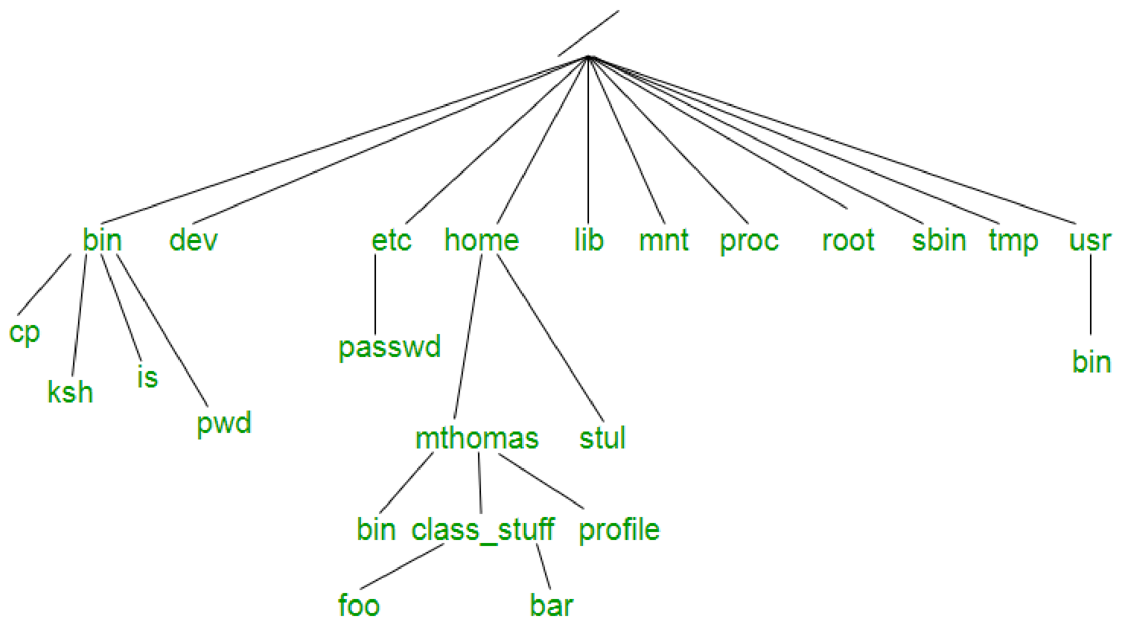
UNIT-V

<u>Files and Directories in UNIX:</u>

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 a 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.



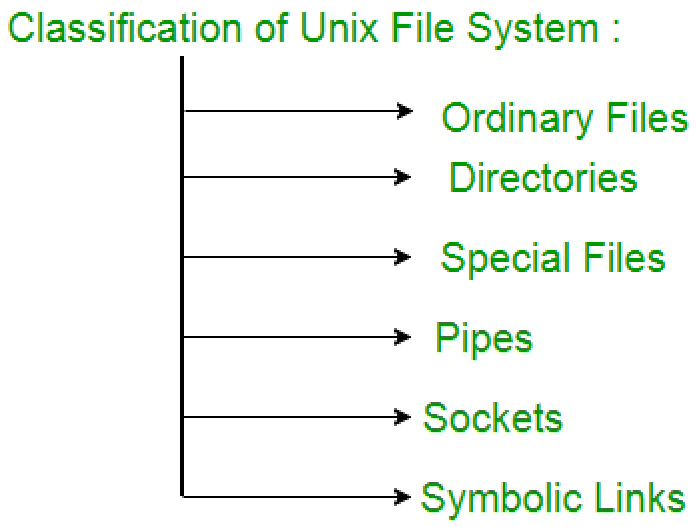
Directories or Files and their description –

- **/ :** 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.
- **/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.
- **/var/tmp :** A place for temporary files which should be preserved between system reboots.

Types of Unix files – The UNIX files system contains several different 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.

2. Directories – 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.

3. Special Files – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. **Device or special files** are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory.

On UNIX systems there are two flavors of special files for each device, character special files and block special files :

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

For terminal devices, it’s one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

- In long-format output of ls -l, character special files are marked by the “c” symbol.
- In long-format output of ls -l, block special files are marked by the “b” symbol.

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

In long-format output of ls –l , named pipes are marked by the “p” symbol.

5. Sockets – A Unix socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server

application framework. In essence, it is a stream of data, very similar to network stream (and network sockets), but all the transactions are local to the filesystem.
In long-format output of ls -l, Unix sockets are marked by “s” symbol.

6. Symbolic Link – Symbolic link is used for referencing some other file of the file system.Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to. If we delete the soft link itself , the data file would still be there.If we delete the source file or move it to a different location, symbolic file will not function properly.

File structure:

A File Structure needs to be predefined format in such a way that an operating system understands. It has an exclusively defined structure, which is based on its type.

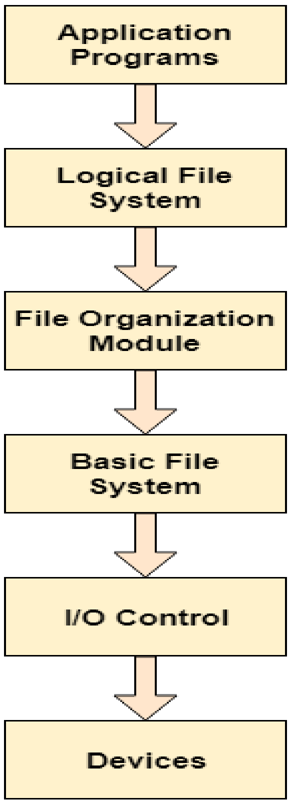
Three types of files structure in OS:

- A text file: It is a series of characters that is organized in lines.
- An object file: It is a series of bytes that is organized into blocks.
- A source file: It is a series of functions and processes.

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.

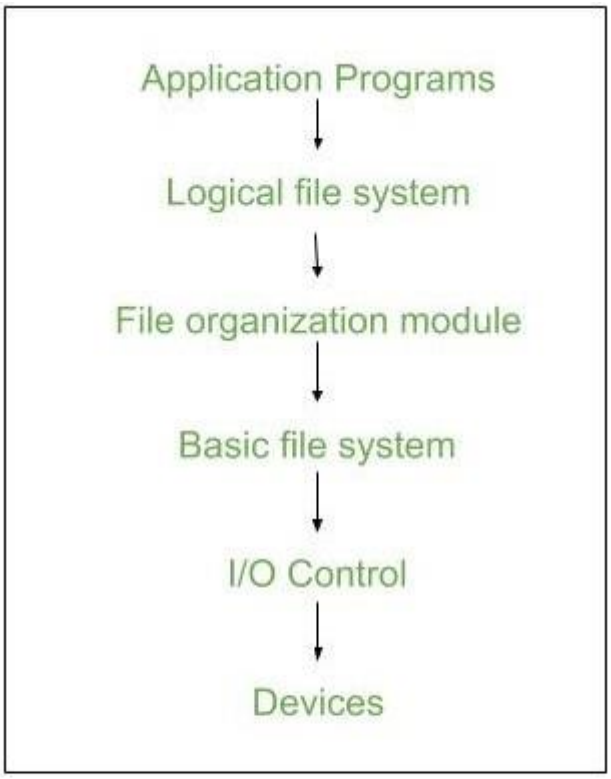


- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

File System Implementation of Operating System Functions:

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

File system organized in many layers :



- **I/O Control level –**

  Device drivers acts as interface between devices and Os, they help to transfer data between disk and main memory. It takes block number a input and as output it gives low level hardware specific instruction.
  /li>
- **Basic file system –**

  It Issues general commands to device driver to read and write physical blocks on disk.It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.
- **File organization Module –**

It has information about files, location of files and their logical and physical blocks.Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.

- **Logical file system –**

  It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

Advantages :

1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

Disadvantages :

If we access many files at same time then it results in low performance.

<u>File permission:</u>

we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows −

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

Basic Operation on Files:

A file is a collection of logically related data that is recorded on the secondary storage in the form of sequence of operations. The content of the files are defined by its creator who is creating the file. The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

1.Create operation:

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open operation:

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write operation:

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read operation:

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek operation:

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. Delete operation:

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate operation:

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close operation:

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

<u>Changing Permission Modes:</u>

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Sr.No.	**Chmod operator & Description**
1	+ Adds the designated permission(s) to a file or directory.

2	**-** Removes the designated permission(s) from a file or directory.
3	**=** Sets the designated permission(s).

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you can combine these commands on a single line −

```
$chmod o+wx,u-x,g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x

2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

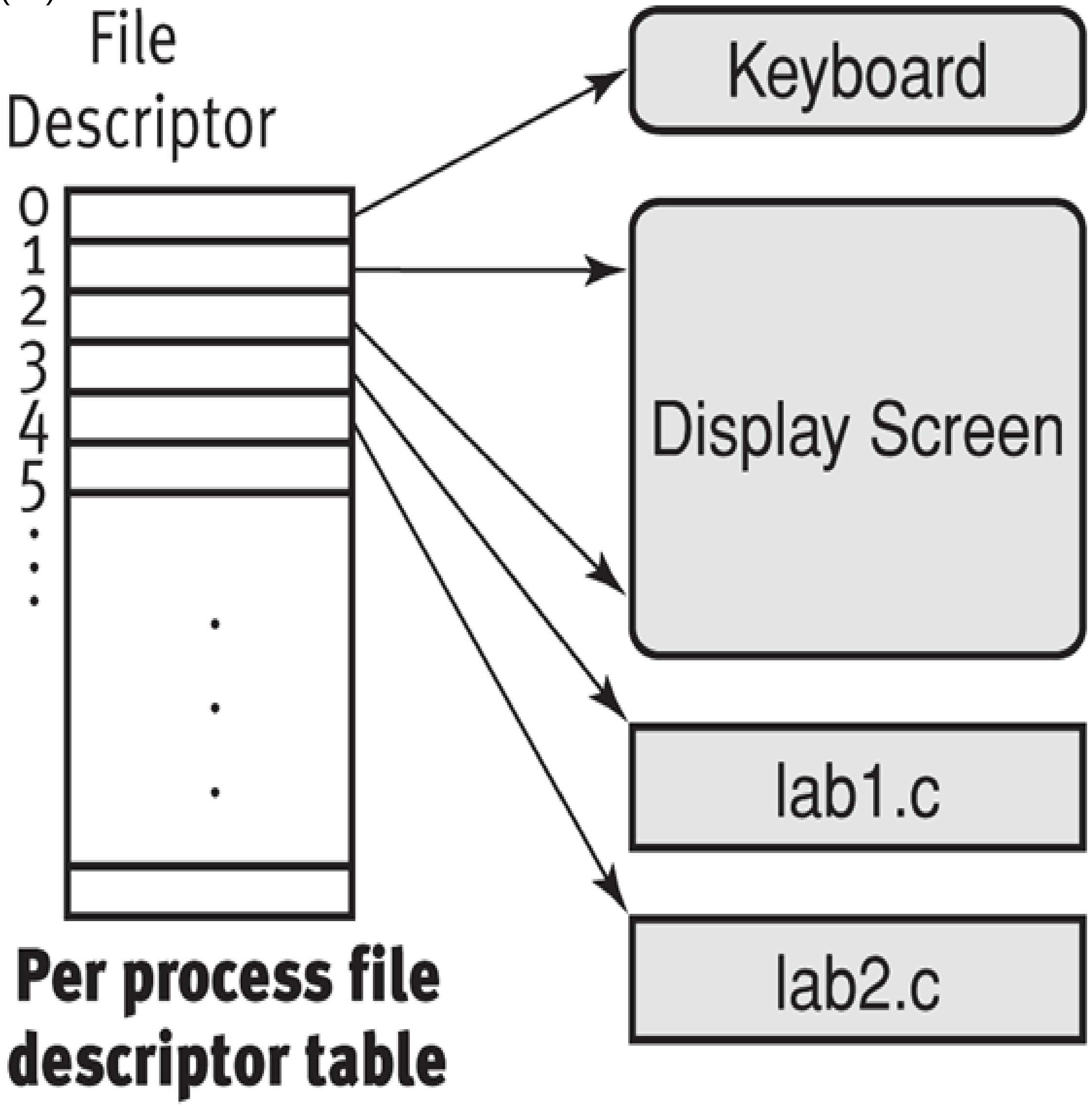
<u>Standard files:</u>

Every process in Linux is provided with three open files(usually called file descriptor). These files are the standard input, output and error files. By default :

- **Standard Input** is the keyboard, abstracted as a file to make it easier to write shell scripts.
- **Standard Output** is the shell window or the terminal from which the script runs, abstracted as a file to again make writing scripts & program easier
- **Standard error** is the same as standard output:the shell window or terminal from which the script runs.

(OR)

- **`stdin`**
- The primary input channel for the program. Default source is the user's keyboard, but it can easily be switched to be from another process via a pipe (|) or a file via a file redirection (<).
- **`stdout`**
- The primary output channel for program's data output. Default destination is the user's screen, but it can easily be switched to another process via a pipe (|) or a file via a file redirection (>).
- **`stderr`**
- Output channel for error messages. Default destination is the user's screen, but it can be switched to another process via a pipe (|) or a file via a file redirection (>). The details of how to do that will be covered later. See File Redirection.



-

    - Logical view of file descriptors

Processes Inspecting Files:

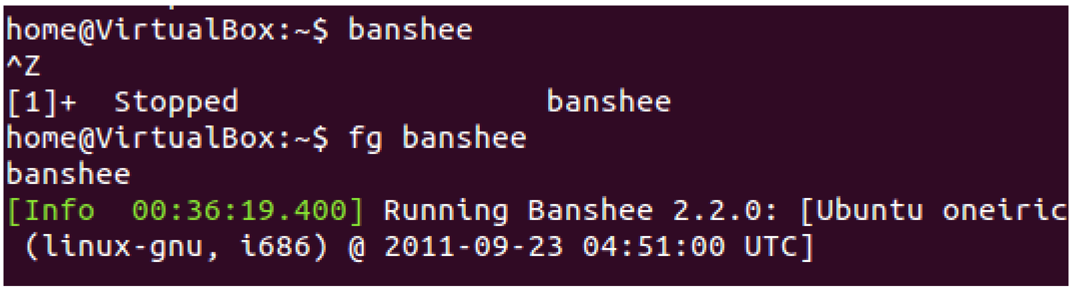
Fg

You can use the command “fg” to continue a program which was stopped and bring it to the foreground.

The simple syntax for this utility is:

fg jobname
Example

1. Launch ‘banshee’ music player
2. Stop it with the ‘ctrl +z’ command
3. Continue it with the ‘fg’ utility.

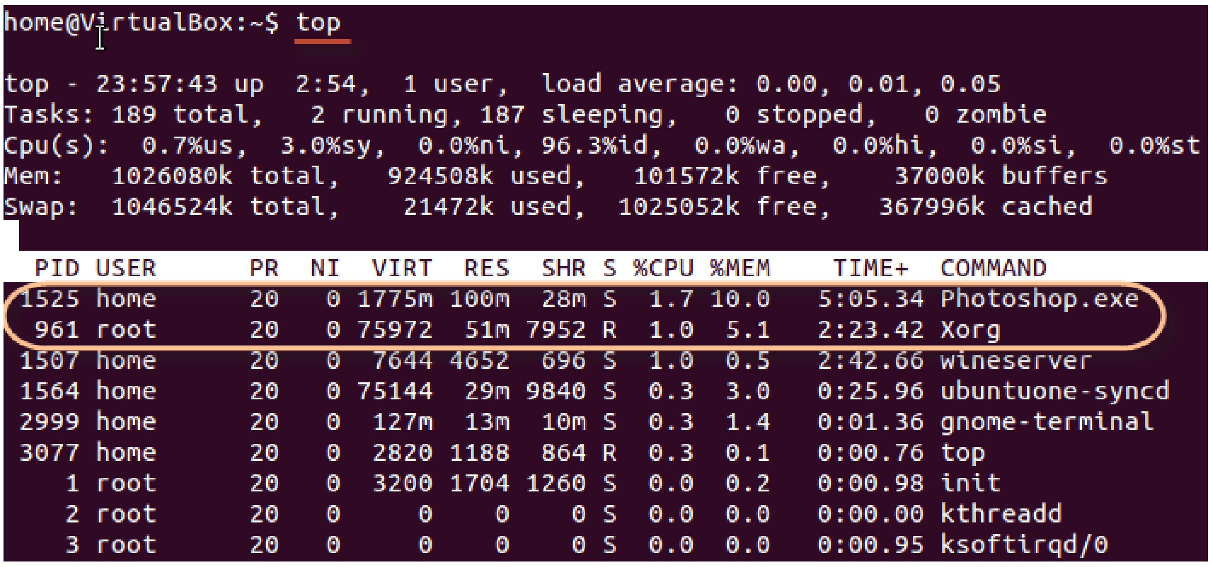


```
home@VirtualBox:~$ banshee
^Z
[1]+  Stopped                 banshee
home@VirtualBox:~$ fg banshee
banshee
[Info  00:36:19.400] Running Banshee 2.2.0: [Ubuntu oneiric
 (linux-gnu, i686) @ 2011-09-23 04:51:00 UTC]
```

Let’s look at other important commands to manage processes –

Top

This utility tells the user about all the running processes on the Linux machine.



```
home@VirtualBox:~$ top

top - 23:57:43 up  2:54,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 189 total,   2 running, 187 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7%us,  3.0%sy,  0.0%ni, 96.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1026080k total,   924508k used,   101572k free,    37000k buffers
Swap:  1046524k total,    21472k used,  1025052k free,   367996k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1525 home      20   0 1775m 100m  28m S  1.7 10.0   5:05.34 Photoshop.exe
  961 root      20   0 75972  51m 7952 R  1.0  5.1   2:23.42 Xorg
 1507 home      20   0  7644 4652  696 S  1.0  0.5   2:42.66 wineserver
 1564 home      20   0 75144  29m 9840 S  0.3  3.0   0:25.96 ubuntuone-syncd
 2999 home      20   0  127m  13m  10m S  0.3  1.4   0:01.36 gnome-terminal
 3077 home      20   0  2820 1188  864 R  0.3  0.1   0:00.76 top
    1 root      20   0  3200 1704 1260 S  0.0  0.2   0:00.98 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.95 ksoftirqd/0
```

Press ‘q’ on the keyboard to move out of the process display.

Method 1: Foreground Process : Every process when started runs in foreground by default, receives input from the keyboard, and sends output to the screen. When issuing pwd command

$ ls pwd

Output:

```
$ /home/geeksforgeeks/root
```

When a command/process is running in the foreground and is taking a lot of time, no other processes can be run or started because the prompt would not be available until the program finishes processing and comes out.

Method 2: Background Process: It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be done in parallel with the process running in the background since they do not have to wait for the previous process to be completed.

Adding & along with the command starts it as a background process

$ pwd &

Since pwd does not want any input from the keyboard, it goes to the stop state until moved to the foreground and given any data input. Thus, on pressing Enter:

Output:

```
[1]   +   Done                pwd

$
```

That first line contains information about the background process – the job number and the process ID. It tells you that the ls command background process finishes successfully. The second is a prompt for another command.

Tracking ongoing processes

ps (Process status) can be used to see/list all the running processes.

$ ps

```
PID       TTY      TIME        CMD
19        pts/1    00:00:00    sh
24        pts/1    00:00:00    ps
```

For more information -f (full) can be used along with ps

$ ps –f

```
UID      PID  PPID C STIME    TTY        TIME CMD
52471     19     1 0 07:20    pts/1  00:00:00f     sh
52471     25    19 0 08:04    pts/1  00:00:00      ps -f
```

For single-process information, ps along with process id is used

$ ps 19

```
PID      TTY      TIME       CMD
19       pts/1    00:00:00   sh
```

For a running program (named process) **Pidof** finds the process id's (pids)

Fields described by ps are described as:

- **UID**: User ID that this process belongs to (the person running it)
- **PID**: Process ID
- **PPID**: Parent process ID (the ID of the process that started it)
- **C**: CPU utilization of process
- **STIME**: Process start time
- **TTY**: Terminal type associated with the process
- **TIME**: CPU time is taken by the process
- **CMD**: The command that started this process

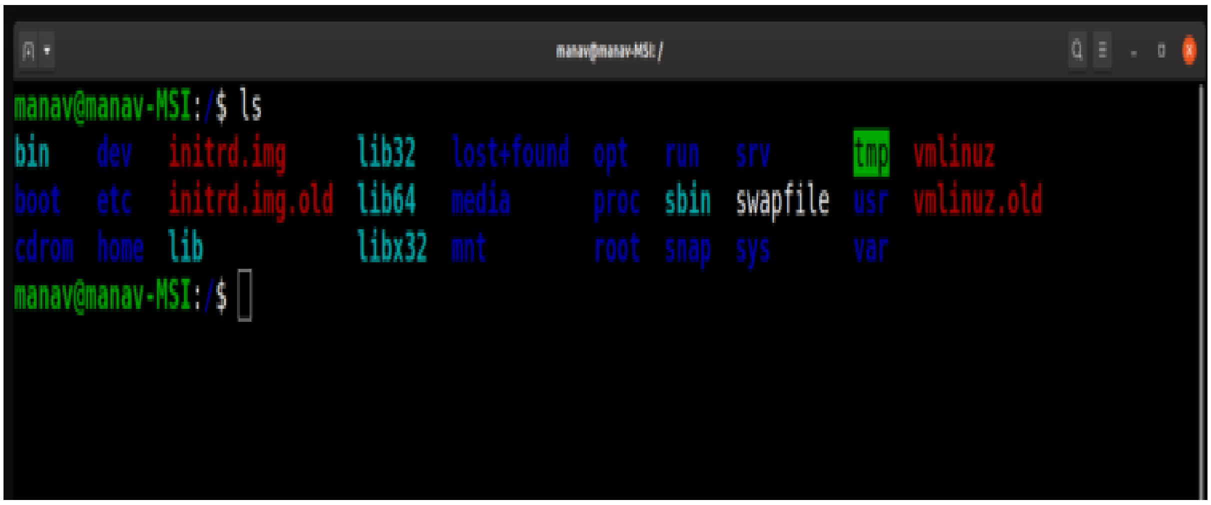
Operating On Files:

In Unix systems, there are two types of special files for each device, i.e. character special files and block special files.

1. **Files Listing**

To perform Files listings or to list files and directories ls command is used

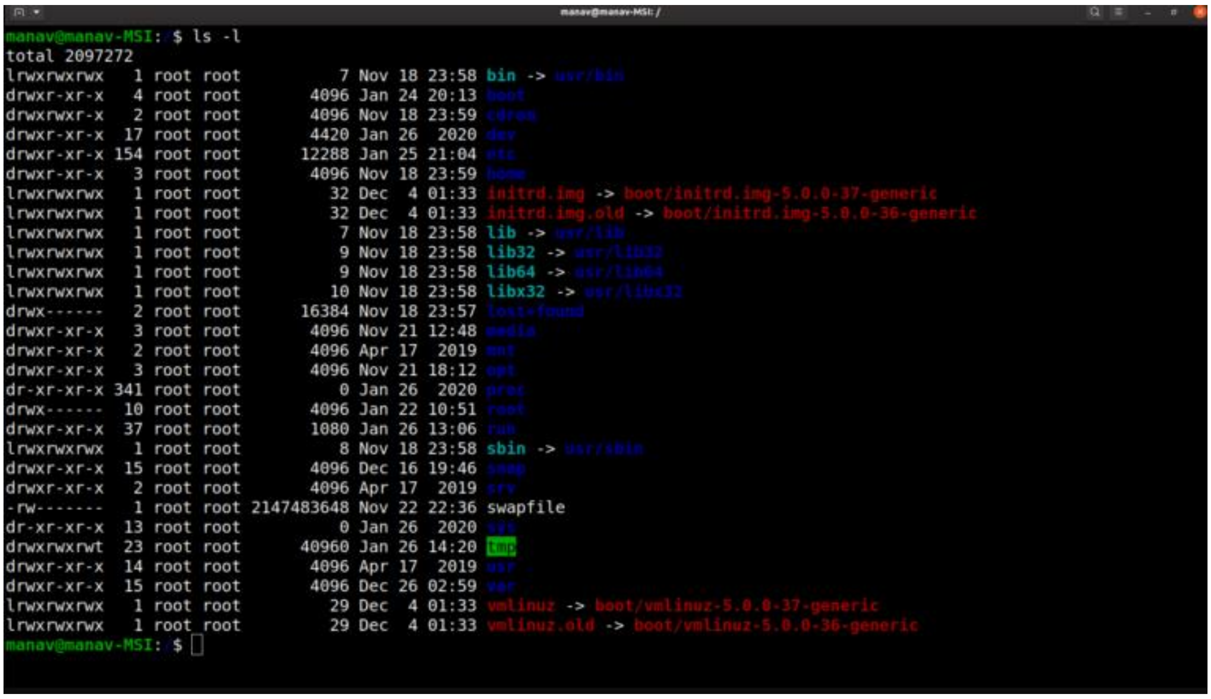
$ls



```
manav@manav-MSI:/$ ls
bin    dev   initrd.img      lib32   lost+found  opt   run   srv       tmp  vmlinuz
boot   etc   initrd.img.old  lib64   media       proc  sbin  swapfile  usr  vmlinuz.old
cdrom  home  lib             libx32  mnt         root  snap  sys       var
manav@manav-MSI:/$
```

All your files and directories in the current directory would be listed and each type of file would be displayed with a different color. Like in the output directories are displayed with dark blue color.

$ls -l



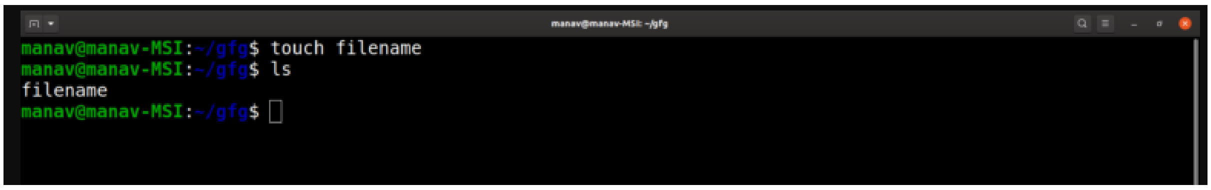
```
manav@manav-MSI:/$ ls -l
total 2097272
lrwxrwxrwx   1 root root          7 Nov 18 23:58 bin -> usr/bin
drwxr-xr-x   4 root root       4096 Jan 24 20:13 boot
drwxrwxr-x   2 root root       4096 Nov 18 23:59 cdrom
drwxr-xr-x  17 root root       4420 Jan 26  2020 dev
drwxr-xr-x 154 root root      12288 Jan 25 21:04 etc
drwxr-xr-x   3 root root       4096 Nov 18 23:59 home
lrwxrwxrwx   1 root root         32 Dec  4 01:33 initrd.img -> boot/initrd.img-5.0.0-37-generic
lrwxrwxrwx   1 root root         32 Dec  4 01:33 initrd.img.old -> boot/initrd.img-5.0.0-36-generic
lrwxrwxrwx   1 root root          7 Nov 18 23:58 lib -> usr/lib
lrwxrwxrwx   1 root root          9 Nov 18 23:58 lib32 -> usr/lib32
lrwxrwxrwx   1 root root          9 Nov 18 23:58 lib64 -> usr/lib64
lrwxrwxrwx   1 root root         10 Nov 18 23:58 libx32 -> usr/libx32
drwx------   2 root root      16384 Nov 18 23:57 lost+found
drwxr-xr-x   3 root root       4096 Nov 21 12:48 media
drwxr-xr-x   2 root root       4096 Apr 17  2019 mnt
drwxr-xr-x   3 root root       4096 Nov 21 18:12 opt
dr-xr-xr-x 341 root root          0 Jan 26  2020 proc
drwx------  10 root root       4096 Jan 22 10:51 root
drwxr-xr-x  37 root root       1080 Jan 26 13:06 run
lrwxrwxrwx   1 root root          8 Nov 18 23:58 sbin -> usr/sbin
drwxr-xr-x  15 root root       4096 Dec 16 19:46 snap
drwxr-xr-x   2 root root       4096 Apr 17  2019 srv
-rw-------   1 root root 2147483648 Nov 22 22:36 swapfile
dr-xr-xr-x  13 root root          0 Jan 26  2020 sys
drwxrwxrwt  23 root root      40960 Jan 26 14:20 tmp
drwxr-xr-x  14 root root       4096 Apr 17  2019 usr
drwxr-xr-x  15 root root       4096 Dec 26 02:59 var
lrwxrwxrwx   1 root root         29 Dec  4 01:33 vmlinuz -> boot/vmlinuz-5.0.0-37-generic
lrwxrwxrwx   1 root root         29 Dec  4 01:33 vmlinuz.old -> boot/vmlinuz-5.0.0-36-generic
manav@manav-MSI:/$ 
```

It returns the detailed listing of the files and directories in the current directory. The command gives os the owner of the file and even which file could be managed by which user or group and which user/group has the right to access or execute which file.

2. Creating Files

touch command can be used to create a new file. It will create and open a new blank file if the file with a filename does not exist. And in case the file already exists then the file will not be affected.

$touch filename

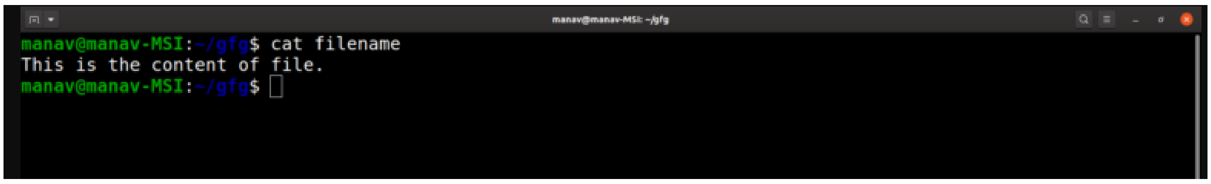


```
manav@manav-MSI:~/gfg$ touch filename
manav@manav-MSI:~/gfg$ ls
filename
manav@manav-MSI:~/gfg$ 
```

3. Displaying File Contents

cat command can be used to display the contents of a file. This command will display the contents of the 'filename' file. And if the output is very large then we could use more or less to fit the output on the terminal screen otherwise the content of the whole file is displayed at once.

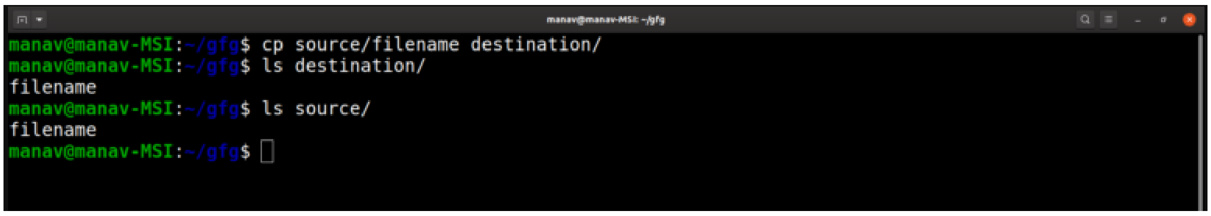
$cat filename



```
manav@manav-MSI:~/gfg$ cat filename
This is the content of file.
manav@manav-MSI:~/gfg$ 
```

4. Copying a File

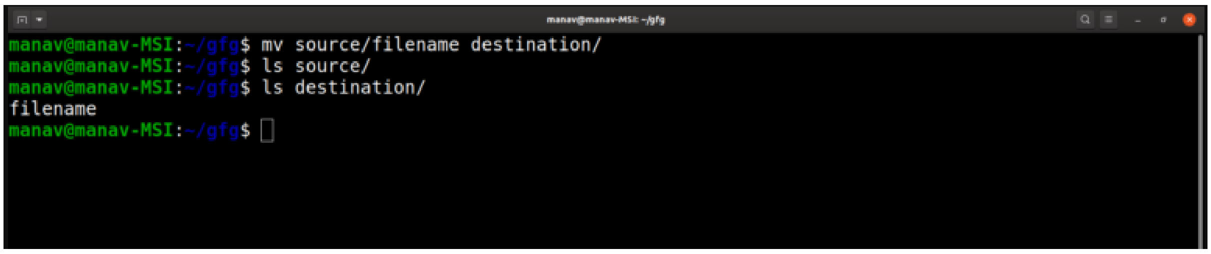
cp command could be used to create the copy of a file. It will create the new file in destination with the same name and content as that of the file ‘filename’.
$cp source/filename destination/



```
manav@manav-MSI:~/gfg$ cp source/filename destination/
manav@manav-MSI:~/gfg$ ls destination/
filename
manav@manav-MSI:~/gfg$ ls source/
filename
manav@manav-MSI:~/gfg$ 
```

5. Moving a File

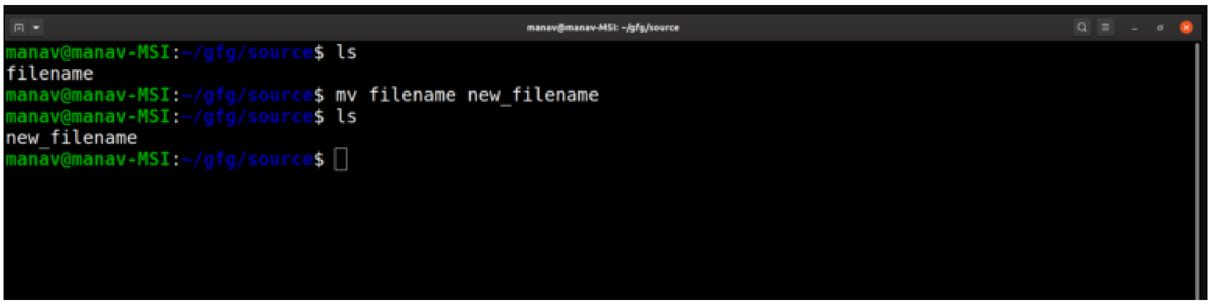
mv command could be used to move a file from source to destination. It will remove the file filename from the source folder and would be creating a file with the same name and content in the destination folder.
$mv source/filename destination/



```
manav@manav-MSI:~/gfg$ mv source/filename destination/
manav@manav-MSI:~/gfg$ ls source/
manav@manav-MSI:~/gfg$ ls destination/
filename
manav@manav-MSI:~/gfg$ 
```

6. Renaming a File

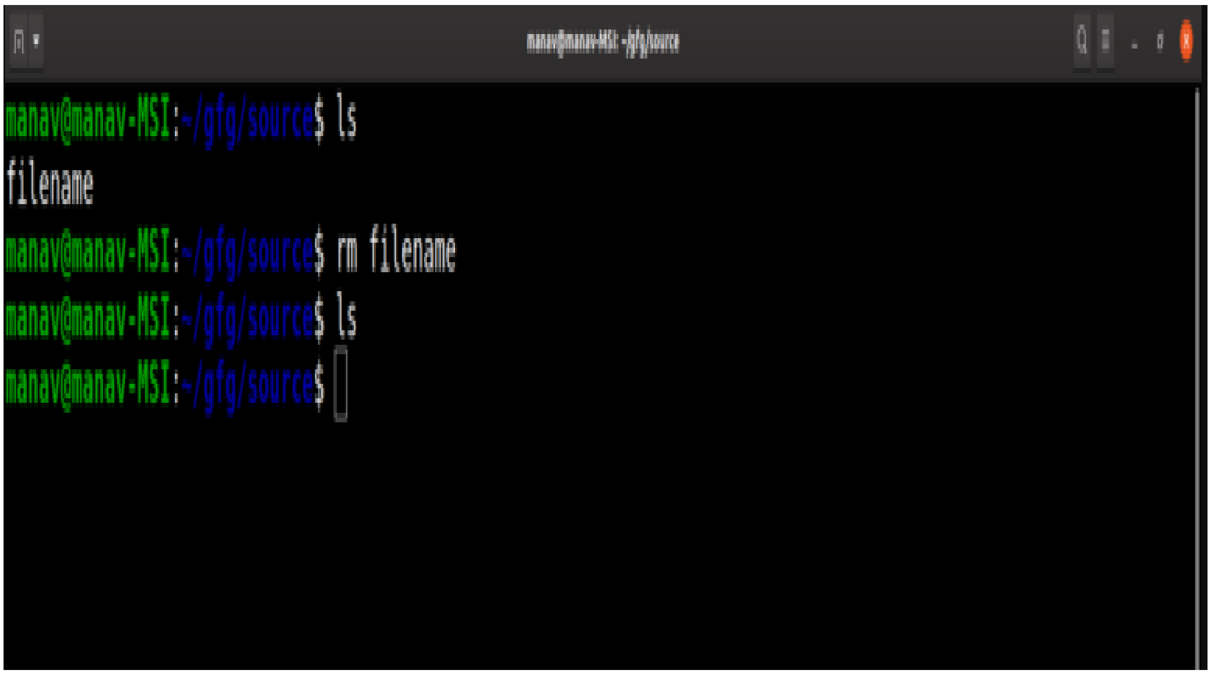
mv command could be used to rename a file. It will rename the filename to new_filename or in other words, it will remove the filename file and would be creating a new file with the new_filename with the same content and name as that of the filename file.
$mv filename new_filename



```
manav@manav-MSI:~/gfg/source$ ls
filename
manav@manav-MSI:~/gfg/source$ mv filename new_filename
manav@manav-MSI:~/gfg/source$ ls
new_filename
manav@manav-MSI:~/gfg/source$ 
```

7. Deleting a File

rm command could be used to delete a file. It will remove the filename file from the directory.
$rm filename



```
manav@manav-MSI:~/gfg/source$ ls
filename
manav@manav-MSI:~/gfg/source$ rm filename
manav@manav-MSI:~/gfg/source$ ls
manav@manav-MSI:~/gfg/source$ 
```