Cengage

Sixth Edition

CompTIA Linux+ and LPIC-1

Guide to Linux Certification

Jason W. Eckert
triOS College

Networking

Sixth Edition

CompTIA Linux+ and LPIC-1

Guide to Linux Certification

Jason W. Eckert
triOS College

Cengage

Australia • Brazil • Canada • Mexico • Singapore • United Kingdom • United States

CompTIA Linux+ and LPIC-1: Guide to Linux Certification, Sixth Edition

Jason W. Eckert and triOS College

SVP, Product: Erin Joyner

VP, Product: Thais Alencar

Senior Product Director: Mark Santee

Portfolio Product Manager: Natalie Onderdonk

Product Assistant: Ethan Wheel

Learning Designer: Carolyn Mako

Content Manager: Marlena Sullivan

Digital Project Manager: Jim Vaughey

Technical Editor: Danielle Shaw

Developmental Editor: Lisa Ruffolo

VP, Product Marketing: Jason Sakos

Director, Product Marketing: Danaë April

Product Marketing Manager: Mackenzie Paine

Content Acquisition Analyst: Ann Hoffman

Production Service: Straive

Senior Designer: Erin Griffin

Cover Image Source: sollia/Shutterstock.com

Library of Congress Control Number: 2023902115

SE ISBN: 979-8-214-00080-0

LLF ISBN: 979-8-214-00085-5

Cengage
200 Pier 4 Boulevard
Boston MA 02210
USA

Cengage is a leading provider of customized learning solutions. Our employees reside in nearly 40 different countries and serve digital learners in 165 countries around the world. Find your local representative at **www.cengage.com.**

To learn more about Cengage platforms and services, register or access your online learning solution, or purchase materials for your course, visit **www.cengage.com.**

Printed in the United States of America
Print Number: 01 Print Year: 2023

Brief Contents

Table of Contents

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

Chapter 10

Chapter 11

Chapter 12

Chapter 13

Chapter 14

Appendix A

Appendix B

Appendix C

Appendix D

Introduction

> "In a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself."
>
> Eric S. Raymond, The Cathedral and the Bazaar

As Eric S. Raymond reminds us, open source software will continue to shape the dynamics of the computer software industry for the next long while, just as it has for the last three decades. Coined and perpetuated by hackers, the term "open source software" refers to software in which the source code is freely available to anyone who wants to improve it (usually through collaboration). At the heart of the open source software movement lies Linux—an operating system whose rapid growth has shocked the world by demonstrating the nature and power of the open source model.

However, as Linux continues to grow, so must the number of Linux-educated users, administrators, developers, and advocates. We now find ourselves in a time in which Linux education is of great importance to the information technology industry. Key to demonstrating Linux skills and knowledge is the certification process. This book, *Linux+ and LPIC-1 Guide to Linux® Certification,* uses carefully constructed examples, questions, and practical exercises to prepare readers with the necessary information to achieve the latest version of the sought-after Linux+ certification from CompTIA, as well as the latest version of the LPIC-1 certification from the Linux Professional Institute (LPI). Whatever your ultimate goal, you can be assured that reading this book in combination with study, creativity, and practice will make the open source world come alive for you as it has for many others.

Intended Audience

Simply put, this book is intended for those who want to learn the Linux operating system and master the topics tested on the Linux+ certification exam from CompTIA or the LPIC-1 certification exams from LPI. It does not assume any prior knowledge of Linux. Although the topics introduced in this book and associated certification exams are geared toward systems administration, they are also well suited for those who will use or develop programs for Linux systems, or want to pursue a career in cloud computing, supercomputing, cybersecurity, web development, cloud development, or Internet of Things (IoT) technology, where Linux knowledge is a prerequisite.

Chapter Descriptions

Chapter 1, "Introduction to Linux," introduces operating systems as well as the features, benefits, and uses of the Linux operating system. This chapter also discusses the history and development of Linux and open source software, as well as the xrole that Linux plays in the cloud.

Chapter 2, "Linux Installation and Usage," outlines the procedures necessary to prepare for and install Linux on a typical computer system. This chapter also describes how to interact with a Linux system via a terminal and enter basic commands into a Linux shell, such as those used to obtain help and to properly shut down the system.

Chapter 3, "Exploring Linux Filesystems," outlines the Linux filesystem structure and the types of files that you find within it. This chapter also discusses commands you can use to view and edit the content of those files.

Chapter 4, "Linux Filesystem Management," covers the commands you can use to locate and manage files and directories on a Linux filesystem. This chapter also discusses how to link files, as well as interpret and set file and directory permissions, special permissions, and attributes.

Chapter 5, "Linux Filesystem Administration," discusses how to create, mount, and manage filesystems in Linux. This chapter also discusses the files that identify storage devices, the various filesystems available for Linux systems, as well as the partitions and logical volumes used to host filesystems.

Chapter 6, "Linux Server Deployment," introduces the types of hardware and storage configurations used within server environments, as well as the considerations for installing a Linux server. This chapter also discusses device driver support, common problems that may occur during installation, system rescue, as well as configuring Linux server storage (SAN, RAID, ZFS, and BTRFS).

Chapter 7, "Working with the Shell," covers the major features of the BASH and Z shells, including redirection, piping, variables, aliases, and environment files. This chapter also explores the syntax of basic shell scripts and the use of Git to provide version control for shell scripts.

Chapter 8, "System Initialization, X Windows, and Localization," covers the GRUB bootloader that you use to start the Linux kernel. This chapter also discusses how to start daemons during system initialization as well as how to start and stop them afterwards. Finally, this chapter discusses the structure and configuration of Linux graphical user interfaces, as well as setting up time, time zone, and locale information.

Chapter 9, "Managing Linux Processes," covers types of processes as well as how to view their attributes, run them in the background, change their priority, and kill them. This chapter also discusses how to schedule processes to occur in the future using various utilities.

Chapter 10, "Common Administrative Tasks," details three important areas of system administration: printer administration, log file administration, and user administration.

Chapter 11, "Compression, System Backup, and Software Installation," describes utilities that are commonly used to compress and back up files on a Linux filesystem. This chapter also discusses how to install software from source code, how to use the Red Hat Package Manager and the Debian Package Manager, as well as how to install and manage sandboxed applications.

Chapter 12, "Network Configuration," introduces networks, network utilities, and the IP protocol, as well as how to configure the IP protocol on a network interface. This chapter also explains how to set up name resolution, IP routing, network services, and the technologies you can use to administer Linux servers remotely.

Chapter 13, "Configuring Network Services and Cloud Technologies," details the configuration of key infrastructure, web, file sharing, email, and database network services. This chapter also examines how virtualization and containers are used within cloud environments, as well as the configuration and usage of the Docker container runtime and Kubernetes orchestrator.

Chapter 14, "Security, Troubleshooting, and Performance," discusses the utilities and processes used to provide security for Linux systems. This chapter also explores the utilities used to monitor system performance, as well as the troubleshooting procedures for solving performance, hardware, application, filesystem, and network problems.

Additional information is contained in the appendices at the rear of the book. Appendix A discusses the certification process, with emphasis on the Linux+ and LPIC-1 certifications. It also explains how the objective lists for the Linux+ and LPIC-1 certifications match each chapter in the textbook. Appendix B explains how to find Linux resources on the Internet and lists some common resources by category. Appendix C applies the Linux concepts introduced within this book to the macOS operating system from Apple, while Appendix D applies these same concepts to the FreeBSD UNIX operating system.

Features

To ensure a successful learning experience, this book includes the following pedagogical features:

- *Chapter objectives*: Each chapter in this book begins with a detailed list of the concepts to be mastered within the chapter. This list provides you with a quick reference to chapter contents as well as a useful study aid.

- *Illustrations and tables*—Numerous illustrations of server screens and components aid you in visualizing common setup steps, theories, and concepts. In addition, many tables provide details and comparisons of both practical and theoretical information and can be used for a quick review of topics.
- *End-of-chapter material*—The end of each chapter includes the following features to reinforce the material covered in the chapter:
  - *Summary*—A bulleted list gives a brief but complete summary of the chapter.
  - *Key Terms list*—This section lists all new terms in the chapter. Definitions for each key term can be found in the Glossary.
  - *Review Questions*—A list of review questions tests your knowledge of the most important concepts covered in the chapter.
  - *Hands-On Projects*—Hands-On Projects help you to apply the knowledge gained in the chapter.
  - *Discovery Exercises*—Additional projects guide you through real-world scenarios and advanced topics.

New to This Edition

This edition has been updated to include the concepts and procedures tested on the latest certification exams from CompTIA and the Linux Professional Institute (LPI). More specifically, this edition contains:

- Content that maps to the latest CompTIA Linux+ (XK0-005) and LPIC-1: System Administrator (101-500, 102-500) certification exams
- Updated information pertinent to the latest Linux distributions and technologies
- New and expanded material related to security tools and configuration, Git, cloud technologies, Docker containers, and Kubernetes
- Continued focus on Linux server administration, including key network services (FTP, NFS, Samba, Apache, DNS, DHCP, NTP, Postfix, SSH, VNC, and SQL)
- Hands-On Projects that configure modern Fedora and Ubuntu Linux, using steps that will work on either a Windows or macOS PC (Intel or Apple Silicon)
- A GitHub resource that outlines modifications to Hands-On Project steps that result from changes made by future Linux releases to ensure that you can always use the latest versions of Fedora and Ubuntu
- Appendices that map the latest Linux+ and LPIC-1 certification objectives to each chapter, identify key Linux-related Internet resources, and apply the Linux administration topics covered within the book to Apple macOS and FreeBSD UNIX

Text and Graphic Conventions

Wherever appropriate, additional information and exercises have been added to this book to help you better understand what is being discussed in the chapter. Special boxes throughout the text alert you to additional materials:

Note

The Note box is used to present additional helpful material related to the subject being described.

Hands-On Projects

The Hands-On Projects box indicates that the projects it contains give you a chance to practice the skills you learned in the chapter and acquire hands-on experience.

Discovery Exercises

The Discovery Exercises box contains exercises that guide you to explore real-world scenarios and advanced topics.

MindTap

MindTap for *CompTIA Linux+ and LPIC-1: Guide to Linux Certification* is an online learning solution designed to help you master the skills needed in today's workforce. Research shows employers need critical thinkers, troubleshooters, and creative problem-solvers to stay relevant in our fast-paced, technology-driven world. MindTap helps you achieve this with assignments and activities that provide hands-on practice, real-life relevance, and mastery of difficult concepts. Students are guided through assignments that progress from basic knowledge and understanding to more challenging problems. MindTap activities and assignments are tied to learning objectives. MindTap features include the following:

- ***Live Virtual Machine labs*** allow you to practice, explore, and try different solutions in a safe sandbox environment. Each module provides you with an opportunity to complete an in-depth project hosted in a live virtual machine environment. You implement the skills and knowledge gained in the chapter through real design and configuration scenarios in a private cloud created with OpenStack.
- ***Simulations*** allow you to apply concepts covered in the chapter in a step-by-step virtual environment and receive immediate feedback.
- ***Linux for Life*** assignments encourage you to stay current with what's happening in the Linux field.
- ***Reflection*** activities encourage classroom and online discussion of key issues covered in the chapters.
- ***Pre- and Post-Quizzes*** assess your understanding of key concepts at the beginning and end of the course and emulate the CompTIA Linux+ XK0-005 and LPIC-1 101-500 and 102-500 Certification Exam.

Instructors, MindTap is designed around learning objectives and provides analytics and reporting so you can easily see where the class stands in terms of progress, engagement, and completion rates. Use the content and learning path as is or pick and choose how your materials will integrate with the learning path. You control what the students see and when they see it. Learn more at https://www.cengage.com/mindtap/.

Instructor Resources

Instructors, please visit cengage.com and sign in to access instructor-specific resources, which include the instructor manual, solutions manual, and PowerPoint presentations.

- *Instructor's Manual*—The Instructor's Manual that accompanies this book includes additional instructional material to assist in class preparation, including items such as overviews, chapter objectives, teaching tips, quick quizzes, class discussion topics, additional projects, additional resources, and key terms. A sample syllabus is also available.

- *Test bank*—Cengage Testing Powered by Cognero is a flexible, online system that allows you to do the following:
  - Author, edit, and manage test bank content from multiple Cengage solutions.
  - Create multiple test versions in an instant.
  - Deliver tests from your LMS, your classroom, or wherever you want.
- *PowerPoint presentations*—This book provides PowerPoint slides to accompany each chapter. Slides can be used to guide classroom presentations, to make available to students for chapter review, or to print as classroom handouts. Files are also supplied for every figure in the book. Instructors can use these files to customize PowerPoint slides, illustrate quizzes, or create handouts.
- *Solution and Answer Guide*—Solutions to all end-of-chapter review questions and projects are available.

Student Resources

MindTap for Linux+ and LPIC-1 Guide to Linux Certification, Sixth Edition, helps you learn on your terms.

- Instant access in your pocket: Take advantage of the MindTap Mobile App to learn on your terms. Read or listen to textbooks and study with the aid of instructor notifications, flashcards, and practice quizzes.
- MindTap helps you create your own potential. Gear up for ultimate success: Track your scores and stay motivated toward your goals. Whether you have more work to do or are ahead of the curve, you'll know where you need to focus your efforts. The MindTap Green Dot will charge your confidence along the way.
- MindTap helps you own your progress. Make your textbook yours; no one knows what works for you better than you. Highlight key text, add notes, and create custom flashcards. When it's time to study, everything you've flagged or noted can be gathered into a guide you can organize.

About the Author

Jason W. Eckert is an experienced technical trainer, consultant, and best-selling author in the Information Technology (IT) industry. With over three decades of IT experience, 45 industry certifications, 4 published apps, and 25 published textbooks covering topics such as UNIX, Linux, Security, Windows Server, Microsoft Exchange Server, PowerShell, BlackBerry Enterprise Server, and Video Game Development, Mr. Eckert brings his expertise to every class that he teaches at triOS College, and to his role as the Dean of Technology. He was named 2019 Outstanding Train-the-Trainer from the Computing Technology Industry Association (CompTIA) for his work providing Linux training to other educators. For more information about Mr. Eckert, visit jasoneckert.net.

Acknowledgments

Firstly, I would like to thank the staff at Cengage for an overall enjoyable experience writing a textbook on Linux that takes a fundamentally different approach than traditional textbooks. Additionally, I wish to thank Lisa Ruffolo, Danielle Shaw, Marlena Sullivan, Carolyn Mako, and Natalie Onderdonk for working extremely hard to pull everything together and ensure that the book provides a magnificent Linux experience. I also wish to thank Frank Gerencser of triOS College for providing me, over the past two decades, with the opportunity to write six editions of a book on a topic about which I'm very passionate. Finally, I wish to thank the Starbucks Coffee Company for keeping me ahead of schedule, and my dog Pepper for continually reminding me that taking a break is always a good idea. Readers are encouraged to email comments, questions, and suggestions regarding *Linux+ and LPIC-1 Guide to Linux® Certification* to Jason W. Eckert: jason.eckert@trios.com.

Dedication

This book is dedicated to my grandson Chase, who was no help at all.

Before You Begin

Linux can be a large and intimidating topic if studied in a haphazard way. So, as you begin your study of Linux, keep in mind that each chapter in this book builds on the preceding one. To ensure that you gain a solid understanding of core Linux concepts, read the chapters in consecutive order and complete the Hands-On Projects. You should also participate in a local Linux Users Group (LUG) and explore the Internet for websites, forums, YouTube videos, and social media articles that will expand your knowledge of Linux.

Lab Requirements

The following hardware is required at minimum for the Hands-On Projects at the end of each chapter:

- A Windows or macOS PC with a 64-bit Intel, AMD, ARM, or Apple Silicon CPU
- 8 GB RAM (16 GB RAM or greater recommended)
- 500 GB hard disk or SSD (SSD strongly recommended)
- A network that provides Internet access

Similarly, the following lists the software required for the Hands-On Projects at the end of each chapter:

- Latest version of Fedora Workstation installation media
- Latest version of Ubuntu Server installation media

Chapter 1

Introduction to Linux

Chapter Objectives

1. Explain the purpose of an operating system.
2. Outline the key features of the Linux operating system.
3. Describe the origins of the Linux operating system.
4. Identify the characteristics of various Linux distributions and where to find them.
5. Explain the common uses of Linux in industry today.
6. Describe how Linux is used in the cloud.

Linux technical expertise is essential in today's computer workplace as more and more companies switch to Linux to meet their computing needs. Thus, it is important to understand how Linux can be used, what benefits Linux offers to a company, and how Linux has developed and continues to develop. In the first half of this chapter, you learn about operating system terminology and features of the Linux operating system, as well as the history and development of Linux. Later in this chapter, you learn about the various types of Linux, as well as the situations and environments in which Linux is used. Finally, you explore the ways that Linux can be hosted in the cloud, as well as the process and technologies used to add web apps to cloud-based Linux systems.

Operating Systems

Every computer has two fundamental types of components: hardware and software. You are probably familiar with these terms, but it's helpful to review their meanings so you can more easily understand how Linux helps them work together.

Hardware consists of the physical components inside a computer that are electrical in nature; they contain a series of circuits that manipulate the flow of information. A computer can contain many different pieces of hardware, including the following:

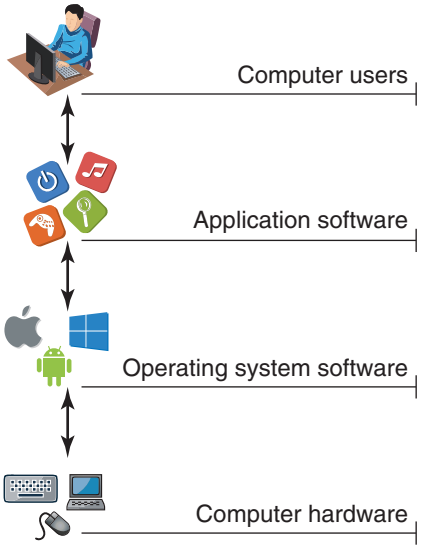
- A processor (also known as the central processing unit or CPU), which computes information
- Physical memory (also known as random access memory or RAM), which stores information needed by the processor
- Hard disk drives (HDDs) and solid state drives (SSDs), which store most of the information that you use
- CD/DVD drives, which read and write information to and from CD/DVD discs

- Flash memory card readers, which read and write information to and from removable memory cards, such as Secure Digital (SD) cards
- Sound cards, which provide audio to external speakers
- Video cards (also known as graphics processing units or GPUs), which display results to the computer monitor
- Network adapter cards, which provide access to wired and wireless (Wi-Fi or Bluetooth) networks
- Ports (such as USB, eSATA, GPIO, and Thunderbolt), which provide access to a wide variety of external devices including keyboards, mice, printers, and storage devices
- Mainboards (also known as motherboards), which provide the circuitry (also known as a bus) for interconnecting all other components

Software, on the other hand, refers to the sets of instructions or **programs** that allow the hardware components to manipulate data (or files). When a bank teller types information into the computer behind the counter at a bank, for example, that bank teller is using a program that understands what to do with your bank records. Programs and data are usually stored on hardware media, such as hard disk drives or SSDs, although they can also be stored on removable media or even embedded in computer chips. These programs are loaded into parts of your computer hardware (such as your computer's memory and processor) when you first turn on your computer and when you start additional software, such as word processors or Internet browsers. After a program is executed on your computer's hardware, that program is referred to as a **process**. In other words, a program is a file stored on your computer, whereas a process is that file in action, performing a certain task.

There are two types of programs. The first type, **applications (apps)**, includes those programs designed for a specific use and with which you commonly interact, such as word processors, computer games, graphical manipulation programs, and computer system utilities. The second type, **operating system** software, consists of a set of software components that control the hardware of your computer. Without an operating system, you would not be able to use your computer. Turning on a computer loads the operating system into computer hardware, which then loads and centrally controls all other application software in the background. At this point, the **user** (the person using the computer) is free to interact with the applications, perhaps by typing on the keyboard or clicking a mouse. Applications take the information the user supplies and relay it to the operating system. The operating system then uses the computer hardware to carry out the requests. The relationship between users, application software, operating system software, and computer hardware is illustrated in Figure 1-1.

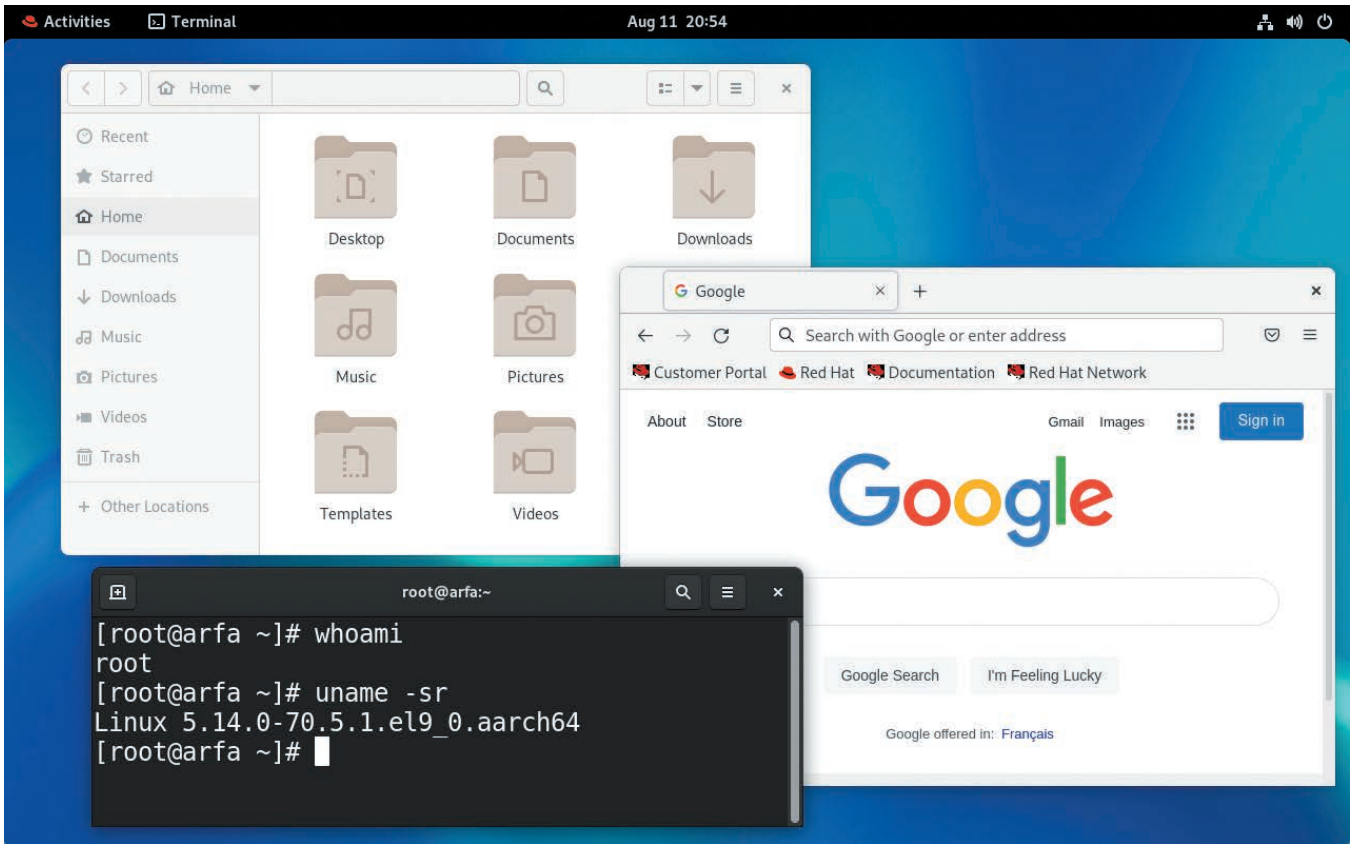
Figure 1-1 The role of operating system software



The operating system carries out many tasks by interacting with different types of computer hardware. For the operating system to accomplish the tasks, it must contain the appropriate device driver software for every hardware device in your computer. Each **device driver** tells the operating system how to use that specific device. The operating system also provides a **user interface**, which is a program that accepts user input indicating what to do, forwards this input to the operating system for completion, and, after it is completed, gives the results back to the user. The user interface can be a command-line prompt, in which the user types commands, or it can be a **graphical user interface (GUI)**, which consists of menus, dialog boxes, and symbols (known as icons) that the user can interact with via the keyboard or the mouse. A typical Linux GUI that also provides a command-line interface via an app is shown in Figure 1-2.

Finally, operating systems offer **services**, which are applications that handle system-related tasks, such as printing, scheduling programs, and network access. These services determine most of the functionality in an operating system. Different operating systems offer different services, and many operating systems allow users to customize the services they offer.

Figure 1-2 A Linux graphical and command-line user interface



The Linux Operating System

Linux (pronounced "Lih-nucks") is an operating system that is used today to run applications on a variety of hardware. Similar to other operating systems, the Linux operating system loads into computer memory when you first power on your computer and initializes (or activates) all of the hardware components. Next, it loads the programs that display the interface. From within the interface, you can execute commands that tell the operating system and other applications to perform specific tasks. The operating system then uses the computer hardware to perform the tasks required by the applications.

Versions of the Linux Operating System

The core component of the Linux operating system is called the Linux **kernel**. The Linux kernel and supporting software (called function libraries) are written almost entirely in the C programming language, which is one of the most common languages that software developers use when creating programs.

Although a variety of software can be used to modify the appearance of Linux, the underlying kernel is common to all types of Linux. The Linux kernel is developed continuously; thus, you should understand the different version numbers of the Linux kernel to decide which kernel version is appropriate for your needs. Because the Linux kernel is directly responsible for controlling the computer's hardware (via device drivers), you might sometimes need to upgrade the Linux kernel after installing Linux to take advantage of new technologies or to fix problems (also known as bugs) related to your computer's hardware. Consequently, a good understanding of your system's hardware is important in deciding which kernel to use.

Note 1

For a complete list of kernels, kernel versions, and their improvements, see kernel.org.

In some cases, you can use updates in the form of a kernel module or a kernel patch to provide or fix hardware supported by the kernel. Kernel modules and kernel patches are discussed later in this book.

Identifying Kernel Versions

Linux kernel versions are made up of the following three components:

- Major number
- Minor number
- Revision number

Let's look at a sample Linux kernel version, 5.18.16. In this example, the **major number** is the number 5, which indicates the major version of the Linux kernel. The **minor number**, represented by the number 18, indicates the minor revision of the Linux kernel. As new features are added to the Linux kernel over time, the minor number is incremented. The major number is usually incremented when a major kernel feature is implemented, when the minor number versioning reaches a high number, or to signify a major event; for example, the 3.0 kernel was introduced to commemorate the twentieth anniversary of Linux.

Linux kernel changes occur frequently. Those changes that are very minor are represented by a **revision number** indicating the most current changes to the version of the particular kernel that is being released. For example, a 5.18.16 kernel has a revision number of 16. This kernel is the sixteenth release of the 5.18 kernel. Some kernels have over 100 revisions as a result of developers making constant improvements to the kernel code.

Note 2

Sometimes, a fourth number is added to a kernel version to indicate a critical security or bug patch. For example, a 5.18.16.1 kernel is a 5.18.16 kernel with a critical patch number of 1.

Modern Linux kernels that have a major, minor, and revision number are referred to as **production kernels**; they have been thoroughly tested by several Linux developers and are declared stable. **Development kernels** are not fully tested and imply instability; they are tested for vulnerabilities by people who develop Linux software. Most development kernels append the minor number with the letters -rc (release candidate) followed by a number that represents the version of the development kernel. For example, the 5.19-rc8 development kernel is the eighth release candidate for the 5.19 kernel; if Linux developers declare it stable after being thoroughly tested, it will become the 5.19.0 production kernel.

Note 3

Until Linux kernel 2.6.0, an odd-numbered minor number was used to denote a development kernel, and an even-numbered minor number was used to denote a production kernel.

Note 4

When choosing a kernel for a mission-critical computer such as a server, ensure that you choose a production kernel. This reduces the chance that you will encounter a bug in the kernel, which saves you the time needed to change kernels.

Table 1-1 shows some sample kernel versions released since the initial release of Linux.

Table 1-1 Sample Linux Kernel version history

Kernel Version	Date Released	Type
0.01	September 1991	First Linux kernel
0.12	January 1992	Production (stable)
0.99.15	March 1994	Development
1.0.8	April 1994	Production (stable)
1.3.100	May 1996	Development
2.0.36	November 1998	Production (stable)
2.3.99	May 2000	Development
2.4.17	December 2001	Production (stable)
2.5.75	July 2003	Development
2.6.35	August 2010	Production (stable)
3.0.0	July 2011	Production (stable)
3.15.10	August 2014	Production (stable)
4.0.0	April 2015	Production (stable)
4.12.5	August 2017	Production (stable)
5.0.0	March 2019	Production (stable)
5.19-rc8	July 2022	Development

Licensing Linux

Companies often choose Linux as their operating system because of the rules governing Linux licensing. Unlike most other operating systems, Linux is freely developed and continuously improved by a large community of software developers. For this reason, it is referred to as **open source software (OSS)**.

To understand OSS, you must first understand how source code is used to create programs. **Source code** refers to the list of instructions that a software developer writes to make up a program; an example of source code is shown in Figure 1-3.

Figure 1-3 Source code

```
#define MODULE
#include <linux/module.h>
int init_module(void){
        printk("My module has been activated.\n");
        return 0;
}
void cleanup_module(void){
        printk("My module has been deactivated.");
}
```

After the software developer finishes writing the instructions, the source code is compiled into a format that your computer's processor can understand and execute directly or via an interpreter program. To edit an existing program, the software developer must edit the source code and then recompile it.

The format and structure of source code follows certain rules defined by the **programming language** in which it was written. Programmers write Linux source code in many programming languages. After being compiled, all programs look the same to the computer operating system, regardless of the programming language in which they were written. As a result, software developers choose a programming language to create source code based on ease of use, functionality, and comfort level.

The fact that Linux is an open source operating system means that software developers can read other developers' source code, modify that source code to make the software better, and redistribute that source code to other developers who might improve it further. Like all OSS, Linux source code must be distributed free of charge, regardless of the number of modifications made to it. People who develop OSS commonly use the Internet to share their source code, manage software projects, and submit comments and fixes for bugs (flaws). In this way, the Internet acts as the glue that binds together Linux developers in particular and OSS developers in general.

Note 5

To read the complete open source definition, visit opensource.org.

Note 6

OSS is also called free and open source software (FOSS) or free/libre and open source software (FLOSS).

Here are some implications of the OSS way of developing software:

- Software is developed rapidly through widespread collaboration.
- Software bugs (errors) are noted and promptly fixed.
- Software features evolve quickly, based on users' needs.
- The perceived value of the software increases because it is based on usefulness and not on price.

As you can imagine, the ability to share ideas and source code is beneficial to software developers. However, a software company's business model changes drastically when OSS enters the picture. The main issue is this: How can a product that is distributed freely generate revenue? After all, without revenue any company will go out of business.

The OSS process of software development was never intended to generate revenue directly. Its goal was to help people design better software by eliminating many of the problems associated with traditional software development, which is typically driven by predefined corporate plans and rigid schedules. By contrast, OSS development relies on the unsystematic contributions of several software developers that have a direct need for the software that they are developing. While this process may seem haphazard, it ensures that software is constantly developed to solve real-world needs. Because the source code is scrutinized and improved by different developers, OSS often solves problems in the best possible way. Each developer contributes their strengths to a project, while learning new techniques from other developers at the same time.

By leveraging existing OSS, organizations can develop software much faster than they could otherwise. A typical software app today contains dozens or hundreds of well-designed OSS software components that the organization's developers didn't have to create themselves. To support this ecosystem, organization developers contribute improvements to existing OSS, as well as create and maintain OSS software components that they'd like other developers to evolve through contributions. In short, OSS saves companies a large amount of time and money when developing software.

Other companies make money by selling computer hardware that runs OSS, by selling customer support for OSS, or by creating **closed source software** programs that run on open source products such as Linux.

The OSS development process is, of course, not the only way to develop and license software. Table 1-2 summarizes the types of software you are likely to encounter. The following section explains these types in more detail.

Table 1-2 Software types

Type	Description
Open source	Software in which the source code and software can be obtained free of charge and optionally modified to suit a particular need
Closed source	Software in which the source code is not available; although this type of software might be distributed free of charge, it is usually quite costly and commonly referred to as commercial software
Freeware	Closed source software that is given out free of charge; it is sometimes referred to as freemium software
Shareware	Closed source software that is initially given out free of charge but that requires payment after a certain period of use

Types of Open Source Licenses

Linux adheres to the **GNU General Public License (GPL)**, which was developed by the **Free Software Foundation (FSF)**. The GPL stipulates that the source code of any software published under its license must be freely available. If someone modifies that source code, that person must also redistribute that source code freely, thereby keeping the source code free forever.

Note 7

"GNU" stands for "GNUs Not UNIX."

Note 8

The GPL is freely available at gnu.org.

The GPL is an example of a **copyleft license** as it contains strict restrictions on how the source code can be used in derivative software. To encourage adoption in software projects, most OSS instead use a **permissive license** that contains far fewer restrictions. The BSD, MIT, and Apache licenses are examples of permissive open source licenses.

Note 9

For a list of copyleft and permissive open source licenses, visit opensource.org.

Types of Closed Source Licenses

Closed source software can be distributed for free or for a cost; either way, the source code for the software is unavailable from the original developers. The majority of closed source software is sold commercially and bears the label of its manufacturer. Each of these software packages can contain a separate license that restricts free distribution of the program and its source code in many ways.

Note 10

Examples of closed source software are software created by companies such as Microsoft, Apple, and Electronic Arts (EA).

Another type of closed source software is **freeware**, in which the software program is distributed free of charge, yet the source code is unavailable. Freeware might also contain licenses that restrict the distribution of source code. Another approach to this style of closed source licensing is **shareware**, which is distributed free of charge, yet after a certain number of hours of usage or to gain certain features of the program, payment is required. Although freeware and shareware do not commonly distribute their source code under an open source license, some people incorrectly refer to them as OSS, assuming that the source code is freely shared as well.

Linux Advantages

The main operating systems in use today include Linux, Microsoft Windows, UNIX, and macOS. Notably, Linux is the fastest growing operating system released to date. Although Linux was only created in 1991, the number of Linux users estimated by Red Hat in 1998 was 7.5 million, and the number of Linux users estimated by Google in 2010 was over 40 million (including the number of Linux-based Android smartphone and device users). In 2013, LinuxCounter.net estimated that the number of Linux users was over 70 million, and Google estimated that over 900 million Linux-based Android devices had shipped by then. In 2018, Linux ran on the top 500 supercomputers, 60 percent of the computers purchased by schools in North America (due to widespread adoption of Linux-based Chromebooks), as well as nearly all web servers and Internet-connected hardware devices. In 2021, Gartner estimated that nearly half of the global population (3.5 billion people) were Linux users, whether they realized it or not.

Organizations have adopted Linux for many reasons. The following advantages are examined in the sections that follow:

- Risk reduction
- Meeting business needs
- Stability and security
- Support for different hardware
- Ease of customization
- Ease of obtaining support
- Cost reduction

Risk Reduction

Companies need software to perform mission-critical tasks, such as managing business operations and providing valuable services to customers over the Internet. However, changes in customer needs and market competition can cause the software a company uses to change frequently. Keeping the software up to date can be costly and time-consuming but is a risk that companies must take. Imagine that a fictitious company, ABC Inc., buys a piece of software from a fictitious software vendor, ACME Inc., to integrate its sales and accounting information with customers via the Internet. What would happen if ACME went out of business or stopped supporting the software due to lack of sales? In either case, ABC would be using a product that had no software support, and any problems that ABC had with the software after that time would go unsolved and could result in lost revenue. In addition, all closed source software is eventually retired after it is purchased, forcing companies to buy new software every so often to obtain new features and maintain software support.

If ABC instead chose to use an OSS product and the original developers became unavailable to maintain it, then ABC would be free to take the source code, add features to it, and maintain it themselves provided the source code was redistributed free of charge. Also, most OSS does not retire after a short period of time because collaborative open source development results in constant software improvement geared to the needs of the users.

Meeting Business Needs

Recall that Linux is merely one product of open source development. Many thousands of OSS programs are available, and new ones are created daily by software developers worldwide. Most open source Internet tools have been developed for quite some time now, and the focus in the Linux community in the past few years has been on developing application software, cloud technologies, and security-focused network services that run on Linux. Almost all of this software is open source and freely available, compared to other operating systems, in which most software is closed source and costly.

OSS is easy to locate on the web, at sites such as SourceForge (sourceforge.net), GitHub (github.com), GitLab (gitlab.com), and GNU Savannah (savannah.gnu.org). New software is published to these sites daily. SourceForge alone hosts over 500,000 software development projects.

Common software available for Linux includes but is not limited to the following:

- Scientific and engineering software
- Software emulators
- Web servers, web browsers, and e-commerce suites
- Desktop productivity software (e.g., word processors, presentation software, spreadsheets)
- Graphics manipulation software
- Database software
- Security software

In addition, companies that run the UNIX operating system (including macOS, which is a flavor of UNIX) might find it easy to migrate to Linux. For those companies, Linux supports most UNIX commands and standards, which eases a transition to Linux because the company likely would not need to purchase additional software or retrain staff. For example, suppose a company that tests scientific products has spent time and energy developing custom software that runs on the UNIX operating system. If this company transitioned to another operating system, its staff would need to be retrained or hired, and much of the custom software would need to be rewritten and retested, which could result in a loss of customer confidence. If, however, that company transitions to Linux, the staff would require little retraining, and little of the custom software would need to be rewritten and retested, hence saving money and minimizing impact on consumer confidence.

Companies that need to train staff on Linux usage and administration can take advantage of several educational resources and certification exams for various Linux skill levels. Certification benefits as well as the CompTIA Linux+ and LPIC-1 certifications are discussed in this book's Appendix A, "Certification."

In addition, for companies that require a certain development environment or need to support custom software developed in the past, Linux provides support for nearly all programming languages and software development frameworks.

Stability and Security

OSS is developed by people who have a use for it. This collaboration among software developers with a common need speeds up the software creation, and when bugs in the software are found, bug fixes are created quickly. Often, the users who identify the bugs can fix the problem because they have the source code, or they can provide detailed descriptions of their problems so that other developers can fix them.

By contrast, customers using closed source operating systems must rely on the operating system vendor to fix any bugs. Users of closed source operating systems must report the bug to the manufacturer and wait for the manufacturer to develop, test, and release a bug fix. This process might take weeks or even months, which is slow and costly for most companies and individuals. The thorough and collaborative open source approach to testing software and fixing software bugs increases the stability of Linux; it is not uncommon to find a Linux system that has been running continuously for months or even years without being turned off.

Security, a vital concern for most companies and individuals, is another Linux strength. Because Linux source code is freely available and publicly scrutinized, security loopholes are quickly identified and fixed by developers. In contrast, the source code for closed source operating systems is not released to the public for scrutiny, which means customers must rely on the operating system vendor to detect and fix security loopholes. A security loophole unnoticed by the vendor can be exploited by the wrong person. Every day, new malicious software (such as viruses and malware) is unleashed on the Internet with the goal of infiltrating operating systems and other software. However, most malicious software targets closed source operating systems and software. As of April 2008, Linux had fewer than 100 known viruses, whereas Windows had more than 1,000,000 known viruses. Compared to other systems, the amount of malicious software for Linux systems remains incredibly low.

Note 11

For a list of recent malicious software, visit cisecurity.org.

Support for Different Hardware

Another important feature of Linux is that it can run on a wide variety of hardware. Although Linux is most commonly installed on workstations and server systems that use an Intel CPU platform, it can also be installed on supercomputers that use an IBM POWER CPU or high-performance cloud servers that use an ARM CPU, such as Amazon's Graviton EC2. Linux can also be installed on other systems, such as point-of-sale terminals and sales kiosks that use an Intel CPU, as well as small custom hardware devices that use an ARM, MIPS, or RISC-V CPU. Small hardware devices that can connect to the Internet are collectively called the **Internet of Things (IoT)**.

The open source nature of Linux combined with the large number of OSS developers at work today makes Linux an attractive choice for manufacturers of mobile, custom, and IoT devices. NASA spacecrafts, Internet routers and firewalls, Google Android smartphones and tablets, Amazon Kindle eBook readers, GPS navigation systems, smart speakers, home automation equipment, and Wi-Fi access points all run Linux.

Few other operating systems run on more than two CPU platforms, making Linux the ideal choice for companies that use and support many different types of hardware. Here is a partial list of CPU platforms on which Linux can run:

- Intel x86/x86_64 (also implemented by AMD)
- ARM/ARM64
- MIPS/MIPS64
- RISC-V
- PPC (PowerPC, POWER)
- Mainframe (S/390, z/Architecture)

Ease of Customization

The ease of controlling the inner workings of Linux is another attractive feature, particularly for companies that need their operating system to perform specialized functions. If you want to use Linux as a web server, you can recompile the Linux kernel to include only the support needed to be a web server. This results in a much smaller and faster kernel.

Note 12

A small kernel performs faster than a large kernel because it contains less code for the processor to analyze. On high performance systems, you should remove any unnecessary features from the kernel to improve performance.

Today, customizing and recompiling the Linux kernel is a well-documented and easy process; however, it is not the only way to customize Linux. Only software packages necessary to perform certain tasks need to be installed; thus, each Linux system can have a unique configuration and set of applications available to the user. Linux also supports several system programming languages, such as shell and PERL, which you can use to automate tasks or create custom tasks.

Consider a company that needs an application to copy a database file from one computer to another computer, yet also needs to manipulate the database file (perhaps by checking for duplicate records), summarize the file, and finally print it as a report. This might seem like a task that would require expensive software; however, in Linux, you can write a short shell script that uses common Linux commands and programs to perform these tasks. This type of customization is invaluable to companies because it allows them to combine several existing applications to perform a certain task, which might be specific only to that company and, hence, not previously developed by another free software developer. Most Linux configurations present hundreds of small utilities, which, when combined with shell or PERL programming, can make new programs that meet many business needs.

Ease of Obtaining Support

For those who are new to Linux, the Internet offers a world of Linux documentation. A search of the word "Linux" on a typical Internet search engine such as google.com displays thousands of results, including Linux-related guides, information portals, and video tutorials.

In addition, several Internet **forums** allow Linux users to post messages and reply to previously posted messages. If you have a specific problem with Linux, you can post your problem on an Internet forum and receive help from those who know the solution. Linux forums are posted to frequently; thus, you can usually expect a solution to a problem within hours. You can find Linux-related forums on several different websites, including linux.org, linuxquestions.org, facebook.com (called groups), discord.com (called servers), and reddit.com (called subreddits).

Note 13

Appendix C, "Finding Linux Resources on the Internet," describes how to navigate Internet resources and lists some resources that you might find useful.

Although online support is the typical method of getting help, other methods are available, including **Linux User Groups (LUGs)**. LUGs are groups of Linux users who meet regularly to discuss Linux-related issues and problems. An average LUG meeting consists of several new Linux users (also known as Linux newbies), administrators, developers, and experts (also known as Linux gurus). LUG meetings are a resource to solve problems and learn about the local Linux community. Most LUGs host websites that contain a multitude of Linux resources, including summaries of past meetings and discussions. One common activity seen at a LUG meeting is referred to as an Installfest; several members bring in their computer equipment to install Linux and other Linux-related software. This approach to transferring knowledge is very valuable to LUG members because concepts can be demonstrated and the solutions to problems can be modeled by more experienced Linux users.

Note 14

To find a list of available LUGs in your region, search for the words "LUG cityname" on an Internet search engine such as google.com (substituting your city's name for "*cityname*"). When searching for a LUG, keep in mind that LUGs might go by several different names; for example, the LUG in the Kitchener-Waterloo area of Ontario, Canada is known as KW-LUG (Kitchener-Waterloo Linux Users Group). Many LUGs today are managed using custom websites, Facebook groups, or meeting sites such as meetup.com.

Cost Reduction

Linux is less expensive than other operating systems such as Windows because there is no cost associated with acquiring the software. In addition, a wealth of OSS can run on different hardware platforms running Linux, and a large community of developers is available to diagnose and fix bugs in a short period of time for free. While Linux and the Linux source code are distributed freely, implementing Linux is not cost free. Costs include purchasing the computer hardware necessary for the computers hosting Linux, hiring people to install and maintain Linux, and training users of Linux software.

The largest costs associated with Linux are the costs associated with hiring people to maintain the Linux system. However, closed source operating systems have this cost in addition to the cost of the operating system itself. The overall cost of using a particular operating system is known as the **total cost of ownership (TCO)**. Table 1-3 shows an example of the factors involved in calculating the TCO for operating systems.

Table 1-3 Calculating the total cost of ownership

Costs	Linux	Closed Source Operating System
Operating system cost	$0	Greater than $0
Cost of administration	Low: Stability is high and bugs are fixed quickly by open source developers.	Moderate/high: Bug fixes are created by the vendor of the operating system, which could result in costly downtime.
Cost of additional software	Low/none: Most software available for Linux is also open source.	Moderate/high: Most software available for closed source operating systems is also closed source.
Cost of software upgrades	Low/none	Moderate/high: Closed source software is eventually retired, and companies must buy upgrades or new products to gain functionality and stay competitive.

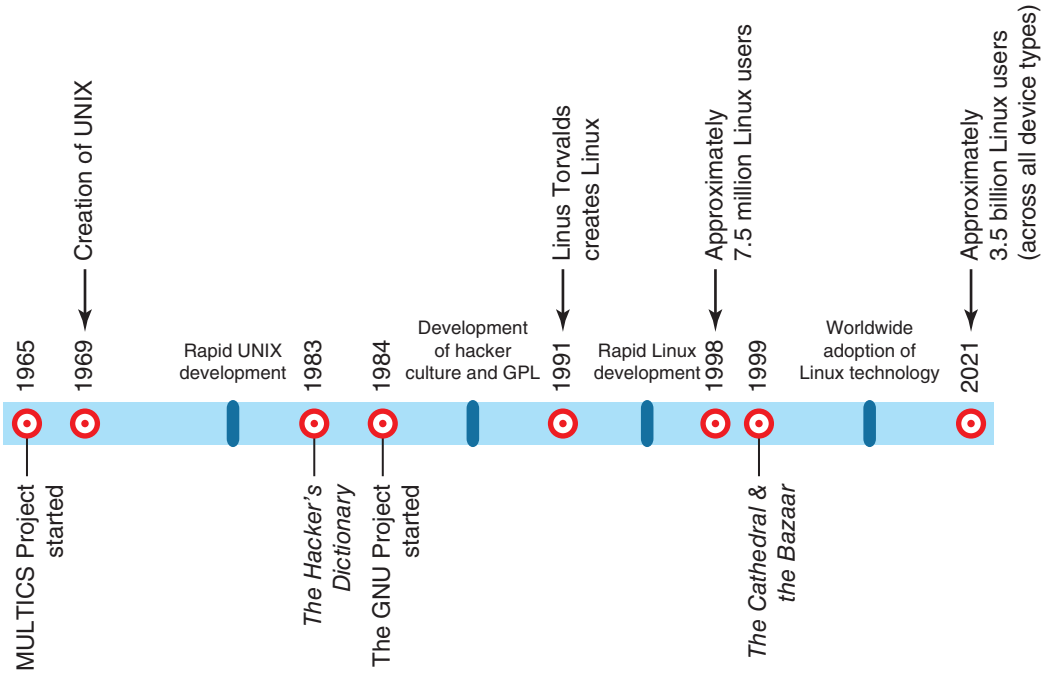
The History of Linux

Linux is based on the UNIX operating system developed by Ken Thompson and Dennis Ritchie of AT&T Bell Laboratories in 1969 and was developed through the efforts of many people as a result of the hacker culture that formed in the 1980s. Therefore, to understand how and why Linux emerged on the operating system market, you must first understand UNIX and the hacker culture. Figure 1-4 illustrates a timeline representing the history of the UNIX and Linux operating systems.

UNIX

The UNIX operating system has roots running back to 1965, when the Massachusetts Institute of Technology (MIT), General Electric, and AT&T Bell Laboratories began developing an operating system called **Multiplexed Information and Computing Service (MULTICS)**. MULTICS was a test project intended to reveal better ways of developing time-sharing operating systems, in which the operating system regulates the amount of time each process has to use the processor. The project was abandoned in 1969. However, Ken Thompson, who had worked on the MULTICS operating system, continued to experiment with operating systems. In 1969, he developed an operating system called **UNIX** that ran on the DEC (Digital Equipment Corporation) PDP-7 computer.

Figure 1-4 Timeline of UNIX and Linux development



Shortly thereafter, Dennis Ritchie invented the C programming language that was used on Ken Thompson's UNIX operating system. The C programming language was a revolutionary language. Most programs at the time needed to be written specifically for the hardware of the computer, which involved referencing volumes of information regarding the hardware in order to write a simple program. However, the C programming language was much easier to use to write programs, and it was possible to run a program on different machines without having to rewrite the code. The UNIX operating system was rewritten in the C programming language, and by the late 1970s, the UNIX operating system ran on different hardware platforms, something that the computing world had never seen until that time. Hence, people called UNIX a portable operating system.

Unfortunately, the company Ken Thompson and Dennis Ritchie worked for (AT&T) was restricted by a federal court order from marketing UNIX. In an attempt to keep UNIX viable, AT&T sold the UNIX source code to several companies, encouraging them to agree to standards among them. Each of these companies developed its own variety, or **flavor**, of UNIX yet adhered to standards agreed upon by all. AT&T also gave free copies of the UNIX source code to certain universities to promote widespread development of UNIX. One result was a UNIX version developed at the University of California, Berkeley in the early 1980s known as BSD (Berkeley Software Distribution). In 1982, one of the companies to whom AT&T sold UNIX source code (Sun Microsystems) marketed UNIX on relatively inexpensive hardware and sold thousands of computers that ran UNIX to companies and universities.

Throughout the 1980s, UNIX found its place primarily in large corporations that had enough money to purchase the expensive computing equipment needed to run UNIX (usually a DEC PDP-11, VAX, or Sun Microsystems computer). A typical UNIX system in the 1980s could cost over $100,000, yet it performed thousands of tasks for client computers (also known as dumb terminals). Today, UNIX still functions in that environment; many large companies employ different flavors of UNIX for their heavy-duty, mission-critical tasks, such as e-commerce and database hosting. Common flavors of UNIX today include FreeBSD, OpenBSD, NetBSD, HP-UX, Solaris, AIX, macOS, and iOS.

The Hacker Culture

The term **hacker** refers to a person who attempts to expand their knowledge of computing through experimentation. It should not be confused with the term **cracker**, which refers to someone who illegally uses computers for personal benefit or to cause damage.

In the early days of UNIX, hackers came primarily from engineering or scientific backgrounds, because those were the fields in which most UNIX development occurred. Fundamental to hacking was the idea of sharing knowledge. A famous hacker, Richard Stallman, promoted the free sharing of ideas while he worked at the Artificial Intelligence Laboratory at MIT. He believed that free sharing of all knowledge in the computing industry would promote development. In the mid-1980s, Stallman formed the Free Software Foundation (FSF) to encourage free software development. This movement was quickly accepted by the academic community in universities around the world, and many university students and other hackers participated in making free software, most of which ran on UNIX. As a result, the hacker culture was commonly associated with the UNIX operating system.

Unfortunately, UNIX was not free software, and by the mid-1980s some of the collaboration seen earlier by UNIX vendors diminished and UNIX development fragmented into different streams. As a result, UNIX did not represent the ideals of the FSF, and so Stallman founded the **GNU Project** in 1984 to promote free development for a free operating system that was not UNIX.

Note 15

For a description of the FSF and GNU, visit gnu.org.

This development eventually led to the publication of the GNU Public License (GPL), which legalized free distribution of source code and encouraged collaborative development. Any software published under this license must be freely available with its source code; any modifications made to the source code must then be redistributed free as well, keeping the software development free forever.

As more and more hackers worked together developing software, a hacker culture developed with its own implied rules and conventions. Most developers worked together without ever meeting each other; they communicated primarily via online forums and email. *The Hacker's Dictionary*, published by MIT in 1983, detailed the terminology regarding computing and computing culture that had appeared since the mid-1970s. Along with the FSF, GNU, and GPL, it served to codify the goals and ideals of the hacker culture. But it wasn't until the publication of Eric S. Raymond's *The Cathedral and the Bazaar*, in 1999, that the larger world was introduced to this thriving culture. Raymond, a hacker himself, described several aspects of the hacker culture:

- Software users are treated as codevelopers.
- Software is developed primarily for peer recognition and not for money.
- The original author of a piece of software is regarded as the owner of that software and coordinates the cooperative software development.
- The use of a particular piece of software determines its value, not its cost.
- Attacking the author of source code is never done. Instead, bug fixes are either made or recommended.
- Developers must understand the implied rules of the hacker culture before being accepted into it.

This hacker culture proved to be very productive, with several thousand free tools and applications created in the 1980s, including the famous Emacs editor, which is a common tool used in Linux today. During this time, many programming function libraries and UNIX commands also appeared