Linux Essentials-Cisco:

1. Welcome!

Hello! Welcome to the NDG Linux Essentials course. NDG (Network Development Group) is very excited that you have decided to immerse yourself in the world of Linux. Before you get started, we want to take the opportunity to introduce you to the course, give you an idea of what you can expect from the material and your interaction with the features we have designed for you.

The NDG Linux Essentials course is designed to prepare you for the Linux Professional Institute Linux Essentials Professional Development Certificate, which validates a demonstrated understanding of:

- FOSS, the various communities, and licenses
- Knowledge of open source applications in the workplace as they relate to closed source equivalents
- Basic concepts of hardware, processes, programs and the components of the Linux Operating System
- How to work on the command line and with files
- How to create and restore compressed backups and archives
- System security, users/groups and file permissions for public and private directories
- How to create and run simple scripts

To obtain the Linux Essentials Professional Development Certificate you must pass Linux Essentials (LPI-010) which covers:

- The Linux community and a career in open source
- Finding your way on a Linux system
- The power of the command line
- The Linux operating system
- Security and file permissions

The Linux Essentials Professional Development Certificate is the beginning of your path to becoming a Linux certified professional. Information about the Linux Professional Institute certifications can be found by going to www.lpi.org.

Do not be concerned if you have little or no Linux experience. This course is the perfect starting place designed to teach all of the concepts from square one. However, if you do not find this material challenging enough, consider starting with NDG Introduction to Linux I, a more rigorous introductory course.

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*.

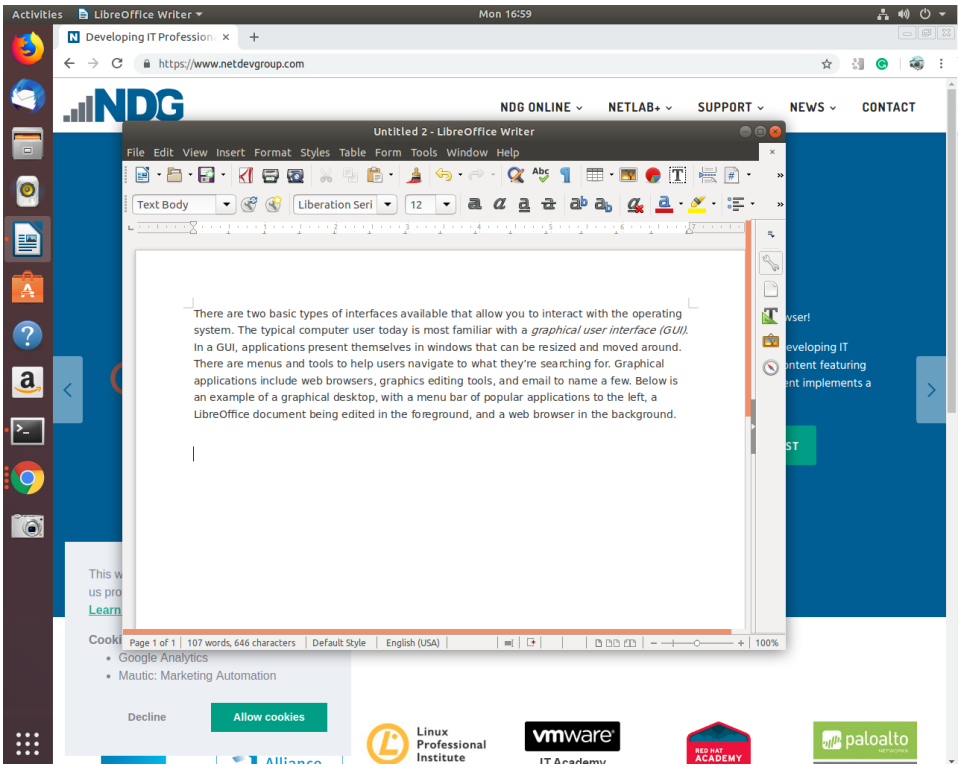
Take Linux and the GNU tools, add some user-facing applications like a web browser and an email client, and you have a full Linux system. Individuals and even companies started bundling all this software into distributions almost as soon as Linux became usable. The distribution includes tools that take 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.

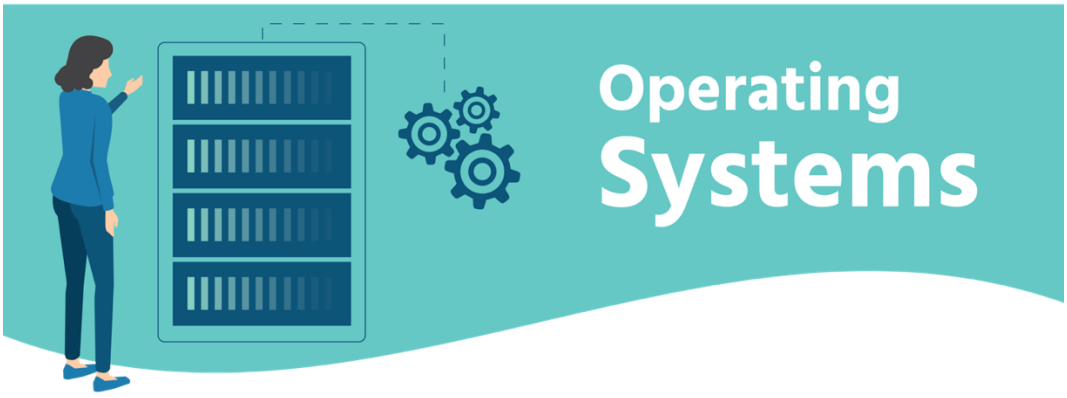


The second type of interface is the *command line interface (CLI)*, a text-based interface to the computer. The CLI relies primarily on keyboard input. Everything the user wants the computer to do is relayed by typing commands rather than clicking on icons. It can be said that when a user clicks on an 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.

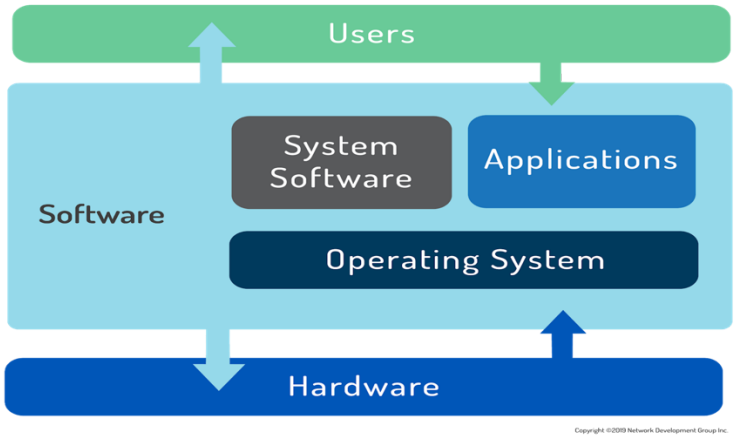
Furthermore, by learning the CLI, a user can easily be productive almost instantly on ANY distribution of Linux, reducing the amount of time needed to familiarize themselves with a system because of variations in a GUI.



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.



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 operating systems 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 10. New versions of this OS come out every 3-5 years and tend to be supported for many years. 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. 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

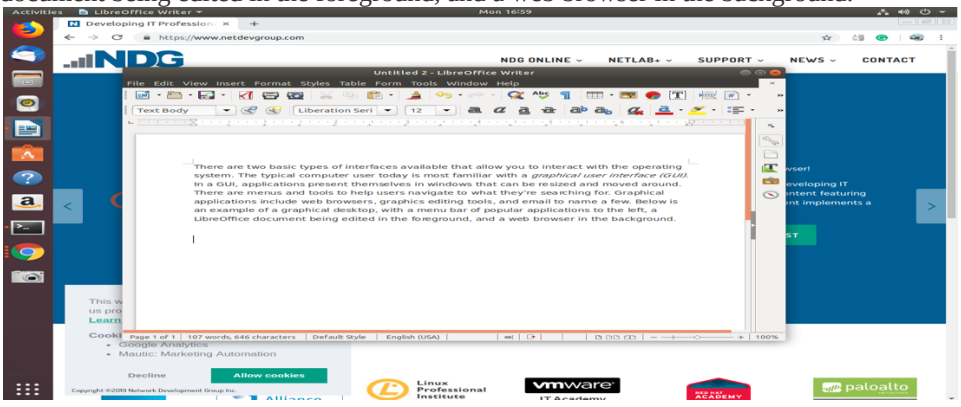
The Debian distribution warns users about the pitfalls of using the “sid” (unstable) release with the following warning:
‘*"sid" is subject to massive changes and in-place library updates. This can result in a very "unstable" system which contains packages that cannot be installed due to missing libraries, 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 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

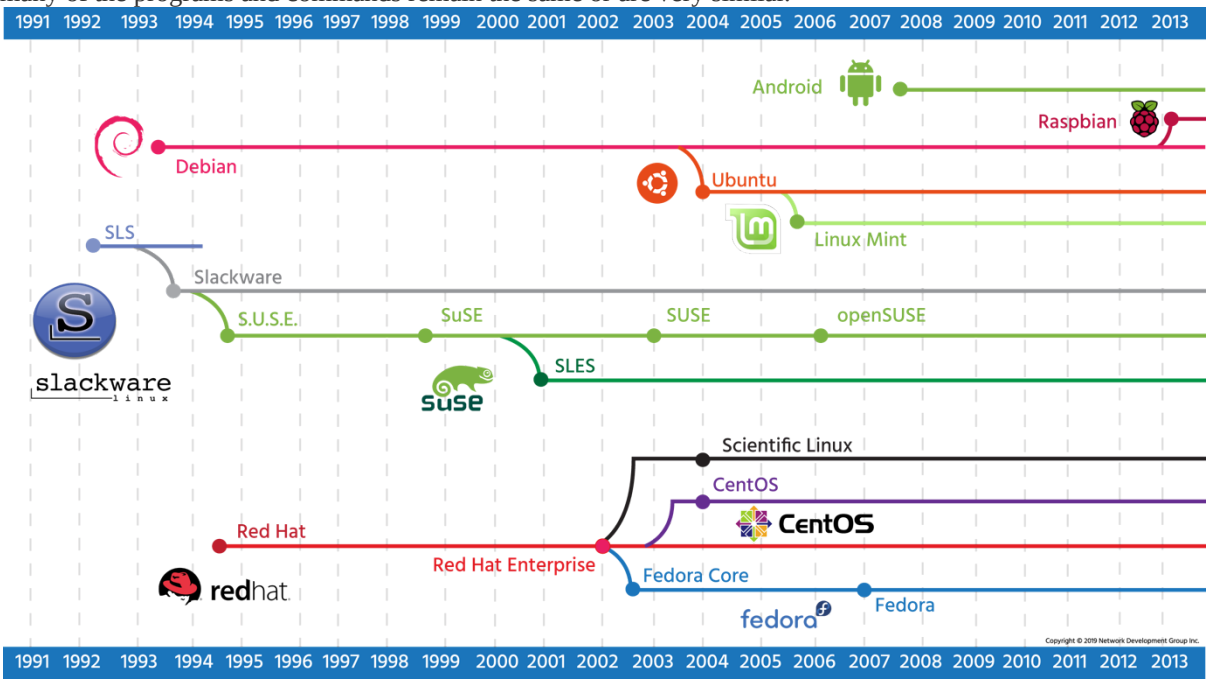
Debian is more of a community effort, and as such, also promotes the use of open source software and adherence to standards. Debian came up with its own package management system based on the .deb file format. While Red Hat leaves non-Intel and AMD platform support to derivative projects, Debian supports 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.



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.

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.

Most Linux distributions allow users to download a “desktop” installation package that can be loaded onto a USB key. This is one of the first things aspiring system administrators should do; download a major distribution and load it onto an old PC. This process is fairly straightforward, and tutorials are available online. The Linux desktop should be familiar to anyone who has used a PC or Macintosh with icons to select different programs and a “settings” application to configure things like user accounts, WiFi networks, and input devices. After familiarizing oneself with the *Linux Graphical User Interface (GUI)*, or desktop, the next step is learning how to perform tasks from the command line.

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.

On systems that boot to a GUI, there are two common ways of accessing the command line—a GUI-based terminal, and a virtual terminal:

- A GUI terminal is a program within the GUI environment that emulates a terminal window. GUI terminals can be accessed through the menu system. For example, on a CentOS machine, you could click on Applications on the menu bar, then System Tools > and, finally, Terminal. If you have a search tools, you can search for terminal, as shown here.

- A virtual terminal can be run at the same time as a GUI but requires the user to log in via the virtual terminal before they can execute commands (as they would before accessing the GUI interface).

Each Linux desktop distribution is slightly different, but the application terminal or x-term will open a terminal window from the GUI. While there are subtle differences between the terms *console* and *terminal* window sessions, they are all the same from an administrators standpoint and require the same knowledge of commands to use.

Ordinary command line tasks are starting programs, parsing scripts, and editing text files used for system or application configuration. Most servers boot directly to a terminal, as a GUI can be resource intensive and is generally not needed to perform server-based operations.

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.

One resulting advantage is that Linux can simulate almost all aspects of a production environment, from development to testing, to verification on scaled-down hardware, which saves costs and time. A Linux administrator could run the same server applications on a desktop or inexpensive virtual server that are run by large internet service providers. Of course, a desktop would not be able to handle the same volume as a major provider would, but almost any configuration can be simulated without needing powerful hardware or server licensing.

Linux software generally falls into one of three categories:

- Server Applications
  Software that has no direct interaction with the monitor and keyboard of the machine it runs on. Its purpose is to serve information to other computers, called *clients*. Sometimes server applications may not talk to other computers but only sit there and crunch data.
- Desktop Applications
  Web browsers, text editors, music players, or other applications with which users interact directly. In many cases, such as a web browser, the application is talking to a server on the other end and interpreting the data. This is the "client" side of a client/server application.
- Tools
  A loose category of software that exists to make it easier to manage computer systems. Tools can help configure displays, provide a Linux shell that users type commands into, or even more sophisticated tools, called compilers, that convert source code to application programs that the computer can execute.

The availability of applications varies depending on the distribution. 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. Some applications, however, like Firefox and LibreOffice are widely supported and available for all major distributions.

The Linux community has come up with lots of creative solutions for both desktop and server applications. These applications, many of which make up the backbone of the Internet, are critical to understanding, and utilizing the power of Linux.

Most computing tasks can be accomplished by any number of applications in Linux. There are many web browsers, web servers, database servers, and text editors from which to choose. Evaluating application software is an important skill to be learned by the aspiring Linux administrator. Determining requirements for performance, stability, and cost are just some of the considerations needed for a comprehensive analysis.

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.

The ownCloud project was launched in 2010 by Frank Karlitschek to provide software to store, sync and share data from private cloud servers. It is available in a standard open source GNU AGPLv3 license and an enterprise version that carries a commercial license.

The Nextcloud project was forked from ownCloud in 2016 by Karlitschek and has been growing steadily since then. It is provided under a GNU AGPLv3 and aims for “an open, transparent development process.”

Both projects focus on providing private cloud software that meets the needs of both large and small organizations that require security, privacy, and regulatory compliance. While several other projects aim to serve the same users, these two are by far the largest in terms of both deployment and project members.

3.2.2.3 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 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

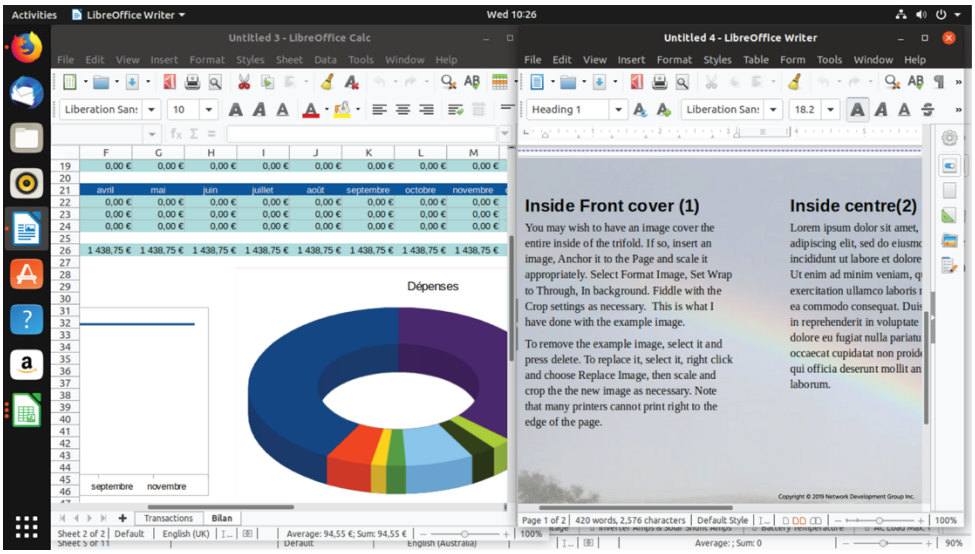
The Mozilla Foundation came out with Thunderbird, a full-featured desktop email client. Thunderbird connects to a POP or IMAP server, displays email locally, and sends email through an external SMTP server.
Other notable email clients are Evolution and KMail which are the GNOME and KDE projects' email 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

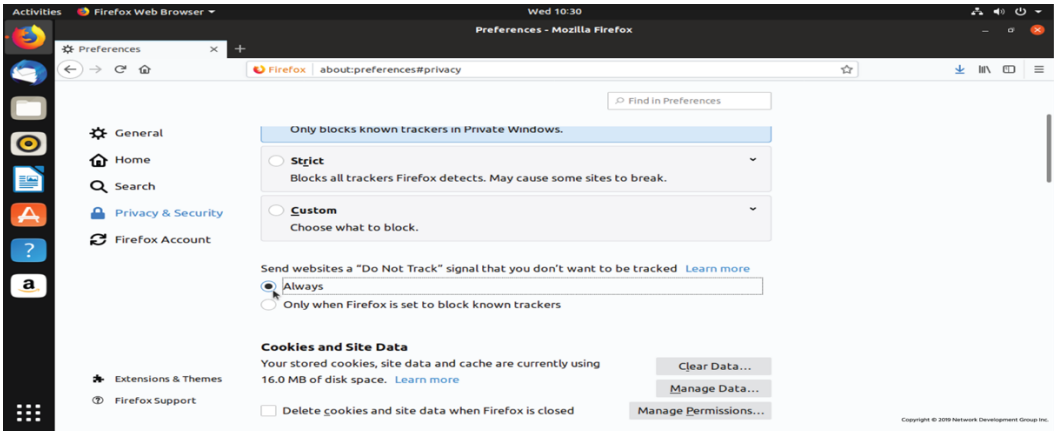
The term *object-oriented* refers to programing that abstracts complex actions and processes so that the end user only deals with basic tasks. To visualize this concept, think of a machine that performs a complex set of tasks by simply pushing a button.

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.

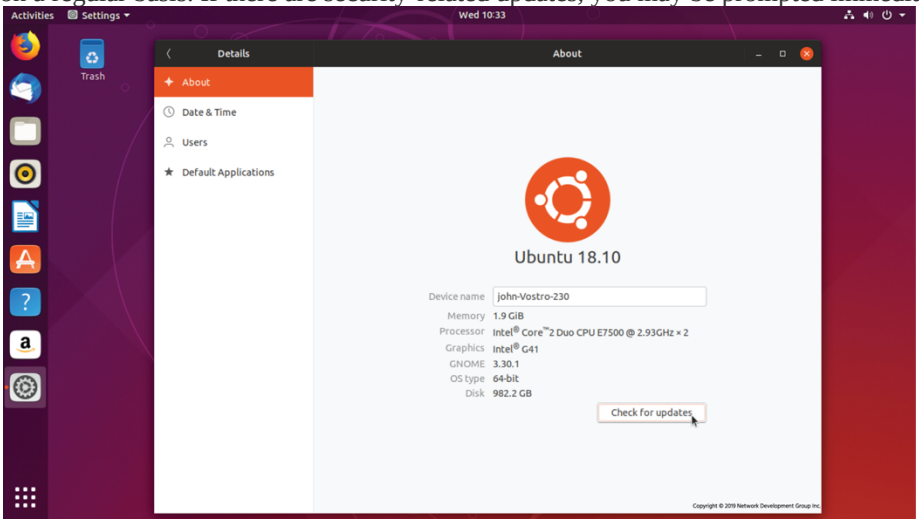
3.6.1 Password Issues

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.

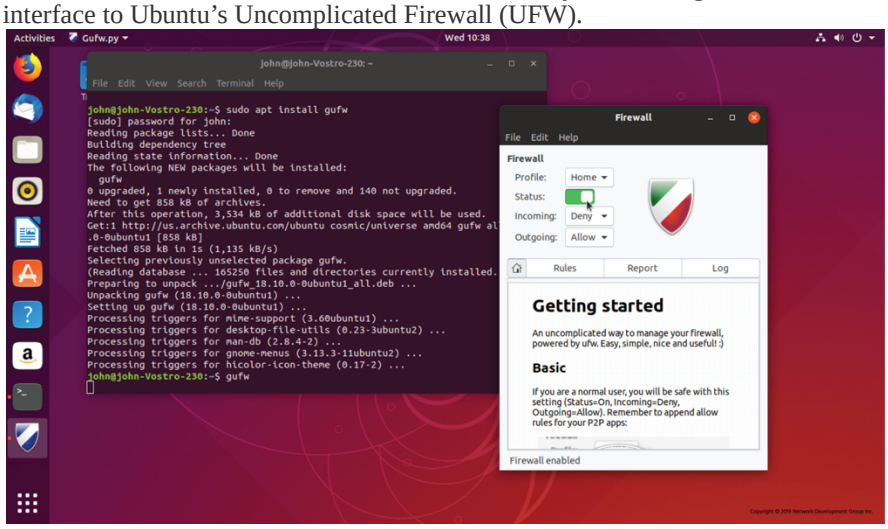
3.6.2 Protecting Yourself

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.



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 from companies like VMWare and Openbox, 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.

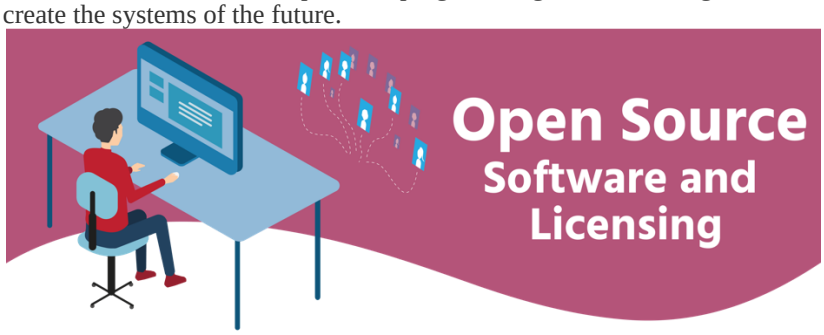
In a virtualized environment one host can run dozens of guest operating systems, and with support from the CPU itself, the guests don't even know they are running on a virtual machine. Each guest gets its own virtual resources and communicates with the network on its own. It is not even necessary to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



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.

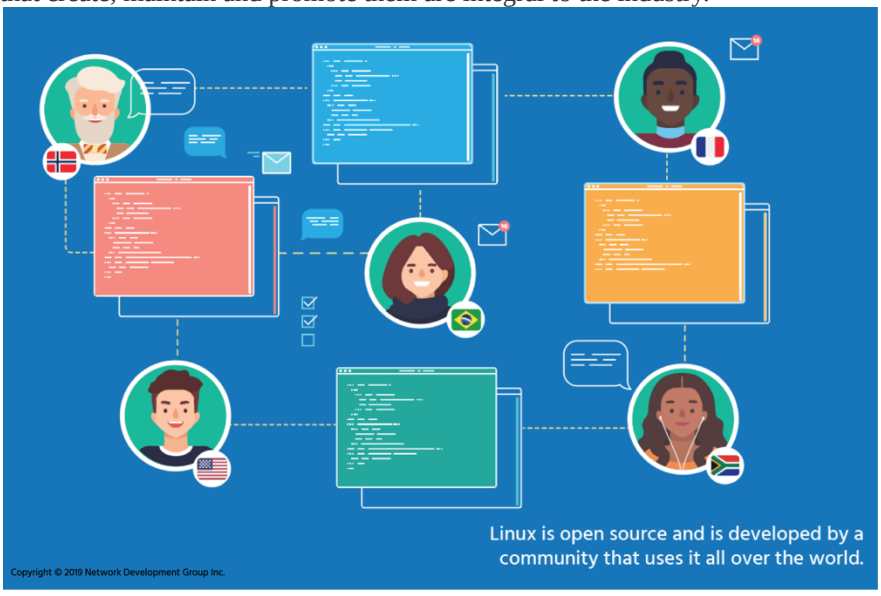
There are many different variants of open source, but all agree that users should have access to the source code. Where they differ is in how one can, or must, redistribute changes.

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.



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.

5.1 Introduction

Most consumer operating systems are designed to shield the user from the ins and outs 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.

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

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

By understanding the foundation of Linux, you have the ability to work on ANY Linux distribution. This could mean one company with a mixed environment or a new company with a different Linux distribution.

5.2 Shell

Once a user has entered a command the terminal then accepts what the user has typed and passes it to a *shell*. The shell is the command line interpreter that translates commands entered by a user into actions to be performed by the operating system. If output is produced by the command, then text is displayed in the terminal. If problems with the command are encountered, an error message is displayed.

The Linux environment allows the use of many different shells, some of which have been around for many years. The most commonly-used shell for Linux distributions is called the Bash shell. Bash provides many advanced features, such as command history and inline editing, which allows a user to easily re-execute previously executed commands or a variation of them via simple editing.

The Bash shell also has other popular features, a few of which are listed below:

- Scripting: The ability to place commands in a file and then interpret (effectively use Bash to execute the contents of) the file, resulting in all of the commands being executed. This feature also has some programming features, such as conditional statements and the ability to create functions (AKA subroutines).
- Aliases: The ability to create short nicknames for longer commands.
- Variables: Used to store information for the Bash shell and for the user. These variables can be used to modify how commands and features work as well as provide vital system information.

Bash has an extensive feature list; this is only a sampling of its capabilities.

When a terminal application is run, and a shell appears, displaying an important part of the interface—the *prompt*. Not only is the prompt there to indicate that commands can be run, but it also conveys useful information to the user. The prompt is fully configurable and can be as sparse or as full-featured as is practical and useful.

The structure of the prompt may vary between distributions, but typically contains information about the user and the system. Below is a common prompt structure:

```
sysadmin@localhost:~$
```

The prompt shown contains the following information:

- User Name:

  ```
  sysadmin@localhost:~$
  ```

- System Name:

```
sysadmin@localhost:~$
```

- Current Directory:

```
sysadmin@localhost:~$
```

The ~ symbol is used as shorthand for the user's home directory. Typically the home directory for the user is under the /home directory and named after the user account name; for example, /home/sysadmin.

5.3 Commands

What is a command? The simplest answer is that a *command* is a software program that, when executed on the CLI, performs an action on the computer.

To execute a command, the first step is to type the name of the command. Click in the terminal on the right. Type ls and hit Enter. The result should resemble the example below:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

Note: By itself, the ls command lists files and directories contained in the current working directory. At this point, you shouldn't worry too much about the output of the command, instead, focus on understanding how to format and execute commands.

The ls command will be covered in complete detail later in the course.

Many commands can be used by themselves with no further input. Some commands require additional input to run correctly. This additional input comes in two forms: *options* and *arguments*.

The typical format for a command is as follows:

```
command [options] [arguments]
```

Options are used to modify the core behavior of a command while arguments are used to provide additional information (such as a filename or a username). Each option and argument is normally separated by a space, although options can often be combined.

Keep in mind that Linux is case-sensitive. Commands, options, arguments, variables, and file names must be entered exactly as shown.

5.3.1 Arguments

```
command [options] [arguments]
```

An argument can be used to specify something for the command to act upon. If the ls command is given the name of a directory as an argument, it lists the contents of that directory. In the following example, the /etc/ppp directory is used as an argument; the resulting output is a list of files contained in that directory:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d  ip-up.d
```

The ls command also accepts multiple arguments. To list the contents of both the /etc/ppp and /etc/ssh directories, pass them both as arguments:

```
sysadmin@localhost:~$ ls /etc/ppp /etc/ssh
/etc/ppp:
ip-down.d  ip-up.d
/etc/ssh:
moduli              ssh_host_dsa_key.pub    ssh_host_rsa_key      sshd_configssh_config
ssh_host_ecdsa_key  ssh_host_rsa_key.pub
ssh_host_dsa_key    ssh_host_ecdsa_key.pub  ssh_import_id
```

5.3.2 Options

```
command [options] [arguments]
```

Options can be used with commands to expand or modify the way a command behaves. For example, using the -l option of the ls command results in a *long listing*, providing additional information about the files that are listed, such as the permissions, the size of the file and other information:

```
sysadmin@localhost:~$ ls -l
total 0
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Desktop
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Documents
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Downloads
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Music
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Pictures
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Public
```

```
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Templates
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  20:13 Videos
```

Often the character is chosen to be mnemonic for its purpose, like choosing the letter *l* for *long* or *r* for *reverse*. By default, the ls command prints the results in alphabetical order, and so by adding the -r option, it prints the results in reverse alphabetical order.

```
sysadmin@localhost:~$ ls -r
Videos  Templates  Public  Pictures  Music  Downloads  Documents  Desktop
```

In most cases, options can be used in conjunction with other options. They can be given as separate options, as in -l -r, or combined, as in -lr. The combination of these two options would result in a long listing output in reverse alphabetical order:

```
sysadmin@localhost:~$ ls -lr
total 32
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Videos
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Downloads
drwxr-xr-x 4 sysadmin sysadmin 4096 Oct 31 20:13 Documents
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Desktop
```

The order of the combined options isn't important. The output of all of these examples would be the same:

```
ls -l -r
ls -rl
ls -lr
```

By default the -l option of the ls command displays files sizes in bytes:

```
sysadmin@localhost:~$ ls -l /usr/bin/perl
-rwxr-xr-x 2 root root 10376 Feb  4  2018 /usr/bin/perl
```

If the -h option is added the file sizes will be displayed in *human-readable* format:

```
sysadmin@localhost:~$ ls -lh /usr/bin/perl
-rwxr-xr-x 2 root root 11K Feb  4  2018 /usr/bin/perl
```

Options are often single letters; however, sometimes they are words or phrases as well. Typically, older commands use single letters while newer commands use complete words for options. Single-letter options are preceded by a single dash - character, like the -l option. Full-word options are preceded by two dash -- characters. The -h option also has an equivalent full-word form; the --human-readable option.

```
sysadmin@localhost:~$ ls -l --human-readable /usr/bin/perl
-rwxr-xr-x 2 root root 11K Feb  4  2018 /usr/bin/perl
```

5.3.3 History

When a command is executed in the terminal, it is stored in a history list. This is designed to make it easy to execute the same command, later eliminating the need to retype the entire command.

Pressing the Up Arrow ↑ key displays the previous command on the prompt line. The entire history of commands run in the current session can be displayed by pressing Up repeatedly to move back through the history of commands that have been run. Pressing the Enter key runs the displayed command again.

When the desired command is located, the Left Arrow ← and Right Arrow → keys can position the cursor for editing. Other useful keys for editing include the Home, End, Backspace and Delete keys.

To view the history list of a terminal, use the history command:

```
sysadmin@localhost:~$ date
Wed Dec 12 04:28:12 UTC 2018
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cal 5 2030
     May 2030
Su Mo Tu We Th Fr Sa
        1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

```
26 27 28 29 30 31
sysadmin@localhost:~$ history
    1  date
    2  ls
    3  cal 5 2030
    4  history
```

If the desired command is in the list that the history command generates, it can be executed by typing an exclamation point ! character and then the number next to the command, for example, to execute the cal command again:

```
sysadmin@localhost:~$ history
    1  date
    2  ls
    3  cal 5 2030
    4  history
sysadmin@localhost:~$ !3
cal 5 2030
     May 2030
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

If the history command is passed a number as an argument, it outputs that number of previous commands from the history list. For example, to show the last three commands:

```
sysadmin@localhost:~$ history 3
    6  date
    7  ls /home
    8  history 3
```

To execute the *nth* command from the bottom of the history list, type !-n and hit Enter. For example, to execute the third command from the bottom of the history list execute the following:

```
sysadmin@localhost:~$ !-3
date
Wed Dec 12 04:31:55 UTC 2018
```

To execute the most recent command type !! and hit Enter:

```
sysadmin@localhost:~$ date
Wed Dec 12 04:32:36 UTC 2018
sysadmin@localhost:~$ !!
date
Wed Dec 12 04:32:38 UTC 2018
```

To execute the most recent iteration of a specific command, type ! followed by the name of the command and hit Enter. For example, to execute the most recent ls command:

```
sysadmin@localhost:~$ !ls
ls /home
sysadmin
```

5.4 Variables

A variable is a feature that allows the user or the shell to store data. This data can be used to provide critical system information or to change the behavior of how the Bash shell (or other commands) work. Variables are given names and stored temporarily in memory. There are two types of variables used in the Bash shell: *local* and *environment*.

5.4.1 Local Variables

Local or *shell variables* exist only in the current shell, and cannot affect other commands or applications. When the user closes a terminal window or shell, all of the variables are lost. They are often associated with user-based tasks and are lowercase by convention.

To set the value of a variable, use the following assignment expression. If the variable already exists, the value of the variable is modified. If the variable name does not already exist, the shell creates a new local variable and sets the value:

```
variable=value
```

The following example creates a local variable named variable1 and assigns it a value of Something:

```
sysadmin@localhost:~$ variable1='Something'
```

The echo command is used to display output in the terminal. To display the value of the variable, use a dollar sign $ character followed by the variable name as an argument to the echo command:

```
sysadmin@localhost:~$ echo $variable1
Something
```

5.4.2 Environment Variables

Environment variables, also called *global variables*, are available system-wide, in all shells used by Bash when interpreting commands and performing tasks. The system automatically recreates environment variables when a new shell is opened. Examples include the PATH, HOME, and HISTSIZE variables. The HISTSIZE variable defines how many previous commands to store in the history list. The command in the example below displays the value of the HISTSIZE variable:

```
sysadmin@localhost:~$ echo $HISTSIZE
1000
```

To modify the value of an existing variable, use the assignment expression:

```
sysadmin@localhost:~$ HISTSIZE=500
sysadmin@localhost:~$ echo $HISTSIZE
500
```

Many variables are available for the Bash shell, as well as variables that affect different Linux commands. A discussion of all variables is beyond the scope of this chapter; however, more shell variables will be covered as this course progresses.

When run without arguments, the env command outputs a list of the environment variables. Because the output of the env command can be quite long, the following examples use a text search to filter that output.

In a previous example variable1 was created as a local variable, so the following search in the environment variables results in no output:

```
sysadmin@localhost:~$ env | grep variable1
```

The pipe | character passes the output of the env command to the grep command, which searches the output. *This text filtering technique will be covered in detail later in the course.*

The export command is used to turn a local variable into an environment variable.

```
export variable
```

After exporting variable1, it is now an environment variable. It is now found in the search through the environment variables:

```
sysadmin@localhost:~$ export variable1
sysadmin@localhost:~$ env | grep variable1
variable1=Something
```

The export command can also be used to make a variable an environment variable upon its creation by using the assignment expression as the argument:

```
sysadmin@localhost:~$ export variable2='Else'
sysadmin@localhost:~$ env | grep variable2
variable2=Else
```

To change the value of an environment variable, use the assignment expression:

```
sysadmin@localhost:~$ variable1=$variable1' '$variable2
sysadmin@localhost:~$ echo $variable1
Something Else
```

Exported variables can be removed using the unset command:

```
sysadmin@localhost:~$ unset variable2
```

5.4.2 Environment Variables

Environment variables, also called *global variables*, are available system-wide, in all shells used by Bash when interpreting commands and performing tasks. The system automatically recreates environment variables when a new shell is opened. Examples include the PATH, HOME, and HISTSIZE variables. The HISTSIZE variable defines how many previous commands to store in the history list. The command in the example below displays the value of the HISTSIZE variable:

```
sysadmin@localhost:~$ echo $HISTSIZE
1000
```

To modify the value of an existing variable, use the assignment expression:

```
sysadmin@localhost:~$ HISTSIZE=500
sysadmin@localhost:~$ echo $HISTSIZE
500
```

Many variables are available for the Bash shell, as well as variables that affect different Linux commands. A discussion of all variables is beyond the scope of this chapter; however, more shell variables will be covered as this course progresses.

When run without arguments, the env command outputs a list of the environment variables. Because the output of the env command can be quite long, the following examples use a text search to filter that output.

In a previous example variable1 was created as a local variable, so the following search in the environment variables results in no output:

```
sysadmin@localhost:~$ env | grep variable1
```

The pipe | character passes the output of the env command to the grep command, which searches the output. *This text filtering technique will be covered in detail later in the course.*

The export command is used to turn a local variable into an environment variable.

```
export variable
```

After exporting variable1, it is now an environment variable. It is now found in the search through the environment variables:

```
sysadmin@localhost:~$ export variable1
sysadmin@localhost:~$ env | grep variable1
variable1=Something
```

The export command can also be used to make a variable an environment variable upon its creation by using the assignment expression as the argument:

```
sysadmin@localhost:~$ export variable2='Else'
sysadmin@localhost:~$ env | grep variable2
variable2=Else
```

To change the value of an environment variable, use the assignment expression:

```
sysadmin@localhost:~$ variable1=$variable1' '$variable2
sysadmin@localhost:~$ echo $variable1
Something Else
```

Exported variables can be removed using the unset command:

```
sysadmin@localhost:~$ unset variable2
```

5.4.3 Path Variable

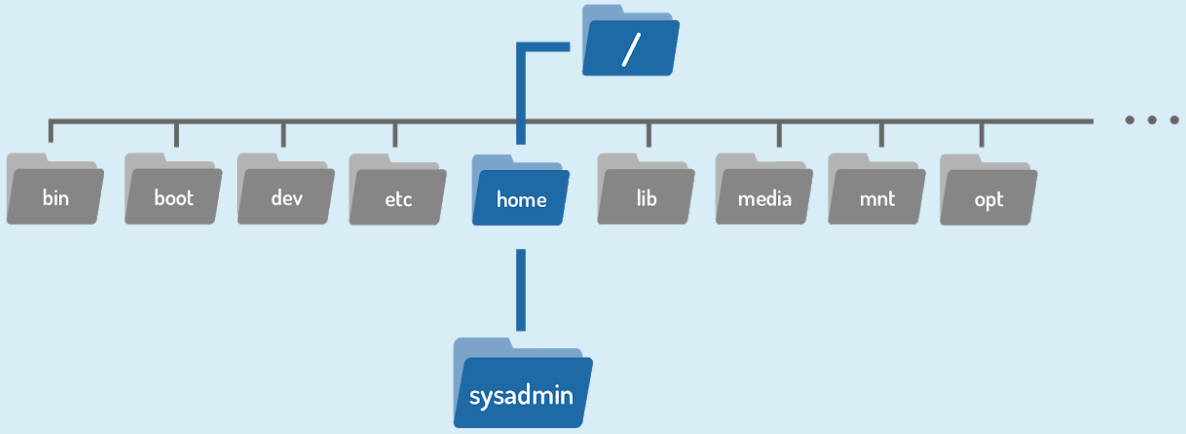
One of the most important Bash shell variables to understand is the PATH variable. It contains a list that defines which directories the shell looks in to find commands. If a valid command is entered and the shell returns a "command not found" error, it is because the Bash shell was unable to locate 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 $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
sysadmin@localhost:~$
```

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
```

Each of these directories is represented by a *path*. A path is a list of directories separated by the / character. If you think of the filesystem as a map, paths are the directory addresses, which include step-by-step navigation directions; they can be used to indicate the location of any file within the filesystem. For example, /home/sysadmin is a path to the home directory:



Directories and paths will be covered in detail later in the course.

If the command is not found in any directory listed in the PATH variable, then the shell returns an error:

```
sysadmin@localhost:~$ zed
-bash: zed: command not found
sysadmin@localhost:~$
```

If custom software is installed on the system it may be necessary to modify the PATH to make it easier to execute these commands. For example, the following will add and verify the /usr/bin/custom directory to the PATH variable:

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

When updating the PATH variable, always include the current path, so as not to lose access to commands located in those directories. This can be accomplished by appending $PATH to the value in the assignment expression. Recall that a variable name preceded by a dollar sign represents the value of the variable.

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
```

There are several different sources of commands within the shell of your CLI including *internal commands*, *external commands*, *aliases*, and *functions*

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 stored in files 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.

If a command does not behave as expected or if a command is not accessible that should be, it can be beneficial to know where the shell is finding the command or which version it is using. It would be tedious to have to manually look in each directory that is listed in the PATH variable. Instead, use the which command to display the full path to the command in question:

```
which command
```

The 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
```

External commands can also be executed by typing the complete path to the command. For example, to execute the ls command:

```
sysadmin@localhost:~$ /bin/ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

For external commands, the type command displays the location of the command:

```
sysadmin@localhost:~$ type cal
cal is /usr/bin/cal
```

In some cases the output of the type command may differ significantly from the output of the which command:

```
sysadmin@localhost:~$ type echo
echo is a shell builtin
sysadmin@localhost:~$ which echo
/bin/echo
```

Using the -a option of the type command displays all locations that contain the command named:

```
sysadmin@localhost:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
```

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
```

For example, the cal 2019 command displays the calendar for the year 2019. Suppose you end up running this command often. Instead of executing the full command each time, you can create an alias called mycal and run the alias, as demonstrated in the following graphic:

```
sysadmin@localhost:~$ alias mycal="cal 2019"
sysadmin@localhost:~$ mycal
                              2019
      January               February               March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
       1  2  3  4  5                  1  2                  1  2
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   3  4  5  6  7  8  9
13 14 15 16 17 18 19  10 11 12 13 14 15 16  10 11 12 13 14 15 16
20 21 22 23 24 25 26  17 18 19 20 21 22 23  17 18 19 20 21 22 23
27 28 29 30 31        24 25 26 27 28        24 25 26 27 28 29 30
                                            31
```

Aliases created this way only persist while the shell is open. Once the shell is closed, the new aliases are lost. Additionally, each shell has its own aliases, so aliases created in one shell won't be available in a new shell that's opened.

The type command can identify aliases to other commands:

```
sysadmin@localhost:~$ type ll
ll is aliased to `ls -alF'
sysadmin@localhost:~$ type -a ls
ls is aliased to `ls --color=auto'
ls is /bin/ls
```

The output of these commands indicates that ll is an alias for ls -alF, and even ls is an alias for ls --color=auto.

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.

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 `. Each set of quotes alerts the shell not to treat the text within the quotes in the normal way.

5.6.1 Double Quotes

Double quotes stop the shell from interpreting some metacharacters (special characters), including glob characters.

Glob characters, also called *wild cards*, are symbols that have special meaning to the shell; they are interpreted by the shell itself before it attempts to run any command. Glob characters include the asterisk * character, the question ? mark character, and the brackets [ ], among others.

Globbing will be covered in greater detail later in the course.

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 [ ]
```

Double quotes still allow for *command substitution*, *variable substitution*, and permit some other shell metacharacters that haven't been discussed yet. 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/local/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 $100
The car costs 00
sysadmin@localhost:~$ echo 'The car costs $100'
The car costs $100
```

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
```

If this sentence is placed in double quotes, $1 and $PATH are considered variables.

```
sysadmin@localhost:~$ echo "The service costs $1 and the path is $PATH"

The service costs  and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

If it is placed in single quotes, $1 and $PATH are not considered variables.

```
sysadmin@localhost:~$ echo 'The service costs $1 and the path is $PATH'
```

```
The service costs $100 and the path is $PATH
```

But what if you want to have $PATH treated as a variable and $1 not?
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 $PATH
The service costs $1 and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

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.
While it may sound confusing, an example should make things more clear. To begin, note the output of the date command:

```
sysadmin@localhost:~$ date
Mon Nov  4 03:35:50 UTC 2018
```

Now, note the output of the echo command:

```
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.
For example, to print the months of January, February and March of 2030, execute the following command:

```
sysadmin@localhost:~$ cal 1 2030; cal 2 2030; cal 3 2030
    January 2030
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

   February 2030
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28

     March 2030
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

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.
With the double pipe, if the first command runs successfully, the second command is skipped; if the first command fails, then the second command is run. In other words, you are essentially telling the shell, "Either run this first command or the second one".
In the following example, the echo command only executes if the ls command fails:

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

LAB-5:

5.1 Introduction

This is Lab 5: Command Line Skills. By performing this lab, students will learn how to use basic features of the shell.
In this lab, you will perform the following tasks:

- Explore Bash features
- Use shell variables
- Be able to make use of quoting

5.2 Files and Directories

In this task, we will access the Command Line Interface (CLI) for Linux to explore how to execute basic commands and what affects how they can be executed.
Most users are probably more familiar with how commands are executed using a Graphical User Interface (GUI). Therefore, this task will likely present some new concepts to you if you have not previously worked with a CLI. To use a CLI, you will need to type the command that you want to run.
The window where you will type your command is known as a terminal emulator application. Inside of the Terminal window the system is displaying a prompt, which currently contains a prompt followed by a blinking cursor:

```
sysadmin@localhost:~$
```

Remember
You may need to press Enter in the window to display the prompt.
The prompt tells you that you are user sysadmin; the host or computer you are using: localhost; and the directory where you are at: ~, which represents your home directory.
When you type a command, it will appear at the text cursor. You can use keys such as the home, end, backspace, and arrow keys for editing the command you are typing.
Equally important is the command line syntax, which is the order in which the command, the option(s), and argument(s) must be entered into the prompt so the shell recognizes how to properly execute the command.
Proper command line syntax looks like the following:

```
command [options] [arguments]
```

Type the ls command at the prompt, which will list the files and directories contained in your current working directory. In this example, no options or arguments will be used.
Once you have typed the command correctly, press Enter to execute it.

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

5.2.1 Step 1

The ls command is used to list information about directories and files and by default it displays information for the current directory. Use the -l option to display this information in the long format, which gives additional information about files located in the current working directory:

```
ls -l
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -l
total 32
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Desktop
drwxr-xr-x 4 sysadmin sysadmin 4096 Oct 31 19:52 Documents
```

```
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Downloads
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Videos
```

Note that directories are considered a type of file in the Linux file system.

5.2.2 Step 2

Arguments can be added to commands as well. Adding the location of a specific directory to the ls command will list information for that directory. Use the argument /tmp to display detailed information about files in the /tmp directory.

```
ls -l /tmp
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -l /tmp
total 4
-rw-rw-r-- 1 root root 1863 Oct 31 19:47 inside_setup.sh
```

By using the option l and the argument /tmp, we can now see that the /tmp directory has a file called inside_setup.sh located inside it.

5.2.3 Step 3

The following command will display the same information that you see in the first part of the prompt. Make sure that you have selected (clicked on) the Terminal window first and then type the following command followed by the Enter key:

```
whoami
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ whoami
sysadmin
sysadmin@localhost:~$
```

The output of the whoami command, sysadmin, displays the user name of the current user. Although in this case your username is displayed in the prompt, this command could be used to obtain this information in a situation when the prompt did not contain this information.

5.2.4 Step 4

The next command displays information about the current system. To be able to see the name of the kernel you are using, type the following command into the terminal:

```
uname
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ uname
Linux
```

Many commands that are executed produce text output like this. You can change what output is produced by a command by using *options* after the name of the command.

Options for a command can be specified in several ways. Traditionally in UNIX, options were expressed by a hyphen followed by another character; for example: -n.

In Linux, options can sometimes also be given by two hyphen characters followed by a word, or hyphenated word; for example: --nodename.

Execute the uname command again twice in the terminal, once with the option -n and again with the option --nodename. This will display the network node hostname, also found in the prompt.

```
uname -n
uname --nodename
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ uname -n
localhost
sysadmin@localhost:~$ uname --nodename
localhost
```

5.2.5 Step 5

The pwd command is used to display your current "location" or current "working" directory. Type the following command to display the working directory:

```
pwd
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

The current directory in the example above is /home/sysadmin. This is also referred to as your *home* directory, a special place where you have control of files and other users normally have no access. By default, this directory is named the same as your username and is located underneath the /home directory.
As you can see from the output of the command, /home/sysadmin, Linux uses the forward slash / to separate directories to make what is called a *path*. The initial forward slash represents the top-level directory, known as the root directory. More information regarding files, directories and paths will be presented in later labs.
The tilde ~ character that you see in your prompt is also indicating what the current directory is. This character is a "shortcut" way to represent your home.

Consider This
pwd stands for "print working directory". While it doesn't actually "print" in modern versions, older UNIX machines didn't have monitors so the output of commands went to a printer, hence the somewhat misleading name of pwd.

5.3 Command History
The Bash shell maintains a history of the commands that you type. Previous commands can be easily accessed in this history in several ways.
The first and easiest way to recall a previous command is to use the up arrow key. Each press of the up arrow key goes backwards one command through your command history. If you accidentally go back too far, then the down arrow key will go forwards through the history of commands.
When you find the command that you want to execute, you can use the left arrow keys and right arrow keys to position the cursor for editing. Other useful keys for editing include the Home, End, Backspace and Delete keys.
Another way to use your command history is to execute the history command to be able to view a *numbered* history list. The number listed to the left of the command can be used to execute the command again. The history command also has a number of options and arguments which can manipulate which commands will be stored or displayed.

5.3.1 Step 1
Execute a new command and then execute the history command:

```
echo Hi
history
```

Remember
The date command will print the time and date on the system. The clear command clears the screen.

Your output should be similar to the following:

```
sysadmin@localhost:~$ history
    1  ls
    2  ls -l
    3  ls -l /tmp
    4  whoami
    5  uname
    6  uname -n
    7  uname --nodename
    8  pwd
    9  echo Hi
   10  history
sysadmin@localhost:~$
```

Your command numbers may differ from those provided above. This is because you may have executed a different number of commands since opening the virtual terminal.

5.3.2 Step 2
To view a limited number of commands, the history command can take a number as a parameter to display exactly that many recent entries. Type the following command to display the last five commands from your history:

```
history 5
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ history 5
    7  uname --nodename
    8  pwd
    9  echo Hi
   10  history
   11  history 5
```

5.3.3 Step 3
To execute a command again, type the exclamation point and the history list number. For example, to execute the 9th command in your history list, you would execute the following:

```
!9
```

```
sysadmin@localhost:~$ !9
echo Hi
Hi
```

5.3.4 Step 4
Next, experiment with accessing your history using the up arrow keys and down arrow keys. Keep pressing the up arrow key until you find a command you want to execute. If necessary, use other keys to edit the command and then press Enter to execute the command.
5.4 Shell Variables
Shell variables are used to store data in Linux. This data is used by the shell itself as well as by programs and users.
The focus of this section is to learn how to display the values of shell variables.
5.4.1 Step 1
The echo command can be used to print text and the value of a variable, and to show how the shell environment expands *metacharacters* (more on metacharacters later in this lab). Type the following command to have it output literal text:

```
echo Hello Student
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo Hello Student
Hello Student
sysadmin@localhost:~$
```

5.4.2 Step 2
Environment variables are available system-wide. The system automatically recreates environment variables when a new shell is opened. Examples include the PATH, HOME, and HISTSIZE variables.
The HISTSIZE variable defines how many previous commands to store in the history list. In the example below, the command will display the value of the HISTSIZE variable:

```
sysadmin@localhost:~$ echo $HISTSIZE
1000
sysadmin@localhost:~$
```

5.4.3 Step 3
Type the following command to display the value of the PATH variable:

```
echo $PATH
```

Your output should be similar to the following:

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

The PATH variable is displayed by placing a $ character in front of the name of the variable.
This variable is used to find the location of commands. Each of the directories listed above are searched when you run a command. For example, if you try to run the date command, the shell will first look for the command in the /home/sysadmin/bin directory and then in the /usr/local/sbin directory and so on. Once the date command is found, the shell "runs it".
5.4.4 Step 4
Use the which command to determine if there is an executable file, in this case named date, that is located within a directory listed in the PATH value:

```
which date
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ which date
/bin/date
sysadmin@localhost:~$
```

The output of the `which` command tells you that when you execute the `date` command, the system will run the command /bin/date. The `which` command makes use of the PATH variable to determine the location of the `date` command.

5.5 Command Types

In this section we will learn about the four types of commands used in Linux. Understanding where these commands come from and how they differ allows an administrator to manage the system more effectively.

5.5.1 Step 1

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
```

There are several different sources of commands within the shell of your CLI:

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 Step 2

External commands are stored in files 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.

```
type ls
```

```
sysadmin@localhost:~$ type ls
ls is aliased to `ls --color=auto'
```

5.5.3 Step 3

Aliases 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.

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

```
sysadmin@localhost:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
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'
```

5.5.4 Step 4

The final command type is the *executable program*. These commands invoke programs installed on the system which perform specific tasks. When a user types the `vi` command, the shell uses the PATH file to locate and execute the program. Programs like `vi` are available on just about every Linux distribution; other programs, like `vlc` (an open source media player often used on Linux desktops), are installed by users or administrators for a specific purpose and will not be listed in the PATH unless they have been installed separately.

```
type vi
cd /bin
type vlc
cd
```

```
sysadmin@localhost:~$ type vi
vi is /usr/bin/vi
sysadmin@localhost:~$ cd /bin
sysadmin@localhost:/bin$ type vlc
-bash: type: vlc: not found
```

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

5.6 Quoting

There are three types of quotes used by the Bash shell: single quotes ('), double quotes (") and back quotes (`). These quotes have special features in the Bash shell as described below.

To understand single and double quotes, consider that there are times that you don't want the shell to treat some characters as "special". For example, as you learned earlier in this lab, the * character is used as a wildcard. What if you wanted the * character to just mean an asterisk?

Single quotes prevent the shell from "interpreting" or expanding all special characters. Often single quotes are used to protect a string (a sequence of characters) from being changed by the shell, so that the string can be interpreted by a command as a parameter to affect the way the command is executed.

Double quotes stop the expansion of glob characters like the asterisk (*), question mark (?), and square brackets ([ ]). Double quotes do allow for both variable expansion and command substitution (see back quotes) to take place.

Back quotes cause "command substitution" which allows for a command to be executed within the line of another command.

When using quotes, they must be entered in pairs or else the shell will not consider the command complete.

While single quotes are useful for blocking the shell from interpreting one or more characters, the shell also provides a way to block the interpretation of just a single character called "escaping" the character. To "escape" the special meaning of a shell metacharacter, the backslash \ character is used as a prefix to that one character.

5.6.1 Step 1

Execute the following command to use back quotes ` (found under the ~ character on some keyboards) to execute the date command within the line of the echo command:

```
echo Today is `date`
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Mon Dec 3 21:29:45 UTC 2018
```

5.6.2 Step 2

You can also place $(before the command and) after the command to accomplish command substitution:

```
echo Today is $(date)
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo Today is $(date)
Today is Mon Dec 3 21:33:41 UTC 2018
```

Why two different methods that accomplish the same thing? Backquotes look very similar to single quotes, making it harder to "see" what a command is supposed to do. Originally, shells used backquotes; the $(*command*) format was added in a later version of the Bash shell to make the statement more visually clear.

5.6.3 Step 3

If you don't want the backquotes to be used to execute a command, place single quotes around them. Execute the following:

```
echo This is the command '`date`'
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo This is the command '`date`'
This is the command `date`
sysadmin@localhost:~$
```

5.6.4 Step 4

Note that you could also place a backslash character in front of each backquote character. Execute the following:

```
echo This is the command \`date\`
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo This is the command \`date\`
This is the command `date`
sysadmin@localhost:~$
```

5.6.5 Step 5

Double quote characters don't have any effect on backquote characters. The shell will still use them as command substitution. Execute the following to see a demonstration:

```
echo This is the command "`date`"
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo This is the command "`date`"
This is the command Mon Dec  3 21:37:33 UTC 2018
```

5.6.6 Step 6
Double quote characters will have an effect on wildcard characters, disabling their special meaning. Execute the following:

```
echo D*
echo "D*"
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
sysadmin@localhost:~$ echo "D*"
D*
sysadmin@localhost:~$
```

Important
Quoting may seem trivial and weird at the moment, but as you gain more experience working in the command shell, you will discover that having a good understanding of how different quotes work is critical to using the shell.

5.7 Control Statements
Typically, you type a single command and you execute it when you press Enter. The Bash shell offers three different statements that can be used to separate multiple commands typed together.
The simplest separator is the semicolon (;). Using the semicolon between multiple commands allows for them to be executed one right after another, sequentially from left to right.
The && characters create a logical "and" statement. Commands separated by && are conditionally executed. If the command on the left of the && is successful, then the command to the right of the && will also be executed. If the command to the left of the && fails, then the command to the right of the && is not executed.
The || characters create a logical "or" statement, which also causes conditional execution. When commands are separated by ||, then only if the command to the left fails, does the command to the right of the || execute. If the command to the left of the || succeeds, then the command to the right of the || will not execute.
To see how these control statements work, you will be using two special executables: true and false.
The true executable always succeeds when it executes, whereas, the false executable always fails. While this may not provide you with realistic examples of how && and || work, it does provide a means to demonstrate how they

5.7.1 Step 1
Execute the following three commands together separated by semicolons:

```
echo Hello; echo Linux; echo Student
```

As you can see the output shows all three commands executed sequentially:

```
sysadmin@localhost:~$ echo Hello; echo Linux; echo Student
Hello
Linux
Student
sysadmin@localhost:~$
```

work without having to introduce new commands.

5.7.3 Step 3
Next, use logical "and" to separate the commands:

```
echo Start && echo Going && echo Gone
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo Start && echo Going && echo Gone
Start
Going
Gone
sysadmin@localhost:~$
```

Because each echo statement executes correctly, a return value of success is provided, allowing the next statement to also be executed.

5.7.4 Step 4
Use logical "and" with a command that fails as shown below:

```
echo Success && false && echo Bye
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo Success && false && echo Bye
Success
sysadmin@localhost:~$
```

The first echo command succeeds and we see its output. The false command executes with a failure result, so the last echo statement is not executed.
5.7.5 Step 5
The "or" characters separating the following commands demonstrates how the failure before the "or" statement causes the command after it to execute; however, a successful first statement causes the command to not execute:

```
false || echo Fail Or
true || echo Nothing to see here
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ false || echo Fail Or
Fail Or
sysadmin@localhost:~$ true || echo Nothing to see here
sysadmin@localhost:~$
```

6.1 Introduction
There are thousands of commands available with many options, making the command line a powerful tool. However, with this power comes complexity. Complexity, in turn, can create confusion. As a result, knowing how to find help while working in Linux is an essential skill for any user. Referring to help provides a quick reminder of how a command works, as well as being an information resource when learning new commands.
6.2 Man Pages
UNIX is the operating system that Linux was modeled after. The developers of UNIX created help documents called *man pages* (short for *manual pages*).
Man pages are used to describe the features of commands. They provide a basic description of the purpose of the command, as well as details regarding available options.
6.2.1 Viewing Man Pages
To view a man page for a command, use the man command:

```
man command
```

For example, the following displays the man page for the ls command:

```
sysadmin@localhost:~$ man ls
```

Navigate the document using the arrow keys:

```
LS(1)                    User Commands                    LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
```

```
List  information  about  the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
fied.

Mandatory  arguments  to  long  options are mandatory for short options
too.

-a, --all
      do not ignore entries starting with .
```

To exit viewing a man page, use the Q key.

Consider This

The man command uses a *pager* to display documents. Usually, this pager is the less command, but on some distributions, it may be the more command. Both are very similar in how they perform.

To view the various movement commands that are available, use the H key while viewing a man page. This displays a help page.

Pagers will be discussed in more detail later in the course.

6.2.2 Sections Within Man Pages

Man pages are broken into sections. Each section is designed to provide specific information about a command. While there are common sections seen in most man pages, some developers also create sections only available on specific man pages.

The following describes some of the more common sections found in man pages:

NAME

Provides the name of the command and a very brief description.

```
NAME
     ls - list directory contents
```

SYNOPSIS

Provides examples of how the command is executed.

```
SYNOPSIS
     ls [OPTION]... [FILE]...
```

The SYNOPSIS section of a man page can be difficult to understand but is very important because it provides a concise example of how to use the command. For example, consider the SYNOPSIS of the man page for the cal command:

```
SYNOPSIS
    cal [-31jy] [-A number] [-B number] [-d yyyy-mm] [[month] year]
```

The square brackets [ ] are used to indicate that this feature is not required to run the command. For example, [-31jy] means options -3, -1, -j, or -y are available, but not required for the cal command to function properly.

The last set of square brackets [[month] year] demonstrates another feature; it means a year can be specified by itself, but to specify a month the year must also be specified.

Another component of the SYNOPSIS that might cause some confusion can be seen in the man page of the date command:

```
SYNOPSIS
     date [OPTION]... [+FORMAT]
     date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

In this SYNOPSIS there are two syntaxes for the date command. The first one is used to display the date on the system while the second is used to set the date.

The ellipsis ... following [OPTION] indicates that one or more of the items before it may be used.

Additionally, the [-u|--utc|--universal] notation means that either the -u option or the --utc option or the --universal option may be used. Typically this means that all three options really do the same thing, but sometimes the use of the | character is used to indicate that the options can't be used in combination, like a logical “or".

DESCRIPTION

Provides a more detailed description of the command.

```
DESCRIPTION
     List  information  about  the FILEs (the current directory by default).
     Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
     fied.
```

OPTIONS
Lists the options for the command as well as a description of how they are used. Often this information is found in the DESCRIPTION section and not in a separate OPTIONS section.

```
-a, --all
      do not ignore entries starting with .

-A, --almost-all
      do not list implied . and ..

--author
      with -l, print the author of each file

-b, --escape
      print C-style escapes for nongraphic characters

--block-size=SIZE
      scale sizes by SIZE before printing them; e.g., '--block-size=M'
      prints sizes in units of 1,048,576 bytes; see SIZE format below
```

FILES
Lists the files that are associated with the command as well as a description of how they are used. These files may be used to configure the command's more advanced features. Often this information is found in the DESCRIPTION section and not in a separate FILES section.
AUTHOR
Provides the name of the person who created the man page and (sometimes) how to contact the person.

```
AUTHOR
      Written by Richard M. Stallman and David MacKenzie.
```

REPORTING BUGS
Provides details on how to report problems with the command.

```
REPORTING BUGS
      GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
      Report ls translation bugs to <http://translationproject.org/team/>
```

COPYRIGHT
Provides basic copyright information.

```
COPYRIGHT
      Copyright (C) 2017 Free Software Foundation, Inc.  License GPLv3+:  GNU
      GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
      This  is  free  software:  you  are free to change and redistribute it.
      There is NO WARRANTY, to the extent permitted by law.
```

SEE ALSO
Provides you with an idea of where you can find additional information. This often includes other commands that are related to this command.

```
SEE ALSO
      Full documentation at: <http://www.gnu.org/software/coreutils/ls>
      or available locally via: info '(coreutils) ls invocation'
```

6.2.3 Searching Man Pages
To search a man page for a term, type the / character followed by a search term, then hit the Enter key. The program searches from the current location down towards the bottom of the page to try to locate and highlight the term.
If a match is found, it will be highlighted. To move to the next match of the term, press n. To return to a previous match of the term, press Shift+N. If the term is not found, or upon reaching the end of the matches, the program will report Pattern not found (press Return).

6.2.4 Man Pages Categorized by Sections
Until now, we have been displaying man pages for commands. However, there are several different types of commands (user commands, system commands, and administration commands), configuration files and other features, such as libraries and kernel components, that require documentation.
As a result, 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
```

To determine which section a specific man page belongs to, look at the numeric value on the first line of the output of the man page. For example, the cal command belongs to the first section of man pages:

```
CAL(1)                BSD General Commands Manual            CAL(1)
```

Sometimes there are man pages with the same name in different sections. In these cases, it may be necessary to specify the section of the correct man page.
For example, there is a command called passwd that allows you to change your password. There is also a file called passwd that stores account information. Both the command and the file have a man page.
The passwd command is a user command, so the associated man page is in the first section. The man command displays the man page for the passwd command by default:

```
sysadmin@localhost:~$ man passwd
PASSWD(1)                  User Commands                PASSWD(1)
```

So how do you display the man page for the passwd file? First, 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
```

Note: On most Linux distributions, the whatis command does the same thing as man -f. On those distributions, both will produce the same output.
To specify a different section, provide the number of the section as the first argument of the man command. The following command displays the passwd man page located in section 5, which is associated with the passwd file:

```
sysadmin@localhost:~$ man 5 passwd
PASSWD(5)              File Formats and Conversions          PASSWD(5)
```

Unfortunately, you won't always remember the exact name of the man page that you want to view. Fortunately, 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 password or gr...
cpio (1)             - copy files to and from archives
cppw (8)             - copy with locking the given file to the password 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...
```

Recall that there are thousands of man pages, so when searching for a keyword, be as specific as possible. Using a generic word, such as "the", could result in hundreds or even thousands of results.

Note: On most Linux distributions, the apropos command does the same thing as man -k. On those distributions, both produce the same output.

6.3 Finding Commands and Documentation

The whatis command (or man -f) returns what section a man page is stored in. This command occasionally returns unusual output, such as the following:

```
sysadmin@localhost:~$ whatis ls
ls (1)               - list directory contents
ls (lp)              - list directory contents
```

Note

The example above is designed to demonstrate a scenario where two commands list directory contents. The output in the example terminal above may not match the output in the VM.

Based on this output, there are two ls commands that list directory contents. The simple reason for this is that UNIX had two main variants, which resulted in some commands being developed "in parallel" and therefore behaving differently on different variants of UNIX. Many modern distributions of Linux include commands from both UNIX variants.

This does, however, pose a bit of a problem: when the ls command is typed, which command is executed?

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. Typically, the command to generate this database is run nightly.

```
sysadmin@localhost:~$ locate gshadow
/etc/gshadow
/etc/gshadow-
/usr/include/gshadow.h
/usr/share/man/cs/man5/gshadow.5.gz
/usr/share/man/da/man5/gshadow.5.gz
/usr/share/man/de/man5/gshadow.5.gz
/usr/share/man/fr/man5/gshadow.5.gz
/usr/share/man/it/man5/gshadow.5.gz
/usr/share/man/man5/gshadow.5.gz
/usr/share/man/ru/man5/gshadow.5.gz
/usr/share/man/sv/man5/gshadow.5.gz
/usr/share/man/zh_CN/man5/gshadow.5.gz
```

Any files created today will not be searchable with the locate command. If root access is available, it's possible to update the locate database manually by running the updatedb command. Regular users cannot update the database file.

Also note that when using the locate command as a regular user, the output may be limited due to file permissions. If the user that is logged in doesn't have access to a file or directory on the filesystem due to permissions, the locate command won't return those names. This security feature is designed to keep users from "exploring" the filesystem by using the locate database. The root user can search for any file in the locate database.

The output of the locate command can be quite large. When searching for a filename, such as passwd, the locate command produces every file that contains the string passwd, not just files named passwd.

In many cases, it is helpful to start by finding out how many files match. Do this by using the -c option to the locate command:

```
sysadmin@localhost:~$ locate -c passwd
98
```

To limit the output produced by the locate command use the -b option. This option only includes listings that have the search term in the basename of the filename. The *basename* is the portion of the filename not including the directory names.

```
sysadmin@localhost:~$ locate -c -b passwd
83
```

As you can see from the previous output, there will still be many results when the -b option is used. To limit the output even further, place a \ character in front of the search term. This character limits the output to filenames that exactly match the term:

```
sysadmin@localhost:~$ locate -b "\passwd"
/etc/passwd
/etc/pam.d/passwd
```

```
/usr/bin/passwd
/usr/share/doc/passwd
/usr/share/lintian/overrides/passwd
```

6.4 Info Documentation

Man pages are excellent sources of information, but they do tend to have a few disadvantages. One example is that each man page is a separate document, not related to any other man page. While some man pages have a SEE ALSO section that may refer to other man pages, they tend to be independent sources of 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.

All of the documentation is merged into a single "book" representing all of the documentation available. Within info documents, information is broken down into categories that work much like a table of contents in a book. Hyperlinks are provided to pages with information on individual topics for a specific command or feature. Another advantage of info over man pages is that the writing style of info documents is typically more conducive to learning a topic. Consider man pages to be more of a reference resource and info documents to be more of a learning guide.

6.4.1 Viewing Info Documentation

To display the info documentation for a command, use the info command.

```
info command
```

For example, to display the info page of the ls command:

```
sysadmin@localhost:~$ info ls
```

Navigate the document using the arrow keys:

```
Next: dir invocation,  Up: Directory listing

10.1 `ls': List directory contents
==================================

   The `ls' program lists information about files (of any type, including directories).  Options and file arguments
can be intermixed arbitrarily, as usual.

   For non-option command-line arguments that are directories, by
default `ls' lists the contents of directories, not recursively, and
omitting files with names beginning with `.'.  For other non-option
arguments, by default `ls' lists just the file name.  If no non-option
argument is specified, `ls' operates on the current directory, acting
as if it had been invoked with a single argument of `.'.

   By default, the output is sorted alphabetically, according to the
locale settings in effect.(1) If standard output is a terminal, the
output is in columns (sorted vertically) and control characters are
output as question marks; otherwise, the output is listed one per line
and control characters are output as-is.

-----Info: (coreutils)ls invocation, 57 lines --Top-----------------------------
Welcome to Info version 6.5.  Type H for help, h for tutorial.
```

This documentation is broken up into nodes, and in the example above the line highlighted in white shows it's currently in the ls invocation node. The first line provides index information about the current location within the document. The next node, like the next chapter in a book, would be the dir invocation node. Going up one level is the Directory listing node.

Scrolling through the document, notice the menu for the ls command:

```
* Menu:

* Which files are listed::
* What information is listed::
* Sorting the output::
* Details about version sort::
* General output formatting::
* Formatting file timestamps::
* Formatting the file names::
```

```
   ---------- Footnotes ----------

 (1) If you use a non-POSIX locale (e.g., by setting `LC_ALL' to
`en_US'), then `ls' may produce output that is sorted differently than
you're accustomed to.  In that case, set the `LC_ALL' environment
variable to `C'.

-----Info: (coreutils)ls invocation, 57 lines --Bot----------------------------
```

The items under the menu are hyperlinks that link to nodes that describe more about the ls command. For example, placing the cursor on the line * Sorting the output:: and pressing the Enter key, leads to a node that describes sorting the output of the ls command:

```
Next: Details about version sort,  Prev: What information is listed,  Up: ls in\
vocation

10.1.3 Sorting the output
-------------------------

These options change the order in which 'ls' sorts the information it
outputs.  By default, sorting is done by character code (e.g., ASCII
order).

'-c'
'--time=ctime'
'--time=status'
     If the long listing format (e.g., '-l', '-o') is being used, print
     the status change timestamp (the ctime) instead of the mtime.  When
     explicitly sorting by time ('--sort=time' or '-t') or when not
     using a long listing format, sort according to the ctime.  *Note
     File timestamps::.

'-f'
     Primarily, like '-U'--do not sort; list the files in whatever order
     they are stored in the directory.  But also enable '-a' (list all
-----Info: (coreutils)Sorting the output, 69 lines --Top-----------------------
```

Note that going into the node about sorting leads into a sub-node of the original. To go back to the previous node, use the U key. While U leads to the start of the node one level up, the L key returns to the same location as before entering the sorting node.

6.4.2 Navigating Info Documents

Like the man command, a listing of movement commands is available by hitting the Shift+H key while reading the info documentation:

```
Basic Info command keys

l          Close this help window.
q          Quit Info altogether.
h          Invoke the Info tutorial.

Up         Move up one line.
Down       Move down one line.
PgUp       Scroll backward one screenful.
PgDn       Scroll forward one screenful.
Home       Go to the beginning of this node.
End        Go to the end of this node.

TAB        Skip to the next hypertext link.
RET        Follow the hypertext link under the cursor.
l          Go back to the last node seen in this window.

[          Go to the previous node in the document.
]          Go to the next node in the document.
p          Go to the previous node on this level.
n          Go to the next node on this level.
u          Go up one level.
```

```
-----Info: *Info Help*, 302 lines -- Top-------------------------------------
```

Note that to close the help screen type the L key, which brings back the current document. To quit entirely, use the Q key.

6.4.3 Exploring Info Documentation

Instead of using info documentation to look up information about a specific command or feature, consider exploring the capabilities of Linux by reading through the info documentation. Execute the `info` command without any arguments to be taken to the top level of the documentation. This is a good starting point to explore many of the features offered:

```
File: dir,      Node: Top,      This is the top of the INFO tree.

This is the Info main menu (aka directory node).
A few useful Info commands:

  'q' quits;
  'H' lists all Info commands;
  'h' starts the Info tutorial;
  'mTexinfo RET' visits the Texinfo manual, etc.

* Menu:

Basics
* Common options: (coreutils)Common options.
* Coreutils: (coreutils).       Core GNU (file, text, shell) utilities.
* Date input formats: (coreutils)Date input formats.
* File permissions: (coreutils)File permissions.
                            Access modes.
* Finding files: (find).        Operating on files matching certain criteria.

C++ libraries
* autosprintf: (autosprintf).   Support for printf format strings in C++.
-----Info: (dir)Top, 211 lines --Top-----------------------------------------
Welcome to Info version 6.5.  Type H for help, h for tutorial.
```

6.5 Additional Sources of Help

In many cases, the man pages or info documentation provide needed information in an efficient manner. However, there are other sources for help to be aware of.

6.5.1 Using the Help Option

Many commands will provide basic information, very similar to the SYNOPSIS found in man pages, by simply using the `--help` option to the command. This option is useful to learn the basic usage of a command quickly without leaving the command line:

```
sysadmin@localhost:~$  cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

  -A, --show-all           equivalent to -vET
  -b, --number-nonblank    number nonempty output lines, overrides -n
  -e                       equivalent to -vE
  -E, --show-ends          display $ at end of each line
  -n, --number             number all output lines
  -s, --squeeze-blank      suppress repeated empty output lines
  -t                       equivalent to -vT
  -T, --show-tabs          display TAB characters as ^I
  -u                       (ignored)
  -v, --show-nonprinting   use ^ and M- notation, except for LFD and TAB
      --help     display this help and exit
      --version  output version information and exit

Examples:
  cat f - g  Output f's contents, then standard input, then g's contents.
  cat        Copy standard input to standard output.

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
```

```
Report cat translation bugs to <http://translationproject.org/team/>
Full documentation at: <http://www.gnu.org/software/coreutils/cat>
or available locally via: info '(coreutils) cat invocation'
```

.5.2 Additional System Documentation

On most systems, there is a directory where additional documentation (such as documentation files stored by third-party software vendors) is found.

These documentation files are often called *readme files* since the files typically have names such as README or readme.txt. The location of these files can vary depending on the distribution that you are using. Typical locations include /usr/share/doc and /usr/doc.

Typically, this directory is where system administrators go to learn how to set up more complex software services. However, sometimes regular users also find this documentation to be useful.

6.1 Introduction

This is Lab 6: Getting Help. By performing this lab, students will learn how to get help on commands and find files.

In this lab, you will perform the following tasks:

- Use several help systems to get help for commands.
- Learn how to locate commands.

6.2 Getting Help

In this task, you will explore the how to get help. This will be a very useful thing to know how to do when you find yourself stuck or when you can't remember how a command works.

In addition to Internet searches, the Linux Operating System provides a variety of techniques to learn more about a given command or feature. Knowing these different techniques will allow you to more easily and quickly find the answer you need.

6.2.1 Step 1

Execute commands in the bash shell by typing the command and then pressing the Enter key. For example, type the following command to display today's date:

```
date
```

The output should be similar to the following:

```
sysadmin@localhost:~$ date
Tue Dec  4 20:59:28 UTC 2018
```

6.2.2 Step 2

To learn more about commands, access the manual page for the command with the man command. For example, execute the following command to learn more about the date command:

```
man date
```

```
sysadmin@localhost:~$ man date
```

Your output should be similar to the following:

```
DATE(1)                    User Commands                    DATE(1)

NAME
       date - print or set the system date and time

SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION
       Display the current time in the given FORMAT, or set the system date.

       -d, --date=STRING
              display time described by STRING, not `now'
       -f, --file=DATEFILE
              like --date once for each line of DATEFILE

       -r, --reference=FILE
              display the last modification time of FILE

       -R, --rfc-2822
              output  date  and time in RFC 2822 format.  Example: Mon, 07 Aug
```

```
Manual page date(1) line 1 (press h for help or q to quit)
```

Note
Documents that are displayed with the man command are called "Man Pages".
If the man command can find the manual page for the argument provided, then that manual page will be displayed using a command called less. The following table describes useful keys that can be used with the less command to control the output of the display:

Key	Purpose
H or h	Display the help
Q or q	Quit the help or manual page
Spacebar or f or PageDown	Move a screen forward
b or PageUp	Move a screen backward
Enter or down arrow	Move down one line
Up arrow	Move up one line
/ *followed by text to search*	Start searching forward
? *followed by text to search*	Start searching backward
n	Move to next text that matches search
N	Move to previous matching text

6.2.3 Step 3
Type the letter h to see a list of movement commands. After reading the movement commands, type the letter q to get back to the document.

```
                SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                           MOVING

e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
b  ^B  ESC-v      *  Backward one window (or N lines).
z                 *  Forward  one window (and set window to N).
w                 *  Backward one window (and set window to N).
ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
d  ^D             *  Forward  one half-window (and set half-window to N)
u  ^U             *  Backward one half-window (and set half-window to N)
ESC-)  RightArrow *  Left  one half screen width (or N positions).
HELP -- Press RETURN for more, or q when done
```

Note that the man pages might be a bit of a mystery to you now, but as you learn more about Linux, you will find they are a very valuable resource.
6.2.4 Step 4
Searches are not case sensitive and do not "wrap" around from the bottom to top, or vice versa. Start a forward search for the word "file" by typing:

```
/file
```

Note that what you are typing will appear at the bottom left portion of the screen.

```
       -r, --reference=FILE
              display the last modification time of FILE

       -R, --rfc-2822
              output  date  and time in RFC 2822 format.  Example: Mon, 07 Aug
/file
```

Press Enter to search the document for the search string (file).

6.2.5 Step 5

Notice that the text matching the search is highlighted. You can move forward to the next match by pressing n. Also try moving backwards through the matches by pressing N:

```
-f, --file=DATEFILE
              like --date once for each line of DATEFILE

       -r, --reference=FILE
              display the last modification time of FILE

      -R, --rfc-2822
              output  date  and time in RFC 2822 format.  Example: Mon, 07 Aug
              2006 12:34:56 -0600

       --rfc-3339=TIMESPEC
              output date and time in RFC 3339 format.  TIMESPEC=`date', `sec-
              onds',  or  `ns'  for  date and time to the indicated precision.
              Date and time  components  are  separated  by  a  single  space:
              2006-08-07 12:34:56-06:00

       -s, --set=STRING
              set time described by STRING

       -u, --utc, --universal
              print or set Coordinated Universal Time

       --help display this help and exit
Manual page date(1) line 18/204 24% (press h for help or q to quit)
```

6.2.7 Step 7

In some cases you may not remember the exact name of the command. In these cases you can use the -k option to the man command and provide a *keyword* argument. For example, execute the following command to display a summary of all man pages that have the keyword "password" in the description:

```
man -k password
```

```
sysadmin@localhost:~$ man -k password
chage (1)            - change user password expiry information
chgpasswd (8)        - update group passwords in batch mode
chpasswd (8)         - update passwords in batch mode
cpgr (8)             - copy with locking the given file to the password or gr...
cppw (8)             - copy with locking the given file to the password or gr...
expiry (1)           - check and enforce password expiration policy
login.defs (5)       - shadow password suite configuration
pam_pwhistory (8)    - PAM module to remember last passwords
pam_unix (8)         - Module for traditional password authentication
passwd (1)           - change user password
passwd (1ssl)        - compute password hashes
passwd (5)           - the password file
pwck (8)             - verify integrity of password files
pwconv (8)           - convert to and from shadow passwords and groups
shadow (5)           - shadowed password file
shadowconfig (8)     - toggle shadow passwords on and off
unix_chkpwd (8)      - Helper binary that verifies the password of the curren...
unix_update (8)      - Helper binary that updates the password of a given user
vipw (8)             - edit the password, group, shadow-password or shadow-gr...
sysadmin@localhost:~$
```

The -k option to the man command will often produce a huge amount of output. We will cover a technique in a later lab to either limit this output or allow you to easily scroll through the data. For now, you can scroll using your mouse in the terminal window to move the display up and down as needed.

6.2.8 Step 8

Note that the apropos command is another way of viewing man page summaries with a keyword. Type the following command:

```
apropos password
```

```
sysadmin@localhost:~$ apropos password
chage (1)            - change user password expiry information
chgpasswd (8)        - update group passwords in batch mode
chpasswd (8)         - update passwords in batch mode
cpgr (8)             - copy with locking the given file to the password or gr...
cppw (8)             - copy with locking the given file to the password or gr...
expiry (1)           - check and enforce password expiration policy
login.defs (5)       - shadow password suite configuration
pam_pwhistory (8)    - PAM module to remember last passwords
pam_unix (8)         - Module for traditional password authentication
passwd (1)           - change user password
passwd (1ssl)        - compute password hashes
passwd (5)           - the password file
pwck (8)             - verify integrity of password files
pwconv (8)           - convert to and from shadow passwords and groups
shadow (5)           - shadowed password file
shadowconfig (8)     - toggle shadow passwords on and off
unix_chkpwd (8)      - Helper binary that verifies the password of the curren...
unix_update (8)      - Helper binary that updates the password of a given user
vipw (8)             - edit the password, group, shadow-password or shadow-gr...
sysadmin@localhost:~$
```

Note

There is no difference between man -k and the apropos command.

6.2.9 Step 9

There are often multiple man pages with the same name. For example, the following command shows three pages for passwd. Execute the following command to view the man pages for the word passwd:

```
man -f passwd
```

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

The fact that there are different man pages for the same "name" is confusing for many beginning Linux users. Man pages are not just for Linux commands, but also for system files and other "features" of the Operating System. Additionally, there will sometimes be two commands with the same name, as in the example provided above.

The different man pages are distinguished by "sections". By default there are nine sections of man pages:

- Executable programs or shell commands
- System calls (functions provided by the kernel)
- Library calls (functions within program libraries)
- Special files (usually found in /dev)
- File formats and conventions, e.g. /etc/passwd
- Games
- Miscellaneous (including macro packages and conventions), e.g. man(7)>, groff(7)
- System administration commands (usually only for root)
- Kernel routines

When you type a command such as man passwd, the first section is searched and, if a match is found, the man page is displayed. The man -f passwd command that you previously executed shows that there is a section 1 man page for passwd: passwd (1). As a result, that is the one that is displayed by default.

6.2.10 Step 10

To display a man page for a different section, provide the section number as the first argument to the man command. For example, execute the following command:

```
man 5 passwd
```

```
PASSWD(5)            File Formats and Conversions            PASSWD(5)
```

```
NAME
       passwd - the password file

DESCRIPTION
       /etc/passwd contains one line for each user account, with seven fields
       delimited by colons (":"). These fields are:

       o   login name

       o   optional encrypted password

       o   numerical user ID

       o   numerical group ID

       o   user name or comment field

       o   user home directory

       o   optional user command interpreter

 Manual page passwd(5) line 1 (press h for help or q to quit)
```

Type q to return to the system prompt.

6.2.11 Step 11

Instead of using man -f to display all man page sections for a name, you can also use the whatis command:

```
whatis passwd
```

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

Note

There is no difference between man -f and the whatis command.

6.2.12 Step 12

Almost all system features (commands, system files, etc.) have man pages. Some of these features also have a more advanced feature called *info* pages. For example, execute the following command:

```
info date
```

```
File: coreutils.info,  Node: date invocation,  Next: arch invocation,  Up: Syst\
em context

21.1 `date': Print or set system date and time
==============================================

Synopses:

     date [OPTION]... [+FORMAT]
     date [-u|--utc|--universal] [ MMDDhhmm[[CC]YY][.ss] ]

   Invoking `date' with no FORMAT argument is equivalent to invoking it
with a default format that depends on the `LC_TIME' locale category.
In the default C locale, this format is `"+%a %b %e %H:%M:%S %Z %Y"',
so the output looks like `Thu Mar  3 13:47:51 PST 2005'.

   Normally, `date' uses the time zone rules indicated by the `TZ'
environment variable, or the system default rules if `TZ' is not set.
*Note Specifying the Time Zone with `TZ': (libc)TZ Variable.

   If given an argument that starts with a `+', `date' prints the
current date and time (or the date and time specified by the `--date'
-----Info: (coreutils)date invocation, 40 lines --Top-------------------------
Welcome to Info version 6.5.  Type H for help, h for tutorial.
```

Many beginning Linux users find info pages to be easier to read. They are often written more like "lessons" while man pages are written purely as documentation.

6.2.13 Step 13

While viewing the info page from the previous step, type **Shift** and the letter **h** to see a list of movement commands. Note that they are different from the movement commands used in man pages. After reading the movement commands, type the letter **l** (lowercase L) to return to viewing the document.

6.2.14 Step 14

Use the movement commands to read the info page for the date command. When you are done, put your cursor anywhere on the line that reads *Examples of date:: and then press the **Enter** key. A new document will be displayed that shows examples of date.

6.2.15 Step 15

Type the **l** key to return to the previous screen. When you are finished reading, type **q** to exit the info page.

6.2.16 Step 16

Another way of getting help is by using the --help option to a command. Most commands allow you to pass an argument of --help to view basic command usage:

```
date --help
```

```
sysadmin@localhost:~$ date --help
Usage: date [OPTION]... [+FORMAT]
  or:  date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT, or set the system date.

  -d, --date=STRING         display time described by STRING, not `now'
  -f, --file=DATEFILE       like --date once for each line of DATEFILE
  -r, --reference=FILE      display the last modification time of FILE
  -R, --rfc-2822            output date and time in RFC 2822 format.
                      Example: Mon, 07 Aug 2006 12:34:56 -0600
      --rfc-3339=TIMESPEC   output date and time in RFC 3339 format.
                      TIMESPEC=`date', `seconds', or `ns' for
                      date and time to the indicated precision.
                      Date and time components are separated by
                      a single space: 2006-08-07 12:34:56-06:00
  -s, --set=STRING          set time described by STRING
  -u, --utc, --universal    print or set Coordinated Universal Time
      --help     display this help and exit
      --version  output version information and exit
```

6.2.17 Step 17

Some system features also have more detailed help documents located in the /usr/share/doc directory structure. Execute the following command to view the contents of this document:

```
ls /usr/share/doc
```

```
sysadmin@localhost:~$ ls /usr/share/doc
adduser             libdrm2             libx11-data
apt                 libedit2            libxau6
ascii               libelf1             libxcb1
base-files          libffi6             libxdmcp6
base-passwd         libgcc1             libxext6
bash                libgcrypt11         libxml2
bind9               libgdbm3            libxmuu1
bind9-host          libgeoip1           locales
bind9utils          libgettextpo0       login
bsdmainutils        libglib2.0-0        logrotate
bsdutils            libgnutls26         lsb-base
busybox-initramfs   libgomp1            makedev
bzip2               libgpg-error0       man-db
ca-certificates     libgpm2             mawk
coreutils           libgssapi-krb5-2    mc
cpio                libgssapi3-heimdal  mc-data
cron                libhcrypto4-heimdal mime-support
curl                libheimbase1-heimdal mlocate
dash                libheimntlm0-heimdal module-init-tools
```

Note that in almost all cases, the man pages and info pages will provide you with the information that you need. However, if you need more in-depth information (something that system administrators sometimes need), then you may find this information in the files located in the /usr/share/doc

6.3 Finding Files

In this task, we will explore how to search for a file on the system. This is useful to know in situations when you can't find a file on the system, either one that you created or one that was created by someone else.

6.3.1 Step 1

An easy way to search for a file is to use the locate command. For example, you can find the location of the crontab file by executing the following command:

```
locate crontab
```

```
sysadmin@localhost:~$ locate crontab
/etc/crontab
/usr/bin/crontab
/usr/share/doc/cron/examples/crontab2english.pl
/usr/share/man/man1/crontab.1.gz
/usr/share/man/man5/crontab.5.gz
sysadmin@localhost:~$
```

6.3.2 Step 2

Note that the output from the previous example includes files that have crontab as part of their name. To find files that are just named crontab, use the following command:

```
locate -b "\crontab"
```

```
sysadmin@localhost:~$ locate -b "\crontab"
/etc/crontab
/usr/bin/crontab
sysadmin@localhost:~$
```

Note: The locate command makes use of a database that is traditionally updated once per day (normally in the middle of the night). This database contains a list of all files that were on the system when the database was last updated.

As a result, any files that you created today will not normally be searchable with the locate command. If you have access to the system as the root user (the system administrator account), you can manually update this file by running the updatedb command. Regular users cannot update the database file.

Another possible solution to searching for "newer" files is to make use of the find command. This command searches the live filesystem, rather than a static database. The find command isn't part of the Linux Essentials objectives for this lab, so it is only mentioned here. Execute man find if you want to explore this command on your own.

6.3.3 Step 3

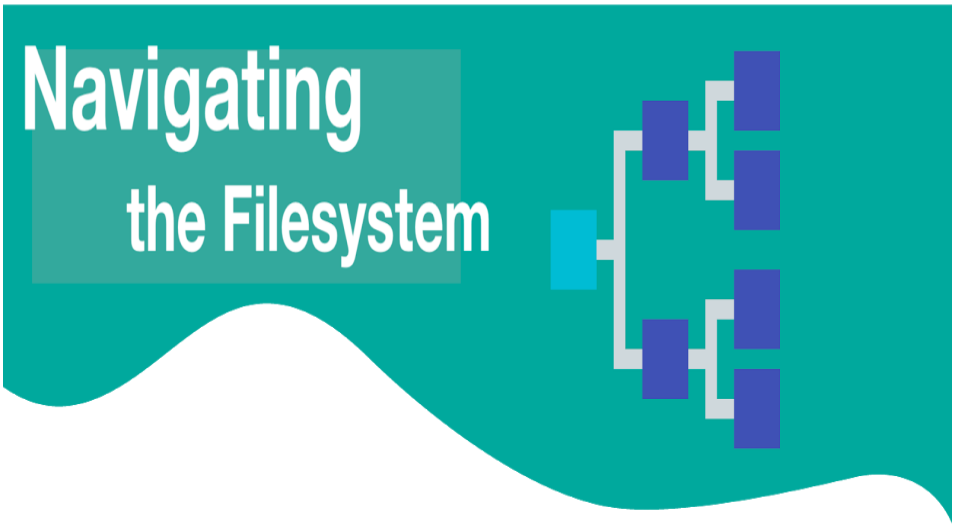
You may just want to find where a command (or its man pages) is located. This can be accomplished with the whereis command:

```
whereis passwd
```

```
sysadmin@localhost:~$ whereis passwd
passwd: /usr/bin/passwd /etc/passwd /usr/share/man/man5/passwd.5.gz /usr/share/man/man1/passwd.1ssl.gz
/usr/share/man/man1/passwd.1.gz
sysadmin@localhost:~$
```

The whereis command does not search for just any file, only for commands and man pages.

Recall from earlier that there is more than one passwd man page on the system. This is why you see multiple file names and man pages (the files that end in .gz are man pages) when you execute the previous command.



7.1 Introduction

In Linux, everything is considered a file. Files are used to store data such as text, graphics, and programs. Directories are a type of file used to store other files; Windows and Mac OS X users typically refer to them as folders. In any case, directories are used to provide a hierarchical organization structure. However, this structure may be somewhat different depending on the type of system in use.

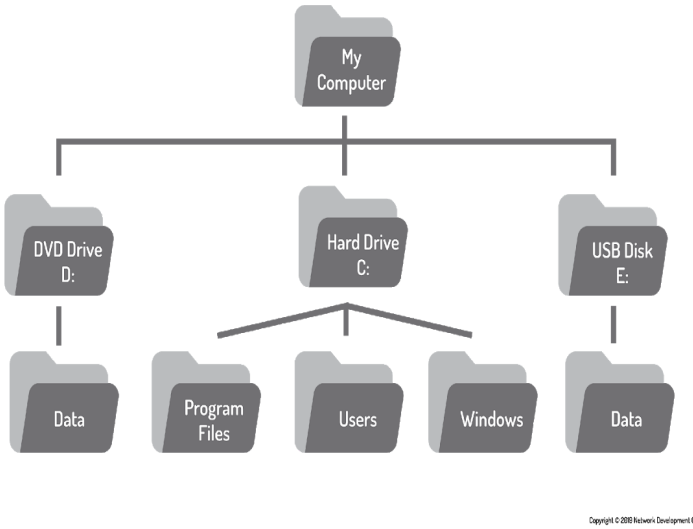
When working in a Linux operating system, it is important to know how to manipulate files and directories. Some Linux distributions have GUI-based applications that allow you to manage files, but it is advantageous to know how to perform these operations via the command line.

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:.

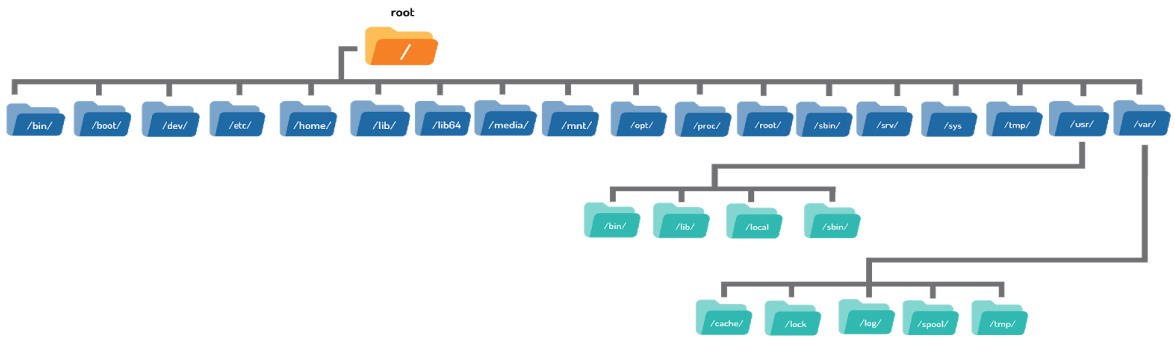
The directory structures shown below are provided as examples only. These directories may not be present within the virtual machine environment of this course.

A visual representation of a Windows directory structure:



Like 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:



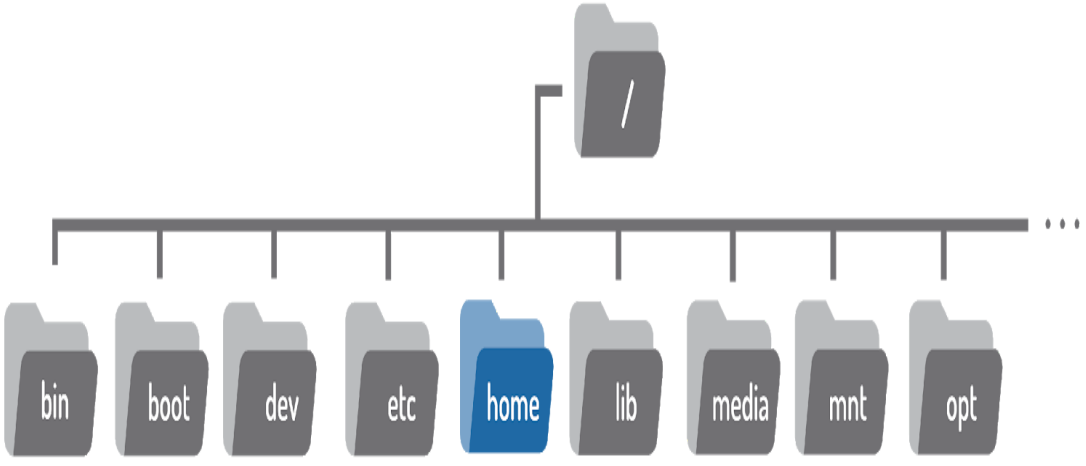
To view the contents of the root directory, use the ls command with the / character as the argument:

```
sysadmin@localhost:~$ ls /
 bin   etc   lib    mnt   root 'sbin'$'\342\200\214'  tmp
 boot  home  lib64  opt   run   srv                   usr
 dev   init  media  proc  sbin  sys                   var
```

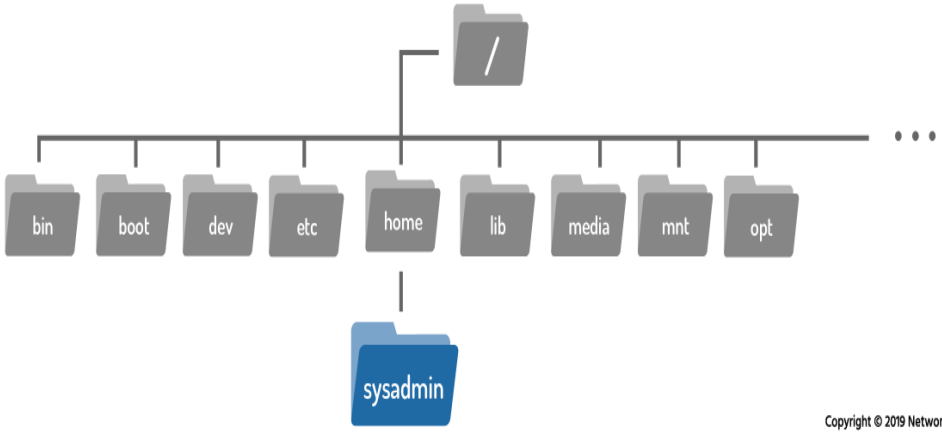
Notice that there are many directories with descriptive names including /boot, which contains files to boot the computer.

7.2.1 Home Directory

The term home directory often confuses new Linux users. 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 an important directory. To begin with, when a user opens a shell, they should automatically be placed in their home directory, as typically this is where they do most of their work. Additionally, 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.

File permissions and ownership will be covered in detail later in the course.

The 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.

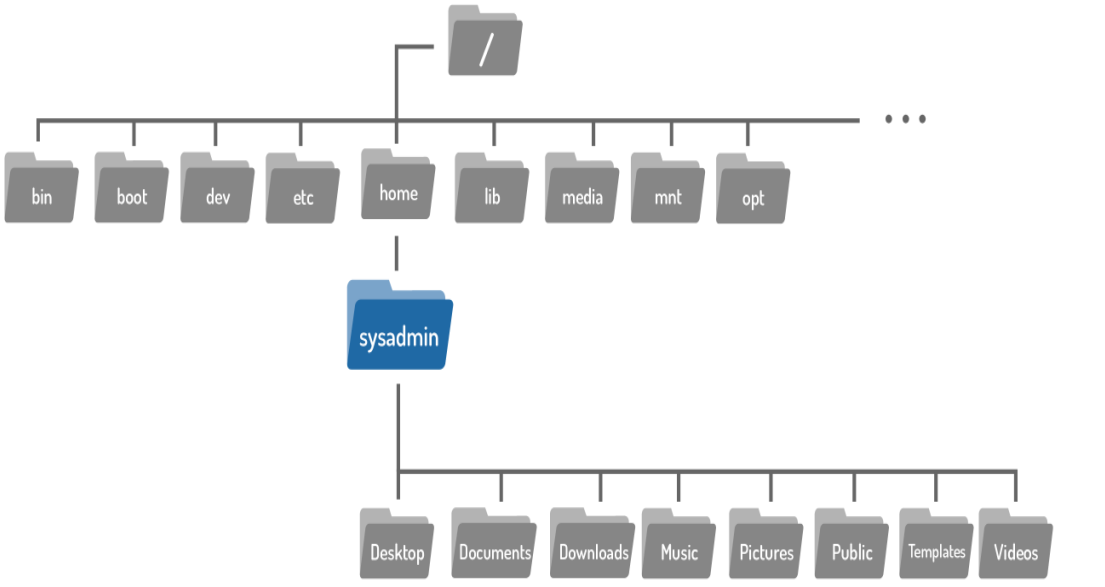
It is also possible to refer to another user's home directory by using the tilde ~ character followed by the name of the user account. For example, ~bob would be the equivalent of /home/bob.

7.2.2 Current Directory

To determine where the user is currently located within the filesystem, the pwd (print working directory) command can be used:

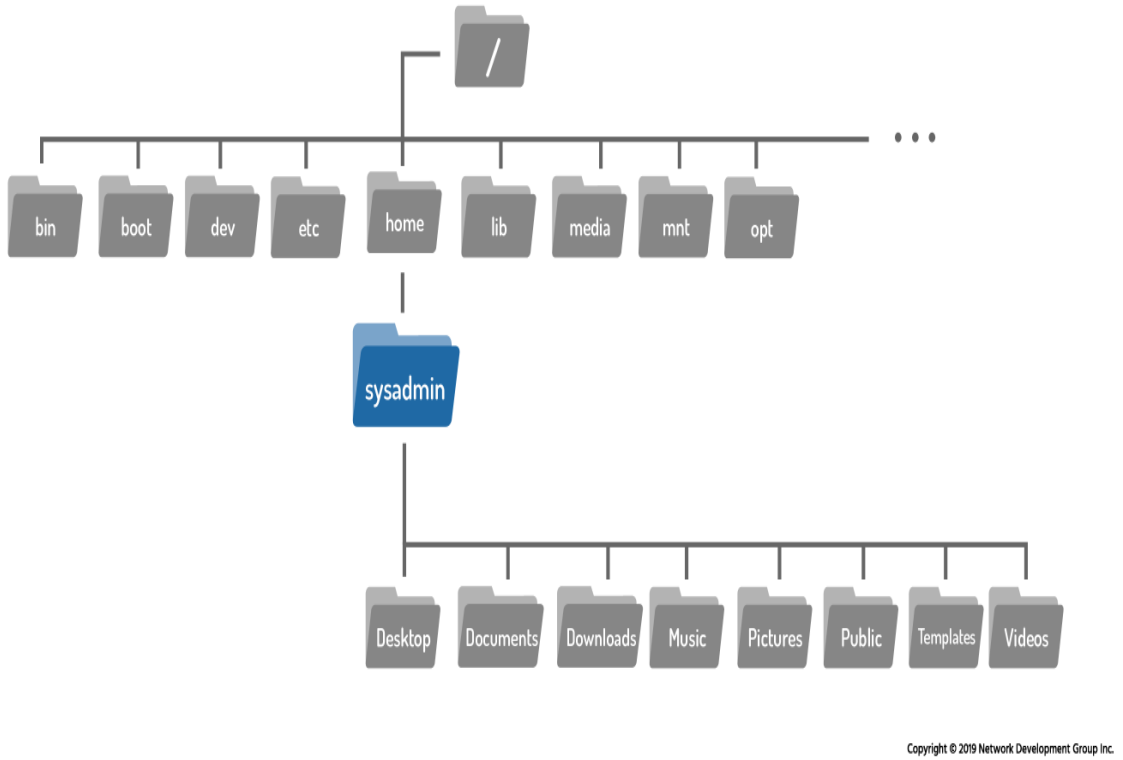
```
pwd [OPTIONS]
sysadmin@localhost:~$ pwd
/home/sysadmin
```

The pwd command prints the working directory, which is the current location of the user within the filesystem. The output of the above command indicates that the sysadmin user is currently in their home directory, shown in the filesystem below:



7.2.3 Changing Directories

When a user opens a shell, they typically begin in their home directory. When you start a fresh virtual machine in our course, either by opening the course or after using the reset button, you are logged in as the sysadmin user, and you begin in the home directory for that user, highlighted in the image below.

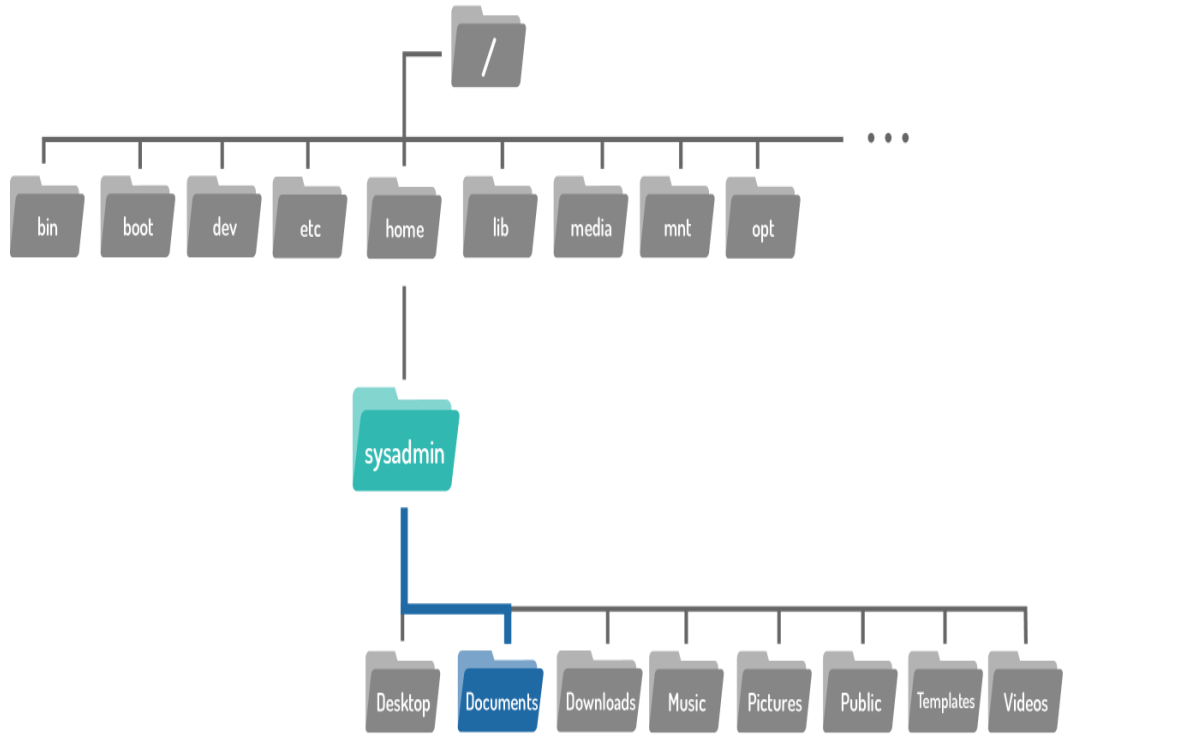


To navigate the filesystem, use the cd (change directory) command.

```
cd [options] [path]
```

There is a directory called Documents located in the home directory of the sysadmin user. To move from the home directory into the Documents directory use the directory name as an argument to the cd command:

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



When used with no arguments, the cd command will take the user to their home directory.

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

Notice our virtual machines employ a prompt that displays the current working directory, emphasized with the color blue. In the first prompt, the tilde ~ character is equivalent to /home/sysadmin, representing the user's home directory.

```
sysadmin@localhost:~$
```

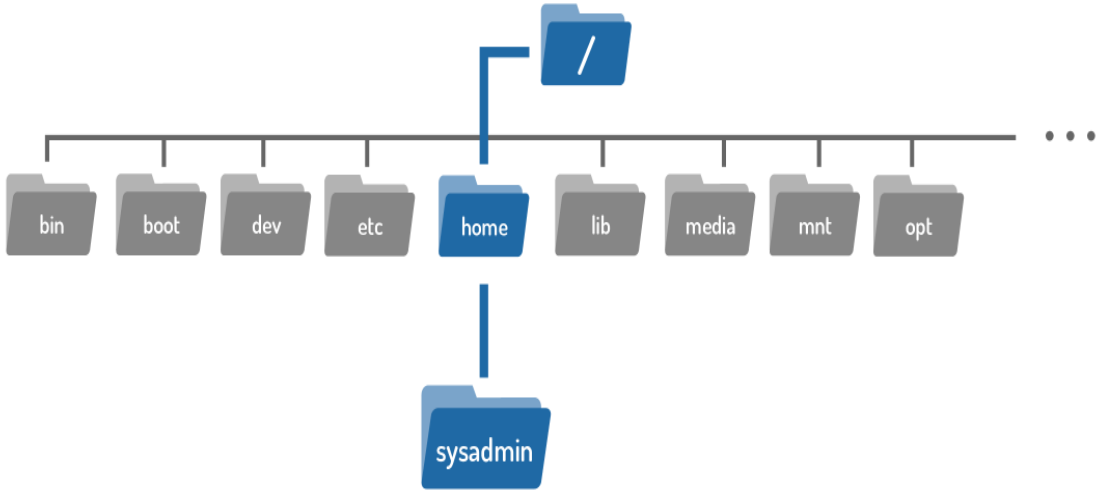
After changing directories, the new location ~/Documents can also be confirmed in the new prompt, again shown in blue.

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

Consider This
For some commands, no news is good news; there is no output if the cd command is successful. If the user tries to change to a directory that does not exist, the command returns an error message:

```
sysadmin@localhost:~$ cd Junk
-bash: cd: Junk: No such file or directory
```

7.3 Paths
The argument to the cd command is more than just the name of a directory, it is actually a path. A path is a list of directories separated by the / character. If you think of the filesystem as a map, paths are the directory addresses, which include step-by-step navigation directions; they can be used to indicate the location of any file within the filesystem.
For example, /home/sysadmin is a path to the home directory:



There are two types of paths: absolute and relative.

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.

If the path /home/sysadmin is used as an argument to the cd command, it moves the user into the home directory for the sysadmin user.

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

Again, no output means the command succeeded. This can be confirmed by looking at the prompt, or using the pwd command:

```
sysadmin@localhost:~$ pwd
/home/sysadmin
```

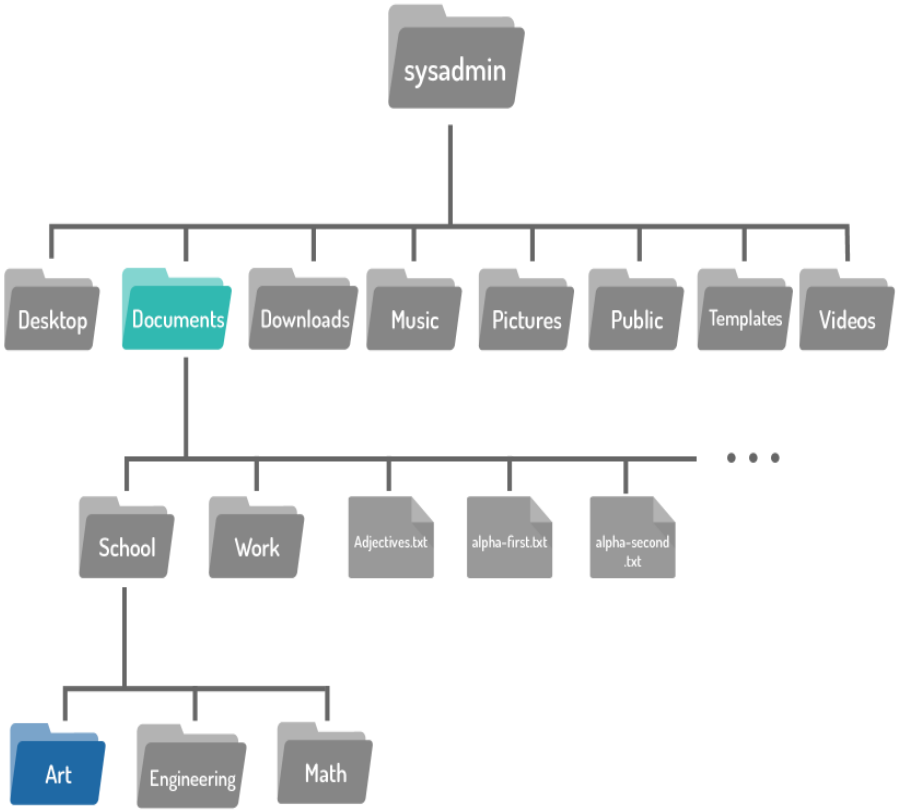
7.3.2 Relative Paths

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$
```

If the user is located in the Documents directory, moving to the Art directory can be accomplished in a number of ways.



The absolute path to the Art directory can be used:

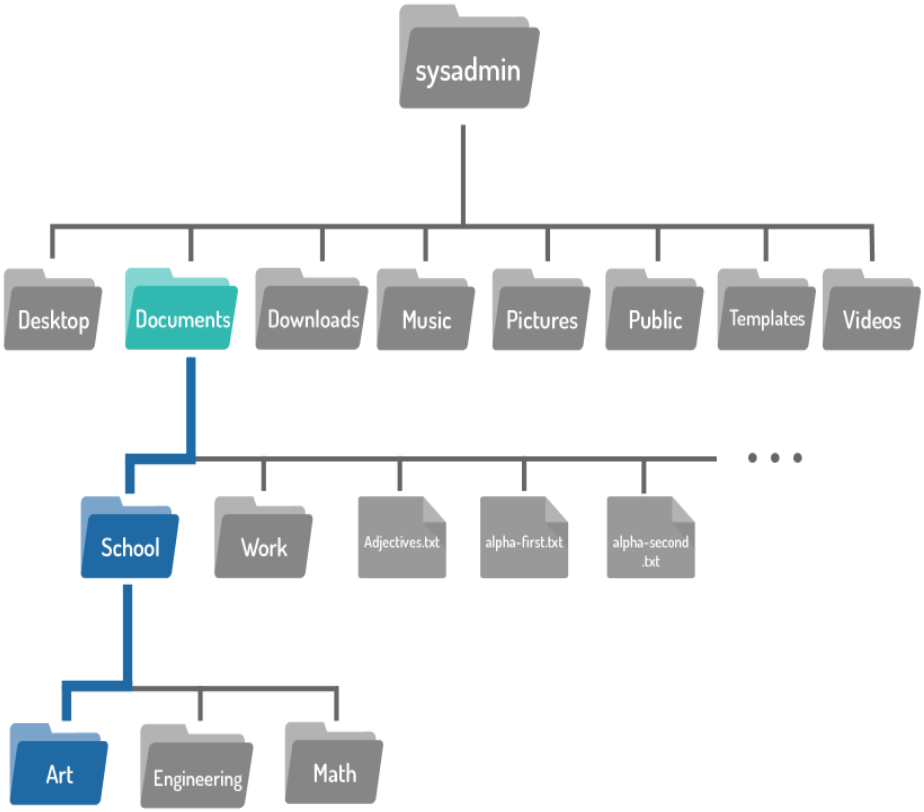
```
sysadmin@localhost:~/Documents$ cd /home/sysadmin/Documents/School/Art
sysadmin@localhost:~/Documents/School/Art$
```

Multiple relative paths can be used:

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

However, the simplest method is to use a single relative path that covers the journey from the origin to the destination directory:

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



Use the pwd command to confirm the change:

```
sysadmin@localhost:~/Documents/School/Art$ pwd
/home/sysadmin/Documents/School/Art
```

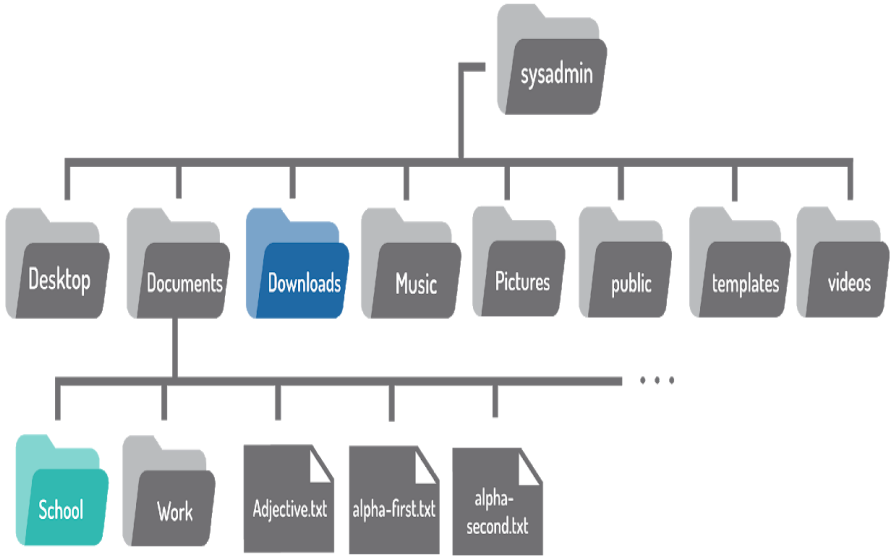
7.3.3 Shortcuts

The .. Characters

Regardless of which directory the user is in, two period .. characters always represents one directory higher relative to the current directory, sometimes referred to as the parent directory. To move from the Art directory back to the School directory:

```
sysadmin@localhost:~/Documents/School/Art$ cd ..
sysadmin@localhost:~/Documents/School$
```

The double dot can also be used in longer paths as well. The following relative path could be used to move from the School directory to the Downloads directory (both highlighted in the image below):




```
sysadmin@localhost:~/Documents/School$ cd ../../Downloads
sysadmin@localhost:~/Downloads$
```

The . Character
Regardless of which directory the user is in, the single period . character always represents the current directory. For the cd this shortcut is not very useful, but it comes in handy for commands covered in subsequent sections.
7.4 Listing Files in a Directory
For the previous examples, images were provided to show the layout of the filesystem. In practice, maps like these aren't provided, and users must rely on what's available in the command line, making the ls (list) command one of the most powerful for navigating the filesystem.

```
ls [OPTION]... [FILE]...
```

This ls command is used to display the contents of a directory and can provide detailed information about the files. By default, when it is used with no options or arguments, it lists the files in the current directory:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

The ls command can also be used to list the contents of any directory in the filesystem. Provide the path to the directory as an argument:

```
sysadmin@localhost:~$ ls /var
backups  cache  lib  local  lock  log  mail  opt  run  spool  tmp
```

7.4.1 Listing Hidden Files
When the ls command is used to display the contents of a directory, not all files are shown automatically. The ls command omits hidden files by default. 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

Each file has details associated with it called metadata. This can include information such as the size, ownership, or timestamps. To view this information, use the `-l` option to the `ls` command. Below, a listing of the /var/log directory is used as an example, since it provides a variety of output:

```
sysadmin@localhost:~$ ls -l /var/log/
total 900
-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
-rw-r----- 1 syslog adm     371 Dec 15 16:38 auth.log
-rw-r--r-- 1 root   root  35330 May 26  2018 bootstrap.log
-rw-rw---- 1 root   utmp      0 May 26  2018 btmp
-rw-r----- 1 syslog adm     197 Dec 15 16:38 cron.log
-rw-r--r-- 1 root   adm   85083 Dec 10 21:33 dmesg
-rw-r--r-- 1 root   root 351960 Jul 19 06:52 dpkg.log
-rw-r--r-- 1 root   root  32064 Dec 10 21:33 faillog
drwxr-xr-x 2 root   root   4096 Jul 19 06:51 journal
-rw-rw-r-- 1 root   utmp 292584 Dec 15 16:38 lastlog
-rw-r----- 1 syslog adm   14185 Dec 15 16:38 syslog
-rw------- 1 root   root  64128 Dec 10 21:33 tallylog
-rw-rw-r-- 1 root   utmp    384 Dec 15 16:38 wtmp
```

In the output above, each line displays metadata about a single file. The following describes each of the fields of data in the output of the `ls -l` command:

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.
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

```
drwxr-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.

Permissions will be covered in detail later in the course.

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.

Links will be covered in detail later in the course.

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.

File ownership will be covered in detail later in the course.

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 ownership will be covered in detail later in the course.
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
```

Symbolic links will be covered in detail later in the course.

7.4.3 Human-Readable Sizes
The -l option to the ls command displays file sizes in bytes. For text files, a byte is 1 character. For smaller files, byte sizes are fine. However, for larger files, it is hard to comprehend how large the file is. For example, consider the output of the following command:

```
sysadmin@localhost:~$ ls -l /var/log/lastlog
-rw-rw-r-- 1 root utmp 292584 Dec 15 16:38 /var/log/lastlog
```

The file size is hard to determine in bytes. Is 292584 a large file or small? It seems fairly large, but it is hard to determine using bytes.
Think of it this way: if someone were to give the distance between Boston and New York using inches, that value would be meaningless. Most people think in terms of miles or kilometers.
Sometimes it is preferable to present the file size in a more human-readable size, like megabytes or gigabytes.
To accomplish this, add the -h option to the -l command:

```
sysadmin@localhost:~$ ls -lh /var/log/lastlog
-rw-rw-r-- 1 root utmp 286K Dec 15 16:38 /var/log/lastlog
```

Important: The -h option must be used with the -l option.

7.4.4 Listing Directories
When the -d option is used, it refers to the current directory, and not the contents within it. Without any other options, it is rather meaningless. Recall that the current directory is always referred to with a single period . character:

```
sysadmin@localhost:~$ ls -d
.
```

To use the -d option in a meaningful way requires the addition of the -l option. In this case, note that the following command lists the details of the contents in the /home/sysadmin directory:

```
sysadmin@localhost:~$ ls -l
total 32
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Desktop
drwxr-xr-x 4 sysadmin sysadmin 4096 Dec 10 21:33 Documents
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Downloads
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Dec 10 21:33 Videos
```

In comparison, the next command lists the /home/sysadmin directory itself:

```
sysadmin@localhost:~$ ls -ld
drwxr-xr-x 1 sysadmin sysadmin 224 Nov  7 17:07 .
```

Note the single period at the end of the long listing. This indicates that the current directory is being listed, and not the contents.

7.4.5 Recursive Listing

There are times when you want to display all of the files in a directory as well as all of the files in all subdirectories under that directory. This is called a recursive listing.

To perform a recursive listing, use the -R option to the ls command:

```
sysadmin@localhost:~$ ls -R /etc/ppp
/etc/ppp:
ip-down.d  ip-up.d

/etc/ppp/ip-down.d:
bind9

/etc/ppp/ip-up.d:
bind9
```

Note that in the previous example, the files in the /etc/ppp directory were listed first. After that, the contents of its subdirectories /etc/ppp/ip-down.d and /etc/ppp/ip-up.d were listed.

Be careful with this option; for example, running the command on the root directory would list every file on the file system, including all files on any attached USB device and DVD in the system. Limit the use of recursive listings to smaller directory structures.

7.4.6 Sort a Listing

By default, the ls command sorts files alphabetically by file name. Sometimes, it may be useful to sort files using different criteria.

To sort files by size, we can use the -S option. Note the difference in the output of the following two commands. The same files and directories are listed, but in a different order:

```
sysadmin@localhost:~$ ls /etc/ssh
moduli              ssh_host_ecdsa_key.pub    ssh_host_rsa_key      sshd_config
ssh_config          ssh_host_ed25519_key      ssh_host_rsa_key.pub
ssh_host_ecdsa_key  ssh_host_ed25519_key.pub  ssh_import_id
sysadmin@localhost:~$ ls -S /etc/ssh
moduli              ssh_host_ed25519_key  ssh_host_ecdsa_key.pub
sshd_config         ssh_host_rsa_key.pub  ssh_host_ed25519_key.pub
ssh_host_rsa_key    ssh_import_id
ssh_config          ssh_host_ecdsa_key
```

Note

The option above uses a capital letter s.

While the -S option works by itself, it is most useful when used with the -l option so the file sizes are visible. The following command lists files from largest to smallest and displays the actual size of the file.

```
sysadmin@localhost:~$ ls -lS /etc/ssh
total 580
-rw-r--r-- 1 root root 553122 Feb 10  2018 moduli
-rw-r--r-- 1 root root   3264 Feb 10  2018 sshd_config
-rw------- 1 root root   1679 Jul 19 06:52 ssh_host_rsa_key
-rw-r--r-- 1 root root   1580 Feb 10  2018 ssh_config
-rw------- 1 root root    411 Jul 19 06:52 ssh_host_ed25519_key
-rw-r--r-- 1 root root    399 Jul 19 06:52 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    338 Jul 19 06:52 ssh_import_id
-rw------- 1 root root    227 Jul 19 06:52 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 Jul 19 06:52 ssh_host_ecdsa_key.pub
-rw-r--r-- 1 root root     99 Jul 19 06:52 ssh_host_ed25519_key.pub
```

It may also be useful to use the -h option to display human-readable file sizes:

```
sysadmin@localhost:~$ ls -lSh /etc/ssh
total 580K
-rw-r--r-- 1 root root 541K Feb 10  2018 moduli
-rw-r--r-- 1 root root 3.2K Feb 10  2018 sshd_config
-rw------- 1 root root 1.7K Jul 19 06:52 ssh_host_rsa_key
-rw-r--r-- 1 root root 1.6K Feb 10  2018 ssh_config
-rw------- 1 root root  411 Jul 19 06:52 ssh_host_ed25519_key
```

```
-rw-r--r-- 1 root root  399 Jul 19 06:52 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root  338 Jul 19 06:52 ssh_import_id
-rw------- 1 root root  227 Jul 19 06:52 ssh_host_ecdsa_key
-rw-r--r-- 1 root root  179 Jul 19 06:52 ssh_host_ecdsa_key.pub
-rw-r--r-- 1 root root   99 Jul 19 06:52 ssh_host_ed25519_key.pub
```

The -t option sorts files based on the time they were modified. It will list the most recently modified files first. This option can be used alone, but again, is usually more helpful when paired with the -l option:

```
sysadmin@localhost:~$ ls -tl /etc/ssh
total 580
-rw------- 1 root root    227 Jul 19 06:52 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 Jul 19 06:52 ssh_host_ecdsa_key.pub
-rw------- 1 root root    411 Jul 19 06:52 ssh_host_ed25519_key
-rw-r--r-- 1 root root     99 Jul 19 06:52 ssh_host_ed25519_key.pub
-rw------- 1 root root   1679 Jul 19 06:52 ssh_host_rsa_key
-rw-r--r-- 1 root root    399 Jul 19 06:52 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    338 Jul 19 06:52 ssh_import_id
-rw-r--r-- 1 root root 553122 Feb 10  2018 moduli
-rw-r--r-- 1 root root   1580 Feb 10  2018 ssh_config
-rw-r--r-- 1 root root   3264 Feb 10  2018 sshd_config
```

It is important to remember that the modified date on directories represents the last time a file was added to or removed from the directory.

If the files in a directory were modified many days or months ago, it may be harder to tell exactly when they were modified, as only the date is provided for older files. For more detailed modification time information you can use the --full-time option to display the complete timestamp (including hours, minutes, seconds). It will assume the -l option automatically:

```
sysadmin@localhost:~$ ls -t --full-time /etc/ssh
total 580
-rw------- 1 root root    227 2018-07-19 06:52:16.000000000 +0000 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 2018-07-19 06:52:16.000000000 +0000 ssh_host_ecdsa_key.pub
-rw------- 1 root root    411 2018-07-19 06:52:16.000000000 +0000 ssh_host_ed25519_key
-rw-r--r-- 1 root root     99 2018-07-19 06:52:16.000000000 +0000 ssh_host_ed25519_key.pub
-rw------- 1 root root   1679 2018-07-19 06:52:16.000000000 +0000 ssh_host_rsa_key
-rw-r--r-- 1 root root    399 2018-07-19 06:52:16.000000000 +0000 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    338 2018-07-19 06:52:16.000000000 +0000 ssh_import_id
-rw-r--r-- 1 root root 553122 2018-02-10 02:31:46.000000000 +0000 moduli
-rw-r--r-- 1 root root   1580 2018-02-10 02:31:46.000000000 +0000 ssh_config
-rw-r--r-- 1 root root   3264 2018-02-10 02:31:46.000000000 +0000 sshd_config
```

It is possible to perform a reverse sort by using the -r option. It can be used alone, or combined with either the -S or -t options. The following command will sort files by size, smallest to largest:

```
sysadmin@localhost:~$ ls -lrS /etc/ssh
total 580
-rw-r--r-- 1 root root     99 Jul 19 06:52 ssh_host_ed25519_key.pub
-rw-r--r-- 1 root root    179 Jul 19 06:52 ssh_host_ecdsa_key.pub
-rw------- 1 root root    227 Jul 19 06:52 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    338 Jul 19 06:52 ssh_import_id
-rw-r--r-- 1 root root    399 Jul 19 06:52 ssh_host_rsa_key.pub
-rw------- 1 root root    411 Jul 19 06:52 ssh_host_ed25519_key
-rw-r--r-- 1 root root   1580 Feb 10  2018 ssh_config
-rw------- 1 root root   1679 Jul 19 06:52 ssh_host_rsa_key
-rw-r--r-- 1 root root   3264 Feb 10  2018 sshd_config
-rw-r--r-- 1 root root 553122 Feb 10  2018 moduli
```

The following command will list files by modification date, oldest to newest:

```
sysadmin@localhost:~$ ls -lrt /etc/ssh
total 580
-rw-r--r-- 1 root root   3264 Feb 10  2018 sshd_config
-rw-r--r-- 1 root root   1580 Feb 10  2018 ssh_config
-rw-r--r-- 1 root root 553122 Feb 10  2018 moduli
-rw-r--r-- 1 root root    338 Jul 19 06:52 ssh_import_id
-rw-r--r-- 1 root root    399 Jul 19 06:52 ssh_host_rsa_key.pub
-rw------- 1 root root   1679 Jul 19 06:52 ssh_host_rsa_key
-rw-r--r-- 1 root root     99 Jul 19 06:52 ssh_host_ed25519_key.pub
```

```
-rw------- 1 root root   411 Jul 19 06:52 ssh_host_ed25519_key
-rw-r--r-- 1 root root   179 Jul 19 06:52 ssh_host_ecdsa_key.pub
-rw------- 1 root root   227 Jul 19 06:52 ssh_host_ecdsa_key
```

LAB 7

7.2.1 Step 1

Type the following command to print the working directory:

```
pwd
```

```
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

The working directory is the directory that your terminal window is currently "in". This is also called the current directory. This will be important for when you are running subsequent commands, as they will behave differently based on the directory you are currently in.

The output of the command (`/home/sysadmin` in the example above) is called the path. The first slash represents the root directory, the top level of the directory structure.

In the output above, `home` is a directory under the root directory and `sysadmin` is a directory under the `home` directory.

When you first open a terminal window, you will be placed in your home directory. This is a directory where you have full access and other users normally have no access by default. To see the path to your home directory, you can execute the following command to view the value of the `HOME` variable:

```
echo $HOME
```

```
sysadmin@localhost:~$ echo $HOME
/home/sysadmin
sysadmin@localhost:~$
```

7.2.2 Step 2

You can use the `cd` command with a path to a directory to change your current directory. Type the following command to make the root directory your current working directory and verify with the `pwd` command:

```
cd /
pwd
```

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

7.2.3 Step 3

To change back to your home directory, the `cd` command can be executed without a path. Change back to your home directory and verify by typing the following commands:

```
cd
pwd
```

```
sysadmin@localhost:/$ cd
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

Notice the change in the prompt. The tilde ~ character represents your home directory. This part of the prompt will tell you what directory you are currently in.

7.2.4 Step 4

The `cd` command may be entered with a path to a directory specified as an argument. Execute the `cd` command with the `/home` directory as an argument by typing the following:

```
cd /home
pwd
```

```
sysadmin@localhost:~$ cd /home
sysadmin@localhost:/home$ pwd
/home
sysadmin@localhost:/home$
```

When the path that is provided as an argument to the `cd` command starts with the forward slash `/`, that path is referred to as an “absolute path”. Absolute paths are always complete paths from the root directory to a subdirectory or file.

7.2.5 Step 5

Change back to your home directory, using the `cd` command with the tilde ~ as an argument:

```
cd ~
pwd
```

```
sysadmin@localhost:/home$ cd ~
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```
When the path that is provided as an argument to the `cd` command starts with a tilde ~ character, the terminal will expand the character to the home directory of a user with an account on the system. If either no other characters or a forward slash follows the tilde, then it will expand to the home directory of **the user currently active** in the shell.
If a user name immediately follows the tilde character, then the shell will expand the tilde and user name to the home directory of that user name. For example, `~bob` would be expanded to `/home/bob`.
Paths that start with a tilde are considered absolute paths because after the shell expands the tilde path, an absolute path is formed.
7.2.6 Step 6
Use the `echo` command below to display some other examples of using the tilde as part of the path:
```
echo  ~ ~sysadmin ~root ~mail ~nobody
sysadmin@localhost:~$ echo ~ ~sysadmin ~root ~mail ~nobody
/home/sysadmin /home/sysadmin /root /var/mail /nonexistent
sysadmin@localhost:~$
```
7.2.7 Step 7
Attempt to change to the home directory of the `root` user by typing the following command:
```
cd ~root
sysadmin@localhost:~$ cd ~root
-bash: cd: /root: Permission denied
sysadmin@localhost:~$
```
Notice the error message; it indicates that the shell attempted to execute `cd` with `/root` as an argument but it failed due to permission being denied. You will learn more about file and directory permissions in a later lab.
7.2.8 Step 8
Using an absolute path, change to the `/usr/bin` directory and display the working directory by using the following commands:
```
cd /usr/bin
pwd
sysadmin@localhost:~$ cd /usr/bin
sysadmin@localhost:/usr/bin$ pwd
/usr/bin
sysadmin@localhost:/usr/bin$
```
7.2.9 Step 9
Use an absolute path to change to the `/usr` directory and display the working directory by issuing the following commands:
```
cd /usr
pwd
sysadmin@localhost:/usr/bin$ cd /usr
sysadmin@localhost:/usr$ pwd
/usr
sysadmin@localhost:/usr$
```
7.2.10 Step 10
Use an absolute path to change to the `/usr/share/doc` directory and display the working directory by issuing the following commands:
```
cd /usr/share/doc
pwd
sysadmin@localhost:/usr$ cd /usr/share/doc
sysadmin@localhost:/usr/share/doc$ pwd
/usr/share/doc
sysadmin@localhost:/usr/share/doc$
```
Absolute vs. Relative pathnames
Suppose you are in the `/usr/share/doc` directory and you want to go to the `/usr/share/doc/bash` directory. Typing the command `cd /usr/share/doc/bash` results in a fair amount of typing. In cases like this, you want to use relative pathnames.
With relative pathnames you provide "directions" of where you want to go from the current directory. The following examples will illustrate using relative pathnames.
7.2.11 Step 11
Using a relative path, change to the `/usr/share/doc/bash` directory and display the working directory by issuing the following commands:
```
cd bash
pwd
```

```
sysadmin@localhost:/usr/share/doc$ cd bash
sysadmin@localhost:/usr/share/doc/bash$ pwd
/usr/share/doc/bash
sysadmin@localhost:/usr/share/doc/bash$
```

Note
If there wasn't a bash directory under the current directory, the previous command would fail.

7.2.12 Step 12
Use a relative path to change to the directory above the current directory:

```
cd ..
pwd
```

```
sysadmin@localhost:/usr/share/doc/bash$ cd ..
sysadmin@localhost:/usr/share/doc$ pwd
/usr/share/doc
sysadmin@localhost:/usr/share/doc$
```

The `..` represents one level above your current directory location.

7.2.13 Step 13
Use a relative path to change up one level from the current directory and then down into the `dict` directory:

```
cd ../dict
pwd
```

```
sysadmin@localhost:/usr/share/doc$ cd ../dict
sysadmin@localhost:/usr/share/dict$ pwd
/usr/share/dict
sysadmin@localhost:/usr/share/dict$
```

7.3 Listing Files and Directories
In this task, you will explore how to list files and directories.

7.3.1 Step 1
To list the contents of the current directory, use the `ls` command:

```
cd
ls
```

Your output should be similar to the following:

```
sysadmin@localhost:/usr/share/dict$ cd
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$
```

In the output of the previous `ls` command, the file names were placed in a light blue color. This is a feature that many distributions of Linux automatically provide through a feature called an alias (more on this feature in a later lab).
The color indicates what type the item is. The following table describes some of the more common colors:

Color	Type of File
Black or White	Regular file
Blue	Directory file
Cyan	Symbolic link file (a file that points to another file)
Green	Executable file (a program)

7.3.2 Step 2
Not all files are displayed by default. There are files, called hidden files, that are not displayed by default. To display all files, including hidden files, use the `-a` option to the `ls` command:

```
ls -a
```

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

The names of hidden files begin with a period (a dot character). Typically these files and often directories are hidden because they are not files you normally want to see.
For example, the `.bashrc` file shown in the example above contains configuration information for the bash shell. This is a file that you normally don't need to view on a regular basis.

Two important "dot files" exist in every directory: `.` (which represents the current directory) and `..` (which represents the directory above the current directory).

7.3.3 Step 3

By itself, the `ls` command just provided the names of the files and directories within the specified (or current) directory. Execute the following command to see how the `-l` option provides more information about a file:

```
ls -l /etc/hosts
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -l /etc/hosts
-rw-r--r-- 1 root root 150 Jan 22 15:18 /etc/hosts
sysadmin@localhost:~$
```

So, what does all of this extra output mean? The following table provides a brief breakdown of what each part of the output of `ls -l` means:

`-`	The first character, a `-` in the previous example, indicates what type of "file" this is. A `-` character is for a plain file while a `d` character would be for a directory.
`rw-r--r--`	This represents the permissions of the file. Permissions are discussed in a later lab.
`1`	This represents something called a hard link count (discussed later).
`root`	The user owner of the file.
`root`	The group owner of the file.
`150`	The size of the file in bytes
`Jan 22 15:18`	The date/time when the file was last modified.

7.3.4 Step 4

Sometimes you want to see not only the contents of a directory, but also the contents of the subdirectories. You can use the `-R` option to accomplish this:

```
ls -R /etc/udev
```

```
sysadmin@localhost:~$ ls -R /etc/udev
/etc/udev:
rules.d  udev.conf

/etc/udev/rules.d:
70-persistent-cd.rules  README
sysadmin@localhost:~$
```

The `-R` option stands for "recursive". All of the files in the `/etc/udev` directory will be displayed as well as all of the files in each subdirectory, in this case the `rules.d` subdirectory.

Be careful of the `-R` option. Some directories are very, very large!

7.3.5 Step 5

You can use file globbing (wildcards) to limit which files or directories you see. For example, the * character can match "zero or more of any characters" in a filename. Execute the following command to display only the files that begin with the letter `s` in the `/etc` directory:

```
ls -d /etc/s*
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -d /etc/s*
/etc/securetty  /etc/sgml     /etc/shells  /etc/ssl         /etc/sysctl.conf
/etc/security   /etc/shadow   /etc/skel    /etc/sudoers     /etc/sysctl.d
/etc/services   /etc/shadow-  /etc/ssh     /etc/sudoers.d   /etc/systemd
sysadmin@localhost:~$
```

Note that the `-d` option prevents files from subdirectories from being displayed. It should always be used with the `ls` command when you are using file globbing.

7.3.6 Step 6

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:~$
```

7.3.7 Step 7
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]*`
Your output should be similar to the following:

```
sysadmin@localhost:~$ ls -d /etc/[abcd]*
/etc/adduser.conf            /etc/blkid.conf            /etc/cron.weekly
/etc/adjtime                 /etc/blkid.tab             /etc/crontab
/etc/alternatives            /etc/ca-certificates       /etc/dbus-1
/etc/apparmor.d              /etc/ca-certificates.conf  /etc/debconf.conf
/etc/apt                     /etc/calendar              /etc/debian_version
/etc/bash.bashrc             /etc/cron.d                /etc/default
/etc/bash_completion.d       /etc/cron.daily            /etc/deluser.conf
/etc/bind                    /etc/cron.hourly           /etc/depmod.d
/etc/bindresvport.blacklist  /etc/cron.monthly          /etc/dpkg
sysadmin@localhost:~$
```

8.1 Introduction
When working in a Linux Operating System, it is important to know how to manipulate files and directories. Some Linux distributions have GUI-based applications that allow you to manage files, but it is advantageous to know how to perform these operations via the command line.
It is important to note that everything in Linux is case-sensitive, a feature carried over from Unix. This means that the shell recognizes a lowercase character as completely different from an uppercase A character. When manipulating files, pay attention to your capitalization: the `hello.txt` file is different from the `HELLO.txt` and `Hello.txt` files.
8.2 Globbing
Glob characters are often referred to as wild cards. These are symbol characters that have special meaning to the shell.
Globs are powerful because they allow you to specify patterns that match filenames in a directory. So instead of manipulating a single file at a time, you can easily execute commands that affect many files. For instance, by using glob characters, it is possible to manipulate all files with a specific extension or with a particular filename length.
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.
The examples provided in this chapter use the `echo` command for demonstration.
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
```

The pattern `t*` matches any file in the `/etc` directory that begins with the character `t` followed by zero or more of any character. In other words, any files that begin with the letter `t`.
You can use the asterisk character at any place within the filename pattern. For example, the following matches any filename in the `/etc` directory that ends with `.d`:

```
sysadmin@localhost:~$ echo /etc/*.d
```

```
/etc/apparmor.d /etc/binfmt.d /etc/cron.d /etc/depmod.d /etc/init.d
/etc/insserv
.conf.d /etc/ld.so.conf.d /etc/logrotate.d /etc/modprobe.d
/etc/modules-load.d /
etc/pam.d /etc/profile.d /etc/rc0.d /etc/rc1.d /etc/rc2.d /etc/rc3.d
/etc/rc4.d
/etc/rc5.d /etc/rc6.d /etc/rcS.d /etc/rsyslog.d /etc/sudoers.d
/etc/sysctl.d /et
c/tmpfiles.d /etc/update-motd.d
```

In the next example, all of the files in the `/etc` directory that begin with the letter `r` and end with `.conf` are displayed:

```
sysadmin@localhost:~$ echo /etc/r*.conf
/etc/resolv.conf /etc/rsyslog.conf
```

8.2.2 Question Mark ? Character

The question mark ? character represents any single character. Each question mark character matches exactly one character, no more and no less.

Suppose you want to display all of the files in the `/etc` directory that begin with the letter `t` and have exactly 7 characters after the `t` character:

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

Glob characters can be used together to find even more complex patterns. The pattern `/etc/*??????????????????` command only matches files in the `/etc` directory with twenty or more characters in the filename:

```
sysadmin@localhost:~$ echo /etc/*??????????????????
/etc/bindresvport.blacklist /etc/ca-certificates.conf
```

The asterisk and question mark could also be used together to look for files with three-letter extensions by using the `/etc/*.???` pattern:

```
sysadmin@localhost:~$ echo /etc/*.???
/etc/issue.net /etc/locale.gen
```

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/gshadow-
/etc/
gss /etc/ucf.conf /etc/udev /etc/ufw /etc/update-motd.d /etc/updatedb.conf
```

Brackets can also be used to a represent a range of characters. For example, the `/etc/[a-d]*` pattern matches all files that begin with any letter between and including a and d:

```
sysadmin@localhost:~$ echo /etc/[a-d]*
/etc/adduser.conf /etc/alternatives /etc/apparmor /etc/apparmor.d /etc/apt
/etc/
bash.bashrc /etc/bind /etc/bindresvport.blacklist /etc/binfmt.d
/etc/ca-certific
ates /etc/ca-certificates.conf /etc/calendar /etc/console-setup /etc/cron.d
/etc
/cron.daily /etc/cron.hourly /etc/cron.monthly /etc/cron.weekly
/etc/crontab /et
c/dbus-1 /etc/debconf.conf /etc/debian_version /etc/default
/etc/deluser.conf /e
tc/depmod.d /etc/dhcp /etc/dpkg
```

The `/etc/*[0-9]*` pattern displays any file that contains at least one number:

```
sysadmin@localhost:~$ echo /etc/*[0-9]*
/etc/X11 /etc/dbus-1 /etc/iproute2 /etc/mke2fs.conf /etc/python3
/etc/python3.6
/etc/rc0.d /etc/rc1.d /etc/rc2.d /etc/rc3.d /etc/rc4.d /etc/rc5.d
/etc/rc6.d
```

The range is based on the **ASCII** text table. This table defines a list of characters, arranging them in a specific standard order. If an invalid order is provided, no matches will be made:

```
sysadmin@localhost:~$ echo /etc/*[9-0]*
```

```
/etc/*[9-0]*
```

Note: The ASCII text table can be viewed in our virtual machines by executing the `ascii` command.

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/updatedb.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.

Keep in mind that it is the shell, not the `echo` or `ls` command, that expands the glob pattern into corresponding file names. In other words, if the `echo /etc/a*` command is executed, what the shell did before executing the echo command was replace `a*` with all of the files and directories within the `/etc` directory that match the pattern.

So, if the `ls /etc/a*` command is run, what the shell would really run would be this:

```
ls /etc/adduser.conf  /etc/alternatives  /etc/apparmor  /etc/apparmor.d
/etc/apt
```

When the `ls` command sees multiple arguments, it performs a list operation on each item separately. In other words, `ls /etc/a*` is the same as running the following commands consecutively:

```
ls /etc/adduser.conf
ls /etc/alternatives
ls /etc/apparmor
ls /etc/apparmor.d
ls /etc/apt
```

Now consider what happens the `ls` command is passed a file, such as `/etc/adduser.conf`:

```
sysadmin@localhost:~$ ls /etc/adduser.conf
/etc/adduser.conf
```

Running the `ls` command on a single file results in the name of the file being printed; typically this is useful if the `-l` option is used to see details about a specific file:

```
sysadmin@localhost:~$ ls -l /etc/adduser.conf
-rw-r--r-- 1 root root 3028 May 26  2018 /etc/adduser.conf
```

However, what if the `ls` command is given a directory name as an argument? In this case, the output of the command is different than if the argument was a regular file:

```
sysadmin@localhost:~$ ls /etc/apparmor
init  parser.conf  subdomain.conf
```

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.named
cache         force-complain  sbin.dhclient  usr.bin.man  usr.sbin.rsyslogd

/etc/apt:
apt.conf.d  preferences.d  sources.list  sources.list.d  trusted.gpg.d
```

When the `ls` command sees a filename as an argument, it just displays the filename. However, for any directory, it displays the contents of the directory, not just the directory name.

This becomes even more confusing in a situation like the following:

```
sysadmin@localhost:~$ ls /etc/x*
autostart  systemd  user-dirs.conf  user-dirs.defaults
```

In the previous example, it seems like the `ls` command is just plain wrong. However, what really happened is that the only thing that matches the glob `/etc/x*` is the `/etc/xdg` directory.

So, the `ls` command only displayed the files in that directory!
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]
```

The source is the file to be copied. The destination is where the copy is to be located. When successful, the `cp` command does not have any output (no news is good news). The following command copies the `/etc/hosts` file to your home directory:

```
sysadmin@localhost:~$ cp /etc/hosts ~
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
Documents  Music      Public    Videos
```

Reminder: The ~ character represents your home directory.
8.3.1 Verbose Mode
The `-v` option causes the `cp` command to produce output if successful. The `-v` option stands for verbose:

```
sysadmin@localhost:~$ cp -v /etc/hosts ~
`/etc/hosts' -> `/home/sysadmin/hosts'
```

When the destination is a directory, the resulting new file keeps the same name as the original file. To give the new file a different name, provide the new name as part of the destination:

```
sysadmin@localhost:~$ cp /etc/hosts ~/hosts.copy
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
Documents  Music      Public    Videos     hosts.copy
```

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.
To illustrate this potential problem, first a new file is created in the home directory by copying an existing file:

```
sysadmin@localhost:~$ cp /etc/hostname example.txt
```

View the information about the file with `ls` command:

```
sysadmin@localhost:~$ ls -l example.txt
-rw-r--r-- 1 sysadmin sysadmin 10 Dec 15 22:55 example.txt
```

View the contents of the file using the `cat` command:

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

In the next example, the `cp` command destroys the original contents of the `example.txt` file:

```
sysadmin@localhost:~$ cp /etc/timezone example.txt
```

Notice that after the `cp` command is complete, the size of the file has changed and the contents are different:

```
sysadmin@localhost:~$ ls -l example.txt
-rw-r--r-- 1 sysadmin sysadmin 8 Dec 15 22:58 example.txt
sysadmin@localhost:~$ cat example.txt
Etc/UTC
```

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 example.txt
cp: overwrite `/home/sysadmin/example.txt'? n
```

If a value of y (yes) were given, then the copy process would have taken place. However, the value of n (no) was given when prompted to overwrite the file, so no changes were made to the file.

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 `-r` option has the `cp` command copy both files and directories.
Be careful with this option. The entire directory structure will be copied which could result in copying a lot of files and directories!
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]`
In the following example, the `hosts` file that was generated previously is moved from the current directory to the `Videos` directory:

```
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
```

Permissions will be covered in detail later in the course.
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:~$ ls
Desktop    Downloads  Pictures  Templates  example.txt
Documents  Music      Public    Videos     hosts.copy
sysadmin@localhost:~$ mv example.txt Videos/newexample.txt
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts.copy
Documents  Music      Public    Videos
sysadmin@localhost:~$ ls Videos
hosts  newexample.txt
```

If a destination directory is not specified, the file is renamed using the destination file name and remains in the source directory. For example, the following commands renames the `newexample.txt` file to `myfile.txt`:

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

Think of the previous `mv` example to mean move the `newexample.txt` file from the current directory to the current directory and give the new file the name `myfile.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.

Important: There is no `-r` option as the `mv` command moves directories by default.

8.5 Creating Files

There are several ways of creating a new file, including using a program designed to edit a file (a text editor).

There is also a way to create an empty file that can be populated with data at a later time. This feature is useful for some operating systems as the very existence of a file could alter how a command or service works. It is also useful to create a file as a "placeholder" to remind you to create the file contents at a later time.

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.

Text editors will be covered in detail later in the course.

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  Videos
```

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
sample.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
sample.txt
sysadmin@localhost:~$ mkdir test
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates   test
Documents  Music      Public    sample.txt
```

LAB *:-

8.1 Introduction

This is Lab 8: Managing Files and Directories. By performing this lab, students will learn how to navigate and manage files and directories.

In this lab, you will perform the following tasks:

- Understand how to use globbing
- Creating, moving and deleting files and directories

8.2 Globbing

The use of glob characters in Linux is similar to what many operating systems refer to as "wildcard" characters. Using glob characters, you match filenames using patterns.

Glob characters are a shell feature, not something that is particular to any specific command. As a result, you can use glob characters with any Linux command.

When glob characters are used, the shell will "expand" the entire pattern to match all files in the specified directory that match the pattern.

For demonstration purposes, we will use the `echo` command to display this expansion process.

8.2.1 Step 1

Use the following `echo` command to display all filenames in the current directory that match the glob pattern `*`:

```
echo *
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo *
Desktop Documents Downloads Music Pictures Public Templates Videos
sysadmin@localhost:~$
```

The asterisk * matches "zero or more" characters in a file name. In the example above, this results in matching all filenames in the current directory.

The `echo` command, in turn, displays the filenames that were matched.

8.2.2 Step 2

The following commands will display all the files in the current directory that start with the letter `D`, and the letter `P`:

```
echo D*
echo P*
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
```

```
sysadmin@localhost:~$ echo P*
Pictures Public
sysadmin@localhost:~$
```

Think of the first example, `D*`, as "match all filenames in the current directory that begin with a capital d character and have zero or more of any other character after the `D`".

8.2.3 Step 3

The asterisk * can be used anywhere in the string. The following command will display all the files in your current directory that end in the letter `s`:

```
echo *s
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo *s
Documents Downloads Pictures Templates Videos
sysadmin@localhost:~$
```

8.2.4 Step 4

Notice that the asterisk can also appear multiple times or in the middle of several characters:

```
echo D*n*s
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo D*n*s
Documents Downloads
sysadmin@localhost:~$
```

The next glob metacharacter that we will examine is the question mark `?`. The question mark matches exactly one character. This single character can be any possible character.

Like the asterisk, it can be used anywhere in a string and can appear multiple times.

8.2.5 Step 5

Since each question mark matches one unknown character, typing six of them will match six-character filenames. Type the following to display the filenames that are exactly six characters long:

```
echo ??????
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo ??????
Public Videos
sysadmin@localhost:~$
```

Important

Each ? character must match exactly one character in a filename-no more and no less than one character.

8.2.6 Step 6

Using the question mark with other characters will limit the matches. Type the following to display the file names that start with the letter `D` and are exactly nine characters long:

```
echo D????????
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo D????????
Documents Downloads
sysadmin@localhost:~$
```

8.2.7 Step 7

Wildcards or glob characters can be combined together. The following command will display file names that are at least six characters long and end in the letter `s`.

```
echo ?????*s
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo ?????*s
Documents Downloads Pictures Templates Videos
sysadmin@localhost:~$
```

Think of the pattern `?????*s` to mean "match filenames that begin with any five characters, then have zero or more of any characters and then end with an `s` character".

8.2.8 Step 8

The next glob is similar to the question mark glob to specify one character.

This glob uses a pair of square brackets `[ ]` to specify which one character will be allowed. The allowed characters can be specified as a range, a list, or by what is known as a character class. The allowed characters can also be negated with an exclamation point `!`.

In the first example, the first character of the file name can be either a `D` or a `P`. In the second example, the first character can be any character except a `D` or `P`:

```
echo [DP]*
echo [!DP]*
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo [DP]*
Desktop Documents Downloads Pictures Public
sysadmin@localhost:~$ echo [!DP]*
```

```
Music Templates Videos
sysadmin@localhost:~$
```
8.2.9 Step 9
In these next examples, a range of characters will be specified. In the first example, the first character of the file name can be any character starting at `D` and ending at `P`. In the second example, this range of characters is negated, meaning any single character will match as long as it is not between the letters `D` and `P`:
```
echo [D-P]*
echo [!D-P]*
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ echo [D-P]*
Desktop Documents Downloads Music Pictures Public
sysadmin@localhost:~$ echo [!D-P]*
Templates Videos
sysadmin@localhost:~$
```
You may be asking yourself "who decides what letters come between `D` and `P`"? In this case, the answer is fairly obvious (E, F, G, H, I, J, K, L, M, N and O), but what if the range was `[1-A]`?
The ASCII text table is used to determine the range of characters. You can view this table by searching for it on the Internet or typing the following command: `ascii`
So, what characters does the glob `[1-A]` match? According to the ASCII text table: `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `:`, `;`, `<`, `=`, `>`, `?`, `@` and `A`.
8.3 Copying, Moving and Renaming Files and Directories
In this task, you will copy, move, and remove files and directories.
8.3.1 Step 1
Make a copy of the `/etc/hosts` file and place it in the current directory. Then, list the contents of the current directory before and after the copy:
```
ls
cp /etc/hosts hosts
ls
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cp /etc/hosts hosts
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
Documents  Music      Public    Videos
sysadmin@localhost:~$
```
Notice how the second `ls` command displays a copy of the `hosts` file.
8.3.2 Step 2
Next, you will remove the file, then copy it again, but have the system tell you what is being done. This can be achieved using the `-v` or `--verbose` option. Enter the following commands:
```
rm hosts
ls
cp –v /etc/hosts hosts
ls
```
Note that the `rm` command is used to delete a file. More information on this command will be provided later in this lab.
Your output should be similar to the following:
```
sysadmin@localhost:~$ rm hosts
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cp -v /etc/hosts hosts
`/etc/hosts' -> `hosts'
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
Documents  Music      Public    Videos
sysadmin@localhost:~$
```
Note that the `-v` switch displays the source and target when the `cp` command is executed:
Source
```
`/etc/hosts'-> `hosts'
```
Target
```
`/etc/hosts'-> `hosts'
```
8.3.3 Step 3
Enter the following commands to copy the `/etc/hosts` file, using the period `.` character to indicate the current directory as the target:
```
rm hosts
```

```
ls
cp –v /etc/hosts .
       ls
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ rm hosts
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cp -v /etc/hosts .
`/etc/hosts' -> `hosts'
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  hosts
Documents  Music      Public    Videos
sysadmin@localhost:~$
```

The period . character is a handy way to say "the current directory". It can be used with all Linux commands, not just the `cp` command.

8.3.4 Step 4

Enter the following commands to copy from the source directory and preserve file attributes by using the `-p` option:

```
rm hosts
ls
cd /etc
ls -l hosts
cp –p hosts /home/sysadmin
cd
       ls –l hosts
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ rm hosts
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cd /etc
sysadmin@localhost:/etc$ ls -l hosts
-rw-r--r-- 1 root root 150 Jan 22 15:18 hosts
sysadmin@localhost:/etc$ cp -p hosts /home/sysadmin
sysadmin@localhost:/etc$ cd
sysadmin@localhost:~$ ls -l hosts
-rw-r--r-- 1 sysadmin sysadmin 150 Jan 22 15:18 hosts
sysadmin@localhost:~$
```

Notice that the date and permission modes were preserved. Note that the timestamp in the output above is the same for both the original and the copy (`Jan 22 15:18`) in the example provided above. Your output may vary.

8.3.5 Step 5

Type the following commands to copy using a different target name:

```
rm  hosts
cp -p /etc/hosts ~
cp hosts newname
ls –l hosts newname
rm hosts newname
```

```
sysadmin@localhost:~$ rm hosts
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ cp -p /etc/hosts ~
sysadmin@localhost:~$ cp hosts newname
sysadmin@localhost:~$ ls -l hosts newname
-rw-r--r-- 1 sysadmin sysadmin 150 Jan 22 15:18 hosts
-rw-r--r-- 1 sysadmin sysadmin 150 Jan 22 16:29 newname
sysadmin@localhost:~$ rm hosts newname
sysadmin@localhost:~$
```

The first copy with the `-p` option preserved the original timestamp. Recall that the tilde ~ represents your home directory (`/home/sysadmin`).

The second copy specified a different filename (`newname`) as the target. Because it was issued without the `-p` option, the system used the current date and time for the target; thus, it did not preserve the original timestamp found in the source file `/etc/hosts`.

Finally, note that you can remove more than one file at a time as shown in the last `rm` command.

8.3.6 Step 6

To copy all files in a directory, use the `-R` option. For this task, we will copy the `/etc/udev` directory into a new directory and display the contents that were copied there. Naturally, the directory must be created before files can be added to it. In this example we will use the default settings for `mkdir` to create the “Myetc” directory. Options are available for the `mkdir` command to set security, permissions and other attributes of a new directory. Once the directory has been copied, the `ls` command is used to list the contents of the directory with both the long and recursive options.

```
mkdir Myetc
cp –R /etc/udev Myetc
ls –l Myetc
ls –lR Myetc
```

```
sysadmin@localhost:~$ mkdir Myetc
sysadmin@localhost:~$ cp -R /etc/udev Myetc
sysadmin@localhost:~$ ls -l Myetc
total 0
drwxr-xr-x 1 sysadmin sysadmin 32 Jan 22 16:35 udev
sysadmin@localhost:~$ ls -lR Myetc
Myetc:
total 0
drwxr-xr-x 1 sysadmin sysadmin 32 Jan 22 16:35 udev

Myetc/udev:
total 4
drwxr-xr-x 1 sysadmin sysadmin  56 Jan 22 16:35 rules.d
-rw-r--r-- 1 sysadmin sysadmin 218 Jan 22 16:35 udev.conf

Myetc/udev/rules.d:
total 8
-rw-r--r-- 1 sysadmin sysadmin  306 Jan 22 16:35 70-persistent-cd.rules
-rw-r--r-- 1 sysadmin sysadmin 1157 Jan 22 16:35 README
sysadmin@localhost:~$
```

8.3.7 Step 7

To remove a directory, use the `-r` option to the `rm` command:

```
ls
rm -r Myetc
ls
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ ls
Desktop    Downloads  Myetc     Public     Videos
Documents  Music      Pictures  Templates
sysadmin@localhost:~$ rm -r Myetc
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$
```

Note that the `rmdir` command can also be used to delete directories, but only if the directory is empty (if it contains no files).

Also note the `-r` option. This option removes directories and their contents recursively.

8.3.8 Step 8

Moving a file is analogous to a "cut and paste". The file is “cut” (removed) from the original location and “pasted” to the specified destination. Move a file in the local directory by executing the following commands:

```
touch premove
ls
mv premove postmove
ls
rm postmove
```

Linux Command	Description
`touch premove`	Creates an empty file called `premove`
`mv premove postmove`	This command “cuts” the `premove` file and “pastes” it to a file called `postmove`

`rm postmove` Removes `postmove` file

Your output should be similar to the following:

```
sysadmin@localhost:~$ touch premove
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  premove
Documents  Music      Public    Videos
sysadmin@localhost:~$ mv premove postmove
sysadmin@localhost:~$ ls
 Desktop    Downloads  Pictures  Templates  postmove
Documents  Music      Public    Videos
sysadmin@localhost:~$
```

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.

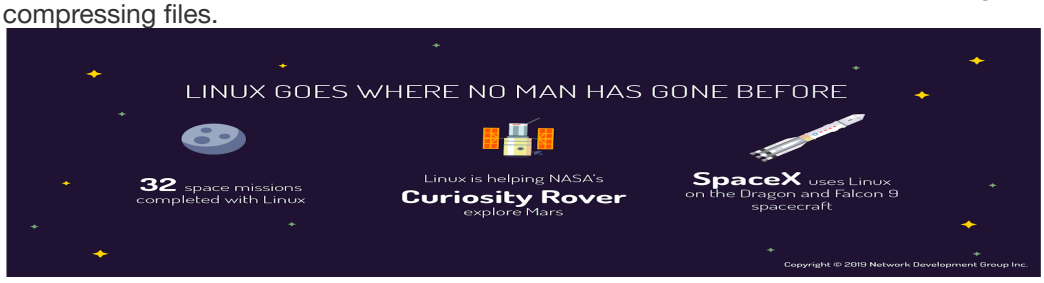
Consider This

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.

When an archive is decompressed, and one or more files are extracted, this is called un-archiving. Even though disk space is relatively cheap, archiving and compression still have value:

- When making a large number of files available, such as the source code to an application or a collection of documents, it is easier for people to download a compressed archive than it is to download files individually.
- Log files have a habit of filling disks, so it is helpful to split them by date and compress older versions.
- When backing up directories, it is easier to keep them all in one archive than it is to version (update) each file.
- Some streaming devices such as tapes perform better if you're sending a stream of data rather than individual files.
- It can often be faster to compress a file before sending it to a tape drive or over a slower network and decompress it at the other end than it would be to send it uncompressed.

It is essential for Linux administrators to become familiar with the tools for archiving and compressing files.



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.
Most image formats, such as GIF, PNG, and JPEG, implement some form of lossy compression. You can generally decide how much quality you want to preserve. A lower quality results in a smaller file, but after decompression, you may notice artifacts such as rough edges or discolorations. High quality will look much like the original image, but the file size will be closer to the original.
Compressing an already compressed file will not make it smaller. This fact is often forgotten when it comes to images since they are already stored in a compressed format. With lossless compression, this multiple compression is not a problem, but if you compress and decompress a file several times using a lossy algorithm, you will eventually have something that is unrecognizable.
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
```

In the preceding example, there is a file called `longfile.txt` that is `66540` bytes. The file is compressed by invoking the `gzip` command with the name of the file as the only argument. After that command completes, the original file is gone, and a compressed version with a file extension of `.gz` is left in its place. The file size is now 341 bytes.
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
sysadmin@localhost:~/Documents$ ls -l longfile*
-rw-r--r-- 1 sysadmin sysadmin 66540 Dec 20  2017 longfile.txt
```

Consider This
The `gunzip` command is just a script that calls `gzip` with the right parameters.
There are other commands that operate virtually identically to `gzip` and `gunzip`. There is `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.

Remembering the modes is key to figuring out the command line options necessary to do what you want. In addition to the mode, remember where to specify the name of the archive, as you may be entering multiple file names on a command line.

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.

All the remaining arguments are considered as input file names, either as a wildcard, a list of files, or both.
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
```

The final size of `alpha_files.tar` is `10240` bytes. Normally, tarball files are slightly larger than the combined input files due to the overhead information on recreating the original files. 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
```

The output is much smaller than the tarball itself, and the resulting file is compatible with `gzip`, which can be used to view the compression details. The uncompressed file is the same size as it would be if you tarred it in a separate step:

```
sysadmin@localhost:~/Documents$ gzip -l alpha_files.tar.gz
         compressed        uncompressed  ratio uncompressed_name
                417               10240  96.1% alpha_files.tar
```

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

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. Pipes and standard inputs will be covered in detail later in the course.

Note: The examples above are designed to demonstrate the `tar` command's ability to recurse into subdirectories. The files displayed in the example are not available within the virtual machine environment of this course.

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
```

The original file is untouched, and the new directory is created. Inside the directory, are the original directories and files.

```
sysadmin@localhost:~/Downloads$ cd School
sysadmin@localhost:~/Downloads/School$ ls -l
total 12
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:45 Art
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:47 Engineering
drwx------ 2 sysadmin sysadmin 4096 Oct 31 17:46 Math
```

Add the `-v` flag and you will get a verbose output of the files processed, making it easier to keep track of what's happening:

Option	Function
`-v`	Verbosely list the files processed.

The next example repeats the prior example, but with the addition of the `-v` option:

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

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/linux.txt
School/Art/linux.txt
```

The `tar` command has many more features, such as the ability to use patterns when extracting files, excluding certain files, or outputting the extracted files to the screen instead of disk. The documentation for `tar` has in-depth information.

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%)
```

The output shows the files and the compression ratio.
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%)
```

In the example above, all files under the `School` directory are added because it uses the `-r` option. The first lines of output indicate that directories were added to the archive, but otherwise the output is similar to the previous example.
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]ename: 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
```

Here, we extract all the files in the archive to the current directory. Just like `tar`, you can pass filenames on the command line. The examples below show three different attempts to extract a file.
First, just the name of the file is passed without the directory component. Like `tar`, the file is not matched.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip linux.txt
Archive:  School.zip
caution: filename not matched:  linux.txt
```

A second attempt passes the directory component along with the file name, which extracts just that file.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip School/Math/numbers.txt
Archive:  School.zip
 extracting: School/Math/numbers.txt
```

The third version uses a wildcard, which extracts the four files matching the pattern, just like `tar`.

```
sysadmin@localhost:~/Documents/tmp$ unzip School.zip School/Art/*t
Archive:  School.zip
  inflating: School/Art/linux.txt
  inflating: School/Art/red.txt
  inflating: School/Art/hidden.txt
  inflating: School/Art/animals.txt
```

The `zip` and `unzip` man pages describe the other things you can do with these tools, such as replace files within the archive, use different compression levels, and even use encryption.

9.1 Introduction

This is Lab 9: Archiving and Compression. By performing this lab, students will learn how to work with archive files.

In this lab, you will perform the following tasks:

- Create archive files using `tar` with and without compression
- Compress and uncompress files into a `gzip` archive file
- Compress and uncompress files into a `bzip2` archive file
- Compress and uncompress files into an `xz` archive file
- Use `zip` and `unzip` to compress and uncompress archive files

9.2 Archiving Commands

In this task, we will use `gzip`, `bzip2`, `tar`, `zip`/`unzip` and `xz`/`unxz` to archive and restore files. These commands are designed to either merge multiple files into a single file or compress a large file into a smaller one. In some cases, the commands will perform both functions.

The task of archiving data is important for several reasons, including but not limited to the following:

1. Large files may be difficult to transfer. Making these files smaller helps make transfer quicker.
2. Transferring multiple files from one system to another can become tedious when there are many files. Merging them into a single file for transport makes this process easier.
3. Files can quickly take up a lot of space, especially on smaller removable media like thumb drives. Archiving reduces this problem.

One potential area of confusion that a beginner Linux user may experience stems from the following question: why are there so many different archiving commands? The answer is that these commands have different features (for example, some of them allow you to password protect the archive file) and compression techniques used.

The most important thing to know for now is how these different commands function. Over time you will learn to pick the correct archive tool for any given situation.

9.2.1 Step 1

Use the following `tar` command to create an archive of the `/etc/udev` directory. Save the backup in the `~/mybackups` directory:

```
cd
mkdir mybackups
tar –cvf mybackups/udev.tar /etc/udev
ls mybackups
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ cd
sysadmin@localhost:~$ mkdir mybackups
sysadmin@localhost:~$ tar -cvf mybackups/udev.tar /etc/udev
tar: Removing leading `/' from member names
/etc/udev/
/etc/udev/udev.conf
/etc/udev/hwdb.d/
/etc/udev/rules.d/
sysadmin@localhost:~$ ls mybackups
udev.tar
sysadmin@localhost:~$
```

The `tar` command is used to merge multiple files into a single file. By default, it does not compress the data.

The `-c` option tells the `tar` command to create a `tar` file. The `-v` option stands for "verbose", which instructs the `tar` command to demonstrate what it is doing. The `-f` option is used to specify the name of the `tar` file.

Consider This

`tar` stands for **T**ape **AR**chive. This command was originally used to create tape backups, but today it is more commonly used to create archive files.

Important

You are not required to use the `.tar` extension to the archive file name, however, it is helpful for identifying the file type. It is considered "good style" when sending an archive file to another person.

9.2.2 Step 2

Display the contents of a `tar` file by using the available options (`t` = list contents, `v` = verbose, `f` = filename):

```
tar –tvf mybackups/udev.tar
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ tar -tvf mybackups/udev.tar
drwxr-xr-x root/root         0 2018-07-19 06:51 etc/udev/
-rw-r--r-- root/root       153 2018-04-20 16:55 etc/udev/udev.conf
drwxr-xr-x root/root         0 2018-04-20 16:55 etc/udev/hwdb.d/
drwxr-xr-x root/root         0 2018-04-20 16:55 etc/udev/rules.d/
sysadmin@localhost:~$
```

Notice that files were backed up recursively using relative path names. This is important because when you extract the files, they will be placed in your current directory, not override the current file.

9.2.4 Step 4

Extract the contents of an archive. Data is restored to the current directory by default:

```
cd mybackups
ls
tar –xvf udev.tar.gz
ls
ls etc
ls etc/udev
ls etc/udev/rules.d
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ cd mybackups
sysadmin@localhost:~/mybackups$ ls
udev.tar  udev.tar.gz
sysadmin@localhost:~/mybackups$ tar -xvf udev.tar.gz
etc/udev/
etc/udev/udev.conf
etc/udev/hwdb.d/
etc/udev/rules.d/
etc/udev/rules.d/70-persistent-cd.rules
etc/udev/rules.d/README
sysadmin@localhost:~/mybackups$ ls
```

```
etc  udev.tar  udev.tar.gz
sysadmin@localhost:~/mybackups$ ls etc
udev
sysadmin@localhost:~/mybackups$ ls etc/udev
hwdb.d  rules.d  udev.conf
sysadmin@localhost:~/mybackups$ ls etc/udev/rules.d
70-persistent-cd.rules  README
sysadmin@localhost:~/mybackups$
```

If you wanted the files to "go back" into their original location, you could first `cd` to the `/` directory and then run the `tar` command. However, in this example, this would require you to be logged in as an administrator because creating files in the `/etc` directory can only be done by the administrator.

9.2.5 Step 5

To add a file to an existing archive, use the `-r` option to the `tar` command. Execute the following commands to perform this action and verify the existence of the new file in the `tar` archive:

```
tar -rvf udev.tar /etc/hosts
tar –tvf udev.tar
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ tar -rvf udev.tar /etc/hosts
tar: Removing leading `/' from member names
/etc/hosts
sysadmin@localhost:~/mybackups$ tar -tvf udev.tar
drwxr-xr-x root/root         0 2018-07-19 06:51 etc/udev/
-rw-r--r-- root/root       153 2018-04-20 16:55 etc/udev/udev.conf
drwxr-xr-x root/root         0 2018-04-20 16:55 etc/udev/hwdb.d/
drwxr-xr-x root/root         0 2018-04-20 16:55 etc/udev/rules.d/
-rw-r--r-- root/root       172 2018-12-09 21:17 etc/hosts
sysadmin@localhost:~/mybackups$
```

9.2.6 Step 6

In the following examples, you will use `gzip` and `gunzip` to compress and uncompress a file. Execute the following commands to compress a copy of the words file:

```
cp /usr/share/dict/words .
ls -l words
gzip words
ls -l words.gz
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ cp /usr/share/dict/words .
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 938848 Jan 25 07:39 words
sysadmin@localhost:~/mybackups$ gzip words
sysadmin@localhost:~/mybackups$ ls -l words.gz
-rw-r--r-- 1 sysadmin sysadmin 255996 Jan 25 07:39 words.gz
sysadmin@localhost:~/mybackups$
```

Notice the size of the zipped file (255996 bytes in the example above) is much smaller than the original file (938848 bytes in the example above).

Very important

When you use `gzip`, the original file is replaced by the zipped file. In the example above, the `words` file was replaced with `words.gz`.

When you unzip the file, the zipped file will be replaced with the original file.

9.2.7 Step 7

Execute the following commands to uncompress the `words.gz` file:

```
ls -l words.gz
gunzip words.gz
ls -l words
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ ls -l words.gz
-rw-r--r-- 1 sysadmin sysadmin 259983 Dec 10 18:18 words.gz
sysadmin@localhost:~/mybackups$ gunzip words.gz
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 971578 Dec 10 18:18 words
sysadmin@localhost:~/mybackups$
```

Linux provides a large number of compression utilities in addition to `gzip`/`gunzip`. Each of them has pros and cons (faster compression, better compression rates, more flexible, more portable, faster decompression, etc.).

The `gzip`/`gunzip` commands are very popular in Linux, but you should be aware that `bzip2`/`bunzip2` are also popular on some Linux distributions. It is fortunate that most of the functionality (the way you run the commands) and options are the same as `gzip`/`gunzip`.

9.2.8 Step 8
Using `bzip2` and `bunzip2` to compress and uncompress a file is very similar to using `gzip` and `gunzip`. The compressed file is created with a `.bz2` extension. The extension is removed when uncompressed. Execute the following commands to compress a copy of the `words` file:

```
ls -l words
bzip2 words
ls -l words.bz2
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 971578 Dec 10 18:18 words
sysadmin@localhost:~/mybackups$ bzip2 words
sysadmin@localhost:~/mybackups$ ls -l words.bz2
-rw-r--r-- 1 sysadmin sysadmin 345560 Dec 10 18:18 words.bz2
```

If you compare the resulting `.bz2` file size (345560) to the `.gz` file size (259983) from Step 7, you will notice that `gzip` did a better job compressing this particular file.

9.2.9 Step 9
Execute the following commands to uncompress the `words.bz2` file:

```
ls -l words.bz2
bunzip2 words.bz2
ls -l words
```

```
sysadmin@localhost:~/mybackups$ ls -l words.bz2
-rw-r--r-- 1 sysadmin sysadmin 345560 Dec 10 18:18 words.bz2
sysadmin@localhost:~/mybackups$ bunzip2 words.bz2
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 971578 Dec 10 18:18 words
```

9.2.10 Step 10
Using `xz` and `unxz` to compress and uncompress a file is also very similar to using `gzip` and `gunzip`. The compressed file is created with a `.xz` extension. The extension is removed when uncompressed. Execute the following commands to compress a copy of the `words` file:

```
ls -l words
xz words
ls -l words.xz
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 971578 Dec  9 23:35 words
sysadmin@localhost:~/mybackups$ xz words
sysadmin@localhost:~/mybackups$ ls -l words.xz
-rw-r--r-- 1 sysadmin sysadmin 198756 Dec  9 23:35 words.xz
sysadmin@localhost:~/mybackups$
```

If you compare the resulting `.xz` file size (198756) to the `.gz` file size (259983) from Step 7, you will notice that `xz` did a better job compressing this particular file.

9.2.11 Step 11
Execute the following commands to uncompress the `words.xz` file:

```
ls -l words.xz
unxz words.xz
ls -l words
```

```
sysadmin@localhost:~/mybackups$ ls -l words.xz
-rw-r--r-- 1 sysadmin sysadmin 198756 Dec  9 23:35 words.xz
sysadmin@localhost:~/mybackups$ unxz words.xz
sysadmin@localhost:~/mybackups$ ls -l words
-rw-r--r-- 1 sysadmin sysadmin 971578 Dec  9 23:35 words
```

While `gzip`, `bzip` and `xz` archive files are commonly used in Linux, the `zip` archive type is more commonly used by other operating systems, such as Windows. In fact, the Windows Explorer application has built-in support to extract `zip` archive files.

Therefore, if you are planning to share an archive with Windows users, it is usually preferred to use the `zip` archive type. Unlike `gzip` and `bzip2`, when a file is compressed with the `zip` command, a copy of the original file is compressed and the original remains uncompressed.

9.2.12 Step 12
Use the `zip` command to compress the `words` file:

```
zip words.zip words
ls -l words.zip
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ zip words.zip words
  adding: words (deflated 73%)
sysadmin@localhost:~/mybackups$ ls -l words.zip
-rw-rw-r-- 1 sysadmin sysadmin 260119 Dec  9 23:48 words.zip
```

```
sysadmin@localhost:~/mybackups$
```
In the example above, the first argument (`words.zip`) of the `zip` command is the file name that you wish to create. The remaining arguments (`words`) are the files you want placed in the compressed file.

Important

You are not required to use the `.zip` extension to the compressed file name; however, it is helpful for determining the file type. It is also considered "good style" when sending an archive file to another person.

9.2.13 Step 13

Compress the `/etc/udev` directory and its contents with `zip` compression:

```
zip -r udev.zip /etc/udev
ls -l udev.zip
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ zip -r udev.zip /etc/udev
  adding: etc/udev/ (stored 0%)
  adding: etc/udev/udev.conf (deflated 24%)
  adding: etc/udev/hwdb.d/ (stored 0%)
  adding: etc/udev/rules.d/ (stored 0%)
sysadmin@localhost:~/mybackups$ ls -l udev.zip
-rw-rw-r-- 1 sysadmin sysadmin 771 Dec  9 23:50 udev.zip
sysadmin@localhost:~/mybackups$
```

The `tar` command discussed earlier in this lab automatically descends through any subdirectories of a directory specified to be archived. With the `bzip2`, `gzip`, and `zip` commands the `-r` option must be specified in order to perform recursion into subdirectories.

9.2.14 Step 14

To view the contents of a `zip` archive, use with the `-l` option with the `unzip` command:

```
unzip -l udev.zip
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ unzip -l udev.zip
Archive:  udev.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2018-07-19 06:51   etc/udev/
      153  2018-04-20 16:55   etc/udev/udev.conf
        0  2018-04-20 16:55   etc/udev/hwdb.d/
        0  2018-04-20 16:55   etc/udev/rules.d/
---------                     -------
      153                     4 files
sysadmin@localhost:~/mybackups$
```

9.2.15 Step 15

To extract the `zip` archive, use the `unzip` command without any options. In this example we first need to delete the files that were created in the earlier `tar` example:

```
rm -r etc
unzip udev.zip
```

Your output should be similar to the following:

```
sysadmin@localhost:~/mybackups$ rm -r etc
sysadmin@localhost:~/mybackups$ unzip udev.zip
Archive:  udev.zip
   creating: etc/udev/
  inflating: etc/udev/udev.conf
   creating: etc/udev/hwdb.d/
   creating: etc/udev/rules.d/
sysadmin@localhost:~/mybackups$
```


10.1 Introduction

A large number of the files in a typical file system are text files. Text files only contain text, no formatting features that you might see in a word processing file.

Because there are so many of these files on a typical Linux system, a significant number of commands exist to help users manipulate text files. There are commands to both view and modify these files in various ways.

Additionally, there are features available for the shell to control the output of commands, so instead of having the output placed in the terminal window, the output can be redirected into another file or another command. These redirection features provide users with a much more flexible and powerful environment to work within.

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. One of the most popular uses of `cat` is to display the content of text files. To display a file in the standard output using the `cat` command, type the command followed by the filename:

```
sysadmin@localhost:~$ cd Documents
sysadmin@localhost:~/Documents$ cat food.txt
Food is good.
```

Although the terminal is the default output of this command, the `cat` command can also be used for redirecting file content to other files or input for another command by using redirection characters.

10.1.2 Viewing Files Using a Pager

While viewing small files with the `cat` command poses no problems, it is not an ideal choice for large files. The `cat` command doesn't provide any easy ways to pause and restart the display, so the entire file contents are dumped to the screen.

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. There are two commonly used pager commands:

- 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.

The `more` and `less` commands allow users to move around the document using keystroke commands. Because developers based the `less` command on the functionality of the `more` command, all of the keystroke commands available in the `more` command also work in the `less` command.

The focus of our content is on the more advanced `less` command. The `more` command is still useful to remember for times when the `less` command isn't available. Remember that most of the keystroke commands provided work for both commands.

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
```

There are many movement commands for the `less` command, each with multiple possible keys or key combinations. While this may seem intimidating, it is not necessary to memorize all of these movement commands. When viewing a file with the `less` command, use the **H** key or **Shift**+**H** to display a help screen:

```
                  SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------
                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
  d  ^D             *  Forward  one half-window (and set half-window to N).
  u  ^U             *  Backward one half-window (and set half-window to N).
  ESC-)  RightArrow *  Left  one half screen width (or N positions).
  ESC-(  LeftArrow  *  Right one half screen width (or N positions).
HELP -- Press RETURN for more, or q when done
```

The first group of movement commands to focus on are the ones that are most commonly used. To make it even more convenient, the keys that are identical in `more` and `less` are summarized below in order to demonstrate how to move in `more` and `less` at the same time:

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

When using `less` as a pager, the easiest way to advance forward a page is to press the **Spacebar**.
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 a match can be found, then the cursor moves in the document to the match. For example, in the following graphic the expression "frog" was searched for in the `words` file:

```
bullfrog
bullfrog's
bullfrogs
bullheaded
bullhorn
bullhorn's
```

Notice that "frog" didn't have to be a word by itself. Also notice that while the `less` command moved to the first match from the current position, all matches were highlighted.
If no matches forward from your current position can be found, then the last line of the screen will report `Pattern not found`:

```
Pattern not found (press RETURN)
```

To search backward from your current position, press the question mark **?** key, then type the text or pattern to match and press the **Enter** key. The cursor moves backward to the first match it can find or reports that the pattern cannot be found.
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.
The search terms actually use patterns called regular expressions. More details regarding regular expressions are provided later in this chapter.
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, respectively (or, when used with a pipe, the output of a previous command). By default, the `head` and `tail` commands display ten lines of the file that is provided as an argument.
For example, the following command displays the first ten lines of the `/etc/sysctl.conf` file:

```
sysadmin@localhost:~/Documents$ cd
sysadmin@localhost:~$ head /etc/sysctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com
```

```
# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3
```

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
# Protects against creating or following links under certain conditions
# Debian kernels have both set to 1 (restricted)
# See https://www.kernel.org/doc/Documentation/sysctl/fs.txt
#fs.protected_hardlinks=0
#fs.protected_symlinks=0
```

The `-n` option can also be used to indicate how many lines to output. Pass a number as an argument to the option:

```
sysadmin@localhost:~$ head -n 3 /etc/sysctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables
```

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.

In previous examples the `head` and `tail` commands were given files as arguments to operate on. However, the pipe character allows you to utilize these commands not only on files, but on the output of other commands. This can be useful when listing a large directory, for example the `/etc` directory:

```
sysadmin@localhost:~$ ls /etc
X11                     gss              mke2fs.conf     rpc
adduser.conf            host.conf        modprobe.d      rsyslog.conf
alternatives            hostname         modules         rsyslog.d
apparmor                hosts            modules-load.d  securetty
apparmor.d              hosts.allow      motd            security
apt                     hosts.deny       mtab            selinux
bash.bashrc             init.d           nanorc          services
bind                    initramfs-tools  netplan         shadow
```

```
bindresvport.blacklist  inputrc          network         shadow-
binfmt.d                insserv.conf.d   networks         shells
ca-certificates          iproute2         newt             skel
ca-certificates.conf     issue            nsswitch.conf   ssh
calendar                issue.net        opt             ssl
console-setup            kernel           os-release      subgid
cron.d                  ld.so.cache      pam.conf         subgid-
cron.daily              ld.so.conf       pam.d           subuid
cron.hourly              ld.so.conf.d     passwd          subuid-
cron.monthly             ldap             passwd-         sudoers
cron.weekly             legal            perl            sudoers.d
crontab                  libaudit.conf     pinforc          sysctl.conf
dbus-1                  locale.alias     ppp             sysctl.d
debconf.conf             locale.gen        profile         systemd
debian_version          localtime        profile.d       terminfo
default                 logcheck         protocols       timezone
deluser.conf             login.defs       python3          tmpfiles.d
depmod.d                logrotate.conf   python3.6       ucf.conf
dhcp                     logrotate.d       rc0.d            udev
dpkg                    lsb-release      rc1.d            ufw
environment              machine-id       rc2.d            update-motd.d
fstab                    magic            rc3.d           updatedb.conf
gai.conf                 magic.mime       rc4.d            vim
groff                   mailcap          rc5.d           vtrgb
group                    mailcap.order    rc6.d            wgetrc
group-                   manpath.config   rcS.d            xdg
gshadow                 mc               resolv.conf
gshadow-                 mime.types       rmt
```

The previous command lists a large number of files. If you execute this in our terminal, the output is cut off and can only be viewed if scrolling up. 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
X11
adduser.conf
alternatives
apparmor
apparmor.d
apt
bash.bashrc
bind
bindresvport.blacklist
binfmt.d
```

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
```

In the next example, note that the `ls` command is executed first and its output is sent to the `nl` command, numbering all of the lines from the output of the `ls` command. Then the `tail` command is executed, displaying the last five lines from the output of the `nl` command:

```
sysadmin@localhost:~$ ls /etc/ssh | nl | tail -5
     6  ssh_host_ed25519_key.pub
     7  ssh_host_rsa_key
     8  ssh_host_rsa_key.pub
     9  ssh_import_id
    10  sshd_config
```

Compare the output above with the next example:

```
sysadmin@localhost:~$ ls /etc/ssh | tail -5 | nl
     1  ssh_host_ed25519_key.pub
     2  ssh_host_rsa_key
     3  ssh_host_rsa_key.pub
     4  ssh_import_id
     5  sshd_config
```

Notice how the line numbers are different. Why is this?
In the second example, the output of the `ls` command is first sent to the `tail` command which only grabs the last five lines of the output. Then the `tail` command sends those five lines to the `nl` command, which numbers them 1-5.
Pipes can be powerful, but it is necessary to consider how commands are piped to ensure that the desired output is displayed.
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.

1. 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.
2. 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.
3. 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
```

This command displays no output because STDOUT was sent to the file `example.txt` instead of the screen. You can see the new file with the output of the `ls` command.

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

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
```

The original contents of the file are gone, replaced with the output of the new `echo` command. 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
```

Instead of being overwritten, the output of the `echo` command is added to the bottom of the file.

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
```

In the example above, STDOUT was redirected to the `output.txt` file. So, the output that is displayed can't be STDOUT because it would have been placed in the `output.txt` file instead of the terminal. Because all command output goes either to STDOUT or STDERR, the output displayed above must be STDERR.
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
```

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
```

Note that when you use &>, the output appears in the file with all of the STDERR messages at the top and all of the STDOUT messages below all STDERR messages:

```
sysadmin@localhost:~$ ls /fake /etc/ppp /junk /etc/sound &> all.txt
sysadmin@localhost:~$ cat all.txt
ls: cannot access '/fake': No such file or directory
ls: cannot access '/junk': No such file or directory
ls: cannot access '/etc/sound': 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
```

The order in which the streams are specified does not matter.

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**.

In the preceding example, the `cat` command isn't provided a filename as an argument. So, it asks for the data to display on the screen from STDIN. The user types `hello`, and then the `cat` command displays `hello` on the screen. While this is mildly entertaining, it isn't particularly useful.
However, if the output of the `cat` command were redirected to a file, then this method could be used either to add text to an existing file or to place text into a new file.
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
```

While the previous example demonstrates another advantage of redirecting STDOUT, it doesn't address why or how STDIN can be directed. To understand this, 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
```

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. Compare this output to the output of the previous `cat` command.

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.

```
sysadmin@localhost:~$ sort -t: -n -k3 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
```

Another commonly used option to the `sort` command is the `-r` option, which is used to perform a reverse sort. The following shows the same command as the previous example, with the addition of the `-r` option, making the higher numbers in the third field appear at the top of the output:

```
sysadmin@localhost:~$ sort -t: -n -r -k3 mypasswd
```

```
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/usr/sbin/nologin
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
```

Lastly, you may want to perform more complex sorts, such as sorting by a primary field and then by a secondary field. For example, consider the following comma-separated value file, a file where the comma character is the field delimiter:

Note
This example is for demonstration purposes. The `os.csv` file does not exist in our VM, therefore output may differ in the VM.

```
sysadmin@localhost:~$ 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:~$ 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
`-k1n`	Numerically sort by field #1
`-k3`	Sort by field #3

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
```

The `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
```

To make it easier to see what exactly is matched, use the `--color` option. This option will highlight the matched items in red:

```
sysadmin@localhost:~$ grep --color bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

Note: On our virtual machines, the grep command is aliased to include the --color option automatically.
In some cases, it may not be important to find the specific lines that match the pattern, but rather how many lines match the pattern. The `-c` option provides a count of how many lines match:

```
sysadmin@localhost:~$ grep -c bash /etc/passwd
2
```

When viewing the output from the `grep` command, it can be hard to determine the original line numbers. This information can be useful when going back into the file (perhaps to edit the file) to quickly find one of the matched lines.
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/bash
```

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 beginn 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 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.
While some of the regular expressions are unnecessarily quoted with single quotes, it is good practice to use single quotes around regular expressions to prevent the shell from trying to interpret special meaning from them.

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:

```
sysadmin@localhost:~/Documents$ cat profile.txt
Hello my name is Joe.
I am 37 years old.
3121991
My favorite food is avocados.
```

```
I have 2 dogs.
123456789101112
```

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:

```
     041  33  21  !                                  141   97  61  a
      042  34  22  “                                   142   98  62  b
      043  35  23  #                                   143   99  63  c
      044  36  24  $                                   144   100 64  d
      045  37  25  %                                   145   101 65  e
      046  38  26  &                                   146   102 66  f
```

The octal value of a is `141` and the numeric value of d is `144`. The numeric value of a is smaller than the numeric value of d; therefore the range `a-d` 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.
```

Consider This
Do not mistake `[^0-9]` to match lines which do not contain numbers. It actually matches lines which contain non-numbers. Look at the original file to see the difference. The third and sixth lines only contain numbers; they do not contain non-numbers, so those lines do not match.

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$ cat red.txt
red
reef
rot
reeed
rd
rod
roof
reed
root
reel
read
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
```

When used with only one other character, * isn't very helpful. Any of the following patterns would match every string or line in the file: `'.*' 'e*' 'b*' 'z*'` because the asterisk * character can match zero occurrences of a pattern.

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

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

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?
```

10.1 Introduction

This is Lab 10: Working with Text. By performing this lab, students will learn how to redirect text streams, use regular expressions and use commands for filtering text files.
In this lab, you will perform the following tasks:

- Learn how to redirect and pipe standard input, output and error channels.
- Use regular expressions to filter the output of commands or file content.
- View large files or command output with programs for paging, and viewing selected portions.
- Previous
- Next

10.2 Command Line Pipes and Redirection

Normally, when you execute a command, the output is displayed in the terminal window. This output (also called a channel) is called standard output, symbolized by the term `stdout`. The file descriptor number for this channel is 1.
Standard error (`stderr`) occurs when an error appears during the execution of a command; it has a file descriptor of 2. Error messages are also sent to the terminal window by default.
In this lab, you will use characters that redirect the output from standard output (`stdout`) and standard error (`stderr`) to a file or to another command instead of the terminal screen.
Standard input, `stdin`, usually is provided by you to a command by typing on the keyboard; it has a file descriptor of 0. However, by redirecting standard input, files can also be used as `stdin`.

10.2.1 Step 1

Use the redirection symbol > along with the `echo` command to redirect the output from the normal output of `stdout` (to the terminal) to a file. The `cat` command can be used to display file contents and will be used in this example to verify redirected output to the file. Type the following:

```
echo "Hello World"
echo "Hello World" > mymessage
cat mymessage
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo "Hello World"
```

```
Hello World
sysadmin@localhost:~$ echo "Hello World" > mymessage
sysadmin@localhost:~$ cat mymessage
Hello World
sysadmin@localhost:~$
```

The first command echoes the message (`stdout`) to the terminal.
The second command redirects the output; instead of sending the output to the terminal, it is sent to a file called `mymessage`.
The last command displays the contents of the `mymessage` file.

10.2.2 Step 2

When you use the > symbol to redirect `stdout`, the contents of the file are first destroyed. Type the following commands to see a demonstration:

```
echo "Greetings" > mymessage
cat mymessage
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ echo "Greetings" > mymessage
sysadmin@localhost:~$ cat mymessage
Greetings
sysadmin@localhost:~$
```

Notice that using one redirection symbol overwrites an existing file. This is called "clobbering" a file.

10.2.3 Step 3

You can avoid clobbering a file by using >> instead of >. By using >> you append to a file.
Execute the following commands to see a demonstration of this:

```
cat mymessage
echo "How are you?" >> mymessage
cat mymessage
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ cat mymessage
Greetings
sysadmin@localhost:~$ echo "How are you?" >> mymessage
sysadmin@localhost:~$ cat mymessage
Greetings
How are you?
sysadmin@localhost:~$
```

Notice that by using >> all existing data is preserved and the new data is appended at the end of the file.

10.2.4 Step 4

The `find` command is a good command to demonstrate how `stderr` works. This very flexible command allows searching with a host of options such as filename, size, date, type and permission. The `find` command will begin the search in the directory specified and recursively search all of the subdirectories.
The `find` command will begin the search in the directory specified and recursively search all of the subdirectories. For example, to search for files beginning in your home directory containing the name `bash`:

```
find ~ -name "*bash*"
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ find ~ -name "*bash*"
/home/sysadmin/.bash_logout
/home/sysadmin/.bashrc
sysadmin@localhost:~$
```

Remember that ~ is used to represent your home directory.
Now, run the following command and observe the output:

```
find /etc -name hosts
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ find /etc -name hosts
/etc/hosts
find: '/etc/ssl/private': Permission denied
sysadmin@localhost:~$
```

Notice the error message indicating you do not have permission to access certain files/directories. This is because as a regular user, you don't have the right to "look inside" some directories. These types of error messages are sent to `stderr`, not `stdout`.
The `find` command is beyond the scope of this course. The purpose of using the command is to demonstrate the difference between `stdout` and `stderr`.

10.2.5 Step 5

To redirect `stderr` (error messages) to a file, issue the following command:

```
find /etc -name hosts 2> err.txt
cat err.txt
```

Your output should be similar to the following:

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

Recall that the file descriptor for `stderr` is the number 2, so it is used along with the > symbol to redirect the `sdterr` output to a file called `err.txt`. Note that `1>` is the same as `>`.
The previous example demonstrates why knowing redirection is important. If you want to "ignore" the errors that the `find` command displays, you can redirect those messages into a file and look at them later, making it easier to focus on the rest of the output of the command.

10.2.6 Step 6

You can also redirect `stdout` and `stderr` into two separate files.

```
find /etc -name hosts > std.out 2> std.err
cat std.err
cat std.out
```

Your output should be similar to the following:

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

Note that a space is permitted but not required after the > redirection symbol.

10.2.7 Step 7

To redirect both standard output (`stdout`) and standard error (`stderr`) to one file, first redirect `stdout` to a file and then redirect `stderr` to that same file by using the notation `2>&1`.

```
find /etc -name hosts > find.out 2>&1
cat find.out
```

Your output should be similar to the following:

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

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

10.2.8 Step 8

Standard input (`stdin`) can also be redirected. Normally `stdin` comes from the keyboard, but sometimes you want it to come from a file instead. For example, the `tr` command translates characters, but it only accepts data from `stdin`, never from a file name given as an argument. This is great when you want to do something like capitalize data that is inputted from the keyboard (Note: Press **Control**+**d**, to signal the `tr` command to stop processing standard input):

```
tr a-z A-Z
this is interesting
how do I stop this?
```

```
^D
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ tr a-z A-Z
this is interesting
THIS IS INTERESTING
how do I stop this?
HOW DO I STOP THIS?
sysadmin@localhost:~$
```
Note: ^D symbolizes **Control**+**d**

10.2.9 Step 9

The `tr` command accepts keyboard input (`stdin`), translates the characters and then redirects the output to `stdout`. To create a file of all lower-case characters, execute the following:
```
tr A-Z a-z > myfile
Wow, I SEE NOW
This WORKS!
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ tr A-Z a-z > myfile
Wow, I SEE NOW
This WORKS!
sysadmin@localhost:~$
```
Press the **Enter** key to make sure your cursor is on the line below "This works!", then use **Control**+**d** to stop input. To verify you created the file, execute the following command:
```
cat myfile
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ cat myfile
wow, i see now
this works!
sysadmin@localhost:~$
```

10.2.10 Step 10

Execute the following commands to use the `tr` command by redirecting `stdin` from a file:
```
cat myfile
tr a-z A-Z < myfile
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ cat myfile
wow, i see now
this works!
sysadmin@localhost:~$ tr a-z A-Z < myfile
WOW, I SEE NOW
THIS WORKS!
sysadmin@localhost:~$
```

10.2.11 Step 11

Another popular form of redirection is to take the output of one command and send it into another command as input. For example, the output of some commands can be massive, resulting in the output scrolling off the screen too quickly to read. Execute the following command to take the output of the `ls` command and send it into the `more` command, which displays one page of data at a time:
```
ls -l /etc | more
```
Your output should be similar to the following:
```
sysadmin@localhost:~$ ls -l /etc | more
total 372
-rw-r--r-- 1 root root    2981 Jan 28  2015 adduser.conf
-rw-r--r-- 1 root root      10 Jan 28  2015 adjtime
drwxr-xr-x 1 root root     900 Jan 29  2015 alternatives
drwxr-xr-x 1 root root     114 Jan 29  2015 apparmor.d
drwxr-xr-x 1 root root     168 Oct  1  2014 apt
-rw-r--r-- 1 root root    2076 Apr  3  2012 bash.bashrc
drwxr-xr-x 1 root root      72 Jan 28  2015 bash_completion.d
drwxr-sr-x 1 root bind     342 Jan 29  2015 bind
-rw-r--r-- 1 root root     356 Apr 19  2012 bindresvport.blacklist
-rw-r--r-- 1 root root     321 Mar 30  2012 blkid.conf
lrwxrwxrwx 1 root root      15 Jun 18  2014 blkid.tab -> /dev/.blkid.tab
```

```
drwxr-xr-x 1 root root      16 Jan 29  2015 ca-certificates
-rw-r--r-- 1 root root    7464 Jan 29  2015 ca-certificates.conf
drwxr-xr-x 1 root root      14 Jan 29  2015 calendar
drwxr-xr-x 1 root root      24 Jan 29  2015 cron.d
drwxr-xr-x 1 root root     134 Jan 29  2015 cron.daily
drwxr-xr-x 1 root root      24 Jan 29  2015 cron.hourly
drwxr-xr-x 1 root root      24 Jan 29  2015 cron.monthly
-rw-r--r-- 1 root root    2969 Mar 15  2012 debconf.conf
--More--
```

You will need to press the **spacebar** to continue or you can also press **CTRL**+**c** to escape this listing.

The `cut` command is useful for extracting fields from files that are either delimited by a character, like the colon `:` in `/etc/passwd`, or that have a fixed width. It will be used in the next few examples as it typically provides a great deal of output that we can use to demonstrate using the `|` character.

10.2.12 Step 12

In the following example, you will use a command called `cut` to extract all of the usernames from a database called `/etc/passwd` (a file that contains user account information). First, try running the `cut` command by itself:

```
cut -d: -f1 /etc/passwd
```

A portion of the command output is shown in the graphic below.

```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
irc
gnats
nobody
libuuid
syslog
bind
sshd
operator
```

10.2.13 Step 13

The output in the previous example was unordered and scrolled off the screen. In the next step you are going to take the output of the `cut` command and send it into the `sort` command to provide some order to the output:

```
cut -d: -f1 /etc/passwd | sort
```

A portion of the command output is shown in the graphic below.

```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd | sort
backup
bin
bind
daemon
games
gnats
irc
libuuid
list
lp
```

```
mail
man
news
nobody
operator
proxy
root
sshd
sync
sys
```

10.2.14 Step 14

Now the output is sorted, but it still scrolls off the screen. Send the output of the `sort` command to the `more` command to solve this problem:

```
cut -d: -f1 /etc/passwd | sort | more
```

```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd | sort | more
backup
bin
bind
daemon
games
gnats
irc
libuuid
list
lp
mail
man
news
nobody
operator
proxy
root
sshd
sync
sys
sysadmin
syslog
uucp
--More--
```

10.3 Viewing Large Text Files

Although large text files can be viewed with the `cat` command, it is inconvenient to scroll backwards towards the top of the file. Additionally, really large files can't be displayed in this manner because the terminal window only stores a specific number of lines of output in memory.

The use of the `more` or `less` commands allows for the user to the view data a "page" or a line at a time. These "pager" commands also permit other forms of navigation and searching that will be demonstrated in this section.

Note

Examples are given using both the `more` and `less` commands. Most of the time, the commands work the same, however, the `less` command is more advanced and has more features. The `more` command is still important to know because some Linux distributions don't have the `less` command, but all Linux distributions have the `more` command.

If you are not interested in viewing the entire file or output of a command, then there are numerous commands that are able to filter the contents of the file or output. In this module, you will learn the use of the `head` and `tail` commands to be able to extract information from the top or bottom of the output of a command or file contents.

10.3.1 Step 1

The `/etc/passwd` is likely too large to be displayed on the screen without scrolling the screen. To see a demonstration of this, use the `cat` command to display the entire contents of the `/etc/passwdfile`:

```
cat /etc/passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ cat /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
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/bash
sysadmin@localhost:~$
```

10.3.2 Step 2

Use the `more` command to display the entire contents of the `/etc/passwd` file:

```
more /etc/passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
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
--More--(92%)
```

Note

The `--More--(92%)` indicates you are "in" the `more` command and 92% through the current data.

10.3.3 Step 3

While you are in the `more` command, you can view the help screen by pressing the **h** key:

```
h
```

Your output should be similar to the following:

```
Most commands optionally preceded by integer argument k.  Defaults in brackets
```

```
Star (*) indicates argument becomes new default.
-------------------------------------------------------------------------------
<space>                 Display next k lines of text [current screen size]
z                       Display next k lines of text [current screen size]*
<return>                Display next k lines of text [1]*
d or ctrl-D             Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt>   Exit from more
s                       Skip forward k lines of text [1]
f                       Skip forward k screenfuls of text [1]
b or ctrl-B             Skip backwards k screenfuls of text [1]
'                       Go to place where previous search started
=                       Display current line number
/<regular expression>   Search for kth occurrence of regular expression [1]
n                       Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd>       Execute <cmd> in a subshell
v                       Start up /usr/bin/vi at current line
ctrl-L                  Redraw screen
:n                      Go to kth next file [1]
:p                      Go to kth previous file [1]
:f                      Display current file name and line number
.                       Repeat previous command
-------------------------------------------------------------------------------
--More--(92%)
```

10.3.4 Step 4

Press the **Spacebar** to view the rest of the document:

`<SPACE>`

In the next example, you will learn how to search a document using either the `more` or `less` commands.

Searching for a pattern within both the `more` and `less` commands is done by typing the slash **/**, followed by the pattern to find. If a match is found, the screen should scroll to the first match. To move forward to the next match, press the **n** key. With the `less` command you can also move backwards to previous matches by pressing the **N** (capital n) key.

10.3.5 Step 5

Use the `less` command to display the entire contents of the `/etc/passwd` file. Then search for the word `bin`, use **n** to move forward, and **N** to move backwards. Finally, quit the less pager by typing the letter **q**:

```
less /etc/passwd
/bin
nnnNNNq
```

Important

Unlike the `more` command which automatically exits when you reach the end of a file, you must press a quit key such as **q** to quit the `less` program.

10.3.6 Step 6

You can use the `head` command to display the top part of a file. By default, the `head` command will display the first ten lines of the file:

```
head /etc/passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
sysadmin@localhost:~$
```

10.3.7 Step 7

Use the `tail` command to display the last ten lines of the `/etc/passwd` file:
`tail /etc/passwd`
Your output should be similar to the following:
```
sysadmin@localhost:~$ tail /etc/passwd
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:/usr/sbin/nol
ogin
syslog:x:103:106::/home/syslog:/usr/sbin/nologin
messagebus:x:104:107::/nonexistent:/usr/sbin/nologin
bind:x:105:110::/var/cache/bind:/usr/sbin/nologin
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
operator:x:1000:37::/root:/bin/sh
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:~$
```

10.3.8 Step 8

Use the `head` command to display the first two lines of the `/etc/passwd` file:
`head -2 /etc/passwd`
Your output should be similar to the following:
```
sysadmin@localhost:~$ head -2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
sysadmin@localhost:~$
```

10.3.9 Step 9

Execute the following command line to pipe the output of the `ls` command to the `tail` command, displaying the last five file names in the `/etc` directory:
`ls /etc | tail -5`
Your output should be similar to the following:
```
sysadmin@localhost:~$ ls /etc | tail -5
updatedb.conf
vim
vtrgb
wgetrc
xdg
sysadmin@localhost:~$
```
As you've seen, both `head` and `tail` commands output ten lines by default. You could also use an option `-#` (or you can use the option `-n #`, where # is a number of lines to output). Both commands can be used to read standard input from a pipe that receives output from a command.
You have also seen where `head` and `tail` commands are different: the `head` command starts counting its lines to output from the top of the data, whereas the `tail` command counts the number of lines to output from the bottom of the data. There are some additional differences between these two commands as demonstrated in the next few tasks.

10.3.10 Step 10

Another way to specify how many lines to output with the `head` command is to use the option `-n -#`, where # is the number of lines counted from the bottom of the output to exclude. Notice the minus symbol `-` in front of the #. For example, if the `/etc/passwd` contains 27 lines, the following command will display lines 1-7, excluding the last twenty lines:
`head -n -20 /etc/passwd`
```
sysadmin@localhost:~$ head -n -20 /etc/passwd
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
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

```
sysadmin@localhost:~$
```

10.4 Searching Text Using Regular Expressions

In this task, you will use the `grep` family of commands with regular expressions in order to search for a specific string of characters in a stream of data (for example, a text file).
The `grep` command uses basic regular expressions, special characters like wildcards that match patterns in data. The `grep` command returns the entire line containing the pattern that matches.
The `-E` option to the `grep` command can be used to perform searches with extended regular expressions, essentially more powerful regular expressions. Another way to use extended regular expressions is to use the `egrep` command.
The `fgrep` command is used to match literal characters, ignoring the special meaning of regular expression characters.

10.4.1 Step 1

The use of `grep` in its simplest form is to search for a given string of characters, such as `sshd` in the `/etc/passwd` file. The `grep` command will print the entire line containing the match:

```
cd /etc
grep sshd passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ cd /etc
sysadmin@localhost:/etc$ grep sshd passwd
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

10.4.2 Step 2

Regular expressions are "greedy" in the sense that they will match every single instance of the specified pattern:

```
grep root passwd
```

Your output should be similar to the following:

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

Note the red highlights indicate what exactly was matched. You can also see that all occurrences of root were matched on each line.

10.4.3 Step 3

To limit the output, you can use regular expressions to specify a more precise pattern. For example, the caret ^ character can be used to match a pattern at the beginning of a line; so, when you execute the following command line, only lines that begin with `root` should be matched and displayed:

```
grep '^root' passwd
```

Your output should be similar to the following:

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

Note that there are two additional instances of the word `root` but only the one appearing at the beginning of the line is matched (displayed in red).

Best Practice
Use single quotes (not double quotes) around regular expressions to prevent the shell program from trying to interpret them.

10.4.4 Step 4

Match the pattern `sync` anywhere on a line:

```
grep 'sync' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep 'sync' passwd
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:/etc$
```

10.4.5 Step 5

Use the $ symbol to match the pattern sync at the end of a line:

```
grep 'sync$' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep 'sync$' passwd
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:/etc$
```

The command in the previous step matched every instance; the second only matches the instance at the end of the line.

10.4.6 Step 6

Use the period character . to match any single character. For example, execute the following command to match any character followed by a 'y':

```
grep '.y' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep '.y' passwd
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologi
n
nobody:x:65534:65534:nobody:/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:/usr/sbin/nol
ogin
syslog:x:103:106::/home/syslog:/usr/sbin/nologin
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:/etc$
```

10.4.7 Step 7

The pipe character, `|`, or "alternation operator", acts as an "or" operator. For example, execute the following to attempt to match either `sshd`, `root` or `operator`:

```
grep 'sshd|root|operator' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep 'sshd|root|operator' passwd
sysadmin@localhost:/etc$
```

Observe that the `grep` command does not recognize the pipe as the alternation operator by default. The `grep` command is actually including the pipe as a plain character in the pattern to be matched. The use of either `grep -E` or `egrep` will allow the use of the extended regular expressions, including alternation.

10.4.8 Step 8

Use the `-E` switch to allow `grep` to operate in extended mode in order to recognize the alternation operator:

```
grep -E 'sshd|root|operator' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ grep -E 'sshd|root|operator' passwd
root:x:0:0:root:/root:/bin/bash
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
operator:x:1000:37::/root:/bin/sh
sysadmin@localhost:/etc$
```

10.4.9 Step 9

Use another extended regular expression, this time with `egrep` with alternation in a group to match a pattern. The strings nob and non will match:

```
egrep 'no(b|n)' passwd
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ egrep 'no(b|n)' passwd
```

```
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
messagebus:x:104:107::/nonexistent:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

Note
The parentheses, ( ), were used to limit the "scope" of the | character. Without them, a pattern such as nob|n would have meant "match either nob or n".

10.4.10 Step 10

The [ ] characters can also be used to match a single character. However, unlike the period character ., the [ ] characters are used to specify exactly what character you want to match. For example, if you want to match a numeric character, you can specify [0-9]. Execute the following command for a demonstration:

```
head passwd | grep '[0-9]'
```

Your output should be similar to the following:

```
sysadmin@localhost:/etc$ head passwd | grep '[0-9]'
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
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

Note
The `head` command was used to limit the output of the `grep` command.

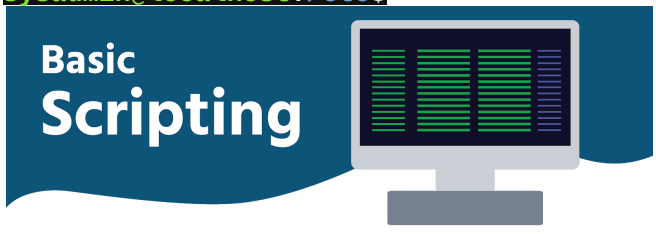
10.4.11 Step 11

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:/usr/sbin/nol
ogin
syslog:x:103:106::/home/syslog:/usr/sbin/nologin
messagebus:x:104:107::/nonexistent:/usr/sbin/nologin
bind:x:105:110::/var/cache/bind:/usr/sbin/nologin
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
operator:x:1000:37::/root:/bin/sh
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:/etc$
```



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.

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, which leads to the shortened form of “shebang” when they’re used at the beginning of a script.

Incidentally, the shebang (or crunchbang) is used for traditional shell scripts and other text-based languages like Perl, Ruby, and Python. Any text file marked as executable will be run under the interpreter specified in the first line as long as the script is run directly. If the script is invoked directly as an argument to an interpreter, such as `sh script` or `bash script`, the given shell will be used no matter what's in the shebang line.
It helps to become comfortable using a text editor before writing shell scripts, since you will need to create files in plain text. Traditional office tools like LibreOffice that output file formats containing formatting and other information are not appropriate for this task.

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                             modified

#!/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 Cur Po
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To 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                             modified

#!/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 Cur Po
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To 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.

You will be back at the shell prompt after saving. Return to the editor. This time press **Ctrl** and **O** together to save your work without exiting the editor. The prompts are largely the same, except that you're back in the editor.
This time use the arrow keys to move your cursor to the line that has `"The time is"`. Press **Control** and **K** twice to cut the last two lines to the copy buffer. Move your cursor to the remaining line and press **Control** and **U** once to paste the copy buffer to the current position. This makes the script echo the current time before greeting you and saved you needing to re-type the lines.
Other helpful commands you might need are:

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

You got your first taste of scripting earlier in this chapter where we introduced a very basic script that ran a single command. The script started with the shebang 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:

1. Variables, which hold temporary information in the script
2. Conditionals, which let you do different things based on tests you write
3. 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.
Next, the script echos a string to the console. The string contains the name of the variable preceded by a dollar sign. When the interpreter sees that dollar sign it recognizes that it will be substituting the contents of the variable, which is called interpolation. The output of the script is then `My favorite animal is a penguin`.
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!

```
#!/bin/bash

ANIMAL=penguin
```

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

`ANIMAL` contains the string `penguin` (as there are no spaces; in this example, the alternative syntax without using quotes is shown). `SOMETHING` is then assigned the contents of `ANIMAL` (because `ANIMAL` has the dollar sign in front of it).
If you wanted, you could assign an interpolated string to a variable. This is quite common in larger scripts, as you can build up a larger command and execute it!
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"
```

This pattern is often used to process text. You might take text from one variable or an input file and pass it through another command like `sed` or `awk` to extract certain parts and keep the result in a variable. The `sed` command is used to edit streams (STDIN) and the `awk` command is used for scripting. These commands are beyond the scope of this course.
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`	`-o` is OR: either can be the same

`test 1 -eq 1 -a 2 -eq` `-a` is AND: both must be the same

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.
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.”

```
#!/bin/bash

i=0
while [ $i -lt 10 ]; do
  echo $i
  i=$(( $i + 1))
done
echo “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.

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.

Keeping virtual machines in mind, it's imperative to understand what makes up the physical hardware that makes up a computer. Without an understanding of what components make up a computer, how they are integrated and communicate between each other, and how a computer should function, you cannot properly install, configure, administer, secure or troubleshoot a system.

There are hundreds of specific types of computers in service, from set-top boxes to dedicated firewalls to laptops, servers and many special-purpose devices, but they all feature at least a few of the same basic components that actually make them a computer. 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:

```
sysadmin@localhost:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
```

```
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
```

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     cached
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):

Note: The example below uses the `lspci` command. This command is not available within the virtual machine environment of this course.

```
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 (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 Interface
(rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
03:00.0 Serial Attached SCSI controller: VMware PVSCSI SCSI Controller (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:

Note

The example uses the `lsusb` command. This command is not available within the virtual machine environment of this course.

```
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
```

Note: Creating and modifying partitions is beyond the scope of this course. For more information, check out NDG Linux I.

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 and the very widely used High-Definition Multimedia Interface (HDMI) which supports resolutions up to the 4K or Ultra HD range.
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.

12.1 Introduction

This is Lab 12: Understanding Computer Hardware. By performing this lab, students will learn about commands to display information about the computer's hardware.
In this lab, you will perform the following tasks:

- Use commands to list hardware.

12.2 Listing Computer Hardware

In this task, you will execute several commands and examine several files to display your hardware configuration.

12.2.1 Step 1

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:    2
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Stepping:              2
CPU MHz:               2394.008
BogoMIPS:              4787.98
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15
sysadmin@localhost:~$
```

12.2.2 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       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           E5620  @ 2.40GHz
stepping        : 2
microcode       : 0x1a
cpu MHz         : 2394.008
cache size      : 12288 KB
 physical id     : 1
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 32
initial apicid  : 32
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
pdpe1gb rdt scp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc ap erfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2
ssse3 cx16 xtpr pdc m pcid dca sse4_1 sse4_2 popcnt aes lahf_lm tpr_shadow
vnmi flexpriority ept vpi d dtherm ida arat
```
Although these flags are beyond the scope of this course, finding them may be useful for planning or troubleshooting production systems.

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:         128920       53910       67178           8        7832
74394
Swap:        131050         177      130873
sysadmin@localhost:~$ free -g
              total        used        free      shared  buff/cache   available
Mem:            125          52          65           0           7
72
Swap:           127           0         127
sysadmin@localhost:~$
```
In the output above, you can see that the system has `128920` megabytes (roughly 128 gigabytes) of physical memory (RAM). Of that, only `31274` 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 5520 I/O Hub to ESI Port (rev 13)
00:01.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root
Por
t 1 (rev 13)
00:03.0 PCI bridge: Intel Corporation 5520/5500/X58 I/O Hub PCI Express Root
Por
t 3 (rev 13)
00:04.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 4
(
rev 13)
00:05.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 5
(
rev 13)
00:06.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 6
(
rev 13)
Output Omitted...
```

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 Host bridge (
rev 01)
        Subsystem: VMware Virtual Machine Chipset
        Kernel driver in use: agpgart-intel
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 Interface (rev 1
0)
        Subsystem: VMware Virtual Machine Communication Interface
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
        Subsystem: VMware SVGA II Adapter
        Kernel driver in use: vmwgfx
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-
MPT Dual Ultra320 SCSI (rev 01)
        Subsystem: VMware LSI Logic Parallel SCSI Controller
        Kernel driver in use: mptspi
```

12.2.6 Step 6

Attempt to list the USB connected devices:

`lsusb`

```
sysadmin@localhost:~$ lsusb
Bus 002 Device 003: ID 0624:0249 Avocent Corp. Virtual Keyboard/Mouse
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 002: ID 0624:0248 Avocent Corp. Virtual Hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 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
Output omitted...
```

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 getting for 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 /
```

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 time=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 ---
1 packets transmitted, 0 received, 100% packet loss, time 10000
```

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:

```
sysadmin@localhost:~$ ps
  PID TTY          TIME CMD
 6054 ?        00:00:00 bash
 6070 ?        00:00:01 xeyes
 6090 ?        00:00:01 firefox
 6146 ?        00:00:00 ps
```

If you run `ps` with the option `--forest`, then, similar to the `pstree` command, it shows lines indicating the parent and child relationship:

```
sysadmin@localhost:~$ ps --forest
  PID TTY          TIME CMD

 6054 ?        00:00:00 bash
 6090 ?        00:00:02   \_ firefox
 6180 ?        00:00:00   \_ dash
 6181 ?        00:00:00        \_ xeyes
 6188 ?        00:00:00        \_ ps
```

To be able to view all processes on the system execute either the `ps aux` command or the `ps -ef` command:

```
sysadmin@localhost:~$ ps aux | head
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  17872  2892 ?        Ss   08:06   0:00 /sbin?? /init
syslog      17  0.0  0.0 175744  2768 ?        Sl   08:06   0:00
/usr/sbin/rsyslogd -c5
root        21  0.0  0.0  19124  2092 ?        Ss   08:06   0:00
/usr/sbin/cron
root        23  0.0  0.0  50048  3460 ?        Ss   08:06   0:00
/usr/sbin/sshd
bind        39  0.0  0.0 385988 19888 ?        Ssl  08:06   0:00
/usr/sbin/named -u bind
```

```
root         48  0.0  0.0  54464  2680 ?        S    08:06   0:00 /bin/login -f
sysadmin     60  0.0  0.0  18088  3260 ?        S    08:06   0:00 -bash
sysadmin    122  0.0  0.0  15288  2164 ?        R+   16:26   0:00 ps aux
sysadmin    123  0.0  0.0  18088   496 ?        D+   16:26   0:00 -bash
```

```
sysadmin@localhost:~$ ps -ef | head
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 08:06 ?        00:00:00 /sbin?? /init
syslog        17     1  0 08:06 ?        00:00:00 /usr/sbin/rsyslogd -c5
root          21     1  0 08:06 ?        00:00:00 /usr/sbin/cron
root          23     1  0 08:06 ?        00:00:00 /usr/sbin/sshd
bind          39     1  0 08:06 ?        00:00:00 /usr/sbin/named -u bind
root          48     1  0 08:06 ?        00:00:00 /bin/login -f
sysadmin      60    48  0 08:06 ?        00:00:00 -bash
sysadmin     124    60  0 16:46 ?        00:00:00 ps -ef
sysadmin     125    60  0 16:46 ?        00:00:00 head
```

The output of all processes running on a system can be overwhelming. In the preceding examples, the output of the `ps` command was filtered by the `head` command, so only the first ten processes were shown. If you don't filter the output of the `ps` command, then you are likely to have to scroll through hundreds of processes to find what might interest you. 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 -e | 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 - 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 si,  0.0
st
KiB Mem : 13201464+total, 76979904 free, 47522152 used,  7512580 buff/cache
KiB Swap: 13419622+total, 13415368+free,    42544 used. 83867456 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18376   3036   2764 S   0.0  0.0   0:00.12 init
    9 syslog    20   0  191328   5648   3100 S   0.0  0.0   0:00.04 rsyslogd
   13 root      20   0   28356   2552   2320 S   0.0  0.0   0:00.00 cron
   15 root      20   0   72296   3228   2484 S   0.0  0.0   0:00.00 sshd
   24 bind      20   0  878888  39324   7148 S   0.0  0.0   0:02.72 named
   43 root      20   0   78636   3612   3060 S   0.0  0.0   0:00.00 login
```

```
  56 sysadmin  20   0   18508   3440   3040 S   0.0  0.0   0:00.00 bash
  72 sysadmin  20   0   36600   3132   2696 R   0.0  0.0   0:00.03 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.
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.
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 si,  0.0
st
KiB Mem : 13201464+total, 76979904 free, 47522152 used,  7512580 buff/cache
KiB Swap: 13419622+total, 13415368+free,    42544 used. 83867456 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18376   3036   2764 S   0.0  0.0   0:00.12 init
    9 syslog    20   0  191328   5648   3100 S   0.0  0.0   0:00.04 rsyslogd
   13 root      20   0   28356   2552   2320 S   0.0  0.0   0:00.00 cron
   15 root      20   0   72296   3228   2484 S   0.0  0.0   0:00.00 sshd
   24 bind      20   0  878888  39324   7148 S   0.0  0.0   0:02.72 named
   43 root      20   0   78636   3612   3060 S   0.0  0.0   0:00.00 login
   56 sysadmin  20   0   18508   3440   3040 S   0.0  0.0   0:00.00 bash
  72 sysadmin  20   0   36600   3132   2696 R   0.0  0.0   0:00.03 top
```

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
```

1. **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.
2. **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.

3. **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.

```
sysadmin@localhost:~$ free
             total       used       free     shared    buffers     cached
Mem:      32953528   26171772    6781756          0       4136   22660364
-/+ buffers/cache:    3507272   29446256
Swap:            0          0          0
```

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:

```
sysadmin@localhost:~$ free -s 10
              total        used        free      shared  buff/cache
available
Mem:      132014640    47304084    77189512        3008     7521044
84085528
Swap:     134196220       42544   134153676

              total        used        free      shared  buff/cache
available
Mem:      132014640    47302928    77190668        3008     7521044
84086684
Swap:     134196220       42544   134153676
```

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
```

1.

Physical Memory Statistics:

```
             total       used       free     shared    buffers     cached
Mem:      32953528   26171772    6781756          0       4136   22660364
-/+ buffers/cache:    3507272   29446256
Swap:            0          0          0
```

2.

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
```

3.

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
```

4.

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 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:

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 execut 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 s or received.
`messages`	Messages from the kernel and other processes that don't belong elsewhere. Sometimes named `syslog` instead of `messages` afte the daemon that writes this file.
`secure`	Messages from processes that required authorization or authentication (such as the login process).
`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/.

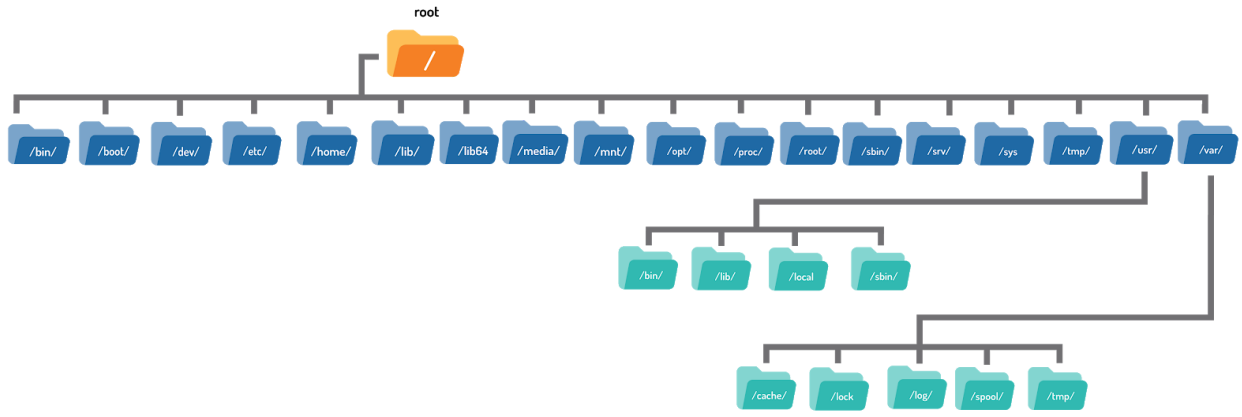
A standard is a set of rules or guidelines that it is recommended to follow. However, these guidelines certainly can be broken, either by entire distributions or by administrators on individual machines.

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 partitior removable devices or network shares
`/bin`	Essential binaries like the `ls`, `cp`, and `rm` commands, and be a pa 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, sucl 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
`/lib`	Essential libraries to support the executable files in the `/bin` and `/sbin` directories
`/lib64`	Essential libraries built for a specific architecture. For example, th `/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 connec to the system
`/srv`	Location where site-specific services may be hosted
`/tmp`	Directory where all users are allowed to create temporary files an 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
`/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.

13.1 Introduction

This is Lab 13: Where Data is Stored. By performing this lab, students will learn about the locations of kernel information, process information, libraries, log files, and software packages.
In this lab, you will perform the following tasks:

- Investigate how the `/proc` filesystem is used by the kernel
- Use the `ps` command to view process information
- Learn how to manage processes by starting, stopping, and resuming them
- Viewing log files
- Manage the ability to load shared libraries

13.2 The kernel and /proc

In this task, you will explore the `/proc` directory and commands that communicate with the Linux kernel. The `/proc` directory appears to be an ordinary directory, like `/usr` or `/etc`, but it is not. Unlike the `/usr` or `/etc` directories, which are usually written to a disk drive, the `/proc` directory is a pseudo filesystem maintained in the memory of the computer.

The /proc Directory

The `/proc` directory contains a subdirectory for each running process on the system. Programs such as `ps` and `top` read the information about running processes from these directories. The `/proc` directory also contains information about the operating system and its hardware in files like `/proc/cpuinfo`, `/proc/meminfo` and `/proc/devices`.
The `/proc/sys` subdirectory contains pseudo files that can be used to alter the settings of the running kernel. Since these files are not "real" files, an editor should not be used to change them; instead you should use either the `echo` or the `sysctl` command to overwrite the contents of these files. For the same reason, do not attempt to view these files in an editor, but use the `cat` or `sysctl` command instead.
For permanent configuration changes, the kernel uses the `/etc/sysctl.conf` file. Typically, this file is used by the kernel to make changes to the /proc files when the system is starting up.

13.2.1 Step 1

In this task, you will examine some of the files contained in the `/proc` directory. Use the password `netlab123` when prompted:

```
su
ls /proc
```

Your output should be similar to the following:

```
sysadmin@localhost:~$ su
Password:
root@localhost:~# ls /proc
1           consoles     irq          mounts          sysvipc
17          cpuinfo      kallsyms     mtrr            thread-self
21          crypto       kcore        net             timer_list
23          devices      key-users    pagetypeinfo    timer_stats
39          diskstats    keys         partitions      tty
60          dma          kmsg         sched_debug     uptime
72          driver       kpagecgroup  schedstat       version
84          execdomains  kpagecount   scsi            version_signature
85          fb           kpageflags   self            vmallocinfo
96          filesystems  loadavg      slabinfo        vmstat
acpi        fs           locks        softirqs        zoneinfo
buddyinfo   interrupts   mdstat       stat
bus         iomem        meminfo      swaps
cgroups     ioports      misc         sys
cmdline     ipmi         modules      sysrq-trigger
root@localhost:~#
```

Recall that the directories that have numbers for names represent running processes on the system. The first process is always `/sbin/init`, so the directory `/proc/1` will contain files with information about the running `init` process.

The `cmdline` file inside the process directory (`/proc/1/cmdline`, for example) will show the command that was executed. The order in which other processes are started varies greatly from system to system. Since the content of this file does not contain a new line character, an `echo` command will be executed to cause the prompt to go to a new line.

13.2.2 Step 2

Use `cat` then `ps` to view information about the `/sbin/init` process (Process IDentifier (PID) of 1):

```
cat /proc/1/cmdline; echo
ps -p 1
```

Your output should look something like this:

```
root@localhost:~# cat /proc/1/cmdline; echo
/bin/bash/init
```

Note

The `echo` command in this example is executed immediately after the `cat` command. Since it has no argument, it functions only to put the next command prompt on a new line. Execute the `cat` command alone to see the difference.

```
root@localhost:~# ps -p 1
 PID TTY          TIME CMD
   1 ?        00:00:00 init
```

The other files in the `/proc` directory contain information about the operating system. The following tasks will be used to view and modify these files.

13.2.3 Step 3

View the `/proc/cmdline` file to see what arguments were passed to the kernel at boot time:

```
cat /proc/cmdline
```

The output of the command should be similar to this:

```
root@localhost:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.4.0-72-generic
root=UUID=8db73d4f-e220-4e04-a8cb-c609
2db5b507 ro cgroup_enable=memory swapaccount=1
```

This output contains all of the information, such as command line parameters, special instructions, etc., that was passed to the kernel when it was first started.

13.3 Managing Processes

In this task you will start and stop processes.

13.3.1 Step 1

From the terminal, type the following command:

```
ping localhost > /dev/null
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null
_
```

The output of the `ping` is being redirected to the `/dev/null` file (which is commonly known as the bit bucket).

Note

Notice that the terminal appears to hang up at this command. This is due to running this command in the “foreground”. A process that is run in the foreground will prevent the user from using the shell until the process is complete. On the other hand, a process that is run in the background will allow the user to use the shell and execute other commands.

The system will continue to `ping` until the process is terminated or suspended by the user.

13.3.2 Step 2

Terminate the foreground process by pressing **Ctrl**-**C**.

```
root@localhost:~# ping localhost > /dev/null


^C
root@localhost:~#
```

13.3.3 Step 3

Next, to start the same process in the background, type:

```
ping localhost > /dev/null &
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null &
[1] 107
```

By adding the ampersand & to the end of the command, the process is started in the background, allowing the user to maintain control of the terminal.

Consider This

An easier way to enter the above command would be to take advantage of the command history. You could have pressed the **Up Arrow Key** ↑ on the keyboard, added a **Space** and & to the end of the command and then pressed the **Enter** key. This is a time-saver when entering similar commands.

Notice that the previous command returns the following information:

```
[1] 107
```

This means that this process has a job number of 1 (as demonstrated by the output of `[1]`) and a Process ID (PID) of `107`. Each terminal/shell will have unique job numbers. The PID is system-wide; each process having a unique ID number.

This information is important when performing certain process manipulations, such as stopping processes or changing their priority value.

Your Process ID will likely be different than the one in the example.

13.3.4 Step 4

To see which commands are running in the current terminal, type the following command:

```
jobs
```

Your output should be similar to the following:

```
root@localhost:~# jobs
[1]+  Running                 ping localhost > /dev/null &
```

13.3.5 Step 5

Next, start another `ping` command in the background by typing the following:

```
ping localhost > /dev/null &
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null &
[2] 108
```

Notice the different job number and process ID for this new command

13.3.6 Step 6

Now, there should be two `ping` commands running in the background. To verify, issue the `jobs` command again:

```
jobs
```

Your output should be similar to the following:

```
root@localhost:~# jobs
[1]-  Running                 ping localhost > /dev/null &
[2]+  Running                 ping localhost > /dev/null &
```

13.3.7 Step 7

Once you have verified that two `ping` commands are running, bring the first command to the foreground by typing the following:

```
fg %1
```

Your output should be similar to the following:

```
root@localhost:~# fg %1
ping localhost > /dev/null
```

13.3.8 Step 8

Notice that, once again, the `ping` command has taken control of the terminal. To suspend (pause) the process and regain control of the terminal, type **Ctrl**-**Z**:

```
root@localhost:~# fg %1
ping localhost > /dev/null
^Z
[1]+  Stopped                 ping localhost > /dev/null
```

13.3.9 Step 9

To have this process continue executing in the background, execute the following command:

```
bg %1
```

Your output should be similar to the following:

```
root@localhost:~# bg %1
[1]+ ping localhost > /dev/null &
```

13.3.10 Step 10

Issue the `jobs` command again to verify two running processes:

```
jobs
```

Your output should be similar to the following:

```
root@localhost:~# jobs
[1]-  Running                 ping localhost > /dev/null &
[2]+  Running                 ping localhost > /dev/null &
```

13.3.11 Step 11

Next, start one more `ping` command by typing the following:

```
ping localhost > /dev/null &
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null &
[3] 110
```

13.3.12 Step 12

Issue the `jobs` command again to verify three running processes :

```
jobs
```

```
sysadmin@localhost:~$ jobs
[1]   Running                 ping localhost > /dev/null &
[2]-  Running                 ping localhost > /dev/null &
[3]+  Running                 ping localhost > /dev/null &
```

13.3.13 Step 13

Using the job number, stop the last `ping` command with the `kill` command and verify it was stopped executing the `jobs` command:

```
kill %3
jobs
```

Your output should be similar to the following:

```
root@localhost:~# kill %3
root@localhost:~# jobs
[1]   Running                 ping localhost > /dev/null &
```

```
[2]-   Running                 ping localhost > /dev/null &
[3]+  Terminated              ping localhost > /dev/null
```

13.3.14 Step 14

Finally, you can stop all of the `ping` commands with the `killall` command. After executing the `killall` command, wait a few moments, and then run the `jobs` command to verify that all processes have stopped:

```
killall ping
      jobs
```

Your output should be similar to the following:

```
root@localhost:~# killall ping
[1]-  Terminated              ping localhost > /dev/null
[2]+  Terminated              ping localhost > /dev/null
root@localhost:~# jobs
```

13.4 Use Top to View Processes

In this task, you will use the `top` command to work with processes. By default, the `top` program sorts the processes in descending order of percentage of CPU usage, so the busiest programs will be at the top of its listing.

13.4.1 Step 1

From the terminal window, type the following commands:

```
ping localhost > /dev/null &
ping localhost > /dev/null &
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null &
[1] 122
root@localhost:~# ping localhost > /dev/null &
[2] 123
```

Make note of the PIDs output by the above commands! They will be different than the examples that we are providing. You will use the PIDs in subsequent steps.

13.4.2 Step 2

Next, start the `top` command by typing the following in the terminal:

```
top
```

```
root@localhost:~# top
```

Your output should be similar to the following:

```
top - 19:32:10 up 15 days, 11:09,  1 user,  load average: 7.52, 4.62, 3.01
Tasks:  12 total,   1 running,  11 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.6 us, 10.0 sy,  0.0 ni, 81.6 id,  0.0 wa,  0.0 hi,  0.8 si,  0.0
st
KiB Mem : 13201464+total, 50793780 free, 55954760 used, 25266100 buff/cache
KiB Swap: 13419622+total, 13401251+free,   183704 used. 75423536 avail Mem

  PID USER      PR  NIVIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18376   3092   2816 S   0.0  0.0   0:00.13 init
    9 syslog    20   0  191328   3748   3244 S   0.0  0.0   0:00.03 rsyslogd
   13 root      20   0   28356   2648   2420 S   0.0  0.0   0:00.00 cron
   15 root      20   0   72296   3096   2352 S   0.0  0.0   0:00.00 sshd
   31 bind      20   0  878108  38180   6888 S   0.0  0.0   0:00.56 named
   51 root      20   0   78636   3676   3120 S   0.0  0.0   0:00.00 login
   64 sysadmin  20   0   18508   3416   2960 S   0.0  0.0   0:00.00 bash
   90 root      20   0   60084   3288   2832 S   0.0  0.0   0:00.01 su
   91 root      20   0   18608   3528   3040 S   0.0  0.0   0:00.02 bash
  122 root      20   0   15576   2148   1996 S   0.0  0.0   0:00.05 ping
  123 root      20   0   15576   2056   1904 S   0.0  0.0   0:00.04 ping
  124 root      20   0   36604   3220   2780 R   0.0  0.0   0:00.08 top
```

Note

The output of the `top` command changes every 2 seconds.

13.4.3 Step 3

The `top` command is an interactive program, which means that you can issue commands within the program. You will use the `top` command to kill the `ping` processes. First type the letter **k**. Notice a prompt has appeared:

```
top - 22:18:59 up 15 days, 13:56,  1 user,  load average: 1.90, 1.52, 1.16
Tasks:  12 total,   1 running,  11 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.5 us,  3.0 sy,  0.0 ni, 90.2 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0
st
KiB Mem : 13201464+total, 75271912 free, 49921140 used,  6821588 buff/cache
KiB Swap: 13419622+total, 13401260+free,   183608 used. 81460888 avail Mem
PID to signal/kill: [default pid = 1]
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18376   3092   2816 S   0.0  0.0   0:00.13 init
    9 syslog    20   0  191328   3748   3244 S   0.0  0.0   0:00.04 rsyslogd
   13 root      20   0   28356   2648   2420 S   0.0  0.0   0:00.00 cron
   15 root      20   0   72296   3096   2352 S   0.0  0.0   0:00.00 sshd
   31 bind      20   0 1271584  38440   6888 S   0.0  0.0   0:02.58 named
   51 root      20   0   78636   3676   3120 S   0.0  0.0   0:00.00 login
   64 sysadmin  20   0   18508   3416   2960 S   0.0  0.0   0:00.00 bash
   90 root      20   0   60084   3288   2832 S   0.0  0.0   0:00.01 su
   91 root      20   0   18608   3528   3040 S   0.0  0.0   0:00.02 bash
  122 root      20   0   15576   2148   1996 S   0.0  0.0   0:01.90 ping
  123 root      20   0   15576   2056   1904 S   0.0  0.0   0:02.02 ping
  124 root      20   0   36604   3236   2780 R   0.0  0.0   0:04.3
```

13.4.4 Step 4

At the `PID to kill:` prompt, type the PID of the first running `ping` process, then press **Enter**. Notice that the prompt changes as below:

```
top - 02:03:27 up  9:24,  1 user,  load average: 0.53, 0.27, 0.23
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  1.8%sy,  0.0%ni, 95.9%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:  65969788k total,  7445364k used, 58524424k free,   288612k buffers
Swap:  2097148k total,        0k used,  2097148k free,  1096724k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  114 sysadmin  20   0 17212 2192 1960 R    1  0.0   0:00.10 top
    1 root      20   0 17868 2872 2624 S    0  0.0   0:00.14 init
   19 syslog    20   0  171m 2796 2420 S    0  0.0   0:00.07 rsyslogd
   23 root      20   0 19120 2020 1824 S    0  0.0   0:00.00 cron
   25 root      20   0 50048 3416 2812 S    0  0.0   0:00.00 sshd
   41 bind      20   0  376m  19m 5988 S    0  0.0   0:00.07 named
   50 root      20   0 54460 2584 2176 S    0  0.0   0:00.00 login
   62 sysadmin  20   0 18084 3200 2700 S    0  0.0   0:00.01 bash
  112 sysadmin  20   0  6504 1844 1712 S    0  0.0   0:00.05 ping
  113 sysadmin  20   0  6504 1812 1676 S    0  0.0   0:00.05 ping
```

13.4.5 Step 5

At the `Kill PID with signal [15]:` prompt, enter the signal to send to this process. In this case, just press the **Enter** key to use the default signal. Notice that the first `ping` command is removed and only one `ping` command remains in the listing (you may need to wait a few seconds as the `top` command refreshes):

```
top - 02:03:27 up  9:24,  1 user,  load average: 0.53, 0.27, 0.23
Tasks:  10 total,   1 running,   9 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  1.8%sy,  0.0%ni, 95.9%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:  65969788k total,  7445364k used, 58524424k free,   288612k buffers
Swap:  2097148k total,        0k used,  2097148k free,  1096724k cached
Kill PID 112 with signal [15]:
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  114 sysadmin  20   0 17212 2192 1960 R    1  0.0   0:00.10 top
    1 root      20   0 17868 2872 2624 S    0  0.0   0:00.14 init
   19 syslog    20   0  171m 2796 2420 S    0  0.0   0:00.07 rsyslogd
   23 root      20   0 19120 2020 1824 S    0  0.0   0:00.00 cron
   25 root      20   0 50048 3416 2812 S    0  0.0   0:00.00 sshd
   41 bind      20   0  376m  19m 5988 S    0  0.0   0:00.07 named
   50 root      20   0 54460 2584 2176 S    0  0.0   0:00.00 login
   62 sysadmin  20   0 18084 3200 2700 S    0  0.0   0:00.01 bash
  112 sysadmin  20   0  6504 1844 1712 S    0  0.0   0:00.05 ping
  113 sysadmin  20   0  6504 1812 1676 S    0  0.0   0:00.05 ping
```

Consider This

There are several different numeric values that can be sent to a process. These are predefined values, each with a different meaning. If you want to learn more about these values, type `man kill` in a terminal window.

The prompt indicates that the default signal is the terminate signal indicated by `SIGTERM` or the number `15`.

13.4.7 Step 7

Type **q** to exit the `top` command. The following screen reflects that both `ping` commands were terminated:

```
[1]-  Terminated              ping localhost > /dev/null
[2]+  Killed                  ping localhost > /dev/null
```

13.5 Use pkill and kill To Terminate Processes

In this task, we will continue to work with processes. You will use `pkill` and `kill` to terminate processes.

13.5.1 Step 1

To begin, type the following commands in the terminal:

```
sleep 888888 &
sleep 888888 &
```

Your output should be similar to the following:

```
root@localhost:~# sleep 888888 &
[1] 134
root@localhost:~# sleep 888888 &
[2] 135
```

The `sleep` command is typically used to pause a program (shell script) for a specific period of time. In this case it is used just to provide a command that will take a long time to run.

Make sure to note the PIDs of the sleep processes on your system for the next steps! Your PIDs will be different than those demonstrated in the lab.

13.5.2 Step 2

Next, determine which jobs are currently running by typing:

```
jobs
```

Your output should be similar to the following:

```
root@localhost:~# jobs
[1]-  Running                 sleep 888888 &
[2]+  Running                 sleep 888888 &
```

13.5.3 Step 3

Now, use the `kill` command to stop the first instance of the `sleep` command by typing the following (substitute PID with the process id of your first `sleep` command). Also, execute `jobs` to verify the process has been stopped:

```
ps
kill PID
jobs
```

Your output should be similar to the following:

```
root@localhost:~# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 init
   51 ?        00:00:00 login
   90 ?        00:00:00 su
   91 ?        00:00:00 bash
  134 ?        00:00:00 sleep
  135 ?        00:00:00 sleep
  136 ?        00:00:00 ps
root@localhost:~# kill 134
root@localhost:~# jobs
[1]-  Terminated              sleep 888888
[2]+  Running                 sleep 888888 &
```

Helpful Hint

If you can't remember the PID of the first process, just type the `ps` command, as shown above.

13.5.4 Step 4

Next, use the `pkill` command to terminate the remaining `sleep` command, using the name of the program rather than the PID:

```
pkill -15 sleep
```

Your output should be similar to the following:

```
root@localhost:~# pkill -15 sleep
[1]-  Terminated              sleep 888888
[2]+  Terminated              sleep 888888
```

13.6 Using ps to Select and Sort Processes

The `ps` command can be used to view processes. By default, the `ps` command will only display the processes running in the current shell.

13.6.1 Step 1

Start up a background process using `ping` and view current processes using the `ps` command:

```
ping localhost > /dev/null &
ps
```

Your output should be similar to the following:

```
root@localhost:~# ping localhost > /dev/null &
[1] 138
root@localhost:~# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 init
   51 ?        00:00:00 login
   90 ?        00:00:00 su
   91 ?        00:00:00 bash
  138 ?        00:00:00 ping
  139 ?        00:00:00 ps
```

Make note of the PID of the `ping`, you will use the PID in a subsequent step.

13.6.2 Step 2

Execute the `ps` command using the option `-e`, so all processes are displayed.

```
ps -e
```

Your output should be similar to the following:

```
root@localhost:~# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    9 ?        00:00:00 rsyslogd
```

```
  13 ?        00:00:00 cron
  15 ?        00:00:00 sshd
  31 ?        00:00:02 named
  51 ?        00:00:00 login
  64 ?        00:00:00 bash
  90 ?        00:00:00 su
  91 ?        00:00:00 bash
 138 ?        00:00:00 ping
 140 ?        00:00:00 ps
```

Due to this environment being a virtualized operating system, there are far fewer processes than what would normally be shown with Linux running directly on hardware.

13.6.3 Step 3

Use the `ps` command with the `-o` option to specify which columns to output.

```
ps -o pid,tty,time,%cpu,cmd
```

Your output should be similar to the following:

```
root@localhost:~# ps -o pid,tty,time,%cpu,cmd
  PID TT           TIME %CPU CMD
    1 ?        00:00:00  0.0 /bin/bash /init
   51 ?        00:00:00  0.0 /bin/login -f
   90 ?        00:00:00  0.0 su
   91 ?        00:00:00  0.0 bash
  138 ?        00:00:00  0.0 ping localhost
  141 ?        00:00:00  0.0 ps -o pid,tty,time,%cpu,cmd
```

13.6.4 Step 4

Use the `--sort` option to specify which column(s) to sort by. By default, a column specified for sorting will be in ascending order, this can be forced with placing a plus + symbol in front of the column name. To specify a descending sort, use the minus - symbol in front of the column name.

Sort the output of `ps` by `%mem`:

```
ps -o pid,tty,time,%mem,cmd --sort %mem
```

Your output should be similar to the following:

```
root@localhost:~# ps -o pid,tty,time,%cpu,cmd --sort %mem
  PID TT           TIME %CPU CMD
  142 ?        00:00:00  0.0 ps -o pid,tty,time,%cpu,cmd --sort %mem
  138 ?        00:00:00  0.0 ping localhost
    1 ?        00:00:00  0.0 /bin/bash /init
   90 ?        00:00:00  0.0 su
   91 ?        00:00:00  0.0 bash
   51 ?        00:00:00  0.0 /bin/login -f
```

13.6.5 Step 5

While the `ps` command can show the percentage of memory that a process is using, the `free` command will show overall system memory usage:

```
free
```

Your output should be similar to the following:

```
root@localhost:~# free
               total        used        free      shared  buff/cache
available
Mem:      132014640    50276284    74632368       10864     7105988
81105444
Swap:     134196220      183608   134012612
```

13.6.6 Step 6

Stop the `ping` command with the following `kill` command and verify with the `jobs` command:

```
kill PID
jobs
```

Your output should be similar to the following:

```
root@localhost:~# kill 138
root@localhost:~# jobs
[1]+  Terminated              ping localhost > /dev/null
```

13.7 Viewing System Logs

System logs are critical for many tasks, including fixing operating system problems and ensuring that your system is secure. Knowing where the system log files are stored and how to maintain them is important for a system administrator.

There are two daemons that handle log messages: the `syslogd` daemon and the `klogd` daemon. Typically you don't need to worry about `klogd`; it only handles kernel log messages and it sends its log information to the `syslogd` daemon.

Messages that were generated by the kernel at boot time are stored in the `/var/log/dmesg` file. The `dmesg` command allows viewing of current kernel messages, as well as providing control of whether those messages will appear in a terminal console window.

The main log file that is written to by `syslogd` is `/var/log/messages`.

In addition to the logging done by `syslogd`, many other processes perform their own logging. Some examples of processes that do their own logging include the Apache web server (log file resides in the `/var/log/httpd` directory), the Common Unix Printing System (`/var/log/cups`) and the `auditd` daemon (`/var/log/audit`).

Note

On CentOS systems, the `syslogd` is called `rsyslogd`.

13.7.1 Step 1

System logs are stored in the `/var/log` directory. List the files in this directory:

```
ls /var/log
```

```
root@localhost:~# ls /var/log
alternatives.log  auth.log       btmp      dmesg     faillog  lastlog
tallylog
apt               bootstrap.log  cron.log  dpkg.log  journal  syslog   wtmp
root@localhost:~#
```

13.7.2 Step 2

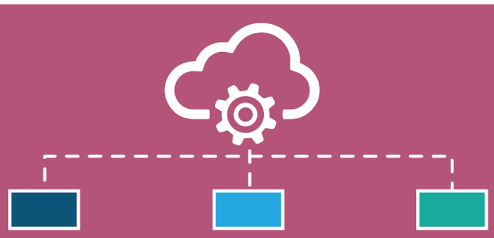
Each log file represents a service or feature. For example, the `auth.log` file displays information regarding authorization or authentication, such as user login attempts. New data is stored at the bottom of the file. Execute the following commands to see an example:

```
ssh localhost
{At the first prompt, type yes}
{At the second prompt, type abc}
{At the third prompt, type abc}
{At the fourth prompt, type abc}
tail -5 /var/log/auth.log
```

```
root@localhost:~# ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 5f:e2:43:0f:f9:26:e5:d5:77:ba:9e:95:72:9e:ee:64.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied (publickey,password).
root@localhost:~# tail -5 /var/log/auth.log
Apr  8 20:25:13 localhost sshd[117]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=localhost  user=root
Apr  8 20:25:16 localhost sshd[117]: Failed password for root from ::1 port
58940 ssh2
Apr  8 20:25:28  sshd[117]: last message repeated 2 times
Apr  8 20:25:28 localhost sshd[117]: Connection closed by ::1 [preauth]
Apr  8 20:25:28 localhost sshd[117]: PAM 2 more authentication failures;
logname= uid=0 euid=0 tty=ssh ruser= rhost=localhost  user=root
root@localhost:~#
```

The `ssh` command was used to generated data in the `/var/log/auth.log` file. Note the failed login attempts are logged in the `/var/log/auth.log` file.

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.

14.2 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 ab to communicate with each other. This communication can be via a wir 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 wo Many people use the Internet to surf web pages and exchange emails but the Internet has many additional capabilities besides these activiti
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 serv For example, a web server stores, processes and delivers web pages. email server receives incoming mail and delivers outgoing mail.

Service	A feature provided by a host is a service. An example of a service wou 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 computer surfing the Internet, you are considered to be on a client hos
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 nam are more natural for humans to remember than numbers, making it easier for us to address network packets to another host. Hostname are translated into IP addresses before the network packet is sent o 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 intern 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 Configuratio Protocol) server. In the world of computers, a protocol is a well-defir 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 addresse 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 yo 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 t 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 desigr to support different communication speeds, the lowest being 10 Mb (10 Megabits per second) and the highest being 100 Gbps (100 gigabits per second). The most common speeds are 100 Mbps and Gbps.
TCP/IP	The Transmission Control Protocol/Internet Protocol (TCP/IP) is a far name for a collection of protocols (remember, protocol = set of rules that are used to define how network communication should take pla between hosts. While it isn't the only collection of protocols used to define network communication, it is the most often utilized one. As a 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`
Essentially, this provides for a much larger address pool, so large that running out of addresses any time in the near future is very unlikely.
It is important to note that the difference between IPv4 and IPv6 isn't just a larger address pool. IPv6 has many other advanced features that address some of the limitations of IPv4, including better speed, more advanced package management and more efficient data transportation.
Considering all the advantages, you would think that by now all hosts would be using IPv6. However, the majority of network-attached devices in the world still use IPv4 (something like 98-99% of all devices).
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.

Nonetheless, most experts agree that IPv6 will eventually replace IPv4, so understanding the basics of both is recommended for those who work in the IT industry.

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.

Generally speaking, desktop machines use wired networks, while laptops use wireless networks. Normally a wired machine uses a static IP address, but these can also often be assigned via a DHCP server. In almost all cases, wireless machines use DHCP since they are almost always mobile and attached to different networks.

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

The primary configuration file for an IPv4 network interface is the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. The following demonstrates 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/ActiveConnection/1
                                                           [
```

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`

`/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.

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
```

3. 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.

Consider This
Two other keywords may appear in the system's `/etc/resolv.conf` file. Although these are beyond the scope of this course, they are routinely included in default `/etc/resolv.conf` files and so we include explanations of these terms below:

`domain`	Followed by a qualified domain, such as `snowblower.example.com`, allows the query for the host `polaris` to be tried both just as the host `polaris`, or failing tha appending the rest of the domain name to it and hopefully havin resolved by the server as that name (e.g. `polaris.snowblower.example.com.`).
`search`	Followed by a set of separate domains which can be queried on after the other hopefully to resolve the name.

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. Not all network settings are covered in this course, but it is important to note from the output below that the IP address of the primary network device `eth0` is `192.168.1.2` and that the device is currently active `UP`:

```
root@localhost:~# ifconfig
eth0      Link encap:Ethernet  HWaddr b6:84:ab:e9:8f:0a
          inet addr:192.168.1.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::b484:abff:fee9:8f0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:95 errors:0 dropped:4 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25306 (25.3 KB)  TX bytes:690 (690.0 B)
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:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:460 (460.0 B)  TX bytes:460 (460.0 B)
```

The `lo` device is referred to as the loopback device. It is a special network device used by the system when sending network-based data to itself.

The `ifconfig` command can also be used to modify network settings temporarily. Typically these changes should be permanent, so using the `ifconfig` command to make such changes is relatively rare.

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:255.255.255.0
          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 group
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    Use 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 package sent to a machine in the `192.168.1` 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, look at the following and focus on where the output used to display default:

```
root@localhost:~# route -n
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use 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 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.
```

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.
By default, the `ping` command continues sending packages endlessly. To limit how many pings to send, use the `-c` option followed by a number indicating how many iterations you desire. The following examples show `ping` being limited to 4 iterations.
If the `ping` command is successful, it looks like the following example:

```
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 TX-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  irtt 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         State
tcp        0      0 192.168.1.2:53          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

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. Sometimes showing the names can be more useful. This can be achieved by dropping the `-n` option:

```
root@localhost:~# netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 cserver.example.:domain *:*                     LISTEN
tcp        0      0 localhost:domain        *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:953           *:*                     LISTEN
tcp6       0      0 [::]:domain             [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:953           [::]:*                  LISTEN
```

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. Replacement 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), ports 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.
While the `ss` command offers many different options for gathering and displaying information, the examples above are the most common ones, and anything else would be outside of the scope of the exam's objectives at this level.

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:c9.
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 attack)!
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 checking.
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/RZ0V4DTxgq9wzd+ohy006SWDSGPA
+nafzlHDPOW7vdI4mZ5ew18KL4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSrg0cda3Pbv7kOdJ
/MTyBlWXFCRH+Cv3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7BA
t5FaiKoAfncM1Q8x3+2V0Ww71/eIFmb1zuUFljHYTprrX88XypNDvjYNby6vw/Pb0rwprz/Tn
mZAW3UX+PnTPI89ZPmNBLuxyrD2cE86Z/il8b+gw3r3+1nJotmIkjn2so1d01QraTlMqVSsbx
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:c9.
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
```

14.1 Introduction

This is Lab 14: Network Configuration. By performing this lab, students will learn about the configuration of their computers on the network.
In this lab, you will perform the following tasks:

1. Examine network configuration information

14.2 Exploring the Network

In this task, you will execute several commands and examine several files to display your network configuration.

14.2.1 Step 1

In order to determine your Internet Protocol (IP) address, execute the `ifconfig` command:

```
sysadmin@localhost:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 192.168.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:c0ff:fea8:102  prefixlen 64  scopeid 0x20<link>
        ether 02:42:c0:a8:01:02  txqueuelen 0  (Ethernet)
        RX packets 11  bytes 878 (878.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 1  bytes 49 (49.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 49 (49.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The output shows two main blocks of information. The first block, indented by `eth0`, reflects information about your first Ethernet network card. The second block, indented by `lo`, reflects information about the loopback or internal network interface.

The second line in each block contains the pertinent information for version 4 of the Internet Protocol (called IPv4) while the third line has the information for version 6 of the Internet Protocol (IPv6). IPv4 is an older method of identifying machines with a series of numbers. It is still widely used today despite the fact that the improved IPv6 method has been available for many years.

The IPv4 addresses are displayed as four decimal numbers ranging from `0` to `255` separated by periods.

The IPv6 addresses are 128-bit numbers which are displayed as hexadecimal digits ranging from `0` to `f`. The hexadecimal digits are generally organized into groups of four digits separated by colons. If a number of consecutive hexadecimal digits have the value of zero, then they are replaced with two colons.

14.2.2 Step 2

Having an IP address will allow your system to communicate with other systems on the same network. With routing devices, you are able to communicate with systems on other networks. To view the table of routing information, use the `route` command:

```
route
```

```
sysadmin@localhost:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

When connecting to other computers, either an IP address or a hostname may be used. Hostnames can be used if they are entered into the `/etc/hosts` file along with their associated IP address or if a Domain Name Server (DNS) provides IP address to host name translation.

A couple of names that are commonly in the `/etc/hosts` file are `localhost`, and `localhost.localdomain`, both of which are used to refer to the current machine.

14.2.3 Step 3

Verify that the IP address `127.0.0.1` has an entry in the `/etc/hosts` file:

```
grep 127.0.0.1 /etc/hosts
```

The output should appear as follows, defining the `localhost` names:

```
sysadmin@localhost:~$ grep 127.0.0.1 /etc/hosts
127.0.0.1       localhost
```

The `ping` command may be used to tell if a system is presently connected to a network. Sometimes, a system may be configured to not respond to `ping` requests. Therefore, the lack of a response to a `ping` command does not mean a system is not connected to a network. A quick response to a `ping` command does indicate, however, that a system is connected to a network.

14.2.4 Step 4

Test to see if the `localhost` machine will respond to four `ping` requests:

```
ping -c4 localhost
```

```
sysadmin@localhost:~$ ping -c4 localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.035 ms

--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.029/0.032/0.035/0.007 ms
```

Unlike the `ping` command that is available in the Microsoft Windows operating system, the Linux `ping` command does not stop making requests by default. If you forget to specify the `-c` option, then you will have to manually stop the command by holding the **Control** key and pressing **C** (**CTRL**+**C**).
Hostnames can also be used if they are registered with a Domain Name System (DNS) server. If your system is connected to a network with DNS servers, then the `nameserver` entry in the `/etc/resolv.conf` file configures your system to use these servers to resolve hostnames into IP addresses.

14.2.5 Step 5

View the `/etc/resolv.conf` file to see if any `nameserver` entries exists:
`cat /etc/resolv.conf`
The output should show one `nameserver` entry:

```
sysadmin@localhost:~$ cat /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
```

14.2.6 Step 6

Use the `dig` command to resolve the `localhost.localdomain` name to an IP address:
`dig localhost.localdomain`

```
sysadmin@localhost:~$ dig localhost.localadmin

; <<>> DiG 9.11.3-1ubuntu1.1-Ubuntu <<>> localhost.localadmin
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 33426
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: e6699a7142d13c74ff6931735c006f9689635efdef906d4c (good)
;; QUESTION SECTION:
;localhost.localadmin.          IN      A

;; Query time: 1556 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Thu Nov 29 23:00:38 UTC 2018
;; MSG SIZE  rcvd: 77
```

Notice the output shows that the first `nameserver` that was listed in the `/etc/resolv.conf` file is the one that responded with the answer in the output (`;; SERVER: 127.0.0.11#53(127.0.0.11)`).

14.2.7 Step 7

You can use the `dig` command to resolve other fully-qualified domain names. Use the `dig` command to resolve the `cserver.example.com` hostname to an IP address:
`dig cserver.example.com`

```
sysadmin@localhost:~$ dig cserver.example.com

; <<>> DiG 9.11.3-1ubuntu1.1-Ubuntu <<>> cserver.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25402
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c33d9913b915efc1e055c1575c006fe19ca3975ce1951a1c (good)
;; QUESTION SECTION:
;cserver.example.com.           IN      A

;; ANSWER SECTION:
cserver.example.com.    86400   IN      A       192.168.1.2

;; AUTHORITY SECTION:
```

```
example.com.            86400   IN      NS      example.com.

;; ADDITIONAL SECTION:
example.com.            86400   IN      A       192.168.1.2

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Thu Nov 29 23:01:53 UTC 2018
;; MSG SIZE  rcvd: 122
```

A fully-qualified domain name (FQDN) includes not just the hostname, but also the domain that the hostname is "in". For the FQDN `cserver.example.com`, `cserver` is the hostname and `example.com` is the domain.

14.2.8 Step 8

Use the `dig` command to resolve the IP address `192.168.1.2` to a hostname:

```
dig -x 192.168.1.2
```

```
sysadmin@localhost:~$ dig -x 192.168.1.2

; <<>> DiG 9.11.3-1ubuntu1.1-Ubuntu <<>> -x 192.168.1.2
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59190
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;2.1.168.192.in-addr.arpa.      IN      PTR

;; ANSWER SECTION:
2.1.168.192.in-addr.arpa. 600   IN      PTR
84a6444e-f044-4754-98e6-64cc9f7d
adea.1.84a6444e-f044-4754-98e6-64cc9f7dadea.LAN.

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Thu Nov 29 23:03:58 UTC 2018
;; MSG SIZE  rcvd: 159
```

The hostname can be found in the `ANSWER SECTION` of the output of the `dig` command. The hostname in the example above looks like the following:

```
;; ANSWER SECTION:
2.1.168.192.in-addr.arpa. 600   IN      PTR
84a6444e-f044-4754-98e6-64cc9f7dadea.1.84a6444e-f044-4754-98e6-64cc9f7dadea.LA
N.
```

Although it may seem like a long hostname, this is detectable as a hostname in our virtual environment. An example of a more simple hostname may look like the following:

```
;; ANSWER SECTION:
192.168.1.2. 8640   IN      PTR     example.com
```

14.2.9 Step 9

The `netstat` command performs a large variety of tasks related to networking. To get an idea of some of its capabilities, execute the command with the `--help` option:

```
netstat --help
```

```
sysadmin@localhost:~$ netstat --help
usage: netstat [-vWeenNcCF] [<Af>] -r         netstat {-V|--version|-h|--help}
       netstat [-vWnNcaeol] [<Socket> ...]
       netstat { [-vWeenNac] -i | [-cnNe] -M | -s [-6tuw] }

        -r, --route              display routing table
        -i, --interfaces         display interface table
        -g, --groups             display multicast group memberships
        -s, --statistics         display networking statistics (like SNMP)
        -M, --masquerade         display masqueraded connections

        -v, --verbose            be verbose
        -W, --wide               don't truncate IP addresses
        -n, --numeric            don't resolve names
```

```
        --numeric-hosts          don't resolve host names
        --numeric-ports          don't resolve port names
        --numeric-users          don't resolve user names
        -N, --symbolic           resolve hardware names
        -e, --extend             display other/more information
        -p, --programs           display PID/Program name for sockets
        -o, --timers             display timers
        -c, --continuous         continuous listing

        -l, --listening          display listening server sockets
        -a, --all                display all sockets (default: connected)
        -F, --fib                display Forwarding Information Base (default)
        -C, --cache              display routing cache instead of FIB
        -Z, --context            display SELinux security context for sockets

  <Socket>={-t|--tcp} {-u|--udp} {-U|--udplite} {-S|--sctp} {-w|--raw}
           {-x|--unix} --ax25 --ipx --netrom
  <AF>=Use '-6|-4' or '-A <af>' or '--<af>'; default: inet
  List of  possible address families (which support routing):
    inet (DARPA Internet) inet6 (IPv6) ax25 (AMPR AX.25)
    netrom (AMPR NET/ROM) ipx (Novell IPX) ddp (Appletalk DDP)
    x25 (CCITT X.25)
```

One of the common uses of `netstat` is to determine which services are listening to or waiting for an incoming connection. For example, a service that is used to allow users to perform remote or network logins is called Secure SHell or SSH. SSH normally will listen to TCP port 22. Well-known ports are the port numbers in the range of 0-1023, typically used by system processes to provide network services. A list of service names and associated port numbers can be found in the `/etc/services` file.

14.2.10 Step 10

Use the `netstat` command to see if the TCP port for `ssh`, 22, has a process listening:

```
netstat -tl
netstat -tln
```

```
sysadmin@localhost:~$ netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:41855        0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 localhost:953           0.0.0.0:*               LISTEN
tcp6       0      0 [::]:domain             [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:953           [::]:*                  LISTEN
sysadmin@localhost:~$ netstat -tln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:41855        0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.2:53          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*
```

14.2.11 Step 11

The `-t` option to the `netstat` command limits the listing to TCP ports; the `-l` option limits the output to ports with listening services; the `-n` shows the network addresses numerically:

```
sysadmin@localhost:~$ netstat -ltn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:41855        0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.2:53          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
```

```
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*
```

14.2.12 Step 12

A more modern approach is to use the `ss` command to view which connections are currently established between the 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 by itself, as shown in the example below. We will use a simple script to generate some traffic first:

```
start_webserver
```

```
sysadmin@localhost:~$ start_webserver
Connecting ...
Connecting ...
Connecting ...
Connecting ...
```

Then we will use the `ss` command to display network statistics:

```
ss
```

```
sysadmin@localhost:~$ ss
Netid State       Recv-Q  Send-Q    Local Address:Port      Peer Address:Port
tcp   CLOSE-WAIT  116     0             127.0.0.1:58006        127.0.0.1:8000
tcp   CLOSE-WAIT  116     0             127.0.0.1:58010        127.0.0.1:8000
tcp   FIN-WAIT-2  0       0             127.0.0.1:8000         127.0.0.1:58014
tcp   CLOSE-WAIT  116     0             127.0.0.1:58014        127.0.0.1:8000
tcp   FIN-WAIT-2  0       0             127.0.0.1:8000         127.0.0.1:58010
tcp   FIN-WAIT-2  0       0             127.0.0.1:8000         127.0.0.1:58006
```

The script started a webserver and created the traffic which is displayed by the `ss` command. This is one way the `ss` command can be used for troubleshooting.

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)
```

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$8HbxqsMEIBTZ0YomlMffYCV9VE1SQ4T2H3SHXw41M02SQtfAdDVE9mqGp2
hr20q.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/bash
```

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:

1. **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.
2. **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.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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/
```

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/221QL5
AX5RjKXpXPJRQ0uVN35TY3/..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:

1. **Username**
   `sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::`
   This field contains the username of the account, which matches the account name in the `/etc/passwd` file.

Password
`sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::`

2. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::`

3. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::`

4. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050:`

5. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::`

6. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5
AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::
```

7. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5
AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::
```

8. 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$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2cQ/221QL5
AX5RjKXpXPJRQ0uVN35TY3/..c7v0.n0:16874:5:30:7:60:15050::
```

9. 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:/b
```

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.

1. **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.
2. **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.
3. **GID**
   `mail:x:12:mail,postfix`
   Each group is associated with a unique group ID (GID) which is placed in this field.
4. **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),27(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),27(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),27(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, to print this information, use the `-G` option:

```
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@localhost:~$ cat /etc/group | grep sysadmin
adm:x:4:syslog,sysadmin
sudo:x:27:sysadmin
sysadmin:x:1001:
```

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.
For example:

```
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:

1. **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)
   ```

2. 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
   ```

3. This column indicates when the user logged in.

4. **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 a value of 100 would mean full CPU usage during that period of time.
The following describes the rest of the output of the `w` command:

Command	Example	Description
US	root	The name of the user who is logged in.
TT	tty2	Which terminal window the user is working
FR	example om	Where the user logged in from.
LO N@	10:00	When the user logged in.
ID	43:44	How long the user has been idle since the la command was executed.
JC	0.01s	The total cpu time used by all processes rur since login.

PC	0.01s	The total cpu time for the current process.
WH	-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.

15.1 Introduction

This is Lab 15: System and User Security. By performing this lab, students will be able to monitor who has been attempting to log in to the system, and view user and group permissions. In this lab, you will perform the following tasks:

- Learn the difference between the superuser account and regular user accounts.
- View user account information.

15.2 Running Commands as an Administrator

In this task, you will learn two ways to run commands as an administrative user. This is often necessary for making changes that affect the whole system.

15.2.1 Step 1

To access the root user account, the `su` or `sudo` commands are normally used.

The `su` command is usually used to switch users and start a new shell as another user, with the default being the root user. The `su` command is often used when a series of commands need to be executed as the root user.

The `sudo` command is typically used to execute a single command as the root user by prefixing that command with `sudo`. The `sudo` command must be configured by the root user before an ordinary user can use it. By default, the `sudo` command stays in effect for 15 minutes on Ubuntu systems where the root account is not enabled by default. Root access has been enabled on the virtual machine used in this lab allowing the `su` command to be used.

When executed without arguments, the `su` command opens a new shell as the root user. There is some confusion as to what the initials “su” stand for (substitute user, switch user, and superuser are all often referenced) , but the main thing to note is that it allows an administrator to change their login to any user on the system.

When entering this command without a username, the system will assume the root user. Most systems will display the current user at the command prompt, but it can be helpful to confirm what user is logged in with the `id` command as shown below. This step will ensure changes required for specific users (such as service accounts) are executed properly.

Switch users to the root user and provide the root password of `netlab123` when prompted:

`su -`

```
sysadmin@localhost:~$ su -
Password:
```

Confirm the new user identity using the `id` command:

```
id
root@localhost:~# id
uid=0(root) gid=0(root) groups=0(root)
```

15.2.2 Step 2

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. Confirm the user identity change using the `id` command.

```
exit
id
root@localhost:~# exit
logout
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
```

Exiting the shell is important to avoid executing commands as root that could damage the system.

15.2.3 Step 3

The `sudo` command works on systems that do not allow root access by default. It is preferred for most administrative tasks since root access times out automatically without having to exit. First type a command as the sysadmin non-privileged user.

```
head /etc/shadow
sysadmin@localhost:~$ head /etc/shadow
head: cannot open `/etc/shadow' for reading: Permission denied
```

Notice the error message that the `head` command displays. This is because the sysadmin user has no rights to view this file. The root user, however, can display this file.

15.2.4 Step 4

Type the same command using `sudo`. Use the `netlab123` password when prompted:

```
sudo head /etc/shadow
sysadmin@localhost:~$ sudo head /etc/shadow
[sudo] password for sysadmin:
root:$6$4Yga95H9$8HbxqsMEIBTZ0YomlMffYCV9VE1SQ4T2H3SHXw41M02SQtfAdDVE9mqGp2hr20q
.ZuncJpLyWkYwQdKlSJyS8.:16464:0:99999:7:::
daemon:*:16463:0:99999:7:::
bin:*:16463:0:99999:7:::
sys:*:16463:0:99999:7:::
sync:*:16463:0:99999:7:::
games:*:16463:0:99999:7:::
man:*:16463:0:99999:7:::
lp:*:16463:0:99999:7:::
mail:*:16463:0:99999:7:::
news:*:16463:0:99999:7:::
sysadmin@localhost:~$
```

The system will prompt for the current user's password, not the root password. If the current user is part of the `sudo` group the command will be executed. All commands entered for the next 15 minutes will be executed as root by default on Ubuntu systems. Other systems may have different timeouts.

15.3 User Accounts

In this task, you will learn about user accounts and the files and commands that display user account information.

15.3.1 Step 1

User and system accounts are defined in the `/etc/passwd` and `/etc/shadow` files. View the first ten lines from the `/etc/passwd` file. While the `passwd` file contains general information about a user such as username, UID, GID, home directory and login shell, the modern `shadow` file has additional details including encrypted password and password policy:

```
head /etc/passwd
sysadmin@localhost:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

Notice that this file contains a colon delimited database of all user and system accounts available on this system.

User accounts are assigned to users, to allow them access to the operating system. The `sysadmin` account that you used to log in to the system is a typical user account.

System accounts are used by the operating system or services running processes on it to perform background functions. These accounts often need access to hardware or system files that normally would only be available to the root user. The default permissions for these accounts normally give access to just the sensitive areas needed rather than the broad access of a root or administrator account, thereby limiting the damage that a compromised service account could cause. System accounts are never used directly by regular users.

15.3.2 Step 2

Use the `grep` command to view the record for your `sysadmin` account:

```
grep sysadmin /etc/passwd
```

```
sysadmin@localhost:~$ grep sysadmin /etc/passwd
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

By using the `grep` command, the output only includes the account information for that one username.

15.4 Passwords

The `/etc/shadow` file contains information about users' passwords. In this exercise you will use several commands to view the data in this file.

15.4.1 Step 1

Try to view the first few lines of `/etc/shadow` file, a file that contains users' encrypted passwords and information about aging them:

```
head -3 /etc/shadow
```

```
sysadmin@localhost:~$ head -3 /etc/shadow
head: cannot open `/etc/shadow' for reading: Permission denied
sysadmin@localhost:~$
```

Notice the error message that the `head` command displays. This is because the `sysadmin` user has no rights to view this file. The `root` user, however, can display this file.

15.4.2 Step 2

Notice that the permissions on the `/etc/shadow` file indicate that only members of the `shadow` group have permission to view the file:

```
ls -l /etc/shadow
```

```
sysadmin@localhost:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 838 Mar 14 17:34 /etc/shadow
sysadmin@localhost:~$
```

Keep in mind that the `root` user can view any file. This is due to the root account having special privileges that transcend regular file permissions.

15.4.3 Step 3

Use the `sudo` command to view the first few lines of the `/etc/shadow` file. Provide the password of the `sysadmin` user, `netlab123`, when prompted.

```
sudo head -3 /etc/shadow
```

```
sysadmin@localhost:~$ sudo head -3 /etc/shadow
[sudo] password for sysadmin:
root:$6$T3W2rbrt$N/2Jrt1EzQ8TqOvxWkYjEpIf3tCbPOyFwU7ZYkToosXB4AGmtb0.W6f8Gb7Vm
ihnj76yZezNPwMbTGoQFs5Kx1:16874:0:99999:7:::
daemon:*:16863:0:99999:7:::
bin:*:16863:0:99999:7:::
sysadmin@localhost:~$
```

Important
The password that you provided was for your sysadmin account, not the root account. Once `sudo` has been configured for your account, you don't need to know the `root` password to run `sudo` commands as the `root` user.

15.4.4 Step 4

Another way to retrieve the account information for a user is by running the following command: `getent passwd username`. The `getent` command has the advantage over the `grep` command as it is also able to access user accounts that are not defined locally. In other words, the `getent` command is able to get user information for users who may be defined on network directory servers such as LDAP, NIS, Windows Domain, or Active Directory Domain servers.
Use the `getent` command to retrieve the information about the sysadmin:

```
getent passwd sysadmin
```

```
sysadmin@localhost:~$ getent passwd sysadmin
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:~$
```

Note
In this case, we don't have any network accounts, so the output displayed is just like looking at the `/etc/passwd` file.
The colon delimited `/etc/passwd` file has the following fields:
`name:password:UID:GID:Comment:directory:shell`
A breakdown of these fields:

1. **Name**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
2. **Password Placeholder**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
3. **User ID**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
4. **Primary Group ID**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
5. **Comment**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
6. **Home Directory**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash`
7. **Shell**
   `sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:`

15.4.5 Step 5

You can view the documentation of the fields in the `/etc/passwd` file with the following command:

```
man 5 passwd
```

```
PASSWD(5)                File Formats and Conversions                PASSWD(5)

NAME
       passwd - the password file

DESCRIPTION
       /etc/passwd contains one line for each user account, with seven fields
       delimited by colons (":"). These fields are:

       o   login name

       o   optional encrypted password

       o   numerical user ID

       o   numerical group ID
```

```
        o   user name or comment field

        o   user home directory

        o   optional user command interpreter
```

Important

Remember while viewing a man page, press **Enter** to move forward line by line, **Space** page by page and **q** to quit.

15.4.6 Step 6

You can view account information for your account, or a specified user account, using the `id` command:

```
id
id root
```

```
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
sysadmin@localhost:~$ id root
uid=0(root) gid=0(root) groups=0(root)
sysadmin@localhost:~$
```

The output of the commands shows the following:

User identity:

```
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
```

Primary group identity:

```
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
```

Groups that you belong to:

```
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo)
```

In this case, your user account only belongs to three groups.

Note

The file `/etc/group`, together with `/etc/passwd`, determines your group memberships. Your default primary group is determined by matching your GID found in `/etc/passwd` to the GID defined for a group in the `/etc/group`. Any secondary group memberships are defined in the `/etc/group` file.

The format of entries in the `/etc/group` file for each line is:

```
group_name:password:GID:user_list
```

15.5 Who is On the System

In this task, you will execute some commands to see who is logged into the system.

15.5.1 Step 1

Use the `who` command to get the current list of users on the system:

```
who
```

```
sysadmin@localhost:~$ who
sysadmin console      Apr 11 14:32
sysadmin@localhost:~$
```

The output of the `who` command has four columns:

Username

```
sysadmin console      Apr 11 14:32
```

This column indicates the name of the user who is logged in.

Terminal

```
sysadmin console      Apr 11 14:32
```

This column indicates which terminal window the user is working in.

Date

```
sysadmin console      Apr 11 14:32
```

This column indicates when the user logged in.

Host

Although there is no output for the fourth column in this case, it can be the name or IP address of a local or remote host.

15.5.2 Step 2

Use the `w` command to get a more detailed view of the users who are currently on your system:

```
w
```

```
sysadmin@localhost:~$ w
```

```
 15:17:08 up 6 days, 15 min,  1 user,  load average: 0.39, 0.34, 0.37
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
sysadmin console                  14:32    4.00s  0.16s  0.00s w
sysadmin@localhost:~$
```

Output from the `w` command displays a summary of how long the system has been running, how many users are logged in and the system load averages for the past 1, 5, and 15 minutes. Also displayed is an entry for each user with their login name, tty name (terminal name), host, login time, idle time, JCPU (CPU time used by background jobs), PCPU (CPU time used by the current process) and what is executing on the current command line.

15.6 Viewing Login History

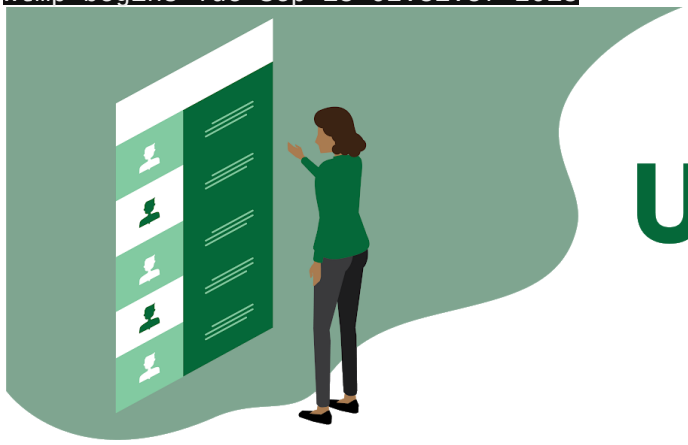
The `last` command reads the entire login history from the `/var/log/wtmp` file and displays all logins and reboot records by default.

15.6.1 Step 1

Use the `last` command to view the `/var/log/wtmp` file which keeps a log of all users who have logged in and out the system.

`last`

```
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
```



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:

```
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

The following describes each of these values:

1. **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.
2. **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.
3. **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.

4. **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.
5. **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.
6. **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.
7. **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:

1. **Mail Directory**
   `MAIL_DIR  /var/mail/spool`
   The directory in which the user's mail spool file is created.
2. **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:`
3. **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:`
4. **Password Minimum Length**
   `PASS_MIN_LEN5`
   This indicates the minimum number of characters that a password must contain.
5. **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:`
6. **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.
7. **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.
8. **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.
9. **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.
10. **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.

11. **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.
12. **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`.
13. **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.
14. **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  Videos
```

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
```

16.3.4 Creating a User

Once you've verified which default values to use and you've gathered the information about the user, then you are ready to create a user account. An example of a `useradd` command using a few options looks like the following:

```
root@localhost:~# useradd -u 1009 -g users -G sales,research -m -c 'Jane Doe'
jane
```

This example of the `useradd` command creates a user with a UID of `1009`, a primary group of users, supplementary memberships in the `sales` and `research` groups, a comment of `"Jane Doe"`, and an account name of `jane`.
After executing the previous command, the information about the `jane` user account is automatically added to the `/etc/passwd` and `/etc/shadow` files, while the information about supplemental group access is automatically added to the `/etc/group` and `/etc/gshadow` files:

```
root@localhost:~# grep jane /etc/passwd
jane:x:1009:100:Jane Doe:/home/jane:/bin/sh
root@localhost:~# grep jane /etc/shadow
jane:!:17883:0:99999:7:30::
root@localhost:~# grep jane /etc/group
research:x:1005:jane
sales:x:999:jane
root@localhost:~# grep jane /etc/gshadow
research:!::jane
sales:!::jane
```

Note that the account doesn't have a valid password yet!
In addition, if the `CREATE_MAIL_SPOOL` is set to `yes` then the mail spool file `/var/spool/mail/jane` is created:

```
root@localhost:~# ls /var/spool/mail
jane root rpc sysadmin
```

Finally, because the `-m` option is used, the `/home/jane` directory is created with permissions only permitting the `jane` user access, and the contents of the `/etc/skel` directory would be copied into the directory:

```
root@localhost:~# ls /home
jane sysadmin
```

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.
It's easy to make a bad password! If you use any information in your password that is related to you, then this information can also be known or discovered by others, resulting in your password being easily compromised. Your password should never contain information about you or anyone you know, such as your first name, middle name, last name, birthday, phone, pet names, drivers license, or social security number.
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 informatic
`-d LAST_DAY`	`--lastday LAST_DAY`	Set the date of the last passwor 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 day before the password can be changed to `MIN_DAYS`
`-M MAX_DAYS`	`--maxdays MAX_DAYS`	Set the maximum number of da 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
```

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 li specified in `GROUPS`.
`-a`	`--append`	Append the user's supplemental groups with those specified by t `-G` option.
`-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
```

16.1 Introduction

This is Lab 16: Creating Users and Groups. By performing this lab, students will learn how to create a new user account, establish the initial password for this account, and make other modifications such as making the new user a member of a secondary group.
In this lab, you will perform the following tasks:

- Create a new group with the `groupadd` command
- Make changes to groups using the `groupmod` command
- Create a new user with the `useradd` command
- Set and reset a user's password with the `passwd` command
- Make changes to the user account with the `usermod` command

16.2 Creating Groups

In this task, you will create group and user accounts.
Group accounts can be helpful to use in order to be able to assign permissions on files shared by a group of users.
User accounts in Linux distributions based upon RedHat, like the CentOS distribution, start with the first User ID (UID) at 500, the next UID given at 501, and so on. The current trend followed by many other distributions is to have the first UID be 1000, the second to be 1001, and so on. Starting with RedHat 7, standard user accounts begin at 1000, a consideration when migrating older systems with existing user accounts.
If managing accounts for multiple systems, then it is desirable to have a network-based authentication server, where accounts can be created once, but used on many machines. Otherwise, managing multiple accounts on multiple machines can be challenging as it can be difficult to ensure that the user, and all the groups they belong to, all have the same UIDs and GIDs on all machines.
Another issue with multiple machine accounts is that it can be difficult to keep the passwords to each account synchronized across all machines.
Managing accounts for local users is still useful for individual machines, even if they have access to a network-based authentication server. In this lab, you will manage local group and user accounts.

16.2.1 Step 1

In order to administer the user and group accounts, you will want to switch users to the root account with the following command. Provide the `root` password `netlab123` when prompted.

`su -`

```
sysadmin@localhost:~$ su -
Password:
root@localhost:~#
```

16.2.2 Step 2

Use the `groupadd` command to create groups called `research` and `sales`:

`groupadd -r research`

```
root@localhost:~#  groupadd -r research
root@localhost:~#  groupadd -r sales
root@localhost:~#
```

The `research` and `sales` groups that were just added were added in the reserved range (between 1-999) because the `-r` option was used. With this option, Group Identifiers (GIDs) are automatically assigned with a value of less than the lowest normal user UID. The `groupadd` command modifies the `/etc/group` file where group account information is stored.

The `groupmod` command can be used with a `-n` option to change the name of either of these groups or with the `-g` option in order to change the GID for either of the groups. The `groupdel` command can be used to delete either of the groups, as long as neither of them have been made the primary group for a user.

16.2.3 Step 3

Use the `getent` command to retrieve information about the new `research` group:

`getent group research`

```
root@localhost:~# getent group research
research:x:999:
```

Your output should appear similar to the example above although the GID that was assigned may be different. Now that the `research` group has been created, existing or new users can be made members of the group.

16.2.4 Step 4

Use the `grep` command to retrieve information about the new `sales` group:

`grep sales /etc/group`

```
root@localhost:~# grep sales /etc/group
sales:x:998:
```

Your output should appear similar to the example above although the GID that was assigned may be different. Now that the `sales` group has been created, existing or new users can be made members of this group.

16.2.5 Step 5

Use the `groupmod` command with the `-n` option to change the name of the `sales` group.

`groupmod -n clerks sales`

```
root@localhost:~# groupmod -n clerks sales
```

Now use the `groupmod` command with the `-g` option to change the GID for the group.

`groupmod -g 10003 clerks`

```
root@localhost:~# groupmod -g 10003 clerks
```

Use the `grep` command to verify the changes made above.

`grep clerks /etc/group`

```
root@localhost:~# grep clerks /etc/group
clerks:x:10003:
```

Important

Note that any files that had been in the `sales` group will now have no group name and will now be orphaned files.

16.2.6 Step 6

Delete the `clerks` group using the `groupdel` command along with the name of the group:

`groupdel clerks`

```
root@localhost:~# groupdel clerks
```

Use the `grep` command to verify that the `clerks` group has been removed:

```
root@localhost:~# grep clerks /etc/group
root@localhost:~#
```

Important

If you decide to delete a group with the `groupdel` command, be aware that any files that are owned by that group will also become orphaned.

16.3 User Configuration

User configuration begins with properly configuring the groups that users will be placed in.

16.3.1 Step 1

View the default values used by the `useradd` command using the `-D` option:

```
useradd -D
```

```
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

The SKEL value provides administrators with an easy way to populate a new user account with key configuration files. It determines which skeleton directory will have its contents copied into the new user's home directory. The `-k` option on the `useradd` command allows a different SKEL directory than the default to be used when creating a new user account. This is useful because most systems have users that need access to different resources as appropriate to their job functions.

16.3.2 Step 2

Set the INACTIVE parameter to allow users with expired passwords to log in for up to thirty days before their accounts are disabled, then view the new default values. The `-D` option to the `useradd` command will allow you to view or change some of the default values used by the `useradd` command.

In the example below, the `-D` option specifies changes to the default values used when creating a new user. The `-f 30` option specifies that users who have expired passwords can still log in for up to thirty days before their accounts are inactivated. Using the `-D` option by itself displays the current defaults, which have been changed by the previous command.

```
useradd -D -f 30
useradd -D
```

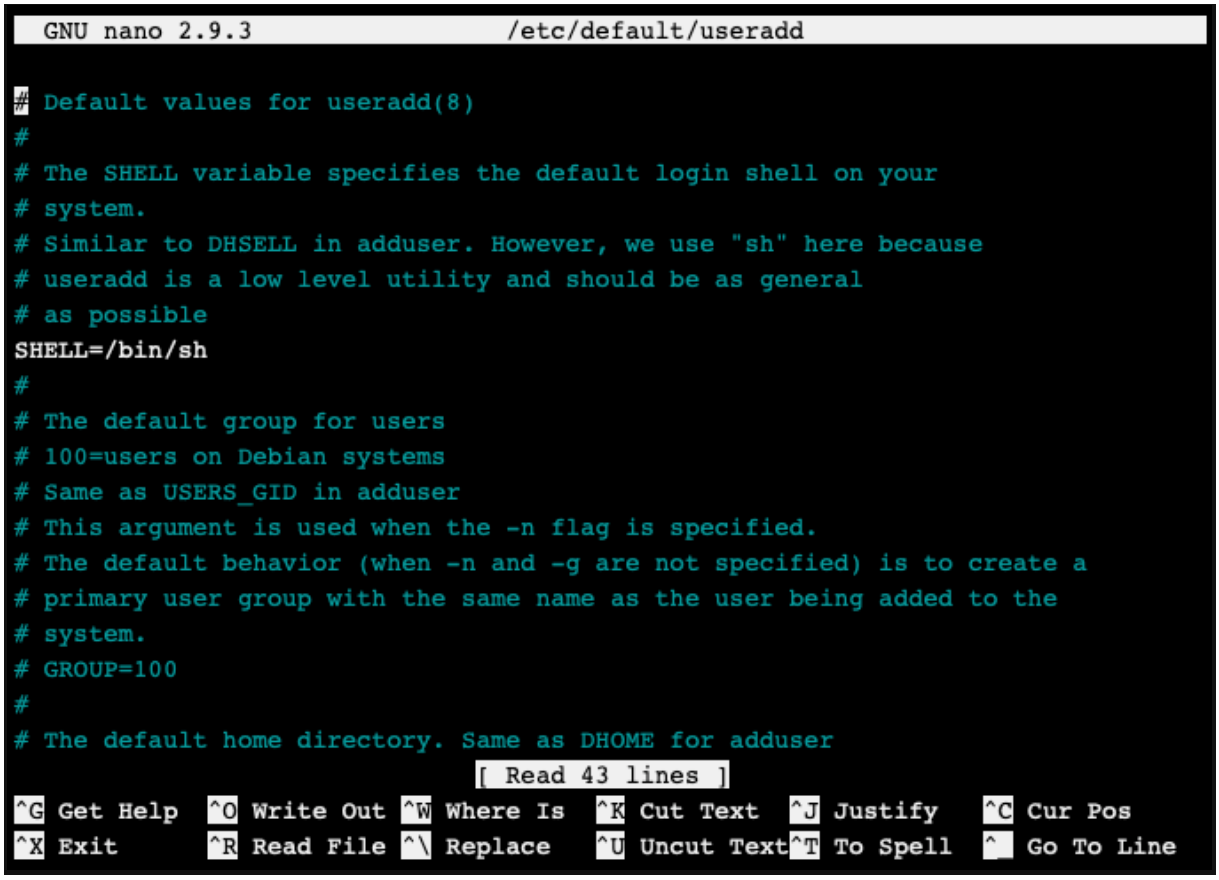
```
root@localhost:~# useradd -D -f 30
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=30
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
root@localhost:~#
```

16.3.3 Step 3

Modify the CREATE_MAIL_SPOOL value in the `/etc/default/useradd` file using the `nano` text editor:

```
nano /etc/default/useradd
```

```
root@localhost:~# nano /etc/default/useradd
```

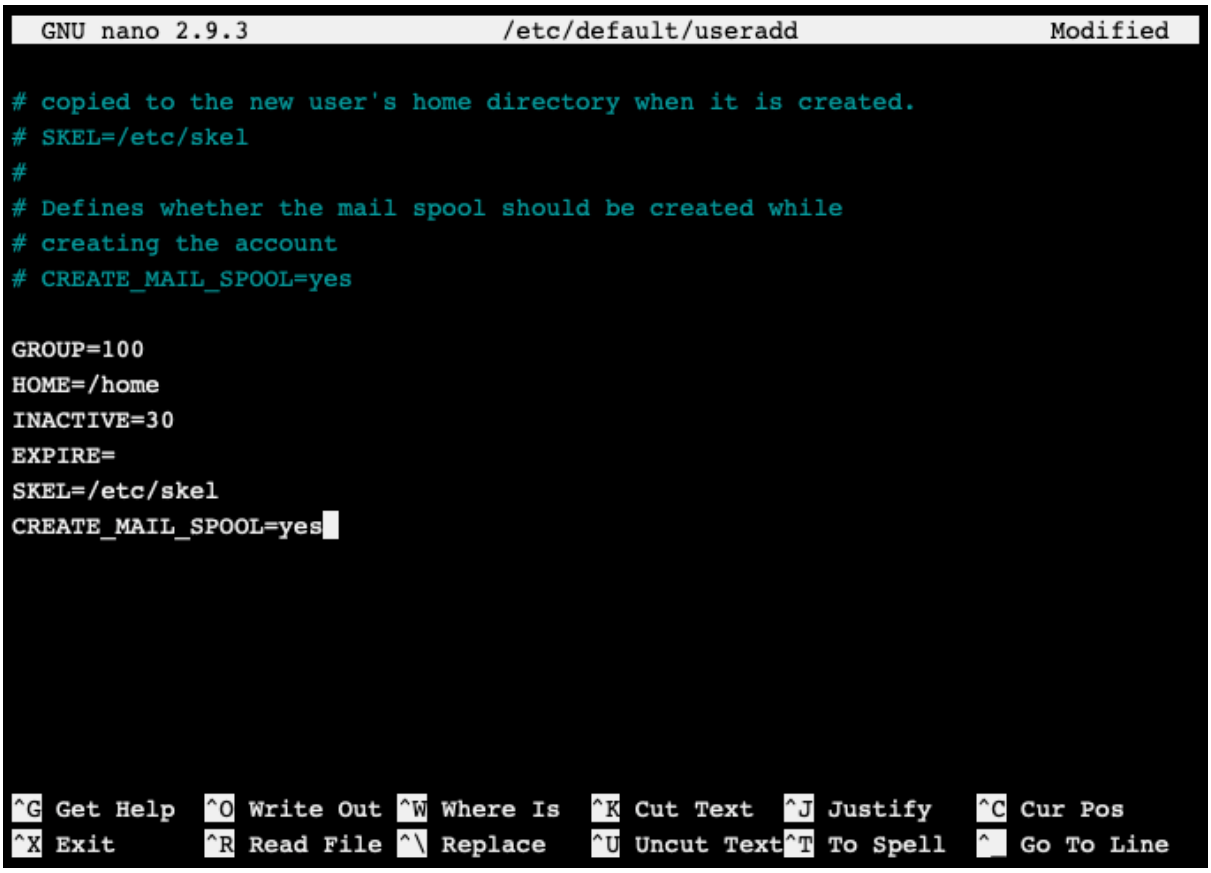


16.3.4 Step 4

Press the **down arrow key** to scroll to the bottom of the file:

16.3.5 Step 5

On the `CREATE_MAIL_SPOOL=no` line, backspace over the `no` and type `yes`:



Press **Ctl** + **X** to exit and type **Y**. Press **Enter** to save your changes then type `useradd -D` at the prompt to confirm the new setting:

```
useradd -D
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=30
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

The `useradd` command will now create a mail spool file.

16.3.6 Step 6

Create a new user named `student` who is a secondary member of the `research` group and a primary member of their own private group. Use a comment of `Linux Student` that will appear as the full name of the user when they do a graphical login. Make sure that their home directory will be created by specifying the `-m` option. Then use `grep` to verify the new user and their group memberships:

```
useradd -G research -c 'Linux Student' -m student
grep student /etc/passwd
grep student /etc/group
root@localhost:~# useradd -G research -c 'Linux Student' -m student
root@localhost:~# grep student /etc/passwd
student:x:1002:1002:Linux Student:/home/student:/bin/sh
root@localhost:~# grep student /etc/group
research:x:999:student
student:x:1002:
```

The user's account information is stored in the `/etc/passwd` and `/etc/shadow` files. The user's group information can be found in the `/etc/passwd` and `/etc/group` files.

16.3.7 Step 7

Use the `usermod` command to add the `research` group as a secondary group for the `sysadmin` user:

```
usermod -aG research sysadmin
root@localhost:~# usermod -aG research sysadmin
root@localhost:~#
```

Users who are actively logged into the system will not be able to use any new group memberships until the next time they log into the system.

16.3.8 Step 8

Using the `getent` command, view the `research` group members again:

```
getent group research
root@localhost:~# getent group research
research:x:999:student,sysadmin
```

Use `getent` to show the `student` group:

```
getent group student
root@localhost:~# getent group student
student:x:1002:
```

Next, use `getent` to show the `passwd` and `shadow` databases for the `student` user:

```
getent passwd student
getent shadow student
root@localhost:~# getent passwd student
student:x:1002:1002:Linux Student:/home/student:/bin/sh
root@localhost:~# getent shadow student
student:!:16902:0:99999:7:30::
root@localhost:~#
```

The output should now show that both `sysadmin` and `student` are secondary members of the `research` group.

The GID of the `student` group matches the fourth field of the `passwd` information for the `student` user. This is what makes the student a primary member of the `student` group.

Finally, the `!` appearing in the second password field of the `shadow` file, shows that the password for the `student` has not been set.

16.3.9 Step 9

Use the `passwd` command to set the password, `netlab123`, for the `student` user. Enter the password twice then view the `shadow` file entry for the `student` user again:

```
passwd student
root@localhost:~# passwd student
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Note

No characters will appear when typing the password.

The output from the `/etc/shadow` file now shows an encrypted password in the second field:

```
getent shadow student
root@localhost:~# getent shadow student
student:$6$pIEEdvAX$GBo0beYhojL3/vDrOP2UAQR6uVCWMZXxMPqImREJWw/5o
```

16.3.10 Step 10

Just because a user has a password doesn't mean that they have ever logged into the system. Use the `last` command to see if the `student` user has ever logged in:

```
last
last student
root@localhost:~# last
sysadmin console                      Thu Jan  3 21:44   still logged in
wtmp begins Thu Jan  3 21:44:06 2019
root@localhost:~# last student
wtmp begins Thu Jan  3 21:44:06 2019
root@localhost:~#
```

The output of the `last` command should show that the `sysadmin` user has logged in before, but not the `student` user. There is also a `lastb` command, which works similar to the `last` command except that it shows "bad" or failed login attempts.

If you no longer wanted the `student` user to have access to the system, then the `usermod -L student` command could be used to "lock" the account. The account could be unlocked with the `usermod -U student` command.

16.3.11 Step 11

A more permanent solution to preventing access to the `student` account would be to delete the account with either the `userdel student` or `userdel -r student` commands. Using the `-r` option with the `userdel` command removes the user's home directory and mail, in addition to deleting the user's account.

Delete the `student` account and remove the user's home directory:

`userdel -r student`

```
root@localhost:~# userdel -r student
Use grep to verify the student user has been removed.
root@localhost:~# grep student /etc/group
root@localhost:~#
```


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:

1. **Permissions**
   `-rw-rw-r--. 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/filetest1`
2. **User Owner**
   `-rw-rw-r--. 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/filetest1`
3. **Group Owner**
   `-rw-rw-r--. 1 sysadmin sysadmin 0 Oct 21 10:18 /tmp/file`

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 empty file
Device: fd00h/64768d    Inode: 31477       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1001/sysadmin)   Gid: ( 1001/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
```

17.5 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 `abc.txt` 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
```

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 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.

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
```

17.7.1.1 Scenario #1 - Answer

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.

17.7.2 Scenario #2 - Viewing Directory Contents

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
```

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 Scenario #3 - Deleting Directory Contents

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
```

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 Scenario #4 - Accessing the Contents of a Directory

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
```

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 Scenario #5 - The Complexity of Users and Groups

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
```

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 Scenario #6 - Permission Priority

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
```

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.
The following examples will use a sample file:

```
root@localhost:~# touch abc.txt
root@localhost:~# ls -l abc.txt
-rw-r--r-- 1 root root 0 Dec 19 18:58 abc.txt
```

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
```

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:

```
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.

17.1 Introduction

This is Lab 17: Ownerships and Permissions. By performing this lab, students will learn how to set and view the ownerships and permissions of files and directories.
In this lab, you will perform the following tasks:

- View and understand permissions on files and directories.
- Use the `chmod` command to change permissions.
- Change ownerships with the `chown` and `chgrp` commands.

17.2 File Permissions

In this task, you will create files and directories, and view and set their permissions.

17.2.1 Step 1

In this step, you will create two directories and two files in the `/tmp` directory.
Move to the `/tmp` directory:

```
cd /tmp
```

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

Use the `mkdir` command to create two directories called `priv-dir` and `pub-dir`:

```
mkdir priv-dir pub-dir
```

```
sysadmin@localhost:/tmp$ mkdir priv-dir pub-dir
```

Create two files, one file called `priv-file` in the `priv-dir` directory and another file called `pub-file` in the `pub-dir` directory:

```
touch priv-dir/priv-file
touch pub-dir/pub-file
```

```
sysadmin@localhost:/tmp$ touch priv-dir/priv-file
sysadmin@localhost:/tmp$ touch pub-dir/pub-file
sysadmin@localhost:/tmp$
```

17.2.2 Step 2

File ownership can be confirmed using the long listing `-l` option of the `ls` command. Use this command to view the contents of the new directories:

```
ls -l priv-dir
ls -l pub-dir
```

Your output should contain the following:

```
sysadmin@localhost:/tmp$ ls -l priv-dir
total 0
-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:26 priv-file
sysadmin@localhost:/tmp$ ls -l pub-dir
total 0
-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-file
sysadmin@localhost:/tmp$
```

Notice that for each file listed, the first character of the line is the hyphen, `-`. This conveys the information that the items are regular files. The first character of the listing indicates the type of file. These file types are listed in the table below:

Character	Meaning
d	indicates a directory
-	is a regular file
l	is a symbolic link
b	is a block device file
c	is a character device file
p	is a pipe file
s	is a socket file

The next nine characters are in three groups of three characters.
The first group of three characters is the user owner's permissions:
`-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-file`
The next three characters are the group owner's permissions:
`-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-file`
The last three characters represent everyone else's permissions (referred to as "others"):
`-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-file`
When viewing permissions, `r` indicates the read permission, `w` indicates the write permission and `x` indicates execute permission. A `-` character indicates that that permission has not been granted.
Following the permissions is a link count number, indicating how many files are linked to this file. Next, you see the user owner, the group owner, the size of the file, the date/time the file was last modified and the file name.

17.2.3 Step 3

File ownership also applies to hidden files in the system. Hidden files, which begin with the dot `.` character are listed using the `-a` option of the `ls` command. The first two hidden “files” listed are actually 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.
`ls -la`

```
sysadmin@localhost:/tmp$ ls -la
total 12
drwxrwxrwt 1 root root 4096 Jan  2 04:17 .
drwxr-xr-x 1 root root 4096 Jan  2 04:14 ..
-rw-rw-r-- 1 root root 2042 Dec 10 21:32 inside_setup.sh
drwxrwxr-x 2 sysadmin sysadmin 4096 Nov 20 01:51 priv-dir
drwxrwxr-x 2 sysadmin sysadmin 4096 Nov 20 01:51 pub-dir
```

17.2.4 Step 4

If you want to make a directory more private use the `chmod` command to remove permissions that others have on the directory. Using the `-d` option with the `ls` command will list directory entries instead of contents.

Use the `ls -ld` command to view permissions for the `priv-dir` directory, then use the `chmod` command to remove the others' permissions for read and execute:

```
ls -ld priv-dir/
chmod o-rx priv-dir/
```

```
sysadmin@localhost:/tmp$ ls -ld priv-dir/
drwxrwxr-x 2 sysadmin sysadmin 4096 Apr 11 21:26 priv-dir/
sysadmin@localhost:/tmp$ chmod o-rx priv-dir/
```

Finally, use the same ls command to verify the change in permissions. The output now shows that others have no permission or access to the priv-dir:

```
ls -ld priv-dir/
```

```
sysadmin@localhost:/tmp$ ls -ld priv-dir/
drwxrwx--- 2 sysadmin sysadmin 4096 Apr 11 21:26 priv-dir/
sysadmin@localhost:/tmp$
```

You can use the `chmod` command to modify the permissions for others by using an o character followed by either a + character or a - character to add or subtract permissions. The = character can be used to set an exact permission.

You could also use a u character instead of an o character to modify the permissions of the user owner. Use a g character if you want to change permissions for the group owner. This method of changing permissions is referred to as symbolic since it uses letters to represent permission groups. The u, g, o or user, group, owner notation may be easier to use than the octal method which relies on a binary translation of an octal (base 8) number:

To modify everyone's permissions use an a character instead of o, u or g.

Examples:

`chmod a+x file`	Gives everyone execute permission
`chmod g-w file`	Removes write permission for group owners
`chmod go+r file`	Adds read permission for group owner and others
`chmod o=rwx`	Sets others permissions to read, write and execute

17.2.5 Step 5

If you want to make a directory more public, then you can use the `chmod` command to add write permission for others:

```
ls -ld pub-dir/
chmod o+w pub-dir/
ls -ld pub-dir/
```

Your output now shows that others have write permission on the directory (the ability to add or delete files inside the directory):

```
sysadmin@localhost:/tmp$ ls -ld pub-dir/
drwxrwxr-x 2 sysadmin sysadmin 4096 Apr 11 21:27 pub-dir/
sysadmin@localhost:/tmp$ chmod o+w pub-dir/
sysadmin@localhost:/tmp$ ls -ld pub-dir/
drwxrwxrwx 2 sysadmin sysadmin 4096 Apr 11 21:27 pub-dir/
sysadmin@localhost:/tmp$
```

17.2.6 Step 6

Use the `chmod` command to remove any permission from the group and others on the `priv-file`:

```
ls -l priv-dir/priv-file
chmod g-rw,o-r priv-dir/priv-file
ls -l priv-dir/priv-file
```

Your output should appear similar to the following:

```
sysadmin@localhost:/tmp$ ls -l priv-dir/priv-file
-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:26 priv-dir/priv-file
sysadmin@localhost:/tmp$ chmod g-rw,o-r priv-dir/priv-file
sysadmin@localhost:/tmp$ ls -l priv-dir/priv-file
-rw------- 1 sysadmin sysadmin 0 Apr 11 21:26 priv-dir/priv-file
sysadmin@localhost:/tmp$
```

17.2.7 Step 7

Grant all users the same read and write permission for the `pub-file`:

```
ls -l pub-dir/pub-file
chmod a=rw pub-dir/pub-file
ls -l pub-dir/pub-file
```

Your output should appear similar to the following:

```
sysadmin@localhost:/tmp$ ls -l pub-dir/pub-file
-rw-rw-r-- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-dir/pub-file
sysadmin@localhost:/tmp$ chmod a=rw pub-dir/pub-file
sysadmin@localhost:/tmp$ ls -l pub-dir/pub-file
-rw-rw-rw- 1 sysadmin sysadmin 0 Apr 11 21:27 pub-dir/pub-file
sysadmin@localhost:/tmp$
```

17.2.8 Step 8

Execute permissions are required for users to run executable files. Create a `test.sh` file in the `/tmp` containing the content `"date"`:

```
echo "date" > test.sh
sysadmin@localhost:/tmp$ echo "date" > test.sh
sysadmin@localhost:/tmp$
```

If a file contains commands, then those commands can be run, or executed, only if the file has execute permission for the user. The process of making a file into an executable file requires giving the execute permission on the file. Without this permission, the file can't be treated as a program.

17.2.9 Step 9

Attempt to execute the `test.sh` file; it should fail. View the permissions on the file to see why:

```
./test.sh
ls -l test.sh
sysadmin@localhost:/tmp$ ./test.sh
-bash: ./test.sh: Permission denied
sysadmin@localhost:/tmp$ ls -l test.sh
-rw-rw-r-- 1 sysadmin sysadmin 5 Apr 11 22:27 test.sh
sysadmin@localhost:/tmp$
```

17.2.10 Step 10

Only the user owner of a file (or the `root` user) is allowed to change permissions on a file. Give yourself, the user owner, execute permission and then execute `test.sh`:

```
chmod u+x test.sh
ls -l test.sh
./test.sh
```

The output shows the added execute permission for the user owner and the current date and time from executing the `date` command inside of your `test.sh` script file:

```
sysadmin@localhost:/tmp$ chmod u+x test.sh
sysadmin@localhost:/tmp$ ls -l test.sh
-rwxrw-r-- 1 sysadmin sysadmin 5 Apr 11 22:27 test.sh
sysadmin@localhost:/tmp$ ./test.sh
Tue Nov 20 13:09:08 UTC 2018
sysadmin@localhost:/tmp$
```

So far, you have seen how to use the `chmod` command with symbolic notation, where symbols are used to represent who (u, g, o, and a), how (+, -, or =), and what to change (r, w, and x).
The `chmod` command can also be used with a numeric value representing the permissions of the user owner, group owner and others in what is called octal notation.
To use the `chmod` command with octal notation, you must first understand the octal value of the permissions:

Read (r)	4
Write (w)	2
Execute (x)	1

From the last listing of the `test.sh` file, the permissions were shown to be `rwx` for the user owner, `rw` for the group owner, and `r` for others. To express these permissions in octal notation, a total is calculated for each ownership.
As a result, the user's permission total would be calculated as 4 + 2 + 1, or 7, where 4 is for the read, 2 is for the write, and 1 is for the execute permission.
The group's permission total would be 4 + 2 or 6, where 4 is for the read permission, and 2 is for the write permission.
The others' ownership permission would simply be 4 for the read permission that they have.
Putting it all together, the octal value for the current permissions would be 764.

17.2.11 Step 11

The `stat` command displays more detailed information about a file, including providing the group ownership both by group name and GID number. Use the `stat` command to verify the octal value for the permissions (access) to `test.sh`:

```
stat test.sh
```

```
sysadmin@localhost:/tmp$ stat test.sh
  File: test.sh
  Size: 5               Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 96600205    Links: 1
Access: (0764/-rwxrw-r--)  Uid: ( 1001/sysadmin)   Gid: ( 1001/sysadmin)
Access: 2018-11-20 13:05:54.386842683 +0000
Modify: 2018-11-20 13:05:54.386842683 +0000
Change: 2018-11-20 13:08:56.566838726 +0000
 Birth: -
```

The permissions are listed in the `Access:` field of the `stat` output. If you wanted to change these permissions using octal notation to give the group and others execute permission, then you would use the following three numbers:

- 7 (read, write and execute) for the user owner
- 7 (read, write and execute) for the group owner
- 5 (read and execute) for others

The new mode, or octal number, for the permissions would then be 775.

17.2.12 Step 12

Using octal notation, modify the permissions of the `test.sh` file, so everyone would be able to execute the file:

```
chmod 775 test.sh
ls -l test.sh
```

```
sysadmin@localhost:/tmp$ chmod 775 test.sh
sysadmin@localhost:/tmp$ ls -l test.sh
-rwxrwxr-x 1 sysadmin sysadmin 5 Apr 11 22:27 test.sh
sysadmin@localhost:/tmp$
```

Now all users on this system will be able to execute the `test.sh` file.

17.3 File Ownership

In this task, you will view and change the ownership of files and directories.

17.3.1 Step 1

There are two commands that can affect the ownership of files: the `chown` command and the `chgrp` command. The `chown` command can only be executed by the `root` user and it can change both the user and group that owns a file.
The `chgrp` command can be used by either the user who owns a file or by the `root` user.
The `chgrp` command only changes the group that owns a file.
When a non-root user uses the `chgrp` command, they can only change the group ownership to a group of which they are a member. The `root` user can use `chgrp` to change the group ownership of any file to any group.
Switch to the `root` user, using the root password of `netlab123` when prompted, so you will be able to execute both the `chown` and `chgrp` commands to change group ownerships to any group:

```
su -
```

```
sysadmin@localhost:/tmp$ su -
Password:
root@localhost:~#
```

17.3.2 Step 2

Change back to the `/tmp` directory and list the details of the `pub-dir`, and then its contents:

```
cd /tmp
ls -ld pub-dir
ls -l pub-dir/pub-file
```

Notice the output shows the directory and the file owned by the `sysadmin` user, and the `sysadmin` group:

```
root@localhost:~# cd /tmp
root@localhost:/tmp# ls -ld pub-dir/
drwxrwxrwx 2 sysadmin sysadmin 4096 Apr 11 22:48 pub-dir/
root@localhost:/tmp# ls -l pub-dir/pub-file
-rw-rw-rw- 1 sysadmin sysadmin 0 Apr 11 22:48 pub-dir/pub-file
root@localhost:/tmp#
```

17.3.3 Step 3

Use the `chown` command to change the user and group owner of `pub-dir` to the `root` user and the `root` group. Then, view the details of the directory:

```
chown root:root pub-dir
ls -ld pub-dir
```

Your output should show both the user and group owners have changed:

```
root@localhost:/tmp# chown root:root pub-dir
root@localhost:/tmp# ls -ld pub-dir/
drwxrwxrwx 2 root root 4096 Apr 11 22:48 pub-dir/
root@localhost:/tmp#
```

17.3.4 Step 4

Use the `chown` command to change the user owner of the `pub-file` to the `bin` user:

```
chown bin pub-dir/pub-file
ls -l pub-dir/pub-file
```

The output now shows that the user owner has been updated to `bin`:

```
root@localhost:/tmp# chown bin pub-dir/pub-file
root@localhost:/tmp# ls -l pub-dir/pub-file
-rw-rw-rw- 1 bin sysadmin 0 Apr 11 22:48 pub-dir/pub-file
root@localhost:/tmp#
```

17.3.5 Step 5

View the details of the `priv-dir` and its contents:

```
ls -ld priv-dir
ls -l priv-dir/priv-file
```

The output should show that `priv-dir` is owned by the `sysadmin` user and `sysadmin` group:

```
root@localhost:/tmp# ls -ld priv-dir
drwxrwx--- 2 sysadmin sysadmin 4096 Apr 11 22:48 priv-dir
root@localhost:/tmp# ls -l priv-dir/priv-file
-rw------- 1 sysadmin sysadmin 0 Apr 11 22:48 priv-dir/priv-file
root@localhost:/tmp#
```

17.3.6 Step 6

To change the group ownership of all of the files of a directory structure, use the recursive `-R` option to the `chgrp` command. Change the group owner of the `priv-dir` and `priv-file` to the `users` group recursively with the `chgrp` command and view the updated files:

```
ls -ld priv-dir
ls -l priv-dir/priv-file
chgrp -R users priv-dir
ls -ld priv-dir
ls -l priv-dir/priv-file
```

```
root@localhost:/tmp# ls -ld priv-dir
drwxrwx--- 2 sysadmin sysadmin 4096 Apr 11 22:48 priv-dir
root@localhost:/tmp# ls -l priv-dir/priv-file
-rw------- 1 sysadmin sysadmin 0 Apr 11 22:48 priv-dir/priv-file
root@localhost:/tmp# chgrp -R users priv-dir
root@localhost:/tmp# ls -ld priv-dir
drwxrwx--- 2 sysadmin users 4096 Apr 11 22:48 priv-dir
root@localhost:/tmp# ls -l priv-dir/priv-file
```

```
-rw------- 1 sysadmin users 0 Apr 11 22:48 priv-dir/priv-file
root@localhost:/tmp#
```
Your output reflects the fact that when applying changes recursively to a directory, the changes apply to the directory and anything it contains. This would mean that every subdirectory under `priv-dir` and every file in `priv-dir` and all of its subdirectories would have this change applied.

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 /tmp
drwxrwxrwt 1 root root 4096 Mar 14  2016 /tmp
```

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/valuable-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
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:~$ echo data > file.original
sysadmin@localhost:~$ ls -li file.*
278772 -rw-rw-r--. 1 sysadmin sysadmin 5 Oct 25 15:42 file.original
```

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:~$ ln file.original file.hard.1
sysadmin@localhost:~$ ls -li file.*
278772 -rw-rw-r--. 2 sysadmin sysadmin 5 Oct 25 15:53 file.hard.1
278772 -rw-rw-r--. 2 sysadmin sysadmin 5 Oct 25 15:53 file.origin
```

18.5.2 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/grub/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:~$  ln -s /etc/passwd mypasswd
sysadmin@localhost:~$  ls -l mypasswd
lrwxrwxrwx. 1 sysadmin sysadmin 11 Oct 31 13:17 mypasswd -> /etc/passwd
```

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.txt
```

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/passwd
```

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 Linux.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
```

18.1 Introduction

This is Lab 18: Special Directories and Files. By performing this lab, students will learn about special permissions and different kinds of link files.
In this lab, you will perform the following tasks:

- View files with special permissions
- Create hard and soft links

18.2 Viewing Special Permissions

In this task, you will find and understand the purpose of special permissions beyond read, write, and execute.

18.2.1 Step 1

Using the `-d` option for the `ls` command lists directory information; combined with the `-l` option it shows ownership and permissions for the directory files. List the details of the `/tmp` and `/var/tmp` directories:

```
ls -ld /tmp
ls -ld /var/tmp
```

The output shows the permissions on these directories to be the same:

```
sysadmin@localhost:~$ ls -ld /tmp
drwxrwxrwt 1 root root 4096 Dec  5 20:17 /tmp
sysadmin@localhost:~$ ls -ld /var/tmp
drwxrwxrwt 2 root root 4096 May 26  2018 /var/tmp
sysadmin@localhost:~$
```

The `/tmp` and `/var/tmp` directories are read, write and executable for everyone. Besides the users' home directories, these two "temporary" directories are the locations in the filesystem where ordinary users can create new files or directories.
This does pose a problem: if all users can create new files, they are also able to delete existing files. This is because the write permission on a directory grants users the ability to add and delete files in a directory.
The `t` in the execute column for the others permissions indicates that this directory has the sticky bit permission set. This special permission means that even though everyone can add files in these directories, only the user who creates a file can delete that file.
The root user is not affected by this permission as that account can delete all files in the directory, regardless of ownership.

18.2.2 Step 2

View the permissions on the `/etc/shadow` file:
`ls -l /etc/shadow`
The output shows that the root user has read and write permissions, members of the shadow group have read permission, and others have no permission on this file:
```
sysadmin@localhost:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 941 Oct 31 19:52 /etc/shadow
sysadmin@localhost:~$
```
Like the other files found in the `/etc` directory, the `/etc/shadow` file contains host-specific configuration information. Specifically, the `/etc/shadow` file contains the encrypted passwords of all local user accounts and information about password aging (how long a password is good). Since this is very sensitive information, access to this file is limited to the root user and commands executing as root, as well as members of the `shadow` group.
When a user updates their password with the `passwd` command, the `passwd` command executes with a special permission called setuid. The setuid permission causes a file to execute as the user that owns the file, instead of the user that is actually running the command.
If you are coming from a Microsoft Windows background, you can think of setuid to be like the "Run as administrator" feature provided in Windows.

18.2.3 Step 3

View the permissions of the `/usr/bin/passwd` file:
`ls -l /usr/bin/passwd`
The listing of this file shows:
```
sysadmin@localhost:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Jan 25  2018 /usr/bin/passwd
sysadmin@localhost:~$
```
Notice the `s` in the user's execute permission column. This indicates that this file has the setuid permission set, so it executes as the user who owns it (`root`) instead of the user running the command.
Thus, the `passwd` command is able to update the `/etc/shadow` file, as it executes as the `root` user (recall that the `root` user can edit any file, regardless of the permissions on the file).

18.2.4 Step 4

View the permissions on the `/usr/bin/wall` command:
`ls -l /usr/bin/wall`
The output now shows the `s` in the execute column for the group:
```
sysadmin@localhost:~$ ls -l /usr/bin/wall
-rwxr-sr-x 1 root tty 30800 May 16  2018 /usr/bin/wall
sysadmin@localhost:~$
```
The `s` in the group execute column indicates that this file has the setgid permission set, so it executes as the group that owns it (`tty`) instead of the group of the user running the command.
Thus, the `wall` command is able to write to all terminals (`ttys`) as it executes as the `tty` group.

Note

This is very similar to the setuid permission, but instead of running the command as the user owner of the program, the command runs as the group owner of the program.

Consider This

So far, you've seen three types of special permissions: sticky bit on a directory, setuid on an executable file and setgid on an executable file. As you have seen, these three types exist on a typical system.
One more special permission type can be used; the setgid permission can also be applied to a directory:
`drwxrwsrwx. 2 root demo 4096 Oct 30 23:20 /tmp/data`
If you see the `s` in the execute column for the group that owns directory, then the directory has the setgid permission. When a directory has the setgid permission, then any new file or directory created in that directory will automatically be owned by the group that owns the directory, not by the group of the user who created the file.

18.3 Hard and Soft Links

In this task, you will create and use hard and soft links.

18.3.1 Step 1

Change to your home directory:
`cd`

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

18.3.2 Step 2

Create a file named `source` containing the text "`data`" by using redirection:

```
echo "data" > source
```

```
sysadmin@localhost:~$ echo “data” > source
```

18.3.3 Step 3

Using the `-i` option with the `ls` command prints the index number of the file. View the details and inode information of the `source` file:

```
ls -li source
```

The highlighted output shows that the file has an inode number of `2076` (this number will vary from one system to another) and a link count of `1`:

```
sysadmin@localhost:~$ ls -li source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

The Linux operating system uses inodes to keep track of the information about a file. A directory entry associates an inode with a file name.

Creating a hard link creates another directory entry associated with an existing inode, and will increase the link count number.

Hard links allow you to use multiple names to refer to the same file. If any one of these names is removed, then the other names can still be used to refer to the file.

In fact, these other names can even be used to create additional links. All of these names are considered equivalent as they all refer to an existing inode.

Important

You cannot create hard links to directories. Also, a hard link to a file must exist within the same filesystem (partition) as the file that it links to.

18.3.4 Step 4

In order 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 that will link to the target. Use the `ln` command to create a hard link. View the details and inode information of the source and new hard link file:

```
ln source hardlink
ls -li source hardlink
```

Notice that the `hardlink` file and the original `source` file share the same inode:

```
sysadmin@localhost:~$ ln source hardlink
sysadmin@localhost:~$ ls -li source hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

18.3.5 Step 5

Use the `ln` command to create another hard link to the `source` file. View the details and inode information of the source and new hard link files:

```
ln hardlink hardlinktwo
ls -li hardlink hardlinktwo source
```

Notice the output continues to show that the hard links share the same inode and the link count increases on all links when a link is added:

```
sysadmin@localhost:~$ ln source hardlinktwo
sysadmin@localhost:~$ ls -li hardlink hardlinktwo source
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 hardlink
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 hardlinktwo
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

18.3.6 Step 6

The `rm` command is used to remove files. Remove the last link that was created and list the `source` and `hardlink` files:

```
rm hardlinktwo
ls -li source hardlink
```

Notice that the link count decreases when one of the hard-linked files is removed:

```
sysadmin@localhost:~$ rm hardlinktwo
```

```
sysadmin@localhost:~$ ls -li source hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

18.3.7 Step 7

Remove the `hardlink` file and list the `source` file details:

```
rm hardlink
ls -li source
```

```
sysadmin@localhost:~$ rm hardlink
sysadmin@localhost:~$ ls -li source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

18.3.8 Step 8

Another type of link that can be created is known as a symbolic link or soft link. Symbolic links do not increase the link count of files with which they are linked.

Symbolic link files have their own inode and type of file. Instead of linking and sharing an inode, they link to the file name. Unlike hard links, soft links can be linked to directories and can cross devices and partitions to their targets.

The `-s` option for the `ln` command creates a symbolic link instead of a hard link. Create a symbolic link to the `source` file and view the details of both files:

```
ln -s source softlink
ls -li source softlink
```

The highlighted output shows that the `softlink` and the `source` file have different inodes. Notice that the link type of file is created when making a symbolic link. The permissions of the link are irrelevant, as it is the permissions of the target file that determine access:

```
sysadmin@localhost:~$ ln -s source softlink
sysadmin@localhost:~$ ls -li source softlink
2086 lrwxrwxrwx 1 sysadmin sysadmin 6 Apr 12 13:56 softlink -> source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

18.3.9 Step 9

Create a symbolic link to the /proc directory and display the link:

```
ln -s /proc crossdir
ls -l crossdir
```

The success of these commands shows that soft links can refer to directories, and that they can cross from one filesystem to another, which are two things that hard links cannot do:

```
sysadmin@localhost:~$ ln -s /proc crossdir
sysadmin@localhost:~$ ls -l crossdir
lrwxrwxrwx 1 sysadmin sysadmin 5 Apr 12 14:00 crossdir -> /proc
sysadmin@localhost:~$
```