Sixth Edition

- Administer, secure, and virtualize Fedora, Red Hat, CentOS, Debian, and Ubuntu
- Manage users, files, software, networking, and Internet and intranet services
- Seamlessly interoperate with Windows-based systems and networks

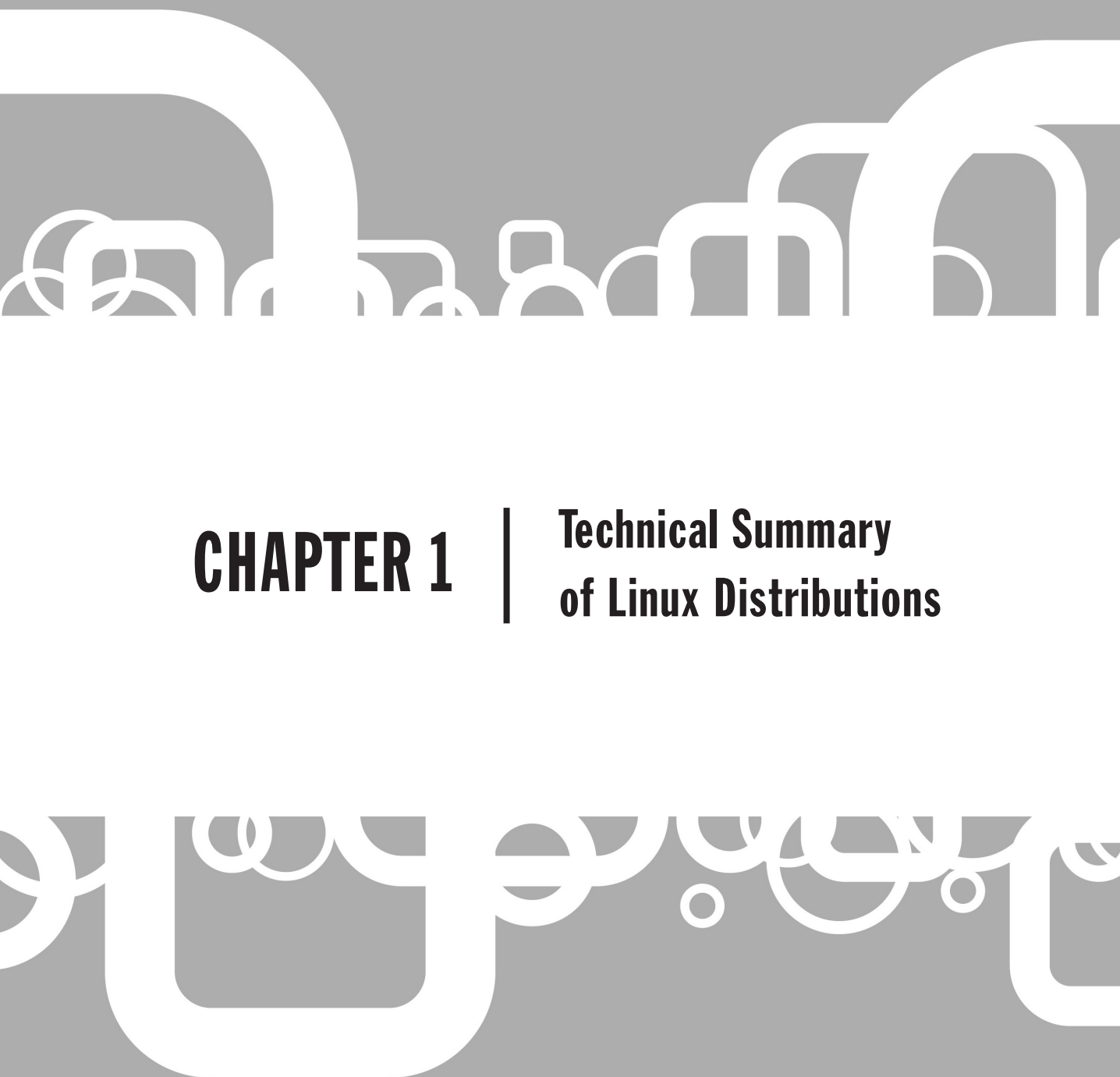
Linux Administration

A Beginner's Guide

Wale Soyinka

Covers Linux kernel 3.x series, B-tree file system (Btrfs), systemd, GlusterFS, UEFI, KVM, and IPv6

PART I | Introduction, Installation, and Software Management



CHAPTER 1 | Technical Summary of Linux Distributions

Linux has hit the mainstream. Hardly a day goes by without a mention of Linux (or open source software) in widely read and viewed print or digital media. What was only a hacker's toy several years ago has grown up tremendously and is well known for its stability, performance, and extensibility.

If you need more proof concerning Linux's penetration, just pay attention to the frequency with which "Linux" is listed as a *desirable* and *must have* skill for technology-related job postings of Fortune 500 companies, small to medium-sized businesses, tech start-ups, and government, research, and entertainment industry jobs—to mention a few. The skills of good Linux system administrators and engineers are highly desirable!

With the innovations that are taking place in different open source projects (such as K Desktop Environment, GNOME, Unity, LibreOffice, Android, Apache, Samba, Mozilla, and so on), Linux has made serious inroads into consumer desktop, laptop, tablet, and mobile markets. This chapter looks at some of the core server-side technologies as they are implemented in the Linux (open source) world and in the Microsoft Windows Server world (possibly the platform you are considering replacing with Linux). But before delving into any technicalities, this chapter briefly discusses some important underlying concepts and ideas that form the genetic makeup of Linux and Free and Open Source Software (FOSS).

Linux: The Operating System

Usually, people (mis)understand Linux to be an entire software suite of developer tools, editors, graphical user interfaces (GUIs), networking tools, and so forth. More formally and correctly, such software *collectively* is called a *distribution,* or *distro.* The distro is the entire software suite that makes Linux useful.

So if we consider a distribution everything you need for Linux, what then *is* Linux exactly? Linux itself is the core of the operating system: the *kernel.* The kernel is the program acting as chief of operations. It is responsible for starting and stopping other programs (such as editors), handling requests for memory, accessing disks, and managing network connections. The complete list of kernel activities could easily fill a chapter in itself, and, in fact, several books documenting the kernel's internal functions have been written.

The kernel is a nontrivial program. It is also what puts the Linux badge on all the numerous Linux distributions. All distributions use essentially the same kernel, so the fundamental behavior of all Linux distributions is the same.

You've most likely heard of the Linux distributions named Red Hat Enterprise Linux (RHEL), Fedora, Debian, Mandrake, Ubuntu, Kubuntu, openSUSE, CentOS, Gentoo, and so on, which have received a great deal of press.

Linux distributions can be broadly categorized into two groups. The first category includes the purely commercial distros, and the second includes the noncommercial distros, or *spins*. The commercial distros generally offer support for their distribution—at a cost. The commercial distros also tend to have a longer release life cycle. Examples of commercial flavors of Linux-based distros are RHEL and SUSE Linux Enterprise (SLE).

The noncommercial distros, on the other hand, are free. These distros try to adhere to the original spirit of the open source software movement. They are mostly community supported and maintained—the community consists of the users and developers. The community support and enthusiasm can sometimes supersede that provided by the commercial offerings.

Several of the so-called noncommercial distros also have the backing and support of their commercial counterparts. The companies that offer the purely commercial flavors have vested interests in making sure that free distros exist. Some of the companies use the free distros as the proofing and testing ground for software that ends up in the commercial spins. Examples of noncommercial flavors of Linux-based distros are Fedora, openSUSE, Ubuntu, Linux Mint, Gentoo, and Debian. Linux distros such as Gentoo might be less well known and have not reached the same scale of popularity as Fedora, openSUSE, and others, but they are out there and in active use by their respective (and dedicated) communities.

What's interesting about the commercial Linux distributions is that most of the programs with which they ship were not written by the companies themselves. Rather, other people have released their programs with licenses, allowing their redistribution with source code. By and large, these programs are also available on other variants of UNIX, and some of them are becoming available under Windows as well. The makers of the distribution simply bundle them into one convenient and cohesive package that's easy to install. In addition to bundling existing software, several of the distribution makers also develop value-added tools that make their distribution easier to administer or compatible with more hardware, but the software that they ship is generally written by others. To meet certain regulatory requirements, some commercial distros try to incorporate/implement more specific security requirements that the FOSS community might not care about but that some institutions/corporations do care about.

What Is Open Source Software and GNU All About?

In the early 1980s, Richard Matthew Stallman began a movement within the software industry. He preached (and still does) that software should be free. Note that by *free*, he doesn't mean in terms of price, but rather free in the same sense as *freedom* or *libre*. This means shipping not just a product, but the entire source code as well. To clarify the meaning of free software, Stallman was once famously quoted as saying:

> *"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer."*

Stallman's policy was, somewhat ironically, a return to classic computing, when software was freely shared among hobbyists on small computers and provided as part of the hardware by mainframe and minicomputer vendors. It was not until the late 1960s that IBM considered selling application software. Through the 1950s and most of the 1960s, IBM considered software as merely a tool for enabling the sale of hardware.

This return to *openness* was a wild departure from the early 1980s convention of selling prepackaged software, but Stallman's concept of open source software was in line with the initial distributions of UNIX from Bell Labs. Early UNIX systems did

contain full source code. Yet by the late 1970s, source code was typically removed from UNIX distributions and could be acquired only by paying large sums of money to AT&T (now SBC). The Berkeley Software Distribution (BSD) maintained a free version, but its commercial counterpart, BSDi, had to deal with many lawsuits from AT&T until it could be proved that nothing in the BSD kernel came from AT&T.

Kernel Differences

Each company that sells a Linux distribution of its own will be quick to tell you that its kernel is better than others. How can a company make this claim? The answer comes from the fact that each company maintains its own patch set. To make sure that the kernels largely stay in sync, most companies do adopt patches that are posted on www.kernel.org, the "Linux Kernel Archives." Vendors, however, typically do not track the release of every single kernel version that is released onto www.kernel.org. Instead, they take a foundation, apply their custom patches to it, run the kernel through their quality assurance (QA) process, and then take it to production. This helps organizations have confidence that their kernels have been sufficiently baked, thus mitigating any perceived risk of running open source–based operating systems.

The only exception to this rule revolves around security issues. If a security issue is found with a version of the Linux kernel, vendors are quick to adopt the necessary patches to fix the problem immediately. A new release of the kernel with the fixes is often made within a short time (commonly less than 24 hours) so that administrators who install it can be sure their installations are secure. Thankfully, exploits against the kernel itself are rare.

So if each vendor maintains its own patch set, what exactly is it patching? This answer varies from vendor to vendor, depending on each vendor's target market. Red Hat, for instance, is largely focused on providing enterprise-grade reliability and solid efficiency for application servers. This might be different from the mission of the Fedora team, which is more interested in trying new technologies quickly, and even more different from the approach of a vendor that is trying to put together a desktop-oriented or multimedia-focused Linux system.

What separates one distribution from the next are the value-added tools that come with each one. Asking, "Which distribution is better?" is much like asking, "Which is better, Coke or Pepsi?" Almost all colas have the same basic ingredients—carbonated water, caffeine, and high-fructose corn syrup—thereby giving the similar effect of quenching thirst and bringing on a small caffeine-and-sugar buzz. In the end, it's a question of requirements: Do you need commercial support? Did your application vendor recommend one distribution over another? Does the software (package) updating infrastructure suit your site's administrative style better than another distribution? When you review your requirements, you'll find that there is likely a distribution that is geared toward your exact needs.

The idea of giving away source code is a simple one: A user of the software should never be forced to deal with a developer who might or might not support that user's intentions for the software. The user should never have to wait for bug fixes to be published. More important, code developed under the scrutiny of other programmers is typically of higher quality than code written behind locked doors. One of the great benefits of open source software comes from the users themselves: Should they need a new feature, they can add it to the original program and then contribute it back to the source so that everyone else can benefit from it.

This line of thinking sprung a desire to release a complete UNIX-like system to the public, free of license restrictions. Of course, before you can build any operating system, you need to build tools. And this is how the GNU project was born.

NOTE GNU stands for GNU's Not UNIX—recursive acronyms are part of hacker humor. If you don't understand why it's funny, don't worry. You're still in the majority.

What Is the GNU Public License?

An important thing to emerge from the GNU project is the *GNU Public License (GPL).* This license explicitly states that the software being released is free and that no one can ever take away these freedoms. It is acceptable to take the software and resell it, even for a profit; however, in this resale, the seller must release the full source code, including any changes. Because the resold package remains under the GPL, the package can be distributed for free and resold yet again by anyone else for a profit. Of primary importance is the liability clause: The programmers are not liable for any damages caused by their software.

It should be noted that the GPL is not the only license used by open source software developers (although it is arguably the most popular). Other licenses, such as BSD and Apache, have similar liability clauses but differ in terms of their redistribution. For instance, the BSD license allows people to make changes to the code and ship those changes without having to disclose the added code. (Whereas the GPL requires that the added code is shipped.) For more information about other open source licenses, check out www.opensource.org.

Historical Footnote

Many, many moons ago, Red Hat started a commercial offering of its erstwhile free product (Red Hat Linux). The commercial release gained steam with the Red Hat Enterprise Linux (RHEL) series. Because the foundation for RHEL is GPL, individuals interested in maintaining a free version of Red Hat's distribution have been able to do so. Furthermore, as an outreach to the community, Red Hat created the Fedora Project, which is considered the testing grounds for new software before it is adopted by the RHEL team. The Fedora Project is freely distributed and can be downloaded from http://fedoraproject.org.

Upstream and Downstream

To help you understand the concept of upstream and downstream components, let's start with an analogy. Picture, if you will, a pizza with all your favorite toppings.

The pizza is put together and baked by a local *pizza shop*. Several things go into making a great pizza—cheeses, vegetables, flour (dough), herbs, meats, to mention a few. The pizza shop will often make some of these ingredients in-house and rely on other businesses to supply other ingredients. The pizza shop will also be tasked with assembling the ingredients into a complete finished pizza.

Let's consider one of the most common pizza ingredients—cheese. The cheese is made by a *cheesemaker* who makes her cheese for many other industries or applications, including the pizza shop. The cheesemaker is pretty set in her ways and has very strong opinions about how her product should be paired with other food stuffs (wine, crackers, bread, vegetables, and so on). The pizza shop owners, on the other hand, do not care about other food stuffs—they care only about making a great pizza. Sometimes the cheesemaker and the pizza shop owners will bump heads because of differences in opinion and objectives. And at other times they will be in agreement and cooperate beautifully. Ultimately (and sometimes unbeknown to them), the pizza shop owners and cheesemaker care about the same thing: producing the best product that they can.

The pizza shop in our analogy here represents the Linux distributions vendors/projects (Fedora, Debian, RHEL, openSUSE, and so on). The cheesemaker represents the different software project maintainers that provide the important programs and tools (such as the Bourne Again Shell [BASH], GNU Image Manipulation Program [GIMP], GNOME, KDE, Nmap, and GNU Compiler Collection [GCC]) that are packaged together to make a complete distribution (pizza). The Linux distribution vendors are referred to as the *downstream* component of the open source food chain; the maintainers of the accompanying different software projects are referred to as the *upstream* component.

Standards

One argument you hear regularly against Linux is that too many different distributions exist, and that by having multiple distributions, fragmentation occurs. The argument opines that this fragmentation will eventually lead to different versions of incompatible Linuxes.

This is, without a doubt, complete nonsense that plays on "FUD" (fear, uncertainty, and doubt). These types of arguments usually stem from a misunderstanding of the kernel and distributions.

Ever since becoming so mainstream, the Linux community understood that it needed a formal method and standardization process for how certain things should be done among the numerous Linux spins. As a result, two major standards are actively being worked on.

The *File Hierarchy Standard (FHS)* is an attempt by many of the Linux distributions to standardize on a directory layout so that developers have an easy time making sure their applications work across multiple distributions without difficulty. As of this writing, several major Linux distributions have become completely compliant with this standard.

The *Linux Standard Base (LSB)* specification is a standards group that specifies what a Linux distribution should have in terms of libraries and tools.

A developer who assumes that a Linux machine complies only with LSB and FHS is almost guaranteed to have an application that will work with all compliant Linux installations. All of the major distributors have joined these standards groups. This should ensure that all desktop distributions will have a certain amount of commonality on which a developer can rely.

From a system administrator's point of view, these standards are interesting but not crucial to administering a Linux environment. However, it never hurts to learn more about both. For more information on the FHS, go to their web site at www.pathname.com/fhs. To find out more about LSB, check out www.linuxbase.org.

The Advantages of Open Source Software

If the GPL seems like a bad idea from the standpoint of commercialism, consider the surge of successful open source software projects—they are indicative of a system that does indeed work. This success has evolved for two reasons. First, as mentioned earlier, errors in the code itself are far more likely to be caught and quickly fixed under the watchful eyes of peers. Second, under the GPL system, programmers can release code without the fear of being sued. Without that protection, people might not feel as comfortable to release their code for public consumption.

NOTE The concept of free software, of course, often begs the question of why anyone would release his or her work for free. As hard as it might be to believe, some people do it purely for altruistic reasons and the love of it.

Most projects don't start out as full-featured, polished pieces of work. They often begin life as a quick hack to solve a specific problem bothering the programmer at the time. As a quick-and-dirty hack, the code might not have a sales value. But when this code is shared and consequently improved upon by others who have similar problems and needs, it becomes a useful tool. Other program users begin to enhance it with features they need, and these additions travel back to the original program. The project thus evolves as the result of a group effort and eventually reaches full refinement. This polished program can contain contributions from possibly hundreds, if not thousands, of programmers who have added little pieces here and there. In fact, the original author's code is likely to be little in evidence.

There's another reason for the success of generously licensed software. Any project manager who has worked on commercial software knows that the *real* cost of development software isn't in the development phase. It's in the cost of selling, marketing, supporting, documenting, packaging, and shipping that software. A programmer carrying out a weekend hack to fix a problem with a tiny, kludged program might lack the interest, time, and money to turn that hack into a profitable product.

When Linus Torvalds released Linux in 1991, he released it under the GPL. As a result of its open charter, Linux has had a notable number of contributors and analyzers. This participation has made Linux strong and rich in features. It is estimated that since the v.2.2.0 kernel, Torvalds's contributions represent less than 2 percent of the total code base.

NOTE This might sound strange, but it is true. Contributors to the Linux kernel code include the companies with competing operating system platforms. For example, Microsoft was one of the top code contributors to the Linux version 3.0 kernel code base (as measured by the number of changes or patches relative to the previous kernel version). Even though this might have been for self-promoting reasons on Microsoft's part, the fact remains that the open source licensing model that Linux adopts permits this sort of thing to happen. Everyone and anyone who knows how-to, can contribute code subject to peer review from which everyone can benefit!

Because Linux is free (as in speech), anyone can take the Linux kernel and other supporting programs, repackage them, and resell them. A lot of people and corporations have made money with Linux doing just this! As long as these individuals release the kernel's full source code along with their individual packages, and as long as the packages are protected under the GPL, everything is legal. Of course, this also means that packages released under the GPL can be resold by other people under other names for a profit.

In the end, what makes a package from one person more valuable than a package from another person are the value-added features, support channels, and documentation. Even IBM can agree to this; it's how the company made most of its money from 1930 to 1970, and again in the late 1990s and early 2000s with IBM Global Services. The money isn't necessarily in the product alone; it can also be in the services that go with it.

The Disadvantages of Open Source Software

This section was included to provide a detailed, balanced, and unbiased contrast to the previous section, which discussed some of the advantages of open source software.

Unfortunately we couldn't come up with any disadvantages at the time of this writing! Nothing to see here.

Understanding the Differences Between Windows and Linux

As you might imagine, the differences between Microsoft Windows and the Linux operating system cannot be completely discussed in the confines of this section. Throughout this book, topic by topic, you'll read about the specific contrasts between

the two systems. In some chapters, you'll find no comparisons, because a major difference doesn't really exist.

But before we attack the details, let's take a moment to discuss the primary architectural differences between the two operating systems.

Single Users vs. Multiple Users vs. Network Users

Windows was originally designed according to the "one computer, one desk, one user" vision of Microsoft's co-founder, Bill Gates. For the sake of discussion, we'll call this philosophy "single-user." In this arrangement, two people cannot work in parallel running (for example) Microsoft Word on the same machine at the same time. You can buy Windows and run what is known as Terminal Server, but this requires huge computing power and extra costs in licensing. Of course, with Linux, you don't run into the cost problem, and Linux will run fairly well on just about any hardware.

Linux borrows its philosophy from UNIX. When UNIX was originally developed at Bell Labs in the early 1970s, it existed on a PDP-7 computer that needed to be shared by an entire department. It required a design that allowed for *multiple users* to log into the central machine at the same time. Various people could be editing documents, compiling programs, and doing other work at the exact same time. The operating system on the central machine took care of the "sharing" details so that each user seemed to have an individual system. This multiuser tradition continues through today on other versions of UNIX as well. And since Linux's birth in the early 1990s, it has supported the multiuser arrangement.

NOTE Most people believe that the term "multitasking" was invented with the advent of Windows 95. But UNIX has had this capability since 1969! You can rest assured that the concepts included in Linux have had many years to develop and prove themselves.

Today, the most common implementation of a multiuser setup is to support *servers*—systems dedicated to running large programs for use by many clients. Each member of a department can have a smaller workstation on the desktop, with enough power for day-to-day work. When someone needs to do something requiring significantly more processing power or memory, he or she can run the operation on the server.

"But, hey! Windows can allow people to offload computationally intensive work to a single machine!" you may argue. "Just look at SQL Server!" Well, that position is only half correct. Both Linux and Windows are indeed capable of providing services such as databases over the network. We can call users of this arrangement *network users*, since they are never actually logged into the server, but rather send requests to the server. The server does the work and then sends the results back to the user via the network. The catch in this case is that an application must be specifically written to perform such server/client duties. Under Linux, a user can run any program allowed by the system administrator on the server without having to redesign that program. Most users find the ability to run arbitrary programs on other machines to be of significant benefit.

The Monolithic Kernel and the Micro-Kernel

Two forms of kernels are used in operating systems. The first, a *monolithic* kernel provides all the services the user applications need. The second, a *micro-kernel* is much more minimal in scope and provides only the bare minimum core set of services needed to implement the operating system.

Linux, for the most part, adopts the monolithic kernel architecture; it handles everything dealing with the hardware and system calls. Windows, on the other hand, works off a micro-kernel design. The Windows kernel provides a small set of services and then interfaces with other executive services that provide process management, input/output (I/O) management, and other services. It has yet to be proved which methodology is truly the best way.

Separation of the GUI and the Kernel

Taking a cue from the Macintosh design concept, Windows developers integrated the GUI with the core operating system. One simply does not exist without the other. The benefit with this tight coupling of the operating system and user interface is consistency in the appearance of the system.

Although Microsoft does not impose rules as strict as Apple's with respect to the appearance of applications, most developers tend to stick with a basic look and feel among applications. One reason this is dangerous, however, is that the video card driver is now allowed to run at what is known as "Ring 0" on a typical *x*86 architecture. Ring 0 is a protection mechanism—only privileged processes can run at this level, and typically user processes run at Ring 3. Because the video card is allowed to run at Ring 0, it could misbehave (and it does!), and this can bring down the whole system.

On the other hand, Linux (like UNIX in general) has kept the two elements—user interface and operating system—separate. The X Window System interface is run as a user-level application, which makes it more stable. If the GUI (which is complex for both Windows and Linux) fails, Linux's core does not go down with it. The GUI process simply crashes, and you get a terminal window. The X Window System also differs from the Windows GUI in that it isn't a complete user interface. It defines only how basic objects should be drawn and manipulated on the screen.

One of the most significant features of the X Window System is its ability to display windows across a network and onto another workstation's screen. This allows a user sitting on host A to log into host B, run an application on host B, and have all of the output routed back to host A. It is possible for two people to be logged into the same machine, running a Linux equivalent of Microsoft Word (such as OpenOffice or LibreOffice) at the same time.

In addition to the X Window System core, a window manager is needed to create a useful environment. Linux distributions come with several window managers, including the heavyweight and popular GNOME and KDE environments (both of which are available on other variants of UNIX as well). Both GNOME and KDE offer an environment that is friendly, even to the casual Windows user. If you're concerned

with speed, you can look into the F Virtual Window Manager (FVWM), Lightweight X11 Desktop Environment (LXDE), and Xfce window managers. They might not have all the glitz of KDE or GNOME, but they are really fast and lightweight.

So which approach is better—Windows or Linux—and why? That depends on what you are trying to do. The integrated environment provided by Windows is convenient and less complex than Linux, but out of the box, Windows lacks the X Window System feature that allows applications to display their windows across the network on another workstation. The Windows GUI is consistent, but it cannot be easily turned off, whereas the X Window System doesn't have to be running (and consuming valuable hardware resources) on a server.

NOTE With its latest server family (Windows Server 8 and newer), Microsoft has somewhat decoupled the GUI from the base operating system (OS). You can now install and run the server in a so-called "Server Core" mode. Windows Server 8 Server Core can run without the usual Windows GUI. Managing the server in this mode is done via the command line or remotely from a regular system, with full GUI capabilities.

The Network Neighborhood

The native mechanism for Windows users to share disks on servers or with each other is through the Network Neighborhood. In a typical scenario, users *attach* to a share and have the system assign it a drive letter. As a result, the separation between client and server is clear. The only problem with this method of sharing data is more people-oriented than technology-oriented: People have to know which servers contain which data.

With Windows, a new feature borrowed from UNIX has also appeared: *mounting.* In Windows terminology, it is called *reparse points.* This is the ability to mount a CD-ROM drive into a directory on your C drive. The concept of mounting resources (optical media, network shares, and so on) in Linux/UNIX might seem a little strange, but as you get used to Linux, you'll understand and appreciate the beauty in this design. To get anything close to this functionality in Windows, you have to map a network share to a drive letter.

Right from inception, Linux was built with support for the concept of mounting, and as a result, different types of file systems can be mounted using different protocols and methods. For example, the popular Network File System (NFS) protocol can be used to mount remote shares/folders and make them appear local. In fact, the Linux Automounter can dynamically mount and unmount different file systems on an as-needed basis.

A common example of mounting partitions under Linux involves mounted home directories. The user's home directories can reside on a remote server, and the client systems can automatically mount the directories at boot time. So the **/home** directory exists on the client, but the **/home/username** directory (and its contents) can reside on the server.

Under Linux NFS and other Network File Systems, users never have to know server names or directory paths, and their ignorance is your bliss. No more questions about which server to connect to. Even better, users need not know when the server configuration must change. Under Linux, you can change the names of servers and adjust this information on client-side systems without making any announcements or having to reeducate users. Anyone who has ever had to reorient users to new server arrangements will appreciate the benefits and convenience of this.

Printing works in much the same way. Under Linux, printers receive names that are independent of the printer's actual host name. (This is especially important if the printer doesn't speak Transmission Control Protocol/Internet Protocol, or TCP/IP.) Clients point to a print server whose name cannot be changed without administrative authorization. Settings don't get changed without you knowing it. The print server can then redirect all print requests as needed. The unified interface that Linux provides will go a long way toward improving what might be a chaotic printer arrangement in your network environment. This also means you don't have to install print drivers in several locations.

The Registry vs. Text Files

Think of the Windows Registry as the ultimate configuration database—thousands upon thousands of entries, only a few of which are completely documented.

"What? Did you say your Registry *got corrupted?*" <maniacal laughter> "Well, yes, we can try to restore it from last night's backups, but then Excel starts acting funny and the technician (who charges $65 just to answer the phone) said to reinstall.…"

In other words, the Windows Registry system can be at best, difficult to manage. Although it's a good idea in theory, most people who have serious dealings with it don't emerge from battling it without a scar or two.

Linux does not have a registry, and this is both a blessing and a curse. The blessing is that configuration files are most often kept as a series of text files (think of the Windows .ini files). This setup means you're able to edit configuration files using the text editor of your choice rather than tools such as **`regedit`**. In many cases, it also means you can liberally comment those configuration files so that six months from now you won't forget why you set up something in a particular way. Most software programs that are used on Linux platforms store their configuration files under the **/etc** directory or one of its subdirectories. This convention is widely understood and accepted in the FOSS world.

The curse of a no-registry arrangement is that there is no standard way of writing configuration files. Each application can have its own format. Many applications are now coming bundled with GUI-based configuration tools to alleviate some of these problems. So you can do a basic setup easily, and then manually edit the configuration file when you need to do more complex adjustments.

In reality, having text files hold configuration information usually turns out to be an efficient method. Once set, they rarely need to be changed; even so, they are straight text files and thus easy to view when needed. Even more helpful is that it's

easy to write scripts to read the same configuration files and modify their behavior accordingly. This is especially helpful when automating server maintenance operations, which is crucial in a large site with many servers.

Domains and Active Directory

If you've been using Windows long enough, you might remember the Windows NT domain controller model. If twinges of anxiety ran through you when reading the last sentence, you might still be suffering from the shell shock of having to maintain Primary Domain Controllers (PDCs), Backup Domain Controllers (BDCs), and their synchronization.

Microsoft, fearing revolt from administrators all around the world, gave up on the Windows NT model and created Active Directory (AD). The idea behind AD was simple: Provide a repository for any kind of administrative data, whether it is user logins, group information, or even just telephone numbers. In addition, provide a central place to manage authentication and authorization for a domain. The domain synchronization model was also changed to follow a Domain Name System (DNS)–style hierarchy that has proved to be far more reliable. NT LAN Manager (NTLM) was also dropped in favor of Kerberos. (Note that AD is still somewhat compatible with NTLM.)

While running **`dcpromo`** might not be anyone's idea of a fun afternoon, it is easy to see that AD works pretty well.

Out of the box, Linux does not use a tightly coupled authentication/authorization and data store model the way that Windows does with AD. Instead, Linux uses an abstraction model that allows for multiple types of stores and authentication schemes to work without any modification to other applications. This is accomplished through the Pluggable Authentication Modules (PAM) infrastructure and the name resolution libraries that provide a standard means of looking up user and group information for applications. It also provides a flexible way of storing that user and group information using a variety of schemes.

For administrators looking to Linux, this abstraction layer can seem peculiar at first. However, consider that you can use anything from flat files, to Network Information Service (NIS), to Lightweight Directory Access Protocol (LDAP) or Kerberos for authentication. This means you can pick the system that works best for you. For example, if you have an existing UNIX infrastructure that uses NIS, you can simply make your Linux systems plug into that. On the other hand, if you have an existing AD infrastructure, you can use PAM with Samba or LDAP to authenticate against the domain. Use Kerberos? No problem. And, of course, you can choose to make your Linux system not interact with any external authentication system. In addition to being able to tie into multiple authentication systems, Linux can easily use a variety of tools, such as OpenLDAP, to keep directory information centrally available as well.

Summary

In this chapter, we offered an overview of what Linux is and what it isn't. We discussed a few of the guiding principles, ideas, and concepts that govern open source software and Linux by extension. We ended the chapter by covering some of the similarities and differences between core technologies in the Linux and Microsoft Windows Server worlds. Most of these technologies and their practical uses are dealt with in greater detail in the rest of this book.

If you are so inclined and would like to get more detailed information on the internal workings of Linux itself, you might want to start with the source code. The source code can be found at www.kernel.org. It is, after all, open source!