Summary for Linux Essentials

29/12/2024

Chapter 1: introdution to Linux

1.1 Linux is Everywhere

Hello and welcome to **NDG Linux Essentials**!

Linux is everywhere; Linux jobs are everywhere. There is a demand for Linux skills in just about every industry and job category on the planet, and not enough Linux talent to meet this growing demand. It's also fun and rewarding to work with something that's so much a part of our modern lives yet which so few people understand.

If someone says they have Linux experience, it might refer to configuring systems, running web servers, or any number of other services and programs that operate on top of Linux. Over time, Linux administration has evolved to encompass just about every task that a modern business, educational institution or government organization might use in their daily operations.

The journey of learning you are beginning today has no ending point. It can take you in a myriad of different directions, from cybersecurity to application and game development, system administration, networking, big data, and artificial intelligence; all of these fields are rooted in Linux.

Every time you execute a search on the internet, watch a video on your phone or order something online, that's likely Linux at work. It epitomizes a whole that is greater than the sum of its parts, a vast undertaking, done voluntarily, by some of the smartest people on the planet.

While your journey will be ongoing, be comforted that you are learning a set of technologies, commands, and methods that have stood the test of time. Linux utilizes and expands upon many of the commands and ways of accomplishing computing that UNIX began, with a rate of change per year that's very manageable. Now, some 30+ years on, many of those learned commands are still active and used every day by sysadmins, devops, and architects. Linux is a revolution of evolution, allowing you to learn the majority once and keep up with the small percentage of changes in a continual learning process.

1.2 Linux is a Kernel

The definition of the word Linux depends on the context in which it is used. Linux means the kernel of the system, which is the central controller of everything that happens on the computer.

When most people refer to Linux, they are really referring to a combination of software called **GNU/Linux**, which defines the operating system. **GNU** is the free software that provides open source equivalents of many common UNIX commands. The **Linux** part of this combination is the Linux kernel, which is the core of the operating system. The kernel

is loaded at boot time and stays running to manage every aspect of the functioning system.

The story of Linux begins with **UNIX**, an operating system developed at AT&T Bell Labs in the 1970s. UNIX is written in the **C** language making it uniquely portable amongst competing operating systems, which were typically closely tied to the hardware for which they were written. It quickly gained popularity in research and academic settings, as well as amongst programmers who were attracted to its modularity. Over time it was modified and forked (that is, people modified it, and those modifications served as the basis for other systems) such that at present there are many different variants of UNIX. However, UNIX is now both a trademark and a specification, owned by an industry consortium called the **Open Group**. Only software that has been certified by the Open Group may call itself UNIX.

Linux started in 1991 as a hobby project of **Linus Torvalds**, a Finnish-born computer scientist studying at the University of Helsinki. Frustrated by the licensing of MINIX, a UNIX-like operating system designed for educational use, and its creator's desire not to make it a full operating system, Linus decided to create his own OS kernel.

From this humble beginning, Linux has grown to be the dominant operating system on the Internet, and arguably the most important computer program of any kind. Despite adopting all the requirements of the UNIX specification, Linux has not been certified, so Linux really isn't UNIX! It's just… UNIX-like.

Prior to and alongside this development was the **GNU Project**, created by **Richard Stallman** in 1983. While GNU initially focused on building their own operating system, they ultimately were far more effective at building tools that go along with a UNIX-like operating system, such as the editors, compilers and user interfaces that make a kernel usable. Since the source was all freely available, Linux programmers were able to incorporate the GNU tools to provide a complete operating system. As such, many of the tools and utilities that are part of the Linux system evolved from these early GNU tools.

Consider This

Linus originally named the project Freax, however, an administrator of the server where the development files were uploaded renamed it Linux, a portmanteau of Linus' name and UNIX. The name stuck.

GNU is a recursive acronym for "GNU's Not Unix," and it's pronounced just like the African horned antelope that is its namesake.

1.3 Linux is Open Source

Historically, most software has been issued under a closed-source license, meaning that you get the right to use the machine code, but cannot see the source code. Often the license explicitly says that you may not attempt to reverse engineer the machine code back to source code to figure out what it does!

The development of Linux closely parallels the rise of open source software. Open source takes a source-centric view of software. The open source philosophy is that you have a right to obtain the software source code and to modify it for your own use.

Linux adopted this philosophy to great success. Linus made the source programming code (the instructions a computer uses to operate) freely available, allowing others to join in and shape this fledgling operating system. It was not the first system to be developed by a volunteer group, but since it was built from scratch, early adopters could influence the project's direction. People took the source, made changes, and shared them back with the rest of the group, greatly accelerating the pace of development, and ensuring mistakes from other operating systems were not repeated.

Consider This

The source code may be written in any of hundreds of different languages. Linux happens to be written in **C**, a versatile and relatively easy language to learn, which shares history with the original UNIX. This decision, made long before it's utility was proven, turned out to be crucial in its nearly universal adoption as the primary operating system for internet servers.

1.4 Linux Has Distributions

People that say their computer runs Linux usually refer to the kernel, tools, and suite of applications that come bundled together in what is referred to as a distribution.

Take Linux and the GNU tools, add some user-facing applications like a web browser and an email client, and you have a full Linux system. Individuals and even companies started bundling all this software into distributions almost as soon as Linux became usable. The distribution includes tools that take care of setting up the storage, installing the kernel, and installing the rest of the software. The full-featured distributions also include tools to manage the system and a package manager to help you add and remove software after the installation is complete.

Like UNIX, there are distributions suited to every imaginable purpose. There are distributions that focus on running servers, desktops, or even industry-specific tools such as electronics design or statistical computing. The major players in the market can be traced back to either Red Hat, Debian or Slackware. The most visible difference between Red Hat and Debian derivatives is the package manager though there are other differences in everything from file locations to political philosophies.

1.5 Linux Embraces the CLI

There are two basic types of interfaces available that allow you to interact with the operating system. The typical computer user today is most familiar with a graphical user interface (GUI). In a GUI, applications present themselves in windows that can be resized and moved around. There are menus and tools to help users navigate. Graphical applications include web browsers, graphics editing tools and email, to name a few.

Below is an example of a graphical desktop, with a menu bar of popular applications to the left, a LibreOffice document being edited in the foreground and a web browser in the background.
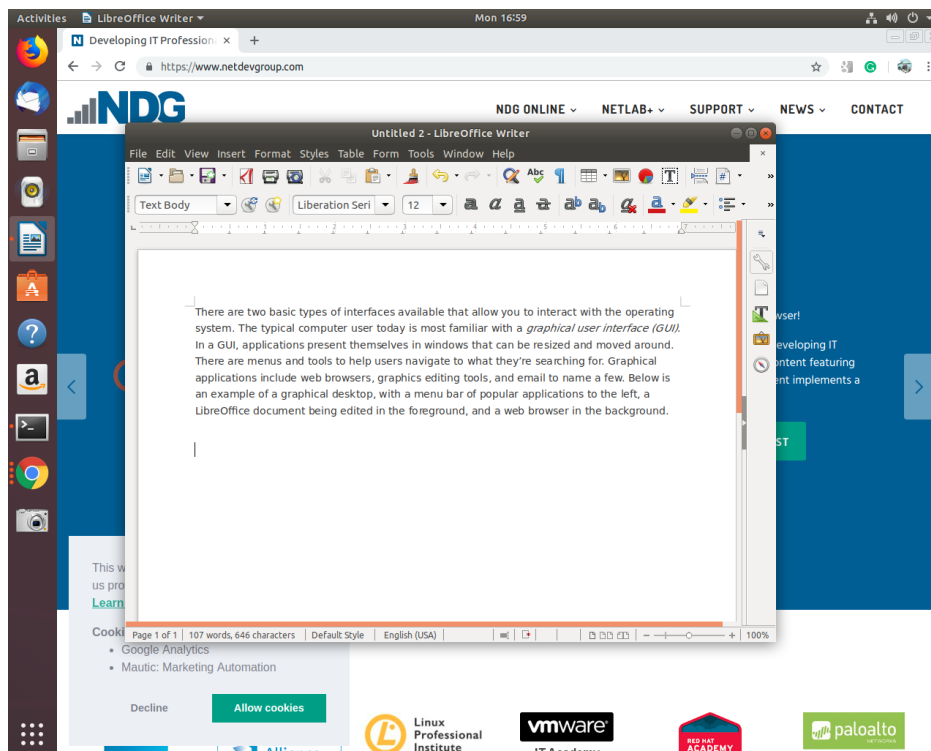
The second type of interface is the command line interface (CLI), a text-based interface to the computer. The CLI relies primarily on keyboard input. Everything the user wants the computer to do is relayed by typing commands rather than clicking on icons. It can be said that when a user clicks on an icon, the computer is telling the user what to do, but, when the user types a command, they are telling the computer what to do.
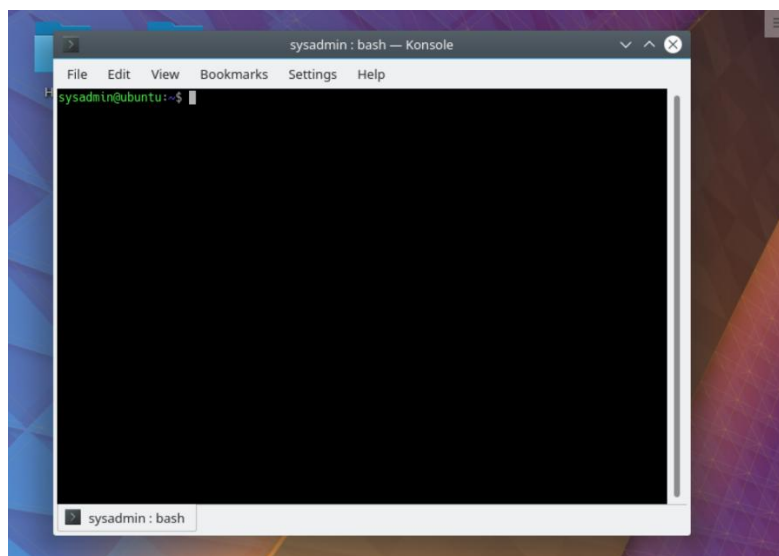
Typically operating systems offer both GUI and CLI interfaces. However, most consumer operating systems (Windows, macOS) are designed to shield the user from the complexity of the CLI. The Linux community is different in that it positively celebrates the CLI for its power, speed and ability to accomplish a vast array of tasks with a single command line instruction. The virtual machines used for the chapters and labs in this course provide a CLI for you to practice on without fear of damaging anything.

When a user first encounters the CLI, they can find it challenging because it requires memorizing a dizzying amount of commands and their options. However, once a user has learned the structure of how commands are used, where the necessary files and directories are located and how to navigate the hierarchy of a filesystem, they can be

immensely productive. This capability provides more precise control, greater speed and the ability to easily automate tasks through scripting.

Furthermore, by learning the CLI, a user can easily be productive almost instantly on ANY distribution of Linux, reducing the amount of time needed to familiarize themselves with a system because of variations in a GUI.
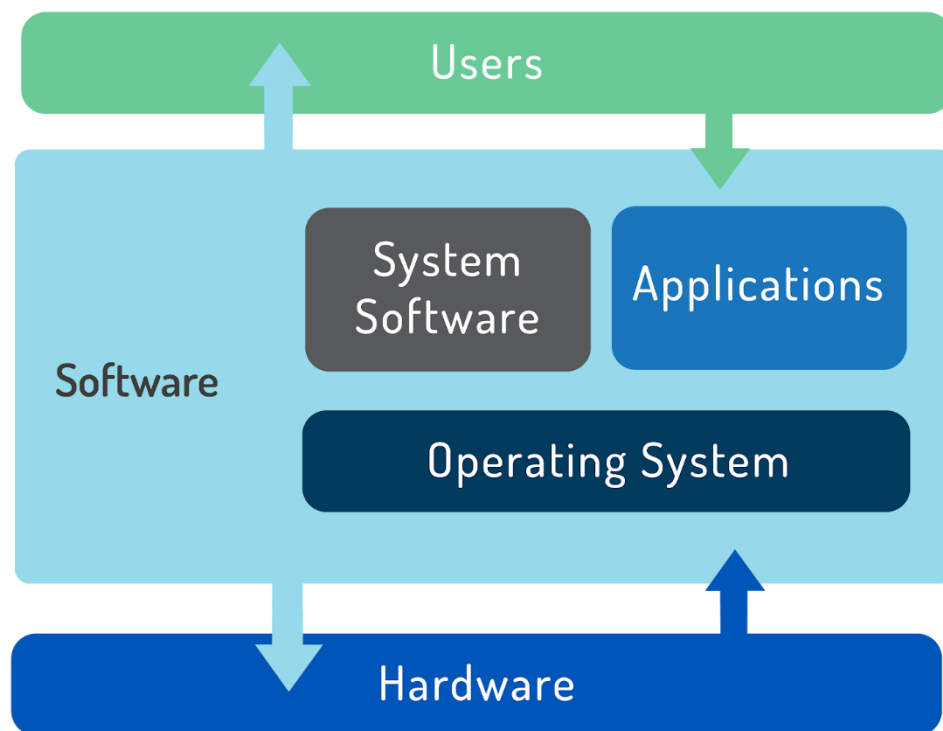
Chapter 2: Operating System

2.1 Operating Systems

An operating system is software that runs on a computing device and manages the hardware and software components that make up a functional computing system.

Modern operating systems don't just manage hardware and software resources, they schedule programs to run in a multi-tasking manner (sharing the processor so that multiple tasks can occur apparently simultaneously), provide standard services that allow users and programs to request something happen (for example a print job) from the operating system, and provided it's properly requested, the operating system will accept the request and perform the function needed.

Desktop and server operating systems are by nature more complex than an operating system that runs on a single-purpose device such as a firewall, or a mobile phone. From a simple set-top box that provides a menu interface for a cable provider, to supercomputers and massive, parallel computing clusters, the generic term operating system is used to describe whatever software is booted and run on that device.

Copyright ©2019 Network Development Group Inc.

Computer users today have a choice mainly between three major operating systems: **Microsoft Windows**, **Apple macOS**, and **Linux**.

Of the three major operating systems listed only Microsoft Windows is unique in its underlying code. Apple's macOS is a fully-qualified UNIX distribution based on BSD Unix (an operating system distributed until 1995), complemented by a large amount of proprietary code. It runs on hardware specifically optimized to work with Apple software. Linux can be any one of hundreds of distribution packages designed or optimized for whatever task is required. Only Microsoft Windows is based on a proprietary code base that isn't either UNIX- or Linux-based.

A user can easily interact with any of these systems by pointing and clicking their way through everyday productivity tasks that all behave similarly regardless of the underlying operating system. Except for Windows, which is mostly administered via the GUI, most system administration tasks are performed using typed commands in a terminal. An administrator that is familiar with UNIX can typically perform tasks on a Linux system and vice versa. Many Linux command line functions also have Microsoft equivalents that administrators use to do their work efficiently.

2.1.1 Decision Points

Role

The first decision when specifying any computer system is the machine's role. Will you be sitting at the console running productivity applications or web browsing? If so, a familiar desktop is best. Will the machine be accessed remotely by many users or provide services to remote users? Then it's a server.

Servers typically sit in a rack and share a keyboard and monitor with many other computers, since console access is generally only used for configuration and troubleshooting. Servers generally run as a CLI, which frees up resources for the real purpose of the computer: serving information to clients (any user or system that accesses resources remotely). Desktop systems primarily run a GUI for the ease of use of their users.

Function

Next, determine the functions of the machine. Is there specific software it needs to run, or specific functions it needs to perform? Will there be hundreds, even thousands, of these machines running at the same time? What is the skill-set of the team managing the computer and software?

Life Cycle

The service lifetime and risk tolerance of the server also needs to be determined. Operating systems and software upgrades come on a periodic basis, called a release cycle. Vendors only support older versions of software for a certain period of time before not offering any updates; this is called a maintenance cycle or life cycle.

In an enterprise server environment, maintenance and release cycles are critical considerations because it is time-consuming and expensive to do major upgrades. Instead, the server hardware itself is often replaced because increased performance is worth the extra expense, and the resources involved are often many times more costly than the hardware.

Consider This

There is a fair amount of work involved in upgrading a server due to specialized configurations, application software patching and user testing, so a proactive organization will seek to maximize their return on investment in both human and monetary capital.

Modern data centers are addressing this challenge through virtualization. In a virtual environment, one physical machine can host dozens, or even hundreds of virtual machines, decreasing space and power requirements, as well as providing for automation of many of the tasks previously done manually by systems administrators. Scripting programs allow virtual machines to be created, configured, deployed and removed from a network without the need for human intervention. Of course, a human still needs to write the script and monitor these systems, at least for now.

The need for physical hardware upgrades has also been decreased immensely with the advent of cloud services providers like **Amazon Web Services**, **Rackspace**, and **Microsoft Azure**. Similar advances have helped desktop administrators manage upgrades in an automated fashion and with little to no user interruption.

Stability

Individual software releases can be characterized as beta or stable depending on where they are in the release cycle. When a software release has many new features that haven't been tested, it's typically referred to as beta. After being tested in the field, its designation changes to stable.

Users who need the latest features can decide to use beta software. This is often done in the development phase of a new deployment and provides the ability to request features not available on the stable release.

Production servers typically use stable software unless needed features are not available, and the risk of running code that has not been thoroughly tested is outweighed by the utility provided.

Software in the open source realm is often released for peer review very early on in its development process, and can very quickly be put into testing and even production environments, providing extremely useful feedback and code submissions to fix issues found or features needed.

Conversely, proprietary software will often be kept secret for most of its development, only reaching a public beta stage when it's almost ready for release.

Compatibility

Another loosely-related concept is backward compatibility which refers to the ability of later operating systems to be compatible with software made for earlier versions. This is usually a concern when it is necessary to upgrade an operating system, but an application software upgrade is not possible due to cost or lack of availability.

The norm for open source software development is to ensure backward compatibility first and break things only as a last resort. The common practice of maintaining and versioning libraries of functions helps this greatly. Typically, a library that is used by one or more programs is versioned as a new release when significant changes have occurred but also keeps all the functions (and compatibility) of earlier versions that may be hard-coded or referred to by existing software.

Cost

Cost is always a factor when specifying new systems. Microsoft has annual licensing fees that apply to users, servers and other software, as do many other software companies. Ultimately, the choice of operating system will be affected by available hardware, staff resources and skill, cost of purchase, maintenance, and projected future requirements.

Virtualization and outsourced support services offer the modern IT organization the promise of having to pay for only what it uses rather than building in excess capacity. This not only controls costs but offers opportunities for people both inside and outside the organization to provide expertise and value.

Interface

The first electronic computer systems were controlled by means of switches and plugboards similar to those used by telephone operators at the time. Then came punch cards and finally a text-based terminal system similar to the Linux command line interface (CLI) in use today. The graphical user interface (GUI), with a mouse and buttons to click, was pioneered at Xerox PARC (Palo Alto Research Center) in the early 1970s and popularized by Apple Computer in the 1980s.

Today, operating systems offer both GUI and CLI interfaces, however, most consumer operating systems (Windows, macOS) are designed to shield the user from the ins and outs of the CLI.

2.2 Microsoft Windows

Microsoft offers different versions of its operating system according to the machine's role: desktop or server? The desktop version of Windows has undergone various naming schemes with the current version (as of this writing) being simply **Windows 11**. While new versions of most Linux distributions come out twice a year, around March and September, new versions of Windows tend to be released only every few years. In all, there have been 16 versions of Windows since 1985. Backward compatibility is a priority for Microsoft, even going so far as to bundle virtual machine technology so that users can run older software.

Windows Server currently (as of this writing) is at version **2019** to denote the release date. The server can run a GUI but recently Microsoft, largely as a competitive response to Linux, has made incredible strides in its command line scripting capabilities through PowerShell and Windows Subsystem for Linux (WSL). There is also an optional Desktop Experience package which mimics a standard productivity machine. Microsoft also actively encourages enterprise customers to incorporate its Azure cloud service.

2.3 Apple macOS

Apple makes the **macOS** operating system, which is partially based on software from the FreeBSD project and has undergone UNIX certification. macOS is well known for being "easy to use", and as such has continued to be favored by users with limited access to IT resources like schools and small businesses. It is also very popular with programmers due to its robust UNIX underpinnings.

On the server side, **macOS Server** is primarily aimed at smaller organizations. This low-cost addition to macOS desktop allows users to collaborate, and administrators to control access to shared resources. It also provides integration with iOS devices like the iPhone and iPad.

Some large corporate IT departments allow users to choose macOS since users often require less support than standard Microsoft productivity deployments. The continued popularity of macOS has ensured healthy support from software vendors. macOS is also quite popular in the creative industries such as graphics and video production. For many of these users, application choice drives the operating system decision. Apple hardware, being integrated so closely with the operating system, and their insistence on adherence to standards in application programming gives these creative professionals a stable platform to perform many computing-intense functions with fewer concerns about compatibility.

2.4 Linux

Linux users typically obtain an operating system by downloading a distribution. A Linux distribution is a bundle of software, typically comprised of the Linux kernel, utilities, management tools, and even some application software in a package which also includes the means to update core software and install additional applications.

The distribution takes care of setting up the storage, building the kernel and installing hardware drivers, as well as installing applications and utilities to make a fully functional computer system. The organizations that create distributions also include tools to manage the system, a package manager to add and remove software, as well as update programs to provide security and functionality patches.

The number of Linux distributions available numbers in the hundreds, so the choice can seem daunting at first. However, the decision points are mostly the same as those highlighted for choosing an operating system.

Role

With Linux, there are multiple options to choose from depending on organizational needs. The variety of distributions and accompanying software allows the operating system to be significantly more flexible and customizable. Distributions are available for a much wider variety of systems, from commercial offerings for the traditional server or desktop roles, to specialized distributions designed to turn an old computer into a network firewall; from distributions created to power a supercomputer, to those that enable embedded systems. These might focus on running application or web servers, productivity desktops, point-of-sale systems, or even tools dedicated to electronics design or statistical computing.

Function

Governments and large enterprises may also limit their choices to distributions that offer commercial support because paying for another tier of support may be better than risking extensive outages. For the most part, concerns over security have been addressed through the large open source community, which monitors kernel changes for vulnerabilities and provides bug reporting and fixes at a much larger scale than closed source vendors can achieve.

Support for necessary applications may vary and is, therefore, an additional consideration. Often application vendors choose a subset of distributions to support. Different distributions have different versions of key libraries, and it is difficult for a company to support all these different versions. However, some applications like Firefox and LibreOffice are widely supported and available for all major distributions.

Life Cycle

Most distributions have both major and minor update cycles to introduce new features and fix existing bugs. Additionally, there are development packages where users can contribute code and submit patches for possible inclusion into new releases.

Linux distributions can be broadly classed in two main categories: enthusiast and enterprise. An enthusiast distribution such as openSUSE's Tumbleweed has a fast update cycle, is not supported for enterprise and may not contain (or drop) features or software in the next version that are in the current one. Red Hat's Fedora project uses a similar method of development and release cycle, as does Ubuntu Desktop.

Enterprise distributions are almost the exact opposite, in that they take care to be stable and consistent, and offer enterprise-grade support for extended periods, anywhere from 5-13 years in the case of SUSE. Enterprise distributions are fewer by far, being offered mainly by Red Hat, Canonical and SUSE.

Application software may be written such that it only supports a specific release of a distribution, requiring users to remain on an older, less secure operating system than they might like. Therefore, some Linux releases are considered to have long-term support (LTS) of 5 years or more while others are only supported for two years or less.

Stability

Some distributions offer stable, testing, and unstable releases. When choosing an unstable release for required features, consideration must be given to the fact that those features may change at any point during the development cycle. When features have been integrated into the system for a long time, with most of the bugs and issues addressed, the software moves through testing into the stable release.

Other releases depend on beta distributions. For instance, the Fedora distribution releases beta or pre-releases of its software ahead of the full release to minimize bugs. Fedora is often considered the community-oriented beta release of RedHat. Features are added and changed in the Fedora release before finding their way into the enterprise-ready RedHat distribution.

openSUSE and its enterprise counterpart SLES (SUSE Linux Enterprise Server) are similar, in that the community edition is used as a testing ground for the features and functions that will eventually be migrated into the enterprise version. Previously somewhat dissimilar, later versions of the openSUSE and SLES distribution codebases are nearly identical, allowing for easier migration of features and code from one to the other.

Consider This

The Debian distribution warns users about the pitfalls of using the "sid" (unstable) release with the following warning:

'"sid" is subject to massive changes and in-place library updates. This can result in a very "unstable" system which contains packages that cannot be installed due to missing libraries, dependencies that cannot be fulfilled, etc. Use it at your own risk!'

Cost

Your chosen Linux distribution itself might be zero cost, but paying for support may be worthwhile depending on organizational needs and capabilities.

Interface

Like most operating systems, Linux can be used in one of two ways: graphical (GUI) and non-graphical (CLI). Below is an example of a graphical desktop, with a menu bar of popular applications to the left, a LibreOffice document being edited in the foreground, and a web browser in the background.
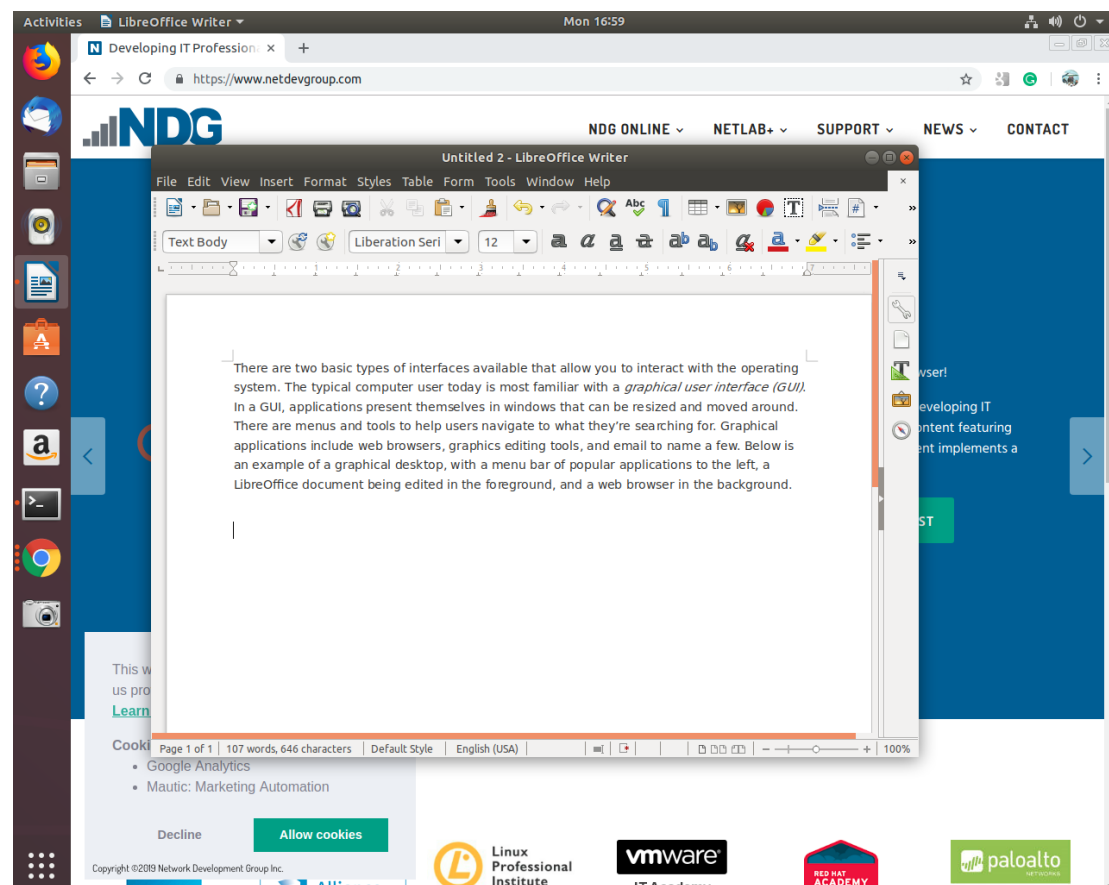
In graphical mode, users can have several different windows with terminal applications (shells) open, which is very helpful when performing tasks on multiple remote computers. Administrators and users can log-in with their username and password through a graphical interface.

The second type of interface, the CLI, is a text-based interface to the computer, where the user types in a command and the computer then executes it. The CLI environment is provided by an application on the computer known as a terminal. The terminal accepts what the user types and passes to a shell. The shell interprets what the user has typed into instructions that can be executed by the operating system. If output is produced by the command, then this text is displayed in the terminal. If problems with the command are encountered, then an error message is displayed.

The CLI starts with a text-based login as shown below. In a successful login, after being prompted for a username and password, you are taken to a CLI shell customized for the particular user.

```
ubuntu 18.04 ubuntu tty2


ubuntu login:
```

In CLI mode there are no windows to move around. Text editors, web browsers, and email clients are all presented in text format only. This is how UNIX operated before graphical environments were the norm. Most servers run in this mode too, since people don't log into them directly, making a graphical interface a waste of resources. Here is an example of a CLI screen after logging in:

```
ubuntu 18.04 ubuntu tty2


ubuntu login: sue

Password:


The programs included with the Ubuntu system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/*/copyright.



Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by

applicable law.


Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.4.0-72-generic x86_64)


 * Documentation:  https://help.ubuntu.com/


212 packages can be updated.

91 updates are security updates.


sue@ubuntu:~$ w

 17:27:22 up 14 min,  2 users,  load average: 1.73, 1.83, 1.69
```

```
USER      TTY      FROM              LOGIN@    IDLE   JCPU   PCPU WHAT
sue     tty2                        20:08    14.35  0.05s  0.00s w
```

The original login prompt is at the top, with newer text added below. During login there may be some text displayed called the message of the day (MOTD). This is an opportunity for the systems administrator to pass information to users, or just make a silly joke. Following the MOTD is the command prompt, in the example above, the user has entered the `w` command which shows who is logged in. As new commands are entered and processed, the window scrolls up and older text is lost across the top. The terminal itself is responsible for keeping any history, such as to allow the user to scroll up and see previously entered commands. As far as Linux is concerned, what is on the screen is all that there is. There's nothing to move around.

2.4.1 Linux Distributions

Red Hat

Red Hat started as a simple distribution that introduced the Red Hat Package Manager (RPM). The developer eventually formed a company around it, which tried to commercialize a Linux desktop for business. Over time, Red Hat started to focus more on the server applications, such as web- and file-serving and released **Red Hat Enterprise Linux (RHEL)**, which was a paid service on a long release cycle. The release cycle dictates how often software is upgraded. A business may value stability and want long release cycles, while a hobbyist or a startup may want the latest software and opt for a shorter release cycle. To satisfy the latter group, Red Hat sponsors the **Fedora Project** which makes a personal desktop comprising the latest software but is still built on the same foundations as the enterprise version.

Because everything in Red Hat Enterprise Linux is open source, a project called **CentOS** came to be. It recompiled all the RHEL packages (converting their source code from the programming language they were written into language usable by the system) and gave them away for free. CentOS and others like it (such as Scientific Linux) are largely compatible with RHEL and integrate some newer software, but do not offer the paid support that Red Hat does.

Scientific Linux is an example of a specific-use distribution based on Red Hat. The project is a Fermilab-sponsored distribution designed to enable scientific computing. Among its many applications, Scientific Linux is used with particle accelerators including the Large Hadron Collider at CERN.

SUSE

SUSE, originally derived from **Slackware**, was one of the first comprehensive Linux distributions, it has many similarities to Red Hat Enterprise Linux. The original company was purchased by Novell in 2003, which was then purchased by the Attachmate Group in 2011. The Attachmate group then merged with Micro Focus International in 2014, and in 2018 SUSE announced plans to go forward as an independent business. Through all of the mergers and acquisitions, SUSE has managed to continue and grow.

While SUSE Linux Enterprise contains proprietary code and is sold as a server product, **openSUSE** is a completely open, free version with multiple desktop packages similar to CentOS and Linux Mint.

Debian

Debian is more of a community effort, and as such, also promotes the use of open source software and adherence to standards. Debian came up with its own package

management system based on the `.deb` file format. While Red Hat leaves non-Intel and AMD platform support to derivative projects, Debian supports many of these platforms directly.

Ubuntu is the most popular Debian-derived distribution. It is the creation of **Canonical**, a company that was made to further the growth of Ubuntu and makes money by providing support. Ubuntu has several different variants for desktop, server and various specialized applications. They also offer an LTS version that is kept up-to-date for 3 years on desktops and 5 years on servers, which gives developers and the companies they work for confidence to build solutions based on a stable distribution.

Linux Mint was started as a fork of Ubuntu Linux, while still relying upon the Ubuntu repositories. There are various versions, all free of cost, but some include proprietary codecs, which cannot be distributed without license restrictions in certain countries.

Android

Linux is a kernel, and many of the commands covered in this course are actually part of the GNU package. That is why some people insist on using the term **GNU/Linux** instead of Linux alone.

Android, sponsored by Google, is the world's most popular Linux distribution. It is fundamentally different from its counterparts. Android uses the **Dalvik** virtual machine with Linux, providing a robust platform for mobile devices such as phones and tablets. However, lacking the traditional packages that are often distributed with Linux (such as GNU and Xorg), Android is generally incompatible with desktop Linux distributions.
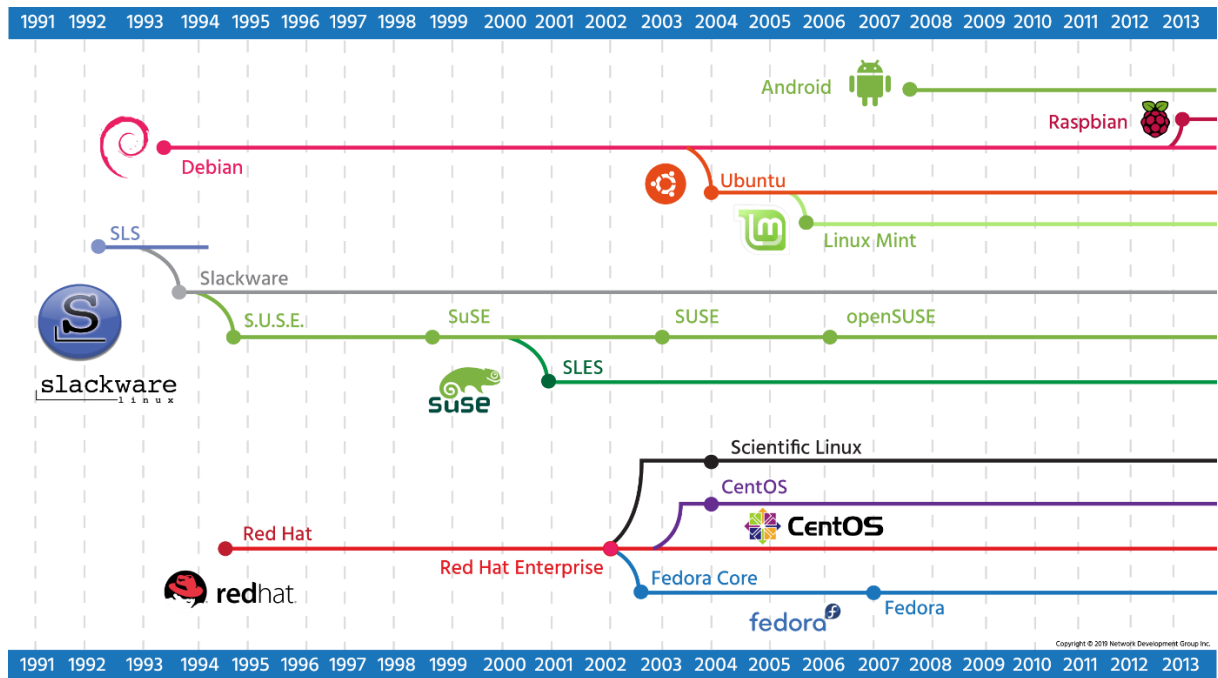
This incompatibility means that a Red Hat or Ubuntu user cannot download software from the Google Play store. Likewise, a terminal emulator in Android lacks many of the commands of its Linux counterparts. It is possible, however, to use BusyBox with Android to enable most commands to work.

Other

Raspbian is a specialized Linux distribution optimized to run on **Raspberry Pi** hardware. This combination has seen significant use in training for programmers and hardware designers at all levels. Its low cost and ease of use have made it a favorite of educators worldwide, and many add-on devices are available to extend its capabilities into the physical world. There is a multitude of labs and projects available that teach everything from environmental monitoring to circuit design, machine learning, and robotics.

Linux From Scratch (LFS) is more of a learning tool than a working distribution. This project consists of an online book, and source code, with "step-by-step instructions" for building a custom Linux distribution from the source code up. This "distribution" embodies the true spirit of Linux whereby users can modify any aspect of the operating system and learn how all the pieces work together. It's also a good starting point for anyone who needs specialized functionality or an ultra-compact build for an embedded system project.

We have discussed the distributions explicitly mentioned in the Linux Essentials objectives. Be aware that there are hundreds, if not thousands more that are available. While there are many different distributions of Linux, many of the programs and commands remain the same or are very similar.

| 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 |

2.4.2 Embedded Systems

Linux started out as something that would only run on a computer like Linus Torvald's: an Intel 386 PC with a specific hard drive controller, but since anyone could add to or change Linux, people started building support for other hardware. Eventually, Linux started supporting other chips with an emphasis on small size and low power consumption.

Because of this flexibility, a significant number of device makers have used Linux as the operating system for their hardware products. Today we call these embedded systems because they are designed to do a specific task on hardware optimized for only that purpose. These systems encompass a tremendous diversity of devices that are used today, from cell phones to smart TVs and appliances, to remote monitoring systems for pipelines and factories.

As Linux evolved, specialized processor chips were developed for consumer and industrial devices to take advantage of its capabilities. Support for Linux has become so ubiquitous that it is possible to prototype and bring to market new devices using off-the-shelf components. The rise of cheap, small, adaptable single-board computers like the Raspberry Pi has given experimenters and entrepreneurs everywhere tools to quickly build custom solutions, powered by Linux, that would have taken months of work by specialized teams just a few years ago.

While consumers are familiar with embedded Linux entertainment devices like digital video recorders (DVRs) and "smart TVs," the real impact of embedded Linux is just starting to be realized. The internet of things (IoT) is just ramping up with cheap, ubiquitous devices being deployed on everything from oil wells to solar generating farms. These networks of smart sensors and controllers enable engineers to adjust critical processes in real time while monitoring and reporting back to central control stations. As more processes are being monitored and more data is being integrated with machine learning and artificial intelligence (AI) we can anticipate gains in efficiency, safety and productivity only dreamed of by past generations.

Chapter 3: Working in Linux

3.1 Navigating the Linux Desktop

To be a Linux systems administrator, it is necessary to be comfortable with Linux as a desktop operating system and have proficiency with basic Information and Communication Technology (ICT) skills. Using Linux for productivity tasks, rather than depending on Windows or Macintosh systems, accelerates learning by working with Linux tools on a daily basis. Systems administrators do far more than manage servers; they are often called upon to assist users with configuration issues, recommend new software, and update documentation among other tasks.

3.1.1 Getting to the Command Line

The command line interface (CLI) is a simple text input system for entering anything from single-word commands to complicated scripts. Most operating systems have a CLI that provides a direct way of accessing and controlling the computer.

3.2 Applications

The kernel of the operating system is like an air traffic controller at an airport, and the applications are the airplanes under its control. The kernel decides which program gets which blocks of memory, it starts and kills applications, and it handles displaying text or graphics on a monitor.

Applications make requests to the kernel and in return receive resources, such as memory, CPU, and disk space. If two applications request the same resource, the kernel decides which one gets it, and in some cases, kills off another application to save the rest of the system and prevent a crash.

The kernel also abstracts some complicated details away from the application. For example, the application doesn't know if a block of disk storage is on a solid-state drive, a spinning metal hard disk, or even a network file share. Applications need only follow the kernel's Application Programming Interface (API) and therefore don't have to worry about the implementation details. Each application behaves as if it has a large block of memory on the system; the kernel maintains this illusion by remapping smaller blocks of memory, sharing blocks of memory with other applications, or even swapping out untouched blocks to disk.

The kernel also handles the switching of applications, a process known as multitasking. A computer system has a small number of central processing units (CPUs) and a finite amount of memory. The kernel takes care of unloading one task and loading a new one if there is more demand than resources available. When one task has run for a specified amount of time, the CPU pauses it so that another may run. If the computer is doing several tasks at once, the kernel is deciding when to switch focus between tasks. With the tasks rapidly switching, it appears that the computer is doing many things at once.

When we, as users, think of applications, we tend to think of word processors, web browsers, and email clients, however, there are a large variety of application types. The kernel doesn't differentiate between a user-facing application, a network service that talks to a remote computer, or an internal task. From this, we get an abstraction called a process. A process is just one task that is loaded and tracked by the kernel. An application may even need multiple processes to function, so the kernel takes care of

running the processes, starting and stopping them as requested, and handing out system resources.

3.2.1 Major Applications

The Linux kernel can run a wide variety of software across many hardware platforms. A computer can act as a server, which means it primarily handles data on others' behalf, or as a desktop, which means a user interacts with it directly. The machine can run software or be used as a development machine in the process of creating software. A machine can even adopt multiple roles as Linux makes no distinction; it's merely a matter of configuring which applications run.

3.2.2 Server Applications

Linux excels at running server applications because of its reliability and efficiency. The ability to optimize server operating systems with just needed components allows administrators to do more with less, a feature loved by startups and large enterprises alike.

3.2.2.1 Web Servers

One of the early uses of Linux was for web servers. A web server hosts content for web pages, which are viewed by a web browser using the **HyperText Transfer Protocol (HTTP)** or its encrypted flavor, **HTTPS**. The web page itself can either be static or dynamic. When the web browser requests a static page, the web server sends the file as it appears on disk. In the case of a dynamic site, the request is sent by the web server to an application, which generates the content.

WordPress is one popular example. Users can develop content through their browser in the WordPress application, and the software turns it into a fully functional dynamic website.

Apache is the dominant web server in use today. Apache was originally a standalone project, but the group has since formed the **Apache Software Foundation** and maintains over a hundred open source software projects. **Apache HTTPD** is the daemon, or server application program, that "serves" web page requests.

Another web server is **NGINX**, which is based out of Russia. It focuses on performance by making use of more modern UNIX kernels and only does a subset of what Apache can do. Over 65% of websites are powered by either NGINX or Apache.

3.2.2.2 Private Cloud Servers

As individuals, organizations, and companies start to move their data to the cloud, there is a growing demand for private cloud server software that can be deployed and administered internally.

.2.2.3 Database Servers

Database server applications form the backbone of most online services. Dynamic web applications pull data from and write data to these applications. For example, a web program for tracking online students might consist of a front-end server that presents a web form. When data is entered into the form, it is written to a database application such as **MariaDB**. When instructors need to access student information, the web application queries the database and returns the results through the web application.

MariaDB is a community-developed fork of the **MySQL** relational database management system. It is just one of many database servers used for web development as different requirements dictate the best application for the required tasks.

A database stores information and also allows for easy retrieval and querying. Some other popular databases are **Firebird** and **PostgreSQL**. You might enter raw sales figures into the database and then use a language called **Structured Query Language (SQL)** to aggregate sales by product and date to produce a report.

3.2.2.4 Email Servers

Email has always been a widespread use for Linux servers. When discussing email servers, it is always helpful to look at the 3 different tasks required to get email between people:

- **Mail Transfer Agent (MTA)**

  The most well known MTA (software that is used to transfer electronic messages to other systems) is **Sendmail**. **Postfix** is another popular one and aims to be simpler and more secure than Sendmail.

- **Mail Delivery Agent (MDA)**

  Also called the **Local Delivery Agent**, it takes care of storing the email in the user's mailbox. Usually invoked from the final MTA in the chain.

- **POP/IMAP Server**

  The **Post Office Protocol (POP)** and **Internet Message Access Protocol (IMAP)** are two communication protocols that let an email client running on your computer talk to a remote server to pick up the email.

Dovecot is a popular POP/IMAP server owing to its ease of use and low maintenance. **Cyrus IMAP** is another option. Some POP/IMAP servers implement their own mail database format for performance and include the MDA if the custom database is desired. People using standard file formats (such as all the emails in one text file) can choose any MDA.

There are several significant differences between the closed source and open source software worlds, one being that of inclusion of other projects as components to a project or package.In the closed source world, **Microsoft Exchange** is shipped primarily as a software package/suite that includes all the necessary or approved components, all from Microsoft, so there are few if any options to make individual selections. In the open source world, many options can be modularly included or swapped out for package components, and indeed some software packages or suites are just a well-packaged set of otherwise individual components all harmoniously working together.

3.2.2.5 File Sharing

For Windows-centric file sharing, **Samba** is the clear winner. Samba allows a Linux machine to look and behave like a Windows machine so that it can share files and participate in a Windows domain. Samba implements the server components, such as making files available for sharing and certain Windows server roles, and also the client end so that a Linux machine may consume a Windows file share.

The **Netatalk** project lets a Linux machine perform as an Apple Macintosh file server. The native file sharing protocol for UNIX/Linux is called the **Network File System (NFS)**.

NFS is usually part of the kernel which means that a remote file system can be mounted (made accessible) just like a regular disk, making file access transparent to other applications.

As a computer network becomes more substantial, the need for a directory increases. One of the oldest network directory systems is the **Domain Name System (DNS)**. It is used to convert a name like https://www.icann.org/ to an IP address like 192.0.43.7, which is a unique identifier of a computer on the Internet. DNS also holds global information like the address of the MTA for a given domain name. An organization may want to run their own DNS server to host their public-facing names, and also to serve as an internal directory of services. The **Internet Software Consortium** maintains the most popular DNS server, simply called bind after the name of the process that runs the service.

The DNS is focused mainly on computer names and IP addresses and is not easily searchable. Other directories have sprung up to store information such as user accounts and security roles. The **Lightweight Directory Access Protocol (LDAP)** is one common directory system which also powers Microsoft's Active Directory. In LDAP, an object is stored in a tree, and the position of that object on the tree can be used to derive information about the object and what it stores. For example, a Linux administrator may be stored in a branch of the tree called "IT Department," which is under a branch called "Operations." Thus one can find all the technical staff by searching under the "IT Department" branch. **OpenLDAP** is the dominant program used in Linux infrastructure.

One final piece of network infrastructure to discuss here is called the **Dynamic Host Configuration Protocol (DHCP)**. When a computer boots up, it needs an IP address for the local network so it can be uniquely identified. DHCP's job is to listen for requests and to assign a free address from the DHCP pool. The Internet Systems Consortium (known until January 2004 as the Internet Software Consortium) also maintains the **ISC DHCP** server, which is the most common open source DHCP server.

3.2.3 Desktop Applications

The Linux ecosystem has a wide variety of desktop applications. There are games, productivity applications, creative tools, web browsers and more.

3.2.3.1 Email

The Mozilla Foundation came out with **Thunderbird**, a full-featured desktop email client. Thunderbird connects to a POP or IMAP server, displays email locally, and sends email through an external SMTP server.

Other notable email clients are **Evolution** and **KMail** which are the GNOME and KDE projects' email clients. Standardization through POP and IMAP and local email formats means that it's easy to switch between email clients without losing data.

3.2.3.2 Creative

For the creative types, there is **Blender**, **GIMP (GNU Image Manipulation Program)**, and **Audacity** which handle 3D movie creation, 2D image manipulation, and audio editing respectively. They have had various degrees of success in professional markets. Blender is used for everything from independent films to Hollywood movies, for example. GIMP supports high-quality photo manipulation, original artwork creation, graphic design elements, and is extensible through scripting in multiple languages. Audacity is a free and open source audio editing tool that is available on multiple operating systems.

3.2.3.3 Productivity

Use of common open source applications in presentations and projects is one way to strengthen Linux skills. The basic productivity applications, such as a word processor, spreadsheet, and presentation package are valuable assets. Collectively they're known as an office suite, primarily due to Microsoft Office, the dominant player in the market.

LibreOffice is a fork of the **OpenOffice** (sometimes called **OpenOffice.org**) application suite. Both offer a full office suite, including tools that strive for compatibility with Microsoft Office in both features and file formats.

Shown below is the spreadsheet and the document editor of LibreOffice. Note how the spreadsheet, **LibreOffice Calc**, is not limited to rows and columns of numbers. The numbers can be the source of a graph, and formulas can be written to calculate values based on information, such as pulling together interest rates and loan amounts to help compare different borrowing options.

Using **LibreOffice Writer**, a document can contain text, graphics, data tables, and much more. You can link documents and spreadsheets together, for example, so that you can summarize data in a written form and know that any changes to the spreadsheet will be reflected in the document.
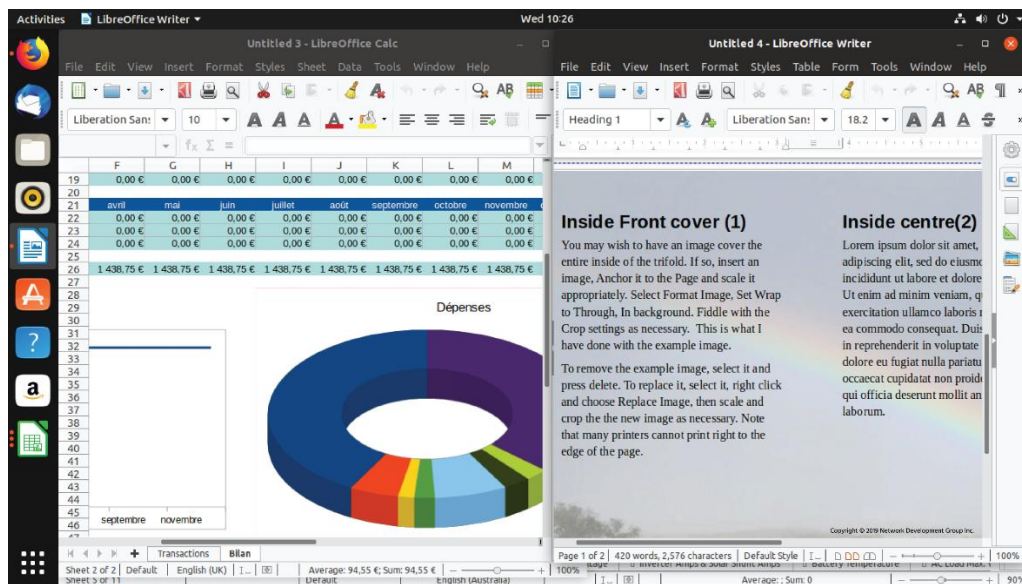
LibreOffice can also work with other file formats, such as Microsoft Office or **Adobe Portable Document Format (PDF)** files. Additionally, through the use of extensions, LibreOffice can be made to integrate with Wiki software to give you a powerful intranet solution.

3.2.3.4 Web Browsers

Linux is a first class citizen for the **Mozilla Firefox** and **Google Chrome** browsers. Both are open source web browsers that are fast, feature-rich, and have excellent support for web developers. These packages are an excellent example of how competition helps to drive open source development – improvements made to one browser spur the development of the other browser. As a result, the Internet has two excellent browsers that push the limits of what can be done on the web, and work across a variety of platforms. Using a browser, while second nature for many, can lead to privacy concerns. By understanding and modifying the configuration options, one can limit the amount of information they share while searching the web and saving content.

3.3 Console Tools

Historically, the development of UNIX shows considerable overlap between the skills of software development and systems administration. The tools for managing systems have features of computer languages such as loops (which allow commands to be carried out repeatedly), and some computer programming languages are used extensively in automating systems administration tasks. Thus, one should consider these skills complementary, and at least a basic familiarity with programming is required for competent systems administrators.

3.3.1 Shells

At the basic level, users interact with a Linux system through a shell whether connecting to the system remotely or from an attached keyboard. The shell's job is to accept commands, like file manipulations and starting applications, and to pass those to the Linux kernel for execution. The Linux shell provides a rich language for iterating over files and customizing the environment, all without leaving the shell. For example, it is possible to write a single command line that finds files with contents matching a specific pattern, extracts useful information from the file, then copies the new information to a new file.

Linux offers a variety of shells to choose from, mostly differing in how and what can be customized, and the syntax of the built-in scripting language. The two main families are the **Bourne shell** and the **C shell**. The Bourne shell was named after its creator Stephen Bourne of Bell Labs. The C shell was so named because its syntax borrows heavily from the C language. As both these shells were invented in the 1970s, there are more modern versions, the **Bourne Again Shell (Bash)** and the **tcsh** (pronounced as tee-cee-shell). Bash is the default shell on most systems, though tcsh is also typically available.

Programmers have taken favorite features from Bash and tcsh and made other shells, such as the **Korn shell (ksh)** and the **Z shell (zsh)**. The choice of shells is mostly a personal one; users who are comfortable with Bash can operate effectively on most Linux systems. Other shells may offer features that increase productivity in specific use cases.

3.3.2 Text Editors

Most Linux systems provide a choice of text editors which are commonly used at the console to edit configuration files. The two main applications are **Vi** (or the more modern **Vim**) and **Emacs**. Both are remarkably powerful tools to edit text files; they differ in the format of the commands and how plugins are written for them. Plugins can be anything from syntax highlighting of software projects to integrated calendars.

Both Vi and Emacs are complex and have a steep learning curve, which is not helpful for simple editing of a small text file. Therefore, **Pico** and **Nano** are available on most systems and provide very basic text editing.

Consider This

The Nano editor was developed as a completely open source editor that is loosely based on Pico, as the license for Pico is not an open source license and forbids making changes and distributing it.

While Nano is simple and easy to use, it doesn't offer the extensive suite of more advanced editing and key binding features that an editor like Vi does. Administrators should strive to gain some basic familiarity with Vi, though, because it is available on almost every Linux system in existence. When restoring a broken Linux system by

running in the distribution's recovery mode, Vi can be a critical tool, and the best time to learn Vim or any editor is before you desperately need it to fix a broken system.

3.4 Package Management

Every Linux system needs to add, remove, and update software. In the past this meant downloading the source code, setting it up, compiling it, and copying files onto each system that required updating. Thankfully, modern distributions use packages, which are compressed files that bundle up an application and its dependencies (or required files), greatly simplifying the installation by making the right directories, copying the proper files into them, and creating such needed items as symbolic links.

A package manager takes care of keeping track of which files belong to which package and even downloading updates from repositories, typically a remote server sharing out the appropriate updates for a distribution. In Linux, there are many different software package management systems, but the two most popular are those from Debian and Red Hat.

3.4.1 Debian Package Management

The Debian distribution, and its derivatives such as Ubuntu and Mint, use the Debian package management system. At the heart of Debian package management are software packages that are distributed as files ending in the `.deb` extension.

The lowest-level tool for managing these files is the `dpkg` command. This command can be tricky for novice Linux users, so the **Advanced Package Tool**, `apt-get` (a front-end program to the `dpkg` tool), makes management of packages easier. Additional command line tools which serve as front-ends to `dpkg` include `aptitude` and GUI front-ends like **Synaptic** and **Software Center**.

3.4.2 RPM Package Management

The **Linux Standards Base**, which is a **Linux Foundation** project, is designed to specify (through a consensus) a set of standards that increase the compatibility between conforming Linux systems. According to the Linux Standards Base, the standard package management system is RPM.

RPM makes use of an `.rpm` file for each software package. This system is what distributions derived from Red Hat, including Centos and Fedora, use to manage software. Several other distributions that are not Red Hat derived, such as SUSE, OpenSUSE, and Arch, also use RPM.

Like the Debian system, RPM Package Management systems track dependencies between packages. Tracking dependencies ensures that when a package is installed, the system also installs any packages needed by that package to function correctly. Dependencies also ensure that software updates and removals are performed properly.

The back-end tool most commonly used for RPM Package Management is the `rpm` command. While the `rpm` command can install, update, query and remove packages, the command line front-end tools such as `yum` and `up2date` automate the process of resolving dependency issues.

Note

A back-end program or application either interacts directly with a front-end program or is "called" by an intermediate program. Back end programs would not interact directly with

the user. Basically, there are programs that interact with people (front-end) and programs that interact with other programs (back-end).

There are also GUI-based front-end tools such as **Yumex** and **Gnome PackageKit** that also make RPM package management easier.

Some RPM-based distributions have implemented the **ZYpp** (or **libzypp**) package management style, mostly openSUSE and SUSE Linux Enterprise, but mobile distributions MeeGo, Tizen and Sailfish as well.

The `zypper` command is the basis of the ZYpp method, and it features short and long English commands to perform functions, such as `zypper in packagename` which installs a package including any needed dependencies.

Most of the commands associated with package management require root privileges. The rule of thumb is that if a command affects the state of a package, administrative access is required. In other words, a regular user can perform a query or a search, but to add, update or remove a package requires the command to be executed as the root user.

3.5 Development Languages

It should come as no surprise that as software built on contributions from programmers, Linux has excellent support for software development. The shells are built to be programmable, and there are powerful editors included on every system. There are also many development tools available, and many modern programming languages treat Linux as a first-class citizen.

Computer programming languages provide a way for a programmer to enter instructions in a more human readable format, and for those instructions to eventually become translated into something the computer understands. Languages fall into one of two camps: interpreted or compiled. An interpreted language translates the written code into computer code as the program runs, and a compiled language is translated all at once.

Linux itself was written in a compiled language called **C**. The main benefit of C is that the language itself maps closely to the generated machine code so that a skilled programmer can write code that is small and efficient. When computer memory was measured in kilobytes, this was very important. Even with large memory sizes today, C is still helpful for writing code that must run fast, such as an operating system.

C has been extended over the years. There is **C++**, which adds object support to C (a different style of programming), and **Objective C** that took another direction and is in heavy use in Apple products.

The **Java** language puts a different spin on the compiled approach. Instead of compiling to machine code, Java first imagines a hypothetical CPU called the **Java Virtual Machine (JVM)** and then compiles all the code to that. Each host computer then runs JVM software to translate the JVM instructions (called bytecode) into native instructions.

The additional translation with Java might make you think it would be slow. However, the JVM is relatively simple so it can be implemented quickly and reliably on anything from a powerful computer to a low power device that connects to a television. A compiled Java file can also be run on any computer implementing the JVM!

Another benefit of compiling to an intermediate target is that the JVM can provide services to the application that usually wouldn't be available on a CPU. Allocating memory to a program is a complex problem, but it's built into the JVM. As a result, JVM makers can focus their improvements on the JVM as a whole, so any progress they make is instantly available to applications.

Interpreted languages, on the other hand, are translated to machine code as they execute. The extra computer power spent doing this can often be recouped by the

increased productivity the programmer gains by not having to stop working to compile. Interpreted languages also tend to offer more features than compiled languages, meaning that often less code is needed. The language interpreter itself is usually written in another language such as C, and sometimes even Java! This means that an interpreted language is being run on the JVM, which is translated at runtime into actual machine code.

JavaScript is a high-level interpreted programming language that is one of the core technologies on the world wide web. It is similar to but fundamentally different from Java, which is a completely object-oriented programming language owned by Oracle. JavaScript is a cross-platform scripting language for adding interactive elements to web pages, that is in wide use across the internet. By using JavaScript libraries, web programmers can add everything from simple animations to complex server-side applications for internet users. JavaScript is continuously evolving to meet the functionality and security needs of internet users and is capable of being released under a GNU GPL License.

Consider This

The term object-oriented refers to programing that abstracts complex actions and processes so that the end user only deals with basic tasks. To visualize this concept, think of a machine that performs a complex set of tasks by simply pushing a button.

Perl is an interpreted language. Perl was originally developed to perform text manipulation. Over the years, it gained favor with systems administrators and continues to be improved and used in everything from automation to building web applications.

PHP is a language that was initially built to create dynamic web pages. A PHP file is read by a web server such as Apache. Special tags in the file indicate that parts of the code should be interpreted as instructions. The web server pulls all the different parts of the file together and sends it to the web browser. PHP's main advantages are that it is easy to learn and available on almost any system. Because of this, many popular projects are built on PHP. Notable examples include WordPress (for blogging), cacti (for monitoring), and even parts of Facebook.

Ruby is another language that was influenced by Perl and Shell, along with many other languages. It makes complex programming tasks relatively easy, and with the inclusion of the Ruby on Rails framework, is a popular choice for building complex web applications. Ruby is also the language that powers many of the leading automation tools like **Chef** and **Puppet**, which make managing a large number of Linux systems much simpler.

Python is another scripting language that is in general use. Much like Ruby it makes complex tasks easier and has a framework called **Django** that makes building web applications very easy. Python has excellent statistical processing abilities and is a favorite in academia.

A computer programming language is just a tool that makes it easier to tell the computer what you want it to do. A library bundles common tasks into a distinct package that can be used by the developer. **ImageMagick** is one such library that lets programmers manipulate images in code. ImageMagick also ships with some command line tools that enable programmers to process images from a shell and take advantage of the scripting capabilities there.

OpenSSL is a cryptographic library that is used in everything from web servers to the command line. It provides a standard interface for adding cryptography into a Perl script, for example.

At a much lower level is the **C library**. The C library provides a basic set of functions for reading and writing to files and displays, and is used by applications and other languages alike.

3.6 Security

Administrators and computer users are increasingly aware of privacy concerns in both their personal and professional lives. High-profile data breaches have been in the news all too often recently, and the cost of these break-ins can reach into the millions of dollars for the institutions that fall victim to hackers and ransomware attacks. Many times the cause of these breaches is simply human error such as opening a suspicious email or entering passwords into a phony login page.

Cookies are the primary mechanism that websites use to track you. Sometimes this tracking is good, such as to keep track of what is in your shopping cart or to keep you logged in when you return to the site.

As you browse the web, a web server can send back the cookie, which is a small piece of text, along with the web page. Your browser stores this information and sends it back with every request to the same site. Cookies are normally only sent back to the site they originated from, so a cookie from example.com wouldn't be sent to example.org.

However, many sites have embedded scripts that come from third parties, such as a banner advertisement or Google analytics pixel. If both example.com and example.org have a tracking pixel, such as one from an advertiser, then that same cookie will be sent when browsing both sites. The advertiser then knows that you have visited both example.com and example.org.

With a broad enough reach, such as placement on social network sites with "Like" buttons and such, a website can gain an understanding of which websites you frequent and figure out your interests and demographics.

There are various strategies for dealing with this. One is to ignore it. The other is to limit the tracking pixels you accept, either by blocking them entirely or clearing them out periodically.

Browsers typically offer cookie-related settings; users can opt to have the browser tell the site not to track. This voluntary tag is sent in the request, and some sites will honor it. The browser can also be set never to remember third-party cookies and remove regular cookies (such as from the site you are browsing) after being closed.

Tweaking privacy settings can make you more anonymous on the Internet, but it can also cause problems with some sites that depend on third-party cookies. If this happens, you might have to explicitly permit some cookies to be saved.
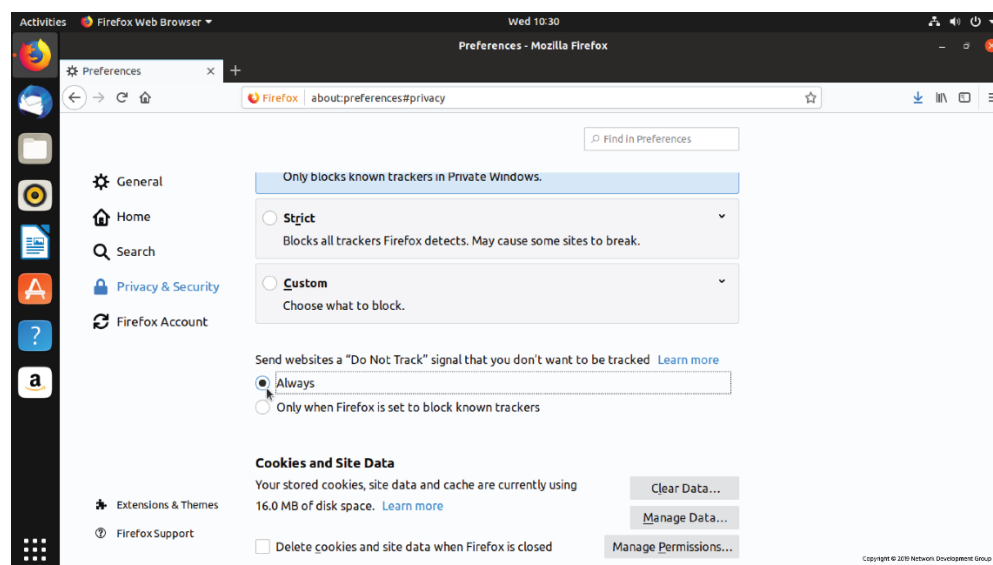
Browsers also offer a private or incognito mode where cookies and tracking pixels are deleted upon exiting the window. This mode can be helpful if you would like to search for something without letting other websites know what you are looking for.

3.6.1 Password Issues

Good password management is essential to security in any computing environment. The Linux systems administrator is often the person responsible for setting and enforcing password policies for users at all levels. The most privileged user on any Linux system is root; this account is the primary administrator and is created when the operating system is installed. Often administrators will disable root access as the first line of defense against intrusion since computer hackers will try to gain root access in order to take control of the system.

There are many levels of access and various means of password management on a Linux system. When users are created, they are given different login permissions depending on what groups they are assigned to. For example, administrators can create and manage users while regular users cannot. Services that run on systems such as databases can also have login permissions with their own passwords and privileges. Additionally, there are specific passwords for accessing systems remotely through SSH, FTP, or other management programs.

Managing all these accounts, and their accompanying passwords is a complicated and necessary part of the systems administrator role. Passwords need to be complex enough not to be easily guessed by hackers, yet easy to remember for users. Increasingly users and administrators are turning to password manager programs to store login credentials in encrypted form. Another trend is two-factor authentication (2FA), a technique where a password is supplemented by a second "factor," often a passcode sent to the user's phone or other devices. Keeping up with current security trends, while ensuring authorized users' ease of access, is an ongoing challenge that must be met.

3.6.2 Protecting Yourself

As you browse the web, you leave a digital footprint. Much of this information goes ignored; some of it is gathered to collect statistics for advertising, and some can be used for malicious purposes.

The easiest thing you can do is to use a good, unique password everywhere you go, especially on your local machine. A good password is at least 10 characters long and contains a mixture of numbers, letters (both upper and lower case) and special symbols. Use a password manager like **KeePassX** to generate passwords, and then you only need to have a login password to your machine and a password to open up your KeePassX file.

Also, limit the information you give to sites to only what is needed. While giving your mother's maiden name and birthdate might help unlock your social network login if you lose your password, the same information can be used to impersonate you to your bank.

After that, make a point of checking for updates periodically. The system can be configured to check for updates on a regular basis. If there are security-related updates, you may be prompted immediately to install them.

Activities   ◎ **Settings** ▼

Trash

‹

✦ **About**

🕓 **Date & T**

👤 **Users**

★ **Default**
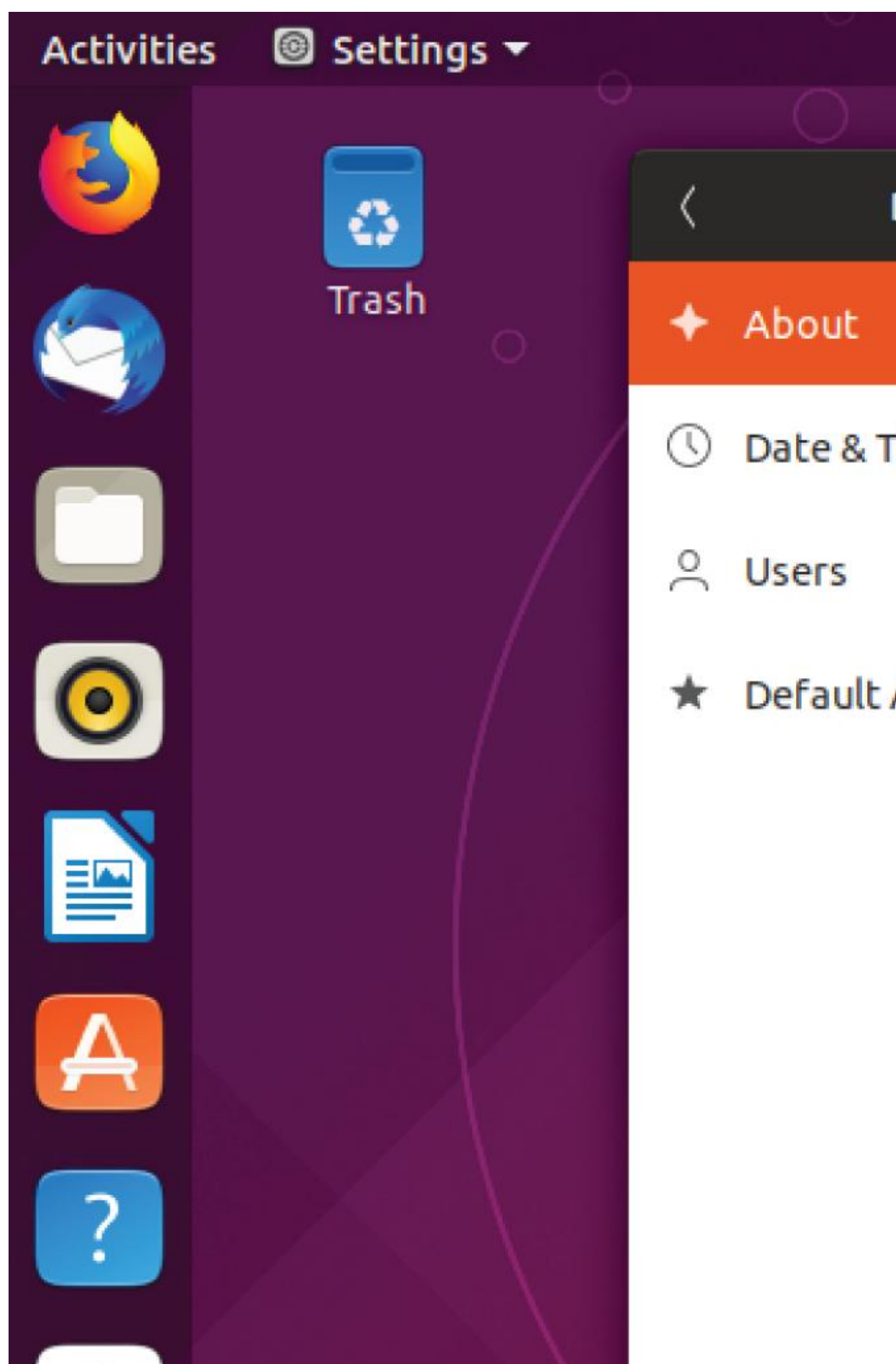
Finally, you should protect your computer from accepting incoming connections.
A firewall is a device that filters network traffic, and Linux has one built-in. If you are using Ubuntu, then the **Gufw** is a graphical interface to **Ubuntu's Uncomplicated Firewall (UFW)**.
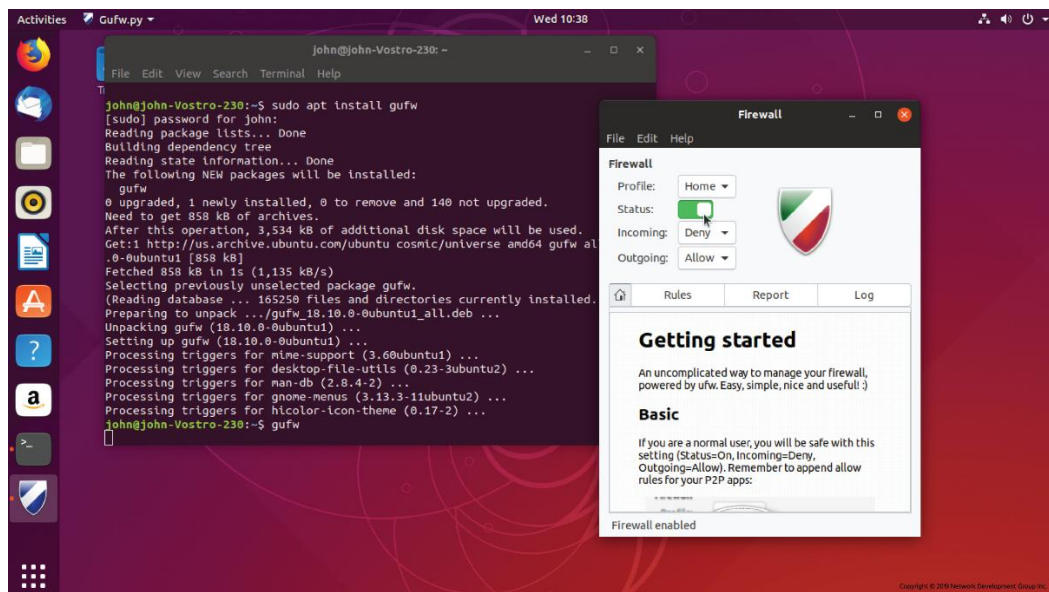
Under the hood, you are using **iptables**, which is the built-in firewall system. Instead of entering complicated iptables commands you use a GUI. While this GUI lets you build an effective policy for a desktop, it barely scratches the surface of what iptables can do.

3.6.3 Privacy Tools

The use of modern privacy tools, both at the server and user level, can help prevent system intrusions and unauthorized access to data.

The good news is that Linux is by default one of the most secure operating systems ever created. Many of the exploits that plague other operating systems simply won't work on Linux due to the underlying architecture. However, there are still many known weaknesses that hackers can take advantage of so the proactive systems administrator is wise to deploy privacy tools that protect their users as well as the systems they use.

Encryption is probably the best-known and most widely-deployed privacy tool in use today. Administrators deploy encryption with authentication keys on almost every system that communicates with the outside world. One well-known example is the **HyperText Transfer Protocol Secure (HTTPS)** standard used on web servers to ensure that data transmitted between users and online resources cannot be intercepted as it travels on the open Internet.

Virtual private networks (VPN) have been in use by companies to connect their remote servers and employees for many years. Now they are gaining popularity amongst ordinary users looking to protect their privacy online. They work by creating an encrypted channel of communication between two systems, so the data transmitted between them is scrambled by an algorithm only the systems know.

The **Tor** project has long been involved in creating privacy tools like it's **Tor Browser** that works by relaying internet requests through a network of servers that prevents websites and others from learning the identity of the person making the request.

These tools are constantly evolving and choosing which ones are appropriate for the users and systems involved is an essential part of the systems administrator's role.

3.7 The Cloud

No doubt you've heard of the cloud. Whether you're using Google Docs for your homework or storing music and photos on iCloud, you probably have at least some of your digital content hosted on a cloud server somewhere.

Cloud computing has revolutionized the way we access technology. As Internet connectivity and speeds have increased, it's become easier to move computing resources to remote locations where content can be accessed, manipulated and shared around the globe. Organizations are increasingly looking at the cloud as essential to their businesses and operations. The migration of an organization's IT applications and processes to cloud services, known as cloud adoption, is rapidly becoming a strategic business decision for many. With cloud adoption rising significantly all over the globe, cloud computing is not the catchphrase that it once was. Cloud computing is seen as one of the major disruptive technologies of the coming decade which will significantly transform businesses, economies, and lives globally.

Physically, a cloud can be described as computing resources from one or many off-site data centers which can be accessed over the internet. The cloud builds on the benefits of a data center and provides computing solutions to organizations who need to store and process data, and it allows them to delegate management of IT infrastructure to a third-party. The data and resources that organizations store in the cloud can include data, servers, storage, application hosting, analytics and a myriad of other services.

A cloud deployment model provides a basis for how cloud infrastructure is built, managed, and accessed. There are four primary cloud deployment models:

- **Public Cloud**: A public cloud is a cloud infrastructure deployed by a provider to offer cloud services to the general public and organizations over the Internet. In the public cloud model, there may be multiple tenants (consumers) who share common cloud resources. More than likely, many of us have accessed public cloud resources at some point through providers such as Amazon, Google, and other popular public cloud providers.

- **Private Cloud**: A private cloud is a cloud infrastructure that is set up for the sole use of a particular organization. When compared to a public cloud, a private cloud offers organizations a greater degree of privacy, and control over the cloud infrastructure, applications, and data. It can be hosted either on servers managed by the company that is using it or through a managed private cloud provider such as Rackspace or IBM.

- **Community Cloud**: A community cloud is a cloud infrastructure that is set up for the sole use by a group of organizations with common goals or requirements. The organizations participating in the community typically share the cost of the community cloud service. This option may be more expensive than the public cloud; however, it may offer a higher level of control and protection against external threats than a public cloud.

- **Hybrid Cloud**: A hybrid cloud is composed of two or more individual clouds, each of which can be a private, community, or public cloud. A hybrid cloud may change over time as component clouds join and leave. The use of such technology enables data and application portability. It also allows companies to leverage outside resources while retaining control of sensitive resources.

3.7.1 Linux in the Cloud

Linux plays a pivotal role in cloud computing. It powers 90% of the public cloud workload, most virtual servers are based on some version of the Linux kernel, and Linux is often used to host the applications behind cloud computing services. So what makes Linux uniquely suited to enabling cloud computing?

Flexibility

Cloud computing provides the capability to provision IT resources quickly and at any time. This agility enables rapid development and experimentation that, in turn, facilitates innovation which is essential for research and development, the discovery of new markets and revenue opportunities, creating new customer segments, and the development of new products.

As a result, cloud computing must compensate for the fact that each organization has a unique, evolving set of resource requirements.

Linux stands out here because it is highly adaptable. For starters, Linux is modular by design, and at the center of an enormous ecosystem of open source applications providing endless configuration options to suit various systems and use cases. On top of that, Linux scales efficiently, allowing it to run anything from a tiny remote sensor to an entire server farm.

Accessibility

In a traditional environment, IT resources are accessed from dedicated devices, such as a desktop or a laptop. In cloud computing, applications and data reside centrally and are accessed from anywhere over a network from any device, such as desktop, mobile, or thin client, and there is a version of Linux for every single one of these devices.

Cost-Effective

Cloud computing is attractive as it has the potential for consumers to reduce their IT costs. In cloud computing, consumers can unilaterally and automatically scale IT resources to meet workload demand, thereby eliminating overhead from underutilized resources. Additionally, the expenses associated with IT configuration, management, floor space, power, and cooling are reduced.

Cloud providers absorb these infrastructure costs but must remain a low-cost alternative. Choosing Linux is one of the most cost-effective solutions providers can deploy. Linux is one of the most power efficient operating systems, and the Linux kernel is completely free, as are many associated applications, utilities, and additional software components.

Enterprise and government organizations can opt to pay for commercially-supported distributions, which are still more cost-effective when compared to licensed competitors. Non-commercial distributions that support cloud computing also are a viable option for many organizations.

Not only can vendors pass these savings onto the customers, offering Linux-based solutions can be cheaper for the client to implement. Setting up Linux on their own systems eliminates expensive user licensing fees potentially associated with competing operating systems.

Manageability

While Linux began as a niche operating system, its widespread presence in the IT industry has made Linux use and administration a necessary skill for IT professionals. It is becoming increasingly easy for cloud vendors and consumers to acquire the necessary talent, or reallocate existing team members.

The nature of Linux, built on the C programming language, also lends itself to automated management tools. A significant portion of Linux servers operating in the cloud are created and managed by automated management programs rather than human operators. This process frees up administrators to monitor computing operations rather than manually configuring and updating systems.

Security

When using a cloud solution, especially a public cloud, an organization may have concerns related to privacy, external threats, and lack of control over the IT resources and data.

Linux can help offset these issues because it is one of the most secure and reliable operating systems available. Linux is open source, meaning its source code is available for anyone to obtain, review, and modify. This also means the code can be inspected for vulnerabilities and compatibility issues, resulting in an extensive community effort to rectify these issues and uphold the robust reputation of Linux.

Virtualization

Virtualization is one of the most significant advancements that has contributed to the enablement cloud of computing.

Linux is a multi-user operating system, which means that many different users can work on the same system simultaneously and for the most part can't do things to harm other users. However, this does have limitations – users can hog disk space or take up too much memory or CPU resources and make the system slow for everyone. Sharing the system in multi-user mode also requires that everyone run as unprivileged users, so letting each user run their own web server, for example, is challenging.

Virtualization is the process where one physical computer, called the host, runs multiple copies of an operating system, each copy called a guest. These guest images can be pre-configured for specific functions to allow rapid deployment, often automatically, when needed. The host system runs software called a hypervisor that switches resources between the various guests just like the Linux kernel does for individual processes. With bare metal hypervisors, the hypervisor runs directly on computer hardware rather than on top of an OS freeing up more resources for guest images.

Virtualization works because servers spend most of their time idling and don't need physical resources such as a monitor and keyboard. With software like **Workstation** and **VirtualBox** from companies like **VMWare** and **Oracle**, you can now take a powerful CPU and by using it to run multiple virtual machines administrators can optimize usage of physical resources and dramatically reduce costs over the previous one-machine, one-OS data center model. The main limitation is usually memory, however, with advances in hypervisor technology and CPUs, it is possible to put more virtual machines on one host than ever.

In a virtualized environment one host can run dozens of guest operating systems, and with support from the CPU itself, the guests don't even know they are running on a virtual machine. Each guest gets its own virtual resources and communicates with the network on its own. It is not even necessary to run the same operating system on all the guests, which further reduces the number of physical servers needed.

Virtualization offers a way for an enterprise to lower power usage and reduce data center space over an equivalent fleet of physical servers. Guests are now just software configurations, so it is easy to spin up a new machine for testing and destroy it when its usefulness has passed.

Since it is possible to run multiple instances of an operating system on one physical machine and connect to it over the network, the location of the machine doesn't matter. Cloud computing takes this approach and allows administrators to have virtual machines

in a remote data center owned by another company, and only pay for the resources used. Cloud computing vendors can take advantage of scales of economy to offer computing resources at far lower prices than operating an on-site data center.

Containers and Bare Metal Deployments

With the rise of containerization technologies like **Docker** and **Kubernetes** application software is now being written that runs in a serverless environment. Essentially, programmers are creating software that does one single function of a system (like database processing or storage) that runs in a container. These containers are organized in pods that run within a node and can talk with each other, and the outside world if needed. Nodes, in turn, are organized and controlled by a master node that provides services to each component within the structure. Building applications in this way decouples each of the components from the others, and from the overhead of running an OS. Since each piece of the puzzle can be automatically destroyed and recreated by the master node they no longer need to be as robust as software that runs on top of an OS. Although these new programming architectures are in many ways bypassing the need for a traditional OS the underlying technology that makes them work is still Linux. So, working in Linux will increasingly be working within a development team that draws on the disciplines of programming, database design, networking, and systems administration to create the systems of the future.

Chapter 4: Open-Source Software and Licensing

4.1 Introduction

Software projects take the form of source code, which is a human-readable set of computer instructions. Since source code is not understood directly by the computer, it must be compiled into machine instructions by a compiler. The compiler is a special program that gathers all of the source code files and generates instructions that can be run on the computer, such as by the Linux kernel.

Historically, commercial software has been sold under a closed source license, meaning that users have the right to use the machine code, also known as the binary or executable, but cannot see the source code. Often the license explicitly states that users may not attempt to reverse engineer the machine code back to source code to figure out what it does.

Consider This

Source code compiled into binary programs is one method of creating programs and running computing instructions. Another is the many types of interpreted languages, such as PERL, Python and even BASH scripting, where the code is not compiled, but fed to an interpreting program, typically a binary executable that understands and implements the instructions contained in the source code or scripts.

The development of Linux closely parallels the rise of open source software. Early on there was shareware, freely available programs where users did not necessarily have access to the source code. There were a lot of good things about this, but it was also problematic because malicious programs could be disguised as innocent-looking games, screensavers, and utilities.

Open source takes a source-centric view of software. The open source philosophy is that users have the right to obtain the software source code, and to expand and modify programs for their own use. This also meant the code could be inspected for backdoors, viruses, and spyware. By creating a community of developers and users, accountability for bugs, security vulnerabilities, and compatibility issues became a shared responsibility. This new, global community of computer enthusiasts was empowered by the growing availability of faster internet services and the world wide web.

There are many different variants of open source, but all agree that users should have access to the source code. Where they differ is in how one can, or must, redistribute changes.

Linux has adopted this philosophy to great success. Since Linux was written in the C programming language, and it mirrored the design and functionality of already established UNIX systems, it naturally became a forum where people could develop and share new ideas. Freed from the constraints of proprietary hardware and software platforms, large numbers of very skilled programmers have been able to contribute to the various distributions, making for software that is often more robust, stable, adaptable, and, frankly, better than the proprietary, closed source offerings which dominated the previous decades.

Large organizations were understandably suspicious about using software built in this new way, but over time they realized their best programmers were working on Linux-based open source projects in their spare time. Soon, Linux servers and open source programs began to outperform the expensive, proprietary systems already in place. When it came time to upgrade outdated hardware the same programmers, engineers, and system administrators who had started working on Linux as a hobby were able to convince their bosses to give Linux a try. The rest is, as they say, history.
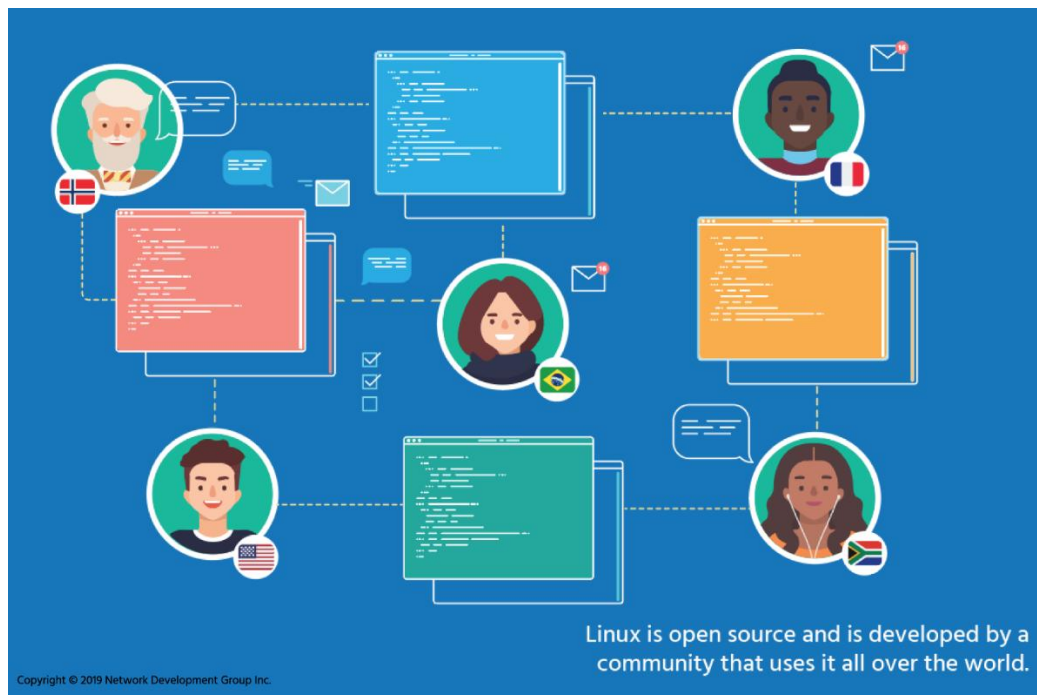
Before the development of Linux, many corporate and scientific applications ran on proprietary UNIX systems. Companies, universities, and governments that run large server farms liked the stability and relative ease of application development these platforms offered.

UNIX was initially created in 1969. By its fourth edition, in 1973, it had been rewritten in the C programming language that is still prominent today. In 1984 the University of California Berkeley released 4.2BSD which introduced TCP/IP, the networking specification that underpins the Internet. By the early 1990's, when Linux development started, different companies developing UNIX operating systems realized their systems needed to be compatible, and they started working on the X/Open specification that is still used today.

Over the years, computer scientists and the organizations that employ them have realized the benefit of systems that provide familiar tools and consistent ways of accomplishing specific tasks. The standardization of application programming interfaces (APIs) allows programs written for one specific UNIX or Linux operating system to be ported (converted) relatively easy to run on another. So, while proprietary UNIX systems are still in use throughout the world in environments where "certified" solutions are preferred, the interoperability of these systems alongside Linux computers is valued by industry, academia, and governments that use them.

The importance of standards organizations cannot be overstated. Groups like the **IEEE (Institute of Electrical and Electronics Engineers)** and **POSIX (Portable Operating System Interface)**, allow professionals from different companies and institutions to collaborate on specifications that make it possible for different operating systems and programs to work together. It doesn't matter if a program is closed or open source, simple or complex, if it is written to these standards others will be able to use and modify it in the future. Every innovation in computing is built on the work of others who came before. Open source software is a collaboration of different people with different needs and backgrounds all working together to make something better than any one of them

could have made individually. Standards are what makes this possible, and the many organizations that create, maintain and promote them are integral to the industry.

Linux is open source and is developed by a community that uses it all over the world.

Copyright © 2019 Network Development Group Inc.

4.2 Open Source Licensing

When talking about buying software, there are three distinct components:

- **Ownership** – Who owns the intellectual property behind the software?
- **Money Transfer** – How does money change hands, if at all?
- **Licensing** – What do you get? What can you do with the software? Can you use it on only one computer? Can you give it to someone else?

In most cases, the ownership of the software remains with the person or company that created it. Users are only granted a license to use the software; this is a matter of copyright law. The money transfer depends on the business model of the creator. It's the licensing that differentiates open source software from closed source software.

Two contrasting examples will get things started.

With Microsoft Windows, the Microsoft Corporation owns the intellectual property. The license itself, the **End User License Agreement (EULA)**, is a custom legal document that you must click through, indicating your acceptance, in order to install the software. Microsoft keeps the source code and distributes only binary copies through authorized channels. For most consumer products you are allowed to install the software on one computer and are not allowed to make copies of the disk other than for a backup. You are not allowed to reverse engineer the software. You pay for one copy of the software, which gets you minor updates but not major upgrades.

Linux is owned by Linus Torvalds. He has placed the code under a license called **GNU General Public License version 2 (GPLv2)**. This license, among other things, says that the source code must be made available to anyone who asks and that anyone is allowed to make changes. One caveat to this is that if someone makes changes and distributes them, they must put the changes under the same license so that others can benefit.

GPLv2 also says that no one is allowed to charge for distributing the source code other than the actual costs of doing so (such as copying it to removable media).

In general, when someone creates something, they also get the right to decide how it is used and distributed. **Free and Open Source Software (FOSS)** refers to software where this right has been given up; anyone is allowed to view the source code and redistribute it. Linus Torvalds has done that with Linux – even though he created Linux he can't forbid someone from using it on their computer because he has given up that right through the GPLv2 license.

Software licensing is a political issue, therefore it should come as no surprise that there are many different opinions. Organizations have come up with their own license that embodies their particular views, so it is easier to choose an existing license than come up with your own. For example, universities like the Massachusetts Institute of Technology (MIT) and University of California have come up with licenses, as have projects like the Apache Foundation. Also, groups like the Free Software Foundation have created their own licenses to further their agenda.

4.2.1 The Free Software Foundation

Two groups can be considered the most influential forces in the world of open source: the Free Software Foundation and the Open Source Initiative.

Only a few years after the development of the GNU project, Richard Stallman founded the **Free Software Foundation (FSF)** in 1985 with the goal of promoting free software. In this context, the word "free" does not refer to the price, but to the freedom to share, study, and modify the underlying source code. According to their website, the FSF believes that users should have "control over the technology we use in our homes, schools, and businesses".

FSF also advocates that software licenses should enforce the openness of modifications. It is their view that if someone modifies free software that they should be required to share any changes they have made when they share it again. This specific philosophy is called copyleft. According to FSF, "copyleft is a general method for making a program (or other work) free (in the sense of freedom, not "zero price"), and requiring all modified and extended versions of the program to be free as well".

The FSF also advocates against software patents and acts as a watchdog for standards organizations, speaking out when a proposed standard might violate the free software principles by including items like Digital Rights Management (DRM) which restrict what can be done with compliant programs.

The FSF have developed their own set of licenses which are free for anyone to use based on the original **GNU General Public License (GPL)**. FSF currently maintains GNU General Public License version 2 (GPLv2) and version 3 (GPLv3), as well as the GNU Lesser General Public Licenses version 2 (LGPLv2) and version 3 (LGPLv3). These licenses are meant to be included in the actual source code to ensure that all future variants and modifications of the original program continue to have the same freedom of use as the original. The GPL license and its variants are powerful legal tools to advance the cause of free software worldwide. What started off in 1983 as one man's desire to share and improve software by letting others change it has ended up changing the world.

Consider This

The changes between GPLv2 and GPLv3 largely focused on using free software on a closed hardware device which has been coined Tivoization. TiVo is a company that builds a television digital video recorder on their own hardware and used Linux as the

base for their software. While TiVo released the source code to their version of Linux as required under GPLv2, the hardware would not run any modified binaries. In the eyes of the FSF, this went against the spirit of the GPLv2, so they added a specific clause to version 3 of the license. Linus Torvalds agrees with TiVo on this matter and has chosen to stay with GPLv2.

4.2.2 The Open Source Initiative

The **Open Source Initiative (OSI)** was founded in 1998 by Bruce Perens and Eric Raymond. They believed that the Free Software Foundation was too politically charged and that less extreme licenses were necessary, particularly around the copyleft aspects of FSF licenses. OSI believes that not only should the source be freely available, but also that no restrictions should be placed on the use of the software, no matter what the intended use. Unlike the FSF, the OSI does not have its own set of licenses. Instead, the OSI has a set of principles and adds licenses to that list if they meet those principles, called open source licenses. Software that conforms to an Open Source license is, therefore, open source software.

One type of Open Source license is the **BSD (Berkeley Software Distribution)** and its derivatives, which are much simpler than GPL. There are currently two actual "BSD" licenses approved by OSI, a 2-Clause and a 3-Clause. These licenses state that you may redistribute the source and binaries as long as you maintain copyright notices and don't imply that the original creator endorses your version. In other words "do what you want with this software, just don't say you wrote it." According to FSF, the original BSD license had a serious flaw in that it required developers to add a clause acknowledging the University of California, Berkeley in every advertisement for software licensed this way. As others copied this simple license, they included acknowledgment for their own institutions which led to over 75 such acknowledgments in some cases.

FSF licenses, such as GPLv2, are also open source licenses. However, many open source licenses such as BSD and MIT do not contain the copyleft provisions and are thus not acceptable to the FSF. These licenses are called permissive free software licenses because they are permissive in how you can redistribute the software. You can take BSD licensed software and include it in a closed software product as long as you give proper attribution.

Rather than dwell over the finer points of Open Source and Free Software, the community has started referring to them collectively as **Free and Open Source Software (FOSS)**. The English word "free" can mean "free as in lunch" (as in no cost) or "free as in speech" (as in no restrictions). This ambiguity led to the inclusion of the word "libre" to refer to the latter definition. Thus, we end up with **Free/Libre/Open Source Software (FLOSS)**.

4.2.3 Creative Commons

FOSS licenses are mostly related to software. People have placed works such as drawings and plans under FOSS licenses, but this was not the intent.

When software has been placed in the public domain, the author has relinquished all rights, including the copyright on the work. In some countries, this is the default when the work is done by a government agency. In some countries, copyrighted work becomes public domain after the author has died and a lengthy waiting period has elapsed.

The **Creative Commons (CC)** organization has created the Creative Commons Licenses which try to address the intentions behind FOSS licenses for non-software entities. CC licenses can also be used to restrict commercial use if that is the desire of the copyright holder. The CC licenses are made up of the following set of conditions the creator can apply to their work:

- **Attribution (BY)** – All CC licenses require that the creator must be given credit, without implying that the creator endorses the use.

- **ShareAlike (SA)** – This allows others to copy, distribute, perform, and modify the work, provided they do so under the same terms.

- **NonCommercial (NC)** – This allows others to distribute, display, perform, and modify the work for any purpose other than commercially.

- **NoDerivatives (ND)** – This allows others to distribute, display, and perform only original copies of the work. They must obtain the creator's permission to modify it.

These conditions are then combined to create the six main licenses offered by Creative Commons:

- **Attribution (CC BY)** – Much like the BSD license, you can use CC BY content for any use but must credit the copyright holder.

- **Attribution ShareAlike (CC BY-SA)** – A copyleft version of the Attribution license. Derived works must be shared under the same license, much like in the Free Software ideals.

- **Attribution NoDerivs (CC BY-ND)** – You may redistribute the content under the same conditions as CC-BY but may not change it.

- **Attribution-NonCommercial (CC BY-NC)** – Just like CC BY, but you may not use it for commercial purposes.

- **Attribution-NonCommercial-ShareAlike (CC BY-NC-SA)** – Builds on the CC BY-NC license but requires that your changes be shared under the same license.

- **Attribution-NonCommercial-NoDerivs (CC BY-NC-ND)** – You are sharing the content to be used for non-commercial purposes, but people may not change the content.

- **No Rights Reserved (CC0)** – This is the Creative Commons version of public domain.

4.3 Open Source Business Models

If all this software is free, how can anyone make money off of it?

First, you must understand there isn't anything in the GPL that prohibits selling software. In fact, the right to sell software is part of the GPL license. Again, recall that the word free refers to freedom, not price. Companies that add value to these free programs are encouraged to make as much money as they can, and put those profits back into developing more and better software.

One of the simplest ways to make money is to sell support or warranty around the software. Companies like Canonical, the developer of Ubuntu, and Redhat have grown into huge enterprises by creating Linux distributions and tools that enable commercial users to manage their enterprises and offer products and services to their customers.

Many other open source projects have also expanded into substantial businesses. In the 1990s, Gerald Combs was working at an Internet service provider and started writing his own network analysis tool because similar tools at the time were costly. Over 600 people have now contributed to the project, called Wireshark. It is now often considered better than commercial offerings and led to a company being formed to sell products and support. Like many others, this company was purchased by a larger enterprise that supports its continued development.

Companies like Tivo have packaged hardware or add extra closed source software to sell alongside the free software. Appliances and embedded systems that use Linux are a multi-billion dollar business and encompass everything from home DVRs to security cameras and wearable fitness devices. Many consumer firewalls and entertainment devices follow this model.

Today, both large and small employers have individuals and sometimes whole groups devoted to working on open source projects. Technology companies compete for the opportunity to influence projects that will shape the future of their industries. Other companies dedicate resources towards projects they need for internal use. As more business is done on cloud resources, the opportunity for open source programmers continues to expand.

Chapter 5: Command Line Skills

5.4.3 Path Variable

One of the most important Bash shell variables to understand is the `PATH` variable. It contains a list that defines which directories the shell looks in to find commands. If a valid command is entered and the shell returns a "command not found" error, it is because the Bash shell was unable to locate a command by that name in any of the directories included in the path. The following command displays the path of the current shell:

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games
```

Each directory in the list is separated by a colon `:` character. Based on the preceding output, the path contains the following directories. The shell will check the directories in the order they are listed:

```
/home/sysadmin/bin

/usr/local/sbin

/usr/local/bin

/usr/sbin

/usr/bin

/sbin

/bin

/usr/games
```

5.5 Command Types

One way to learn more about a command is to look at where it comes from.
The `type` command can be used to determine information about command type.

```
type command
```

example:

```
sysadmin@localhost:/bin$ type nc
nc is /bin/nc
```

5.5.1 Internal Commands

Also called built-in commands, internal commands are built into the shell itself. A good example is the `cd` (change directory) command as it is part of the Bash shell. When a user types the `cd` command, the Bash shell is already executing and knows how to interpret it, requiring no additional programs to be started.

The `type` command identifies the `cd` command as an internal command:

```
sysadmin@localhost:~$ type cd

cd is a shell builtin
```

5.5.2 External Commands

External commands are binary executables stored in directories that are searched by the shell. If a user types the `ls` command, then the shell searches through the directories that are listed in the PATH variable to try to find a file named `ls` that it can execute.

`which` command searches for the location of a command by searching the PATH variable.

```
sysadmin@localhost:~$ which ls

/bin/ls

sysadmin@localhost:~$ which cal

/usr/bin/cal
```

5.5.3 Aliases

An alias can be used to map longer commands to shorter key sequences. When the shell sees an alias being executed, it substitutes the longer sequence before proceeding to interpret commands.

For example, the command `ls -l` is commonly aliased to `l` or `ll`. Because these smaller commands are easier to type, it becomes faster to run the `ls -l` command line.

To determine what aliases are set on the current shell use the `alias` command:

```
sysadmin@localhost:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

The aliases from the previous examples were created by initialization files. These files are designed to make the process of creating aliases automatic.

New aliases can be created using the following format, where *name* is the name to be given the alias and *command* is the command to be executed when the alias is run.

```
alias name=command
```

```
sysadmin@localhost:~$ alias mycal="cal 2019"
sysadmin@localhost:~$ mycal
```

5.5.4 Functions

Functions can also be built using existing commands to either create new commands, or to override commands built-in to the shell or commands stored in files. Aliases and functions are normally loaded from the initialization files when the shell first starts.

Functions are more advanced than aliases and typically are used in Bash shell scripts. Typically, functions are used to execute multiple commands. To create a function, the following syntax is used:

```
function_name ()
{
    commands
}
```

Functions are useful as they allow for a set of commands to be executed one at a time instead of typing each command repeatedly. In the example below, a function called `my_report` is created to execute the `ls`, `date`, and `echo` commands.

```
sysadmin@localhost:~$ my_report () {
> ls Documents
```

```
> date

> echo "Document directory report"

> }
```

5.6 Quoting

Quotation marks are used throughout Linux administration and most computer programming languages to let the system know that the information contained within the quotation marks should either be ignored or treated in a way that is very different than it would normally be treated. There are three types of quotes that have special significance to the Bash shell: double quotes ", single quotes ', and back quotes `

5.6.1 Double Quotes

Within double quotes an asterisk is just an asterisk, a question mark is just a question mark, and so on, which is useful when you want to display something on the screen that is normally a special character to the shell. In the `echo` command below, the Bash shell doesn't convert the glob pattern into filenames that match the pattern:

```
sysadmin@localhost:~$ echo "The glob characters are *, ? and [ ]"

The glob characters are *, ? and [ ]
```

et. The following demonstration shows that the value of the `PATH` variable is still displayed:

```
sysadmin@localhost:~$ echo "The path is $PATH"

The path is /usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/l
ocal/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

5.6.2 Single Quotes

Single quotes prevent the shell from doing any interpreting of special characters, including globs, variables, command substitution and other metacharacters that have not been discussed yet.

For example, to make the $ character simply mean a $, rather than it acting as an indicator to the shell to look for the value of a variable, execute the second command displayed below:

```
sysadmin@localhost:~$ echo The car costs $100

The car costs 00
```

5.6.3 Backslash Character

There is also an alternative technique to essentially single quote a single character. Consider the following message:

```
The service costs $1 and the path is $PATH
```

In this case, use a backslash \ character in front of the dollar sign $ character to prevent the shell from interpreting it. The command below demonstrates using the \ character:

```
sysadmin@localhost:~$ echo The service costs \$1 and the path is $PAT
H
The service costs $1 and the path is /usr/bin/custom:/home/sysadmin/b
in:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
```

5.6.4 Backquotes

Backquotes, or backticks, are used to specify a command within a command, a process called command substitution. This allows for powerful and sophisticated use of commands.

```
sysadmin@localhost:~$ echo Today is date
Today is date
```

In the previous command, the word `date` is treated as regular text, and the shell passes `date` to the `echo` command. To execute the `date` command and have the output of that command sent to the `echo` command, put the `date` command in between two backquote characters:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Mon Nov 4 03:40:04 UTC 2018
```

5.7 Control Statements

Control statements allow you to use multiple commands at once or run additional commands, depending on the success of a previous command. Typically these control statements are used within scripts, but they can also be used on the command line as well.

5.7.1 Semicolon

```
command1; command2; command3
```

The semicolon ; character can be used to run multiple commands, one after the other. Each command runs independently and consecutively; regardless of the result of the first

command, the second command runs once the first has completed, then the third and so on.

5.7.2 Double Ampersand

```
command1 && command2
```

The double ampersand `&&` acts as a logical "and"; if the first command is successful, then the second command will also run. If the first command fails, then the second command will not run.

In the following example, the first command succeeds because the `/etc/ppp` directory exists and is accessible while the second command fails because there is no `/junk` directory:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d   ip-up.d
sysadmin@localhost:~$ ls /etc/junk
ls: cannot access /etc/junk: No such file or directory
```

5.7.3 Double Pipe

```
command1 || command2
```

The double pipe `||` is a logical "or". Depending on the result of the first command, the second command will either run or be skipped.

Chapter6: Getting Help

6.2.4 Man Pages Categorized by Sections

there are thousands of man pages on a typical Linux distribution. To organize all of these man pages, they are categorized by sections.

By default, there are nine sections of man pages:

1. General Commands
2. System Calls
3. Library Calls
4. Special Files
5. File Formats and Conventions
6. Games
7. Miscellaneous
8. System Administration Commands

9. Kernel Routines

The man command searches each of these sections in order until it finds the first match. For example, if you execute the command man cal, the first section (General Commands) is searched for a man page called cal. If not found, then the second section is searched. If no man page is found after searching all sections, an error message is returned:

```
sysadmin@localhost:~$ man zed
No manual entry for zed
```

, determine in which section the man page is located. To search for man pages by name, use the -f option to the man command. It displays man pages that match, or partially match, a specific name and provide the section number and a brief description of each man page:

```
sysadmin@localhost:~$ man -f passwd
passwd (5)            - the password file
passwd (1)            - change user password
passwd (1ssl)         - compute password hashes
```

```
sysadmin@localhost:~$ man -f sudo
sudo (8)              - execute a command as another user
```

 each man page has a short description associated with it. The -k option to the man command searches both the names and descriptions of the man pages for a keyword.

For example, to find a man page that displays how to copy directories, search for the keyword copy:

```
sysadmin@localhost:~$ man -k copy
cp (1)                - copy files and directories
cpgr (8)              - copy with locking the given file to the passwo
rd or gr...
cpio (1)              - copy files to and from archives
cppw (8)              - copy with locking the given file to the passwo
rd or gr...
dd (1)                - convert and copy a file
debconf-copydb (1)    - copy a debconf database
install (1)           - copy files and set attributes
scp (1)               - secure copy (remote file copy program)
ssh-copy-id (1)       - use locally available keys to authorize logins
on a re...
```

6.3.2 Find Any File or Directory

The `whereis` command is specifically designed to find commands and man pages. While this is useful, it is often necessary to find a file or directory, not just files that are commands or man pages.

To find any file or directory, use the `locate` command. This command searches a database of all files and directories that were on the system when the database was created.

6.4 Info Documentation

The `info` command also provides documentation on operating system commands and features. The goal is slightly different from man pages: to provide a documentation resource that gives a logical organizational structure, making reading documentation easier.

Chapter 7: navigating The File System

7.2 Directory Structure

On a Windows system, the top level of the directory structure is called My Computer. Physical devices, such as hard drives, USB drives, network drives, show up under My Computer and are each assigned a drive letter, such as C: or D:.

A visual representation of a Windows directory structure:
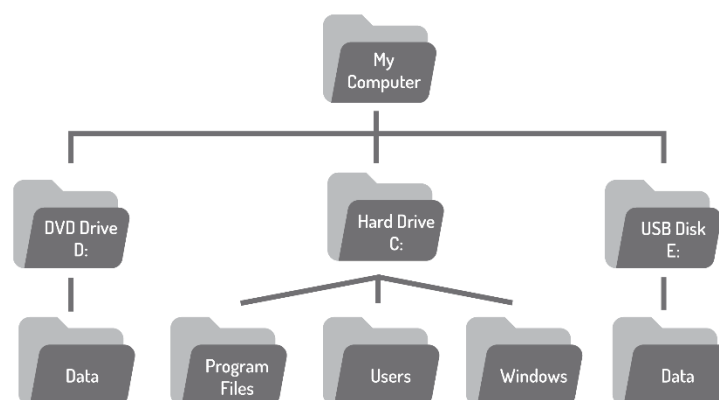
ike Windows, the Linux directory structure, typically called a filesystem, also has a top level. However instead of My Computer, it is called the root directory, and it is symbolized

by the slash / character. Additionally, there are no drives in Linux; each physical device is accessible under a directory, as opposed to a drive letter.

The following image shows a visual representation of a typical Linux filesystem:
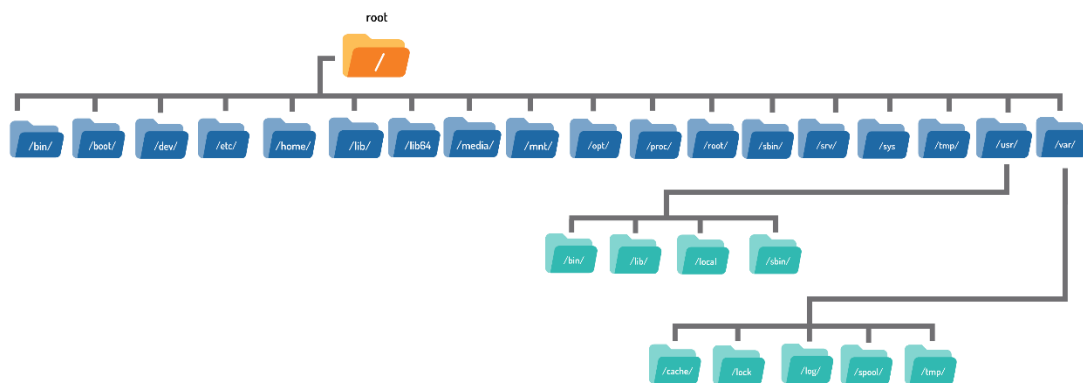
7.2.1 Home Directory

To begin with, on most Linux distributions there is a directory called `home` under the root / directory.

Under this `/home` directory there is a directory for each user on the system. The directory name is the same as the name of the user, so a user named `sysadmin` would have a home directory called `/home/sysadmin`.
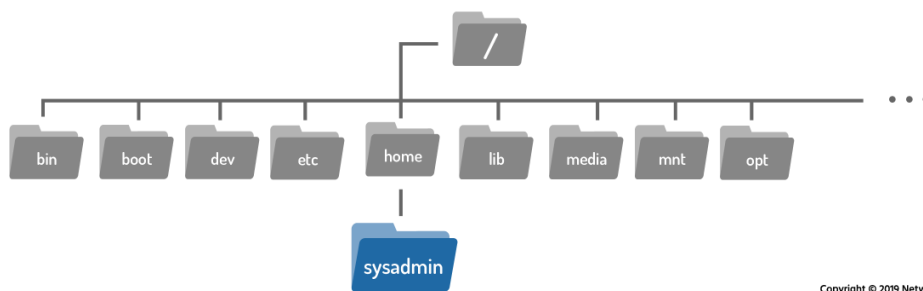
the home directory is one of the few directories where the user has full control to create and delete additional files and directories. On most Linux distributions, the only users who can access the files in a home directory are the owner and the administrator on the system. Most other directories in a Linux filesystem are protected with *file permissions*.

he home directory has a special symbol used to represent it; the tilde ~ character. So if the sysadmin user is logged in, the tilde ~ character can be used in place of the `/home/sysadmin` directory.

It is also possible to refer to another user's home directory by using the tilde ~ character followed by the name of the user account. For example, `~bob` would be the equivalent of `/home/bob`.
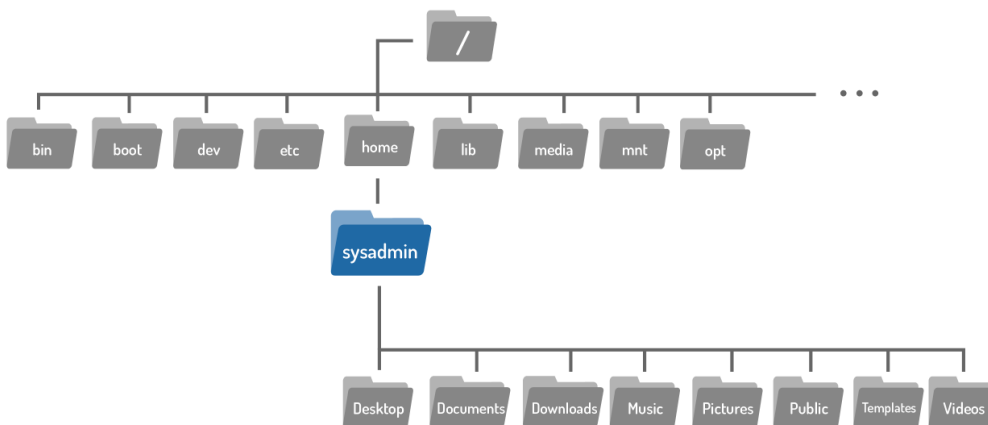
7.3.1 Absolute Paths

Absolute paths allow the user to specify the exact location of a directory

It always starts at the root directory, and therefore it always begins with the `/` character. The path `/home/sysadmin` is an absolute path; it tells the system to begin at the root `/` directory, move into the `home` directory, and then into the `sysadmin` directory.

7.3.2 Relative Paths

Relative paths start from the current directory. A relative path gives directions to a file relative to the current location in the filesystem. They do not start with the / character. Instead, they start with the name of a directory. More specifically, relative paths start with the name of a directory contained within the current directory.

Take another look at the first `cd` command example. The argument is an example of the simplest relative path: the name of a directory within the current working directory.

```
sysadmin@localhost:~$ cd Documents
sysadmin@localhost:~/Documents$
```

7.4.1 Listing Hidden Files

A hidden file is any file (or directory) that begins with a dot `.` character.

To display all files, including hidden files, use the `-a` option to the `ls` command:

```
sysadmin@localhost:~$ ls -a
.               .bashrc     .selected_editor   Downloads   Public
..              .cache      Desktop            Music       Templates
.bash_logout    .profile    Documents          Pictures    Videos
```

Why are files hidden in the first place? Most of the hidden files are customization files, designed to customize how Linux, your shell or programs work. For example, the `.bashrc` file in the home directory customizes features of the shell, such as creating or modifying variables and aliases.

These customization files are not ones that you work with on a regular basis, and having them displayed makes it more difficult to find other files.

7.4.2 Long Display Listing

File Type

```
-rw-r--r-- 1 root    root   15322 Dec 10 21:33 alternatives.log
drwxr-xr-x 1 root    root    4096 Jul 19 06:52 apt
```

The first character of each line indicates the type of file. The file types are:

Symbol	File Type	Description
d	directory	A file used to store other files.
-	regular file	Includes readable files, images files, binary files, and compressed files.

Symbol	File Type	Description
l	symbolic link	Points to another file.
s	socket	Allows for communication between processes.
p	pipe	Allows for communication between processes.
b	block file	Used to communicate with hardware.
c	character file	Used to communicate with hardware.

The first file `alternatives.log` is a regular file (-), while the second file `apt` is a directory (d).

Permissions

```
d rwxr-xr-x 2 root   root   4096 Jul 19 06:51 journal
```

The next nine characters demonstrate the permissions of the file. Permissions indicate how certain users can access a file.

Hard Link Count

```
-rw-r----- 1 syslog adm     371 Dec 15 16:38 auth.log
drwxr-xr-x 2 root   root   4096 Jul 19 06:51 journal
```

This number indicates how many hard links point to this file.

User Owner

```
-rw-r----- 1 syslog adm     197 Dec 15 16:38 cron.log
```

Every file is owned by a user account. This is important because the owner has the rights to set permissions on a file.

Group Owner

```
-rw-rw-r-- 1 root   utmp   292584 Dec 15 16:38 lastlog
```

Indicates which group owns this file. This is important because any member of this group has a set of permissions on the file.

File Size

```
-rw-r----- 1 syslog adm    14185  Dec 15 16:38 syslog
```

Displays the size of the file in bytes.

For directories, this value does not describe the total size of the directory, but rather how many bytes are reserved to keep track of the filenames in the directory. In other words, ignore this field for directories.

Timestamp

```
 -rw-rw---- 1 root   utmp      0 May 26  2018  btmp
```

Indicates the time that the file's contents were last modified. For directories, this timestamp indicates the last time a file was added or deleted from the directory.

File Name

```
-rw-r--r-- 1 root   root  35330 May 26  2018 bootstrap.log
```

The final field contains the name of the file or directory.

In the case of symbolic links, the link name is displayed along with an arrow and the pathname of the original file.

```
lrwxrwxrwx. 1 root root 22 Nov 6 2018 /etc/grub.conf -> ../boot/grub
/grub.conf
```

test lab:

The ? character can be used to match exactly 1 character in a file name. Execute the following command to display all of the files in the /etc directory that are exactly four characters long:

```
ls -d /etc/????
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -d /etc/????
/etc/bind  /etc/init  /etc/motd  /etc/perl  /etc/skel
/etc/dpkg  /etc/ldap  /etc/mtab  /etc/sgml  /etc/udev
sysadmin@localhost:~$
```

test lab:

By using square brackets [ ] you can specify a single character to match from a set of characters. Execute the following command to display all of the files in the /etc directory that begin with the letters a, b, c or d:

```
ls -d /etc/[abcd]*
```

Chapter 8 - Managing Files and Directories

8.2 Globbing

Glob characters are often referred to as wild cards. These are symbol characters that have special meaning to the shell.

Unlike commands that the shell runs, or options and arguments that the shell passes to commands, glob characters are interpreted by the shell itself before it attempts to run any command. As a result, glob characters can be used with *any* command.

8.2.1 Asterisk * Character

The asterisk `*` character is used to represent zero or more of any character in a filename. For example, to display all of the files in the `/etc` directory that begin with the letter `t`:

```
sysadmin@localhost:~$ echo /etc/t*
/etc/terminfo /etc/timezone /etc/tmpfiles.d
```

8.2.2 Question Mark ? Character

The question mark `?` character represents any single character

```
sysadmin@localhost:~$ echo /etc/t???????
/etc/terminfo /etc/timezone
```

8.2.3 Bracket [ ] Characters

The bracket `[]` characters are used to match a single character by representing a range of characters that are possible match characters. For example, the `/etc/[gu]*` pattern matches any file that begins with either a `g` or `u` character and contains zero or more additional characters:

```
sysadmin@localhost:~$ echo /etc/[gu]*
/etc/gai.conf /etc/groff /etc/group /etc/group- /etc/gshadow /etc/gsh
adow- /etc/
gss /etc/ucf.conf /etc/udev /etc/ufw /etc/update-motd.d /etc/updatedb
.conf
```

8.2.4 Exclamation Point ! Character

The exclamation point `!` character is used in conjunction with the square brackets to negate a range. For example, the pattern `/etc/[!DP]*` matches any file that does not begin with a `D` or P.

```
sysadmin@localhost:~$ echo /etc/[!a-t]*

/etc/X11 /etc/ucf.conf /etc/udev /etc/ufw /etc/update-motd.d /etc/upd
atedb.conf

/etc/vim /etc/vtrgb /etc/wgetrc /etc/xdg
```

8.2.5 Listing With Globs

The `ls` command is normally used to list files in a directory; as a result, using the `echo` command may seem to have been a strange choice. However, there is something about the `ls` command that causes problems when listing files using glob patterns.

If the `ls` command is given a directory name, the command displays the contents of the directory (the names of the files in the directory), not just the directory name. The filenames in the previous example are the names of the files in the `/etc/apparmor` directory.

Why is this a problem when using globs? Consider the following output:

```
sysadmin@localhost:~$ ls /etc/ap*
/etc/apparmor:

init  parser.conf  subdomain.conf


/etc/apparmor.d:

abstractions  disable         local           tunables     usr.sbin.na
med

cache         force-complain  sbin.dhclient  usr.bin.man  usr.sbin.rs
yslogd


/etc/apt:

apt.conf.d  preferences.d  sources.list  sources.list.d  trusted.gpg.
d
```

```
sysadmin@localhost:~$ echo  /etc/ap*

/etc/apm /etc/apparmor /etc/apparmor.d /etc/apt
```

There is a simple solution to this problem: always use the `-d` option with globs, which tells the `ls` command to display the name of directories, instead of their contents:

```
sysadmin@localhost:~$ls -d /etc/x*
/etc/xdg
```

8.3 Copying Files

The `cp` command is used to copy files. It requires a source and a destination. The structure of the command is as follows:

```
cp source destination
```

sysadmin@localhost:~$ cp /etc/hosts ~/hosts.copy

sysadmin@localhost:~$ ls

Desktop   **Downloads** **Pictures** **Templates** hosts.copy

Documents **Music**   **Public**   **Videos**

sysadmin@localhost:~$ cat hosts.copy

127.0.0.1   localhost

::1   localhost ip6-localhost ip6-
loopback
fe00::0 ip6-
localnet

ff00::0 ip6-
mcastprefix

ff02::1 ip6-
allnodes

ff02::2 ip6-
allrouters

192.168.1.2   localhost

8.3.2 Avoid Overwriting Data

The `cp` command can be destructive to existing data if the destination file already exists. In the case where the destination file exists, the `cp` command overwrites the existing file's contents with the contents of the source file.

sysadmin@localhost:~$ cp /etc/hostname ~/hosts.copy

sysadmin@localhost:~$ cat hosts.copy

localhost
Two options can be used to safeguard against accidental overwrites. With the `-i` *interactive* option, the `cp` command prompts the user before overwriting a file. The following example demonstrates this option, first restoring the content of the original file:

sysadmin@localhost:~$ cp -i /etc/hosts ~/hosts.copy
cp: overwrite '/home/sysadmin/hosts.copy'? n

```
sysadmin@localhost:~$ cat hosts.copy

localhost
```

The `-i` option requires you to answer `y` or `n` for every copy that could end up overwriting an existing file's contents. This can be tedious when a bunch of overwrites occur, such as the example demonstrated below:

```
sysadmin@localhost:~$ cp -i /etc/skel/.* ~
cp: -r not specified; omitting directory '/etc/skel/.'
cp: -r not specified; omitting directory '/etc/skel/..'
cp: overwrite `/home/sysadmin/.bash_logout'? n
cp: overwrite `/home/sysadmin/.bashrc'? n
cp: overwrite `/home/sysadmin/.profile'? n
cp: overwrite `/home/sysadmin/.selected_editor'? n
```

As you can see from the example above, the `cp` command tried to overwrite four existing files, forcing the user to answer four prompts. If this situation happened for 100 files, it could become very annoying, very quickly.

To answer `n` to each prompt automatically, use the `-n` option. It stands for no clobber, or no overwrite.

```
sysadmin@localhost:~$ cp -n /etc/skel/.* ~
cp: -r not specified; omitting directory '/etc/skel/.'
cp: -r not specified; omitting directory '/etc/skel/..'
```

8.3.3 Copying Directories

By default, the `cp` command will not copy directories; any attempt to do so results in an error message:

```
sysadmin@localhost:~$ cp -n /etc/skel/.* ~
cp: -r not specified; omitting directory '/etc/skel/.'
cp: -r not specified; omitting directory '/etc/skel/..'
```

However, the recursive `-r` option allows the `cp` command to copy both files and directories.

```
cp -r source_directory destination_directory
```

8.4 Moving Files

To move a file, use the `mv` command. The syntax for the `mv` command is much like the `cp` command:

```
mv source destination
```

```
sysadmin@localhost:~$ ls
Desktop     Downloads   Pictures   Templates   example.txt   hosts.copy
Documents   Music       Public     Videos      hosts
sysadmin@localhost:~$ mv hosts Videos
sysadmin@localhost:~$ ls
Desktop     Downloads   Pictures   Templates   example.txt
Documents   Music       Public     Videos      hosts.copy
sysadmin@localhost:~$ ls Videos
hosts
```

When a file is moved, the file is removed from the original location and placed in a new location. Moving files can be somewhat tricky in Linux because users need specific permissions to remove files from a directory. Without the right permissions, a `Permission denied` error message is returned:

```
sysadmin@localhost:~$ mv /etc/hosts .
mv: cannot move `/etc/hosts' to `./hosts': Permission denied
```

8.4.1 Renaming Files

The `mv` command is not just used to move a file, but also to rename a file. If the destination for the `mv` command is a directory, the file is moved to the directory specified. The name of the file only changes if a destination file name is also specified.

```
sysadmin@localhost:~$ mv example.txt Videos/newexample.txt
sysadmin@localhost:~$ ls Videos
hosts   newexample.txt
```

8.4.2 Additional Move Options

Like the `cp` command, the `mv` command provides the following options:

Option	Meaning
`-i`	Interactive: Ask if a file is to be overwritten.
`-n`	No Clobber: Do not overwrite a destination file's contents.
`-v`	Verbose: Show the resulting move.

8.5 Creating Files

To create an empty file, use the `touch` command as demonstrated below:

```
sysadmin@localhost:~$ touch sample
sysadmin@localhost:~$ ls -l sample
-rw-rw-r-- 1 sysadmin sysadmin 0 Nov  9 16:48 sample
```

Notice the size of the new file is `0` bytes. As previously mentioned, the `touch` command doesn't place any data within the new file.

8.6 Removing Files

To delete a file, use the `rm` command:

```
sysadmin@localhost:~$ ls
Desktop     Downloads  Pictures   Templates   hosts.copy
Documents   Music      Public     Videos      sample
sysadmin@localhost:~$ rm sample
sysadmin@localhost:~$ rm hosts.copy
sysadmin@localhost:~$ ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates   Vi
deos
```

Note that the files were deleted with no questions asked. This could cause problems when deleting multiple files by using glob characters. Because these files are deleted without question, a user could end up deleting files that were not intended to be deleted.

Warning

The files are permanently deleted. There is no command to undelete a file and no trash can from which to recover deleted files.

As a precaution, users should use the `-i` option when deleting multiple files:

```
sysadmin@localhost:~$ touch sample.txt example.txt test.txt
sysadmin@localhost:~$ ls
Desktop     Downloads  Pictures   Templates   example.txt   test.txt
Documents   Music      Public     Videos      sample.txt
sysadmin@localhost:~$ rm -i *.txt
rm: remove regular empty file `example.txt'? y
rm: remove regular empty file `sample.txt'? n
rm: remove regular empty file `test.txt'? y
sysadmin@localhost:~$ ls
Desktop     Downloads  Pictures   Templates   sample.txt
Documents   Music      Public     Videos
```

8.6.1 Removing Directories

You can delete directories using the `rm` command. However, the default behavior (no options) of the `rm` command is to not delete directories:

```
sysadmin@localhost:~$ rm Videos

rm: cannot remove `Videos': Is a directory
```

To delete a directory with the `rm` command, use the `-r` recursive option:

```
sysadmin@localhost:~$ ls

Desktop    Downloads  Pictures  Templates  sample.txt

Documents  Music      Public    Videos

sysadmin@localhost:~$ rm -r Videos

sysadmin@localhost:~$ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  sa
mple.txt
```

When a user deletes a directory, all of the files and subdirectories are deleted without any interactive question. It is best to use the `-i` option with the `rm` command.

You can also delete a directory with the `rmdir` command, but only if the directory is empty.

```
sysadmin@localhost:~$ rmdir Documents

rmdir: failed to remove 'Documents': Directory not empty
```

8.7 Creating Directories

To create a directory, use the `mkdir` command:

```
sysadmin@localhost:~$ ls

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  sa
mple.txt

sysadmin@localhost:~$ mkdir test

sysadmin@localhost:~$ ls

Desktop    Downloads  Pictures  Templates   test

Documents  Music      Public    sample.txt
```

Module 9: Archiving and Compression

9.1 Introduction

In this chapter, we discuss how to manage archive files at the command line. File archiving is used when one or more files need to be transmitted or stored as efficiently as possible. There are two fundamental aspects which this chapter explores:

- Archiving: Combines multiple files into one, which eliminates the overhead in individual files and makes the files easier to transmit.
- Compression: Makes the files smaller by removing redundant information.

Consider This

Files can be compressed individually, or multiple files can be combined into a single archive and then subsequently compressed. The latter is still referred to as archiving.

9.2 Compressing Files

Compression reduces the amount of data needed to store or transmit a file while storing it in such a way that the file can be restored. A file with human-readable text might have frequently used words replaced by something smaller, or an image with a solid background might represent patches of that color by a code. The compressed version of the file is not typically viewed or utilized, instead, it is decompressed before use.

The compression algorithm is a procedure the computer uses to encode the original file, and as a result, make it smaller. Computer scientists research these algorithms and come up with better ones that can work faster or make the input file smaller.

When talking about compression, there are two types:

- Lossless: No information is removed from the file. Compressing a file and decompressing it leaves something identical to the original.
- Lossy: Information might be removed from the file. It is compressed in such a way that uncompressing a file will result in a file that is slightly different from the original. For instance, an image with two subtly different shades of green might be made smaller by treating those two shades as the same. Often, the eye can't pick out the difference anyway.

Generally, human eyes and ears don't notice slight imperfections in pictures and audio, especially as they are displayed on a monitor or played over speakers. Lossy compression often benefits media because it results in smaller file sizes and people can't tell the difference between the original and the version with the changed data. For things that must remain intact, such as documents, logs, and software, you need lossless compression.

Linux provides several tools to compress files; the most common is `gzip`. Here we show a file before and after compression:

```
sysadmin@localhost:~$ cd Documents
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 66540 Dec 20  2017 longfile.txt
sysadmin@localhost:~/Documents$ gzip longfile.txt
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 341 Dec 20  2017 longfile.txt.gz
```

The `gzip` command will provide this information, by using the `-l` option, as shown here:

```
sysadmin@localhost:~/Documents$ gzip -l longfile.txt.gz
        compressed        uncompressed  ratio uncompressed_name
            341                66540  99.5% longfile.txt
```

The compression ratio is given as `99.5%`, an impressive reduction helped by the repetitive information in the original file. Additionally, when the file is decompressed, it will be called `longfile.txt` again.

Compressed files can be restored to their original form using either the `gunzip` command or the `gzip -d` command. This process is called decompression. After `gunzip` does its work, the `longfile.txt` file is restored to its original size and file name.

```
sysadmin@localhost:~/Documents$ gunzip longfile.txt.gz
```

There are other commands that operate virtually identically to `gzip` and `gunzip`. These include `bzip2` and `bunzip2`, as well as `xz` and `unxz`.

The `gzip` command uses the **Lempel-Ziv** data compression algorithm, while the `bzip` utilities use a different compression algorithm called **Burrows-Wheeler** block sorting, which can compress files smaller than `gzip` at the expense of more CPU time. These files can be recognized because they have a `.bz` or `.bz2` extension instead of a `.gz` extension.

The `xz` and `unxz` tools are functionally similar to `gzip` and `gunzip` in that they use the **Lempel-Ziv-Markov (LZMA)** chain algorithm, which can result in lower

decompression CPU times that are on par with `gzip` while providing the better compression ratios typically associated with the `bzip2` tools. Files compressed with the `xz` command use the `.xz` extension.

9.3 Archiving Files

If you had several files to send to someone, you could choose to compress each one individually. You would have a smaller amount of data in total than if you sent uncompressed files, however, you would still have to deal with many files at one time.

Archiving is the solution to this problem. The traditional UNIX utility to archive files is called `tar`, which is a short form of TApe aRchive. It was used to stream many files to a tape for backups or file transfer. The `tar` command takes in several files and creates a single output file that can be split up again into the original files on the other end of the transmission.

The `tar` command has three modes that are helpful to become familiar with:

- Create: Make a new archive out of a series of files.
- Extract: Pull one or more files out of an archive.
- List: Show the contents of the archive without extracting.

9.3.1 Create Mode

```
tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
```

Creating an archive with the `tar` command requires two named options:

Option	Function
`-c`	Create an archive.
`-f ARCHIVE`	Use archive file. The argument `ARCHIVE` will be the name of the resulting archive file.

The following example shows a tar file, also called a tarball, being created from multiple files. The first argument creates an archive called `alpha_files.tar`. The wildcard option `*` is used to include all files that begin with `alpha` in the archive:

```
sysadmin@localhost:~/Documents$ tar -cf alpha_files.tar alpha*
sysadmin@localhost:~/Documents$ ls -l alpha_files.tar
-rw-rw-r-- 1 sysadmin sysadmin 10240 Oct 31 17:07 alpha_files.tar
```

Tarballs can be compressed for easier transport, either by using `gzip` on the archive or by having `tar` do it with the `-z` option.

Option	Function
-z	Compress (or decompress) an archive using the `gzip` command.

The next example shows the same command as the prior example, but with the addition of the -z option.

```
sysadmin@localhost:~/Documents$ tar -czf alpha_files.tar.gz alpha*
sysadmin@localhost:~/Documents$ ls -l alpha_files.tar.gz
-rw-rw-r-- 1 sysadmin sysadmin 417 Oct 31 17:15 alpha_files.tar.gz
```

Or to compress an existing tar

```
sysadmin@localhost:~/Documents$ gzip alpha_files.tar
sysadmin@localhost:~/Documents$ ls -
lh alpha_files.tar.gz
-rw-rw-r-- 1 sysadmin sysadmin 432 Jan 15 09:09 alpha_files.tar.gz
```

While file extensions don't affect the way a file is treated, the convention is to use `.tar` for tarballs, and `.tar.gz` or `.tgz` for compressed tarballs.

The `bzip2` compression can be used instead of `gzip` by substituting the -j option for the -z option and using `.tar.bz2`, `.tbz`, or `.tbz2` as the file extension.

Option	Function
-j	Compress (or decompress) an archive using the `bzip2` command.

For example, to archive and compress the `School` directory:

```
sysadmin@localhost:~/Documents$ tar -cjf folders.tbz School
```

9.3.2 List Mode

```
tar -t [-f ARCHIVE] [OPTIONS]
```

Given a `tar` archive, compressed or not, you can see what's in it by using the -t option. The next example uses three options:

Option	Function
-t	List the files in an archive.

Option	Function
-j	Decompress with an bzip2 command.
-f ARCHIVE	Operate on the given archive.

sysadmin@localhost ~/Documents$ tar -tf alpha_files.tar.gz

alpha-first.txt

alpha-second.txt

alpha-third.txt

alpha.txt

To list the contents of the `folders.tbz` archive:

```
sysadmin@localhost:~/Documents$ tar -tjf folders.tbz
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
School/Art/linux.txt
School/Math/
School/Math/numbers.txt
```

In the example, the directory `School/` is prefixed to the files. The tar command will recurse into subdirectories automatically when compressing and will store the path info inside the archive.

Consider This

To show that this file is still nothing special, we will list the contents of the file in two steps using a pipeline, the | character.

```
sysadmin@localhost:~/Documents$ bunzip2 -c folders.tbz | tar -t
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
School/Art/linux.txt
School/Math/
School/Math/numbers.txt
```

The left side of the pipeline is `bunzip2 -c folders.tbz`, which decompresses the file, but the `-c` option sends the output to the screen. The output is redirected to `tar -t`. If

you don't specify a file with `-f` then tar will read from the standard input, which in this case is the uncompressed file.

9.3.3 Extract Mode

```
tar -x [-f ARCHIVE] [OPTIONS]
```

Creating archives is often used to make multiple files easier to move. Before extracting the files, relocate them to the `Downloads` directory:

```
sysadmin@localhost:~/Documents$ cd ~
sysadmin@localhost:~$ cp Documents/folders.tbz Downloads/folders.tbz
sysadmin@localhost:~$ cd Downloads
```

Finally, you can extract the archive with the `-x` option once it's copied into a different directory. The following example uses a similar pattern as before, specifying the operation, the compression, and a file name to operate on.

Option	Function
`-x`	Extract files from an archive.
`-j`	Decompress with the `bzip2` command.
`-f ARCHIVE`	Operate on the given archive.

```
sysadmin@localhost:~/Downloads$ tar -xjf folders.tbz
sysadmin@localhost:~/Downloads$ ls -l
total 8
drwx------ 5 sysadmin sysadmin 4096 Dec 20  2017 School
-rw-rw-r-- 1 sysadmin sysadmin  413 Oct 31 18:37 folders.tbz
```

It is important to keep the `-f` flag at the end, as `tar` assumes whatever follows this option is a file name. In the next example, the `-f` and `-v` flags were transposed, leading to `tar` interpreting the command as an operation on a file called `v`, which does not exist.

```
sysadmin@localhost:~/Downloads$ tar -xjfv folders.tbz
tar (child): v: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

If you only want some files out of the archive, add their names to the end of the command, but by default, they must match the name in the archive exactly, or use a pattern.

The following example shows the same archive as before, but extracting only the `School/Art/linux.txt` file. The output of the command (as verbose mode was requested with the `-v` flag) shows only the one file has been extracted:

```
sysadmin@localhost:~/Downloads$ tar -xjvf folders.tbz School/Art/linu
x.txt

School/Art/linux.txt
```

9.4 ZIP Files

The de facto archiving utility in Microsoft is the ZIP file. ZIP is not as prevalent in Linux but is well supported by the `zip` and `unzip` commands. Albeit, with `tar` and `gzip`/`gunzip` the same commands and options can be used interchangeably to do the creation and extraction, but this is not the case with `zip`. The same option has different meanings for the two different commands.

The default mode of `zip` is to add files to an archive and compress it.

```
zip [OPTIONS] [zipfile [file…]]
```

The first argument `zipfile` is the name of the archive to be created, after that, a list of files to be added. The following example shows a compressed archive called `alpha_files.zip` being created:

```
sysadmin@localhost:~/Documents$ zip alpha_files.zip alpha*

  adding: alpha-first.txt (deflated 32%)

  adding: alpha-second.txt (deflated 36%)

  adding: alpha-third.txt (deflated 48%)

  adding: alpha.txt (deflated 53%)

  adding: alpha_files.tar.gz (stored 0%)
```

It should be noted that `tar` requires the `-f` option to indicate a filename is being passed, while `zip` and `unzip` require a filename and therefore don't need you to inform the command a filename is being passed.

The `zip` command will not recurse into subdirectories by default, which is different behavior than the `tar` command. That is, merely adding `School` will only add the empty directory and not the files under it. If you want `tar` like behavior, you must use the `-r` option to indicate recursion is to be used:

```
sysadmin@localhost:~/Documents$ zip -r School.zip School

updating: School/ (stored 0%)

updating: School/Engineering/ (stored 0%)

updating: School/Engineering/hello.sh (deflated 88%)

updating: School/Art/ (stored 0%)
```

```
updating: School/Art/linux.txt (deflated 49%)

updating: School/Math/ (stored 0%)

updating: School/Math/numbers.txt (stored 0%)

  adding: School/Art/red.txt (deflated 33%)

  adding: School/Art/hidden.txt (deflated 1%)

  adding: School/Art/animals.txt (deflated 2%)
```

The -l list option of the unzip command lists files in .zip archives:

```
sysadmin@localhost:~/Documents$ unzip -l School.zip
Archive:  School.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2017-12-20 16:46   School/
        0  2018-10-31 17:47   School/Engineering/
      647  2018-10-31 17:47   School/Engineering/hello.sh
        0  2018-10-31 19:31   School/Art/
       83  2018-10-31 17:45   School/Art/linux.txt
        0  2018-10-31 17:46   School/Math/
       10  2018-10-31 17:46   School/Math/numbers.txt
       51  2018-10-31 19:31   School/Art/red.txt
       67  2018-10-31 19:30   School/Art/hidden.txt
       42  2018-10-31 19:31   School/Art/animals.txt
---------                     -------
      900                     10 files
        0  2018-10-31 17:46   School/Math/
       10  2018-10-31 17:46   School/Math/numbers.txt
---------                     -------
      740                     7 files
```

Extracting the files is just like creating the archive, as the default operation of
the unzip command is to extract. It gives several options if unzipping files will overwrite
existing ones:

```
sysadmin@localhost:~/Documents$ unzip School.zip
Archive:  School.zip
replace School/Engineering/hello.sh? [y]es, [n]o, [A]ll, [N]one, [r]e
name: n
replace School/Art/linux.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Math/numbers.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename
: n
```

```
replace School/Art/red.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
replace School/Art/hidden.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename:
n
replace School/Art/animals.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename:
n
```

This can be avoided by copying the zip file into a new directory:

```
sysadmin@localhost:~/Documents$ mkdir tmp
sysadmin@localhost:~/Documents$ cp School.zip tmp/School.zip
sysadmin@localhost:~/Documents$ cd tmp
sysadmin@localhost:~/Documents/tmp$ unzip School.zip
Archive:  School.zip
   creating: School/
   creating: School/Engineering/
  inflating: School/Engineering/hello.sh
   creating: School/Art/
  inflating: School/Art/linux.txt
   creating: School/Math/
 extracting: School/Math/numbers.txt
  inflating: School/Art/red.txt
  inflating: School/Art/hidden.txt
  inflating: School/Art/animals.txt
```

###################33

Chapter 10 - Working with Text

10.1.1 Viewing Files in the Terminal

The `cat` command, short for concatenate, is a simple but useful command whose functions include creating and displaying text files, as well as combining copies of text files.

10.1.2 Viewing Files Using a Pager

For larger files, use a *pager* command to view the contents. Pager commands display one page of data at a time, allowing you to move forward and backward in the file by using movement keys.

- The `less` command provides a very advanced paging capability. It is usually the default pager used by commands like the `man` command.
- The `more` command has been around since the early days of UNIX. While it has fewer features than the `less` command, however, the `less` command isn't included with all Linux distributions. The `more` command is always available.

10.1.2.1 Pager Movement Commands

To view a file with the `less` command, pass the file name as an argument:

```
sysadmin@localhost:~/Documents$ less words
```

Key	Movement
Spacebar	Window forward
B	Window backward
Enter	Line forward
Q	Exit
H	Help

10.1.2.2 Pager Searching Commands

There are two ways to search in the `less` command: searching forward or backward from your current position.

To start a search to look forward from your current position, use the slash **/** key. Then, type the text or pattern to match and press the **Enter** key.

```
Abdul

Abdul's

Abe

/frog
```

If more than one match can be found by a search, then use the **n** key to move the *next* match and use the **Shift+N** key combination to go to a *previous* match.

10.1.3 Head and Tail

The `head` and `tail` commands are used to display only the first few or last few lines of a file

Passing a number as an option will cause both the `head` and `tail` commands to output the specified number of lines, instead of the standard ten. For example to display the last five lines of the `/etc/sysctl.conf` file use the `-5` option:

```
sysadmin@localhost:~$ tail -5 /etc/sysctl.conf
```

Negative Value Option

Traditionally in UNIX, the number of lines to output would be specified as an option with either command, so `-3` meant to show three lines. For the `tail` command, either `-3` or `-n -3` still means show three lines.

However, the GNU version of the `head` command recognizes `-n -3` as show all but the last three lines, and yet the `head` command still recognizes the option `-3` as show the first three lines.

Positive Value Option

The GNU version of the `tail` command allows for a variation of how to specify the number of lines to be printed. If the `-n` option is used with a number prefixed by the plus sign, then the `tail` command recognizes this to mean to display the contents starting at the specified line and continuing all the way to the end.

For example, the following displays the contents of the `/etc/passwd` from line `25` to the end of the file:

```
sysadmin@localhost:~$ nl /etc/passwd | tail -n +25
    25  sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
    26  operator:x:1000:37::/root:/bin/sh
    27  sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:
/bin/bash
```

Consider This

Live file changes can be viewed by using the `-f` option to the `tail` command—useful when you want to see changes to a file as they are happening.

A good example of this would be when viewing log files as a system administrator. Log files can be used to troubleshoot problems and administrators often view them "interactively" with the `tail` command while performing commands in a separate window.

For example, if you were to log in as the `root` user, you could troubleshoot issues with the email server by viewing live changes to the `/var/log/mail.log` log file.

10.2 Command Line Pipes

The pipe `|` character can be used to send the output of one command to another. Typically, when a command has output or generates an error, the output is displayed to the screen; however, this does not have to be the case. Instead of being printed to the screen, the output of one command becomes input for the next command. This tool can be powerful, especially when looking for specific data; piping is often used to refine the results of an initial command.

. To more easily view the beginning of the output, pipe it to the `head` command. The following example displays only the first ten lines:

```
sysadmin@localhost:~$ ls /etc | head
```

The full output of the `ls` command is passed to the `head` command by the shell instead of being printed to the screen. The `head` command takes this output from the `ls` command as input data, and the output of `head` is then printed to the screen.

Multiple pipes can be used consecutively to link multiple commands together. If three commands are piped together, the output of the first command is passed to the second command. Then, the output of the second command is passed to the third command. The output of the third command would then be printed to the screen.

It is important to carefully choose the order in which commands are piped, as each command only sees input from the previous command. The examples below illustrate this using the `nl` command, which adds line numbers to the output. In the first example, the `nl` command is used to number the lines of the output of the preceding `ls` command:

```
sysadmin@localhost:~$ ls /etc/ssh | nl
     1  moduli
     2  ssh_config
     3  ssh_host_ecdsa_key
     4  ssh_host_ecdsa_key.pub
```

```
 5  ssh_host_ed25519_key

 6  ssh_host_ed25519_key.pub

 7  ssh_host_rsa_key

 8  ssh_host_rsa_key.pub

 9  ssh_import_id

10  sshd_config
```

10.3 Input/Output Redirection

Input/Output (I/O) redirection allows for command line information to be passed to different streams. Before discussing redirection, it is important to understand the standard streams.

- ## STDIN

  Standard input, or STDIN, is information entered normally by the user via the keyboard. When a command prompts the shell for data, the shell provides the user with the ability to type commands that, in turn, are sent to the command as STDIN.

- ## STDOUT

  Standard output, or STDOUT, is the normal output of commands. When a command functions correctly (without errors) the output it produces is called STDOUT. By default, STDOUT is displayed in the terminal window where the command is executing. STDOUT is also known as stream or channel #1.

- ## STDERR

  Standard error, or STDERR, is error messages generated by commands. By default, STDERR is displayed in the terminal window where the command is executing. STDERR is also known as stream or channel #2.

I/O redirection allows the user to redirect STDIN so that data comes from a file and STDOUT/STDERR so that output goes to a file. Redirection is achieved by using the arrow < > characters.

10.3.1 STDOUT

STDOUT can be directed to files. To begin, observe the output of the following `echo` command which displays to the screen:

```
sysadmin@localhost:~$ echo "Line 1"

Line 1
```

Using the > character, the output can be redirected to a file instead:

```
sysadmin@localhost:~$ echo "Line 1" > example.txt
```

The file contains the output of the `echo` command, which can be viewed with the `cat` command:

```
sysadmin@localhost:~$ cat example.txt
Line 1
```

It is important to realize that the single arrow overwrites any contents of an existing file:

```
sysadmin@localhost:~$ cat example.txt
Line 1
sysadmin@localhost:~$ echo "New line 1" > example.txt
sysadmin@localhost:~$ cat example.txt
New line 1
```

It is also possible to preserve the contents of an existing file by appending to it. Use two arrow >> characters to append to a file instead of overwriting it:

```
sysadmin@localhost:~$ cat example.txt
New line 1
sysadmin@localhost:~$ echo "Another line" >> example.txt
sysadmin@localhost:~$ cat example.txt
New line 1
Another line
```

10.3.2 STDERR

STDERR can be redirected similarly to STDOUT. When using the arrow character to redirect, stream #1 (STDOUT) is assumed unless another stream is specified. Thus, stream #2 must be specified when redirecting STDERR by placing the number 2 preceding the arrow > character.

To demonstrate redirecting STDERR, first observe the following command which produces an error because the specified directory does not exist:

```
sysadmin@localhost:~$ ls /fake
ls: cannot access /fake: No such file or directory
```

Note that there is nothing in the example above that implies that the output is STDERR. The output is clearly an error message, but how could you tell that it is being sent to STDERR? One easy way to determine this is to redirect STDOUT:

```
sysadmin@localhost:~$ ls /fake > output.txt
ls: cannot access /fake: No such file or directory
```

The STDERR output of a command can be sent to a file:

```
sysadmin@localhost:~$ ls /fake 2> error.txt
```

In the example, the `2>` indicates that all error messages should be sent to the
file `error.txt`, which can be confirmed using the `cat` command:

```
sysadmin@localhost:~$ cat error.txt

ls: cannot access /fake: No such file or directory
```

```
sysadmin@localhost:~$ cat example.txt
ls: cannot access '/check': No such file or directory
sysadmin@localhost:~$ ls /Aisha  2>> example.txt
sysadmin@localhost:~$ cat example.txt
ls: cannot access '/check': No such file or directory
ls: cannot access '/Aisha': No such file or directory
sysadmin@localhost:~$
```

10.3.3 Redirecting Multiple Streams

It is possible to direct both the STDOUT and STDERR of a command at the same time.
The following command produces both STDOUT and STDERR because one of the
specified directories exists and the other does not:

```
sysadmin@localhost:~$ ls /fake /etc/ppp

ls: cannot access /fake: No such file or directory

/etc/ppp:

ip-down.d  ip-up.d
```

If only the STDOUT is sent to a file, STDERR is still printed to the screen:

```
sysadmin@localhost:~$ ls /fake /etc/ppp > example.txt

ls: cannot access /fake: No such file or directory

sysadmin@localhost:~$ cat example.txt

/etc/ppp:

ip-down.d

ip-up.d
```

If only the STDERR is sent to a file, STDOUT is still printed to the screen:

```
sysadmin@localhost:~$ ls /fake /etc/ppp 2> error.txt

/etc/ppp:

ip-down.d

ip-up.d

sysadmin@localhost:~$ cat error.txt
```

```
ls: cannot access /fake: No such file or directory
```

Both STDOUT and STDERR can be sent to a file by using the ampersand `&` character in front of the arrow `>` character. The `&>` character set means both `1>` and `2>`:

```
sysadmin@localhost:~$ ls /fake /etc/ppp &> all.txt
sysadmin@localhost:~$ cat all.txt
ls: cannot access /fake: No such file or directory
/etc/ppp:
ip-down.d
ip-up.d
```

If you don't want STDERR and STDOUT to both go to the same file, they can be redirected to different files by using both `>` and `2>`. For example, to direct STDOUT to `example.txt` and STDERR to `error.txt` execute the following:

```
sysadmin@localhost:~$ ls /fake /etc/ppp > example.txt 2> error.txt
sysadmin@localhost:~$ cat error.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$ cat example.txt
/etc/ppp:
ip-down.d
ip-up.d
```

10.3.4 STDIN

The concept of redirecting STDIN is a difficult one because it is more difficult to understand why you would want to redirect STDIN. With STDOUT and STDERR, their purpose is straightforward; sometimes it is helpful to store the output into a file for future use.

Most Linux users end up redirecting STDOUT routinely, STDERR on occasion, and STDIN very rarely.

There are very few commands that require you to redirect STDIN because with most commands if you want to read data from a file into a command, you can specify the filename as an argument to the command.

For some commands, if you don't specify a filename as an argument, they revert to using STDIN to get data. For example, consider the following `cat` command:

```
sysadmin@localhost:~$ cat
hello
hello
how are you?
how are you?
goodbye
```

```
goodbye
```

Note

If you do attempt the `cat` command without arguments, kill the process and return to the prompt by using **Ctrl+C**.

The first command in the example below redirects the output of the `cat` command to a newly created file called `new.txt`. This action is followed up by providing the `cat` command with the `new.txt` file as an argument to display the redirected text in STDOUT.

```
sysadmin@localhost:~$ cat > new.txt
Hello
How are you?
Goodbye
sysadmin@localhost:~$ cat new.txt
Hello
How are you?
Goodbye
```

consider a new command called `tr`. This command takes a set of characters and translates them into another set of characters.

For example, to capitalize a line of text use the `tr` command as follows:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z'
watch how this works
WATCH HOW THIS WORKS
```

The `tr` command took the STDIN from the keyboard and converted all lower-case letters before sending STDOUT to the screen.

It would seem that a better use of the `tr` command would be to perform translation on a file, not keyboard input. However, the `tr` command does not support file name arguments:

```
sysadmin@localhost:~$ cat example.txt
/etc/ppp:
ip-down.d
ip-up.d
sysadmin@localhost:~$ tr 'a-z' 'A-Z' example.txt
tr: extra operand `example.txt'
Try `tr --help' for more information
```

It is possible, however, to tell the shell to get STDIN from a file instead of from the keyboard by using the < character:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z' < example.txt

/ETC/PPP:

IP-DOWN.D

IP-UP.D
```

Most commands do accept file names as arguments, so this use case is relatively rare. However, for those that do not, this method could be used to have the shell read from the file instead of relying on the command to have this ability.

One last note to save the resulting output, redirect it into another file:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z' < example.txt > newexample.txt

sysadmin@localhost:~$ cat newexample.txt

/ETC/PPP:

IP-DOWN.D

IP-UP.D
```

```
sysadmin@localhost:~$ cat example2.txt
hello
myname is Aisha Ahmad
sysadmin@localhost:~$ tr 'a-z' 'A-Z' < example2.txt > newfile.txt
sysadmin@localhost:~$ cat newfile.txt
HELLO
MYNAME IS AISHA AHMAD
sysadmin@localhost:~$
```

10.4 Sorting Files or Input

The `sort` command can be used to rearrange the lines of files or input in either dictionary or numeric order. The following example creates a small file, using the `head` command to grab the first 5 lines of the `/etc/passwd` file and send the output to a file called `mypasswd`.

```
sysadmin@localhost:~$ head -5 /etc/passwd > mypasswd

sysadmin@localhost:~$ cat mypasswd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin

sys:x:3:3:sys:/dev:/usr/sbin/nologin

sync:x:4:65534:sync:/bin:/bin/sync
```

Now we will `sort` the `mypasswd` file:

```
sysadmin@localhost:~$ sort mypasswd
```

```
bin:x:2:2:bin:/bin:/usr/sbin/nologin

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

root:x:0:0:root:/root:/bin/bash

sync:x:4:65534:sync:/bin:/bin/sync

sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

Upon close examination of the output in the preceding example, the `sort` command has arranged the lines of the file in alphabetical order.

10.4.1 Fields and Sort Options

The `sort` command can rearrange the output based on the contents of one or more fields. Fields are determined by a field delimiter contained on each line. In computing, a delimiter is a character that separates a string of text or data; it defaults to whitespace, like spaces or tabs.

The following command can be used to sort the third field of the `mypasswd` file numerically. Three options are used to achieve this sort:

Option	Function
`-t:`	The `-t` option specifies the field delimiter. If the file or input is separated by a delimiter other than whitespace, for example a comma or colon, the `-t` option will allow for another field separator to be specified as an argument. The `mypasswd` file used in the previous example uses a colon `:` character as a delimiter to separate the fields, so the following example uses the `-t:` option.
`-k3`	The `-k` option specifies the field number. To specify which field to sort by, use the `-k` option with an argument to indicate the field number, starting with `1` for the first field. The following example uses the `-k3` option to sort by the third field.
`-n`	This option specifies the sort type. The third field in the `mypasswd` file contains numbers, so the `-n` option is used to perform a numeric sort.

```
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sysadmin@localhost:~$ sort -t: -k1 -n newfile.txt
bin:x:2:2:bin:/bin:/usr/sbin/nologin
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sysadmin@localhost:~$ sort -t: -k3 -n newfile.txt
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:~$
```

Another commonly used option to the `sort` command is the `-r` option, which is used to perform a *reverse* sort

```
sysadmin@localhost:~$ cd ~/Documents

sysadmin@localhost:~/Documents$ cat os.csv

1970,Unix,Richie

1987,Minix,Tanenbaum

1970,Unix,Thompson

1991,Linux,Torvalds
```

To sort first by the operating system (field #2) and then year (field #1) and then by last name (field #3), use the following command:

```
sysadmin@localhost:~/Documents$ sort -t, -k2 -k1n -k3 os.csv

1991,Linux,Torvalds

1987,Minix,Tanenbaum

1970,Unix,Richie

1970,Unix,Thompson
```

The following table breaks down the options used in the previous example:

Option	Function
-t,	Specifies the comma character as the field delimiter
-k2	Sort by field #2

Option	Function
-k1n	Numerically sort by field #1
-k3	Sort by field #3

```
sysadmin@localhost:~/Documents$ sort -t, -k1n -k2 -k3 os.csv
1970,Unix,Richie
1970,Unix,Thompson
1987,Minix,Tanenbaum
1991,Linux,Torvalds
sysadmin@localhost:~/Documents$
```

10.5 Viewing File Statistics

The `wc` command provides the number of lines, words and bytes (1 byte = 1 character in a text file) for a file, and a total line count if more than one file is specified. By default, the `wc` command allows for up to three statistics to be printed for each file provided, as well as the total of these statistics if more than one filename is provided:

```
sysadmin@localhost:~$ wc /etc/passwd /etc/passwd-
  35   56 1710 /etc/passwd
  34   55 1665 /etc/passwd-
  69  111 3375 total
```

The output of the previous example has four columns:

1. Number of lines
2. Number of words
3. Number of bytes
4. File name

It is also possible to view only specific statistics, by using the `-l` option to show just the number of lines, the `-w` option to show just the number of words, the `-c` option to show just the number of bytes, or any combination of these options.

The `wc` command can be useful for counting the number of lines output by some other command through a pipe. For example, if you wanted to know the total number of files in the `/etc` directory, pipe the output of `ls` to `wc` and count only the number of lines:

```
sysadmin@localhost:~$ ls /etc/ | wc -l
142
```

10.6 Filter File Sections

The cut command can extract columns of text from a file or standard input. It's primarily used for working with delimited database files. Again, delimited files are files that contain columns separated by a delimiter. These files are very common on Linux systems.

By default, the cut command expects its input to be separated by the tab character, but the -d option can specify alternative delimiters such as the colon or comma.

The -f option can specify which fields to display, either as a hyphenated range or a comma-separated list.

In the following example, the first, fifth, sixth and seventh fields from the mypasswd database file are displayed:

```
sysadmin@localhost:~$ cut -d: -f1,5-7 mypasswd

root:root:/root:/bin/bash

daemon:daemon:/usr/sbin:/usr/sbin/nologin

bin:bin:/bin:/usr/sbin/nologin

sys:sys:/dev:/usr/sbin/nologin

sync:sync:/bin:/bin/sync
```


he cut command is also able to extract columns of text based upon character position with the -c option—useful when working with fixed-width database files or command outputs.

For example, the fields of the ls -l command are always in the same character positions. The following will display just the file type (character 1), permissions (characters 2-10), a space (character 11), and filename (characters 50+):

```
sysadmin@localhost:~$ ls -l | cut -c1-11,50-

total 44

drwxr-xr-x Desktop

drwxr-xr-x Documents

drwxr-xr-x Downloads
```

```
drwxr-xr-x Music

drwxr-xr-x Pictures

drwxr-xr-x Public

drwxr-xr-x Templates

drwxr-xr-x Videos

-rw-rw-r-- all.txt

-rw-rw-r-- example.txt

-rw-rw-r-- mypasswd

-rw-rw-r-- new.txt
```

10.7 Filter File Contents

The `grep` command can be used to filter lines in a file or the output of another command that matches a specified pattern. That pattern can be as simple as the exact text that you want to match or it can be much more advanced through the use of regular expressions.

For example, to find all the users who can log in to the system with the BASH shell, the `grep` command can be used to filter the lines from the `/etc/passwd` file for the lines containing the pattern `bash`:

```
sysadmin@localhost:~$ grep bash /etc/passwd

root:x:0:0:root:/root:/bin/bash

sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

```
sysadmin@localhost:~/Documents$ ls -l | grep root
-rw-r--r-- 1 root      root       971578 Feb  8  2021 words
sysadmin@localhost:~/Documents$
```

The `-c` option provides a count of how many lines match:

```
sysadmin@localhost:~$ grep -c bash /etc/passwd

2
```

The `-n` option to the `grep` command will display original line numbers. To display all lines and their line numbers in the `/etc/passwd` file which contain the pattern `bash`:

```
sysadmin@localhost:~$ grep -n bash /etc/passwd

1:root:x:0:0:root:/root:/bin/bash

27:sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

The `-v` option inverts the match, outputting all lines that do not contain the pattern. To display all lines not containing `nologin` in the `/etc/passwd` file:

```
sysadmin@localhost:~$ grep -v nologin /etc/passwd

root:x:0:0:root:/root:/bin/bash

sync:x:4:65534:sync:/bin:/bin/sync

operator:x:1000:37::/root:/bin/sh

sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bas
h
```

The `-i` option ignores the case (capitalization) distinctions. The following searches for the pattern `the` in `newhome.txt`, allowing each character to be uppercase or lowercase:

```
sysadmin@localhost:~$ cd Documents

sysadmin@localhost:~/Documents$ grep -i the newhome.txt

There are three bathrooms.

**Beware** of the ghost in the bedroom.

The kitchen is open for entertaining.

**Caution** the spirits don't like guests.
```

The `-w` option only returns lines which contain matches that form whole words. To be a word, the character string must be preceded and followed by a non-word character. Word characters include letters, digits, and the underscore character.

The following examples search for the `are` pattern in the `newhome.txt` file. The first command searches with no options, while the second command includes the `-w` option. Compare the outputs:

```
sysadmin@localhost:~/Documents$ grep are newhome.txt

There are three bathrooms.

**Beware** of the ghost in the bedroom.

sysadmin@localhost:~/Documents$ grep -w are newhome.txt

There are three bathrooms.
```

10.8 Basic Regular Expressions

Regular expressions, also referred to as regex, are a collection of normal and special characters that are used to find simple or complex patterns, respectively, in files. These characters are characters that are used to perform a particular matching function in a search.

Normal characters are alphanumeric characters which match themselves. For example, an `a` would match an `a`. Special characters have special meanings when used within patterns by commands like the `grep` command. They behave in a more complex manner and do not match themselves.

There are both Basic Regular Expressions (available to a wide variety of Linux commands) and Extended Regular Expressions (available to more advanced Linux commands). Basic Regular Expressions include the following:

Character	Matches
.	Any single character
[ ]	A list or range of characters to match one character If the first character within the brackets is the caret ^, it means any character not in the list
*	The previous character repeated zero or more times
^	If the first character in the pattern, the pattern must be at the beginning of the line to match, otherwise just a literal ^ character
$	If the last character in the pattern, the pattern must be at the end of the line to match, otherwise just a literal $ character

The `grep` command is just one of the many commands that support regular expressions. Some other commands include the `more` and `less` commands.

10.8.1 The Period . Character

One of the most useful expressions is the period `.` character. It matches any character except for the new line character. Consider the unfiltered contents of the `~/Documents/red.txt` file:

```
sysadmin@localhost:~/Documents$ cat red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read
```

The pattern `r..f` would find any line that contained the letter `r` followed by exactly two characters and then the letter `f`:

```
sysadmin@localhost:~/Documents$ grep 'r..f' red.txt
```

```
reef

roof
```

The line does not have to be an exact match, it simply must contain the pattern, as seen here when `r..t` is searched for in the `/etc/passwd` file:

```
sysadmin@localhost:~/Documents$ grep 'r..t' /etc/passwd

root:x:0:0:root:/root:/bin/bash

operator:x:1000:37::/root:
```

The period character can be used any number of times. To find all words that have at least four characters, the following pattern can be used:

```
sysadmin@localhost:~/Documents$ grep '....' red.txt

reef

reeed

roof

reed

root

reel

read
```

10.8.2 The Bracket [ ] Characters

When using the `.` character, any possible character could match it. In some cases, you want to specify exactly which characters you want to match, such as a lowercase alphabet character or a number character.

The square brackets `[ ]` match a single character from the list or range of possible characters contained within the brackets. For example, given the `profile.txt` file:

To find all the lines in `profile.txt` which have a number in them, use the pattern `[0123456789]` or `[0-9]`:

```
sysadmin@localhost:~/Documents$ grep '[0-9]' profile.txt

I am 37 years old.

3121991

I have 2 dogs.

123456789101112
```

Note that each possible character can be listed out `[abcd]` or provided as a range `[a-d]`, as long as the range is in the correct order. For example, `[d-a]` wouldn't work because it isn't a valid range:

```
sysadmin@localhost:~/Documents$ grep '[d-a]' profile.txt

grep: Invalid range end
```

The range is specified by a standard called the ASCII table. This table is a collection of all printable characters in a specific order. You can see the ASCII table with the `ascii` command. A small sample:

```
Dec Hex     Dec Hex     Dec Hex    Dec Hex    Dec Hex    Dec Hex     Dec Hex     Dec Hex
  0 00 NUL  16 10 DLE   32 20       48 30 0   64 40 @    80 50 P      96 60 `    112 70 p
  1 01 SOH  17 11 DC1   33 21 !     49 31 1   65 41 A    81 51 Q      97 61 a    113 71 q
  2 02 STX  18 12 DC2   34 22 "     50 32 2   66 42 B    82 52 R      98 62 b    114 72 r
  3 03 ETX  19 13 DC3   35 23 #     51 33 3   67 43 C    83 53 S      99 63 c    115 73 s
  4 04 EOT  20 14 DC4   36 24 $     52 34 4   68 44 D    84 54 T     100 64 d    116 74 t
  5 05 ENQ  21 15 NAK   37 25 %     53 35 5   69 45 E    85 55 U     101 65 e    117 75 u
  6 06 ACK  22 16 SYN   38 26 &     54 36 6   70 46 F    86 56 V     102 66 f    118 76 v
  7 07 BEL  23 17 ETB   39 27 '     55 37 7   71 47 G    87 57 W     103 67 g    119 77 w
  8 08 BS   24 18 CAN   40 28 (     56 38 8   72 48 H    88 58 X     104 68 h    120 78 x
  9 09 HT   25 19 EM    41 29 )     57 39 9   73 49 I    89 59 Y     105 69 i    121 79 y
 10 0A LF   26 1A SUB   42 2A *     58 3A :   74 4A J    90 5A Z     106 6A j    122 7A z
 11 0B VT   27 1B ESC   43 2B +     59 3B ;   75 4B K    91 5B [     107 6B k    123 7B {
 12 0C FF   28 1C FS    44 2C ,     60 3C <   76 4C L    92 5C \     108 6C l    124 7C |
 13 0D CR   29 1D GS    45 2D -     61 3D =   77 4D M    93 5D ]     109 6D m    125 7D }
 14 0E SO   30 1E RS    46 2E .     62 3E >   78 4E N    94 5E ^     110 6E n    126 7E ~
 15 0F SI   31 1F US    47 2F /     63 3F ?   79 4F O    95 5F _     111 6F o    127 7F DEL
sysadmin@localhost:~/Documents$
```

The ASCII value of the letter `a` is `97` while the value of `d` is `100`. Since `97` is smaller than `100`, the range `a-d` (`97-100`) is a valid range.

What about exempting characters?, For instance, to match a character that can be anything except an `x`, `y` or `z`? It would be inefficient to provide a set with all of the characters except `x`, `y` or `z`.

To match a character that is not one of the listed characters, start the set with a `^` symbol. To find all the lines which contain any non-numeric characters, insert a `^` as the first character inside the brackets. This character negates the characters listed:

```
sysadmin@localhost:~/Documents$ grep '[^0-9]' profile.txt
Hello my name is Joe.
I am 37 years old.
My favorite food is avocados.
I have 2 dogs.
```

10.8.3 The Asterisk * Character

The asterisk `*` character is used to match zero or more occurrences of a character or pattern preceding it. For example, `e*` would match zero or more occurrences of the letter `e`:

```
sysadmin@localhost:~/Documents$ grep 're*d' red.txt
red
reeed
rd
```

```
reed
```

It is also possible to match zero or more occurrences of a list of characters by utilizing the square brackets. The pattern `[oe]*` used in the following example matches zero or more occurrences of the `o` character or the `e` character:

```
sysadmin@localhost:~/Documents$ grep 'r[oe]*d' red.txt
red
reeed
rd
rod
reed
```

To make the asterisk character useful, it is necessary to create a pattern which includes more than just the one character preceding it. For example, the results above can be refined by adding another `e` to make the pattern `ee*` effectively matching every line which contains at least one `e`.

```
sysadmin@localhost:~/Documents$ grep 'ee*' red.txt
red
reef
reeed
reed
reel
read
```

10.8.4 Anchor Characters

When performing a pattern match, the match could occur anywhere on the line. Anchor characters are one of the ways regular expressions can be used to narrow down search results. They specify whether the match occurs at the beginning of the line or the end of the line.

For example, the pattern `root` appears many times in the `/etc/passwd` file:

```
sysadmin@localhost:~/Documents$ grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:1000:37::/root:
```

The caret (circumflex) `^` character is used to ensure that a pattern appears at the beginning of the line. For example, to find all lines in `/etc/passwd` that start with `root` use the pattern `^root`. Note that `^` must be the first character in the pattern to be effective:

```
sysadmin@localhost:~/Documents$ grep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

The second anchor character `$` can be used to ensure a pattern appears at the end of the line, thereby effectively reducing the search results. To find the lines that end with an `r` in the `alpha-first.txt` file, use the pattern `r$`:

```
sysadmin@localhost:~/Documents$ cat alpha-first.txt

A is for Animal

B is for Bear

C is for Cat

D is for Dog

E is for Elephant

F is for Flower

sysadmin@localhost:~/Documents$ grep 'r$' alpha-first.txt

B is for Bear

F is for Flower
```

Again, the position of this character is important. The `$` must be the last character in the pattern to be effective as an anchor.

10.8.5 The Backslash \ Character

In some cases, you may want to match a character that happens to be a special regular expression character. For example, consider the following:

```
sysadmin@localhost:~/Documents$ cat newhome.txt
Thanks for purchasing your new home!!


**Warning** it may be haunted.


There are three bathrooms.


**Beware** of the ghost in the bedroom.


The kitchen is open for entertaining.


**Caution** the spirits don't like guests.


Good luck!!!
sysadmin@localhost:~/Documents$ grep 're*' newhome.txt
Thanks for purchasing your new home!!
**Warning** it may be haunted.
```

```
There are three bathrooms.

**Beware** of the ghost in the bedroom.

The kitchen is open for entertaining.

**Caution** the spirits don't like guests.
```

In the output of the `grep` command above, the search for `re*` matched every line which contained an `r` followed by zero or more of the letter `e`. To look for an actual asterisk `*` character, place a backslash `\` character before the asterisk `*` character:

```
sysadmin@localhost:~/Documents$ grep 're\*' newhome.txt

**Beware** of the ghost in the bedroom.
```

10.8.6 Extended Regular Expressions

The use of extended regular expressions often requires a special option be provided to the command to recognize them. Historically, there is a command called `egrep`, which is similar to `grep`, but can understand extended regular expressions. Now, the `egrep` command is deprecated in favor of using `grep` with the `-E` option.

The following regular expressions are considered extended:

Character	Meaning
?	Matches previous character zero or one time, so it is an optional character
+	Matches previous character repeated one or more times
|	Alternation or like a logical "or" operator

To match `colo` followed by zero or one `u` character followed by an `r` character:

```
sysadmin@localhost:~/Documents$ grep -E 'colou?r' spelling.txt
American English: Do you consider gray to be a color or a shade?
British English: Do you consider grey to be a colour or a shade?
```

To match one or more `e` characters:

```
sysadmin@localhost:~/Documents$ grep -E 'e+' red.txt
red
reef
reeed
reed
reel
```

```
read
```

To match either `gray` or `grey`:

```
sysadmin@localhost:~/Documents$ grep -E 'gray|grey' spelling.txt
American English: Do you consider gray to be a color or a shade?
British English: Do you consider grey to be a colour or a shade?
```

Lab:

```
sysadmin@localhost:~$ find /etc -name hosts > find.out 2>&1
sysadmin@localhost:~$ cat find.out
/etc/hosts
find: '/etc/ssl/private': Permission denied
```

The `2>&1` part of the command means "send the `stderr` (channel 2) to the same place where `stdout` (channel 1) is going".

Suppose you want to search for a pattern containing a sequence of three digits. You can use `{ }` characters with a number to express that you want to repeat a pattern a specific number of times; for example: `{3}`. The use of the numeric qualifier requires the extended mode of `grep`:

```
grep -E '[0-9]{3}' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep -E '[0-9]{3}' passwd
sync:x:4:65534:sync:/bin:/bin/sync
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd/
netif:/usr/
sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd/resolve:/u
sr/sbi
```

Chapter 11: Basic Scripting

11.1 Introduction

In this chapter, we will discuss how the tools you have learned so far can be turned into reusable scripts.

11.2 Shell Scripts in a Nutshell

A shell script is a file of executable commands that has been stored in a text file. When the file is run, each command is executed. Shell scripts have access to all the commands of the shell, including logic. A script can therefore test for the presence of a file or look for particular output and change its behavior accordingly. You can build scripts to automate repetitive parts of your work, which frees your time and ensures consistency each time you use the script. For instance, if you run the same five commands every day, you can turn them into a shell script that reduces your work to one command.

A script can be as simple as one command:

```
echo "Hello, World!"
```

The script, `test.sh`, consists of just one line that prints the string `Hello, World!` to the console.

The script, `test.sh`, consists of just one line that prints the string `Hello, World!` to the console.

Note

This file is not available within the virtual machine environment of this course.
Running a script can be done either by passing it as an argument to your shell or by running it directly:

```
sysadmin@localhost:~$ sh test.sh
Hello, World!
sysadmin@localhost:~$ ./test.sh
-bash: ./test.sh: Permission denied
sysadmin@localhost:~$ chmod +x ./test.sh
sysadmin@localhost:~$ ./test.sh
Hello, World!
```

In the example above, first, the script is run as an argument to the shell. Next, the script is run directly from the shell. It is rare to have the current directory in the binary search path `$PATH` so the name is prefixed with `./` to indicate that it should be run out of the current directory.

The error `Permission denied` means that the script has not been marked as executable. A quick `chmod` later and the script works. `chmod` is used to change the permissions of a file, which will be explained in detail in a later chapter.

There are various shells with their own language syntax. Therefore, more complicated scripts will indicate a particular shell by specifying the absolute path to the interpreter as the first line, prefixed by `#!` as shown:

```
#!/bin/sh

echo "Hello, World!"
```

or

```
#!/bin/bash

echo "Hello, World!"
```

The two characters `#!` are traditionally called the hash and the bang respectively

11.3 Editing Shell Scripts

UNIX has many text editors. The merits of one over the other are often hotly debated. Two are specifically mentioned in the LPI Essentials syllabus: The GNU nano editor is a very simple editor well suited to editing small text files. The Visual Editor, `vi`, or its newer version, VI improved (`vim`), is a remarkably powerful editor but has a steep learning curve. We'll focus on `nano`.

Type `nano test.sh` and you'll see a screen similar to this:

```
GNU nano 2.2.6              File: test.sh                        mod
ified


#!/bin/sh


echo "Hello, World!"
echo -n "the time is "
date













^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C C
ur Po
```

```
^X Exit       ^J Justify    ^W Where Is   ^V Next Page ^U UnCut Text^T T
o Spell
```

The nano editor has few features to get you on your way. You simply type with your keyboard, using the **arrow keys** to move around and the **delete/backspace** button to delete text. Along the bottom of the screen you can see some commands available to you, which are context-sensitive and change depending on what you're doing. If you're directly on the Linux machine itself, as opposed to connecting over the network, you can also use the mouse to move the cursor and highlight text.

To get familiar with the editor, start typing out a simple shell script while inside nano:

```
GNU nano 2.2.6              File: test.sh                          mod
ified


#!/bin/sh


echo "Hello, World!"
echo -n "the time is "
date









^G Get Help  ^O WriteOut   ^R Read File ^Y Prev Page ^K Cut Text   ^C C
ur Po
^X Exit       ^J Justify    ^W Where Is   ^V Next Page ^U UnCut Text^T T
o Spell
```

Note that the bottom-left option is ^X Exit which means "press **control** and **X** to exit". Press **Ctrl** and **X** together and the bottom will change:

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No              ^C Cancel
```

At this point, you can exit the program without saving by pressing the **N** key, or save first by pressing **Y** to save. The default is to save the file with the current file name. You can press the **Enter** key to save and exit.

Command	Description
Ctrl + **W**	search the document

Command	Description
Ctrl + **W**, then **Control** + **R**	search and replace
Ctrl + **G**	show all the commands possible
Ctrl + **Y/V**	page up / down
Ctrl + **C**	show the current position in the file and the file's size

11.4 Scripting Basics

The script started with the shebang (or hashbang) line, telling Linux that `/bin/bash` (which is Bash) is to be used to execute the script.

Other than running commands, there are 3 topics you must become familiar with:

- Variables, which hold temporary information in the script
- Conditionals, which let you do different things based on tests you write
- Loops, which let you do the same thing over and over

11.4.1 Variables

Variables are a key part of any programming language. A very simple use of variables is shown here:

```
#!/bin/bash


ANIMAL="penguin"
echo "My favorite animal is a $ANIMAL"
```

After the shebang line is a directive to assign some text to a variable. The variable name is `ANIMAL` and the equals sign assigns the string `penguin`. Think of a variable like a box in which you can store things. After executing this line, the box called `ANIMAL` contains the word `penguin`.

It is important that there are no spaces between the name of the variable, the equals sign, and the item to be assigned to the variable. If you have a space there, you will get an odd error such as "`command not found`". Capitalizing the name of the variable is not necessary but it is a useful convention to separate variables from commands to be executed.

So remember this: to assign to a variable, just use the name of the variable. To access the contents of the variable, prefix it with a dollar sign. Here, we show a variable being assigned the contents of another variable!

Another way to assign to a variable is to use the output of another command as the contents of the variable by enclosing the command in back ticks:

```
#!/bin/bash

CURRENT_DIRECTORY=`pwd`

echo "You are in $CURRENT_DIRECTORY"
```

It is possible to get input from the user of your script and assign it to a variable through the read command:

```
#!/bin/bash


echo -n "What is your name? "

read NAME

echo "Hello $NAME!"
```

The read command can accept a string right from the keyboard or as part of command redirection like you learned in the last chapter.

There are some special variables in addition to the ones you set. You can pass arguments to your script:

```
#!/bin/bash
echo "Hello $1"
```

A dollar $ sign followed by a number N corresponds to the Nth argument passed to the script. If you call the example above with ./test.sh World the output will be Hello World. The $0 variable contains the name of the script itself.

After a program runs, be it a binary or a script, it returns an exit code which is an integer between 0 and 255. You can test this through the $? variable to see if the previous command completed successfully.

```
sysadmin@localhost:~$ grep -q root /etc/passwd
sysadmin@localhost:~$ echo $?
0
sysadmin@localhost:~$ grep -q slartibartfast /etc/passwd
sysadmin@localhost:~$ echo $?
1
```

The grep command was used to look for a string within a file with the -q flag, which means "quiet". The grep, while running in quiet mode, returns 0 if the string was found and 1 otherwise. This information can be used in a conditional to perform an action based on the output of another command.

Likewise you can set the exit code of your own script with the `exit` command:

```
#!/bin/bash

# Something bad happened!

exit 1
```

The example above shows a comment `#`. Anything after the hash mark is ignored, which can be used to help the programmer leave notes. The `exit 1` returns exit code 1 to the caller. This even works in the shell, if you run this script from the command line and then type `echo $?` you will see it returns 1.

By convention, an exit code of `0` means "everything is OK". Any exit codes greater than `0` mean some kind of error happened, which is specific to the program. Above you saw that `grep` uses 1 to mean the string was not found.

11.4.2 Conditionals

Now that you can look at and set variables, it is time to make your script do different functions based on tests, called branching. The if statement is the basic operator to implement branching.

A basic `if` statement looks like this:

```
if somecommand; then

  # do this if somecommand has an exit code of 0

fi
```

The next example will run "somecommand" (actually, everything up to the semicolon) and if the exit code is `0` then the contents up until the closing `fi` will be run. Using what you know about `grep`, you can now write a script that does different things based on the presence of a string in the password file:

```
#!/bin/bash

if grep -q root /etc/passwd; then

  echo root is in the password file

else

  echo root is missing from the password file

fi
```

From previous examples, you might remember that the exit code of `grep` is `0` if the string is found. The example above uses this in one line to print a message if `root` is in the password file or a different message if it isn't. The difference here is that instead of an `fi` to close off the `if` block, there's an `else`. This lets you do one action if the condition is true, and another if the condition is false. The `else` block must still be closed with the `fi` keyword.

Other common tasks are to look for the presence of a file or directory and to compare strings and numbers. You might initialize a log file if it doesn't exist, or compare the

number of lines in a file to the last time you ran it. The `if` command is clearly the one to help here, but what command do you use to make the comparison?

The `test` command gives you easy access to comparison and file test operators. For example:

Command	Description
`test –f /dev/ttyS0`	`0` if the file exists
`test ! –f /dev/ttyS0`	`0` if the file doesn't exist
`test –d /tmp`	`0` if the directory exists
`test –x `which ls``	substitute the location of `ls` then `test` if the user can execute
`test 1 –eq 1`	`0` if numeric comparison succeeds
`test ! 1 –eq 1`	`NOT – 0` if the comparison fails
`test 1 –ne 1`	Easier, `test` for numeric inequality
`test "a" = "a"`	`0` if the string comparison succeeds
`test "a" != "a"`	`0` if the strings are different
`test 1 –eq 1 –o 2 –eq 2`	`-o` is OR: either can be the same
`test 1 –eq 1 –a 2 –eq 2`	`-a` is AND: both must be the same

```
if test ! -f /dev/ttyS0; then
echo ttyS0 file is missing from the dev directory
else
echo ttyS0 file is in the dev directory
fi

echo "Testing is tmp directory exists"
if test -d /tmp; then
echo tmp directory is exists
else
echo tmp directory is missing
fi

echo "Substitute the location of ls then test if the user can execute"
if test -X `which is`; then
echo yes
else
echo  no
fi
                              [ Read 29 lines ]
^G Get Help    ^O Write Out  ^W Where Is    ^K Cut Text   ^J Justify     ^C Cur Pos
^X Exit        ^R Read File  ^\ Replace     ^U Uncut Text ^T To Linter   ^_ Go To Line
```

```
sysadmin@localhost:~$ bash test2.sh
Testing if ttyS0 file is in dev directory
ttyS0 file is missinng from the dev directory file
Testing if ttyS0 is not in dev directory
ttyS0 file is missing from the dev directory
Testing is tmp directory exists
tmp directory is exists
Substitute the location of ls then test if the user can execute
yes
```

It is important to note that `test` looks at integer and string comparisons differently. `01` and `1` are the same by numeric comparison, but not by string comparison. You must always be careful to remember what kind of input you expect.

There are many more tests, such as `-gt` for greater than, ways to test if one file is newer than the other, and many more. Consult the `test man` page for more.

`test` is fairly verbose for a command that gets used so frequently, so there is an alias for it called `[` (left square bracket). If you enclose your conditions in square brackets, it's the same as running `test`. So, these statements are identical.

```
if test -f /tmp/foo; then

if [ -f /tmp/foo]; then
```

While the latter form is most often used, it is important to understand that the square bracket is a command on its own that operates similarly to `test` except that it requires the closing square bracket.

The `if` statement has a final form that lets you do multiple comparisons at one time using `elif` (short for `else if`).

```
#!/bin/bash


if [ "$1" = "hello" ]; then

  echo "hello yourself"

elif [ "$1" = "goodbye" ]; then

  echo "nice to have met you"

  echo "I hope to see you again"

else

  echo "I didn't understand that"

fi
```

The code above compares the first argument passed to the script. If it is `hello`, the first block is executed. If not, the script checks to see if it is `goodbye` and echos a different message if so. Otherwise, a third message is sent. Note that the `$1` variable is quoted and the string comparison operator is used instead of the numeric version (`-eq`).

The `if`/`elif`/`else` tests can become quite verbose and complicated.
The `case` statement provides a different way of making multiple tests easier.

```
#!/bin/bash


case "$1" in

hello|hi)

  echo "hello yourself"

  ;;

goodbye)

  echo "nice to have met you"

  echo "I hope to see you again"

  ;;

*)

  echo "I didn't understand that"

esac
```

The `case` statement starts off with a description of the expression being tested: `case EXPRESSION in`. The expression here is the quoted `$1`.

Next, each set of tests are executed as a pattern match terminated by a closing parenthesis. The previous example first looks for `hello` or `hi`; multiple options are separated by the vertical bar `|` which is an OR operator in many programming languages. Following that are the commands to be executed if the pattern returns true, which are terminated by two semicolons. The pattern repeats.

The `*` pattern is the same as an `else` because it matches anything. The behavior of the `case` statement is similar to the `if`/`elif`/`else` statement in that processing stops

after the first match. If none of the other options matched, the * ensures that the last one will match.

With a solid understanding of conditionals you can have your scripts take actions only if necessary.

11.4.3 Loops

Loops allow code to be executed repeatedly. They can be useful in numerous situations, such as when you want to run the same commands over each file in a directory, or repeat some action 100 times. There are two main loops in shell scripts: the `for` loop and the `while` loop.

`for` loops are used when you have a finite collection over which you want to iterate, such as a list of files, or a list of server names:

```
#!/bin/bash


SERVERS="servera serverb serverc"

for S in $SERVERS; do

  echo "Doing something to $S"

done
```

The script first sets a variable containing a space separated list of server names. The `for` statement then loops over the list of servers, each time it sets the S variable to the current server name. The choice of S was arbitrary, but note that the S has no dollar sign but the $SERVERS does, showing that $SERVERS will be expanded to the list of servers. The list does not have to be a variable. This example shows two more ways to pass a list.

```
#!/bin/bash


for NAME in Sean Jon Isaac David; do

  echo "Hello $NAME"

done


for S in *; do

  echo "Doing something to $S"

done
```

The first loop is functionally the same as the previous example, except that the list is passed to the `for` loop directly instead of using a variable. Using a variable helps the clarity of the script as someone can easily make changes to the variable rather than looking at a loop.

The second loop uses a * which is a file glob. This gets expanded by the shell to all the files in the current directory.

```
  GNU nano 2.9.3                          loop.sh

#!/bin/bash


SERVERS="servera serverb serverc"

for S in $SERVERS;do
echo "Doing something to $S"
done



for NAME in Sean Jon David;do
echo "Hello $NAME"
done

echo "this will loop through the files in the current directory"
for S in *;do
echo " $S"
done



                         [ Read 17 lines ]
^G Get Help   ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit       ^R Read File ^\ Replace   ^U Uncut Text^T To Linter ^_ Go To Line
```

```
sysadmin@localhost:~$ nano loop.sh
sysadmin@localhost:~$ sh loop.sh
Doing something to servera
Doing something to serverb
Doing something to serverc
Hello Sean
Hello Jon
Hello David
this will loop through the files in the current directory
 Desktop
 Documents
 Downloads
 Music
 Pictures
 Public
 Templates
 Videos
 case.sh
 loop.sh
 test.sh
sysadmin@localhost:~$
```

The other type of loop, a `while` loop, operates on a list of unknown size. Its job is to keep running and on each iteration perform a `test` to see if it should run another time. You can think of it as "while some condition is true, do stuff."

```bash
#!/bin/bash

i=0

while [ $i -lt 10 ];do
echo $i
i=$(( $i + 1 ))
done
echo "Done counting"
```

```
sysadmin@localhost:~$ sh while.sh
0
1
2
3
4
5
6
7
8
9
Done counting
```

The example above shows a `while` loop that counts from 0 to 9. A counter variable, `i`, is initialized to `0`. Then a `while` loop is run with the `test` being "is `$i` less than `10`?" Note that the `while` loop uses the same notation as an `if` statement!

Within the `while` loop the current value of `i` is echoed and then `1` is added to it through the `$(( arithmetic ))` command and assigned back into `i`. Once `i` becomes 10 the `while` statement returns false and processing continues after the loop

Lab

11.2.45 Step 45

Save the file and close the `vi` editor using any one of the following methods
that saves changes:

Command	Function/Keys
`:x`	Will save and close the file.
`:wq`	Will write to file and quit.
`:wq!`	Will write to a read-only file, if possible, and quit.

Command	Function/Keys
ZZ	Will save and close. Notice that no colon ： is used in this case.
:q!	Exit without saving changes
:e!	Discard changes and reload file
:w!	Write to read-only, if possible.

Simple script using vi editor

```
#!/bin/bash
echo "Please enter your age"
read age
if test $age -lt 16
then
echo "you are not old enough to drive"
else
echo "You can dirve"
fi




~
~
~
~
~
~
~
~
~
~
~
~
~
"dive.sh" 10 lines, 145 characters
```

```
sysadmin@localhost:~$ sh dive.sh
Please enter your age
23
You can dirve
sysadmin@localhost:~$ vi dive.sh
sysadmin@localhost:~$ sh dive.sh
Please enter your age
13
you are not old enough to drive
sysadmin@localhost:~$
```

```
#!/bin/bash
echo "Please enter a number greater than 100"
read num
while [ $num -le 100 ]
do
echo "$num is not greater than 100"
read num
done
echo "Finally , $num is greater than 100"
~
~
~
```

```
sysadmin@localhost:~$ sh  num.sh
Please enter a number greater than 100
12
12 is not greater than 100
190
Finally , 190 is greater than 100
```

Scripting code is part of the BASH shell, which means you can use these statements on the command line just like you use them in a shell script. This can be useful for a statement like the `for` statement, a statement that will assign a list of values one at a time to a variable. This allows you to perform a set of operations on each value. For example, run the following on the command line:

```
sysadmin@localhost:~$ for name in /etc/passwd /etc/hosts /etc/group
> do
> wc $name
> done
  28   37 1455 /etc/passwd
   7  16 172 /etc/hosts
  51  51 666 /etc/group
sysadmin@localhost:~$
```

Often the `seq` command is used in conjunction with the `for` statement.
The `seq` command can generate a list of integer values, for instance from 1 to 10. For example, run the following on the command line to create 12 files
named `test1`, `test2`, `test3`, etc. (up to `test12`):

```
sysadmin@localhost:~$ for num in `seq 1 5`
> do
> touch test$num
> done
sysadmin@localhost:~$ ls
Desktop     Music      Templates  myfile     test1  test4
Documents   Pictures   Videos     num.sh     test2  test5
Downloads   Public     dive.sh    sample.sh  test3
sysadmin@localhost:~$
```

Chapter 12: Understanding Computer Hardware

12.1 Introduction

Computers begin as a hardware device but can grow to encompass virtual machines that are entirely constructed out of software and appear to all purposes to be a remote machine that users can access via remote protocols and methods.

This chapter introduces and explains the basics of what makes up a computer and expands to a few optional items that are fairly commonplace.

12.2 Motherboards

The motherboard, or system board, is the main hardware board in the computer through which the central processing unit (CPU), random-access memory (RAM) and other components are all connected. Depending on the computer type (laptop, desktop, server) some devices, such as processors and RAM, are attached directly to the motherboard, while other devices, such as add-on cards, are connected via a bus.

12.3 Processors

A central processing unit (CPU or processor) is one of the most critical hardware components of a computer. The processor is the brain of the computer, where the execution of code takes place and where most calculations are done. It is directly connected (soldered) to the motherboard, as motherboards are typically configured to work with specific types of processors.

If a hardware system has more than one processor, the system is referred to as a multiprocessor. If more than one processor is combined into a single overall processor chip, then it is called multi-core.

Although support is available for more types of processors in Linux than any other operating system, there are primarily just two types of processors used on desktop and

server computers: x86 and x86_64. On an x86 system, the system processes data 32 bits at a time; on an x86_64 the system processes data 64 bits at a time. An x86_64 system is also capable of processing data 32 bits at a time in a backward compatible mode. One of the main advantages of a 64-bit system is the ability to work with more memory, while other advantages include increased efficiency of processing and increased security.

Intel originated the x86 family of processors in 1978 with the release of the 8086 processor. Since that time, Intel has produced many other processors that are improvements to the original 8086; they are known generically as x86 processors. These processors include the 80386 (i386), 80486 (i486), the Pentium series (i586) and the Pentium Pro series (i686). In addition to Intel, other companies such as AMD and Cyrix have also produced x86 compatible processors. While Linux is capable of supporting processors back to the i386 generation, many distributions (especially the ones that feature corporate support, such as SUSE, Red Hat and Canonical) limit their support to i686 or later.

The x86_64 family of processors, including the 64-bit processors from Intel and AMD, have been in production since around the year 2000. As a result, almost all of the modern processors shipped today are of the x86_64 type. While the hardware has been available for over a decade now, the software to support this family of processors has been much slower to develop. Although it's 2018 at the time of this update, there remain a number of packages that are still available for the x86 architecture.

To see which family the CPU of the current system belongs to, use the `arch` command:

```
sysadmin@localhost:~$ arch
x86_64
```

For more information concerning the CPU, use the `lscpu` command:

```
x86_64
sysadmin@localhost:~$ lscpu
Architecture:         x86_64
CPU op-mode(s):       32-bit, 64-bit
Byte Order:           Little Endian
CPU(s):               16
On-line CPU(s) list:  0-15
Thread(s) per core:   1
Core(s) per socket:   1
Socket(s):            16
NUMA node(s):         1
Vendor ID:            AuthenticAMD
CPU family:           25
Model:                1
Model name:           AMD EPYC 7443P 24-Core Processor
Stepping:             1
CPU MHz:              2844.656
BogoMIPS:             5689.31
Virtualization:       AMD-V
Hypervisor vendor:    VMware
Virtualization type:  full
```

The first line of this output shows that the CPU is being used in a 64-bit mode, as the architecture reported is x86_64. The second line of output shows that the CPU is capable of operating in either a 32- or 64-bit mode, therefore, it is actually a 64-bit CPU.

12.4 Random Access Memory

The motherboard typically has slots where random-access memory (RAM) can be connected to the system. The 32-bit architecture systems can use up to 4 gigabytes (GB) of RAM, while 64-bit architectures are capable of addressing and using far more RAM.

In some cases, the RAM a system has might not be enough to handle all of the operating system requirements. Once invoked, programs are loaded in RAM along with any data they need to store, while instructions are sent to the processor when they execute.

When the system is running low on RAM, virtualized memory called swap space is used to temporarily store data to disk. Data stored in RAM that has not been used recently is moved over to the section of the hard drive designated as swap, freeing up more RAM for use by currently active programs. If needed, this swapped data can be moved back into RAM at a later time. The system does all of this automatically as long as swap is configured.

To view the amount of RAM in your system, including the swap space, execute the `free` command. The `free` command has a `-m` option to force the output to be rounded to the nearest megabyte (MB) and a `-g` option to force the output to be rounded to the nearest gigabyte (GB):

```
sysadmin@localhost:~$ free -m
             total       used       free     shared    buffers     ca
ched
Mem:          1894        356       1537          0         25
177
-/+ buffers/cache:         153       1741
Swap:         4063          0       4063
```

The output shows that this system has a total of 1,894 MB and is currently using 356 MB.

The amount of swap appears to be approximately 4 GB, although none of it appears to be in use. This makes sense because so much of the physical RAM is free, so there is no need at this time for virtual RAM (swap space) to be used.

12.5 Buses

A bus is a high-speed connection that allows communication between computers or the components inside a computer. The motherboard has buses that allow for multiple devices to connect to the system, including the Peripheral Component Interconnect (PCI) and Universal Serial Bus (USB). The motherboard also has connectors for monitors, keyboards and mice.

Different system types will use a bus differently to connect components. In a desktop or server computer, there is a motherboard with the processor, RAM, and other components directly attached, and then a high-speed bus that allows additional components to be attached via card slots, such as video, network and other peripheral devices.

Increasingly for laptop and light/thin/small form factor computers, the majority of the computer's components may be directly connected to the motherboard, including the usual processor, RAM, as well as additional components like the network card. In this

configuration, the bus still exists, but effectively one of the main convenience factors of being able to swap out or upgrade devices is no longer a possibility.

Peripheral Devices

Peripheral devices are components connected to a computer that allow input, output or data storage. Keyboards, mice, monitors, printers and hard disks are all considered peripherals and are accessed by the system in order to perform functions outside of central processing. To view all of the devices connected by the PCI bus, the user can execute the `lspci` command. The following is a sample output of this command. The highlighted sections show this system has a VGA controller (a monitor connector), an SCSI storage controller (a type of hard drive) and an Ethernet controller (a network connector):

```
sysadmin@localhost:~$ lspci

00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Ho
st bridge (rev 01)

00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP
bridge (rev 01)

00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08
)

00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev
01)

00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)

00:07.7 System peripheral: VMware Virtual Machine Communication Inter
face (rev 10)

00:0f.0 VGA compatible controller: VMware SVGA II Adapter
03:00.0 Serial Attached SCSI controller: VMware PVSCSI SCSI Controlle
r (rev 02
0b:00.0 Ethernet controller: VMware VMXNET3 Ethernet Controller (rev
01)
```

Universal Serial Bus Devices

While the PCI bus is used for many internal devices such as sound and network cards, an ever-increasing number of external devices (or peripherals) are connected to the computer via USB.

Devices connected internally are usually cold-plug, meaning the system must be shut down in order to connect or disconnect a device. USB devices are hot-plug, meaning they can be connected or disconnected while the system is running.

Though USB devices are hot-plug by design, it's important to ensure that any mounted filesystems are correctly unmounted, or data loss and corruption of the filesystem may occur.

To display the devices connected to the system via USB, execute the `lsusb` command:

```
sysadmin@localhost:~$ lsusb

Bus 001 Device 002: ID 0e0f:000b VMware, Inc.

Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 002 Device 004: ID 0e0f:0008 VMware, Inc.
```

```
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub

Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse

Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

12.6 Hard Drives

Hard drives, also called hard disks, disk devices, or storage devices, may be attached to the system in a number of ways; the controller may be integrated into the motherboard, on a PCI card, or as a USB device. For the purposes of most Linux systems, a hard drive can generally be defined as any electromechanical or electronic storage device that holds data to be accessed by the system.

Hard drives are divided into one or more partitions. A partition is a logical division of a hard drive, designed to take a large amount of available storage space and break it up into smaller areas. While it is common on Microsoft Windows to have a single partition for each hard drive, on Linux distributions, multiple partitions per hard drive is common.

Some hard drives make use of a partitioning technology called Master Boot Record (MBR) while others make use of a partitioning type called GUID Partitioning Table (GPT). The MBR type of partitioning has been used since the early days of the Personal Computer (PC), and the GPT type has been available since the year 2000.

An old term used to describe an internal hard disk is fixed disk, as the disk is fixed (not removable). This term gave rise to several command names: the `fdisk`, `cfdisk` and `sfdisk` commands, which are tools for working with the MBR partitioned disks.

The GPT disks use a newer type of partitioning, which allows the user to divide the disk into more partitions than what MBR supports. GPT also allows having partitions which can be larger than two terabytes (MBR does not). The tools for managing GPT disks are named similarly to their `fdisk` counterparts: `gdisk`, `cgdisk`, and `sgdisk`.

There is also a family of tools that attempt to support both MBR and GPT type disks. This set of tools includes the `parted` command and the graphical `gparted` tool.

Note

Many Linux distributions use the `gparted` tool in the installation routine, as it provides a graphical interface to the powerful options available for creating new partitions, resizing existing ones (such as Windows partitions) and many other advanced tasks.

Hard drives are associated with file names (called device files) that are stored in the `/dev` directory. Each device file name is made up of multiple parts.

- **File Type**

  The file name is prefixed based on the different types of hard drives. IDE (Intelligent Drive Electronics) hard drives begin with `hd`, while USB, SATA (Serial Advanced Technology Attachment) and SCSI (Small Computer System Interface) hard drives begin with `sd`.

- **Device Order**

  Each hard drive is then assigned a letter which follows the prefix. For example, the first IDE hard drive would be named `/dev/hda` and the second would be `/dev/hdb`, and so on.

- **Partition**

Finally, each partition on a disk is given a unique numeric indicator. For example, if a USB hard drive has two partitions, they could be associated with the `/dev/sda1` and `/dev/sda2` device files.

The following example shows a system that has three `sd` devices: `/dev/sda`, `/dev/sdb` and `/dev/sdc`. Also, there are two partitions on the first device, as evidenced by the `/dev/sda1` and `/dev/sda2` files, and one partition on the second device, as evidenced by the `/dev/sdb1` file:

```
root@localhost:~$ ls /dev/sd*
/dev/sda /dev/sda1  /dev/sda2  /dev/sdb  /dev/sdb1  /dev/sdc
```

The `fdisk` command can be used to display further information about the partitions:

Note

The following command requires root access.

```
root@localhost:~$ fdisk -l /dev/sda

Disk /dev/sda: 21.5 GB, 21474836480 bytes

255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors

Units = sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x000571a2


   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    39845887    19921920   83  Linux
/dev/sda2          39847934    41940991     1046529    5  Extended
/dev/sda5          39847936    41940991     1046528   82  Linux swap /
Solaris
```

12.7 Solid State Disks

While the phrase hard disk is typically considered to encompass traditional spinning disk devices, it can also refer to the newer and very different solid state drives or disks.

Consider the difference between a traditional spinning platter hard disk and a USB thumb drive. The former literally has spinning disk platters in it that are read by drive heads, and the spinning disks are laid out to take advantage of the spinning nature of the device. Data is written (and read) in long strings of sequential blocks that a drive head encounters as the platter spins around.

Solid state disks are essentially larger capacity USB thumb drives in construction and function. As there are no moving parts, and no spinning disks, just memory locations to be read by the controller, a solid state disk is measurably and visibly faster in accessing the information stored in its memory chips.

A solid state disk is controlled by an embedded or onboard processor that makes the decisions as to where and how data is written and read back from the memory chips when requested.

Advantages of a solid state disk include lower power usage, time savings in system booting, faster program loads, and less heat and vibration from no moving parts. Disadvantages include higher costs in comparison to spinning hard disks, lower capacity due to the higher cost, and if soldered directly on the motherboard/mainboard, no ability to upgrade by swapping out the drive.

Note

For the purposes of this course, and for understanding, we will refer to all disks, spinning or solid state, as hard disks unless there is an important distinction that requires us to differentiate between the two.

12.8 Optical Drives

Optical drives, often referred to as CD-ROMs, DVDs, or Blu-Ray, are removable storage media. While some devices used with optical disks are read-only, others are capable of burning (writing to) disks, when using a writable type of disk. There are various standards for writable and rewritable disks, such as CD-R, CD+R, DVD+RW, and DVD-RW.

Where these removable disks are mounted in the file system is an important consideration for a Linux administrator. Modern distributions often mount the disks under the `/media` folder, while older distributions typically mount them under the `/mnt` folder. For example, a USB thumb drive might be mounted on the `/media/usbthumb` path.

Upon mounting, most GUI interfaces prompt the user to take some action, such as open the contents of the disk in a file browser or start a media program. When the user is finished using the disk, it is necessary to unmount it using a menu or the `umount` command.

12.9 Managing Devices

Linux by nature has a number of distributions or versions, a large portion of which are focused on mainstream hardware, while others run on relatively obscure types or models of hardware. It's safe to say that there is a Linux distribution specifically designed for just about all modern hardware platforms, and for many historical platforms as well.

Each of these hardware platforms has a great deal of variety in the components that are available. In addition to several different types of hard drives, there are many different graphics cards, monitors and printers. With the popularity of USB devices, such as USB storage devices, cameras, and mobile phones, the number of available devices you would want to connect to a Linux system can number in the thousands.

The sheer number of different devices poses problems as these hardware devices typically need drivers, software that allows them to communicate with the installed operating system.

The driver could be compiled as part of the Linux kernel, loaded into the kernel as a module, or loaded by a user command or application. External devices, like scanners and printers, typically have their drivers loaded by an application and these drivers, in turn, communicate through the device via the kernel through an interface such as USB.

Hardware manufacturers often provide this software, but typically for Microsoft Windows, not for Linux. This has been changing with the increasing popularity of Linux overall, but the desktop market share of Linux remains a very distant third to Microsoft and Apple systems.

Partly because there is relatively sparse vendor support, there is a great deal of community support from developers who strive to provide drivers for Linux systems. Although not all hardware has the necessary drivers, there is a decent amount that does, making the challenge for Linux users and administrators to either find the correct drivers or choose hardware that has some level of support in Linux.

In order to be successful in enabling devices in Linux, it is best to check the Linux distribution to see if the device is certified to work with that distribution. Commercial distributions like Red Hat and SUSE have web pages dedicated to listing hardware that is certified or approved to work with their software.

Additional tips on being successful with connecting your devices: avoid brand new or highly-specialized devices and check with the vendor of the device to see if they support Linux before making a purchase.

12.10 Video Display Devices

In order to display output to the monitor, the computer system must have a video display device (video card) and a monitor. Video display devices are often directly built into or attached to the motherboard, although they can also be connected through the PCI bus slots on the motherboard.

With the large array of video device vendors, each video display device usually requires a proprietary driver provided by the vendor. All drivers, but most especially video device drivers, must be written for the specific operating system, something that is commonly done for Microsoft Windows, but not always for Linux. Fortunately, most of the larger video display vendors now provide at least some level of Linux support. The Linux community has done an incredible amount of work developing standard drivers for video cards.

Note

The video support issues that Linux has experienced are mostly focused on the desktop market for Linux. Server market share is mostly unaffected by the GUI, as the vast majority of Linux server implementations are text mode.

There are three types of video cables commonly used:

- the older but still used analog 15-pin Video Graphics Array (VGA) cable
- the more recent 29-pin Digital Visual Interface (DVI) interface
- the very widely-used High-Definition Multimedia Interface (HDMI) which supports resolutions up to the 4K or Ultra HD range, and has either 19 or 29 pins
- the newest digital display interface, the 20-pin DisplayPort (DP). Also its miniaturized counterpart, the Mini DisplayPort, used mainly for Apple products.

For monitors to work correctly with video display devices, they must be able to support the same resolution as the video display device. Normally, the software driving the video display device can automatically detect the maximum resolution that the video display and monitor can both support and set the screen resolution to that value.

12.11 Power Supplies

Power supplies are the devices that convert alternating current (120v, 240v) into direct current, which the computer uses at various voltages (3.3v, 5v, 12v). Power supplies are generally not programmable; however, their proper function has a major impact on the rest of the system.

Although they are not typically designed as surge suppressors, these devices often protect the computer from fluctuations in voltage coming from the power source. It is wise for a network administrator to choose a power supply based on quality rather than price since a failing power supply can result in significant damage to a computer system.

Desktop, server tower, rack and standalone systems are more vulnerable to power fluctuations than laptop systems are. If the power fluctuates, it can cause much havoc on the non-battery based systems, whereas a laptop simply keeps pulling power from its internal battery until depleted.

lab

In order to determine the type of CPU execute the `lscpu` command:

```
lscpu
```

Your output will be similar to the following:

```
sysadmin@localhost:~$ lscpu
Architecture:         x86_64
CPU op-mode(s):       32-bit, 64-bit
Byte Order:           Little Endian
CPU(s):               16
On-line CPU(s) list:  0-15
Thread(s) per core:   1
Core(s) per socket:   1
Socket(s):            16
```

Step 2

Being able to display CPU information can be important when trying to determine if more advanced Linux features can be used on your system. For even more details about your CPU(s), you can examine the `/proc/cpuinfo` file, especially the "flags" that are listed which determine whether or not your CPU has certain features.

Use the `head` command with the `-n` option to list the first 20 lines of the `cpuinfo` file:

```
head -n 20 /proc/cpuinfo
```

```
sysadmin@localhost:~$ head -n 20 /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 25
model           : 1
model name      : AMD EPYC 7443P 24-Core Processor
```

```
stepping        : 1

microcode       : 0xa001173

cpu MHz         : 2844.656

cache size      : 512 KB
```

12.2.3 Step 3

To discover how much RAM and swap space is being used, use the `free` command:

```
free -m
free -g
```

The output shows the amount of memory in megabytes when the `-m` option is used and in gigabytes when the `-g` option is used:

```
sysadmin@localhost:~$ free -m
              total        used        free      shared  buff/cache
available
Mem:          64298       12937        5837          26       45523
50616
Swap:          8191           0        8191
sysadmin@localhost:~$ free -g
              total        used        free      shared  buff/cache
available
Mem:             62          12           5           0          44
49
Swap:             7           0           7
```

In the output above, you can see that the system has `64298` megabytes (roughly 64 gigabytes) of physical memory (RAM). Of that, only `12937` megabytes are being used, a good sign that you have enough memory for your system's needs.

In the event that you run out of memory, swap is used. Swap is hard drive space that is used to temporarily store data that is supposed to be stored in RAM.

12.2.4 Step 4

To see what devices are connected to the PCI bus, use the `lspci` command:

```
lspci
```

Notice from the partial output below, that many of the devices connected to the system board are displayed:

```
sysadmin@localhost:~$ lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX
Host bridge (
```

```
rev 01)

00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX
AGP bridge (re

v 01)

00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev
08)

00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (
rev 01)

00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08
)

00:07.7 System peripheral: VMware Virtual Machine Communication In
terface (rev 1

0)

00:0f.0 VGA compatible controller: VMware SVGA II Adapter

00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030
PCI-X Fusion-

MPT Dual Ultra320 SCSI (rev 01)

00:11.0 PCI bridge: VMware PCI bridge (rev 02)

00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)

00:15.1 PCI bridge: VMware PCI Express Root Port (rev 01)
```

```
...
```

The output of the `lspci` command can be helpful for identifying devices that are not
supported by the Linux kernel. Some devices such as video cards may only provide basic
functionality without installing proprietary driver software. However, new distributions are
rapidly addressing this problem and advanced hardware functionality is more commonly
available today.

12.2.5 Step 5

Use the `lspci` command with the `-k` option to show devices along with the kernel driver
and modules used:

```
sysadmin@localhost:~$ lspci -k
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Ho
st bridge (

rev 01)

        Subsystem: VMware Virtual Machine Chipset

        Kernel driver in use: agpgart-intel

lspci: Unable to load libkmod resources: error -12

00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP
bridge (re
```

```
v 01)

00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08
)

        Subsystem: VMware Virtual Machine Chipset

00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev
01)

        Subsystem: VMware Virtual Machine Chipset

        Kernel driver in use: ata_piix

00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)

        Subsystem: VMware Virtual Machine Chipset

00:07.7 System peripheral: VMware Virtual Machine Communication Inter
face (rev 1

0)

        Subsystem: VMware Virtual Machine Communication Interface

        Kernel driver in use: vmw_vmci

00:0f.0 VGA compatible controller: VMware SVGA II Adapter

        Subsystem: VMware SVGA II Adapter

        Kernel driver in use: vmwgfx

...
```

2.2.6 Step 6

Attempt to list the USB connected devices:

```
lsusb
```

```
sysadmin@localhost:~$ lsusb

Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 001 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub

Bus 001 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse

Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub

sysadmin@localhost:~$
```

This is how you can get information on any USB memory sticks that you may want to mount, as well as peripheral devices like mice and keyboards.

12.2.7 Step 7

For hardware to function, the Linux kernel usually loads a driver or module. Use the `lsmod` command to view the currently loaded modules:

```
lsmod
```

Partial output of the command is shown below. The first column is the module name, and the second is the amount of memory used by the module. The number in the "`Used by`" column indicates how many other modules are using the module. The names of the other modules using the module may also be listed in the "`Used by`" column, but is often incomplete:

```
sysadmin@localhost:~$ lsmod
Module                    Size  Used by
isofs                    40960  0
dccp_diag                16384  0
dccp                     73728  1 dccp_diag
tcp_diag                 16384  0
udp_diag                 16384  0
inet_diag                20480  3 tcp_diag,dccp_diag,udp_diag
unix_diag                16384  0
xt_nat                   16384  1558
xt_tcpudp                16384  1641
veth                     16384  0
vxlan                    49152  0
ip6_udp_tunnel           16384  1 vxlan
udp_tunnel               16384  1 vxlan
ipt_MASQUERADE           16384  46
nf_nat_masquerade_ipv4    16384  1 ipt_MASQUERADE
xfrm_user                32768  1
...
```

12.2.8 Step 8

The system board of many computers contains what is known as BIOS, or a Basic Input and Output System. System Management BIOS, or SMBIOS, is a standard defining the data structures and how to communicate information about computer hardware.

The `fdisk` command is useful for identifying and manipulating disk storage resources on a system. Since it can be used to create, format and delete partitions, as well as for getting information, it should be used with care in administrator mode to avoid data loss. The `fdisk` command can be used in two ways: interactively and non-interactively.

When the `-l` option is used with `fdisk`, then the command will non-interactively list block devices, which includes disks (hard drives) and logical volumes.

Without the `-l` option, the `fdisk` command enters an interactive mode that is typically used to modify partitions on a disk device.

12.2.9 Step 9

Execute the `fdisk` command to list the disk devices. The `-l` option lists the partition tables for the specified devices and then exits. If no devices are given, those mentioned in `/proc/partitions` (if that file exists) are used:

```
fdisk -l
```

```
sysadmin@localhost:~$ fdisk -l
Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000571a2

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    39845887    19921920   83  Linux
/dev/sda2           39847934    41940991     1046529    5  Extended
/dev/sda5           39847936    41940991     1046528   82  Linux swap /
Solaris
```

Chapter 13: where data is stored

13.1 Introduction

> **Reminder**
>
> When most people refer to Linux, they are really referring to a combination of software called **GNU/Linux**, which defines the operating system. GNU is the free software that surrounds the kernel and provides open source equivalents of many common UNIX commands. The Linux part of this combination is the Linux kernel, which is the core of the operating system. The kernel is loaded at boot time and stays running to manage every aspect of the functioning system.

An implementation of the Linux kernel includes many subsystems that are a part of the kernel itself and others that may be loaded in a modular fashion when needed. Key functions of the Linux kernel include a system call interface, process management, memory management, virtual filesystems, networking, and device drivers.

In short, via a shell, the kernel accepts commands from the user and manages the processes that carry out those commands by giving them access to devices such as memory, disks, network interfaces, keyboards, mice, monitors and more.

A typical Linux system has thousands of files. The **Filesystem Hierarchy Standard** provides a guideline for distributions on how to organize these files. It is important to understand the role of the Linux kernel and how it both processes and provides information about the system under the `/proc` and `/sys` pseudo filesystems.

13.2 Processes

The kernel provides access to information about active processes through a pseudo filesystem that is visible under the /proc directory. Hardware devices are made available through special files under the /dev directory, while information about those devices can be found in another pseudo filesystem under the /sys directory.

Pseudo filesystems appear to be real files on disk but exist only in memory. Most pseudo file systems such as /proc are designed to appear to be a hierarchical tree off the root of the system of directories, files and subdirectories, but in reality only exist in the system's memory, and only appear to be resident on the storage device that the root file system is on.

The /proc directory not only contains information about running processes, as its name would suggest, but it also contains information about the system hardware and the current kernel configuration.

The /proc directory is read, and its information utilized by many different commands on the system, including but not limited to top, free, mount, umount and many many others. It is rarely necessary for a user to mine the /proc directory directly—it's easier to use the commands that utilize its information.

See an example output below:

```
sysadmin@localhost:~$ ls /proc
1            cpuinfo       irq           modules        sys
128          crypto        kallsyms      mounts         sysrq-trigger
17           devices       kcore         mtrr           sysvipc
21           diskstats     key-users     net            thread-self
23           dma           keys          pagetypeinfo   timer_list
39           driver        kmsg          partitions     timer_stats
60           execdomains   kpagecgroup   sched_debug    tty
72           fb            kpagecount    schedstat      uptime
acpi         filesystems   kpageflags    scsi           version
buddyinfo    fs            loadavg       self           version_signature
bus          interrupts    locks         slabinfo       vmallocinfo
cgroups      iomem         mdstat        softirqs       vmstat
cmdline      ioports       meminfo       stat           zoneinfo
consoles     ipmi          misc          swaps
```

The output shows a variety of named and numbered directories. There is a numbered directory for each running process on the system, where the name of the directory matches the process ID (PID) for the running process.

For example, the numerals `72` denote PID 72, a running program, which is represented by a directory of the same name, containing many files and subdirectories that describe that running process, it's configuration, use of memory, and many other items.

On a running Linux system, there is always a process ID or PID 1.

There are also a number of regular files in the `/proc` directory that provide information about the running kernel:

File	Contents
`/proc/cmdline`	Information that was passed to the kernel when it was first started, such as command line parameters and special instructions
`/proc/meminfo`	Information about the use of memory by the kernel
`/proc/modules`	A list of modules currently loaded into the kernel to add extra functionality

Again, there is rarely a need to view these files directly, as other commands offer more user-friendly output and an alternative way to view this information.

Consider This

While most of the "files" underneath the `/proc` directory cannot be modified, even by the root user, the "files" underneath the `/proc/sys` directory are potentially meant to be changed by the root user. Modifying these files changes the behavior of the Linux kernel.

Direct modification of these files causes only temporary changes to the kernel. To make changes permanent (persistent even after rebooting), entries can be added to the appropriate section of the `/etc/sysctl.conf` file.

For example, the `/proc/sys/net/ipv4` directory contains a file named `icmp_echo_ignore_all`. If that file contains a zero `0` character, as it normally does, then the system will respond to `icmp` requests. If that file contains a one `1` character, then the system will not respond to `icmp` requests:

```
sysadmin@localhost:~$ su -
Password:
root@localhost:~# cat /proc/sys/net/ipv4/icmp_echo_ignore_all
0
root@localhost:~# ping -c1 localhost
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=1 ttl=64 ti
me=0.026 ms


--- localhost.localdomain ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.026/0.026/0.026/0.000 ms

root@localhost:~# echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all

root@localhost:~# ping -c1 localhost

PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.


--- localhost.localdomain ping statistics ---
```

13.2.1 Process Hierarchy

When the kernel finishes loading during the boot procedure, it starts the init process and assigns it a PID of 1. This process then starts other system processes, and each process is assigned a PID in sequential order.

Consider This

On a **System V**-based system, the init process would be the `/sbin/init` program. On a **systemd**-based system, the `/bin/systemd` file is typically executed but is almost always a link to the `/lib/system/systemd` executable.

Regardless of which type of system init process that is being run, the information about the process can be found in the `/proc/1` directory.

As either of the init processes starts up other processes, they, in turn, may start up processes, which may start up other processes, on and on. When one process starts another process, the process that performs the starting is called the parent process and the process that is started is called the child process. When viewing processes, the parent PID is labeled PPID.

When the system has been running for a long time, it may eventually reach the maximum PID value, which can be viewed and configured through the `/proc/sys/kernel/pid_max` file. Once the largest PID has been used, the system "rolls over" and continues seamlessly by assigning PID values that are available at the bottom of the range.

Processes can be "mapped" into a family tree of parent and child couplings. If you want to view this tree, the command `pstree` displays it:

```
sysadmin@localhost:~$ pstree
init-+-cron


    |-login---bash---pstree

    |-named---18*[{named}]

    |-rsyslogd---2*[{rsyslogd}]

    `-sshd
```

If you were to examine the parent and child processes relationship using the output of the previous command, it could be described as the following:

- init is the parent of login
- login is the child of init

- login is the parent of bash
- bash is the child of login

- bash is the parent of pstree
- pstree is the child of bash

13.2.2 Viewing Process Snapshot

Another way of viewing processes is with the ps command. By default, the ps command only shows the current processes running in the current shell. Ironically, even though you are trying to obtain information about processes, the ps command includes itself in the output:

```
root@localhost:~# ps
    PID TTY          TIME CMD
      1 pts/0    00:00:00 init
      7 pts/0    00:00:00 login
     61 pts/0    00:00:00 su
     62 pts/0    00:00:00 bash
     87 pts/0    00:00:00 ps
root@localhost:~# pstree
init-+-cron
     |-login---bash---su---bash---pstree
     |-named---3*[{named}]
     |-rsyslogd---2*[{rsyslogd}]
     `-sshd
root@localhost:~#
```

If you run ps with the option --forest, then, similar to the pstree command, it shows lines indicating the parent and child relationship:

```
root@localhost:~# ps --forest
    PID TTY          TIME CMD
     61 pts/0    00:00:00 su
     62 pts/0    00:00:00  \_ bash
     89 pts/0    00:00:00      \_ ps
      1 pts/0    00:00:00 init
      7 pts/0    00:00:00 login
root@localhost:~#
```

To be able to view all processes on the system execute either the ps aux command or the ps -ef command:

```
        7 pts/0    00:00:00 login
root@localhost:~# ps aux
USER          PID %CPU %MEM     VSZ    RSS TTY       STAT START   TIME COMMAND
root            1  0.0  0.0    4388    864 pts/0     Ss   08:42   0:00 /sbin/init
root            7  0.0  0.0   78644   3772 pts/0     S    08:42   0:00 /bin/login -f
syslog         10  0.0  0.0  191336   3836 ?         Ssl  08:42   0:00 /usr/sbin/rsy
root           14  0.0  0.0   28368   2700 ?         Ss   08:43   0:00 /usr/sbin/cro
root           16  0.0  0.0   72312   3372 ?         Ss   08:43   0:00 /usr/sbin/ssh
bind           26  0.0  0.0  217060  18452 ?         Ssl  08:43   0:00 /usr/sbin/nam
sysadmin       47  0.0  0.0   19228   4228 pts/0     S    08:43   0:00 -bash
root           61  0.0  0.0   60092   3344 pts/0     S    09:00   0:00 su -
root           62  0.0  0.0   19232   4228 pts/0     S    09:00   0:00 -su
root           90  0.0  0.0   34416   3044 pts/0     R+   09:26   0:00 ps aux
root@localhost:~#
```

A common way to reduce the number of lines of output that the user might have to sort through is to use the `grep` command to filter the output display lines that match a keyword, such as a process name. For example, to only view information about the `firefox` process, execute a command like:

```
sysadmin@localhost:~$ ps -ef | grep firefox
 6090 pts/0    00:00:07 firefox
```

Sending the output to a pager such as the `less` command can also make the output of the `ps` command more manageable.

An administrator may be more concerned about the processes of another user. There are several styles of options that the `ps` command supports, resulting in different ways to view an individual user's processes. To use the traditional UNIX option to view the processes of a specific user, use the `-u` option:

```
sysadmin@localhost:~$ ps -u root
  PID TTY          TIME CMD
    1 ?        00:00:00 init
   13 ?        00:00:00 cron
   15 ?        00:00:00 sshd
   43 ?        00:00:00 login
```

13.2.3 Viewing Processes in Real Time

Whereas the `ps` command provides a snapshot of the processes running at the instant the command is executed, the `top` command has a dynamic, screen-based interface that regularly updates the output of running processes. The `top` command is executed as follows:

```
sysadmin@localhost:~$ top
```

By default, the output of the `top` command is sorted by the percentage % of CPU time that each process is currently using, with the higher values listed first, meaning more CPU-intensive processes are listed first:

```
top - 09:36:32 up 19 days, 17:57,  1 user,  load average: 0.16, 0.30, 0.34
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.1 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65841564 total, 33116080 free,  8871844 used, 23853640 buff/cache
KiB Swap:  8388604 total,  8388604 free,        0 used. 56201120 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0    4388    864    804 S   0.0  0.0   0:00.03 init
    7 root      20   0   78644   3772   3224 S   0.0  0.0   0:00.01 login
   10 syslog    20   0  191336   3836   3336 S   0.0  0.0   0:00.39 rsyslogd
   14 root      20   0   28368   2700   2436 S   0.0  0.0   0:00.00 cron
   16 root      20   0   72312   3372   2628 S   0.0  0.0   0:00.00 sshd
   26 bind      20   0  217060  18452   7308 S   0.0  0.0   0:00.03 named
   47 sysadmin  20   0   19228   4228   3024 S   0.0  0.0   0:00.03 bash
   61 root      20   0   60092   3344   2896 S   0.0  0.0   0:00.01 su
   62 root      20   0   19232   4228   3024 S   0.0  0.0   0:00.07 bash
   98 root      20   0   38716   3284   2832 R   0.0  0.0   0:00.02 top
```

There is an extensive amount of interactive commands that can be executed from within the running `top` program. Use the **H** key to view a full list.

```
Help for Interactive Commands - procps-ng 3.3.12
Window 1:Def: Cumulative mode Off.  System: Delay 3.0 secs; Secure mode Off.

  Z,B,E,e   Global: 'Z' colors; 'B' bold; 'E'/'e' summary/task memory scale
  l,t,m     Toggle Summary: 'l' load avg; 't' task/cpu stats; 'm' memory info
  0,1,2,3,I Toggle: '0' zeros; '1/2/3' cpus or numa node views; 'I' Irix mode
  f,F,X     Fields: 'f'/'F' add/remove/order/sort; 'X' increase fixed-width

  L,&,<,> . Locate: 'L'/'&' find/again; Move sort column: '<'/'>' left/right
  R,H,V,J . Toggle: 'R' Sort; 'H' Threads; 'V' Forest view; 'J' Num justify
  c,i,S,j . Toggle: 'c' Cmd name/line; 'i' Idle; 'S' Time; 'j' Str justify
  x,y     . Toggle highlights: 'x' sort field; 'y' running tasks
  z,b     . Toggle: 'z' color/mono; 'b' bold/reverse (only if 'x' or 'y')
  u,U,o,O . Filter by: 'u'/'U' effective/any user; 'o'/'O' other criteria
  n,#,^O  . Set: 'n'/'#' max tasks displayed; Show: Ctrl+'O' other filter(s)
  C,...   . Toggle scroll coordinates msg for: up,down,left,right,home,end

  k,r       Manipulate tasks: 'k' kill; 'r' renice
  d or s    Set update interval
  W,Y       Write configuration file 'W'; Inspect other output 'Y'
  q         Quit
            ( commands shown with '.' require a visible task display window )
Press 'h' or '?' for help with Windows,
Type 'q' or <Esc> to continue
```

One of the advantages of the `top` command is that it can be left running to stay on top of processes for monitoring purposes. If a process begins to dominate, or run away with the system, then by default it will appear at the top of the list presented by the `top` command. An administrator that is running the top command can then take one of two actions:

Key	Action
K	Terminate the runaway process.

Key	Action
R	Adjust the priority of the process.

Pressing the **K** key while the `top` command is running will prompt the user to provide the PID and then a signal number. Sending the default signal requests the process terminate, but sending signal number `9`, the `KILL` signal, forces the process to terminate.

Pressing the **R** key while the `top` command is running will prompt the user for the process to renice, and then for a niceness value. Niceness values can range from -20 to 19, and affect priority. Only the root user can use a niceness value that is a lower number than the current one, or a negative niceness value, which causes the process to run with an increased priority. Any user can provide a niceness value that is higher than the current niceness value, which causes the process to run with a lowered priority.

Another advantage of the `top` command is that it can provide an overall representation of how busy the system is currently and the trend over time.

```
top - 00:26:56 up 28 days, 20:53,  1 user,  load average: 0.11, 0.15,
0.17
Tasks:   8 total,   1 running,   7 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 s
i,  0.0 st
KiB Mem : 13201464+total, 76979904 free, 47522152 used,  7512580 buff
/cache
KiB Swap: 13419622+total, 13415368+free,    42544 used. 83867456 avai
l Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ C
OMMAND
     1 root      20   0   18376   3036   2764 S   0.0  0.0   0:00.12 i
nit
     9 syslog    20   0  191328   5648   3100 S   0.0  0.0   0:00.04 r
syslogd
    13 root      20   0   28356   2552   2320 S   0.0  0.0   0:00.00 c
ron
    15 root      20   0   72296   3228   2484 S   0.0  0.0   0:00.00 s
shd
    24 bind      20   0  878888  39324   7148 S   0.0  0.0   0:02.72 n
amed
    43 root      20   0   78636   3612   3060 S   0.0  0.0   0:00.00 l
ogin
    56 sysadmin  20   0   18508   3440   3040 S   0.0  0.0   0:00.00 b
ash
    72 sysadmin  20   0   36600   3132   2696 R   0.0  0.0   0:00.03 t
op
```

The load averages shown in the first line of output from the `top` command indicate how busy the system has been during the last one, five and fifteen minutes. This information can also be viewed by executing the `uptime` command or directly by displaying the contents of the `/proc/loadavg` file:

Note: To exit the `top` program and return to the prompt, type **q**.

```
sysadmin@localhost:~$ cat /proc/loadavg
0.12 0.46 0.25 1/254 3052
```

- **Load Average:**

```
0.12  0.46  0.25  1/254 3052
```

The first three numbers in this file indicate the load average over the last one, five and fifteen minute intervals.

- **Number of Processes:**

```
0.12 0.46 0.25  1/254  3052
```

The fourth value is a fraction which shows the number of processes currently executing code on the CPU `1` and the total number of processes `254`.

- **Last PID:**

```
0.12 0.46 0.25 1/254  3052
```

The fifth value is the last PID value that executed code on the CPU.

The number reported as a load average is proportional to the number of CPU cores that are able to execute processes. On a single-core CPU, a value of one (`1`) would mean that the system is fully-loaded. On a four core CPU, a value of `1` would mean that the system is only 1/4 or 25% loaded.

Another reason administrators like to keep the `top` command running is the ability to monitor memory usage in real-time. Both the `top` and the `free` command display statistics for how overall memory is being used.

The `top` command can also show the percentage of memory used by each process, so a process that is consuming an inordinate amount of memory can quickly be identified

13.3 Memory

Memory on a modern Linux system is governed and managed by the kernel. The hardware memory on the system is shared by all the processes on the system, through a method called virtual addressing. The physical memory can be referenced by a number of processes, any of which may think they are able to address more memory than they actually can. Virtual addressing allows many processes to access the same memory without conflicts or crashes. It does this by allocating certain areas of a physical (or virtual) hard disk to be used in place of physical RAM. Memory is divided into blocks of equally sized units that can be addressed like any other resource on the system. Not only can the system access memory from local system addresses, but it can also access memory that is located elsewhere, such as on a different computer, a virtual device, or even on a volume that is physically located on another continent!

While a detailed review of Linux memory addressing is beyond the scope of this course, it's important to note the difference between user space and kernel space. Kernel space is where code for the kernel is stored and executed. This is generally in a "protected" range of memory addresses and remains isolated from other processes with lower privileges. User space, on the other hand, is available to users and programs. They communicate with the Kernel through "system call" APIs that act as intermediaries between regular programs and the Kernel. This system of separating potentially unstable or malicious programs from the critical work of the Kernel is what gives Linux systems the stability and resilience that application developers rely on.

13.3.1 Viewing Memory

Executing the `free` command without any options provides a snapshot of the memory being used at that moment.

If you want to monitor memory usage over time with the `free` command, then you can execute it with the `-s` option (how often to update) and specify that number of seconds. For example, executing the following `free` command would update the output every ten seconds:

```
root@localhost:~# free
              total        used        free      shared  buff/cache   available
Mem:       65841564     8642340    33339224       32652    23860000    56431388
Swap:       8388604           0     8388604
root@localhost:~# free -s 10
              total        used        free      shared  buff/cache   available
Mem:       65841564     8596416    33387036       32172    23858112    56477788
Swap:       8388604           0     8388604

              total        used        free      shared  buff/cache   available
Mem:       65841564     8514460    33469000       32172    23858104    56559740
Swap:       8388604           0     8388604

              total        used        free      shared  buff/cache   available
Mem:       65841564     8515896    33467488       32172    23858180    56558320
Swap:       8388604           0     8388604
```

To make it easier to interpret what the `free` command is outputting, the `-m` or `-g` options can be useful by showing the output in either megabytes or gigabytes, respectively. Without these options, the output is displayed in bytes:

When reading the output of the `free` command:

- **Descriptive Header:**

  -
	total	used	free	shared	buffers
cached					
  - Mem: 32953528   26171772   6781756   0   4136   22660364
  - -/+ buffers/cache: 3507272   29446256
  - Swap: 0   0   0

- **Physical Memory Statistics:**

  -
	total	used	free	shared	buffers
cached					
  - Mem: 32953528   26171772   6781756   0   4136   22660364
  - -/+ buffers/cache: 3507272   29446256

```
Swap:            0           0           0
```

- **Memory Adjustment:**

  The third line represents the amount of physical memory after adjusting those values by not taking into account any memory that is in use by the kernel for buffers and caches. Technically, this "used" memory could be "reclaimed" if needed:

```
              total        used        free      shared     buffers
cached

Mem:       32953528    26171772     6781756           0        4136
22660364

-/+ buffers/cache:       3507272    29446256
Swap:              0           0           0
```

- **Swap Memory:**

  The fourth line of output refers to swap memory, also known as virtual memory. This is space on the hard disk that is used like physical memory when the amount of physical memory becomes low. Effectively, this makes it seem that the system has more memory than it does, but using swap space can also slow down the system:

```
              total        used        free      shared     buffers
cached

Mem:       32953528    26171772     6781756           0        4136
22660364

-/+ buffers/cache:       3507272    29446256

Swap:              0           0           0
```

If the amount of memory and swap that is available becomes very low, then the system will begin to automatically terminate processes, making it critical to monitor the system's memory usage. An administrator that notices the system becoming low on free memory can use `top` or `kill` to terminate the processes of their own choice, rather than letting the system choose.

13.4 Log Files

As the kernel and various processes run on the system, they produce output that describes how they are running. Some of this output is displayed as standard output and error in the terminal window where the process was executed, though some of this data is not sent to the screen. Instead, it is written to various files. This information is called log data or log messages.

Log files are useful for many reasons; they help troubleshoot problems and determine whether or not unauthorized access has been attempted.

Some processes can log their own data to these files, other processes rely on a separate process (a daemon) to handle these log data files.

Note

Syslog is the term that is used almost generically to describe logging in Linux systems as it has been in place quite some time. In some cases, when an author says syslog what they really mean is whatever logging system is currently in use on this distribution. Logging daemons (processes) differ in two main ways in recent distributions. The older method of doing system logging is two daemons (named `syslogd` and `klogd`) working together, but in more recent distributions, a single service named `rsyslogd` combines these two functions and more into a single daemon.

In yet more recent distributions, those based on systemd, the logging daemon is named `journald`, and the logs are designed to allow for mainly text output, but also binary. The standard method for viewing `journald`-based logs is to use the `journalctl` command.

Regardless of what the daemon process being used, the log files themselves are almost always placed into the `/var/log` directory structure. Although some of the file names may vary, here are some of the more common files to be found in this directory:

```
root@localhost:~# cd  /var/log
root@localhost:/var/log# ls
alternatives.log  bootstrap.log  dist-upgrade  faillog  syslog
apt               btmp           dmesg         journal  tallylog
auth.log          cron.log       dpkg.log      lastlog  wtmp
root@localhost:/var/log#
```

File	Contents
boot.log	Messages generated as services are started during the startup of the system.
cron	Messages generated by the `crond` daemon for jobs to be executed on a recurring basis.
dmesg	Messages generated by the kernel during system boot up.
maillog	Messages produced by the `mail` daemon for e-mail messages sent or received.
messages	Messages from the kernel and other processes that don't belong elsewhere. Sometimes named `syslog` instead of `messages` after the daemon that writes this file.
secure	Messages from processes that required authorization or authentication (such as the login process).

File	Contents
`journal`	Messages from the default configuration of the `systemd-journald.service`; can be configured in the `/etc/journald.conf` file amongst other places.
`Xorg.0.log`	Messages from the X Windows (GUI) server.

You can view the contents of various log files using two different methods. First, as with most other files, you can use the `cat` command, or the `less` command to allow for searching, scrolling and other options.

The second method is to use the `journalctl` command on systemd-based systems, mainly because the `/var/log/journal` file now often contains binary information and using the `cat` or `less` commands may produce confusing screen behavior from control codes and binary items in the log files.

Log files are rotated, meaning older log files are renamed and replaced with newer log files. The file names that appear in the table above may have a numeric or date suffix added to them: for example, `secure.0` or `secure-20181103`.

Rotating a log file typically occurs on a regularly-scheduled basis: for example, once a week. When a log file is rotated, the system stops writing to the log file and adds a suffix to it. Then a new file with the original name is created, and the logging process continues using this new file.

With most modern daemons, a date suffix is used. So, at the end of the week ending November 3, 2018, the logging daemon might stop writing to `/var/log/messages` (or `/var/log/journal`), rename that file `/var/log/messages-20181103`, and then begin writing to a new `/var/log/messages` file.

Although most log files contain text as their contents, which can be viewed safely with many tools, other files such as the `/var/log/btmp` and `/var/log/wtmp` files contain binary. By using the `file` command, users can check the file content type before they view it to make sure that it is safe to view. The following `file` command classifies `/var/log/wtmp` as `data`, which usually means the file is binary:

```
sysadmin@localhost:~$ file /var/log/wtmp
/var/log/wtmp: data
```

For the files that contain binary data, there are commands available that will read the files, interpret their contents and then output text. For example, the `lastb` and `last` commands can be used to view the `/var/log/btmp` and `/var/log/wtmp` files respectively.

The `lastb` requires root privileges to be executed in our virtual machine environment.

13.5 Kernel Messages

The `/var/log/dmesg` file contains the kernel messages that were produced during system startup. The `/var/log/messages` file contains kernel messages that are produced as the system is running, but those messages are mixed in with other messages from daemons or processes.

Although the kernel doesn't have its own log file normally, one can be configured for it by modifying either the `/etc/syslog.conf` file or the `/etc/rsyslog.conf` file. In addition, the `dmesg` command can be used to view the kernel ring buffer, which holds a large number of messages that are generated by the kernel.

On an active system, or one experiencing many kernel errors, the capacity of this buffer may be exceeded, and some messages might be lost. The size of this buffer is set at the time the kernel is compiled, so it is not trivial to change.

Executing the `dmesg` command can produce up to 512 kilobytes of text, so filtering the command with a pipe to another command like `less` or `grep` is recommended. For example, if a user were troubleshooting problems with a USB device, then searching for the text `USB` with the `grep` command is helpful. The `-i` option is used to ignore case:

```
sysadmin@localhost:~$ dmesg | grep -i usb
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
ohci_hcd 0000:00:06.0: new USB bus registered, assigned bus number 1
usb usb1: New USB device found, idVendor=1d6b, idProduct=0001
usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
```

13.6 Filesystem Hierarchy Standard

Among the standards supported by the Linux Foundation is the **Filesystem Hierarchy Standard (FHS)**, which is hosted at the URL http://www.pathname.com/fhs/.

A standard is a set of rules or guidelines that it is recommended to follow. However, these guidelines certainly can be broken, either by entire distributions or by administrators on individual machines.

The FHS standard categorizes each system directory in a couple of ways:

- A directory can be categorized as either shareable or not, referring to whether the directory can be shared on a network and used by multiple machines.
- The directory is put into a category of having either static files (file contents won't change) or variable files (file contents can change).

To make these classifications, it is often necessary to refer to subdirectories below the top level of directories. For example, the `/var` directory itself cannot be categorized as either shareable or not shareable, but one of its subdirectories, the `/var/mail` directory, is shareable. Conversely, the `/var/lock` directory should not be shareable.

Not Shareable	Shareable

Variable	/var/lock	/var/mail
Static	/etc	/opt
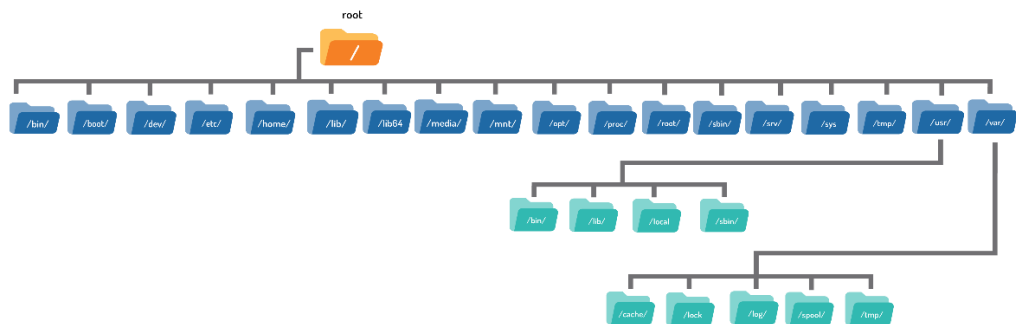
The FHS standard defines four hierarchies of directories used in organizing the files of the filesystem. The top-level or root hierarchy follows:

Directory	Contents
/	The base of the structure, or root of the filesystem, this directory unifies all directories regardless of whether they are local partitions, removable devices or network shares
/bin	Essential binaries like the `ls`, `cp`, and `rm` commands, and be a part of the root filesystem
/boot	Files necessary to boot the system, such as the Linux kernel and associated configuration files
/dev	Files that represent hardware devices and other special files, such as the /dev/null and /dev/zero files
/etc	Essential host configurations files such as the /etc/hosts or /etc/passwd files
/home	User home directories

Directory	Contents
/lib	Essential libraries to support the executable files in the /bin and /sbin directories
/lib64	Essential libraries built for a specific architecture. For example, the /lib64 directory for 64-bit AMD/Intel x86 compatible processors
/media	Mount point for removable media mounted automatically
/mnt	Mount point for temporarily mounting filesystems manually
/opt	Optional third-party software installation location
/proc	Virtual filesystem for the kernel to report process information, as well as other information
/root	Home directory of the root user
/sbin	Essential system binaries primarily used by the root user
/sys	Virtual filesystem for information about hardware devices connected to the system
/srv	Location where site-specific services may be hosted
/tmp	Directory where all users are allowed to create temporary files and that is supposed to be cleared at boot time (but often is not)
/usr	Second hierarchy Non-essential files for multi-user use
/usr/local	Third hierarchy Files for software not originating from distribution
/var	Fourth hierarchy Files that change over time

Directory	Contents
/var/cache	Files used for caching application data
/var/log	Most log files
/var/lock	Lock files for shared resources
/var/spool	Spool files for printing and mail
/var/tmp	Temporary files to be preserved between reboots

The second and third hierarchies, located under the /usr and /usr/local directories, repeat the pattern of many of the key directories found under the first hierarchy or root filesystem. The fourth hierarchy, the /var directory, also repeats some of the top-level directories such as lib, opt and tmp.

While the root filesystem and its contents are considered essential or necessary to boot the system, the /var, /usr and /usr/local directories are deemed non-essential for the boot process. As a result, the root filesystem and its directories may be the only ones available in certain situations like booting into single-user mode, an environment designed for troubleshooting the system.

The /usr directory is intended to hold software for use by multiple users.
The /usr directory is sometimes shared over the network and mounted as read-only.

The /usr/local hierarchy is for installation of software that does not originate with the distribution. Often this directory is used for software that is compiled from the source code.

13.6.1 Organization Within the Filesystem Hierarchy

Although the FHS standard is helpful for a detailed understanding of the layout of the directories used by most Linux distributions, the following provides a more generalized description of the layout of directories as they exist on a typical Linux distribution.

User Home Directories

The /home directory has a directory underneath it for each user account. For example, a user bob will have a home directory of /home/bob. Typically, only the user bob will have access to this directory. Without being assigned special permissions on other directories, a user can only create files in their home directory, the /tmp directory, and the /var/tmp directory.

Binary Directories

Binary directories contain the programs that users and administrators execute to start processes or applications running on the system.

User-Specific Binaries

The binary directories that are intended to be used by non-privileged users include:

- `/bin`
- `/usr/bin`
- `/usr/local/bin`

Sometimes third-party software also store their executable files in directories such as:

- `/usr/local/application/bin`
- `/opt/application/bin`

In addition, it is not unusual for each user to have their own `bin` directory located in their home directory; for example, `/home/bob/bin`.

Root-Restricted Binaries

On the other hand, the `sbin` directories are primarily intended to be used by the system administrator (the root user). These usually include:

- `/sbin`
- `/usr/sbin`
- `/usr/local/sbin`

Some third-party administrative applications could also use directories such as:

- `/usr/local/application/sbin`
- `/opt/application/sbin`

Depending on the distribution, the `PATH` variable may not contain all of the possible `bin` and `sbin` directories. To execute a command in one of these directories, the directory needs to be included in the `PATH` variable list, or the user needs to specify the path to the command.

Software Application Directories

Unlike the Windows operating system, where applications may have all of their files installed in a single subdirectory under the `C:\Program Files` directory, applications in Linux may have their files in multiple directories spread out throughout the Linux filesystem. For Debian-derived distributions, you can execute the `dpkg -L packagename` command to get the list of file locations. In Red Hat-derived distributions, you can run the `rpm -ql packagename` command for the list of the locations of the files that belong to that application.

The executable program binary files may go in the `/usr/bin` directory if they are included with the operating system, or else they may go into the `/usr/local/bin` or `/opt/application/bin` directories if they came from a third party.

The data for the application may be stored in one of the following subdirectories:

- `/usr/share`
- `/usr/lib`

- `/opt/application`
- `/var/lib`

The file related to documentation may be stored in one of the following subdirectories:

- `/usr/share/doc`
- `/usr/share/man`
- `/usr/share/info`

The global configuration files for an application are most likely to be stored in a subdirectory under the `/etc` directory, while the personalized configuration files (specific for a user) for the application are probably in a hidden subdirectory of the user's home directory.

Library Directories

Libraries are files which contain code that is shared between multiple programs. Most library file names end in a file extension of `.so`, which means shared object.

Multiple versions of a library may be present because the code may be different within each file even though it may perform similar functions as other versions of the library. One of the reasons that the code may be different, even though it may do the same thing as another library file, is that it is compiled to run on a different kind of processor. For example, it is typical for systems that use code designed for 64-bit Intel/AMD type processors to have both 32-bit libraries and 64-bit libraries.

The libraries that support the essential binary programs found in the `/bin` and `/sbin` directories are typically located in either `/lib` or `/lib64`.

To support the `/usr/bin` and `/usr/sbin` executables, the `/usr/lib` and `/usr/lib64` library directories are typically used.

For supporting applications that are not distributed with the operating system, the `/usr/local/lib` and `/opt/application/lib` library directories are frequently used.

Variable Data Directories

The `/var` directory and many of its subdirectories can contain data that changes frequently. If your system is used for email, then either `/var/mail` or `/var/spool/mail` is normally used to store users' email data. If you are printing from your system, then the `/var/spool/cups` directory is used to store the print jobs temporarily.

Depending on what events are being logged and how much activity is occurring, the system determines how large your log file becomes. On a busy system, there could be a considerable amount of data in the log files. These files are stored in the `/var/log` directory.

While the log files can be handy for troubleshooting problems, they can cause problems. One major concern for all of these directories is that they can fill up the disk space quickly on an active system. If the `/var` directory is not a separate partition, then the root filesystem could become full and cause the system to crash.

Chapter 14: Network Configuration

14.1 Introduction

Having access to the network is a key feature of most Linux systems. Users want to surf the net, send and receive email and transfer files with other users.

Typically the programs that perform these functions, such as web browsers and email clients, are reasonably easy to use. However, they all rely on an important feature: the ability of your computer to communicate with another computer. To have this communication, you need to know how to configure your system's network.

Linux provides you with several tools to both configure your network as well as monitor how it is performing.

4.2 Basic Network Terminology

Before setting up a network or accessing an existing network, it is beneficial to know some key terms that are related to networking. This section explores the terms with which you should be familiar. Some of the terms are basic, and you may already be familiar with them. However, others are more advanced.

Host	A host is a computer. Many people automatically think of a desktop computer or laptop when they hear the term computer. In reality, many other devices, such as cell phones, digital music players and many modern televisions, are also computers. In networking terms, a host is any device that communicates via a network with another device.
Network	A network is a collection of two or more hosts (computers) that are able to communicate with each other. This communication can be via a wired connection or wireless.
Internet	The Internet is an example of a network. It consists of a publicly accessible network that connects millions of hosts throughout the world. Many people use the Internet to surf web pages and exchange emails, but the Internet has many additional capabilities besides these activities.
Wi-Fi	The term Wi-Fi refers to wireless networks.
Server	A host that provides a service to another host or client is called a server. For example, a web server stores, processes and delivers web pages. An email server receives incoming mail and delivers outgoing mail.

Service	A feature provided by a host is a service. An example of a service would be when a host provides web pages to another host.
Client	A client is a host that is accessing a server. When you are working on a computer surfing the Internet, you are considered to be on a client host.
Router	Also called a gateway, a router is a machine that connects hosts from one network to another network. For example, if you work in an office environment, the computers within the company can all communicate via the local network created by the administrators. To access the Internet, the computers would have to communicate with a router that would be used to forward network communications to the Internet. Typically when you communicate on a large network (like the Internet), several routers are used before your communication reaches its final destination.

14.3 Networking Features Terminology

In addition to the networking terms discussed in the last section, there are some additional terms with which you should be familiar. These terms focus more on the different types of networking services that are commonly used, as well as some of the techniques that are used to communicate between machines.

Packet	A network packet is used to send network communication between hosts. By breaking down communication into smaller chunks (packets), the data delivery method is much more efficient.
IP Address	An Internet Protocol (IP) address is a unique number assigned to a host on a network. Hosts use these numbers to address network communication.
Mask	Also called a netmask, subnet mask or mask, a network mask is a number system that can be used to define which IP addresses are considered to be within a single network. Because of how routers perform their functions, networks have to be clearly defined.
Hostname	Each host on a network could have its own hostname because names are more natural for humans to remember than numbers, making it easier for us to address network packets to another host. Hostnames are translated into IP addresses before the network packet is sent on the network.
URL	A Uniform Resource Locator (URL), also commonly called a web address, is used to locate a resource, like a web page, on the internet. It's what you type into your web browser to access a web page. For example, http://www.netdevgroup.com. It includes the protocol http:// and the hostname www.netdevgroup.com.

DHCP	Hosts can be assigned hostnames, IP addresses and other network-related information by a DHCP (Dynamic Host Configuration Protocol) server. In the world of computers, a protocol is a well-defined set of rules. DHCP defines how network information is assigned to client hosts, and the DHCP server is the machine that provides this information.
DNS	As mentioned previously, hostnames are translated into IP addresses, prior to the network packet being sent on the network. So your host needs to know the IP address of all of the other hosts with which you are communicating. When working on a large network (like the Internet), this can pose a challenge as there are so many hosts. A Domain Name System (DNS) provides the service of translating domain names into IP addresses.
Ethernet	In a wired network environment, Ethernet is the most common way to physically connect the hosts into a network. Ethernet cables are connected to network cards that support Ethernet connections. Ethernet cables and devices (such as routers) are specifically designed to support different communication speeds, the lowest being 10 Mbps (10 Megabits per second) and the highest being 100 Gbps (100 gigabits per second). The most common speeds are 100 Mbps and 1 Gbps.
TCP/IP	The Transmission Control Protocol/Internet Protocol (TCP/IP) is a fancy name for a collection of protocols (remember, protocol = set of rules) that are used to define how network communication should take place between hosts. While it isn't the only collection of protocols used to define network communication, it is the most often utilized one. As an example, TCP/IP includes the definition of how IP addresses and network masks work.

14.4 IP Addresses

As previously mentioned, hosts address network packets by using the IP address of the destination machine. The network packet also includes a return address, which is the IP address of the sending machine.

There are, in fact, two different types of IP addresses: **IPv4** and **IPv6**. To understand why there are two different types, you need to understand a brief bit of IP addressing history.

For many years, the IP addressing technique that was used by all computers was IPv4. In an IPv4 address, a total of four 8-bit numbers are used to define the address. This is considered a 32-bit address (4 x 8 = 32). For example:

```
192.168.10.120
```

8-bit refers to numbers from 0 to 255.

Each host on the Internet must have a unique IP address. In an IPv4 environment, there is a technical limit of about 4.3 billion IP addresses. However, many of these IP addresses are not usable for various reasons. Also, many organizations haven't made use of all of the IP addresses they have available.

While it seems like there should be plenty of IP addresses to go around, various factors have led to a problem: the Internet started running out of IP addresses.

This issue encouraged the development of IPv6. IPv6 was officially created in 1998. In an IPv6 network the addresses are much larger, 128-bit addresses that look like this:

```
2001:0db8:85a3:0042:1000:8a2e:0370:7334
```

So, why hasn't the world embraced the superior technology of IPv6?

There are primarily two reasons:

- **NAT:** Invented to overcome the possibility of running out of IP addresses in an IPv4 environment, Net Address Translation (NAT) used a technique to provide more hosts access to the Internet. In a nutshell, a group of hosts is placed into a private network with no direct access to the Internet; a special router provides Internet access, and only this one router needs an IP address to communicate on the Internet. In other words, a group of hosts shares a single IP address, meaning a lot more computers can attach to the Internet. This feature means the need to move to IPv6 is less critical than before the invention of NAT.

- **Porting:** Porting is switching over from one technology to another. IPv6 has a lot of great new features, but all of the hosts need to be able to utilize these features. Getting everyone on the Internet (or even just some) to make these changes poses a challenge.

14.5 Configuring Network Devices

When you are configuring network devices, there are two initial questions that you need to ask:

- Wired or wireless? Configuring a wireless device is slightly different to configuring a wired device because of some of the additional features typically found on wireless devices (such as security).

- DHCP or static address? Recall that a DHCP server provides network information, such as your IP address and subnet mask. If you don't make use of a DHCP server, then you will need to manually provide this information to your host, which is called using a static IP address

14.5.1 Configuring the Network Using Configuration Files

There will be times when no GUI-based tool is available. In those cases, it is helpful to know the configuration files that are used to store and modify network data.

These files may vary depending on the Linux distribution that you are working on.

14.5.1.1 Primary IPv4 Configuration File

On a CentOS system, the primary configuration file for an IPv4 network interface is the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. The VM in this chapter is Debian-based, and so does not have the `sysconfig` folder. However, for demonstration purposes only, the following shows what this file looks like when configured for a static IP address

```
root@localhost:~# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
BOOTPROTO=none
NM_CONTROLLED="yes"
ONBOOT=yes
TYPE="Ethernet"
UUID="98cf38bf-d91c-49b3-bb1b-f48ae7f2d3b5"
DEFROUTE=yes
IPV4 _FAILURE_FATAL=yes
IPV6INOT=no
NAME="System eth0"
IPADDR=192.168.1.1
PREFIX=24
GATEWAY=192.168.1.1
DNS1=192.168.1.2
HWADDR=00:50:56:90:18:18
LAST_CONNECT=1376319928
```

If the device were configured to be a DHCP client, the `BOOTPROTO` value would be set to `dhcp`, and the `IPADDR`, `GATEWAY` and `DNS1` values would not be set.

14.5.1.2 Primary IPv6 Configuration File

On a CentOS system, the primary IPv6 configuration file is the same file where IPv4 configuration is stored; the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. If you want to have your system have a static IPv6 address, add the following to the configuration file:

```
IPV6INIT=yes
IPV6ADDR=<IPv6 IP Address>
IPV6_DEFAULTGW=<IPv6 IP Gateway Address>
```

If you want your system to be a DHCP IPv6 client, then add the following setting:

```
DHCPV6C=yes
```

You also need to add the following setting to the `/etc/sysconfig/network` file:

```
NETWORKING_IPV6=yes
```

Consider This

The widely accepted method of making changes to a network interface is to take the interface down using a command such as `ifdown eth0`, make the desired changes to the configuration file, and then bring the interface back up and into service with a command such as `ifup eth0`.

Another less specific method is to restart the system's networking entirely, with a command such as `service network restart`, which takes down ALL interfaces, re-reads all related configuration files, and then restarts the networking for the system.

Restarting the network service can disrupt much more than just the single interface a user wanted to change, so use the most limited and specific commands to restart the interface if possible.

The following example demonstrates how the `service` command would need to be executed on a CentOS system:

```
[root@localhost ~]# service network restart

Shutting down interface eth0:  Device state: 3 (disconnected)

                                                                [  OK
]

Shutting down loopback interface:                               [  OK
]

Bringing up loopback interface:                                 [  OK
]

Bringing up interface eth0:  Active connection state: activated

Active connection path: /org/freedesktop/NetworkManager/ActiveConn
ection/1

                                                                [  OK
]
```

14.5.1.3 Domain Name System (DNS)

When a computer is asked to access a website, such as www.example.com, it does not necessarily know what IP address to use. For the computer to associate an IP address with the URL or hostname request, the computer relies upon the DNS service of another computer. Often, the IP address of the DNS server is discovered during the DHCP request, while a computer is receiving important addressing information to communicate on the network.

The address of the DNS server is stored in the `/etc/resolv.conf` file. A typical `/etc/resolv.conf` file is automatically generated and looks like the following:

```
sysadmin@localhost:~$ cat /etc/resolv.conf
nameserver 127.0.0.1
```

The `nameserver` setting is often set to the IP address of the DNS server. The following example uses the `host` command, which works with DNS to associate a hostname with an IP address. Note that the example server is associated with the IP address `192.168.1.2` by the DNS server:

```
sysadmin@localhost:~$ host example.com
example.com has address 192.168.1.2
```

It is also common to have multiple `nameserver` settings, in the event that one DNS server isn't responding.

14.5.1.4 Network Configuration Files

Name resolution on a Linux host is accomplished by 3 critical files: the `/etc/hosts`, `/etc/resolv.conf` and `/etc/nsswitch.conf` files. Together, they describe the location of name service information, the order in which to check resources, and where to go for that information.

Files	Explanation
`/etc/hosts`	This file contains a table of hostnames to IP addresses. It can be used to supplement a DNS server. ```sysadmin@localhost:~$ cat /etc/hosts` `127.0.0.1 localhost```
`/etc/resolv.conf`	This file contains the IP addresses of the name servers the system should consult in any attempt to resolve names to IP addresses. These servers are often DNS servers. It also can contain additional keywords and values that can affect the resolution process. ```sysadmin@localhost:~$ cat /etc/resolv.conf` `nameserver 127.0.0.11```

Files	Explanation
/etc/nsswitch.conf	This file can be used to modify where hostname lookups occur. It contains a particular entry that describes in what order name resolution sources are consulted.

```
sysadmin@localhost:~$ cat /etc/nsswitch.conf

# /etc/nsswitch.conf

#


Output Omitted...


hosts:          files dns


Output Omitted...
```

The /etc/hosts file is searched first, the DNS server second:

```
hosts: files dns
```

The DNS server would be searched first, local files second:

```
hosts: dns files
```

Commands or programs on the system, such as the browser, request a connection with a remote computer by DNS name. Then the system consults various files in a particular order to attempt to resolve that name into a usable IP address.

1. First, the /etc/nsswitch.conf file is consulted:

```
hosts:          files dns
```

This line indicates that the system should consult local files first in an attempt to resolve hostnames, which means that the /etc/hosts file will be parsed for a match to the requested name.

2. Second, the system will consult the /etc/hosts file to attempt to resolve the name. If the name matches an entry in /etc/hosts, it is resolved.

It will not failover (or continue) to the DNS option, even if the resolution is inaccurate. This can occur if the entry in /etc/hosts points to a non-assigned IP address.

3. Third, if the local /etc/hosts file doesn't result in a match, the system will use the configured DNS server entries contained in the /etc/resolv.conf file to attempt to resolve the name.

The /etc/resolv.conf file should contain at least two entries for name servers, such as the example file below:

```
nameserver 10.0.2.3
nameserver 10.0.2.4
```

The DNS resolution system will use the first name server for an attempted lookup of the name. If that is unavailable, or a timeout period is reached, the second server will then be queried for the name resolution. If a match is found, it is returned to the system and used for initiating a connection and is also placed in the DNS cache for a configurable time period.

14.6 Network Tools

There are several commands that you can use to view network information. These tools can also be useful when you are troubleshooting network issues.

14.6.1 The ifconfig Command

The `ifconfig` command stands for interface configuration and is used to display network configuration information

14.6.2 The ip Command

The `ifconfig` command is becoming obsolete in some Linux distributions (deprecated) and is being replaced with a form of the `ip` command, specifically `ip addr show`.

The `ip` command differs from `ifconfig` in several important manners, chiefly that through its increased functionality and set of options, it can almost be a one-stop shop for configuration and control of a system's networking. The format for the `ip` command is as follows:

```
ip [OPTIONS] OBJECT COMMAND
```

While `ifconfig` is limited primarily to modification of networking parameters, and displaying the configuration details of networking components, the `ip` command branches out to do some of the work of several other legacy commands such as `route` and `arp`.

Note

Linux and Unix commands don't usually just disappear when they become obsolete; they stick around as a legacy command, sometimes for many years, as the number of scripts that depend on those commands, and the amount of muscle memory amongst system administrators, makes it a good idea to keep them around for compatibility sake.

The `ip` command can initially appear to be a little more verbose than the `ifconfig` command, but it's a matter of phrasing and a result of the philosophy behind the operation of the `ip` command.

In the example below, both the `ifconfig` command and `ip` command are used to show all interfaces on the system.

```
root@localhost:~# ifconfig
eth0      Link encap:Ethernet   HWaddr 00:0c:29:71:f0:bb
          inet addr:172.16.241.140  Bcast:172.16.241.255  Mask:255.25
5.255.0
```

```
        inet6 addr: fe80::20c:29ff:fe71:f0bb/64 Scope:Link

        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

        RX packets:8506 errors:0 dropped:0 overruns:0 frame:0

        TX packets:1201 errors:0 dropped:0 overruns:0 carrier:0

        collisions:0 txqueuelen:1000

        RX bytes:8933700 (8.9 MB)  TX bytes:117237 (117.2 KB)


lo        Link encap:Local Loopback

        inet addr:127.0.0.1  Mask:255.0.0.0

        inet6 addr: ::1/128 Scope:Host

        UP LOOPBACK RUNNING  MTU:65536  Metric:1

        RX packets:285 errors:0 dropped:0 overruns:0 frame:0

        TX packets:285 errors:0 dropped:0 overruns:0 carrier:0

        collisions:0 txqueuelen:1

        RX bytes:21413 (21.4 KB)  TX bytes:21413 (21.4 KB)
root@localhost:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN g
roup default qlen 1

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

    inet 127.0.0.1/8 scope host lo

        valid_lft forever preferred_lft forever

    inet6 ::1/128 scope host

        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000

    link/ether 00:0c:29:71:f0:bb brd ff:ff:ff:ff:ff:ff

    inet 172.16.241.140/24 brd 172.16.241.255 scope global eth0

        valid_lft forever preferred_lft forever

    inet6 fe80::20c:29ff:fe71:f0bb/64 scope link

        valid_lft forever preferred_lft forever
```

Both show the type of interface, protocols, hardware and IP addresses, network masks and various other information about each of the active interfaces on the system.

14.6.3 The route Command

Recall that a router (or gateway) is a machine that allows hosts from one network to communicate with another network. To view a table that describes where network packages are sent, use the `route` command:

```
root@localhost:~# route

Kernel IP routing table

Destination     Gateway        Genmask         Flags Metric Ref    U
se Iface
192.168.1.0     *              255.255.255.0   U     0      0
0 eth0
default         192.168.1.1    0.0.0.0         UG    0      0
0 eth0
```

The first highlighted line in the preceding example indicates that any network packet sent to a machine in the 192.168.0 network is not sent to a gateway machine (the * indicates no gateway). The second highlighted line indicates that all other network packets are sent to the host with the IP address of 192.168.1.1 (the router).

Some users prefer to display this information with numeric data only, by using the -n option to the route command. For example, compare the following and focus on where the previous output displayed the word default:

```
root@localhost:~# route -n

Kernel IP routing table

Destination     Gateway        Genmask         Flags Metric Ref    U
se Iface
192.168.1.0     0.0.0.0        255.255.255.0   U     0      0
0 eth0

0.0.0.0         192.168.1.1    0.0.0.0         UG    0      0
0 eth0
```

The 0.0.0.0 refers to all other machines, and is the same as default.

The route command is becoming obsolete in some Linux distributions (deprecated) and is being replaced with a form of the ip command, specifically ip route or ip route show. Note that the same information highlighted above can also be found using this command:

```
root@localhost:~# ip route show

default via 192.168.1.254 dev eth0 proto static

192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.2
```

14.6.4 The ping Command

The ping command can be used to determine if another machine is reachable. If the ping command can send a network package to another machine and receive a response, then you should be able to connect to that machine.

```
root@localhost:~# ping -c 4 192.168.1.2

PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.

64 bytes from 192.168.1.2: icmp_req=1 ttl=64 time=0.051 ms

64 bytes from 192.168.1.2: icmp_req=2 ttl=64 time=0.064 ms

64 bytes from 192.168.1.2: icmp_req=3 ttl=64 time=0.050 ms
```

```
64 bytes from 192.168.1.2: icmp_req=4 ttl=64 time=0.043 ms


--- 192.168.1.2 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 2999ms

rtt min/avg/max/mdev = 0.043/0.052/0.064/0.007 ms
```

If the `ping` command fails, a message stating, `Destination Host Unreachable` displays:

```
root@localhost:~# ping -c 4 192.168.1.1

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

From 192.168.1.2 icmp_seq=1 Destination Host Unreachable

From 192.168.1.2 icmp_seq=2 Destination Host Unreachable

From 192.168.1.2 icmp_seq=3 Destination Host Unreachable

From 192.168.1.2 icmp_seq=4 Destination Host Unreachable


--- 192.168.1.1 ping statistics ---

4 packets transmitted, 0 received, +4 errors, 100% packet loss, time
2999ms

pipe 4
```

It is important to note that just because the `ping` command fails does not mean that the remote system is unreachable. Some administrators configure their machines (and even entire networks!) to not respond to `ping` requests because a server can be attacked by something called a denial of service attack. In this sort of attack, a server is overwhelmed by a massive number of network packets. By ignoring `ping` requests, the server is less vulnerable.

As a result, the `ping` command may be useful for checking the availability of local machines, but not always for machines outside of your own network.

Consider This

Many administrators use the `ping` command with a hostname, and if that fails then use the IP address to see if the fault is in resolving the device's hostname. Using the hostname first saves time; if that `ping` command is successful, there is proper name resolution, and the IP address is functioning correctly as well.

14.6.5 The netstat Command

The `netstat` command is a powerful tool that provides a large amount of network information. It can be used to display information about network connections as well as display the routing table similar to the `route` command.

For example, to display statistics regarding network traffic, use the `-i` option to the `netstat` command:

```
root@localhost:~# netstat -i

Kernel Interface table
```

```
Iface    MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR  TX-DRP T
X-OVR Flg
eth0        1500 0        137        0        4 0        12        0        0
0 BMRU
lo         65536 0         18        0        0 0        18        0        0
0 LRU
```

The most important statistics from the output above are the TX-OK and TX-ERR. A high percentage of TX-ERR may indicate a problem on the network, such as too much network traffic.

To use the netstat command to display routing information, use the -r option:

```
root@localhost:~# netstat -r

Kernel IP routing table

Destination     Gateway          Genmask          Flags   MSS Window  i
rtt Iface

192.168.1.0     *                255.255.255.0    U         0 0
0 eth0

default         192.168.1.1      0.0.0.0          UG        0 0
0 eth0
```

The netstat command is also commonly used to display open ports. A port is a unique number that is associated with a service provided by a host. If the port is open, then the service is available for other hosts.

For example, you can log into a host from another host using a service called SSH. The SSH service is assigned port #22. So, if port #22 is open, then the service is available to other hosts.

It is important to note that the host also needs to have the services running itself; this means that the service (in this case the ssh daemon) that allows remote users to log in needs to be started (which it typically is, for most Linux distributions).

To see a list of all currently open ports, use the following command:

```
root@localhost:~# netstat -tln

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address            Foreign Address        S
tate

tcp       0        0 192.168.1.2:53           0.0.0.0:*              L
ISTEN

tcp       0        0 127.0.0.1:53             0.0.0.0:*              L
ISTEN

tcp       0        0 0.0.0.0:22               0.0.0.0:*              L
ISTEN

tcp       0        0 127.0.0.1:953            0.0.0.0:*              L
ISTEN

tcp6      0        0 :::53                    :::*                   L
ISTEN

tcp6      0        0 :::22                    :::*                   LI
STEN
```

```
tcp6       0       0 ::1:953                    :::*                          L
ISTEN
```

As you can see from the output above, port #22 is listening, which means it is open.

In the previous example, `-t` stands for TCP (recall this protocol from earlier in this chapter), `-l` stands for listening (which ports are listening) and `-n` stands for show numbers, not names.

On some distributions you may see the following message in the man page of the `netstat` command:

```
NOTE
    This program is obsolete. Replacement for netstat is ss. Replace
ment for

    netstat -r is ip route. Replacement for netstat -i is ip -s link
.

    Replacement for netstat -g is ip maddr.
```

While no further development is being done on the `netstat` command, it is still an excellent tool for displaying network information. The goal is to eventually replace the `netstat` command with commands such as the `ss` and `ip` commands. However, it is important to realize that this may take some time

14.6.6 The ss Command

The `ss` command is designed to show socket statistics and supports all the major packet and socket types. Meant to be a replacement for and to be similar in function to the `netstat` command, it also shows a lot more information and has more features.

The main reason a user would use the `ss` command is to view what connections are currently established between their local machine and remote machines, statistics about those connections, etc.

Similar to the `netstat` command, you can get a great deal of useful information from the `ss` command just by itself as shown in the example below.

```
root@localhost:~# ss

Netid  State      Recv-Q Send-Q       Local Address:Port
Peer Address:Port

u_str  ESTAB      0      0                        * 104741
* 104740

u_str  ESTAB      0      0       /var/run/dbus/system_bus_socket 14623
* 14606

u_str  ESTAB      0      0       /var/run/dbus/system_bus_socket 13582
* 13581

u_str  ESTAB      0      0       /var/run/dbus/system_bus_socket 16243
* 16242

u_str  ESTAB      0      0                        * 16009
* 16010

u_str  ESTAB      0      0       /var/run/dbus/system_bus_socket 10910
* 10909
```

```
u_str  ESTAB       0        0            @/tmp/dbus-LoJW0hGFkV 15706
* 15705

u_str  ESTAB       0        0                        * 24997
        * 24998

u_str  ESTAB       0        0                        * 16242
* 16243

u_str  ESTAB       0        0            @/tmp/dbus-opsTQoGE 15471
* 15470
```

The output is very similar to the output of the `netstat` command with no options. The columns above are:

`Netid`	The socket type and transport protocol
`State`	Connected or Unconnected, depending on protocol
`Recv-Q`	Amount of data queued up for being processed having been received
`Send-Q`	Amount of data queued up for being sent to another host
`Local Address`	The address and port of the local host's portion of the connection
`Peer Address`	The address and port of the remote host's portion of the connection

The format of the output of the `ss` command can change dramatically, given the options specified, such as the use of the `-s` option, which displays mostly the types of sockets, statistics about their existence and numbers of actual packets sent and received via each socket type, as shown below:

```
root@localhost:~# ss -s
Total: 1000 (kernel 0)
TCP:   7 (estab 0, closed 0, orphaned 0, synrecv 0, timewait 0/0), po
rts 0


Transport Total     IP        IPv6
*             0     -         -
RAW           0     0         0
UDP           9     6         3
TCP           7     3         4
INET          16    9         7
FRAG          0     0         0
```

> **Consider This**
>
> The `ss` command typically shows many rows of data, and it can be somewhat daunting to try to find what you want in all that output. Consider sending the output to the `less` command to make the output more manageable. Pagers allow the user to scroll up and down, do searches and many other useful functions inside the parameters of the `less` command.

14.6.7 The dig Command

There may be times when you need to test the functionality of the DNS server that your host is using. One way of doing this is to use the `dig` command, which performs queries on the DNS server to determine if the information needed is available on the server.

In the following example, the `dig` command is used to determine the IP address of the example.com host:

```
root@localhost:~# dig example.com

; <<>> DiG 9.8.1-P1 <<>> example.com

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45155

;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0


;; QUESTION SECTION:
;example.com.                    IN      A


;; ANSWER SECTION:
example.com.             86400   IN      A       192.168.1.2
;; AUTHORITY SECTION:
example.com.             86400   IN      NS      example.com.

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Dec  8 17:54:41 2015
;; MSG SIZE  rcvd: 59
```

Note that the response included the IP address of `192.168.1.2`, meaning that the DNS server has the IP address to hostname translation information in its database.

If the DNS server doesn't have the requested information, it is configured to ask other DNS servers. If none of them have the requested information, an error message displays:

```
root@localhost:~# dig sample.com

; <<>> DiG 9.8.1-P1 <<>> sample.com

;; global options: +cmd

;; connection timed out; no servers could be reached
```

14.6.8 The host Command

In its simplest form, the `host` command works with DNS to associate a hostname with an IP address. As used in a previous example, example.com is associated with the IP address of `192.168.1.2`:

```
root@localhost:~# host example.com

example.com has address 192.168.1.2
```

The `host` command can also be used in reverse if an IP address is known, but the domain name is not.

```
root@localhost:~# host 192.168.1.2

2.1.168.192.in-addr.arpa domain name pointer example.com.

2.1.168.192.in-addr.arpa domain name pointer cserver.example.com.
```

Other options exist to query the various aspects of a DNS such as a `CNAME` canonical name -alias:

```
root@localhost:~# host -t CNAME example.com

example.com has no CNAME record
```

Since many DNS servers store a copy of example.com, `SOA` Start of Authority records indicate the primary server for the domain:

```
root@localhost:~# host -t SOA example.com

example.com has SOA record example.com. cserver.example.com. 2 604800
86400 2419200 604800
```

A comprehensive list of DNS information regarding example.com can be found using the `-a` all option:

```
root@localhost:~# host -a example.com
Trying "example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3549
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL:
1


;; QUESTION SECTION:
;example.com.                        IN      ANY


;; ANSWER SECTION:
example.com.            86400   IN      SOA     example.com. cserver.
example.com. 2 604800 86400 2419200 604800

example.com.            86400   IN      NS      example.com.
```

```
example.com.                86400   IN     A       192.168.1.2


;; ADDITIONAL SECTION:

example.com.                86400   IN     A       192.168.1.2


Received 119 bytes from 127.0.0.1#53 in 0 ms
```

14.6.9 The ssh Command

The `ssh` command allows you to connect to another machine across the network, log in and then perform tasks on the remote machine.

If you only provide a machine name or IP address to log into, the `ssh` command assumes you want to log in using the same username that you are currently logged in as. To use a different username, use the syntax:

```
username@hostname
```

```
root@localhost:~# ssh bob@test
The authenticity of host 'test (127.0.0.1)' can't be established.
RSA key fingerprint is c2:0d:ff:27:4c:f8:69:a9:c6:3e:13:da:2f:47:e4:c
9.
Are you sure you want to continue connection (yes/no)? yes
Warning: Permanently added 'test' (RSA) to the list of known hosts.
bob@test's password:
bob@test:~$ date
Fri Oct   4 16:14:43 CDT 2013
```

To return back to the local machine, use the `exit` command:

```
bob@test:~$ exit
logout
Connection to test closed.
root@localhost:~#
```

Warning

Be careful, if you use the `exit` command too many times, you will close the terminal window that you are working in!

14.6.9.1 RSA Key Fingerprint

When using the `ssh` command, the first prompt asks you to verify the identity of the machine you are logging into. In most cases, you are going to want to answer `yes`. While you can check with the administrator of the remote machine to make sure that the RSA key fingerprint is correct, this isn't the purpose of this query. It is designed for future login attempts.

After you answer `yes`, the RSA key fingerprint of the remote machine is stored on your local system. When you attempt to `ssh` to this same machine in the future, the RSA key fingerprint provided by the remote machine is compared to the copy stored on the local machine. If they match, then the username prompt appears. If they don't match, an error like the following displays:

```
sysadmin@localhost:~$ ssh bob@test

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!

Someone could be eavesdropping on you right now (man-in-the-middle at
tack)!

It is also possible that the RSA host key has just been changed.

The fingerprint for the RSA key sent by the remote host is

c2:0d:ff:27:4c:f8:69:a9:c6:3e:13:da:2f:47:e4:c9.

Please contact your system administrator.

Add correct host key in /home/sysadmin/.ssh/known_hosts to get rid of
this message.

Offending key in /home/sysadmin/.ssh/known_hosts:1

RSA host key for test has changed and you have requested strict check
ing.

Host key verification failed.
```

This error could indicate that a rogue host has replaced the correct host. Check with the administrator of the remote system. If the system were recently reinstalled, it would have a new RSA key, and that would be causing this error.

In the event that this error message is due to a remote machine reinstall, you can remove the `~/.ssh/known_hosts` file from your local system (or just remove the entry for that one machine) and try to connect again:

```
sysadmin@localhost:~$ cat ~/.ssh/known_hosts

test ssh-rsa AAAAB3NzaC1yc2EAAAAmIwAAAQEAklOUpkDHrfHY17SbrmTIp/RZ0V4D
Txgq9wzd+ohy006SWDSGPA+nafzlHDPOW7vdI4mZ5ew18KL4JW9jbhUFrviQzM7xlELEV
f4h9lFX5QVkbPppSrg0cda3Pbv7kOdJ/MTyBlWXFCRH+Cv3FXRitBqxiX1nKhXpHAZsMc
iLq8V6RjsNAQwdsdMFvSlVK/7BA

t5FaiKoAfncM1Q8x3+2V0Ww71/eIFmb1zuUFljHYTprrX88XypNDvjYNby6vw/Pb0rwpr
z/Tn

mZAW3UX+PnTPI89ZPmNBLuxyrD2cE86Z/il8b+gw3r3+1nJotmIkjn2so1d01QraTlMqV
Ssbx

NrRFi9wrf+ghw==

sysadmin@localhost:~$ rm ~/.ssh/known_hosts

sysadmin@localhost:~$ ssh bob@test

The authenticity of host 'test (127.0.0.1)' can't be established.
```

```
RSA key fingerprint is c2:0d:ff:27:4c:f8:69:a9:c6:3e:13:da:2f:47:e4:c
9.

Are you sure you want to continue connection (yes/no)? yes

Warning: Permanently added 'test' (RSA) to the list of known hosts.

bob@test's password:

Last login: Fri Oct   4 16:14:39 CDT 2013  from localhost
```

Chapter 15: system and user security

15.1 Introduction

User accounts are designed to provide security on a Linux operating system. Each person on the system must log in using a user account which either allows the person to access specific files and directories or disallows such access, accomplished using file permissions, which are file and directory permissions given by the system to users, groups and everyone else who logs in. These permissions can be edited by the root user.

User accounts also belong to groups, which can also be used to provide access to files/directories. Each user belongs to at least one group (often many) to allow users to more easily share data that is stored in files with other users.

User and group account data is stored in database files. Knowing the content of these files allows you to better understand which users have access to files and directories on the system. These database files also contain vital security information that may affect the ability of a user to log in and access the system.

Several commands provide the ability to see user and group account information, as well as to switch from one user account to another (provided you have the appropriate authority to do so). These commands are valuable for investigating usage of the system, troubleshooting system problems and for monitoring unauthorized access to the system.

15.2 Administrative Accounts

There are many different ways to execute a command that requires administrative or root privileges. Logging in to the system as the root user allows you to execute commands as the administrator. This access is potentially dangerous because you may forget that you are logged in as root and might run a command that could cause problems on the system. As a result, it is not recommended to log in as the root user directly.

Because using the root account is potentially dangerous, you should only execute commands as root if administrative privileges are needed. If the root account is disabled, as it is on the Ubuntu distribution, then administrative commands can be executed using

the `sudo` command. If the root account is enabled, then a regular user can execute the `su` command to switch accounts to the root account.

When you log in to the system directly as root to execute commands, then everything about your session runs as the root user. If using the graphical environment, this is especially dangerous as the graphical login process is comprised of many different executables (programs that run during login). Each program that runs as the root user represents a greater threat than a process run as a standard user, as those programs would be allowed to do nearly anything, whereas standard user programs are very restricted in what they can do.

The other potential danger with logging into the system as root is that a person that does this may forget to log out to do their non-administrative work, allowing programs such as browsers and email clients to be run as the root user without restrictions on what they could do. The fact that several distributions of Linux, notably Ubuntu, do not allow users to log in as the root user should be enough indication that this is not the preferred way to perform administrative tasks.

15.2.1 Switching Users

The `su` command allows you to run a shell as a different user. While switching to the `root` user is what the `su` command is used for most frequently, it can also switch to other users as well.

```
su [options] [username]
```

When switching users utilizing the login shell option is recommended, as the login shell fully configures the new shell with the settings of the new user, ensuring any commands executed run correctly. If this option is omitted, the new shell changes the UID but doesn't fully log in the user. The login shell option can be specified one of three ways:

```
su -

su -l

su --login
```

By default, if a username is not specified, the `su` command opens a new shell as the `root` user. The following two commands are equivalent ways to start a shell as the `root` user:

```
su - root

su -
```

After pressing **Enter** to execute either one of these commands, the user must provide the password of the `root` user to start the new shell. If you don't know the password of the account that you are shifting to, then the `su` command will fail.

Notice in the example below, and in our virtual machines, the command prompt changes to reflect the current user.

```
sysadmin@localhost:~$ su -
Password: netlab123
root@localhost:~# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

After using the shell started by the `su` command to perform the necessary administrative tasks, return to your original shell (and original user account) by using the `exit` command.

```
root@localhost:~# exit

logout

sysadmin@localhost:~$ id

uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27
(sudo)
```

15.2.2 Executing Privileged Commands

The `sudo` command allows users to execute commands as another user. Similar to the `su` command, the `root` user is assumed by default.

```
sudo [options] command
```

In distributions that do not allow the `root` user to login directly or via the `su` command, the installation process automatically configures one user account to be able to use the `sudo` command to execute commands as if the `root` user executed them. For example, administrative privileges are necessary to view the `/etc/shadow` file:

```
sysadmin@localhost:~$ head /etc/shadow

head: cannot open '/etc/shadow' for reading: Permission denied
```

When using the `sudo` command to execute a command as the `root` user, the command prompts for the user's own password, not that of the `root` user. This security feature could prevent unauthorized administrative access if the user were to leave their computer unattended. The prompt for the password will not appear again as long as the user continues to execute `sudo` commands less than five minutes apart.

The following `sudo` command would run the `head` command from the previous example as the `root` user. It prompts for the password of the `sysadmin` user:

```
sysadmin@localhost:~$ sudo head /etc/shadow

[sudo] password for sysadmin: netlab123

root:$6$4Yga95H9$8HbxqsMEIBTZ0YomlMffYCV9VE1SQ4T2H3SHXw41M02SQtfAdDVE
9mqGp2hr20q.ZuncJpLyWkYwQdKlSJyS8.:16464:0:99999:7:::

daemon:*:16463:0:99999:7:::

bin:*:16463:0:99999:7:::

sys:*:16463:0:99999:7:::

sync:*:16463:0:99999:7:::

games:*:16463:0:99999:7:::

man:*:16463:0:99999:7:::

lp:*:16463:0:99999:7:::
```

```
mail:*:16463:0:99999:7:::

news:*:16463:0:99999:7:::
```

Using the `sudo` command to execute an administrative command results in an entry placed in a log file. Each entry includes the name of the user who executed the command, the command that was executed and the date and time of execution. This allows for increased accountability, compared to a system where many users might know the `root` password and can either log in directly as `root` or use the `su` command to execute commands as the `root` user.

One big advantage of using `sudo` to execute administrative commands is that it reduces the risk of a user accidentally executing a command as `root`. The intention to execute a command is clear; the command is executed as `root` if prefixed with the `sudo` command. Otherwise, the command is executed as a regular user.

15.3 User Accounts

There are several text files in the `/etc` directory that contain the account data of the users and groups defined on the system. For example, to see if a specific user account has been defined on the system, then the place to check is the `/etc/passwd` file.

The `/etc/passwd` file defines some of the account information for user accounts. The following example shows the last five lines of a typical `/etc/passwd` file:

```
sysadmin@localhost:~$ tail -5 /etc/passwd

syslog:x:101:103::/home/syslog:/bin/false

bind:x:102:105::/var/cache/bind:/bin/false

sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin

operator:x:1000:37::/root:/bin/sh

sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bas
h
```

Each line contains information pertaining to a single user. The data is separated into fields by colon characters. The following describes each of the fields in detail, from left to right, using the last line of the output of the previous graphic:

- **Name**

  ```
  sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/
  bin/bash
  ```

  The first field contains the name of the user or the username. This name is used when logging in to the system and when file ownership is viewed with the `ls -l` command. It is provided to make it easier for regular users to refer to the account, while the system typically utilizes the user ID internally.

- **Password Placeholder**

  ```
  sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/
  bin/bash
  ```

At one time, the password for the user was stored in this location, however, now the `x` in this field indicates to the system that the password is in the `/etc/shadow` file.

- **User ID**

```
sysadmin:x: 1001 :1001:System Administrator,,,,:/home/sysadmin:/
bin/bash
```

Each account is assigned a user ID (UID). Usernames are not directly used by the system, which typically defines the account by the UID instead. For example, files are owned by UIDs, not by usernames.

- **Primary Group ID**

```
sysadmin:x:1001: 1001 :System Administrator,,,,:/home/sysadmin:/
bin/bash
```

This field indicates that the user is a member of that group, which means the user has special permissions on any file that is owned by this group.

- **Comment**

```
sysadmin:x:1001:1001: System Administrator,,,, :/home/sysadmin:/
bin/bash
```

This field can contain any information about the user, including their real name or other useful information.

- **Home Directory**

```
sysadmin:x:1001:1001:System Administrator,,,,: /home/sysadmin :/
bin/bash
```

This field defines the location of the user's home directory. For regular users, this would typically be `/home/`*`username`*. For example, a username of `bob` would have a home directory of `/home/bob`.

The root user usually has a different place for the home directory, the `/root` directory.

- **Shell**

```
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin: /
bin/bash
```

This field indicates the location of the user's login shell. By default, the user is placed in this shell whenever they log into a command line environment or open a terminal window. The bash shell `/bin/bash` is the most common shell for Linux users.

An efficient way to check if a specific user has been defined on a system is to search the /etc/passwd file using the grep command. For example, to see the account information for the user named `sysadmin`, use the following command:

```
sysadmin@localhost:~$ grep sysadmin /etc/passwd
```

```
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bas
h
```

15.3.1 Passwords

As previously mentioned, the `/etc/shadow` file contains account information related to the user's password. However, regular users can't view the contents of the `/etc/shadow` file for security reasons. To view the contents of this file, log in as the administrator (the root account):

```
sysadmin@localhost:~$ su -

Password: netlab123

root@localhost:~#
```

A typical `/etc/shadow` file would look similar to the following:

```
root@localhost:~# tail -5 /etc/shadow

syslog:*:16874:0:99999:7:::

bind:*:16874:0:99999:7:::

sshd:*:16874:0:99999:7:::

operator:!:16874:0:99999:7:::

sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/
221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050:
```

Again, each line is separated into fields by colon characters. The following describes each of the fields in detail, from left to right, using the last line of the output of the previous graphic:

- **Username**

  ```
  sysadmin :$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc
  1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
  5050::
  ```

  This field contains the username of the account, which matches the account name in the `/etc/passwd` file.

- **Password**

- ```
  sysadmin: $6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc
  1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0 :16874:5:30:7:60:
  15050::
  ```

  The password field contains the encrypted password for the account. This very long string is a one-way encryption, meaning that it can't be "reversed" to determine the original password.

  While regular users have encrypted passwords in this field, system accounts have an asterisk `*` character in this field.

- **Last Change**

- ```
  sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
  aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
  5050::
  ```

This field contains a number that represents the last time the password was changed. The number `16874` is the number of days since January 1, 1970 (called the Epoch).

This value generates automatically when the user's password is modified. It is used by the password aging features provided by the rest of the fields of this file.

- **Minimum**

- ```
  sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
  aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
  5050::
  ```

This field indicates the minimum number of days between password changes. It is one of the password aging fields; a non-zero value in this field indicates that after a user changes their password, the password can't be changed again for the specified number of days, `5` days in this example. This field is important when the maximum field is used.

A value of zero in this field means the user can always change their password.

- **Maximum**

- ```
  sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
  aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
  5050:
  ```

This field indicates the maximum number of days the password is valid. It is used to force users to change their passwords on a regular basis. A value of `30` in this field means the user must change their password at least every 30 days to avoid having their account locked out.

Note that if the minimum field is set to `0`, the user may be able to immediately set their password back to the original value, defeating the purpose of forcing the user to change their password every 30 days. So, if the maximum field is set, the minimum field is ordinarily set as well.

For example, a minimum:maximum of `5:30` means the user must change their password every 30 days and, after changing, the user must wait 5 days before they can change their password again.

If the `max` field is set to `99999`, the maximum possible value, then the user essentially never has to change their password (because 99999 days is approximately 274 years).

- **Warn**

- ```
  sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
  aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
  5050::
  ```

If the maximum field is set, the warn field indicates the number of days before password expiry that the system warns the user. For example, if the warn field is set to 7, then any time during the 7 days before the maximum time frame is reached, the user will be warned to change their password during the login processes.

The user is only warned at login, so some administrators have taken the approach of setting the warn field to a higher value to provide a greater chance of having a warning issued.

If the maximum time frame is set to 99999, then the warn field is basically useless.

- **Inactive**

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
5050::
```

If the user ignores the warnings and exceeds the password timeframe, their account will be locked out. In that case, the inactive field provides the user with a "grace" period in which their password can be changed, but only during the login process.

If the inactive field is set to 60, the user has 60 days to change to a new password. If they fail to do so, then the administrator would be needed to reset the password for the user.

- **Expire**

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:1
5050::
```

This field indicates the day the account will expire, represented by the number of days from January 1, 1970. An expired account is locked, not deleted, meaning the administrator can reset the password to unlock the account.

Accounts with expiration dates are commonly provided to temporary employees or contractors. The account automatically expires after the user's last day of work.

When an administrator sets this field, a tool is used to convert from a real date to an Epoch date. There are also several free converters available on the Internet.

- **Reserved**

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1
aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15
050::
```

Currently not used, this field is reserved for future use.

Consider This

In addition to the `grep` command, another technique for retrieving user information contained in the `/etc/passwd` and `/etc/shadow` files is to use the `getent` command. One advantage of this command is that it can retrieve account information that is defined

locally, in files such as `/etc/passwd` and `/etc/shadow`, or on a network directory server.

The general syntax of a `getent` command is:

```
getent database record
```

For example, the following command would retrieve account information for the `sysadmin` user from the `/etc/passwd` file:

```
sysadmin@localhost:~$ getent passwd sysadmin
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

15.4 System Accounts

Users log into the system using regular user accounts. Typically, these accounts have UID values of greater than 500 (on some systems 1,000). The root user has special access to the system. This access is provided to the account with a UID of 0.

There are additional accounts that are not designed for users to log into. These accounts, typically from UID 1 to UID 499, are called system accounts, and they are designed to provide accounts for services that are running on the system.

System accounts have some fields in the `/etc/passwd` and `/etc/shadow` files that are different than other accounts. For example, system accounts rarely have home directories as they typically are not used to create or store files. In the `/etc/passwd` file, system accounts have a non-login program in the shell field:

```
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
```

In the `/etc/shadow` file, system accounts typically have an asterisk * character in place of the password field:

```
sshd:*:16874:0:99999:7:::
```

Most system accounts are necessary for the system to function correctly. You should not delete a system account unless you are certain that removing the account won't cause problems. Take time to learn what each system account does; system administrators are tasked with ensuring the security of the system, and that includes properly securing the system accounts.

15.5 Group Accounts

Your level of access to a system is not determined solely by your user account. Each user can be a member of one or more groups, which can also affect the level of access to the system.

Traditionally, UNIX systems limited users to belonging to no more than a total of sixteen groups, but the recent Linux kernels support users with over sixty-five thousand group memberships.

The `/etc/passwd` file defines the primary group membership for a user. Supplemental group membership (or secondary group membership) and the groups themselves are defined in the `/etc/group` file.

The `/etc/group` file is another colon-delimited file. The following describes the fields in more detail, using a line that describes a typical group account.

- **Group Name**

  ```
  mail:x:12:mail,postfix
  ```

  This field contains the group name. As with usernames, names are more natural for people to remember than numbers. The system typically uses group IDs rather than group names.

- **Password Placeholder**

  ```
  mail:x:12:mail,postfix
  ```

  While there are passwords for groups, they are rarely used in Linux. If the administrator makes a group password, it would be stored in the `/etc/gshadow` file. The `x` in this field is used to indicate that the password is not stored in this file.

- **GID**

  ```
  mail:x:12:mail,postfix
  ```

  Each group is associated with a unique group ID (GID) which is placed in this field.

- **User List**

  ```
  mail:x:12:mail,postfix
  ```

  This last field is used to indicate who is a member of the group. While primary group membership is defined in the `/etc/passwd` file, users who are assigned to additional groups would have their username placed in this field of the `/etc/group` file. In this case, the `mail` and `postfix` users are secondary members of the `mail` group.

It is very common for a username to also appear as a group name. It is also common for a user to belong to a group with the same name.

To view information about a specific group, either the `grep` or `getent` commands can be used. For example, the following commands display the `mail` group account information:

```
sysadmin@localhost:~$ grep mail /etc/group
mail:x:12:mail,postfix
sysadmin@localhost:~$ getent group mail
mail:x:12:mail,postfix
```

15.6 Viewing User Information

The `id` command is used to print user and group information for a specified user.

```
id [options] username
```

When switching between different user accounts, it can be confusing as to which account is currently logged in. When executed without an argument, the `id` command outputs information about the current user, allowing you to confirm your identity on the system.

```
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27
(sudo)
```

The output of the `id` command always lists the user account information first, using the user ID and username first:

```
uid=1001(sysadmin)  gid=1001(sysadmin) groups=1001(sysadmin),4(adm),2
7(sudo)
```

After the username the primary group is listed, denoted by both the group ID and group name:

```
uid=1001(sysadmin)  gid=1001(sysadmin)  groups=1001(sysadmin),4(adm),2
7(sudo)
```

Other information listed includes the groups the user belongs to, again denoted by group IDs followed by the group names. The user shown belongs to three groups:

```
uid=1001(sysadmin) gid=1001(sysadmin)  groups=1001(sysadmin),4(adm),2
7(sudo)
```

If the command is given a username as an argument, such as `root`, it displays information about the specified account:

```
sysadmin@localhost:~$ id root
uid=0(root) gid=0(root) groups=0(root)
```

To print only the user's primary group, use the `-g` option:

```
sysadmin@localhost:~$ id -g
1001
```

The `id` command can also be used to verify the user's secondary group memberships using the `-G` option. This will print all the groups that a user belongs to, both primary and secondary.

```
sysadmin@localhost:~$ id -G
1001 4 27
```

The output of the previous example aligns with the contents of the `/etc/group` file, as a search for `sysadmin` reveals:

```
sysadmin:x:1001:
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
sysadmin@localhost:~$ id -G
1001 4 27
sysadmin@localhost:~$ id Root
id: 'Root': no such user
sysadmin@localhost:~$ id root
uid=0(root) gid=0(root) groups=0(root)
sysadmin@localhost:~$ id root -G
0
sysadmin@localhost:~$ cat /etc/group | grep sysadmin
adm:x:4:syslog,sysadmin
sudo:x:27:sysadmin
sysadmin:x:1001:
sysadmin@localhost:~$
```

15.7 Viewing Current Users

The `who` command displays a list of users who are currently logged into the system, where they are logged in from, and when they logged in. Through the use of options, this command is also able to display information such as the current runlevel (a functional state of the computer) and the time that the system was booted.

```
sysadmin@localhost:~$ who
sysadmin pts/0        Feb 13 08:21
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ who

root           tty2         2013-10-11 10:00

sysadmin       tty1         2013-10-11 09:58 (:0)

sysadmin       pts/0        2013-10-11 09:59 (:0.0)

sysadmin       pts/1        2013-10-11 10:00 (example.com)
```

The following describes the output of the `who` command:

- **Username**

```
    root                    tty2         2013-10-11 10:00
```

  This column indicates the name of the user who is logged in. Note that by "logged in" we mean "any login process and open terminal window".

- **Terminal**

```
    • root         tty2          2013-10-11 10:00
    • sysadmin     pts/0         2013-10-11 09:59 (:0.0)
```

  This column indicates which terminal window the user is working in.

If the terminal name starts with `tty`, then this is an indication of a local login, as this is a regular command line terminal. If the terminal name starts with `pts`, then this indicates the user is using a pseudo terminal or running a process that acts as a terminal.

- **Date**

  - ```
    root          tty2          2013-10-11 10:00
    ```

  This column indicates when the user logged in.

- **Host**

  After the date and time, some location information may appear. If the location information contains a hostname, domain name or IP address, then the user has logged in remotely:

  ```
  sysadmin pts/1        2013-10-11 10:00  (example.com)
  ```

  If there is a colon and a number, then this indicates that they have performed a local graphical login:

  ```
  sysadmin     tty1        2013-10-11 09:59  (:0)
  ```

  If no location information is shown in the last column, then this means the user logged in via a local command line process:

  ```
  root          tty2          2013-10-11 10:00
  ```

Consider This

The `who` command has several options for displaying system status information. For example, the `-b` option shows the last time the system started (booted), and the `-r` option shows the time the system reached the current runlevel:

```
sysadmin@localhost:~$ who -b -r
        system boot   2013-10-11 09:54
        run-level 5   2013-10-11 09:54
```

There may be instances where more information about users, and what they are doing on the system, is needed. The `w` command provides a more detailed list about the users currently on the system than the `who` command. It also provides a summary of the system status. For example:

```
sysadmin@localhost:~$ w
 10:44:03 up 50 min,  4 users,  load average: 0.78, 0.44, 0.19
USER          TTY     FROM        LOGIN@   IDLE     JCPU      PCPU
WHAT
root          tty2    -           10:00    43:44    0.01s     0.01s
-bash
sysadmin      tty1    :0          09:58    50:02    5.68s     0.16s
pam: gdm-password
```

```
sysadmin      pts/0   :0.0          09:59     0.00s      0.14s
      0.13s   ssh 192.168.1.2

sysadmin      pts/1   example.com 10:00     0.00s      0.03s  0.01s
w
```

The first line of output from the `w` command is identical to that of the `uptime` command. It shows the current time, how long the system has been running, the total number of users currently logged on and the load on the system averaged over the last 1, 5 and 15 minute time periods. Load average is CPU utilization where, for a single-core system, a value of 1 would mean 100% CPU usage during that period of time. For a dual-core system, it would mean 50% CPU usage, and for a quad-core system, it would mean 25% CPU usage.

The following describes the rest of the output of the `w` command:

Column	Example	Description
USER	root	The name of the user who is logged in.
TTY	tty2	Which terminal window the user is working in.
FROM	example.com	Where the user logged in from.
LOGIN@	10:00	When the user logged in.
IDLE	43:44	How long the user has been idle since the last command was executed.
JCPU	0.01s	The total cpu time used by all processes run since login.
PCPU	0.01s	The total cpu time for the current process.
WHAT	-bash	The current process that the user is running.

Note

The `s` character represents seconds.

15.8 Viewing Login History

The `last` command reads the entire login history from the `/var/log/wtmp` file and displays all logins and reboot records by default. An interesting detail of the reboot records is that it displays the version of the Linux kernel that was booted instead of the login location. The `/var/log/wtmp` file keeps a log of all users who have logged in and out the system.

```
sysadmin@localhost:~$ last
sysadmin console Tue Sep 18 02:31   still logged in
sysadmin console                    Tue Sep 18 02:31 - 02:31  (00:00)
wtmp begins Tue Sep 18 02:31:57 2018
```

The `last` command is slightly different from the `who` and `w` commands. By default, it also shows the username, terminal, and login location, not just of the current login sessions, but previous sessions as well. Unlike the `who` and `w` commands, it displays the date and time the user logged into the system. If the user has logged off the system, then it will display the total time the user spent logged in, otherwise it will display `still logged in`.

Consider This

The `who` command reads from the `/var/log/utmp` file which logs current users, while the `last` command reads from the `/var/log/wtmp` file, which keeps a history of all user logins.

Chapter 16: Creating Users and Groups

16.1 Introduction

During the installation process, most installers create a normal user and either give this user the permission to execute administrative commands with `sudo` or require the root user account password be configured as part of the installation process. Most Linux systems are configured to allow for one unprivileged (non-root) user to log in, as well as have the ability to effectively execute commands as the root user, either directly or indirectly.

If the computer is to be used by only one person, then having only one regular user account might be sufficient. However, if a computer needs to be shared by multiple people, then it is desirable to have a separate account for each person who uses it. There are several advantages to individuals having their own separate accounts:

Accounts can be used to grant selective access to files or services. For example, the user of each account has a separate home directory that is generally not accessible to the other users.

The `sudo` command can be configured to grant the ability to execute select administrative commands. If users are required to use the `sudo` command to perform administrative commands, then the system logs when users perform these commands.

Each account can have group memberships and rights associated with it allowing for greater management flexibility.

On some distributions, creating a new user account also automatically creates a group account for the user, called a User Private Group (UPG). On these systems, the group

and username would be the same, and the only member of this new group would be the new user.

For distributions that do not create a UPG, new users are typically given the `users` group as their primary group. The administrator can manually create group accounts that are private for the user, but it's more common for the administrator to create groups for multiple users that need to collaborate. User accounts can be modified at any time to add or remove them from group account memberships, but users must belong to at least one group for use as their primary group.

Before you begin creating users, you should plan how to use groups. Users can be created with memberships in groups that already exist, or existing users can be modified to have memberships in existing groups.

If you already have planned which users and groups you want, it is more efficient to create your groups first and create your users with their group memberships. Otherwise, if you create your users first, and then your groups, you'll need to take an extra step to modify your users to make them members of your groups.

16.2 Groups

The most common reason to create a group is to provide a way for users to share files. For example, if several people who work together on the same project and need to be able to collaborate on documents stored in files for the project. In this scenario, the administrator can make these people members of a common group, change the directory ownership to the new group and set permissions on the directory that allows members of the group to access the files.

After creating or modifying a group, you can verify the changes by viewing the group configuration information in the `/etc/group` file with the `grep` command. If working with network-based authentication services, then the `getent` command can show you both local and network-based groups.

```
grep pattern filename

getent database record
```

For local usage, these commands show the same result, in this case for the `root` group:

```
root@localhost:~# grep root /etc/group

root:x:0:

root@localhost:~# getent group root

root:x:0:
```

16.2.1 Creating A Group

The `groupadd` command can be executed by the root user to create a new group. The command requires only the name of the group to be created. The `-g` option can be used to specify a group id for the new group:

```
root@localhost:~# groupadd -g 1005 research

root@localhost:~# grep research /etc/group

research:x:1005:
```

If the `-g` option is not provided, the `groupadd` command will automatically provide a GID for the new group. To accomplish this, the `groupadd` command looks at the `/etc/group` file and uses a number that is one value higher than the current highest GID number. The execution of the following commands illustrates this:

```
root@localhost:~# groupadd development
root@localhost:~# grep development /etc/group
development:x:1006:
```

16.2.1.1 Group ID Considerations

In some Linux distributions, particularly those based upon Red Hat, when a user ID (UID) is created, a user private group (UPG) is also created with that user as its only member. In these distributions, the UID and the ID of the UPG are supposed to match (be the same number).

Therefore, you should avoid creating GIDs in the same numeric ranges where you expect to create UIDs, to avoid a conflict between a GID you create and a UPG number that is created to match a UID.

GIDs under either 500 (RedHat) or 1000 (Debian) are reserved for system use. There may be times at which you want to assign a lower GID value. To accomplish this, use the `-r` option which assigns the new group a GID that is less than the lowest standard GID:

```
root@localhost:~# groupadd -r sales
root@localhost:~# getent group sales
sales:x:999:
```

16.2.1.2 Group Naming Considerations

Following these guidelines for group names can help to select a group name that is portable (function correctly with other systems or services):

- The first character of the name should be either an underscore _ character or a lowercase alphabetic `a-z` character.
- Up to 32 characters are allowed on most Linux distributions, but using more than 16 can be problematic as some distributions may not accept more than 16.
- After the first character, the remaining characters can be alphanumeric, a dash – character or an underscore _ character.
- The last character should not be a hyphen – character.

Unfortunately, these guidelines are not always enforced. The problem isn't that the `groupadd` command does not necessarily fail, but that other commands or system services may not work correctly.

16.2.2 Modifying a Group

The `groupmod` command can be used to either change the name of a group with the `-n` option or change the GID for the group with the `-g` option.

Changing the name of the group may confuse users who were familiar with the old name and haven't been informed of the new name. However, changing the group name won't cause any problems with accessing files, since the files are owned by GIDs, not group names. For example:

```
root@localhost:~# ls -l index.html
-rw-r-----. 1 root sales 0 Aug  1 13:21 index.html
root@localhost:~# groupmod -n clerks sales
root@localhost:~# ls -l index.html
-rw-r-----. 1 root clerks 0 Aug  1 13:21 index.html
```

Note: The file in the example above is not available within the virtual machine environment of this course.

After the previous `groupmod` command, the `index.html` file has a different group owner name. However, all users who were in the `sales` group are now in the `clerks` group, so all of those users can still access the `index.html` file. Again, this is because the system defines the group by the GID, not the group name.

On the other hand, if you change the GID for a group, then all files that were associated with that group will no longer be associated with that group. In fact, all files that were associated with that group will no longer be associated with any group name. Instead, these files will be owned by a GID only, as shown below:

```
root@localhost:~# groupmod -g 10003 clerks
root@localhost:~# ls -l index.html
-rw-r-----. 1 root 491 13370 Aug  1 13:21 index.html
```

Consider This

These files with no group name are called orphaned files. To search for all files that are owned by just a GID (not associated with a group name) use the `-nogroup` option of the `find` command:

```
root@localhost:~# find / -nogroup
/root/index.html
```

16.2.3 Deleting a Group

If you decide to delete a group with the `groupdel` command, be aware that any files that are owned by that group will become orphaned.

Only supplemental groups can be deleted, so if any group that is the primary group for any user, it cannot be deleted. The administrator can modify which group is a user's primary group, so a group that was being used as a primary group can be made into a supplemental group and then can be deleted.

As long as the group to be deleted is not a user's primary group, deleting the group is accomplished by using the `groupdel` command along with the name of the group:

```
root@localhost:~# groupdel clerks
```

16.3 Users

User account information is stored in the `/etc/passwd` file and user authentication information (password data) is stored in the `/etc/shadow` file. Creating a new user can be accomplished by manually adding a new line to each of these files, but that is generally not the recommended technique.

By using an appropriate command to add a new user, these files can be modified automatically and safely. If you were to modify these files manually, you would risk making a mistake that could prevent all users from being able to log in normally.

Before you begin creating users for your system, you should verify or establish practical values that are used by default with the `useradd` command. These settings can be found in the configuration files that are used by the `useradd` command.

Ensuring that the values in these configuration files are reasonable before adding users can help save you the time and trouble of having to correct user account settings after adding the users

16.3.1 User Configuration Files

The `-D` option to the `useradd` command allows you to view or change some of the default values used by the `useradd` command. The values shown by `useradd -D` can also be viewed or updated by manipulating the `/etc/default/useradd` file:

```
sysadmin@localhost:~$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
sysadmin@localhost:~$
```

The following describes each of these values:

- **Group**

```
GROUP=100
```

  In distributions not using UPG, this is the default primary group for a new user, if one is not specified with the `useradd` command. This is usually the `users` group with a GID of 100.

  This setting affects the primary group ID field of the `/etc/passwd` file highlighted below:

```
bob:x:600: 600 :bob:/home/bob:/bin/bash
```

  The `-g` option to the `useradd` command allows you to use a different primary group than the default when creating a new user account.

- **Home**

```
HOME=/home
```

The `/home` directory is the default base directory under which the user's new home directory is created. This means that a user with an account name of `bob` would have a home directory of `/home/bob`.

This setting affects the home directory field of the `/etc/passwd` file highlighted below:

```
bob:x:600:600:bob: /home/bob :/bin/bash
```

The `-b` option to the `useradd` command allows you to use a different base directory group than the default when creating a new user account.

- **Inactive**

```
INACTIVE=-1
```

This value represents the number of days after the password expires that the account is disabled. A value of `-1` means this feature is not enabled by default and no "inactive" value is provided for new accounts by default.

This setting affects the inactive field of the `/etc/shadow` file highlighted below:

```
bob:pw:15020:5:30:7: 60 :15050:
```

The `-f` option to the `useradd` command allows you to use a different `INACTIVE` value than the default when creating a new user account.

- **Expire**

```
EXPIRE=
```

By default, there is no value set for the expiration date. Usually, an expiration date is set on an individual account, not all accounts by default.

For example, if you had a contractor that was hired to work until the end of the day on November 1, 2019, then you could ensure that they would be unable to log in after that date by using the `EXPIRE` field.

This setting affects the expire field of the `/etc/shadow` file highlighted below:

```
bob:pw:15020:5:30:7:60: 15050 :
```

The `-e` option to the `useradd` command allows you to use a different `EXPIRE` value than the default when creating a new user account.

- **Shell**

```
SHELL=/bin/bash
```

The `SHELL` setting indicates the default shell for a user when they log in to the system.

This setting affects the shell field of the `/etc/passwd` file highlighted below:

```
bob:x:600:600:bob:/home/bob: /bin/bash
```

The `-s` option to the `useradd` command allows you to use a different login shell than the default when creating a new user account.

- **Skeleton Directory**

```
SKEL=/etc/skel
```

The `SKEL` value determines which skeleton directory has its contents copied into the new user's home directory. The contents of this directory are copied into the new user's home directory, and the new user is given ownership of the new files.

This setting provides administrators with an easy way to populate a new user account with key configuration files.

The `-k` option to the `useradd` command allows you to use a different `SKEL` directory than the default when creating a new user account.

- **Create Mail Spool**

```
CREATE_MAIL_SPOOL=yes
```

A mail spool is a file where incoming email is placed.

Currently, the value for creating a mail spool is `yes`, which means that users by default are configured with the ability to receive and store local mail. If you are not planning on using local mail, then this value could be changed to `no`.

To modify one of the `useradd` default values, the `/etc/default/useradd` file could be edited with a text editor. Another (safer) technique is to use the `useradd -D` command.

For example, if you wanted to allow users to have expired passwords that they could still log in with for up to thirty days, then you could execute:

```
root@localhost:~# useradd -D -f 30
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=30
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

16.3.2 User Configuration Files

The `/etc/login.defs` file also contains values that are applied by default to new users you create with the `useradd` command. Unlike the `/etc/default/useradd` file, the `/etc/login.defs` file is usually edited directly by the administrator to alter its values.

This file contains many comments and blank lines, so to only view lines that are not comments or blank lines (the real configuration settings), then you can use the following `grep` command:

```
root@localhost:~#  grep -Ev '^#|^$' /etc/login.defs
MAIL_DIR        /var/mail/spool
PASS_MAX_DAYS  99999
PASS_MIN_DAYS  0
PASS_MIN_LEN   5
PASS_WARN_AGE  7
UID_MIN                   500
UID_MAX                 60000
GID_MIN                   500
GID_MAX                 60000
CREATE_HOME     yes
UMASK           077
USERGROUPS_ENAB yes
ENCRYPT_METHOD SHA512
MD5_CRYPT_ENAB no
```

The above example represents a typical CentOS 6 distribution `/etc/login.defs` file
with its values. The following describes each of these values:

- **Mail Directory**

  ```
  MAIL_DIR            /var/mail/spool
  ```

  The directory in which the user's mail spool file is created.

- **Password Maximum Days**

  ```
  PASS_MAX_DAYS     99999
  ```

  This setting determines the maximum number of days that a user can continue to
  use the same password. Since it defaults to `99999` days (over 200 years) it
  effectively means users never have to change their password.

  Organizations with effective policies for maintaining secure passwords typically
  change this value to `60` or `30` days.

  This setting affects the default setting of the `/etc/shadow` file highlighted below:

  ```
  bob:pw:15020:5: 30 :7:60:15050:
  ```

- **Password Minimum Days**

  ```
  PASS_MIN_DAYS     0
  ```

  With this set to a default value of zero, the shortest time that a user is required to
  keep a password is zero days, which means that they can immediately change a
  password that they have just set.

If the `PASS_MIN_DAYS` value was set to three days, then after setting a new password, the user would have to wait three days before they could change it again.

This setting affects the default setting of the `/etc/shadow` file highlighted below:

```
bob:pw:15020:3:30:7:60:15050:
```

- **Password Minimum Length**

```
PASS_MIN_LEN      5
```

This indicates the minimum number of characters that a password must contain.

- **Password Warning**

```
PASS_WARN_AGE     7
```

This is the default for the warning field. As a user approaches the maximum number of days that they can use their password, the system checks to see if it is time to start warning the user about changing their password at login.

This setting affects the default setting of the `/etc/shadow` file highlighted below:

```
bob:pw:15020:3:30:7:60:15050:
```

- **UID Minimum**

```
UID_MIN                    500
```

The `UID_MIN` determines the first UID that is assigned to an ordinary user. Any UID less than this value would either be for a system account or the root account.

- **UID Maximum**

```
UID_MAX                    60000
```

A UID technically could have a value of over four billion. For maximum compatibility, it's recommended to leave it at its default value of 60000.

- **GID Minimum**

```
GID_MIN                    500
```

The `GID_MIN` determines the first GID that is assigned to an ordinary group. Any group with a GID less than this value would either be a system group or the root group.

- **GID Maximum**

```
GID_MAX                    60000
```

A GID, like a UID, could have a value of over four billion. Whatever value you use for your `UID_MAX`, should be used for `GID_MAX` to support UPG.

- **Home Directory**

```
CREATE_HOME        yes
```

The value of this determines whether or not a new directory is created for the user when their account is created.

Our virtual machines do not include this value. Therefore a home directory is not created for new users unless specified.

- **Umask**

```
UMASK              077
```

UMASK works at the time the user home directory is being created; it determines what default permissions are placed on this directory. Using the default value of 077 for UMASK means that only the user owner has any kind of permission to access their directory.

The *UMASK* value will be covered in more detail in the chapter on permissions.

- **UPG**

```
USERGROUPS_ENAB yes
```

In distributions that feature a private group for each user, as this CentOS example shows, the USERGROUPS_ENAB will have a value of yes. If UPG is not used in the distribution, then this will have a value of no.

- **Encryption**

```
ENCRYPT_METHOD SHA512
```

The encryption method that is used to encrypt the users' passwords in the /etc/shadow file. The ENCRYPT_METHOD setting overrides the MD5_CRYPT_ENAB setting.

- **Encryption (Deprecated)**

```
MD5_CRYPT_ENAB no
```

This deprecated setting originally allowed the administrator to specify using MD5 encryption of passwords instead of the original DES encryption. It has been superseded by the ENCRYPT_METHOD setting.

16.3.3 Account Considerations

Creating a user account for use with a Linux system may require that you gather several pieces of information. While all that may be required is the account name, you may also want to plan the UID, the primary group, the supplementary groups, the home directory, the skeleton directory, and the shell to be used. When planning these values, consider the following:

Username

The only required argument for the `useradd` command is the name you want the account to have. The username should follow the same guidelines as for group names. Following these guidelines can help you to select a username that is portable:

- The first character of the name should be either an underscore _ character or a lower-case alphabetic `a-z` character.
- Up to 32 characters are allowed on most Linux distributions, but using more than 16 can be problematic as some distributions may not accept more than 16.
- After the first character, the remaining characters can be alphanumeric, a dash – character or an underscore _ character.
- The last character should not be a hyphen - character.

If the user needs to access multiple systems, it is usually recommended to have the account name be the same on those systems. The account name must be unique for each user.

```
root@localhost:~# useradd jane
```

User Identifier (UID)

Once you create a user with a specific UID, the system generally increments the UID by one for the next user that you create. If attached to a network with other systems, you may want to ensure that this UID is the same on all systems to help provide consistent access.

Adding the `-u` option to the `useradd` command allows you to specify the UID number. UIDs typically can range anywhere from zero to over four billion, but for greatest compatibility with older systems, the maximum recommended UID value is 60,000.

```
root@localhost:~# useradd -u 1000 jane
```

The root user has a UID of `0`, which allows that account to have special privileges. Any account with a UID of `0` would effectively be able to act as the administrator.

System accounts are generally used to run background services called daemons. By not having services run as the root user, the amount of damage that can be done with a compromised service account is limited. System accounts used by services generally use UIDs that are in the reserved range. One system account that is an exception to this rule is the user `nfsnobody`, which has a UID of `65534`.

The reserved range used for service accounts has expanded over time. Initially, it was for UIDs between 1 and 99. Then, it expanded to be between 1 and 499. The current trend among distributions is that system accounts are any account that has a UID between 1 and 999, but the range 1-499 is also still commonly used.

When setting up a new system, it is a good practice to start UIDs no lower than 1000 ensuring there are sufficient UIDs available for many system services and giving you the ability to create many GIDs in the reserved range.

Primary Group

In distributions which feature UPG, this group is created automatically with a GID and group name that matches the UID and username of the newly created user account. In distributions not using UPG, the primary group ordinarily defaults to the `users` group with a GID of 100.

To specify a primary group with the `useradd` command, use the `-g` option with either the name or GID of the group. For example, to specify `users` as the primary group:

```
root@localhost:~# useradd -g users jane
```

Supplementary Group

To make the user a member of one or more supplementary groups, the -G option can be used to specify a comma-separated list of group names or numbers. For example to specify sales and research as supplementary groups:

```
root@localhost:~# useradd -G sales,research jane
```

Home Directory

By default, most distributions create the user's home directory with the same name as the user account underneath whichever base directory is specified in the HOME setting of the /etc/default/useradd file, which typically specifies the /home directory. For example, if creating a user account named jane, the user's new home directory would be /home/jane.

```
root@localhost:~# useradd jane

root@localhost:~# grep '/home/jane' /etc/passwd

jane:x:1008:1010::/home/jane:/bin/sh
```

There are several options for the useradd command that can affect creating the user's home directory:

- If CREATE_HOME is set to no or this setting is not present, then the directory will not be created automatically. Otherwise, the -M option is used to specify to the useradd command that it should not create the home directory, even if CREATE_HOME is set to yes.

- If the CREATE_HOME setting in the /etc/login.defs file is set to yes, the home directory is created automatically. Otherwise, the -m option can be used to make the home directory.

```
• root@localhost:~# useradd -m jane

• root@localhost:~# ls -ld /home/jane

• drwxr-xr-x 2 jane jane 4096 Dec 18 19:14 /home/jane
```

- The -b option allows you to specify a different base directory under which the user's home directory is created. For example, the following creates the user account jane with a /test/jane created as the user's home directory:

```
• root@localhost:~# useradd -mb /test jane

• root@localhost:~# ls -ld /test/Jane

• drwxr-xr-x 2 jane jane 4096 Dec 18 19:16 /test/jane
```

- The -d option allows you to specify either an existing directory or a new home directory to create for the user. This should be a full path for the user's home directory. For example, the following creates the user account jane with a /test/jane created as the user's home directory:

```
● root@localhost:~# useradd -md /test/jane jane

● root@localhost:~# ls -ld /test/jane

● drwxr-xr-x 2 jane jane 4096 Dec 18 19:19 /test/jane
```

- The `-k` option to specifies a different skeleton directory. When using the `-k` option, the `-m` option is required.

Skeleton Directory

By default, the contents of the `/etc/skel` directory are copied into the new user's home directory. The resulting files are also owned by the new user. By using the `-k` option with the `useradd` command, the contents of a different directory can be used to populate a new user's home directory. When specifying the skeleton directory with the `-k` option, the `-m` option must be used or else the `useradd` command will fail with an error.

The following example uses `/home/sysadmin` as the skeleton directory:

```
root@localhost:~# useradd -mk /home/sysadmin jane
root@localhost:~# ls /home/jane

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Vi
deos
```

Shell

While the default shell is specified in the `/etc/default/useradd` file, it can also be overridden with the `useradd` command using the `-s` option at the time of account creation:

```
root@localhost:~# useradd -s /bin/bash jane
```

It is common to specify the `/sbin/nologin` shell for accounts to be used as system accounts.

Comment

The comment field, originally called the General Electric Comprehensive Operating System (GECOS) field, is typically used to hold the user's full name. Many graphical login programs display this field's value instead of the account name. The `-c` option of the `useradd` command allows for the value of this field to be specified.

```
root@localhost:~# useradd -c 'Jane Doe' jane
```

16.3.5 Passwords

Choosing a good password is not an easy task, but it is critical that it is done properly or the security of an account (perhaps the entire system) could be compromised. Picking a good password is only a start; you need to be very careful with your password so that it is not revealed to other people. You should never tell anyone your password and never let anyone see you type your password. If you do choose to write down your password, then you should store it securely in a place like a safe or safe deposit box.

There are numerous factors to consider when you are trying to choose a password for an account:

- **Length**: The `/etc/login.defs` file allows the administrator to specify the minimum length of the password. While some believe that the longer the password, the better, this isn't really correct. The problem with passwords that are too long is that they are not easily remembered and, as a result, they are often written down in a place where they can easily be found and compromised.
- **Composition**: A good password should be composed of a combination of alphabetic, numeric and symbolic characters.
- **Lifetime**: The maximum amount of time that a password can be used should be limited for several reasons:

  - If an account is compromised and the time that the password is valid is limited, the intruder will ultimately lose access when the password becomes invalid.
  - If an account is not being used, then it can automatically be disabled when the password is no longer valid.
  - If attackers are attempting a "brute-force" attack by trying every possible password, then the password can be changed before the attack can succeed.

  However, requiring a user to change their password too often might pose security problems, including:

  - The quality of the password the user chooses might suffer.
  - The user may start writing their password on paper, increasing the possibility that the password may be discovered.
  - Seldom used user accounts may become expired and require administrative attention to reset.

Opinions vary about how often users should be forced to change their passwords. For highly-sensitive accounts, it is recommended to change passwords more frequently, such as every 30 days. On the other hand, for non-critical accounts without any access to sensitive information, there is less need for frequent change. For accounts with minimal risk, having a duration of 90 days would be considered more reasonable.

16.3.5.1 Setting a User Password

There are several ways for a user password to be changed. The user can execute the `passwd` command, the administrator can execute the `passwd` command providing the username as an argument, or graphical tools are also available.

The administrator can use the `passwd` command to either set the initial password or change the password for the account. For example, if the administrator had created the account `jane`, then executing `passwd jane` provides the administrator a prompt to set the password for the jane account. If completed successfully, then the /etc/shadow file will be updated with the user's new password.

While regular users must follow many password rules, the root user only needs to follow one rule: the password cannot be left blank. When the root user violates all other password rules that normally apply to regular users, it results in a warning being printed to the screen and the rule not being enforced:

```
root@localhost:~# passwd Jane

Enter new UNIX password:

BAD PASSWORD: it is WAY to short

BAD PASSWORD: is too simple
```

```
Retype new UNIX password:
```

Assuming that the administrator has set a password for a user account, the user can then log in with that account name and password. After the user opens a terminal, they can execute the `passwd` command with no arguments to change their own password. They are prompted for their current password and then prompted to enter the new password twice.

As an ordinary user, it may be difficult to set a valid password because all of the rules for the password must be followed. The user is normally allowed three attempts to provide a valid password before the `passwd` command exits with an error.

Using the privileges of the root user, the encrypted passwords and other password-related information can be viewed by viewing the `/etc/shadow` file. Recall that regular users cannot see the contents of this file.

16.3.5.2 Managing Password Aging

The `chage` command provides many options for managing the password aging information found in the `/etc/shadow` file.

Here's a summary of the chage options:

Short Option	Long Option	Description
`-l`	`--list`	List the account aging information
`-d LAST_DAY`	`--lastday LAST_DAY`	Set the date of the last password change to `LAST_DAY`
`-E EXPIRE_DATE`	`--expiredate EXPIRE_DATE`	Set account to expire on `EXPIRE_DATE`
`-h`	`--help`	Show the help for the `chage` command
`-I INACTIVE`	`--inactive INACTIVE`	Set account to permit login for `INACTIVE` days after password expires
`-m MIN_DAYS`	`--mindays MIN_DAYS`	Set the minimum number of days before the

Short Option	Long Option	Description
		password can be changed to `MIN_DAYS`
`-M MAX_DAYS`	`--maxdays MAX_DAYS`	Set the maximum number of days before a password should be changed to `MAX_DAYS`
`-W WARN_DAYS`	`--warndays WARN_DAYS`	Set the number of days before a password expires to start displaying a warning to `WARN_DAYS`

A good example of the `chage` command would be to change the maximum number of days that an individual's password is valid to be 60 days:

```
root@localhost:~# chage -M 60 jane
root@localhost:~# grep jane /etc/shadow | cut -d: -f1,5
jane:60
```

```
sysadmin@localhost:/home/Aisha$ sudo grep jane /etc/shadow
jane:!:20146:0:99999:7:::
sysadmin@localhost:/home/Aisha$ sudo chage -M 60 jane
sysadmin@localhost:/home/Aisha$ sudo grep jane /etc/shadow
jane:!:20146:0:60:7:::
sysadmin@localhost:/home/Aisha$
```

16.3.6 Modifying a User

Before making changes to a user account, understand that some commands will not successfully modify a user account if the user is currently logged in (such as changing the user's login name).

Other changes that you might make won't be effective if the user is logged in, but will become effective as soon as the user logs out and then logs back in again. For example, when modifying group memberships, the new memberships will be unavailable to the user until the next time the user logs in.

In either case, it is helpful to know how to use the `who`, `w`, and `last` commands, so you can be aware of who is logged into the system, as this may impact the changes that you want to make to a user account.

Both the `who` and the `w` commands display who is currently logged into the system. The `w` command is the more verbose of the two, as it shows the system's uptime and load information as well as what process each user is running. The `last` command can be used to determine current and previous login sessions as well as their specific date and time. By providing a username or a `tty` (terminal) name as an argument, the command only shows records that match that name.

The `usermod` command offers many options for modifying an existing user account. Many of these options are also available with the `useradd` command at the time the account is created. The following chart provides a summary of the `usermod` options:

Short Option	Long Option	Description
`-c`	`COMMENT`	Sets the value of the GECOS or comment field to `COMMENT`.
`-d HOME_DIR`	`--home HOME_DIR`	Sets `HOME_DIR` as a new home directory for the user.
`-e EXPIRE_DATE`	`--expiredate EXPIRE_DATE`	Set account expiration date to `EXPIRE_DATE`.
`-f INACTIVE`	`--inactive INACTIVE`	Set account to permit login for `INACTIVE` days after password expires.
`-g GROUP`	`--gid GROUP`	Set `GROUP` as the primary group.
`-G GROUPS`	`--groups GROUPS`	Set supplementary groups to a list specified in `GROUPS`.
`-a`	`--append`	Append the user's supplemental groups with those specified by the `-G` option.

Short Option	Long Option	Description
`-h`	`--help`	Show the help for the `usermod` command.
`-l NEW_LOGIN`	`--login NEW_LOGIN`	Change the user's login name.
`-L`	`--lock`	Lock the user account.
`-s SHELL`	`--shell SHELL`	Specify the login shell for the account.
`-u NEW_UID`	`--uid NEW_UID`	Specify the user's UID to be `NEW_UID`.
`-U`	`--unlock`	Unlock the user account.

Several of these options are worthy of discussion because of how they impact user management. It can be very problematic to change the user's UID with the `-u` option, as any files owned by the user will be orphaned. On the other hand, specifying a new login name for the user with `-l` does not cause the files to be orphaned.

Deleting a user with the `userdel` command can either orphan or remove the user's files on the system. Instead of deleting the account, another choice is to lock the account with the `-L` option for the `usermod` command. Locking an account prevents the account from being used, but ownership of the files remains.

There are some important things to know about managing the supplementary groups. If you use the `-G` option without the `-a` option, then you must list all the groups to which the user would belong. Using the `-G` option alone can lead to accidentally removing a user from all the former supplemental groups that the user belonged to.

If you use the `-a` option with `-G` then you only have to list the new groups to which the user would belong. For example, if the user `jane` currently belongs to the `sales` and `research` groups, then to add her account to the `development` group, execute the following command:

```
root@localhost:~# usermod -aG development jane
```

16.3.7 Deleting a User

The `userdel` command is used to delete users. When you delete a user account, you also need to decide whether to delete the user's home directory. The user's files may be important to the organization, and there may even be legal requirements to keep the data for a certain amount of time, so be careful not to make this decision lightly. Also, unless

you've made backup copies of the data, once you've executed the command to delete the user and their files, there is no reversing the action.

To delete the user `jane` without deleting the user's home directory `/home/jane`, execute:

```
root@localhost:~# userdel jane
```

Beware that deleting a user without deleting their home directory means that the user's home directory files will be orphaned and these files will be owned solely by their former UID and GID.

To delete the user, home directory, and mail spool as well, use the `-r` option:

```
root@localhost:~# userdel -r jane
```

WARNING

The above command will only delete the user's files in their home directory and their mail spool. If the user owns other files outside of their home directory, then those files will continue to exist as orphaned files

Chapter 17: ownership and permissions

17.1 Introduction

File ownership is critical for file security. Every file has a user owner and a group owner.

This chapter focuses on how to specify the user and group ownership of a file. In addition, the concept of file and directory permissions is explored, including how to change the permissions on files and directories. Default permissions are the permissions given to files and directories when they are initially created.

17.2 File Ownership

By default, users own the files that they create. While this ownership can be changed, this function requires administrative privileges. Although most commands usually show the user owner as a name, the operating system is associating the user ownership with the UID for that username.

Every file also has a group owner. By default, the primary group of the user who creates the file is the group owner of any new files. Users are allowed to change the group owner of files they own to any group that they belong to. Similar to user ownership, the association of a file with a group is not done internally by the operating system by name, but by the GID of the group.

Since ownership is determined by the UID and GID associated with a file, changing the UID of a user (or deleting the user) has the effect of making a file that was originally

owned by that user have no real user owner. When there is no UID in
the `/etc/passwd` file that matches the UID of the owner of the file, then the UID (the
number) is displayed as the user owner of the file instead of the username (which no
longer exists). The same occurs for groups.

The `id` command can be useful for verifying which user account you are using and which
groups you have available to use. By viewing the output of this command, you can see
the user's identity information expressed both as a number and as a name.

The output of the `id` command displays the UID and user account name of the current
user followed by the GID and group name of the primary group and the GIDs and group
names of all group memberships:

```
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27
(sudo),1005(research),1006(development)
```

The above example shows the user has a UID of `1001` for the user account `sysadmin`.
It also shows that the primary group for this user has a GID of `1001` for the
group `sysadmin`.

Because the user account and primary group account have the same numeric identifier
and name, this indicates that this user is in a User Private Group (UPG). In addition, the
user in this example belongs to four supplemental groups: the `adm` group with a GID of `4`,
the `sudo` group with a GID of `27`, the `research` group with a GID of `1005` and
the `development` group with a GID of `1006`.

When a file is created, it belongs to the current user and their current primary group. If
the user from the previous example executes the `touch` command to create a file, then
the user owner of the file is the `sysadmin` user, and the group owner is
the `sysadmin` group:

```
sysadmin@localhost:~$ touch /tmp/filetest1
```

The file ownership can be confirmed using the long listing `-l` option of the `ls` command.

```
sysadmin@localhost:~$ ls -l /tmp/filetest1
-rw-rw-r--. 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/filetest1
```

File ownership also applies to hidden files in the system. Hidden files, which begin with
the period `.` character are listed using the `-a` option of the `ls` command. The first two
hidden files listed are the current `.` and parent `..` directories respectively. The ownership
of all files and subdirectories within the current directory can be listed using the `ls -
la` command.

```
sysadmin@localhost:~$ ls -la
total 60
drwxr-xr-x 1 sysadmin sysadmin 4096 Nov  3 22:29 .
drwxr-xr-x 1 root     root     4096 Mar 14  2016 ..
-rw-r--r-- 1 sysadmin sysadmin  220 Apr  3  2012 .bash_logout
-rw-r--r-- 1 sysadmin sysadmin 3768 Mar 14  2016 .bashrc
```

```
drwx------ 2 sysadmin sysadmin 4096 Nov  3 22:29 .cache

-rw-r--r-- 1 sysadmin sysadmin  675 Apr  3  2012 .profile

-rw-r--r-- 1 sysadmin sysadmin   74 Mar 14  2016 .selected_editor

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Desktop

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Documents

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Downloads

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Music

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Pictures

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Public

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Templates

drwxr-xr-x 2 sysadmin sysadmin 4096 Mar 14  2016 Videos
```

Consider This

The output of the `ls -l` command includes multiple pieces of information that are relevant to this chapter including:

- **Permissions**

  ```
  - rw-rw-r-- . 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/filetest1
  ```

- **User Owner**

  ```
  -rw-rw-r--. 1 sysadmin  sysadmin 0 Oct 21 10:18 /tmp/filetest1
  ```

- **Group Owner**

  ```
  -rw-rw-r--. 1 sysadmin  sysadmin  0 Oct 21 10:18 /tmp/filetest1
  ```

17.3 Changing Groups

If you know that the file you are about to create should belong to a group different from your current primary group, then you can use the `newgrp` command to change your current primary group.

```
newgrp group_name
```

The `id` command lists your identity information, including your group memberships. If you are only interested in knowing what groups you belong to, then you can execute the `groups` command:

```
sysadmin@localhost:~$ groups

sysadmin adm sudo research development
```

The output of the `groups` command may not be as detailed as the output of the `id` command, but if all you need to know is what groups you can switch to by using

the `newgrp` command, then the `groups` command provides the information that you need. The `id` command output does show your current primary group, so it is useful for verifying that the `newgrp` command succeeded.

For example, if the `sysadmin` user was planning on having a file owned by the group `research`, but that wasn't the user's primary group, then the user could use the `newgrp` command and then verify the correct primary group with the `id` command before creating the new file:

```
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27
(sudo),1005(research),1006(development)
sysadmin@localhost:~$ newgrp research
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1005(research) groups=1005(research),4(adm),27
(sudo),1001(sysadmin),1006(development)
```

According to the output of the previous commands, initially the user's GID is `1001` for the sysadmin user, then the `newgrp` command is executed, and the user's primary GID becomes `1005`, the `research` group. After these commands were executed, if the user were to create another file and view its details, the new file would be owned by the `research` group:

```
sysadmin@localhost:~$ touch /tmp/filetest2

sysadmin@localhost:~$ ls -l /tmp/filetest2

-rw-r--r--. 1 sysadmin research 0 Oct 21 10:53 /tmp/filetest2
```

The `newgrp` command opens a new shell; as long as the user stays in that shell, the primary group won't change. To switch the primary group back to the original, the user can leave the new shell by running the `exit` command. For example:

```
sysadmin@localhost:~$ id

uid=1001(sysadmin) gid=1005(research) groups=1005(research),4(adm),27
(sudo),1001

(sysadmin),1006(development)

sysadmin@localhost:~$ exit

exit

sysadmin@localhost:~$ id

uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27
(sudo),1005(research),1006(development)
```

Consider This

Administrative privileges are required to change the primary group of the user permanently. The root user would execute the following command:

```
usermod -g groupname username
```

17.4 Changing Group Ownership

To change the group owner of an existing file the `chgrp` command can be used.

```
chgrp group_name file
```

As the root user, the `chgrp` command can be used to change the group owner of any file to any group. As a user without administrative privileges, the `chgrp` command can only be used to change the group owner of a file to a group that the user is already a member of:

```
sysadmin@localhost:~$ touch sample
sysadmin@localhost:~$ ls -l sample
-rw-rw-r-- 1 sysadmin sysadmin 0 Oct 23 22:12 sample
sysadmin@localhost:~$ chgrp research sample
sysadmin@localhost:~$ ls -l sample
-rw-rw-r--. 1 sysadmin research 0 Oct 23 22:12 sample
```

If a user attempts to modify the group ownership of a file that the user doesn't own, they receive an error message:

```
sysadmin@localhost:~$ chgrp development /etc/passwd
chgrp: changing group of '/etc/passwd': Operation not permitted
```

To change the group ownership of all of the files of a directory structure, use the recursive `-R` option to the `chgrp` command. For example, the command in the following example would change the group ownership of the `test_dir` directory and all files and subdirectories of the `test_dir` directory.

```
sysadmin@localhost:~$ chgrp -R development test_dir
```

Consider This

While you can view the ownership of a file with the `-l` option to the `ls` command, the system provides another command that is useful when viewing ownership and file permissions: the `stat` command. The `stat` command displays more detailed information about a file, including providing the group ownership both by group name and GID number:

```
sysadmin@localhost:~$ stat /tmp/filetest1
  File: `/tmp/filetest1'
  Size: 0            Blocks: 0          IO Block: 4096   regular em
pty file
Device: fd00h/64768d   Inode: 31477       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1001/sysadmin)   Gid: ( 1001/sysadm
in)
Access: 2013-10-21 10:18:02.809118163 -0700
Modify: 2013-10-21 10:18:02.809118163 -0700
Change: 2013-10-21 10:18:02.809118163 -0700
```

17.5 Changing User Ownership

The `chown` command allows the root user to change the user ownership of files and directories. A regular user cannot use this command to change the user owner of a file, even to give the ownership of one of their own files to another user. However, the `chown` command also permits changing group ownership, which can be accomplished by either root or the owner of the file.

There are three different ways the `chown` command can be executed. The first method is used to change just the user owner of the file.

```
chown user /path/to/file
```

For example, if the root user wanted to change the user ownership of the `filetest1` file to the user `jane`, then the following command could be executed:

```
root@localhost:~# chown jane /tmp/filetest1

root@localhost:~# ls -l /tmp/filetest1

-rw-rw-r-- 1 jane sysadmin 0 Dec 19 18:44 /tmp/filetest1
```

The second method is to change both the user and the group; this also requires root privileges. To accomplish this, you separate the user and group by either a colon or a period character. For example:

```
chown user:group /path/to/file

chown user.group /path/to/file
```
```
root@localhost:~# chown jane:users /tmp/filetest2

root@localhost:~# ls -l /tmp/filetest2

-rw-r--r-- 1 jane users 0 Dec 19 18:53 /tmp/filetest2
```

If a user doesn't have root privileges, they can use the third method to change the group owner of a file just like the `chgrp` command. To use `chown` only to change the group ownership of the file, use a colon or a period as a prefix to the group name:

```
chown :group /path/to/file

chown .group /path/to/file
```
```
jane@localhost:~$ chown .users /tmp/filetest1

jane@localhost:~$ ls -l /tmp/filetest1

-rw-rw-r-- 1 jane users 0 Dec 19 18:44 /tmp/filetest1
```

```
sysadmin@localhost:~$ touch test_file.txt
sysadmin@localhost:~$ ls -l
total 4
-rw-r--r-- 1 root      develpment    0 Feb 28 18:58 Aisha
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Desktop
drwxr-xr-x 4 sysadmin sysadmin    4096 Feb  8  2021 Documents
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Downloads
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Music
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Pictures
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Public
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Templates
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Videos
-rw-rw-r-- 1 sysadmin sysadmin       0 Feb 28 19:42 test_file.txt
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ cat /etc/passwd | grep Aisha
Aisha:x:1002:1002::/home/Aisha:/bin/bash
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ sudo chown Aisha test_file.txt
sysadmin@localhost:~$ ls -l
total 4
-rw-r--r-- 1 root      develpment    0 Feb 28 18:58 Aisha
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Desktop
drwxr-xr-x 4 sysadmin sysadmin    4096 Feb  8  2021 Documents
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Downloads
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Music
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Pictures
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Public
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Templates
drwxr-xr-x 2 sysadmin sysadmin       6 Feb  8  2021 Videos
-rw-rw-r-- 1 Aisha     sysadmin       0 Feb 28 19:42 test_file.txt
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ grep research /etc/group
research:x:1002:
sysadmin@localhost:~$ sudo chown Aisha:research test_file.txt
sysadmin@localhost:~$ ls -l test_file.txt
-rw-rw-r-- 1 Aisha research 0 Feb 28 19:42 test_file.txt
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ sudo chown :users test_file.txt
sysadmin@localhost:~$ ls -l test_file.txt
-rw-rw-r-- 1 Aisha users 0 Feb 28 19:42 test_file.txt
sysadmin@localhost:~$
```

17.6 Permissions

The output of the `ls -l` command displays ten characters at the beginning of each line. These characters indicate the type of file and the permissions of the file. For example, consider the output of the following command:

```
root@localhost:~# ls -l /etc/passwd
-rw-r--r--. 1 root root 4135 May 27 21:08 /etc/passwd
```

File Type

The first character of each line indicates the type of file:

```
-rw-r--r-- 1 root root 4135 May 27 21:08 /etc/passwd
```

The following table describes the possible values for the file type:

Character	Type of the File
-	A regular file, which may be empty, or contain text or binary data.
d	A directory file, which contains the names of other files and links to them.
l	A symbolic link is a file name that refers (points) to another file.
b	A block file is one that relates to a block hardware device where data is read in blocks of data.
c	A character file is one that relates to a character hardware device where data is read one byte at a time.
p	A pipe file works similar to the pipe symbol, allowing for the output of one process to communicate to another process through the pipe file, where the output of the one process is used as input for the other process.
s	A socket file allows two processes to communicate, where both processes are allowed to either send or receive data.

Consider This

Although all the file types are listed in the table above, typically you don't encounter anything but regular, directory and link files unless you explore the /dev directory.

```
sysadmin@localhost:~$ ls -l /dev
total 0
crw-------  1 sysadmin tty  136, 0 Feb 28 19:52 console
lrwxrwxrwx 1 root      root      11 Feb 28 18:50 core -> /proc/kcore
lrwxrwxrwx 1 root      root      13 Feb 28 18:50 fd -> /proc/self/fd
crw-rw-rw- 1 root      root   1, 7 Feb 28 18:50 full
srw-rw-rw- 1 root      root       0 Feb 28 18:50 log
drwxrwxrwt 2 root      root      40 Feb 28 18:50 mqueue
crw-rw-rw- 1 root      root   1, 3 Feb 28 18:50 null
lrwxrwxrwx 1 root      root       8 Feb 28 18:50 ptmx -> pts/ptmx
drwxr-xr-x 2 root      root       0 Feb 28 18:50 pts
crw-rw-rw- 1 root      root   1, 8 Feb 28 18:50 random
drwxrwxrwt 2 root      root      40 Feb 28 18:50 shm
lrwxrwxrwx 1 root      root      15 Feb 28 18:50 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root      root      15 Feb 28 18:50 stdin  -> /proc/self/fd/0
lrwxrwxrwx 1 root      root      15 Feb 28 18:50 stdout -> /proc/self/fd/1
crw-rw-rw- 1 root      root   5, 0 Feb 28 19:31 tty
crw-rw-rw- 1 root      root   1, 9 Feb 28 18:50 urandom
crw-rw-rw- 1 root      root   1, 5 Feb 28 18:50 zero
sysadmin@localhost:~$
```

Permission Groups

The next nine characters demonstrate the permissions of the file.

```
- rw-r--r-- 1 root root 4135 May 27 21:08 /etc/passwd
```

The permissions set on these files determine the level of access that a user has on the file. When a user runs a program and the program accesses a file, then the permissions are checked to determine whether the user has the correct access rights to the file.

The permissions are grouped into three different roles, representing the different users that may try to access the file.

If you aren't the owner and you're not a member of the file/directory group, then your permissions would be others.

User Owner

```
- rw- r--r-- 1 root root 4135 May 27 21:08 /etc/passwd
```

Characters 2-4 indicate the permissions for the user that owns the file. If you are the owner of the file, then only the user owner permissions are used to determine access to that file.

Group Owner

```
-rw- r-- r-- 1 root root 4135 May 27 21:08 /etc/passwd
```

Characters 5-7 indicate the permissions for the group that owns the file. If you are not the owner but are a member of the group that owns the file, then only group owner permissions are used to determine access to that file.

Other Permissions

```
-rw-r-- r-- 1 root root 4135 May 27 21:08 /etc/passwd
```

Characters 8-10 indicate the permissions for others or what is sometimes referred to as the world's permissions. This group includes all users who are not the file owner or a member of the file's group.

Permission Types

Each group is attributed three basic types of permissions: read, write, and execute.

User Owner			Group Owner			Other		
Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
r	w	x	r	w	x	r	w	x

The permissions themselves are deceptively simple and have a different meaning depending on whether they are applied to a file or a directory.

Read

The first character of each group represents the read permission. There is an `r` character if the group has the read permission, or a – character if the group does not.

- On a file, this allows processes to read the contents of the file, meaning the contents can be viewed and copied.
- On a directory, file names in the directory can be listed, but other details are not available.

Write

The second character of each group represents the write permission. There is a `w` character if the group has the write permission, or a – character if the group does not.

- A file can be written to by the process, so changes to a file can be saved. Note that `w` permission really requires `r` permission on the file to work correctly.
- On a directory, files can be added to or removed from the directory. Note that `w` permission requires `x` permission on the directory to work correctly.

Execute

The third character of each group represents the execute permission. There is an `x` character if the group has the execute permission, or a – character if the group does not.

- A file can be executed or run as a process.
- On a directory, the user can use the `cd` command to "get into" the directory and use the directory in a pathname to access files and, potentially, subdirectories under this directory.

17.7 Understanding Permissions

The descriptions of the permission types can be handy, but just themselves, they don't provide a clear description of how permissions work. To better understand how permissions work, consider the following scenarios.

To understand these scenarios, you should first understand the following diagram:

```
drwxr-xr-x . 17  root root  4096 23:38  /
drwxr-xr-x . 10  root root  128  03:38  /data
-rwxr-xr-- .  1  bob  bob   100  21:08  /data/abc.txt
```

The relevant information is highlighted. The first line represents the `/` directory, with a user owner of `root`, a group owner of `root` and permissions of `rwxr-xr-x`. The second line represents the `/data` directory, a directory that is under the `/` directory. The third line represents the `abc.txt` file, which is stored in the `/data` directory.

17.7.1 Scenario #1 - Directory Access

Question: Based on the following information, what access would the user `bob` have on the file `abc.txt`?

```
drwxr-xr-x. 17 root root 4096 23:38 /

drwxr-xr--. 10 root root 128  03:38 /data

-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Answer: None.

Explanation: Initially it would appear that the user `bob` can view the contents of the `abc.txt` file as well as copy the file, modify its contents and run it like a program. This erroneous conclusion would be the result of looking solely at the file's permissions (`rwx` for the user `bob` in this case).

However, to do anything with the file, the user must first "get into" the `/data` directory. The permissions for `bob` for the `/data` directory are the permissions for "others" (`r--`), which means `bob` can't even use the `cd` command to get into the directory. If the execute permission (`--x`) were set for the directory, then the user `bob` would be able to "get into" the directory, meaning the permissions of the file itself would apply.

Lesson Learned: The permissions of all parent directories must be considered before considering the permissions on a specific file.

For more clarity:

The data directory is owned by the root user, but the file abc.txt in the data directory is owned by bob user. And when bob user try to enter the data directory, it will require him a root permission, because other than the user root will have read permission only.

17.7.2.1 Scenario #2 - Answer

Question: Based on the following information, who can use the `ls` command to display the contents of the `/data` directory (`ls /data`)?

```
drwxr-xr-x. 17 root root 4096 23:38 /

drwxr-xr--. 10 root root 128  03:38 /data

-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Answer: All users.

Explanation: All that is required to be able to view a directory's contents is `r` permission on the directory (and the ability to access the parent directories). The `x` permission for all users in the `/` directory means all users can use `/` as part of a path, so everyone can get through the `/` directory to get to the `/data` directory. The `r` permission for all users in the /data directory means all users can use the `ls` command to view the contents. This includes hidden files, so the `ls -a` command also works on this directory.

However, note that in order to see file details (`ls -l`), the directory would also require `x` permission. So while the `root` user and members of the `root` group have this access on the `/data` directory, no other users would be able to execute `ls -l /data`.

Lesson Learned: The `r` permission allows a user to view a listing of the directory

17.7.3.1 Scenario #3 - Answer

Question: Based on the following information, who can delete the `/data/abc.txt` file?

```
drwxr-xr-x. 17 root root 4096 23:38 /

drwxrw-rw-. 10 root root 128  03:38 /data

-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Answer: Only the `root` user.

Explanation: A user needs no permissions at all on the file itself to delete a file. The `w` permission on the directory that the file is stored in is required to delete a file in a directory. Based on that, it would seem that all users could delete the `/data/abc.txt` file, since everyone has `w` permission on the directory.

However, to delete a file, you must also be able to "get into" the directory. Since only the `root` user has `x` permission on the `/data` directory, only root can "get into" that directory to delete files in this directory.

Lesson Learned: The `w` permission allows a user to delete files from a directory, but only if the user also has `x` permission on the directory.

17.7.4.1 Scenario #4 - Answer

Question: True or False: Based on the following information the user `bob` can successfully execute the following command: `more /data/abc.txt`?

```
drwxr-xr-x. 17 root root 4096 23:38 /

dr-xr-x--x. 10 root root 128  03:38 /data
```

```
-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

Answer: True.

Explanation: As previously mentioned, to access a file, the user must have access to the directory. The access to the directory only requires `x` permission; even though `r` permission would be useful to list files in a directory, it isn't required to "get into" the directory and access files within the directory.

When the command `more /data/abc.txt` is executed, the following permissions are checked: `x` permission on the `/` directory, `x` permission on the `data` directory and `r` permission on the `abc.txt` file. Since the user `bob` has all of these permissions, the command executes successfully.

Lesson Learned: The `x` permission is required to "get into" a directory, but the `r` permission on the directory is not necessary unless you want to list the directory's contents.

17.7.5.1 Scenario #5 - Answer

Question: True or False: Based on the following information the user `bob` can successfully execute the following command: `more /data/abc.txt`?

Note that the `/data` directory has different user and group owners than previous examples

```
drwxr-xr-x. 17 root  root    4096 23:38 /
dr-xr-x---. 10 sue   payroll 128  03:38 /data
-rwxr-xr--.  1 bob   bob     100  21:08 /data/abc.txt
```

Answer: Not enough information to determine.

Explanation: In order to access the `/data/abc.txt` file, the user `bob` needs to be able to "get into" the `/data` directory. This requires `x` permission, which `bob` may or may not have, depending on whether he is a member of the `payroll` group.

If `bob` is a member of the `payroll` group, then his permissions on the `/data` directory are `r-x`, and the command `more` will execute successfully (`bob` also needs `x` on `/` and `r` on `abc.txt`, which he already has).

If he isn't a member of the `payroll` group, his permissions on the `/data` directory are `---`, and the `more` command will fail.

Lesson Learned: You must look at each file and directory permissions separately and be aware of which groups the user account belongs to.

17.7.6.1 Scenario #6 - Answer

Question: True or False: Based on the following information the user `bob` can successfully execute the following command: `more /data/abc.txt`?

Note that the `/data` directory has different user and group owners than the previous example

```
drwxr-xr-x. 17 root root 4096 23:38 /
```

```
dr-xr-x---. 10 bob  bob  128  03:38 /data
----rw-rwx.  1 bob  bob  100  21:08 /data/abc.txt
```

Answer: False.

Explanation: Recall that if you are the owner of a file, then the only permissions that are checked are the user owner permissions. In this case, that would be `---` for `bob` on the `/data/abc.txt` file.

In this case, members of the `bob` group and "others" have more permissions on the file than `bob` has.

Lesson Learned: Don't provide permissions to the group owner and "others" without applying at least the same level of access to the owner of the file.

17.8 Changing Permissions

The `chmod` (change mode) command is used to change permissions on files and directories. There are two techniques that can be used with this command: symbolic and numeric. Both techniques use the following basic syntax:

```
chmod new_permission file_name
```

Important

To change a file's permissions, you either need to own the file or log in as the root user.

17.8.1 Symbolic Method

If you want to modify some of the current permissions, the symbolic method is usually easier to use. With this method, you specify which permissions you want to change on the file, and the other permissions remain as they are.

When specifying the *new_permission* argument of the `chmod` command using the symbolic method three types of information are required.

Start by using one or more of the following characters to indicate which permission group(s) to apply the changes to:

`u`	user owner
`g`	group owner
`o`	others
`a`	all (user owner, group owner, and others)

Then choose one of the following operators to indicate how to modify the permissions:

`+`	add

-	remove

=	equals

Lastly, use the following characters to specify the permissions type(s) to change:

r	read

w	write

x	execute

For example, to give the group owner write permission on a file named `abc.txt`, you could use the following command:

```
root@localhost:~# chmod g+w abc.txt
root@localhost:~# ls -l abc.txt
-rw-rw-r-- 1 root root 0 Dec 19 18:58 abc.txt
```

Only the group owner's permission was changed. All other permissions remained as they were prior to the execution of the `chmod` command.

You can combine values to make multiple changes to the file's permissions. For example, consider the following command which adds the execute permission to the user owner and group owner and removes the read permission for others:

```
root@localhost:~# chmod ug+x,o-r abc.txt
root@localhost:~# ls -l abc.txt
-rwxrwx--- 1 root root 0 Dec 19 18:58 abc.txt
```

Lastly, you could use the = character, which adds specified permissions and causes unmentioned ones to be removed. For example, to give the user owner only read and execute permissions, removing the write permission:

```
root@localhost:~# chmod u=rx abc.txt
root@localhost:~# ls -l abc.txt
-r-xrwx--- 1 root root 0 Dec 19 18:58 abc.txt
```

```
sysadmin@localhost:~$ sudo chmod u+x test_file.txt
[sudo] password for sysadmin:
sysadmin@localhost:~$ ls -l
total 4
-rw-r--r-- 1 root      develpment      0 Feb 28 18:58 Aisha
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Desktop
drwxr-xr-x 4 sysadmin sysadmin      4096 Feb  8  2021 Documents
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Downloads
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Music
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Pictures
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Public
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Templates
drwxr-xr-x 2 sysadmin sysadmin         6 Feb  8  2021 Videos
-rwxrw-r-- 1 Aisha     users           0 Feb 28 19:42 test_file.txt
```

```
sysadmin@localhost:~$ sudo chmod u=rw test_file.txt
sysadmin@localhost:~$ ls -l test_file.txt
-rw-rw-r-- 1 Aisha users 0 Feb 28 19:42 test_file.txt
sysadmin@localhost:~$
```

17.8.2 Numeric Method

The numeric method (also called the octal method) is useful when changing many permissions on a file. It is based on the octal numbering system in which each permission type is assigned a numeric value:

4	Read
2	Write
1	Execute

By using a combination of numbers from 0 to 7, any possible combination of read, write and execute permissions can be specified for a single permission group set. For example:

7	rwx
6	rw-
5	r-x

4	r--
3	-wx
2	-w-
1	--x
0	---

The *new_permission* argument is specified as three numbers, one number for each permission group. When the numeric method is used to change permissions, all nine permissions must be specified. Because of this, the symbolic method is generally easier for changing a few permissions while the numeric method is better for changes that are more drastic.

For example, to set the permissions of a file named `abc.txt` to be `rwxr-xr--` you could use the following command:

R = 4, w = 2, x=1

(rwx= 4+2+1=7)

(r-x= 4+0+1= 5)

(r--= 4+0+0=4)

```
root@localhost:~# chmod 754 abc.txt
root@localhost:~# ls -l abc.txt
-rwxr-xr-- 1 root root 0 Dec 19 18:58 abc.txt
```

Consider This

Recall the `stat` command provides more detailed information than the `ls -l` command. Because of this, you may consider using the `stat` command instead of the `ls -l` command when viewing permissions on a file. One big advantage of the `stat` command is that it shows permissions using both the symbolic and numeric methods, as highlighted below:

```
sysadmin@localhost:~$ stat /tmp/filetest1
  File: `/tmp/filetest1'
  Size: 0               Blocks: 0          IO Block: 4096    regular empty file
Device: fd00h/64768d    Inode: 31477       Links: 1
Access: (0664/-rw-rw-r--)  Uid: (  502/sysadmin)   Gid: (  503/sysadmin)
Access: 2013-10-21 10:18:02.809118163 -0700
Modify: 2013-10-21 10:18:02.809118163 -0700
Change: 2013-10-21 10:18:02.809118163 -0700
```

17.9 Default Permissions

The `umask` command is a feature that is used to determine default permissions that are set when a file or directory is created. Default permissions are determined when the umask value is subtracted from the maximum allowable default permissions. The maximum default permissions are different for files and directories:

file	rw-rw-rw-

directories	rwxrwxrwx

The permissions that are initially set on a file when it is created cannot exceed `rw-rw-rw-`. To have the execute permission set on a file, you first need to create the file and then change the permissions.

The `umask` command can be used to display the current umask value:

```
sysadmin@localhost:~$ umask
0002
```

A breakdown of the output:

- The first `0` indicates that the umask is given as an octal number.
- The second `0` indicates which permissions to subtract from the default user owner's permissions.
- The third `0` indicates which permissions to subtract from the default group owner's permissions.
- The last number `2` indicates which permissions to subtract from the default other's permissions.

Note that different users may have different umasks. Typically the root user has a more restrictive umask than normal user accounts:

```
root@localhost:~# umask
0022
```

To understand how umask works, assume that the umask of a file is set to `027` and consider the following:

File Default	666

Umask	-027

Result	640

The `027` umask means that new files would receive `640` or `rw-r-----` permissions by default, as demonstrated below:

```
sysadmin@localhost:~$ umask 027

sysadmin@localhost:~$ touch sample

sysadmin@localhost:~$ ls -l sample

-rw-r-----. 1 sysadmin sysadmin 0 Oct 28 20:14 sample
```

Because the default permissions for directories are different than for files, a umask of `027` would result in different initial permissions on new directories:

Directory Default	777
Umask	-027
Result	750

The `027` umask means that directories files would receive `750` or `rwxr-x---` permissions by default, as demonstrated below:

```
sysadmin@localhost:~$ umask 027

sysadmin@localhost:~$ mkdir test-dir

sysadmin@localhost:~$ ls -ld test-dir

drwxr-x---. 1 sysadmin sysadmin 4096 Oct 28 20:25 test-dir
```

The new umask is only applied to file and directories created during that session. When a new shell is started, the default umask will again be in effect.

Permanently changing a user's umask requires modifying the `.bashrc` file located in that user's home directory.

Chapter 18: special Directories and files

18.1 Introduction

In most circumstances, the basic Linux permissions, read, write, and execute, are enough to accommodate the security needs of individual users or organizations.

However, when multiple users need to work routinely on the same directories and files, these permissions may not be enough. The special permissions setuid, setgid and the sticky bit are designed to address these concerns.

18.2 Setuid

When the setuid permission is set on an executable binary file (a program) the binary file is run as the owner of the file, not as the user who executed it. This permission is set on a handful of system utilities so that they can be run by normal users, but executed with the permissions of root, providing access to system files that the normal user doesn't normally have access to.

Consider the following scenario in which the user `sysadmin` attempts to view the contents of the `/etc/shadow` file:

```
sysadmin@localhost:~$ more /etc/shadow

/etc/shadow: Permission denied

sysadmin@localhost:~$ ls -l /etc/shadow

-rw-r-----. 1 root root 5195 Oct 21 19:57 /etc/shadow
```

The permissions on `/etc/shadow` do not allow normal users to view (or modify) the file. Since the file is owned by the `root` user, the administrator can temporarily modify the permissions to view or modify this file.

Now consider the `passwd` command. When this command runs, it modifies the `/etc/shadow` file, which seems impossible because other commands that the `sysadmin` user runs that try to access this file fail. So, why can the `sysadmin` user modify the `/etc/shadow` file while running the `passwd` command when normally this user has no access to the file?

The `passwd` command has the special setuid permission set. When the `passwd` command is run, and the command accesses the `/etc/shadow` file, the system acts as if the user accessing the file is the owner of the `passwd` command (the root user), not the user who is running the command.

You can see this permission set by running the `ls -l` command:

```
sysadmin@localhost:~$ ls -l /usr/bin/passwd

-rwsr-xr-x 1 root root 31768 Jan 28 2010 /usr/bin/passwd
```

Notice the output of the `ls` command above; the setuid permission is represented by the `s` in the owner permissions where the execute permission would normally be represented. A lowercase `s` means that both the setuid and execute permission are set, while an uppercase `S` means that only setuid and not the user execute permission is set.

Like the read, write and execute permissions, special permissions can be set with the `chmod` command, using either the symbolic and octal methods.

To add the setuid permission symbolically, run:

```
chmod u+s file
```

To add the setuid permission numerically, add 4000 to the file's existing permissions (assume the file originally had 775 for its permission in the following example):

```
chmod 4775 file
```

To remove the setuid permission symbolically, run:

```
chmod u-s file
```

To remove the setuid permission numerically, subtract 4000 from the file's existing permissions:

```
chmod 0775 file
```

Previously, we set permission with the octal method using three-digit codes. When a three-digit code is provided, the `chmod` command assumes that the first digit before the three-digit code is 0. Only when four digits are specified is a special permission set.

If three digits are specified when changing the permission on a file that already has a special permission set, the first digit will be set to 0, and the special permission will be removed from the file.

18.3 Setgid

The setgid permission is similar to setuid, but it makes use of the group owner permissions. There are two forms of setgid permissions: setgid on a file and setgid on a directory. The behavior of setgid depends on whether it is set on a file or directory.

18.3.1 Setgid on Files

The setgid permission on a file is very similar to setuid; it allows a user to run an executable binary file in a manner that provides them additional (temporary) group access. The system allows the user running the command to effectively belong to the group that owns the file, but only in the setgid program.

A good example of the setgid permission on an executable file is the `/usr/bin/wall` command. Notice the permissions for this file as well as the group owner:

```
sysadmin@localhost:~$ ls -l /usr/bin/wall
-rwxr-sr-x 1 root tty 30800 May 16  2018 /usr/bin/wall
```

You can see that this file is setgid by the presence of the `s` in the group's execute position. Due to this executable being owned by the `tty` group, when a user executes this command, the command is able to access files that are group owned by the `tty` group.

This access is important because the `/usr/bin/wall` command sends messages to terminals, which is accomplished by writing data to files like the following:

```
sysadmin@localhost:~$ ls -l /dev/tty?
crw--w----. 1 root tty  4, 0 Mar 29  2013 /dev/tty0
crw--w----. 1 root tty  4, 1 Oct 21 19:57 /dev/tty1
```

Note that the `tty` group has write permission on the files above while users who are not in the `tty` group ("others") have no permissions on these files. Without the setgid permission, the `/usr/bin/wall` command would fail.

18.3.2 Setgid on Directories

When set on a directory, the setgid permission causes files created in the directory to be owned by the group that owns the directory automatically. This behavior is contrary to

how new file group ownership would normally function, as by default new files are group owned by the primary group of the user who created the file.

In addition, any directories created within a directory with the setgid permission set are not only owned by the group that owns the setgid directory, but the new directory automatically has setgid set on it as well. In other words, if a directory is setgid, then any directories created within that directory inherit the setgid permission.

By default when the `ls` command is executed on a directory, it outputs information on the files contained within the directory. To view information about the directory itself add the `-d` option. Used with the `-l` option, it can be used to determine if the setgid permission is set. The following example shows that the `/tmp/data` directory has the setgid permission set and that it is owned by the `demo` group.

```
sysadmin@localhost:~$ ls -ld /tmp/data
drwxrwsrwx. 2 root demo 4096 Oct 30 23:20 /tmp/data
```

In a long listing, the setgid permission is represented by an `s` in the group execute position. A lowercase `s` means that both setgid and group execute permissions are set:

```
drwxrwsrwx. 2 root demo 4096 Oct 30 23:20 /tmp/data
```

An uppercase `S` means that only setgid and not group execute permission is set. If you see an uppercase `S` in the group execute position of the permissions, then it indicates that although the setgid permission is set, it is not really in effect because the group lacks the execute permission to use it:

```
drwxrwSr-x. 2 root root 5036 Oct 30 23:22 /tmp/data2
```

Typically files created by the user `sysadmin` are owned by their primary group, also called `sysadmin`.

```
sysadmin@localhost:~$ id
uid=500(sysadmin) gid=500(sysadmin)
groups=500(sysadmin),10001(research),10002(development)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

The UID and GID information in the example above may differ from the output in our virtual environment.

However, if the user `sysadmin` creates a file in the `/tmp/data` directory, the setgid directory from the preceding example, the group ownership of the file isn't the `sysadmin` group, but rather the group that owns the directory `demo`:

```
sysadmin@localhost:~$  touch /tmp/data/file.txt
sysadmin@localhost:~$  ls -ld /tmp/data/file.txt
-rw-rw-r--. 1 bob demo 0 Oct 30 23:21 /tmp/data/file.txt
```

Why would an administrator want to set up a setgid directory? First, consider the following user accounts:

- The user `bob` is a member of the `payroll` group.
- The user `sue` is a member of the `staff` group.
- The user `tim` is a member of the `acct` group.

In this scenario, these three users need to work on a joint project. They approach the administrator to ask for a shared directory in which they can work together, but that no one else can access their files. The administrator does the following:

1. Creates a new group called `team`.
2. Adds `bob`, `sue`, and `tim` to the `team` group.
3. Makes a new directory called `/home/team`.
4. Makes the group owner of the `/home/team` directory be the `team` group.
5. Gives the `/home/team` directory the following permissions: `rwxrwx---`

As a result, `bob`, `sue`, and `tim` can access the `/home/team` directory and add files. However, there is a potential problem: when `bob` creates a file in the `/home/team` directory, the new file is owned by his primary group:

```
-rw-r-----. 1 bob  payroll  100 Oct 30 23:21 /home/team/file.txt
```

Unfortunately, while `sue` and `tim` can access the `/home/team` directory, they can't do anything with bob's file. Their permissions for that file are the others permissions (`---`).

If the administrator sets the setgid permission to the `/home/team` directory, then when `bob` creates a file, it is owned the `team` group:

```
-rw-r-----. 1 bob  team  100 Oct 30 23:21 /home/team/file.txt
```

As a result, `sue` and `tim` would have access to the file through the group owner permissions (`r--`).

Certainly, `bob` could change the group ownership or the others permissions after creating the file, but that would become tedious if there were many new files being created. The setgid permission makes it easier for this situation.

18.3.3 Setting Setgid

Use the following syntax to add the setgid permission symbolically:

```
chmod g+s <file|directory>
```

To add the setgid permission numerically, add 2000 to the file's existing permissions (assume in the following example that the directory originally had 775 for its permissions):

```
chmod 2775 <file|directory>
```

To remove the setgid permission symbolically, run:

```
chmod g-s <file|directory>
```

To remove the setgid permission numerically, subtract 2000 from the file's existing permissions:
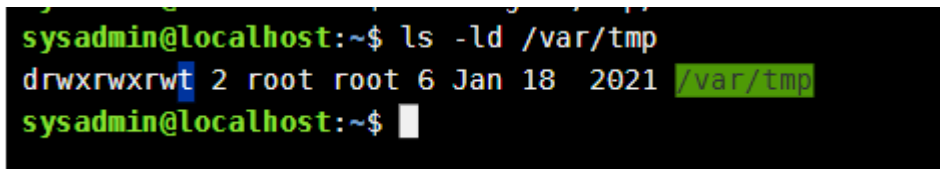
```
chmod 0775 <file|directory>
```

18.4 Sticky Bit

The sticky bit permission is used to prevent other users from deleting files that they do not own in a shared directory. Recall that any user with write permission on a directory can create files in that directory, as well as delete any file in the directory, even if they do not own the file!

The sticky bit permission allows for files to be shared with other users, by changing write permission on the directory so that users can still add and delete files in the directory, but files can only be deleted by the owner of the file or the root user.

A good example of the use of sticky bit directories would be the `/tmp` and `/var/tmp` directories. These directories are designed as locations where any user can create a temporary file.

Because these directories are intended to be writable by all users, they are configured to use the sticky bit. Without this special permission, users would be able to delete any files in this directory, including those that belong to other users.

The output of the `ls -l` command displays the sticky bit by a `t` character in the execute bit of the others permission group:

```
sysadmin@localhost:~$ ls -ld /var/tmp
drwxrwxrwt 2 root root 6 Jan 18  2021 /var/tmp
sysadmin@localhost:~$
```

A lowercase `t` means that both the sticky bit and execute permissions are set for others. An uppercase `T` means that only the sticky bit permission is set.

While the capital `S` indicated a problem with the setuid or setgid permissions, a capital `T` does not necessarily indicate a problem, as long as the group owner still has the execute permission.

To set the sticky bit permission symbolically, execute a command like the following:

```
chmod o+t <directory>
```

To set the sticky bit permission numerically, add 1000 to the directory's existing permissions (assume the directory in the following example originally had 775 for its permissions):

```
chmod 1775 <file|directory>
```

To remove the sticky permission symbolically, run:

```
chmod o-t <directory>
```

To remove the sticky bit permission numerically, subtract 1000 from the directory's existing permissions:

```
chmod 0775 <directory>
```

18.5 Links

Consider a scenario where there is a file deeply buried in the file system called:

```
/usr/share/doc/superbigsoftwarepackage/data/2013/october/tenth/valuab
le-information.txt
```

Another user routinely updates this file, and you need to access it regularly. The long file name is a not an ideal choice for you to type, but the file must reside in this location. It is also updated frequently, so you can't simply make a copy of the file.

In a situation like this, links come in handy. You can create a file that is linked to the one that is deeply buried. This new file could be placed in the home directory or any other convenient location. When you access the linked file, it accesses the contents of the `valuable-information.txt` file.

Each linking method, hard and symbolic, results in the same overall access, but uses different techniques. There are pros and cons to each method, so knowing both techniques and when to use them is important.

18.5.1 Creating Hard Links

To understand hard links, it is helpful to understand a little bit about how the file system keeps track of files. For every file created, there is a block of data on the file system that stores the metadata of the file. Metadata includes information about the file like the permissions, ownership, and timestamps. Metadata does not include the file name or the contents of the file, but it does include just about all other information about the file.

This metadata is called the file's inode table. The inode table also includes pointers to the other blocks on the file system called data blocks where the data is stored.

Every file on a partition has a unique identification number called an inode number. The `ls -i` command displays the inode number of a file.

```
sysadmin@localhost:~$ ls -i /tmp/file.txt
215220874 /tmp/file.txt
```

Like users and groups, what defines a file is not its name, but rather the number it has been assigned. The inode table does not include the file name. For each file, there is also an entry that is stored in a directory's data area (data block) that includes an association between an inode number and a file name.

In the data block for the `/etc` directory, there would be a list of all of the files in this directory and their corresponding inode number. For example:

File Name	Inode Number
passwd	123
shadow	175
group	144

File Name	Inode Number
gshadow	897

When you attempt to access the `/etc/passwd` file, the system uses this table to translate the file name into an inode number. It then retrieves the file data by looking at the information in the inode table for the file.

Hard links are two file names that point to the same inode. For example, consider the following directory entries:

File Name	Inode Number
passwd	123
mypasswd	123
shadow	175
group	144
gshadow	897

Because both the `passwd` and `mypasswd` file have the same inode number, they essentially are the same file. You can access the file data using either file name.

When you execute the `ls -li` command, the number that appears for each file between the permissions and the user owner is the link count number:

```
sysadmin@localhost:~$ ls -l test.txt
-rwxrw-r-- 1 sysadmin sysadmin 12 Mar  5 11:29 test.txt

sysadmin@localhost:~$ ln test.txt linked1_file.txt
```

The link count number indicates how many hard links have been created. When the number is a value of one, then the file has only one name linked to the inode.

To create hard links, the `ln` command is used with two arguments. The first argument is an existing file name to link to, called a target, and the second argument is the new file name to link to the target.

```
ln target link_name
```

When the `ln` command is used to create a hard link, the link count number increases by one for each additional filename:

```
sysadmin@localhost:~$ ls -l test.txt
-rwxrw-r-- 2 sysadmin sysadmin 12 Mar  5 11:29 test.txt
sysadmin@localhost:~$ ls -l linked1_file.txt
-rwxrw-r-- 2 sysadmin sysadmin 12 Mar  5 11:29 linked1_file.txt
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ cat test.txt
hello world
sysadmin@localhost:~$ cat linked1_file.txt
hello world
sysadmin@localhost:~$
```

8.5.2 Creating Symbolic Links

A symbolic link, also called a soft link, is simply a file that points to another file. There are several symbolic links already on the system, including several in the /etc directory:

```
sysadmin@localhost:~$ ls -l /etc/grub.conf

lrwxrwxrwx. 1 root root 22 Feb 15  2011 /etc/grub.conf -> ../boot/gru
b/grub.conf
```

In the previous example, the file /etc/grub.conf "points to" the ../boot/grub/grub.conf file. So, if you were to attempt to view the contents of the /etc/grub.conf file, it would follow the pointer and show you the contents of the ../boot/grub/grub.conf file.

To create a symbolic link, use the -s option with the ln command:

```
ln -s target link_name
```

```
sysadmin@localhost:~$ touch target.txt
sysadmin@localhost:~$ echo this is the target.txt file > target.txt
sysadmin@localhost:~$ ln -s target.txt linked2_file.txt
sysadmin@localhost:~$ ls -l linked2_file.txt
lrwxrwxrwx 1 sysadmin sysadmin 10 Mar  5 13:08 linked2_file.txt -> target.txt
sysadmin@localhost:~$ cat linked2_file.txt
this is the target.txt file
sysadmin@localhost:~$
```

Consider This

Notice that the first bit of the ls -l output is the l character, indicating that the file type is a link.

18.5.3 Comparing Hard and Symbolic Links

While hard and symbolic links have the same result, the different techniques produce different advantages and disadvantages. In fact, the advantage of one technique compensates for a disadvantage of the other technique.

Advantage: Hard links don't have a single point of failure.

One of the benefits of using hard links is that every file name for the file content is equivalent. If you have five files hard linked together, then deleting any four of these files would not result in deleting the actual file contents.

Recall that a file is associated with a unique inode number. As long as one of the hard linked files remains, then that inode number still exists, and the file data still exists.

Symbolic links, however, have a single point of failure: the original file. Consider the following example in which access to the data fails if the original file is deleted. The mytest.txt file is a symbolic link to the text.txt file:

```
sysadmin@localhost:~$ ls -l mytest.txt

lrwxrwxrwx. 1 sysadmin sysadmin 8 Oct 31 13:29 mytest.txt -> test.txt

sysadmin@localhost:~$ more test.txt

hi there

sysadmin@localhost:~$ more mytest.txt

hi there
```

If the original file, the test.txt file is removed, then any files linked to it, including the mytest.txt file, fail:

```
sysadmin@localhost:~$ rm test.txt

sysadmin@localhost:~$ more mytest.txt

mytest.txt: No such file or directory

sysadmin@localhost:~$ ls -l mytest.txt

lrwxrwxrwx. 1 sysadmin sysadmin 8 Oct 31 13:29 mytest.txt ->  test.tx
t
```

Advantage: Soft links are easier to see.

Sometimes it can be difficult to know where the hard links to a file exist. If you see a regular file with a link count that is greater than one, you can use the find command with the -inum search criteria to locate the other files that have the same inode number. To find the inode number you would first use the ls -i command:

```
sysadmin@localhost:~$ ls -i file.original

278772 file.original

sysadmin@localhost:~$ find / -inum 278772 2> /dev/null

/home/sysadmin/file.hard.1

/home/sysadmin/file.original
```

Soft links are much more visual, not requiring any extra commands beyond the ls command to determine the link:

```
sysadmin@localhost:~$ ls -l mypasswd
```

```
lrwxrwxrwx. 1 sysadmin sysadmin 11 Oct 31 13:17 mypasswd -> /etc/pass
wd
```

Advantage: Soft links can link to any file.

Since each file system (partition) has a separate set of inodes, hard links cannot be created that attempt to cross file systems:

```
sysadmin@localhost:~$ ln /boot/vmlinuz-2.6.32-358.6.1.el6.i686 Linux.
Kernel

ln: creating hard link `Linux.Kernel' => `/boot/vmlinuz-2.6.32-358.6.
1.el6.i686': Invalid cross-device link
```

In the previous example, a hard link was attempted to be created between a file in the /boot file system and the / file system; it failed because each of these file systems has a unique set of inode numbers that can't be used outside of the filesystem.

However, because a symbolic link points to another file using a pathname, you can create a soft link to a file in another filesystem:

```
sysadmin@localhost:~$ ln -s /boot/vmlinuz-2.6.32-358.6.1.el6.i686 Lin
ux.Kernel
sysadmin@localhost:~$ ls -l Linux.Kernel
lrwxrwxrwx. 1 sysadmin sysadmin 11 Oct 31 13:17 Linux.Kernel -> /boot
/vmlinuz-2.6.32-358.6.1.el6.i686
```

Advantage: Soft links can link to a directory.

Another limitation of hard links is that they cannot be created on directories. The reason for this limitation is that the operating system itself uses hard links to define the hierarchy of the directory structure. The following example shows the error message that is displayed if you attempt to hard link to a directory:

```
sysadmin@localhost:~$ ln /bin binary
ln: `/bin': hard link not allowed for directory
```

Linking to directories using a symbolic link is permitted:

```
sysadmin@localhost:~$ ln -s /bin binary
sysadmin@localhost:~$ ls -l binary
lrwxrwxrwx. 1 sysadmin sysadmin 11 Oct 31 13:17 binary -> /bin
```

```
sysadmin@localhost:~$ touch test.txt
sysadmin@localhost:~$ echo this is the test.txt file > test.txt
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  test.txt
Documents  Music      Public    Videos
sysadmin@localhost:~$ ln test.txt linked.txt
sysadmin@localhost:~$ cat linked.txt
this is the test.txt file
sysadmin@localhost:~$ rm test.txt
sysadmin@localhost:~$ cat linked.txt
this is the test.txt file
sysadmin@localhost:~$
```

```
sysadmin@localhost:~$ touch test2.txt
sysadmin@localhost:~$ ln -s test2.txt linked2.txt
sysadmin@localhost:~$ ls -l linked2.txt
lrwxrwxrwx 1 sysadmin sysadmin 9 Mar  5 17:43 linked2.txt -> test2.txt
sysadmin@localhost:~$ echo this is test2.txt file > test2.txt
sysadmin@localhost:~$ rm test2.txt
sysadmin@localhost:~$ ls -l linked2.txt
lrwxrwxrwx 1 sysadmin sysadmin 9 Mar  5 17:43 linked2.txt -> test2.txt
sysadmin@localhost:~$ cat linked2.txt
cat: linked2.txt: No such file or directory
sysadmin@localhost:~$
```

Source:

NDG-Linux essentials

GitHub:

https://github.com/aisha-x