**Unit 3 : Getting Started with Unix, Unix Shell Command**

Part : 1

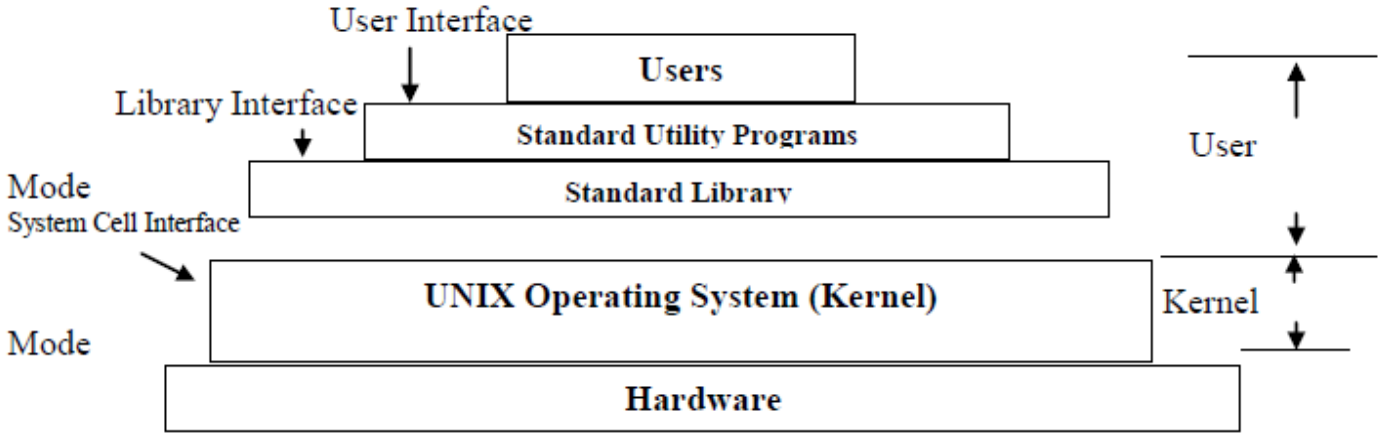
- Unix Architecture
- Unix Features
- Types Of Shell ( C, Bourn, Korn )
- Unix File System
- Types Of Files
  - O Ordinary Files
  - o Directory Files
  - o Device Files
- Unix File & Directory Permissions

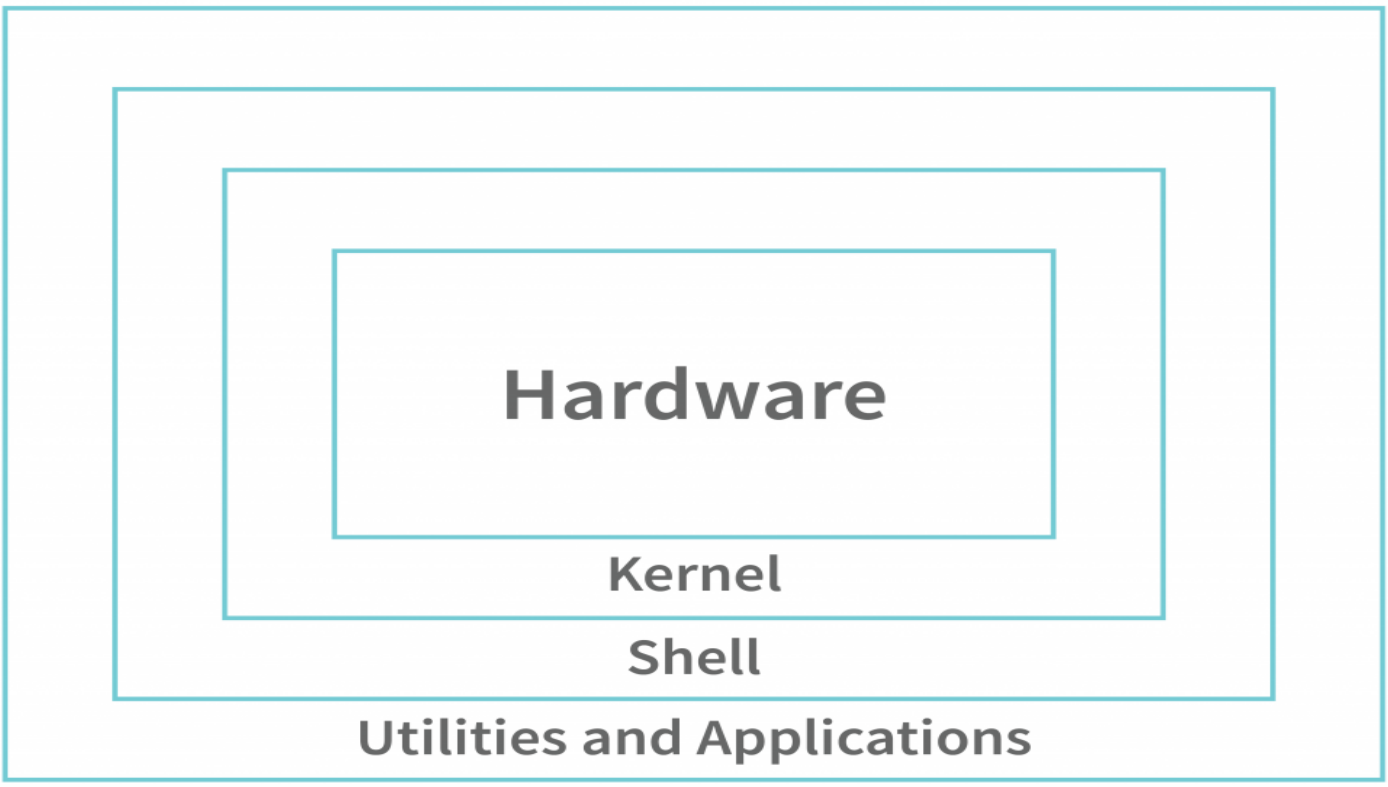
## UNIX operating system :

In 1970, Ken Thompson and Dennis Ritchie at AT&T Bell labs developed UNIX operating system. Earlier this system was known as UNICS (UnIplexed Information and Computer Service), but later it became famous as a UNIX. In 1973, it was re-written in „C" programming language (higher-level language). This made UNIX the world"s first portable operating system, capable of being easily ported (moved) to any hardware. UNIX became easy to understand and modifiable. This was a major advantage for UNIX. This step led to the wide acceptance of UNIX among users at that time. Today"s tremendous popularity of UNIX is also responsible to this step.


## Architecture of UNIX:

The various layers depicted in UNIX architecture are shown in below figure and also described below.




Unix Architecture Layers

**1. Hardware**

- The bottom layer is the hardware.
- It consists of various physical devices such as CPU, memory, disks, monitor, printer, etc…
- These devices provide various services. For example, printers are used for printout purposes.

**2. UNIX Operating System (Kernel):**

- The next higher layer is the UNIX operating system.
- It is also called system kernel, or simply, kernel. (It is described next).

- It manages all the underlying hardware.
- It directly interacts with the hardware and provides user programs required services.
- It hides the complex details of hardware also.
- In short, it provides the simple interface between user programs and hardware.
- The main services of an operating system includes process management, memory management, file system management, I/O management etc.

**3. Standard Library:**
- Above operating system, next layer is for standard library.
- It contains a set of procedures, one procedure per system cell.
- These procedures are written in assembly language and used to invoke various system calls from user programs.

**4. Standard Utility Programs:**
- In addition to operating system and system call library, all versions of UNIX supply a large number of standard programs.
- Such programs include command processor (Shell), compilers, editors, text processing programs, file manipulation utilities, a variety of commands and so on.

Such programs make the user tasks simpler. Users interact with them and they, in turn, interact with the operating system to get services from operating system.

**5. Users:**
- The top most layer is of users.
- User programs come in this layer. They interact with the system either by using library procedures to invoke system calls, or by using utility-program such as shell.

Here, some new terms came in picture such as kernel, shell, and system call. These are described below.

**Kernel:**
- The kernel is the core of the UNIX operating system.
- UNIX uses microkernel approach, where code from the kernel is moved up in higher layers to keep it as small (thin) as possible.
- Kernel is a program, which is loaded in memory when system is turned on. It stays there and provides various services until the system is turned off.
- Kernel interacts with the hardware directly. When user program needs to use any hardware, it has to use services provided by the kernel. Special functions, called system calls, are used to request kernel. Kernel performs the job on behalf of the user process.
- In addition to providing services to user programs, kernel also provides other services like process management, memory management, file system management and so on.
- In short, kernel manages entire computer system.

**Shell:**

- The shell is an interface between the user program and the kernel.
- When user logs-in to the system, process for shell starts execution. It terminates when user logs-out from the system.
- Users can directly interact with the shell.
- It works as a command interpreter. It accepts commands from user and translates them into the form, which the kernel can understand easily.
- It is also a programming language. It provides various programming functionalities such as looping, branching and so on.

**System Call:**

- System calls are special functions.
- They are used to request kernel to provide various services, such as reading from a file stored on hard disk.
- They can be invoked via library procedures, or via command provided by shell, or even directly from C programs in UNIX.
- System calls are similar to user-defined functions. Difference is that they execute in the kernel mode, having fully access to all the hardware; while user defined functions execute in user mode, having no direct access to the hardware.
- Various flavors of UNIX have one thing in common. They all use the same set of system calls provided by POSIX standard. If any operating system is using other system calls, then it will not be a UNIX operating system.

## Features of UNIX:

Many operating systems are there in market, all operating systems having their own functionality which operating system user should chose that is depends on the task. As being an operating system, UNIX contains all the features that any operating system should have, however, in addition to such common features, UNIX contains some special features, which makes it so much popular. . These features are as given below:

**1. Files and Process:**

- In UNIX, every thing is either a file or a process.
- A file is a collection of data. They are used to store large amount of information permanently. Directories are considered as files. In addition, physical devices are also considered as files in UNIX.
- A process is a program in execution. All commands execute as processes and perform their tasks.

**2. A Multi-User System:** -UNIX is a multi user system. (Dos is a single user system.) - This means that it allows multiple users to work simultaneously on the same system.

- Different users can login from different machines into the same machine by using programs like „TELNET".

**3. Multi Tasking System.**

- UNIX is a multi-tasking system too. (Dos is a single tasking system)
- It allows multiple program to run simultaneously.
- Among simultaneously running processes, one process will be foreground process. User can interact with this process directly. While other processes will be background processes. They execute in background without requiring user interaction.

**4. The Building Block Approach:**

- UNIX uses the building block approach to perform complex tasks.
- It provides a few hundred commands each of which can perform one simple job to perform complex tasks, such simple commands can be combined using pipes and filters. Thus, the small is beautiful philosophy is implemented here.

**5. Portability**

- UNIX is written in high-level language, rather than low level assembly language
- This makes it easy to read, understand, change and move to other machine.
- Thus, it provides protability.

**6. Consistent format for files**

- UNIX does not provide special formats for different file types.
- It treats all files same, and provides consistent format for files in for of simple byte streams.
- It is the responsibility of the application programs to interpret file contents as per requirements.

**7. Pattern Matching**

- UNIX provides very sophisticated pattern matching capabilities.
- There is a set of special character like as „*" that can be used in pattern matching.

**8. Programming facility:**

- The UNIX shell is also a programming language.
- It supports all the programming features such as variables, control structures, loops so on.

These features can be used to develop shell programs, called shell scripts. Such programs can be used to control and automate many of the system"s functions.

**9. On-line help**

- UNIX provides an on-line help facility for all the commands
- For this purpose, it provides a command named „man". By using this command, user can have an instant help on any command.

- **Types of Shell:**

The kernel program is usually stored in a file called „UNIX‟ whereas the shell program is un a file called ‘sh’. For each user working with UNIX at any time different shell programs are running. Thus, at a particular point in time there may be several shells running in memory but only one kernel. This is because; at any instance UNIX is capable of executing only one program as the other programs waits for their turn. And since it‟s the kernel which executes the program one kernel is sufficient. However, different users at different terminals are trying to seek kernel‟s attention. And since the user interacts with the kernel through the shell different shells are necessary.

Different people implemented the interpreter function of the shell in different ways. This gave rise to various types f shells, the most prominent of which are outlined below:

**There are 3 types of shells in UNIX:**
1) Bash
2) Korn shell
3) C shell

**Bourne Shell (Bash)**
Among all, Steve Bourne‟s creation, known after him as the Bourne Shell, is the most popular. Probably that‟s why it is bundled with every UNIX system. Or perhaps it is the other way round. Because it was bundled with every system it became popular. Whatever the cause and the effect, the fact remains that this is the shell used by many UNIX users.

**C Shell (Corn)**
This shell is a hit with those who are seriously into UNIX programming. It was created by Bill Joy, then pursuing his graduation at the University of California at Berkeley. It has two advantages over the Bourne Shell.

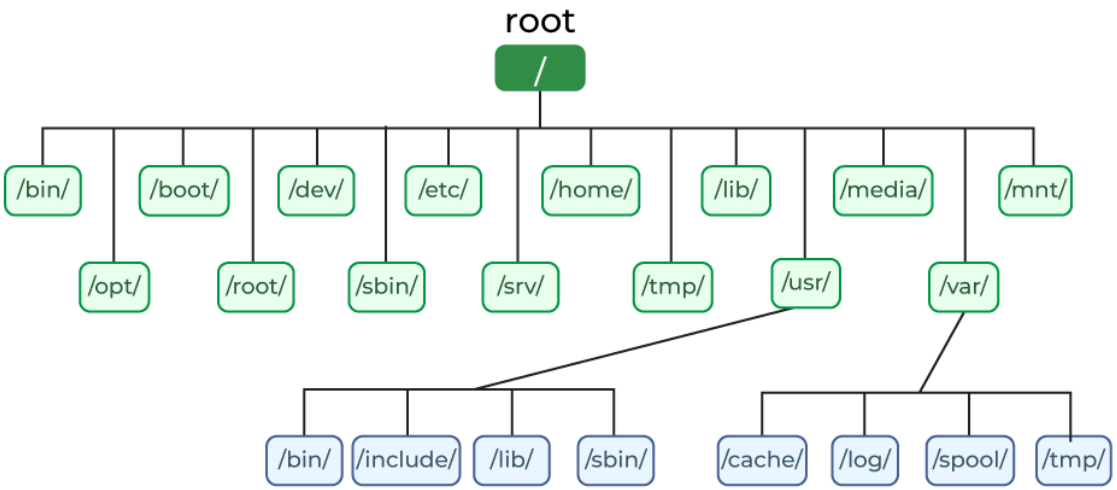
First, it allows aliasing of commands. That is, you can decide what name you want to call a command by. This proves very useful when lengthy commands which are used time and again are renamed by you. Instead of typing the entire command you can simply use the short alias at the command line.

If you want to save even more on the typing work, C shell has command history feature. This is the second benefit that comes with C Shell. Previously typed commands can be recalled, since the C shell keeps track of all commands issued at the command line. This feature is similar to the one provided by the program DOSKEY in MS-DOS environment.

**Korn Shell (Ksh) :**

If there was any doubt about the cause-effect relationship of the popularity of Bourne Shell and it's inclusion in every package, this adds fuel to it. The not-so-widely-used Korn Shell is very powerful, and is a superset of Bourne Shell. It offers a lot more capabilities and is decidedly more efficient than the other. It was designed to be so by David Korn of AT & T‟s Bell Labs.

## Unix File System:




Unix File System is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage. A file is the smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.

The Unix file system is a hierarchical file system used by Unix-based operating systems to store and organize files and directories. It is a tree-like structure that starts with a single directory called the root directory, which is denoted by a forward slash (/) character.

The Unix file system uses a directory hierarchy that allows for easy navigation and organization of files. Directories can contain both files and other directories, and each file or directory has a unique name.

Unix file system also uses a set of permissions to control access to files and directories. Each file and directory has an owner and a group associated with it, and permissions can be set to allow or restrict access to these entities.

**Directories or Files and their Description**

| NAME | DESCRIPTION |
|---|---|
| / | The slash / character alone denotes the root of the filesystem tree. |
| /bin | Stands for “binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users. |
| /boot | Contains all the files that are required for successful booting process. |
| /dev | Stands for “devices”. Contains file representations of peripheral devices and pseudo-devices. |
| /etc | Contains system-wide configuration files and system databases. Originally also contained “dangerous maintenance utilities” such as init,but these have typically been moved to /sbin or elsewhere. |
| /home | Contains the home directories for the users. |
| /lib | Contains system libraries, and some critical files such as kernel modules or device drivers. |
| /media | Default mount point for removable devices, such as USB sticks, media players, etc. |
| /mnt | Stands for “mount”. Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on. |
| /proc | procfs virtual filesystem showing information about processes as files. |

| NAME | DESCRIPTION |
|---|---|
| **/root** | The home directory for the superuser “root” – that is, the system administrator. This account’s home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted. |
| **/tmp** | A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time. |
| **/usr** | Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person). |
| **/usr/bin** | This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc. |
| **/usr/include** | Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language. |
| **/usr/lib** | Stores the required libraries and data files for programs stored within /usr or elsewhere. |
| **/var** | A short for “variable.” A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files. |
| **/var/log** | Contains system log files. |
| **/var/mail** | The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail. |
| **/var/spool** | Spool directory. Contains print jobs, mail spools and other queued tasks. |

| NAME | DESCRIPTION |
|---|---|
| /var/tmp | A place for temporary files which should be preserved between system reboots. |

## Types Of Files

### 1. Ordinary Files

- An ordinary file is a file on the system that contains data, text, or program instructions.
- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.
- Do not contain other files.
- In long-format output of ls -l, this type of file is specified by the "-" symbol.
- **This is the most common file type in UNIX.** All programs you write belong to this type. An ordinary file itself can be divided into to types.

**(A) Text File:** A text file contains only printable characters, and you can often view the contents and make sense out of them. All C and Java program sources, shell and perl scripts are text files. A text file contains lines of characters where every line is terminated with the new line character, also known as line feed. When you press enter while inserting text, the LF character is appended to every line. You won‟t see this character normally, but there is a command which can make it visible.

**(B) Binary File:** On the other hand, contains both printable and nonprintable characters that cover the entire ASCII range 0 – 255. Most UNIX commands are binary files, and the object code and executables that you produce by compiling C programs are also binary files. Picture, sound and video files are binary files as well. Displaying such files with a simple eat command produces unreadable output and may even disturb your terminal‟s settings.

### 2. Directory File:

Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components. (1) The Filename (2) A unique identification number for the file or directory (called the inode number)

- Branching points in the hierarchical tree.
- Used to organize groups of files.
- May contain ordinary files, special files or other directories.
- Never contain “real” information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, ( named / ) located at the top of the tree.
- In long-format output of ls –l , this type of file is specified by the “d” symbol.

**A directory contains no data, but keeps some details of the files and subdirectories that it contains.** The UNIX file system is organized with a number of directories and subdirectories, and you can also create them as and when you need. You often require to do that to group a set of files pertaining to a specific application. This allows two or more files in separate directories to have the same filename. A directory file contains an entry for every file and subdirectory that it houses. If you have 20 files in a directory, there will be 20 entries in the directory. Each entry has two components. (i) **The file name.** (ii) **A unique identification number for the file or directory.**

If directory contains an entry for a file named abc, we commonly say that the directory contains the file abc. Though we shall often be using the phrase “contains the file” rather than “contains the file name”, you must not interpret the statement literally. A directory contains the file name and not the file‟s contents.

### 3. Device File:

**You will also be printing files, installing software from CD-ROM r backing files t tape. All these activities are performed by reading or writing the file representing the device. For instance, you print a file by writing the file representing the printer.** When you restore files from tape, you read the file associated with the tape drive. The kernel takes care of this “reflection” by mapping these special files to their respective devices. It is advantageous to treat devices as files as some of the commands used to access an ordinary file also work with device files.

**Device file names are generally found inside a single directory structure, ./dev.** A device file is indeed special; it‟s not really a stream of character. In fact, it doesn‟t contain anything at all. You will soon learn that every file has some attributes that are not stored in the file but elsewhere

on the disk. It‟s the attributes f a device file that entirely govern the operation of the device. The kernel identifies a device from its attributes and then uses them to operate the device.

## Unix File & Directory Permissions

### File permission:

Before we take up file permission, you need to understand the significance of file ownership. When you create a file, your username shows up in the third column of the file‟s listing; you are the owner of the file. Your group name is seen in the fourth column; your group is the group owner of the file. If you copy someone else‟s file, you are the owner of the copy. If you can‟t create files in other users‟ home directories, it‟s because those directories are not owned by you. UNIX has a simple and well-defined system of assigning permissions to files. Issue the ls –l command once again to view the permissions of a few files.

| Categories of user | Types Of Permission |
|---|---|
| User u | Read r |
| Group g | Write w |
| Other o | Execute x |

**$ls –l**

| | | | | | |
|---|---|---|---|---|---|
| -rwxr-xr-- | 1 | kumar metal | 20500 | may 10 19:21 | chap02 |
| -rwxr-xr-x | 1 | kumar metal | 890 | Jan 29 23:17 | dateval.sh |
| -rw-rw-rw- | 1 | kumar metal | 84 | Feb 12 12:30 | dept.lst |

Observe the first column that represents the file permissions. These permissions are also different for the three files. UNIX follows a three-tired file protection system that determines a file‟s access right. To understand how this system works, let‟s break up the permissions string of the file chap02 into three groups. The initial – (in the first column) represents an ordinary file and is left of the permissions string:

Each group here represents a category and contains three slots, representing the read, write and execute permission of the file – in that order. r indicates read permission, which means cat can display the file. w indicates write permission; you can edit such a file with an editor. x indicates execute permission; the file can be executed as a program. The – shows the absence of the corresponding permission.

**The first group (rwx) has all tree permission. The file is readable, writable and executable by the owner of the file, kumar.** But do we know who the owner is? Yes we do. The third column shows kumar as the owner and the first permissions group applies to kumar. You have to log in with the username kumar for these privileges to aplly to you.

The second group (r-x) has a hyphen in the middle slot, which indicates the absence of write permission by the group owner of the file. This group owner is metal, and all users belonging to the metal group have read and execute permissions only.

The third group (r--) has the write and execute bits absent. This set of permissions is applicable to others, i.e., those who are neither the owner kumar nor belong to the metal group. This category (others) is often referred to as the world. This file is not world-writable.

You can set different permissions for the three categories of users – owner, group and others. It‟s important that you understand them because a little learning here can be a dangerous thing. A faulty file permission is a sure recipe for disaster.

**(B) Directory Permission:**

**Directory also has their own permissions and the significance of these permissions differ a great deal from those of ordinary files.** You may not have expected this, but be aware that read and write access to an ordinary file are also influenced by the permission of the directory housing them. It‟s possible that a file can‟t be accessed even though it has read permission, and can be removed even when it‟ write-protected. In fact, it‟s very easy to make it behave that way.

If the default directory permission are not altered, the chmod theory still applies. However, if they are changed, unusual things can happen. Though directory permissions are taken up later, it‟s worthwhile to know what the default permissions are on your system:

| Categories of user | | Types Of Permission | |
|---|---|---|---|
| User | u | Read | r |
| Group | g | Write | w |
| Other | o | Execute | x |

```
$mkdir c_progs

$ls –ld c_progs

drwxr-xr-x   2   kumar metal  512   may   9    09:57  c_progs
```

First character d displays that c_progs is directory, second, third and fourth character rwx is showing that user is having all permission read, write and execute, fifth, sixth and seventh character r-x showing that group is having only read and execute permission, eighth, ninth and tenth character r-x showing that others having read and execute permission.

The default permissions of a directory on this system are rwxr-xr-x (or 755); that is what they should be. A directory must never be writable by group and others. If you find that your files are being tampered with even though they appear to be protected, check up the directory permissions.

Part: 2

- Connecting Unix Shell : Telnet
- Login Commands

passwd, logout, who, who am i, clear,uname

- File / Directory Related Command

ls, cat, cd, pwd, mv, cp, ln, rm, rmdir, mkdir, chmod,

chown, chgrp, find, more, less, head, tail, wc, touch, stat,

alias, type

- Operators in Redirection & Piping

<, >, <<, >>, |

- Finding Patterns in Files

grep, fgrep, egrep

- Working with columns and fields

cut, paste, join

- Tools for sorting :sort, uniq
- Comparing files : cmp, comm, diff
- Changing Information in Files: tr, sed
- Examining File Contents : od
- Tools for mathematical calculations: bc, factor
- Monitoring Input and Output :tee, script
- Tools For Displaying Date and Time: cal, date
- Communications : telnet, ping
- Process Related Commands: ps, sleep

## login commands

1. **passwd:** This command is used to change the password for logged in user. If your account doesn‟t have a password or has one that is already known to others, you should change it immediately. This is done with the passwd command.

   **Syntax**
   $passwd
   Enter login password: ksc123
   New password: ******* (As your choice alphabet and number)
   Re-enter new password: ********
   passwd (SYSTEM): passwd successfully changed for the particular user

   passwd expects you to respond three times. First, it prompt for the old password. Next, it checks whether you have entered a valid password, and if you have, it then prompts for the new password.

2. **Logout:** This command is used to logout or terminate the session there is other way to logout with exit command and also you can terminate session with the press of ctrl +d but these commands may not be work every where to terminate the session the most reliable command is logout.

   **Syntax:**
   $logout

3. **Who**: This command is used to get information about users (including yourself), you can use several commands. The who command reports on users who are presently logged in. The who command normally reports certain information about logged-in users. By using its options, you can specify that it report information about the processes initiated by init, and that it report reboots changes to the system clock, and logoffs. If you invoke who with no option or argument, you get the following output:

   **Syntax:**
   $who

   **Output:**

**Here's how to read the output :**

1st column show the user name

2nd column show how the user connected. Tty means the user is connected directly to the computer, while pts means the user is connected from remote

3rd and 4th columns show the date and time

5th column show the IP Address where the users are connected

1. **Command to display the hostname and user associated with the input/output devices like a keyboard-**

**Syntax:**

$who -m -H

**Output:**

```
ksc      pts/0        2025-03-02 15:04 (:0.0)
ksc@ubuntu:~$ who -m -H
NAME     LINE         TIME             COMMENT
ksc      pts/0        2025-03-02 15:04 (:0.0)
```

2. **To display all details of currently logged in users-**

With this command's help, one sees all the details of every user logged in to the current system.

**Syntax:**

```
$ who -a
```

**Output:**

```
ksc@ubuntu:~$ who -a
           system boot  2025-03-02 15:02
           run-level 2  2025-03-02 15:02
LOGIN      tty4         2025-03-02 15:02              1020 id=4
LOGIN      tty5         2025-03-02 15:02              1025 id=5
LOGIN      tty2         2025-03-02 15:02              1034 id=2
LOGIN      tty3         2025-03-02 15:02              1035 id=3
LOGIN      tty6         2025-03-02 15:02              1039 id=6
LOGIN      tty1         2025-03-02 15:02              1181 id=1
ksc      + tty7         2025-03-02 15:02  old         1393 (:0)
ksc      + pts/0        2025-03-02 15:04   .          1672 (:0.0)
```

**Or**

You can use "who -all" instead of "who -a" as it displays all information.

**Syntax:**

```
$who  -all
```

**Output:**

```
ksc@ubuntu:~$ who -all
           system boot  2025-03-02 15:02
           run-level 2  2025-03-02 15:02
LOGIN      tty4         2025-03-02 15:02             1020 id=4
LOGIN      tty5         2025-03-02 15:02             1025 id=5
LOGIN      tty2         2025-03-02 15:02             1034 id=2
LOGIN      tty3         2025-03-02 15:02             1035 id=3
LOGIN      tty6         2025-03-02 15:02             1039 id=6
LOGIN      tty1         2025-03-02 15:02             1181 id=1
ksc      + tty7         2025-03-02 15:02  old        1393 (:0)
ksc      + pts/0        2025-03-02 15:04   .         1672 (:0.0)
ksc@ubuntu:~$ who -p -h
```

**3. To display information about all active processes that are spawned by the NIT process-**

This command will help you to display essential information as well as each and every active process.

**Syntax:**

$who -p -H

**Output:**

```
ksc@ubuntu:~$ who -p -H
NAME     LINE         TIME                 PID COMMENT
```

**4. To display the status of the user's message as -, + or?**

This command will help us to display the status of the user's message.

**Syntax**

$who -T -H

**Output**

```
NAME     LINE         TIME                 PID COMMENT
ksc@ubuntu:~$ who -T -H
NAME       LINE         TIME             COMMENT
ksc      + tty7         2025-03-02 15:02 (:0)
ksc      + pts/0        2025-03-02 15:04 (:0.0)
```

**5. To display the whole list of logged-in users-**

This command will help us to display the whole list of the logged-in users.

**Syntax**

$who -u

**Output**


**6. To display the whole list of dead processes-**

One can use this command to see the complete list of all dead processes.

**Syntax**

$who -d -H

**Output**

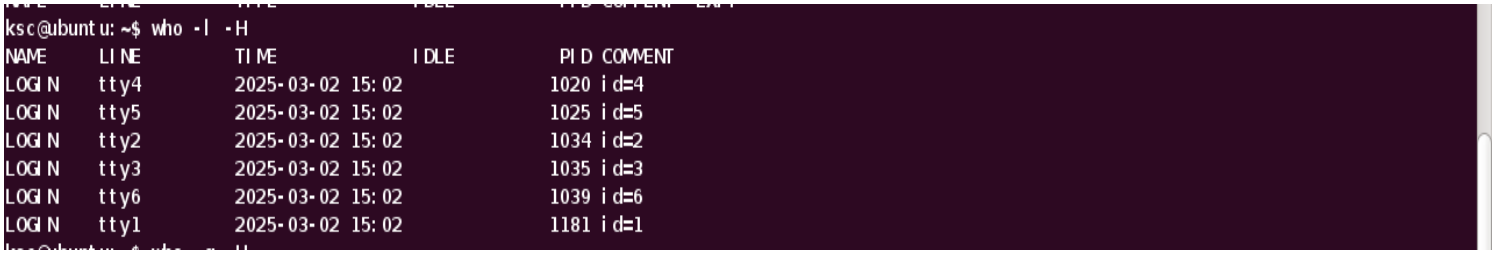

**7. To display system login process details-**

One can use this command to see the login process.

**Syntax:**

$who -l -H

**Output**




**8. To count the numbers of all logged-in users-**

We can use this command to see how many users logged in in the form of numbers

**Syntax:**

$who -q -H

**Output**

```
ksc@ubuntu:~$ who -q -H
ksc ksc
# users=2
```

**9. To display the current run level of the system-**

**Syntax:**

$who -r

**Output**

```
# users=2
ksc@ubuntu:~$ who -r
         run-level 2  2025-03-02 15:02
```

**10. To display the system's username-**

This command is generally used to know about the actual system's username.

**Syntax**

$whoami

**Output**

```
ksc@ubuntu:~$ whoami
ksc
```

**11. To display the list of users and their activities-**

With the help of this command, you can see the complete list of users and their activities as well, which are logged-in the current system.

**Syntax**

$w

**Output**

```
ksc@ubuntu:~$ w
 15:08:51 up 7 min,  2 users,  load average: 0.17, 0.77, 0.52
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
ksc      tty7     :0               15:02    6:59   8.18s  0.35s gnome-session
ksc      pts/0    :0.0             15:04    0.00s  0.13s  0.01s w
```

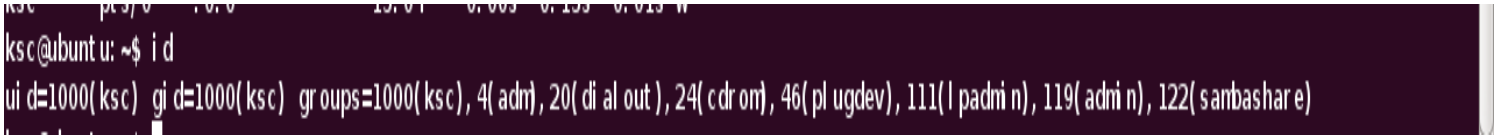
### 12. To display user identification information-

One can use this command to see the user identification information.

**Syntax**

$id

**Output**




**3.Clear:** This command is used to clear the screen.

**Syntax:**

$clear

**4.uname:** The uname command in Unix and Linux is used to display system information

Syntax:

$uname

## File / Directory Related Command

**(1) ls:** This command is used see the list of files and directory at current location or specified path. This command supports us to see the list of file and directory.

```
$ls
ksc@ubuntu:~$ ls
Add.sh  Ayushi     Documents  examples.desktop  Music   output.pdf  Pictures  read.sh  SYBCAA  Videos
AK      Desktop    Downloads  extra             odd.vi  p1.sh       Public    sum.sh   SYBCAB  xyz.txt
```

**Options:**

$ls –x : Multicolumn output

```
ksc@ubuntu:~$ ls -x
Add.sh  AK      Ayushi   Desktop  Documents  Downloads  examples.desktop  extra  Music  odd.vi  output.pdf  p1.sh  Pictures  Public  read.sh
sum.sh  SYBCAA  SYBCAB   Videos   xyz.txt
```

$ls -F : Marks executable with * directories with / and symbolic links with @

```
ksc@ubuntu:~$ ls -F
Add.sh*  Ayushi/   Documents/  examples.desktop  Music/  output.pdf  Pictures/  read.sh  SYBCAA/  Videos/
AK/      Desktop/  Downloads/  extra             odd.vi  p1.sh*      Public/    sum.sh*  SYBCAB/  xyz.txt
```

$ls -a : Show all filenames beginning with a dot including . and ..

```
ksc@ubuntu:~$ ls -a
.               .cache      .evolution      .gstreamer-0.10  Music       .profile                 sum.sh  .themes
..              .config     examples.desktop  .gtk-bookmarks   .nautilus   Public                   .swl    .thumbnails
Add.sh          .dbus       extra           .gvfs            odd.vi      .pulse                   .swm    Videos
AK              Desktop     .fontconfig     .ICEauthority    .openoffice.org  .pulse-cookie       .swn    .xsession-errors
Ayushi          .dmrc       .gconf          .icons           output.pdf  read.sh                  .swo    .xsession-errors.old
.bash_history   Documents   .gconfd         .local           p1.sh       .recently-used.xbel      .swp    xyz.txt
.bash_logout    Downloads   .gnome2         .mission-control Pictures    .romil.sh.swp            SYBCAA
.bashrc         .esd_auth   .gnome2_private  .mozilla        .pki        .sudo_as_admin_successful  SYBCAB
```

$ls -R : Recursive list

```
ksc@ubuntu: ~$ ls -R
.:
Add.sh  Ayushi     Documents  examples.desktop  Music     output.pdf  Pictures  read.sh  SYBCAA  Videos
AK      Desktop    Downloads  extra             odd.vi    p1.sh       Public    sum.sh   SYBCAB  xyz.txt

./AK:
read.sh  xyz.txt

./Ayushi:
age.sh   Ayu      date.sh  hello.sh  info.sh  loginuser.sh  name.sh        sweping.sh          while1.sh
area.sh  Ayu.sh   Hello    if1.sh    last.sh  mark_tot.sh   posi_para.sh   system_variable.sh

./Ayushi/Ayu:

./Desktop:
p1.sh  Screenshot.png

./Documents:

./Downloads:

./Music:

./Pictures:

./Public:

./SYBCAA:
echo.sh  print.sh

./SYBCAB:
echo.sh  print.sh

./Videos:
```

## $ls -r : Sorts filenames in reverse order (ASCII collating sequence by default)

```
ksc@ubuntu: ~$ ls -r
xyz.txt  SYBCAA  Public     output.pdf  extra             Documents  AK
Videos   sum.sh  Pictures   odd.vi      examples.desktop  Desktop    Add.sh
SYBCAB   read.sh p1.sh      Music       Downloads         Ayushi
ksc@ubuntu: ~$
```

## $ls -1 : One filename in each line

```
ksc@ubuntu: ~$ ls -1
Add.sh
AK
Ayushi
Desktop
Documents
Downloads
examples.desktop
extra
Music
odd.vi
output.pdf
p1.sh
Pictures
Public
read.sh
sum.sh
SYBCAA
SYBCAB
Videos
xyz.txt
```

## $ls -l : Long listing in ASCII collating sequence showing seven attributes of a file

```
ksc@ubuntu:~$ ls -l
total 160
-rwxr-xr-x 2 ksc ksc   165 2025-02-08 09:04 Add.sh
drwxr-xr-x 2 ksc ksc  4096 2014-09-12 15:52 AK
drwxr-xr-x 3 ksc ksc  4096 2025-02-24 06:15 Ayushi
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:33 Desktop
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Documents
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Downloads
-rw-r--r-- 1 ksc ksc   179 2013-03-30 02:49 examples.desktop
-rw-r--r-- 3 ksc ksc   162 2014-09-12 17:57 extra
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Music
-rw-r--r-- 4 ksc ksc   102 2018-01-22 07:41 odd.vi
-rw-r--r-- 1 ksc ksc 85368 2025-03-02 15:22 output.pdf
-rwxr-xr-x 2 ksc ksc   133 2025-02-08 07:15 p1.sh
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Pictures
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Public
-rw-r--r-- 3 ksc ksc   162 2014-09-12 17:57 read.sh
-rwxr-xr-x 1 ksc ksc   141 2025-02-19 15:27 sum.sh
drwxr-xr-x 2 ksc ksc  4096 2025-02-19 09:08 SYBCAA
drwxr-xr-x 2 ksc ksc  4096 2025-02-19 08:06 SYBCAB
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Videos
-rw-r--r-- 2 ksc ksc    34 2014-09-12 15:34 xyz.txt
```

## $ls -t : Sorts filenames by last modification time

```
ksc@ubuntu:~$ ls -t
Desktop     Ayushi   SYBCAA  Add.sh  odd.vi  read.sh  xyz.txt    Downloads  Pictures  Videos
output.pdf  sum.sh   SYBCAB  p1.sh   extra   AK       Documents  Music      Public    examples.desktop
```

## $ls -lt : Sorts filenames listing by last modification time

```
ksc@ubuntu:~$ ls -lt
total 160
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:33 Desktop
-rw-r--r-- 1 ksc ksc 85368 2025-03-02 15:22 output.pdf
drwxr-xr-x 3 ksc ksc  4096 2025-02-24 06:15 Ayushi
-rwxr-xr-x 1 ksc ksc   141 2025-02-19 15:27 sum.sh
drwxr-xr-x 2 ksc ksc  4096 2025-02-19 09:08 SYBCAA
drwxr-xr-x 2 ksc ksc  4096 2025-02-19 08:06 SYBCAB
-rwxr-xr-x 2 ksc ksc   165 2025-02-08 09:04 Add.sh
-rwxr-xr-x 2 ksc ksc   133 2025-02-08 07:15 p1.sh
-rw-r--r-- 4 ksc ksc   102 2018-01-22 07:41 odd.vi
-rw-r--r-- 3 ksc ksc   162 2014-09-12 17:57 extra
-rw-r--r-- 3 ksc ksc   162 2014-09-12 17:57 read.sh
drwxr-xr-x 2 ksc ksc  4096 2014-09-12 15:52 AK
-rw-r--r-- 2 ksc ksc    34 2014-09-12 15:34 xyz.txt
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Documents
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Downloads
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Music
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Pictures
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Public
drwxr-xr-x 2 ksc ksc  4096 2013-03-30 02:53 Videos
-rw-r--r-- 1 ksc ksc   179 2013-03-30 02:49 examples.desktop
```

$ls -u : Sorts filenames by last access time

```
ksc@ubuntu:~$ ls -u
Desktop     xyz.txt  Ayushi  Documents  Music  SYBCAA          Pictures  Videos  read.sh  p1.sh
output.pdf  AK       SYBCAB  Downloads  Public  examples.desktop  odd.vi    extra   Add.sh   sum.sh
```

$ls -lu :Sort by ASCII collating sequence but listing shows last access time

```
ksc@ubuntu:~$ ls -lu
total 160
-rwxr-xr-x 2 ksc ksc   165 2025-02-27 10:42 Add.sh
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 AK
drwxr-xr-x 3 ksc ksc  4096 2025-03-02 15:15 Ayushi
```

$ls -lut : As above but sorted by last access time

```
ksc@ubuntu:~$ ls -lut
total 160
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:33 Desktop
-rw-r--r-- 1 ksc ksc 85368 2025-03-02 15:22 output.pdf
-rw-r--r-- 2 ksc ksc    34 2025-03-02 15:15 xyz.txt
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 AK
drwxr-xr-x 3 ksc ksc  4096 2025-03-02 15:15 Ayushi
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 SYBCAB
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Documents
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Downloads
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Music
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Public
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 SYBCAA
-rw-r--r-- 1 ksc ksc   179 2025-03-02 15:15 examples.desktop
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Pictures
-rw-r--r-- 4 ksc ksc   102 2025-03-02 15:15 odd.vi
drwxr-xr-x 2 ksc ksc  4096 2025-03-02 15:15 Videos
-rw-r--r-- 3 ksc ksc   162 2025-03-02 15:15 extra
-rw-r--r-- 3 ksc ksc   162 2025-03-02 15:15 read.sh
-rwxr-xr-x 2 ksc ksc   165 2025-02-27 10:42 Add.sh
-rwxr-xr-x 2 ksc ksc   133 2025-02-27 05:55 p1.sh
-rwxr-xr-x 1 ksc ksc   141 2025-02-19 15:27 sum.sh
```

$ls -i : Display inode number

```
ksc@ubuntu:~$ ls -i
138317 Add.sh  130037 Desktop    149043 examples.desktop  138365 odd.vi    130043 Pictures  138424 sum.sh  130044 Videos
134484 AK      130041 Documents  134498 extra             134486 output.pdf  130040 Public    138372 SYBCAA  134490 xyz.txt
134441 Ayushi  130038 Downloads  130042 Music             134483 p1.sh     134498 read.sh   134416 SYBCAB
```

$ls -lh : (h stands for human readable form) : To display file size in easy-to-read format. i.e i.e M for MB, K for KB, G for GB.

```
ksc@ubuntu:~$ ls -lh
total 164K
-rwxr-xr-x 2 ksc ksc  165 2025-02-08 09:04 Add.sh
drwxr-xr-x 2 ksc ksc 4.0K 2014-09-12 15:52 AK
drwxr-xr-x 3 ksc ksc 4.0K 2025-02-24 06:15 Ayushi
-rw-r--r-- 1 ksc ksc   57 2025-03-02 20:19 demo.sh
drwxr-xr-x 2 ksc ksc 4.0K 2025-03-02 20:20 Desktop
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Documents
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Downloads
-rw-r--r-- 1 ksc ksc  179 2013-03-30 02:49 examples.desktop
-rw-r--r-- 3 ksc ksc  162 2014-09-12 17:57 extra
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Music
-rw-r--r-- 4 ksc ksc  102 2018-01-22 07:41 odd.vi
-rw-r--r-- 1 ksc ksc  84K 2025-03-02 15:22 output.pdf
-rwxr-xr-x 2 ksc ksc  133 2025-02-08 07:15 p1.sh
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Pictures
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Public
-rw-r--r-- 3 ksc ksc  162 2014-09-12 17:57 read.sh
-rwxr-xr-x 1 ksc ksc  141 2025-02-19 15:27 sum.sh
drwxr-xr-x 2 ksc ksc 4.0K 2025-02-19 09:08 SYBCAA
drwxr-xr-x 6 ksc ksc 4.0K 2025-03-08 06:49 SYBCAB
drwxr-xr-x 2 ksc ksc 4.0K 2013-03-30 02:53 Videos
-rw-r--r-- 2 ksc ksc   34 2014-09-12 15:34 xyz.txt
```

$ls -lah :

| | |
|---|---|
| -l | Long format (detailed list with permissions, owner, size, date) |
| -a | Show hidden files (files starting with .) |
| -h | Human-readable sizes |

```
ksc@ubuntu:~$ ls -lah
total 416K
drwxr-xr-x 35 ksc  ksc  4.0K 2025-03-08 06:45 .
drwxr-xr-x  6 root root 4.0K 2025-03-04 08:37 ..
-rwxr-xr-x  2 ksc  ksc   165 2025-02-08 09:04 Add.sh
drwxr-xr-x  2 ksc  ksc  4.0K 2014-09-12 15:52 AK
drwxr-xr-x  3 ksc  ksc  4.0K 2025-02-24 06:15 Ayushi
-rw-------  1 ksc  ksc  4.6K 2025-03-08 07:04 .bash_history
-rw-r--r--  1 ksc  ksc   220 2013-03-30 02:49 .bash_logout
-rw-r--r--  1 ksc  ksc  3.3K 2013-03-30 02:49 .bashrc
drwx------  7 ksc  ksc  4.0K 2025-03-08 06:45 .cache
drwxr-xr-x 10 ksc  ksc  4.0K 2025-03-08 06:46 .config
drwx------  3 ksc  ksc  4.0K 2013-03-30 02:53 .dbus
-rw-r--r--  1 ksc  ksc    57 2025-03-02 20:19 demo.sh
drwxr-xr-x  2 ksc  ksc  4.0K 2025-03-02 20:20 Desktop
-rw-r--r--  1 ksc  ksc    41 2025-03-08 06:45 .dmrc
drwxr-xr-x  2 ksc  ksc  4.0K 2013-03-30 02:53 Documents
drwxr-xr-x  2 ksc  ksc  4.0K 2013-03-30 02:53 Downloads
-rw-------  1 ksc  ksc    16 2013-03-30 02:53 .esd_auth
drwx------  8 ksc  ksc  4.0K 2025-03-02 15:23 .evolution
-rw-r--r--  1 ksc  ksc   179 2013-03-30 02:49 examples.desktop
-rw-r--r--  3 ksc  ksc   162 2014-09-12 17:57 extra
drwxr-xr-x  2 ksc  ksc  4.0K 2013-03-30 02:53 .fontconfig
drwx------  4 ksc  ksc  4.0K 2025-03-08 06:45 .gconf
```

$ ls -n: lists the files and directories in the current directory or files with numeric user and group IDs (UIDs and GIDs).

```
ksc@ubuntu:~$ ls -n
total 164
-rwxr-xr-x 2 1000 1000   165 2025-02-08 09:04 Add.sh
drwxr-xr-x 2 1000 1000  4096 2014-09-12 15:52 AK
drwxr-xr-x 3 1000 1000  4096 2025-02-24 06:15 Ayushi
-rw-r--r-- 1 1000 1000    57 2025-03-02 20:19 demo.sh
drwxr-xr-x 2 1000 1000  4096 2025-03-08 07:07 Desktop
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Documents
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Downloads
-rw-r--r-- 1 1000 1000   179 2013-03-30 02:49 examples.desktop
-rw-r--r-- 3 1000 1000   162 2014-09-12 17:57 extra
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Music
-rw-r--r-- 4 1000 1000   102 2018-01-22 07:41 odd.vi
-rw-r--r-- 1 1000 1000 85368 2025-03-02 15:22 output.pdf
-rwxr-xr-x 2 1000 1000   133 2025-02-08 07:15 p1.sh
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Pictures
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Public
-rw-r--r-- 3 1000 1000   162 2014-09-12 17:57 read.sh
-rwxr-xr-x 1 1000 1000   141 2025-02-19 15:27 sum.sh
drwxr-xr-x 2 1000 1000  4096 2025-02-19 09:08 SYBCAA
drwxr-xr-x 6 1000 1000  4096 2025-03-08 06:49 SYBCAB
drwxr-xr-x 2 1000 1000  4096 2013-03-30 02:53 Videos
-rw-r--r-- 2 1000 1000    34 2014-09-12 15:34 xyz.txt
```

$ ls -m: This command is used to list the files and directories in the current directory in a comma-separated format.

```
ksc@ubuntu:~$ ls -m
Add.sh, AK, Ayushi, demo.sh, Desktop, Documents, Downloads, examples.desktop,
extra, Music, odd.vi, output.pdf, p1.sh, Pictures, Public, read.sh, sum.sh,
SYBCAA, SYBCAB, Videos, xyz.txt
```

$ ls -Q: This command is used to list files and directories in the current directory, with each filename enclosed in double quotes (").

```
ksc@ubuntu:~$ ls -Q
"Add.sh"    "Desktop"           "extra"       "p1.sh"     "sum.sh"   "xyz.txt"
"AK"        "Documents"         "Music"       "Pictures"  "SYBCAA"
"Ayushi"    "Downloads"         "odd.vi"      "Public"    "SYBCAB"
"demo.sh"   "examples.desktop"  "output.pdf"  "read.sh"   "Videos"
```

$ ls *.file extension : This command is used to list files with a specific extension

```
ksc@ubuntu:~$ ls *.sh
Add.sh  demo.sh  p1.sh  read.sh  sum.sh
```

$ ls *.{sh,txt}: This command is used to list all files with the .sh or .txt extensions in the current directory.

```
ksc@ubuntu:~$ ls *.{sh,txt}
Add.sh  demo.sh  p1.sh  read.sh  sum.sh  xyz.txt
```

$ls /Directory : This command is used to list of subfolder and files

```
ksc@ubuntu:~$ ls / Ayushi
/:
bin   dev   initrd.img  media  proc  selinux  tmp  vmlinuz
boot  etc   lib         mnt    root  srv      usr
cdrom home  lost+found  opt    sbin  sys      var

Ayushi:
age.sh   Ayu.sh   hello.sh  last.sh       name.sh       system_variable.sh
area.sh  date.sh  if1.sh    loginuser.sh  posi_pera.sh  while1.sh
Ayu      Hello    info.sh   mark_tot.sh   sweping.sh
```

$ls -d */: This command is used to Show only directories

```
ksc@ubuntu:~$ ls -d */
AK/      Desktop/    Downloads/  Pictures/  SYBCAA/  Videos/
Ayushi/  Documents/  Music/      Public/    SYBCAB/
```

(2) **cat:** This command is use to displays files contents, creates new file, or appends data to an existing file.

**Syntax 1 To create a new file**

$cat > File Name

Enter the file content

Ctrl + d

**Example**

$cat > ayushi.sh

echo “HELLO SYBCA STUDENTS”

Ctrl+d

```
ksc@ubuntu:~$ cat > ayushi.sh
echo "HELLO SYBCA STUDENTS"
```

**Syntax 2 To display the content of file**

$cat File Name

**Example**

$cat ayushi.sh

```
ksc@ubuntu:~$ cat ayushi.sh
echo "HELLO SYBCA STUDENTS"
```

**Syntax 3 To append existing file**

$cat >> File Name

Enter the file content want to append

Ctrl+d

**Example**

$cat >>demo.sh

echo “GOOD MORNING”

Ctrl+d

```
ksc@ubuntu:~$ cat >> ayushi.sh
echo "GOOD MORNING"
```

(3) **cd (Change Directory):** This command is used to change one directory forward, backward and supporting to go to root directory directly.

**Syntax**

| | |
|---|---|
| Cd | Change working directory to home directory. |
| cd .. | Change working directory to parent directory. |
| cd | Path Change working directory to specified directory. |
| cd / | Change working directory to root directory. |

**Example**

**If current directory /home/ksc and user home directory is /home/ayushi then**

**$cd will take you to home directory**

now current directory is /home/ayushi

```
ksc@ubuntu:~$ cd ../..
ksc@ubuntu:/$ cd home/ayushi
ksc@ubuntu:/home/ayushi$ pwd
/home/ayushi
```

**$cd .. will take you to home directory**

now current directory is /home

```
ksc@ubuntu:~$ cd ..
ksc@ubuntu:/home$ pwd
/home
```

**$cd /home/ksc**

now current directory is /home/ksc

```
ksc@ubuntu:/home/ayushi$ cd ../..
ksc@ubuntu:/$ cd home/ksc
ksc@ubuntu:~$ pwd
/home/ksc
```

**$cd /**

now current directory is / (Root)

```
/home/ksc
ksc@ubuntu:~$
ksc@ubuntu:~$ cd ../..
ksc@ubuntu:/$ pwd
/
```

(4) **pwd: (Path for working directory):** This command is used to see path for working directory in UNIX you can not see the prompt with path at that time if we want to know the what is our current path. The pwd command supports to know path for working directory.

**Syntax**

**$pwd**

**Example**

**If current directory is /home/ksc then**

**$pwd will show you /home/ksc**

```
ksc@ubuntu:~$ pwd
/home/ksc
```

**(5) mv command (Renaming Files and Moving a file from one place to another place) The mv command renames files. It has two distinct functions:**

- It renames a file or directory.
- It moves a group of files to a different directory.

**To rename a file or directory**

**Syntax**
**$mv Old File Name New File Name**

mv doesn‟t create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming. To rename the file abc.txt to xyz.txt you can write following command.

**$mv ayushi.sh bca.sh**
If the destination file doesn‟t exist, it will be created. For the above example, mv simply replaces the filename in the existing directory entry with the new name. By default, mv doesn‟t prompt for overwriting the destination file if it exists. So be careful.

```
ksc@ubuntu:~$ mv ayushi.sh bca.sh
ksc@ubuntu:~$ ls
Add.sh  bca.sh   Documents      extra   output.pdf  Public   SYBCAA  xyz.txt
AK      demo.sh  Downloads      Music   p1.sh        read.sh  SYBCAB
Ayushi  Desktop  examples.desktop  odd.vi  Pictures     sum.sh   Videos
```

**To move a file from one place to another**

**Syntax**
**$mv file name Destination folder or path of Destination folder**

**Example**
To move bca.sh file from root to home/ksc/SYBCAB directory then following command is used from root.

**$mv bca.sh /home/ksc/SYBCAB**

Above command will move file named bca.sh from root to /home/ksc/SYBCAB

```
ksc@ubuntu:~$ mv bca.sh SYBCAB
ksc@ubuntu:~$ ls
Add.sh  demo.sh    Downloads          Music       p1.sh     read.sh  SYBCAB
AK      Desktop    examples.desktop   odd.vi      Pictures  sum.sh   Videos
Ayushi  Documents  extra              output.pdf  Public    SYBCAA   xyz.txt
ksc@ubuntu:~$ cd SYBCAB
ksc@ubuntu:~/SYBCAB$ ls
a  ayushi.sh  b  bca.sh  c  d  echo.sh  hello.sh  print.sh
ksc@ubuntu:~/SYBCAB$ mv bca.sh /home/ksc/SYBCAA/
ksc@ubuntu:~/SYBCAB$ ls
a  ayushi.sh  b  c  d  echo.sh  hello.sh  print.sh
```

**(6) cp (Copying a file)** This command is used to copy a file.

**Syntax**

**$cp [option] Source file path or file Target path**

**Option**

-i Asks before overwriting a destination file

```
ksc@ubuntu:~$ cp -i Add.sh sum.sh
cp: overwrite `sum.sh'? y
ksc@ubuntu:~$ cat sum.sh
int a, b;
echo Enter No 1:
read a
echo Enter No 2:
read b
echo sum is = $(($a + $b))
echo mul is = $(($a * $b))
echo div is = $(($a / $b))
echo sub is= $(($a - $b))
```

-r It copies directory structure.

```
ksc@ubuntu:~$ cp -r AK Ayu
ksc@ubuntu:~$ ls
Add.sh  Ayushi    Documents         extra   output.pdf  Public   SYBCAA  xyz.txt
AK      demo.sh   Downloads         Music   p1.sh       read.sh  SYBCAB
Ayu     Desktop   examples.desktop  odd.vi  Pictures    sum.sh   Videos
```

**Example**

**To copy a bca.sh file from /home/ksc/SYBCAA to /home/ksc/SYBCAB following command is used.**

**$cp bca.sh /home/ksc/SYBCAB**

Above command will copy bca.sh file from /home/ksc/SYBCAA to /home/ksc/SYBCAB directory

```
ksc@ubuntu:~/SYBCAB$ cd ../SYBCAA
ksc@ubuntu:~/SYBCAA$ cp bca.sh /home/ksc/SYBCAB
ksc@ubuntu:~/SYBCAA$ ls
bca.sh  echo.sh  print.sh
ksc@ubuntu:~/SYBCAA$ cd ../SYBCAB
ksc@ubuntu:~/SYBCAB$ ls
a  ayushi.sh  b  bca.sh  c  d  echo.sh  hello.sh  print.sh
```

**(7) ln (Link a file):** This command is used to link a file. Links are used for sharing of files from two different locations in directory structure. There are two types of link 1) soft link 2) hard link.

**Soft Link:-** It links file name if source is deleted then link will not be found

**Hard Link:-** It links physical storage area of the file if source file is deleted still it will target the physical part of the file and data can be found.

**Syntax:**

**$ln [Option] Source File Name Link File Name**

**Option**

-s Creates soft link if this option is not specified with ln command then hard link will be created.

**Example**

File Name demo.sh following is the content of the file hello dear students

**$ ln –s demo.sh link.sh**

```
ksc@ubuntu:~$ ln -s demo.sh symbolic.sh
ksc@ubuntu:~$ ls
a         Ayushi   demo        Downloads          Music       Pictures   SYBCAA
abc.txt   b        demo.sh     examples.desktop   odd.vi      Public     SYBCAB
Add.sh    c        Desktop     extra              output.pdf  read.sh    symbolic.sh
AK        d        Documents   link.sh            p1.sh       sum.sh     Videos
ksc@ubuntu:~$ ls -F
a*        Ayushi/  demo/       Downloads/         Music/      Pictures/  SYBCAA/
abc.txt*  b/       demo.sh     examples.desktop   odd.vi      Public/    SYBCAB/
Add.sh*   c/       Desktop/    extra*             output.pdf  read.sh*   symbolic.sh@
AK/       d/       Documents/  link.sh            p1.sh*      sum.sh*    Videos/
```

```
lrwxrwxrwx 1 ksc ksc     7 2025-03-16 09:16 symbolic.sh -> demo.sh
```

Soft link will be created if demo.sh file will be deleted then user will not be able to see the content of link.sh because it is related to demo.sh file name.

**$ ln demo.sh link.sh**

```
ksc@ubuntu:~$ ln demo.sh link.sh
ksc@ubuntu:~$ ls
a         Ayushi   demo        Downloads          Music       Pictures   SYBCAA
abc.txt   b        demo.sh     examples.desktop   odd.vi      Public     SYBCAB
Add.sh    c        Desktop     extra              output.pdf  read.sh    Videos
AK        d        Documents   link.sh            p1.sh       sum.sh
```

Hard link will be created if abc.txt file will be deleted still user will be able to work will the content of demo.sh file.

(8) **rm (Remove File)** This command is used to remove (delete) a file.

**Syntax**

**$rm [Option] [Path]File Name**

```
ksc@ubuntu:~$ ls
abc.sh  Ayushi   Desktop    examples.desktop  odd.vi      Pictures  read.sh  SYBCAB
Add.sh  BCA      Documents  extra             output.pdf  Public    sum.sh   Videos
AK      DEMO.sh  Downloads  Music             p1.sh       read1.sh  SYBCAA
ksc@ubuntu:~$ rm read1.sh
ksc@ubuntu:~$ ls
abc.sh  Ayushi   Desktop    examples.desktop  odd.vi      Pictures  sum.sh  Videos
Add.sh  BCA      Documents  extra             output.pdf  Public    SYBCAA
AK      DEMO.sh  Downloads  Music             p1.sh       read.sh   SYBCAB
```

**Options**

-i Ask before removing a file

```
ksc@ubuntu:~$ cat > Ayushi.sh
echo "HELLO DEAR STUDENTS"
ksc@ubuntu:~$ rm -i Ayushi.sh
rm: remove regular file `Ayushi.sh'? y
```

-r Performs recursive deletion in directory

```
ksc@ubuntu:~$ rm -r BCA
ksc@ubuntu:~$ ls
abc.sh  Ayushi   Documents         extra   output.pdf  Public   SYBCAA
Add.sh  DEMO.sh  Downloads         Music   p1.sh       read.sh  SYBCAB
AK      Desktop  examples.desktop  odd.vi  Pictures    sum.sh   Videos
```

-f Removes write protected file also

**Example**

**To remove a file a.txt from /home/ksc/SYBCAA current directory is root then following command will be written.**

**$rm /home/ksc/SYBCAA/a.txt**

Above command will remove abc.txt file.

```
ksc@ubuntu:~$ ls SYBCAA
a.txt  bca.sh  demo.sh  echo.sh  example.sh  print.sh
ksc@ubuntu:~$ rm /home/ksc/SYBCAA/a.txt
ksc@ubuntu:~$ ls
a       a.txt   Desktop    examples.desktop  odd.vi      Pictures  sum.sh  Videos
Add.sh  Ayushi  Documents  extra             output.pdf  Public    SYBCAA
AK      b.txt   Downloads  Music             p1.sh       read.sh   SYBCAB
ksc@ubuntu:~$ cd /home/ksc/SYBCAA/
ksc@ubuntu:~/SYBCAA$ ls
bca.sh  demo.sh  echo.sh  example.sh  print.sh
```

**rmdir: (Remove directory):** This command is used to remove directory form current path and specified path

**Syntax:**

**$rmdir [Path] Directory Name**

```
ksc@ubuntu:~$ rmdir b
ksc@ubuntu:~$ ls
a        Ayushi  demo.sh    examples.desktop  odd.vi      Public    SYBCAB
abc.txt  c       Desktop    extra             output.pdf  read.sh   symbolic.sh
Add.sh   d       Documents  link.sh           p1.sh       sum.sh    Videos
AK       demo    Downloads  Music             Pictures    SYBCAA
ksc@ubuntu:~$
ksc@ubuntu: ~
```

This command is useful to remove empty directory the prime condition of this command is this that whatever the directory is, you are going to remove it that must be empty else message will display on the screen that directory is not empty.

```
ksc@ubuntu:~$ ls SYBCAB
a  ayushi.sh  b  bca.sh  c  d  echo.sh  hello.sh  print.sh
ksc@ubuntu:~$ rmdir SYBCAB
rmdir: failed to remove `SYBCAB': Directory not empty
```

**Error Message rmdir: dirname: cannot remove [Directory not empty]**

**(9) mkdir: (Make Directory):** This command is used to create a directory on current path or specified path.

**Syntax:**

**$mkdir Directory Name**

```
ksc@ubuntu:~$ mkdir student
ksc@ubuntu:~$ ls
a         Ayushi   demo.sh    examples.desktop  odd.vi      Public    SYBCAA
abc.txt   c        Desktop    extra             output.pdf  read.sh   SYBCAB
Add.sh    d        Documents  link.sh           p1.sh       student   symbolic.sh
AK        demo     Downloads  Music             Pictures    sum.sh    Videos
```

Some time you are giving right command still directory is not going to be created for following reasons.

```
ksc@ubuntu:~$ mkdir SYBCAB
mkdir: cannot create directory `SYBCAB': File exists
ksc@ubuntu:~$ mkdir demo
mkdir: cannot create directory `demo': File exists
```

**Error Message Mkdir: directory name: [file exists]**

(1) The directory name is given with mkdir is already exist

(2) There may be a an ordinary file by that name in the current directory

(3) The permission set for the current directory don‟t permit

**(10) Umask:** This command is used to set default file permission. Whatever the file is created by the owner at that time which permission is default given to user, group and other that default

permission will be set using Umask command. Umask stands for user file creation mask. The term mask implying which permissions to mask or hide. The umask value tells UNIX which of the three permission are to be denied rather than granted. The current value of umask can be easily determined by just typing umask.

**To see the default permissions**

**Syntax**
**$umask**

$umask
0022

```
ksc@ubuntu:~$ umask
0022
ksc@ubuntu:~$
```

Here first 0 indicates that for owner no permission is denied, Second 0 indicates that for user no permission is denied, third number 2 indicates that write permission is denied to group and fourth number two is also indicating that write permission denied to others.

**Changing default permission using umask**
**$umask 111**
This would see to it that here onwards any new file that you create would have the permissions 555 and any directory that you create would have the permission 666.

```
ksc@ubuntu:~$ umask 111
ksc@ubuntu:~$ umask
0111
ksc@ubuntu:~$
```

**(11) Chmod (Changing File and Directory Permissions):** A file or directory is created with a default set of permissions, and this default is determined by a simple setting called umask. Generally, the default setting write-protecccts a file from all except the user (new name for owner), though all user may have read access. However, this may not be so on your system. To know your system‟s default, create a file SYBCAB.txt:

$cat > SYBCAB.txt

$ls –l

-rw-r--r-- 1 ksc ksc 1906 Sep 5 23:38 SYBCAB.txt

```
ksc@ubuntu:~$ cat > SYBCAB.txt
echo "Hello Dear Students"
ksc@ubuntu:~$
```

**Syntax Relative Permission**

$chmod <<User Category>> <<Operation>> <<Permission>> <<File Name>>

**User Category :** User, Group, Other

**Operation :** Assign Or remove a permission using + and –

**Permission** : Read, Write, Execute

**Example**

To assign execute permission to the user of the file ayushi.sh, we need to write following command

**$chmod u+x ayushi.sh**

$ls –l

-rwxr--r-- 1 ksc ksc 47 2025-03-09 09:31 ayushi.sh

```
ksc@ubuntu:~/SYBCAB$ chmod u+x ayushi.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rw-r--r-- 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

**To use multiple characters to represent the user category (ugo).**

**$chmod ugo+x bca.sh**

```
ksc@ubuntu:~/SYBCAB$ chmod u+w bca.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rw-r--r-- 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
ksc@ubuntu:~/SYBCAB$ chmod g+w bca.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rw-rw-r-- 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

**or**

**$chmod a+x xstart**

**Or**

**$chmod +x xstart**

$ls –l

-rwxr-xr-x 1 ksc ksc 1906 May 10 20:30 bca.sh

```
ksc@ubuntu:~/SYBCAB$ chmod +x bca.sh
ksc@ubuntu:~/SYBCAB$ ls
a  ayushi.sh  b  bca.sh  c  d  echo.sh  hello.sh  print.sh
```

**To use more then one file at a time**

$chmod u+x note note1 note2 note 3

**To grant and revoke a permission at a time**

$chmod a-x,ugo+r xstart

**Abbreviations Used By chmod**

Category Operation Permission

u user + Assigns Permission r Read Permission

g group - Remove Permission w Write Permission

o Other = Assigns absolute permission x Execute permission

a All(ugo)

```
ksc@ubuntu:~/SYBCAB$ chmod ugo=wx print.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rwxrw---x 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
--wx-wx-wx 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

**Syntax Absolute Permissions**

$chmod Combination of Decimal Code

**Combination of Decimal Code**

- Read Permission - 4 (Octal 100)
- Write Permission - 2 (Octal 010)
- Execute Permission - 1 (Octal 001)

| Binary | Octal | Permissions | Significance |
|---|---|---|---|
| 000 | 0 | --- | No permission |
| 001 | 1 | --x | Executable Only |
| 010 | 2 | -w- | Writable Only |
| 011 | 3 | -wx | Writable and Executable |
| 100 | 4 | r-- | Readable Only |
| 101 | 5 | r-x | Readable and Executable |
| 110 | 6 | rw- | Readable and Writable |

| | | | |
|---|---|---|---|
| 111 | 7 | rwx | Readable, Writable and Executable |

**Example 1**

**$chmod 666 bca.sh**

**$ls –l bca.sh**

-rw-rw-rw- 1 ksc ksc 1906 May 10 20:30 bca.sh

```
ksc@ubuntu:~/SYBCAB$ chmod 666 bca.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rw-rw-rw- 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

The 6 indicates read and write permission (4 + 2). To restore the original permissions to the file, you need to remove the write permission (2) from group and others:

**Example 2**

**$chmod 644 bca.sh**

**$ls –l bca.sh**

-rw-r--r-- 1 ksc ksc 1906 May 10 20:30 bca.sh

To assign all permissions to the owner, read and write permissions to the group, and only execute permission to the other, use following command.

```
ksc@ubuntu:~/SYBCAB$ chmod 644 bca.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rw-r--r-- 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

**Example 3**

**$chmod 761 bca.sh**

-rwxrw---x 1 ksc ksc 1906 May 10 20:30 bca.sh

```
ksc@ubuntu:~/SYBCAB$ chmod 761  bca.sh
ksc@ubuntu:~/SYBCAB$ ls -l
total 36
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:01 a
-rwxr--r-- 1 ksc ksc   47 2025-03-09 09:31 ayushi.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 b
-rwxrw---x 1 ksc ksc   48 2025-03-10 10:23 bca.sh
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-08 06:49 d
-rwxr-xr-x 1 ksc ksc   22 2025-02-19 08:01 echo.sh
-rw-r--r-- 1 ksc ksc   72 2025-03-07 09:07 hello.sh
-rwxr-xr-x 1 ksc ksc   20 2025-02-19 08:07 print.sh
```

**(12) Chown: (Change Owner):** This command is used to change the owner of file. It transfers ownership of a file to a user, and it seems that it can optionally change the group as well. The command requires the user-id (UID) of the recipient, followed by one or more filenames. Changing ownership requires super user permission, so let‟s first change our status to that of super user with the su command.

$su
Password:********
#_

```
ksc@ubuntu:~$ su bca
Password:
```

After the password is successfully entered, su returns a # prompt, the same prompt used by root. su lets us acquire super user status if we know the root password. To now renounce the ownership of the file note to sharma, use chown in the following way:

**Syntax:**

Chown < New Owner Name> file(s)

```
ksc@ubuntu:~$ sudo chgrp -c bca abc.txt
[sudo] password for ksc:
changed group of `abc.txt' to bca
ksc@ubuntu:~$ ls -l abc.txt
-rwxr--r-- 1 bca bca 6 2025-03-12 07:54 abc.txt
ksc@ubuntu:~$
```

**Example**

#ls –s note

-rwxr----x 1 kumar metal 347 May 10 20:30 Note

#chown sharma note;

#ls –l

-rwxr----x 1 sharma metal 347 May 10 20:30 Note

#exit

$_

Once ownership of the file has been given away to sharma, the permissions that previously applied to kumar now apply to sharma. Thus, kumar can no longer edit note since there‟s no write privilege for group and others. He can‟t get back the ownership either. But he can copy the file to his own directory, in which case he becomes the owner of the copy.

**(13) chgrp: (Changing Group Owner):** By default, the group owner of a file is the group to which the owner belongs. **The chgrp (change group) command changes a file's group owner.** A user can change the group owner of a file, but only to a group to which she also belongs. Yes, a user can belong to more than one group.

chgrp shares a similar syntax with chown. In the following example, kumar changes the group ownership of dept.lst to dba (No super user permission required):

**Syntax:**

chgrp <<New Group Name>> <<File Name>>

$ls –l dept.lst

-rw-r--r-- 1 kumar metal 139 Jun 8 16:43 dept.lst

$chgrp dba dept.lst

-rw-r--r-- 1 kumar dba 139 Jun 8 16:43 dept.lst

```
ksc@ubuntu:~$ sudo chgrp -c bca abc.txt
[sudo] password for ksc:
changed group of `abc.txt' to bca
ksc@ubuntu:~$ ls -l abc.txt
-rwxr--r-- 1 bca bca 6 2025-03-12 07:54 abc.txt
```

If kumar is also a member of the dba group then above command will work, if he is not a member of the dba group then super user can make the command work. Note that kumar can reverse this action and restore the previous group ownership (to metal) because he is still owner of the file and consequently retains all right related to it.

14) **Find (Locating Files):** This command is used to find a particular file.

**Syntax**
**$find [Path]/File Name**

**Example 1**
**$find abc.txt**
**$find /home/guest/abc.txt**

```
ksc@ubuntu:~$ find link.sh
link.sh
```

User can use wild card character with file name * means all character and ? means 1 character

**Example 2**
To find the file which having extension name .txt following command is used
**$find *.txt**

```
ksc@ubuntu:~$ find link.sh
link.sh
ksc@ubuntu:~$ find *.txt
abc.txt
SYBCAB.txt
ksc@ubuntu:~$ find *.sh
Add.sh
demo.sh
link.sh
p1.sh
read.sh
sum.sh
symbolic.sh
ksc@ubuntu:~$
```

**Example 3**

To find the file which 2nd character is d then following command is used

**$find ?d*.***

```
ksc@ubuntu:~$ find ?d*.*
Add.sh
odd.vi
ksc@ubuntu:~$ find ??m*
demo
demo.sh
sum.sh
symbolic.sh
```

**(15) Pg commands:** This command is used to see the information of the document page wise.

**Syntax**

**$pg** Starting Line Number  No of Lines Per Page Option  File Name

**Example**

**$pg +2 –4 –p “Page No. %d” –s a.txt**

```
ksc@ubuntu:~$ cat>a.txt
Hi
Hello
BCA
College
Student
ksc@ubuntu:~$ pg +2-4 -p "1%d" -s a.txt
BCA
College
Student
```

**Explanation**

Above command starts displaying the contents of abc.txt file, 04 lines at a time from 2nd line onwards. At the end of each displayed page a prompt comes which displays the page number on view. This prompt overrides the default „:‟ prompt of the pg command. The –s option ensures that the prompt is displayed in reverse video.

**(16) More Command (Paging out)**

This command is used to see the information of the document page wise.

**Syntax**
**$more File Name Press q to exit**

**Example**
**$more odd.vi**

```
ksc@ubuntu:~$ cat odd.vi
echo"enter a number"
read.x
y='exper $x% 2' if[$% -eq 0)
then
echo "no is even"
else
echo"no is all"

ksc@ubuntu:~$ more +2 odd.vi
read.x
y='exper $x% 2' if[$% -eq 0)
then
echo "no is even"
else
echo"no is all"
```

**Explanation**

You will see the contents of odd.vi on the screen, one page at a time. At the bottom of the screen, you‟ll also see the filename and percentage of the file that has been viewed:

**(17) less (Paging Out):** This command is used to see the information of the document page wise. This command work just like more command and it is older version.

**Syntax**
**$less File Name Press q to exit**

**Example**
**$less abc.txt**

**Explanation**
You will see the contents of abc.txt on the screen, one page at a time. At the bottom of the screen, you‟ll also see the filename and : that is ready to accept the navigation command those are described below.

**Abc.txt :**
Less has a couple of internal commands that don‟t show up on the screen when you invoke them. q, the command used to exit less, is an internal command.

**Navigation of less Command**
Irrespective of version, less uses the spacebar to scroll forward a page at a time. You can also scroll by small and large increment of lines or screens. To move forward one page, use

| | |
|---|---|
| f or spacebar | To move page forward |
| b | To move back one page |
| Enter or j | One line forward |
| k | One line back |
| p or 1G | Beginning of file |
| G | End of file |
| q | Quit |

At the end of file user has to give q to quit from less command.

**(18) head:** We are using cat command to view the content of files; however, if the file is bigger than 24 lines then the matter would naturally scroll off the screen. If we want to stop the scrolling we can do so by hitting the pause key and resume it by hitting any other key. This of course needs a bit of a practice otherwise the matter scrolls off the screen before you can reach for the pause

key. To exercise a tighter control over the way we can view files UNIX proves several utilities. Out of these the head commands help in viewing lines at the beginning of the file respectively.

**Syntax:**

**$head –Line Number File Name**


Unless otherwise specified the head command assumes that you want to display first 10 lines in the file. Should you decide to view first fifteen lines you simply have to say.

**$head -15 myfile**

Unless otherwise specified the head command assumes that you want to display first 10 lines in the file. Should you decide to view first fifteen lines you simply have to say.

**$head -15 myfile**

**Here with head command we can specify the number of lines if we decide to override this default value.**

The disadvantage of head is that they cannot display a range of lines. Moreover, what is displayed is final. That is, if we have displayed the first 50 lines in a file we cannot move back and view say the 10th line. UNIX provides two commands which offer more flexibility in viewing files.

**(19) tail:** We are using cat command to view the content of files; however, if the file is bigger than 24 lines then the matter would naturally scroll off the screen. If we want to stop the scrolling we can do so by hitting the pause key and resume it by hitting any other key. This of course needs a bit of a practice otherwise the matter scrolls off the screen before you can reach for the pause key. To exercise a tighter control over the way we can view files UNIX proves several utilities. Out of these the tail commands help in viewing lines at the ending of the file respectively.

**Syntax:**

**$tail –Line Number File Name**

Unless otherwise specified the tail command assumes that you want to display last 10 lines in the file. Should you decide to view last fifteen lines you simply have to say.

**$tail -15 myfile**

```
ksc@ubuntu:~$ tail -3 p1.sh
        echo "$n is odd no";
fi
```

**Here with tail command we can specify the number of lines if we decide to override this default value.**

The disadvantage of tail is that they cannot display a range of lines. Moreover, what is displayed is final. That is, if we have displayed the last 50 lines in a file we cannot move back and view say the 10th line. UNIX provides two commands which offer more flexibility in viewing files. These commands are pg and more.

**(20) wc (Word Count):** This command is used to count number of characters, words and lines in an given input file. If file is not given, it takes input from standard input.

**Syntax**

**$wc [Option] File Name**

```
ksc@ubuntu:~$ wc p1.sh
 11  32 133 p1.sh
ksc@ubuntu:~$ wc -l p1.sh
11 p1.sh
ksc@ubuntu:~$ wc -w p1.sh
32 p1.sh
ksc@ubuntu:~$ wc -c p1.sh
133 p1.sh
```

**Options**

-l Counts number of lines
-w Counts number of words
-c Counts number of characters

**Example**

A file abc.txt having following data
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
**Note: Spacebar and Enter also known as on character**

**(21) touch (Changing the time stamp):** The touch command in Unix/Linux is used to create empty files or update the access and modification timestamps of existing files.

**Syntax**

**$touch File Name**

```
ksc@ubuntu:~$ touch demo.txt
ksc@ubuntu:~$ ls
a         Ayushi   demo.sh     Downloads          Music     Public   SYBCAA       Videos
abc.txt   c        demo.txt    examples.desktop   odd.vi    read.sh  SYBCAB
Add.sh    d        Desktop     extra              p1.sh     student  SYBCAB.txt
AK        demo     Documents   link.sh            Pictures  sum.sh   symbolic.sh
```

(22) **stat:** The 'stat' command is used to display detailed information about files and file systems, including attributes like file size, permissions, ownership, timestamps, and more.

**Syntax**

```
stat filenames
```

```
ksc@ubuntu:~$ stat link.sh
  File: `link.sh'
  Size: 25              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 149614      Links: 2
Access: (0644/-rw-r--r--)  Uid: ( 1000/     ksc)   Gid: ( 1001/     bca)
Access: 2025-03-22 06:47:42.924413309 +0400
Modify: 2025-02-08 09:04:26.057625148 +0400
Change: 2025-03-21 09:59:08.922536814 +0400
```

- **Size:** File size in bytes.
- **Blocks:** Number of allocated blocks.
- **Access:** Last accessed timestamp.
- **Modify:** Last modified timestamp.
- **Change:** Last metadata change timestamp.

(23) **alias :** The alias command in Unix is used to create shortcuts for longer commands. It allows users to define custom command names or abbreviations for frequently used commands.

Syntax: alias name='command'

Create an alias : alias l1='ls -l'

```
ksc@ubuntu:~$ l1
total 176
-rwxr--r-- 1 1001 1001     6 2025-03-12 07:54 abc.txt
-rwxrwxrwx 2 ksc  1001   204 2025-03-23 16:18 Add.sh
drwxr-xr-x 2 ksc  ksc   4096 2014-09-12 15:52 AK
-rwxr-xr-x 1 ksc  ksc    108 2025-03-26 10:17 a.sh
-rw-r--r-- 1 ksc  ksc   2684 2025-03-27 11:01 a.txt
drwxr-xr-x 3 ksc  ksc   4096 2025-02-24 06:15 Ayushi
-rw-r--r-- 1 ksc  ksc   1697 2025-03-24 20:03 b.txt
-rw-r--r-- 1 ksc  ksc    263 2025-03-25 06:41 cal.txt
-rw-r--r-- 1 ksc  ksc     20 2025-03-26 07:48 cmp1.txt
-rw-r--r-- 1 ksc  ksc      9 2025-03-26 07:48 cmp.txt
-rw-r--r-- 1 ksc  ksc     16 2025-03-25 10:47 comm.txt
-rw-r--r-- 1 ksc  ksc      9 2025-03-25 10:46 com.txt
drwxr-xr-x 2 ksc  ksc   4096 2025-03-12 08:05 demo
-rw-r--r-- 2 ksc  1001     0 2025-03-24 20:07 demo.sh
-rw-r--r-- 1 ksc  ksc   1734 2025-03-24 19:57 demo.txt
drwxr-xr-x 3 ksc  ksc   4096 2025-03-27 06:19 Desktop
-rw-r--r-- 1 ksc  ksc     10 2025-03-25 10:52 diff.txt
-rw-r--r-- 1 ksc  ksc     10 2025-03-25 10:52 dif.txt
-rw-r--r-- 1 ksc  ksc   1583 2025-03-23 17:52 dm. ?
```

List all aliases : alias

```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'
')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
ksc@ubuntu:~$ alias l1='ls -l'
ksc@ubuntu:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'
')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias l1='ls -l'
alias la='ls -A'
alias ll='ls -alF'
```

Remove an alias : unalias l1

```
ksc@ubuntu:~$ unalias l1
ksc@ubuntu:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'
')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

(24) **type:** The type command in Unix is used to determine how a command is interpreted by the shell—whether it is a built-in command, an alias, a function, or an external executable.

**Syntax**

type command_name

Check a Built-in Command

type cd

Check an External Command

type ls

```
ksc@ubuntu:~$ type cd
cd is a shell builtin
ksc@ubuntu:~$ type ls
ls is aliased to `ls --color=auto'
ksc@ubuntu:~$ type cat
cat is /bin/cat
ksc@ubuntu:~$ type ln
ln is /bin/ln
ksc@ubuntu:~$
```

## Operators in Redirection & Piping

### 1. Redirection Operators

1. Output Redirection (>, >>)

- Used to redirect the standard output (stdout) to a file.

| Operator | Description | Example |
|---|---|---|
| > | Redirects output to a file (overwrites existing content) | ls > output.txt |
| >> | Appends output to a file (does not overwrite) | ls >> output.txt |

```
ksc@ubuntu:~$ touch sy.txt
ksc@ubuntu:~$ ls > sy.txt
ksc@ubuntu:~$ cat sy.txt
a
abc.txt
Add.sh
AK
Ayushi
c
d
demo
demo.sh
demo.txt
Desktop
Documents
```

```
ksc@ubuntu:~$ ls >> sy.txt
ksc@ubuntu:~$ cat sy.txt
a
abc.txt
Add.sh
AK
Ayushi
c
d
demo
demo.sh
demo.txt
Desktop
Documents
```

*2. Input Redirection (<)*

- Used to take input from a file instead of standard input (stdin).

| Operator | Description | Example |
|---|---|---|
| < | Redirects input from a file | `sort < file.txt` |

```
ksc@ubuntu:~$ sort < new.txt
a
abc.txt
Add.sh
AK
Ayushi
c
d
demo
demo.sh
demo.txt
Desktop
Documents
Downloads
examples.desktop
```

*Here Document (<<)*

```
ksc@ubuntu:~$ ls << new.txt
> bash: warning: here-document at line 3 delimited by end-of-file (wanted `new.txt')
a         c         Desktop           link.sh  Pictures  SYBCAA       Videos
abc.txt   d         Documents         Music    Public    SYBCAB
Add.sh    demo      Downloads         new.txt  read.sh   SYBCAB.txt
AK        demo.sh   examples.desktop  odd.vi   student   symbolic.sh
Ayushi    demo.txt  extra             p1.sh    sum.sh    sy.txt
```

**2. Piping (|) :** Pipes (|) are used to send the output of one command as input to another command.

Filter output with grep : Lists only .txt files.

```
ksc@ubuntu:~$ ls -l | grep .txt
-rwxr--r-- 1 bca bca    6 2025-03-12 07:54 abc.txt
-rw-r--r-- 1 ksc ksc    0 2025-03-23 17:08 demo.txt
-rw-r--r-- 1 ksc ksc    0 2025-03-23 17:41 new.txt
-rwxrw---- 1 ksc ksc   27 2025-03-16 09:26 SYBCAB.txt
-rw-r--r-- 1 ksc ksc  458 2025-03-23 16:54 sy.txt
```

Count lines in a file using wc : Counts the number of lines in file.txt.

```
ksc@ubuntu:~$ cat sy.txt | wc -l
62
```

Display long output page-by-page using less : Lets you scroll through the ls -l output.

```
total 108
-rwxr-xr-x 1 ksc ksc     0 2025-03-11 20:09 a
-rwxr--r-- 1 bca bca     6 2025-03-12 07:54 abc.txt
-rwxrwxrwx 2 ksc bca   204 2025-03-23 16:18 Add.sh
drwxr-xr-x 2 ksc ksc  4096 2014-09-12 15:52 AK
drwxr-xr-x 3 ksc ksc  4096 2025-02-24 06:15 Ayushi
drwxr-xr-x 2 ksc ksc  4096 2025-03-12 08:20 c
drwxr-xr-x 2 ksc ksc  4096 2025-03-12 08:20 d
drwxr-xr-x 2 ksc ksc  4096 2025-03-12 08:05 demo
-rw-r--r-- 2 ksc bca   214 2025-03-23 15:47 demo.sh
```

Sort and remove duplicates : Sorts the contents of names.txt and removes duplicate lines.

```
ksc@ubuntu:~$ cat sy.txt | sort | uniq
a
abc.txt
Add.sh
AK
Ayushi
c
d
demo
demo.sh
demo.txt
Desktop
Documents
Downloads
```

### Finding Patterns in Files

1. **grep :** The grep command is the most commonly used tool for searching patterns in files.

Syntax :
$grep "pattern" filename

```
ksc@ubuntu:~$ grep "SYBCAB" /home/ksc/sy.txt
SYBCAB
SYBCAB.txt
SYBCAB
SYBCAB.txt
```

2. **fgrep** (fixed-string grep): This command is used to search for literal strings in files, without interpreting regular expressions.

Syntax :
$fgrep "pattern" filename

```
ksc@ubuntu:~$ fgrep "SYBCAB" /home/ksc/sy.txt
SYBCAB
SYBCAB.txt
SYBCAB
SYBCAB.txt
ksc@ubuntu:~$ fgrep *.txt
sy.txt:abc.txt
sy.txt:abc.txt
ksc@ubuntu:~$ fgrep *.sh
demo.sh:Add.sh
link.sh:Add.sh
fgrep: read.sh: Permission denied
symbolic.sh:Add.sh
```

3. **egrep** : egrep (Extended grep) is used to search for patterns using **extended regular expressions (ERE)**.

Syntax :
$egrep "pattern" filename

```
ksc@ubuntu:~$ egrep "SYBCAA|SYBCAB" *.sh
demo.sh:SYBCAA
demo.sh:SYBCAB
demo.sh:SYBCAB.txt
link.sh:SYBCAA
link.sh:SYBCAB
link.sh:SYBCAB.txt
egrep: read.sh: Permission denied
symbolic.sh:SYBCAA
symbolic.sh:SYBCAB
symbolic.sh:SYBCAB.txt
```

## Working with columns and fields

1. **cut :** The cut command in Unix is used to extract specific sections (columns) from lines of text or files.

```
ksc@ubuntu:~$ cut -c 1-5 demo.txt
Hello
Good
ksc@ubuntu:~$ cut -c 1-6 demo.txt
Hello
Good M
```

2. **paste :** The paste command is used to merge lines of files **horizontally** (side by side) by joining corresponding lines from multiple files.

```
ksc@ubuntu:~$ paste demo.txt new.txt
Hello Dear Students     Hello SYBCA
Good Morning
ksc@ubuntu:~/SYBCAB$ cat > Sid.txt
1
2
3
4
5
ksc@ubuntu:~/SYBCAB$ cat > snm.txt
a
b
c
d
e
ksc@ubuntu:~/SYBCAB$ paste sid.txt snm.txt
paste: sid.txt: No such file or directory
ksc@ubuntu:~/SYBCAB$ paste Sid.txt snm.txt
1       a
2       b
3       c
4       d
5       e
```

3. **join** : The join command in Unix is used to merge two files based on a common field.

```
ksc@ubuntu:~/SYBCAB$ cat Sid.txt
1
2
3
4
5
ksc@ubuntu:~/SYBCAB$ cat snm.txt
1 a
2 b
3 c
4 d
5 e
ksc@ubuntu:~/SYBCAB$ join Sid.txt snm.txt
1 a
2 b
3 c
4 d
5 e
```

## Tools for sorting

1. **sort :** The sort command in Unix is used to arrange lines of text files in a specified order, either alphabetically or numerically.

```
ksc@ubuntu:~/SYBCAB$ cat > sort.txt
Zebra
Ant
Bat
Cat
Watch
ksc@ubuntu:~/SYBCAB$ sort sort.txt
Ant
Bat
Cat
Watch
Zebra
ksc@ubuntu:~/SYBCAB$ sort -r sort.txt
Zebra
Watch
Cat
Bat
Ant
```

2. **uniq :** The uniq command in Unix is used to remove duplicate lines from a sorted file. It only removes adjacent duplicate lines, so you often use it with sort first.

```
ksc@ubuntu:~/SYBCAB$ sort sort.txt | uniq
Ant
Bat
Cat
Watch
Zebra
```

## Comparing files

1. **cmp :** The cmp command in Unix is used to compare two files **byte by byte** and report the first difference it finds.

```
ksc@ubuntu:~/SYBCAB$ cmp Sid.txt snm.txt
Sid.txt snm.txt differ: byte 2, line 1
ksc@ubuntu:~/SYBCAB$ cp snm.txt snm
ksc@ubuntu:~/SYBCAB$ cmp snm.txt snm
ksc@ubuntu:~/SYBCAB$
```

2. **comm :** The comm command in Unix is used to compare two sorted files **line by line** and display their common and unique lines in three separate columns. It is useful for finding similarities and differences between files.

```
ksc@ubuntu:~/SYBCAB$ comm Sid.txt snm.txt
1
        1 a
2
        2 b
3
        3 c
4
        4 d
5
        5 e
ksc@ubuntu:~/SYBCAB$ cat > id.txt
1
2
3
4
5
ksc@ubuntu:~/SYBCAB$ comm Sid.txt id.txt
                1
                2
                3
                4
                5
```

3. **diff :** The diff command in Unix compares two files **line by line** and shows the differences between them.

```
ksc@ubuntu:~/SYBCAB$ diff Sid.txt snm.txt
1,5c1,5
< 1
< 2
< 3
< 4
< 5
---
> 1 a
> 2 b
> 3 c
> 4 d
> 5 e
ksc@ubuntu:~/SYBCAB$ diff Sid.txt id.txt
```

## Changing Information in Files: tr, sed

1. **tr :** The tr (translate) command in UNIX is used for translating or deleting characters from input. It is commonly used for replacing, squeezing, or deleting specific characters from input text.

Translate characters:

```
ksc@ubuntu:~$ echo "Hello Dear SYBCA Students" | tr 'a-z' 'A-Z'
HELLO DEAR SYBCA STUDENTS
```

Delete characters (-d option):

```
ksc@ubuntu:~$ echo "Hello Dear SYBCA Students" 123456789 | tr -d '0-9'
Hello Dear SYBCA Students
```

Squeeze repeated characters (-s option):

```
ksc@ubuntu:~$ echo "Helloooo Dear SYBCA Students"  | tr -s 'o'
Hello Dear SYBCA Students
```

Replace spaces with newlines:

```
ksc@ubuntu:~$ echo "Helloooo Dear SYBCA Students"  | tr ' ' '\n'
Helloooo
Dear
SYBCA
Students
```

2. **sed :** The sed (Stream Editor) command in UNIX is used for text processing, allowing users to perform find-and-replace operations, delete lines, insert text, and more.

Replace a String (Substitution)

```
ksc@ubuntu:~$ sed 's/Students/Student/' demo.txt
Hello Dear Student
Good Morning
```

Delete Lines

```
ksc@ubuntu:~$ cat demo.txt
Hello Dear Students
Good Morning
ksc@ubuntu:~$ sed '2d' demo.txt
Hello Dear Students
```

Print Specific Lines

```
ksc@ubuntu:~$ sed -n '2,4p' sy.txt
abc.txt
Add.sh
AK
ksc@ubuntu:~$ sed -n '1,10p' sy.txt
a
abc.txt
Add.sh
AK
Ayushi
c
d
demo
demo.sh
demo.txt
```

## Examining File Contents : od

1. **od :** The od (Octal Dump) command in UNIX is used to display the contents of a file in various formats, such as octal, hexadecimal, ASCII, and binary. It's useful for inspecting binary files, debugging, or analyzing non-printable characters in text files.

Default Output (Octal Representation)

```
ksc@ubuntu:~$ od sy.txt
0000000 005141 061141 027143 074164 005164 062101 027144 064163
0000020 040412 005113 074501 071565 064550 061412 062012 062012
0000040 066545 005157 062544 067555 071456 005150 062544 067555
0000060 072056 072170 042012 071545 072153 070157 042012 061557
```

Display Hexadecimal Output

```
ksc@ubuntu:~$ od -x sy.txt
0000000 0a61 6261 2e63 7874 0a74 6441 2e64 6873
0000020 410a 0a4b 7941 7375 6968 630a 640a 640a
0000040 6d65 0a6f 6564 6f6d 732e 0a68 6564 6f6d
0000060 742e 7478 440a 7365 746b 706f 440a 636f
0000100 6d75 6e65 7374 440a 776f 6c6e 616f 7364
```

Display ASCII Characters (-c) (Shows text characters along with escape sequences for non-printable characters.)

```
ksc@ubuntu:~$ od -c sy.txt
0000000   a  \n   a   b   c   .   t   x   t  \n   A   d   d   .   s   h
0000020  \n   A   K  \n   A   y   u   s   h   i  \n   c  \n   d  \n   d
0000040   e   m   o  \n   d   e   m   o   .   s   h  \n   d   e   m   o
0000060   .   t   x   t  \n   D   e   s   k   t   o   p  \n   D   o   c
0000100   u   m   e   n   t   s  \n   D   o   w   n   l   o   a   d   s
0000120  \n   e   x   a   m   p   l   e   s   .   d   e   s   k   t   o
0000140   p  \n   e   x   t   r   a  \n   l   i   n   k   .   s   h  \n
```

Display Decimal Output: (Shows decimal (base 10) representation of file contents.)

```
ksc@ubuntu:~$ od -d sy.txt
0000000  2657 25185 11875 30836  2676 25665 11876 26739
0000020 16650  2635 31041 29557 26984 25354 25610 25610
0000040 28005  2671 25956 28525 29486  2664 25956 28525
0000060 29742 29816 17418 29541 29803 28783 17418 25455
0000100 28021 28261 29556 17418 30575 27758 24943 29540
0000120 25866 24952 28781 25964 11891 25956 27507 28532
```

Display Binary Output (-b) (Shows the file in **octal byte** format.)

```
ksc@ubuntu:~$ od -b sy.txt
0000000 141 012 141 142 143 056 164 170 164 012 101 144 144 056 163 150
0000020 012 101 113 012 101 171 165 163 150 151 012 143 012 144 012 144
0000040 145 155 157 012 144 145 155 157 056 163 150 012 144 145 155 157
0000060 056 164 170 164 012 104 145 163 153 164 157 160 012 104 157 143
0000100 165 155 145 156 164 163 012 104 157 167 156 154 157 141 144 163
```

## Tools for mathematical calculations: bc, factor

1. **bc :** The bc (Basic Calculator) command in UNIX is a command-line calculator that supports **arithmetic operations, variables, functions, and scripting**.

Basic Arithmetic

```
ksc@ubuntu:~$ echo '3+5' | bc
8
ksc@ubuntu:~$ echo '5-3' | bc
2
ksc@ubuntu:~$ echo '5/2' | bc
2
ksc@ubuntu:~$ echo '5*2' | bc
10
ksc@ubuntu:~$ echo '5%2' | bc
1
```

Floating-Point (Decimal) Calculations (scale=5 sets precision to 5 decimal places.)

```
ksc@ubuntu:~$ echo 'scale=5; 5/3' | bc
1.66666
ksc@ubuntu:~$ echo 'scale=2; 5/3' | bc
1.66
ksc@ubuntu:~$ echo 'a=10; b=5; a*b' | bc
50
ksc@ubuntu:~$ echo '2^3' | bc
8
ksc@ubuntu:~$ echo 'scale=5; sqrt(25)' | bc
5.00000
ksc@ubuntu:~$ echo '$((5+3))'
$((5+3))
ksc@ubuntu:~$ echo $((5+3))
```

Using Variables

```
ksc@ubuntu:~$ echo 'a=10; b=5; a*b' | bc
50
```

Exponents (^ Operator)

```
ksc@ubuntu:~$ echo '2^3' | bc
8
```

Square Root (sqrt)

```
ksc@ubuntu:~$ echo 'scale=5; sqrt(25)' | bc
5.00000
```

Using bc as a Shell Calculator

```
ksc@ubuntu:~$ echo $((5+3))
8
```

Convert Decimal to Binary

```
ksc@ubuntu:~$ echo 'obase=2; 10' | bc
1010
```

Convert Binary to Decimal

```
ksc@ubuntu:~$ echo 'ibase=2; 1010' | bc
10
```

**2. factor :** The factor command in Unix is used to display the prime factors of a given integer. It takes an integer as an argument and prints its prime factorization.

**Factorizing a number**:

```
ksc@ubuntu:~$ factor 60
60: 2 2 3 5
```

**Using** factor **without an argument** (interactive mode):

```
ksc@ubuntu:~$ factor
30
30: 2 3 5
15
15: 3 5
20
20: 2 2 5
```

**Using** factor **with multiple numbers**:

```
ksc@ubuntu:~$ factor 45 100 98
45: 3 3 5
100: 2 2 5 5
98: 2 7 7
```

## Monitoring Input and Output :tee, script

1. **tee :** The tee command in UNIX is used to **read from standard input (**stdin**) and write to both standard output (**stdout**) and one or more files simultaneously**. It is commonly used for logging output while still displaying it on the screen.

Save output to a file (This lists files in the directory and saves the output to demo.txt while also displaying it on the screen.)

```
ksc@ubuntu:~$ ls -l | tee demo.txt
total 124
-rwxr-xr-x 1 ksc ksc    0 2025-03-11 20:09 a
-rwxr--r-- 1 bca bca    6 2025-03-12 07:54 abc.txt
-rwxrwxrwx 2 ksc bca  204 2025-03-23 16:18 Add.sh
drwxr-xr-x 2 ksc ksc 4096 2014-09-12 15:52 AK
drwxr-xr-x 3 ksc ksc 4096 2025-02-24 06:15 Ayushi
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:20 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:20 d
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:05 demo
-rw-r--r-- 2 ksc bca  214 2025-03-23 15:47 demo.sh
-rw-r--r-- 1 ksc ksc    0 2025-03-24 19:57 demo.txt
```

Append Output to a File (-a option) (The -a option appends the output to demo.txt instead of overwriting it.)

```
ksc@ubuntu:~$ ls -l | tee -a demo.sh
total 128
-rwxr-xr-x 1 ksc ksc    0 2025-03-11 20:09 a
-rwxr--r-- 1 bca bca    6 2025-03-12 07:54 abc.txt
-rwxrwxrwx 2 ksc bca  204 2025-03-23 16:18 Add.sh
drwxr-xr-x 2 ksc ksc 4096 2014-09-12 15:52 AK
drwxr-xr-x 3 ksc ksc 4096 2025-02-24 06:15 Ayushi
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:20 c
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:20 d
drwxr-xr-x 2 ksc ksc 4096 2025-03-12 08:05 demo
-rw-r--r-- 2 ksc bca 1734 2025-03-24 19:57 demo.sh
```

Write to Multiple Files (Saves the command output to both a.txt and b.txt while displaying it.)

```
ksc@ubuntu:~$ ls -l | tee a.txt b.txt
total 120
-rwxr--r-- 1 bca bca    6 2025-03-12 07:54 abc.txt
-rwxrwxrwx 2 ksc bca  204 2025-03-23 16:18 Add.sh
drwxr-xr-x 2 ksc ksc 4096 2014-09-12 15:52 AK
-rw-r--r-- 1 ksc ksc    0 2025-03-24 20:03 a.txt
drwxr-xr-x 3 ksc ksc 4096 2025-02-24 06:15 Ayushi
-rw-r--r-- 1 ksc ksc    0 2025-03-24 20:02 b.txt
```

2. **script :** The script command in Unix is used to record a terminal session. It captures all input and output during the session and saves it to a file, allowing users to review or share their terminal activity later.

**Record Terminal Session**

**Ex script file name**

```
ksc@ubuntu:~$ script link.sh
Script started, file is link.sh
```

**Append to an Existing Log File (**-a **option) (**Appends new session data to link.sh instead of overwriting it.**)**

```
ksc@ubuntu:~$ script -a link.sh
Script started, file is link.sh
```

Record Without Showing Start and End Messages (-q option) (Runs script in quiet mode, suppressing the "Script started" and "Script done" messages.)

```
ksc@ubuntu:~$ script -q link.sh
```

**Set a Timeout to Auto-Flush Output (**-t **option) (**Flushes output every 2 seconds, ensuring real-time logging.**)**

```
ksc@ubuntu:~$ script -t link.sh
Script started, file is link.sh
0.099316 32
           ksc@ubuntu:~$ 0.336197 1
                                  s7.480225 4
                                            2.583987 34
```

## Tools For Displaying Date and Time: cal, date

1. **cal :** The cal command in Linux and Unix-like operating systems displays a calendar in the terminal.

This shows the current month's calendar

```
ksc@ubuntu:~$ cal
     March 2025
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Specific Year:

```
ksc@ubuntu:~$ cal 2025
                                2025

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

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

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

      October               November                December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
            1  2  3  4                  1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
```

Specific Month and Year:

```
ksc@ubuntu:~$ cal 04 2025
     April 2025
Su Mo Tu We Th Fr Sa
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Julian Calendar: (Displays the calendar with Julian dates (day numbers in the year))

```
ksc@ubuntu:~$ cal -j
        March 2025
 Su   Mo   Tu   We   Th   Fr   Sa
                               60
 61   62   63   64   65   66   67
 68   69   70   71   72   73   74
 75   76   77   78   79   80   81
 82   83   84   85   86   87   88
 89   90
```

Show Previous, Current, and Next Month:

```
ksc@ubuntu:~$ cal -3
   February 2025          March 2025           April 2025
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1                     1           1  2  3  4  5
 2  3  4  5  6  7  8   2  3  4  5  6  7  8   6  7  8  9 10 11 12
 9 10 11 12 13 14 15   9 10 11 12 13 14 15  13 14 15 16 17 18 19
16 17 18 19 20 21 22  16 17 18 19 20 21 22  20 21 22 23 24 25 26
23 24 25 26 27 28     23 24 25 26 27 28 29  27 28 29 30
                      30 31
```

2. **date :** The date command in Linux and Unix-like operating systems displays or sets the system date and time.

Displays the current date and time.

Date

```
ksc@ubuntu:~$ date
Mon Mar 24 20:15:13 RET 2025
```

Only Date: date "+%Y-%m-%d"

```
ksc@ubuntu:~$ date '+%Y-%m-%d'
2025-03-24
```

Only Time: date "+%H:%M:%S"

```
ksc@ubuntu:~$ date '+%H:%M:%s'
20:17:1742833034
```

Setting the Date and Time (Requires Root Privileges): sudo date 032414302025

```
ksc@ubuntu:~$ sudo date 032414302025
[sudo] password for ksc:
Mon Mar 24 14:30:00 RET 2025
```

## Process Related Commands: ps, sleep

1. **ps :** The ps command in Linux and Unix-like operating systems is used to display information about active processes.

```
ksc@ubuntu:~$ ps
  PID TTY          TIME CMD
 1712 pts/0    00:00:00 bash
 1950 pts/0    00:00:00 ps
```

2. **sleep :** The sleep command in Linux and Unix-like operating systems is used to pause script execution for a specified amount of time.

**Pause for 5 seconds**: sleep 5
**Pause for 2 minutes**: sleep 2m
**Pause for 1 hour**: sleep 1h

```
ksc@ubuntu:~$ sleep 5
```

**Use `sleep` in a script**:
echo "Start"
sleep 3
echo "End after 3 seconds"

```
ksc@ubuntu:~$ cat > sleep.sh
echo "Before Sleep"
sleep 2
echo "After Sleep"
ksc@ubuntu:~$ chmod +x sleep.sh
ksc@ubuntu:~$ ./sleep.sh
Before Sleep
After Sleep
ksc@ubuntu:~$
```

### Communications

1. **telnet :** The telnet command in Unix/Linux is used to establish a remote connection to another system over the Telnet protocol. It allows users to log into remote machines and execute commands as if they were physically present.

Connect to a remote server : telnet <hostname or IP> <port>

- hostname or IP address → The remote server's domain name or IP address you want to connect to.
- port → (Optional) The port number on which the Telnet service is running (default is port 23).

Check if a port is open on a remote system : telnet example.com 80

**Connecting to a Remote Server**

```
gfg0622@Anshu-ka-lappi:~$ telnet 192.168.1.1
```

**To check if** port 80 **(HTTP) is open on a website:**

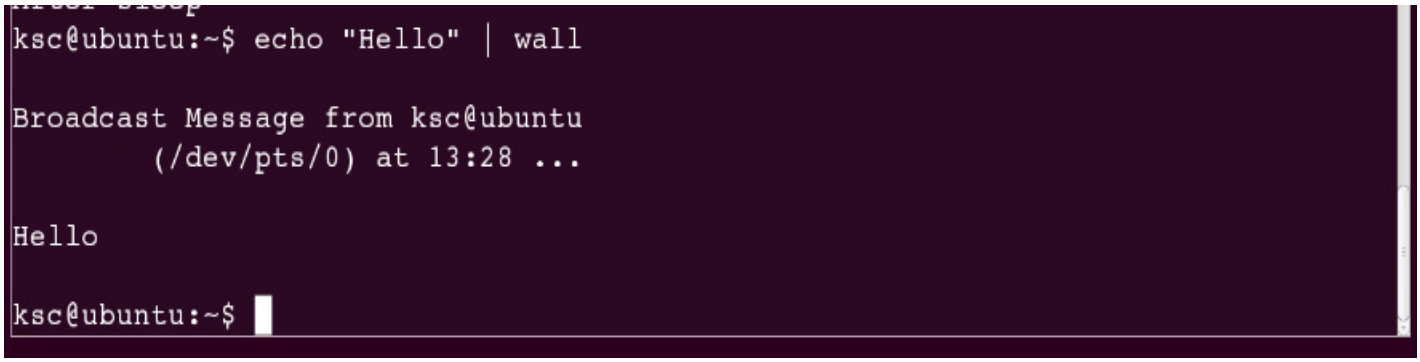
```
gfg0622@Anshu-ka-lappi:~$ telnet example.com 80
Trying 23.215.0.138...
Connected to example.com.
Escape character is '^]'.
Connection closed by foreign host.
```

2. **ping :** The PING (Packet Internet Groper) command is used to check the network connectivity between the host and server/host.

**Syntax** : `ping [options] host_or_IP_address`

```
administrator@GFG19566-LAPTOP:~/practice$ ping www.google.com
PING www.google.com (142.250.194.100) 56(84) bytes of data.
64 bytes from del12s04-in-f4.1e100.net (142.250.194.100): icmp_seq=1 ttl=116 time=1.60 ms
64 bytes from del12s04-in-f4.1e100.net (142.250.194.100): icmp_seq=2 ttl=116 time=1.15 ms
64 bytes from del12s04-in-f4.1e100.net (142.250.194.100): icmp_seq=3 ttl=116 time=1.17 ms
64 bytes from del12s04-in-f4.1e100.net (142.250.194.100): icmp_seq=4 ttl=116 time=1.14 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.136/1.264/1.599/0.193 ms
```

**Wall :** The wall command in Unix is used to send a message to all logged-in users. It's commonly used by system administrators to broadcast important messages, such as system shutdown warnings.



Assignment : 3

1. Explain redirection and piping operators.
2. Explain the architecture of UNIX
3. Explain chmod, chown, chgrp command.
4. Explain rmdir ,mkdir, wc, chmod, cp, mv with example.
5. Explain Process Related Commands