the output diverges from expectation. In other cases, testers are users who compile lists of problems that they detect when using the software. For instance, if the software crashes when opening a file or if a particular feature operates too slowly, these cases are noted and sent back to the developers. Aside from testing, there is also a need for documentation writing.

The open-source ideology has extended beyond software development to other community-involved pursuits. Wikipedia is one of the best examples of this. The online encyclopedia receives contributions from people all over the world who want to share knowledge. Unlike open source software which requires some expertise as a programmer, Wikipedia only requires knowledge of a particular topic and the willingness to contribute. With Wikimedia, knowledge-based contributions extend to other media, namely images (whether drawings or photographic). We see artists sharing their products through the Internet using various forms of social media such as through YouTube and Bandcamp.

SECTION ACTIVITIES

1. We often hear of basement inventors who go on to make millions of dollars on their inventions (of course we mostly only hear about the successes, how many other inventors never profited from their inventions?) Torvalds made no attempt to profit from Linux. Yet, as a software developer he has made an immense success for himself and is estimated to be worth well over $100 million. Research his personal history to learn how he turned Linux into a fortune without actually making much money off of Linux itself.
2. In this section, we provided a URL that shows all of the Linux distributions. Use that link to examine this amazing figure. Try to estimate how many Linux distributions exist.
3. You have likely accessed and used other products of the open-source community like Wikipedia, Wikimedia and YouTube. Come up with a list of other websites where information is freely shared that you have used.

1.5 HOW DO YOU USE LINUX?

In this section, we look at the steps to install Linux. We examine several different Linux distributions. In each case, we make the assumption that the installation will be performed onto a virtual machine. If you want to try any of these on your own computer, it is best to download and install a virtual machine client program. VirtualBox from Oracle and VMWare Workstation are both open-source products. For complete information on installing Linux into a VM, see lab 0 in the lab manual that accompanies this textbook.

To install Linux, we need to get ahold of the Linux installation program, which is in the form of an `iso` file. We download the needed `iso` file directly onto hard disk from which to perform the installation. The versions of Linux we examine here are free (and available in open-source format for those interested in looking at or modifying the code) where the `iso` can be found on the Linux projects' websites.

We perform a default installation in most cases although discuss some of the options. Later in the textbook, we look at the steps required to perform a nonstandard partitioning of the Linux computer's file space. This section ends with a brief look at each of the Linux distributions we installed including their basic desktop setup and how to open and use a terminal window to enter commands.

1.5.1 Installing Debian Linux

The Debian installation image can be obtained from `www.debian.org/distrib`. For users who have slow Internet connections, a small image is available that can be saved onto an optical disc. Otherwise, the full image can be saved to optical disc or hard disk. A cloud image is available if you have an account in one of the commercial clouds although it is likely that your cloud provider already has a Debian image available so that you can skip installation and just create a copy.

Upon starting our VM software, we create a new VM. When asked for the type of OS, we select `Linux` and then more specifically `Debian`. We are asked for a location of the installation image and we select the downloaded `iso` file. Figure 1.4 shows the first four screens that we see during installation. First, we select the type of installation. We have selected `Graphical install` to ensure that our Linux installation comes with a GUI. Next, we are asked for our language (the default is English) and location. The location is used to set the time. We are next asked for the keyboard configuration (not shown in the figure, we selected `American English`). At this point, installation begins. A bar indicates the amount of installation completed. Messages appear during installation describing what is currently being installed. The next step is to specify the root password. Instructions explain the importance of selecting a strong password. We are asked to enter the password twice in this step.

As installation continues, the next step is to create an initial user account. We want to have a non-root account available so that when we log in, we log in as a normal user instead of root. Logging in as root through the GUI can lead to mistakes that could damage components of the OS or accidental deletion of files. It is best to switch to root sparingly and only from the command line. For the new user account, we are asked for the user's full name. Debian creates an account for this user. We are then asked to enter the new user account's initial password, twice, much like we did for root.

At this point of installation, we are asked to select how we want the disk space partitioned. The options are to have a guided approach or to perform partitioning manually. There are three forms of guided approaches, each uses the entire hard disk available (which in the case of a VM is not our computer's full hard disk, just the section reserved for the VM based on the size specified when we created the VM). One guided approach uses a standard partitioning, one uses a logical volume manager (LVM, covered in Chapter 8), and one uses an LVM but employs encryption. Note that if we were installing Linux on a dual boot computer, we would have to perform a manual installation so that we do not wipe out the existing operating system and files.

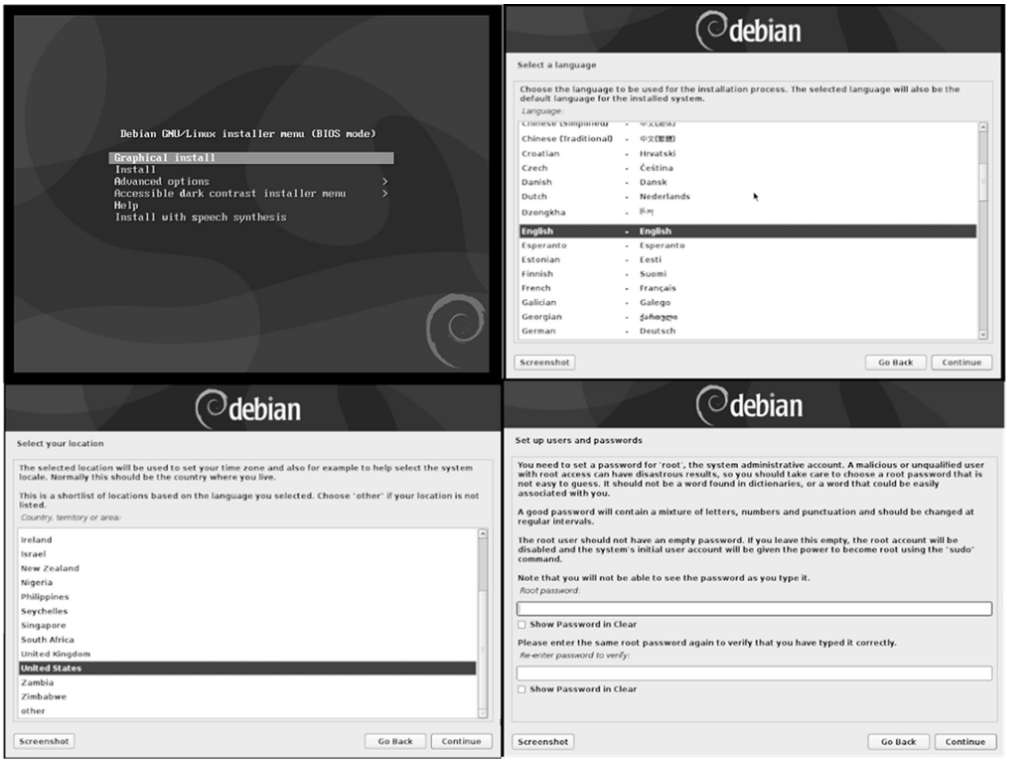

FIGURE 1.4 Four steps during Debian installation.

In Figure 1.5, we see the partition choices and, having selected `Guided – use entire disk`, we see three options. It is preferable to separate `/home` (the partition containing the user directories), `/var` (the partition containing various software data) and `/tmp` (the partition used by software to save temporary information) from the rest of the operating system. We next see in Figure 1.5 the recommended partitioning with sizes. Note that our disk space was limited to 32 GB as we were creating this Linux computer in a VM with limited hard disk space).

With partitioning specified, installation continues with the core packages. We are later asked if the GRUB boot loader should be installed in the master boot record. Our only choices are `no` or `yes`. Selecting `no` will require that we install our own boot loader later. We select `yes` and are asked where to place GRUB. This is shown in the lower right of Figure 1.5. The notation `/dev/sda` represents our hard disk. The rest of the installation is handled with no user interaction.

Upon completion, the VM should boot to our Debian installation and we will be shown the login screen like that shown in the left half of Figure 1.6. Here, there is only one user, the user we had established during installation. Having selected this user, we are asked for the user's password to complete the login process. Notice in the upper right (shown in full in the left side of the figure but not in the right side of the figure) are controls to change accessibility settings, network connection, the volume and "power settings" (log out, shut down, reboot).

Upon logging in, we are presented a mostly blank desktop. Along the upper left is a menu labeled Activities. Selecting this menu brings up a list of icons along the left margin, representing from top-to-bottom, the Firefox web browser, the Evolution Email program, the RhythmBox music program, LibreOffice Writer, the Linux File Browser, the Software program (software store), Debian Help and applications software. Upon selecting the application software icon, the installed titles are displayed in the desktop. On the right side of Figure 1.6, roughly half of the installed titles appear. A search bar appears at the top for easy searching (useful if we had more software titles installed). You will find that this basic setup is similar to both CentOS and Ubuntu.

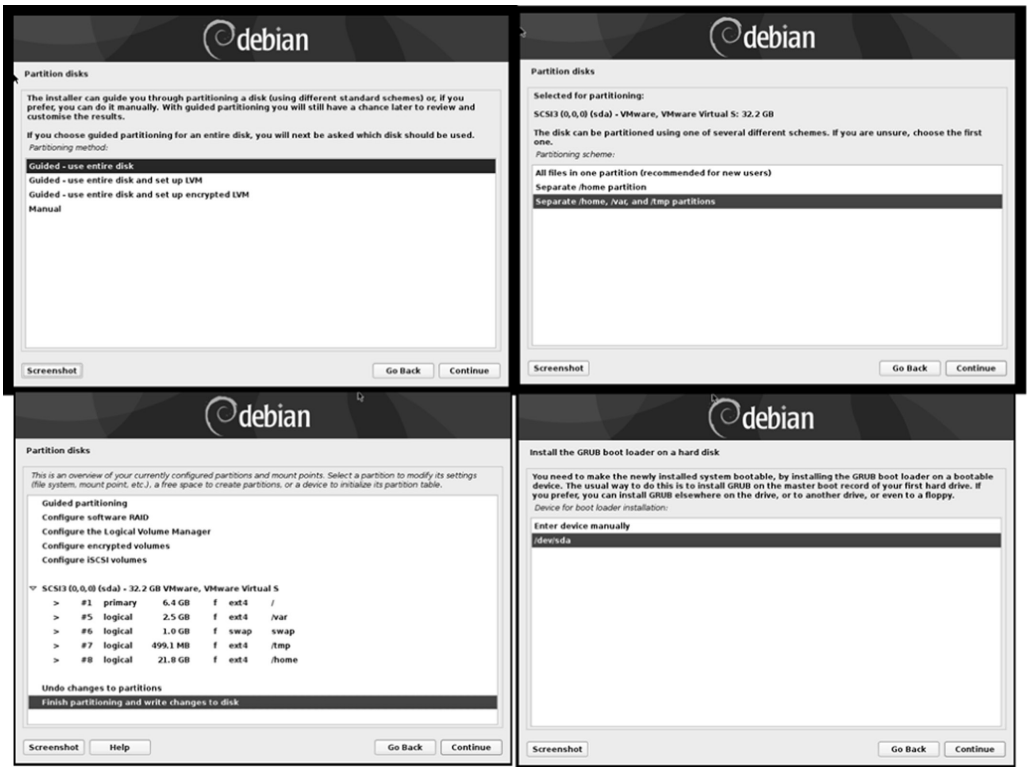

FIGURE 1.5 Partitioning the Debian installation.

FIGURE 1.6 Debian login screen and desktop.

1.5.2 Installing CentOS Linux

At the time of this writing, there are two different "current" versions of CentOS: CentOS Linux version 8 and CentOS Stream. In all previous releases of CentOS, the current version followed from the most recent release of RHEL. Thus, CentOS 8 is what you would expect to install. However, starting in 2020, CentOS announced that rather than releasing new CentOS versions based on the most recent RHEL version, CentOS Stream would offer versions of Red Hat that were, at least in some cases, ahead of the next RHEL version. This means that CentOS Stream releases may include

features that are proposed but not yet available in the most recent RHEL releases. Because of this shift in philosophy, CentOS has announced that CentOS 8's end of life will occur at the end of 2021. By the time you read this, it is likely that CentOS will have moved entirely on to Stream releases. In this section and throughout this textbook, we focus on CentOS Stream rather than CentOS 8. With that said, as of August 2021 the two releases are virtually identical.

We can obtain installation ISOs for both CentOS 8 and CentOS Stream at the `centos.org` website. Specifically, we find versions of CentOS 8 at `www.centos.org/centos-linux` and versions of CentOS Stream at `www.centos.org/centos-stream`. Download the `iso` of the version you prefer (to match the textbook, Stream is recommended). Upon downloading the `iso`, create a new VM and select the `iso` as the installation image.

The first step in installing CentOS is to select whether we want to install the OS, test the media and install the OS or troubleshoot. The troubleshooting selection is for a partially or fully installed OS that is not functioning. As we have just downloaded the `iso` and stored it on disk, there is no need to test the media and so we select Install (which should be the first choice). Like with Debian, we are asked for the language and keyboard layout, both of which default to English. Next, we will be presented with the `Installation Summary` window. We see an example of this in Figure 1.7. From the summary window, we are presented with choices that specify the remainder of the installation. We step through these choices one at a time.

We must respond to selections which appear with an orange triangle and an exclamation mark in it with a description listed in orange font. If all selections are in black font then the `Begin`

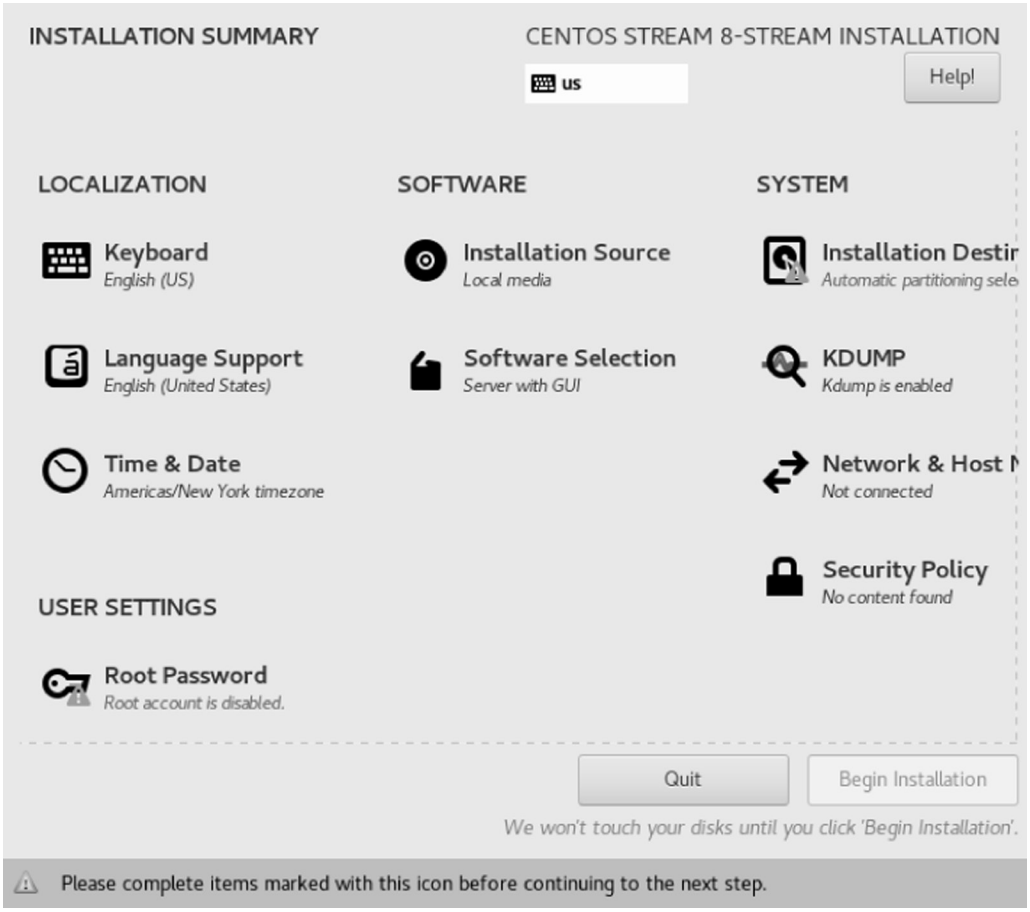

FIGURE 1.7 CentOS installation summary.

Installation button is accessible. In Figure 1.7, Installation Destination and Root Password are the only two selections with the orange triangle. We can select any other item to override the default but do not have to.

Selecting any item will take us to a new window so that we can specify our choice(s). For each window, there will be a Done button in the upper left of the window. Selecting it returns us to the Installation Summary window.

As we had previously selected Keyboard and Language Support, we can skip those. Time & Date is already set for us and we would only select this if we needed to alter the default. We will also not need to select Installation Source as this was specified when we started to create the new VM by specifying the location of the iso. We will not need to modify KDump or Security Policy. So, for us, we will concentrate on Software Selection, Installation Destination, Network & Host Name, and Root Password.

The Software Selection defaults to Server with GUI (note that earlier versions of CentOS defaulted to Minimal Install, which provides a text-based only version of Linux). Although Server with GUI is fine, if you do not intend to run a server (like a webserver) from your Linux VM, it may not be the best choice. The list of choices runs along the left pane, as shown in Figure 1.8. For us, we will select Workstation.

Upon selecting one of these base environments, a list of Additional software is provided in the right pane. The software listed is specific to the base environment selected. Among the items available for a workstation are GNOME applications (GNOME is the GUI desktop), Internet applications, Office suite, legacy UNIX compatibility software and development tools. If we were going to use our Linux computer to write code, we would want the last of these, perhaps the last two. For us, we will choose GNOME applications only. We select Done to return to the Installation Summary window.

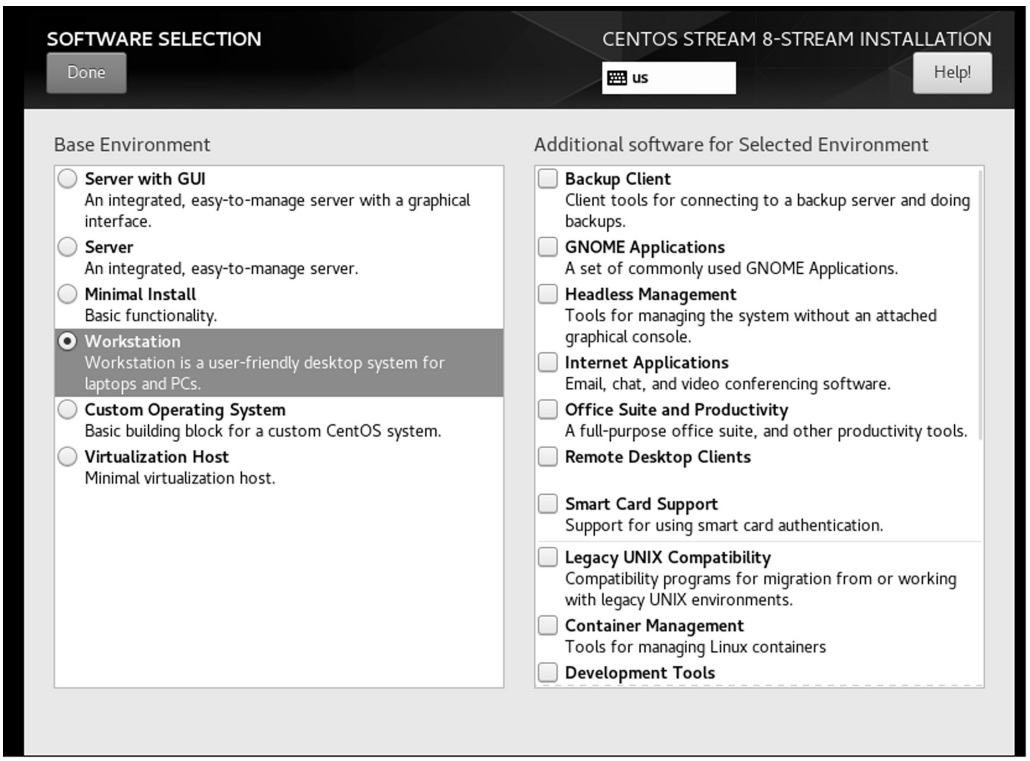

FIGURE 1.8 Base environment and software selection choices.

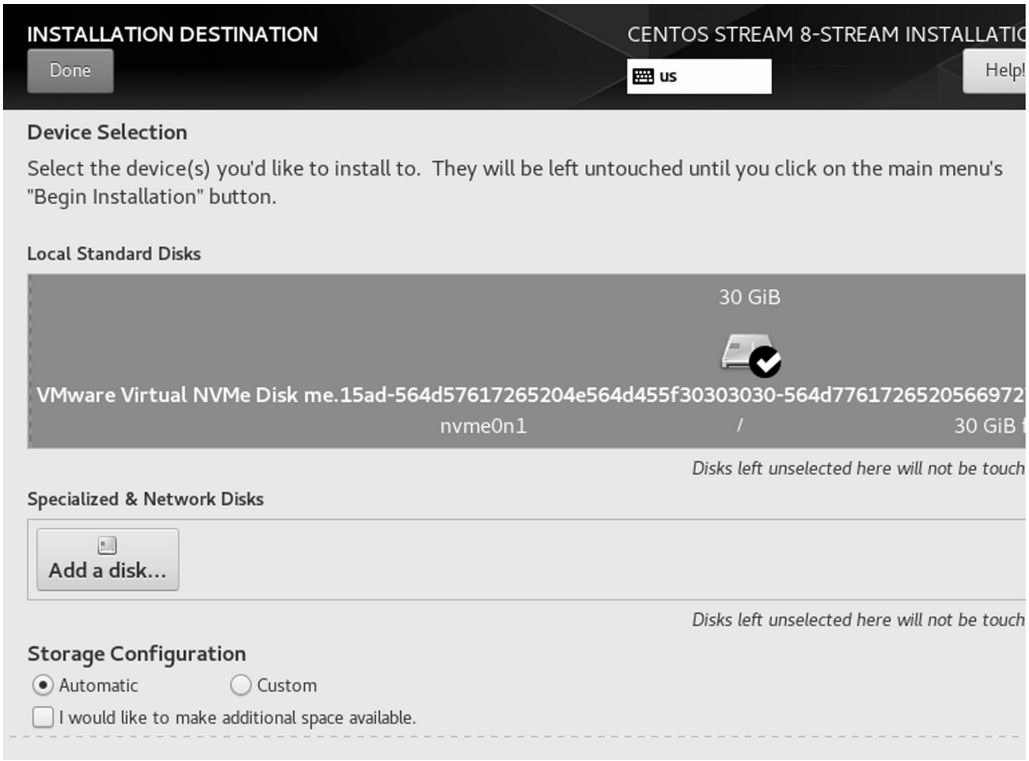

FIGURE 1.9 Installation destination window to specify partitioning.

Next is `Installation Destination`. By default, `Automatic` partitioning is selected. Should we wish to change this, select `Installation Destination`. The `Installation Destination` window is shown in Figure 1.9. Here, we would select the disk(s) onto which to install. There is only one choice in this case, labeled as `VMWare Virtual NVMe Disk me` followed by a lengthy identifier. To select this, click on it twice. The first time highlights it, and the second time adds a checkmark. Without the checkmark, the disk is not selected. Note that as we are installing into a VM, we have a small hard disk size.

Selecting `Done` returns us to the `Installation Summary`. If we had chosen `Custom`, selecting `Done` would take us to a `Manual Partitioning` Window, as shown in Figure 1.10. We cover this step for completeness but suggest that you use the `Automatic` partitioning.

From the `Manual Partitioning` Window, we can select between `Standard Partitioning`, `LVM` and `LVM Thin Partitioning` (the default is LVM). The `Standard Partition` selection allows us to specify the exact number, type and size of partitions. We omit detail for a standard partitioning until we cover this in Chapter 8.

The `LVM` (also covered in Chapter 8) allows our partitions to grow as needed and is the more reasonable choice, at least at this point. `Thin Provisioning` makes more efficient use of disk space at the expense of added run-time overhead. `LVM` will be sufficient for our needs.

One issue with using an LVM is that we need to separate out the boot partition. We need to create one separate partition, for `/boot`. Near the bottom left of the `Manual Partitioning` window are buttons labeled + and -. Select the + to add a new partition. From the `Add A New Mount Point` window (shown at the bottom of Figure 1.10), select `/boot` for the mount point and specify a desired capacity. For CentOS Stream, it is recommended that this partition be 1 GB, so we enter `1024` and select `Add Mount Point`. After being returned to the `Manual Partitioning` window, select `Done`. We will be presented with a summary of the changes. Select `Accept Changes`

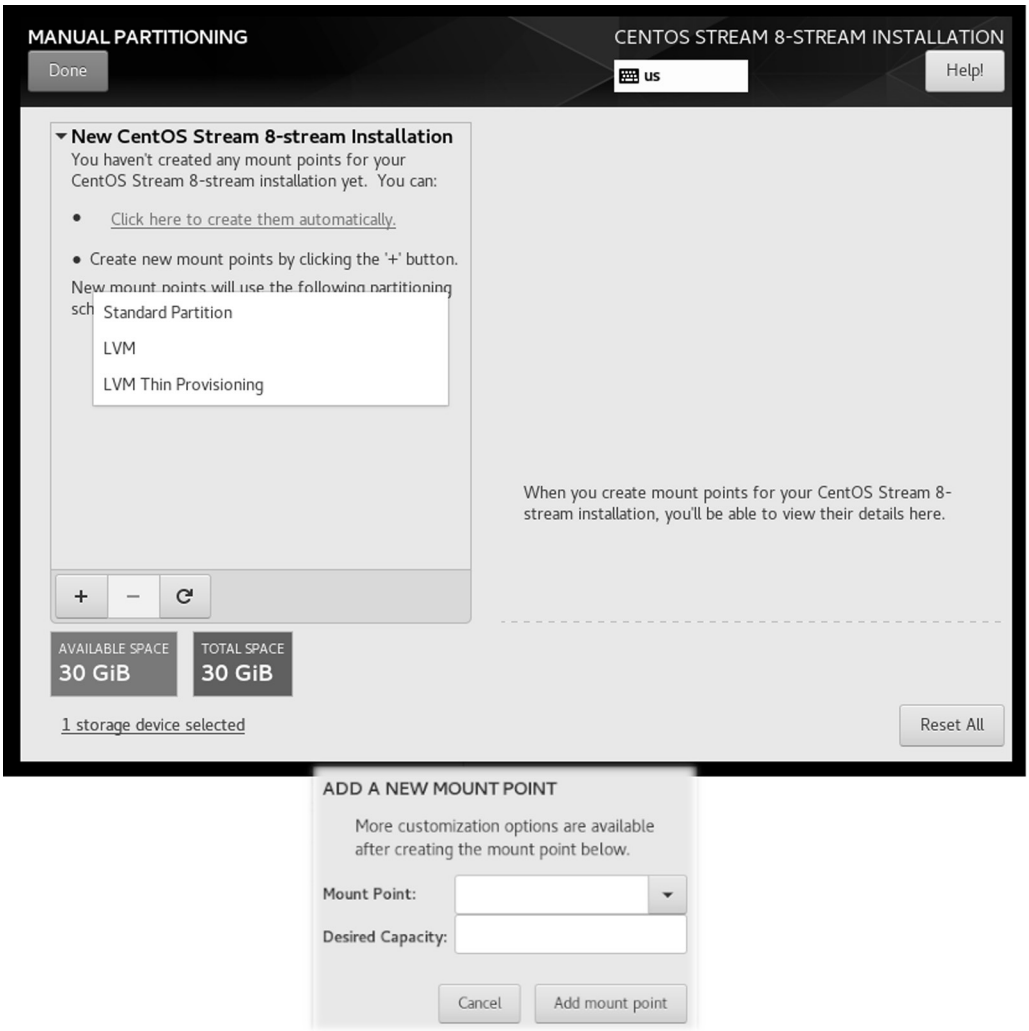

FIGURE 1.10 Manual partitioning.

and we will be returned to the `Installation Summary` window to continue. Note that should we choose, we can return to `Installation Destination`, select `Automatic` and `Done` to continue.

For `Network & Host Name`, the default is to disable the network interface. We might want to change this to enabled. We can also change the default hostname, which is initially set to `localhost.localdomain`. We can also add other interface devices. The default device is an Ethernet connection. Select `Network & Host Name` and from this window, `enable` the network and select `Done`.

Selecting `Root Password` allows us to specify and confirm the root password. A bar indicates how strong the password is. We would prefer to use a strong password. Use a password for root that you will not forget! Upon returning from the `Root Password` window, the `Installation Summary` has another entry that we can select, `User Creation`. Selecting this entry lets us specify an initial user account. As noted in the subsection on installing Debian Linux, it is preferable to have an initial user account so that we can log in as a normal user rather than root.

From the Create User window, we are asked for the user's full name. Typing in the name causes CentOS to automatically create a username using the format first initial last name. We might prefer to alter the default name. For instance, if we enter Frank Zappa as the user's name, we are given the username of fzappa but we might prefer something like zappaf or zappaf1. We are also asked to specify the user's initial password and confirm it.

Two checkboxes allow us to require a password for this account (selected by default) and to make this user an administrator (and thus give this user account access to all administrator programs, this is not selected by default and we want to leave it unselected). The Advanced... button lets us tailor this account manually by changing the home directory, change UID/GID and group membership. Do not bother changing the initial user's default values. We explore such options in Chapter 7. Selecting Done returns us to the Installation Summary where we are finally ready to perform the installation. We select Begin Installation.

Installation takes some time. Upon completion of installation, we are given a new option, Reboot. We must reboot to start using Linux. Upon booting (and in all future instances of booting or rebooting our CentOS VM), the first screen shown asks us which version we want to boot to. By default, CentOS installs with two kernels: the normal kernel and the rescue kernel. The rescue kernel is only needed if our OS develops some faults and we can't successfully boot to it. The normal kernel is selected automatically if we do not make a selection within a few seconds.

After rebooting, we are brought to an Initial Setup window with one choice, to view and accept the License Information. The license agreement is displayed (its brief!); select I accept the license agreement and Done. Upon returning to the Initial Setup window, select Finish Configuration in the lower right of the window. The computer then takes us to a login screen very much like that of Debian (see the left side of Figure 1.6).

The first time a user logs into his or her account, CentOS runs a user setup program. The first window is a Welcome screen whereby the user selects their language (followed by keyboard language). These will default to whatever language we selected during installation (English in our case). Next, the user can select whether to permit location services to run or not (privacy). Finally, the user can select which online accounts the computer should connect to. This step can be skipped. After this initial user setup is completed, the user is presented with a Ready to Go window, which is a tutorial for showing us how to use GNOME. Closing this window brings us to the CentOS GNOME Desktop. We can return to the GNOME Help window any time we like as explained below.

The GNOME Desktop in CentOS is very similar to that of Debian (see the right side of Figure 1.6). The Activities menu brings up our list of favorites, just as it did in Debian, although the initial favorites may differ. One of the items in this list looks like a life preserver. Selecting it returns us to the GNOME Help window. Selecting the last entry expands all installed software. It is likely that the CentOS Stream installation has fewer titles than Debian's installation. One item is Utilities. Selecting this displays the various CentOS utility programs. Two are of particular interest to us, System Monitor (which is similar to Windows Task Manager and a program we explore in Chapter 4) and Terminal. Right click on Terminal and select Add to Favorites. This places the terminal window icon on the activities list for easy access. We can drag this icon to the top of the list for even more convenient access.

Another of the software selections is called Settings. This presents a GUI much like Windows 10's Settings tool, although there are fewer selections available. We see some of the selections in the left pane of Figure 1.11 where we have selected Network. The available network settings are shown in the right pane. Note that while we can use this GUI to modify settings, we will find other ways to modify settings as well, including Cockpit (which we explore next) and through programs launched from the command line.

As Linux users, we would likely not need administrative tools beyond the settings available through the Settings tool. As a system administrator, we will need to be able to change any number of OS settings, most of which are not available through the Settings tool. In older versions of CentOS/RHEL, there were some administrator GUI tools available, including those to manage user and group

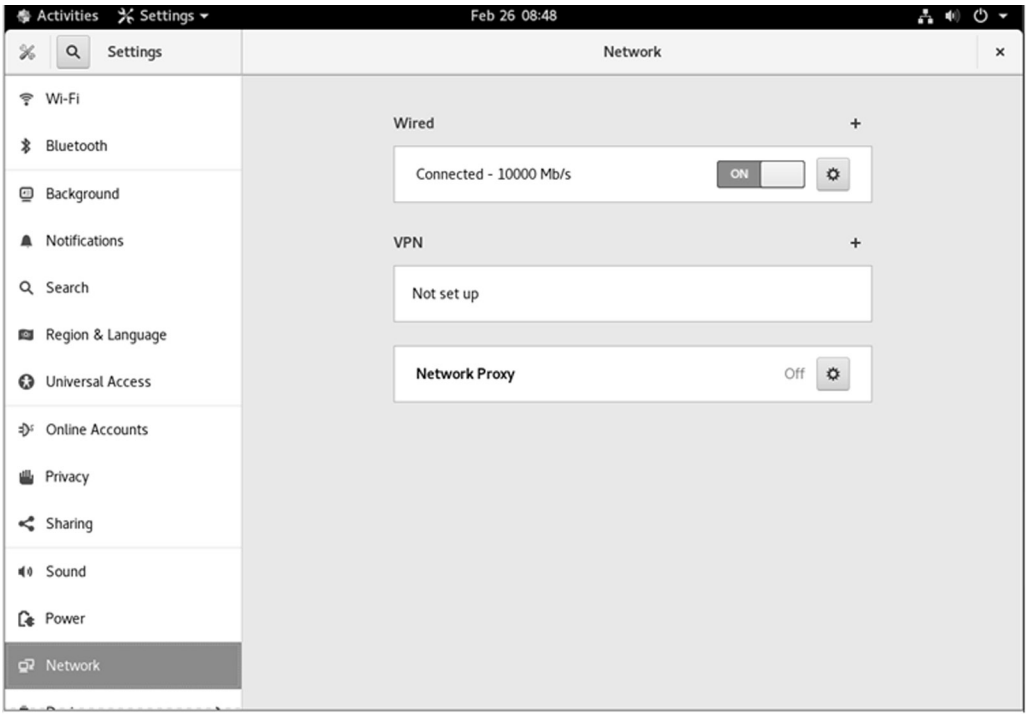

FIGURE 1.11 Changing settings in CentOS.

accounts, control `KDump`, start and stop services and alter the firewall. With Red Hat 8 (CentOS 8, CentOS Stream, RHEL 8), most of the GUI tools no longer exist and instead we manage these tasks either through command line or through the new *Cockpit* interface. Our access to Cockpit is through a web browser using our computer's IP address followed by `:9090`, as in `10.11.12.13:9090`.

As Cockpit loads, we are asked to log in. If we log in under our user account, our access will be restricted to mostly the same items as we saw in Settings. If we log in as root, we are given additional capabilities including user account management. Figure 1.12 shows the Cockpit interface with System selected to show performance information. The controls along the left pane allow us to view log files and examine currently used and available storage, network settings, current user accounts, available software and other items.

1.5.3 Installing Ubuntu Linux

Having started the Ubuntu installation, we are presented with some installation screens like those shown in Figure 1.13. The last two screens of the figure show some of the information that we are presented with during installation. We are asked a few installation questions, similar to that of CentOS: language and keyboard layout, time zone, type of software installation (our choices here are normal, minimal or other), installation type and initial user account information. For the installation type, if there is already a version of Ubuntu installed, we are asked whether to erase it and install the new one or install the new version alongside the existing version. Other choices are to erase the disk, encrypt the installation and use an LVM. We can also select `Something else` to specify our own partitioning. We skip these details.

The initial user account selection differs from that in CentOS and Debian in one significant way. In CentOS and Debian, we are asked to provide a root password so that we can have user accounts and a separate administrator account. In Ubuntu, the root account's password is randomly generated and not given to the user. In place of having root-level access, the first user account is given full

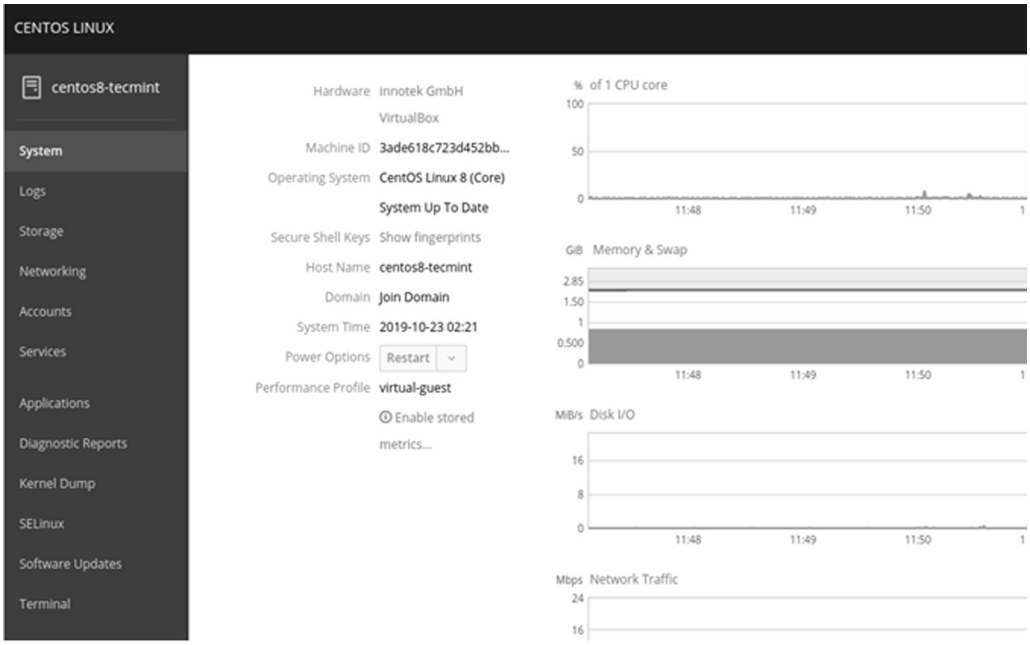

FIGURE 1.12 `Cockpit`.

administrator privileges through `sudo`. The `sudo` program, covered in Chapter 7, allows a user to run programs under different access rights. In the case of Ubuntu, the initial user account is given access to all system administrator programs so that this user never has to log in as root.

This change in approach to installing Linux originated with Ubuntu and is found in several Linux distributions, mimicking to some extent Windows 10 installation for home users where the first user account is given full administrator privileges. To specify the initial user, we are asked for the user's name, the username, the computer's name and a password. We are not, however, asked if this user should be made an administrator because this choice is made automatically. After installation, we are asked to reboot.

After rebooting, we are presented a login screen which will be similar to that of Debian and CentOS except for the color scheme. After logging in, the user is presented with a desktop very similar to that of Debian and CentOS except that there are two desktop icons already created (the user's home directory and a trash can). The Ubuntu desktop is shown in Figure 1.14.

1.5.4 Installing Linux Mint

Linux Mint is the easiest of all installations as there is almost no user interaction. Upon creating a new VM and selecting the Linux Mint `iso`, we are presented with a menu of installation choices much like in Debian (the upper left window of Figure 1.4). Installation proceeds from this point with no interruption. Unlike the other Linux installations, Mint installs with a generic user account and no initial password. This initial account, like in Ubuntu, has full administrator privileges as the root password is unknown.

After installation, the desktop appears, as shown in the left half of Figure 1.15. Here, we see two desktop icons for the computer and for the user home directory, and a list of icons along the bottom left. The leftmost icon brings up a menu similar in style to the Windows 10 Start Button menu. Upon selecting this start button, the menu, which we see in the right half of Figure 1.15, appears. The other icons along the lower left, running from left to run, display the desktop (the default view), open a web browser, open a terminal window and launch the file browser GUI.

FIGURE 1.13 Ubuntu installation.

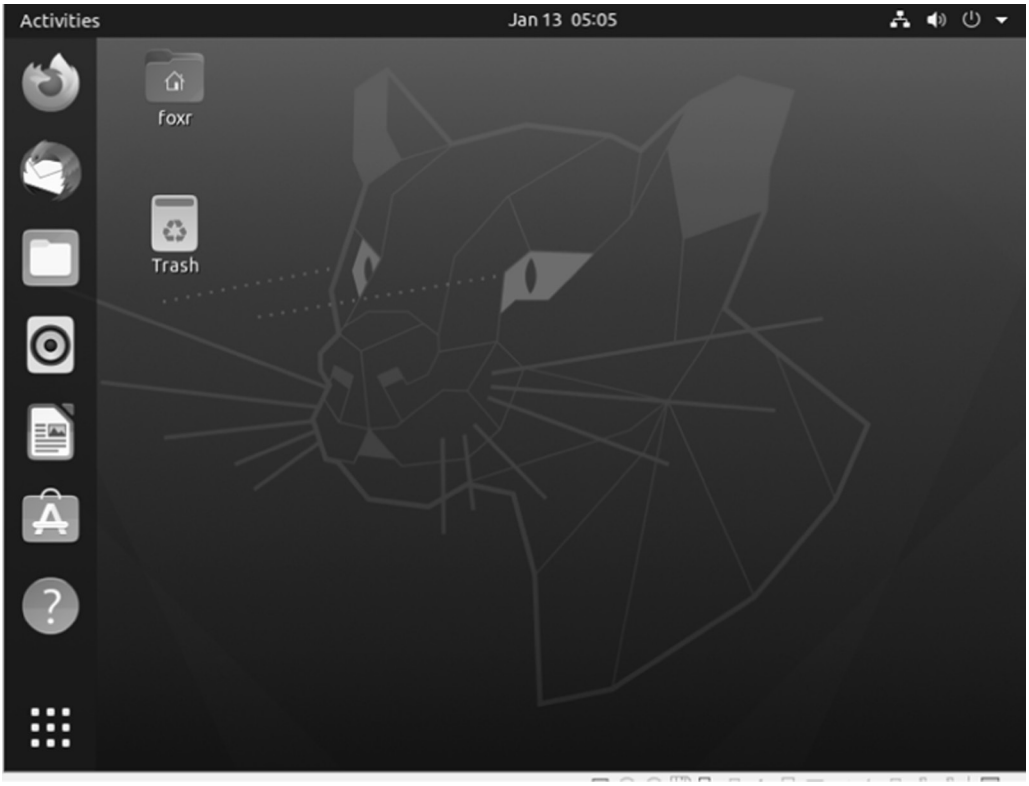

FIGURE 1.14 Ubuntu desktop.

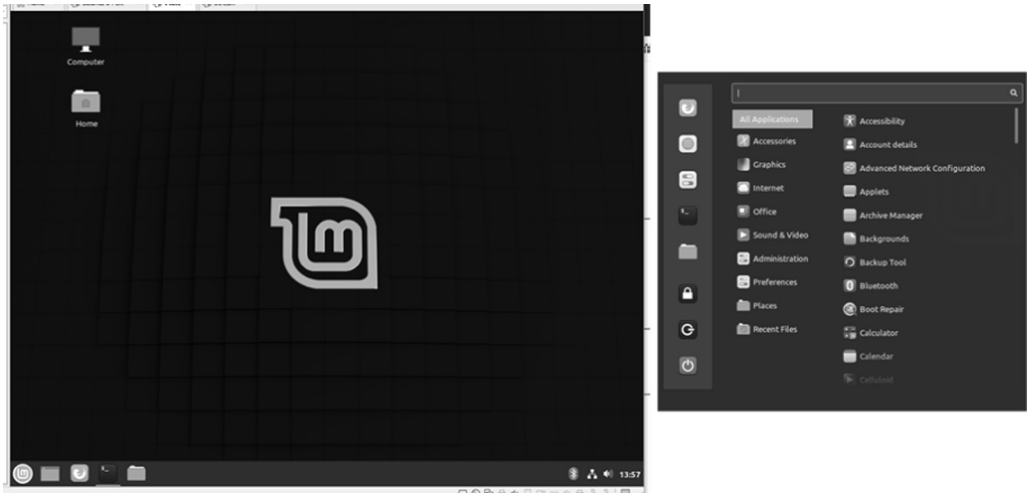

FIGURE 1.15 Linux Mint desktop.

1.5.5 An Introduction to the Shell and Command Line

No matter which version of Linux you may have installed, they all have the capability of opening a terminal window. A terminal window is a GUI component which itself runs a shell and an interpreter. The window has limited capabilities such as minimize, maximize, resize, zoom in and out,

and cut, copy and paste text from within it. The shell running inside the window provides the user with a command line prompt.

From the prompt, the user inputs Linux instructions. Any instruction entered is then handled by the shell's interpreter. Most Linux users have a default shell of Bash, so commands are run by the Bash interpreter. As we will explore in Chapter 2, we can open shell sessions from within a shell and in doing so, change the type of shell (e.g., from `bash` to `tcsh`). In Chapter 7, we cover how to change the default shell of a user.

Opening a terminal window from the desktop is handled differently depending on the distribution. In Debian, CentOS and Ubuntu, select `Activities` and from the favorites list, select the `Terminal Window` (which looks like window with a black background and a white prompt). The CentOS Stream Workstation installation does not have a terminal window icon in this menu and so we have to add it as discussed previously to make it a part of our favorites list. In Debian, the terminal is called `XTerm`.

We see an example of an open terminal window in CentOS in Figure 1.16. Notice at the top of the desktop we have an added menu alongside `Activities` called `Terminal`. From this menu, we can close our terminal window (by selecting `Quit`; we can also close the terminal window by clicking on the `x` in the upper-right hand corner of the terminal window, or typing `exit` from the command prompt) but we can also open additional terminal windows (`New Window`). The terminal window itself has menus for controlling copying and pasting, zooming, etc. The `File` menu also has a selection to open a new window, open a tab, close the window or close the current tab.

The terminal window gives us the ability to enter Linux commands from the command line prompt in the window. Upon pressing `<enter>`, the interpreter executes the command for us. We see some very simple interactions in Figure 1.17 where we first ask who we are logged in as (`whoami`), the current time and date (`date`), where we are located in the file space (`pwd`) and who else is logged in (`who`). These are among the simplest instructions available in Linux. We spend most of the rest of this textbook looking at dozens of other commands.

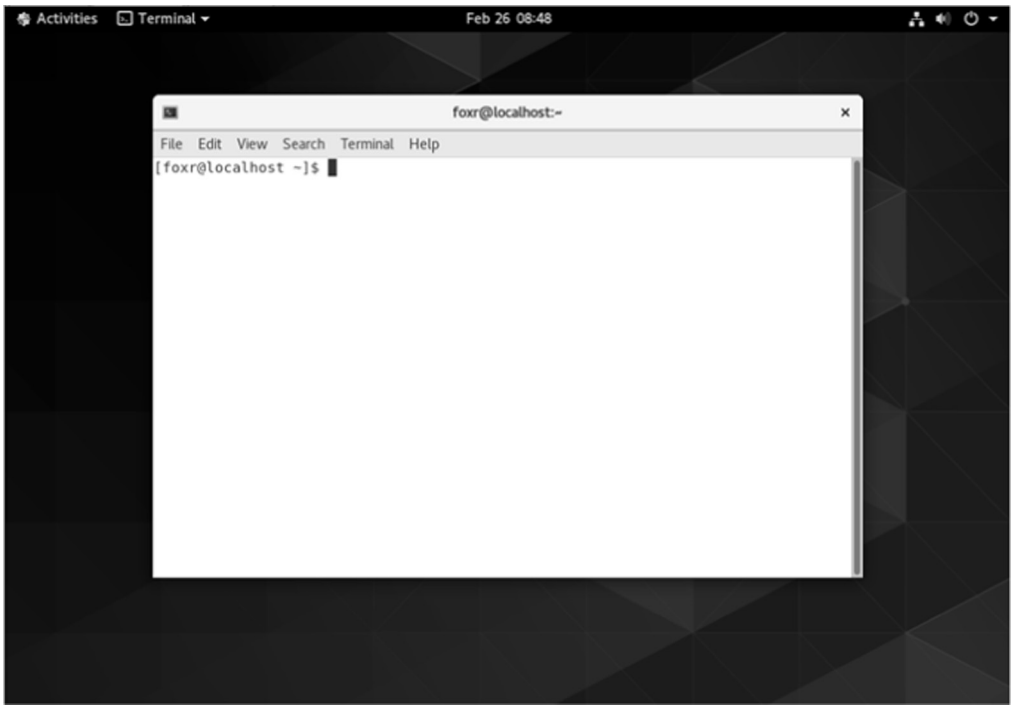

FIGURE 1.16 A terminal window.

```
[foxr@localhost ~]$ whoami
foxr
[foxr@localhost ~]$ date
Fri Mar 12 13:10:58 EST 2021
[foxr@localhost ~]$ pwd
/home/foxr
[foxr@localhost ~]$ who
foxr       tty1       2021-03-12 13:11 (tty1)
zappaf     pts/2      2021-03-10 08:44 (10.11.12.13)
```

FIGURE 1.17 Some simple interaction through a terminal window.

One additional instruction to learn is `passwd`, to change our password. It is likely that a user is given an initial password from the system administrator and is told to change that password at their first login. In our case, this is not necessary because we created our own initial password when we installed Linux.

The `passwd` command, when issued with no parameter, attempts to change our password. We are asked for our current password. As we type in the password, it is not displayed on the screen. We are then asked for the new password and again, as we type it is not displayed on the screen. If the password satisfies whatever strong password requirements are set up in the system, we are asked to confirm the password. As we will explore in Chapter 7 when we look at user accounts, root can also use password to change other users' passwords or change attributes about passwords.

SECTION ACTIVITIES

1. Of the four versions of Linux we examined in this installation section, which would you pick to use? Would you select the one easiest to install? The one that looked the most user-friendly? The one with the most available preinstalled software? The one that has the most games available?
2. If you are taking a Linux class, you will probably do some labs using Linux. If this is not the case, download and install an open-source virtual machine client like VirtualBox and then download and install one of the Linux distributions covered in this section.

1.6 CHAPTER REVIEW

Concepts and terms introduced in this chapter:

- Bare machine – a computer with no operating system installed to ensure the fastest possible hardware execution.
- BSD Unix – one of the most significant spinoffs of Unix, developed by Ken Thompson and faculty and students at UC Berkeley; widely distributed in the 1980s and 1990s.
- Commercial support – the ability to ask for (and expect) assistance with software when purchased; support includes timely bug fixes and help when you do not know how to accomplish some task that the software should be able to perform.
- Device driver – a piece of software tailored for a specific hardware device allowing the CPU to communicate with that hardware device.
- CentOS – a distribution of Red Hat Linux that is open source, lightweight and a popular choice for servers.
- Cockpit – a recent addition to Linux to give the user and system administrator a uniform means of viewing the system and changing settings; Cockpit runs through a web browser.

- Copyleft – a term coined by Richard Stallman about the GPL to express that the license supports freedom rather than restrictions on software usage.
- Debian – one of the earliest distributions of Linux; popular because of its stable releases.
- Fedora – a distribution of Red Hat Linux that is open source and supported entirely by the open source community.
- File descriptor – a means for the OS kernel to refer to a resource such as an open file.
- Free software foundation – the name originally given to the open-source community but after a splintering is the group that feels that all software should be free.
- GNU – a project to develop a Unix-like operating system that was entirely free with no licensing restrictions; the project was headed by Richard Stallman but never completed; members of the GNU project contributed time and already implemented code to the development of Linux.
- GPL – the GNU Public License, a copyleft which supports users' (and developers') rights rather than restrictions on using a piece of software.
- HeartBleed – a flaw found in the encryption technology of OpenSSL which was patched less than a week after the flaw was discovered, showing the responsiveness of the open-source community.
- Hybrid kernel – a type of OS kernel which combines aspects of microkernels and monolithic kernels.
- ISO – the abbreviation references the International Organization of Standards where ISO 9660 specifies optical disc storage; in this context, an ISO is the format used to store Linux installation programs.
- Kernel module – in order to make a Linux kernel more efficient, components of the kernel, called modules, can be loaded into and out of memory.
- Knoppix – a distribution of Linux descended from Ubuntu; Knoppix was the first Linux distribution that featured live booting.
- Lightweight – the idea that an OS (or other piece of software) can run on limited resources; many Linux distributions are considered lightweight particularly when compared to OSs like Windows.
- Linux distribution – a specific version of Linux; there are hundreds of different distros that are all based on the same Linux kernel but have different features, services, available applications software and GUI components.
- Linux kernel – a monolithic kernel with modules to make it more efficient; most Linux distributions are based on the same kernel.
- Linux Mint – a Linux distribution based on Debian Linux known for being lightweight.
- Live boot – the ability to boot from and run the OS off of an optical disc (or USB flash drive) rather than one that is installed on hard disk.
- Microkernel – a form of OS kernel which hands off some of its duties to lesser programs called servers.
- Monolithic kernel – a form of OS kernel which handles all of its tasks internally and thus makes the kernel more complex; Linux, while based on a monolithic kernel, hands off some of its tasks to modules and services (not to be confused with servers).
- MULTICS – an early 1960s operating system that introduced many new OS innovations and was a basis for many of Unix's original features.
- Open-source community – people who freely contribute their time and effort to developing and supporting open-source software.
- Open-source initiative (OSI) – a splinter of the open-source community who believe that some software can be commercially developed and distributed as opposed to the *Free Software Foundation*; the OSI makes up the larger fraction of the open-source community.

- Open-source software – a software product that is made available in its source code format so that developers can download and modify the code; as it is available in source code format it is freely available and usually also available in an easy-to-install executable format as well for non-developers.
- Operating System (OS) – a piece of software responsible for monitoring the resources of a computer and for acting as an intermediary between the user, the application software and computer hardware.
- OS kernel – the core program of any OS; the kernel is loaded into memory at the time the computer boots and remains in memory while the computer runs; the kernel is responsible for handling the most important tasks of the OS including process management, resource management, memory management, interprocess communication and interrupt handling.
- OS utility – a standalone program that is typically invoked by the user to help manage the computer system; unlike the kernel and OS services, a utility runs in the foreground and often presents to the user a GUI-based tool to interface with.
- Portability – the aspect of software where the program is written to run on multiple platforms; Unix was the first OS developed to be platform-independent.
- Privileged mode – most processors run in one of two modes: user mode to run user programs and privileged (or supervisor) mode to run system software.
- Red Hat Enterprise Linux (RHEL) – the version of Red Hat that is only available as a commercial product but comes with Red Hat, Inc. commercial support; CentOS is an open source version related to RHEL.
- Red Hat Linux – one of the earliest distributions of Linux and one of the most popular; Red Hat is somewhat unique because one of its spinoffs, Red Hat Enterprise Linux, is a commercial product.
- Resident monitor – before OSs were created, many mainframes loaded and ran a resident monitor to support simple user interactions such as loading and running programs off of magnetic tape.
- Server – in the context of this chapter, a server is a piece of the OS separate from the kernel and invoked by the kernel so that the kernel can be smaller in size; this strategy is used in support of microkernels.
- Service – an OS program separate from the kernel used to support some functionality of the OS; we find services used in OSs like Linux and Windows.
- Settings – a GUI tool available in most modern Linux distributions so that the user can change personalized settings; similar in style to modern Windows operating systems Settings tool.
- SLS/Slackware/SUSE Linux – SLS was the earliest Linux spinoff distribution which was the basis for the improved Slackware which itself was used to create SUSE Linux.
- System call – an invocation by a user program or OS component to call upon a function of the kernel.
- Ubuntu – a spinoff of Debian Linux which became popular because of its user-friendly approach and a large number of software products.
- Unix – developed in the late 1960s by AT&T Bell Labs and modeled somewhat on MULTICS, Unix became a popular, portable and powerful operating system; Linux' look is very similar to that of Unix.
- User mode – the default mode for processors so that user processes can only make requests of the OS; the processor must switch modes to privileged mode to invoke the OS and access system resources.
- Wrapper function – a system call invokes a wrapper function which itself invokes the OS kernel; the wrapper function is responsible for handling pre- and post-invocation tasks such as reporting an error to the user.

REVIEW QUESTIONS

1. Why is commercial support desired when purchasing software? What are some of the guarantees that come with commercial support?
2. In what way is the HeartBleed bug a good indicator that open-source software does not need commercial support?
3. Are all Linux distributions free?
4. One argument for using Linux over an OS like Windows is greater control. How does Linux offer greater control?
5. Which of the following is true of the earliest mainframe computers: they were bare machines, they had resident monitors, they had operating systems or they were virtual machines?
6. Explain the steps by which a FORTRAN programmer would compile and run their FORTRAN program on a mainframe computer of 1958.
7. At what point in time would a resident monitor be loaded into memory? How long would it remain in memory?
8. What resident monitor/OS was the first to provide time sharing (multitasking)?
9. Which role of the OS kernel allows multiple running processes to share data in memory?
10. Virtual memory would be considered a feature handled by which of the OS kernel's roles?
11. Synchronization and deadlock handling are tasks handled by which of the OS kernel's roles?
12. Upon receiving a system call from a user process, the kernel switches the processor from _____ mode to _____ mode.
13. Linux contains which type of kernel: microkernel, monolithic kernel, hybrid kernel?
14. Which of the forms of kernel is the most challenging to debug because of its complexity: microkernel, monolithic kernel, hybrid kernel?
15. Examine the system calls in Table 1.4 to answer the following questions.
    a. Which system call(s) would be involved in creating a new file?
    b. Which system call(s) would be involved in starting a new process?
16. True/false: All device drivers are preinstalled into the OS.
17. Linux services run in the foreground or background?
18. List three types of OS utilities.
19. What members of AT&T Bell Labs worked on MULTICS and later started developing Unix?
20. What was the original name for Unix?
21. Who was Richard Stallman and what project did he begin?
22. Read Figure 1.3 (the GPL Preamble). Instead of restrictions on software, it talks about users' _____.
23. In what year did Linus Torvalds begin working on Linux? In what year was the first full implementation made available?
24. Between SLS/Slackware, Debian and Red Hat, which appeared first?
25. List three popular distributions based on Debian Linux.
26. List three popular distributions based on Red Hat Linux.
27. In what way is RHEL different from CentOS?
28. How does CentOS Stream differ from previous CentOS releases?
29. What is the main difference between Debian-based Linux distributions and Red Hat-based Linux distributions?
30. List three reasons for why a developer might contribute to open-source software.
31. What is an ISO file?
32. True/false: When installing Linux, you must specify how the disk drive is to be partitioned.
33. When installing CentOS Linux, what two choices are you required to specify from the Installation Summary window? At what point does the Begin Installation button become accessible?

34. When installing a recent version of Debian, CentOS and Ubuntu, there is a similar Desktop. What is the name of the only menu that appears when the desktop first appears?
35. What is GNOME?
36. As a user, what can you do through the Cockpit interface?
37. What is the name of the one menu available in current versions of Debian and CentOS Linux?
38. List five types of settings that a user can change.
39. One difference between installing one of Debian or CentOS Linux from Ubuntu is that in Debian and CentOS you are asked for the root password. How does Ubuntu provide root-level access if the user does not know the root password?

References

Abrams, M. , LaPadula, L. , Eggers, K. , and Olson, I. A. Generalized Framework for Access Control: An Informal Description, Proceedings of the 13th National Computer Security Conference, pp. 135–143, 1990.

Accetta, M. , Baron, R. , Bolosky, W. , Golub, D. , Rashid, R. , Tevanian, A. , and Young, M. Mach: A New Kernel Foundation for Unix Development, Proceedings of the Summer 1986 USENIX Conference, pp. 93–112, 1986.

Adelstein, T. , and Lubanovic, B . Linux System Administration, Sebastopol, CA: O'Reilly, 2007.

Aho, A. , Kernighan, B. , and Weinberger, P . The AWK Programming Language, Boston, MA: Addison-Wesley, 1988.

Al-Hafeedh, A. , Crochemore, M. , Ilie, L. , Kopylova, E. , Smyth, W. , Tishcler, G. , and Yusufu, M. A. Comparison of index-based Lempel-Ziv LZ77 factorization algorithms, ACM Computing Surveys, Vol. 45, Issue 1, 5:9–5:17, Article 5, 2012.

Allen, N . Network Maintenance and Troubleshooting Guide: Field Tested Solutions for Everyday Problems, Boston, MA: Addison-Wesley, 2009.

Almesberger, W . Booting Linux: The History and the Future, Proceedings of the Ottawa Linux Symposium, 2000.

Almgren, M. , Debar, H. , and Dacier, M. A. Lightweight Tool for Detecting Web Server Attacks, Networks and Distributed System Security Symposium, pp. 157–170, 2000.

Anderson, K . Convergence: A holistic approach to risk management, Network Security, Vol. 2007, Issue 5, pp. 4–7, Amsterdam: Elsevier Science Publishers, 2007.

Aulds, C . Linux Apache Web Server Administration, Laguna Beach, CA: SYBEX, 2000.

Bach, M . The Design of the Unix Operating System, Hoboken, NJ: Prentice-Hall, 1999.

Bagherzadeh, M. , Kahani, N. , Bezemer, C. , Hassan, A. , Dingel, J. , and Cordy, J . Analyzing a decade of Linux system calls. Empirical Software Engineering, Vol. 23, Issue 3, pp. 1519–1551, New York: Springer, 2018.

Balasubramanian, K. , and Johnson, D . Linux Memory Management Overview, The Linux Kernel Hacker's Guide, http://www.redhat.com:8080/HyperNews/get/memory/memory.html, 1993.

Bar, M . Linux File Systems, New York: McGraw-Hill, 2001.

Barrett, D. , Silverman, R. , and Byrnes, R . Linux Security Cookbook, Newton, MA: O'Reilly, 2003.

Bembenek, J. , and Klus, A . Grep Pocket Reference, Newton, MA: O'Reilly, 2009.

Bennett, H. CD-E: Call It Erasable, Call It Rewritable, but Will It Fly? CD-ROM Professional, 1996.

Benvenuti, C . Understanding Linux Network Internals, Newton, MA: O'Reilly, 2006.

Berry, D . Copy, Rip, Burn: The Politics of Copyleft and Open Source, London: Pluto Press, 2008.

Billimoria, K . Linux Kernel Programming: A Comprehensive Guide to Kernel Internals, Writing Kernel Modules, and Kernel Synchronization, Birmingham: Packt, 2021.

Black, D. , Golub, D. , Julin, D. , Rashid, R. , Draves, R. , Dean, R. , Forin, A. , Barrera, J. , Tokadu, H. , Malan, G. , and Bohman, D . Microkernel Operating System Architecture and Mach, Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, pp. 11–30, 1992.

Blum, R. , and Bresnahan, C . Linux Command Line and Shell Scripting Bible, Hoboken, NJ: Wiley & Sons, 2021.

Bonaccorsi, A. , and Rossi, C . Why open source software can succeed, Research Policy, Vol. 32, Issue 7, pp. 1149–1292, Elsevier, 2003.

Both, D . systemd, Using and Administering Linux, Vol. 2, pp. 379–410, Berkeley, CA: Apress, 2020.

Bourne, S.R. . The UNIX Shell, The Bell System Technical Journal, NJ: American Telephone and Telegraph Company, Vol. 57, Issue 6, pp. 1971–1990, 1978.

Bovet, D . Understanding the Linux Kernel, Newton, MA: O'Reilly, 2005.

Bradley, D . The Divergent Anarcho-Utopian discourses of the open source software movement, Canadian Journal of Communication, Vol. 30, pp. 585–611, 2005.

Bridger, R . Introduction to Human Factors and Ergonomics, Orlando, FL: CSC Press, 2017.

Brinch, P . Classic Operating Systems: From Batch Processing to Distributed Systems, New York: Springer-Verlag, 2001.

Brookshear, J . Computer Science: An Overview, Boston, MA: Pearson, 2018.

Brown, F . Boolean Reasoning: The Logic of Boolean Equations, New York: Dover, 2012.

Brown, G . zOS JCL (Job Control Language), Hoboken, NJ: John Wiley & Sons, 2002.

Burtch, K . Linux Shell Scripting with Bash, Boston, MA: Pearson, 2004.

Calderon, P . Nmap: Network Exploration and Security Auditing Cookbook, Birmingham: Packt Publishing, 2017.

Callaghan, B . NFS Illustrated, Boston, MA: Addison-Wesley, 2000.

Ceruzzi, P . A History of Modern Computing, Cambridge, MA: The MIT Press, 2003.

Chau, P . Selection of packaged software in small business, European Journal of Information Systems, Vol. 3, Issue 4, pp. 292–302, 1994.

Chen, P. , Lee, E. , Gibson, G. , Katz, R. , and Patterson, D. RAID: High-performance, reliable secondary storage, ACM Computing Surveys, Vol. 26, Issue 2, pp. 145–185, 1994.
Chervenak, A. , Vellanki, V. , and Kurmas, Z . Protecting File Systems: A Survey of Backup Techniques, Proceedings of the Joint NASA and IEEE Mass Storage Conference, 1998.
Cheung, W. , and Loong, A . Exploring issues of operating systems structuring: From microkernel to extensible systems, Operating Systems Review, Vol. 29, pp. 4–16, 1995.
Ciampa, M . Security+Guide to Network Security Fundamentals, Boston, MA: Thomson Course Technologies, 2011.
Clements, A . The Principles of Computer Hardware, New York:Oxford, 2006.
Cole, E . Network Security Bible, Hoboken, NJ: Wiley and Sons, 2009.
Comer, D . Internetworking with TCP/IP, Boston, MA: Addison-Wesley, 2013.
Corner, D . Computer Networks and Internets, Boston, MA: Pearson, 2014.
Cox, R. , Muthitacharoen, A. , and Morris, R . Serving DNS using a peer-to-peer lookup service, Lecture Notes in Computer Science, Vol. 2429, pp. 155–165, New York: Springer, 2002.
Cramer, R. , and Shoup, V . Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack, SIAM Journal on Computing, Vol. 33, pp. 167–226, 2001.
Crawley, D . The Accidental Administrator: Linux Server Step-by-Step Configuration Guide, Seattle, WA: CreateSpace, 2010.
Dada, E. , Bassi, J. , Chiroma, H. , Adetunmbi, A . and Ajibuwa, O . Machine learning for Email spam filtering: Review, approaches and open research problems, Heliyon, Vol. 5, Issue 6, pp. e01802, 2019.
Dallheimer, M. , and Welsh, M . Running Linux, Newton, MA: O'Reilly, 2005.
Davis, D. , and Swick, R . Network security via private-key certificates, ACM SIGOPS Operating Systems Review, Vol. 24, Issue 4, pp. 64–67, 1990.
De Goyeneche, J . Loadable kernel modules, IEEE Software, Vol. 16, Issue 1, pp. 65–71, 1999.
Denning, P . Thrashing: Its Causes and Prevention, Proceedings of the December 9–11 1968 AFIPS Fall Join Computer Conference, pp. 915–922, New York: ACM, 1968.
Denning, P . Virtual memory, ACM Computing Surveys, Vol. 2, Issue 3, pp. 153–189, 1970.
Denning, P. A Short Theory of Multiprogramming, Proceedings of the 3rd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 2–7, 1995.
Dijkstra, E . The structure of the THE multiprogramming system, Communications of the ACM, Vol. 11, Issue 5, pp. 341–346, New York: ACM, 1968.
Doeppner, T . Operating Systems in Depth: Design and Programming, Hoboken, NJ: Wiley and Sons, 2010.
Droms, R . Automated configuration of TCP/IP with DHCP, IEEE Internet Computing, Vol. 3, Issue 4, pp. 45–53, 1999.
Durumeric, Z. , Li, F. , Kasten, J. , Amann, J. , Beekman, J. , Payer, M. , Weaver, N. , Adrian, D. , Paxson, V. , Bailey, M . and Halderman, J . The Matter of Heartbleed, Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 475–488, New York: ACM, 2014.
Easttom, W . Computer Security Fundamentals, Boston, MA: Pearson, 2019.
Ebrahim, M . and Mallett, A . Mastering Linux Shell Scripting: A Practical Guide to Linux Command-line, Bash Scripting and Shell Programming, Birmingham: Packt, 2018.
Economides, N. , and Katsamakas, E . Linux vs. windows: a comparison of application and platform innovation incentives for open source and proprietary software platforms, The Economics of Open Software Development, J. Bitzer and P. Schroder (eds), Amsterdam: Elsevier, 2006.
Elmasri, R. , Carrick, A. , and Levine, D . Operating Systems: A Spiral Approach, New York: McGraw-Hill, 2009.
Fedora DOCS , https://docs.fedoraproject.org/en-US/docs/.
Fenwick, P . The burrows-wheeler transform for block sorting text compression: Principles and improvements, Computer Journal, Vol. 39, Issue 9, pp. 731–740, New York: Oxford University Press, 1996.
Fitzgerald, B . The transformation of open source software, MIS Quarterly, Vol. 30, Issue 30, pp. 587–598, 2006.
Forouzan, B . TCP/IP Protocol Suite, New York: McGraw-Hill, 2010.
Forouzan, B . Data Communications and Networking, New York: McGraw-Hill, 2012.
Fox, R . Information Technology: An Introduction for Today's Digital World, Boca Raton, FL: CRC Press, 2021.
Fox, R. , and Hao, W . Internet Infrastructure: Networking, Web Services and Cloud Computing, Boca Raton, FL: CRC Press, 2017.
Friedl, J . Mastering Regular Expressions, Newton, MA: O'Reilly, 2006.
Frisch, E . Essential System Administration, Newton, MA: O'Reilly, 2002.
Gajewska, H. , Manasse, M. , and McCormack, J . Why X is not our ideal window system, Software—Practice & Experience, Vol. 20, Issue S2, 1990.
Galov, N . 111 + Linux Statistics and Facts – Linux Rocks!, Hosting Tribunal blog, https://hostingtribunal.com/blog/linux-statistics/.
Gancarz, M . Linux and the Unix Philosophy, Daytona Beach, FL: Digital Press, 2003.

Garfinkel, S. , Spafford, G. , and Schwartz, A . Practical Unix and Internet Security, Newton, MA: O'Reilly, 2003.
Garrels, M . Bash Guide for Beginners, CA: Fultus Corporation, 2010.
Garrido, J. , and Schlesinger, R . Principles of Modern Operating Systems, Burlington, MA: Jones and Bartlett, 2011.
Gay, J. , Stallman, R. , and Lessig, L . Free Software, Free Society: Selected Essays of Richard M. Stallman, Seattle, WA: CreateSpace, 2009.
Geisshirt, K . Pluggable Authentication Modules, Birmingham: Packt, 2007.
Gillay, C . Linux User's Guide: Using the Command Line and Gnome Red Hat Linux, OR: Franklin, Beedle & Associates, 2003.
GNU/Free Software Foundation , General Public License, http://www.gnu.org/copyleft/gpl.html.
GNU/Free Software Foundation , GNU Make, https://www.gnu.org/software/make/
Goldman, D. , and Bonzini, P . Definitive Guide to Sed: Tutorial and Reference, Renton, WA: EHDP Press, 2013.
Gomberg, M. , Evard, R. , and Stacey, C. A Comparison of Large-Scale Software Installation Methods on NT and Unix, Proceedings of USENIX Technical Program on Large Installation System Administration of Windows NT Conference, pp. 37–48, 1998.
Goodheart, B. , and Cox, J . The Magic Garden Explained: The Internals of UNIX System V Release 4 An Open Systems Design, Hoboken, NJ: Prentice-Hall, 1994.
Goyal, V. , Horman, N. , Ohmichi, K. , Soni, M. , and Garg, A . Kdump: Smarter, Easier, Trustier, Ottawa Linux Symposium, 2007.
Goyvaerts, J. , and Levithan, S . Regular Expressions Cookbook, Newton, MA: O'Reilly, 2012.
Grampp, F. , and Morris, R. UNIX operating-system security, AT&T Bell Laboratories Technical Journal, Vol. 63, pp. 1649–1672, 1984.
Green, R. , Baird, A. , and Davies, J . Designing a fast, on-line backup system for a log-structured file system, Digital Technical Journal of Digital Equipment Corporation, Vol. 8, Issue 2, pp. 32–45, 1986.
Gregg, J . Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets, Hoboken, NJ: Wiley-IEEE Press, 1998.
Groom, F . The structure and software of the internet, Annual Review of Communications, Vol. 50, pp. 695–707, 1997.
Guttman, E . Service location protocol: automatic discovery of IP network services, IEEE Internet Computing, Vol. 3, Issue 4, pp. 71–80, 1999.
Haff, G . How Open Source Ate Software: Understanding the Open Source Movement and So Much More, Hoboken, NJ: Apress, 2018.
Hagen, S . IPv6 Essentials, Newton, MA: O'Reilly, 2014.
Halsall, F . Data Communications, Computer Networks, and Open Systems, Boston, MA: Addison-Wesley, 1996.
Hamming, R . Error detecting and error correcting codes, Bell System Technical Journal, Vol. 29, Issue 2, pp. 147–160, 1950.
Hansen, P . (editor). Classic Operating Systems: From Batch Processing to Distributed Systems, New York: Springer, 2010.
Harker, J. , Brede, D. , Pattison, R. , Santana, G. , and Taft, L. A quarter century of disk file innovation, IBM Journal of Research and Development, Vol. 25, Issue 5, pp. 677–689, 1981.
Harkins, M . Managing Risk and Information Security: Protect to Enable, Hoboken, NJ: Apress, 2016.
Hartig, H. , Hohmuth, M. , Liedtke, J. , Schonberg, S. , and Wolter, J . The Performance of Micro-Kernel-Based Systems, Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997.
Hecker, F. Setting up shop: the business of open-source software, IEEE Software, Vol. 16, Issue 1, pp. 45–51, 1999.
Helmke, M . Ubuntu Unleashed, Boston, MA: Addison-Wesley, 2020.
Hennessy, J . and Patterson, D . Computer Architecture: A Quantitative Approach, San Francisco, CA: Morgan Kaufman, 2017.
Hicks, B. , Rueda, S. , St. Clair, L. , Jaeger, T. , and McDaniel, P. A logical specification and analysis for SELinux MLS policy, ACM Transactions on Information and System Security, Vol. 13, Issue 3, pp. 26–31, Article 26, 2010.
Hill, B. , Burger, C. , Jesse, J. , and Bacon, J . The Official Ubuntu Book, Boston, MA: Pearson, 2016.
Holcombe, C. , and Holcombe, J . Survey of Operating Systems, New York: McGraw-Hill Education, 2019.
Howser, G . Computer Networks and the Internet: A Hands-On Approach, New York: Springer, 2019.
Hunt, C . TCP/IP Network Administration, Newton, MA: O'Reilly, 2002.
Jacob, B. , and Mudge, T . Virtual memory in contemporary multiprocessors, IEEE Micro Magazine, Vol. 18, pp. 60–75, 1998.
Jacob, B. , and Wang, D . Memory Systems: Cache, DRAM, Disk, San Francisco, CA: Morgan Kaufmann, 2007.
Jang, M . Security Strategies in Linux Platforms and Applications, Burlington, MA: Jones and Bartlett, 2015.

Kalkhanda, S . CentOS Quick Start Guide: Get Up and Running with CentOS Server Administration, Birmingham: Packt, 2018.
Kalkhanda, S. , Learning AWK Programming: A Fast and Simple Cutting-edge Utility for Text-Processing on the Unix-like Environment, Birmingham: Packt, 2018.
Kamel, M. , Keast, J. , and Pal, C . Concrete Architecture of the Linux Kernel, technical report, University of Waterloo, Waterloo, Ontario, N2L 3G1. Department of Electrical and Computer Engineering, Department of Computer Science, 1998.
Katz, J. , and Lindell, Y . Introduction to Modern Cryptography, Boca Raton, FL: CRC Press, 2020.
Kernighan, B. UNIX: A History and a Memoir, Independently Published, 2019.
Kernighan, B. , and Pike, R . The Unix Programming Environment, Boston, MA: Pearson, 2015.
Kernighan, B. , and Ritchie, D . The C Programming Language, Hoboken, NJ: Prentice-Hall, 1988.
Kiddle, O. , Stephenson, P. , and Peek, J . From Bash to Z Shell: Conquering the Command Line, Hoboken, NJ: Apress Media LLC, 2004.
Kirkbride, P . systemd, Basic Linux Terminal Tips and Tricks, pp. 221–234, Berkeley, CA: Apress, 2020.
Kirtch, O . Linux Network Administrator's Guide, Newton, MA: O'Reilly, 1995.
Knuth, D . Dynamic huffman coding, Journal of Algorithms, Vol. 6, Issue 2, pp. 163–180, Amsterdam: Elsevier, 1985.
Krishnamurthy, B. , and Rexford, J . Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching and Traffic Measure, Boston, MA: Addison-Wesley, 2001.
Kumar, A . Migration from Microsoft to Linux on Servers and Desktops, Proceedings of the 20th Annual Conference of the National Advisory Committee on Computing Qualifications, S. Mann and N. Bridgeman (eds), pp. 117–123, 2007.
Langfeldt, N . The Concise Guide to DNS and Bind, Indianapolis, IN: Que Corp, 2000.
Laurent, A . Understanding Open Source and Free Software Licensing, Newton, MA: O'Reilly, 2004.
Leach, R . Advanced Topics in UNIX: Processes, Files and Systems, Hoboken, NJ: Wiley and Sons, 1994.
LeBlanc, D. , and Yates, I . Linux Install and Configuration Little Black Book: The Must-Have Troubleshooting Guide to Installing and Configuring Linux, Scottsdale, AZ: Coriolis Open Press, 1999.
Ledin, J . Modern Computer Architecture and Organization: Learn x86, ARM and RISC-V Architectures and the Design of Smartphones, PCs, and Cloud Servers, Birmingham: Packt, 2020.
Lempel, A. , and Ziv, J . On the complexity of finite sequences, IEEE Transactions on Information Theory, Vol. 22, Issue 1, pp. 75–81, 1976.
Lerner J. , Pathak P. A. , and Tirole, J . The dynamics of open source contributors, American Economic Review, Vol. 96, Issue 2, pp. 114–118, 2006.
Lewine, D . POSIX Programmer's Guide: Writing Portable Unix Programs, Newton, MA: O'Reilly, 1992.
Li, Y. , Li, W. , and Jiang, C. A Survey of Virtual Machine Systems: Current Technology and Future Trends, Proceedings of the Third International Symposium on Electronic Commerce and Security, 2010.
Liang, Y . and Liang, Y. , Introduction to Java Programming and Data Structures, Boston, MA: Pearson, 2017.
Liedtke, J . On Micro-Kernel Construction, Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, 1995.
Liedtke, J . Toward real microkernels, Communications of the ACM, Vol. 39, Issue 9, pp. 70–79, 1996.
Limoncelli, T. , Hogan, C. , and Chalup, S . The Practice of System and Network Administration, Boston, MA: Addison-Wesley, 2016.
Lin, Y. , Hwang, R. , and Baker, F . Computer Networks: An Open Source Approach, New York: McGraw-Hill, 2011.
Liu, C . DNS and BIND on IPv6: DNS for the Next-Generation Internet, Newton, MA: O'Reilly, 2011.
Loscocco, P. , and Smalley, S . Integrating Flexible Support for Security Policies into the Linux Operating System, Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, The USENIX Association, pp. 29–42, CA: USENIX Association, 2001.
Love, R . Linux Kernel Development, Hoboken, NJ: Addison-Wesley, 2010.
Lu, L. , Arpaci-Dusseau, A , Arpaci-Dusseau, R. , and Lu, S. A study of Linux file system evolution, ACM Transactions on Storage, Vol. 10, Issue 1, pp. 1–32, 2014.
Luotonen, A. , and Altis, K . World-Wide Web Proxies, Computer Networks and ISDN Systems, Vol. 27, Issue 2, pp. 147–154, Amsterdam: Elsevier, 1994.
Mallett, A . CentOS System Administration Essentials, Birmingham: Packt, 2014.
Mann, S. , and Mitchell, E . Linux System Security: The Administrator's Guide to Open Source Security Tools, Hoboken, NJ: Prentice-Hall, 2002.
Mansfield, K. , and Antonakos, J . Computer Networking for LANs to WANs: Hardware, Software and Security, Boston, MA: Thomson Course Technology, 2009.
Markatos, E. , Katevenis, M. , Pnevmatikatos, D. , and Flouris, M . Secondary Storage Management for Web Proxies, Proceedings of USITS 99: The 2nd Conference on USENIX Symposium on Internet Technologies and Systems, Vol. 2, pp. 93–114, CA: USENIX, 1999.
Matotek, D. , Turnbull, J . and Lieverdink, P . Pro Linux System Administration: Learn to Build Systems for Your Business Using Free and Open Source Software, Hoboken, NJ: Apress, 2017.

Matthews, J . Computer Networking: Internet Protocols in Action, Hoboken, NJ: Apress, 2005.
Mauelshagen, H . Logical Volume Manager (LVM2). Red Hat Magazine, 2004.
Mauerer, W . Professional Linux Kernel Architecture, Hoboken, NJ: Wrox, 2008.
Meeker, H . The Open Source Alternative: Understanding Risks and Leveraging Opportunities, Hoboken, NJ: Wiley & Sons, 2008.
Melve, I. , Slettjord, L. , Bekker, H. , and Verschuren, T . Building a Web Caching System—Architectural Considerations, Proceedings of the 1997 NLANR Web Cache Workshop, CA: National Laboratory for Applied Network Research, 1997.
Menezes, A. , Oorschot, P. , and Vanstone, S . Handbook of Applied Cryptography (Discrete Mathematics and Its Applications), Boca Raton, FL: CRC Press, 1996.
Miles, S . Linux Closing in on Microsoft Market Share, Study Says, CENT News.com, http://news.com.om/2100-1001-243527.html?tag=mainstry, 2000.
Mockapetris, P. , and Dunlap, K . Development of the Domain Name System, SIGCOMM 88 Symposium Proceedings on Communications Architectures and Protocols, pp. 123–133, New York: ACM, 1988.
Moody, G . Rebel Code: Linux and the Open Source Revolution, New York: Basic Books, 2009.
Morris, R. , and Thompson, K. , Password security: a case history, Communications of the ACM, Vol. 22, Issue 11, pp. 594–597, New York: ACM, 1979.
Moshe, B . Linux File Systems, New York: McGraw-Hill, 2001.
Mustonen, M . Copyleft—the economics of Linux and other open source software, Information Economics and Policy, Vol. 15, Issue 1, pp. 99–121, Amsterdam: Elsevier, 2003.
Negus, C. , and Boronczyk, T . CentOS Bible, Hoboken, NJ: Wiley and Sons, 2009.
Negus, C. , and Bresnahan, C . Linux Bible, Hoboken, NJ: Wiley and Sons, 2020.
Nemeth, E. , Snyder, G. , Hein, T. , Whaley, B. , and Mackin, D . Unix and Linux System Administration Handbook, Boston, MA: Addison-Wesley, 2017.
Newham, C . Learning the Bash Shell: Unix Shell Programming, Newton, MA: O'Reilly, 2005.
Null, L. , and Lobur, J . The Essentials of Computer Organization and Architecture, Burlington, MA: Jones and Bartlett, 2018.
Odom, W . Computer Networking First-Step, Indianapolis, IN: Cisco Press, 2004.
O'Regan, G . Introduction to the History of Computing, New York: Springer, 2016.
Otero, A. Information Technology Control and Audit, Boca Raton, FL: Auerbach Publications, 2020.
Paar, C. , and Pelzl, J . Understanding Cryptography, New York: Springer. 2010.
Pate, S . UNIX Filesystems: Evolution, Design and Implementation, Hoboken, NJ: Wiley and Sons, 2003.
Patterson, D. , Gibson, G. , and Katz, R. A. Case for Redundant Arrays of Inexpensive Disks (RAID), Proceedings of SIGMOD '88, pp. 109–116, New York: ACM, 1988.
Patterson, D. , and Hennessy, J . Computer Organization and Design: The Hardware/Software Interface, San Francisco, CA: Morgan Kaufmann, 2011.
Peek, J. , Powers, S. , O'Reilly, T. , and Loukides, M . Unix Power Tools, Newton, MA: O'Reilly, 2007.
Peterson, L. , and Davie, B . Computer Networks: A Systems Approach, San Francisco, CA: Morgan Kaufmann, 2021.
Petersen, R . Linux: The Complete Reference, New York: McGraw-Hill, 2007.
Pfleeger, C. , and Pfleeger, S . Security in Computing, Boston, MA: Pearson, 2018.
Plank, J. A. Tutorial on reed-solomon coding for fault-tolerance in RAID-like systems, Software Practice and Experience, Vol. 27, Issue 9, pp. 995–1012, Hoboken, NJ: Wiley and Sons, 1997.
Portnoy, M . Virtualization Essentials, Hoboken, NJ: Sybex, 2016.
Prabhakaran, V. , Arpaci-Dusseau, A. , and Arpaci-Dusseau, R. Analysis and Evolution of Journaling File Systems, USENIX '05 Online Proceedings, pp. 105–120, 2005.
Prasse, P. , Sawade, C. , Landwehr, N . and Scheffer, T . Learning to identify concise regular expressions that describe email campaigns, Journal of Machine Learning Research, Vol. 16, Issue 1, pp. 3687–3720, 2015.
Puryear, D . Best Practices for Managing Linux and Unix Servers, New York: Penton, 2006.
Quarterman, J. , and Hoskins, H . Notable computer networks, Communications of the ACM, Vol. 29, Issue 10, pp. 932–971, New York: ACM, 1986.
Quigley, E . UNIX Shells by Example, Hoboken, NJ: Prentice-Hall, 2001.
Ramey, C. , and Fox, B . Bash Reference Manual, http://www.gnu.org/software/bash/manual, Network Theory LTD, 2012.
Rash, M . Linux Firewalls: Attack Detection and Response with iptables, psad and fwsnort, San Fransico, CA: No Starch Press, 2007.
Raymond, E . The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, Newton, MA: O'Reilly, 2001.
Red Hat Inc , Product Documentation for Red Hat Enterprise Linux 8, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/8/, 2021.
Refsdal, A. , Solhaug, B . and Stolen, K . Cyber-Risk Management, New York: Springer, 2015.

Rescorla, E . An introduction to openssl programming, Linux Journal, Vol. 2001, Issue 89, p. 3, Article 3, 2001.
Reynolds, G . Ethics in Information Technology, Boston, MA: Cengage, 2018.
Riehle, D . The economic motivation of open source: stakeholder perspectives, IEEE Computer, Vol. 40, Issue 4, pp. 25–32, 2007.
Ritchie, D . The Evolution of the UNIX Time-Sharing System, Language Design and Programming Methodology, Lecture Notes on Computer Science, Vol. 79, Berlin: Springer-Verlag, 1979.
Ritchie, D. , and Thompson, K . The UNIX time-sharing system, Communications of the ACM, Vol. 17, Issue 7, pp. 365–375, 1974.
Robbins, A . VI Editor Pocket Reference, Newton, MA: O'Reilly, 1999.
Robbins, A . Sed and Awk: A Pocket Reference, Newton, MA: O'Reilly, 2002.
Robbins, A . Bash Pocket Reference, Newton, MA: O'Reilly, 2016.
Robbins, A. , and Beebe, N . Classic Shell Scripting, Newton, MA: O'Reilly, 2005.
Rodeh, O. , Bacik, J . and Mason, C. BTRFS: the Linux B-tree filesystem. ACM Transactions on Storage, Vol. 9, Issue 3, pp. 1–32, 2013.
Rosenfeld, L . and Downey, A . Think Perl 6: How to Think Like a Computer Scientist, Newton, MA: O'Reilly, 2017.
Ross, K . and Kurose, J . Computer Networking: A Top-Down Approach, Boston, MA: Pearson, 2017.
Royon, Y. , and Frenot, S. A Survey of Unix Init Schemes, Technical report arXiv:0706.2748v2, Cornell University Library, 2007.
Rusen, C . Networking Your Computers & Devices Step by Step, Redmond, WA: Microsoft Press, 2011.
Rusling, D. The Linux Kernel, http://sunsite.unc.edu/Linux/LDP/tlk/tlk.html, 2001.
Ryan, P . Linux Market Share Set to Surpass Win 98, OS X Still Ahead of Vista, http://arstechnica.com/apple/2007/09/linux-marketshare-set-to-surpass-windows-98, 2007.
Saini, K . Squid Proxy Server 3.1: Beginner's Guide, Birmingham: Packt Publishing, 2011.
Salas, P . The Daemon, the GNU and the Penguin, Virginia Beach, VA: Reed Media Services, 2008.
Salus, P. (ed). A Quarter Century of Unix, Boston, MA: Addison-Wesley, 1994.
Samar, V . Unified Login with Pluggable Authentication Modules, Proceedings of the 3rd ACM Conference on Computer and Communications Security, pp. 1–10, ACM Press, 1996.
Sandberg, R. , Goldberg, D. , Kleiman, S. , Walsh, D. , and Lyon, B . Design and Implementation of the Sun Network File System, Proceedings of the 1985 USENIX Summer Conference, pp. 119–130, 1985.
Sandhu, R. , and Samarati, P . Access control: principles and practice, IEEE Communications, Vol. 32, pp. 40–48, 1994.
Sarwar, S. , and Koretsky, R . Unix: The Textbook, Boca Raton, FL: CRC Press, 2016.
Sawicki, E . Guide to Apache, Boston, MA: Thomson, 2008.
Schach, S. , Jin, B. , Wright, D. , Heller, G. , and Offutt, A . Maintainability of the Linux Kernel, IEE Proceedings—Software, Vol. 149, Issue 1, pp. 18–23, London: Institute of Engineering and Technology, 2002.
Scheifler, R. , and Gettys, J . X Window System: Core and Extension Protocols: X Version 11, Releases 6 and 6.1, Daytona Beach, FL: Digital Press, 1996.
Schneier, B . Applied Cryptography: Protocols, Algorithms and Source Code in C, Hoboken, NJ: Wiley & Sons, 2015.
Schwartz, M . Linux Job Scheduling, Linux Journal, Vol. 2000, Issue 77, Article 8, 2000.
Sebesta, R . Concepts of Programming Languages, Boston, MA: Addison-Wesley, 2015.
Shoch, J. , Dalal, Y. , Redell, D. , and Crane, R . Evolution of the ethernet local computer network, Computer, Vol. 15, Issue 8, pp.10–27, 1982.
Shotts Jr., W . The Linux Command Line: A Complete Introduction, San Fransico, CA: No Starch Press, 2019.
Siever, E. , Weber, A. , Figgins, S. , Love, R. , and Robbins, A . Linux in a Nutshell: A Desktop Quick Reference, Newton, MA: Riley, 2009.
Silberschatz, A. , Galvin, P. , and Gagne, G . Operating System Concepts, Hoboken, NJ: Wiley & Sons, 2021.
Silva, S . Web Server Administration, Boston, MA: Thomson Course Technology, 2008.
Sloan, J . Network Troubleshooting Tools, Newton, MA: O'Reilly, 2001.
Smalley, S. , and Fraser, T. A. Security Policy Configuration for the Security-Enhanced Linux, NAI Labs Technical Report, 2001.
Smyth, N . Red Hat Enterprise Linux 8 Essentials: Learn to Install, Administer and Deploy RHEL 8 Systems, Cary, NC: Payload, 2019.
Sobell, M . A Practical Guide to Ubuntu Linux, Boston, MA: Pearson, 2015.
Sobell, M . A Practical Guide to Linux Commands, Editors, and Shell Programming, Boston, MA: Addison-Wesley, 2017.
Spafford, E . The internet worm: crisis and aftermath, Communications of the ACM, Vol. 32, Issue 6, pp. 678–687, 1989.
St. Laurent, A . Understanding Open Source and Free Software Licensing, Newton, MA: O'Reilly, 2004.
Stallings, W . Data and Computer Communications, Boston, MA: Pearson, 2013.

Stallings, W . Computer Organization and Architecture, Boston, MA: Pearson, 2015.
Stallings, W . Cryptography and Network Security: Principles and Practices, Boston, MA: Pearson, 2016.
Stallings, W . Operating Systems: Internals and Design Principles, Boston, MA: Pearson, 2017.
Stallings, W. , and Brown, L . Computer Security: Principles and Practices, Boston, MA: Pearson, 2017.
Stallman, R . The GNU Operating System and the Free Software Movement, Open Sources: Voices from the Open Source Revolution, C. DiBona , S. Ockman and M. Stone (eds), Sebastopol, CA: O'Reilly, 1999.
Stallman, R . Why 'Free Software' Is Better than 'Open Source', http://www.gnu.org/philosophy/free-software-for-freedom.html, 2007.
Stallman, R. , and Gay, J . Free Software, Free Society: Selected Essays of Richard M. Stallman, New York: SoHo Books, 2002.
Stankovic, J . Software communication mechanisms: procedure calls versus messages, Computer, Vol. 15, Issue 4, pp. 19–25, 1982.
Statcounter , Operating System Market Share Worldwide, https://gs.statcounter.com/os-market-share.
Stewart, J . and Kinsey, D . Network Security, Firewalls, and VPNs, Burlington, MA: Jones and Bartlett, 2020.
Stubblebine, T . Regular Expression Pocket Reference, Newton, MA: O'Reilly, 2003.
Tam, L. , Glassman, M . and Vandenwauver, M . The psychology of password management: a tradeoff between security and convenience. Behaviour & Information Technology, Vol. 29, Issue 3, pp. 233–244, 2010.
Tanenbaum, A . Structured Computer Organization, Boston, MA: Pearson, 2016.
Tanenbaum, A . Modern Operating Systems, Boston, MA: Pearson, 2016.
Tanenbaum, A . Feamster, N . and Wetherall, D. Computer Networks, Boston, MA: Pearson, 2020.
Tanenbaum, A. , Herder, J. , and Bos, H . Can we make operating systems reliable and secure? Computer, Vol. 39, Issue 5, pp. 44–51, 2006.
Tevanian, A. , Rashid, R. , Golub, D. , Black, Dl , Cooper, E. , and Young, M . Mach Threads and the UNIX Kernel: The Battle for Control, Proceedings of the Summer 1987 USENIX Conference, 1987.
Tirgari, V . Information technology policies and procedures against unstructured data: a phenomenological study of information technology professionals. Journal of Management Information and Decision Sciences, Vol. 15,Issue 2, pp. 87–106, 2012.
Toigo, J . Disaster Recovery Planning: Preparing for the Unthinkable, Hoboken, NJ: Prentice-Hall, 2002.
Tominaga, A. , Nakamura, O. , Teraoka, F. , and Murai, J . Problems and Solutions of DHCP, Proceedings of INET 95, 1995.
Toxen, B . Real World Linux Security: Intrusion Prevention, Detection, and Recovery, Hoboken, NJ: Prentice-Hall, 2003.
Ts'o, T. , and Tweedie, S . Planned Extensions to the Linux Ext2/Ext3 Filesystem, Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, pp. 235–243, 2002.
Ubuntu Documentation Team , Official Ubuntu Documentation, https://help.ubuntu.com/.
Uti, N . Real Time Mobile Video Streaming Using Wavelet Transformation and Run-Length Coding, *Proceedings of the 2009 International Conference on Wireless Networks*, 2 *(ICWN '09)* , pp. 368–373, 2009.
Vacca, R . Computer and Information Security Handbook, San Francisco, CA: Morgan Kaufmann, 2017.
Van Winkle, L . Hands-On Network Programming with C, Birmingham: Packt, 2019.
Vance, N . and Polik, W . Understanding firewalld in Multi-zone Configurations, Linux Journal, Vol. 269, pp. 80–88, 2016.
Vermeulen, S . SELinux Cookbook. Birmingham: Packt, 2014.
Viega, J. , Messier, M . and Chandra, P . Network Security with openSSL: Cryptography for Secure Communications, Newton, MA: O'Reilly, 2002.
Warford, J . Computer Systems, Burlington, MA: Jones and Bartlett, 2016.
Watt, A . Beginning Regular Expressions (Programmer to Programmer), Hoboken, NJ: Wrox, 2005.
Welch, T. A. , A technique for high performance data compression, IEEE Computer, Vol. 17, Issue 6, pp. 8–19, 1984.
Wells, N . The Complete Guide to Linux System Administration, Boston, MA: Thomson Course Technology, 2005.
Wessels, D . Squid: The Definitive Guide, Newton, MA: O'Reilly, 2009.
West, J. , Dean, T . and Andrews, J . Network+Guide to Networks, Boston, MA: Cengage, 2018.
Whitesitt, J . Boolean Algebra and Its Applications, New York: Dover, 2010.
Williams, S . Free as in Freedom: Richard Stallman's Crusade for Free Software, Seattle, WA: CreateSpace, 2009.
Wirzenius, L . (ed). Linux System Administrator's Guide, Linux Documentation Project, http://www.tldp.org/LDP/sag/html/index.html.
Wright, C . Linux Security Module Framework, Proceedings of the Linux Symposium, pp. 604–610, 2002.
Wu, C. , Gerlach, J. , and Young, C . An empirical analysis of open source software developers' motivations and continuance intentions, Information and Management, Vol. 44, pp. 253–262, Amsterdam: Elsevier, 2007.
Ziv, J. , and Lempel, A . Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory, Vol. 24, Issue 5, pp. 530–536, CA: IEEE, 1978.