Linux Essentials

Muhammad Magdy

www.linkedin.com/in/muhammad-magdy-996323131

Introduction to Linux:

- **Linux is a Kernel**

- 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 and quickly gained popularity in research and academic settings, as well as amongst programmers who were attracted to its modularity.

- Linux started in 1991 as a hobby project of **Linus Torvalds** and it has grown to be the dominant operating system on the Internet. 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.

- **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 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.

- 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. 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.

Linux Distributions:

- 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.

- Red Hat:

*** Red Hat** started as a simple distribution that introduced the Red Hat Package Manager (RPM). The developer 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.

* 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.

* **Scientific Linux** is an example of a specific-use distribution based on Red Hat which designed to enable scientific computing.

- 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.

* 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, 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 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.

*** 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.

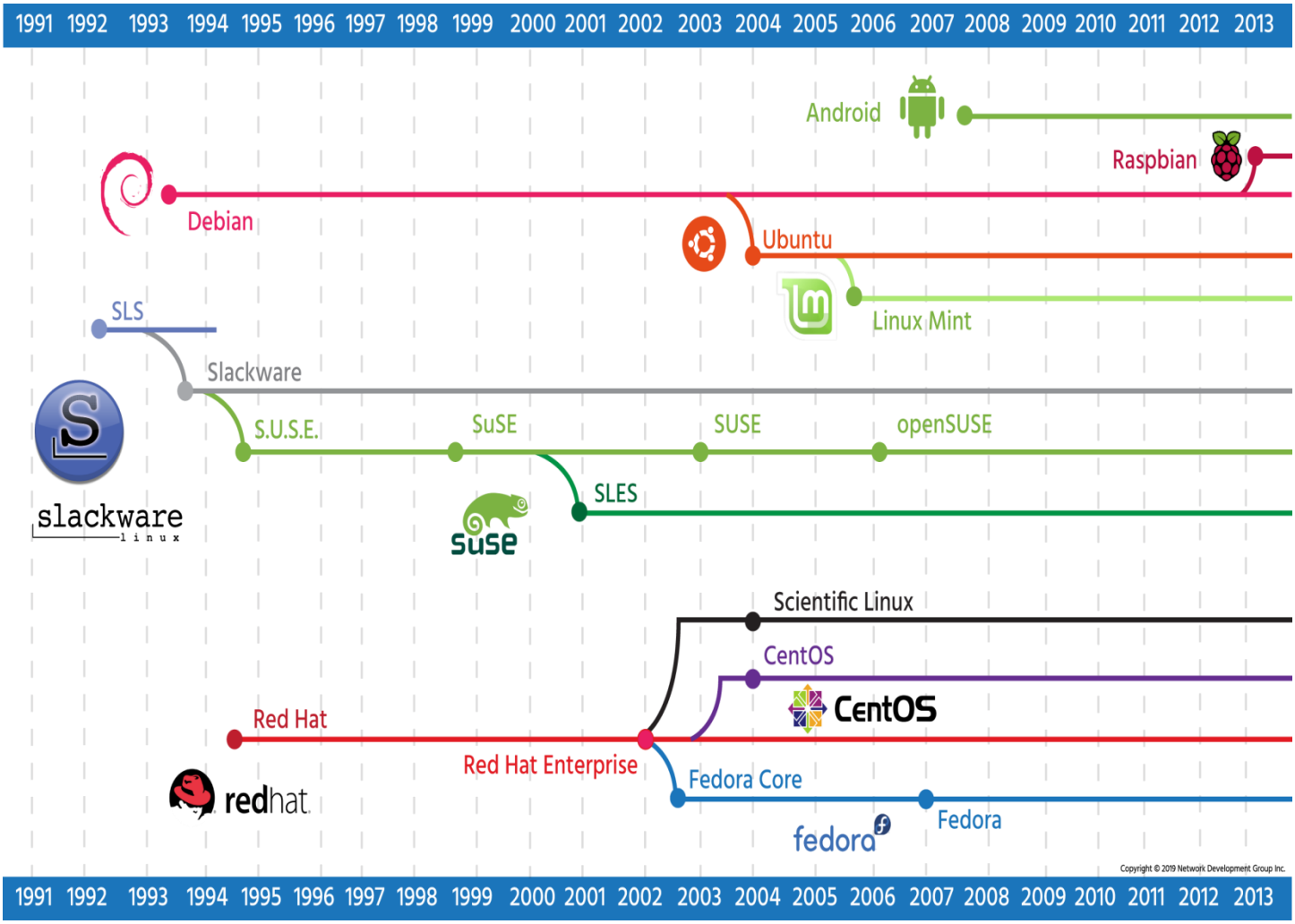
*** Raspbian** is a specialized Linux distribution optimized to run on **Raspberry Pi** hardware. This combination has seen significant use in training for programmers and hardware designers at all levels. Its low cost and ease of use have made it a favorite of educators worldwide, and many add-on devices are available to extend its capabilities into the physical world.

- Android:

* **Android** 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.

* However, lacking the traditional packages that are often distributed with Linux , 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.



Open Source Licensing:

- With Microsoft Windows, the Microsoft Corporation owns the intellectual property. The **End User License Agreement (EULA)**, is a custom legal document that you must click through to install the software. Microsoft keeps the source code and distributes only binary copies through authorized channels. 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 says that the source code must be made available to anyone who asks and that anyone is allowed to make changes. 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. According to **Free and Open Source Software (FOSS)**, anyone is allowed to view the source code and redistribute it.

- There are two groups can be considered the most influential forces in the world of open source:

- **Free Software Foundation**

* **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.

* 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 free and requiring all modified and extended versions of the program to be free as well".

* 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.

- **Open Source Initiative**

* The **Open Source Initiative (OSI)** was founded in 1998 by **Bruce Perens & 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.

* 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."

* 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.

* 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)**. 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)**.

- **Creative Commons**

* 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.

Linux Server Applications:

- **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 **Hyper Text 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.

- **NGINX** is a web server 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.

- **Private Cloud Servers**

- The **ownCloud** project was launched in 2010 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 and has been growing steadily since then. It is provided under a GNU AGPLv3 and aims for “an open, transparent development process.”

- **Database Servers**

- Database server applications form the backbone of most online services. Dynamic web applications pull data from and write data to these applications. When data is entered into the form, it is written to a database application such as MariaDB.

- **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.

- 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.

- **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.

* **POP/IMAP Server**: They 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.

- **File Sharing Servers**

- **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. 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 just like a regular disk, making file access transparent to other applications.

- The **Lightweight Directory Access Protocol (LDAP)** is one common directory system which also powers Microsoft’s Active Directory. **OpenLDAP** is the dominant program used in Linux infrastructure.

- The **Internet Software Consortium** maintains the most popular DNS server, simply called **bind** after the name of the process that runs the service. It also maintains the **ISC DHCP** server, which is the most common **open source DHCP** server.

Linux Desktop Applications:

- **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.

- **Multiple-media**

- 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.

- **Productivity**

- 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** application suite. Both offer a full office suite, including tools that strive for compatibility with Microsoft Office in both features and file formats. LibreOffice can also work with other file formats, such as Microsoft Office or **(PDF)** files.

Linux Console Tools:

- **Shells**

- 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.

- The two main Linux shells 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)**. Other shells may offer features that increase productivity in specific use cases.

- **Text Editors**

- The two main text editors 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.

- 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.

- 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. 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.

Linux 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.

- Modern distributions use ***packages***, which are compressed files that bundle up an application and its *dependencies* (or required files), 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:

- **Debian Package Management**

- The Debian distribution, and its derivatives such as Ubuntu and Mint, uses 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**.

- **RPM Package Management**

- The **Linux Standards Base**, which is a **Linux Foundation** project, is designed to specify 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.

- 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.

- 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.

Basic Command Syntax:

- 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.

- Most commands follow a simple pattern of syntax:

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

* **Argument:** can be used to specify something for the command to act upon. The `ls` command can be given the name of a directory as an argument, and it will list the contents of that directory. In the next example, the `Documents` directory will be used as an argument:

```
sysadmin@localhost:~$ ls Documents
School            alpha-second.txt  food.txt     linux.txt     os.csv
Work              alpha-third.txt   hello.sh     longfile.txt  people.csv
```

* **Options:** can be used to alter the behavior of a command. On the previous page, the `ls` command was used to list the contents of a directory. In the following example, the `-l` option is provided to the `ls` command, which results in a "long display" output, meaning the output gives more information about each of the files listed:

```
sysadmin@localhost:~$ ls -l
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Desktop
drwx------ 4 sysadmin sysadmin 4096 Dec 20  2017 Documents
drwx------ 2 sysadmin sysadmin 4096 Dec 20  2017 Downloads
```

- Typically *options* alter the behavior of the command and arguments are items or values for the command to act upon. Although there are some commands in Linux that aren't entirely consistent with this syntax, most commands use this syntax or something similar.

- Single-letter options are preceded by a single dash `-` character, like the `-h` option. Full-word options are preceded by two dash `--` characters.

Command Types:

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

* **Internal Commands**: Also called ***built-in commands***, are built into the shell itself.

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

```
sysadmin@localhost:~$ type cd
cd is a shell builtin
```

* **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.

- Instead, use the `which` command to display the full path to the command in question:

```
which command
sysadmin@localhost:~$ which ls
/bin/ls
```

- External commands can also be executed by typing the complete path to the 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
```

* **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. 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.

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

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

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

```
alias name=command
```

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

- 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
```

* **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.

Printing Working Directory:

- In order to discover where you are currently located within the filesystem, the `pwd` command can be used. The `pwd` command prints the working directory, your current location within the filesystem:

```
pwd [OPTIONS]
```

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

System & Hardware Information:

- **System Information**

- To displays information about the current system to see the name of the kernel you are using.

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

- To display the network node hostname, also found in the prompt.

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

- To displays the user name of the current user.

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

➢ Hardware Information

- 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
CPU(s):                4
Core(s) per socket:    4
```

- 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.

```
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
cpu MHz         : 2394.008
```

- 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
```

- 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.

```
sysadmin@localhost:~$ lspci
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 (
```

- Use the `lspci` command with the `-k` option to show devices along with the kernel driver and modules used.

- To display the devices connected to the system via USB, execute the `lsusb` command:

```
sysadmin@localhost:~$ lsusb
Bus 001 Device 002: ID 0e0f:000b VMware, Inc.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc.
```

- Modern distributions often mount the removable 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.

- For hardware to function, the Linux kernel usually loads a driver or module. Use the `lsmod` command to view the currently loaded modules:

```
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
```

- 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.

➢ Hard Drives

- 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 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.

- 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 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**

    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:

```
root@localhost:~$ fdisk -l /dev/sda
Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors
   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 /
```

- You can find the partition tables for the specified device in **`/proc/partitions`** file.

Changing Directories:

- To navigate the filesystem structure, use the `cd` (change directory) command to change directories.

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

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

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

- To use a relative path to change up one level from the current directory and then down into the `dict` directory:

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

- If you think of the filesystem as a map, paths are the step-by-step directions; they can be used to indicate the location of any file within the filesystem. There are two types of paths: absolute and relative. Absolute paths start at the root of the filesystem, relative paths start from your current location.

* **Absolute path:** allows you to specify the exact location of a directory. It always starts at the root directory, therefore it always begins with the **/** character. The path to the home directory **/home/sysadmin** is an absolute path.

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

* **Relative path:** gives directions to a file relative to your current location in the filesystem. Relative paths do not start with the **/** character, they start with the name of a directory.

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

Shortcuts:

- The (`..`) 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 (`.`) Character: always represents your current directory. For the `cd` this shortcut is not very useful, but it will come in handy for commands covered in subsequent sections.

- The (`~`) Character: The home directory of the current user is represented by the `~` character. As stated above, you always begin as the **sysadmin** user, whose home is located at **/home/sysadmin**. To return to your home directory at any time execute the following command:

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

Listing Files:

- The `ls` command is used to list the contents of a directory:

```
ls [OPTIONS] [FILE]
```

- By default, when the `ls` command is used with no options or arguments, it will list the files in the current directory.

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

- To view the contents of the root directory, use the `ls /` command

```
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
```

- To learn the details about a file, such as the type of file, the permissions, ownerships or the timestamp, perform a long listing using the `-l` option to the `ls` command.

```
sysadmin@localhost:~$ ls -l /var/log/
-rw-r--r-- 1 root   root  18047 Dec 20  2017 alternatives.log
drwxr-x--- 2 root   adm    4096 Dec 20  2017 apache2
```

- To present the file size in a more human-readable size, like megabytes or gigabytes, add the `-h` option to the `ls` command:

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

- For more detailed modification time information you can use the `--full-time` option to display the complete timestamp (including hours, minutes, seconds).

```
sysadmin@localhost:~$ ls --full-time /etc/ssh
-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_ed2519_key.pub
```

- 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
```

- 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
```

- 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
```

- The `ls` command uses color to distinguish by file type. For example, regular file may be displayed in **Black** or white, directories may be displayed in blue, executable files may be displayed in **Green**, and symbolic links may be displayed in Cyan. The `ls` seems to perform this coloring automatically because there is an alias for the ls command, so it runs with the `--color` option.

* To avoid using the alias, place a backslash character `\` in front of your command:

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

- You can use file ***globbing*** (wildcards) to limit which files or directories you see.

* The `*` character can match "zero or more of any characters" in a filename.

```
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
```

* The `?` character can be used to match exactly 1 character in a file name.

```
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
```

* The `[ ]` can specify a single character to match from a set of characters.

```
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
```

- Each line corresponds to a file contained within the directory. The information can be broken down into fields separated by spaces. The fields are as follows:

* **File Type:** The first field actually contains ten characters, where the first character indicates the type of file and the next nine specify permissions. The file types are:

```
- rw-r--r-- 1 root   root  18047 Dec 20  2017 alternatives.log
d rwxr-x--- 2 root   adm    4096 Dec 20  2017 apache2
```

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.

* **Permissions:** indicate how certain users can access a file. Keep reading to learn more about permissions.

```
d rwxr-xr-x 2 root   root   4096 Apr 11  2014 upstart
```

* **Hard Link Count:** This number indicates how many hard links point to this file.

```
-rw-r----- 1 syslog adm    1346 Oct  2 22:17 auth.log
```

In the case of symbolic links, a file that points to another file, the link name will be displayed along with an arrow and the pathname of the original file.

```
lrwxrwxrwx. 1 root root 22 Nov 6 2012 /etc/grub.conf -> ../boot/grub/grub.conf
```

* **User Owner:** User syslog owns this file. Every time a file is created, the ownership is automatically assigned to the user who created it.

```
-rw-r----- 1 syslog adm     106 Oct  2 19:57 kern.log
```

* **Group Owner:** Indicates which group owns this file

```
-rw-rw-r-- 1 root    utmp  292584 Oct  2 19:57 lastlog
```

* **File Size:** Directories and larger files may be shown in kilobytes since displaying their size in bytes would present a very large number. Therefore, in the case of a directory, it might actually be a multiple of the block size used for the file system. Block size is the size of a series of data stored in the filesystem.

```
-rw-r----- 1 syslog adm    19573 Oct  2 22:57 syslog
```

* **Timestamp:** This indicates the time that the file's contents were last modified.

```
drwxr-xr-x 2 root   root    4096 Dec  7  2017 fsck
```

* **Filename:** The final field contains the name of the file or directory.

```
-rw-r--r-- 1 root   root  47816 Dec  7  2017 bootstrap.log
```

Sorting Files:

- By default the output of the `ls` command is sorted alphabetically by filename. It can sort by other methods as well:

* The `-t` option will sort the files by **timestamp**.

```
sysadmin@localhost:~$ ls -lt /var/log
total 844
-rw-r----- 1 syslog adm   19573 Oct  2 22:57 syslog
-rw-r----- 1 syslog adm     547 Oct  2 22:17 cron.log
-rw-rw-r-- 1 root   utmp 292584 Oct  2 19:57 lastlog
```

* The `-S` option will sort the files by file size.

```
sysadmin@localhost:~$ ls -l -S /var/log
total 844
-rw-r--r-- 1 root   root 325238 Dec 20  2017 dpkg.log
-rw-rw-r-- 1 root   utmp 292584 Oct  2 19:57 lastlog
-rw-r----- 1 root   adm   85083 Dec 20  2017 dmesg
-rw-r--r-- 1 root   root  47816 Dec  7  2017 bootstrap.log
```

* The `-r` option will reverse the order of any type of sort.

```
sysadmin@localhost:~$ ls -lSr /var/log
-rw-rw---- 1 root   utmp      0 Dec  7  2017 btmp
-rw-r----- 1 syslog adm     106 Oct  2 19:57 kern.log
-rw-rw-r-- 1 root   utmp    384 Oct  2 19:57 wtmp
-rw-r----- 1 syslog adm     654 Oct  2 23:17 cron.log
```

The numbers in file size field switch from descending to ascending.

Used alone the `-r` option with list the files in reverse alphabetical order:

```
sysadmin@localhost:~$ ls -r /var/log
wtmp     lastlog   faillog   cron.log       auth.log  alternatives.log
upstart  kern.log  dpkg.log  btmp           apt
syslog   fsck      dmesg     bootstrap.log  apache2
```

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.

- 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:~$ 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.

```
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.

```
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.

```
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 04:32:38 UTC 2018
```

System and User Security

- **Administrative Access**

- 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.

- The (`su`) Command: allows you to temporarily act as a different user. It does this by creating a new shell. The shell is simply a text input console that lets you type in commands. By default, if a user account is not specified, the `su` command will open a new shell as the root user, which provides administrative privileges.

```
su OPTIONS USERNAME
su - OR su -l OR su --login
sysadmin@localhost:~$ su  -
```

```
Password:
root@localhost:~#
```

- 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.

- After using the shell started by the `su` command to perform the necessary administrative tasks, return to your original shell by using the `exit` or `logout` command.

```
root@localhost:~# exit
logout
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(
```

- The (`sudo`) Command: allows a user to execute a command as another user without creating a new shell. Instead, to execute a command with administrative privileges, use it as an argument to the `sudo` command. Like the `su` command, the `sudo` command assumes by default the **`root`** user account should be used to execute commands.

```
sudo [OPTIONS] COMMAND
```

- 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.

- Once the command has completed, notice the prompt has *not* changed, you are still logged in as **`sysadmin`**. The `sudo` command only provides administrative access for the execution of the specified command. This is an advantage as it reduces the risk that a user accidentally executes a command as root.

➢ 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 that 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
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:

* **Name:** 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.

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

*** Password Placeholder:** 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.

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

*** User ID:** 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.

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

*** Primary Group ID:** 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.

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

*** Comment:** This field can contain any information about the user, including their real name or other useful information

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

*** Home Directory:** This field defines the location of the user's home directory. For regular users, this would typically be **/home/*username***. The root user usually has a different place for the home directory, the **/root** directory.

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

*** Shell:** 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.

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

- **Passwords**

- 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):

```
root@localhost:~# cat /etc/shadow
syslog:*:16874:0:99999:7:::
operator:!:16874:0:99999:7:::
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2mFSIfnc1aU2c
```

- Each line is separated into fields by colon characters. The following describes each of the fields in detail from left to right:

*** Username:** This field contains the username of the account, which matches the account name in the **/etc/passwd** file.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

*** Password:** This 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.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- While regular users have encrypted passwords in this field, system accounts have an asterisk ***** character in this field.

*** Last Change:** 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).

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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:** This field indicates the *minimum* number of days between password changes.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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:** 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.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050:
```

- 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:** If the *maximum* field is set, the *warn* field indicates the number of days before password expiry that the system warns the user.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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:** 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.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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:** 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.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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:** Currently not used, this field is reserved for future use.

```
sysadmin:$6$c75ekQWF$.GpiZpFnIXLzkALjDpZXmjxZcIll14OvL2:16874:5:30:7:60:15050::
```

- 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.

- For example, the following command would retrieve account information for the **sysadmin** user from the **/etc/passwd** file:

```
sysadmin@localhost:~$ getent passwd sysadmin
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
```

➢ 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:::
```

➢ 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 16 groups, but the recent Linux kernels support users with over 65000 group memberships.

- The **/etc/passwd** file defines the primary group membership for a user. Supplemental group membership (or secondary group membership) and the groups themselves are defined in the **/etc/group** file.

- The `/etc/group` file is another colon-delimited file. The following describes the fields in more detail, using a line that describes a typical group account:

*** Group Name:** 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.

```
mail:x:12:mail,postfix
```

*** Password Placeholder:** 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.

```
mail:x:12:mail,postfix
```

*** GID:** Each group is associated with a unique *group ID (GID)* which is placed in this field.

```
mail:x:12:mail,postfix
```

*** User List:** 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.

```
mail:x:12:mail,postfix
```

- 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.

```
sysadmin@localhost:~$ grep mail /etc/group
mail:x:12:mail,postfix
sysadmin@localhost:~$ getent group mail
mail:x:12:mail,postfix
```

➢ Viewing User Information

- The `id` command is used to print user and group information for a specified user.

```
id [options] username
```

- 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 using the `-G` option. This will print all the groups that a user belongs to, both primary and secondary.

```
sysadmin@localhost:~$ id -G
1001 4 27
```

- The output of the previous example aligns with the contents of the **/etc/group** file, as a search for **sysadmin** reveals:

```
sysadmin@localhost:~$ cat /etc/group | grep sysadmin
adm:x:4:syslog,sysadmin
sudo:x:27:sysadmin
sysadmin:x:1001:
```

➢ 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. The `who` command reads from the **/var/log/utmp** file which logs current users.

```
sysadmin@localhost:~$ who
root           tty2         2013-10-11 10:00
sysadmin       tty1         2013-10-11 09:58 (:0)
sysadmin       pts/0        2013-10-11 09:59 (:0.0)
sysadmin       pts/1        2013-10-11 10:00 (example.com)
```

- The following describes the output of the `who` command:

* **Username:** 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".

```
root             tty2         2013-10-11 10:00
```

* **Terminal:** 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.

```
root             tty2          2013-10-11 10:00
sysadmin         pts/0      2013-10-11 09:59 (:0.0)
```

* **Date:** This column indicates when the user logged in.

```
root             tty2         2013-10-11 10:00
```

* **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:

- 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 run level:

```
sysadmin@localhost:~$ who -b -r
         system boot   2013-10-11 09:54
         run-level 5    2013-10-11 09:54
```

- 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.

```
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        pts/1    example.com  10:00     0.00s     0.03s   0.01s   w
```

- The first line of output from the `w` command is identical to that of the `uptime` command. It shows the current time, how long the system has been running, the total number of users currently logged on and the load on the system averaged over the last 1, 5 and 15 minute time periods.
- *Load average* is CPU utilization where, for a single-core system, a value of 1 would mean 100% CPU usage during that period of time. For a dual-core system, it would mean 50% CPU usage, and for a quad-core system, it would mean 25% CPU usage.

- The following describes the rest of the output of the `w` command:

Column	Example	Description
USER	root	The name of the user who is logged in.
TTY	tty2	Which terminal window the user is working in.
FROM	example.com	Where the user logged in from.
LOGIN@	10:00	When the user logged in.
IDLE	43:44	How long the user has been idle since the last command was executed.
JCPU	0.01s	The total cpu time used by all processes run since login.
PCPU	0.01s	The total cpu time for the current process.
WHAT	-bash	The current process that the user is running.

➢ 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. The **`/var/log/wtmp`** file keeps a log of all users who have logged in and out the system.

```
sysadmin@localhost:~$ last -w
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`**.

Users and Groups Management

➢ Groups

- 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 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.

- 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.

```
root@localhost:~# grep root /etc/group
root:x:0:
root@localhost:~# getent group root
root:x:0:
```

- 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.

```
root@localhost:~# groupadd development
root@localhost:~# grep development /etc/group
development:x:1006:
```

* 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. 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:
```

* Following these guidelines for group names can help to select a group name that is 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.

- 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 won't cause any problems with accessing files, since the files are owned by GIDs, not group names.

```
root@localhost:~# groupmod -n clerks sales
root@localhost:~# groupmod -g 10003 clerks
```

* 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.

* 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
```

- 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
```

- To delete group even if it is the primary group of a user, use `-f` option:

```
root@localhost:~# groupdel -f sysadmin
```

➢ Users

- User account information is stored in the **/etc/passwd** file and user 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.

- User Configuration Files (1):

* 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:

- **Group:** 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.

```
GROUP=100
```

This setting affects the *primary group ID* field of the **/etc/passwd** file highlighted below:

```
bob:x:600:600:bob:/home/bob:/bin/bash
```

The `-g` option to the `useradd` command allows you to use a different primary group than the default when creating a new user account.

- **Home:** The **`/home`** directory is the *default base directory* under which the user's new home directory is created.

```
HOME=/home
```

This setting affects the home directory field of the **`/etc/passwd`** file highlighted below:

```
bob:x:600:600:bob:/home/bob:/bin/bash
```

The `-b` option to the `useradd` command allows you to use a different base directory group than the default when creating a new user account.

- **Inactive:** 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.

```
INACTIVE=-1
```

This setting affects the *inactive* field of the **`/etc/shadow`** file highlighted below:

```
bob:pw:15020:5:30:7:60:15050:
```

The `-f` option to the `useradd` command allows you to use a different **`INACTIVE`** value than the default when creating a new user account.

- **Expire:** 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.

```
EXPIRE=
```

This setting affects the *expire* field of the **`/etc/shadow`** file highlighted below:

```
bob:pw:15020:5:30:7:60:15050:
```

The `-e` option to the `useradd` command allows you to use a different **`EXPIRE`** value than the default when creating a new user account.

- **Shell:** The **`SHELL`** setting indicates the default shell for a user when they log in to the system.

```
SHELL=/bin/bash
```

This setting affects the *shell* field of the **`/etc/passwd`** file highlighted below:

```
bob:x:600:600:bob:/home/bob:/bin/bash
```

The `-s` option to the `useradd` command allows you to use a different login shell than the default when creating a new user account.

- **Skeleton Directory:** The **`SKEL`** value determines which *skeleton directory* has its contents copied into the new user's home directory.

```
SKEL=/etc/skel
```

The **`/etc/skel`** directory contains files and folders that will be copied in the new user's home directory (login directory), when that user is created with `useradd` or other commands. It's also used to initiate home directory when a user is first created.

This setting provides administrators with an easy way to populate a new user account with key configuration files and also allows the administrator to create new users having the same files and folders in their home directories.

It is not recommended to change the permission of **`SKEL`** directory or its contents because it may break some of the program as there are some profiles that needs the permission of read and trying to give it permission of execute will cause some programs/profiles to work unexpectedly.

When a user is deleted, the data inside the **`/etc/skel`** directory remains unchanged.

The `-k` option to the `useradd` command allows you to use a different `SKEL` directory than the default when creating a new user account.

- **Create Mail Spool:** A *mail spool* is a file where incoming email is placed.

```
CREATE_MAIL_SPOOL=yes
```

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.

* 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.

```
root@localhost:~# useradd -D -f 30
root@localhost:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=30
```

- User Configuration Files (2):

* 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
```

* The following describes each of these values:

- **Mail Directory:** The directory in which the user's mail spool file is created.

```
MAIL_DIR /var/mail/spool
```

- **Password Maximum Days:** This setting determines the maximum number of days that a user can continue to use the same password. Since it defaults to **99999** days it effectively means users never have to change their password.

```
PASS_MAX_DAYS    99999
```

  This setting affects the default setting of the **/etc/shadow** file highlighted below:

```
bob:pw:15020:5:30:7:60:15050:
```

- **Password Minimum Days:** 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.

```
PASS_MIN_DAYS    0
```

  If the **PASS_MIN_DAYS** value was set to 3 days, then after setting a new password, the user would have to wait three days before they could change it again.

  This setting affects the default setting of the **/etc/shadow** file highlighted below:

```
bob:pw:15020:3:30:7:60:15050:
```

- **Password Minimum Length:** This indicates the minimum number of characters that a password must contain.

```
PASS_MIN_LEN     5
```

- **Password Warning:** 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.

```
PASS_WARN_AGE    7
```

  This setting affects the default setting of the **/etc/shadow** file highlighted below:

```
bob:pw:15020:3:30:7:60:15050:
```

- **UID Minimum:** The **UID_MIN** determines the first UID that is assigned to an ordinary user. Any UID less than this value would either be for a system account or the root account.

```
UID_MIN          500
```

- **UID Maximum:** A UID technically could have a value of over 4 billion. For maximum compatibility, it's recommended to leave it at its default value of 60000.

```
UID_MAX          60000
```

- **GID Minimum:** The **GID_MIN** determines the first GID that is assigned to an ordinary group. Any group with a GID less than this value would either be a system group or the root group.

```
GID_MIN          500
```

- **GID Maximum:** A GID, like a UID, could have a value of over 4 billion. Whatever value you use for your **UID_MAX**, should be used for **GID_MAX** to support UPG.

```
GID_MAX                 60000
```

- **Home Directory:** The value of this determines whether or not a new directory is created for the user when their account is created.

```
CREATE_HOME      yes
```

- **Umask:** **UMASK** works at the time the user home directory is being created; it determines what default permissions are placed on this directory.

```
UMASK           077
```

  Using the default value of **077** for **UMASK** means that only the user owner has any kind of permission to access their directory.

- **UPG:** In distributions that feature a private group for each user, 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**.

```
USERGROUPS_ENAB yes
```

- **Encryption:** 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.

```
ENCRYPT_METHOD SHA512
```

- **Encryption (Deprecated):** 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.

```
MD5_CRYPT_ENAB no
```

- User Account Considerations:

* Creating a user account for use with a Linux system may require 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.

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 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.

- ***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
```

- 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
```

* 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
```

- Setting & Managing 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. If setting the password completed successfully, then the **`/etc/shadow`** file will be updated with the user's new password.

```
root@localhost:~# passwd Jane
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

* 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.

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	Description
-l	List the account aging information
-d *LAST_DAY*	Set the date of the last password change to `LAST_DAY`
-E *EXPIRE_DATE*	Set account to expire on `EXPIRE_DATE`
-h	Show the help for the `chage` command
-I *INACTIVE*	Set account to permit login for `INACTIVE` days after password expires
-m *MIN_DAYS*	Set the minimum number of days before the password can be changed to `MIN_DAYS`
-M *MAX_DAYS*	Set the maximum number of days before a password should be changed to `MAX_DAYS`
-W *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
```

- 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.

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	Description
-c	Sets the value of the GECOS or comment field to `COMMENT`.
-d *`HOME_DIR`*	Sets `HOME_DIR` as a new home directory for the user.
-e *`EXPIRE_DATE`*	Set account expiration date to `EXPIRE_DATE`.
-f *`INACTIVE`*	Set account to permit login for `INACTIVE` days after password expires.
-g *`GROUP`*	Set `GROUP` as the primary group.
-G *`GROUPS`*	Set supplementary groups to a list specified in `GROUPS`.
-a	Append the user's supplemental groups with those specified by the `-G` option.
-h	Show the help for the `usermod` command.
-l *`NEW_LOGIN`*	Change the user's login name.
-L	Lock the user account.
-s *`SHELL`*	Specify the login shell for the account.
-u *`NEW_UID`*	Specify the user's UID to be `NEW_UID`.
-U	Unlock the user account.

* 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
```

- Deleting a User:

* The `userdel` command is used to delete users. When you delete a user account, you also need to decide whether to delete the user's home directory. Also, unless you've made backup copies of the data, once you've executed the command to delete the user and their files, there is no reversing the action.

- To delete the user **`jane`** without deleting the user's home directory **`/home/jane`**, execute:

```
root@localhost:~# userdel jane
```

* Beware that deleting a user without deleting their home directory means that the user's home directory files will be orphaned and these files will be owned solely by their former UID and GID.

- To delete the user, home directory, and mail spool as well, use the `-r` option:

```
root@localhost:~# userdel -r jane
```

* The above command will only delete the user's files in their home directory and their mail spool. If the user owns other files outside of their home directory, then those files will continue to exist as orphaned files.

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.

- Also, 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 done by associating the group ownership with the *GID* for that group.

- Since ownership is determined by the UID and GID associated with a file, changing the UID of a 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.

➢ 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.

- 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
```

- For example, if the **sysadmin** user was planning on having a file owned by the group **it**, but that wasn't the user's primary group:

```
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1001(sysadmin) groups=1001(sysadmin),4(adm),27(sudo),1005(it)
sysadmin@localhost:~$ newgrp it
sysadmin@localhost:~$ id
uid=1001(sysadmin) gid=1005(it) groups=1005(research),4(adm),27(sudo),1001(sysadmin)
```

- 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:

- 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.

```
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)
```

- Administrative privileges are required to change the primary group of the user permanently. The root user would execute the following command:

```
sysadmin@localhost:~$ sudo usermod -g sysadmin it
```

➢ Changing Group Ownership

- To change the group owner of an existing file the `chgrp` command can be used.

```
sysadmin@localhost:~$ chgrp research sample
sysadmin@localhost:~$ ls -l sample
-rw-rw-r--. 1 sysadmin research 0 Oct 23 22:12 sample
```

- 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.

- 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
```

- 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
```

➢ Changing User Ownership

- The `chown` command allows the root user to change the user ownership of files and directories. A regular user cannot use this command to change the user owner of a file, even to give the ownership of one of their own files to another user. However, the `chown` command also permits changing group ownership, which can be accomplished by either root or the owner of the file.

- There are three different ways the `chown` command can be executed:

* The first method is used to change just the user owner of the file.

```
chown user /path/to/file
```

- For example, if the root user wanted to change the user ownership of the **`filetest1`** file to the user **`jane`**, then the following command could be executed:

```
root@localhost:~# chown jane /tmp/filetest1
root@localhost:~# ls -l /tmp/filetest1
-rw-rw-r-- 1 jane sysadmin 0 Dec 19 18:44 /tmp/filetest1
```

* The second method is to change both the user and the group; this also requires root privileges. To accomplish this, you separate the user and group by either a colon or a period character. For example:

```
chown user:group /path/to/file
chown user.group /path/to/file
root@localhost:~# chown jane:users /tmp/filetest2
```

```
root@localhost:~# ls -l /tmp/filetest2
-rw-r--r-- 1 jane users 0 Dec 19 18:53 /tmp/filetest2
```

* If a user doesn't have root privileges, they can use the third method to change the group owner of a file just like the `chgrp` command. To use `chown` only to change the group ownership of the file, use a colon or a period as a prefix to the group name:

```
chown :group /path/to/file
chown .group /path/to/file
```

```
jane@localhost:~$ chown .users /tmp/filetest1
jane@localhost:~$ ls -l /tmp/filetest1
-rw-rw-r-- 1 jane users 0 Dec 19 18:44 /tmp/filetest1
```

Permissions:

- Permissions determine the ways different users can interact with a file or directory. When listing a file with the `ls -l` command, the output includes permission information. For the example we will use a script called `hello.sh` located in the `Documents` directory:

```
sysadmin@localhost:~/Documents$ ls -l hello.sh
-rw-r--r-- 1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

- The permissions are broken into three sets of three characters:

* **User Owner:** The first set is for the user who owns the file. If your current account is the user owner of the file, then the first set of the three permissions will apply and the other permissions have no effect.

```
-rw-r--r-- 1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

The user who owns the file, and who these permissions apply to, can be determined by the ***user owner*** field:

```
-rw-r--r-- 1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

* **Group Owner:** The second set is for the group that owns the file. If your current account is not the user owner of the file and you are a member of the group that owns the file, then the group permissions will apply and the other permissions have no effect.

```
-rw-r--r-- 1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

The group for this file can be determined by the *group owner* field:

```
-rw-r--r-- 1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

* **Other:** The last set is for everyone else, anyone who that first two sets of permissions do not apply to. If you are not the user who owns the file or a member of the group that owns the file, the third set of permissions applies to you.

```
-rw-r--r--  1 sysadmin sysadmin 647 Dec 20  2017 hello.sh
```

- There are three different permissions that can be placed on a file or directory: read, write, and execute. The manner in which these permissions apply differs for files and directories, as shown in the chart below:

Permission	Effects on File	Effects on Directory
read (r)	▪ Allows for file contents to be read or copied.	▪ Without execute permission on the directory, allows for a non-detailed listing of files. With execute permission, `ls -l` can provide a detailed listing.
write (w)	▪ Allows for contents to be modified or overwritten. Allows for files to be added or removed from a directory. ▪ This permission requires read permission on the file to work correctly.	▪ Files can be added to or removed from the directory. ▪ For this permission to work, the directory must also have execute permission.
execute (x)	▪ Allows for a file to be run as a process, although script files require read permission, as well.	▪ Allows a user to change to the directory if parent directories have execute permission as well. ▪ 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.

➢ Understanding Permissions

- Scenario #1 - Directory Access:

*** Question:** Based on the following information, what access would the user **bob** have on the file **abc.txt**?

```
drwxr-xr-x. 17 root root 4096 23:38 /
drwxr-xr--. 10 root root 128  03:38 /data
-rwxr-xr--.  1 bob  bob  100  21:08 /data/abc.txt
```

*** Answer:** None.

*** Explanation:** Initially it would appear that the user `bob` can view the contents of the **abc.txt** file as well as copy the file, modify its contents and run it like a program. This erroneous conclusion would be the result of looking solely at the file's permissions (**rwx** for the user **bob** in this case).

However, to do anything with the file, the user must first "get into" the `/data` directory. The permissions for **bob** for the **/data** directory are the permissions for "others" (**r--**), which means **bob** can't even use the `cd` command to get into the directory. If the execute permission (**--x**) were set for the directory, then the user **bob** would be able to "get into" the directory, meaning the permissions of the file itself would apply.

*** Lesson Learned:** The permissions of all parent directories must be considered before considering the permissions on a specific file.

- 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
```

*** 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.

- 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
```

*** 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.

- 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
```

* **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.

- 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`?

```
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.

- 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`?

```
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.

➢ Changing File Permissions

- The `chmod` command is used to change the permissions of a file or directory. Only the root user or the user who owns the file is able to change the permissions of a file

- There are two techniques for changing permissions with the `chmod` command:

- **Symbolic method:** is good for changing one set of permissions at a time.
- **Octal method:** or numeric requires knowledge of the octal value of each of the permissions and requires all three sets of permissions (user, group, other) to be specified every time.

- The 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.

```
chmod [<SET><ACTION><PERMISSIONS>]... FILE
```

- To use the symbolic method of `chmod` first indicate which set of permissions is being changed:

```
chmod [ <SET> <ACTION><PERMISSIONS>]... FILE
```

Symbol	Meaning
u	User: The user who owns the file.
g	Group: The group who owns the file.
o	Others: Anyone other than the user owner or member of the group owner.
a	All: Refers to the user, group and others.

- Next, specify an action symbol:

```
chmod [<SET> <ACTION> <PERMISSIONS>]... FILE
```

Symbol	Meaning
+	Add the permission, if necessary
=	Specify the exact permission
-	Remove the permission, if necessary

After an action symbol, specify one or more permissions to be acted upon.

```
chmod [<SET><ACTION><PERMISSIONS>]... FILE
```

Symbol	Meaning
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
```

* 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
```

* 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
```

- 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:

Number	Permission
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:

Number	Permisson
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
```

- **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 Type	Default Value	permission
File	666	rw-rw-rw-
Directory	777	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
```

- 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	-27
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:

- 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.

Special Permissions:

➢ **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.
- When `passwd` 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.
- 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
```

- 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
```

➢ 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.

- 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 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.

- For example of the setgid permission on an executable file is the **`/usr/bin/wall`** command.

```
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.

- 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.

* 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
```

* 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. 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, it is not really in effect because the group lacks the execute permission to use it.

* 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>
```

* Use Case: 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--**).

➢ Sticky Bit

- The sticky bit permission is used to prevent other users from deleting files that they do not own in a shared directory.

- 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.

- Use the following syntax to set the sticky bit permission symbolically:

```
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>
```

Viewing Files:

- There are a few Linux commands available to view the content of files. The `cat` command, which stands for “concatenate”, is often used to quickly view the contents of small files.

- To view the contents of a file using the `cat` command, simply type the command and use the name of the file you wish to view as the argument:

```
cat [OPTIONS] [FILE]
```

```
sysadmin@localhost:~/Documents$ cat animals.txt
```

```
1 retriever
2 badger
3 bat
4 wolf
```

- 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.

- <u>Pager Movement Commands</u>: 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:

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

  ---------------------------------------------------------------------------

                           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).
HELP -- Press RETURN for more, or q when done
```

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

- <u>Pager Searching Commands</u>: 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
```

- For example, in the following graphic the expression "frog" was searched for in the `words` file:

```
bullfrog
bullfrog's
bullfrogs
bullheaded
```

* 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.

- Another way to view the content of files is by using the `head` and `tail` commands. These commands are used to view a select number of lines from the top or bottom of a file:

```
head [OPTIONS] [FILE]
tail [OPTIONS] [FILE]
```

* To filter the output and view lines from the top of the `alpha.txt` file, use the `head` command:

```
sysadmin@localhost:~/Documents$ head alpha.txt
```

* Then, to view lines at the bottom of the `alpha.txt` file, you use the `tail` command:

```
sysadmin@localhost:~/Documents$ tail alpha.txt
```

- The `-n` option with the `head` and `tail` commands can be used to specify the amount of lines to display. To use the `-n` option, specify the amount of lines from the file you want to display after the option and use the filename as an argument:

```
head -n number_of_lines filename
```

```
sysadmin@localhost:~/Documents$ head -n 5 alpha.txt
A is for Apple
B is for Bear
C is for Cat
D is for Dog
E is for Elephant
sysadmin@localhost:~/Documents$ tail -n 5 alpha.txt
```

```
V is for Velvet
W is for Walrus
X is for Xenon
Y is for Yellow
Z is for Zebra
```

- 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.

- 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/sysadm
```

Command Line Pipes:

- The ***pipe*** `|` character can be used to send the output of one command to another. Typically, the output of one command becomes input for the next command. This tool can be powerful, especially when looking for specific data; *piping* is often used to refine the results of an initial command.

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

```
sysadmin@localhost:~$ ls /etc | head
X11
adduser.conf
alternatives
apparmor
apparmor.d
```

- Multiple pipes can be used 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.

```
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
```

Sorting File Contents:

- 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:~$ 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
```

- 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.
-k	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
```

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

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

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

Option	Function
-t,	Specifies the comma character as the field delimiter
-k2	Sort by field #2
-k1n	Numerically sort by field #1
-k3	Sort by field #3

Filtering File Sections:

- The `cut` command can extract columns of text from a file or standard input. It's used for working with delimited database files.

- 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
```

- 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.

- 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
```

Filtering 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
```

- 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
```

- 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
```

- 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
```

- 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
```

- 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:~/Documents$ grep -i the newhome.txt
There are three bathrooms.
**Beware** of the ghost in the bedroom.
```

- The `-w` option only returns lines which contain matches that form whole words. 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:

```
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.
```

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.

- 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
```

Creating Files & Directories:

- 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
```

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

```
sysadmin@localhost:~$ mkdir test
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates   test
```

Links:

- Consider a scenario where there is a file deeply buried in the file system called:

```
/usr/share/doc/superbigsoftwarepackage/data/2013/october/tenth/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, 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 **`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.

- Creating Hard Links:

* 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. It 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 ID 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. 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.

* 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.original
```

- 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
```

- Hard Links Vs Symbolic Links:

* 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
```

* 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
```

* 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
```

* 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 -> /bi
```

Copying Files (1):

```
cp [OPTIONS] SOURCE DESTINATION
```

- The `cp` command is used to copy files. Similar to the `mv` command, it requires at least two arguments: a source and a destination.

- To copy the `/etc/passwd` file to the current directory, use the following command:

```
sysadmin@localhost:~/Documents$ cp /etc/passwd .
sysadmin@localhost:~$ cp /etc/hosts ~
```

- 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
```

- 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'
```

- The second argument is the `.` & `~` character represent the current directory. The result of executing the previous command would create a copy of the contents of the **`/etc/passwd`** file in the **`Documents`** directory, since that is our current directory.

- To copy a file, it is necessary to have execute permission to access the directory where the file is located and the read permission for the file being copied. It is also necessary to have write and execute permission on the directory the file is being copied to.

- 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.

- With the `-i` ***interactive*** option, the `cp` command prompts the user before overwriting a file.

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

- 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.

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

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

- By default, the `cp` command will not copy directories. However, the *recursive* `-r` option has the `cp` command copy both files and directories.

Copying Files (2):

- The `dd` command is a utility for copying files or entire partitions at the bit level.

```
dd [OPTIONS] OPERAND
```

- This command has several useful features, including:

- It can be used to clone or delete (wipe) entire disks or partitions.
- It can be used to copy raw data to removable devices, such as USB drives and CDROMs.
- It can backup and restore the MBR (Master Boot Record).
- It can be used to create a file of a specific size that is filled with binary zeros, which can then be used as a swap file (virtual memory).

- The `dd` command creates a file named **`/tmp/swapex`** with 50 blocks of zeros that are one megabyte in size:

```
sysadmin@localhost:~/Documents$ cd ~
sysadmin@localhost:~$ dd if=/dev/zero of=/tmp/swapex bs=1M count=50
50+0 records in
50+0 records out
52428800 bytes (52 MB) copied, 0.825745 s, 635 MB/s
```

- The `dd` command uses special arguments to specify how it will work. The following illustrates some of the more commonly used arguments:

Argument	Description
if	Input File: The input file to be read from.
of	Output File: The output file to be written.
bs	Block Size: The block size to be used. By default, the value is considered to be in bytes. Use the following suffixes to specify other units: `K`, `M`, `G`, and `T` for kilobytes, megabytes, gigabytes and terabytes respectively.
count	Count: The number of blocks to be read from the input file.

- No block size or count needs to be specified when copying over entire devices. For example, to clone from one hard drive (**`/dev/sda`**) to another (**`/dev/sdb`**) execute the following command:

```
dd if=/dev/sda of=/dev/sdb
```

Moving & Renaming Files:

- The `mv` command is used to move a file from one location in the filesystem to another.

```
mv SOURCE DESTINATION
```

- The `mv` command requires at least two arguments. The first argument is the source, a path to the file to be moved. The second argument is the destination, a path to where the file will be moved to. The files to be moved are sometimes referred to as the source, and the place where the files are to be placed is called the destination.

```
sysadmin@localhost:~/Documents$ mv people.csv Work
```

- The `mv` command is able to move multiple files, as long as the final argument provided to the command is the destination. For example, to move three files into the **`School`** directory:

```
sysadmin@localhost:~/Documents$ mv numbers.txt letters.txt alpha.txt School
```

- Moving a file within the same directory is an effective way to rename it. For example, in the following example the **`animals.txt`** file is given a new name of **`zoo.txt`**:

```
sysadmin@localhost:~/Documents$ mv animals.txt zoo.txt
```

- If a destination directory is not specified, the file is renamed using the destination file name and remains in the source directory.

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

- Like the `cp` command, the `mv` command provides options such as `-i`, `-n` & `-v` but there is no `-r` option as the `mv` command moves directories by default.

Removing Files & Directories:

- The `rm` command is used to delete files and directories. It is important to keep in mind that deleted files and directories do not go into a "trash can" as with desktop-oriented operating systems. When a file is deleted with the `rm` command, it is almost always permanently gone.

```
rm [OPTIONS] FILE
```

- Without any options, the `rm` command is typically used to remove regular files:

```
sysadmin@localhost:~/Documents$ rm linux.txt
sysadmin@localhost:~/Documents$ ls linux.txt
ls: cannot access linux.txt: No such file or directory
```

- Users should use the `-i` option when deleting multiple files:

```
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
```

- To delete a directory, use a recursive option, either the `-r` or `-R` options. Just be careful since these options are "recursive", this will delete all files and all subdirectories:

```
sysadmin@localhost:~/Documents$ rm -r Work
sysadmin@localhost:~/Documents$ ls Work
ls: cannot access Work: No such file or directory
```

- 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.

- To delete a file within a directory, a user must have write and execute permission on a directory. Regular users typically only have this type of permission in their home directory and its subdirectories.

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.

- The ***compression algorithm*** is a procedure the computer uses to encode the original file, and as a result, make it smaller. 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.

- 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.

- 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`. Compressed files can be restored to their original form using either the `gunzip` command or the `gzip -d` command.

- To compress files using `gzip`:

```
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
```

- To display all information about the compressed file, use `-l` option:

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

- To restore compressed file to its original form using `gunzip` or `gzip -d`.

```
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
```

- 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.

Archiving Files:

- ***Archiving*** combines multiple files into one, which eliminates the overhead in individual files and makes the files easier to transmit.

- 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 has three modes that are helpful to become familiar with:

*** Create:** Make a new archive out of a series of files.

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

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

Option	Function
-c	Create an archive.
-f ARCHIVE	Use archive file. The argument ARCHIVE will be the name of the resulting archive file.
-z	Compress (or decompress) an archive using the gzip command.
-j	Compress (or decompress) an archive using the `bzip2` command.

- The following example shows a ***tar file***, also called a ***tarball***, being created from multiple files.

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

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

```
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 `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.

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

*** List:** Show the contents of the archive without extracting.

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

Listing an archive with the `tar` command requires 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.tar`** archive:

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

*** Extract:** Pull one or more files out of an archive.

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

Extract an archive with the `tar` command requires options once it's copied into a different directory.

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

ARCHIVE	
-v	Verbosely list the files processed.

- You can extract the archive with the `-x` option:

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

- You can get a verbose output of the files processed by adding the `-v` flag:

```
sysadmin@localhost:~/Downloads$ tar -xvf folders.tar
School/
School/Engineering/
School/Engineering/hello.sh
School/Art/
```

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…]]
```

```
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%)
```

- The `zip` command will not recurse into subdirectories by default, which is different behavior than the `tar` command. For instance, `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/Math/ (stored 0%)
updating: School/Math/numbers.txt (stored 0%)
```

```
  adding: School/Art/red.txt (deflated 33%)
  adding: School/Art/hidden.txt (deflated 1%)
```

- 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
---------                     -------
      900                     10 files
```

- 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
```

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 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**

- The Period (.) Character: It will match any character except for the new line character.

```
sysadmin@localhost:~/Documents$ grep 'r..f' red.txt
reef
roof
sysadmin@localhost:~/Documents$ grep 'r..d' red.txt
reed
read
```

* This 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
```

* 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 Bracket [ ] Characters: it will match a ***single*** character from the list or range of possible characters contained within the brackets.

* To find all the lines in the **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
```

* To find all the lines which contain any non-numeric characters, insert a **`^`** as the first character inside the brackets.

```
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.
```

***** When other regular expression characters are placed inside of square brackets, they are treated as literal characters. For example, the (**`.`**) normally matches any one character, but placed inside the square brackets, then it will just match itself.

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

- The Asterisk (*) Character: it's used to match zero or more occurrences of a character or pattern preceding it. For example **`e*`** would match zero or more occurrences of the letter **`e:`**

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

* 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 will match 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
```

* 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*.`**

```
sysadmin@localhost:~/Documents$ grep 'z*' red.txt
red
reef
rot
sysadmin@localhost:~/Documents$ grep 'e*' red.txt
red
reef
rot
```

* To make the **`*`** useful, it is necessary to create a pattern which includes more than just the one character preceding **`*`**.

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

- 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' 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$ grep 'r$' alpha-first.txt
B is for Bear
F is for Flower
```

- The Backslash \ Character

* In some cases, you may want to match a character that happens to be a special regular expression character. 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.
```

Quick Brief: The following table summarizes basic regular expression characters:

Basic Regex Character(s)	Meaning
.	Any one single character
[ ]	Any one specified character
[^]	Not the one specified character
*****	Zero or more of the previous character
^	If first character in the pattern, then pattern must be at beginning of the line to match, otherwise just a literal `^`
$	If last character in the pattern, then pattern must be at the end of the line to match, otherwise just a literal `$`

➢ 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 table summarizes the extended regular expressions, which must be used with either the `egrep` command or the `-E` option with the `grep` command:

Extended Regex Character(s)	Meaning
+	Matches previous character repeated one or more times
?	Matches previous character zero or one time, so it is an optional character
{ }	Specify minimum, maximum or exact matches of the previous pattern

|	Alternation or like a logical "or" operator
( )	Used to create groups

- 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
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?
```

Shutting Down:

- The `shutdown` command arranges for the system to be brought down in a safe way. All logged-in users are notified that the system is going down and within the last five minutes leading up to the shutdown, new logins are prevented.

```
shutdown [OPTIONS] TIME [MESSAGE]
```

```
sysadmin@localhost:~$ su -
Password:
root@localhost:~# shutdown now
Broadcast message from sysadmin@localhost
        (/dev/pts/0) at 2:05 ...
The system is going down for maintenance NOW!
```

- The `shutdown` command requires a time argument specifying when the shutdown should begin. Formats of this time argument can be the word **now**, a time of day in the format **hh:mm** or the number of minutes to delay in the format ***+minutes***.

```
weekday month day hour:minute:second UTC year
```

```
root@localhost:~# date
```

```
Sat Oct  3 22:15:58 UTC 2020
root@localhost:~# shutdown 01:51
```

- The `shutdown` command also has an optional message argument, indicating a message that will appear in the terminals of all users.

```
root@localhost:~# shutdown +1 "Goodbye World!"
The system is going down for maintenance in 1 minute!
Goodbye World!
```

Network Configuration:

- **Network Configuration Files**

- The configuration files that are used to store and modify network data may vary depending on the Linux distribution that you are working on.

- On a CentOS system, the primary configuration file for an IPv4 & IPv6 network interface is the **/etc/sysconfig/network-scripts/ifcfg-eth0** file.

```
root@localhost:~# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
BOOTPROTO=none
TYPE="Ethernet"
UUID="98cf38bf-d91c-49b3-bb1b-f48ae7f2d3b5"
IPV6INOT=no
NAME="System eth0"
IPADDR=192.168.1.1
PREFIX=24
GATEWAY=192.168.1.1
DNS1=192.168.1.2
```

- 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.
- The 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.

```
[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  [  OK  ]
```

- The address of the DNS server is stored in the **/etc/resolv.conf** file that is automatically generated and looks like the following:

```
sysadmin@localhost:~$ cat /etc/resolv.conf
nameserver 127.0.0.1
```

- Name resolution on a Linux host is accomplished by 3 critical files that describe the location of name service information, the order in which to check resources, and where to go for that information:

* **/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**.

Two other keywords may appear in the system's **/etc/resolv.conf** file; **domain** > followed by a qualified domain and **search** > followed by a set of separate domains which can be queried one after the other to resolve the name.

- 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. This 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. It will not failover (or continue) to the DNS option, even if the resolution is inaccurate. This can occur if the entry in **`/etc/hosts`** points to a non-assigned IP address.
3. Third, if the local **`/etc/hosts`** file doesn't result in a match, the system will use the configured DNS server entries contained in the **`/etc/resolv.conf`** file to attempt to resolve the name. The **`/etc/resolv.conf`** file should contain at least two entries for name servers; 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.

➢ **Network Tools**

- The ifconfig Command:

* The `ifconfig` command stands for ***interface configuration*** and is used to display network configuration information.

```
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
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
```

* 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.

* The `iwconfig` command is similar to the `ifconfig` command, but it is dedicated to wireless network interfaces.

- The ip Command:

* The `ifconfig` command is becoming obsolete in some Linux distributions 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 `ip` command branches out to do some of the work of several other legacy commands such as `route` and `arp`.

* The format for the `ip` command is as follows:

```
ip [OPTIONS] OBJECT COMMAND
```

```
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
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group defau
lt 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
```

- The route Command:

* 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
```

* Some users prefer to display this information with numeric data only, by using the `-n` option to the `route` command.

```
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 `route` command is becoming obsolete in some Linux distributions and is being replaced with a form of the `ip` command, specifically `ip route show`.

```
root@localhost:~# ip route show
default via 192.168.1.254 dev eth0 proto static
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.2
```

- The ping Command:

* By default, the `ping` command will continue sending packets until the break command (**CTL** + **C**) is entered at the console. To limit how many pings are sent, use the `-c` option followed by the number of pings to be sent.

```
root@localhost:~# ping -c 2 192.168.1.2
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
--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.043/0.052/0.064/0.007 ms
```

- The netstat Command:

* The `netstat` can be used to display information about network connections as well as display the routing table similar to the `route` command.

- 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.

- To see a list of all currently open TCP 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
```

- In the previous example, `-t` stands for *TCP*, `-l` stands for *listening* (which ports are listening) and `-n` stands for *show numbers, not names*.

- The ss Command:

* The `ss` command is designed to show socket statistics and supports all the major packet and socket types. It meant to be a replacement for and to be similar in function to the `netstat` command.

* 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.

- You can get a great deal of useful information from the `ss -l` command as shown in the example below.

```
root@localhost:~# ss -l
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
```

* The output is very similar to the output of the `netstat` command with no options. The columns above are:

Field	Description
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         -         -
UDP            9         6         3
TCP            7         3         4
INET           16        9         7
```

- The dig Command:

* The `dig` command performs queries on the DNS server to determine if the information needed is available on the server.

- For 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
;; QUESTION SECTION:
;example.com.                   IN      A
;; ANSWER SECTION:
example.com.            86400   IN      A       192.168.1.2
```

* 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
```

* You can use the `dig` command with `-x` option to resolve the IP address **`192.168.1.2`** to a hostname:

```
sysadmin@localhost:~$ dig -x 192.168.1.2
```

- The host Command:

* 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 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.
```

* 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 60480
```

* A list of DNS information regarding **example.com** can be found using the `-a` option:

```
root@localhost:~# host -a example.com
Trying "example.com"
;; QUESTION SECTION:
;example.com.                   IN      ANY
;; ANSWER SECTION:
example.com.            86400   IN      SOA     example.com. cserver.example.com.
example.com.            86400   IN      NS      example.com.
example.com.            86400   IN      A       192.168.1.2
```

- 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
bob@test's password:
bob@test:~$
```

- To return back to the local machine, use the `exit` command:

```
bob@test:~$ exit
logout
Connection to test closed.
root@localhost:~#
```

* 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.
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:~$ rm ~/.ssh/known_hosts
```

Processes:

- **Processes Files**

- 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, 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 like `top`, `free`, `mount`, `umount` and 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.

```
sysadmin@localhost:~$ ls /proc
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
```

- 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.

- 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

- 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 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:

```
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.
1 packets transmitted, 0 received, 100% packet loss, time 1000
```

➢ 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 starts other system processes, and each process is assigned a PID in sequential order.

- 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.

- 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.

- 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}]
```

- 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`

➢ Viewing Process Snapshot

- Regular users, like the **`sysadmin`** user, cannot control another user's processes. Users who have administrative privileges, like the **`root`** account, can control any user processes, including stopping any user process.

- The `ps` command can be used to list processes.

```
sysadmin@localhost:~$ ps
  PID TTY          TIME CMD
   80 pts/0        00:00:00 bash
   94 pts/0        00:00:00 ps
```

- The `ps` command will display the processes that are running in the current terminal by default. The output includes the following columns of information:

- **PID:** The process identifier, which is unique to the process. This information is useful for controlling the process by its ID number.
- **TTY:** The name of the terminal where the process is running. This information is useful for distinguishing between different processes that have the same name.
- **TIME:** The total amount of processor time used by the process. Typically, this information isn't used by regular users.
- **CMD:** The command that started the process.

- The `-e` option will display every process running on the system and also the `-ef` & `aux` option is used as it provides more detail in the output of the command, including options and arguments:

```
sysadmin@localhost:~$ ps -e
  PID TTY          TIME CMD
    1 pts/0        00:00:00 init
   33 ?            00:00:00 rsyslogd
   69 pts/0        00:00:00 login
   79 pts/0        00:00:00 bash
```

```
   94 pts/0        00:00:00 ps
sysadmin@localhost:~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 19:16 pts/0        00:00:00 /sbin??? /init
syslog      33     1  0 19:16 ?            00:00:00 /usr/sbin/rsyslogd
root        37     1  0 19:16 ?            00:00:00 /usr/sbin/cron
sysadmin    79    69  0 19:16 pts/0        00:00:00 -bash
sysadmin    95    79  0 19:43 pts/0        00:00:00 ps -ef
```

- You can use `head` or `tail` to filter the output of ps command to avoid scrolling through hundreds of processes to find what might interest you.

```
sysadmin@localhost:~$ ps -ef | head
```

- Sending the output to a pager such as the `less` command can also make the output of the `ps` command more manageable.

```
sysadmin@localhost:~$ ps -ef | less
```

- 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.

```
sysadmin@localhost:~$ ps -e | grep firefox
 6090 pts/0    00:00:07 firefox
```

- Use the `ps` command with the `-o` option to specify which columns to output.

```
root@localhost:~# ps -o pid,tty,time,%cpu,cmd
  PID TT           TIME %CPU CMD
    1 ?        00:00:00  0.0 /bin/bash /init
   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
```

- 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.

```
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
   91 ?        00:00:00  0.0 bash
```

```
   51 ?        00:00:00  0.0 /bin/login -f
```

- 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
```

- You can manage processes by using the following:

- To start the `ping` process in the background, type 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.
- To see which commands are running in the current terminal, type the following command:

```
root@localhost:~# jobs
[1]-  Running                 ping localhost > /dev/null &
[2]+  Running                 ping localhost > /dev/null &
```

- Once you have verified that two `ping` commands are running, bring the first command to the foreground by typing the following:

```
root@localhost:~# fg %1
ping localhost > /dev/null
```

- To have this process continue executing in the background, execute the following command:

```
root@localhost:~# bg %1
[1]+ ping localhost > /dev/null &
```

- Using the job number, stop the last `ping` command with the `kill` command and verify it was stopped executing the `jobs` command:

```
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
```

- To terminate processes, you can also use **`pkill`** and **`kill`** with the PID:

```
root@localhost:~# kill 134
root@localhost:~# pkill -9 sleep
```

➢ Viewing Processes in Real Time

- 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:

- There is an extensive amount of interactive commands that can be executed from within the running `top` program:

Key	Action
H	View a full list
k	Terminate the runaway process.
r	Adjust the priority of the process.
q	Exit the top program and return to the prompt

- 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.

- To kill a remaining process, at the signal prompt **`Kill PID with signal [15]:`** prompt, use a value of **`9`** instead of accepting the default of **`15`**. Press **Enter** to accept the entry.
- The kill signal **`9`** or **`SIGKILL`** is a "forceful" signal that cannot be ignored, unlike the default value of **`15`**.

- 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.

- The `top` 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
```

- 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:

```
sysadmin@localhost:~$ cat /proc/loadavg
0.12 0.46 0.25 1/254 3052
```

- **Load Average:**

```
0.12 0.46 0.25 1/254 3052
```

  The first three numbers in this file indicate the load average over the last one, five and fifteen minute intervals.

- **Number of Processes:**

```
0.12 0.46 0.25 1/254 3052
```

  The fourth value is a fraction which shows the number of processes currently executing code on the CPU `1` and the total number of processes `254`.

- **Last PID:**

```
0.12 0.46 0.25 1/254 3052
```

  The fifth value is the last PID value that executed code on the CPU.

- The number reported as a load average is proportional to the number of CPU cores that are able to execute processes. On a single-core CPU, a value of one (`1`) would mean that the system is fully-loaded. On a four core CPU, a value of `1` would mean that the system is only 1/4 or 25% loaded.

Memory

- **Linux System 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***.

- 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.

- 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:** 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.

➢ 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. Without these options, the output is displayed in bytes:

- When reading the output of the `free` command:

- **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 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
Swap:            0          0          0
```

- If the amount of memory and swap that is available becomes very low, then the system will begin to automatically terminate processes, making it critical to monitor the system's memory usage.

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***.

- Some processes can log their own data to these files, other processes rely on a separate process (**a *daemon*** = *a background process that handles requests for services such as print spooling and file transfers, and is dormant when not required.*) to handle these log data files.

- ***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 executed on a recurring basis.
dmesg	Messages generated by the kernel during system boot up.
maillog	Messages produced by the **`mail`** daemon for e-mail messages sent or received.
messages	Messages from the kernel and other processes that don't belong elsewhere. Sometimes named **`syslog`** instead of **`messages`** after the daemon that writes this file.
secure	Messages from processes that required authorization or authentication (such as the login process).
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.

- You can use the `cat` command, or the `less` command to allow for searching, scrolling and other options.
- You can 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.

- 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 **`/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, 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
```

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/.

- 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.

- The FHS standard defines four hierarchies of directories used in organizing the files of the filesystem. The top-level or root hierarchy follows:

Directory	Contents
/	The base of the structure, or root of the filesystem, this directory unifies all directories regardless of whether they are local partitions, removable devices or network shares
/bin	Essential binaries like the `ls`, `cp`, and `rm` commands, and be a part of the root filesystem
/boot	Files necessary to boot the system, such as the Linux kernel and associated configuration files
/dev	Files that represent hardware devices and other special files, such as the `/dev/null` and `/dev/zero` files
/etc	Essential host configurations files such as the `/etc/hosts` or `/etc/passwd` files
/home	User home directories
/lib	Essential libraries to support the executable files in the `/bin` and `/sbin` directories
/lib64	Essential libraries built for a specific architecture. For example, the `/lib64` directory for 64-bit AMD/Intel x86 compatible processors
/media	Mount point for removable media mounted automatically
/mnt	Mount point for temporarily mounting filesystems manually
/opt	Optional third-party software installation location
/proc	Virtual filesystem for the kernel to report process information, as well as other information
/root	Home directory of the root user
/sbin	Essential system binaries primarily used by the root user
/sys	Virtual filesystem for information about hardware devices connected to the system
/srv	Location where site-specific services may be hosted
/tmp	Directory where all users are allowed to create temporary files and that is supposed to be cleared at boot time (but often is not)
/usr	Second hierarchy Non-essential files for multi-user use
/usr/local	Third hierarchy Files for software not originating from distribution

/var	Fourth hierarchy Files that change over time
/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.

- 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. Typically, only the user will have access to his 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.

- Root-Restricted Binaries:

The **sbin** directories are primarily intended to be used by 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.

* If the **`/var`** directory is ***not*** a separate partition, then the root filesystem could become full and cause the system to crash.

Package Management:

- Installing Packages: Package files are commonly installed by downloading them directly from repositories located on Internet servers. The Debian repositories contain more than 65,000 different packages of software. Before installing a package, it is good practice to use the refresh the list of available packages using the `apt-get update` command.

```
sudo apt-get update
```

```
sysadmin@localhost:~$ sudo apt-get update
[sudo] password for sysadmin:
Ign file: amd64/ InRelease
Ign file: amd64/ Release.gpg
Reading package lists... Done
```

* To search for keywords within these packages, you can use the `apt-cache search` command. The keyword that is used should match part of the name or description of the package that is to be located. Multiple keywords can be used to further clarify the search.

```
apt-cache search [keyword]
```

```
sysadmin@localhost:~$ apt-cache search cow
cowsay - configurable talking cow
```

* Once you've found the package that you want to install, you can install it with the `apt-get install` command:

```
sudo apt-get install [package]
```

```
sysadmin@localhost:~$ sudo apt-get install cowsay
[sudo] password for sysadmin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
The following NEW packages will be installed:
```

- Updating Packages: The `apt-get install` command can also update a package, if that package is installed and a newer version is available.

* Updating all packages of the system should be done in two steps. First, update the cache of all packages available with `apt-get update`. Second, execute the `apt-get upgrade` command and all packages and dependencies will be updated.

```
apt-get update
apt-get upgrade
```

```
sysadmin@localhost:~$ sudo apt-get update
[sudo] password for sysadmin:
Ign file: amd64/ InRelease
Ign file: amd64/ Release
Reading package lists... Done
sysadmin@localhost:~$ sudo apt-get upgrade
Reading package lists... Done
Reading state information... Done
Calculating upgrade... Done
```

- Removing Packages: The `apt-get` command is able to either remove or purge a package. The difference between the two is that purging deletes all package files, while removing deletes all but the configuration files for the package.

```
apt-get remove [package]
apt-get purge [package]
```

```
sysadmin@localhost:~$ sudo apt-get purge cowsay
Do you want to continue? [Y/n] y
Removing cowsay (3.03+dfsg1-6) ...
```

Updating User Passwords:

- The `passwd` command is used to update a user's password. Users can only change their own passwords, whereas the root user can update the password for any user.

```
passwd [OPTIONS] [USER]
```

```
sysadmin@localhost:~$ passwd
Changing password for sysadmin.
(current) UNIX password: netlab123
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

- If the user wants to view ***status*** information about their password, they can use the `-S` option:

```
sysadmin@localhost:~$ passwd -S sysadmin
sysadmin P 12/20/2017 0 99999 7 -1
```

Field	Example	Meaning
User Name	sysadmin	The name of the user.
Password Status	P	P >> indicates a usable password. L >> indicates a locked password. NP >> indicates no password.
Change Date	42064	The date when the password was last changed.
Minimum	0	The minimum number of days that must pass before the current password can be changed by the user.
Maximum	99999	The maximum number of days remaining for the password to expire.
Warn	7	The number of days prior to password expiry that the user is warned.
Inactive	-1	The number of days after password expiry that the user account remains active.

- The root user can change the password of any user. If the root user wants to change the password for **`sysadmin`**, they would execute the following command:

```
root@localhost:~# passwd sysadmin
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Redirection:

- There is a way in Linux to quickly add content to a file using a command line feature called ***input/output (I/O) redirection***. I/O redirection allows for information in the command line to be sent to files, devices, and other commands. Redirection is achieved by using the arrow `< >` characters.

- The input or output of a command is redirected from its default destination to a different location. There are three called ***file descriptors*** to command input and output:

- <u>Standard Input (STDIN)</u>:

* The concept of redirecting STDIN is a difficult one because it is more difficult to understand why you would want to redirect STDIN.

* It receives and processes when it is executed, essentially what a user types on the keyboard.

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

* 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.

* 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
```

- 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 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
How are you?
sysadmin@localhost:~$ cat new.txt
How are you?
```

- 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.

```
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
```

- 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
```

- <u>Standard Output (STDOUT)</u>:

*It displays the output of the command and can be directed to files.

When a command functions correctly without errors the output it produces is called STDOUT (also known as ***stream*** or ***channel*** #1)

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

- 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
```

```
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  example.txt
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
```

- 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
```

- <u>Standard Error (STDERR)</u>:

* It's the error messages generated by commands that are not correctly executed. STDERR is also known as ***stream*** or ***channel*** #2.

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

* STDERR can be redirected 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.

- 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
```

- 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.
- 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
```

- 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
```

- 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
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
```

- 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
```

- 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
```

Text Editor:

- The premier text editor for Linux and UNIX is a program called `vi`. While there are numerous editors available for Linux that range from the tiny editor `nano` to the massive `emacs` editor.

➢ Vi Text Editor

- Most Linux systems don't include the original `vi`, but an improved version of it known as `vim`, for *vi improved*. For the most part, `vim` works just like `vi`, but has additional features.

- To get started using `vi`, type the command followed by the pathname to the file to edit or create:

```
sysadmin@localhost:~$ vi newfile.txt
```

- There are three modes used in `vi`:

*** Command Mode Movement:** is used to type commands, such as those used to move around a document, manipulate text, and access the other two modes. To return to command mode at any time, press the **Esc** key.

Movement commands in `vi` have two aspects, a motion and an optional number prefix, which indicates how many times to repeat that motion. The general format is as follows:

```
[count] motion
```

Motion	Result
h	Moves cursor to the left one character
j	Moves cursor down one line
k	Moves cursor up line
l	Moves cursor to the right one character
w	Moves cursor to beginning of next word
b	One word back
^	Beginning of line
$	End of the line
e	Moves cursor to end of word
b	Moves cursor to beginning of previous word

These motions can be prefixed with a number to indicate how many times to perform the movement. For example, `5h` would move the cursor five characters to the left and `3w` would move the cursor three words to the right.

To move the cursor to a specific line number, type that line number followed by the `G` character. For example, to get to the fifth line of the file type `5G`. `1G` or `gg` can be used to go to the first line of the file, while a lone `G` will take you to the last line. To find out which line the cursor is currently on, use **CTRL+G**.

*** Command Mode Actions:** is to use copy, cut, and paste. The `vi` program has none of these. Instead, `vi` uses the following three commands:

Standard	Vi	Meaning
cut	d	delete
copy	y	yank
paste	P | `p`	put

✓ Delete

Delete removes the indicated text from the page and saves it into the buffer. The following table provides some common usage examples:

Action	Result
dd	Delete current line
3dd	Delete the next three lines
dw	Delete the current word
d3w	Delete the next three words
d4h	Delete four characters to the left

✓ Change

Change is very similar to delete; the text is removed and saved into the buffer. The following table provides some common usage examples:

Action	Result
cc	Change current line
cw	Change current word
c3w	Change the next three words
c5h	Change five characters to the left

✓ Yank

Yank places content into the buffer without deleting it. The following table provides some common usage examples:

Action	Result
yy	Yank current line
3yy	Yank the next three lines
yw	Yank the current word
y$	Yank to the end of the line

✓ Put

Put places the text saved in the buffer either before or after the cursor position. Notice that these are the only two options, put does not use the motions like the previous action commands.

Action	Result
p	Put (paste) after cursor
P	Put before cursor

✓ Searching in vi

To search forward from the current position of the cursor, use the `/` to start the search, type a search term, and then press the **Enter** key to begin the search. The cursor will move to the first match that is found.

To proceed to the next match using the same pattern, press the `n` key. To go back to a previous match, press the `N` key. If the end or the beginning of the document is reached, the search will automatically wrap around to the other side of the document.

To start searching backwards from the cursor position, start by typing `?`, then type the pattern to search for matches and press the **Enter** key.

*** Insert Mode:** Insert mode is used to add text to the document. The following table covers the most common:

Input	Purpose
a	Enter insert mode right after the cursor
A	Enter insert mode at the end of the line
i	Enter insert mode right before the cursor
I	Enter insert mode at the beginning of the line
o	Enter insert mode on a blank line after the cursor
O	Enter insert mode on a blank line before the cursor

*** Ex Mode:** the `vi` editor was called the `ex` editor which only allowed users to see and modify one line at a time. In the visual mode, users could see as much of the document that will fit on the screen.

The following table lists some common actions performed in ex mode:

Input	Purpose
:w	Write the current file to the filesystem
:w ***filename***	Save a copy of the current file as *filename*
:w!	Force writing to the current file
:1	Go to line number 1 or whatever number is given
:e ***filename***	Open *filename*
:q	Quit if no changes made to file
:q!	Quit without saving changes to file

- **Nano Text Editor**

- The GNU nano editor is a very simple editor well suited to editing small text files. 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.

- 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.

- 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

---

Scripting Basics:

- 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 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.

- 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
Hello, World!
```

- 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.

- 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.

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

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

➢ 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 echoes 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***.

- 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"
```

– Another way to assign to a variable is to use the output of another command as the contents of the variable by enclosing the command in back ticks:

```
#!/bin/bash
CURRENT_DIRECTORY=`pwd`
echo "You are in $CURRENT_DIRECTORY"
```

- It is possible to get input from the user of your script and assign it to a variable through the `read` command:

```
#!/bin/bash
echo -n "What is your name? "
read NAME
echo "Hello $NAME!"
```

- 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`**.

➢ Conditionals

- Now that you can look at and set variables, it is time to make your script do different functions based on tests, called ***branching***. The **if** statement is the basic operator to implement branching.

- A basic `if` statement looks like this:

```
if somecommand; then
  # do this if somecommand has an exit code of 0
fi
```

- The next example will run “**somecommand**” (actually, everything up to the semicolon) and if the exit code is `0` then the contents up until the closing `fi` will be run. Using what you know about `grep`, you

can now write a script that does different things based on the presence of a string in the password file:

```
#!/bin/bash
if grep -q root /etc/passwd; then
  echo root is in the password file
else
  echo root is missing from the password file
fi
```

- From previous examples, you might remember that the exit code of `grep` is `0` if the string is found. The example above uses this in one line to print a message if **`root`** is in the password file or a different message if it isn't. The difference here is that instead of an **`fi`** to close off the **`if`** block, there's an **`else`**. This lets you do one action if the condition is true, and another if the condition is false. The **`else`** block must still be closed with the **`fi`** keyword.

- Other common tasks are to look for the presence of a file or directory and to compare strings and numbers. You might initialize a log file if it doesn't exist, or compare the number of lines in a file to the last time you ran it. The **`if`** command is clearly the one to help here, but what command do you use to make the comparison?

- The `test` command gives you easy access to comparison and file test operators. For example:

Command	Description
test –f /dev/ttyS0	0 if the file exists
test ! –f /dev/ttyS0	0 if the file doesn't exist
test –d /tmp	0 if the directory exists
test –x `which ls`	substitute the location of `ls` then `test` if the user can execute
test 1 –eq 1	0 if numeric comparison succeeds
test ! 1 –eq 1	NOT – 0 if the comparison fails
test 1 –ne 1	Easier, `test` for numeric inequality
test "a" = "a"	0 if the string comparison succeeds
test "a" != "a"	0 if the strings are different
test 1 –eq 1 –o 2 –eq 2	-o is OR: either can be the same
test 1 –eq 1 –a 2 –eq 2	-a is AND: both must be the same

- 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.

- **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.

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 work. Variables are given names and stored temporarily in memory.

- There are two types of variables used in the Bash shell:

* **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.

- 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
```

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

- 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
```

*** Environment Variables:** *Environment* or *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.

- 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
```

- 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.

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

The pipe | character passes the output of the `env` command to the `grep` command, which searches the output.

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

```
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.

```
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
```

Path Variable:

- The path variable 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/game
```

- 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:/sbi
n:/bin:/usr/games
```

Quoting:

- Quotation marks are used 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.

*** Double Quotes (`"`):** stop the shell from interpreting some metacharacters (special characters), including glob characters (wild cards) that have special meaning to the shell; they are interpreted by the shell itself before it attempts to run any command.

- 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.

```
sysadmin@localhost:~$ echo "The path is $PATH"
The path is /usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr
```

*** 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.

- 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
```

*** Backslash (`\`):** 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:/us
r/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 $1 and the path is $PATH
```

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

```
sysadmin@localhost:~$ echo The service costs \$1 and the path is $PATH
The service costs $1 and the path is /usr/bin/custom:/home/sysadmin/bi
```

*** Backquotes (`):** Also called *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
sysadmin@localhost:~$ echo Today is date
Today is date
```

- 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
```

Globbing:

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

- 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.

*** Asterisk (*):** is used to represent zero or more of any character in a filename.

- 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
```

- You can use the asterisk character at any place within the filename pattern:

```
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
```

*** Question Mark (?):** represents any single character. Each question mark character matches exactly one character, no more and no less.

- 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
```

- 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/*.???`**

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

*** Square Bracket [ ]:** are used to match a single character by representing a range of characters that are possible match characters. Brackets can also be used to a represent a range of characters.

- 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/
```

- The **`/etc/[a-d]*`** pattern matches all files begin with any letter between and including **`a`** & **`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 /
```

- 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 /
```

*** Exclamation point (!):** is used in conjunction with the square brackets to negate a range.

- 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
```

Control Statements:

- Control statements allow you to use multiple commands at once or run additional commands, depending on the success of a previous command:

*** Semicolon (;):** 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.

```
command1; command2; command3
```

```
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
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26
```

*** Double Ampersand (&&):** acts as a logical "and"; if the first command is successful, then the second command will also run. If the first command fails, then the second command will *not* run.

```
command1 && command2
```

- To use the success or failure of the `ls` command in conjunction with `&&` execute commands like the following. In the first example, the `echo` command is executed because the `ls` command succeeds:

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

- In the second example, the `echo` command isn't executed because the `ls` command fails:

```
sysadmin@localhost:~$ ls /etc/junk && echo success
ls: cannot access /etc/junk: No such file or directory
```

* **Double Pipe (||):** acts 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 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
failed
```

Man Pages:

- Man pages (short for *manual 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.

- Man pages are easily distinguished from commands as they are typically compressed with a program called `gzip`, resulting in a filename that ends in **`.gz`**.

- To view a man page for a command, use the `man` command:

```
man command
```

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

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

- Man pages are broken into sections. Each section is designed to provide specific information about a command. 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. It can be difficult to understand but is very important because it provides a concise example of how to use the command.

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

* **DESCRIPTION:** Provides a more detailed description of the command.

```
DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
```

* **OPTIONS:** Lists the options for the command as well as a description of how they are used.

```
       -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
```

*** 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.

*** AUTHOR:** Provides the name of the person who created the man page and (sometimes) how to contact the person.

*** 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/>
```

*** 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.

- 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 & Conventions
6. Games
7. Miscellaneous
8. System Administration Commands
9. Kernel Routines

- 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.

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

- 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.

- To search for man pages by name, use the `-f` option to the `man` command.

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

- To specify a different section, provide the number of the section as the first argument of the man command.

```
sysadmin@localhost:~$ man 5 passwd
```

- The `-k` option to the `man` command searches both the names and descriptions of the man pages for a keyword.
- Note that the `apropos` command is another way of viewing man page summaries with a keyword.

```
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
dd (1)               - convert and copy a file
```

Finding Commands & 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
```

- To search for the location of a command or the man pages for a command, use the `whereis` command. This command searches for commands, source files and man pages in specific locations where these files are typically stored:

```
sysadmin@localhost:~$ whereis ls
ls: /bin/ls /usr/share/man/man1p/ls.1.gz /usr/share/man/man1/ls.1.gz
```

Finding File & Directory:

- 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.

```
sysadmin@localhost:~$ locate gshadow
```

- 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. 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.

- 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
```

- 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
```

Info Documentation:

- The `info` command also provides documentation on operating system commands and features.

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

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

- This documentation is broken up into nodes. The first line provides index information about the current location within the document.

- To go back to the previous node, use the **U** key. The **L** key returns to the same location as before entering the sorting node.
- To get the listing of movement commands is available, hit the **Shift**+**H** key while reading the info documentation.
- To close the help screen type the **L** key.
