CHAPTER 1

Starting with Linux

IN THIS CHAPTER

- Learning what Linux is
- Learning where Linux came from
- Choosing Linux distributions
- Exploring professional opportunities with Linux
- Becoming certified in Linux

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

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

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

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

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

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

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

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

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

Understanding What Linux Is

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Understanding How Linux Differs from Other Operating Systems

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

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

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

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

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

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

Exploring Linux History

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

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

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

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

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

Note

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

Free-flowing UNIX culture at Bell Labs

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Commercial UNIX

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

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

Berkeley Software Distribution arrives

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

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

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

UNIX Laboratory and commercialization

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

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

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

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

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

Note

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

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

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

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

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

GNU transitions UNIX to freedom

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BSD loses some steam

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

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

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

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

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

Note

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

Linus builds the missing piece

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

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

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

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

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

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

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

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

Linus Torvalds continues to maintain and improve the Linux kernel.

Note

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

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

OSI open source definition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Understanding How Linux Distributions Emerged

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

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

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

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

Choosing a Red Hat distribution

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

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

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

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

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

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

Using Red Hat Enterprise Linux

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

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

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

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

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

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

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

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

Using Fedora

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

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

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

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

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

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

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

Choosing Ubuntu or another Debian distribution

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

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

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

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

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

Finding Professional Opportunities with Linux Today

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

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

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

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

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

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

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

Understanding how companies make money with Linux

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

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

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

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

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

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

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

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

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

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

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

Becoming Red Hat certified

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

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

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

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

RHCSA

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

RHCE

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

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

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

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

RHCSA topics

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

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

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

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

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

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

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

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

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

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

RHCE topics

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

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

System configuration and management

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

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

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

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

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

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

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

Installing and configuring network services

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

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

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

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

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

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

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

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

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

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

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

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

Summary

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

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

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

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

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

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