SYLLABUS

Linux Administration

Unit-1

Overview of Linux : What is Linux, Linux's root in Unix, Common Linux Features, advantage of Linux, Overview of Unix and Linux architectures, Linux files system, hardware requirements for Linux, Linux standard directories. Commands for files and directories cd, ls, cp, rm, mkdir, rmdir, pwd, file, more, less, Creating and viewing files using cat, file comparisons.

Unit 2

Essential Linux commands: Processes in Linux, process fundamentals, connecting processes with pipes, redirecting input/output, Background processing, managing multiple processes, process scheduling – (at, batch), nohup command, kill, ps, who, find, sort, touch, file, file processing commands - wc, cut, paste etc Mathematical commands - expr, factor etc. Creating and editing files with vi editor.

Unit 3

Shell programming - Basics of shell programming, various types of shell available in Linux, comparisons between various shells, shell programming in bash. Conditional and looping statements, case statement, parameter passing and arguments, Shell variables, system shell variables, shell keywords, Creating Shell programs for automating system tasks.

Unit-4

System administration - Common administrative tasks, identifying administrative files configuration and log files, Role of system administrator, Managing user accounts-adding & deleting users, changing permissions and ownerships, Creating and managing groups, modifying group attributes, Temporary disabling of users accounts, creating and mounting file system, checking and monitoring system performance - file security & Permissions, becoming super user using su. Getting system information with uname, host name, disk partitions & sizes, users, kernel, installing and removing packages with rpm command.

Unit-5:

Simple filter commands: pr, head, tail, cut, sort, uniq, tr - Filter using regular expression grep, egrep, sed **Understanding various Servers :**DHCP, DNS, Squid, Apache, Telnet, FTP,Samba.

UNIT 1

OVERVIEW OF LINUX

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner. So, to work on your computer, you need an Operating System (OS).

Linux is an operating system or a kernel which germinated as an idea in the mind of young and bright **Linus Torvalds** when he was a computer science student. He used to work on the UNIX OS (proprietary software) and thought that it needed improvements.

However, when his suggestions were rejected by the designers of UNIX, he thought of launching an OS which will be receptive to changes, modifications suggested by its users. So Linus devised a Kernel named Linux in 1991. Though he would need programs like File Manager, Document Editors, Audio -Video programs to run on it. So around 1991, a working Linux operating system with some applications was officially launched, and this was the start of one of the most loved and open-source OS options available today.

Linux is a **free open source operating system** (OS) based on UNIX that was created in 1991 by **Linus Torvalds**. Users can modify and create variations of the source code, known as distributions, for computers and other devices. The most common use is as a server, but

Linux is also used in desktop computers, smartphones, e-book readers and gaming consoles, etc.

The benefits of using Linux

Linux now enjoys popularity at its prime, and famous among programmers as well as regular computer users around the world. Its main benefits are -

- It offers a free operating system. You do not have to spend hundreds of dollars to get the OS like Windows.
- Being open-source, anyone with programming knowledge can modify it.
- The Linux operating systems now offer millions of programs/ applications to choose from, most of them free.
- Once you have Linux installed you no longer need an antivirus. Linux is a highly secure system. More so, there is a global development community constantly looking at ways to enhance its security. With each upgrade, the OS becomes more secure and robust.
- Linux is the OS of choice for Server environments due to its stability and reliability (Mega-companies like Amazon, Facebook, and Google use Linux for their Servers). A Linux based server could run non-stop without a reboot for years on end.

Linux's root in Unix

UNIX is called the mother of operating systems which laid out the foundation to Linux. Unix is designed mainly for mainframes and is in enterprises and universities. While Linux is fast becoming a household name for computer users, developers, and server environment. You may have to pay for a Unix kernel while in Linux it is free.

But, the commands used on both the operating systems are usually the same. There is not much difference between UNIX and Linux. Though they might seem different, at the core, they are essentially the

same. Since Linux is a clone of UNIX. So learning one is same as learning another.

Common Linux Features

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

- **Portable** – Portability means software can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** – Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.
- **Graphical User Interface-** Linux is command line based OS but it can be converted to GUI based by installing packages.
- **Support's customized keyboard-**As it is used worldwide, hence supports different languages keyboards.
- **Application support-**It has its own software repository from where users can download and install many applications. Linux can also run Windows applications if needed.

- **Live CD/USB** - Almost all Linux distributions have Live CD/ USB feature by which user can run/try the OS even without installing it on the system.

Advantages of Linux

The main Advantages of Linux over other Operating system are follows:

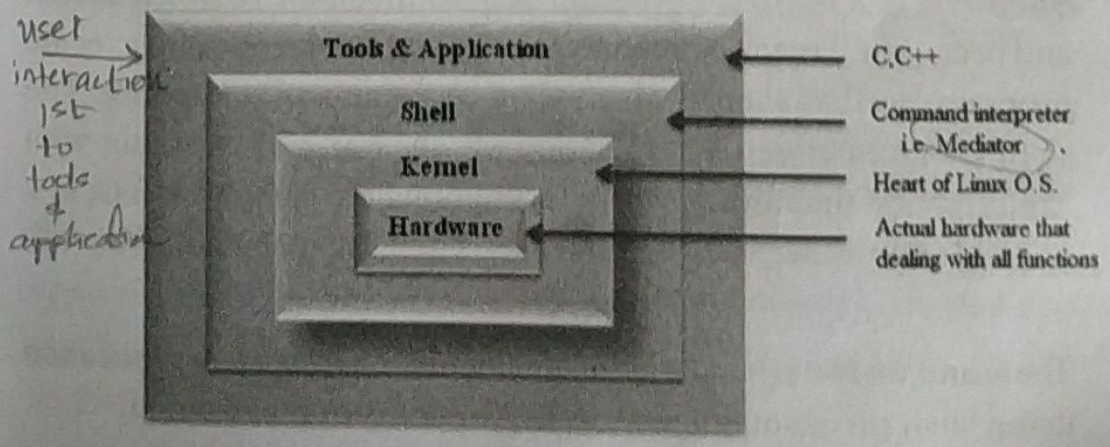
- **Open source**:- Linux is an Open source operating systems. You can easily get the source code for Linux and edit it to develop your personal operating system. Today, Linux is widely used for both basic home and office uses. It is the main operating system used for high performance business and in web servers. Linux has made a high impact in this world.
- **Low cost**:- There is no need to spend time and huge amount money to obtain licenses since Linux and much of its software come with the GNU General Public License. There is no need to worry about any software's that you use in Linux.
- **Stability**:- Linux has high stability compared with other operating systems. There is no need to reboot the Linux system to maintain performance levels. Rarely it freeze up or slow down. It has continuous up-times of hundreds of days or more.
- **Performance**:- Linux provides high performance on various networks. It has the ability to handle large numbers of users simultaneously.
- **Flexibility**:-Linux is very flexible. Linux can be used for high performance server applications, desktop applications, and embedded systems. You can install only the needed components for a particular use. You can also restrict the use of specific computers.
- **Compatibility**:- It runs all common Unix software packages and can process all common file formats.

- **Security**:- Linux is one of the most secure operating systems. File ownership and permissions make Linux more secure.
- **Networking**:- Linux provides a strong support for network functionality; client and server systems can be easily set up on any computer running Linux. It can perform tasks like network backup faster than other operating systems.
- **Multitasking**:- Linux is a multitasking operating system. It can handle many things at the same time.
- **Fast and easy installation**:- Linux distributions come with user-friendly installation.
- **Better use of hard disk**:- Linux uses its resources well enough even when the hard disk is almost full.
- **Wider Choice**:- There are a large number of Linux distributions which gives you a wider choice. Each organization develops and support different distribution. You can pick the one you like best; the core functions are the same.

The most popular Linux distributions are:*Red Hat, Ubuntu, Linux Mint, Arch Linux, Deepin, Fedora, Debian, openSUSE,Debian, CentOS, Scientific Linux* etc.

Overview of Unix and Linux architectures (Linux System Organization)

The following illustration shows the architecture of a Linux system –



The architecture of a Linux System consists of the following layers –

1. **Hardware:-** It includes the actual parts of computer through which a computer system works such as all the peripheral devices like CPU, Hard Disk Drive and RAM etc.

2. **Kernel:-** It is the heart of Linux operating system which is responsible for all major activities. Kernel interacts directly with actual hardware. It is collection of programs written in C language which will directly communicate with hardware. There is only one kernel for each system. It is loaded in memory, when system is booted it manages system resources allocated time between users and processes, decides priority, perform requests from hardware and all other things related to user. It is the core component of Operating System. The kernel has various functions as follows-

   - To manages the file system
   - To manages computer memory
   - To control access to the computer
   - To handle the interrupts
   - To handle the errors to perform i/p services
   - To carries out all data transfer between file system and hardware
   - To calculate resources of computer among the users
   - To schedule various programs running in memory to allocate C.P.U time to all running programs.

3. **Shell:-** It is a software program acts as mediator between kernel and user. Shell reads command prompt and send request to execute program, so that shell is also called as command interpreter. The shell program stored at file called as '**sh**'.For each working with Linux at any time different shell programs are running. Thus, at a particular time there may several shells running in memory but only one kernel.

4. **Tools and applications:-** This layer includes user written application using shell program language C,C++ application and so on.

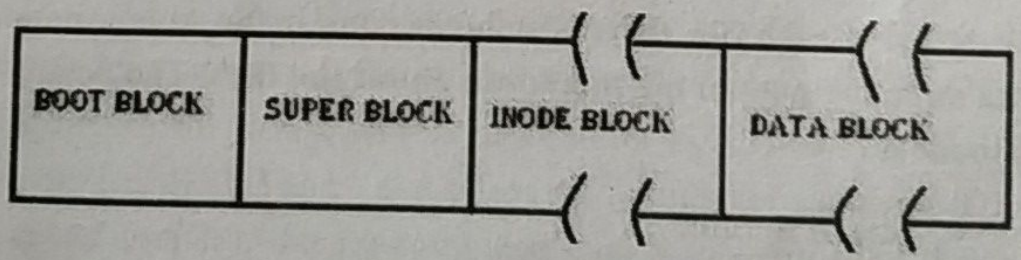
Linux files system

Each physical drive can be divided into several partitions. A partition is a container for information and can span an entire hard drive if desired. Each partition can contain one file system.

A file system is a **logical collection of files** on a partition or disk. It is defined as the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the file system.

When a partition or disk is formatted, the sectors in the hard disk are first divided into small groups. This group of sectors is called as *blocks*. The size of each block is 512 bytes. All the blocks belonging to the file system are logically divided into four parts.

- ❖ Boot Block
- ❖ Super Block
- ❖ Inode Table
- ❖ Data Block



1. Boot Block

A Boot block located in the first few sectors of a file system. The boot block contains the initial **bootstrap program** used to load the operating system. Typically, the first sector contains a boot strap program that reads in a larger bootstrap program from the next few sectors, and so forth.

2. Super Block

A Super block describes the **state of the file system**. It contains:

- The total size of the partition
- The block size of the file system
- How many files it can store
- Pointers to a list of free blocks
- Number of allocated and free nodes
- Link to lists of allocated and free nodes
- Size and allocation of inode tables
- The inode number of the root directory

The superblocks contain information about the general size and "shape" of the file system. To access a file in a file system requires access to the super block to get information about the file. Super block is backed up into multiple areas of a disk.

3. Inode block

We know that all entities in Linux are treated as files. The information related to all these files (not the contents) is stored in an *inode table* on the disk. For each file, there is an inode entry in the table. An inode is a data structure containing **metadata about the files**. The details stored in inode are:

- Size of the file
- User ID (owner) of file
- Group ID of the file
- Type of file (executable, block special etc)
- File access permissions (read, write, execute)
- The timestamps for file creation, modification etc
- Location of file on hard disk.

- Pointers to the blocks storing file's contents
- File protection flags
- Some other metadata about file.

4. Data Block

Data blocks start at the end of the inode list and contain the **actual file contents**. Data blocks contain several types of files. It contains users files, special files related to user data-regular files, directory files, symbolic link files, character special files, block special files etc. An allocated block can belong to only one file in the file system. This block cannot be used for storing any other file's contents unless the file to which it originally belonged is deleted.

Some Linux-supported File Systems

- **Ext**- stands for "Extended file system", and was the first created specifically for Linux. It's had four major revisions. "Ext" is the first version of the file system, introduced in 1992. It was a major upgrade from the Minix file system used at the time, but lacks important features. Many Linux distributions no longer support Ext.
- **Ext2, Ext3, Ext4**:- These file systems are robust. ext2 was the default file system under the 2.2 kernel. ext3 is simply the enhanced ext2 filesystem with a journaling feature. ext3 is the default file system for RHEL 3 and 4 and offers the best performance (in terms of speed and CPU usage) combined with data security of the file systems supported under Linux due to its journaling feature. ext4 was developed as the successor of ext3. It provides features for large filesystems, performance, increased limits, and reliability.
- **XFS**:- XFS is designed for high scalability and provides near native I/O performance even when the file system spans multiple storage devices.

- **OCFS**:- Oracle Cluster File System (OCFS) is a shared file system designed specifically for Oracle Real Application Cluster (RAC). OCFS eliminates the requirement for Oracle database files to be linked to logical drivers. OCFS volumes can span one shared disk or multiple shared disks for redundancy and performance enhancements.
- **OCFS2**:- OCFS2 is the next generation of the Oracle Cluster File System for Linux. It is an extent based, POSIX compliant file system. Unlike the previous release (OCFS), OCFS2 is a general-purpose file system that can be used for shared Oracle home installations making management of Oracle Real Application Cluster (RAC) installations even easier.
- **NFS**:- nfs is a network filesystem used to access remote disks.
- **JFS**:-Journaled File System", was developed by IBM for the IBM AIX operating system in 1990 and later ported to Linux. It boasts low CPU usage and good performance for both large and small files. JFS partitions can be dynamically resized, but not shrunk.

Types of Files in Linux

The Linux file system contains several different types of files. They are:

- ❖ Ordinary file
- ❖ Directories
- ❖ Special files
- ❖ Pipe file
- ❖ Socket file
- ❖ Symbolic link file

1. Ordinary files

All the files created by a user are ordinary files. Ordinary files include all data, program, object, and executable files. A user can modify ordinary files.

2. Directory Files

When you create a directory in Linux, a directory file is automatically created by the operating system. The directory file has the same name as the directory and contains information about the files stored in the directory. A user cannot modify a directory file. However when a file or subdirectory is added to the home directory, the directory file is modified automatically by the Linux operating system.

3. Special Files/ Device Files

Most of the system files in Linux are special files. Special files are usually associated with input/output devices. Special files are stored in standard Linux directories such as */dev* and */etc*. A user cannot modify special files. On Linux systems there are two flavours of special files for each device, **character special files** and **block special files**.

When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called *raw device access*. When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called *block device access*.

4. Pipes

Linux allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another. A Linux pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands.

5. Socket file

A socket file is used to pass information between applications for communication purpose. Socket is used in a client-server application framework.

6. Symbolic Link

Symbolic link is used for referencing some other file of the file system. These are linked files to other files. There are two types of link files available in Linux: **soft link** and **hard link**.

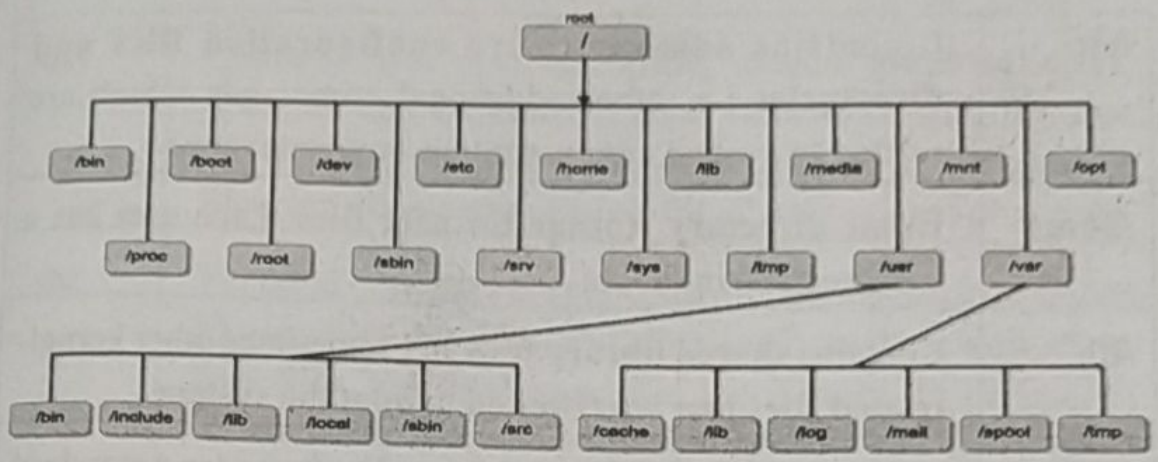
Soft link is an actual link to the original file, whereas a hard link is a mirror copy of the original file. If you delete the original file, the soft link has no value, because it points to a non-existent file. But the hard link can still has the data of the original file, because hard link acts as a mirror copy of the original file.

Linux standard Directories

There is a solid difference between the way the user sees the Linux file system and the way the kernel actually stores the files. To the user, the file system appears as a hierarchical arrangement of directories that contain files and other directories i.e., sub directories. Directories and files are identified by their names. The Linux file system resembles an upside down tree. And this hierarchy starts from a single directory called '*root*', which is represented by a "/" i.e. forward slash. Branching from root directory there are several standard sub directories.

A Linux file system is a collection of files and directories that has the following properties:

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an *inode*.
- By convention, the root directory has an inode number of 2, inode numbers 0and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self-contained. There are no dependencies between one filesystem and another.



Directory	Description
/(root)	Root directory is top level directory in the file system hierarchy and this directory contains all other directories and their subdirectories. It must contain **all of the filesrequired to boot the Linux system** before other file systems are mounted. It must include all of the required executable and libraries required to boot the remaining file systems. After the system is booted, all other file systems are mounted on standard, well-defined mount points as subdirectories of the root file system. The root directory also contains a file called as Linux which is Linux kernel itself.
/bin	It contains essential **binary files**, such as commands that are needed by both the system administrator and normal users. Usually also contains the shells, such as Bash.
/boot	Contains the static **boot loader and kernel executable and configuration files** required to boot a Linux computer.
/dev	This directory contains the **device files for every hardware device** attached to the system. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices. These include terminal devices (tty*), floppy disks (fd*), hard disks (hd* or sd*), RAM (ram*), and CD-ROM (cd*). Users can access these devices directly through these device files; however, some applications hide the actual device names to end users.

/etc	It contains **administrative configuration files and directories** i.e. other additional commands which are related to system administration & maintenance.
/home	**Home directory storage for user** files. Each user has a subdirectory in /home.
/lib	Contains **shared library files** and sometimes other kernel-related files that are required to boot the system.
/media	**Mount points for removable media**. Provides a standard location for auto mounting devices (removable media in particular). If the medium has a volume name, that name is typically used as the mount point. For example, a USB drive with a volume name of *myusb* would be mounted on */media/myusb*.
/mnt	A **temporary mount point** for regular file systems that can be used while the administrator is repairing or working on a file system.
/opt	**Optional files** such as vendor supplied application programs should be located here. Typically contains extra and third party software.
/root	It is the **home directory for the root user**. The home directory for root does not reside beneath /home for security reasons.
/sbin	System binary files. These are **executables used for system administration**. /sbin is similar to /bin, but it contains applications that only the super user will need. Contains administrative commands and daemon processes.
/tmp	Used by the operating system and many programs to **store temporary files**. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.

/usr	Contains user documentation, games, graphical files, libraries, and a variety of other commands and files that are not needed during the boot process. ie, programs, libraries, documentation etc. for all user-related programs.
/var	**Variable data files** are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more. In particular, this is where you would place files that you share as an FTP server (/var/ftp) or a web server (/var/www). It also contains all system log files (/var/log) and spool files in /var/spool (such as mail, cups, and news).
/proc	Contains **information about system resources**. It contains information about your computer, such as information about your CPU and the kernel your Linux system is running.
/srv	Contains **data for servers**. If you are running a web server from your Linux box, your HTML files for your sites would go into /srv/http (or /srv/www). If you were running an FTP server, your files would go into /srv/ftp.
/sys	/sys is another virtual directory like /proc and /dev and also contains **information from devices** connected to your computer.

Following are the salient features of the Linux file system:

- It has a hierarchical file structure.
- Files can grow dynamically
- Files have access permissions
- All devices are implemented as files.

Hardware requirements for Linux

You can get a Linux distribution that runs on hand-held devices or an old PC in your closet with as little as 24MB of RAM and a 486

processor. To have a good desktop PC experience with Linux, however, you should consider what you want to be able to do with Linux when you are choosing your computer.

- **Processor** — A 400 MHz Pentium processor is the minimum for a GUI installation. For most applications, a 32-bit processor is fine (x86). However, if you want to set up the system to do virtualization, you need a 64-bit processor (x86_64).
- **RAM** — Needs at least 1GB of RAM, but at least 2GB or 3GB would be much better.
- **Disk space** — Needs at least 10GB of disk space for an average desktop installation, although installations can range (depending on which packages you choose to install) from 600MB (for a minimal server with no GUI install) to 7GB (to install all packages from the installation media). Consider the amount of data you need to store. While documents can consume very little space, videos can consume massive amounts of space.
- **DVD or CD drive** — You need to be able to boot up the installation process from a DVD, CD, or USB drive. If you can't boot from a DVD or CD, there are ways to start the installation from a hard disk or USB drive. After the installation process is started, more software can sometimes be retrieved from different locations (over the network or from hard disk, for example).
- **Network card** — You need a wired or wireless networking hardware to be able to add more software or get software updates.
- **Special hardware features** — Some Linux features require special hardware features. For example, to use as a virtualization host using KVM, the computer must have a processor that supports virtualization. These include AMD-V or Intel-VT chips.

Man Page in Linux

The manual (man) pages provide detailed descriptions and usage of the commands. You can use the man command to display the man

page entry that explains a given command. The syntax of the man command is as follows.

```
man <commandname>
```

Example: man ls

This command displays the man page *ls* command. ls is used for listing.

Commands for files and directories

cd:

To **change directory** - change the current working directory to a specific directory.

Syntax:

```
cd [Options] [Directory]
```

Option	Meaning
cd ~	change to your home directory
cd /	change to root directory
cd ..	change to parent directory
cd -	change to previous directory where you working earlier

Example :

```
cd /bin
```

The directory in which you find yourself when you first login is called your *home directory*. You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

Eg 1:- You can go in your home directory anytime using the following command

```
$cd ~
$
```

Here ~ indicates the home directory.

Eg 2:- Suppose you have to go in any other user's home directory, use the following command

```
cd ~username
```

Eg 3:- To move inside a directory from a directory

```
cd dir1/dir2/dir3
```

Eg 4:- To navigate to a directory with white spaces, specify directory name in double quotes or single quotes.

```
cd "dir name"
```

```
[salmiya@localhost ~]$ cd dir1/dir2/dir3
[salmiya@localhost dir3]$ cd ..
[salmiya@localhost dir2]$ cd ~
[salmiya@localhost ~]$ cd /bin
[salmiya@localhost bin]$ cd ~
[salmiya@localhost ~]$ cd /
[salmiya@localhost /]$ cd -
/home/salmiya
[salmiya@localhost ~]$
```

ls

ls command is used to **list the content** of the specified directory. ie. directory and files in a directory.

```
ls [option] [directory]
```

Option	Meaning
-a	List all files including hidden files starting with '.'
-d	List directory
-l	List with long format
-r	List in reverse order

-s	Sort by size (Display the number of storage blocks used by a file.)
-t	Sort by time and date
-X	Sort by extension name
-R	List the file and folder of directory and sub directory
-C	List entries by columns
-i	Print inode number of each file

Example : ls -a

Following is the example to list all the files contained in **/usr/local** directory "

```
ls /usr/local
X11         bin     gimp    jikes   sbin
ace         doc     include lib     share
atalk       etc     info    man     ami
```

cp

cp command is used to **copy the content** of one file into another.

cp [options] <source-file><destination-file>

If the destination is an existing file, the file is overwritten: If the destination is an existing directory, the file is copied into that directory.

Option	Meaning
-a	Archive files
-f	Force copy by removing the destination file
-n	Do not overwrite an existing file
-R	Recursive copy (including hidden file)
-v	Print information message (Explain what is being done)
-i	Prevent cp from overwriting existing files. Prompt before overwriting destination file.
-p	Preserve all information, including owner, group, and permissions.

Example : cp test.txt sample.txt

Copy the file "test.txt" to the file "sample.txt" in the current working directory. This command will create the file "sample" if it doesn't exist. It will normally overwrite it without warning if it exists.

- To copy multiple files using the *cp* command, pass the names of files followed by the destination directory.

```
cp file1 file2 file3 dir1
```

- To copy entire directory structures from one place to another use the -R option to perform a recursive copy. cp copies all files of the source directory to the destination directory, creating any files or directories needed.

```
cp -R Src_directory Dest_directory
```

In the above command, if the *Dest_directory* doesn't exist, cp creates it and copies content of Src_directory recursively as it is. But if *Dest_directory* exists then copy of *Src_directory* becomes sub-directory under *Dest_directory.*

- If you want to be prompted before over writing a file, use the –i option.

```
cp -i file1.txt sample.txt
```

If *sample.txt* already exists, you will be prompted:

cp: overwrite 'sample.txt'?

If you type y (or yes) then *sample.txt* will be overwritten with a copy of *file1.txt.*

- To show files that are being copied use the -v option to the cp. This prints the files and folders that are being copied to standard output.

```
cp -v oldfile newfile
```

```
[salmiya@localhost ~]$ cp -v test.txt sample.txt
'test.txt' -> 'sample.txt'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cp -v test.txt sample.txt dir1
'test.txt' -> 'dir1/test.txt'
'sample.txt' -> 'dir1/sample.txt'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cp file1.txt -iv sample.txt
cp: overwrite 'sample.txt'? y
'file1.txt' -> 'sample.txt'
[salmiya@localhost ~]$
```

rm

The rm command is used to **remove files or directory**. To remove a file, you must have write permission for the directory that contains the file.

rm [options] <filename>

Options	Meaning
-i	Prompt before every removal
-I	Prompt once before removing more than three files, or when removing recursively
-r, -R	Remove directories and their contents recursively
-d	Remove empty directories
-f	Remove write-protected file, never prompt before removing
-v	Explain what is being done(print the name of each file before removing it)

Eg-1: Remove the file *test.txt*. If the file is write-protected, you will be prompted to confirm that you really want to delete it:

```
rm test.txt
```

Eg-2:Remove the file *myfile.txt*. You will not be prompted, even if the file is write-protected:

```
rm -f test.txt
```

Eg-3: Remove all files in the working directory. If it is write-protected, you will be prompted before rm removes it:

```
rm *
```

Eg-4: Remove all files in the working directory. rm will not prompt you for any reason before deleting them:

```
rm -f *
```

Eg-5: Attempt to remove every file in the working directory, but prompt before each file to confirm:

```
rm –i *
```

Eg-6: Remove every file in the working directory; prompt for confirmation if more than three files are being deleted:

```
rm –I *
```

Eg 7:- Remove all files with .txt extension

```
rm –i *.txt
```

Eg-8:Remove the directory *mydirectory*, and any files and directories it contains. If a file or directory that *rm* tries to delete is write-protected, you will be prompted to make sure that you really want to delete it:

```
rm -r mydirectory
```

Eg-9:Same as the above command, but you will never be prompted; if rm can delete the files, it will:

```
rm -rf mydirectory
```

```
[salmiya@localhost ~]$ rm -v test.txt
removed 'test.txt'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ rm -i file1.txt
rm: remove regular file 'file1.txt'? y
[salmiya@localhost ~]$
[salmiya@localhost ~]$ rm -rv dir1
removed directory: 'dir1/dir2/dir3'
removed directory: 'dir1/dir2'
removed 'dir1/test.txt'
removed 'dir1/sample.txt'
removed directory: 'dir1'
[salmiya@localhost ~]$
```

mkdir

The mkdir command is used to **create a new directory**. This command can create multiple directories at once and also set permissions when creating the directory.

mkdir [option] <directory-name>

Options	Meaning
-m	Set permission mode
-p	Create parent directories as necessary. When this option is specified, no error is reported if a directory already exists.
-v	Print a message for each created directory

Eg-1: Creates a new directory called '*mydirectory*' whose parent is the current directory.

```
mkdir mydirectory
```

Eg-2: Create the directory 'mydirectory', and set its permissions such that all users may read, write, and execute the contents.

```
mkdir -m a=rwx mydirectory
```

Eg-3: To create multiple directories at one time:

```
mkdir dir1 dir2 dir3
```

Eg-4: To create several subdirectories at one time, use –p option to create intermediate directories if not exists.

```
mkdir -pfirst/second/third
```

```
[salmiya@localhost ~]$ mkdir -v mydirectory
mkdir: created directory 'mydirectory'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ mkdir -v dir1 dir2 dir3
mkdir: created directory 'dir1'
mkdir: created directory 'dir2'
mkdir: created directory 'dir3'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ mkdir -pv first/second/third
mkdir: created directory 'first'
mkdir: created directory 'first/second'
mkdir: created directory 'first/second/third'
[salmiya@localhost ~]$
```

rmdir

The rmdir command is used to **remove the directory**. The directory should be empty before removing it.

rmdir [options] <directoryname>

Options	Meaning
-v	Print a message for every directory processed, explain what is being done
-p	Each of the directory argument is treated as a pathname of which all components will be removed, if they are already empty, starting from the last component.

Eg-1: To delete the directory test:

```
rmdir test
```

Eg-2: To delete nested empty directories:

```
rmdir -p first/second/third
```

Eg-3:Remove the directories dir1, dir2, and dir3, if they are empty. If any are not empty, an error message will be printed for that directory, and the others will be removed.

```
rmdir  dir1  dir2  dir3
```

```
[salmiya@localhost ~]$ rmdir -v mydirectory
rmdir: removing directory, 'mydirectory'
[salmiya@localhost ~]$
[salmiya@localhost ~]$ rmdir -pv first/second/third
rmdir: removing directory, 'first/second/third'
rmdir: removing directory, 'first/second'
rmdir: removing directory, 'first'
[salmiya@localhost ~]$
```

pwd

The pwd command displays the **present working directory** or current directory.

```
pwd  [option]
```

Options	Meaning
-L	Display the content as absolute name (Print symbolic path)
-P	Print fully resolved name for the current directory (Print actual path)

```
[salmiya@localhost ~]$ cd /bin
[salmiya@localhost bin]$
[salmiya@localhost bin]$ pwd
/bin
[salmiya@localhost bin]$
[salmiya@localhost bin]$ pwd -L
/bin
[salmiya@localhost bin]$
[salmiya@localhost bin]$ pwd -P
/usr/bin
[salmiya@localhost bin]$
```

file

The file command is used to determine the **type of a file**. It reports the file type in human readable format (e.g. 'ASCII text') or MIME type (e.g. 'text/plain; charset=us-ascii'). File can be a useful command to determine how to view or work with a file. The file name along with the file type will be printed to standard output.

```
file [options] <filename>
```

Options	Meaning
-b	To show just the file type
-i	To view the mime type of a file rather than the human readable format
-F	File and file type are separated by : But we can change separator using -F option
-s	Used for special files.
z	To view compressed files without decompressing

Eg 1:-

```
file sample.txt
```

Eg 2:- To display the types of all files.

```
file *
```

Eg 3:- To see the file type without filename.

```
file -b sample.txt
```

Eg 4:- To display all files file types in particular directory.

```
file directoryname/*
```

Eg 5:- To view mime type of file.

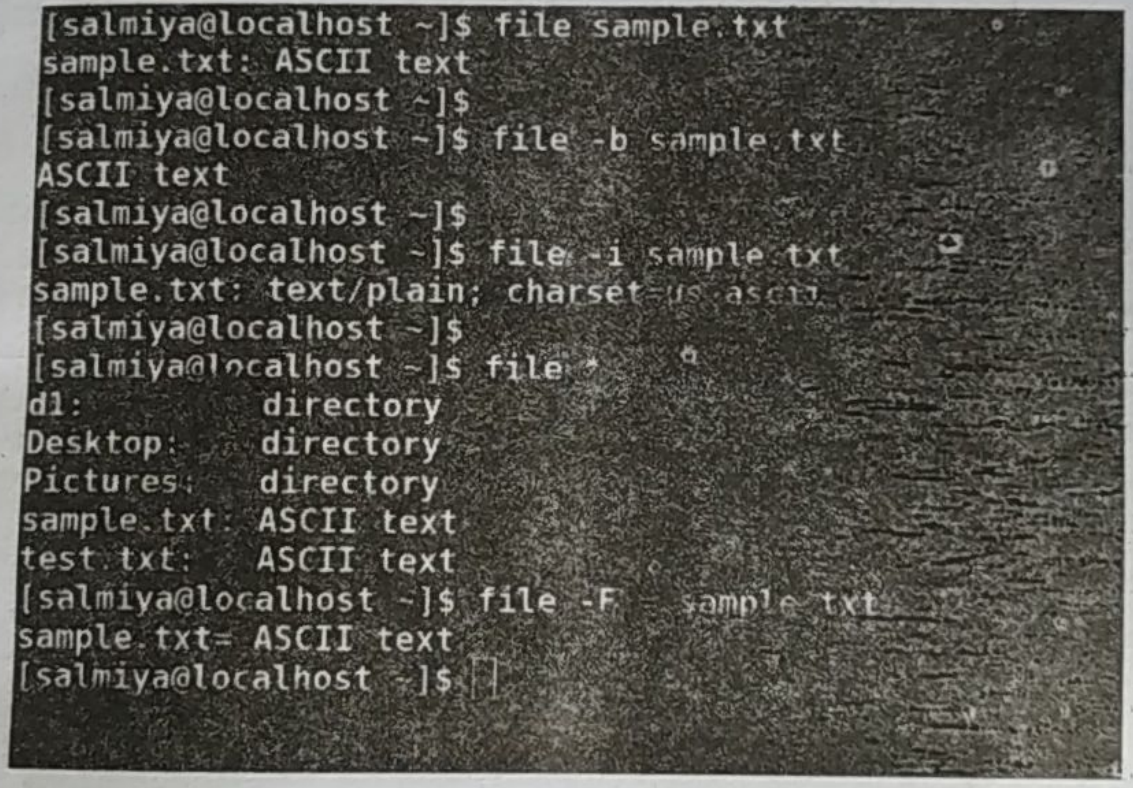
```
file -i sample.txt
```

Eg 6:- To display special files

```
file -s /dev/sda
```

Eg 7:- To display file types of multiple files

```
file file1 file2 file3
```



more

If the information to be displayed on the screen is very long, it scrolls up on the screen fastly. So the user cannot be able to read it. The more command is used to **display the output page by page** (without scrolling up on the screen fastly). While viewing the file use the following controls:

Enter key	:	To scroll down line by line.
Space bar or **f** key	:	To go to the next page.
b key	:	To go to back one page.
q key	:	To quit displaying

The syntax is as follows:

```
more <filename>
```

Options	Meaning
-d	Prompt the user with the message "[Press space to continue, 'q' to quit.]" and will display "[Press 'h' for instructions.]" instead of ringing the bell when an illegal key is pressed.
-s	Squeeze multiple blank lines into one.
-u	Omits the underlines.
-NUM	Specify the number of lines per screenful
+NUM	Display file beginning from line number NUM

Example:

```
more sample.txt
man ls | more
```

This command will display the man page of *ls* command page by page.

less

The less command has the same syntax and functions of more.ie, used to read contents of text file **one page per time** (one screen). It has faster access because if file is large, it don't access complete file, but access it page by page.For example, if it's a large file and you are reading it using any text editor, then the complete file will be loaded to main memory, but less command don't load entire file, but load it part by part, which makes it faster.

General format is:

```
less <filename>
```

Options	Meaning
-F	causes less to exit if entire file can be displayed on first screen
-s	Squeeze multiple blank lines into one.
-u	Omits the underlines.
-n	Suppress line number

Example:

```
less sample.txt
man ls | less
```

Creating and viewing files using cat

The cat command is used to **display the content of text file** and to combine several files to one file. It is also use to **create a file**.

The cat command (short for "**concatenate** ") is one of the most frequently used command in Linux. The **cat** command allows us to create single or multiple files, view content of file, concatenate files and redirect output in terminal or files. The cat command display files content to a screen. The cat command can read and write data from standard input and output devices. It has three main functions related to manipulating text files: **creating them, displaying them, and combining them**. It is also used to copy text files into a new document.

cat [options] <filename>

Options	Meaning
-b	add line number to non-blank lines
-n	add line number to all lines
-s	suppress repeated empty output lines
-E	display $ at end of each line

- To view the content of a file, use the following command:

  cat <filename>

- To create a new file, use:

  cat > [name-of-new-file]

When run, the command requires you to enter the information on the terminal. Once you're done, just press Ctrl+d to save the file.

- To copy the contents of one file to another file:

  cat [filename-whose-contents-is-to-be-copied] > [destination-filename]

- To display contents of multiple files:

  cat filename1 filename2 filename3 ...

- To append the contents of one file to another, you can use the double greater than '>>' symbol with the cat command.

  cat filename1 >> filename2

  Here, the content of *filename1* is added at the end of *filename2*.

- To display contents of all txt files, use the following command.

  cat *.txt

- To display the contents of a file with line number, use the following command.

  cat -n filename

Examples:

Eg 1:- To create a new file:

```
cat >sample.txt
```

Eg 2:- To view the content of a file:

```
cat sample.txt
```

Eg 3:- To copy the contents of one file to another file

```
cat sample.txt > samplecopy.txt
```

Eg 4:- To append the contents of one file to another:

```
cat sample.txt >>test.txt
```

```
[salmiya@localhost ~]$ cat > sample.txt
creating new file named sample.txt
The file is created using cat command
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample.txt
creating new file named sample.txt
The file is created using cat command
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample.txt > samplecopy.txt
[salmiya@localhost ~]$ cat samplecopy.txt
creating new file named sample.txt
The file is created using cat command
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample.txt
creating new file named sample.txt
The file is created using cat command
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat test.txt
This is a file named test.txt
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample.txt >> test.txt
[salmiya@localhost ~]$ cat test.txt
This is a file named test.txt
creating new file named sample.txt
The file is created using cat command
[salmiya@localhost ~]$
```

File Comparisons Commands

The file comparison command helps us to compare the files and find the similarities and differences between these files. The different file comparison commands used in Linux are cmp, comm, diff, and uniq.

1. cmp:

This command is used to compare two files **character by character**.

```
cmp [options] <filename1> <filename2>
```

cmp command is used to compare the two files **byte by byte** and helps you to find out whether the two files are identical or not. When *cmp* is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found and displays no message and simply returns the prompt if the files compared are identical. It also displays line number if a difference is found.

Options	Meaning
-b	Displays the differing bytes in the output
-i [bytes-to-be-skipped]	Skip a particular number of initial bytes from both the files and then after skipping it compares the files.
-i SKIP1:SKIP2	Skip the first SKIP1 bytes of File1 and the first SKIP2 bytes of File2
-l	Print byte position and byte value for all differing bytes.
-s	Suppress all normal output, print exit status only.
-n [no. of bytes]	Limit the number of bytes you want to compare.

Eg:-1

```
cmp file1 file2
```

If the files are not identical the command will display the first differing byte and line number.If the files are identical,it displays no message and you will see something like this on your screen:

$ _

/*indicating that the files are identical*/

Eg 2:-

```
cmp -i 10 file1  file2
```

This command skips first 10 bytes from both files and then compares the files. Note that in cases like these (where you use -i to skip bytes), the byte at which the comparison begins is treated as byte number zero.

Eg 3:-

```
cmp -i 10:15 file1 file2
```

Here 10 bytes skipped from first file and 15 bytes skipped from second file and the compare them.

Eg 4:-

```
cmp -l file1 file2
```

If the files are different this command displays the position of differing bytes along with the differing bytes in both file. The first column in the output represents the position (byte number) of differing bytes. The second column represents the byte value of the differing byte in the first file, while the third column represents the byte value of the differing byte in the second file.

Eg 5:-

```
cmp -s file1 file2
```

This allows you to suppress the output normally produced by cmp command i.e it compares two files without writing any messages. This gives an exit value of 0 if the files are identical, a value of 1 if different, or a value of 2 if an error message occurs.

Eg 6:-

```
cmp -n 50 file1 file2
```

This command compares first 50 bytes.

```
[salmiya@localhost ~]$ cat file1
Apple
Orange
Grapes
[salmiya@localhost ~]$ cat file2
Apple
Orange
Banana
[salmiya@localhost ~]$ cmp file1 file2
file1 file2 differ: byte 14, line 3
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cmp -l file1 file2
14 107 102
15 162 141
16 141 156
17 160 141
18 145 156
19 163 141
[salmiya@localhost ~]$
```

2. diff

diff stands for difference. This command is used to display the differences in the files by **comparing the files line by line**. It outputs a set of instructions for how to change one file to make it identical to the second file.

The syntax is:

diff [options] <filename1> <filename2>

Option	Meaning
-i	Ignore case differences in file contents, consider upper and lower case letters equivalent
-b	Ignore changes in the amount of white space (such as spaces or tabs)
-w	Ignore all white space.
-q	Prompt when two files are differ
-s	Report when two files are the same.

The important thing to remember is that *diff* uses certain special symbols and instructions that are required to make two files identical. It tells you the instructions on how to change the first file to make it match the second file. Special symbols used are:

a : Add

c : Change

d : Delete

The first line of the diff output will contain:

- line numbers corresponding to the first file,
- a letter (**a** for *add*, **c** for *change*, or **d** for *delete*), and
- line numbers corresponding to the second file.

The three dashes ("- - -") merely separate the lines of file1 and file2. And also

- Lines preceded by a < are lines from the first file;
- Lines preceded by > are lines from the second file.

Example: we have two files with names *file1* and *file2* containing the following data. Then applying diff command without any option we get the following output:

```
[salmiya@localhost ~]$ cat file1
Apple
Orange
Grapes
Mango
[salmiya@localhost ~]$ cat file2
Apple
Banana
Grapes
Mango
Watermelon
[salmiya@localhost ~]$ diff file1 file2
2c2
< Orange
---
> Banana
4a5
> Watermelon
[salmiya@localhost ~]$
```

Here, 2c2 means line 2 in the *file1*needs to be changed to match the line number 2 in the *file2*. Next line contains 4a5, which means, after line 4 you have to add 'Watermelon' to match the second file line number 5. As a summary, to make both the files identical, Change the line number 2 in the *file1* ie, 'Orange' with line number 2 of *file2* ie, 'Banana'. Then add 'Watermelon' in *file1* after line number 4 to match the line number 5 of *file2*.

Example 2:-Now let's see what it looks like when diff tells us that we need to **delete a line.**

```
[salmiya@localhost ~]$ cat test1
Apple
Orange
Grapes
Mango
[salmiya@localhost ~]$ cat test2
Apple
Orange
Mango
[salmiya@localhost ~]$ diff test1 test2
3d2
< Grapes
[salmiya@localhost ~]$
```

Here above output 3d2 means delete line 3rd of *file1* i.e. 'Grapes' so that both the files sync up at line 2.

Linux system offers two different ways to view the *diff* command output i.e. **context mode** and **unified mode**.

- **Context Mode (-c)** : To view differences in context mode, use the **-c** option.

  Example: We have two files *file1* and *file*.

```
[salmiya@localhost ~]$ cat file1
Apple
Orange
Grapes
Mango
[salmiya@localhost ~]$ cat file2
Apple
Banana
Grapes
Mango
Watermelon
[salmiya@localhost ~]$ diff -c file1 file2
*** file1       2019-10-08 20:42:04.310680050 +0530
--- file2       2019-10-08 20:42:37.436490383 +0530
***************
*** 1,4 ****
  Apple
! Orange
  Grapes
  Mango
--- 1,5 ----
  Apple
! Banana
  Grapes
  Mango
+ Watermelon
[salmiya@localhost ~]$
```

The first file is indicated by ***, and the second file is indicated by —. The first two lines of this output show us information about file1 and file2. It lists the file name, modification date, and modification time of each of our files, one per line. The line "**********" is just a separator.

The next line has three asterisks ("***") followed by a line range from the first file (in this case lines 1 through 4, separated by a comma). Then four asterisks ("****").After that it shows the contents of the first file with the following indicators:

1. If the line needs to be unchanged, it is prefixed by two spaces.
2. If the line needs to be changed, it is prefixed by a symbol and a space. The symbol means are as follows:

Character	Meaning
+	It indicates a line in the second file that needs to be added to the first file to make them identical.
-	It indicates a line in the first file that needs to be deleted to make them identical.
!	It indicates a line that is to be changed.

After the lines from the first file, there are three dashes ("—"), then a line range from the second file, then four dashes ("—").Then it shows the contents of the second file.

If there is more than one section that needs to change, *diff* will show these sections one after the other. Lines from the first file will still be indicated with "***", and lines from the second file with "—".

- **Unified mode (-u)**: To view differences in unified mode, use the -u option. It is similar to context mode but it **doesn't display any redundant information** or it shows the information in concise form.

Example:

```
[salmiya@localhost ~]$ cat file1
Apple
Orange
Grapes
Mango
[salmiya@localhost ~]$ cat file2
Apple
Banana
Grapes
Mango
Watermelon
[salmiya@localhost ~]$ diff -u file1 file2
--- file1       2019-10-08 20:42:04.310680050 +0530
+++ file2       2019-10-08 20:42:37.436490383 +0530
@@ -1,4 +1,5 @@
 Apple
-Orange
+Banana
 Grapes
 Mango
+Watermelon
[salmiya@localhost ~]$
```

The output is similar to above, but as you can see, the differences are "unified" into one set.

The first file is indicated by —, and the second file is indicated by +++.The first two lines of this output show us information about file1 and file2. It lists the file name, modification date, and modification time of each of our files, one per line.

After that the next line has two at sign @ followed by a line range from the first file (in our case lines 1 through 4, separated by a comma) prefixed by – and then space and then again followed by a line range from the second file prefixed by + and at the end two at sign @. Followed by the file content in output tells us which line remain unchanged and which lines needs to added or deleted (indicated by symbols) in the file 1 to make it identical to file 2.

3. comm

Used to **compare two sorted files line by line** and write to standard output: the lines that are common, plus the lines that are unique.

comm [options] <File1><File2>

With no options, it produces three-column output. Column one contains lines unique to File1, column two contains lines unique to File2, and column three contains lines common to both files. *comm* command only works right if you are comparing two files which are already sorted.

Option	Meaning
-1	suppress column 1 (lines unique to FILE1)
-2	suppress column 2 (lines unique to FILE2)
-3	suppress column 3 (lines that appear in both files)

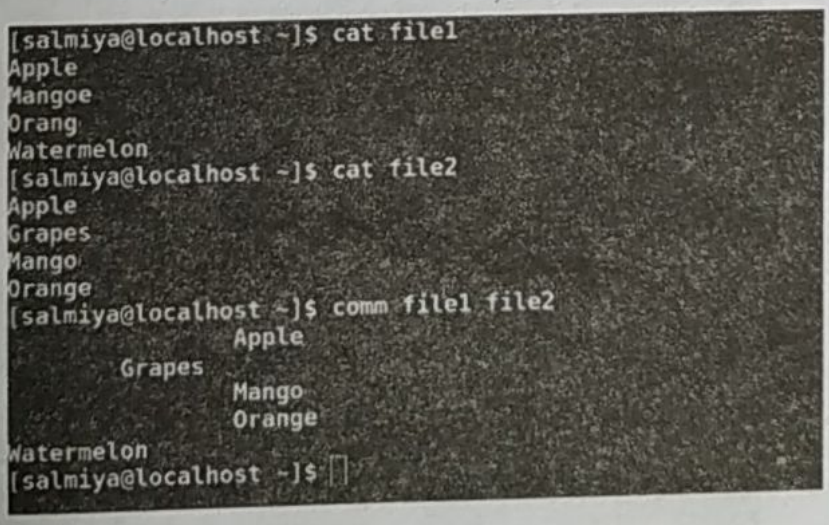
Eg 1:

```
comm file1 file2
```

Eg 2:

```
comm -1 file1 file2
```

This command suppress column 1 , ie, lines unique to file1



4. uniq

With the help of uniq command you can form a sorted list in which every word will occur only once. ie,the uniq command reports or **filters out repeated lines** in a file.

uniq [option] <filename>

Option	Meaning
-d	Displays only the duplicate lines
-u	Displays only unique lines
-i	Normally, comparisons are case-sensitive. This option performs case-insensitive comparisons instead
-c	Displays line by eliminating duplicate lines and prefix lines with a number representing how many times they occurred.
-w	It only compares N characters in a line.

To sore the output of *uniq* command, you should specify the destination filename as shown below:

uniq <source-file> <destination-file>

```
[salmiya@localhost ~]$ cat file1
Apple
Apple
Banana
Grapes
Grapes
Grapes
Mangoe
Mangoe
Watermelon
[salmiya@localhost ~]$ uniq file1
Apple
Banana
Grapes
Mangoe
Watermelon
[salmiya@localhost ~]$ uniq file1 file2
[salmiya@localhost ~]$ cat file2
Apple
Banana
Grapes
Mangoe
Watermelon
[salmiya@localhost ~]$ uniq -d file1
Apple
Grapes
Mangoe
[salmiya@localhost ~]$ uniq -u file1
Banana
Watermelon
[salmiya@localhost ~]$ uniq -c file1
      2 Apple
      1 Banana
      3 Grapes
      2 Mangoe
      1 Watermelon
[salmiya@localhost ~]$
```

REVIEW QUESTIONS

Part A

1. What is Linux?
2. What do you mean by open source?
3. What is kernel?
4. What is super block?
5. What is inode?
6. What is the use of *file* command?
7. Compare *more* and *less* command.
8. How to create a directory in Linux?
9. What is *cp* command?
10. What is the use of *pwd* command?
11. What is *rmdir* command?
12. What is *rm* command?
13. What are the different options of *ls* command?
14. Write the command to change a directory in Linux.

Part B

15. Explain Linux system Architecture.
16. What are the features of Linux OS?
17. Write note on creating and viewing files using *cat* command.
18. Write a note on creating and deleting directory.
19. Explain different types of files in Linux.

Part C

20. Explain Linux file systems in detail.
21. Explain Linux standard directories.
22. Explain file comparison commands with examples.

UNIT 2

ESSENTIAL LINUX COMMANDS

Processes

A process refers to a **program in execution**; it's a running instance of a program. It is made up of the program instruction, data read from files, other programs or input from a system user.

It is a dynamic entity, constantly changing as the machine code instructions are executed by the processor. As well as the program's instructions and data, the process also includes the program counter and all of the CPU's registers as well as the process stacks containing temporary data such as routine parameters, return addresses and saved variables. During the lifetime of a process it will use many system resources. It will use the CPUs in the system to run its instructions and the system's physical memory to hold it and its data.

Processes in Linux

In Linux a process is a running instance of a command. Whenever you issue a command in Linux, it creates, or starts, a new process. For example, when you use the ***ls*** command to list the directory contents, you started a process.

A process is identified on the system by what is referred to as a process ID. That process ID is unique for the current system. In other words, no other process can use that number as its process ID while that first process is still running. However, once a process is ended, another process can

reuse that number. Along with a process ID number, there are other attributes associated with a process. Each process, when it is run, is associated with a particular user account and group account. That account information helps determine what system resources the process can access. For example, processes run as the root user have much more access to system files and resources than a process running as a regular user.

States of a Process in Linux

As a process executes it changes state according to its circumstances. Linux processes have the following states:

- **Running:-** The process is either running (it is the current process in the system) or it is ready to run (it is waiting to be assigned to one of the system's CPUs).
- **Waiting (Blocked):-** The process is waiting for an event or for a resource. Linux differentiates between two types of waiting process; interruptible and uninterruptible. Interruptible waiting processes can be interrupted by signals whereas uninterruptible waiting processes are waiting directly on hardware conditions and cannot be interrupted under any circumstances.
- **Stopped:-**Once the process is completed, this state occurs. A process that is being debugged can be in a stopped state. This process can be restarted.
- **Zombie:-** This is a halted process or a dead process. The process will be terminated and the information will still be available in the process table.

Types of Processes in Linux

1. Parent and Child process

Each Linux process has two ID numbers assigned to it: The Process ID (pid) and the Parent process ID (ppid). Each user process in the system has a parent process. When you run a program in your shell, a process is

created. This new process is called a child process of the shell. The originating process (the shell from which you ran the command) is called the parent process of the child. For each user process there's a parent process in the system, with most of the commands having shell as their parent.

2. Zombie process (or defunct process)

A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done, the zombie process is eliminated from the process table. This is known as reaping the zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

3. Orphan process

An Orphan process is a running process whose parent process has finished or terminated. i.e, a process whose parent process no more exists, either finished or terminated, without waiting for its child process to terminate is called an orphan process. The orphaned process will be immediately adopted by the special *init* system process once its parent process dies.

4. Daemon process

A daemon process is a process which runs in background and has no controlling terminal. Since a daemon process usually has no controlling terminal so almost no user interaction is required. Daemon processes are used to provide services that can well be done in background without any user interaction. They are system-related background processes that often run with the permissions of root and services requests from other processes.

Process Attributes

Following are the characteristics associated with each process:

- **The process ID** or PID: a unique identification number used to refer to the process.

- **The parent process ID** or PPID: the number of the process (PID) that started this process.
- **Nice number**: the degree of friendliness of the process towards other processes (process priority is calculated from nice numbers and recent CPU usage).
- **Terminal** or TTY: the terminal to which the process is connected.
- **User name** of the real and effective user (RUID and EUID): the owner of the process. The real owner is the user issuing the command; the effective user is the one determining access to system resources. RUID and EUID are usually the same, and the process has the same access rights the issuing user would have.
- Real and effective **group** owner (RGID and EGID): The real group owner of a process is the primary group of the user who started the process. The effective group owner is usually the same as RGID.

Starting a Process

When you start a process (run a command), there are two ways you can run it

1. **Foreground Processes**: They run on the screen and need input from the user. For example Office Programs
2. **Background Processes**: They run in the background and usually do not need user input. For example Antivirus.

Foreground Processes

Foreground processes also referred to as *interactive processes* are initialized and controlled through a terminal session. In other words, there has to be a user connected to the system to start such processes; they haven't started automatically as part of the system functions/services. By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt

would not be available until the program finishes processing and comes out.

To start a foreground process, you can run it from the terminal. If you start a foreground program/process from the terminal, then you cannot work on the terminal, till the program is up and running. Particular, data-intensive tasks take lots of processing power and may even take hours to complete. You do not want your terminal to be held up for such a long time. To avoid such a situation, you can run the program and send it to the background so that terminal remains available to you.

Background Processes

Background processes also referred to as *non-interactive/automatic process* are processes not connected to a terminal; they don't expect any user input. It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be done in parallel with the process running in background since they do not have to wait for the previous process to be completed. The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another. The simplest way to start a background process is to add an ampersand (**&**) at the end of the command.

Example:

```
cp sample samplecopy &
```

Connecting Processes with Pipes

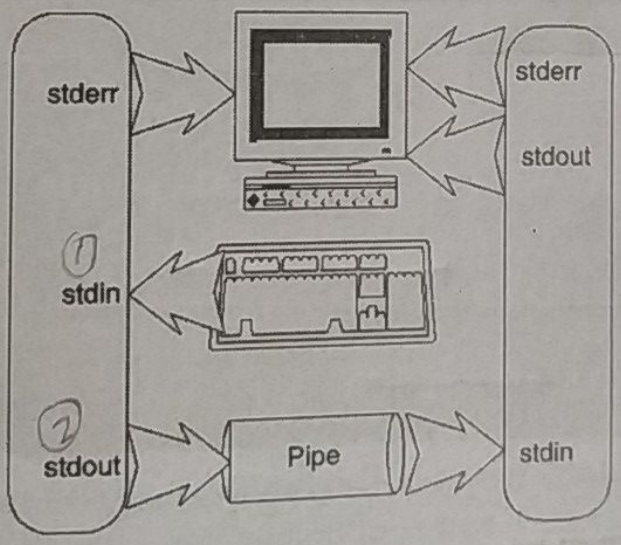
A pipe is a form of redirection that is used in Linux operating systems to **send the output of one program to another program** for further processing. It also uses the output of one command as input of another command. We call this process piping due to its similarity to the real-world use of a pipe. At the command line, both of the processes (commands) are connected using the **vertical bar symbol "|"**. This symbol is often called a pipe symbol. When two commands are connected through a pipe, the first command sends its output to the pipe instead of sending it to the terminal

screen. The second command reads its input from the pipe rather than from the keyboard. Both of the commands still send error messages to the terminal screen.

That is, a pipe is used to combine two or more command and in this the output of one command act as input to another command and this command output may act as input to next command and so on. It can also be visualized as a temporary connection between two or more commands/ programs/ processes. This direct connection between commands/ programs/ processes allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen. Pipes are unidirectional, i.e data flow from left to right through the pipeline.

The syntax is as follows:

Command1 | Command2 | Command3 | .... | CommandN



The first command takes input from the keyboard (stdin), and the second command sends its output to the terminal screen (stdout). If the second command needs input data but nothing is available in the pipe, it just waits for the first command to send something into the pipe. Pipes are often used to filter, modify, or manipulate data output of one command. Multiple levels of pipes can be used in one command line. Similarly, pipes can also be used in combination with I/O redirection symbols.

Eg 1:- Listing all files and directories and give it as input to more command

```
ls -l | more
```

The more command takes output of *ls -l* as its input. The net effect of this command is that the output of *ls -l* is displayed one screen at a time. The pipe act as a container which take output of *ls -l* and giving it to *more* as input. This command does not use a disk to connect standard output of *ls -l* to standard input of more because pipe is implemented in the main memory.

Eg 2:- Use *sort* and *uniq* command to sort a file and print unique values.

```
sort file1 | uniq
```

This will sort the given file and print the unique values only.

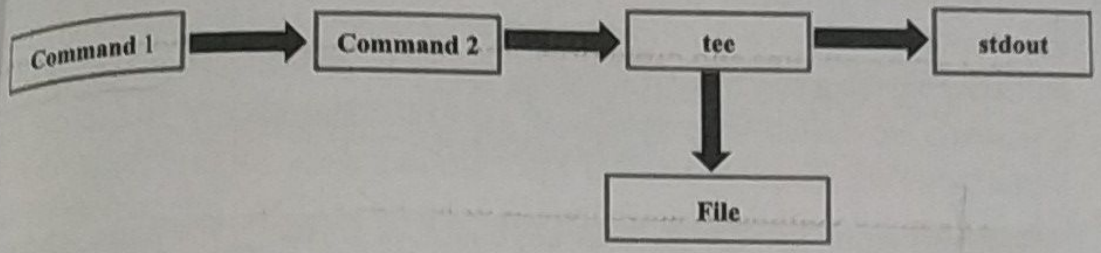
```
[salmiya@localhost ~]$ cat file1
orange
apple
mango
orange
apple
grape
[salmiya@localhost ~]$ sort file1 | uniq
apple
grape
mango
orange
[salmiya@localhost ~]$
```

tee command

The *tee* command reads the standard input and writes it to both the standard output and one or more files. It basically breaks the output of a program so that it can be both **displayed and saved** in a file. It does both the tasks simultaneously, copies the result into the specified files and also displays the result.

Syntax:

Command1 | Command2 | ... | tee <filename>



Example:

```
sort file1 | uniq | tee sortfile
```

This will sort the given file *file1* and print the unique values only and write the same information to the file *sortfile*.

```
ls –l | tee file1 file2 file3
```

This will print the output of *ls –l* and stores the output in file1, file2 and file3.

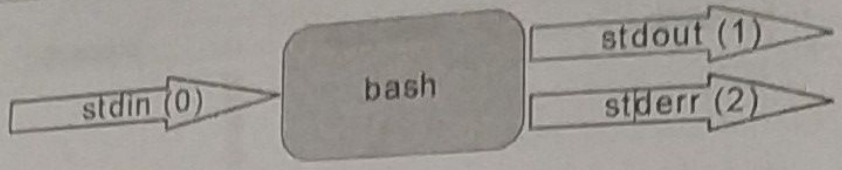
Redirecting Input/Output

Redirection is a feature in Linux such that when executing a command, you can **change the standard input/output devices**. The basic workflow of any Linux command is that it takes an input and gives an output.

There are three standard streams in I/O redirection:

- **standard input (stdin) :** The *stdin* stream is numbered as *stdin (0)*. The shell takes input from *stdin*. By default, keyboard is used as input.
- **standard output (stdout) :** The *stdout* stream is numbered as *stdout (1)*. The shell sends output to *stdout*. Output goes to display.
- **standard error (stderr) :** The *stderr* stream is numbered as *stderr (2)*. The shell sends error message to *stderr*. Error message goes to display.

Every Linux command takes input data from *stdin* and sends its normal output to *stdout* and error messages to *stderr*. These data streams are often called **standard input/output**. The drawing below has a graphical interpretation of these three streams.



Standard input, usually the user keyboard, is normally the place where a program reads its input from. Standard output, usually your terminal screen, is where the results of a command or program are displayed. In normal cases, standard error messages are also displayed on the terminal screen, but it is always possible to separate *stdout* from *stderr*. With redirection, the above standard input/output can be changed. The Linux shell can redirect any of these streams to a file, a device, or some other command, as required by the user.

Linux includes redirection commands for each stream. These commands write standard output to a file. If a non-existent file is targeted, a new file with that name will be created prior to writing. Commands with a single bracket overwrite the destination's existing contents and commands with a double bracket *do not* overwrite (append) the destination's existing contents.

Overwrite

- > - standard output
- < - standard input
- 2> - standard error

Append

- >> - standard output
- << - standard input
- 2>> - standard error

1. Output Redirection

Redirection of *stdout* is controlled by ">" the greater-than symbol. The process of redirecting output takes input from the keyboard but sends its output to a file. The syntax is:

Command-options-and-arguments > output-file

If a command line contains the output redirection symbol (>) followed by a file name, the standard output from the command will go to the specified file instead of to the terminal. If the file didn't exist before the command was invoked, then the file is automatically created. If the file did exist before the command was invoked, then the file will be overwritten; the command's output will completely replace the previous contents of the file.

If you want to append to a file instead of overwriting, you can use the output redirection append symbol (>>). This will also create the file if it didn't already exist.

Example 1:

```
cat sample
```

This command will display the content of sample file. To redirect the output of the cat command, use the following command.

```
cat sample > test
```

This *cat* command displayed nothing, as the output of the command is redirected to a file. If we check the contents of file test, it will contain the same text as sample (the output of the cat command).

Example 2:

```
ls –l > filelist
```

This command stores the result of ls –l to the file *filelist*.

```
[salmiya@localhost ~]$ cat sample
This is a file named sample
Showing output redirection
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample > test
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat test
This is a file named sample
Showing output redirection
[salmiya@localhost ~]$
```

2. Input Redirection

Linux commands can send output anywhere when using output redirection, and we can also get input from places other than the keyboard. We use the "less-than" symbol (<) for input redirection. Syntax is as follows:

Command-options-and-arguments < input-file

Example:

```
more <file1
```

This command sends the *file.txt* file to the *more* command

```
sort < file1
```

Here, *sort* command is used for sorting the content. Here the input to *sort* command is *file1*.

```
[salmiya@localhost ~]$ cat file1
Kerala
Tamilnadu
New Delhi
Assam
[salmiya@localhost ~]$
[salmiya@localhost ~]$ sort < file1
Assam
Kerala
New Delhi
Tamilnadu
[salmiya@localhost ~]$
```

3. Error Redirection

The *stderr* stream can be redirected in a similar fashion as *stdin* or *stdout*. There is no special symbol for redirecting *stderr*. The same ">" symbol is used but with the number 2 attached in front of it. We use "2>" for *stderr* redirection to tell the shell that we want to redirect the error messages instead of *stdout*. The syntax is:

Command-options-and-arguments 2> output-file

Example:- To display the content of the file named *myfile*. If *myfile* is not exists, the error message is normally displayed on our terminal. If you use error redirection (2>) symbol, nothing is displayed on the screen because the error message has been stored in the specified file (*errorfile*).

```
cat myfile 2> errorfile
```

```
[salmiya@localhost ~]$ cat myfile
cat: myfile: No such file or directory
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat myfile 2> errorfile
[salmiya@localhost ~]$ cat errorfile
cat: myfile: No such file or directory
[salmiya@localhost ~]$
```

Background processing

Most of the system processes run in background, while the user executes their processes in the foreground. If the user so desires even he can run his processes in the background. Using this facility, the user can run time consuming tasks like sorting a huge file in the background. This way he would not be required to wait till the sorting is over to be able to run the next process. He can immediately concentrate on another task the moment sorting process is submitted to run in the background.

There are several ways to place an active program in the background. One is to add an ampersand (&) to the end of a command line when you first run the command. While executing a command, if this symbol is placed at the end of the command then the command will be executed in background. When you run a process in the background a number (ie, PID of the process) is displayed on the screen. You will see a message on the screen when a background process is finished running.

Example:

```
cp sample samplecopy &
```

This command will copy the content of *sample* to *samplecopy* and is executed in background.

Switching Between Processes

If a foreground process is taking too much time, stop it by pressing Ctrl + Z. A stopped job still exists, but its execution is suspended. To resume the job, but in the background, type **bg** to send the stopped job to background execution. To resume a suspended process in the foreground, type **fg** and that process will take over the active session.

Job Control Commands

Job control commands enable you to place jobs in the foreground or background, and to start or stop jobs. The commonly used job control commands are:

- **Jobs**:- Lists all active jobs
- **bg %n**:- Places the current or specified job in the background, where n is the job ID
- **Fg %n**:- Brings the current or specified job into the foreground, where n is the job ID
- **Control-Z**:- Stops the foreground job and places it in the background as a stopped job

Process Scheduling

Any multitasking operating system must provide tools to permit scheduling of processes as per the user's or system's requirements. Linux can schedule processes to get executed within next few seconds to next few years. Once the user has submitted a process directing it to execute at a specified time and specified date in future. Linux manages to remember the processes to be executed and goes about executing them whenever the time arises without needing any further directions from the user. For this purpose, Linux use *cron* daemon.

itable record
, who had joined as an
ant coach in the pre-
season

here. I never think about the
nine matches," Jamil said on
the eve of the match. "This is
a very serious game and we

the tournament but slips in
the last two matches cost it
the League Winners Shield.
Coach Antonio Habas, who
the title twice with ATK

played so far. Our target
be winning the title and
have to put out best effort
realise it," Habas said.
ATKMB is likely to miss

Cron (stands for chronograph) is a daemon to run schedule tasks. *cron* wakes up every minute and checks schedule tasks in *crontab*.Crontab (CRON TABle) is a table which contains the list of tasks scheduled to run at regular time intervals on the system. When *cron* wakes up it checks whether any scheduled job is available for it to execute. If it is, it executes the job and goes back to sleep again, only to wake up the next minute to once again carry out the check. This cycle goes on till the system isn't shut down.

The commands for *cron* scheduling are: **at** and **batch**.

at:-

This command is used to execute the specified Linux command at **future date and time**. General format is:

at <time>

at> <commands>

^d (press Ctrl+d at the end)

Here *time* specifies the time at which the specified commands are to be executed.

Example:- at 12:30 PM

at> Echo "LUNCH BREAK"

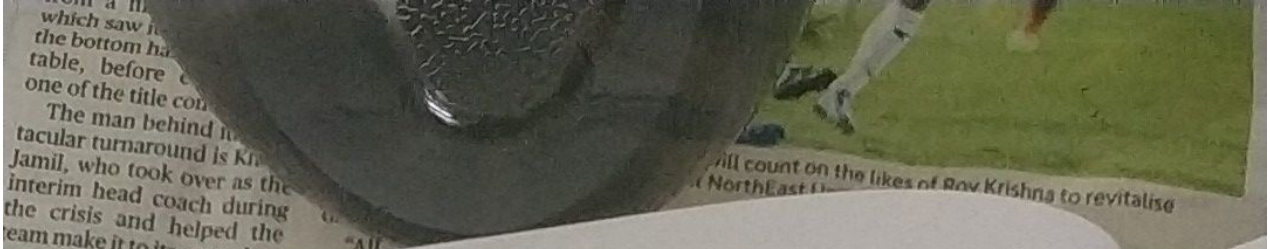
^d

at offers the keywords now, noon, midnight, today and tomorrow and they convey special meanings. It also offers the keywords minutes, hour, days, weeks, months and years, can be used with + operator.

at 4:00 PM tomorrow

at now + 2 days

at 3:00 AM Jan 20, 2020



Options

-l :- for listing the submitted jobs

-r :- for removing a submitted job

atq: You can use *atq* command (or *at -l*), to display all the *at* command jobs that are scheduled (pending jobs) or currently running. Syntax is:

atq

atrm:- You can use *atrm* command (or *at -d*), to delete a particular job. Syntax is:

atrm <jobid>

```
[salmiya@localhost ~]$ at 12:30 PM
at> echo "LUNCH BREAK"
at> <EOT>
job 7 at Sat Oct 12 12:30:00 2019
[salmiya@localhost ~]$
[salmiya@localhost ~]$ at 4:00 PM tomorrow
at> sort file1
at> <EOT>
job 8 at Sat Oct 12 16:00:00 2019
[salmiya@localhost ~]$
[salmiya@localhost ~]$ atq
7       Sat Oct 12 12:30:00 2019 a salmiya
8       Sat Oct 12 16:00:00 2019 a salmiya
[salmiya@localhost ~]$
```

batch:-

This command is used to execute the specified commands **when the system load is light** (when CPU becomes nearly free). General format is:

batch

<commands>

^d

Any job scheduled with *batch* also goes to the *at* queue.

Example: batch
sort file1
sort file2
cp sample samplecopy
^d

nohup Command

If a user wants a process that he has executed not to die even when he logged-out from the system, you can use this *nohup* (no hangup) command. Note that normally all the processes of a user are terminated if he logs out. General format is:

nohup <command> &

Example:- nohup sort file1&
1105

Here, 1105 is PID of the process. Then the command *sort file1* will be executed even if you logged out from the system. The output of this command will be stored in a file named *nohup.out*. The *nohup.out* file is always created in the current directory.

kill

This command is used to **terminate a process**. General format is:

kill [-signal number] <PID>

The PID is the process identification number of the process that we want to terminate. By default, this kill command uses the signal number 15 to terminate a process. But some programs cannot be terminated by this default signal. In this case, you can use the signal number 9, the 'sure kill' signal to forcefully terminate a process.

Example:- kill 120 (Terminate a process with PID 120)
kill -9 0

This command kills all the processes including login shell, ie, this command kills the Kernel, so all the processes are killed. (The PID of Kernel process is 0.)

ps (Process Status)

This command is used to **show which processes are running** in a system. General format is:

ps [options]

The *ps* command without arguments lists the running processes in the current shell. Result contains four columns of information. Where,

PID:- the unique process ID

TTY:- terminal type that the user is logged into

TIME:- amount of CPU in minutes and seconds that the process has been running

CMD:- name of the command that launched the process.

Option	Meaning
-A, -e	Viewing all the running processes
-T	select all processes on this terminal
-a	To view all processes with the exception of processes associated with the terminal and session leaders A session leader is a process that starts other processes
-u <user-name>	Lists the processes which are running for the specified user.
-t <terminal-name>	Lists the processes which are running on the specified terminal.
-x	Lists the system processes.

```
[salmiya@localhost ~]$ ps
  PID TTY          TIME CMD
 3042 pts/0    00:00:01 bash
 6418 pts/0    00:00:00 ps
[salmiya@localhost ~]$
```

who

Since Linux is a multiuser operating system, several users may work on this system. This command is used to display the **users who are logged on the system currently.** General format is:

who [options]

Option	Meaning
-b	The time of last system boot
-a	lists all available output for each user on your system.
-d -H	Display dead processes Print line of column headings
-l	Print system login processes
-q	Print all login names and number of users logged on
-s	Print only name, line, and time (default)
-t	Print the last time the system clock was changed, if the information is available.

The first column of the output represents the user names. The second column represents the corresponding terminal names and the remaining column represents the time at which the users are logged on.

who am i:- This command tells you who you are.

```
[salmiya@localhost ~]$ who
salmiya  :0           2019-10-13 12:34 (:0)
salmiya  pts/0        2019-10-13 12:35 (:0)
[salmiya@localhost ~]$
[salmiya@localhost ~]$ who am i
salmiya  pts/0        2019-10-13 12:35 (:0)
[salmiya@localhost ~]$
```

find

The find command helps in **locating files which meet the search criteria**. This command recursively examines the specified directory tree to look for files matching some file attributes, and then takes some specified action on those files. General format is:

find <path-list> <selection-criteria> <action>

It recursively examines all files in the directories specified in <path-list> and then matches each file for <selection-criteria> (file attributes). Finally, it takes the specified <action> on those selected files.

The <selection-criteria> may be as follows:

Selection Criteria	Meaning
–name <filename>	Selects the file specified in <filename>.
–user <username>	Selects file owned by <username>.
–type d	Selects directories.
–size +n or –n	Selects files that are greater than/ less than "n" blocks. Generally one block is 512 bytes
–mtime n or +n or –n	Selects files that have been modified on exactly n days/ more than n days/ less than n days.
-mmin n or +n or –n	Selects files that have been modified on exactly n minutes/ more than n minutes/ less than n minutes.
-atime n or +n or –n	Selects files that have been accessed on exactly n days/ more than n days/ less than n days.
-amin n or +n or –n	Selects files that have been accessed on exactly n minutes/ more than n minutes/ less than n minutes.
-empt	Returns true if a file is empty (contains nothing) and is either a regular file or a directory.

The <action> may be as follows:

Action	Meaning
-print	Displays the selected files on the screen.
-exec <command>	Execute the specified Linux command ends with {}\;
-delete	Delete files

If <path-list> and <action> are not specified, the *current directory* and *–print* are taken as default arguments.

Eg 1:-

```
find –name sample  –print
```

This command search the file sample from current directory and print it.

Eg 2:-

```
find –name [ab]* -print
```

This command finds all files begin with 'a' or 'b'and print them.

Eg 3:-

```
find /home –name *.txt -print
```

Find all the files under /home directory with *.txt* extension.

Eg 4:-

```
find  -empty -delete
```

This command finds all empty folders and files in the current directory or sub-directories and deletes them.

Eg 5:-

```
find –name sample –exec sort {} \;
```

This command searches the file sample from current directory and sort the file.

```
[salmiya@localhost ~]$ find -name sample -print
./sample
[salmiya@localhost ~]$
[salmiya@localhost ~]$ cat sample
New Delhi
Kerala
Tamil Nadu
Assam
Karnataka
[salmiya@localhost ~]$
[salmiya@localhost ~]$ find -name sample -exec sort {} \;
Assam
Karnataka
Kerala
New Delhi
Tamil Nadu
[salmiya@localhost ~]$
```

sort

This command is use to **sort the contents** of a given file based on ASCII values of characters. General format is:

sort [options] <filename>

Options	Meaning
-m <filelist>	Merges sorted files specified in <filelist>
-o <filename>	Stores output in the specified <filename>
-r	Sorts the content in reverse order
-u	Removes duplicate lines and display the sorted content
-n	Numeric-sort
-c	Checks if the file is sorted or not. If the file is sorted, no message will be displayed. Otherwise it will indicate the unsortedness.
-f	Ignore case
-b	Ignore leading spaces
+pos	Starts sort after skipping the posth field
-pos	Stops sorting after the posth field
-t "char"	Uses the specified "char" as delimiter to identify fileds.

Examples:

1. To sort the content of the file use:

```
sort file1
```

2. To sort the content in reverse order:

```
sort –r file1
```

3. To store the output to a file use :

```
sort –o sortfile  file1
```

This command not displays anything, but stores the result in *sortfile.*

4. To sort the file with numeric data:

```
sort –n number.txt
```

```
[salmiya@localhost ~]$ cat file1
Kerala
New Delhi
Tamil Nadu
Karnataka
Assam
[salmiya@localhost ~]$ sort file1
Assam
Karnataka
Kerala
New Delhi
Tamil Nadu
[salmiya@localhost ~]$ sort -r file1
Tamil Nadu
New Delhi
Kerala
Karnataka
Assam
[salmiya@localhost ~]$
```

```
[salmiya@localhost ~]$ cat number
25
10
5
15
30
20
[salmiya@localhost ~]$ sort -n number
5
10
15
20
25
30
[salmiya@localhost ~]$
```

touch

This command is used to **create empty files**. Touch does not allow you to store anything in a file. This command also allows you to **change the modification and access times of a file** into the specified time. ie, you can change these times without really accessing or modifying the file. General format is;

```
touch [options] <filename> [filename2...]
```

Options	Meaning
-a	Change only the access time
-c	If the file does not exist, do not create it
-m	Change only the modification time
-r	Ue the access and modification times of file
-t	Creates a file using a specified time

Examples:

1. To set the modification and access time to current time

```
touch sample
```

2. To set the access time to current time

```
touch –a sample
```

3. Sets the last modified and access time of all files into the current date and time.

```
touch *
```

file

This command lists the general classification of a specified file.ie, used to determine the **type of a file**. It lets you to know if the content of the specified file is ASCII text, C program text, data, separate executable, empty or others. General format is:

```
file [options] <filename>
```

Options	Meaning
-b	Display just file type in brief mode.
-i	Output mime type strings rather than the more traditional human readable ones.
-s	For special files

Examples:

1. To display just file type in brief mode.

```
file –b file1
```

2. To view mime type of file:

```
file –i file1
```

3. To displays the all files's file type:

```
file *
```

4. To display all files filetypes in particular directory.

```
file directoryname/*
```

```
[salmiya@localhost ~]$ file file1
file1: ASCII text
[salmiya@localhost ~]$ file -b file1
ASCII text
[salmiya@localhost ~]$ file -i file1
file1: text/plain; charset=us-ascii
[salmiya@localhost ~]$
```

File Processing Commands

There are various commands available for operating and manipulating files. Many of these file related commands are called filters. The commonly used commands are: wc, cut, paste, sort.

wc

This command is used to display the **number of lines, words, and characters** of information stored on the specified file. General format is:

```
wc [-options] <filename>
```

Options	Meaning
-l	Displays the number of lines in the file
-w	Displays the number of words in the file
-c	Displays the number of bytes in the file
-m	Displays the number of characters in the file

The *wc* command is capable of accepting input directly from the keyboard. By entering *wc* without any arguments, it waits for the user to type in the input. On terminating input (using Ctrl+d), the appropriate counts are displayed for the input that you supplied.

Without any option, the command displays total number of lines, words, and characters.

Examples:

1. wc sample
2. wc –l sample

When more than file name is specified in argument then command will display four-columnar output for all individual files plus one extra row displaying total number of lines, words and characters of all the files specified in argument, followed by keyword *total*.

```
wc file1 file2 file3
```

```
[salmiya@localhost ~]$ cat sample
This is a file named sample
Showing the use of wc command
[salmiya@localhost ~]$
[salmiya@localhost ~]$ wc sample
 2 12 58 sample
[salmiya@localhost ~]$
[salmiya@localhost ~]$ wc -l sample
2 sample
[salmiya@localhost ~]$
[salmiya@localhost ~]$ wc file1 file2 file3
 5  6 43 file1
 3  5 25 file2
 3  3 14 file3
11 14 82 total
[salmiya@localhost ~]$
```

cut

This command is used to **cut the columns/fields** of a specified file. ie, used for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by byte position, character and field. General format is:

```
cut [option] <filename>
```

Options	Meaning
-b	select only these bytes
-c	select only these characters
-f	select only these fields
-d DELIM	Use character DELIM instead of a tab for the field delimiter

Note: Without any option specified, the cut command displays an error displays an error.

Examples:

1. To display the first, second and fifth bytes:

```
cut –b 1,2,5sample
```

2. To display bytes in the range:

```
cut –b 1-3,6-9 sample
```

3. To print the second, fifth and seventh character from each line of the file.

```
cut –c 2,5,7 sample
```

4. To print the first field of the file:

```
cut –d " " –f 1sample
```

```
[salmiya@localhost ~]$ cat sample
Kerala
Karnataka
New Delhi
Tamilnadu
Uttar Pradesh
[salmiya@localhost ~]$ cut -b 1,2,5 sample
Kel
Kaa
NeD
Tal
Utr
[salmiya@localhost ~]$ cut -d " " -f 1 sample
Kerala
Karnataka
New
Tamilnadu
Uttar
[salmiya@localhost ~]$
```

paste

This command **concatenates the contents of the specified files** into a single file vertically. It is used to **merge lines of files**. General format is:

paste <filename1><filename2>...

Options	Meaning
-s	paste one file at a time instead of in parallel(merge in sequential manner)
-d DELIM	Use character DELIM as delimiter instead of a tab

Without any option *paste* merges the files in parallel. The paste command writes corresponding lines from the files with tab as a delimiter on the terminal.

Examples:

1. Let us consider two files having name state and capital. state and capital file contains 5 names of the Indian states and capitals respectively. To merge these files use:

```
paste state capital
```

2. To merge the files in sequential manner:

```
paste –s state capital
```

3. To use : as delimiter

```
paste –d ":" state capital
```

```
[salmiya@localhost ~]$ cat state
Kerala
Tamilnadu
Assam
Karnataka
[salmiya@localhost ~]$ cat capital
Trivandrum
Chennai
Dispur
Bengaluru
[salmiya@localhost ~]$ paste state capital
Kerala  Trivandrum
Tamilnadu       Chennai
Assam   Dispur
Karnataka       Bengaluru
[salmiya@localhost ~]$ paste -s state capital
Kerala  Tamilnadu       Assam   Karnataka
Trivandrum      Chennai Dispu   Bengaluru
[salmiya@localhost ~]$

[salmiya@localhost ~]$ paste -d ":" state capital
Kerala:Trivandrum
Tamilnadu:Chennai
Assam:Dispur
Karnataka:Bengaluru
[salmiya@localhost ~]$
```

Mathematical commands

There are different commands are available to perform mathematical operations.

expr

This command is used to perform **arithmetic operations on integers**. It can't handle floating point arithmetic. General format is:

expr expression

The arithmetic operators and their corresponding functions are given below.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder of divisio

A white space must be used on either side of an operand. Multiplication operator should be preceded by \ symbol because in Linux * represents all files in the current directory. If we directly use * in *expr*, we will get error message.

Examples:

```
[salmiya@localhost ~]$ expr 10 + 5
15
[salmiya@localhost ~]$
[salmiya@localhost ~]$ expr 10 - 5
5
[salmiya@localhost ~]$
[salmiya@localhost ~]$ expr 10 \* 5
50
[salmiya@localhost ~]$
[salmiya@localhost ~]$ expr 10 / 5
2
[salmiya@localhost ~]$
```

bc

bc (Basic Calculator) is a command-line utility that provides all features you expect from a simple scientific or financial calculator. It is used to perform **arithmetic operations on integers as well as on floating point numbers.**

Type the arithmetic expression in line and press *Enter* key. Then the answer will be displayed on the next line. After you have finished your work, press Ctrl+d to end up.

By default, *bc* performs integer division. If you require float division, then set *scale* to the number of digits of precision before the operation.

```
[salmiya@localhost ~]$ bc
10 + 5.5
15.5

8 / 3
2

scale=1
8 / 3
2.6

[salmiya@localhost ~]$
```

factor

This command is used to print the **prime factors** of a number. General format is:

factor [number]

Note: The number is given from command line or read from standard input. If the number is given through standard input it waits for another number. It exits if you press Ctrl+d.

Example:

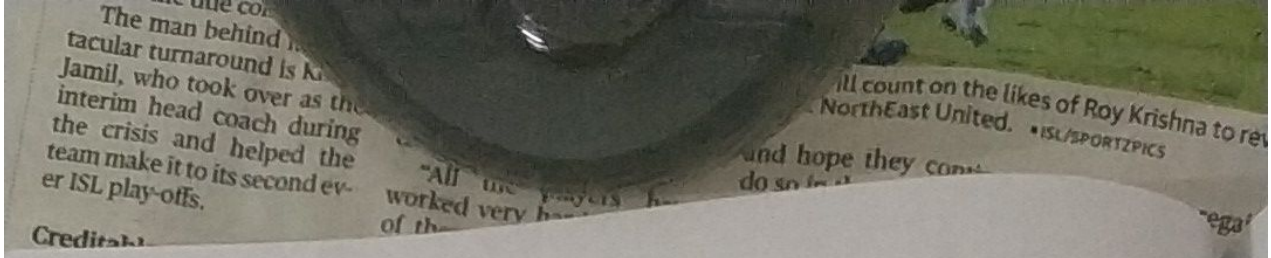
```
[salmiya@localhost ~]$ factor

30
30: 2 3 5

25
25: 5 5

100
100: 2 2 5 5

[salmiya@localhost ~]$ factor 40
40: 2 2 2 5
[salmiya@localhost ~]$
```



ill count on the likes of Roy Krishna to re
NorthEast United. •ISL/SPORTZPICS

The man behind
tacular turnaround is K
Jamil, who took over as th
interim head coach during
the crisis and helped the
team make it to its second ev-
er ISL play-offs.

Creditab

"All
worked very
of th

and hope they com
do so

ega

Creating and editing files with vi editor

A *vi* is a screen-oriented text editor, which is the most widely used editor on Linux and Unix. The *vi* editor (visual full screen editor) is available with every distribution of the Linux operating system. *William Joy* developed *vi* at the University of California at Berkeley. It is a screen-oriented interactive editor. The contents of the file will be displayed to your screen, and as you make changes to the file, they are immediately displayed on the screen. This editor is capable of handling multiple files simultaneously. All open files are called *buffers* in vi terminology. You can cut, copy, and paste text, search and replace text, export and import text to and from other files, and spell check. In addition to the use of *vi* for general file editing, it is also used for typing email and editing the command line.

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

There are mainly three modes used in the vi editor:

» Command mode
» Insert or Input mode
» Line mode

1 Command mode:

vi is a command-driven editor. When you start a *vi* edit session, you are in command mode. Therefore, if you type any keys, *vi* will try to execute the associated commands. Almost every key on the keyboard is assigned to some *vi* function. Commands are available to input text, move the cursor, modify text, delete text, and paste text. Generally, *vi* commands are silent, which means that as you enter *vi* commands they will not be echoed to the screen. You will only see their effects. In this mode, whatever you type is interpreted as a command. It is the default mode when we start up *vi* editor.

2 Insert or Input mode: (normal alphabets)

This mode enables you to **insert text** into the file. Everything that's typed in this mode is interpreted as input and placed in the file. To return to the command mode just press the *Esc* key.

3 Line Mode or ex mode: executable mode

The *ex* mode commands can be entered at the last line of the screen. We can enter into this mode from command mode by pressing [:] key. *ex* commands are entered at the colon prompt, and are echoed to the screen and are submitted by entering a Return .

Note: *vi* always starts in the command mode. To enter text, you must be in the insert mode for which simply type **'i'**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

Starting and stopping the vi Editor

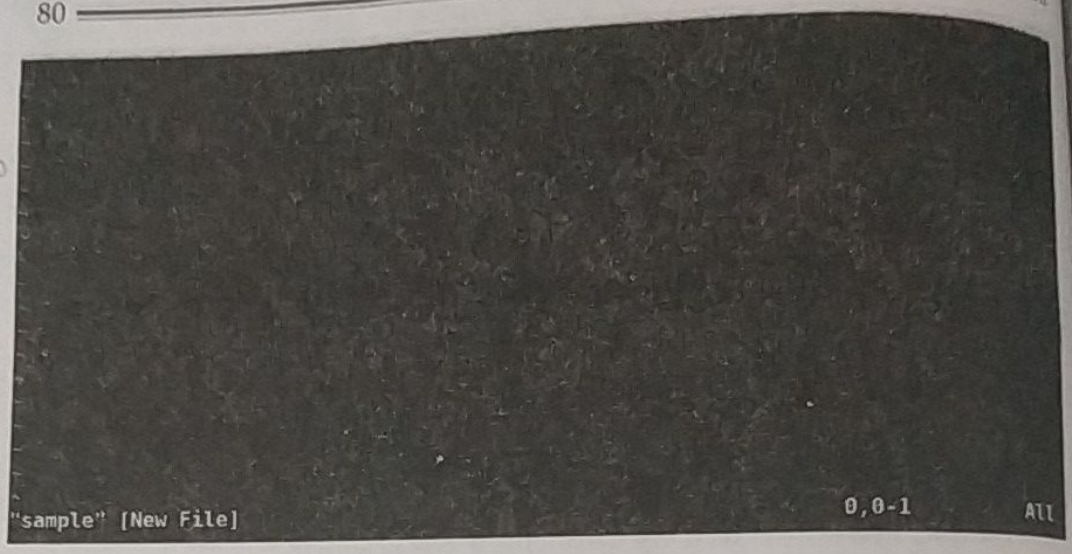
The editor is started when you use a *vi* command and give a file name as its argument. To create a new file use:

vi <filename>

Eg: if you want to create a new file with the name 'samplefile', you will use :

vi samplefile

This command starts *vi* and allocates a memory buffer for file 'samplefile'. If the file already exists, text in the file will be displayed on the screen. The bottom line, which is used as a status line, will display a file name, line number, and number of characters in the file. Remember that you can't add any text to the file until you go into insert mode because *vi* started in the command mode.



The tilde character (~) you see in the screen shows that the file has no text in these lines. To switch to insert mode, you press the **i** character just after starting *vi*. When you have finished entering text into the file, you press the **Esc** and **:** key to return to command mode. To save the file, you can use the **:w** command. After you save the file, you can quit the editor with the **:q** command. You can also use **:x** or **:wq** to save and quit in one step.

If you want to specify any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called 'filename2', you would type ':w filename2' and return.

Multiple files can be opened with *vi* by supplying multiple file names at the command line. If you want to open three files (file1, file2, file3) simultaneously, the command will be as follows:

```
vi file1 file2 file3
```

Cursor Movement in vi Editor

To move the cursor, you must be in command mode. On most of the modern terminals, you can move the cursor with the arrow keys on your keyboard.

Symbol	Function
l	Move one character right
h	Move one character left
j	Move one line down
k	Move one line up
< space >	Move one character right
G	Go to last line of the file
<n>G	Go to line number n in the file
$	Go to end of current line
^	Go to start of line
w	Go to beginning of next word
b	Go to beginning of previous word
e	Move to end of word
H	Go to first line of screen
M	Go to middle line of screen
L	Go to last line of screen
(	Go to beginning of sentence
)	Go to end of sentence
{	Go to beginning of paragraph
}	Go to end of paragraph

Before using any of these commands, make sure that you are in the command mode by pressing the ESC & : key. You can also instruct the vi editor to display the line number with each line using the *:set number* command.

Inserting and Deleting Text

Text insertion takes place only when you are in insert mode. Text deletion tasks are performed in command mode.

List of text entry commands

Command	Effect
i	Start inserting text at the current cursor location
I	Start inserting text at the beginning of the current line
a	Appends after cursor
A	Appends at the end of the current line
o	Insert a blank line just below the current line and start inserting text from the beginning of that line
O	Insert a blank line just above the current line and start inserting text from the beginning of that line

List of text delete commands

Command	Effect
x	Delete character at current cursor location.
<n>x	Delete n characters starting at current cursor location.
X	Deletes previous character from the current cursor location.
<n>X	Delete n previous characters from the current cursor location.
dd	Delete current line.
db	Delete previous word.
dw	Delete from current cursor location to the end of word.
dG	Delete to the end of file including current line.

Replacing Text

In addition to inserting and deleting text, you can also replace existing text with new text. You can replace a single character or many lines of the text. The **r** command is used to replace a single character. When you press **r** while in command mode, nothing happens, but as soon as you press the next character, it appears on the screen replacing the character at the current cursor location.

Command	Effect
r	Replace current character remaining in command mode
R	Replace multiple characters until the Esc key is pressed
s	Replace current character and go to insert mode
S	Replaces the entire line
cw	Change to the beginning of next word
cc	Removes the contents of the line, leaving you in insert mode.
cG	Change to the end of file

Search and Replace

Search and replace is a necessary editing feature found in all good editors. If you want to find a text pattern in vi, you can use the / text command, where *text* is the string you want to search. This command searches the text in the forward direction from the current cursor position. You can search in the backward direction, use **?text** command. To repeat the search once you find a string, just use / or ? without typing the string again. You can also replace text after a search is made.

Command	Effect
/text	Search text in forward direction starting from current cursor location
?text	Search text in backward direction starting from current cursor location
/	Repeat previous search in forward direction
?	Repeat previous search in backward direction
n	Repeat search in the same direction
N	Repeat search in the opposite direction

:s/oldtext/newtext	Search *oldtext* in the forward direction and replace it with *newtext*
:s/oldtext/newtext/g	Search *oldtext* in the entire file and replace it with *newtext*
:m,n s/oldtext/newtext	Search *oldtext* in the forward direction from line m to line n and replace it with *newtext*
/ text	If you put a space between the / and the text to be searched, only whole words are searched
/^text	Search text only in the beginning of a line
/text$	Search text only in the end of a line
/(More than one word)	Use parenthesis to search multiple words

Cut, Copy and Paste

You can copy lines or words from one place and then you can paste them at another place. The *dd* and *dw* commands cut the text and put it on a cut buffer. Text from the cut buffer can be pasted any place using the **p** command.

Command	Effect
yy	Copy or yank current line
<n>yy	Copy n lines starting from current line position
p	Paste yanked text after the current cursor position
P	Paste yanked text before the current cursor position
:m <a>	Move current line and paste after line number a
:<a>,<b> m <c>	Move lines from a to b and paste after line number c
:<a>, <b> t <c>	Copy lines from a to b and paste after line number c

Set Commands

You can change the look and feel of your vi screen using the following set commands. Once you are in the command mode, type **:set** followed by any of the following commands.

Command	Effect
:set number	Sets line number in front of each line in Vi
:set all	Lists all available options
:set autoindent	The next line is indented the same number of character as the current line
:set readonly	Sets the current file as read-only. No change is saved
:set wrapmargin = n	Sets the right wrap margin equal to n. If we are using 80-column display and the wrap margin is set to 6, every line will be wrapped to the next line after 74 characters
:set showmode	Displays the currently working mode

Running Commands

The *vi* has the capability to run commands from within the editor. To run a command, you only need to go to the command mode and type **:! <command>**. It executes the specified command without exiting from *vi*.

REVIEW QUESTIONS

Part A

1. Define process in Linux.
2. What do you mean by foreground processing?
3. What do you mean by background processing?
4. What are the different process states?
5. What are the different types of processes?
6. What is redirection in Linux?
7. What is the use of pipe symbol?
8. What is the use of at command?
9. What is batch command?
10. What is the use of kill command?
11. What is nohup command?
12. What is ps command?
13. Which command is used to search a particular file?
14. What is file command?
15. What is paste command?
16. What is cut command?
17. What is touch command?
18. What are the different modes of vi editor?

Part B

19. What is Linux redirection? Explain the different types of redirection with examples.
20. List out the usage of find command in locating files and directories
21. What are the commands used for process scheduling?
22. Write note on connecting process with pipes.
23. What is the use of sort command?
24. What are the different file processing commands?
25. Explain different mathematical commands in Linux.
26. What are the different text replacement commands in vi editor?

Part C

27. What is Linux redirection? Explain the different types of redirection with examples.
28. Explain creating and editing files with vi editor.