UNIT I

Open source definition:

Open source software is software with source code that anyone can inspect, modify, and enhance. "Source code" is the part of software that most computer users don't ever see; it's the code computer programmers can manipulate to change how a piece of software—a "program" or "application"—works.

ADVANTAGES

1. **Cheaper than commercially marketed products.**

   Open source software collectively help business owners save around $60 billion a year. This might seem unbelievable at first, but it’s not really surprising since these programs are developed to be accessible to anyone (especially those who can’t afford to buy commercial products). For one thing, they’re usually offered for free and don’t require you to pay for any additional copy you download. Since many of these programs are created to work with almost any type of computer, they can also help you lengthen the life of your old hardware and avoid the need to replace them every now and then.

2. **Created by skilful and talented people**.

Large and well-established software companies have the financial capability to hire the best talent in the business to create their products. Because of this,

many people opt to buy computer programs from these firms because they think they'll get great value for their money by doing so. Many software developers turn to open source products as an outlet for their ideas and creativity. By doing so, they won't be confined by the rigid rules of the corporate world, and they'll have the freedom to experiment and come up with high-quality programs. These, in turn, benefit consumers since they'll have access to world-class and state-of-the-art software without having to pay too much.

3. Highly reliable

There are two main reasons why open source software are reliable. First of all, they're developed chiefly by skilful and talented experts who do their best to create high-quality programs. Second, they're worked on by tens or hundreds of people, which means there are numerous eyes that can monitor for the presence of bugs and many pairs of hands that can fix these defects within the shortest amount of time. Both of these factors lead to products that have excellent quality and helpful features and perform well most (if not all) of the time.

4. Help you become more flexible.

Since you're not tied to a proprietary product, you don't need to abide by a specific IT architecture that might require you to upgrade your software and

even hardware often. Rather, you can mix and match your software and create a unique IT infrastructure that best suits your needs. There's no need to fret since there's a wide range of options in the market, so you only have to browse through them and pick one those that match your requirements and specifications .if Can't find anything you like? You can modify existing open sources software or hire someone who can do it for you.

DISADVANTAGES:

1. **Vulnerable to malicious users**.

Many people have access to the source code of open source software, but not all of them have good intentions. While a lot of people utilize their access to spot defects and make improvements to the program, others use this privilege to exploit the product's vulnerabilities and create bugs that can infect hardware, steal identities or just annoy other users. These rarely happen with commercially produced software since the companies who make them have stringent quality control processes and ensure that the program is almost perfect when released to the market.

2. Might not be as user-friendly as commercial versions.

This is not true for all open source software since many of them (such as LibreOffice, Mozilla Firefox and the Android operating system) are incredibly easy to use. However, there are several programs which are created mainly to cater to the developer's wishes and bring his ideas to life. As a result, not much attention is given to the software's user interface, making it difficult to use especially for those who aren't really tech-savvy.

3. Don't come with extensive support.

Those who favour commercially produced programs say that these software gives them peace of mind. After all, since they know exactly who designed, created and distributed the product, they have a clear idea of who they can hold liable if the program doesn't function properly or causes damage to their hardware. This isn't exactly the case for open source software. Since it's developed by numerous people, users exactly don't have a specific person or company they can point a finger to.

APPLICATIONS:

1: Server software

The flagship open source web server software is one of the most widely used on the planet. It's free, incredibly reliable, easy to manage, and doesn't require the enormous overhead needed for other software's.

2: Development

There are a lot of options for developing for open source or with open source tools (as are there with proprietary development). The biggest difference between open source and proprietary is the access you have to the software code. Within the world of FOSS (free open source software) the code is readily available. For many developers, the Linux operating system has everything they need to develop, built right in (especially those who code without a full-blown IDE). If you do require GUI development tools, open source has you covered.

3: Security

The open source security route will take a bit more time to deploy (with a much higher learning curve), but the end results are generally incredible. This doesn't even address the idea that using open source on the desktop is, generally speaking, a more secure platform than most proprietary systems. Deploy Linux on the desktops and your security woes will drop dramatically.

4: Desktops

This area is where most of the pushback happens. However, you must take into consideration the fact that the daily workflow has undergone a major paradigm shift. Most of what we do now is done via a web browser. So why not deploy Linux on the desktop? Not only does it work with the majority of today's tasks, it will do so without suffering from viruses, malware, and updates that cripple a

system. It's not perfect — what platform is? But it's solid, and in the end, it can save you money. That's a win-win.

5: Workflow

Every business depends upon workflow. For some businesses, a smooth workflow depends upon tools. Open source has this arena covered.open source handles just about every possible acronym you can think of — and it does it very well.

6: E-commerce: powerful e-commerce solutions are available — regardless of license. With just about every feature you could possible want (and some you probably haven't even thought of), the open source platform excels at e-commerce on every level.

FOSSS

Fee and open-source software (FOSS) is computer software that can be classified as both free software and open-source software. That is, anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software. This is in contrast to proprietary software, where the software is under restrictive copyright and the source code is usually hidden from the users.

The benefits of using FOSS can include decreasing software costs, increasing security and stability (especially in regard to malware), protecting privacy, and giving users more control over their own hardware. Free, open-source operating systems such as Linux are widely utilized today, powering millions of servers, desktops, smart phones (e.g. Android), and other devices. Free software licenses and open-source licenses are used by many software packages.

FREE SOFTWARE MOVEMENT

The free software movement (FSM) or free / open source software movement (FOSSM) or free / libber open source software (FLOSS) is a social movement with the goal of obtaining and guaranteeing certain freedoms for software users, namely the freedom to run the software, to study and change the software, and to redistribute copies with or without changes. Although drawing on traditions and philosophies among members of the 1970s hacker culture and academia, Richard Stallman formally founded the movement in 1983 by launching the GNU Project. Stallman later established the Free Software Foundation in 1985 to support the movement.

The philosophy of the movement is that the use of computers should not lead to people being prevented from cooperating with each other. In practice, this means rejecting "proprietary software", which imposes such restrictions, and

promoting free software, with the ultimate goal of liberating everyone in cyberspace – that is, every computer user. Stallman notes that this action will promote rather than hinder the progression of technology, since "it means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art".

Members of the free software movement believe that all users of software should have the freedoms listed in The Free Software Definition. Many of them hold that it is immoral to prohibit or prevent people from exercising these freedoms and that these freedoms are required to create a decent society where software users can help each other, and to have control over their computers.

Some free software users and programmers do not believe that proprietary software is strictly immoral, citing an increased profitability in the business models available for proprietary software or technical features and convenience as their reasons.

The Free Software Foundation also believes all software needs free documentation, in particular because conscientious programmers should be able to update manuals to reflect modification that they made to the software, but deems the freedom to modify less important for other types of written works. Within the free software movement, the FLOSS Manuals foundation specialises on the goal of providing such documentation. Members of the free software

movement advocate that works which serve a practical purpose should also be free.

UNIT II

Define Linux:

Linux is a freely distributed, open source; complete multitasking, multi user operating system that behaves similar to the UNIX operating system. The Linux operating system was specifically designed for personal computers and workstations. Linux operating system was developed by a computer science student namely Linus Torvald's in 1990. After that developers across the globe have joined with torvald's and today we have a large community of open source developers who are contributing in evolution of Linux environment.

Features of Linux operating system.

- Multi-tasking.
- Multi-user
- Multi-threading
- Memory protection
- Demand paging.
- Supports several file systems.

Advantages of Linux operating system.

- Linux is free.
- Open source.
- Supports various types of hardware.

- High performance report.
- Runs on various types of computers.
- Great networking support.
- Various types of distributions.
- Fast installation.
- Secure.

HISTORY OF LINUX

All modern operating systems have their roots in 1969 when Dennis Ritchie and Ken Thompson developed the C language and the Unix operating system at AT&T

Bell Labs. They shared their source code (yes, there was open source back in the Seventies) with the rest of the world. By1975, when AT&T started selling Unix commercially, about half of the source codewas written by others.

In the Eighties many companies started developing their own Unix The result was a mess of Unix dialects and a dozen different ways to do thesame thing. And here is the first real root of Linux, when Richard Stallman aimedto end this era of Unix separation and everybody re-inventing the wheel by startingthe GNU project (GNU is Not Unix). His goal was to make an operating system that was freely available to everyone, and where everyone could work together (like in the Seventies).

The Nineties started with Linus Torvalds, a Swedish speaking Finnish student,buying a 386 computer and writing a brand new POSIX compliant kernel. He put the source code online, thinking it would never support anything but 386 hardware.

Many people embraced the combination of this kernel with the GNU tools.Today more than 90 percent of supercomputers morethan half of all smartphones, many millions of desktop computers, around 70 percentof all web servers, a large chunk of tablet computers, and several appliances (dvdplayers,washing machines, dsl modems, routers, ...) run Linux. It is by far the most commonly used operating system in the world.

FEATURES OF LINUX

Following are some of the important features of Linux Operating System.

- Portable – Portability means softwares can works on different types of hardwares in same way.Linux kernel and application programs supports their installation on any kind of hardware platform.
- Open Source – Linux source code is freely available and it is community based development project. Multiple team's works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- Multi-User – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- Multiprogramming – Linux is a multiprogramming system means multiple applications can run at same time.
- Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.
- Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

DRAWBACKS OF LINUX

No Standard Edition

While Windows and Mac have several definite versions, there is no one standard edition of Linux. In fact, there are hundreds of different user-developed editions. It can be challenging to figure out which one is best for you, and making that decision can be overwhelming for a new user.

Learning Curve

Linux is not as easy to use as Windows or Mac. It requires a broader base of knowledge about computing than other operating systems, and this can be very challenging for a beginning user. If you are used to using Windows or Mac, you will have to unlearn and relearn many different functions and processes. It can take some time, and the less technical understanding you have the more it will require of you to learn. While it is certainly possible to gain a functional understanding of Linux with practice and self-teaching, it will require more effort than with other operating systems.

Non-Compatible Software

A disadvantage to using a Linux OS is that the majority of your favourite programs will not run on it. If you are used to certain software, you will have to find a comparable Linux option. There are hundreds of choices of programs, and there are many that are similar to specific Windows or Mac software. However, a lot of times the user interface is very different and not every function you want is always available. You will have to do some searching and testing of different programs until you find ones that you like and meet your needs.

Unsupported Hardware

There is less computer hardware that is compatible with Linux, also. There is a much smaller selection of drivers that will work with Linux, although more are being added on a consistent basis. Often times it takes a while for new hardware to be supported, and you may find that a lot of the hardware you already have will be tough to run on Linux. For some reason, many people encounter problems with running their printers on Linux. Blu-ray discs are also not able to be played using Linux.

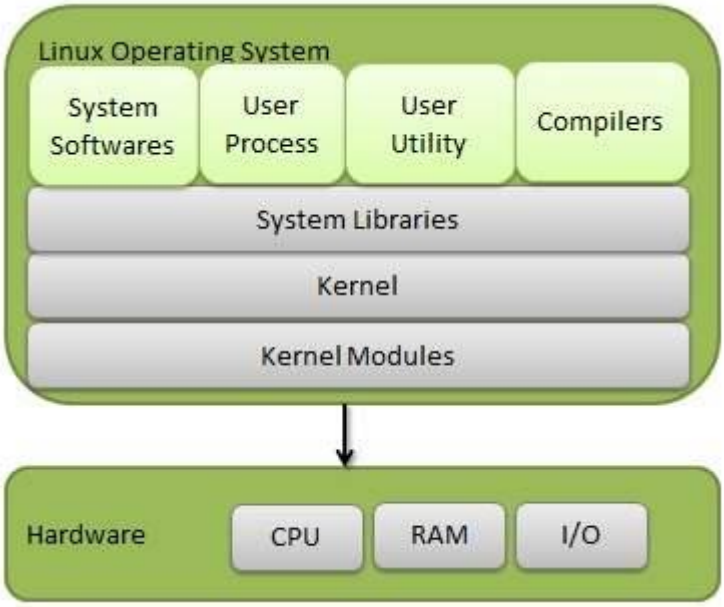
Tech Support

While there is a large community of Linux users that are very helpful in answering your technical questions in forums and chat rooms, it can be more difficult to get assistance for your issues. You can ask questions of Linux users, but sometimes their answers will be difficult to understand if your technical knowledge is lacking. There is also no immediate help because you will be waiting for someone to write a response rather than calling a central tech support hotline where someone will answer right away. Also, it can be difficult to find a computer repair person who is versed in Linux.

Components of Linux System

Linux Operating System has primarily three components:

- Kernel − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- System Library − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.
- System Utility − System Utility programs are responsible to do specialized, individual level tasks.



DIFFRENCE BETWEEN LINUX AND WINDOWS

1: Full access vs. no access

Having access to the source code is probably the single most significant difference between Linux and Windows. The fact that Linux belongs to the GNU Public License ensures that users (of all sorts) can access (and alter) the code to the very kernel that serves as the foundation of the Linux operating system. Unless you are a member of a very select (and elite, to many) group, you will never lay eyes on code making up the Windows operating system.

2: Licensing freedom vs. licensing restrictions

Along with access comes the difference between the licenses. I'm sure that every IT professional could go on and on about licensing of PC software. But let's just look at the key aspect of the licenses (without getting into legalese). With a Linux GPL-licensed operating system, you are free to modify that software and use and even republish or sell it (so long as you make the code available). Also, with the GPL, you can download a single copy of a Linux distribution (or application) and install it on as many machines as you like. With the Microsoft license, you can do none of the above. You are bound to the number of licenses you purchase, so if you purchase 10 licenses, you can legally install that operating system (or application) on only 10 machines.

3: Online peer support vs. paid help-desk support

This is one issue where most companies turn their backs on Linux. But it's really not necessary. With Linux, you have the support of a huge community via forums, online search, and plenty of dedicated Web sites. And of course, if you feel the need, you can purchase support contracts from some of the bigger Linux companies (Red Hat and Novell for instance).

However, when you use the peer support inherent in Linux, you do fall prey to time. You could have an issue with something, send out e-mail to a mailing list or post on a forum, and within 10 minutes be flooded with suggestions. Or these suggestions could take hours of days to come in. It seems all up to chance sometimes. Still, generally speaking, most problems with Linux have been encountered and documented. So chances are good you'll find your solution fairly quickly.

On the other side of the coin is support for Windows. Yes, you can go the same route with Microsoft and depend upon your peers for solutions. There are just as many help sites/lists/forums for Windows as there are for Linux. And you can purchase support from Microsoft itself. Most corporate higher-ups easily fall victim to the safety net that having a support contract brings. But most higher-ups haven't had to depend up on said support contract.

4: Full vs. partial hardware support

One issue that is slowly becoming nonexistent is hardware support. Years ago, if you wanted to install Linux on a machine you had to make sure you hand-picked each piece of hardware or your installation would not work 100 percent.

With Windows, you know that most every piece of hardware will work with the operating system. Of course, there are times (and I have experienced this over and over) when you will wind up spending much of the day searching for the correct drivers for that piece of hardware you no longer have the install disk for. But you can go out and buy that 10-cent Ethernet card and know it'll work on your machine (so long as you have, or can find, the drivers). You also can rest assured that when you purchase that insanely powerful graphics card, you will probably be able to take full advantage of its power.

5: Centralized vs. Non centralized application installation

The heading for this point might have thrown you for a loop. But let's think about this for a second. With Linux you have (with nearly every distribution) a centralized location where you can search for, add, or remove software. such as Synaptic. With Synaptic, you can open up one tool, search for an application (or

group of applications), and install that application without having to do any Web searching (or purchasing).

Windows has nothing like this. With Windows, you must know where to find the software you want to install, download the software (or put the CD into your machine), and run setup.exe or install.exe with a simple double-click. For many years, it was thought that installing applications on Windows was far easier than on Linux. And for many years, that thought was right on target. Not so much now. Installation under Linux is simple, painless, and centralized.

DIFFERENCE BETWEEN LINUX AND UNIX

Linux	Unix
Linux is an example of Open Source software development and Free Operating System (OS).	Unix is an operating system that is very popular in universities, companies, big enterprises etc.
Cost: Linux can be freely distributed, downloaded freely, distributed through magazines, Books etc. There are priced versions for Linux also, but they are normally cheaper than Windows.	Cost: Different flavours of Unix have different cost structures
Price: Free but support is available for a price.	Price: Some free for development use (Solaris) but support is available for a price.
User: Everyone From home users to developers and computer enthusiasts alike.	User: Unix operating systems were developed mainly for mainframes, servers and workstations. The Unix environment and the client-server program model were essential elements in the development of the Internet
Manufacturer: Linux kernel is developed by the community. Linus Torvalds oversees things.	Manufacturer: Three bigest distributions are Solaris (Oracle), AIX (IBM) & HP-UX Hewlett Packard.
Usage: Linux can be installed on a wide variety of computer hardware, ranging from mobile phones, tablet computers and video game consoles, to mainframes and supercomputers.	Usage: The UNIX operating system is used in internet servers, workstations & PCs. Backbone of the majority of finance infastructure and many 24x365 high availability solutions.
Processors: Dozens of different	Processors: x86/x64, Sparc, Power,

kinds.	Itanium, PA-RISC and many others.
Architectures: Originally developed for Intel's x86 hardware, ports available for over two dozen CPU types including ARM	Processors: is available on PA-RISC and Itanium machines. Solaris also available for x86/x64 based systems.
Killer Features: Ksplice - kernel update without reboot	Killer Features: ZFS - Next generation filesystemDtrace - dynamic kernel tracing
GUI: Linux typically provides two GUIs, KDE and Gnome. But Linux GUI is optional.	GUI: Initially Unix was a command based OS, but later a GUI was created called Common Desktop Environment. Most distributions now ship with Gnome.
File System Support: Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, FAT32, NTFS	File System Support: jfs,gpfs,hfs,ufs,xfs,zfs format
Text mode interface: BASH (Bourne Again SHell) is the Linux default shell. It can support multiple command interpreters.	Text mode interface: Originally the Bourne Shell. Now it's compatible with many others including BASH, Korn& C.
Security: Linux has had about 60-100 viruses listed till date. None of them actively spreading nowadays.	Security: A rough estimate of UNIX viruses is between 85 -120 viruses reported till date.
Development and Distribution: Linux is developed by Open Source development i.e. through sharing and collaboration of code and features through forums etc and it is distributed by various vendors such as Debian, Red Hat, SUSE, Ubuntu,	Development and Distribution: Unix systems are divided into various other flavors, mostly developed by AT&T as well as various commercial vendors and non-profit organizations.

GentuX etc.	
Threat detection and solution: In case of Linux, threat detection and solution is very fast, as Linux is mainly community driven and whenever any Linux user posts any kind of threat, several developers start working on it from different parts of the world	Thread detection and solution: Because of the proprietary nature of the original Unix, users has to wait for a while, to get the proper bug fixing patch. But these are not as common.
Inception: Inspired by MINIX (a Unix-like system) and eventually after adding many features of GUI, Drivers etc, Linus Torvalds developed the framework of the OS that became LINUX in 1992. The LINUX kernel was released on 17th September, 1991	Inception: In 1969, it was developed by a group of AT&T employees at Bell Labs and Dennis Ritchie. It was written in "C" language and was designed to be a portable, multi-tasking and multi-user system in a time-sharing configuration.

LINUX distributions

A Linux distribution is a collection of (usually open source) software on top of a Linux kernel. A distribution (or short, distro) can bundle server software, system management tools, documentation and many desktop applications in a central secure software repository. A distro aims to provide a common look and feel, secure and easy software management and often a specific operational purpose.

Some popular distributions are as follows:

Red Hat

Red Hat is a billion dollar commercial Linux Company that puts a lot of effort in developing Linux. They have hundreds of Linux specialists and are known for their excellent support. They give their products (Red Hat Enterprise Linux and Fedora) away for free. While Red Hat Enterprise Linux (RHEL) is well tested before release and supported for up to seven years after release, Fedora is a distro with faster updates but without support.

Ubuntu

Canonical started sending out free compact discs with Ubuntu Linux in 2004 and quickly became popular for home users (many switching from Microsoft Windows). Canonical wants Ubuntu to be an easy to use graphical Linux desktop without need to ever see a command line. Of course they also want to make a profit by selling support for Ubuntu.

Debian

There is no company behind Debian. Instead there are thousands of well organised developers that elect a Debian Project Leader every two years. Debian is seen as one of the most stable Linux distributions. It is also the basis of every release of Ubuntu.

Debian comes in three versions: *stable, testing and unstable.*

Other

Distributions like CentOS, Oracle Enterprise Linux and Scientific Linux are based on Red Hat Enterprise Linux and share many of the same principles, directories and system administration techniques. Linux Mint, Edubuntu and many other distributions are based on Ubuntu and thus share a lot with Debian.

LINUX IS VIRUS PROOF

No Operating system on this earth can be ever be 100% immune to Viruses and Malware. But still Linux never had a widespread malware-infection as compared to Windows. Some people believe that Linux still has a minimal usages share, and a Malware is aimed for mass destruction. No programmer will give his valuable time, to code day and night for such group and hence Linux is known to have little or no viruses. Linux should be the primary target of Malware infection because more than 90% of high end server runs on Linux today.

Destroying or infecting one server means collapse of thousands of computer and then Linux would have been the soft target of hackers. So certainly usages share ratio is not in consideration for the above said fact.

Linux is architecturally strong and hence very much immune (not totally) to security threats. Linux is Kernel and GNU/Linux is the OS. There are hundreds of distributions of Linux. At Kernel Level they all are more or less the same but not at the OS Level.

If root password is confidential and strong enough, the OS is literally secure. You can not set a Linux System without setting up root password and user password. It means every user in a Linux System must have a password except 'Guest'. Whereas Windows allow you to set user and even root account without password. A user cannot run a program be it install/uninstall without permission provided or root password. But this is not the case with Windows.

Linux is so much secure in architecture that you even don't need to go behind a firewall until you are on Network. The access control Security Policy in Linux that is called Security-Enhanced Linux (SELinux) is a set of Kernel modification and user-space tools which implement security policies in a Linux system. Even SELinux is not must for normal users however it is important for users on network and Administrators.

Some Known Linux Threats are:

Tron horses
Local Scripts
Web Scripts
Worms
Targeted Attacks
Rootkits, etc.

These days a new trend of cross platform viruses is getting common. Some of the measures one should implement, for Linux System protection:

Protect boot loader
Encrypt Disk
Check rootkits on regular basis

Protect Root with strong Password
Provide correct permission to files
Provide proper roles to users
Use Antivirus
Go behind a Firewall
Don't keep un-necessary packages and programs (It may result into security Flaw).

PROS AND CONS OF LINUX

List of Pros of Linux

1. **Linux is Easy to Use.**

Linux is an easy operating system to use. It is ideal in a business setting. You can use Linux as your server system and reap the benefits of a low cost operating system.

2. **Linux is Super Compatible**.

Linux is compatible with many different types of software. Linux is an extremely adaptable operating system.

3. **Very Stable Environment**.

Linux is graded very stable by experts. It does not need periodic reboots to make it function at its optimal level. This is a nice switch from other operating systems that stall, get hung up and need a reboot.

5. **It'sFast.**

Linux is great at multi-tasking. It can handle a pretty heavy workload and not lose any of its speed. It is a fast system that does not disappoint.

6. **Update Less.**

You do not have to spend hours updating Linux. It is ready to rock and roll and if there are any needed patches or repairs they are very quick to install.

7. **Very Secure.**
Linux is not plagued by viruses like other operating systems tend to be. There are built in safety features that keep all the bad stuff out.

8. **No Licensing or Registration Needed.**

You do not have to deal with registering Linux or getting a license key to use it since it is largely freeware.

9. **You Can Run It With Windows.**

You can run Linux with Windows you just have to add a partition.

List of Cons of Linux

1. **Advanced Software**
Linux does not have the ability to run some advanced financial programs and some other multimedia files. It can be a source of frustration but if you do not typically use these types of applications than it will not actually apply to you.

2. **You Can Not Run some MS Applications.**
You cannot just plug and play some MS applications but you can download an emulator that will help you with MS applications. This can be a bit of a barrier but it is not something that cannot be overcome.

Overall

Linux is a secure, fast, cost effective operating system that can meet a host of needs. While you can easily download it for free from many different sites you may want to consider paying for it because you will have the support available to you that you will need should something not work right.

It is a good choice in a host of situations. It is a cost effective option that can easily enhance your computing experiences. Linux is not going anywhere it is constantly being looked at as a potential replacement for other operating systems that cannot bode as well when it comes to security and ease of use.

UNDERSTANDING FILES AND DIRECTORIES IN LINUX

General overview of the Linux file system

Files:

A simple description of the UNIX system, also applicable to Linux, is this:

"On a UNIX system, everything is a file; if something is not a file, it is a process."

This statement is true because there are special files that are more than just files (named pipes and sockets, for instance), but to keep things simple, saying that everything is a file is an acceptable generalization.

A Linux system, just like UNIX, makes no difference between a file and a directory, since a directory is just a file containing names of other files. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files, according to the system.

In order to manage all those files in an orderly fashion, man likes to think of them in an ordered tree-like structure on the hard disk, as we know from MS-DOS (Disk Operating System) for instance. The large branches contain more branches, and the branches at the end contain the tree's leaves or normal files.

Sorts of files

Most files are just files, called *regular* files; they contain normal data, for example text files, executable files or programs, input for or output from a program and so on.

While it is reasonably safe to suppose that everything you encounter on a Linux system is a file, there are some exceptions.

- *Directories*: files that are lists of other files.
- *Special files*: the mechanism used for input and output. Most special files are in /dev.
- *Links*: a system to make a file or directory visible in multiple parts of the system's file tree.
- *(Domain) sockets*: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.

- *Named pipes*: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

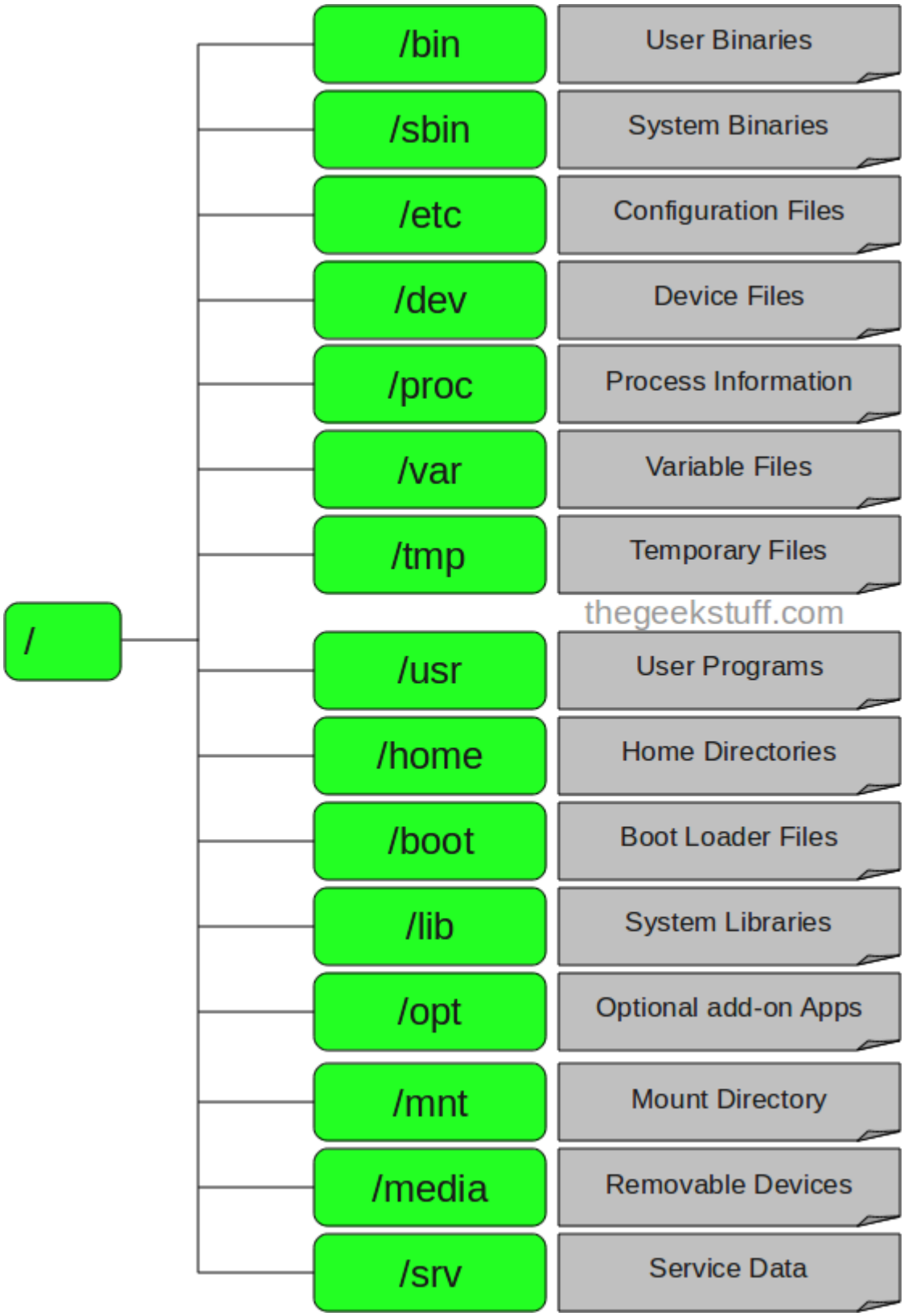
LIST OF FILE TYPES

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

In order not to always have to perform a long listing for seeing the file type, a lot of systems by default don't issue just ls, but ls -F, which suffixes file names with one of the characters "/=*|@" to indicate the file type.

As a user, you only need to deal directly with plain files, executable files, directories and links. The special file types are there for making your system do what you demand from it and are dealt with by system administrators and programmers.

Linux Directory Structure (File System Structure) Explained with Examples



1. / – Root

- Every single file and directory starts from the root directory.
- Only root user has write privilege under this directory.

2. /bin – User Binaries

- Contains binary executables.
- Common linux commands you need to use in single-user modes are located under this directory.
- Commands used by all the users of the system are located here.
- For example: ps, ls, ping, grep, cp.

3. /sbin – System Binaries

- Just like /bin, /sbin also contains binary executables.
- But, the linux commands located under this directory are used typically by system aministrator, for system maintenance purpose.
- For example: iptables, reboot, fdisk, ifconfig, swapon

4. /etc – Configuration Files

- Contains configuration files required by all programs.
- This also contains startup and shutdown shell scripts used to start/stop individual programs.
- For example: /etc/resolv.conf, /etc/logrotate.conf

5. /dev – Device Files

- Contains device files.
- These include terminal devices, usb, or any device attached to the system.
- For example: /dev/tty1, /dev/usbmon0

6. /proc – Process Information

- Contains information about system process.
- This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
- This is a virtual filesystem with text information about system resources. For example: /proc/uptime

7. /var – Variable Files

- var stands for variable files.

- Content of the files that are expected to grow can be found under this directory.
- This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);

8. /tmp – Temporary Files

- Directory that contains temporary files created by system and users.
- Files under this directory are deleted when system is rebooted.

9. /usr – User Programs

- Contains binaries, libraries, documentation, and source-code for second level programs.
- /usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
- /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
- /usr/lib contains libraries for /usr/bin and /usr/sbin
- /usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2

10. /home – Home Directories

- Home directories for all users to store their personal files.
- For example: /home/john, /home/sunny

11. /boot – Boot Loader Files

- Contains boot loader related files.
- Kernel initrd, vmlinux, grub files are located under /boot
- For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

12. /lib – System Libraries

- Contains library files that supports the binaries located under /bin and /sbin
- Library filenames are either ld* or lib*.so.*
- For example: ld-2.11.1.so, libncurses.so.5.7

13. /opt – Optional add-on Applications

- opt stands for optional.

- Contains add-on applications from individual vendors.
- add-on applications should be installed under either /opt/ or /opt/ sub-directory.

14. /mnt – Mount Directory

- Temporary mount directory where sysadmins can mount filesystems.

15. /media – Removable Media Devices

- Temporary mount directory for removable devices.
- For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

16. /srv – Service Data

- srv stands for service.
- Contains server specific services related data.
- For example, /srv/cvs contains CVS related data.

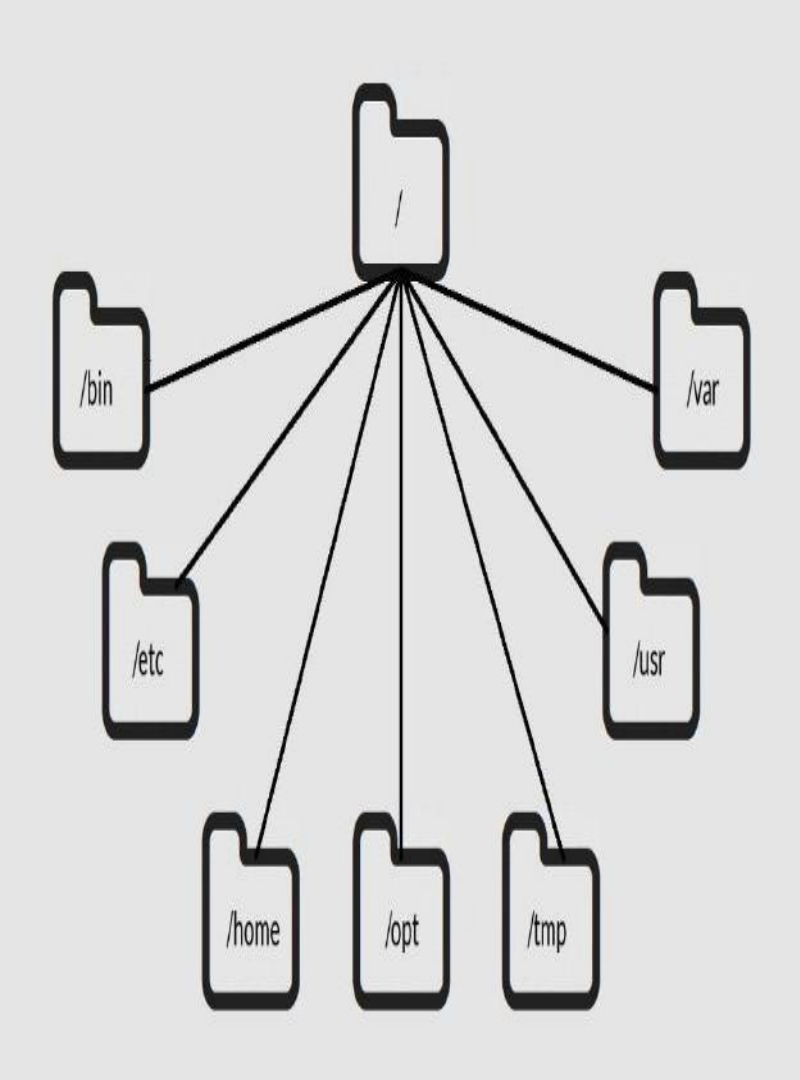
Linux Directory Structure and File System Hierarchy

Common Top Level Directories

Here are the most common top level directories that you need to be aware of and may interact with as a user of a Linux system.

Dir	Description
/	The directory called “root.” It is the starting point for the file system hierarchy. Note th not related to the root, or superuser, account.
/bin	Binaries and other executable programs.

Dir	Description
/etc	System configuration files.
/home	Home directories.
/opt	Optional or third party software.
/tmp	Temporary space, typically cleared on reboot.
/usr	User related programs.
/var	Variable data, most notably log files.


/
/bin
/var
/etc
/usr
/home
/opt
/tmp

Linux Folders

The Linux directory structure is like a tree. The base of the Linux file system hierarchy begins at the root. Directories branch off the root, but everything starts at root.

The directory separator in Linux is the forward slash (/). When talking about directories and speaking directory paths, "forward slash" is abbreviated to "slash." Often the root of the file system is referred to as "slash" since the full path to it is /. If you hear someone say "look in slash" or "that file is in slash," they are referring to the root directory.

The /bin directory is where you will find binary or executable files. Programs are written in source code which is human readable text. Source code is then compiled into machine readable binaries. They are called binaries because machine code is a series of zeros and ones. The important thing to know is that commands, programs, and applications that you can use are sometimes located in /bin.

Configuration files live in the /etc directory. Configuration files control how the operating system or applications behave. For example, there is a file in /etc that tells the operating system whether to boot into a text mode or a graphical mode.

User home directories are located in /home. If your account name is "pat" your home directory will be /home/pat. Linux systems can and often do have multiple user accounts. Home directories allow each user to separate their data from the other users on the system. The patdirectory is known as a subdirectory. A subdirectory is simply a directory that resides inside another directory.

The /opt directory houses optional or third party software. Software that is not bundled with the operating system will often been installed in /opt. For example, the Google Earth application is not part of the standard Linux operating system and gets installed in the /opt/google/earthdirectory.

Temporary space is allocated in /tmp. Most Linux distributions clear the contents of /tmp at boot time. Be aware that if you put files in /tmp and the Linux system reboots, your files will more than likely be gone. The /tmp directory is a great place to store temporary files, but do not put anything in /tmp that you want to keep long term.

The /usr directory is called “user.” You will find user related binary programs and executables in the /usr/bin directory.

Variable data such as log files reside in /var. Specifically, the /var/log directory contains logs generated by the operating system and other applications.

Understanding Linux File Permissions

Although there are already a lot of good security features built into Linux-based systems, one very important potential vulnerability can exist when local access is granted - - that is file permission based issues resulting from a user not assigning the correct permissions to files and directories. So based upon the need for proper permissions, I will go over the ways to assign permissions and show you some examples where modification may be necessary.

Basic File Permissions

Permission Groups

Each file and directory has three user based permission groups:

- owner - The Owner permissions apply only the owner of the file or directory, they will not impact the actions of other users.
- group - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
- all users - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Permission Types

Each file or directory has three basic permission types:

- read - The Read permission refers to a user's capability to read the contents of the file.
- write - The Write permissions refer to a user's capability to write or modify a file or directory.
- execute - The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Viewing the Permissions

You can view the permissions by checking the file or directory permissions in your favorite GUI File Manager (which I will not cover here) or by reviewing the output of the *\"ls -l\"* command while in the terminal and while working in the directory which contains the file or folder.

The permission in the command line is displayed as: *_rwxrwxrwx 1 owner:group*

1. User rights/Permissions
    1. The first character that I marked with an underscore is the special permission flag that can vary.
    2. The following set of three characters (rwx) is for the owner permissions.
    3. The second set of three characters (rwx) is for the Group permissions.
    4. The third set of three characters (rwx) is for the All Users permissions.
2. Following that grouping since the integer/number displays the number of hardlinks to the file.
3. The last piece is the Owner and Group assignment formatted as Owner:Group.

Explicitly Defining Permissions

To explicity define permissions you will need to reference the Permission Group and Permission Types.

The Permission Groups used are:

- u - Owner
- g - Group
- o or a - All Users

The potential Assignment Operators are + (plus) and - (minus); these are used to tell the system whether to add or remove the specific permissions.

The Permission Types that are used are:

- r - Read
- w - Write
- x - Execute

So for an example, lets say I have a file named file1 that currently has the permissions set to _rw_rw_rw, which means that the owner, group and all users have read and write permission. Now we want to remove the read and write permissions from the all users group.

To make this modification you would invoke the command: *chmod a-rw file1* To add the permissions above you would invoke the command: *chmoda+rw file1*

As you can see, if you want to grant those permissions you would change the minus character to a plus to add those permissions.

Root

Definition:

Root is the user name or account that by default has access to all commands and files on a Linux or other Unix-like operating system. It is also referred to as the root account, root user and the super user.

The word root also has several additional, related meanings when used as part of other terms, and thus it can be a source of confusion to people new to Unix/Linux-like systems.

One of these is the root directory, which is the top level directory on a system. That is, it is the directory in which all other directories, including their sub-directories, and files reside. The root directory is designated by a forward slash (/).

Another is /root (pronounced slash root), which is the root user's home directory. A home directory is the primary repository of a user's files, including that user's configuration files, and it is usually the directory in which a user finds itself when it logs into a system. /root is a subdirectory of the root directory, as indicated by the forward slash that begins its name, and should not to be confused with that directory. Home directories for users other than root are by default created in the /home directory, which is another standard subdirectory of the root directory.

The root account is the most privileged on the system and has absolute power over it (i.e., complete access to all files and commands). Among

root's powers are the ability to modify the system in any way desired and to grant and revoke access permissions (i.e., the ability to read, modify and execute specific files and directories) for other users, including any of those that are by default reserved for root.

A root-kit is a set of software tools secretly installed by an intruder into a computer that allows such intruder to use that computer for its own, usually nefarious, purposes when desired. Well designed root-kits are able to obtain root access (i.e., access to the root account rather than just to a user account) and to hide most or all traces of their presence and activities.

The use of the term root for the all-powerful administrative user may have arisen from the fact that root is the only account having write permissions (i.e., permission to modify files) in the root directory. The root directory, in turn, takes its name from the fact that the file systems (i.e., the entire hierarchy of directories that is used to organize files) in Unix-like operating systems have been designed with a tree-like (although inverted) structure in which all directories branch off from a single directory that is analogous to the root of a tree.

The original UNIX/Linux operating system, on which Linux and other Unix-like systems are based, was designed from the very beginning as a multi-user system because personal computers did not yet exist and each user was connected to the mainframe computer (i.e., a large, centralized computer) via a dumb (i.e., very simple) terminal. Thus it was necessary to have a mechanism for separating and protecting the files of the individual users while allowing them to use the system simultaneously. It was also necessary to have a means for enabling a system administrator to perform such tasks as entering user directories and files to correct individual problems, granting and revoking powers for ordinary users, and accessing critical system files to repair or upgrade the system.

Every user account is automatically assigned an identification number, the UID (i.e., user ID), by a Unix-like system, and the system uses these numbers instead of the user names to identify and keep track of the users. Root always has a UID of zero. This can be verified by logging in as root (if using a home computer or other system that permits this operation) and running the echo-command to display the UID of the current user, i.e.,

```
echo $UID
```

echo is used to repeat on the screen what is typed in after it. The dollar sign

preceding UID tells echo to display its value rather than its name.

The UID for root (as well as for all other users) can also be seen by looking at /etc/passwd, which is the configuration file for user data. This file can be viewed (by default by all users) by using the cat command (which is commonly employed to read files), i.e.,

cat /etc/passwd | less

The output of cat /etc/passwd in this example is piped (i.e., transferred) to the less command to allow it to be read one screenful at a time, which is useful if the file is a long one. The line of output for root will look something like root:x:0:0:root:/root:/bin/bash. The first column shows the user name and the third column shows the UID, which can be seen to be zero.

The permissions system in Unix-like operating systems is set by default to prevent access by ordinary users to critical parts of the system and to files and directories belonging to other users. Thus, it can be very tempting for users new to such systems, especially those who are accustomed to systems with a weak permissions system or without any permissions system (e.g., Microsoft Windows or the older versions of the Macintosh), to bypass this permissions system on their personal computers by logging directly into the root account and staying there. Although this provides momentary relief, it should be avoided and ordinary work on the system should be done via an ordinary user account.

This is because it is very easy to damage a Unix-like system when using it as root -- much easier than to damage most other types of operating systems. The designers of most other operating systems devised methods of protecting the system and data to compensate for the lack of a robust permissions system.

However, an important principle of Unix-like operating systems is the provision of maximum flexibility to configure the system, and thus the root user is fully empowered. Unix-like systems assume that the system administrator knows exactly what he or she is doing and that only such individual(s) will be using the root account. Thus, there is virtually no safety net for the root user in the event of a careless error, such as damaging or deleting a critical system file (which could make the entire system inoperable).

Adding to the danger of routinely using the system as root is the fact that

all processes (i.e., instances of programs in execution) started by the root user have root privileges. Because even the most widely used and well-tested application programs contain numerous programming errors (due to the huge amount of code required and its great complexity), a skilled attacker can often find and exploit such an error to obtain control of a system when a program is run with root privileges rather than using an ordinary user account, with its very limited privileges.

A critical means for preventing users from directly damaging Unix-like systems or increasing the vulnerability of such systems to damage by others is the avoidance of using the root account except when absolutely necessary, even by knowledgeable and experienced system administrators. That is, rather than routinely logging into the system as root, administrators should log in with their ordinary user accounts and then use commands, such as su, kdesu and sudo, that provide them with root privileges only as needed and without requiring a new login.

For example, to become root with su merely requires typing su at the command line (i.e., in the all-text mode), pressing the Enter key and supplying the root password. The account of the previous user can be returned to by pressing the Ctrl and d keys simultaneously or by typing the word exit and then pressing the Enter key.

The security associated with using su can be increased by using its -c option(su –c) which terminates it and causes an immediate return to the former user account after the current command has completed execution or after any program that it has launched has been closed.

Tasks that require root privileges include moving files or directories into or out of system directories (i.e., directories that are critical to the functioning of the operating system), copying files into system directories, granting or revoking user privileges, some system repairs, and the installation of some application programs. By default, it is not necessary to be root to be able to read most configuration files and documentation files in system directories, although it is necessary to be root to modify them.

Root privileges are usually required for installing software in RPM (Red Hat Package Manager) package format because of the need to write to system directories. If an application program is being compiled (i.e., converted into runnable form) from source code (i.e., its original, human-readable form), however, it can usually be configured to install and run from a user's home directory. Root privileges are not needed by an

ordinary user to compile and install software in its home directory.

Shell in Linux

A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. Its primary function is to read commands that are typed into a console (i.e., an all-text display mode) or terminal window (an all-text window) in a GUI (graphical user interface) and then execute (i.e., run) them.

The term shell derives its name from the fact that it is an outer layer of an operating system. A shell is an interface between the user and the internal parts of the operating system (at the very core of which is the kernel).

A user is in a shell (i.e., interacting with the shell) as soon as that user has logged into the system. A shell is the most fundamental way that a user can interact with the system, and the shell hides the details of the underlying operating system from the user.

A shell prompt, also referred to as a command prompt is a character or set of characters at the start of the command line that indicates that the shell is ready to receive commands. It usually is, or ends with, a dollar sign ($) for ordinary users and a pound sign (#) for the root (i.e., administrative) user. The term command line is sometimes used interchangeably with the shell prompt, because that is where the user enters commands. For example, instructions for performing some activity might say "Enter the following at the command line," which is the same as saying "Enter the following at the shell prompt." However, a command line is not a program but rather just the space to the right of a shell prompt.

Shells, although small in size, are sophisticated and powerful programs that are used for the following: program execution (i.e., launching), substitution of variables and of file names, input/output (I/O), redirection (i.e., sending the

output from a program to a destination other than its default destination, including to be used as the input for another program), user environment control (e.g., changing the shell or the shell prompt), and serving as a programming language (i.e., a language that can be used to write shell scripts). Shells in Unix-like operating systems are unusual in that they are both an interactive command language (i.e., a language that a user can employ interactively to issue commands) and a programming language.

On systems with GUIs, many users rarely interact with the shell directly. However, GUIs are merely front ends for shells; that is, they are merely attractive interface layers that are built on top of a shell and which use the shell's commands. As shells are usually more powerful and feature-rich than GUIs, most intermediate and advanced users of Unix-like operating systems find them to be preferable to GUIs for many tasks.

A number of different shells have been developed for Unix-like operating systems. They share many similarities, but there are also some differences with regard to commands, syntax and functions that are important mainly for advanced users. Every Unix-like operating system has at least one built-in shell, and most have several. Some are discussed below:

sh (the Bourne Shell) is the original UNIX shell, and it is still in widespread use today. Written by **Stephen Bourne** at Bell Labs in 1974, it is a simple shell with a small size and few features, perhaps the fewest of any shell for a Unix-like operating system. Bell Labs was the research and development arm of AT&T (The American Telephone and Telegraph Company), the former U.S. telecommunications monopoly. The first version of UNIX was developed at Bell Labs in 1969.

Among the functions that most users have come to expect in a shell but which are missing in sh are file name completion, command editing, command history and ease of executing multiple background processes (also referred to as jobs). A process is an instance of an executing program; a background process operates generally unnoticed while users are working with foreground processes. File name completion is the completion by the system of the names of files that have only partially typed in by a user. Command history allows users to conveniently find and reissue previously issued commands. Every Unix-like system contains sh or another shell which incorporates its commands.

bash (Bourne-again shell) is the default shell on Linux. It also runs on nearly every other Unix-like operating system as well, and versions are also available for other operating systems including the Microsoft Windows systems. Bash is a superset of sh (i.e., commands that work in sh also work in bash, but the

reverse is not always true), and it has many more commands than sh, making it a powerful tool for advanced users. But it is also intuitive and flexible, and thus it is probably the most suitable shell for beginners. bash was written for the GNU project (whose goal is to develop a complete, Unix-compatible, high performance and entirely free operating system), primarily by **Brian Fox and Chet Ramey**. Its name is a pun on the name of Steve Bourne.

Csh (the C shell) has a syntax that resembles that of the highly popular C programming language (also developed at Bell Labs), and thus it is sometimes preferred by programmers. It was created in 1978 by **Bill Joy** (who also wrote the vi text editor and later co-founded Sun Microsystems) at the University of California at Berkeley (UCB).

ksh (the Korn shell) is a superset of sh developed by **David Korn** at Bell Labs in 1983. It contains many features of the C shell as well, including a command history, which was inspired by the requests of Bell Labs users. It also features built-in arithmetic evaluation and advanced scripting capabilities similar to those found in powerful programming languages such as awk, sed and perl.

tcsh (the TENEX C shell) is based on csh but also has programmable file name completion, command line editing, a command history mechanism and other features lacking in csh. It is named after the TENEX operating system, which inspired the author of tcsh. tcsh replaced the csh as the default shell on some BSD operating systems (i.e., FreeBSD and Darwin).

zsh (the Z shell) is similar to ksh, but it also includes many features from csh. That is, it attempts to combine the programmability and syntax of the Korn shell with useful features from the C shell (which has some disadvantages as a programming language). It was written by Paul Falstad around 1990.

The great flexibility of shells on Unix-like operating systems is further enhanced by the fact that it is extremely easy to change the current shell. The shell for any user can be switched temporarily, or that user's default shell can be changed.

Some other families of operating systems also have shells. For example, MS-DOS became the shell on earlier versions of Microsoft Windows. However, it was replaced with a shell emulator on later versions (e.g., Windows 2000 and Windows XP), apparently because of the Microsoft philosophy of attempting to make the GUI all-powerful and de-emphasizing the command line.

VI editor and commands associated with it

There are many ways to edit files in Linux and Unix like operating systems and one of the best ways is using screen-oriented text editor vi. This editor enable you to edit lines in context with other lines in the file.

Now a days you would find an improved version of vi editor which is called VIM. Here VIM stands for Vi Improved.

The vi is generally considered the de facto standard in Unix/Linux editors because −

- It's usually available on all the flavors of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user friendly than any other editors like ed or ex.

You can use vi editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

Starting the vi Editor

There are following way you can start using vi editor −

Command	Description
vi filename	Creates a new file if it already does not exist, otherwise opens existing file.
vi -R filename	Opens an existing file in read only mode.
view filename	Opens an existing file in read only mode.

Following is the example to create a new file testfile if it already does not exist in the current working directory –

```
$vitestfile
```

As a result you would see a screen something like as follows −

```
|
~
~
~
~
~
~
~
~
~
~
~
~
"testfile"[NewFile]
```

You will notice a tilde (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

So now you have opened one file to start with. Before proceeding further let us understanding few minor but important concepts explained below.

Operation Modes

While working with vi editor you would come across following two modes −

- Command mode − This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.
- Insert mode − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it

is put in the file .

The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode you simply type i. To get out of insert mode, press the Esc key, which will put you back into command mode.

If you are not sure which mode you are in, press the Esc key twice, and then you'll be in command mode. You open a file using vi editor and start type some characters and then come in command mode to understand the difference.

Getting Out of vi

The command to quit out of vi is **:q**. Once in command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is :q!. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is :w. You can combine the above command with the quit command, or :wq and return.

The easiest way to save your changes and exit out of vi is the ZZ command. When you are in command mode, type ZZ and it will do the equivalent of :wq.

You can specify a different file name to save to by specifying the name after the :w. For example, if you wanted to save the file you were working as another filename called filename2, you would type :w filename2 and return.

Moving within a File

To move around within a file without affecting your text, you must be in command mode (press Esc twice). Here are some of the commands you can use to move around one character at a time −

Command	Description
K	Moves the cursor up one line.
J	Moves the cursor down one line.

H	Moves the cursor to the left one character position.
L	Moves the cursor to the right one character position.

There are following two important points to be noted −

- The Vi is case-sensitive, so you need to pay special attention to capitalization when using commands.
- Most commands in vi can be prefaced by the number of times you want the action to occur. For example, 2j moves cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Here are some more commands you can use to move around the file −

Command	Description
0 or |	Positions cursor at beginning of line.
$	Positions cursor at end of line.
W	Positions cursor to the next word.
B	Positions cursor to previous word.
(	Positions cursor to beginning of current sentence.
)	Positions cursor to beginning of next sentence.
E	Move to the end of Blank delimited word
{	Move a paragraph back

}	Move a paragraph forward
[[	Move a section back
]]	Move a section forward
n|	Moves to the column n in the current line
1G	Move to the first line of the file
G	Move to the last line of the file
nG	Move to nth line of the file
:n	Move to nth line of the file
Fc	Move forward to c
Fc	Move back to c
H	Move to top of screen
nH	Moves to nth line from the top of the screen
M	Move to middle of screen
L	Move to botton of screen
nL	Moves to nth line from the bottom of the screen

:x	Colon followed by a number would position the cursor on line number represented by x

Control Commands

There are following useful command which you can use along with Control Key −

Command	Description
CTRL+d	Move forward 1/2 screen
CTRL+f	Move forward one full screen
CTRL+u	Move backward 1/2 screen
CTRL+b	Move backward one full screen
CTRL+e	Moves screen up one line
CTRL+y	Moves screen down one line
CTRL+u	Moves screen up 1/2 page
CTRL+d	Moves screen down 1/2 page
CTRL+b	Moves screen up one page
CTRL+f	Moves screen down one page
CTRL+I	Redraws screen

Editing Files

To edit the file, you need to be in the insert mode. There are many ways to enter insert mode from the command mode −

Command	Description
I	Inserts text before current cursor location.
I	Inserts text at beginning of current line.
A	Inserts text after current cursor location.
A	Inserts text at end of current line.
O	Creates a new line for text entry below cursor location.
O	Creates a new line for text entry above cursor location.

Deleting Characters

Here is the list of important commands which can be used to delete characters and lines in an opened file −

Command	Description
X	Deletes the character under the cursor location.
X	Deletes the character before the cursor location.
Dw	Deletes from the current cursor location to the next word.
d^	Deletes from current cursor position to the beginning of the line.

d$	Deletes from current cursor position to the end of the line.
D	Deletes from the cursor position to the end of the current line.
Dd	Deletes the line the cursor is on.

As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, 2x deletes two character under the cursor location and 2dd deletes two lines the cursor is on.

I would highly recommend to exercise all the above commands properly before proceeding further.

Change Commands

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

Command	Description
Cc	Removes contents of the line, leaving you in insert mode.
Cw	Changes the word the cursor is on from the cursor to the lowercase w end of the word.
R	Replaces the character under the cursor. vi returns to command mode after the replacement is entered.
R	Overwrites multiple characters beginning with the character currently under the cursor. You must use Esc to stop the overwriting.
S	Replaces the current character with the character you type. Afterward, you are left in insert mode.

S	Deletes the line the cursor is on and replaces with new text. After the new text is entered, vi remains in insert mode.

Copy and Paste Commands

You can copy lines or words from one place and then you can past them at another place using following commands −

Command	Description
Yy	Copies the current line.
Yw	Copies the current word from the character the lowercase w cursor is on until the end of the word.
P	Puts the copied text after the cursor.
P	Puts the yanked text before the cursor.

Advanced Commands

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

Command	Description
J	Join the current line with the next one. A count joins that many lines.
<<	Shifts the current line to the left by one shift width.
>>	Shifts the current line to the right by one shift width.
~	Switch the case of the character under the cursor.

^G	Press CNTRL and G keys at the same time to show the current filename and the status.
U	Restore the current line to the state it was in before the cursor entered the line.
U	Undo the last change to the file. Typing 'u' again will re-do the change.
J	Join the current line with the next one. A count joins that many lines.
:f	Displays current position in the file in % and file name, total number of file.
:f filename	Renames current file to filename.
:w filename	Write to file filename.
:e filename	Opens another file with filename.
:cd dirname	Changes current working directory to dirname.
:e #	Use to toggle between two opened files.
:n	In case you open multiple files using vi, use :n to go to next file in the series.
:p	In case you open multiple files using vi, use :p to go to previous file in the series.
:N	In case you open multiple files using vi, use :N to go to

	previous file in the series.
:r file	Reads file and inserts it after current line
:nr file	Reads file and inserts it after line n.

Word and Character Searching

The vi editor has two kinds of searches: string and character. For a string search,**the / and ?** commands are used. When you start these commands, the command just typed will be shown on the bottom line, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place –

- The / command searches forwards (downwards) in the file.
- The ?command searches backwards (upwards) in the file.

The n and N commands repeat the previous search command in the same or opposite direction, respectively. Some characters have special meanings while using in search command and preceded by a backslash (\) to be included as part of the search expression.

Character	Description
^	Search at the beginning of the line. (Use at the beginning of a search expression.)
.	Matches a single character.
*	Matches zero or more of the previous character.
$	End of the line (Use at the end of the search expression.)

[	Starts a set of matching, or non-matching expressions.
<	Put in an expression escaped with the backslash to find the ending or beginning of a word.
>	See the '<' character description above.

The character search searches within one line to find a character entered after the command. The f and F commands search for a character on the current line only. f searches forwards and F searches backwards and the cursor moves to the position of the found character.

The t and T commands search for a character on the current line only, but for t, the cursor moves to the position before the character, and T searches the line backwards to the position after the character.

Set Commands

You can change the look and feel of your vi screen using the following :set commands. To use these commands you have to come in command mode then type :set followed by any of the following options −

Command	Description
:set ic	Ignores case when searching
:set ai	Sets autoindent
:set noai	To unset autoindent.
:set nu	Displays lines with line numbers on the left side.
:set sw	Sets the width of a software tabstop. For example you would set a shift width of 4 with this command: :set sw=4

:set ws	If *wrapscan* is set, if the word is not found at the bottom of the file, it will try to search for it at the beginning.
:set wm	If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: :set wm=2
:set ro	Changes file type to "read only"
:set term	Prints terminal type
:set bf	Discards control characters from input

Running Commands

The vi has the capability to run commands from within the editor. To run a command, you only need to go into command mode and type :! command.

For example, if you want to check whether a file exists before you try to save your file to that filename, you can type :! ls and you will see the output of ls on the screen.

When you press any key (or the command's escape sequence), you are returned to your vi session.

Replacing Text

The substitution command (:s/) enables you to quickly replace words or groups of words within your files. Here is the simple syntax −

```
:s/search/replace/g
```

The g stands for globally. The result of this command is that all occurrences on the cursor's line are changed.

UNIT III

BASIC LINUX COMMANDS

touch
touch command is used to create an empty file
syntax: $ touch filename
example:$ touch ABC

mkdir
It is commonly used directory command. It is a counter part of MD or MKDIR of DOS.
Syntax: $mkdir directory name
Example: $mkdir book
The above command creates a directory named book.
Among the options available with mkdir is –p, which allows you to create multiple generations of directories, at one go. That means it creates all the parent directories specified in the given path too.
For example:-
$ mkdir –p works/bpb/Linux/book
The –p option tells Linux to first create works ,then within it bpb, next is child directory Linux and lastly book.

rmdir
Delete the named directories (not the contents). directories are deleted from the parent directory and must be empty (if not, rm -r can be used instead).
Syntax: rmdir directory name
Example: rmdir book

ls
This command is used to list all files and directories.
Syntax:$ ls
$ls –a:to list hidden files

Pwd: print working command is used to display the name of current working directory.

Syntax:
Pwd

cd
we can change your current directory with the cd command (Change Directory).

paul@laika$ cd /etc

```
paul@laika$ pwd
/etc
paul@laika$ cd /bin

paul@laika$ pwd
/bin
paul@laika$ cd /home/paul/
paul@laika$ pwd
/home/paul
```

Chmod: it is used to change the permissions of a file.
syntax: $ chmod weight filename
Example : $ chmod 700 file 1

$chmod +w filename: to give write permissions to all i.e to user,group and others.
$chmod .
$chmod +r :to give read permissions to others for a file.

df: command displays the amount of disk space available on the file system
Syntax: #df

dd: dd is a command-line utility for Unix and Unix-like operating systems whose primary purpose is to convert and copy files.

On Unix, device drivers for hardware (such as hard disk drives) and special device files (such as /dev/zero and /dev/random) appear in the file system just like normal files; dd can also read and/or write from/to these files, provided that function is implemented in their respective driver. As a result, dd can be used for tasks such as backing up the boot sector of a hard drive, and obtaining a fixed amount of random data. The dd program can also perform conversions on the data as it is copied, including byte order swapping and conversion to and from the ASCII and EBCDIC text encodings.

ADD USER: The useradd command creates a new user account using the values specified on the command line and the default values from the system. The new user account will be entered into the system files (/etc/passwd) as needed, the home directory (/home/username) will be created, and initial files copied, depending on the command line options.

Syntax is as follows for useradd command:
useradd<username>

By default user account is locked, you need to setup a new password:
passwd<username>
For example add a new user called tom and set password to jerry:

```
# adduser tom
```

```
# passwd jerry
```

Passwd
Passwords of users can be set with the passwd command. Users will have to provide their old password before twice entering the new one.
For example
[harry@RHEL4 ~]$ passwd
Changing password for user harry.
Changing password for harry
(current) Linux password:
New Linux password:
BAD PASSWORD: it's WAY too short
New Linux password:
Retype new Linux password:
passwd: all authentication tokens updated successfully.
[harry@RHEL4 ~]$

date
The date command can display the date, time, time zone and more.
paul@rhel55 ~$ date
Sat oct 17 2016

cat: cat command is used to create a file and add contents to it.
Syntax: $ cat filename

$ cat>>filename : to add the contents to file while retaining its previous content.
$ cat filename> filename2: to copy the contents of file name1 to filename2.

tail: The tail command will display the last ten lines of a file.
syntax: tail filename
You can give tail the number of lines you want to see.
For example
$ tail -3 filename

wc: The wc command is used to count the number of lines, word, and characters for a specified file.

Syntax: $ wc filename

rm: The Linux rm command is used to remove files and directories.

Let's take a look at some rm command examples, starting from easy examples to more complicated examples.

Unix/Linux rm command examples - Deleting files

In its most basic use, the rm command can be used to remove one file, like this:

```
rm oldfile.txt
```

You can also use the rm command to delete multiple Linux files at one time, like this:

```
rm file1 file2 file3
```

If you prefer to be careful when deleting files, use the -i option with the rm command. The -i stands for "inquire", so when you use this option the rm command prompts you with a yes/no prompt before actually deleting your files:

```
rm -i files file2 file3
```

Linux rm command examples - Deleting directories

To delete Linux directories with the rm command, you have to specify the -r option, like this:

```
rm -r OldDirectory
```

The -r option means "recursive", and therefore this command recursively deletes all files and directories in the directory named *OldDirectory*.

cmp :is used to compare two files byte by byte. If a difference is found, it reports the byte and line number where the first difference is found. If no differences are found, by default, cmp returns no output.
cmp syntax

```
cmp [OPTION]... FILE1 [FILE2 [SKIP1 [SKIP2]]]
```

Options

The optional SKIP1 and SKIP2 specify the number of bytes to skip at the beginning of each file (zero by default).
SKIP values may be followed by the following multiplicative suffixes:

kB	kilobytes	1000
K	kibibytes	1024
MB	megabytes	1,000,000
M	mebibytes	1,048,576
GB	gigabytes	1,000,000,000
G	gibibytes	1,073,741,824

If a FILE is specified as '-' or not specified, data is read from standard input. cmp's exit status is 0 if inputs are the same, 1 if different, or 2 if the program encounters a problem.
cmp examples

```
cmp file1.txt file2.txt
```

Compares file1 to file2, reading each file byte-by-byte and comparing them until one of the byte pairs is not equal. When a difference is found, it will output the location in the file where the difference was found, and exit. Example output:
file.txt file2.txt differ: char 1011, line 112

diff : analyzes two files and prints the lines that are different. Essentially, it outputs a set of instructions for how to change one file in order to make it identical to the second file.

The sort Command

The sort command arranges lines of text alphabetically or numerically. The example below sorts the lines in the food file −

```
$sort food
kashmiriCuisine
BangkokWok
BigAppleDeli
Isle of Java
Mandalay
SushiandSashimi
SweetTooth
TioPepe's Peppers
$
```

The sort command arranges lines of text alphabetically by default. There are many options that control the sorting −

Option	Description
-n	Sort numerically (example: 10 will sort after 2), ignore blanks and tabs.

-r	Reverse the order of sort.
-f	Sort upper- and lowercase together.
+x	Ignore first x fields when sorting.

more

Displays text one screen at a time.
Description

more is a filter for paging through text one screen at a time. It does not provide as many options or enhancements as less, but is nevertheless quite useful and simple to use.
more syntax

more [-dlfpcsu] [-num*lines*] [+/*pattern*] [+*linenum*] [*file* ...]

Options

-num *lines*	Sets the number of lines that makes up a screenful. *lines* must be an integer.
-d	With this option, more will prompt the user with the message "[Press space to conti and display "[Press 'h' for instructions.]" when an illegal key is pressed, instead of r
-l	more usually treats ^L (CONTROL-L, the form feed) as a special character, and wi line that contains it. The -l option will prevent this behavior.
-f	Causes more to count logical, rather than screen lines (i.e., long lines are not wrapp
-p	Do not scroll. Instead, clear the whole screen and then display the text. This option automatically if the more executable is named page.

-c	Do not scroll. Instead, paint each screen from the top, clearing the remainder of eac displayed.
-s	Squeeze multiple blank lines into one blank line.
-u	Do not display underlines.
+/*string*	Search for the string *string*, and advance to the first line containing *string* when the
+*num*	Start displaying text at line number *num*.

chmod

chmod is used to change the permissions of files or directories.

Overview

On Linux and other Unix-like operating systems, there is a set of rules for each file which defines who can access that file, and how they can access it. These rules are called file permissions or file modes. The command name chmod stands for "change mode", and it is used to define the way a file can be accessed.

In general, chmod commands take the form:

chmod*optionspermissionsfilename*

If no *options* are specified, chmod modifies the permissions of the file specified by *filename* to the permissions specified by *permissions*.

permissions defines the permissions for the owner of the file (the "user"), members of the group who owns the file (the "group"), and anyone else ("others"). There are two ways to represent these permissions: with symbols (alphanumeric characters), or with octal numbers (the digits 0 through 7).

Let's say you are the owner of a file named myfile, and you want to set its permissions so that:

1. the user can read, write, and execute it;
2. members of your group can read and execute it; and
3. others may only read it.

This command will do the trick:

```
chmod u=rwx,g=rx,o=r myfile
```

This is an example of using symbolic permissions notation. The letters u, g, and o stand for "user", "group", and "other". The equals sign ("=") means "set the permissions exactly like this," and the letters "r", "w", and "x" stand for "read", "write", and "execute", respectively. The commas separate the different classes of permissions, and there are no spaces in between them.
Here is the equivalent command using octal permissions notation:

```
chmod 754 myfile
```

Here the digits 7, 5, and 4 each individually represent the permissions for the user, group, and others, in that order. Each digit is a combination of the numbers 4, 2, 1, and 0:

- 4 stands for "read",
- 2 stands for "write",
- 1 stands for "execute", and
- 0 stands for "no permission."

So 7 is the combination of permissions 4+2+1 (read, write, and execute), 5 is 4+0+1 (read, no write, and execute), and 4 is 4+0+0 (read, no write, and no execute).
chmod syntax

```
chmod [OPTION]... MODE[,MODE]... FILE...

chmod [OPTION]... OCTAL-MODEFILE...

chmod [OPTION]... --reference=RFILE FILE...
```

Options

-c, --changes	Like --verbose, but gives verbose output only when a change is actually made.
-f, --silent, --quiet	Quiet mode; suppress most error messages.

-v, --verbose	Verbose mode; output a diagnostic mesage for every file processed.
--no-preserve-root	do not treat '/' (the root directory) in any special way. This is the default.
--preserve-root	Do not operate recursively on '/'.
--reference=*RFILE*	Set permissions to match those of file *RFILE*, ignoring any specified *MODE*.
-R, --recursive	change files and directories recursively.
--help	Display a help message and exit.
--version	Output version information and exit.

Kill command

kill is used to send a signal to a process.
kill syntax

kill [-s] [-l] %*pid*

Description

The default signal for kill is TERM (which will terminate or "kill" the process). Use -l or -L to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and 0. Alternate signals may be specified in three ways: -9, -SIGKILL or -KILL. Negative PID (process ID) values may be used to choose whole process groups

pid [...]	Send a signal to every *pid* listed.
-*signal*, -s *signal*, --signal *signal*	Specify the signal to be sent. The *signal* can be specified by using name or number. The behavior of signals, and their identifying names, are listed on our signals page.
-l, -- list [*signal*]	List signal names. This option has optional argument, which will convert signal number to signal name, or other way around.
-L, --table	List signal names in a nice table.

Write command

Write sends a message to another user.
Description

The write utility allows you to communicate with other users, by copying lines from your terminal to theirs.
When you run the write command, the user you are writing to gets a message of the format:
Message from yourname@yourhost.
Any further lines you enter will be copied to the specified user's terminal. If the other user wants to reply, they must run write as well.
When you are done, type an end-of-file or interrupt character. The other user will see the message 'EOF' indicating that the conversation is over.
You can prevent people (other than the super-user) from writing to you with the mesgcommand.
If the user you want to write to is logged in on more than one terminal, you can specify which terminal to write to by specifying the terminal name as the second operand to the write command. Alternatively, you can let write select one of the terminals; it will pick the one with the shortest idle time. This is so that if the user is logged in at work and also dialed up from home, the message will go to the right place.
The traditional protocol for writing to someone is that the string '-o', either at the end of a line or on a line by itself, means that it is the other person's turn to talk. The string 'oo' means that the person believes the conversation to be over.
write syntax
writeuser [tty]
Options

User	The user to write to.

Tty	The specific terminal to write to, if the user is logged in to more than one session.

Wall command

wall writes a message to other users.

Description

wall displays the contents of file or, by default, its standard input, on the terminals of all currently logged in users. The command will cut any lines that are over 79 characters to new lines. Short lines are white space padded to have 79 characters. The command will always put carriage return and new line at the end of each line.

Only the super-user can write on the terminals of users who have chosen to deny messages or are using a program which automatically denies messages.

Reading from a file is refused when the invoker is not superuser and the program is suidorsgid.

wall syntax

```
wall [-n] [-t TIMEOUT] [file]
```

Options

-n, --nobanner	Supress banner
-t, --timeout *TIMEOUT*	Write timeout to terminals in seconds. *TIMEOUT* must be positive integ is 300 seconds, which is a legacy from time when people ran terminals c
-V, --version	Display version information and exit.
-h, --help	Display a help message and exit.

wall examples

```
sudo wall message.txt
```

Using the sudo command to run wall as the super user, sends the contents of message.txt to all users.

Merge command

merge is used to perform a three-way file merge.
merge analyzes three files -- an original file, and two modified versions of the original -- and compares them, line-by-line, attempting to resolve the differences between the two sets of modifications in order to create a single, unified file which represents both sets of changes.
Depending on the differences between the two sets of changes, this may be an automatic process or may require user input.
If neither set of changes conflicts with the other, merge can usually figure out what to do on its own. But if the two sets of changes conflict — for example, if the same line of text is worded differently in both modified files — merge will show the conflict in the resulting merged file.
Description

merge incorporates all changes that lead from *file2* to *file3* into *file1*. The result ordinarily goes into *file1*.
Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then merge combines both changes.
A conflict occurs if both *file1* and *file3* have changes in a common segment of lines. If a conflict is found, merge normally outputs a warning and brackets the conflict with "<<<<<<<" and ">>>>>>>" lines. For instance, a typical conflict will look like this:
<<<<<<<file A

lines in file A

=======

lines in file B

>>>>>>> file B

If there are conflicts, the user should edit the result and delete one of the alternatives.

merge syntax

```
merge [options] file1 file2 file3
```

mail command

Send and receive mail.
mail syntax

```
mail [OPTION...] [address...]
```

mail examples

```
mail
```

Opens the mail program and displays the first message in the mailbox, if any. If mail is found, the user is placed at a mail command prompt; type for a list of commands.

```
mail support@computerhope.com
```

Starts a new e-mail addressed to support@computerhope.com. When finished composing the message, type CTRL-D on a new line.

News command

Displays all news items distributed system wide. These items are usually stored in /user/news or / var/news and set up by the system administrator .The news command is normally invoked by any user to read any message that is sent by the system administrator.

Syntax: $ news [options] [news item]
Description : Displays message to all the users.

Option:

-a	displays all of the news items
-s	displays the names of all of the news items.
-c	displays a count of all of the news items

Pipes and filters

we can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a vertical bar (|) on the command line between two commands.

When a program takes its input from another program, performs some operation on that input, and writes the result to the standard output, it is referred to as a filter.

The grep Command

The grep program searches a file or files for lines that have a certain pattern. The syntax is −

```
$grep pattern file(s)
```

The name "grep" derives from the ed (a UNIX line editor) command g/re/p which means "globally search for a regular expression and print all lines containing it."

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work −

```
$ls-l |grep"Aug"
-rw-rw-rw-1 john  doc     11008Aug614:10 ch02
-rw-rw-rw-1 john  doc      8515Aug615:30 ch07
-rw-rw-r--1 john  doc      2488Aug1510:51 intro
-rw-rw-r--1 carol doc      1605Aug2307:35 macros
$
```

There are various options which you can use along with grep command −

Option	Description
-v	Print all lines that do not match pattern.
-n	Print the matched line and its line number.
-l	Print only the names of files with matching lines (letter "l")
-c	Print only the count of matching lines.
-i	Match either upper- or lowercase.

The sort Command

The sort command arranges lines of text alphabetically or numerically. The example below sorts the lines in the food file −

```
$sort food
AfghaniCuisine
BangkokWok
BigAppleDeli
Isle of Java
Mandalay
SweetTooth
TioPepe's Peppers
$
```

The sort command arranges lines of text alphabetically by default. There are many options that control the sorting –

Option	Description
-n	Sort numerically (example: 10 will sort after 2), ignore blanks and tabs.
-r	Reverse the order of sort.
-f	Sort upper- and lowercase together.
+x	Ignore first x fields when sorting.

The pg and more Commands

A long output would normally zip by you on the screen, but if you run text through more or pg as a filter, the display stops after each screenful of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through more as follows −

```
$ls-l |grep"Aug"| sort +4n| more
-rw-rw-r--1 carol doc      1605Aug2307:35 macros
-rw-rw-r--1 john  doc      2488Aug1510:51 intro
-rw-rw-rw-1 john  doc       8515Aug615:30 ch07
-rw-rw-r--1 john  doc     14827Aug912:40 ch03
            .
            .
            .
-rw-rw-rw-1 john  doc     16867Aug615:56 ch05
--More--(74%)
```

The screen will fill up with one screenful of text consisting of lines sorted by order of file size. At the bottom of the screen is the more prompt where you can type a command to move through the sorted text.

whoami

whoami prints the effective user ID.
This command prints the username associated with the current effective user ID.
Running whoami is the same as running the idcommand with the options -un.
whoami syntax

whoami [OPTION]

Options

--help	Display a help message, and exit.
--version	Display version information, and exit.

Who

Who command is used to know about all the users who are logged in.

Syntax: $who

UNIT IV

Shell variables:

variables are an integral part of shell programming. They provide the ability to store and manipulate information within a shell program. The variables use are completely under our control. We can control and destroy any number of variables as needed to solve the problem at hand.

The various rules forbuilding shell variables are as:

1. A variable name is any combination of alphabets, digits and an underscore(“_”).
2. No commas or blanks are allowed within a variable name.
3. The first character of variable name must either be an alphabet or an underscore”_”.
4. Variable names should be of any reasonable length.
5. Variable names are case sensitive i,e Name,NAME,name,Name are all different variables.

Here are some sample variable names:

Si_int

M_hra

Salary

Salary

In programming we need variables to hold data. Like other programming language, the shell scripts also support creation and usage of variables.

There are two types of variables in Linux shell namely System variables and User defined Variables.

The system variables are created and maintained by Linux itself. These types of variables are defined in Capital Letters.

Whereas the user defined variables are created and maintained by the User and these variables are defined in lower letters.

Shell special characters (meta characters)

Normally, these characters have special meaning to the shell:

\ ' " ` <>| ;<Space><Tab><Newline> ( ) [ ] ? # $ ^ & * =

Here is the meaning of some of them:

\ ' " and ' are used for quoting.
< and > are used for input/output redirection.
| pipes the output of the command to the left of the pipe symbol.
"|" to the input of the command on the right of the pipe symbol.
; separates multiple commands written on a single line.
<Space> and <Tab> separate the command words.
<Newline> completes a command or set of commands.
( ) enclose command(s) to be launched in a separate shell (subshell). E.g. (dir).

{ } enclose a group of commands to be launched by the current shell. E.g. { dir }. It needs the spaces.
& causes the preceding command to execute in the background (i.e., asynchronously, as its own separate process) so that the next command does not wait for its completion.

* when a filename is expected, it matches any filename except those starting with a dot (or any part of a filename, except the initial dot).

? when a filename is expected, it matches any single character.
[ ] when a filename is expected, it maches any single character enclosed inside the pair of [ ].

&& is an "AND" connecting two commands.

command1 && command2 will execute command2 only if command1 exits with the exit status 0 (no error). For example: cat file1 && cat file2 will display file2 only if displaying file1 succeeded.

|| is an "OR" connecting two commands.

command1 || command2 will execute command2 only if command1 exits with the exit status of non-zero (with an error). For example: cat file1 || cat file2will display file2 only if displaying file1 didn't succeed.

= assigns a value to a variable.

Example. This command:
me=blahblah
assigns the value "blahblah" to the variable called "me". I can print the name of the variable using:
echo $me

$ preceeds the name of a variable to be expanded.

The variables are either assigned using "=" or are one of the pre-defined variables (which cannot be assigned to):
$0 name of the shell or the shell script being executed.
$# number of the positional parameters to the command
$1 the value of the first positional parameter passed to the command. $2 is the second positional parameter passed to the command. etc. up to $9.
$* expands to all positional parameters passed to the command
$@ expands to all positional parameters passed to the command, but individually quoted when "$@" is used.

ITERATION STAEMENTS (LOOPS)

Loops are a powerful programming tool that enable you to execute a set of commands repeatedly. Let's examine the following types of loops available to shell programmers −

- The while loop
- The for loop
- The until loop
- The select loop

we would use different loops based on dfferent situation. For example while loop would execute given commands until given condition remains true where as until loop would execute until a given condition becomes true.

Nesting Loops

All the loops support nesting concept which means you can put one loop inside another similar or different loops. This nesting can go upto unlimited number of times based on your requirement.

Here is an example of nesting while loop and similar way other loops can be nested based on programming requirement −

Nesting while Loops

It is possible to use a while loop as part of the body of another while loop.

Syntax

```
while command1 ; # this is loop1, the outer loop
do
   Statement(s) to be executed if command1 is true

while command2 ; # this is loop2, the inner loop
do
     Statement(s) to be executed if command2 is true
done
```

```
  Statement(s) to be executed if command1 is true
done
```

Example

Here is a simple example of loop nesting, let's add another countdown loop inside the loop that you used to count to nine −

```
#!/bin/sh

a=0
while [ "$a" -lt 10 ]    # this is loop1
do
  b="$a"
while [ "$b" -ge 0 ]  # this is loop2
do
echo -n "$b "
    b=`expr $b - 1`
done
echo
  a=`expr $a + 1`
done
```

This will produce following result. It is important to note how echo -n works here. Here -n option let echo to avoid printing a new line character.

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
```

```
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

The infinite Loop

All the loops have a limited life and they come out once the condition is false or true depending on the loop.

A loop may continue forever due to required condition is not met. A loop that executes forever without terminating executes an infinite number of times. For this reason, such loops are called infinite loops.

Example

Here is a simple example that uses the while loop to display the numbers zero to nine −

```
#!/bin/sh

a=10

until[ $a -lt10]
do
echo $a
  a=`expr $a + 1`
done
```

This loop would continue forever because a is always greater than or equal to 10 and it would never become less than 10.

The break statement

The break statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.

Syntax

The following break statement would be used to come out of a loop −

```
break
```

The break command can also be used to exit from a nested loop using this format −

```
break n
```

Here n specifies the nth enclosing loop to exit from.

Example

Here is a simple example which shows that loop would terminate as soon as a becomes 5:

```
#!/bin/sh

a=0

while[ $a -lt10]
do
echo $a
if[ $a -eq5]
then
break
fi
  a=`expr $a + 1`
done
```

This will produce following result −

```
0
1
2
3
4
5
```

Here is a simple example of nested for loop. This script breaks out of both loops if var1 equals 2 and var2 equals 0 −

```
#!/bin/sh

for var1 in123
do
for var2 in05
do
if[ $var1 -eq2-a $var2 -eq0]
then
break2
else
echo"$var1 $var2"
fi
done
done
```

This will produce following result. In the inner loop, you have a break command with the argument 2. This indicates that if a condition is met you should break out of outer loop and ultimately from inner loop as well.

```
10
15
```

The continue statement

The continue statement is similar to the break command, except that it causes the current iteration of the loop to exit, rather than the entire loop.

This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Syntax

```
continue
```

Like with the break statement, an integer argument can be given to the continue command to skip commands from nested loops.

```
continue n
```

Here n specifies the nth enclosing loop to continue from.

Example

The following loop makes use of continue statement which returns from the continue statement and start processing next statement −

```
#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
  Q=`expr $NUM % 2`
if[ $Q -eq0]
then
echo"Number is an even number!!"
continue
fi
echo"Found odd number"
done
```

This will produce following result −

```
Found odd number
Numberis an even number!!
Found odd number
Numberis an even number!!
Found odd number
Numberis an even number!!
Found odd number
```

CONDITIONAL STATEMENTS:

While writing a shell script, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform right actions.

Unix/Linux Shell supports conditional statements which are used to perform different actions based on different conditions. Here we will explain following two decision making statements −

- The if...else statements
- The case...esac statement

The if...else statements:

If else statements are useful decision making statements which can be used to select an option from a given set of options.

Unix/Linux Shell supports following forms of if..else statement −

- if...fi statement
- if...else...fi statement
- if...elif...else...fi statement

Most of the if statements check relations using relational operators discussed in previous chapter.

The case...esac Statement

You can use multiple if...elif statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Unix/Linux Shell supports case...esac statement which handles exactly this situation, and it does so more efficiently than repeated if...elif statements.