UNIT I

1

INTRODUCTION

Unit Structure

1.0 OBJECTIVES

Introducing various tools and techniques commonly used by Linux programmers, system administrators and end users to achieve their day-to-day work in Linux environment.

1.1 INTRODUCTION

Linux is similar to Unix. It is coming under OS (Open source) and community-developed operating system. Various devices like computers, servers, mainframes, mobile devices, and embedded devices. Linux is highly configurable. It depends on a modular design which provides flexibility to users to customize their own versions of Linux.

1.2 AN OVERVIEW

Depending on the application, Linux can be optimized for different purposes such as:

- Networking performance.
- Computation performance.
- Deployment on specific hardware platforms.

Users can choose different Linux distributions.

1.2.1 History of Linux:

In 1991, the Linux history started with the starting of a particular project by the Finland student **Linus Torvalds** for creating a new free **OS kernel**. The final Linux Kernel was remarked by continuous development throughout the history since then.

- Linux was proposed by the Finland student Linus Torvalds in 1991.
- HP-UX (**Hewlett Packard**) 8.0 version was published.
- Hewlett Packard 9.0 version was published in 1992.
- FreeBSD 1.0 version and **NetBSD**8 version was released in 1993.
- Red Hat Linux was proposed in 1994. Caldera was detected by Ransom love and Bryan Sparks and NetBSD 1.0 version published.
- HP-UX 10.0 version and FreeBSD 2.0 version was released in 1995.
- K Desktop Environment was established by **Matthias Ettrich** in 1996.
- HP-UX 11.0 version was released in 1997.
- The IRIX 6.5 version, i.e., the fifth SGI UNIX generation, Free BSD 3.0 version, and Sun Solaris 7 OS was released in 1998.
- The **Caldera System** agreement with professional services division and SCO server software division was released in 2000.
- **Linus Torvalds** published the Linux version 2.4 source code in 2001.
- **Microsoft** filed the Trademark collection against Lindows.com in 2001.
- Lindows name was modified to Linspire in 2004.
- The first publication of **Ubuntu** was published in 2004.
- The openSUSE project started a free distribution from the community of **Novell** In 2005.
- **Oracle** published its Red Hat distribution in 2006.

- **Dell** begun laptop distribution with Ubuntu which was pre-installed on it in 2007.
- Linux kernel version 3.0 was released in 2011.
- Linux-based android of Google insisted 75% of the market share of the Smartphone, based on the number of phones exported in 2013.
- Ubuntu insisted on 20000000+ users in 2014.

1.2.2 Philosophy:

All operating systems have some philosophy. When Unix was being developed in the late 1960s and early 1970s, the inventors were intent upon building an operating system that was significantly different from the operating systems that ante ceded. The idea of Unix was markedly different from that of other operating systems. And the Linux philosophy is relatively naturally deduced directly from the Unix philosophy.

1.2.3 Community:

A distribution is largely driven by its inventor and communities. Some develop and fund their distributions on a levy base, Debian being a well-known illustration. Others maintain a community interpretation of their commercial distributions. Numerous Internet communities also give support to Linux users. Utmost distributions and free software / open-source projects have IRC chatrooms or newsgroups. Online forums are another means for support, with notable exempli cations being LinuxQuestions.org and the various distribution specific support and community forums, such as ones for Ubuntu, Fedora, and Gentoo. Linux distributions host mailing lists: commonly there will be a specific topic such as usage or development for a given list.

1.2.4 Distributions:

Other operating systems like Microsoft combine each bit of coding internally and release it as a single package. You have to choose from one of the interpretations they offer.But Linux is different from them.

Different parts include kernel, shell utilities, X server, system environment, graphical programs, etc. If you want, you can access the codes of all these parts and assemble them yourself. But its not an easy task seeking a lot of time and all the parts has to be assembled correctly.

Linux Distributions List:

Approximately an average of six hundred Linux distributors providing different features.

1) Ubuntu:

- Came into Existence in 2004
- Used as Graphical Linux without the use of command line.

- Well known distribution
- Lot of pre-installed apps
- Easy to use

2) Linux Mint:

- Based on Ubuntu
- Uses repository software.
- Media codec and proprietary software are included
- It uses cinnamon and desktop instead of Ubuntu's unity desktop environment

3) Debian:

- Came into existence in 1993.
- Most Stable Linux distribution.
- User Friendly.
- Every release is based on the name of the movie Toy Story.

4) Red Hat Enterprise / CentOS:

- Commercial Linux distributor.
- Red hat uses trademark law to prevent their software from being redistributed.
- CentOS is a community project that uses red hat enterprise Linux code
- It is a free version of RHEL

5) Fedora:

- Mainly focuses on free software.
- Used 'upstream' software
- Less stable

Choosing a Linux Distro

Distribution	Why To Use
Ubuntu	It works like Mac OS and easy to use.
Linux mint	It works like windows and should be use by newcomers.

Debian	It provides stability but not recommended to a new user.
Fedora	If you want to use red hat and latest software.
Red hat enterprise	To be used commercially.
CentOS	If you want to use red hat but without its trademark.
OpenSUSE	It works same as Fedora but slightly older and more stable.
Arch Linux	It is not for the beginners because every package has to be installed by yourself.

1.3 HOW IS THE LINUX OPERATING SYSTEM USED?

Every version of the Linux OS manages hardware resources, launches, and handles applications, and provides some form of user interface.

The Linux OS can be found in many different settings, supporting many different use cases. Linux is used in the following ways:

- **Server OS**: web servers, database servers, file servers, email servers and any other type of shared server. Designed to support high-volume and multithreading applications, Linux is well-suited for all types of server applications.
- **Desktop OS:** for personal productivity computing. It is an OS and freely available.
- **Headless server OS:** for systems that do not require a graphical user interface (GUI) or directly connected terminal and keyboard. Headless systems are often used for remotely managed networking server and other devices.
- **Embedded device or appliance OS**: needs less computing function. It is used as as embedded OS for a variety of application like including household appliances, automotive entertainment systems and network file system appliances.
- **Network OS:** for routers, switches, domain name system servers, home networking devices and more. For example, Cisco that uses the Linux kernel.
- **Software development OS:** for enterprise software development. Although many development tools have been ported to Windows or other OSes, Linux is home to some of the most widely used open-source software development tools. For example, git for distributed source control; vim and emacs for source code editing; and compilers and interpreters for almost every programming language.

- **Cloud OS:** for cloud instances like Linux for cloud servers, desktops and other services.

1.4 HOW THE LINUX OPERATING SYSTEM WORKS

The Linux OS follows a modular design that's the key to its numerous variations and distributions. All Linux distributions are based on Linux kernel, but they can differ depending on factors such as:

- **Kernel version:** configured with more recent versions, to incorporate newer features or with older versions to be more stable.
- **Kernel modules:** This is software that can be loaded and unloaded into the kernel to extend functionality without rebooting. Kernel modules are often used to support:
  - device drivers, which use code that controls how attached devices operate.
  - file system drivers, which use code that controls how the kernel works with different file systems; and
  - system calls, which use code that controls how programs request services from the kernel.
- **Configuration options:** It is compiled with configuration options set to include only device or file system drivers are used for some specialized distributions; for example, compiling a kernel for a wireless device without any wired network device drivers.

The Linux kernel is the one thing that all systems running Linux have in common. Linux works by:

- Loading and booting a Linux kernel.
- Kernel manages all system input and output. The system is initialized, and processes can be started.
- As system processes are started, the system can be used for processes that include network server functions, commands entered interactively via command line, desktop applications or any application or program.

When using Linux with a desktop environment as a GUI, Linux works much an equivalent as any GUI-based OS. Applications and other resources are often opened by clicking on icons, and files are often moved, copied or deleted employing a mouse or trackpad

1.5 LINUX KERNEL VS DISTRIBUTION

When we discuss Linux Operating Systems, we actually mention Whole OS or the Linux Distribution, not just the Kernel. Technically Linux is that the Kernel of the OS. A kernel is that

the core a part of any OS which basically handles Hardware. once we use Linux kernel and add other important things just like the shell, Various Libraries, GUI and other programs like Multimedia Apps etc. Then we refer those systems as a Linux Distribution which generally are often considered as complete OS. there's there are dozens of Linux distributions available but few of them which are very fashionable and widely used round the world.

1.6 WHY LEARN LINUX?

Since Linux has many advantages and features to use it,

Some of them are as follows:

1. **Free:** Linux is License Free software.
2. **Security (Virus Free with inbuilt Firewall protection):** The security aspect of Linux is much stronger; Inbuilt Firewall protection is available hence Linux is not prone to viruses.
3. **OpenSource:** Linux is a opensource software hence source code is open & easily available on internet.
4. **Customizable:** It is Customized with different types of hardware and software
5. **Flexibility:** Linux is a flexible freeware operating system.
6. **Cost:** It is mostly free to obtain.
7. **Linux is versatile:** You can use Linux on virtually anything you develop
8. **Linux is a community:** You can work with other Linux developers to share knowledge and learnings
9. **Linux is very stable:** Linux systems rarely crash, and when they do, the whole system normally does not go down.

1.6.1 Importance of Linux in software ecosystem:

- GUI (Graphical User Interface)
- Multitasking: No of programs running at same time.
- Multiuser: Several users on the same machine at the same time
- Multiplatform: Number of processors at a time. runs on many different CPUs, not just Intel.
- Multiprocessor: Kernel supports multiple independently thread of a single process or multiple process.
- Multithreading: Has native kernel support for multiple independent threads of control within a single process memory space.

- Linux runs in a protected mode on 386 machines. It has memory protection between processes so that that one program can't bring whole system down.
- It supports virtual memory using paging i.e., separate partition or file in file system created.
- Dynamically linked shared libraries & static libraries.
- It is an open-source software hence all source code is available including the kernel and all drivers.
- Multiple virtual consoles
- Linux has different filesystem depending upon file system like FAT32, VFAT, NTFS or NFS, ext3, ext4, swap (RAM)
- It supports Network connectivity
- It supports Network Servers like It supports TCP/IP Networking including FTP, Telnet, NFS, etc.
- It has Hardware Support
- Firewall Protection inbuilt available so that no outsider introducer can attack on our System.
- It has its Own K-office Introduction to Linux 13
- Linux supports NetWare client and server with static Routing & Dynamic Routing with the help of DHCP.
- Linux supports 'Samba Server' for connectivity of windows & Linux file sharing.
- It Supports different Time Servers with send mail facility.
- It supports ftp & http services with Apache web server.

1.6.2 web servers:

- **Economical:** Linux is an open-source operating system, Its all versions are available with lower price other than web servers., hence Linux web servers are the best to choose for web hosting services.
- **Flexibility:** Linux provides a flexible hosting environment with plenty of high-performance applications.
- **High Up time:** high up time decides how long a web server functions well. Linux servers have high up-time because of its robust performance and reliable.
- **Stability and Performance:** If hosting package includes Linux based server, then performance is good of web servers. This operating system is the most stable and doesn't slow down over time or freeze

up. Linux web servers don't experience memory leaks and the up times are often much better than other servers.

- **Inexpensive Hosting:** Linux is an open-source operating system, which means it's free to use.
- **Multitasking:** Linux server can run multiple programs simultaneously and its enables programs to run continuously in the background while user works with some other programs. Hence Linux web servers to have multitasking capabilities.

1.6.3 Supercomputers:

- Customization: Linux open-source nature make source code available to modify code or make customization of code with supercomputer administration. Hence custom server implantation is possible.
- Less overhead, i.e., way faster: Linux does not require extra software to update or upgrade.
- No need for reboots.
- 10,000 times more stable: Linux with supercomputers is more stable.
- Easier to automate with scripts.
- Easier backup facility available of Linux servers with supercomputers.

1.6.4 Mobile:

- Google's Android developers modified the Linux kernel and created Android operating system which is based on Linux kernel (core of operating system).
- powered by the Linux kernel, which can be found on a wide range of devices.
- Android is an open-source operating system which allows developers to access unlocked hardware and develop new programs as they wish.
- Android manages processes and different Apps to keep minimum power consumption.

1.6.5 Servers:

Linux servers are very powerful of their outstanding characteristics like security, stability, and flexibility. These Linux servers has in built web servers and business applications which supports network administration and web and database management services.

Following are the Key features of Linux servers:

- High Level Security: Since security is main concern hence Linux servers provides high level of security with the help of firewall protection and powerful system administration and file access system to provide authorization.
- Ease of administration: Linux servers are easily administrated. They can be controlled and managed remotely. It reduces cost because no additional software setup for the administration is required.
- Supports multiple applications: Linux servers supports many software applications because of its inbuilt technical strength.
- Customization is easy.
- Reliable: These servers are very reliable as they offer consistent services without any failures.

1.7 PROS AND CONS OF LINUX

Some advantages of using Linux include:

- Open-source software
- Licensing cost is Nil.
- Reliability.
- Backward compatibility
- Many choices of distributions.

Some disadvantages of using Linux include:

- **Lack of established standard.**
- **Support costs:** Most enterprise Linux distributors like SUSE and Red Hat offer support contracts. Depending on the circumstances, these license fees can reduce savings significantly.
- **Proprietary software.** Desktop productivity software like Microsoft Office cannot be used on Linux desktops, and other proprietary software may be unavailable for Linux platforms.
- **Unsupported hardware.**
- **Steep learning curve.** Many users struggle to learn to use the Linux desktop or Linux-based applications.

In some cases, the same Linux attribute can be either an advantage or disadvantage. For example, having many options for customizing the Linux OS is advantageous for manufacturers looking for an embedded OS, but it is a disadvantage for enterprises that want a desktop OS that can be used by a wide range of end users.

1.8 LET US SUM UP

Linux is an **open-source** operating system. As it is open source, it is special and different from other operating systems, which means that you can customize it by editing source code. *It provides programming as well as a graphical user interface.* Linux is built by **Linux Torvalds** because he wanted to create a free operating system kernel that anyone can use.

Linux is a collection of operating systems that are based on Linux **kernel**. The first version of Linux was released in the **year 1991.** The Linux system is most commonly used for servers; however, it is available in desktop versions as well.

Ubuntu, Devian, and **Fedora** are some popular Linux distributions. Also, we have **SUSE Linux Enterprise Server (SLES)** and **RedHat Enterprise** Linux for the commercial distribution of Linux. As it is open source, we can modify the source code and make variations in the operating system.

1.11 UNIT END QUESTIONS

1. Define what are different operating systems in market. Explain Linux operating System in detail.
2. Explain features of Linux in detail.
3. Explain features of Linux in detail with different Linux distributions.
4. Explain Linux Architecture with neat diagram.
5. What is Linux? Define History of Linux.
6. Explain about Philosophy of Linux.
7. What is Linux Community and state the names of Linux communities.
8. Explain Linux Terminology.
9. Explain Linux different Distributions in detail.
10. Explain Linux kernel vs different distribution.
11. Define different reasons Why we learn Linux.
12. Explain Importance of Linux in software ecosystem 1.
13. Explain importance of Linux in web servers.
14. Explain importance of Linux in Supercomputers.
15. Explain importance of Linux in Mobile.
16. Explain importance of Linux servers.

1.9 LIST OF REFERENCES

1. The Linux Programming Interface: A Linux and UNIX System Programming Handbook 1st Edition by Michael Kerrisk.
2. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward.
3. The Linux Command Line: A Complete Introduction 1st Editionby by William E. Shotts Jr.
4. Fundamentals of Linux by Pelz Oliver.

1.10 BIBLIOGRAPHY

1. Linux Command Line and Shell Scripting Bible, 3rd Edition by Richard Blum
2. Linux*: The Complete Reference, Sixth Edition by Richard Petersen*
3. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward
4. The Linux Command Line: A Complete Introduction 1st Edition by by William E. Shotts Jr.
5. Fundamentals of Linux by Pelz Oliver
6. https://www.javatpoint.com/linux-distributions
7. https://searchdatacenter.techtarget.com/definition/Linux-operating-system
8. https://opensource.com/resources/linux
9. https://www.linuxfoundation.org/tools/participating-in-open-source-communities/
10. https://www.tutorialspoint.com/operating_system/os_linux.htm

*****

2

INSTALLATION

Unit Structure

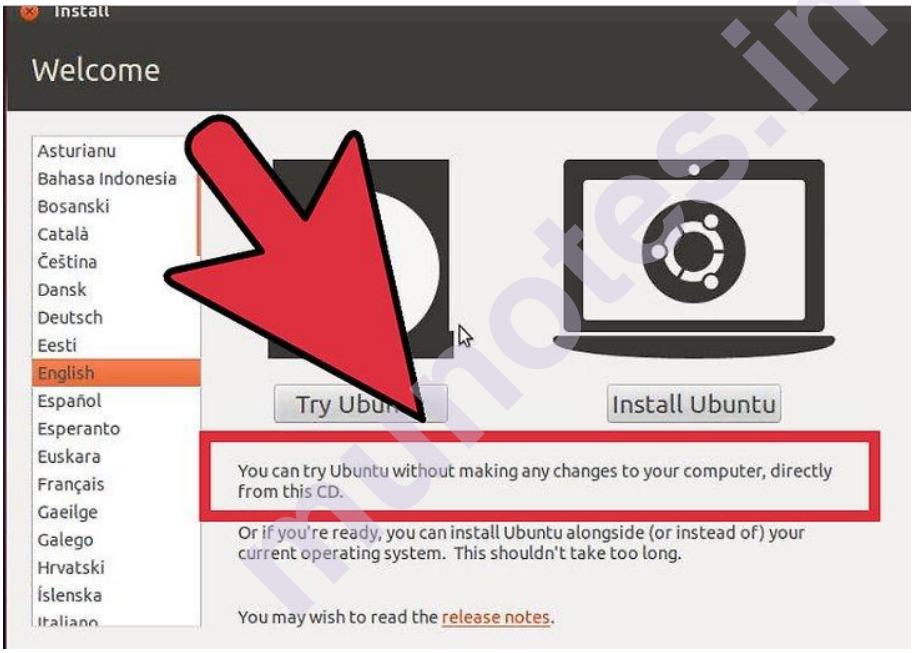
2.0 INTRODUCTION

Linux is the foundation of thousands of open-source operating systems designed to replace Windows and Mac OS. It is free to download and install on any computer. Because it is open source, there are a variety of different interpretation, or distributions, available developed by different groups. Follow the guidelines for installing any version of Linux.

2.2 DOWNLOAD THE LINUX DISTRIBUTION OF YOUR CHOICE.

Still, consider trying a featherlight and easy to use distribution, similar as Ubuntu or Linux Mint. Linux distributions (known as "distros") are generally available for free to download in ISO format. You can find the ISO for the distribution of your choice at the distribution's website. This format needs to be burned to a CD or USB stick before you can use it to install Linux. This will produce a Live CD or Live USB.

- A Live CD or Live USB is a fragment that you can boot into, and frequently contains a interpretation of the operating system that can be run directly from the CD or USB stick.
- Install an image burning program or use your system's built-in burning tool if you are using Windows 7, 8, or Mac OS X. Pen Drive Linux and UNetBootin are two popular tools for burning ISO files to USB sticks.



2.3 BOOT INTO THE LIVE CD OR LIVE USB

Utmost computers are set to boot into the hard drive first, which means you'll need to change some settings to boot from your recently burned CD or USB. Start by rebooting the computer.

- Once the computer reboots, press the key used to enter the menu. The key for your system will be displayed on the same screen as the manufacturer's logo. Typical keys include F12, F2, or Del.
- For Windows 8 users,

✓ hold the Shift key and click restart.

This will load the Advanced Start-up Options, where you can boot from CD.

- For Windows 10 users,

✓ go to advanced boot and then "Restart Now."

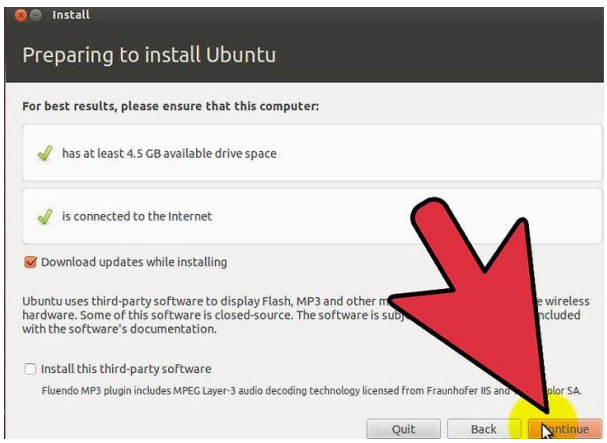
- Once you're in the boot menu,

✓ select CD or USB.

Once you've changed the settings, save and exit the BIOS setup or boot menu.

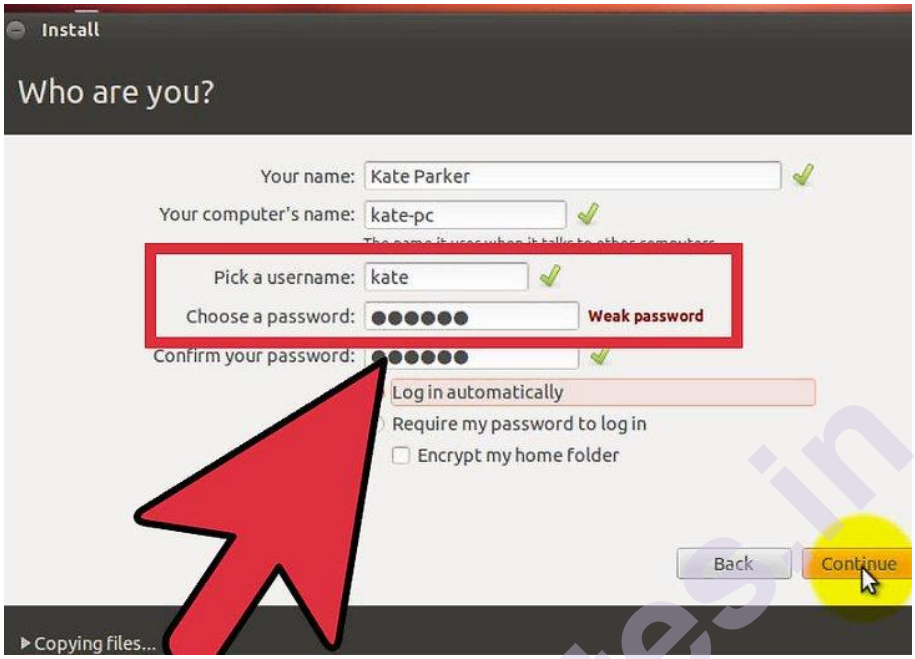
2.4 TRY OUT THE LINUX DISTRIBUTION BEFORE INSTALLING

Live CDs and USBs can launch a "live environment", giving you the capability to test it out before making the switch. It is not possible to create files but can navigate around the interface.



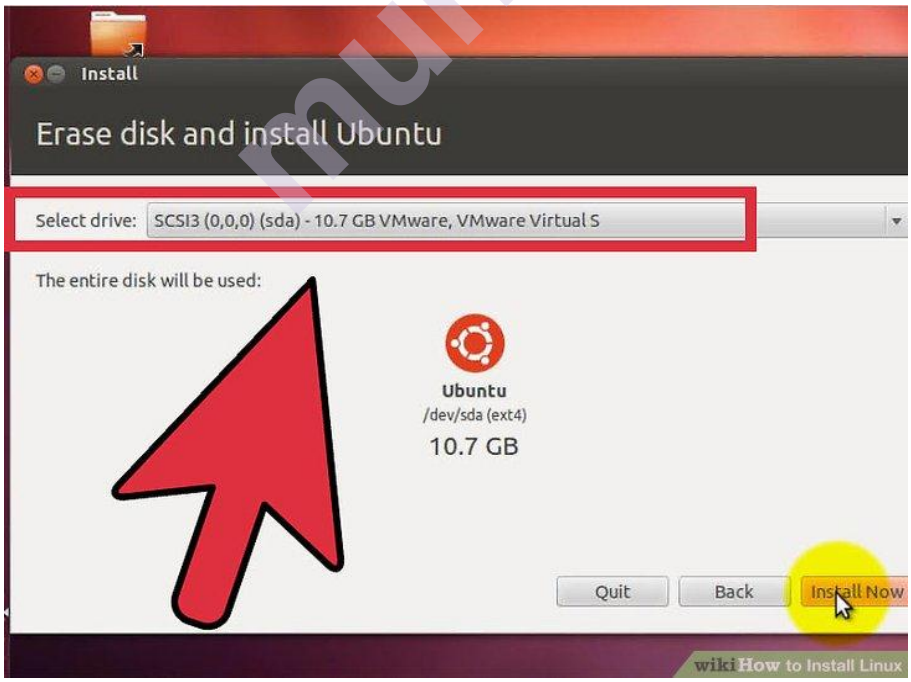
2.5 START THE INSTALLATION PROCESS.

If you are trying distro, then it can be launched from the application on the desktop. If it is not distro, we can start the installation from the boot menu.



2.6 CREATE A USERNAME AND PASSWORD

We need to create login information to install Linux:



2.7 SET UP THE PARTITION

Linux needs to be installed on a separate partition from any other OS on your computer if you intend binary booting Linux with another OS. A partition is a portion of the hard drive that is formatted specifically for that operating system.

Ubuntu will set a partition automatically and Linux installation require at least 20 GB

- If partitions are not given by installation process, then check the formatted as EX4. If the copy of Linux you are installing is the only operating system on the computer, you will most likely have to manually set your partition size.

2.8 BOOT INTO LINUX

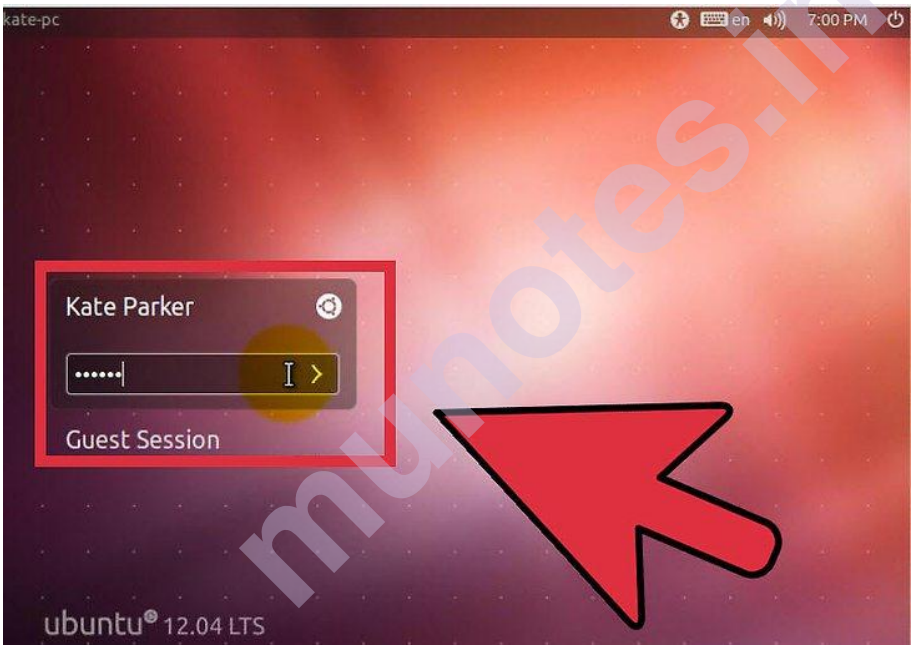
After installation, computer will be rebooted. We can see a new screen. “GNU GRUB” is a boot loader and handles Linux installation.

- If you install multiple distros on your computer, they will all be listed here.

2.9 CHECK YOUR HARDWARE

Utmost H/W should work out of the box with your Linux distro, though you may need to download some additional drivers to get everything working.

- Some hardware requires proprietary drivers to work correctly in Linux. This is most common with graphics cards.
- In Ubuntu, we can download proprietary drivers through the System Settings menu.
- Select the Additional Drivers option, and then select the graphics driver from the list. Other distros have specific methods for obtaining extra drivers.
- You can find other drivers from this list as well, such as Wi-Fi drivers.



2.10 START USING LINUX

If installation is complete and verified if all H/W is working properly then we are ready to start using Linux. Many popular programs can be installed from their respective repositories.

2.11 LINUX STRUCTURE

Let's first start with the basic knowledge of the Linux operating system.

2.11.1 Linux operating system:

An OS can be defined as an interface between the computer hardware and the user. OS is a group of software that handles the resources of the computer and provides basic services for computer programs.

It is an essential component of system software. The objective is to provide a platform for the user to run any program efficiently. Linux is one of the famous versions of the UNIX OS. It is coming under free open source.

2.11.2 Architecture of Linux system:

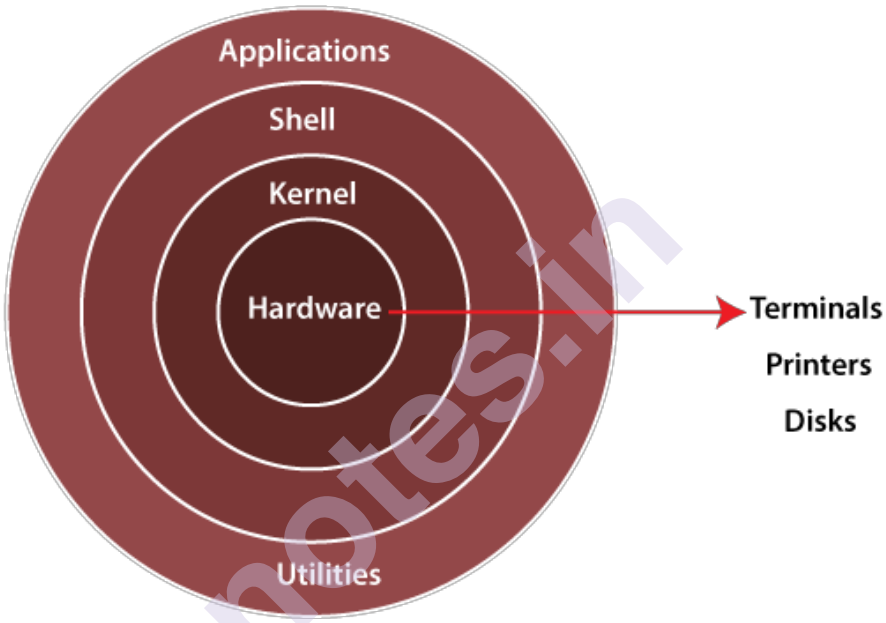


Fig 1. Architecture of Linux System

The Linux operating system's architecture mainly contains some of the components: **the Kernel, System Library, Hardware layer, System,** and **Shell utility**.

1. Kernel: - The kernel is one of the core section of an operating system. It is responsible for each of the major actions of the Linux OS. This operating system contains distinct types of modules and cooperates with underlying hardware directly. The kernel facilitates required abstraction for hiding details of low-level hardware or application programs to the system. There are some of the important kernel types which are mentioned below:

- Monolithic Kernel
- Micro kernels
- Exo kernels
- Hybrid kernels

2. System Libraries: These libraries can be specified as some special functions. These are applied for implementing the operating system's functionality and don't need code access rights of the modules of kernel.

3. System Utility Programs: It is responsible for doing specialized level and individual activities.

4. Hardware layer: Linux operating system contains a hardware layer that consists of several peripheral devices like CPU, HDD, and RAM.

5. Shell: It is an interface among the kernel and user. It can afford the services of kernel. It can take commands through the user and runs the functions of the kernel. The shell is available in distinct types of OSes. These operating systems are categorized into two different types, which are the **graphical shells** and **command-line shells**.

The graphical line shells facilitate the graphical user interface, while the command line shells facilitate the command line interface. Thus, both of these shells implement operations. However, the graphical user interface shells work slower as compared to the command-line interface shells.

There are a few types of these shells which are categorized as follows:

- Korn shell
- Bourne shell
- C shell
- POSIX shell

2.12 UNIT END QUESTIONS

1. How to Download the Linux distribution of your choice.
2. How to create a username and password.
3. How to set up the partition.
4. Briefly explain Linux structure
5. Short notes on Architecture of linux sytem

2.13 LIST OF REFERENCES

1. The Linux Programming Interface: A Linux and UNIX System Programming Handbook 1st Edition by Michael Kerrisk
2. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward
3. The Linux Command Line: A Complete Introduction 1st Editionby by William E. Shotts Jr.

4. Fundamentals of Linux by Pelz Oliver.

2.14 BIBLIOGRAPHY

1. Linux Command Line and Shell Scripting Bible, 3rd Edition by Richard Blum.
2. Linux: The Complete Reference, Sixth Edition by Richard Petersen.
3. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward.
4. The Linux Command Line: A Complete Introduction 1st Edition by by William E. Shotts Jr.
5. Fundamentals of Linux by Pelz Oliver.

*****

3

LINUX STRUCTURE

Unit Structure

3.1 INTRODUCTION

A Linux file system is a structured collection of files, which may be in a disk drive or a partition. Mostly a partition is a segment of memory and contains some data. Our system may contain various partitions of the memory. Generally, every partition contains a file system. Some reasons for maintaining the file system are given below.

- Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off.
- Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space.

File system contains the following sections:

- The root directory (/)
- A specific data storage format (EXT3, EXT4, BTRFS, XFS and so on)
- A partition or logical volume having a particular file system.

3.2 WHAT IS THE LINUX FILE SYSTEM?

Linux file system is generally a built-in layer of a Linux operating system which is used to handle the storage. Usually, it manages name of the file, size of the file, date of creation and much more about a file.

3 Basic File Types are:

- **Ordinary Files:** Contains data, text, or program instructions.
- **Directories:** Directories are equivalent to folders.
- **Special Files:** Provides access to H/W

3.3 LINUX FILE SYSTEM STRUCTURE

It has a hierarchal file structure which contains:

- a root directory and
- its subdirectories.
- Linux file system contains two-part file system software implementation architecture

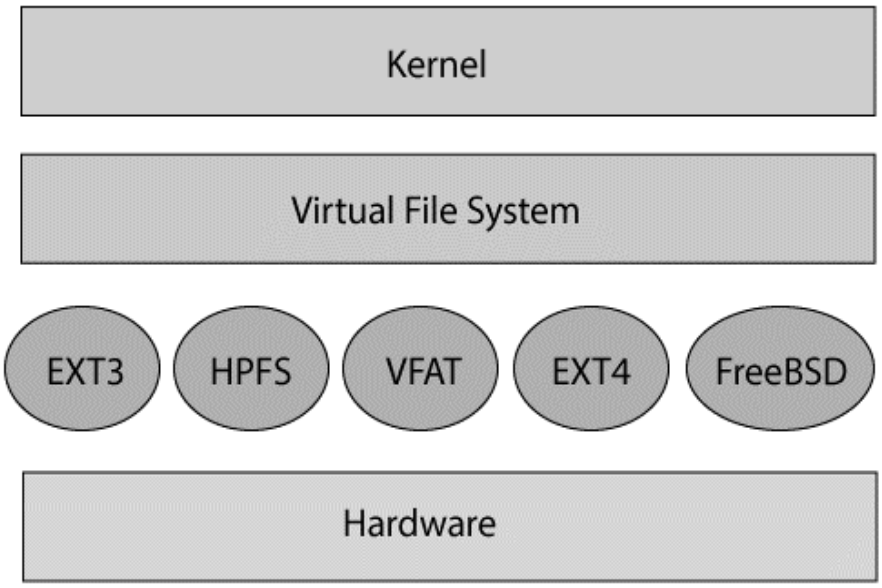


Fig 1. Linux File System Architecture

To interact with file system, Application Programming Interface is required. API facilitates algorithm for arranging files on a file system and facilitates tasks such as creating, deleting, and copying the files. Linux virtual file system provides a single set of commands for the kernel. It requires the specific system driver to give an interface to the file system.

1.4 TYPES OF LINUX FILE SYSTEM

Linux operating system installation offers many file systems like **Ext, Ext2, Ext3, Ext4, JFS, ReiserFS, XFS, btrfs,** and **swap**.

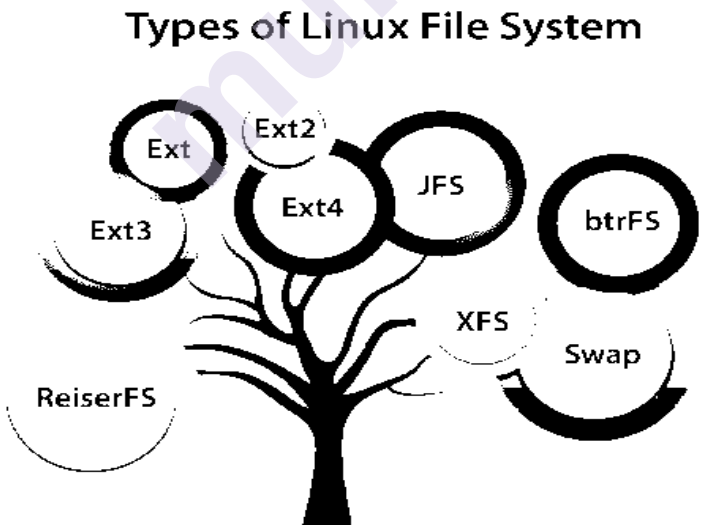


Fig 2. Types of Linux File System

Let's understand each of these file systems in detail:

3.4.1 Ext, Ext2, Ext3 and Ext4 file system:

Ext1 stands for **Extended File System**. It was primarily developed for **MINIX OS**. Due to some limitations, this file system no longer used.

Ext2 is managing 2 terabytes of data.

Ext3 is developed through Ext2; it is an upgraded version of Ext2 and contains backward compatibility. The major drawback of Ext3 is that it does not support servers because this file system does not support file recovery and disk snapshot.

Ext4 file system is the fastest file system and compatible. It is the default file system in linux distribution.

3.4.2 JFS File System:

JFS stands for **Journaled File System**, and it is developed by **IBM for AIX Unix**. It is an alternative to the Ext file system. It can also be used in place of Ext4, where stability is needed with few resources. It is a handy file system when CPU power is limited.

3.4.3 ReiserFS File System:

ReiserFS is an alternative to the Ext3 file system. It has improved performance and advanced features. In the earlier time, the ReiserFS was used as the default file system in SUSE Linux. This file system dynamically supports the file extension, but it has some drawbacks in performance.

3.4.4 XFS File System:

XFS file system was considered as high-speed JFS, which is developed for parallel I/O processing. NASA still using this file system with its high storage server (300+ Terabyte server).

3.4.5 Btrfs File System:

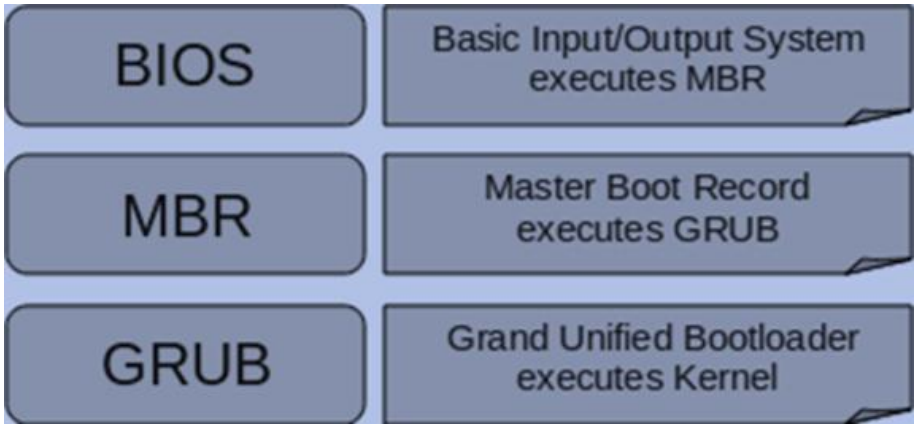
Btrfs stands for the **B tree file system**. It is used for fault tolerance, repair system, fun administration, extensive storage configuration, and more. It is not a good suit for the production system.

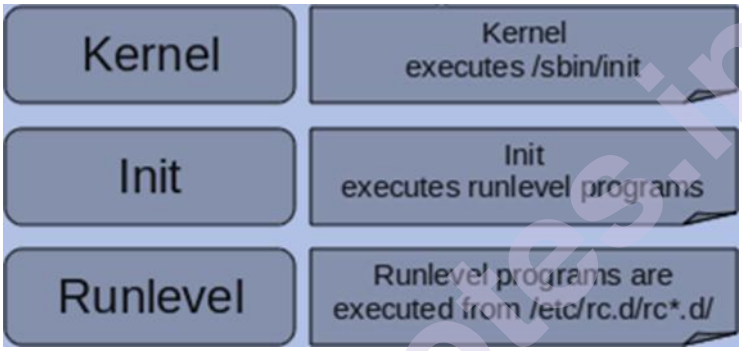
3.4.6 Swap File System:

The swap file system is used for memory paging in Linux operating system during the system hibernation. A system that never goes in hibernate state is required to have swap space equal to its RAM size.

3.5 LINUX BOOT PROCESS

The following are the 6 high level stages of a typical Linux boot process.





3.5.1 BIOS:

- BIOS stands for Basic Input/Output System.
- Performs some system integrity checks.
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, cd-rom, or hard drive. You can press a key (typically F12 of F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

3.5.2. MBR:

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically, /dev/hda, or /dev/sda

- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

3.5.3 GRUB

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this). The following is sample grub.conf of CentOS.

```
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-194.el5PAE)
        root (hd0,0)
        kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/
        initrd /boot/initrd-2.6.18-194.el5PAE.img
```

- As you notice from the above info, it contains kernel and initrd image.
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

3.5.4 Kernel:

- Mounts the root file system as specified in the "root=" in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a 'ps -ef | grep init' and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains

necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

3.5.5 Init:

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
- 0 – halt
- 1 – Single user mode
- 2 – Multiuser, without NFS
- 3 – Full multiuser mode
- 4 – unused
- 5 – X11
- 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
- Execute 'grep initdefault /etc/inittab' on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

3.5.6. Runlevel programs

- When the Linux system is booting up, you might see various services getting started. For example, it might say "starting sendmail …. OK". Those are the runlevel programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
- Run level 0 – /etc/rc.d/rc0.d/
- Run level 1 – /etc/rc.d/rc1.d/
- Run level 2 – /etc/rc.d/rc2.d/
- Run level 3 – /etc/rc.d/rc3.d/
- Run level 4 – /etc/rc.d/rc4.d/
- Run level 5 – /etc/rc.d/rc5.d/

- Run level 6 – /etc/rc.d/rc6.d/
- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

3.6 SHUTDOWN

The shutdown command brings down system in a secure way. All the logged-in users are notified about the system shutdown.

Signal SIGTERM notifies all the processes that the system is going down, so that processes can be saved and exit properly.

Command shutdown signals the init process to change the runlevel.

Runlevel 0 halts the system

Runlevel 6 reboots the system

Runlevel 1 is default state.

Five minutes before shutdown sequence starts, file **/etc/nologin** is created when shutdown is scheduled for future which does not allow new user logins.

If by any reason, command shutdown is stopped before signalling init, this file is removed. It is also removed to change runlevel before signalling init.

To run shutdown command root user access is required.

3.7 VERY BASIC INSTRUCTIONS TO LINUX PROCESS

Anytime you run a program, you have created a process.

3.7.1 List processes:

To display currently active processes, use the `ps` command:

```
[tcarrigan@client ~]$ ps
```

```
  PID TTY          TIME CMD
 2648 pts/0    00:00:00 bash
 3293 pts/0    00:00:00 sleep
 3300 pts/0    00:00:00 ps
```

Here you will get information about the active processes on your system. You will want to pay attention to the **PID** (unique process ID), the **TIME** (amount of time that the process has been running), and the **CMD** (the command executed to launch the process).

3.7.2 Verbose list (processes):

To see an incredibly detailed list of processes, you can use the `ps aux` command.

- a - all users
- u - shows the user/owner
- x - displays processes not executed in the terminal (making the output rather long)

You can see the command here (output edited for length):

```
[tcarrigan@client ~]$ ps aux
USER         PID %CPU %MEM    VSZ   RSS   TTY      STAT START
 TIME COMMAND
tcarrig+    3293  0.0  0.0 215292   520 pts/0    T    13:41   0:00 sleep 500
root        3380  0.0  0.0      0     0 ?        I    13:45   0:00 [kworker/1:1-
mm_percpu_wq]
root        3381  0.0  0.0      0     0 ?        I    13:45   0:00 [kworker/1:3]
root        3398  0.0  0.0      0     0 ?        I    13:46   0:00 [kworker/3:2-
ata_sff]
root        3481  0.0  0.0      0     0 ?        I    13:50   0:00 [kworker/u8:2-
flush-253:0]
root        3482  0.0  0.0      0     0 ?        I    13:50   0:00 [kworker/0:1-
events]
root        3483  0.0  0.0      0     0 ?        I    13:50   0:00 [kworker/0:2]
root        3508  0.0  0.0      0     0 ?        I    13:51   0:00 [kworker/3:0-
ata_sff]
root        3511  0.0  0.0  18892  7732 ?        S    13:52   0:00 systemd-
userwork
root        3512  0.0  0.0  18892  7656 ?        S    13:52   0:00 systemd-
userwork
```

```
root        3513  0.0  0.0  18892  7656 ?        S    13:52   0:00 systemd-userwork

root        3566  0.4  0.0 432792  8024 ?        Ssl  13:54   0:00 /usr/libexec/fprintd

tcarrig+    3598  0.0  0.0 228208  3948 pts/0    R+   13:54   0:00 ps aux
```

3.7.3. Kill by PID:

Inevitably, a process will get hung, and you will need to `kill` it. The more time you spend at the CLI, the more likely it is you will need the `kill` command. The most accurate way to identify a process is by process ID (PID).

Use the following syntax:

```
[tcarrigan@client ~]$ kill PID
```

This command sends the **SIGTERM** signal. However, if you are dealing with a stuck process, add the `-9` option.

```
[tcarrigan@client ~]$ ps
  PID TTY          TIME CMD
 2648 pts/0    00:00:00 bash
 3293 pts/0    00:00:00 sleep
 4684 pts/0    00:00:00 sleep
40527 pts/0    00:00:00 sleep
40540 pts/0    00:00:00 ps
[tcarrigan@client ~]$ sudo kill -9 3293
[sudo] password for tcarrigan:
[1]   Killed                  sleep 500
```

3.7.4 Kill by name/keyword:

Use the `killall` command to kill a process by name. This command will kill all processes with the keyword/name that you specify.

The syntax is:

```
[tcarrigan@client ~]$ killall sleep
```

This would kill all `sleep` processes active on the system (the `-9` option works here as well). Here is an example:

```
[tcarrigan@client ~]$ ps
```

```
  PID TTY          TIME CMD
 2648 pts/0    00:00:00 bash
 4684 pts/0    00:00:00 sleep
40527 pts/0    00:00:00 sleep
40540 pts/0    00:00:00 ps
```

```
[tcarrigan@client ~]$ killall -9 sleep
[2]-  Killed                  sleep 500
[3]+  Killed                  sleep 500
```

These next two commands go hand in hand. They allow you to move/manage background commands.

3.7.5 List background jobs and resume background jobs:

To list and manage background jobs, we will use the `bg` command. I started a new `sleep 500` process and then stopped it, sending it to the background. Thus we see it listed when running `bg` below:

```
[tcarrigan@client ~]$ bg
[1]+ sleep 500 &
```

3.7.6 Bring the most recent job to the foreground:

To do this, we are going to use the `fg` command. This brings the most recently run job/process to the foreground. The following example is a continuation of the above command. The `sleep 500` process that is in the background is now active in the background. Let's bring it into the light...

```
[tcarrigan@client ~]$ fg
sleep 500
```

This command brings us to our final command in this list.

3.7.7 Bring a specific job to the foreground:

Use the `fg` command again, but select a specific job to bring to the foreground (instead of the most recent). To do this, we are just going to add the job/process name to the command.

```
[tcarrigan@client ~]$ fg XXXample
```

This brings job **XXXample** to the foreground.

3.8 PACKAGING SYSTEMS

Almost all the software that is installed on a modern Linux system will be found on the Internet. It can either be provided by the distribution vendor

through central repositories (which can contain several thousands of packages, each of which has been specifically built, tested, and maintained for the distribution) or be available in source code that can be downloaded and installed manually.

Because different distribution families use different packaging systems (Debian: ***.deb** / CentOS: ***.rpm** / openSUSE: ***.rpm** built specially for openSUSE), a package intended for one distribution will not be compatible with another distribution. However, most distributions are likely to fall into one of the three distribution families covered by the LFCS certification.

3.8.1 High and low-level package tools:

In order to perform the task of package management effectively, you need to be aware that you will have two types of available utilities: **low-level** tools (which handle in the backend the actual installation, upgrade, and removal of package files), and **high-level** tools (which are in charge of ensuring that the tasks of dependency resolution and metadata searching -"data about the data"- are performed).

DISTRIBUTION	LOW-LEVEL TOOL	HIGH-LEVEL TOOL
Debian and derivatives	Dpkg	apt-get / aptitude
CentOS	Rpm	yum
openSUSE	Rpm	zypper

Let us see the descrption of the low-level and high-level tools.

dpkg is a low-level package manager for Debian-based systems. It can install, remove, provide information about and build *.deb packages but it can't automatically download and install their corresponding dependencies.

apt-get is a high-level package manager for Debian and derivatives, and provides a simple way to retrieve and install packages, including dependency resolution, from multiple sources using the command line. Unlike dpkg, apt-get does not work directly with *.deb files, but with the package proper name.

aptitude is another high-level package manager for Debian-based systems, and can be used to perform management tasks (installing, upgrading, and removing packages, also handling dependency resolution automatically) in a fast and easy way. It provides the same functionality as apt-get and additional ones, such as offering access to several versions of a package.

rpm is the package management system used by Linux Standard Base (LSB)-compliant distributions for low-level handling of packages. Just

like dpkg, it can query, install, verify, upgrade, and remove packages, and is more frequently used by Fedora-based distributions, such as RHEL and CentOS.

yum adds the functionality of automatic updates and package management with dependency management to RPM-based systems. As a high-level tool, like apt-get or aptitude, yum works with repositories.

3.9 GRAPHICAL VS COMMAND LINE

CLI is that the word form used for **Command Line Interface.** CLI permits users to put in writing commands associate degree exceedingly in terminal or console window to interact with an operating system. CLI is a platform or medium wherever users answer a visible prompt by writing a command and get the response from system, for this users have to be compelled to kind command or train of command for performing the task. CLI is suitable for the pricey computing wherever input exactitude is that the priority.

GUI stands for **Graphical User Interface.** GUI permits users to use the graphics to interact with an operating system. In graphical user interface, menus are provided such as : windows, scrollbars, buttons, wizards, painting pictures, alternative icons etc. It's intuitive, simple to find out and reduces psychological feature load. In GUI, the information is shown or presented to the user in any form such as: plain text, videos, images, etc.

Let's see that the difference between GUI and CLI:

S.NO.	CLI	GUI
1.	CLI is difficult to use.	Whereas it is easy to use.
2.	It consumes low memory.	While consumes more memory.
3.	In CLI we can obtain high precision.	While in it, low precision is obtained.
4.	CLI is faster than GUI.	The speed of GUI is slower than CLI.
5.	CLI operating system needs only keyboard.	While GUI operating system need both mouse and keyboard.
6.	CLI's appearance can not be modified or changed.	While it's appearance can be modified or changed.
7.	In CLI, input is entered only at command prompt.	While in GUI, input can be entered anywhere on the screen.
8.	In CLI, the information is shown or presented to the user in plain text and files	While in GUI, the information is shown or presented to the user in any form such as: plain text, videos, images, etc.
9.	In CLI, there are no menus provided.	While in GUI, menus are provided.
10.	There are no graphics in CLI.	While in GUI, graphics are used.
11.	CLI do not use any pointing devices.	While it uses pointing devices for selecting and choosing items.

12.	In CLI, spelling mistakes and typing errors are not avoided.	Whereas in GUI, spelling mistakes and typing errors are avoided.

3.10 UNIT END QUESTIONS

1. What is Linux File System?
2. Briefly explain Linux File System Structure
3. Explain various types of Linux File System
4. Explain Linux Boot Process
5. Briefly explain about Packaging systems
6. Write difference between CUI and GUI

3.11 LIST OF REFERENCES

1. The Linux Programming Interface: A Linux and UNIX System Programming Handbook 1st Edition by Michael Kerrisk
2. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward
3. The Linux Command Line: A Complete Introduction 1st Editionby by William E. Shotts Jr.
4. Fundamentals of Linux by Pelz Oliver

3.12 BIBLIOGRAPHY

1. Linux Command Line and Shell Scripting Bible, 3rd Edition by Richard Blum
2. Linux: The Complete Reference, Sixth Edition by Richard Petersen
3. How Linux Works, 2nd Edition: What Every Superuser Should Know Second Edition by Brian Ward
4. The Linux Command Line: A Complete Introduction 1st Edition by by William E. Shotts Jr.
5. Fundamentals of Linux by Pelz Oliver

*****

UNIT II

4

GRAPHICAL DESKTOP

Unit Structure

4.1 GRAPHICAL DESKTOP

The Linux Desktop Environment:

In the early days of Linux simple text interface to the Linux operating system was available. This text interface allowed administrators to start programs, control program operations, and move files around on the system.

But now due to Microsoft Windows awareness the Linux graphical desktops are introduced.

The X Windows System:

X windows is designed for flexibility and there are various ways you can configure it. on X windows you can run most of the different video cards available & different graphics cards. The X Windows software is the core element in presenting graphics. It provides an graphics operations.

To run X-windows, the X free 86 server for appropriate system video card has to be installed and configuration information provided about your monitor, mouse and keyboard. This information resides in the configuration file called /etc/xF86 config. The file uses technical information that is best generated by an X-windows.

There are two basic elements that control your video environment — the video card in your PC and your monitor. The X Windows software is a low-level program that works directly with the video card and monitor in the PC, and controls how Linux applications can present fancy windows and graphics on your computer.

In the Linux world, there are only two software packages that can implement it.

The XFree86 software package is the older of the two, and for a long time was the only X Windows package available for Linux. As its name implies, it's a free open source version of the X Windows software.

A new package called X.org has come onto the Linux scene. It too provides an open source software implementation of the X Windows system. Both packages work the same way, controlling how Linux uses your video card to display content on your monitor. To do that, they have to be configured for your specific system. During installation it automatically happens.

The core X Windows software produces a graphical display environment, but nothing else. There is no desktop environment allowing users to manipulate files or launch programs. To do that, you need a desktop environment on top of the X Windows system software.

The KDE Desktop:

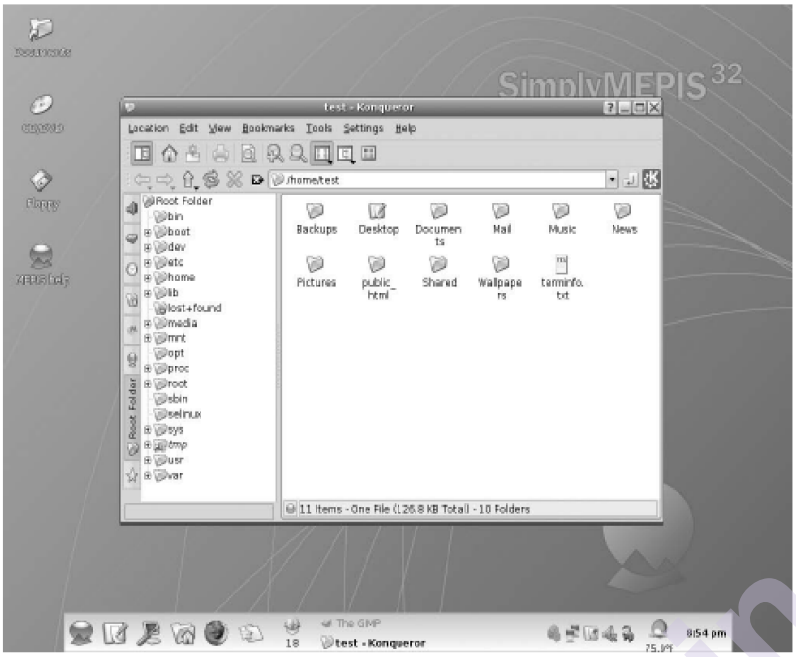
The K Desktop Environment (KDE) was first released in 1996 as an open source project to produce a graphical desktop similar to the Microsoft Windows environment. The KDE desktop incorporates all of the features you are probably familiar with if you are a Windows user.

The KDE desktop allows you to place both application and file icons on the desktop area. If you single-click an application icon, the Linux system starts the application. If you single-click on a file icon, the KDE desktop attempts to determine what application to start to handle the file.

The bar at the bottom of the desktop is called the Panel. The Panel consists of four parts:

- **The K menu:** Similarly to the Windows Start menu, the K menu contains links to start installed applications.
- **Program shortcuts**: These are quick links to start applications directly from the Panel.
- **The taskbar:** The taskbar shows icons for applications currently running on the desktop.
- **Applets:** These are small applications that have an icon in the Panel that often can change depending on information from the application.

The KDE desktop on a SimplyMEPIS Linux system



KDE Applications:

KDE Applications	
Application	**Description**
amaroK	Audio file player
digiKam	Digital camera software
K3b	CD-burning software
Kaffeine	Video player
Koffice	E-mail client
Konqueror	File and Web browser
Kontact	Personal information manager
Kopete	Instant messaging client

Application Description:

All of the Panel features are similar to what you would find in Windows. Besides the desktop features, the KDE project has produced a wide area of applications that run in the KDE environment.

The GNOME Desktop:

The GNU Network Object Model Environment (GNOME) is another popular Linux desktop environment.

First released in 1999, GNOME has become the default desktop environment for many Linux distributions (the most popular being Red Hat Linux).

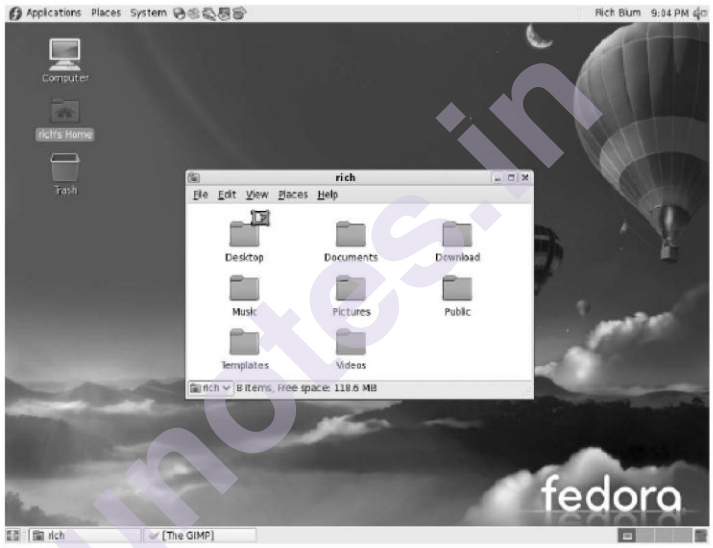
While GNOME choose to depart from the standard Microsoft Windows look-and-feel, it incorporates many features that most Windows users are comfortable with:

- A desktop area for icons.
- Two panel areas.
- Drag-and-drop capabilities.

GNOME developers have also produced a host of graphical applications that integrate with the GNOME desktop. As you can see, there are also quite a few applications available for the GNOME desktop. Besides all of these applications, most Linux distributions that use the GNOME desktop also incorporate the KDE libraries, allowing you to run KDE applications on your GNOME desktop.

Figure shows the standard GNOME desktop used in the Fedora Linux distribution.

A GNOME desktop on a Fedora Linux system



Other Desktops:

The downside to a graphical desktop environment is that they require a fair amount of system resources to operate properly. In the early days of Linux, a hallmark and selling feature of Linux was its ability to operate on older, less powerful PCs that the newer Microsoft desktop products couldn't run on. However, with the popularity of KDE and GNOME desktops, this hallmark has changed, as it takes just as much memory to run a KDE or GNOME desktop as the latest Microsoft desktop environment.

4.2 SESSION MANAGEMENT

The screen or GNU screen is a terminal multiplexer. Using this, you can run any number of console-based-applications, interactive command shells, course-based applications, etc. When screen is called, it creates a single window with a shell in it (or the specified command) and then gets out of your way so that you can use the program as you normally would. Then, at any time, you can create new (full-screen) windows with other

programs in them (including more shells), kill the current window, view a list of the active windows, copy text between windows, switch between windows, etc.

Screen manages a *session* consisting of one or more *windows* each containing a shell or other program. Furthermore, screen can divide a terminal display into multiple *regions*, each displaying the contents of a window. All windows run their programs completely independent of each other. Programs continue to run when their window is currently not visible and even when the whole screen session is detached from the user's terminal. This is practical to prevent involuntary ssh timeout session.

1. Screen:

screen

You can start a new window within the screen and also gives a name to the window, for example aloft. It creates a session with identified by that name. The name can be used to reattach at a later stage.

OnlyFans founder resigns from CEO position

screen -S aloft

Note that you can do all your work as you are in the normal CLI environment.

2. List all the screen processes:

As we are able to start new windows within the screen, it is possible to display the currently opened screens including those running in the background. It will list all the existing screen sessions.

```
# screen -ls
```

There is a screen on:

10437.aloft (Attached)

1 Socket in /var/run/screen/S-root.

3. Main command of screen:

Ctrl-a followed by c: create a new windows

Ctrl-a followed by w: display the list of all the windows currently opened

Ctrl-a followed by A: rename the current windows. The name will appear when you will list the list of windows opened with Ctrl-a followed by w.

Ctrl-a followed by n: go to the next windows

Ctrl-a followed by p: go to the previous windows

Ctrl-a followed by Ctrl-a: back to the last windows used.

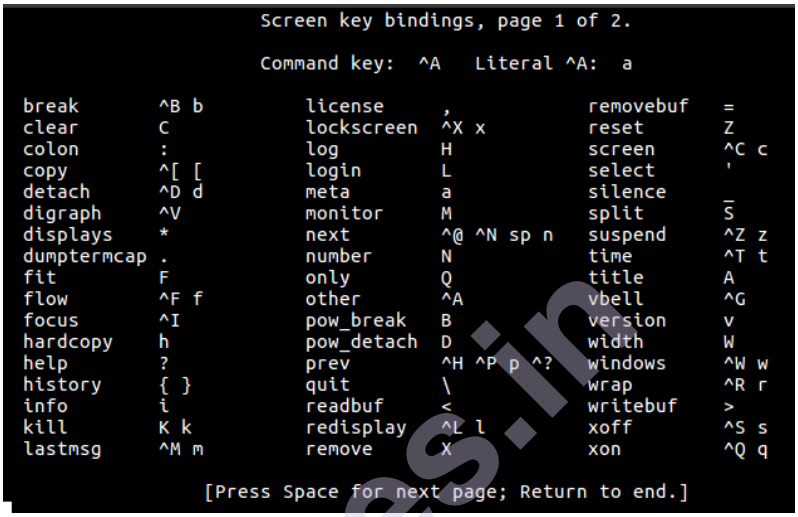
Ctrl-a followed by a number from 0 to X: go the windows n° X.

Ctrl-a followed by ": choose the windows into which to move on.

Ctrl-a followed by k: close the current windows (kill)

4. Show screen parameter:

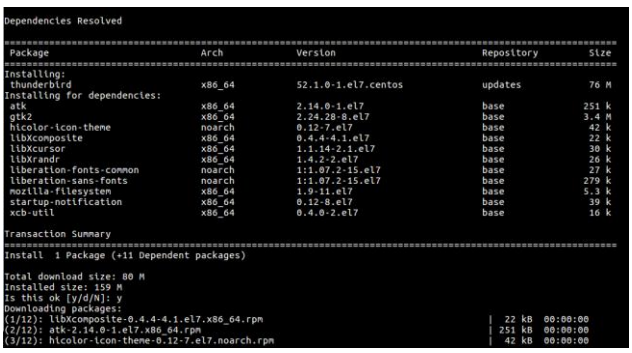
You can list all screen's parameters for help. To do this, type Ctrl-a followed by the character ?. It will display a list of all the commands.



5. Detaching session:

The best advantage of the screen command is the possibility to detach a screen session. You can start a screen session on one computer at the office, detach the session from the local terminal, go home, log into our office computer remotely and reattach the screen session to our home computer's terminal. During the intervening time, all jobs on your office computer have continued to execute. This function is used to prevent the lost of data which occur suddenly during dropped ssh connection.

To good understand what we are talking about, let us take an example. We launch an installation process.



Now we will detach the screen with Ctrl-a followed by d. We can check with the command below.

```
# screen -ls
```

There is a screen on:

12449.win (Detached)

```
1 Socket in /var/run/screen/S-root.
```

It is possible to detach screen with screen -d command followed by the screen id or its name. It means that you will need to open another windows or console to detach the session if the current console have a process in progress. You first need to list current attached screen.

```
# screen -ls
```

There is a screen on:

13686.win200 (Attached)

```
1 Socket in /var/run/screen/S-root.
```

Now on a new terminal, enter the command below.

```
# screen -d 13686
```

or you can use the name

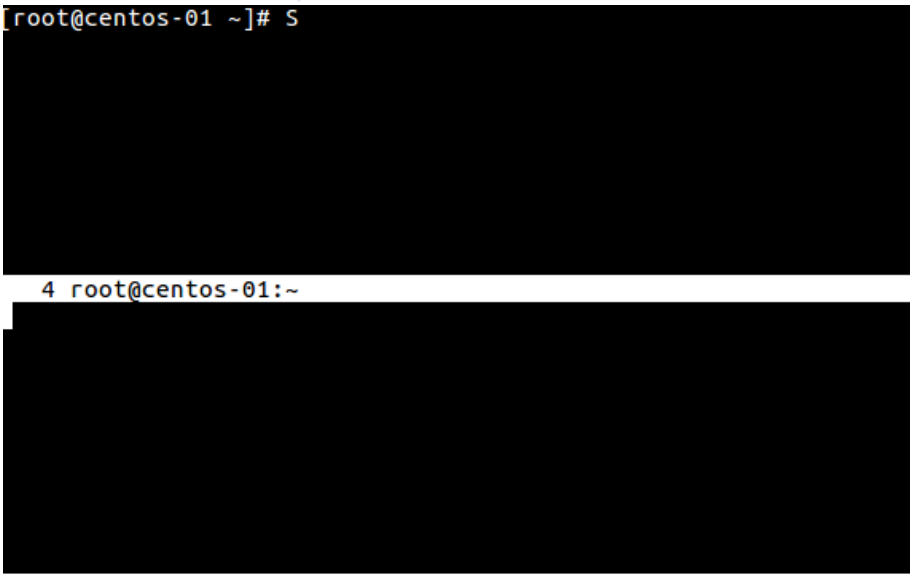
```
# screen -d win200
```

You will have an output as below which indicates that the screen was detached.

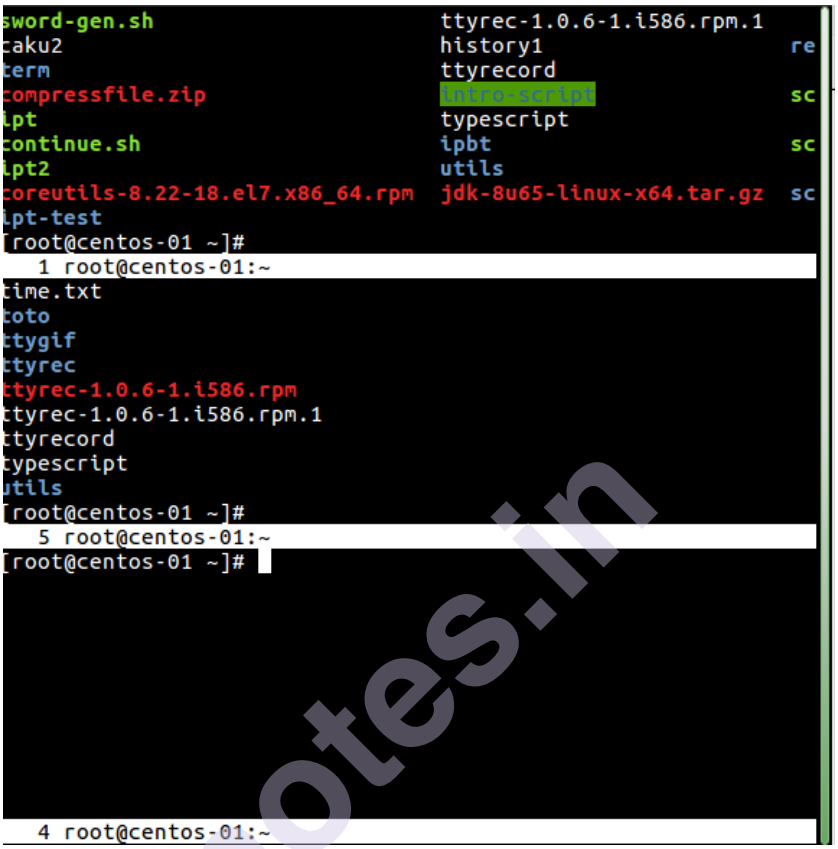
```
[remote detached from 13686.win200]
```

6. Split windows:

To have a global view of your work, you can need to split your windows instead of having multiple windows. Ctrl-a followed by S or | split your screen horizontally or vertically. It is possible to repeat the operation with no limit. To move another windows, use Ctrl-a followed by Tab.



When the cursor is on the bottom windows, you can create a new window (Ctrl-a followed by c) or call an existing window (Ctrl-a followed by a number).



To close a splitted windows, use Ctrl-a followed by X (Note that it is the uppercase character).

7. Reconnect to a disconnected ssh session:

When you first log in,

run screen to start a screen session. You get another shell, run commands in that.

screen -S remote_session

When you have finished, detach the screen session then logout to the ssh

[detached from 20995.remote_session]

You can list all the screen session first

screen -ls

There are screens on:

20995.remote_session (Detached)

14331.daby (Attached)

3 Sockets in /var/run/screen/S-root.

Reconnect to your screen session and continue your work

screen -d -r remote_ression

The screen command is most used for ssh session because it helps to continue your work after a disconnection without losing the current processes in progress.

8. Scroll up in screen windows:

Since screen takes over managing your remote programs, you can't use your terminal emulator's scroll features while running screen. You must use the Screen commands to access the scrollback buffer.

Use Ctrl-a followed by escape

Press the Up and Down arrow keys or the PgUp and PgDn keys to scroll through previous output.

You can see where is my cursor on the screenshot. Each virtual terminal has its own scrollback buffer.

9 See the owner of each screen session:

The screen -ls or screen -list commands only show you your own screen sessions even for root. As far as I know that's as good as it gets for screen itself.

If you want to see which screen sessions have been started by which users look in each users directory in /var/run/screen/.

4.3 BASIC DESKTOP OPERATIONS

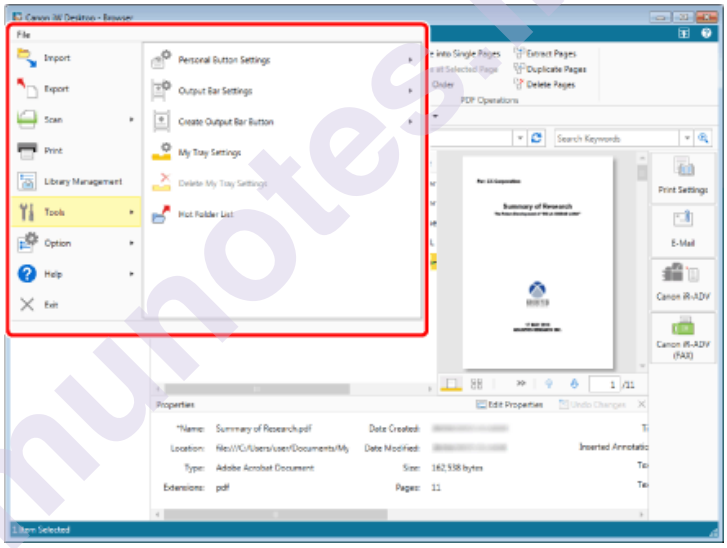
Desktop Browser Screen Layout

The Desktop Browser screen consists of a "[File] menu", "ribbon", "windows", and "bars". Documents and folders can be searched on a separate Search Screen.

[File] menu
Ribbon
Windows
Bars
Search Screen

[File] menu

The [File] menu contains the functions relating to Desktop Browser, such as printing and scanning, and items regarding the configuration of all of Desktop.



Ribbon:

The ribbon includes tabs with commands for performing operations on and editing documents stored in libraries and commands for sending and receiving faxes and printing materials.

Commands are grouped by function and located on tabs. You can switch the displayed commands by clicking the tabs.

The following tabs are displayed on the ribbon of Desktop Browser by default.

[Home]

[Fax]

[Print Meeting Materials]

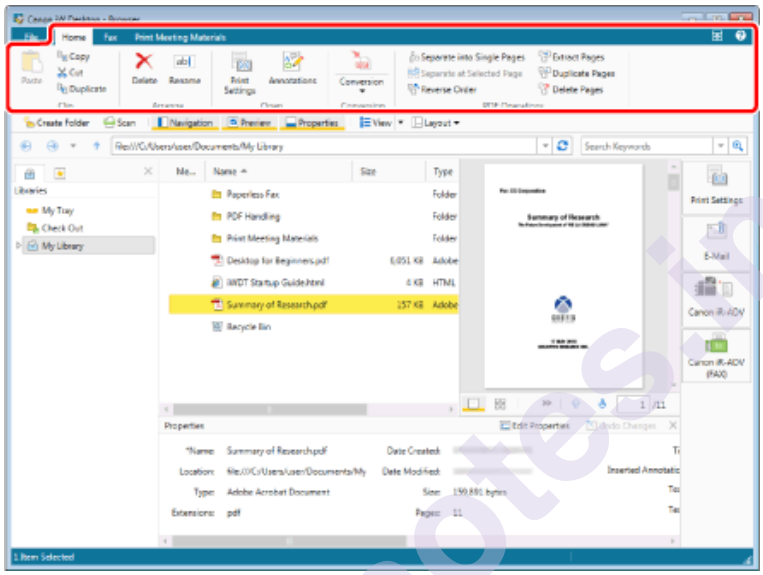
You can also click the following areas of the Navigation Window to display commands related to the area on a tab.

My Tray: Displays the [Edit] tab of the My Tray tool.

Checkout Folder: Displays the [Operations] tab of the document management tool.

Document Server Library: Displays the [Operations] tab of the document management tool.

Recycle Bin: Displays the [Manage] tab of the Recycle Bin tool.



NOTE:

You can hide or display the ribbon by clicking (Hide the Ribbon)/ (Display the Ribbon). For information on hiding/displaying the ribbon, see the following.

Displaying/Hiding Ribbons

If you click (Open Manual), this manual is displayed.

When you use a wheel mouse, you can switch the tabs displayed on the ribbon by rolling the wheel while above the ribbon.

Ribbons can be customized. For more information, see the following.

Customizing Ribbons

For information on the Navigation Window, see the following.

Navigation Window

Windows

Areas of the Desktop Browser screen mainly used for displaying content are called "windows".

This section describes the "windows" of Desktop Browser.

Navigation Window

File List View Window

Preview Window

Properties Window

You can resize Desktop Browser screen and other windows by dragging with the mouse.

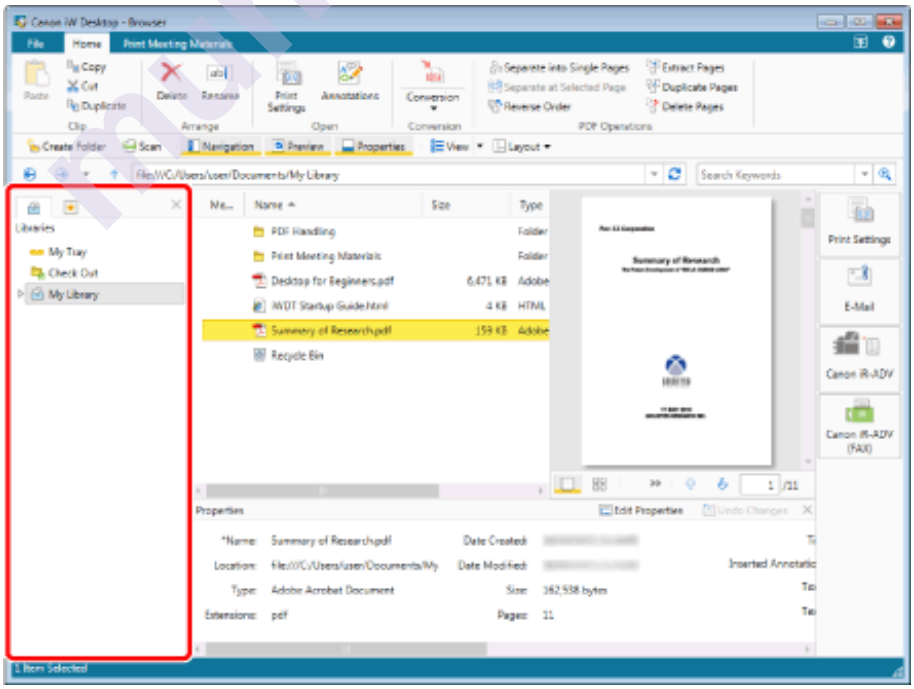
Navigation Window

The Navigation Window is used to organize multiple libraries and list the folder structure inside libraries. It also enables you to list saved search conditions.

The following content is displayed on the Navigation Window of Desktop Browser.

[Libraries] tab: Displays a tree view of libraries, checkout folders, and My Tray folders.

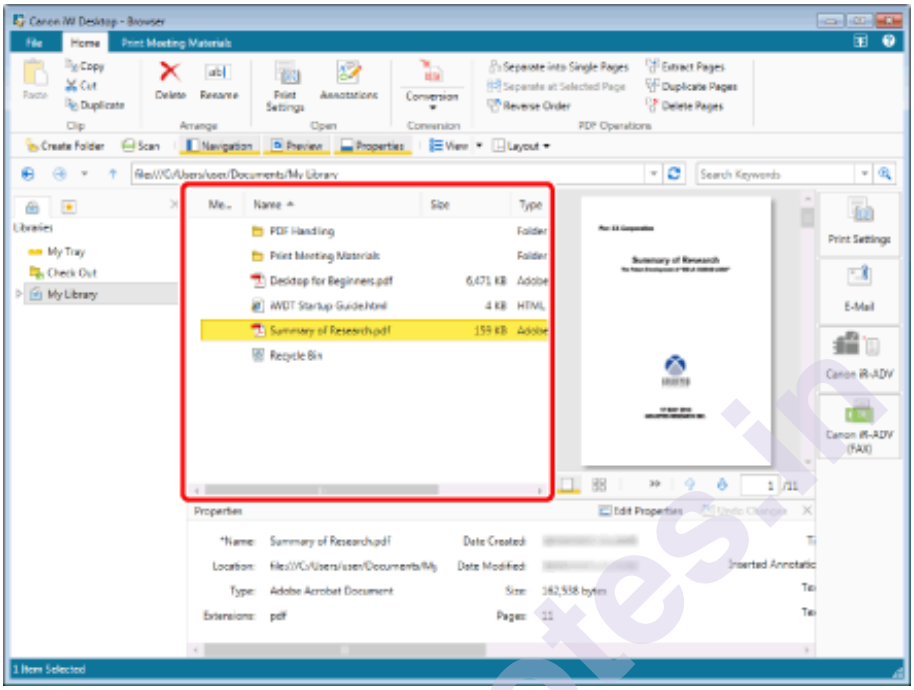
[Search Conditions] tab: Displays the saved search conditions.



File List View Window:

The File List View Window displays the content of the libraries and folders selected on the Navigation Window. You can change the File List View Window to the list view or thumbnail view.

You can also add memos to documents on the File List View Window.



NOTE

The list view can be displayed in ascending or descending order.

The thumbnail view can be changed to [Large Thumbnail], [Medium Thumbnail], and [Small Thumbnail].

You can return the File List View Window to the previous display with (Back), or move forward with (Forward) on the address bar. For information on the address bar, see the following.

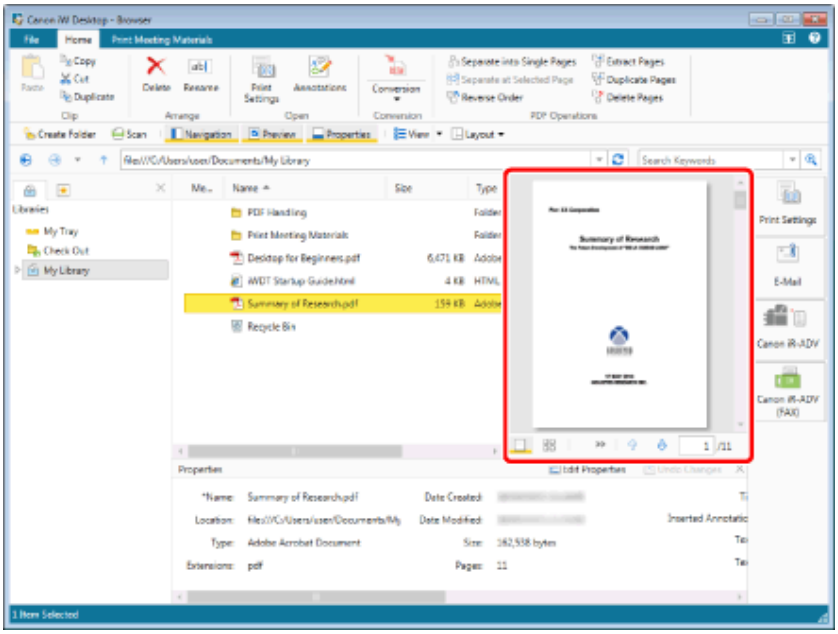
Address Bar

For more information on how to switch the display format of the File List View Window, see the following.

Selecting from the File List View Window

Preview Window:

The Preview Window displays a preview of the document selected on the File List View Window.



NOTE

You can switch between displaying/hiding the Preview Window. For more information, see the following.

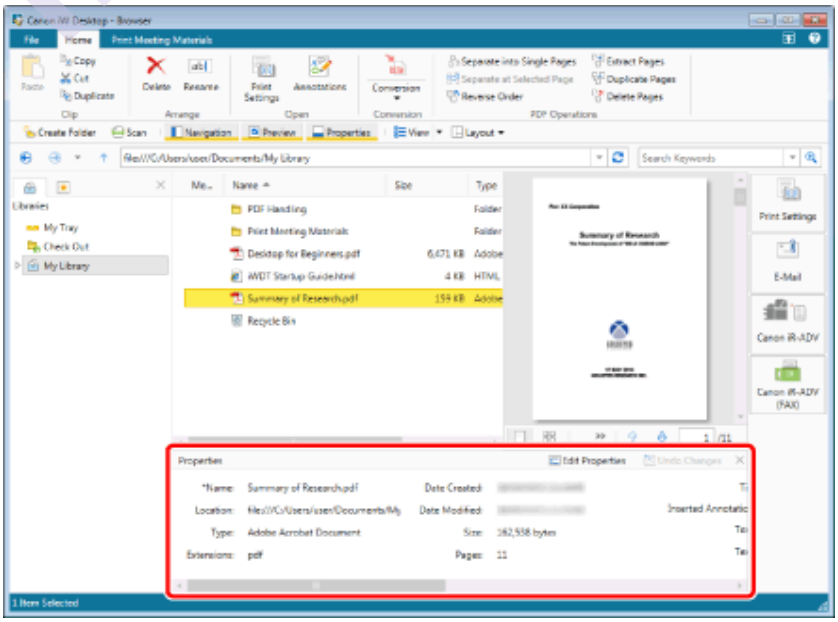
Displaying/Hiding the Preview Window

For information on operating the Preview Window, see the following.

Displaying Documents in the Preview Window

Properties Window:

The Properties Window displays the information (properties) of the document or folder selected on the Navigation Window or File List View Window.



NOTE

You can switch between displaying/hiding the Properties Window. For more information, see the following.

Displaying/Hiding the Properties Window

Bars

Areas of the Desktop Browser screen mainly used for operations and with items arranged horizontally or vertically are called "bars".

This section describes the "bars" of Desktop Browser.

Address Bar

Simple Search Bar

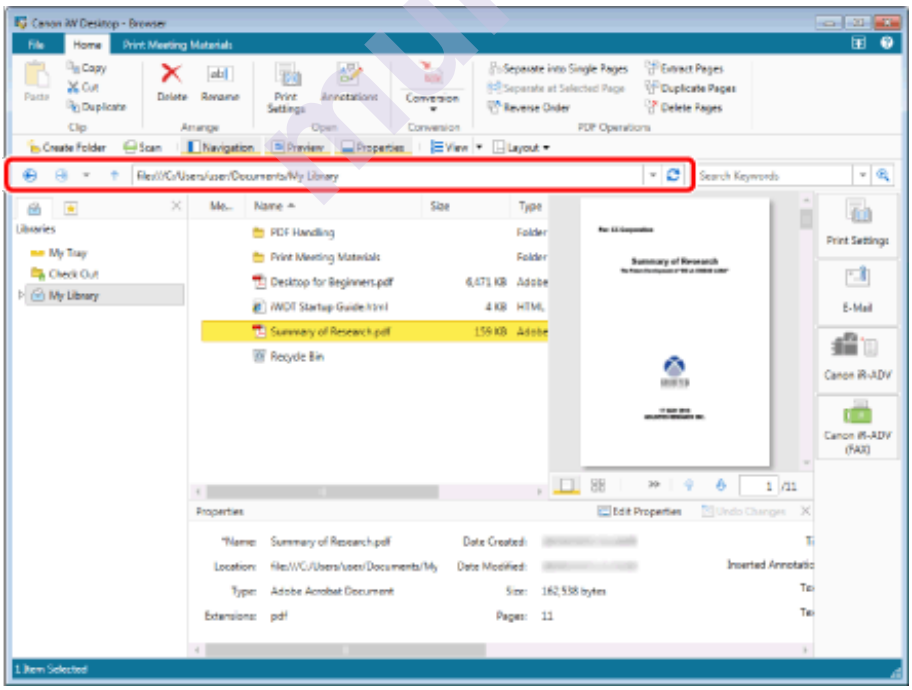
Toolbar

Output Bar

Status Bar

Address Bar

The address bar shows the path of a selected library, My Tray, or folder. You can also click (Back), (Forward), (Up), (Refresh), or (Go) on the address bar to switch the view of the File List View Window.



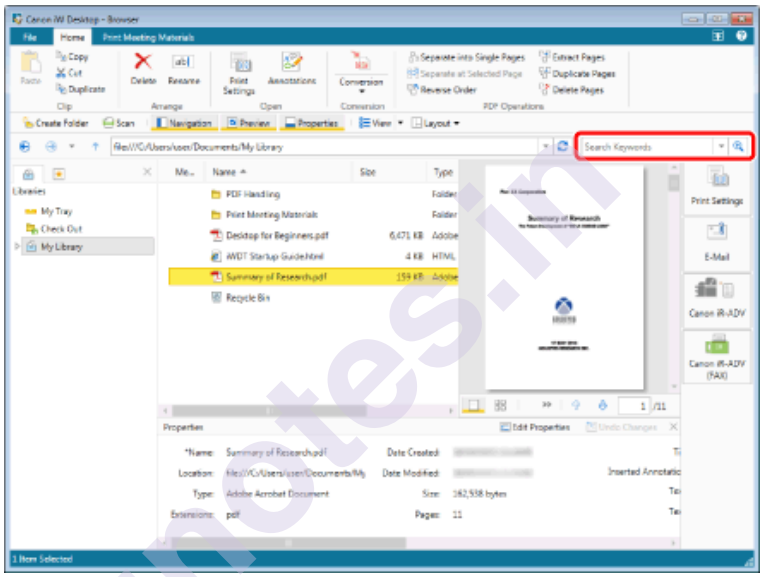
NOTE

By clicking (Refresh), the File List View Window display is updated.

If you select the address bar, (Refresh) changes to (Go). If you click (Go), the screen moves to the location entered in the address bar.

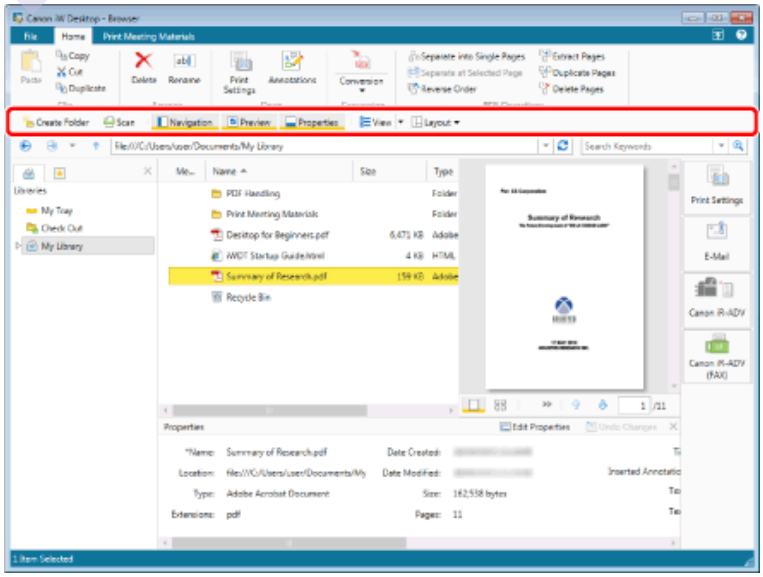
Simple Search Bar:

The simple search bar provides the bar for entering a keyword(s) and performing a simple search.



Toolbar:

The toolbar enables you to set frequently used ribbon commands on the toolbar.



NOTE

For information on the ribbon commands that can be set as buttons on the toolbar, see the following.

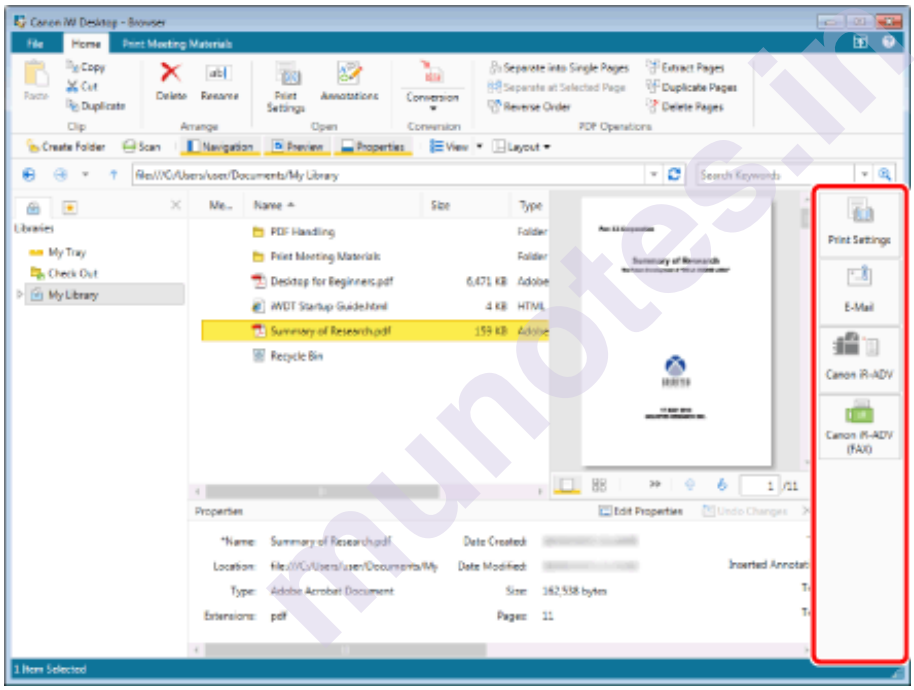
Lists of Ribbon Commands/Toolbars

The toolbar can be customized. For more information, see the following.

Customizing Toolbars

Output Bar:

The output bar enables you to configure settings for linkage with applications and devices and use them. You can also use it for specifying shared folders and circulating documents.



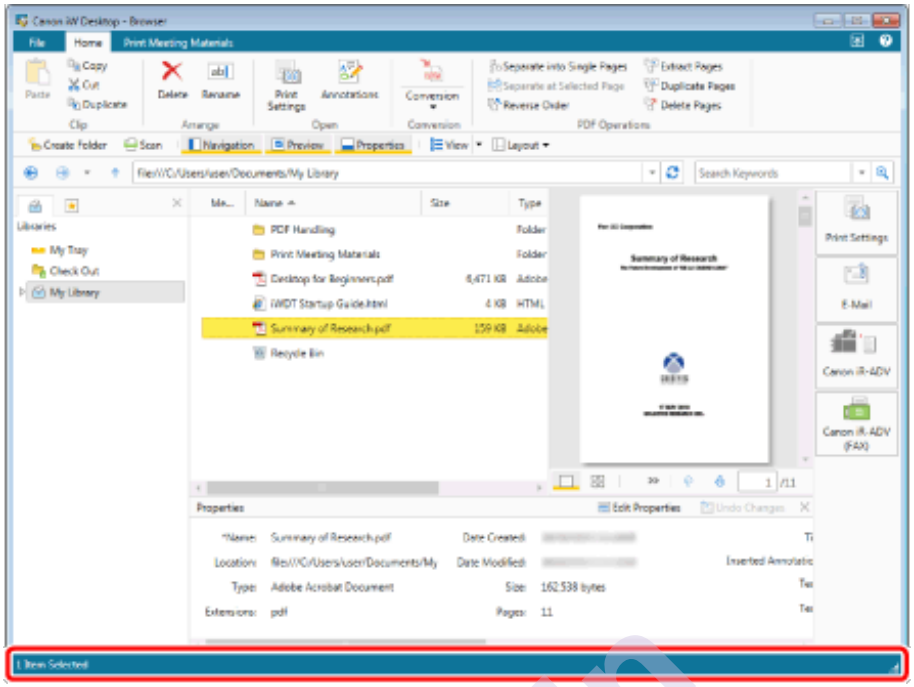
NOTE

The output bar can be customized. For more information, see the following.

Changing the Color of Output Bar Buttons

Status Bar:

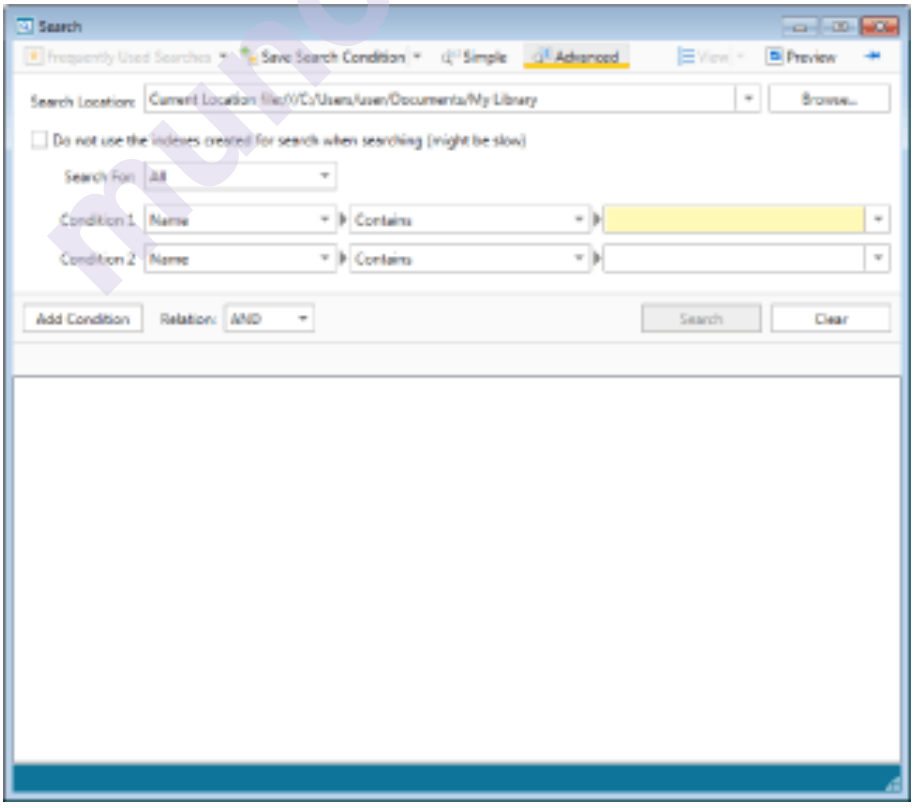
The status bar displays the number of files selected on the File List View Window. If no files are selected, the status bar displays the total number of folders and files displayed on the File List View Window.



Search Screen

The Search Screen displays the results of searching from the simple search bar.

The Search Screen also provides the items for configuring detailed search conditions, performing a search, and saving search conditions.



NOTE

To display the Search Screen always in front of the Desktop Browser screen, click (Display in Front of Browser) to make it . Click it again to cancel.

For information on searching for documents and folders in Desktop Browser, see the following.

Searching for Documents/Folders

4.4 Network Management

4.4.1 TCP/IP Basics:

Remote Access with Linux: Remote Access with Linux can be performed using TCP/IP or UUCP network protocols.

TCP / IP Remote Access operations: the TCP / IP remote commands allow you to log in remotely to accounts on other system. You can also copy files and execute linux commands on those systems. You can also obtain information about other systems, such as who is currently logged on. However, for your remote commands to work on a remote system, you must first given access by that remote system. To provide such a access by that remote system needs to have a **.rhosts** file that lists your system name and login name. The various TCP / IP commands are as follows: All the commands are preceded by alphabet **r.**

$ rwho: It displays **all** the users currently logged into the network

$ ruptime: It displays information about each system on your network. The **information** shows how each system has been performing, whether a system is up or down, how long, it has been up or down, the no of users on the system etc.

$ ping: The ping **command** detects whether or not a system is up and running. It takes the name of the system which you want to check as argument.

Eg **$ ping violet**.

Output: **violet is alive**.

$ ping garnet

Output: **no answer from garnet**

Remote Access Permissions (rhosts): To control the access to our system (from other users who are using TCP/IP) we use rhosts file. To achieve communication between you and others the rhosts file on both system (yours and others) should have each others system name and login name. The rhosts file which is located in users home directory is a simple way to

allow people access to your account without giving out your password. To deny access to a user, simply delete the user's system name and login name from your rhosts file

$ cat .rhosts

Output: garnet chris

Violet Robert

Remote Login: It is quite possible that we have no of account on different systems in a network. If sitting on one system if we want to log into our account on other system on a network then it can be done with rlogin command. rlogin command takes system name as argument. **$ rlogin violet**

As soon as we execute the rlogin command we are immediately prompted for the password. During executing this command we have passed only the system name but not the login name. This is because rlogin command assumes that the login name on your local system is same as that login name on the remote system.(login name on local system means the login name of current system on which you are currently working). But suppose that you are having different login name on remote system then we use the following command.

$ rlogin *system-name* – l *login-name:* (-l option is used to specify the login name on **remote** system is different from the current system).

$ rlogin violet – l robert: (The user logs into the system violet using login name robert).

Remote file copy (rcp): The rcp command is used to copy files to and from remote and local systems. rcp is a file transfer utility that operates like the cp command but across the network. The rcp command requires that the remote system have your local system name and login name in its .rhosts file. The syntax is as follows:

$ rcp *system-name:source file system-name:copy-file.*

When copying to remote system, the copy file will be a remote file and will require the **remote** system's name. If the source file is one which is present on your own system then it will not require the system name. For e.g.

$ rcp abc violet:xyz . The file abc since it is present on your own system it will not require the system name, only file name is sufficient. The file abc is copied from the current system to violet system and named as xyz.

$ rcp violet:pqr jkl . The file pqr from violet system is copied to current system and **named** as jkl.

Copying directories: The rcp command with –r option will copy the complete directories to or from the remote systems.

$ rcp –r ***source-directory remote-system-name:copy-directory***

$ rcp –r vijay violet:vijay1 (vijay **directory** from current system is copied to directory vijay1 on violet system)

$ rcp –r violet:manoj manoj1 (**manoj** directory from violet system is copied to directory manoj1 on current system)

$ rcp –r violet:*.c . (All files with .c extension is copied from violet system to current directory on current or local system (**.**))

$ rcp –r abc violet:\ . (Directory abc is copied from current system to current directory of **violet** system).

The asterisk i.e. * and the dot are special shell characters which are evaluated by your local system (**because** you executing the shell command), not by the remote system. If you want that the special characters to be evaluated by the remove system, you must quote it (by using backward slash).

Remote Execution: Some times we need to execute a shell script on remote system, for which we use rsh command. The rsh command will execute a command or shell script on remote system and display the results on your system. Your system name and login name must, of course, be in the remote system's .rhosts file. The syntax of the command is

$ rsh ***remote-system-name Linux-command***

$ rsh violet ls /home/robert (**listing** all the contents of robert directory on violet system)

$ rsh violet ls /home/robert > xyz (all **the** contents from robert directory on violet system is redirected to xyz file on local or current system)

$ rsh violet ls /home/robert ' >' xyz (If we use the quotes for redirection operator then it becomes the part of Linux command evaluated on the remote system. The working of the above command is same as that of earlier command but the only difference is that the file xyz is formed on remote system instead of current system)

The same is true with **all** the Linux commands. For e.g. with pipes

$ rsh violet ls /home/robert | lpr (piped to local or current printer)

$ rsh violet ls /home/robert '|' lpr (piped to remote system's printer)

- **TCP/IP commands:**

(1) You can find out who is logged in, get information about a user an another system.

$ rwho

violet Rebert: tty I sept 10 10:34

garnet cris: tty 2 sept 10 29:22

(2) The ruptime **command** displays information about each system on your network

$ ruptime

Violet up 11+04:10 8 users load 1:20 1:10 1:00 ruptime shows whether a system is up or down, the number of users on the system, and the average load on the system for the last 5, 10 and 15 minutes.

(3) The ping command detects whether or not a system is up and running. The ping command takes as its argument the name of the system you want to check.

$ ping violet → system number or name

violet is a line

(4) It is possible that you could have accounts on different systems in your network. You can perform such a remote login using the rlogin command takes as its argument a system name.

$ rlogin system-name -1 login name

For example, **the** user logs into a system called violet using the login name

Robert

$ rlogin violet -1 robert

password

The option –1 allows you to enter a different login name for the account on the remote system. Because most people use rlogin to access accounts **they** have on other systems with their own login name.

(5) To copy files to and from remote and local system, use the rcp command that operates like the Cp command but across a network connection to a remote system. The rcp command begins with the keyword rcp and has its arguments the source file and copy file names.

$ rcp system-names : source-file system-name : copy file

When copying to a remote systems, the copy file will be a remote file and require the remote system name. The source file is one on your own system **and** does not require a system name. For e.g. The user copies the file whether from his own system to the remote system violet and renames the file Monday.

$ rcp whether violet : Monday

In the next example, the user copies the file Wednesday from the remote system **violet** to his own system and renames the file today.

$ rcp violet : Wednesday today.

(6) At times, you may need to execute a single command on a remote system. The rsh command will execute a linux command on another system and display the results on your own.

The rsh command takes two general arguments, a system name and a Linux command.

The syntax is as follows:

$ rsh remote-system-name Linux-command

In the next example, the rsh command executes an IS command on the remote system. Violet to list the files in the /home/Robert directory on violet.

$ rsh violet ls/more/Robert

or

$ rsh violet Is/more/Robert> my files

This example shows you the list of files on the remote system and sends them to standard output to local system.

4.4.2 Resolving Ip Addresses:

(1) TCP/IP Network Addresses:

TCP/IP address is organized into four segments consisting of numbers separated by periods. This is called the IP address. Part of this address is used for network address, and the other part is used to identify a particular host in the network. The network address identifies the network that a particular host is a part of. Usually, the network part of the address takes up the first three segments and the host takes the last segment. Altogether this forms a unique address with which to identify any computer on a TCP/IP network. For example, in the IP address 199.35.209.72, the network part is 199.35.209 and the host part is 72. The host is a part of a network whose own address is 199.35.209.0.

The IP address of the host, or host address is only one of several addresses you will need in order to connect the host to a network in addition, you will need a network address, or a network mask or netmask. If you provided them at that time, they will all be automatically entered into the appropriate configuration files.

(2) Network Address:

You can easily figure out the network address using your host address. It is a network part of your host address; with the host part set to 0.50 the network address for the host address 199.35.209.72 is 199.35.209.0

Therefore system device the network address form the host address using the netmask.

(3) Broadcast Address:

The broadcast address allows a system to send the same message to all systems on your network at once. As with the network address, you can easily figure it out using your host address; it has the host part of your address set to 255. The network part remains untouched. So the broadcast address for the host address 199.35.209.72 is 199.35.209.255. (You combine the network part with 255 in the host part.)

(4) Gateway Address:

Some networks will have a computer designated as the gateway to other networks. Every connection to and from the network to other networks passes through this gateway computer. If you are on this type of network, you will have to provide the gateway address. If your network does not have a gateway or you use a stand-alone system or dial into an Internet Service Provider, you do not need a gateway address.

Usually a gateway address has the same network part of a hostname, with the host part set to 1. The gateway address for the host address 199.35.209.72 may be 199.35.209.1.

(5) Nameserver Addresses:

Many networks, including the internet, have computers that operate as domain nameservers to translate the domain names of networks and hosts into IP addresses. This makes your computer identifiable on a network, using just your domain name rather than your IP address. You can also use the domain names of other system to reference them, so you don't have to know their IP addresses. You do, however have to know the IP addresses of any domain nameservers for your network.

Even if you are using an internet service provider, you will have to know the address of the domain nameservers that your ISP operates for the internet.

(6) Netmask:

The netmask is used to derive the address of the network you are connected to. The netmask is determined using your host address as a template. All the numbers in the network part of your host address are set to 255, and the host part is set to 0. This, then is your netmask. So the netmask for the host address is 199.35.209.72 is 255.255.0. The network part 199.35.209, has been set to 255.255.255 and the host part, 72 has

been set to 0. Systems can then use your netmask to derive your network address from your host address. They can determine what part of your host address makes up your network address and what those numbers are.

(7) TCP/IP Configuration Files:

A set of configuration files in the /etc directory, are used to set up and manage your TCP/IP network. They specify such network information as host and domain names, IP addresses, and interface options. It is in these files that the IP address and domain names of other Internet hosts that you want you access are entered. If you configured your network during installation, you will already find that information in these files. The netcfg program on your desktop and netconfig program on your command line both provide an easy interface for entering the configuration data for these files.

File	Function
/etc/hosts	Associates hostnames with IP addresses
/etc/networks	Associates domain names with network addresses
/etc/rc.d/init.d/inct	Contain commands to configure your network interface when you boot up
/etc/HOSTNAME	Holds the hostname of your system
/etc/host.conf	Resolves options
/etc/resolu.conf	Contains list of domain nameservers

(8) Identifying Hostnames: /etc/hosts:

Without the unique IP address that the TCP/IP network uses to identify computers, a particular computer could not be located. Since IP addresses are difficult to use or remember, domain names are used instead. For each IP address there is a domain name. When you use a domain name to reference a computer on the network, your system translates it into its associated IP address. This address can then be used by your network to locate that computer.

The responsibility for associating domain names and IP addresses has been taken over by domain nameservers. However, the hosts file is still used to hold the domain names an IP addresses of frequently accessed hosts. Your system will always check your hosts file for the IP address of a domain name before taking the added step of accessing a nameserver.

The format of a domain name entry in the hosts file is the IP address followed by the domain name, separated by a space. You will already find an entry in your hosts file for “localhost” with the IP address 127.0.0.1. Localhost is a special identification used by your computer to enables users on you’re your system to communicate locally with each other. The IP address 127.0.0.1 is a special reserved address used by every computer for this purpose. It identifies what is technically referred to as a loopback device.

(9) Network Name: /etc/networks:

The /etc/networks file holds the domain names and IP addresses of networks that you are connected to, not to domain names of particular computers. Every IP address consists of a network part and a host part. The network part is the network address you will find in the networks file. You will always have an entry in this file for the network portion on your computer's IP address. This is the network address of the network, your computer is connected to.

/etc/HOSTNAME:

The /etc/HOSTNAME file holds your system's hostname. To change your hostname, you change this entry. The netcfg program allows you to change your hostname and will place the new name in /etc/HOSTNAME. Instead of displaying this file to find your hostname, you can use the hostname command.

$ hostname

turtle.trek.com

(1) Network Interfaces and Routes: ifconfig and route:

Your connection to a network is made by your system through a particular hardware interface such as an Ethernet card or a modem. Data passing through this interface is then routed to your network. The ifconfig command configures your network interfaces and the route command will route them accordingly.

Every time you start your system, the network interfaces and their routes have to be established. You can have this done automatically for you when you boot up by placing the ifconfig and route commands for each interface in the /etc/rc.d/init.d/inet initialization fle, which is executed whenever you start your system.

Ifconfig:

The ifconfig command takes as its arguments the name of an interface and an IP address as well as options. Ifconfig then assigns the IP address to the interface. Your system now knows that there is such an interface and that it references a particular IP address. In addition, you can specify whether the IP address is a host of network address. You can use a domain name for the IP address, provided the domain name is listed along with its IP address in the /etc/hosts file. The syntax for the ifconfig command is as follows.

ifconfig interface – host – net – flag address options.

The host – net – flag can be either – host or – net to indicate a host or network IP address.

In the next example, the ifconfig command configures an Ethernet interface.

The ifconfig command can have several options, which set different features of the interface, such as the maximum number of bytes. It can transfer (mtu) or the broadcast address. The up and down option activate and deactivate the interface.

The ifconfig command is very useful for checking on the status of an interface if you enter the ifconfig command along with the name of the interface, information about that interface is displayed

if config etho

Routing:

A packet that is part of transmission takes a certain route to reach its destination. On a large network, packets are transmitted from one computer to another until the destination computer is reached. The route determines where the process starts and what computer your system needs to send the packet to in order for it to reach its destination. On small network routing may be static, that is, the route from one system to another is fixed. One system knows how to reach another, moving through fixed paths. However, on larger networks and on the Internet, routing is dynamic. Your system knows the first computer to send its packet off to, and then that computer takes it from there, passing it on to another that then determines where to pass it on to. For dynamic routing, your system needs to know very little. Static routing however can become very complex, since you have to keep track of all the network connections.

Your routes are listed in your routing table in the /proc/net/route file. To display the routing table, enter route with no arguments.

route

kernel routing table:

Destination	Gateway	Fenmask	flags	metric	Ref	Use	Iface
Loopback	*	255.0.0.0	U	0	0	12	Lo
Pongol.train.com	255.255.255.0	U	0	0	0	0	etho

The different fields are listed in the following table:

Field	**Description**
Destination	Description IP address of the route
Gateway	IP address or hostname of the gateway the route uses * indicates no gateway is used.
Genmask	The network for the route
Flags	Type of route: U = up, H = host, G = Gateway, D = dynamic, M = modifies.

Metric	Metric cost of route
Ref	Number of routes that depend on this one
Window	TCP window for Ax.25 networks.
Use	Number of times used
Iface	Type of interface this route uses

You should have at least one entry in the routing table for the loopback interface. If not, you will have to route the loopback interface using the route command. The IP address for an interface has to be added to the routing table before you can use that interface. You add an address with the route command and the add option.

Route add address:

The next example adds the IP address for the loopback interface to the routing table.

route add 127.0.0.1

If your system is connected to a network, there should be at least one entry in your routing table that specifies the default route. This is the route taken by a message packet when not other route entry leads to its destination. The destination for a default route is the keyword default.

Monitoring Your Network: Ping and Netstat:

With the ping program, you can check to see if you can actually access another host on your network. Ping will send a request to the host for a reply. The host then sends a reply back, and it is displayed on your screen. Ping will continually send such a request until you stop it with a break command, a Ctrl+C. you will see one reply after another scroll by on your screen until you stop the program. If ping can not access a host, it will issue a message saying that the host is unreachable.

The netstat program provides real-time information on the status of your network connections, as well as network status and the routing table.

The netstat command with no options will list the network connections on your system. First, active TCP connections are listed and then the active domain sockets. The domain sockets contain processes used to set up communications between your system and other systems. The various fields are described in the following table.

Field	Description
Proto	Protocol used for the connection: TCP, UDP
Recu-Q	Bytes received but not yet used by the system
Send-Q	Bytes sent to remote system, but not yet confirmed as received.
Local Address	Local hostname and port number
Foreign Address	Remote hostname and port number assigned to a connection, port number can be connection type, such

	as telnet or ftp.
(State)	State of connection to remote host ESTABLISHD, connection established SYN_SENT, trying to make connection SYN_REC, connection being created Fin_WAIT1, connection shutting down CLOSED, connection closed LISTEN, listening for remote connection UNKNOWN, unknown state

Domain socket:

Proto protocol for socket, usually unix

RefCnt number of processes currently in socket

Flag

Type	Mode Socket is access
State	State of the socket FREE, Socket is not used LISTENING, waiting for connection UNCONNECTED, no current connection CONNECTING, trying to make connection CONNECTED, currently connected DISCONNECTING, closing a connection
Path	Path name used by processes to access socket

You can use netstat with the –r option to display the routing table, and netstat with the –i option displays the us are for the different network interfaces. The following table explains the coded information.

netstat –i

kernel interface table:

Iface	Mtu	met	Rx-OK	Rx-ERR	Rx-DRP	Rx-OUR	Tx-OK	Tx-ERR	Tx-DRP	Tx-OVR	flags
LO	2000	0	0	0	0	0	58	0	0	0	BLRU

MTU	Maximum number of bytes for one transmission
RX-OK	Packets received with no errors
RX-ERR	Packets received with errors
RX-DRP	Packets dropped
RX-OVR	Packet overrun errors
TX-OK	Packets sent with no errors
TX-ERR	Packets sent with errors
TX-DRP	Packets dropped in transmission
TX-OVR	Packets dropped in transmission with overrun errors.

Flags	Interface Characteristics
A	Receives packets for multicast addresses
B	Receives broadcasts
D	Debugging is on
I	Loopback interface
M	Promiscuous mode
N	No trailers processed on packets
O	Address resolution protocol is off
P	Point-to-point interface
R	Interface is running
U	Interface is activated, up

Domain Name Service (DNS):

Each computer connected to a TCP/IP network such as the Internet is identified by its own IP address. An IP address is a set of four numbers specifying the location of a network and of a host (a computer) within that network. IP addresses are difficult to remember, so a domain name version of each IP address is also used to identify a host. A domain name consists of two ports, the host name and the domain. The host name is the computer's specific name, and the domain identifies the network that the computer is a part of. The combination of a hostname domain and extension forms a unique name by which a computer can be referenced. The domain can, in turn, be split into further sub-domains.

As you know, a computer on a network can still only be identified by its IP address, even if it has a domain name. You can use a domain name to reference a computer on a network, but this involves using the domain name to look up the corresponding IP address in a database. The network then uses the IP address, not the domain name, to access the computer.

As networks become larger, it becomes impractical and, in the case of the Internet impossible for each computer to maintain its own list of all the domain names and IP addresses. To provide the service of translating domain addresses to IP addresses databases of domain names were developed and placed on their own servers. To find the IP addresses of a domain name, a query is sent to a name server that then looks up the IP address for you and sends it back. In a large network there can be several name servers covering different parts of the network. If a nameserver cannot find a particular IP address, it will send the query on to another name server that is more likely to have it. Nameservers can also provide information such as the company name and street address of a computer or even the person maintaining it.

Nameservers are queried by resolvers. These are programs specially designed o obtain addresses from nameservers.

4.4.3 Telnet:

What is Telnet? Telnet is a user command and an underlying TCP/IP protocol for accessing remote computers. Through Telnet, an

administrator or another user can access someone else's computer remotely. On the Web, HTTP and FTP protocols allow you to request specific files from remote computers, but not to actually be logged on as a user of that computer. With Telnet, you log on as a regular user with whatever privileges you may have been granted to the specific application and data on that computer.If you have an account on a host in local network, you can use telnet with the hostname or IP address as argument:

$ telnet 192.168.35.12

Connected to 192.168.35.12

Login:

Now user can enter the Login name at this prompt and then the password to gain access to the remote machine. After login you can work on any command at remote location.

telnet> !ls –l

4.4.4 FTP:

ftp command can also be used with or without arguments.

ftp ip-address

After establishing a connection with the destination, ftp prompts for the username and password. The local username is prompted as default and if pressed enter, the system would have logged in as default. Termination of ftp is done in two stages. Firstly one has to disconnect from the remote machine with close and then quit ftp either with bye or quit. ftp has all the basic facilities needed to handle files and directories On the remote machine like pwd, ls, cd, mkdir, rmdir, chmod. User can delete single file with delete and multiple files with mdelete or rename a file(rename).

Transferring files:

For the purpose of transfer files can be seen as belonging to two types- ascii and binary. All executables, graphics, word processing and multimedia files belong to binary type. Uploading of files is done with put for single file and for multiple files mput. For downloading get for single file and mget for multiple files.

ftp displays the ftp> prompt when used without argument. Then connection can be established with open command.

ftp works in two stages. First it makes a connection with a remote machine. This is done by invoking ftp with the hostname or later with the open command. After the connection has been established, ftp asks for the username and password. To login after this user command is used along with the username.

Anonymous ftp:

On the internet there are several sites which offer trial and public domain software for downloading. Where a separate account is not there for every user. These sites offer a special user account “anonymous” that has to be used for logging in. these sites are known as anonymous ftp sites. User can only download files from an anonymous site.

4.5 INSTALLING AND UPDATING SOFTWARE

Most people are surprised to see that they have a running, usable computer after installing Linux; most distributions contain ample support for video and network cards, monitors and other external devices, so there is usually no need to install extra drivers. Also common tools such as office suites, web browsers, E-mail and other network client programs are included in the main distributions. Even so, an initial installation might not meet your requirements.

If you just can't find what you need, maybe it is not installed on your system. It may also be that you have the required software, but it does not do what it is supposed to do. Remember that Linux moves fast, and software improves on a daily basis. Don't waste your time troubleshooting problems that might already be resolved.

You can update your system or add packages to it at any time you want. Most software comes in packages. Extra software may be found on your installation CDs or on the Internet. The website of your Linux distribution is a good place to start looking for additional software and contains instructions about how to install it on your type of Linux, see Appendix A. Always read the documentation that comes with new software, and any installation guidelines the package might contain. All software comes with a README file, which you are very strongly advised to read.

4.5.1. What is RPM?

RPM, the RedHat Package Manager, is a powerful package manager that you can use to install, update and remove packages. It allows you to search for packages and keeps track of the files that come with each package. A system is built-in so that you can verify the authenticity of packages downloaded from the Internet. Advanced users can build their own packages with RPM.

An RPM package consists of an archive of files and meta-data used to install and erase the archive files. The meta-data includes helper scripts, file attributes, and descriptive information about the package. Packages come in two varieties: binary packages, used to encapsulate software to be installed, and source packages, containing the source code and recipe necessary to produce binary packages.

Types of RPM Packages:

RPM packages come in two categories: source and binary.

A source RPM can always be recognized because the filename ends with the string ".src.rpm". In a source RPM are not only the original program source code files but scripts that allow the code to be recompiled automatically, to be installed automatically, and to be removed automatically. There are no end-user executable files in a source RPM. Usually, only developers are interested in a source RPM.

A binary RPM contains the end-user compoments of an RPM. Binary RPM filenames identify the host architecture for the contents. For example, the binary RPM file:

bash-3.1-16.1.x86_64.rpm

It contains files only usable on a 64-bit Intel X86 architecture CPU. Other common architecture values include "i386" for 32-bit Intel hosts. Some binary RPM's may be installed on any CPU architecture because their files will work on any host; an example of these ".noarch.rpm" packages is the "tzdata" RPM which contains information about world timezones. To update your system with the latest version of a package, you will need the most recent binary RPM for it.

RPM Naming Scheme:

Each RPM package is contained in a single file. The filename has several fields to fully identify the contents of the package. While the RPM tools themselves do not rely upon the filename itself, you should understand the filename convention to help you identify or download the proper package. Here is an example RPM filename:

bash-3.1-16.1.x86_64.rpm

This RPM is for the BASH shell ("/bin/bash"). The filename is composed of several parts:

[name]-[version]-[release].[arch].rpm

Where,

[name] is the name of the program or package. The [name] is usually assigned by the program's author. In our example, the developers decided to name their product "bash" for reasons that seemed amusing to them.

version] identifies which edition of the software this RPM contains. The [version] number is assigned by the program's author. Using the number allows one to determine which version of the author's sources were used to generate the RPM.

[release] provides the edition number of the RPM file itself and not the version of the author's source files. An updated RPM may be issued to supply a patched version of the author's original software. The patch need

not have come from the original developer, so the RPM [release] gets incremented instead of the [version].

[arch] describes the contents of the RPM and tells whether this file contains the product source (a ".src.rpm"), architecture-independent files (a ".noarch.rpm"), or files which may only be installed on a particular host type (a ".sh.rpm" will work only on a STRONGHOLD embedded processor).

Installing and Removing Files

Note: Usually only one or of an RPM may be installed at once.

Later versions are usually installed using the "-U" (update) RPM function instead of the "-i" RPM function. Common exceptions to the only-one RPM rule are the kernel RPM's. A system commonly has several versions of kernels installed; RPM has a list of which RPM's may have multiple versions installed. To delete one version when several are installed, you must fully-specify the package name and version.

On the x86_64 architecture, it is common to have both the 32-bit ".i386" and the 64-bit ".x86_64" RPM packages installed tosupport both 32-bit and 64-bit applications. Normally, RPM does not display the architecture of a package on a query but you can manually display it.

Installation and Removal:

rpm -i --install (install new RPM; error if already installed)

rpm -U --upgrade (delete existing RPM, if any; install new)

rpm -F --freshen (update RPM only if package already installed)

rpm -e --erase (remove, delete, expunge)

Common Options

Output: -v (verbose – file name), -h (hash)

Preconditions: –nodeps, –replacefiles, –force (BE CAREFUL HERE !!!)

Relocating: –excludepath, –prefix, –relocate, –badreloc, –root

URL Support: ftp, http

Examples:

rpm -ivh binutils-2.11.90.0.8-12.i386.rpm

rpm -Uvh finger-0.17-9-i386.rpm

rpm -Fvh ftp://updates.redhat.com/current/i386/*.rpm

rpm -e diffutils

rpm -e kernel-enterprise-2.4.9-e.12

Hint: Never, ever, use the "-U" option to install a new kernel RPM. The "-U" update function first deletes the current RPM from the system and then attempts to install the new RPM. Any problem that prevents the new RPM from installing will leave the system unbootable. This is not what you want, so always use the "-i" switch to install a kernel RPM.

Queries (Packages and/or Information):

Use a query for information about installed packages. You may query against all installed packages, or a single installed package. You may also find out which RPM supplies a particular file.

rpm -q [packages] [information]

rpm -qa (all installed packages)

rpm -q package_name

rpm -qf (filename)

rpm -qp (package filename)

Information

default (package name)

-i: general information

-l: file list

Examples:

rpm -qa

rpm -q kernel -i (information)

rpm -q kernel -l (files contained in package)

rpm -q kernel --requires (prereqs)

rpm -q kernel --provides (capabilities provided by package)

rpm -q kernel --scripts (scripts run during installation and removal)

rpm -q kernel --changelog (revision history)

rpm -q kernel -queryformat format (rpm --querytags for list of options)

Queries – Verification (Files):

The RPM database contains many attributes about each and every file installed by an RPM. You may verify the current status of the file against the information cataloged by RPM when the package was installed.

rpm -V package_name

rpm -Va (verify all)

rpm -Vf (filename)

rpm -Vp (package filename)

4.6 TEXT EDITORS

4.6.1 Gedit

4.6.2 Vi

4.6.3 Vim

4.6.4 Emacs

4.6.5 Graphics editors

4.6.1 Gedit:

gedit is a full-featured text editor for the GNOME desktop environment. You can use it to prepare simple notes and documents, or you can use some of its advanced features, making it your own software development environment.

Once *gedit* launches, you can start writing right away. To save your text, just click the Save icon in the *gedit* toolbar.

Tab-related Shortcut keys:

Action	Keyboard shortcut
Switch to the next tab to the left	Ctrl + Alt + PageUp
Switch to the next tab to the right	Ctrl + Alt + PageDown
Close tab	Ctrl + W
Save all tabs	Ctrl + Shift + L
Close all tabs	Ctrl + Shift + W
Reopen the most recently closed tab	Ctrl + Shift + T
Jump to *n*th tab	Alt + *n*
New tab group	Ctrl + Alt + N
Previous tab group	Shift + Ctrl + Alt + Page up
Next tab group	Shift + Ctrl + Alt + Page down

Shortcut keys for working with files:

Action	Keyboard shortcut
Create a new document in a new window	Ctrl + N
Create a new document in a new tab	Ctrl + T
Open a document	Ctrl + O
Open the Quick Open window	Alt + O
Save the current document	Ctrl + S
Save the current document with a new filename	Ctrl + Shift + S
Print the current document	Ctrl + P
Print preview	Ctrl + Shift + P
Close the current document	Ctrl + W
Quit gedit	Ctrl + Q

Shortcut keys for editing files:

Action	Keyboard shortcut
Move to the beginning of the current line	Home
Move to the end of the current line	End
Move to the beginning of the document	Ctrl + Home
Move to the end of the document	Ctrl + End
Move the selected word right one word	Alt + Right Arrow
Move the selected word left one word	Alt + Left Arrow
Undo the last action	Ctrl + Z
Redo the last undone action	Ctrl + Shift + Z
Cut the selected text or region and place it on the clipboard	Ctrl + X
Copy the selected text or region onto the clipboard	Ctrl + C
Paste the contents of the clipboard	Ctrl + V
Select all text in the file	Ctrl + A
Delete the current line	Ctrl + D
Move the selected line up one line	Alt + Up Arrow
Move the selected line down one line	Alt + Down Arrow

Replace text in gedit:

Open the Replace tool by clicking Menu button ▸ Find and Replace… or press Ctrl+H.

Enter the text that you wish to replace into the Find field.

Enter the new, replacement text into the Replace with field.

Once you have entered the original and replacement text, you can add extra parameters to the search. You can also choose what you want to replace:

To replace only the next match, click Replace.

To replace all occurrences of the searched-for text, click Replace All.

Print Preview:

Prior to printing your document, you can preview how the printed document will look by using Print Preview. To preview the document:

Select File ▸ Print Preview. Alternatively, you can press Shift+Ctrl+P.

Printing To Paper:

You can print your documents to paper using a local or remote printer. To print a file:

Select File ▸ Print ▸ General.

Select the desired printer from the list of printers available.

You can preview the file using Print Preview and once you are satisfied with the settings, click Print to send the file to printer.

Printing To File:

You can also use gedit to print to a file. To print your document to file of a different format:

Select File ▸ Print ▸ Print to File.

Printing is enabled for the following file formats, you may select from:

Portable Document Format (.pdf)

PostScript (.ps)

Scalable Vector Graphic (.svg)

To print the document to file, click Print

Search for text:

The Find tool can help you find specific sequences of text within in your file.

Finding text

Open the search window by clicking Menu Button ▸ Find… or pressing Ctrl+F. This will move your cursor to the start of the search window.

Type the text you wish to search for in the search window.

As you type, gedit will begin highlighting the portions of text that match what you have entered.

To scroll through the search results, do any of the following:

Click on the up or down facing arrows next to the search window.

Press the up arrow or down arrow keys on your keyboard.

Press Ctrl+G or Ctrl+Shift+G.

To close the search window, press either Esc or Enter. Pressing Esc will return the cursor to where it was before you began your search. Pressing Enter will return the cursor to the current position in the search results.

Undo a recent action:

If you make a mistake while using gedit, you can undo it by pressing Ctrl+Z.

Create a new file:

The easiest way to create a new file in gedit, is to click the Create a new document button on the left side of the toolbar, or press Ctrl+T.

Any one of these actions will create a new file in the gedit window. If you have other files open in gedit, the new file that you create will appear as a new tab to the right of those files.

Open a file or set of files:

To open a file in gedit, click the Open button, or press Ctrl+O.

This will cause the Open dialog to appear. Use your mouse or keyboard to select the file that you wish to open, and then click Open. The file that you've selected will open in a new tab.

To close the Open dialog without opening a file, click Cancel.

Save a file:

To save a file in gedit, click on the Save button on the right side of the toolbar or just press Ctrl+S.

If you are saving a new file, a dialog will appear, and you can select a name for the file, as well as the directory where you would like the file to be saved.

Reopen a recently-used file:

By default, gedit provides easy access to five of your most recently-used files. Here is how you can open a recently-used file:

Click the Open button.

gedit will display a list of the five most-recently used files.

Select the desired file, and it will open in a new tab.

Ref : https://help.gnome.org/

4.6.2 vi/vim Editor:

In the Linux family, the VI editor is the most popular and classic text editor. Here are some of the reasons why it is such a popular editor.

1) It's included in practically every Linux distribution.

2) It is compatible with a variety of systems and distributions.

3) It is user-friendly. Hence, millions of Linux users love it and use it for their editing needs

Nowadays, there are advanced versions of the vi editor available, and the most popular one is VIM which is Vi Improved.

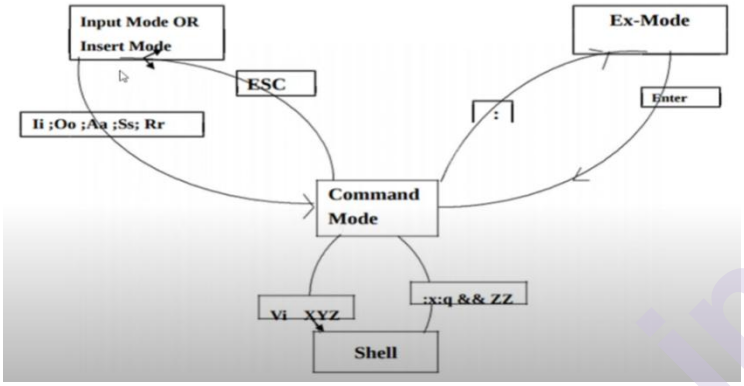
Modes of vi editor:

They are divided into three main parts:

Command Mode:

Insert Mode:

Escape Mode:



1. Command Mode:

Command Mode is the first screen of VI editor. It is case sensitive. Any character that is typed during this mode is treated as a command. These are character are not visible on the window. We can cut, copy, paste or delete a piece of text or even move through the file in this mode

[ESC] used to enter the Command Mode from another mode (Insert Mode)

2. Insert Mode:

We can easily move from Command mode à Insert mode by pressing 'i' or 'Insert' key from the keyboard. Characters typed in this mode is treated as input and add text to your file

Pressing ESC will take you from Insert Mode -> Command Mode

3. Escape Mode

Press [:] to move to the escape mode. This mode is used to save the files & execution of the commands

1. Open/ Create a File:

This will create a file with the name 'filename' or open the file with the name 'filename' if already exists.

vi Filename

2. Moving out of a file:

:q Quit out of a file

:q! Quit the file without saving the changes

:w Save the content of the editor

:wq Save the changes and quit the editor (*Combing the commands: q &: w)

ZZ In command mode, this works similar to wq

3. Rename a File:

:w newFileName – This will rename the file that you are currently working into 'new filename'. A command is used in Escape Mode.

4. Move within a file:

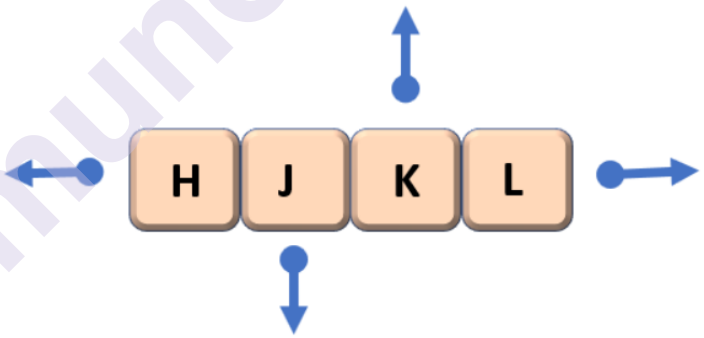
To move around in a file without actually editing the content of a file we must be in the Command mode and keep the below commands handy.

h Moves the cursor left one character position

l Moves the cursor right one character position

k Moves the cursor one line up

j Moves the cursor one line down.



Cursor Movement Keys:

5. Inserting or Adding Text

Following is the command used to put the editor in the insert mode. Once the ESC is pressed it will take the editor back to command mode.

i Insert text before the cursor

I Insert at the beginning of the current line

a Append after the cursor

A Append at the end of the current line

o Open & places the text in a new line below the current line

O Open & places the text in a new line above the current line

6. Searching the Text:

Similar to the find & replace command in windows editor we have certain Search & replace command available in the VI editor as well.

/string Search the mentioned 'String' in the forward direction

?string Search the mentioned 'String' in the backward direction

n Move to the next available position of the searched string

N Move to the next available position of the searched string in the opposite direction

Cutting & Pasting Text:

These commands allow you to copy and paste the text

yy Copy (yank, cut) the current line into the buffer

Nyy or yNy Copy 'N' lines along with the current line into the buffer

p Paste / Put the lines in the buffer into the text after the current line

4.1.6.3 Emacs:

GUI text editors and coding environments and are not used to a primarily text-based program, running commands in the editor itself, and/or using large amounts of keyboard shortcuts.

List of shortcuts

C-h C-h : help

C-g : quit

C-x b : switch buffers

C-x right : right-cycle through buffers

C-x left : left-cycle through buffers

C-x k : kill buffer

C-x 0 : close the active window

C-x 1 : close all windows except the active window

C-x 2 : split the active window vertically into two horizontal windows

C-x 3 : split the active window horizontally into two vertical windows

C-x o : change active window to next window

C-x C-f : open file

C-x C-s : save file

C-x C-w : save file as

C-space : set region mark

C-w : kill region

C-k : kill region between point and end of current line

M-w : kill region without deleting

C-y : yank region from kill ring

M-y : move to previous item in the kill ring

M-Y : move to next item in the kill ring

C-_ : undo

C-s : search forwards

C-r : search backwards

M-% : query replace ('space' to replace, 'n' to skip, '!' to replace all)

M-q : wrap text

C-left : move one word left

C-right : move one word right

C-up : move one paragraph up

C-down : move one paragraph down

home : move to the beginning of the line

end : move to the end of the line

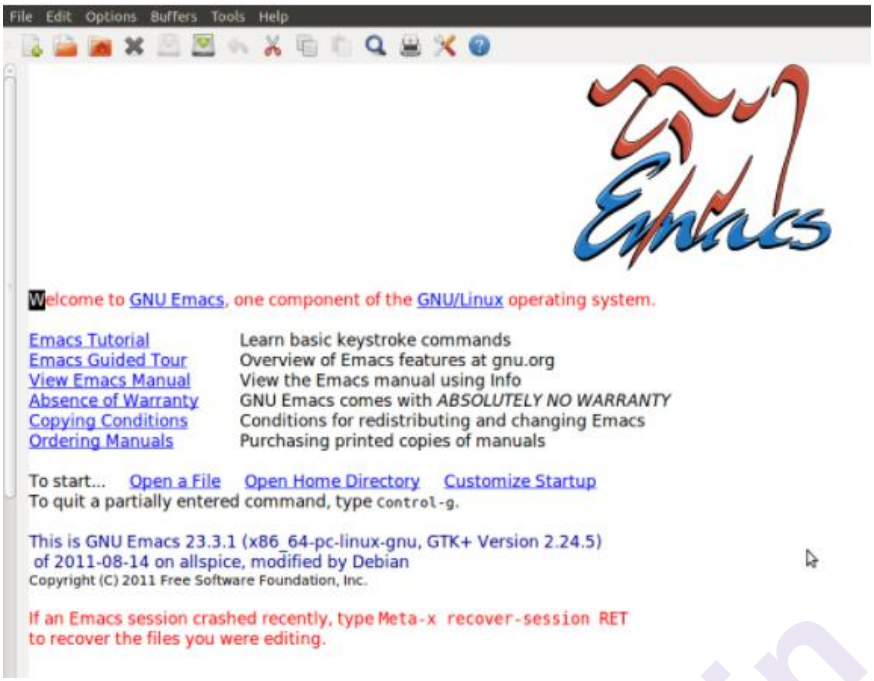
page up : move up a page

page down : move down a page

M- : move to end of buffer

Opening Emacs:

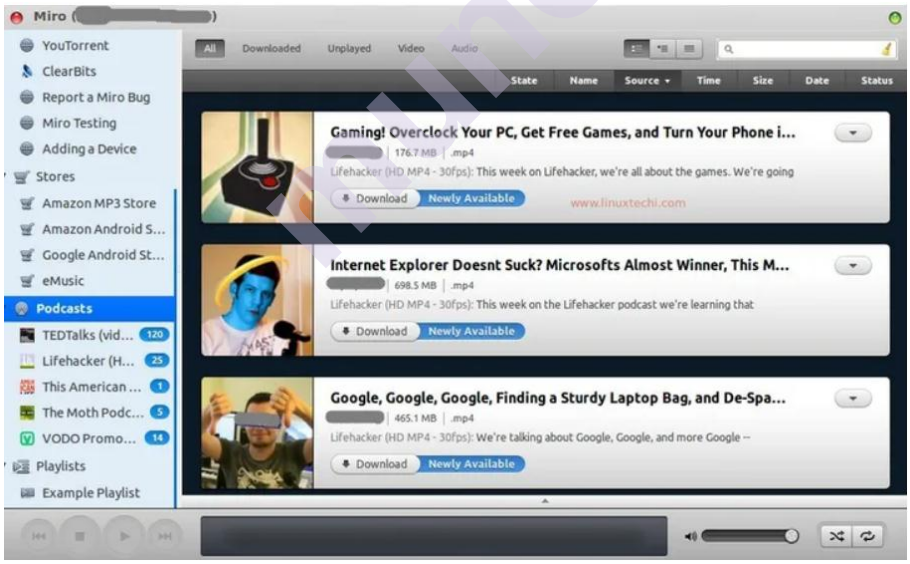
When you first open Emacs, you will see a window that looks something like this.



4.7 MULTIMEDIA APPLICATIONS

1) Miro Media Player:

Miro is one of the best multimedia player for Internet TV for Linux Desktop. It was developed by the Participatory Culture Foundation. It plays almost all audio and video formats.



Features:

Remembers last played position

Supports all major video formats including MPEG, DivX, AVI, Quicktime, WMV, FLV etc.

Also support Bit torrent file download.

2) VLC Media Player

VLC is a popular media player in Windows and also made available for Linux Desktop Environment like Ubuntu too. It is a free and open source multimedia player that supports most video formats.

Features:

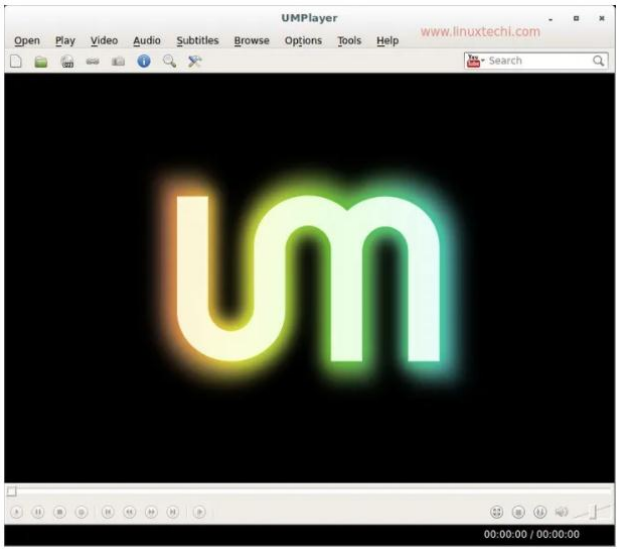
Remembers last played position

Supports all major video formats including MPEG, DivX, AVI, Quicktime, WMV, FLV etc.

Also support Bit torrent file download

3) UMPlayer:

UMPlayer stands for Universal Media Player as it can play all kinds of media formats and also platform independent. The user interface is aesthetically pleasing, simple and easy to use. It is licensed under the GNU license and is free to download. You can also watch YouTube videos on the UMPlayer.



Features:

Search and play YouTube Videos

Supports almost 270 media formats including AC3, AVI, WMV, Mp4, MPEG, XVID etc.,

You can also record YouTube Videos using UMPlayer

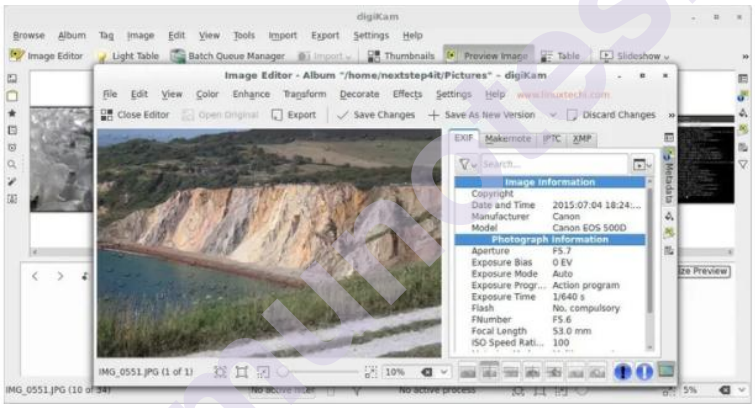
Skinnable Interface

Remember media position

Localization support

4) DigiKam:

DigiKam is an advanced photo editing and management software available for all major operating systems including Linux, Windows and Mac OS. The people behind the creation of DigiKam are real professional photographers who saw the need to developing an application that can enable them to view, edit and organize their pictures and also tag and share them with other people through social media.



*****

5

COMMAND LINE

Unit Structure

5.1 COMMAND LINE

Once you start a terminal it will log in Linux console, you get access to the shell (command line interface) CLI prompt. The prompt is your gateway to the shell. This is the place where you enter shell commands.

The default prompt symbol for the bash shell is the dollar sign ($) indicates the normal user & # sign indicates the Root user of the Linux. This symbol indicates that the shell is waiting for you to enter text. However, you can change the format of the prompt used by your shell. On my Fedora Linux system, the bash shell prompt looks like this: rich@1[~]$

On my Fedora Linux system, it looks like this:

[rich@testbox ~]$

You can change the prompt by ps1 & ps2 command. You can configure the prompt to provide basic information about your environment. The first example above shows three pieces of information in the prompt:

- The username that started the shell Like rich.
- The current virtual console number Like @1.
- The current directory (the tilde sign (~) is shorthand for the home directory) i.e. Absoulte path of Home.

Directory:

In this example: [rich@testbox ~]$ it provides similar information, except that it uses the hostname (Like testbox) instead of the virtual console number.

There are two environment variables that control the format of the command line prompt:

- PS1: Controls the format of the default command line prompt
- PS2: Controls the format of the second-tier command line prompt.

The shell uses the default PS1 prompt for initial data entry into the shell. If you enter a command that requires additional information, the shell displays the second-tier prompt specified by the PS2 environment variable.

ps2 variable holds the secondary prompt symbol which is used for commands that takes several lines to complete.

The default secondary prompt is '>'

The PS1 & P2s2 variables contains primary & secondary prompt symbols PS1 has defaults symbol value is $, you can change the prompt symbol by assigning a new set of character to the PS1 variable.

$ PS1 = "Hello>" <Enter>
Hello> new prompt

i.e. prompt will change to Hello>.

There should not be blank space before & after '=' symbol. The new prompt should be closed within (" ") double quotes.

$ psl = "→"

Primary symbol prompt change to → new prompt.

You are having specific prompt codes specified for primary symbol.

Example 2:

$psl = "\w>"

~> cd/user/doc

/usr/doc>

~ (tilde) sign will indicate current working directory absolute path

Example 3:

$ psl = "\t\!→"

This will display time & current history no. followed by

ps2:

Ps2 is a Secondary prompt it appears when 1st prompt value of the command is incomplete by pressing ctr +Z to exit command$ echo "This is incomplete command (Enter)

You can change the secondary prompt

$ ps2 = '^'

$ ps2 will change to '^'

$ ps2 = "@"

$ echo"

@

@

To display the current settings for your prompts, use the echo command: rich@1[~]$ **echo $PS1**

It displays the current primary prompt of the system.
(Eg. \u@\l[\W]\$)

rich@1[~]$ **echo $PS2**

It displays the secondary prompt of the system. **(Eg. >)**

rich@1[~]$

The shell uses special characters to signify elements within the command line prompt. Following:

List shows the special characters that you can use in the prompt string.

Bash Shell Prompt Characters:

Character	Description
\a	**The bell character**
\d	**Current date in the format "Day Month Date"**
\e	**The ASCII escape character**
\h	**The local hostname**
\H	**The fully qualified domain hostname**
\j	**The number of jobs currently managed by the shell**
\l	**The basename of the shell's terminal device name**
\n	**The ASCII newline character**
\r	**The ASCII carriage return**
\s	**The name of the shell ie. Shell currently active**
\t	**The current time in 24-hour HH:MM:SS format**

\T	**The current time in 12-hour HH:MM:SS format**
\@	**The current time in 12-hour am/pm format**
\u	**The username of the current user**
\v	**The version of the bash shell**
\V	**The release level of the bash shell**
\w	**The current working directory**
\W	**The basename of the current working directory**
\!	**The bash shell history number of this command ie. Current history number.**
\#	**The command number of this command**
\$	**Use a dollar ($) sign if a normal user, or a pound sign (#) if the root user**
\nnn	**The character corresponding to the octal value nnn**
****	**A backslash**
\[	**Begins a control code sequence**
\]	**Ends a control code sequence**

Notice that all of the special prompt characters begin with a backslash (\). The prompt contained both prompt characters and a normal character. You can create any combination of prompt characters in your prompt. To create a new prompt, just assign a new string to the PS1 variable:

[rich@testbox ~]$ PS1="[\t][\u]\$ "

[14:40:32][rich]$

This new shell prompt now shows the current time(\t), along with the username(\u), along with that $ sign indicates the normal user. The new PS1 definition only lasts for the duration of the shell session. When you start a new shell, the default shell prompt definition is reloaded.

5.2 SHELL

The GNU/Linux shell is a special interactive utility.It is a program started after you log on to the LINUX. It provides a command line interface or shell between user & LINUX kernel. Hence it is called as fundamental interface to O.S "Kernel". The core of the shell is the command prompt. The command prompt is the interactive part of the shell.

Typed 'cmds are interpreted by the shell & send to the kernel which in turns open, closes, reads or writes files. The shell runs like any other program under the LINUX system. A shell is simply a macro processor that executes commands that you enter at the command prompt. A Linux shell is understood to be both a command interpreter and a programming language. You can enter the commands at the command prompt (#) and the shell will run them. If you have several commands that you need to run, you can put them in a special text file called a script file and the shell will run it also.

There are quite a few Linux shells available to use on a Linux system. Different shells have different characteristics, some being more useful for creating scripts and some being more useful for managing processes. The default shell used in all Linux distributions is the bash shell. The bash shell was developed by the GNU project as a replacement for the standard Unix shell, called the Bourne shell (after its creator).

Linux Shells:

There are no. of shells available for linux:

1. Bash shell (Bourne again shell)
2. C shell
3. Korn shell
4. Restricted shell
5. Bourne shell
6. Tcsh shell
7. A shell
8. Z shell
9. PDKSH shell (public domain kom shell)

There are various shell available for LINUX system:

1. Bourne shell:

a. It is fastest unix command processor available & it is available on all the unix sys.

b. It is most widely used shell at present for UNIX sys.

c. The executable file name is 'sh' & it is installed as /bin/sh. It is developed by AT&T.

2. C shell:

- This is another and processor developed by William joy it gets it's name from its programming language in syntax.
- This C shell is not compatible with bourne shell.
- The C shell was developed to provide a programming interface similar to C programming language.
- The name of executable file is esh.

3. Korn shell or ksh shell:

- It was developed by David korn. It is also a product of AT&T.
- It contains the best feature of both the above shell i.e. C shell & Bourne shell.

- The executable file name is ksh.
- This is a public domain distribution for Red Hat called pdksh.
- The korn shell language is an interactive, complete, high level programming language that can be used to write shell scripts and programs.
- It is useful for writing applications with the development time being less than most other programming languages.
- Ksh is called command line completion that means when you are typing a command at the command line ksh will attempt to guess what you are typing and type it for you.
- A programming shell compatible with the Bourne shell but supporting advanced programming features like associative arrays and floating-point arithmetic

4. Restricted shell:

When you want use of O.S. to have limited access to LINUX serve to restricted shell is used & it is typically used for guest users who are not part of system &insurance installation where users must be restricted to work only in their own limited environment. These are called rsh shells.

5. Bash shell:

Bash is known as Bourne again shell. It was written by Stephen Bourne. It is an enhancement of Bourne shell since bourne shell is default shell of Unix which is already registerd hence Linux O.S legally can't use bourne shell of Unix. Hence this bourne shell is newly created with modification in bourne shell of Unix hence it is called as bourne again shell. It is a default shell of most Linux system. The Bash shell is executed as /bin/bash.

A simple, lightweight shell that runs in low-memory environments.

6. Tcsh:

Tcsh stands for Tom's C shell. Also known as Tc shell. It is enhancement of C shell. The symbolic link available for Tcsh shell on Linux is csh. You can execute Tcsh shell by typing either csh or tcsh. At cmd prompt. The C & Tc shell are not compatible with bourne shell.

A shell that incorporates elements from the C programming language into shell scripts

7. A shell:

The A shell (ash) was developed by "Kenneth alnquist". It is light weight bourne shell. It is usually suitable for computers that have very limited memory. This is a light weight Bourne compatible shell.

8. Z shell:

The Z shell can be executed by zsh. It has best features of Tesh set shell. Also it has features of korn shell& having large no. of utilities & extensive documentation. It is designed for interactive use with a powerful scripting language. An advanced shell that incorporates features from bash, tcsh, and korn, providing advanced programming features, shared history files, and themed prompts.

5.3 BASIC COMMANDS

File Management becomes easy if you know the right basic command in Linux. Following commands are the basic commands in linux.

ls	Lists all files and directories in the present working directory
ls – R	Lists files in sub-directories as well
ls – a	Lists hidden files as well
ls – al	Lists files and directories with detailed information like permissions, size, owner, etc.
cat > filename	Creates a new file
cat filename	Displays the file content
cat file1 file2 > file3	Joins two files (file1, file2) and stores the output in a new file (file3)
mv file "new file path"	Moves the files to the new location
mv filename new_file_name	Renames the file to a new filename
sudo	Allows regular users to run programs with the security privileges of the superuser or root
rm filename	Deletes a file
wc	Word Count
touch	Create blank files.

5.4 GENERAL PURPOSE UTILITIES

Along with controlling hardware devices, operating system needs utilities to perform standard functions, such as controlling files and programs.

The GNU organization (GNU stands for GNU's not UNIX) developed a complete set of Unix utilities, but had no kernel system to run them on.

Linux is developed by thousand of programmers hence GNU public licensed s/w provides a programming development tools, editors & work processors. Once Linux is installed you can start creating your own program. Hence Linux is distributed freely under GNU i.e. general public license (GPL) specified by free S/W foundation hence called as freeware.

These utilities were developed under a software philosophy called open source software (OSS). Since it is an open source S/W hence source code for application is freely distributed along with application over the internet & easy to downward, upgrade & share. The concept of OSS allows programmers to develop software and then release it to the world with no licensing fees attached. Anyone can use the software, modify it, or incorporate it into his or her own system without having to pay a license fee.

The core GNU Utilities:

The GNU project was mainly designed for Unix system administrators to have a Unix-like environment available. The core bundle of utilities supplied for Linux systems is called the *coreutils* package.

The GNU coreutils package consists of three parts:

- Utilities for handling files
- Utilities for manipulating text
- Utilities for managing processes

These three main groups of utilities each contain several utility programs that are invaluable to the Linux system administrator and programmer.

5.5 INSTALLING SOFTWARE

Instructions how to install new software in Linux: as this point is exceptionally challenging and called-for among former Windows users. The most common methods are below:

Installing RPM packages

Installing DEB packages

Installing from tarballs (esp. Source code)

Firstly, any Linux user should be aware of such thing as software repositories. Repository is storage for packages (both source and binary) accessible via Internet to install any required software on your computer.

You can easily select which to use or even create your own one: the list of connected repositories is stored here by default (examples for the most popular utilities):

– YUM: in files repo in the directory /etc/yum.repos.d/;

– APT: in file /etc/apt/sources.list and in the files in the directory /etc/apt/source.list.d/.

#1) Redhat RPM is common for Linux free software package management tool developed by Red Hat. This method is popular because users don't need to compile the code by themselves. The software is ready to be installed and you can find a brief instruction below.

As for RPM, user needs to perform the extraction of files by already defined options (such as destination, name etc.) which are hidden within the responsible utilities (rpm, yum). Installing RPM packages is fairly straight forward. To install such software package, you can run the following command: rpm -i RPMPackage.rpm.

An alternative tool here is yum: the main difference is automatic upgrades and package management (including necessary dependencies). YUM is analog for APT (DEB packages) and manage repositories. Example: yum install RPMPackage.rpm; yum update RPMPackage.rpm; yum remove RPMPackage.rpm.

#2) Debian packages are almost the same as RPM but for usage in Debian GNU/Linux systems. Obviously, the extension of such packages are *.deb.

To install such packages (whether source or binary) use APT (Advance Packaging Tool). This is package management system for Debian and also includes a lot of different tools. So, installing new software will be quite simple as well: just run the command apt-get install DEBPackage.deb. Just for understanding the common flow, here is an example: apt-get update DEBPackage.deb; apt-get remove DEBPackage.deb.

#3) Tarballs is so-called archives distributed with the following extensions ".tar.gz", ".tar.bz2", or ".zip" (there are even more regarding the type of compression and archivers). Originally tarballs are used for programs which are not compiled, i.e. they are presented as source code. That's why there significant differences how to install software this way. The main idea here: if you cannot find your program in the repositories, just download the source code from any open source program website and then install it according simple instruction below.

To extract data form such tarballs we should use the corresponding commands. Some variants are below:

- for files ending in .tar.gz, run: tar -zxvf <TarBallName>,

- for files ending in .tar.bz2, run: tar -jxvf <TarBallName>,
- for files ending in .zip, run: unzip <TarBallName>.

As you got the point that the program has not been prepared for installation one should perform the preparation procedure called pre-installation configuration. Just run ./configure and your system will be checked for any necessary libraries or configurations required.

To fulfill test of your system, preparation of the package and make installation instructions for the next step apply the following command make.

To install the program from source code after preparation phase run make install. Though, according to Linux forums, checkinstall is strongly recommended here due to the problem with further updates of installed software if using make install.

5.6 USER MANAGEMENT

The /etc/passwd file:

/etc/passwd: all information except the password encryption is stored in /etc/passwd. This file contains the password once. The encryption itself is stored in /etc/shadow. There are 7 fields area present in the file and the fields and their description is as follows:

- username – the name with which user logs in.
- password – no longer stores the password encryption but contains an x.
- UID- the use's numerical identification. No 2 users should have the same UID.
- GID- the user's numerical group identification. This number is also the third field in the /etc/group.
- Comment- user details.
- Home directory- the dir where the user ends up on logging in. the variable HOME is set by the login program by reading this field.
- Login shell – the first program executed after logging in. this is usually the shell. Login also set the variable shell by reading this entry, and also fork-execs the shell process.

For every line in /etc/passwd, there's a corresponding entry in /etc/shadow. The relevant line in this file could look something like this:

Oracle:sdfsdfsdfsd:12032::::::::

The password encryption is shown in the second field.

/dev:

Devices are also files. One can open a device, read and write to it and then close it like any other file. The functions for doing all this is built into the kernel for each and every device of the system. The same device can often be accessed with several different filenames. All device files are stored in /dev or in its subdirectories. The device files can be grouped into mainly depending on the first char of the permission field. It does have the following dirs., cdrom, default floppy drive, first hard disk, printer, tape drive and terminal.

There are two kinds of devices exist block devices and char devices. For both these types of devices, the device file exist in the /dev dir.

```
$ cat /etc/passwd

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

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

Understanding Linux File Permissions:

The root user account is the administrator for the Linux system and is always assigned UID 0. The Linux system creates lots of user accounts for various functions that aren't actual users. These are called *system accounts*. A system account is a special account that services running on the system use to gain access to resources on the system. All services that run in background mode need to be logged in to the Linux system under a system user account.

These services often just logged in using the root user account if an unauthorized person broke into one of these services, he instantly gained access to the system as the root user. To prevent this, now just about every service that runs in background on a Linux server has its own user account to log in with.

You probably noticed that the /etc/passwd file contains lots more than just the login name and UID for the user. The fields of the /etc/passwd file contain the following information:

- The login username.

- The password for the user.
- The numerical UID of the user account.
- The numerical group ID (GID) of the user account.
- A text description of the user account (called the comment field).
- The location of the HOME directory for the user.
- The default shell for the user.

The password field in the /etc/passwd file is set to an x. This doesn't mean that all of the user accounts have the same password. In the old days of Linux, the /etc/passed file contained an encrypted version of the user's password. However, since lots of programs need to access the /etc/passwd file for user information, this became somewhat of a security problem.

Now, most Linux systems hold user passwords in a separate file (called the *shadow* file, located at /etc/shadow). Only special programs (such as the login program) are allowed access to this file.

The /etc/shadow file:

The /etc/shadow file provides more control over how the Linux system manages passwords.

Only the root or admin user has access to the /etc/shadow file, making it more secure than the /etc/passwd file.

The /etc/shadow file contains one record for each user account on the system. A record looks like this:

rich:1.FfcK0ns$f1UgiyHQ25wrB/hykCn020:11627:0:99999:7:::

There are nine fields in each /etc/shadow file record:

(1) The login name corresponding to the login name in the /etc/passwd file.

(2) The encrypted password.

(3) The number of days since January 1, 1970 that the password was last changed.

(4) The minimum number of days before the password can be changed.

(5) The number of days before the password must be changed.

(6) The number of days before password expiration that the user is warned to change the password.

(7) The number of days after a password expires before the account will be disabled.

(8) The date (stored as the number of days since January 1, 1970) since the user account was disabled.

(9) A field reserved for future use.

Using the shadow password system, it can control how often a user must change his or her password, and when to disable the account if the password hasn't been changed.

Adding a new user:

The primary tool used to add new users to your Linux system is useradd. This command provides an easy way to create a new user account and set up the user's HOME directory structure all at once. The useradd command uses a combination of system default values and command line parameters to define a user account. To see the system default values used on your Linux distribution, enter the useradd command with the -D parameter:

```
# /usr/sbin/useradd -D

GROUP=100

HOME=/home

INACTIVE=-1

EXPIRE=

SHELL=/bin/bash

SKEL=/etc/skel

CREATE_MAIL_SPOOL=yes
```

/usr/sbin directory, which may not be in your PATH environment variable. The -D parameter shows what defaults the useradd command uses if you don't specify them in the command line when creating a new user account.

This example shows the following default values:

(1) The new user will be added to a common group with group ID 100.

(2) The new user will have a HOME account created in the directory /home/loginname.

(3) The account will not be disabled when the password expires.

(4) The new account will not be set to expire at a set date.

(5) The new account will use the bash shell as the default shell.

(6) The system will copy the contents of the /etc/skel directory to the user's HOME directory.

(7) The system will create a file in the mail directory for the user account to receive mail.

The next-to-the-last value is interesting. The useradd command allows an administrator to create a default HOME directory configuration, then uses that as a template to create the new user's HOME directory. This allows you to place default files for the system in every new user's HOME directory automatically. On my Linux system, the /etc/skel directory has the following files:

 # ls -al /etc/skel

total 48

```
drwxr-xr-x 2 root root 4096 2001-11-01 00:23 .

drwxr-xr-x 107 root root 12288 2007-09-20 16:53
..

-rw-r--r-- 1 root root 33 2007-02-12 10:18
.bash_logout

-rw-r--r-- 1 root root 176 2007-02-12 10:18
.bash_profile

-rw-r--r-- 1 root root 124 2007-02-12 10:18
.bashrc
```

The useradd command created the new HOME directory, using the files in the /etc/skel directory.

Useradd Parameter Description:

-c *comment* Add text to the new user's comment field.

-d *home_dir* Specify a different name for the home directory other than the login name.

-e *expire_date* Specify a date, in YYYY-MM-DD format, when the account will expire.

-f *inactive_days* Specify the number of days after a password expires when the account will be disabled. A value of 0 disables the account as soon as the password expires; a value of -1 disables this feature.

-g *initial_group* Specify the group name or GID of the user's login group.

-G *group*. . . Specify one or more supplementary groups the user belongs to.

-k Copy the /etc/skel directory contents into the user's HOME directory (must use -m as well).

-m Create the user's HOME directory.

-M Don't create a user's HOME directory (used if the default setting is to create one).

-n Create a new group using the same name as the user's login name.

-r Create a system account

-p *passwd* Specify a default password for the user account.

-s *shell* Specify the default login shell.

-u *uid* Specify a unique UID for the account.

Removing a user:

If you want to remove a user from the system, the userdel command is what you need. By default, the userdel command only removes the user information from the /etc/passwd file.

It doesn't remove any files the account owns on the system.

If you use the -r parameter, userdel will remove the user's HOME directory, along with the user's mail directory. However, there may still be other files owned by the deleted user account on the system. This can be a problem in some environments.

Here's an example of using the userdel command to remove an existing user account:

```
# /usr/sbin/userdel -r test

# ls -al /home/test

ls: cannot access /home/test: No such file or
directory
```

User Account Modification Utilities:

Command Description:

Usermod Edits user account fields, as well as specifying primary and secondary group membership passwd Changes the password for an existing user chpasswd Reads a file of login name and password pairs, and updates the passwords chage Changes the password's expiration date chfn Changes the user account's comment information chsh Changes the user account's default shell Each of these utilities provides a specific function for changing information about user accounts.

The following sections describe each of these utilities.

usermod:

The usermod command is the most robust of the user account modification utilities. It provides options for changing most of the fields in the /etc/passwd file. To do that you just need to use the command line

parameter that corresponds to the value you want to change. The parameters are mostly the same as the useradd parameters (such as -c to change the comment field, -e to change the expiration date, and -g to change the default login group). However, there are a couple of additional parameters that might come in handy:

- -l to change the login name of the user account
- -L to lock the account so the user can't log in
- -p to change the password for the account
- -U to unlock the account so that the user can log in

The -L parameter is especially handy. Use this to lock an account so that a user can't log in without having to remove the account and the user's data. To return the account to normal, just use the -U parameter.

passwd and chpasswd:

A quick way to change just the password for a user is the passwd command:

passwd test

Changing password for user test.

New UNIX password:

Retype new UNIX password:

passwd: all authentication tokens updated successfully.

#

If you just use the passwd command by itself, it'll change your own password. Any user in the system can change their own password, but only the root user can change someone else's password.

The -e option is a handy way to force a user to change the password on the next log in. This allows you to set the user's password to a simple value, then force them to change it to something harder that they can remember.

If you ever need to do a mass password change for lots of users on the system, the

chpasswd:

The chpasswd command reads a list of login name and password pairs (separated by a colon) from the standard input, and automatically encrypts the password and sets it for the user account.

chsh, chfn, and chage:

The chsh, chfn, and chage utilities are specialized for specific functions. The chsh commandallows you to quickly change the default login shell for a user. You must use the full pathname for the shell, and not just the shell name:

```
# chsh -s /bin/csh test
```

Changing shell for test.

```
Shell changed.
```

#

The chfn command provides a standard method for storing information in the comments field in the /etc/passwd file. Instead of just inserting random text, such as names, nicknames, or even just leaving the comment field blank, the chfn command uses specific information used in the Unix finger command to store information in the comment field. The finger command allows you to easily find information about people on your Linux system:

finger rich

Login: rich Name: Rich Blum

Directory: /home/rich Shell: /bin/bash

On since Thu Sep 20 18:03 (EDT) on pts/0 from 192.168.1.2

No mail.

No Plan.

#

If you use the chfn command with no parameters, it queries you for the appropriate values to enter in to the comment field:

 # chfn test:

Changing finger information for test.

Name []: Ima Test

Office []: Director of Technology

Office Phone []: (123)555-1234

Home Phone []: (123)555-9876

Finger information changed.

 # finger test:

Login: test Name: Ima Test

Directory: /home/test Shell: /bin/csh

Office: Director of Technology Office Phone: (123)555-1234

Home Phone: (123)555-9876

Never logged in.

No mail.

No Plan.

#

If you now look at the entry in the /etc/passwd file, it looks like this:

 # grep test /etc/passwd:

```
test:x:504:504:Ima          Test,Director          of
Technology,(123)555-

1234,(123)555-9876:/home/test:/bin/csh

#
```

All of the finger information is neatly stored away in the /etc/passwd file entry.

Finally, the chage command helps us manage the password aging process for user accounts.

There are several parameters to set individual values, shown in Table 6-4.

The chage date values can be expressed using one of two methods:

- A date in YYYY-MM-DD format
- A numerical value representing the number of days since January 1, 1970

One neat feature of the chage command is that it allows you to set an expiration date for an account. Using this feature, you can create temporary user accounts that automatically expire on a set date, without your having to remember to delete them! Expired accounts are similar to locked accounts. The account still exists, but the user can't log in with it.

The change Command Parameters:

Parameter Description:

- d Set the number of days since the password was last changed.
- E Set the date the password will expire.
- I Set the number of days of inactivity after the password expires to lock the account.

- m Set the minimum number of days between password changes.
- W Set the number of days before the password expires that a warning message appears.

5.7 ENVIRONMENT VARIABLES

The bash shell uses a feature called environment variables to store information about the shell session and the working environment (thus the name environment variables).

This feature also allows you to store data in memory that can be easily accessed by any program or script running from the shell.

There are two types of environment variables in the bash shell:

(1) Global Variables.

(2) Local Variables.

(1) Global Environment Variables:

Global environment variables are visible from the shell session, and any child processes that the shell spawns. Local variables are only available in the shell that creates them. This makes global environment variables useful in applications that spawn child processes that require information from the parent process.

To display the value of an individual environment variable, use the echo command. When referencing an environment variable, you must place a dollar sign before the environment variable name:

```
$ echo $HOME

/home/rich
```

Global environment variables are also available to child processes running under the current shell session:

```
$ bash

$ echo $HOME

/home/rich
```

(2) Local Environment Variables:

Local environment variables, as their name implies, can be seen only in the local process in which they are defined.

```
Set} example
```

5.7.1 Setting Environment Variables:

Once you start a bash shell (or spawn a shell script), you're allowed to create local variables that are visible within your shell process. You can assign either a numeric or a string value to an environment variable by assigning the variable to a value using the equal sign:

```
$ test=testing

$ echo $test

testing
```

If you need to assign a string value that contains spaces, you'll need to use a single quotation mark to delineate the beginning and the end of the string:

```
$ test=testing a long string

-bash: a: command not found

$ test='testing a long string'

$ echo $test testing a long string
```

Note: It's extremely important that there are no spaces between the environment variable name, the equal sign, and the value. If you put any spaces in the assignment, the bash shell interprets the value as a separate command:

```
$ test2 = test

-bash: test2: command not found
```

Once you set a local environment variable, it's available for use anywhere within your shell process. However, if you spawn another shell, it's not available in the child shell:

```
$ bash

$ echo $test

$ exit

exit

$ echo $test

testing a long string

$
```

The test environment variable is not available in the child shell (it contains a blank value)

Similarly, if you set a local environment variable in a child process, once you leave the child process the local environment variable is no longer available:

```
$ bash

$ test=testing

$ echo $test

testing

$ exit

exit

$ echo $test
```

Setting Global Environment Variables:

Global environment variables are visible from any child processes created by the process that sets the global environment variable. The method used to create a global environment variable is to create a local environment variable, then export it to the global environment.

This is done by using the export command:

```
$ echo $test

testing a long string

$ export test

$ bash

$ echo $test

testing a long string

$
```

After exporting the local environment variable test, I started a new shell process and viewed the value of the test environment variable. This time, the environment variable kept its value, as the export command made it global.

5.7.2 Removing Environment Variables:

If you can create a new environment variable, it makes sense that you can also remove an existing environment variable. This is done by using the unset command:

```
$ echo $test

testing

$ unset test
```

```
$ echo $test
```

Note: *When referencing the environment variable in the unset command, remember not to use the dollar sign.*

If you're in a child process and unset a global environment variable, it only applies to the child process. The global environment variable is still available in the parent process:

```
$ test=testing
$ export test
$ bash
$ echo $test
testing
$ unset test
$ echo $test
$ exit
exit
$ echo $test
testing
$
```

5.7.3 Default Shell Environment Variables:

Table shows the environment variables the bash shell provides that are compatible with the original Unix Bourne shell.

Variable	Description
CDPATH	A colon-separated list of directories used as a search path for the cd command.
HOME	The current user's home directory.
IFS	A list of characters that separate fields used by the shell to split text strings.
MAILPATH	A colon-separated list of multiple filenames for the current user's mailbox. The
OPTARG	The value of the last option argument processed by the getopts command.
PATH A	colon-separated list of directories where the shell looks for commands.
PS1	The primary shell command line interface prompt string.
PS2	The secondary shell command line interface prompt string.

MAIL	The filename for the current user's mailbox. The bash shell checks this file
For new mail.	bash shell checks each file in this list for new mail.
OPTIND	The index value of the last option argument processed by the getopts command

5.7.4 Setting the PATH Environment Variables:

The most important environment variable in this list is the PATH environment variable.

When you enter a command in the shell command line interface (CLI), the shell must search the system to find the program. The PATH environment variable defines the directories it searches looking for commands. On my Linux system, the PATH environment variable looks like this:

```
$ echo $PATH

/usr/kerberos/bin:/usr/lib/ccache:/usr/local/bin:
/bin:/usr/bin:/home/rich/bin
```

This shows that there are six directories where the shell looks for commands. Each directory in the PATH is separated by a colon. There's nothing at the end of the PATH variable indicating the end of the directory listing.

PATH environment variable without having to rebuild it from scratch. The individual directories listed in the PATH are separated by a colon. All you need to do is reference the original PATH value, and add any new directories to the string.

This looks something like this:

```
$ echo $PATH

/usr/kerberos/bin:/usr/lib/ccache:/usr/local/bin:
/bin:/usr/bin:/home

/rich/bin

$ PATH=$PATH:/home/rich/test

$ echo $PATH

/usr/kerberos/bin:/usr/lib/ccache:/usr/local/bin:
/bin:/usr/bin:/home

/rich/bin:/home/rich/test

$ myprog
```

The factorial of 5 is 120.

5.7.5 Locating System Environment Variables:

The Linux system uses environment variables to identify itself in programs and scripts. This provides a convenient way to obtain system information for your programs. The trick is in how these environment variables are set.

By default bash checks several files for commands. These files are called *startup files*. The startup files bash processes depend on the method use to start the bash shell.

There are three ways of starting a bash shell:

(1) Default login shell at login time.

(2) Interactive shell that is not the login shell.

(3) Non-interactive shell to run a script.

(1) Default Login Shell at Login Time:

Linux system, the bash shell starts as a login shell. The login shell looks for four different startup files to process commands from. The order in which the bash shell processes the files is:

- /etc/profile
- $HOME/.bash profile
- $HOME/.bash login
- $HOME/.profile

The /etc/profile file is the main default startup file for the bash shell. Whenever you log in to the Linux system, bash executes the commands in the /etc/profile startup file.

The remaining three startup files are all used for the same function to provide a user-specific startup file for defining user-specific environment variables.

(2) Interactive Shell that is not the Login Shell:

If you start a bash shell without logging into a system (such as if you just type bash at a CLI prompt), you start what's called an *interactive* shell. The interactive shell doesn't act like the login shell, but it still provides a CLI prompt for you to enter commands.

If bash is started as an interactive shell, it doesn't process the /etc/profile file. Instead, it checks for the .bashrc file in the user's HOME directory.

The .bashrc file does two things.

First, it checks for a common bashrc file in the /etc directory.

Second, it provides a place for the user to enter personal aliases and private script functions.

The common/etc/bashrc startup file is run by everyone on the system who starts an interactive shell session.

(3) Non-interactive Shell:

The last type of shell is a non-interactive shell. This is the shell that the system starts to execute a shell script.

This is different in that there isn't a CLI prompt to worry about. However, there may still be specific startup commands you want to run each time you start a script on your system.

To accommodate that situation, the bash shell provides the BASH ENV environment variable. When the shell starts a non-interactive shell process, it checks this environment variable for the name of a startup file to execute.

If one is present, the shell executes the commands in the file.

5.7.6 Variable Arrays:

Feature of environment variables is that they can be used as *arrays*. An array is a variable that can hold multiple values. Values can be referenced either individually or as a whole for the entire array.

To set multiple values for an environment variable, just list them in parentheses, with each value separated by a space:

```
$ mytest=(one two three four five)

$ echo $mytest

one

$
```

Only the first value in the array appears. To reference an individual array element, you must use a numerical index value, which represents its place in the array. The numeric value is enclosed in square brackets:

```
$ echo ${mytest[2]} three
```

Note: Environment variable arrays start with an index value of zero.

To display an entire array variable use the asterisk wildcard character as the index value:

```
$ echo ${mytest[*]}

one two three four five

$
```

Change the value of an individual index position:

```
$ mytest[2]=seven

$ echo ${mytest[*]}

one two seven four five

$
```

use the unset command to remove an individual value within the array

```
$ unset mytest[2]

$ echo ${mytest[*]}

one two four five

$

$ echo ${mytest[2]}

$ echo ${mytest[3]}

four
```

This example uses the unset command to remove the value at index value 2. When you display the array, it appears that the other index values just dropped down one. However, if you specifically display the data at index value 2, you'll see that that location is empty.

Remove the entire array just by using the array name in the unset command:

```
$ unset mytest

$ echo ${mytest[*]}
```


5.8 USING COMMAND ALIASES

A command alias allows you to create an alias name for common commands (along with their parameters) to help keep your typing to a minimum.

To see a list of the active aliases, use the alias command with the -p parameter:

```
$ alias -p

alias l.='ls -d .* --color=tty'

alias ll='ls -l --color=tty'

alias ls='ls --color=tty'

alias vi='vim'
```

```
alias which='alias | /usr/bin/which --tty-only --
readalias--

show-dot --show-tilde'
```

use an alias to override the standard ls command. It automatically provides the --color parameter, indicating that the terminal supports color mode listings.

Using the alias command:

```
[rich@testbox ~]$ alias li='ls -il'

[rich@testbox ~]$ li

total 989292

360621 drwxrwxr-x 2 rich rich 4096 2007-08-24
22:04 4rich

301871 drwxr-xr-x 4 rich rich 4096 2007-09-18
08:38 Desktop

301875 drwxr-xr-x 2 rich rich 4096 2001-11-01
01:10 Documents

301872 drwxr-xr-x 2 rich rich 4096 2001-11-01
04:06 Download
```

Once you define an alias value, you can use it at any time in your shell, including in shell scripts.

Command aliases act like local environment variables. They're only valid for the shell process in which they're defined:

```
$ alias li='ls -il'

$ bash

$ li
```

bash: li: command not found

5.9 SUMMARY

This chapter consist of different types of shell in command line, How to install rpm packages with the help of yum, different files when we create a user, command with options, how to change password of the user using passwd , different types of environment variables.

5.10 UNIT END QUESTIONS

1. Write a short note on following

   Bash shell (Bourne again shell)

C shell

Korn shell

Restricted shell

Bourne shell

Tcsh shell

A shell

Z shell

2. How many GNU coreutils package are available? Explain any one of them.
3. Differentiate rpm and yum.
4. Write the contain of /etc/passwd file.
5. Describe/etc/shadow file
6. Describe useradd Parameter
7. How to change password of the particular user.
8. Explain types of environment variables in the bash shell.
9. Explain how to setup and removing Environment Variables.
10. Describe default Shell Environment Variables.
11. How COMMAND ALIASES used in command line

5.11 LIST OF REFERENCES

1) Unix Concepts and Applications by Sumitabha Das.

2) Official Ubuntu Book, 8th Edition, by Matthew Helmke & Elizabeth K. Joseph with Jose Antonio Rey and Philips Ballew, Prentice Hall

*****

6

LINUX DOCUMENTATION

Unit Structure

6.0 OBJECTIVES

In this chapter In this chapter you will learn about:

- Linux Documentation man pages, GNU info help command
- More documentation sources, File Operations, Filesystem, Filesystem
- Architecture, File types, File attributes, Working with
- Files, Backup and compression

6.1 LINUX DOCUMENTATION

if you are new to LINUX operating system and having trouble dealing with the command-line utilities provided by LINUX then you really need to know first of all about the help command which as its name says help you to learn about any built-in command.
help command as told before just displays information about shell built-in commands. Here's the syntax for it:

// syntax for help command

$help [-dms] [pattern ...]

The pattern specified in the syntax above refers to the command about which you would like to know and if it is matched with any shell built-in command then help give details about it and if it is not matched

then help prints the list of help topics for your convenience. And the d, m and s here are options that you can use with the help command.

$help help

help: help [-dms] [pattern...]

Display information about builtin commands.

Displays brief summaries of builtin commands. If PATTERN IS

specified, gives detailed help on all commands matching PATTERN,

otherwise the list of help topics is printed.

Options:

-d output short description for each topic

-m display usage in pseudo-manpage format

-s output only a short usage synopsis for each topic matching

6.2 MAN PAGES, GNU INFO

GNU Info accepts several options to control the initial node being viewed, and to specify which directories to search for Info files. Here is a template showing an invocation of GNU Info from the shell:

info [option]... [menu-item...]

The program accepts the following options:

--apropos=string

Specify a string to search in every index of every Info file installed on your system. Info looks up the named string in all the indices it can find, prints the results to standard output, and then exits. If you are not sure which Info file explains certain issues, this option is your friend. Note that if your system has a lot of Info files installed, searching all of them might take some time.

6.3 HELP COMMAND

$help help

help: help [-dms] [pattern...]

Display information about builtin commands.

Options for help command

-d

option: It is used when you just want to get an overview about any shell built-in command i.e it only gives short description.

-m option: It displays usage in pseudo-manpage format.

-s option: It just displays only a short usage synopsis for each topic matching.

6.4 MORE DOCUMENTATION SOURCES

The commercial Linux distributions-for example, Red Hat, SuSE, Mandrake, Xandros, and Linspire-supply excellent user manuals. Every major Linux distribution provides a feast of online resources. Search engines, user mailing lists, Usenet, and all sorts of Linux web sites also supply a wealth of help and information.

6.5 FILE OPERATIONS

The most basic feature of the shell is the ability to see what files are available on the system. The list command (ls) is the tool that helps do that.

Basic listing:

The ls command at its most basic form displays the files and directories located in your current directory:

$ ls

4rich Desktop Download Music Pictures store store.zip test backup Documents Drivers myprog Public store.sql Templates Videos.

$

Notice that the ls command produces the listing in alphabetical order (in columns rather than rows). If you're using a terminal emulator that supports color, the ls command may also show different types of entries in different colors. The LS COLORS environment variable controls this feature. Different Linux distributions set this environment variable depending on the capabilities of the terminal emulator.

If you don't have a color terminal emulator, you can use the -F parameter with the ls command to easily distinguish files from directories. Using the -F parameter produces the following output:

$ ls -F

4rich/ Documents/ Music/ Public/ store.zip Videos/

backup.zip Download/ myprog* store/ Templates/

Desktop/ Drivers/ Pictures/ store.sql test

$

The -F parameter now flags the directories with a forward slash, to help identify them in the listing. Similarly, it flags executable files with an asterisk, to help you find the files that can be run on the system easier.

The basic ls command can be somewhat misleading. It shows the files and directories contained in the current directory, but not necessarily all of them. Linux often uses hidden files to store configuration information. In Linux, hidden files are files with filenames that start with a period.

These files don't appear in the default ls listing (thus they are called hidden).

To display hidden files along with normal files and directories, use the -a parameter.

Displaying File Information with different ls options:

(1)	$ ls	:	This ls command is shortened form of list & lists the file in your directory. It works as' dir' command in Dos.
(2)	$ ls – l	:	This command will give listing of files in long format with all permission i.e. reading, writing, & executing with users & group info of life.
(3)	$ ls–al	:	This command will give listing of file with all info along with all hidden files in long format.
(4)	$ ls -c	:	Sort by time of last modification.
(5)	$ ls -d	:	Directory List directory entries instead of contents, and don't dereference symbolic links.
(6)	$ ls -F	:	Classify Append file-type indicator to entries.
(7)	$ ls -g	:	List full file information except for the file's owner.
(8)	$ ls -G	:	No-group In long listing don't display group names.
(9)	$ ls -n	:	Numeric-uid-gid Show numeric userid and groupid instead of names.
(10)	$ ls -o	:	In long listing don't display owner names.
(11)	$ ls -r	:	Reverse Reverse the sorting order when displaying files and directories.
(12)	$ ls -u	:	Display file last access time instead of last modification time.
(13)	$ ls -v	:	sort=version Sort the output by file version.
(14)	$ ls – m	:	It lists all the files separated by commas.
(15)	$ ls – Y	:	It sorts the file name in columns horizontally.
(16)	$ ls – A	:	It lists all the files & directories without two directories 1). 2) ..
(17)	$ ls – C	:	It sorts the file in column vertically.
(18)	$ ls – F	:	It identifies directories links & executable files.
(19)	$ ls – s	:	It sorts file by size.
(20)	$ ls – color	:	It use color for identification.

There are different wildcard characters and Regular Expression supported by ls command.

The ls command recognizes standard wildcard characters and uses them to match patterns within the filter:

- A question mark (?) to represent one character.
- An asterisk (*) to represent zero or more characters.

The question mark can be used to replace exactly one character anywhere in the filter string.

For example:

$ ls -l mypro?

-rw-rw-r-- 1 rich rich 0 2007-09-03 16:38 myprob

-rwxr--r-- 1 rich rich 30 2007-08-23 21:42 myprog

$

The filter mypro? matched two files in the directory. Similarly, the asterisk can be used to match

zero or more characters:

$ ls -l myprob*

-rw-rw-r-- 1 rich rich 0 2007-09-03 16:38 myprob

-rw-rw-r-- 1 rich rich 0 2007-09-03 16:40 myproblem

$

The asterisk matches zero characters in the myprob file, but it matches three characters in the myproblem file.

This is a powerful feature to use when searching for files when you're not quite sure of the filenames.

Ls supports the regular expressions Like the range of characters is specified by [ ].

Following are some of examples of using Wild card chracters & Regular expressions:

1. $ ls doc *:	This command will give listing of files starting with Doc.
2. $ ls * day:	This command gives listing of all files ending with day.
3. $ ls * .c:	Listing of all files having extension '.c'

4. $ ls doc ?:	This command gives listing of all files having

	starting 3 characters as doc & followed by one single unknown character.
5. $ ls *.[co] :	It displays the files having extension either c or o main.c arg.c main.o
6. $ ls doc[1A]:	You can specify a set of characters as a range, rather than listing them one by one. The filename begining with doc followed by 1 or A.
7. $ ls doc[1-3]:	It displays the files begin with pattern doc & ending in character 1 through 3.

Output of ls –l is as shown below:

$ $ ls –l

File type	File permission	No. of links	Owner group	Other	Size in byte	Date & time last modified	file name
–	rwxrwxrwx	2	ABC other	"	660 4096	2007-08-24 15:34	abc.doc
	d wxrwxrwx	3	XYZ ABC	"	1250	"	
	d wxrwxrwx	1	ABC other	"	950	"	xyz

The file type (such as directory (d), Ordinary file (-), character device (c), or block device (b)

The permissions for the file:

(1) Read → With this permission user can read the data or file, user can not write into it.

(2) Write → With this permission user can write the data or file.

(3) Execute → With this permission user can execute the application.

The number of hard links to the file (Links of the file in same directory)

The username of the owner of the file.

The group name of the group the file belongs to.

The size of the file in bytes.

The time the file was modified last.

The file or directory namePart I.

The Linsux is divided into 3 groups owner, group and others.

1) Owner:

It is the user who creates a particulars file called as owner of that file. The owner is a person who has all rights on the files created by him and also decides the rights of other user associated to that file. The owner group is abbrivated as ‘u’

2) Group:

In a typical software development one large project is divided among different members of a team. So software developers requires that file belonging their project should not accessible to other users hence they will apply right to group users. It is abbreviated as ‘g’

3) Others:

All the users that are neither owner nor engaged to with a group is called as other normally guest user Abbrivated d as ‘o’.

The Linux Command Line:

The -R parameter is another command ls parameter to use. It shows files that are contained within directories in the current directory. If more number of directories are available then this can be quite a long listing. Here’s a simple example of what the -R parameter produces:

$ ls -F -R

.:

file1 test1/ test2/

./test1:

myprog1* myprog2*

./test2:

GURE 3-3

Notice that first, the -R parameter shows the contents of the current directory, which is a file (file1) and two directories (test1 and test2). Following that, it traverses each of the two directories, showing if any files are contained within each directory. The test1 directory shows two files (myprog1 and myprog2), while the test2 directory doesn’t contain any files. If there had been further subdirectories within the test1 or test2 directories, the -R parameter would have continued to traverse those as well. As you can see, for large directory structures this can become quite a large output listing.

The complete parameter list

The ls command uses two types of command line parameters:

- Single-letter parameters.
- Full-word (long) parameters.

The single-letter parameters are always preceded by a single dash. Full-word parameters are more descriptive and are preceded by a double dash. Many parameters have both a single-letter and full-word version, while some have only one type. These are some of parameters that are used with ls command.

Single Letter	Full Word Description
O	In long listing don't display owner names.
R	Reverse the sorting order when displaying files and directories.
R	Recursive List subdirectory contents recursively.
S	Size Print the block size of each file.
S	sort=size Sort the output by file size.
T	sort=time Sort the output by file modification time.
U	Display file last access time instead of last modification time.
U	sort=none Don't sort the output listing.
V	sort=version Sort the output by file version.
X	List entries by line instead of columns.
X	sort=extension Sort the output by file extension

You can use more than one parameter at a time if you want to. A common combination to use is the -a parameter to list all files, the -i parameter to list the inode for each file, the -l parameter to produce a long listing, and the -s parameter to list the block size of the files. The inode of a file or directory is a unique identification number the kernel assigns to each object in the filesystem. Combining all of these parameters creates the easy-to-remember -sail parameter:

$ ls -sail

total 2360

301860 8 drwx------ 36 rich rich 4096 2007-09-03 15:12 .

65473 8 drwxr-xr-x 6 root root 4096 2007-07-29 14:20 ..

360621 8 drwxrwxr-x 2 rich rich 4096 2007-08-24 22:04 4rich

301862 8 -rw-r--r-- 1 rich rich 124 2007-02-12 10:18 .bashrc

361443 8 drwxrwxr-x 4 rich rich 4096 2007-07-26 20:31 .ccache

301879 8 drwxr-xr-x 3 rich rich 4096 2007-07-26 18:25 .config

301871 8 drwxr-xr-x 3 rich rich 4096 2007-08-31 22:24 Desktop

301870 8 -rw------- 1 rich rich 26 2001-11-01 04:06 .dmrc

301872 8 drwxr-xr-x 2 rich rich 4096 2001-11-01 04:06 Download

360207 8 drwxrwxr-x 2 rich rich 4096 2007-07-26 18:25 Drivers

301882 8 drwx------ 5 rich rich 4096 2007-09-02 23:40 .gconf

301883 8 drwx------ 2 rich rich 4096 2007-09-02 23:43 .gconfd

360338 8 drwx------ 3 rich rich 4096 2007-08-06 23:06 .gftp

File Handling:

The bash shell provides lots of commands for manipulating files on the Linux filesystem. This section walks you through the basic commands you will need to work with files from the CLI (command level interface) for all your file-handling needs.

➢ **File:**

(1) The file is container for storing information.

(2) A file's size is not stored in the file, nor even it's name. All file attributes are kept in a separate area of the hard disk, not directly to humans, but only to the kernel.

(3) UNIX treats directories and devices as file as well. A directory is simply a folder where you store filenames and other directories. All physical devices like the hard disk, memory, CD-ROM, printer and modem are treated as files.

➢ **Creating files:**

(1) Using cat to create a file:

- Cat is also useful for creating a file.
- Enter the command cat, followed by the > (the right chevron) character and the filename(for e.g. kiran):

$ cat > kiran

A > symbol following the command means that the output goes to the filename following it. Cat used in this way represents a rudimentary editor.

[Ctrl-d]

$ _ prompt returns

- **$ cat >file1:**

This command is used to create a new file known as file1.

To save this file ctrl +d option is used, ctrl+d indicate end of file mark ie eof mark.

To interrupt in between ctrl + z is used.

This command defines that redirect the contents from standard output ie. Console to the newly created file named as file1.

- When the command line is terminated with [Enter], the prompt vanishes. cat now waits to take input from the user. Enter the three lines, each followed by [Enter]. Finally press [Ctrl+d] to signify the end of input to the system. This is the eof character used by UNIX systems and is shown in the sty output.
- When this character is entered, the system understand that no further text input will be made.
- The file is written and the prompt returned.
- **cat** is a versatile command. It can be used to create, display, concatenate and append to files.
- **$ cat kiran >> new:**
  This command is used to redirect the contents of file kiran to the standard output and redirect these contents for appending (adding) into file new. ie. kiran contents will be appended into the new. if new is not created first then it will create the file new and redirect the contents of kiran to new and if it is already created then it will append the contents of kiran to new.
- **$ cat < chirag >newfile:**

  This command is used to redirect the contents of file chirag to the standard output and redirect these contents to the file newfile. ie. chirag contents will be redirected (overwritten) to the newfile.

(2) Using touch command to create a file:

Every once in a while you will run into a situation where you need to create an empty file. Sometimes applications expect a log file to be present before they can write to it. In these situations, you can use the touch command to easily create an empty file:

$ touch test1

$ ls -il test1

1954793 -rw-r--r-- 1 rich rich 0 Sep 1 09:35 test1

$

The Linux Command Line:

The touch command creates the new file you specify, and assigns your username as the file owner. Since I used the -il parameters for the ls command, the first entry in the listing shows the inode number assigned to the file. Every file on the Linux system has a unique inode number.

Notice that the file size is zero, since the touch command just created an empty file. The touch command can also be used to change the access and modification times on an existing file without changing the file contents:

$ touch test1

$ ls -l test1

-rw-r--r-- 1 rich rich 0 Sep 1 09:37 test1

$

The modification time of test1 is now updated from the original time. If you want to change only the access time, use the -a parameter. To change only the modification time, use the –m parameter. By default touch uses the current time. You can specify the time by using the –t parameter with a specific timestamp:

$ touch -t 200812251200 test1

$ ls -l test1

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test1

$

Now the modification time for the file is set to a date significantly in the future from the current time.

Copying files:

Copying files and directories from one location in the filesystem to another is a common practice for system administrators. The cp command provides this feature.

In it's most basic form, the cp command uses two parameters: the source object and the destination object:

cp source destination

When both the source and destination parameters are filenames, the cp command copies the source file to a new file with the filename specified as the destination. The new file acts like a brand new file, with an updated file creation and last modified times:

$ cp kiran newfile

$ ls -il

total 0

1954793 -rw-r--r-- 1 rich rich 0 Dec 25 2008 kiran

1954794 -rw-r--r-- 1 rich rich 0 Sep 1 09:39 newfile

$

The new file newfile shows a different inode number, indicating that it's a completely new file.

You'll also notice that the modification time for the newfile file shows the time that it was created.

cp: Copying a file:

(1) The cp (copy) command copies a file or a group of files. It creates an exact image of the file on disk with a different name

(2) The syntax requires at least two filenames to be specified in the command line.

(3) When both are ordinary files, the first id copied to the second:

cp chap01 unit1

(4) If the destination file(unit1)doesn't exist, it will first be created before copying takes place.

(5) If not, it will simply be overwritten without any warning from the system.

(6) If there is only one file to be copied, the destination can be either an ordinary or directory.

(7) You then have the option of choosing your destination filename.

(8) The following example shows two ways of copying a file to the progs directory:

cp chap01 progs/unit1

chap01 copied to unit1 under progs

(9) cp is often used with the shorthand notation, . (dot), to signify the current directory as the destination.

(10) For instance, to copy the file .profile from /home/chirag to your current directory, you can use either of the two commands:

cp /home/chirag/.profile.profile Destination is a file

cp/home/chirag/.profile. Destination is the current directory

(11) cp can also be used to copy more than one file with a single invocation of the command.

In that case, the last filename must be a directory:

- You have already seen how the UNIX system uses the * to frame a pattern for matching more than one filename.
- If there were only three filenames in the current directory having the common string chap, you can compress the above sequence using the * as a suffix to chap:

cp chap* progs Copies all files beginning with chap.

cp Options:

(a) Interactive copying (-i) the –i(interactive) option warns the user before overwriting the destination file. If unit1 exists, cp prompts for a response:

$ cp –i chap01 unit1

(b) cp: overwrite unit1 (yes/no)? y

A y at this prompt overwrites the file, any other responses leaves it uncopied.

Copying directory structures (-R) Many UNIX commands are capable of recursive behavior. This means that the command can descend a directory and examine all files in its subdirectories. The cp –R command behaves recurvely to copy an entire directory structure.

cp –R progs newprogs newprogs must not exist

(3) Basic bash Shell Commands 3:

If the destination file already exists, the cp command will prompt you to answer whether or not you want to overwrite it:

$ cp test1 test2

cp: overwrite `test2'? y

$

If you don't answer y, the file copy will not proceed. You can also copy a file to an existing directory:

$ cp test1 dir1

$ ls -il dir1

total 0

1954887 -rw-r--r-- 1 rich rich 0 Sep 6 09:42 test1

$

The new file is now under the dir1 directory, using the same filename as the original. These examples all used relative pathnames, but you can just as easily use the absolute pathname for both the source and destination objects.

To copy a file to the current directory you're in, you can use the dot symbol:

$ cp /home/rich/dir1/test1 .

cp: overwrite `./test1'?

Use the -p parameter to preserve the file access or modification times of the original file for the copied file.

$ cp -p test1 test3

$ ls -il

total 4

1954886 drwxr-xr-x 2 rich rich 4096 Sep 1 09:42 dir1/

1954793 -rw-r--r-- 1 rich rich 0 Dec 25 2008 test1

1954794 -rw-r--r-- 1 rich rich 0 Sep 1 09:39 test2

1954888 -rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

$

Now, even though the test3 file is a completely new file, it has the same timestamps as the original test1 file.

The-R parameter is extremely powerful. It allows you to recursively copy the contents of an entire directory in one command:

$ cp -R dir1 dir2
$ ls -l
total 8

The Linux Command Line:

drwxr-xr-x 2 rich rich 4096 Sep 6 09:42 dir1/

drwxr-xr-x 2 rich rich 4096 Sep 6 09:45 dir2/

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test1

-rw-r--r-- 1 rich rich 0 Sep 6 09:39 test2

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

$

Now dir2 is a complete copy of dir1. You can also use wildcard characters in your cp commands:

$ cp -f test* dir2

$ ls -al dir2

total 12

drwxr-xr-x 2 rich rich 4096 Sep 6 10:55 ./

drwxr-xr-x 4 rich rich 4096 Sep 6 10:46 ../

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test1

-rw-r--r-- 1 rich rich 0 Sep 6 10:55 test2

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

$

TABLE 3-6

The cp Command Parameters:

Parameter Description:

-a Archive files by preserving their attributes.

-b Create a backup of each existing destination file instead of overwriting it.

-d Preserve

-f Force the overwriting of existing destination files without prompting.

-i Prompt before overwriting destination files.

-l Create a file link instead of copying the files.

-p Preserve file attributes if possible.

-r Copy files recursively.

-R Copy directories recursively.

-s Create a symbolic link instead of copying the file.

-S Override the backup feature.

-u Copy the source file only if it has a newer date and time than the destination (update).

-v Verbose mode, explaining what's happening.

-x Restrict the copy to the current filestytem.

Some Examples of cp command:

E.g. $ cp kiran /home/root/mydoc/kiran

copies Kiran file from current directory into specified path

E.g. $ cp file* /tmp

(copies multiple files to /tmp directory)

E.g. $ cp –i file1 file2

(copies file while copying it will prompt for coping)

cp : overwrite 'file2' ? y

E.g. $ cp –bi file1 file2

(copies file while copying it will prompt for coping and creates the backup file)

cp: overwrite 'file2' ? y

$ ls file*

file2 file2~

~ This sign shows the backup file of file2.

(4) Linking files:

If you need to maintain two (or more) copies of the same file on the system, instead of having separate physical copies, you can use one physical copy and multiple virtual copies, called links. A link is a placeholder in a directory that points to the real location of the file.

ln: This command is used to link a file

Links → In linux you can use links to give the same file with two entirely different names or to pretend that the file is in one location in the file system and in actual stored in an entirely different location.

Linking is useful when you have two different programs that look for the same file in different places. So you need to make sure that the file is in both locations.

There are two types of links:

(1) Hard Link.

(2) Soft Link.

(1) Hard Link:

A hard link is just another name for an existing file. The two files share the same inode, so in reality they are same file.

This is different from a copy, where there are two separate files with separate inodes, taking up different blocks on the hard disk. The hard link is to be performed in the same directory in which you are currently logged in.

The inode is a special file that tells the kernel which blocks on the hard disk holds the file because the hard link to a file is actually the same as the original (target) file, you can't tell which is the original file and which is the hard link.

You can create Hard Link by the given command:

In command takes the two argument original file and new file name.

$ ln originalfile linkfile:

The link file should not be created first.

$ ln old new:

To check whether a link file is created properly or not

$ ls –I old new

old

new

since first field displays the inode number and second field display the name of the file. If the inode number of both the file is same then it is sure that both files are linked.

it will create a file new which is link file of the original file old.

$ ln today tom1

$ ln today tom2

$ ls –l today tom1

file	type	file permissions	no. of links	owner	group	other	size	file name
–	rwx rwx rwx		2	ABC	xyz	"	660	today
–	rwx rwx rwx 2		2	ABC	xyz	"	660	tom1

Long listing format shows the number of links for the file today and tom1 as 2.

You can only create a hard link between files on the same physical medium. You can't create a hard link between files under separate mount points. In that case, you'll have to use a soft link.

Symbol Link:

A symbol link is different from a hard link in that it is special file type that contains the name of the original file some what link a shortcut in windows.

Symbolic link is also called as symlink.

A symbolic link file is created that contains a pointer to the original, target file.

To create a symbolic link

$ ln –s abc.doc /kk/xy.doc

It will create symbolic link of abc.doc in the specified directory to a file xy.doc. In this xy.doc should be created first. The linking can identified by red blinking on the path of abc.doc when ls–l ie. long lisitng is to be shown. The symbolic link is identified by file type field of ls–l option which is indicated by l (l in lower case).

The inode number of symbolic link files are not same.

Symbol Link to a directory →

You can create a symbolic link to current directory to specified directory.

$ ln –s /home/chris/letter gifts

$ cd gifts

$ pwd

/home/chris/letter

If you want to display the name of symbolic link cwd variable is used.

cwd is a special system variable names of directory which is symbolic link.

$ pwd

/home/chris/letter

$ cwd

/home/chris/gifts

Difference between Hard Link and Symbolic Link:

Symbolic Link	Hard Link
It contains/hold the pathname of the file to which it is linking.	In hard link the linking is not possible in other directory the linking is performed in current root path only
It can be created by ln with –s option.	It is created directly by ln command.
Symbolic link can be created from one directory to other directory.	Hard links can not be created from one file system to other file system in other directory.
Command is $ ln – s abc /home/xyz/veg	Command is $ ln abc xyz
The symbolic link is identified by l in file type field of long listing format.	The Hard link is identified by number of links option in ls –l option.
In symbolic link the inode	In Hard link the inode number

number of original file and linked file is not same.	of original file and linked file is same. You can see the inode number by a command ls –i original linkfile

Renaming files:

In the Linux world, renaming files is called moving. The mv command is available to move both files and directories to another location:

$ mv test2 test6

$ ls -il test*

1954793 -rw-r--r-- 2 rich rich 6 Sep 1 09:51 test1

1954888 -rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

1954793 -rw-r--r-- 2 rich rich 6 Sep 1 09:51 test4

1954891 lrwxrwxrwx 1 rich rich 5 Sep 1 09:56 test5 -> test1

1954794 -rw-r--r-- 1 rich rich 0 Sep 1 09:39 test6

$

Notice that moving the file changed the filename but kept the same inode number and the timestamp value. Moving a file with soft links is a problem:

$ mv test1 test8

$ ls -il test*

total 16

1954888 -rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

1954793 -rw-r--r-- 2 rich rich 6 Sep 1 09:51 test4

1954891 lrwxrwxrwx 1 rich rich 5 Sep 1 09:56 test5 -> test1

1954794 -rw-r--r-- 1 rich rich 0 Sep 1 09:39 test6

1954793 -rw-r--r-- 2 rich rich 6 Sep 1 09:51 test8

[rich@test2 clsc]$ mv test8 test1

The test4 file that uses a hard link still uses the same inode number, which is perfectly fine.

However, the test5 file now points to an invalid file, and it is no longer a valid link.

You can also use the mv command to move directories:

- The mv command renames (moves) files. It has two distinct functions:
- It renames a file (or directory).
- It moves a group of files to a different directory.
- If the destination doesn't exist, it will be created. For the above example, mv simply replaces the filename in the existing directory entry with the new name.
- Like cp, a group of files can be moved to a directory. The following command moves three files to the progs directory:

  mv chap01 chap02 chap03 progs
- There's a –i option available with mv also, and behaves exactly like in cp.

Deleting files:

rm: Deleting Files

In Linux if you want to delete existing files. Whether it's to clean up a filesystem or to remove a software package, there's always opportunities to delete files.

Deleting is called removing. The command to remove files in the bash shell is rm.

(1) The rm(remove) command deletes one or more files.

(2) The following command deletes three files:

rm chap01 chap02 chap03

rm chap* could be dangerous to use!

(3) A file once deleted can't be recovered. rm won't normally remove a directory, but it can remove files from one.

(4) You may sometimes need to delete all files in a directory as part of a cleanup operation

The *, when used by itself, represents all files, you can then use rm like this:

$ rm * All file gone!

$ _

rm options:

- **Interactive deletion (-i):** Like in **cp,** the –i(interactive) option makes the command ask the user for confirmation before removing each file:

$ rm –i chap01 chap02 chap03

rm: remove chap01 (yes/no)? ?y

rm: remove chap02 (yes/no)? ?n

rm: remove chap03 (yes/no)? [enter]

No response – file is not deleted.

A y removes the file, any other response leaves the file undeleted.

- **Recursive deletion (-r or -R):** With the –r (or -R) option, rm performs a tree walk – a thorough recursive search for all subdirectories and files within these subdirectories. At each stage, it deletes everything it finds. rm won't normally remove directories, but when used with this option, it will. Therefore, when you issue the command

rm -r * Behaves partially like rmdir

- **Forcing removal (-f) rm prompts for removal if a file is write-protected:** The –f option overrides this minor protection and forces removal. When you combine with the –r option, it could be the most risky thing to do:

rm -rf * Deletes everything in the current directory and below

The basic form of the rm command is pretty simple:

$ rm -i test2

rm: remove `test2'? y

$ ls -l

total 16

drwxr-xr-x 2 rich rich 4096 Sep 1 09:42 dir1/

drwxr-xr-x 2 rich rich 4096 Sep 1 09:45 dir2/

-rw-r--r-- 2 rich rich 6 Sep 1 09:51 test1

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

-rw-r--r-- 2 rich rich 6 Sep 1 09:51 test4

lrwxrwxrwx 1 rich rich 5 Sep 1 09:56 test5 -> test1

Notice that the command prompts you to make sure that you're serious about removing the file.

Once you remove a file it's gone forever. Now, here's an interesting tidbit about deleting a file that has links to it:

$ rm test1

$ ls -l

total 12

drwxr-xr-x 2 rich rich 4096 Sep 1 09:42 dir1/

drwxr-xr-x 2 rich rich 4096 Sep 1 09:45 dir2/

-rw-r--r-- 1 rich rich 0 Dec 25 2008 test3

-rw-r--r-- 1 rich rich 6 Sep 1 09:51 test4

lrwxrwxrwx 1 rich rich 5 Sep 1 09:56 test5 -> test1

$ cat test4

hello

$ cat test5

cat: test5: No such file or directory

$

I removed the test1 file, which had both a hard link with the test4 file and a soft link with the test5 file. Noticed what happened. Both of the linked files still appear, even though the test1 file is now gone (although on my color terminal the test5 filename now appears in red). When I look at the contents of the test4 file that was a hard link, it still shows the contents of the file.

When I look at the contents of the test5 file that was a soft link, bash indicates that it doesn't exist any more.

Directory Handling:

- **Creating Directories:**

(1) mkdir command is used to create a new directory.

Syntax is mkdir <dir-name>.

The command is followed by the names of the directories to be

created.

Example: mkdir Kiran

Kiran directory is created in the current root path.

(2) Directories and subdirectories are created with the mkdir (make directory) command.

(3) You can create a number of subdirectories with one mkdir command:

mkdir kiran chirag abc

(4) The system creates a new directory and assigns it a new inode number.

(5) The following command creates a directory tree:

mkdir college college/kiran college/chirag

Creates the directory tree:

This creates three subdirectories–college and two subdirectories under college. The order of specifying the arguments is important; you obviously can't create a subdirectory before creation of it's parent directory for instance, you can't enter

$mkdir college/chirag college/kiran college

mkdir: failed to make directory "college/chirag"; No such file or directory

mkdir: failed to make directory "college/kiran"; No such file or directory

Note that even though the system failed to create the two subdirectories, kiran and chirag, it has still created the college directory.

(6) Sometimes, the system refuses to create a directory:

$ mkdir malcalm

mkdir: failed to make directory "malcalm"; Permission denied

This can happen due to these reasons:

The directory malcalm may already exist.

There may be an ordinary file by that name in the current directory.

The permissions set for the current directory don't permit the creation of files and directories by the users.

Deleting Directories:

The basic command for removing a directory is rmdir:

Syntax : rmdir <dir-name>

$ rmdir Kiran

Removes the directory Kiran.

The following Rules are to be followed while removing or deleting a directory.

(1) The **rmdir** (remove directory) command removes directories. You simply have to do this to remove the directory pis:

rmdir college directory must be empty

The rmdir command only works for removing empty directories.

(2) like mkdir, rmdir can also create and delete more than one directory

in one shot.

(3) For instance, the directories and subdirectories that they were just created with mkdir can be removed by using rmdir with a reversed set of argument:

rmdir college/kiran college/chirag college

(4) The following directory sequence used by mkdir is invalid in rmdir:

$ rmdir college college/kiran college/chirag

rmdir: directory “college”: Directory not empty

(5) This error message leads to two important rules that you should:

Remember when deleting directories:

(a) You can’t delete a directory with rmdir unless it is empty. In this case, the college directory couldn’t be removed because of the existence of the subdirectories, Kiran and chirag, under it.

(b) You can’t remove a subdirectory unless you are placed in a directory which is hierarchically above the one you have chosen to remove.

(6) To illustrate the second cardinal rule, try removing the kiran directory by executing the command from the same directory itself:

$cd kiran

$ pwd

/home/user/college/kiran

$rmdir /home/user/college/kiran

rmdir: directory “/home/user/college/kiran”: Directory does not exist.

(7) To remove the directory, you must position yourself in the directory above Kiran (i.e. You should be out of that directory), i.e., move to college directory, and then remove it from there:

$ cd /home/user/college

$ pwd

/home/user/college

$ rmdir kiran

The mkdir and rmdir commands work only in directories owned by the user.

Viewing File Contents:

We can view the contents of the file and how to view or how to peek inside of them. There are several commands available for taking a look inside files without having to pull out an editor.

- **Viewing file statistics:**

ls command with –l option is used to provide lots of useful information about files in the long listing format. There's still some more information that you can't see in the ls command.

The stat command provides a complete rundown of the status of a file on the filesystem:

$ stat kiran

File: "kiran"

Size: 6 Blocks: 8 Regular File

Device: 306h/774d Inode: 1954891 Links: 2

Access: (0644/-rw-r--r--) Uid: (501/ rich) Gid: (501/ rich)

Access: Sat Sep 1 12:10:25 2009

Modify: Sat Sep 1 12:11:17 2010

Change: Sat Sep 1 12:16:42 2010

$

The results from the stat command show just about everything you'd want to know about the file being examined, even down the major and minor device numbers of the device where the file is being stored.

- **Viewing the file type:**

Stat command produces all status of the file, there's still one piece of information missing – the file type. Before you want to view the contents of the file, it's needed to know what type of file it is, i.e. To identify whether it is binary or octal or text or ascii or device or ordinary or link file or shell script file or it is directory. For that file command is used. It

has the ability to peek inside of a file and determine just what kind of file it is:

$ file test1

test1: ASCII text

$ file myscript

myscript: Bourne shell script text executable

$ file myprog

myprog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), not stripped

$ file Monday reports

monday: text

reports: directory

$ file today mydata

today: Ascii text

mydata: empty

The file command classifies files into three categories:

- Text files: Files that contain printable characters.
- Executable files: Files that you can run on the system.
- Data files: Files that contain nonprintable binary characters, but that you can't run on the system.

The linux file commands helps you to determine for what a file is used. It examines a few lines of files & tries to determine classification for it. The file command is used to display the type of file.

The file cmd looks for a special keybwords or special numbers in those first few lines only but it is not fully accurate. Hence OD' cmd is used to examine the entire file byte by byte 'OD' is the 'octal dump' which performs the dump of a file & it prints every bite in its octal representation.

File <filename> [directory name]

File OD: it prints the contents of file byte by byte either in octal character or a decimal or hexadecimal.

Options	Descriptions
-c	It O/Ps the character of byte value.

-d	It O/Ps a decimal form of byte value.
-x	It O/Ps hexadecimal form of byte value.
-o	It O/Ps octal form of byte value.

- **Viewing the whole file:**

If you have a large text file on your hands, you may want to be able to see what's inside of it.

There are three different commands in Linux to view the files contents.

- **The cat command:**

The cat command is a used for displaying all of the data inside a text file:

$ cat test1

hello

This is a test file.

That we'll use to test the cat command.

$

Contents of the text file will be displayed, there are a few parameters you can use with the cat command. The -n parameter numbers all of the lines for us:

$ cat -n test1

1 hello

2

3 This is a test file.

4

5

6 That we'll use to test the cat command.

$

when you're examining scripts. If you just want to number the lines that have text in them, the -b parameter is for you:

$ cat -b test1

1 hello

2 This is a test file.

3 That we'll use to test the cat command.

$

If you need to compress multiple blank lines into a single blank line, use the -s parameter:

$ cat -s test1

hello

This is a test file.

That we'll use to test the cat command.

$

if you don't want tab characters to appear, use the -T parameter:

$ cat -T test1

hello

This is a test file.

That we'll use to^Itest the cat command.

$

The -T parameter replaces any tabs in the text with the ^I character combination.

Using the more command to display a text file

For large files, the cat command is not advisable. The text in the file will just quickly scroll off of the monitor without stopping more command will this problem.

➢ **The More Command:**

The more command displays a text file, but stops after it displays each page of data. At the bottom of the screen the more command displays a tag showing that you're still in the more application and how far along in the text file you are. Like – more – This is the prompt for the more command more command is also called as **PAGING OUTPUT.**

- The man command displays its output a page at a time. This is possible because it sends its output to a pager program. UNIX offers the more pager (originally from Berkeley) which has today replaced pg, the original pager of UNIX. Linux also offers more but less is its standard pager.

- To view the file chap01, enter the command with the filename:

  more chap01 Press q to exit

- You'll see the contents of chap01 on the screen, one page at a time. At the bottom of the screen, you'll also see the filename and percentage of the file that has been viewed:

  – more – (17%)

- more has a couple of internal commands that don't show up on the screen when you invoke them.q, the command used to exit more, is an internal command.

Navigation:

Irrespective of version, more uses the spacebar to scroll forward a page at a time. You can scroll by small and large increments of lines or screens. To move forward one page, use f or the spacebar And to move back one page, use b

The Repeat Features:

- **The Repeat factor:** Many navigation commands in **more,** including **f** and **b**, use a repeat factor. This is the term used in **vi** to prefix a number to vi internal command use of the repeat factor as a command prefix simply repeats the command that many times.

- This means you can use **10f** for scrolling forward by 10 pages and **30b** for scrolling back 30 pages just remember that the commands themselves are not displayed on the screen – even for a moment.

- **Repeating the last command ( ) more** has a repeat command, the dot (same command used by **vi**), that repeats the last command you used. If you scroll forward with **10f,** you can scroll another 10 pages by simply pressing a dot.

Searching for a Pattern:

You can perform a search for a pattern with / command followed by the string. For instance, to look for the first while loop in your programs, you'll have to enter this:

/while Press [enter]also

You can repeat this search for viewing the next while loop section by pressing n, and you can do that repeatedly until you have scanned the entire file. Move back with b (using a repeat factor, if necessary) to arrive at the first page.

The more Command Options Option Description:

H Display a help menu.

spacebar Display the next screen of text from the file.

z Display the next screen of text from the file.

ENTER Display one more line of text from the file.

d	Display a half-screen (11 lines) of text from the file.
q	Exit the program.
s	Skip forward one line of text.
f	Skip forward one screen of text.
b	Skip backward one screen of text.
	/expression Search for the text expression in the file.
n	Search for the next occurrence of the last specified expression.
_	Go to the first occurrence of the specified expression.
!	cmd Execute a shell command.
v	Start up the vi editor at the current line.
	CTRL-L Redraw the screen at the current location in the file.
=	Display the current line number in the file.
·	Repeat the previous command.

➢ **The Less Command:**

Less command is actually a play on words and is an advanced version of the more command (the less command uses the phrase ‘‘less is more’’).

(a) It provides several features for scrolling both forward and backward through a text file

(b) It provides some advanced searching capabilities.

(c) The less command is used to display the contents of a file before it finishes reading the entire file. This is a drawback for both the cat and more commands when using extremely large files.

(d) The less command operates much the same as the more command, displaying one screen of text from a file at a time. Kj

(e) Less command provides additional information in its prompt, showing the total number of lines in the file, and the range of lines currently displayed.

(f) The less command supports the same command set as the more command, plus lots more options. 3

(g) The less command recognizes the up and down arrow keys, as well as the page up and page down keys. This gives you full control when viewing a file.

Internal commands of more and less:

More	Less	Action
Spacebar or f	Spacebar or f or z	One page forward
20f	-	20 pages forward
B	B	One page backward
15b	-	15 pages back
[Enter]	J or [Enter]	One line forward
-	K	One line back
-	P or 1g	Beginning of file
-	G	End of file
/pat	/pat	Searches forward for expression pat
N	N	Repeats search forward
-	?pat	Searches back for expression pat
.(a dot)	-	Repeats last command
V	V	Starts up vi editor
!cmd	!cmd	Executes UNIX command cmd
Q	Q	Quit
H	H	View Help

- **Viewing parts of a file:**

If we want to view the contents which are located right at top or at bottom of a file. Head & Tail command is used to achieve this.

Head:

Displays the first ten lines of a file, unless otherwise stated.

Syntax:

head [-number | -n number] filename

-number	The number of the you want to display.
-n number	The number of the you want to display.
filename	The file that you want to display the x amount of lines of.

Examples:

head-15 myfile.txt: Would display the first fifteen lines of myfile.txt.

Tail Command:

Syntax:

tail [+ number] [-l] [-b] [-c] [-r] [-f] [-c number |-n number] [file].

<table>
<tr><td>+number
-number</td><td>This option is only recognized if it is specified first. COUNT is a decimal number optionally followed by a size letter (`b', `k', `m') as in `-c', or `l' to mean count by lines, or other option letters (`cfqv').</td></tr>
<tr><td>-l</td><td>Units of lines.</td></tr>
<tr><td>-b</td><td>Units of blocks.</td></tr>
<tr><td>-c</td><td>Units of bytes.</td></tr>
<tr><td>-r</td><td>Reverse. Copies lines from the specified starting point in the file in reverse order. The default for r is to print the entire file in reverse order.</td></tr>
<tr><td>-f</td><td>Follow. If the input-file is not a pipe, the program will not terminate after the line of the input-file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input-file. Thus it may be used to monitor the growth of a file that is being written by some other process.</td></tr>
<tr><td>-c number</td><td>The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes, to begin the copying:
<table>
<tr><td>+</td><td>Copying starts relative to the beginning of the file.</td></tr>
<tr><td>-</td><td>Copying starts relative to the end of the file.</td></tr>
<tr><td>none</td><td>Copying starts relative to the end of the file.</td></tr>
</table>
The origin for counting is 1; that is, -c+1 represents the first byte of the file, -c-1 the last.</td></tr>
<tr><td>-n number</td><td>Equivalent to -c number, except the starting location in the file is measured in lines instead of bytes. The origin for counting is 1; that is, -n+1 represents the first line of the file, -n-1 the last.</td></tr>
<tr><td>File</td><td>Name of the file you wish to display</td></tr>
</table>

Examples:

tail myfile.txt

The above example would list the last 10 (default) lines of the file myfile.txt.

tail myfile.txt -n 100

The above example would list the last 100 lines in the file myfile.txt.

6.6 FILE SYSTEM

As you can see from the shell prompt, when you start a shell session you are usually placed in your home directory. Most often, you will want to break out of your home directory and want to explore other areas in the Linux system.

The Linux filesystem OR The File structure:

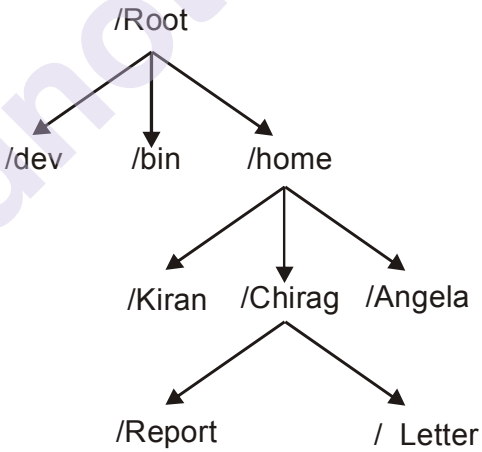
The linux file organizes into tree structure format connected set of directories each directory contains either files or directories.

Directory perform 2 main functions:

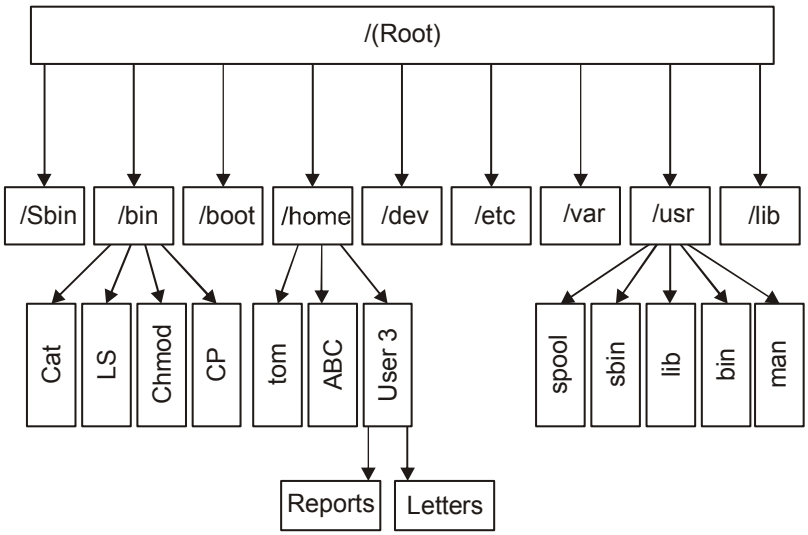
1) A directory hold files.

2) Directory connects to other directories like a branch in a tree which can be connected to other branches also

Files and directories are shown in the tree structure format.



The tree can be shown by root at the top. Extending down from the root are the branches. Each branch grows out of other branch but it can have many lower branches it can said to be parent child structure. In the same way each directory is a subdirectory of one other directory i.e. each directory is a child of parent directory. Root is identified by a forward slash (/), within the root directory number of system directories are built, root directory also contain home directory which contains the info of all users in the system and each user home directory i.e. Chirag in turn contains the directory which the user has made for his use.
The Full Linux file structure.

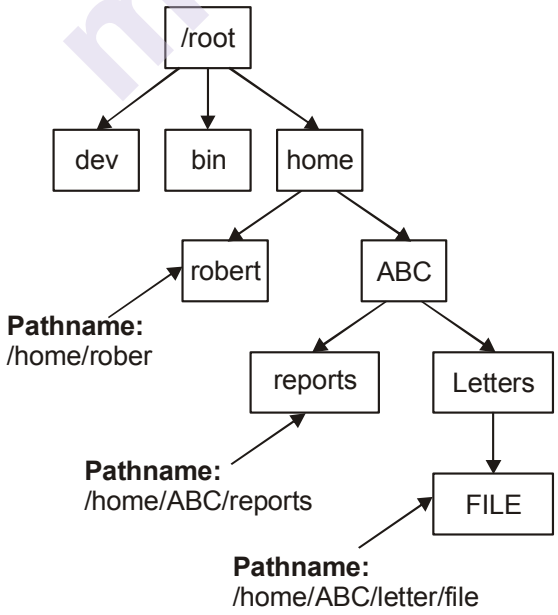


Home Directories:

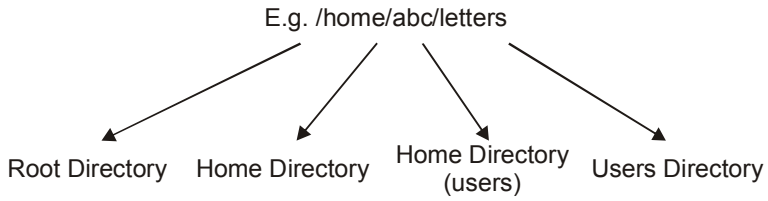
When you log on to system you are placed within home directory. The name given to this directory by system is the same as your login name. You can crate files in home directory also you can create more directories. You can change to these directories and stores file in them. Same is true for users on the system i.e. each and every users will have his own directory identified by login name and user in turn can create their own directories and subdirectories.

Path Name:

The full name of the directory to identify that directory is the path name. The hierarchically nested relationship among directories forms path and these path can be used to identify and reference any directory or file. A path exist from '/' i.e. root directly to home directory.



While writing the pathname by listing of each directory the pathname is separated by '/'. When we are writing any path starts with "/" indicate root directory receiving directory name



Traversing Directories:

The change directory command (cd) is what you'll use to move your shell session to another directory in the Linux filesystem.

The format of the cd command is:

cd destination

The cd command may take a single parameter, destination, which specifies the directory name you want to go to. If you don't specify a destination on the cd command, it will take you to your home directory.

The destination parameter, though, can be expressed using two different methods:

- An absolute filepath
- A relative filepath

Pathnames are of two types:

(1) Absolute path.

(2) Relative path.

Absolute path is complete pathname of a directory or file always begins with root directory. The absolute filepath defines exactly where the directory is in the virtual directory structure, starting at the root of the virtual directory. Sort of like a full name for a directory.

E.g: /usr/lib/apache

(1) If the first character of a pathname is / , the file's location must be determined with respect to root (the first /). Such a pathname is called an **absolute pathname.**

(2) When you have more than one / in a pathname, for each such / , you have to descend one level in the file system.

Relative Pathnames:

You would have noted that in a previous example, we didn't use an absolute pathname to move the directory progs.

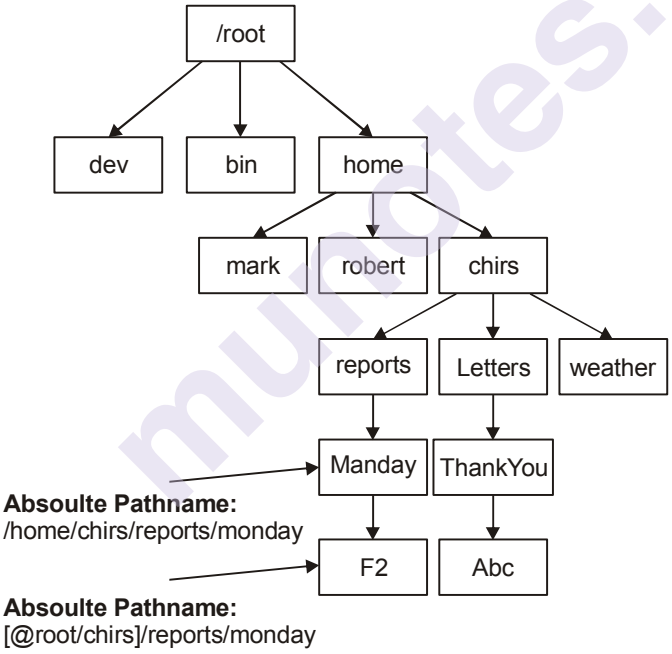
Relative pathname begins from your working directory working directory is the one you are currently in. Relative filepaths allow you to specify a destination filepath relative to your current location, without having to start at the root. A relative filepath doesn't start with a forward slash, indicating the root directory. Instead, a relative filepath starts with either a directory name (if you're traversing to a directory under your current directory), or a special character indicating a relative location to your current directory location. The two special characters used for this are:

- The dot (.) to represent the current directory.
- The double dot (..) to represent the parent directory.

The double dot character is extremely handy when trying to traverse a directory hierarchy. For example, if you are in the Documents directory under your home directory and need to go to your Desktop directory, also under your home directory, you can do this:

rich@1[Documents]$ cd ../Desktop

rich@1[Desktop]$



Sometimes the absolute path could be complex hence to refer absolute path. You can use a special character tilde ' ~ ' which represents absolute path name of your home directory. Suppose currently you are in 'thank you' directory and you want to see contents of 'abc' file by using absolute path, so you can write $ cat ~ / abc. Hence tilde (~) will give you absolute path as '/home/chris/letters/thankyou', Same way if you are in chris directory & you want to see contents of file F2 of Monday directory. So you can use tilde sign (~) before using it check you absolute path by command 'pwd' which display the present working directory. Hence we

can write $ cat ~/reports/Monday/F2. So it displays the contents of F2 Where tilde (~) indicates the path /home/chris.

6.7 FILESYSTEM ARCHITECTURE

The root directory that begins with linux file structure contains several system directories. The system directory contains files and programs used to run and maintain the system. Many directories contains other sub directories with programs for executing specific features of LINUX

e.g. /usr/bin

Contains various LINUX commands that user executes such as cp – for copy and mv – for renaming or moving to other directory & the directory /bin holds interfaces with different system devices such as printer or the terminal also consist of the executable file for the commands.

	Directory	Function
(1)	Root '/ '	It begin file system structure call the root, The root of the virtual directory.
(2)	/home	Contains users home directory.
(3)	/bin	The binary directory holds all standard commands and utility programs (like Vi editor), also executable files for the commands.
(4)	/usr	Holds those files & commands used by the system. This directory breaks down into several sub directories. The user-installed software directory.
(5)	/usr/sbin	Holds system administration commands with its executable files.
(6)	/usr/lib	It holds libraries for programming language
(7)	/usr/doc	Holds linux documentations.
(8)	/usr/man	Holds online manual man files i.e. Help files.
(9)	/usr/spool	Holds spool files such as those generated for printing jobs, waiting printing jobs will be stored in spool directory.
(10)	/sbin	Holds system administration command for booting of system. The system binary directory, where many GNU admin-level utilities are stored.
(11)	/var	It holds the file that vary. Hence this directory has information about different utilities of linux e.g. /val/log. – directory contains files that stores operating system log files or error reports file. The variable directory, for files that change frequently, such as log files.
(12)	/dev	Holds file interfaces for devices such as terminal and printers. It stores the device files for input and output hardware devices.
(13)	/etc	Holds system configuration files and any other system files. This stores operating system related data which users and operating system needs to refer such as password file.

(14)	/tmp	The temporary directory, where temporary work files can be created and destroyed.
(15)	/mnt	The mount directory, another common place for mount points used for removable media.
(16)	/Lib	The library directory, where system and application library files are stored.
(17)	/boot	The boot directory, where boot files are stored.

6.9 FILE ATTRIBUTES

You can decode the cryptic file permissions you've seen when using the ls command. Here we will specify how to decipher the permissions.

Using File Permission Symbols:

The ls command displays the file permissions for files, directories, and devices on the Linux system:

```
$ ls -l

total 68

-rw-rw-r--  1  rich  rich  50  2007-09-13  07:49
file1.gz

-rw-rw-r-- 1 rich rich 23 2007-09-13 07:50 file2

-rwxrwxr-x  1  rich  rich  4882  2007-09-18  13:58
myprog

-rw-rw-r--  1  rich  rich  237  2007-09-18  13:58
myprog.c

drwxrwxr-x  2  rich  rich  4096  2007-09-03  15:12
test1

drwxrwxr-x  2  rich  rich  4096  2007-09-03  15:12
test2

$
```

The first field in the output listing is a code that describes the permissions for the files and directories.

The first character in the field defines the type of the object, These are the different options of file types.

- for files
- d for directories
- l for links
- c for character devices

- b for block devices
- n for network devices

After that, there are three sets of three characters. Each set of three characters defines an access permission triplet:

- r for read permission for the object
- w for write permission for the object
- x for execute permission for the object

If a permission is denied, a dash appears in the location. The three sets relate the three levels of security for the object:

- The owner of the object
- The group that owns the object
- Everyone else on the system

-rwxrwxr-x 1 rich rich 4882 2007-09-18 13:58 myprog

The file myprog has the following sets of permissions:

- rwx for the file owner (set to the login name rich)
- rwx for the file group owner (set to the group name rich)
- r-x for everyone else on the system

These permissions indicate that the user login name rich can read, write, and execute the file (considered full permissions). Likewise, members in the group rich can also read, write, and execute the file. However, anyone else not in the rich group can only read and execute the file; the w is replaced with a dash, indicating that write permissions are not assigned to this security level.

Default File Permissions:

The umask command sets the default permissions for any file or directory you create:

$ touch newfile

```
$ ls -al newfile
-rw-r--r-- 1 rich rich 0 Sep 20 19:16 newfile
$
```

The touch command created the file using the default permissions assigned to my user account. The umask command shows and sets the default permissions:

$ umask

$

Unfortunately, the umask command setting isn't overtly clear, and trying to understand exactly how it works makes things even muddier. The first digit represents a special security feature called the *sticky bit*. The next three digits represent the octal values of the umask for a file or directory. To understand how umask works, you first need to understand octal mode security settings. *Octal mode* security settings take the three rwx permission values and convert them into a 3-bit binary value, represented by a single octal value. In the binary representation, each position is a binary bit. Thus, if the read permission is the only permission set, the value becomes r--, relating to a binary value of 100, indicating the octal value of 4.

Octal mode takes the octal permissions and lists three of them in order for the three security levels (user, group, and everyone). Thus, the octal mode value 664 represents read and write permissions for the user and group, but read-only permission for everyone else.

The permission for a file is broken into three parts. Each group represents a category and contains three slots representing the read, write and execute permissions of the file.

The first group (rwx) has all three permissions. The file is readable, write able and executable by the owner.

The second group (r-x) has a hyphen in the middle slot, which indicates the absence of write permission by the group user of the file.

The third group (r--) has the write and execute bit absent. The file permissions are also categorized for the kinds of user ie it can be other/ guest user.

6.10 WORKING WITH FILES

The **chmod** command is used for changing the permissions of a file.

Permissions can be changed by two ways:

(1) Binary Masking Method (Absolute Permission Method):

In this method binary 1 or 0 is assigned to the permissions

1 → Assigning (Granting permission)

0 → Removing (Revoking permission)

Syntax for Binary Masking:

Chmod	Binary Number	filename
Owner	Group	Other
111	101	001

7 5 1

Chmod 751 hello.txt

7 indicate (111) Owner has all three permissions (Read, Write, Execute)

5 indicate (101) Group user has Read & Execute Permission.

1 indicate (001) Other or Guest User has Only Execute Permission.

2) Symbolic Masking Method (Relative Method of changing permissions):

In this method symbols (Abbreviations) are used for assigning & Removing Permissions:

r → Read

w → Write

x → Execute

g → group

o → Other

u → Owner

a → All users

\+ → Assinging a permission (plus sign)

\- → Removing a permission. (minus sign)

Syntax:

chmod category operation permission filenames

category	can be user, group or other
operation	can be assign or remove
permission	can be read, write and execute

Example:

Chmod g+rw hello.txt

Chmod a-wx xyz.txt

6.11 BACKUP, COMPRESSION

Compressing Data:

The zip utility allows you to easily compress large files (both text and executable) into smaller files that take up less space.

Linux contains several file compression utilities. While this may sound great, it often leads to confusion and chaos when trying to download files. Lists the file compression utilities available for Linux.

The compress file compression utility is not often found on Linux systems. If you download a file with a .Z extension, you can usually install the compress package (called ncompress in many Linux distributions) and then uncompress the file with the uncompress command.

The gzip command compresses every file in the directory that matches the wildcard pattern.

The zip utility:

The zip utility is compatible with the popular PKZIP package created by Phil Katz for MS-DOS and Windows. There are four utilities in the Linux zip package:

- zip creates a compressed file containing listed files and directories.
- zipcloak creates an encrypted compress file containing listed files and directories.
- zipnote extracts the comments from a zip file.
- zipsplit splits a zip file into smaller files of a set size (used for copying large zip files to floppy disks).

unzip extracts files and directories from a compressed zip file.

zip [-options] [-b path] [-t mmddyyyy] [-n suffixes] [zipfile list]

[-xi list]

The default action is to add or replace zipfile entries from list, which can include the special name - to compress standard input.

If zipfile and list are omitted, zip compresses stdin to stdout.

-f freshen: only changed files -u update: only changed or new files.

-d delete entries in zipfile -m move into zipfile (delete files).

-r recurse into directories -j junk directory names.

-0 store only -l convert LF to CR LF.

-1 compress faster -9 compress better.

-q quiet operation -v verbose operation.

-c add one-line comments -z add zipfile comment.

-@ read names from stdin -o make file as old as latest entry.

-x exclude the following names -i include only the following names.

-F fix zipfile (-FF try harder) -D do not add directory entries.

-A adjust self-extracting exe -J junk zipfile prefix (unzipsfx).

-T test zipfile integrity -X eXclude eXtra file attributes.

-y store symbolic links as the link instead of the referenced file.

-R PKZIP recursion (see manual).

-e encrypt -n don't compress these suffixes.

$

The power of the zip utility is its ability to compress entire directories of files into a single compressed file. This makes it ideal for archiving entire directory structures:

$ zip -r testzip test

adding: test/ (stored 0%)

adding: test/test1/ (stored 0%)

adding: test/test1/myprog2 (stored 0%)

adding: test/test1/myprog1 (stored 0%)

adding: test/myprog.c (deflated 39%)

adding: test/file3 (deflated 2%)

adding: test/file4 (stored 0%)

adding: test/test2/ (stored 0%)

adding: test/file1.gz (stored 0%)

adding: test/file2 (deflated 4%)

adding: test/myprog.gz (stored 0%)

$

This example creates the zip file named testzip.zip, and recurses through the directory test,

Adding each file and directory found to the zip file. Notice from the output that not all of the files stored in the zip file could be compressed. The zip utility automatically determines the best compression type to use for each individual file.

When you use the recursion feature in the zip command, files are stored in the same.

Directory structure in the zip file. Files contained in subdirectories are stored in the zip.

File within the same subdirectories. You must be careful when extracting the files, the unzip command will rebuild the entire directory structure in the new location.

File Compression:

If you are transferring the file across a N/W to save transmission time. You can effectively reduce the size by creating a compressed copy of it. Anytime you need file again decompress it.

$ gzip mydata

$ mydata.gz drip1

To compress a gzip file

$ gzip –d mydata.gz

To decompress a gzip

$ gunzip mydata.gz

$ ls

$ gzip 0-cmydata preface>mufiles.gz sends compressed version of a file to standard o/p each file listed is separately compressed extension will be gz.

-h displays help help listing

-l file list

$ gzip -l myfiles.gz

-r dir name

-v file list

-num

You can also zip with bzipz (Burrows-wheeler block sorting text compression algorithm)

The file created with extension .bzz bzip command compresses file in block & enables you to specify their size.

$ bzipz mydata

$ ls

mydata bzz

zip also created zip files:

$ zip mydata

$ zip –r reports

$ ls

$ unzip mydata.zip

mydata.zip

Archiving (tar):

The tar command is used to store backup transport & archive files. A tar file can be made on tape drive or on local hard disk. tar command serves many file together in a single tape or disk archive & can single tape or disk archive & can restore individual file from the archive i.e. tar file is a single file that contains the contents of many file also store file attributes like file access permission the user, group, size time. The files in tar file are called the members of that archive.

The tar utility creates archive for files & directories with tar you can archive specific files, update them in the archive & add new files as you want to that archive. You can even archive entire directories with all their files & subdirectories all of which can be restored from archive.

The tar utility was originally designed to create archives on tapes.

The term **tar stands for tap archive**, also you can create archives on any device such as floppy disk or you can create an archive file to hold the archive (for devices archiving is not possible in DOS only for files it is possible)

The tar utility is ideal for making backups of your files or combining several files into a single file for transmission across a N/W.

On Linux, tar is used to create archives on devices of files.

You can direct tar to archive files to a specific device or a file by using **'f '** option with the name of device or file.

The syntax for the tar command with f option

$ tar option f archive name tar Directory & file name

When creating a file for a tar archive, the file name is usually given the extension .tar

If directory name is specified for archive then all its subdirectories & included files are in archive.

To create an archive use c option along with f option or file.

c creates on archive on a file or device

$ tar cvf myarch.tar mydir:

v→ verbose mode which means that it displays detailed comments as the operation proceeds like verification.

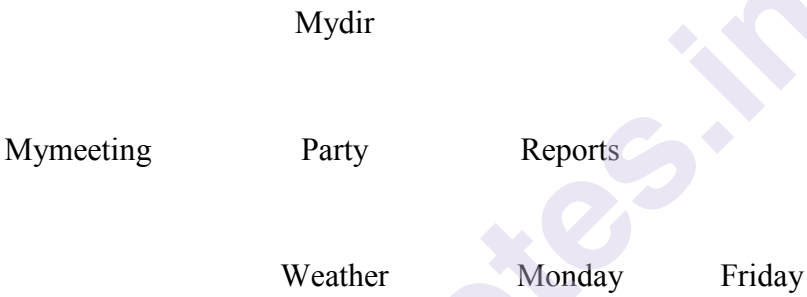
c→ creating a new archive file i.e. tar file display each file name is it archive for verifying

f→ file i.e. specify the name of tar file or location where it is to be created.

In this example the directory mydir & its all subdirectories are saved in the fie myarch.tar

myarch.tar → name of archive to be created .

mydir → directory to be archived.




```
$ tar cvf myarch.tar mydir
mydir/
mydir/reports/
mydir/reports/weather
mydir/reports/Monday
mydir/reports/Friday
mydir/mymeeting
mydir/party
```

Command		Execution
tar options	**:**	Back up files to tape device or archive file
tar option archive name file list	**:**	Backs up file to specific file or device specified as archive name, filelist, can be filenm or directory

Options:

c **:** creates a new archive

t	**:**	lists the names of files in an archive
r	**:**	append file to an archive
u	**:**	Update an archive with new & changed files, adds only those files modified ♀ they were archived or files not already present in archive
w	**:**	waits for a confirmation from the user before archiving each file
x	**:**	extracts files from an archive
m	**:**	created a multiple volume archive that may be stored on several floppy drives
f archive – nm	**:**	saves the tape archives to the file archive name instead of the default tape device, when in this option gives an archive name, the f option saves the tar archive in file of that name which is specified as archive name
f device –nm	**:**	Saves a tar archive to a device such as floppy disk or tape /dev/fdo is the device name For your floppy disk the default devices is held in /etc/default/tar file
v	**:**	Displays each filename an it is archived
z	**:**	Compresses or decompresses archived files using g zip

The user can extract the directories from the tape using X option. The xf option extracts file from an archive file or device. The tar extension operation generates all subdirectories.

xf option directs for to extract all the files & subdirectories form the tar files.

myarch.tar

```
$ tar xvf myarch.tar
```

```
$ tar xvf myarch.tar
```

pathname denotes the relative pathnames. The above command creates the root directory under the current directory if it does not already exist.

```
mydir/

mydir/reports

mydir/reports/weather

mydir/reports/Monday

mydir/reports/Friday

mydir/mymeeting

mydir/party
```

You can use r option to add files to an already created archive. The r option appends the files to the archive

$ tar rvf myarch.tar mydocs:

The user appends the files in the myarch.tar archive. Here the directory mydocs & its files are added to the myarch.tar archive.

If you want to do change or update to the previously created archived, you can use **u** option to instruct tar to update the archive with an modified file.

The tar command compares the time of the last update for each archive file with those in users directory & copies into the archive any files that have been changed since they were last archived.

Any newly created files in these directories are also added to the archive suppose mydir directory you have added two or 3 files & you want to update that in tar**.**

$ tar uvf myarch.tar mydir

mydir/

mydir/gifts

To see tar archive file stored in archive tar without option

$ gzip mydata

$ gunzip mydata

Since tar files are capable of preserving files info. & directory structure tar is commonly used to perform full & incremented backups of disks.

Whenever any file corrupt the chances of recovering it are higher if it is uncompressed tar file as well as you cannot update compressed file.

$ tar –cvf trial.tar /root/test/* tar;

Removing leading '/' from absolute path names in archieve

```
root/test/a out

root/test/m.c.
```

tar file will not store the leading slash (/) it removes the slash while retrieving a file as the tar file does not store the absolute filename, it restores the file with relation to your current directory & prevents accidental overwriting of original data.

If you want absolute path to be stored use –p (capital P) option

-tar - -absolute -paths -paths -cvf trial.tar/root/test/*

Or

 # -tar -p -cvf trial.tar/root/test/*

Storing the absolute pathnames we can created archive filename will long names also instead of using only single character # tar –c –v -f trial.tar/root/user/

 # tar --create - - verbose - - file = trial.tar/root/user/*

Deleting files form an Archive:

use -delete option

```
# tar - - delete –f trial.tar tmp

# tar -tvf trial.tar

a.txt

m.c.

tmp will be deleted
```

Concatenating Tar.Archives

 # tar –Af trial.tar script.tar

The above command adds all contents of script.tar archive to the trial.tar archive. To backup the files to a specific device, specify the device as the archive for a floppy disk, you can specify the floppy drive. Be sure use Blank floppy otherwise any data previously placed on it will be erased by this operation.

The user created an archive on floppy disk in the /dev/fdo device device & copies into floppy disk all the files which all in mydir directory.

- **$ tar cf /dev/fdo mydir:**

To extract the backed up files on the hard disk in a device

$ tar xf /dev/fdo

If the files you are archiving take up more space than would be available on a device such as floppy disk, you can create a for archive that uses the multiple labels. The m option instruct for to prompt you for a new storage component when the current one is filled when archiving to a floppy drive with m option tar prompts you to put in a new floppy disk when one becomes full. You can then save your tar archive on several floppy disks.

$ tar cmf /dev/fdo mydirect

To unpack the multiple disk archive i.e. to take the multiple floppy disk data form floppy to hard disk.

Place the 1st one in the floppy drive, & then issue the following tar command using both the x & m options. You are then prompted to put in the other floppy drive as they are needed.

$ tar xmf /dev/fdo

The tar operation does not perform compression on archived files. if you want to compress the archive files you can instruct tar to involve the gzip utility to compress them with the lower case z option tar first uses gzip to compress file before archiving them. The same z option involves gzip to decompress them when extracting files.

$ tar czf myarch.tar.gz mydir

Normally an archive is created for transferring several files at one as one tar file to shorten transmission time the archive should be as emall as possible extention of zip files i.e. tar.gz

To view the contents of the compressed tar file

```
# tar tufz trial.tar.gz

# gzip myarch.tar

$ ls

$ myarch.targz
```

6.12 SUMMARY

In this chapter we learn documentation and types of compression utilities and file system architecture

6.13 UNIT END QUESTIONS

1. Write a short note on man pages, GNU info help command.
2. Write different file operation used in linux.
3. Write a short note on Filesystem.
4. Describe Filesystem architecture.
5. Describe different file system.
6. Write different file attributes
7. Explain 1. Backup 2.Compression

List Of References:

1) Unix Concepts and Applications by Sumitabha Das.

2) Official Ubuntu Book, 8th Edition, by Matthew Helmke & Elizabeth K. Joseph with Jose Antonio Rey and Philips Ballew, Prentice Hall

*****

UNIT III

7

SECURITY

Unit Structure

7.0 OBJECTIVES

In this chapter you will learn about:

- Basic of linux security under the use of sudo command and password policies. Also the ssh-secure shell uses will enhance the linux security.

7.1 INTRODUCTION

This chapter introduces linux security measures regarding network criteria by using ssh command features. The password policies which are modified by using user management command. The sudo command which are helping to manage all linux security and password policies.

7.2 UNDERSTANDING LINUX SECURITY

- Security is a set of appropriate procedures to protect your data, account against risks.
- Risks for Linux users are compromised account, system compromise, infrastructure related issues, not good user and system administration.
- One important task is to understand why we need to secure a system.
- Linux being treated as highly secure operating system, it has some security flaws.

- Causes of security problem in linux are local security, root security, file system security.
- Security system is in two parts:
  - Authentication: Responsible for ensuring that a user requesting access to the system is really the user with the account.
  - Access Control: Responsible for controlling which resources each account has access to and what kind of access is permitted.

Security Requirements:

- **Authorization:** Allowing authorized user to access data
- **Authenticity:** Verifying them.
- **Confidentiality:** Personal information not been compromised.
- **Integrity:** Data not been changed or modified.
- **Availability:** Ensure that data is available.

Linux Security Systems and Tools:

- Reason for Linux is less vulnerable to attack is in Linux file system model. Files are handled differently in a Linux system. One way is that each file on the system has the concept of permission built in. These permissions separate who can use the file and how.
- Most of the files on a Linux system belong by default to 'root', or the system administrator account. Within this category, no non-root user can write to system files, and some programs which are risky security-wise can additionally be restricted so that only root can run them.
- The user access levels also apply to any program that the user runs. That is even if a user downloads and runs a malicious program, that program inherits the user's permissions and so cannot do anything that the user themselves could not do.
- Each file has the system's file type identifier embedded in the file itself instead of relying in an extension. Thus executables do not necessarily have ".exe" at the end, and plain text files do not need ".txt" at the end. Thus one cannot fools the system by making an executable file with a ".pdf" extension, since even if the user naively double-clicks on the PDF file to open it for reading the system will know that it's really an executable program and refuse to run it.

Firewalls:

- Firewalls are network packet filters that are capable of blocking unwanted network traffic, while passing through allowed traffic.

- A firewall uses a set of rules which determines which traffic is allowed to pass and in which direction. These are normally used to separate internal networks from external ones.
- Firewalls are often the first line defense against crackers and internet worms, which can be blocked by denying the means of network ingress.

7.3 USES OF ROOT

- Root is username or account who has access to all commands and files referred to as root user or superuser.
- In Linux system two people can change the permission of a file or directory.
  - The owner of file
  - The root user
- Root user is a superuser who can do anything on the system to maintain the system.
- It's actually a directory represented by "/".
- It has number of sub directories such as bin,dev,home,lib etc.
- It has write permission i.e to modify files.
- Root user can start up or shut down the system and change operating mode such as single user mode.
- It can add or remove users, file systems, back up and restore files.
- Create new process or kill process if required for running of system.
- Any user account's password can be change
- Setting of system date and clock.
- Communicate with concurrent user.
- Set the limitations to the user account like creating number of files, disk space allowed to user.
- Schedule services using cron.
- Configuration of networking services FTP, SSH etc.

7.4 SUDO COMMAND

- Its "superuser do" which allows user with proper permission to execute a command as another user.

- In Ubuntu Linux by default root account is not configured
- Linux sudo command is used to give permission to any particular command such as when user tries to install,remove and change piece of software that user wants to execute.
- During installation of Ubuntu a default user is created and default user is set up with sudo permission
- Sudo requires that users authenticate with a password.
- By default it is user's password not the root password.

Attribute	Description
-v	Prints version number and exist
-l	List will print out the commands allowed the user on the current host
-h	Help prints a usage message and exist.
-b	Background runs the given command in background
-K	Sure kill removes the user's timestamp entirely
-u	User option to run the specified command as a user other than root
-s	Shell option runs the shell specified
-e	Edit option indicates that instead of running a command user wish to edit one or more files.

For example user wants to update the operating system by passing command:
apt-get update

7.5 WORKING WITH PASSWORDS

Password and authentication are important concepts when working in Linux environment

Rules for Good Password:

- Choose different password: If you have a fear that your password has been hacked them immediately change it.
- Choose uncommon password: Don't choose common names of your friends, relatives, pets etc.
- Use mixture characters: Use characters, numbers and special characters in mixed form.
- Length of password: Minimum characters should be of 6 in length
- Don't keep password written anywhere.

Password Security:

- As password is an important issue if at the time of login when we enter user ID and password and if is accurate then login successes otherwise it fails.
- It is important for users to secure passwords and it should be unguessable.
- Recently Linux distribution includes passwd programs that do not allow to set easily guessable passwords.
- It has one store area where password information is stored i.e. /etc/passwd
- Linux file system is case-sensitive
- No person can regenerate original password string from the encryptd string, thus password is secured.

7.6 BYPASSING USER AUTHENTICATION

- Most of the system require access to private information
- It is possible to bypass authentication measure by tempering with request and tricking the application into thinking that we are already authenticated.
- In authentication mechanism we ask a user name and password at login and password at login page, then allow authorized users unrestricted access to other web pages without any further checking.
- The problem is if users go directly to configuration pages, bypassing authentication.
- For e.g. One user to run a command as another user without supplying a password
- The solution to this is use sudo nopasswd tag. Which indicates to sudo that no password is needed for authentication.

7.7 UNDERSTANDING SSH

- ssh means secure shell was created to provide the best security when we are accessing another computer remotely.
- ssh provides better authentication facility, secure file transfer.
- ssh encrypts any communication between the remote user and a system on your network.
- Two implementation of ssh are ssh1 which uses original ssh protocol and ssh2 uses rewritten version of ssh protocol.
- Openssh is version supplied with Linux distributions.

- When you connect through ssh, you are in shell session, a text based interface where we can interact with server.
- In ssh session, any commands that we type in local terminal are sent through an encrypted ssh tunnel and executed on server.
- The ssh connection is implemented by client server model where remote machine must be running a piece machine must be running a pience of software for connection to be established.
- The user's computer has a ssh client.

ssh Encryption and authentication:

- ssh secures connections by authenticating users and encrypting transmissions.
- ssh first authenticates particular host,verifies it if is valid ssh host for srcure communication.

Encryption:

- Public key encryption in ssh authentication uses two keys: public key and private key.
- Public key is used to encrypt data and private ey to decrypt it.
- For eg. When host sends data to a user on another system, the host encrypts the authentication data with public key previously received from that user.
- Data can be decrypted by user's private key.

Authentication:

- ssh authentication is first carried out with the host and then with users.
- When a remote user receives the encrypted challenge, that user decrypts the challenge with its private key.
- The remote user first encrypts a session identifier using its private key.
- The encrypted session identifier is then decrypted by the account using the remote user's public key.

ssh Tools:

- sssh is implemented on Linux systems with openSSH.
- OpenSSH packages include general openSSH package, openSSH server and openSSH clients.

- The following are the tools:

Application	Description
Sshs	ssh client
Sshd	ssh server
Scp	ssh copy command client
Sftp	ssh ftp client
sftp-server	ssh ftp server
ssh-keygen	utility for generating keys
slogin	remote login
ssh-agent	ssh authentication agent

7.7 LET US SUM UP

Thus, we have studied the basic concepts of security and linux command which is used to modify security measures. The uses of sudo command and password policies are explained in chapter.

7.8 UNIT END QUESTIONS

1) What are the uses of root user in Linux system? Give the purpose of sudo command.

2) Write any five privileges of administrator.

7.9 LIST OF REFERENCES

1) Unix Concepts and Applications by Sumitabha Das.

2) Official Ubuntu Book, 8th Edition, by Matthew Helmke & Elizabeth K. Joseph with Jose Antonio Rey and Philips Ballew, Prentice Hall.

*****

8

NETWORKING

Unit Structure

8.0 OBJECTIVES

In this chapter you will learn about:

- Basic networking concepts and networking protocol.
- Transferring files through FTP, TELNET etc.
- Networking GUI.

8.1 INTRODUCTION

This chapter introduces network concept through OSI layers and protocols. The transferring files can be done by protocols like FTP, TELNET etc. The interface for users provided by network command.

8.2 BASIC INTRODUCTION TO NETWORKING

- All pieces of hardware and software programs speak different languages.
- For communication between two computers we need a special program in operating system that performs this function
- There are 7 layers of communication protocols which is known as OSI(Open system Interconnection) links.

Goals of Computer Network:

- **Sharing of Resources:** Resources are been shared. For e.g. one printer is shared by different nodes of computers.

8.3 NETWORK PROTOCOLS

- Protocols are set of rules used for communication.
- **Different types of Network Protocols are as follows:**
  - HTTP (Hypertext transfer protocol)
  - FTP (File transfer protocol)
  - TCP/IP
  - UDP
  - ICMP
  - Mail Protocols POP3 and SMTP
- **HTTP (Hypertext Transfer Protocol):**
  - HTTP has separate client and server components.
  - Client request the server for a document and server respond by sending it.
  - The protocol is also stateless in that each connection is unaware of the other.
  - Life cycle of connection using http is.
- The client contacts the source and opens a connection at port number 80.
- Client request the web server for some service. The request may consist of request header followed by data sent by the client.
- Server now sends a response which has response header, followed by data.
- Server waits for more requests and finally closes the connection.
- **FTP (File Transfer Protocol):**
  - FTP command is used to transfer files between hosts.
  - Like telnet, ftp can also be involved with or without the address
  - FTP is simple and convenient protocol.
  - FTP hostname by default will connect you to system, you must have a login id to be able to transfer the files.
  - ASCII and Binary files can be transferred.
  - **FTP establishes two types of connections:**

- **Data transfer:** It transfer data from one place to another.
- **Control transfer:** It transfers control to another system and also control activities of remote machine.

- **FTP command uses various options:**

Tag	Description
-p	**Use passive mode for data transfer**
-i	**Interactive prompting during multiple file transfer**
-e	**Disables command editing and history support**
-d	**Enabling debugging**

- **TCP/IP:**
  - **TCP (Transmission Control Protocol)** and **IP (Internet Protocol)** are two different procedures that are often linked together. When information is sent over the Internet, it is generally broken up into smaller pieces or "packets"
  - The use of packets facilitates speedy transmission since different parts of message can be sent by different routes and then reassembled at the destination.
  - TCP is the means for creating the packets, putting them back together in the correct order at the end, and checking to make sure that no packets got lost in transmission. If necessary, TCP will request that a packet be resent.
  - Internet Protocol(IP) is the method used to route information to the proper address. Every computer on the Internet has to have its own unique address known as the IP address. Every packet sent will contain an IP address showing where it is supposed to go.

- **UDP:**
  - This protocol is used together with IP when small amounts of information are involved.
  - It is simpler than TCP and lacks the flow-control and error-recovery functions of TCP. Thus it uses fewer system resources.

- **ICMP:**
  - A different type of protocol is Internet Control Message Protocol (ICMP).
  - It defines a small number of messages used for diagnostic and management purposes.
  - It is also used by ping and traceroute.

- **Mail Protocols POP3 and SMTP:**
  - Email has its set of protocols and there are a diversity of it, both for sending mail and for receiving mail.
  - The most common protocol for receiving mail is Post Office Protocol (POP) which is now in version 3 called POP3.
  - Both SMTP and POP3 use TCP for managing the transmission and delivery of mail across the Internet.
  - The most common protocol for sending mail is Simple Mail Transfer Protocol (SMTP).
  - For reading mail there is Interactive Mail Access Protocol (IMAP).
- **IP Address:**
  - Every host in the network has an address has an address called IP address, used by other machines to communicate with it.
  - It's a series of four dots delimited numbers.
  - The maximum value of each octect is 255.
  - It uses Internet Protocol for communication. For example: 211.162.0.1
  - TCP/IP application can address a host by its hostname as well as its IP address:

    telnet abc

    ftp 211.162.0.1
  - The network administrator makes the IP address unique in all connected networks.
- **DNS (Domain Name System)**
  - It is a distributed system which has three key concepts.
    - A hierarchical organization of hostnames
    - A distributed database for mapping
    - Authorities at individual level
  - Host belongs to domains which further belong to sub domain.
  - Root at the top signified by. (dot)
  - There are number of top level domains as shown in below fig 8.1

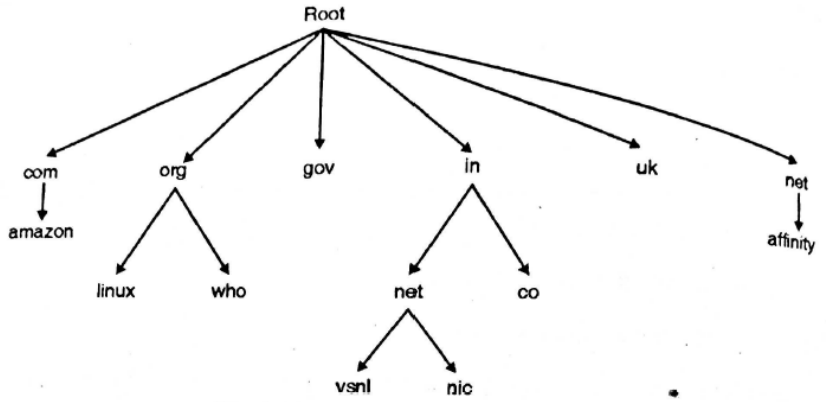


Fig 8.1 Internet Domain Hierarchy

o In the hierarchy related to these dot-separated strings, in is above net which is in turn above vsnl represents fully qualifies domain name.

o BIND (Berkeley Internet Name Domain) maintains the DNS-related software that runs under Linux.

o Below table shows various internet domains:

Domain Name	Significance
Int	International organization
Edu	Educational Institution
Gov	Government
Com	Commercial organization
Net	Networking organization
In	India
Biz	Business

- **Browsers:**

o Browsers are the most used applications as it's important to choose stable browser that suits all your needs.

o Browsers can be light weight, command line, free to cross platform and extremely extensible one.

o Best browsers in Linux are Firefox, chrome, opera, Pale Moon.

o It's a HTTP client which accepts a URL from URL window and gets the resource from the server.

8.4 TRANSFERRING FILES

- Files that can be transferred are ASCII (text) and binary.
- Executable, graphics, word processing are binary types.

- Uploading-put commands sends signal file i.e abc.gif to remote machine. We can copy multiple files with mput.mput behaves interactively and has confirmation for every file which has to be transferred.
- Downloading-To download use get and mget commands.

➢ **ssh (Secure shell):**

o ssh, or secure shell is a protocol used to securely log onto remote systems.

o It is the most common way to access remote Linux and Unix-Like servers.

o For Example command is:

ssh remote_host

o The remote_host is the IP address or domain name that we are trying to connect. This command assumes that your username on the remote system is the same as your username on your local system.

➢ **telnet:**

o telnet is terminal network.

o Popular client server application process for terminal services.

o telnet use in time sharing system.

o Responsible for establishing connection to the remote system.

o **It provides two types of login:**

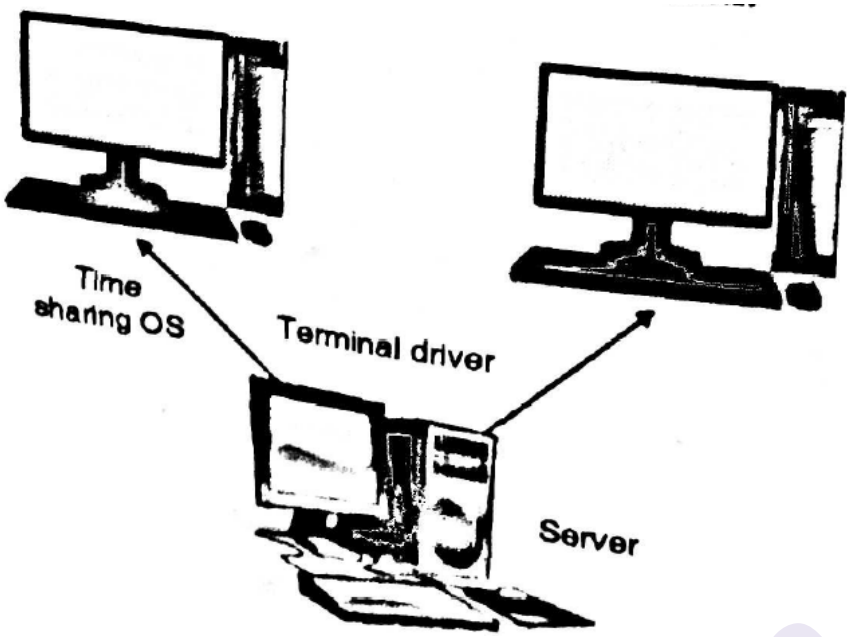
1) Local Login

2) Remote Login

1) Local Login:

a. Many terminal users are connected to one CPU.

b. CPU allocates time slot to each terminal.



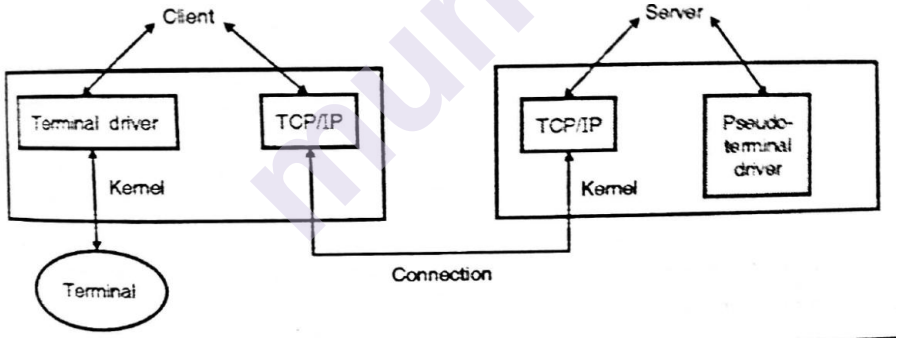
2) Remote Login:

a. Used as client server system

b. User logins from remote places so that remote terminal can access application program of another machine.

c. Server can communicate with one or more client.

d. Whenever client needs services it runs, request for the services and use it.



➢ **Ping (Packet Internet Groper):**

- Ping command is used for checking the network and also to test connectivity between two nodes.
- It uses ICMP(Internet Control Message Protocol) to communicate to other devices.
- Pinging a host does not require server process to run at other end.
- Ping command sends the ICMP ECHO_REQUEST packets to network host.

- PING (packet Internet Groper) command is the best way to check connectivity between two nodes in Local Area Network(LAN) or Wide Area Network(WAN).
- Host name or IP address can be used along with ping command. Ping uses the ICMP protocols mandatory.
- ECHO_REQUEST datagram to evoke an ICMP ECHO_RESPONSE from a host or gateway.

Example:

1) Ping the host to see if it's alive.
2) Increase ping Time Interval

   Wait for 5 seconds before sending the next packet.

   $ping –I 5 google.com
3) Send N packets and stop

   $ping –c 4 google.com
4) Timeout –w

Ping –w option specifies the deadline to terminate the ping output. This specifies the total number of seconds the ping command should send packets to the remote host.

The following example will ping for 5 seconds. i.e ping command will exit after 5 seconds irrespective of how many packets are sent or reciv

$ping –w 5 localhost

- **Traceroute:**
  - Prints the route that packets take to a network host.
  - Attempts to trace the route an IP packet would follow to some intenet host with time to live then listening for an ICMP "time exceeded" reply from a gateway.

traceroute command uses various options:

Tag	Description
-help	Display a help message and exit
-d	Enable socket level debugging
-f	Specifies with what TTL to start
-v	Print version information and exit
-e	Show ICMP extensions

$traceroute google.com

- **Route:**

  - Route command is used to show manipulate the IP routing table.
  - Primarily used to setup route to specific host or network through interface

route command uses various options:

Tag	Description
-A	Use the specified address family
-F	Operate on kernel's forwarding Information base routing table
-C	Operate on kernel routing cache
-n	Shows numerical addresses
-net	Target is network
-del	Delete route
-add	Add a new route

- **Hostname:**

  - Hostname command shows or sets system hostname
  - To display the system's DNS name
  - Hostname is usually set at system startup by reading the contents of file which contains hostname.
  - For example/etc/hostname

Hostname command uses various options:

Tag	Options
-a	Displays the alias name of the host
-b	Always set a hostname
-d	Displays the name of the DNS domain
-F	Read the hostname from the specified file
-i	Display the network address of the host name
-f	Display the FQDN (Fully Qualified Domain Name)
-s	Display the short host name
-h	Print a help message and exit

8.5 NETWORKING GUI

- The network manager service dynamically detects and configures network connections.

- Network manager does not have its own graphical user interface.
- In Ubuntu the graphical configuration tool to configure network interfaces is called network-admin
- System should include the network-manager-gnome package that is able to run Network Manager's GUI connection editor.

8.6 LET US SUM UP

Thus, we have studied the basic concepts of network and linux command which is used to perform communication among user over internet. The uses of network protocol and commands are explained in chapter.

8.7 UNIT END QUESTIONS

1) Write a note on FTP.
2) What is the purpose of commands:- ssh, ping, hostname, telnet, route. Give suitable example.
3) Define network protocol. Explain in detail HTTP.

8.8 LIST OF REFERENCES

1) Unix Concepts and Applications by Sumitabha Das.
2) Official Ubuntu Book, 8th Edition, by Matthew Helmke & Elizabeth K. Joseph with Jose Antonio Rey and Philips Ballew, Prentice Hall

*****

9

BASIC SHELL SCRIPTING

Unit Structure

9.0 OBJECTIVES

In this chapter you will learn about:

- Basic of shell scripting through environment variables, conditional, looping statements and commands.

9.1 INTRODUCTION

This chapter introduces shell scripting through variables, commands, conditional and looping statements. The sed and awk commands are supporting for file manipulation for analyzing data.

9.2 FEATURES AND CAPABILITIES

- A shell script is defined as it plain text file with a set of Linux command, flow of control and Input Outptut facilities. And it is created by using any text editor like vi, emac etc.

- Shell script allows use of variable, file and directory management features and it interpreted directly.

- Shell script provides many features like loop, construct, array, functions, logic with other utilities etc.
- Shell script allows reading input and parsing the command line.
- Shell supports advanced features such as Functions and Arrays, regular expressions.
- Easy to use and understand.
- It is much quicker than programming in any other languages.

9.3 SYNTAX

Following is syntax basic structure of shell script:

#!/bin/bash (Shebang)

#(comments)

chmod +x scriptfilename (make script executable)

echo " " (to print message of variables contains)

./scriptfilename.sh (execute script)

Where,

#!/bin/bash:-It define which shell will be used to run the shell script.

#comments:-By using '#' symbol you can pass the comments.

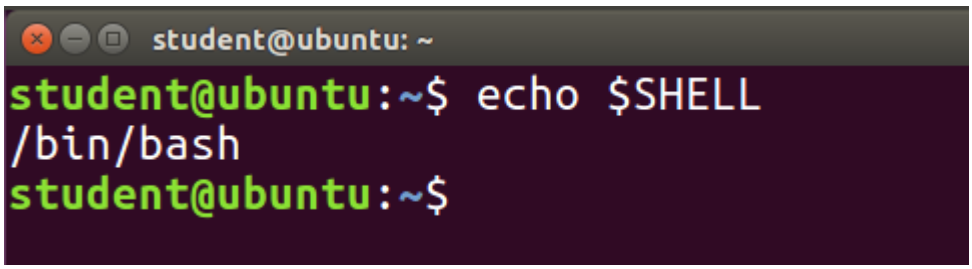
chmod +x scriptfilename.sh:-It define file is executable and tell to Linux that file is executable.

./scriptfilename.sh:-It define execute the script

Examples,

1) To check your current shell use following command as

echo $SHELL



Here dollar sign ($) stands for a shell variable

2) #!/bin/bash

#we learn shell scripting

echo “This is my first program by using shell script”

```
student@ubuntu: ~
student@ubuntu:~$ #!/bin/bash
student@ubuntu:~$ #we learn shell scripting
student@ubuntu:~$ echo "This is my first program by using shell script"
This is my first program by using shell script
student@ubuntu:~$
```

Output:

```
This is my first program by using shell script
student@ubuntu:~$
```

Constructs:

Construct shell script you can use text editor such as vi or cat command.

vi command:

First type following vi command and rest of text as:

```
student@ubuntu: ~
student@ubuntu:~$ vi demofile.sh
```

```
student@ubuntu: ~
#this is my first program
echo "we learn basic shell scripting"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

```
student@ubuntu: ~
student@ubuntu:~$ vi demofile.sh
student@ubuntu:~$ ls
demofile.sh  Documents  examples.desktop  Pictures  Templates
Desktop      Downloads  Music             Public    Videos
```

```
student@ubuntu:~$ cat demofile.sh
#this is my first program
echo "we learn basic shell scripting"

student@ubuntu:~$ chmod +x demofile.sh
```

Output:

We learn basic shell scripting, following are screen snaps for creating first shell script program.

```
student@ubuntu:~$ ls -l demofile.sh
-rwxrwxr-x 1 student student 65 Nov 19 19:52 demofile.sh
student@ubuntu:~$ ./demofile.sh
we learn basic shell scripting
student@ubuntu:~$
```

cat command:

First type following cat command and rest of text as:

```
student@ubuntu: ~
student@ubuntu:~$ cat>demofile2.sh
#print your name using input from keyboard
echo "enter your name"
read name
echo "your name is:"$name
student@ubuntu:~$
```

```
student@ubuntu:~$ ls
demofile2.sh  Desktop    Downloads         Music     Public     Videos
demofile.sh   Documents  examples.desktop  Pictures  Templates
student@ubuntu:~$ ls -l demofile2.sh
-rw-rw-r-- 1 student student 102 Nov 19 19:56 demofile2.sh
```

When you finished your writing script press CTRL+D to save and then by using chmod command give executable permission to file.

```
student@ubuntu:~$ chmod +x demofile2.sh
student@ubuntu:~$ ls -l demofile2.sh
-rwxrwxr-x 1 student student 102 Nov 19 19:56 demofile2.sh
student@ubuntu:~$
```

For execute file give command as ./demofile2.sh

```
student@ubuntu:~$ ./demofile2.sh
enter your name
Rich
your name is:Rich
student@ubuntu:~$
```

Sometimes arguments are specified with shell procedure ten they are assigned to special variable or propositional parameters.

1) $1,$2,$3 The positional parameters.

2) $* The complete set of positional parameters as single string.

3) $# The number of arguments specified in command line.

4) $0 Name of executed command.

5) $? Exit status of the last command

6) $! PID of last command

Example:

```
student@ubuntu: ~
student@ubuntu:~$ cat>scriptfile.sh
#Example to print standard input from user to standard output using parameters
echo "Program:$0 The number of arguments are:$# The arguments are:$*"
echo "Your name is:$1\n Your name is:$2"
student@ubuntu:~$
```

When you finished your script press CTRL+D to save and By using chmod command give executable permission to file.

```
student@ubuntu:~$ ls -l scriptfile.sh
-rw-rw-r-- 1 student student 190 Nov 19 20:07 scriptfile.sh
student@ubuntu:~$ chmod +x scriptfile.sh
student@ubuntu:~$ ls -l scriptfile.sh
-rwxrwxr-x 1 student student 190 Nov 19 20:07 scriptfile.sh
student@ubuntu:~$
```

Here we pass optional parameters as ashwini and 18889 as shown above.

```
student@ubuntu:~$ ./scriptfile.sh Rich 19980
Program:./scriptfile.sh The number of arguments are:2 The arguments are:Rich 19980
Your name is:Rich\n Your name is:19980
student@ubuntu:~$
```

9.4 MODIFYING FILES

Consider a file studentinfo.txt which is already created and contains information of student like Roll.No, Name, Class and PH.No. etc if you want to add some text in studentinfor.txt file then it is modify by using vi editor following key for modify the file.

Name of Key	Use of Key
h	It is used to move left one character
l	It is used to move right one character
k	It is used to move up one character
J	It is used to move down one character

Also for modifying file you sed command which modify each line of line and replace specified parts of the line.

9.5 SED

"sed" means stream editor. Sed command allows:

1) Performing basic text transformations on an input stream.
2) To modify each line of a file
3) To replace specified parts of the line.

```
student@ubuntu:~$ vi file.txt
```

Example

Consider a file file.txt

```
student@ubuntu:~$ cat file.txt
ROLLNO  NAME            ADDRESS         TOTALFEE
1       ANJU            MUMBAI          5000
2       RAJ             MUMBAI          7000
3       SAMEER          PUNE            9000
4       BABU            VASHI           4000
5       SUMAN           VASHI           3000
6       SEJAL           KHARGHAR        8000
7       MANISH          DADAR           3000
8       SARTH           BORIVALI        11000

student@ubuntu:~$
```

If the file name "file.txt" and you want to change all occurrences of VASHI to PAREL to the modified file to "file.txt then use the following command.

Command:

```
student@ubuntu:~$ sed 's/VASHI/PAREL/' file.txt >file1.txt
student@ubuntu:~$ ls
demofile2.sh  Desktop    Downloads         file1.txt  Music     Public
demofile.sh   Documents  examples.desktop  file.txt   Pictures  scriptfile.sh
```

Output:

```
student@ubuntu:~$ cat file1.txt
ROLLNO  NAME            ADDRESS         TOTALFEE
1       ANJU            MUMBAI          5000
2       RAJ             MUMBAI          7000
3       SAMEER          PUNE            9000
4       BABU            PAREL           4000
5       SUMAN           PAREL           3000
6       SEJAL           KHARGHAR        8000
7       MANISH          DADAR           3000
8       SARTH           BORIVALI        11000

student@ubuntu:~$
```

sed is also frequently used to filter lines in a file or stream.

Example:

If you only want see the lines containing "SA" you could use:

```
student@ubuntu:~$ sed -n '/SA/p' file.txt > file3.txt
```

Output:

```
student@ubuntu:~$ cat file3.txt
3       SAMEER          PUNE            9000
8       SARTH           BORIVALI        11000
student@ubuntu:~$
```

Replacing or substituting string:

sed command is mostly used to replace the text in a file. The below simple sed command replaces the word “UNIX” with “LINUX” in the file.

```
student@ubuntu: ~
student@ubuntu:~$ cat>file.txt
linux is great OS.
unix is opensource.
unix is free OS.
learn Operating system.
```

Output:

```
student@ubuntu:~$ sed 's/unix/linux/' file.txt
linux is great OS.
linux is opensource.
linux is free OS.
learn Operating system.
student@ubuntu:~$
```

Here the “s” specifies the substitution operation. The “/” are delimiters. The “Unix” is the search pattern and the “Linux” is the replacement string.

9.6 AWK COMMAND

This command is used for processing or analyzing text files, in particular data files that are organized by lines (rows) and columns.

Syntax:

awk ‘pattern {action}’ input-filename > output-filename

This command is worked as taking each line of the input file. And if the line contains the pattern then apply the action to the line and write the resulting line to the output-file.

If the pattern is omitted, the action is applied to all line.

Examples:

```
student@ubuntu: ~
student@ubuntu:~$ cat file.txt
ROLLNO   NAME            ADDRESS         TOTALFEE
1        ANJU            MUMBAI          5000
2        RAJ             MUMBAI          7000
3        SAMEER          PUNE            9000
4        BABU            PAREL           4000
5        SUMAN           PAREL           3000
6        SEJAL           KHARGHAR        8000
7        MANISH          DADAR           3000
8        SARTH           BORIVALI        11000

student@ubuntu:~$
```

1) $awk '{print $1}' file.txt > outputfie.txt:

```
student@ubuntu:~$ awk '{print $1}' file.txt > outputfile.txt
```

```
student@ubuntu:~$ cat outputfile.txt
ROLLNO
1
2
3
4
5
6
7
8

student@ubuntu:~$
```


This statement takes the element of the 1st column of each line and writes it as a line in the output file "outputfile.txt". As shown above

2) $awk '{print $2,$3}" file.txt > opfile.txt:

```
student@ubuntu: ~
student@ubuntu:~$ awk '{print $2,$3}'  file.txt > opfile.txt
```

```
student@ubuntu:~$ cat opfile.txt
NAME ADDRESS
ANJU MUMBAI
RAJ MUMBAI
SAMEER PUNE
BABU PAREL
SUMAN PAREL
SEJAL KHARGHAR
MANISH DADAR
SARTH BORIVALI

student@ubuntu:~$
```

In this command we pass the second, third column, with $2,$3. By default columns are assumed to be separated by spaces or tabs, comma.

You can use regular expression as condition.

3) $awk '/15/{print $2}' file.txt:

Here regular expression is string between the two slashes ('/'). In this case string "15". It indicates that if a line contains the string "15", the system prints out the element at the 2nd column of that line.

4) If the table elements are numbers awk can run calculations on them as in this example:

```
student@ubuntu:~$ cat > awkfile.txt
```

$cat awkfile.txt:

```
student@ubuntu:~$ cat  awkfile.txt
1 2 3 4
2 2 3 4
3 2 3 4
4 2 3 4
5 2 3 4
student@ubuntu:~$
```

$awk '{print ($1 * $2) + $4}' awkfile.txt:

```
student@ubuntu:~$ awk '{print ($1 * $2) + $4}' awkfile.txt
6
8
10
12
14
student@ubuntu:~$
```


9.7 FILE MANIPULATION UTILITIES

1. tac:

This command is used for file manipulation tac command is used for print file in reverse (liat line first). That is this command prints the file in reverse order with the last line first.

```
student@ubuntu: ~
student@ubuntu:~$ cat file.txt
ROLLNO   NAME            ADDRESS         TOTALFEE
1        ANJU            MUMBAI          5000
2        RAJ             MUMBAI          7000
3        SAMEER          PUNE            9000
4        BABU            PAREL           4000
5        SUMAN           PAREL           3000
6        SEJAL           KHARGHAR        8000
7        MANISH          DADAR           3000
8        SARTH           BORIVALI        11000

student@ubuntu:~$
```

Syntax: tac filename.txt

Example: consider a file file.txt

Command:

```
student@ubuntu:~$ tac file.txt

8        SARTH           BORIVALI        11000
7        MANISH          DADAR           3000
6        SEJAL           KHARGHAR        8000
5        SUMAN           PAREL           3000
4        BABU            PAREL           4000
3        SAMEER          PUNE            9000
2        RAJ             MUMBAI          7000
1        ANJU            MUMBAI          5000
ROLLNO   NAME            ADDRESS         TOTALFEE
student@ubuntu:~$
```


By using tac command file.txt display print in reverse.

2. rev:

This command is used for reverse the characters in every line. Difference between tac command and rev command is-

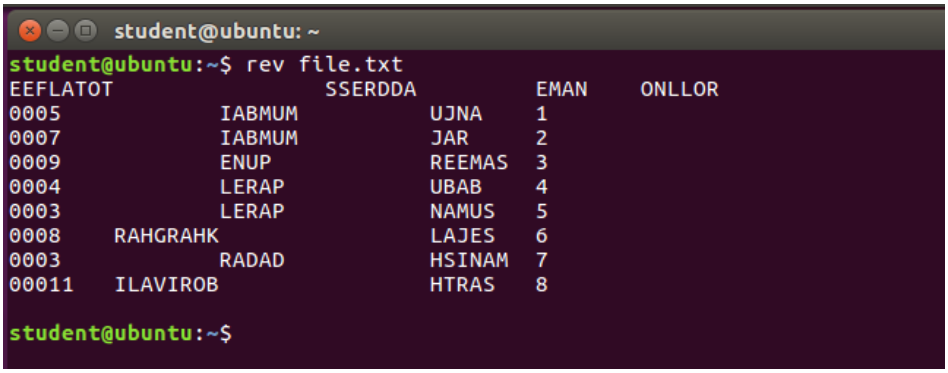
rev command reverse each character of the line

tac command reverse each line of the file.

Syntax: rev filename.txt

Example: Consider a file file.txt

Command:



By using rev command file.txt display reverse the characters in every line.

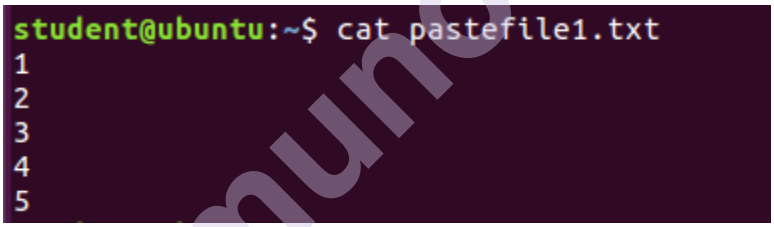
3. paste:

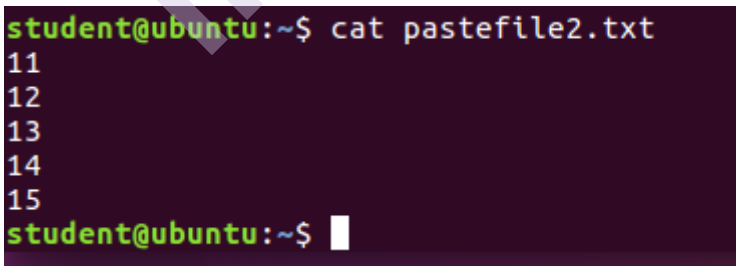
This command is used for merge file lines that means. This command paste the line1 of file1, line1 of file2,.. line1 of fileN. It will repeat the same for all lines. Each file's line is separated by tab.

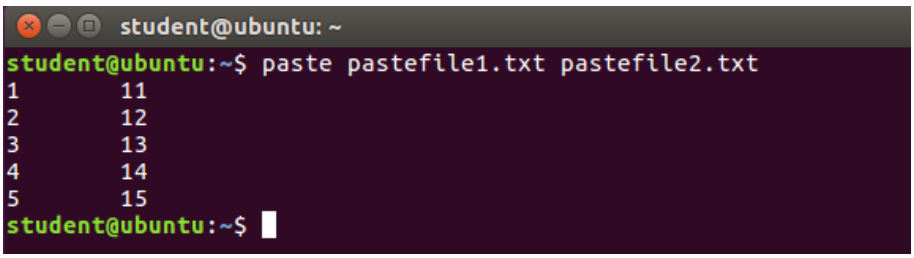
Syntax: $paste filename1.txt filename2.txt filename3.txt

Example 1:

Consider a two file pastefile1.txt and pastefile2.txt are shown combined by using cat commands:







Example 2:

Let us consider a file with the sample contents as below:

```
student@ubuntu:~$ cat file1
linux
unix
solaris
redhat
fedora
student@ubuntu:~$
```

i) Join all lines in a file:

```
student@ubuntu:~$ paste -s file1
linux	unix	solaris	redhat	fedora
student@ubuntu:~$
```

-s option of paste joins all the lines in a file. Since no delimiter is specified. Default delimiter tab is used to separate the columns.

ii) Merge a file by pasting the data into 2 columns using colon separator:

```
student@ubuntu:~$ paste -d':' - - < file1
linux:unix
solaris:redhat
fedora:
student@ubuntu:~$
```

iii) Join all lines using the comma delimiter:

```
student@ubuntu:~$ paste -d, -s file1
linux,unix,solaris,redhat,fedora
student@ubuntu:~$
```

4. join:

This command is used for join lines of two files which is based on a common field, this can specify by using field.

Syntax: $join –t':' -1 N -2 N file1 file2

Where,

-t':' – it indicates that field separator

-1 N – it indicate that Nth field in 1st file

-2 N – it indicate that Nth field in 2nd file

file1 file2 – it indicate that files name that should be joined

Example: Consider two file joinfile.txt and joinfile1.txt

```
student@ubuntu: ~
student@ubuntu:~$ cat joinfile1.txt
name    salary
asha    30k
rahul   40k
raj     50k
student@ubuntu:~$ cat joinfile2.txt
name    salary
anju    40k
vinay   80k
neha    20k
student@ubuntu:~$
```

Now by using join command you join this two file joinfile.txt joinfile1.txt

Command:

```
student@ubuntu:~$ join -1 2 -2 1 joinfile1.txt joinfile2.txt
```

Output:

```
join: joinfile1.txt:2: is not sorted: asha      30k
join: joinfile2.txt:4: is not sorted: neha      20k
student@ubuntu:~$
```

9.8 DEALING WITH LARGE FILES AND TEXT

To view and manipulate large log files use the following command

1. To Display Specific Lines of a file use sed command:

Here to view specific lines you have to use line numbers.

Syntax: $sed –n –e Xp –e Yp FILENAME

Where,

Sed	print all the lines by default
-n	output
-e	Command to be executed
Xp	Print line number X
Yp	Print line number Y
FILENAME	name of the file

Example:

1. $sed –n –e 50p –e 100p –e 100p /var/log/file:

i) Here print the lines 50,100,1000 from the file.

ii) You can view the content of var/log/file from line number 100 to 300:

Syntax: sed –n M, Np FILENAME

Where,

M	Starting line number
N	Ending line number

$sed –n 100,300p /var/log/file

2. To display First N Lines of a file use head Command:

To displays only first 20 lines of /var/log/file

Syntax: head –n N FILENAME

$head –n 20 /var/log/file

3. Ignore last N lines of file use head command:

To shows how to ignore the last N lines, and display only the remaining lines from the top of file.

To display all the lines of the /var/log/file except the last 50 lines.

Syntax: head –n N FILENAME

$head –n -50 /var/log/file

4. Display last N lines of the file use tail command

To displays only last 30 lines of /var/log/file.

Syntax: tail –n N FILENAME

$tail –n 30 /var/log/messages

9.9 STRING MANIPULATION

We know that when you use $(dollar sign) followed by variable name it indicates that variable with its values which is known as parameter expansion.

1. String:

This command is used to get the length of the given variable in your shell script.

Syntax: $(#string)

Example:

```
student@ubuntu: ~
student@ubuntu:~$ cat > length.sh
echo "this command is used to get length of given variable"
var="we learn linux"
echo ${#var}
```

```
student@ubuntu:~$ chmod +x length.sh
```

```
student@ubuntu:~$ ./length.sh
this command is used to get length of given variable
14
student@ubuntu:~$
```

Output:

```
this command is used to get length of given variable
14
```

2. Position:

This command is used to extract a substring from a string. Character substring from $sting starting from $position

Syntax: $(string:position)

$(string:position:length)

Example:

```
student@ubuntu: ~
student@ubuntu:~$ cat > substring.sh
echo "this command is used to extract substring from string"
var="we learn shell scripting"
echo ${var:6}
echo ${var:16:6}k
```

```
student@ubuntu:~$ chmod +x substring.sh
```

Output:

```
student@ubuntu:~$ ./substring.sh
this command is used to extract substring from string
rn shell scripting
criptik
student@ubuntu:~$
```

Where, First variable returns the substring from 6th position.

Second variable returns the 6 characters starting from 16th position.

3. Substring for shortest match:

To match shortest substring use following command:

Syntax:

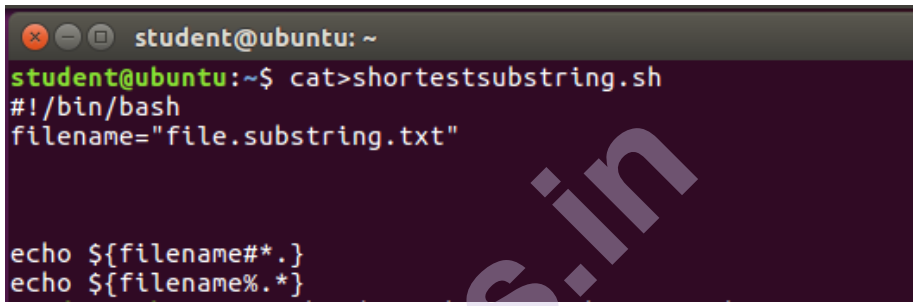
1) ${string#substring}:

- deletes the shortest match of $substring from **front** of $string

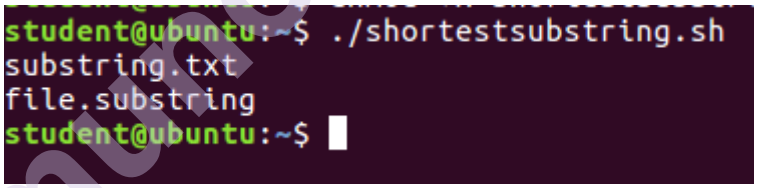
2) ${string%substring}:

% - deletes the shortest match of $substring from **back** of $string

Example:



Output:



Here first echo statement substring is '*' matches the substring starts with dot and # strips from the front of the string, so it strips the substring is "substrings"

Second echo statement substring '*' matches the substring starts with dot, and % strips from back of the string, so it deletes the substring '.txt'

4. Substring for longest match:

To match longest $substring use following command

Syntax:

1) ${string##substring}

- deletes the longest match of $substring from front of $string.

2) ${string%%substring}

%% - deletes the longest match of $substring from back of $string

Example:

```
student@ubuntu: ~
student@ubuntu:~$ cat > longestsubstring.sh
#!/bin/bash
filename=" file.substrings.txt "
echo "After deletion of longest match from FRONT:" ${filename##*.}
echo "After deletion of longest match from BACK:" ${filename%%.*}
student@ubuntu:~$
```

```
student@ubuntu:~$ chmod +x longestsubstring.sh
```

Output:

```
student@ubuntu:~$ ./longestsubstring.sh
After deletion of longest match from FRONT: txt
After deletion of longest match from BACK: file
student@ubuntu:~$
```

Here first echo statement ##* match for '*' longest match which matches 'file.substrings' so after striping it returns remaining txt.

Second echo statement %%* match for '*' longest match which matches 'subtrings.txt' so after striping it returns 'file'.

9.10 BOOLEAN EXPRESSIONS

Following are Boolean operators used in shell script:

Name of operator	operator	Use
logical negation	! operator	Display result as a true condition into false and vice versa
logical OR	-o operator	Display if one of the operands is true then condition should be true
logical AND	-a operator	Display if both the operands are true then condition should be true otherwise it should be false.

Example:

1) [!false] return it is true

2) [$a –lt 20 –o $b –gt 100] return it is true

3) [$a –lt 20 –a $b –gt 100] return it is false

Whete 'lt' means less than and 'gt' means greater than.

```
student@ubuntu: ~                                          8:02 AM
student@ubuntu:~$ cat>boolean.sh
a=10
b=20
if [ $a != $b ]
then
echo "$a!=$b:a is not equal to b"
else
echo "$a!=$b:a is equal to b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
echo "$a -lt 100 -a $b -gt 15:returns true"
else
echo "$a -lt 100 -a $b -gt 15:returns false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
echo "$a -lt 100 -o $b -gt 100:returns true"
else
echo "$a -lt 100 -o $b -gt 100:returns false"
fi
student@ubuntu:~$
```

Ubuntu Software

Output:

```
student@ubuntu:~$ chmod +x boolean.sh
student@ubuntu:~$ ./boolean.sh
10!=20:a is not equal to b
10 -lt 100 -a 20 -gt 15:returns true
10 -lt 100 -o 20 -gt 100:returns true
student@ubuntu:~$
```

9.11 FILE TESTS

When you are using files in shell script, to do some file tests on your file before using it.

File test allow:

1) Checking for existence of your file.
2) Your file is readable, writable or executable.
3) Type of the file.

File test is done by if clause.

Syntax:

if [-option filename to be test]
then,
---
else
---
fi

Following are some file test operators:

File test operator name	Use
a	It returns true if the file exists
c	It returns true if the file exists and is a character special file
d	It returns true if the file exists and is a directory
e	It returns true if the file exists
f	It returns true if the file exists and is a regular file
p	It returns true if the file exists and is a regular file
r	It returns true if the file exists and is readable
s	It returns true is the file exists and has a size greater than zero
t	It returns true if file descriptor is open and refers to a terminal
w	It returns true if the file exists and is writable
x	It returns true if the file exists and is executable

Example:

1. Shell script checks for existence of a regular file.

```
student@ubuntu:~$ cat>filedemo
FILE="boolean.sh"
if [ -f $FILE ]
then
echo "$FILE exists and is a regular file"
else
echo "Either $FILE does not exist or is not a regular file"
fi
```

Output:

```
student@ubuntu:~$ chmod +x filedemo
student@ubuntu:~$ ./filedemo
boolean.sh exists and is a regular file
student@ubuntu:~$ ls
boolean.sh  Desktop    examples.desktop  Music      Templates
calc.py     Documents  filedemo          Pictures   Videos
cprogram    Downloads  hello.py          Public
student@ubuntu:~$
```

2. All the file test operators:

Consider a variable name is 'file' which holds an existing file name as "/var/www/test/linux/test.sh" whose size is 100 bytes and has read, write and execute permission off-

```
student@ubuntu:~$ gedit test.sh
```

```
student@ubuntu:~$ cat test.sh
file="/home/student/boolean.sh"
if [ -r $file ]
then
each "File has read access"
else
echo "File does not have read access"
fi

if [ -w $file ]
then
echo "File has write permission"
else
echo "File does not have write permission"
fi

if [ -x $file ]
then "file has execute permission"
else
echo "file does not have execute permission"
fi

if [ -f $file ]
then
echo "file is an ordinary file"
else
echo "This is special file"
fi

if [ -d $file ]
then
echo "File is a directory"
else
echo "This is not a directory"
fi

if [ -s $file ]
then
echo "file size is zero"
else
echo "file size is not zero"
fi

if [ -e $file ]
then
echo "file exists"
else
echo "file does not exist"
fi
```


Output:

```
student@ubuntu:~$ gedit test.sh
student@ubuntu:~$ chmod +x test.sh
student@ubuntu:~$ ./test.sh
./test.sh: line 4: each: command not found
File has write permission
./test.sh: line 17: file has execute permission: comman
d not found
file is an ordinary file
This is not a directory
file size is zero
file exists
student@ubuntu:~$
```

Here read, write and execute permission off-hence output is obtained with every else statements.

9.12 CASE

Case statement is similar to switch statement used in C. By using case statement user can test simple values for integers and characters and testing can be done by string pattern that can contains wild card char actors (special characters)

Syntax:

Case expression in

pattern1)

Statement to be executed if pattern1 matches

;;

Pattern2)

Statement to be executed if pattern2 matches

;;

esac

case required at least one pattern

Finally case statements expand the expression and try to it against each pattern. Here expression is compared against every pattern until match is found and then the statements following the pattern matching is executed.

When statement part is executed until;; (double semicolon) which indicates that program flow should jump to the end of the entire case statement. If there is no match, exit status of case is zero.

Example:

Once finished save it and exit.

For run above shell script use chmod com

mand as shown in below.

Command:

```
student@ubuntu:~$ gedit casefile.sh
student@ubuntu:~$ cat casefile.sh
echo "Enter the string"
read s
case $s in
[aeiou]*)echo "The string begins with small case letter
"
;;
[AEIOU]*)echo "The string begins with capital case lett
er" ;;
[0-9]*) echo "The string begins with a digit"
;;
*[0-9]) echo "The string ends with a digit"
;;
?????)echo "You entered a five letter word"
;;
esac;

student@ubuntu:~$
```

Output:

```
student@ubuntu:~$ ./casefile.sh
Enter the string
2states
The string begins with a digit
student@ubuntu:~$
```


9.13 DEBUGGING

We know that with the x-option, run the entire script in debug mode.

Each command with its arguments is printed to standard output after the commands have been expanded but before they are executed.

-x option is used to debug a shell script

Run the shell script using –x option

For e.g. $ -x scriptname.sh

Following are debugging options used for turn on or off with set command.

set –x: it display command and their arguments as they are executed

set –v: it display shell input lines as they are read.

9.14 REGULAR EXPRESSIONS

Regular expressions are special characters. It is used for search data, matching patterns grep command is used to search for a specific string in a file.

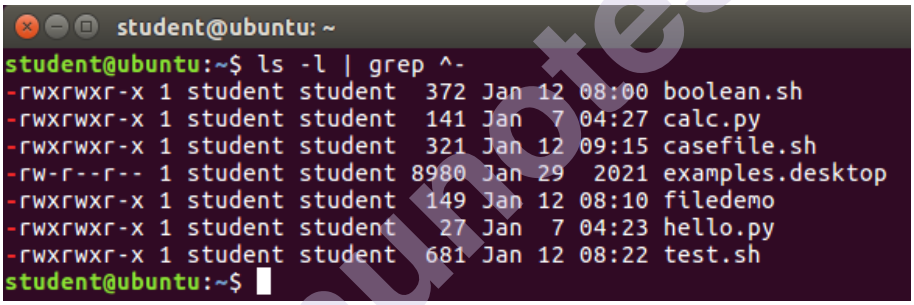
Following are some basic regular expressions symbols:

1) It is used to replaces any character
2) ^ It is used for matches start of string
3) $ It is used for matches end of string
4) * It is used for matches up zero or more times the preceding character.
5) \ It is used for Represent special characters
6) () It is used for Groups regular expressions.
7) ? It is used for Matches up exactly one character
8) \+ It is used for Matches one or more occurrence of the previous character
9) \? It is used for Marches zero or one occurrence of the previous character.

Examples:

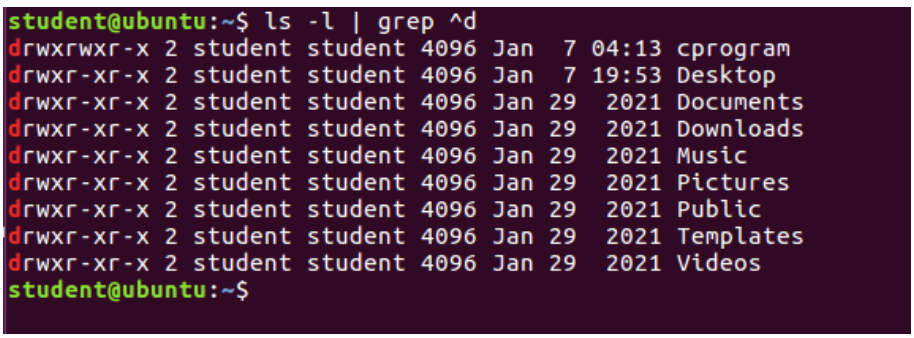
1. Search for matches start of-

$ls –l | grep ^ -



2. Search for content that STARTS with ‘d’

$ls –l | grp ^ d



3. Search for content containing letter ‘r’

$cat test.sh | grep r

```
student@ubuntu: ~
student@ubuntu:~$ cat test.sh | grep r
if [ -r $file ]
each "File has read access"
echo "File does not have read access"
echo "File has write permission"
echo "File does not have write permission"
then "file has execute permission"
echo "file does not have execute permission"
echo "file is an ordinary file"
echo "File is a directory"
echo "This is not a directory"
echo "file size is zero"
echo "file size is not zero"
student@ubuntu:~$
```

4. Search for content that STARTS with ‘e’

$cat test.sh | grep ^e

```
student@ubuntu:~$ cat test.sh | grep ^e
each "File has read access"
else
echo "File does not have read access"
echo "File has write permission"
else
echo "File does not have write permission"
else
echo "file does not have execute permission"
echo "file is an ordinary file"
else
echo "This is special file"
echo "File is a directory"
else
echo "This is not a directory"
echo "file size is zero"
else
echo "file size is not zero"
echo "file exists"
else
echo "file does not exist"
student@ubuntu:~$
```


5. Select only those line that end with ‘e’

$cat test.sh | grep e$

```
student@ubuntu: ~
student@ubuntu:~$ cat test.sh | grep e$
else
else
else
else
else
else
else
student@ubuntu:~$
```

6. Filter out all lines that contain character ‘o’

$cat test.sh | grep o

```
student@ubuntu: ~
student@ubuntu:~$ cat test.sh | grep o
file="/home/student/boolean.sh"
echo "File does not have read access"
echo "File has write permission"
echo "File does not have write permission"
then "file has execute permission"
echo "file does not have execute permission"
echo "file is an ordinary file"
echo "This is special file"
echo "File is a directory"
echo "This is not a directory"
echo "file size is zero"
echo "file size is not zero"
echo "file exists"
echo "file does not exist"
student@ubuntu:~$
```


7. Searching for all characters ‘v’

$cat test.sh | grep v

```
student@ubuntu: ~
student@ubuntu:~$ cat test.sh | grep v
echo "File does not have read access"
echo "File does not have write permission"
echo "file does not have execute permission"
student@ubuntu:~$
```

8. Filter out lines where character ‘a’ proceeds characters ‘r’

$cat test.sh | grep “a\+r”

```
student@ubuntu:~$ cat test.sh | grep "a\+r"
echo "file is an ordinary file"
student@ubuntu:~$
```

9.15 LET US SUM UP

Thus, we have studied the basic concepts shell scripting. Test cases, conditional, looping statements have been described briefly. The debugging also explained.

9.16 UNIT END QUESTIONS

1) Give the purpose of HOME, PS2, PS1, SHELL, USER shell variables.

2) Define regular expression. What is the purpose of the following regular expression characters:- **^ , $, * , ?**

3) Write a shell script to read a month number from the user and display corresponding month name

4) List any three features of awk. Give its general syntax. Explain it with two different examples

5) Write a shell script to accept 2 numbers from user and one operator. Based on the operator entered perform addition, subtraction, multiplication and division.

9.17 LIST OF REFERENCES

1) Unix Concepts and Applications by Sumitabha Das.

2) Official Ubuntu Book, 8th Edition, by Matthew Helmke & Elizabeth K. Joseph with Jose Antonio Rey and Philips Ballew, Prentice Hall.

*****