PART I

Introducing Unix

- Chapter 1: History and Background of Unix
- Chapter 2: Which Unix?
- Chapter 3: Some Basic Unix Concepts

CHAPTER

ONE

1

History and Background of Unix

- What Is Unix?
- Creation and History of Unix
- The Unix Philosophy
- Summary

Welcome to *Mastering Unix*! As we explained in the introduction, we've written this book with a variety of users in mind. You might be an old hand at using Unix systems and you've picked this book up (heavy, isn't it?) to serve as a reference guide. You could be an intermediate user of Unix or Unix-based operating systems who's looking for that extra information that will take you to the next skill level. You may be someone who knows enough about Unix to get around your shell Internet account, reading mail and news, but not doing much else. You might even be completely new to Unix and its derivatives, and have picked this book up out of idle curiosity. No matter who you are, you'll find something of use in this book. Both of us have been using Unix or Unix-based operating systems for over a decade now, and we learn something new about this magnificent beast almost every day.

If you're reading this book because you've been told, or have decided, that you need to learn how to use a Unix system, odds are that you already know at least a little bit about Unix. If you picked up this book because of its striking cover or size, or because you've heard the term *Unix* and you're wondering what it's all about, it's possible that you might not have any idea whatsoever what Unix actually is—and how it's different from the other operating systems you're probably familiar with.

One position that we hold strongly is that computer users should know the background of the software they are using. In many cases, all that's really necessary is a bit of basic history; everyone seems to know that Microsoft Windows is the brainchild of the Microsoft Corporation. Microsoft is in the news so frequently that even people who don't use computers know about Windows. The Macintosh is slightly less well-known, but it has a reputation of being user-friendly, easy to learn, and the challenger to Microsoft and the personal computer (PC).

So, you might ask, what's the point of knowing all that? Well, there are several points. If you know that Microsoft is responsible for your operating system, your integrated office suite, and your Internet Web browser, you have some idea of where that software came from. There's a company you can point to. With Unix, it's a little different—and with all the operating systems that have grown out of the original Unix, it's even more different.

To give you an understanding of Unix and where it fits into the world of computing, we've decided to start the book with Part I: "Introducing Unix." This part of the book contains information about Unix: the history of the operating system,

the various Unix variants, an introduction to the concept of Free Software, and some basic Unix concepts that you should know before reading further.

In this chapter, we provide a brief introduction to what Unix is and explain a little bit about its development, history, and philosophy. Chapter 2: "Which Unix?" introduces the wide variety of Unix variants now available and covers in more detail the three variants we've selected for this book: Linux, FreeBSD, and Sun Solaris. In addition, we will present a brief history of the Free Software movement, which affects Unix users in a significant manner. Finally, we give the opportunity to start building your Unix skills in Chapter 3: "Some Basic Unix Concepts." We've designed this part of the book to help you to understand why Unix is what it is and how that affects the concepts, skills, and programs that we describe in the rest of the book.

What Is Unix?

In the simplest terms, Unix is an *operating system.* An operating system is the software that runs behind the scenes and allows the user to operate the machine's hardware, start and stop programs, and set the parameters under which the computer operates. Modern operating systems also do a lot of other things, such as controlling network connections, but in the strictest sense, these can be thought of as extra capabilities. The most basic requirement of an operating system is that it permits the user to operate the computer.

Anyone who has used a computer in the past 10 or 15 years has used an operating system. The most common personal operating systems in use today are Microsoft's Windows family (Windows 95 and Windows 98) and Apple's MacOS. These systems were developed for use with the new generations of low-cost, personal-use computers that became available in the 1980s. As these desktop computers became more powerful and more popular, these personal operating systems saw a commensurate increase in popularity.

However, the popularity of personal operating systems such as Apple's and Microsoft's is only part of the operating-system story. Well before these systems existed, academics and computing professionals were using a variety of operating systems. Most of these are now extinct, but a few—especially Unix—survived and continued to evolve.

What we now know as *Unix* is actually an entire family of operating systems. From IBM's AIX, Xerox's Xenix, and Hewlett Packard's HP-UX to the publicly licensed Linux and FreeBSD, versions of Unix are produced by a variety of companies and organizations. All of these versions have slight differences, but it is what they have in common that makes them important.

All Versions of Unix Are Multiuser

Unix was originally designed to be used on large mainframe computers with many users. Consequently, Unix has support for user accounts and varying levels of file security, allowing users to keep their files private from one another. Even if you install a Unix-based operating system on a standalone computer and you are the only person who will ever use the computer, you will still create at least two accounts: the root account and a personal user account. Many administrators set up accounts for nonexistent people so that they can test configurations or programs under different account settings.

All Versions of Unix Are Multitasking

Unix systems can perform many tasks at once. Unix does this by means of *time slicing* (also called *true* multitasking), which means that each running process gets to use the computer for a specific period of time. This behavior is in contrast to *task switching*, which is the "multitasking" system used by personal operating systems. Task switching means that each running process gets to use the computer until it has completed a particular task; it's not really multitasking in the true sense of the term, so we've put quotation marks around it. When we talk about multitasking in this book, we are talking about time slicing, the true form of multitasking.

All Versions of Unix Can Use the Same Commands

It doesn't matter what kind of Unix-based operating system you're using, whether it's Linux, FreeBSD, Solaris, or some commercial Unix. When using a Unix-derived operating system, users can issue commands to the system by means of a *command shell.* The command shell is separate from the operating system; in fact, the shell acts as a translator between the commands you enter with the keyboard and the operating system itself. A multitude of shells is available to the Unix user. These shells can be run on any version of Unix, so that the same

commands will work on any machine using that shell. We've devoted an entire part of this book to the bash shell, which is one of the most commonly used command shells.

What Does This Mean to the End User?

To the user, then, all versions of Unix look pretty much alike. With only some minor differences, a user will use one given Unix machine in the exact same way as she would use any other Unix machine. The display might be a bit different, and the exact syntax of commands might be altered (if a command shell different from her regular shell is installed), but she can still perform her regular tasks with the same commands. The differences between the various *Unices* (the plural of Unix) come into play when you reach the level of programmers and system administrators. These are the people to whom the nuts and bolts of different systems become critically important.

If you are wondering whether it's better to use Unix A or Unix B, or if you're caught in the Linux vs. FreeBSD dilemma, don't worry. Pick one and get to know it. When you're comfortable with that one, you might want to explore another. However, you will never find that learning one particular Unix makes all other Unices incomprehensible; Unix just doesn't work that way.

When non-Unix people hear Unix people talking about command languages, shell environments, and so on, they often get the idea that Unix is an obscure and old-fashioned operating system that makes computing difficult by requiring the user to memorize complicated command syntaxes. Although it is true that Unix can be operated entirely from a command-line interface, it may come as a surprise to some of these folks to learn that Unix has a windowing system that is both older and more sophisticated than the ones that form the basis of the personal operating systems. Hundreds of graphical applications, including word processors, spreadsheets, image manipulation software, and others, can be run on Unix machines. With the continuing development of applications for Unix platforms and the transfer of popular Windows-based programs to Unix, the popularity (and ease-of-use) of this powerful operating system is bound to blossom.

So what is Unix? Unix is a powerful multiuser, multitasking family of operating systems. Unix is mature technology, having its genesis in the late 1960s, but it is thoroughly modern—it runs on just about any computing hardware you can think of.

Creation and History of Unix

Once upon a time, every computer came with its own operating system and cost thousands of dollars. The idea that an operating system could be independent of the hardware had not been developed. How is that different from today? Today, the computer you buy at Best Buy has an operating system preinstalled, but you can change that operating system if you want. For example, the Windows operating system is not integrated into the hardware of your new Compaq.

In the very earliest days of computing, of course, there were no operating systems. Computing was done by human operators on *bare machines.* This meant that for every computing task that needed to be done, the computer would have to be configured for that specific task. This was a very cumbersome way to do computing tasks, and computer scientists were always looking for ways that the machine itself could take over more of the work of processing data.

As hardware got more powerful, and the computers' internal switches were further automated, programmers began writing programs that could reconfigure the machine on the fly. Each computer manufacturer would write operating programs that were specific to the particular hardware they'd designed. This was more or less the state of affairs until the late 1970s and early 1980s, when the popular personal operating systems were first conceived and developed. Apple, for example, wrote its operating systems specifically for the hardware they'd designed, while Microsoft developed its system specifically for Intel's processors.

The Story of C

Meanwhile, others were looking for ways to use operating-system software to get the same behaviors from different types of hardware, so that a new operating system didn't need to be written for each new computer. In 1965, two computer scientists at Bell Labs, now known as Lucent Technologies, wrote the first incarnation of Unix, which ran on a Digital Equipment Corporation (DEC) PDP-7. When they acquired a PDP-11/20, the scientists (Dennis Ritchie and Ken Thompson) decided to *port* Unix to the new computer. (To port a piece of software is to rewrite it for a different platform.) The experience they gained in this exercise resulted in Ritchie's conception and design of the C programming language, still one of the most useful programming languages for Unix users.

The idea behind C was to create a programming language suitable for creating an operating system. Once C was usable, programmers could then create *compilers* for the various hardware devices; the compilers would translate C instructions into the machine's native command language, no matter what that language was. C turned out to be very successful, because it filled a need that everyone had. In fact, it was so successful that in 1973, Ritchie and Thompson completely rewrote Unix in C.

In the meantime, Bell Labs' parent company, AT&T, had been declared a monopoly by the United States Federal Trade Commission. As a result of this declaration, AT&T was subject to certain restrictions on its behavior. Partly because of these new requirements, Bell Labs began making Unix available to universities, free of charge. This was quite popular, and Unix became widely used in the academic environment. It subsequently began to propagate into the private sector when students began to graduate or leave school, taking their knowledge and affection for Unix with them.

The Rise of Unix Derivations

In 1978, AT&T announced that they would begin charging everyone, including academic institutions, for the Unix source code. In response, computer scientists at the University of California at Berkeley announced that they would create their own Unix-like system, to be called BSD (Berkeley Software Distribution) Unix. BSD was released under a very permissive license and has gone on to form the basis of many other Unix variants.

In 1987, around the same time that version 4.3 of BSD was being released, AT&T and Sun Microsystems agreed to cooperate on a plan to reintegrate the AT&T and BSD versions of Unix. Other vendors who had created their own Unices in the intervening years, such as IBM and Hewlett Packard, felt threatened by this plan and formed an organization called the Open Software Foundation. Although OSF-1, the Foundation's 1991 version of Unix, was never a major hit, parts of it managed to find their way into other distributions.

The Internet and Unix

In the mid to late 1980s, other events were occurring that would affect the growth and development of Unix. The Internet began to establish a real presence in universities and research labs. This rapid access to information and colleagues made

possible a new type of software development. In previous years, programmers and developers worked in laboratories together. The physical proximity of other team members and the computers fueled innovation and hard work; this method of development was typified by MIT's Research Lab, home of many of the inventions we take for granted today.

However, the Internet changed everything. Programmers were no longer required to be in the same building or city. With instant communication via e-mail and the ability to share code files with negligible cost, programmers soon realized that they could work on software projects with colleagues thousands of miles away or on different continents. The result of this realization was that Unix variants began appearing that were free for the downloading. Anyone with a yen to hone their programming skills could work on these distributions and contribute their work back to the project.

These free Unices had the effect of reenergizing enthusiasm for Unix on college campuses, because students could download them for free and install them on their personal computers. The result was that computer science students now had the same programming environment in their dorm rooms or apartments as they used in their classes—no more fighting for time on a mainframe computer or waiting in line for a computer in the campus-research laboratory. The additional time has meant that college students are now as involved in the Unix community as those who are professional Unix administrators or programmers.

Unix Today

All the developments of the last 40 years have brought us to the vibrant Unix community of today. Linux and FreeBSD, two free Unices, are very popular on college campuses, and Linux is beginning to make inroads into business and the popular consciousness. CNN's online news site, `http://www.cnn.com`, even runs regular columns on Linux in their Technology section.

Although nowhere near as popular as Microsoft's operating systems, Linux and FreeBSD are beginning to establish a toehold in the personal-computer market, as consumers are beginning to learn that they can have a full-power, industrial-strength operating system at low cost. Businesses are beginning to take advantage of Linux and FreeBSD to save money on small servers for their internal use.

Meanwhile, Unix and Unix-derived operating systems are the de facto standard for large servers. AIX, HP-UX, and Sun's Solaris are extremely popular for serving large Internet sites and databases. We've heard several reports from system administrators at large corporations who use a Unix-derived operating system on their Web and e-mail servers to provide reliability and lengthy up-times, even if the majority of the company's computers are managed with Windows NT so that Windows software programs can be used.

The Unix Philosophy

We've covered the history of Unix, but is that what makes Unix special? Not completely. From the very beginning, a number of assumptions have been built into the design of Unix. Over time, these ideas have proven themselves as valid and have taken on the quality of an entire philosophy. Some of the main ideas of this Unix philosophy are explained below; you'd probably get quite a few suggestions for other main components of the concept, were you to ask around, but these seem to be the core of everyone's idea of Unix.

Keep things small: Each component of the system should be as small and simple as possible. Each component may not be especially powerful by itself, but small components can be combined into powerful and flexible complex objects. Small programs are easy to understand and maintain, and simple programs can often be adapted to unforeseen uses. Small modules can be used to affect the kernel's behavior, so that only one action or setting is controlled by each module.

Everything is configurable: The behavior of any particular program or command can be configured in as many different ways as imaginable. Users can configure their individual accounts as they like, while administrators can configure general system settings or regular routines to save time and effort. If you find a Unix program that isn't configurable, it's an anomaly.

Everything is consistent: Every aspect of a Unix system is represented as a file. Text documents, executable programs, system features, hardware devices, and just about anything else you can think of are represented by the system as a file. A set of consistent ways of dealing with system features

has been developed based on this idea. We explain this concept in more detail in Chapter 3: "Some Basic Unix Concepts."

Captive user interfaces are avoided: The more popular personal operating systems, such as MacOS and Windows, are based on the assumption that the user of a program is always a human. This ignores the fact that the user of a program might be another program. In those operating systems, the user interface is therefore captive to the human user; if you don't click the button in a dialog box, the operating system patiently waits until you do. This can take hours or days. Unix avoids this problem wherever possible by allowing programs to function in noninteractive modes. These modes allow programs to be chained together to perform complex tasks without any intervention from the user.

Automation is possible: Many aspects of the Unix interface allow for automation. The Unix shells, in addition to being simple command interpreters, are also program interpreters. Anything that can be done from the keyboard can also be done from within a program. This means that you can write scripts that will call certain programs automatically at a given time or system state. Most system administrators automate routine tasks, such as backups, to avoid having to do such jobs by hand. Unix is powerful enough to handle most of its administration tasks by itself, with the only human intervention required being a check of the results.

Summary

Unix is an operating system with a long and rich history, as computer history goes. Since Unix's roots are found in mainframe computers, the operating system includes support for multiple users, wise allocation of system resources through multitasking, configurability and flexibility for user and administrator preferences, and the ability to use the same commands regardless of the Unix variant being used.

The underlying Unix philosophy keeps the operating system flexible. Unix is small and modular, effectively organized, responsive to user needs yet able to run multiple automatic processes, and consistent in its operation and output. With Unix, you can perform simple tasks or complicated programming operations. Whatever you choose to do with a Unix computer, Unix will be able to keep up.