# Unit–IV

# LINUX

Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

Linux is a community of open-source UNIX like operating systems that are based on the Linux Kernel. It was initially released by **Linus Torvalds** on September 17, 1991.

It is a free and open-source operating system and the source code can be modified and distributed to anyone commercially or noncommercial under the GNU(General_Public_License).
Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc.

Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smart watches, etc.

The biggest success of Linux is Android (operating system) it is based on the Linux kernel that is running on smartphones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution.

## Linux Distribution

Linux distribution is an operating system that is made up of a collection of software based on Linux kernel or we can say distribution contains the Linux kernel and supporting libraries and software.

We can get Linux based operating system by downloading one of the Linux distributions and these distributions are available for different types of devices like embedded devices, personal computers, etc.

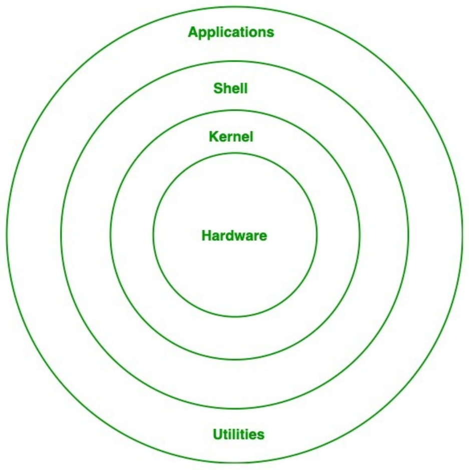
Around **600 + Linux Distributions** are available and some of the popular Linux distributions are:

- MX Linux

- Manjaro
- Linux Mint
- Elmentary
- Ubuntu
- Debian
- Solus
- Fedora
- openSUSE
- Deepin
- Red Hat

## Architecture of Linux

Linux architecture has the following components:



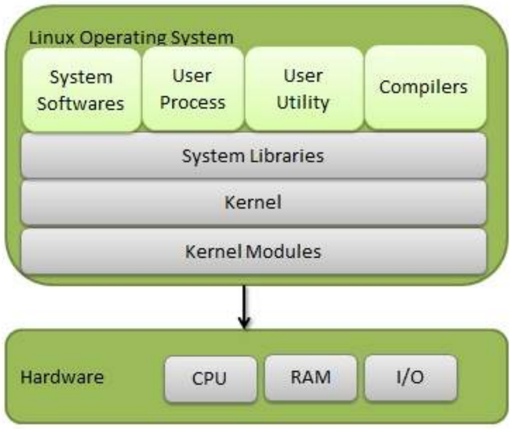
- **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
- **Utilities** − Utility programs that provide the user most of the functionalities of an operating systems.

## Components of Linux System

Linux Operating System has primarily three components

- **Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

- **System Library** − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
- **System Utility** − System Utility programs are responsible to do specialized, individual level tasks.



## Basic Features

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

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

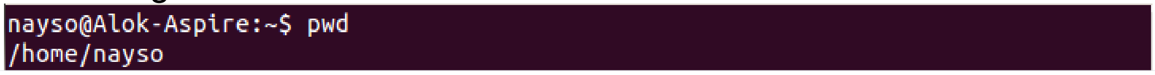
## The Terminal

The terms “terminal,” “shell,” and “command line interface” are often used interchangeably, but there are subtle differences between them:

- A *terminal* is an input and output environment that presents a text-only window running a shell.
- A *shell* is a program that exposes the computer’s operating system to a user or program. In Linux systems, the shell presented in a terminal is a command line interpreter.
- A *command line interface* is a user interface (managed by a command line interpreter program) which processes commands to a computer program and outputs the results.

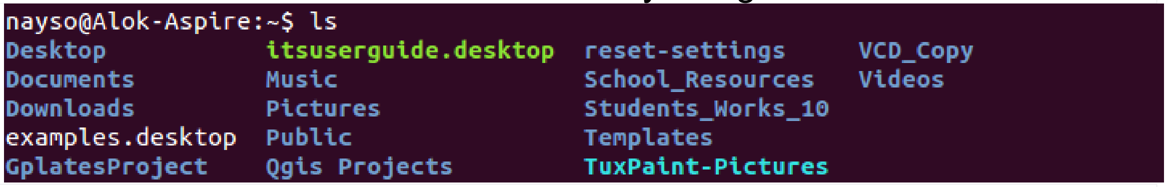
## Basic Commands

**1. pwd** ” When you first open the terminal, you are in the home directory of your user. To know which directory you are in, you can use the **pwd** command. It gives us the absolute path, which means the path that starts from the root. The root is the base of the Linux file system. It is denoted by a forward slash ( / ). The user directory is usually something like "/home/username".



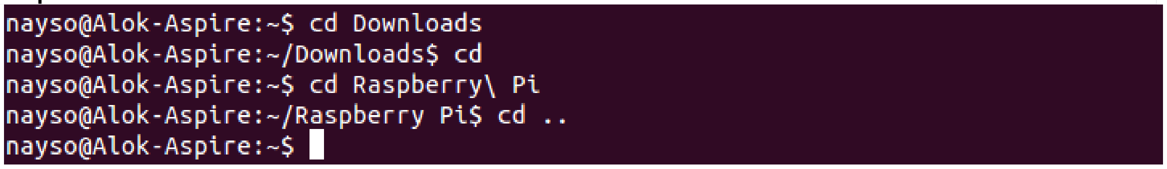
```
nayso@Alok-Aspire:~$ pwd
/home/nayso
```

**2. ls** ” Use the **"ls"** command to know what files are in the directory you are in. You can see all the hidden files by using the command **ls** .



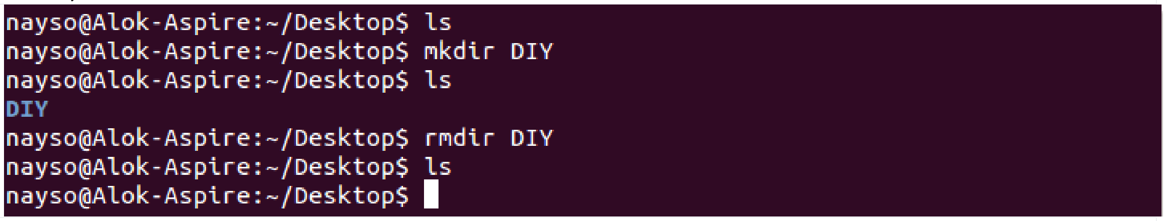
```
nayso@Alok-Aspire:~$ ls
Desktop            itsuserguide.desktop  reset-settings     VCD_Copy
Documents          Music                 School_Resources   Videos
Downloads          Pictures              Students_Works_10
examples.desktop   Public                Templates
GplatesProject     Qgis Projects         TuxPaint-Pictures
```

**3. cd**” Use the **"cd"** command to go to a directory. For example, if you are in the home folder, and you want to go to the downloads folder, then you can type in **cd Downloads** . Remember, this command is case sensitive, and you have to type in the name of the folder exactly as it is. But there is a problem with these commands. Imagine you have a folder named Raspberry Pi . In this case, when you type in **cd Raspberry Pi** , the shell will take the second argument of the command as a different one, so you will get an error saying that the directory does not exist. Here, you can use a backward slash. That is, you can use **cd Raspberry\ Pi** in this case. Spaces are denoted like this: If you just type **cd** and press enter, it takes you to the home directory. To go back from a folder to the folder before that, you can type **cd** ..” . The two dots represent back.



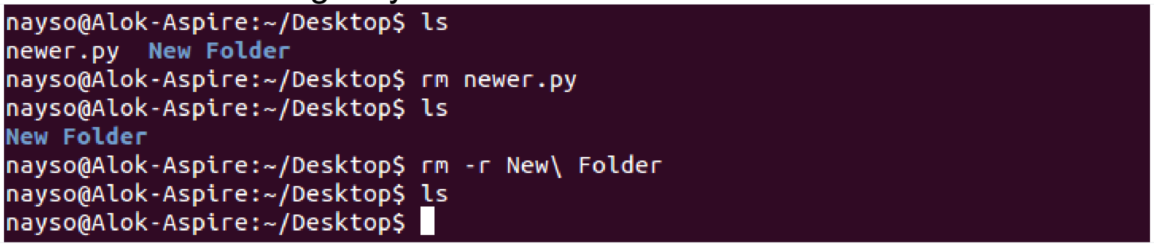
```
nayso@Alok-Aspire:~$ cd Downloads
nayso@Alok-Aspire:~/Downloads$ cd
nayso@Alok-Aspire:~$ cd Raspberry\ Pi
nayso@Alok-Aspire:~/Raspberry Pi$ cd ..
nayso@Alok-Aspire:~$
```

**4. mkdir & rmdir** ” Use the **mkdir** command when you need to create a folder or a directory. For example, if you want to make a directory called â€œDIYâ€ , then you can type **mkdir DIY** . Remember, as told before, if you want to create a directory named DIY Hacking , then you can type mkdir **DIY\ Hacking**. Use **rmdir** to delete a directory. But **rmdir** can only be used to delete an empty directory. To delete a directory containing files, use **rm**.



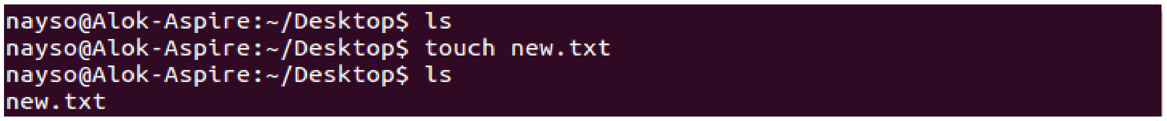
```
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$ mkdir DIY
nayso@Alok-Aspire:~/Desktop$ ls
DIY
nayso@Alok-Aspire:~/Desktop$ rmdir DIY
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$
```

**5. rm** - Use the **rm** command to delete files and directories.  Use "**rm -r**" to delete just the directory. It deletes both the folder and the files it contains when using only the **rm** command.



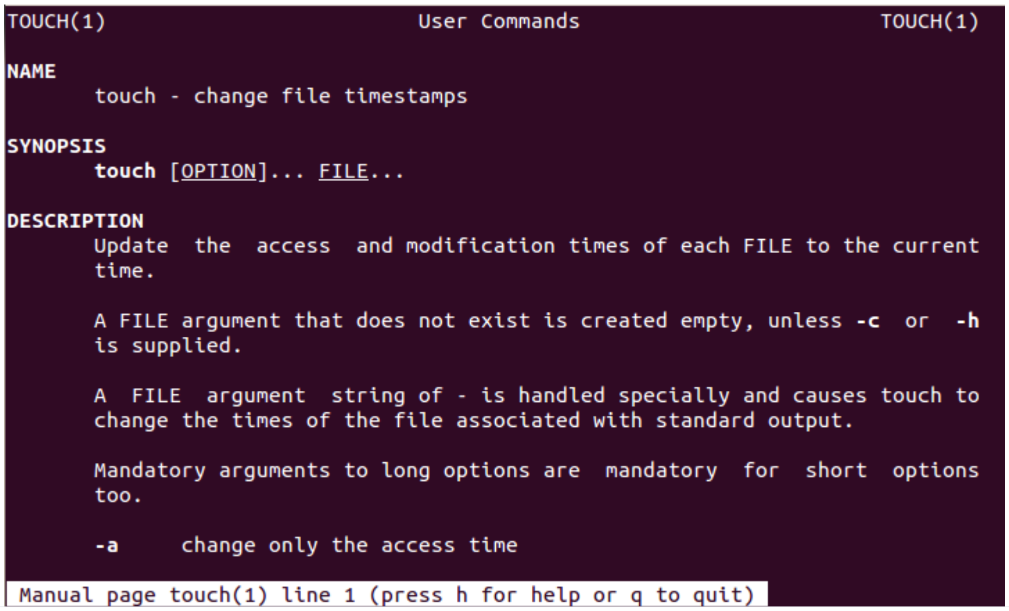
```
nayso@Alok-Aspire:~/Desktop$ ls
newer.py  New Folder
nayso@Alok-Aspire:~/Desktop$ rm newer.py
nayso@Alok-Aspire:~/Desktop$ ls
New Folder
nayso@Alok-Aspire:~/Desktop$ rm -r New\ Folder
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$
```

**6. touch**” The **touch** command is used to create a file. It can be anything, from an empty txt file to an empty zip file. For example, **touch new.txt**.



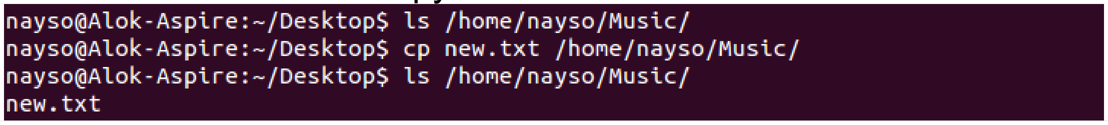
```
nayso@Alok-Aspire:~/Desktop$ ls
nayso@Alok-Aspire:~/Desktop$ touch new.txt
nayso@Alok-Aspire:~/Desktop$ ls
new.txt
```

**7. man & --help**” To know more about a command and how to use it, use the **man** command. It shows the manual pages of the command. For example, **man cd**  shows the manual pages of the **cd** command. Typing in the command name and the argument helps it show which ways the command can be used (e.g., **cd “help**).



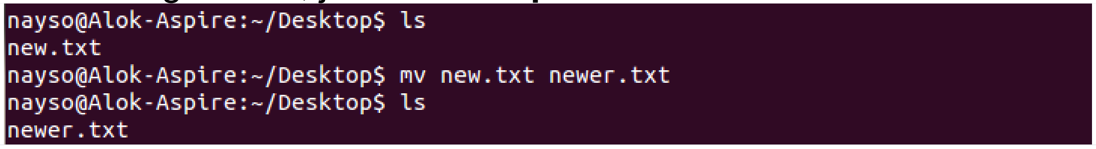
```
TOUCH(1)                         User Commands                        TOUCH(1)

NAME
       touch - change file timestamps

SYNOPSIS
       touch [OPTION]... FILE...

DESCRIPTION
       Update  the  access  and modification times of each FILE to the current
       time.

       A FILE argument that does not exist is created empty, unless -c  or  -h
       is supplied.

       A  FILE  argument  string of - is handled specially and causes touch to
       change the times of the file associated with standard output.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a     change only the access time

 Manual page touch(1) line 1 (press h for help or q to quit)
```

**8. cp**” Use the **cp** command to copy files through the command line. It takes two arguments: The first is the location of the file to be copied , the second is where to copy.



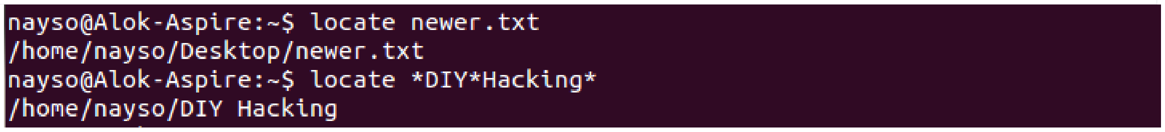
```
nayso@Alok-Aspire:~/Desktop$ ls /home/nayso/Music/
nayso@Alok-Aspire:~/Desktop$ cp new.txt /home/nayso/Music/
nayso@Alok-Aspire:~/Desktop$ ls /home/nayso/Music/
new.txt
```

**9. mv**” Use the **mv** command to move files through the command line. We can also use the **mv** command to rename a file. For example, if we want to rename the file **text** to **new** , we can use **mv text new**. It takes the two arguments, just like the **cp** command.



```
nayso@Alok-Aspire:~/Desktop$ ls
new.txt
nayso@Alok-Aspire:~/Desktop$ mv new.txt newer.txt
nayso@Alok-Aspire:~/Desktop$ ls
newer.txt
```

**10. locate**” The **locate** command is used to locate a file in a Linux system, just like the search command in Windows. This command is useful when you don't know where a file is saved or the actual name of the file. Using the -i argument with the command helps to ignore the case (it doesn't matter if it is uppercase or lowercase). So, if you want a file that has the word “hello”, it gives the list of all the files in your Linux system containing the word "hello" when you type in “**locate -i hello**” . If you remember two words, you can separate them using an asterisk (*). For example, to locate a file containing the words "hello" and "this", you can use the command “**locate -i *hello*this” .**

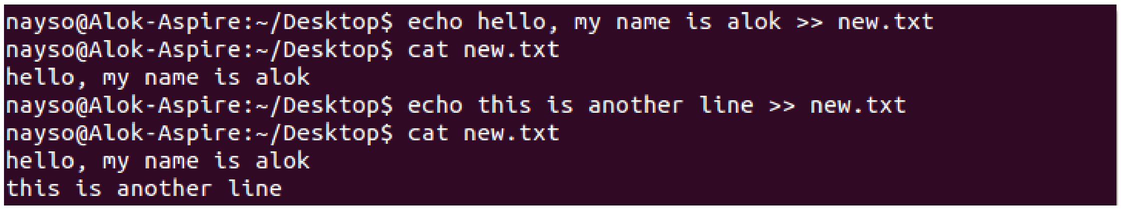


```
nayso@Alok-Aspire:~$ locate newer.txt
/home/nayso/Desktop/newer.txt
nayso@Alok-Aspire:~$ locate *DIY*Hacking*
/home/nayso/DIY Hacking
```

**Intermediate Commands**

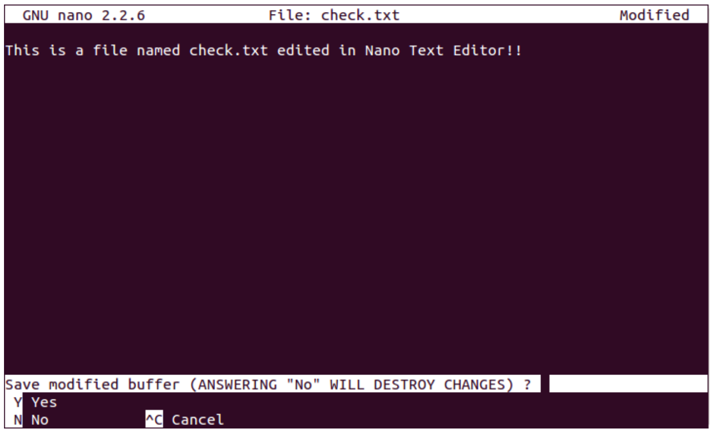
**1. echo** ” The "**echo**" command helps us move some data, usually text into a file. For example, if you want to create a new text file or add to an already made text file, you just need to type in, **echo hello, my name is alok >> new.txt** . You do not need to separate the spaces by using the backward slash here, because we put in two triangular brackets when we finish what we need to write.

**2. cat** ” Use the **cat** command to display the contents of a file. It is usually used to easily view programs.



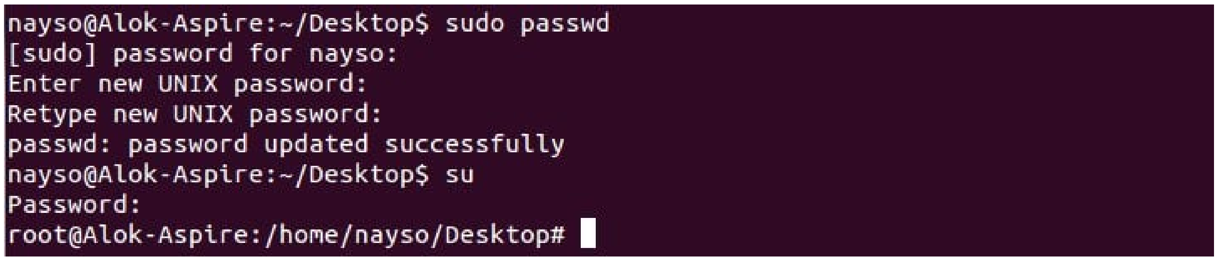
```
nayso@Alok-Aspire:~/Desktop$ echo hello, my name is alok >> new.txt
nayso@Alok-Aspire:~/Desktop$ cat new.txt
hello, my name is alok
nayso@Alok-Aspire:~/Desktop$ echo this is another line >> new.txt
nayso@Alok-Aspire:~/Desktop$ cat new.txt
hello, my name is alok
this is another line
```

**3. nano, vi, jed” nano** and **vi** are already installed text editors in the Linux command line. The **nano** command is a good text editor that denotes keywords with color and can recognize most languages. And **vi** is simpler than **nano**. You can create a new file or modify a file using this editor. For example, if you need to make a new file named **"check.txt**", you can create it by using the command **nano check.txt**. You can save your files after editing by using the sequence Ctrl+X, then Y (or N for no). In my experience, using **nano** for HTML editing doesn't seem as good, because of its color, so I recommend **jed** text editor.



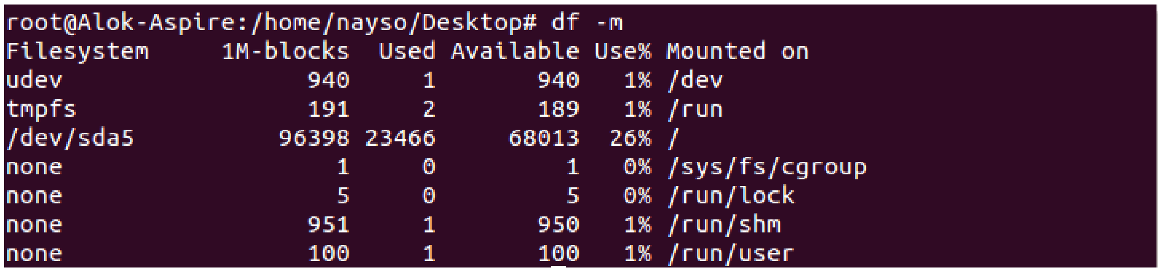
```
  GNU nano 2.2.6              File: check.txt                        Modified

This is a file named check.txt edited in Nano Text Editor!!

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No           ^C Cancel
```

**4. sudo**” A widely used command in the Linux command line, **sudo** stands for "SuperUser Do". So, if you want any command to be done with administrative or root privileges, you can use the **sudo** command. For example, if you want to edit a file like **viz. alsa-base.conf**, which needs root permissions, you can use the command " **sudo nano alsa-base.conf**. You can enter the root command line using the command **sudo bash** , then type in your user password. You can also use the command **su** to do this, but you need to set a root password before that. For that, you can use the command **sudo passwd** (not misspelled, it is **passwd**). Then type in the new root password.



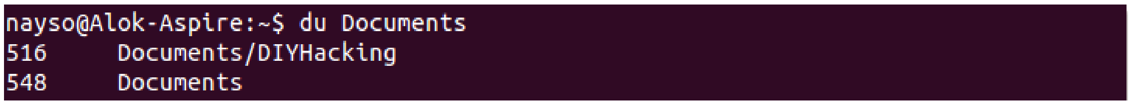
```
nayso@Alok-Aspire:~/Desktop$ sudo passwd
[sudo] password for nayso:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
nayso@Alok-Aspire:~/Desktop$ su
Password:
root@Alok-Aspire:/home/nayso/Desktop#
```

**5. df** " Use the **df** command to see the available disk space in each of the partitions in your system. You can just type in **df** in the command line and you can see each mounted partition and their used/available space in % and in KBs. If you want it shown in megabytes, you can use the command **df -m**.



```
root@Alok-Aspire:/home/nayso/Desktop# df -m
Filesystem     1M-blocks  Used Available Use% Mounted on
udev                 940     1       940   1% /dev
tmpfs                191     2       189   1% /run
/dev/sda5          96398 23466     68013  26% /
none                   1     0         1   0% /sys/fs/cgroup
none                   5     0         5   0% /run/lock
none                 951     1       950   1% /run/shm
none                 100     1       100   1% /run/user
```

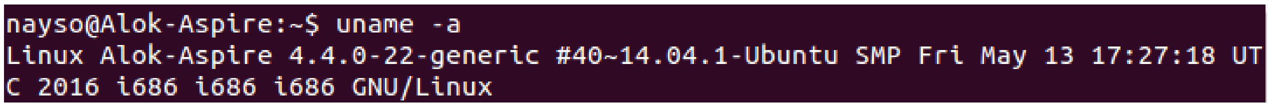
**6. du**" Use **du** to know the disk usage of a file in your system. If you want to know the disk usage for a particular folder or file in Linux, you can type in the command **df** and the name of the folder or file. For example, if you want to know the disk space used by the documents folder in Linux, you can use the command **du Documents**. You can also use the command **ls –lah** to view the file sizes of all the files in a folder.



```
nayso@Alok-Aspire:~$ du Documents
516     Documents/DIYHacking
548     Documents
```

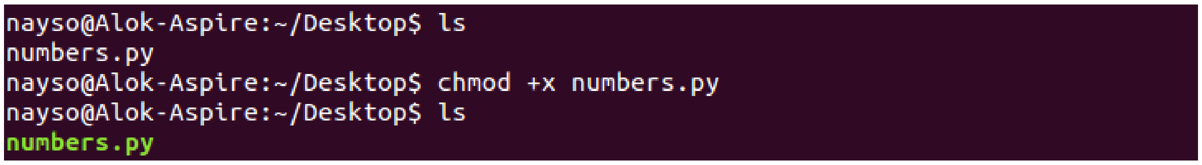
**7. zip, unzip** Use **zip** to compress files into a zip archive, and **unzip** to extract files from a zip archive.

**8. uname**” Use **uname** to show the information about the system your Linux distro is running. Using the command **uname -a** prints most of the information about the system. This prints the kernel release date, version, processor type, etc.



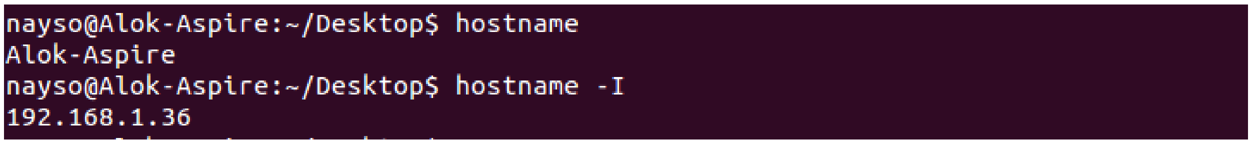
```
nayso@Alok-Aspire:~$ uname -a
Linux Alok-Aspire 4.4.0-22-generic #40~14.04.1-Ubuntu SMP Fri May 13 17:27:18 UT
C 2016 i686 i686 i686 GNU/Linux
```

**9. chmod** ” Use **chmod** to make a file executable and to change the permissions granted to it in Linux. Imagine you have a python code named **numbers.py** in your computer. You'll need to run **python numbers.py** every time you need to run it. Instead of that, when you make it executable, you'll just need to run **numbers.py** in the terminal to run the file. To make a file executable, you can use the command **chmod +x numbers.py** in this case. You can use **chmod 755 numbers.py** to give it root permissions or **sudo chmod +x numbers.py** for root executable.



```
nayso@Alok-Aspire:~/Desktop$ ls
numbers.py
nayso@Alok-Aspire:~/Desktop$ chmod +x numbers.py
nayso@Alok-Aspire:~/Desktop$ ls
numbers.py
```

**10. hostname**” Use **hostname** to know your name in your host or network. Basically, it displays your hostname and IP address. Just typing **hostname** gives the output. Typing in **hostname -I** gives you your IP address in your network.



```
nayso@Alok-Aspire:~/Desktop$ hostname
Alok-Aspire
nayso@Alok-Aspire:~/Desktop$ hostname -I
192.168.1.36
```

## Linux Text Editors

Linux text editors can be used for **editing text files, writing codes, updating user instruction files,** and more. A Linux system supports multiple text editors. There are two types of text editors in Linux, which are given below:

- **Command-line text editors** such as Vi, nano, pico, and more.
- **GUI text editors** such as gedit (for Gnome), Kwrite, and more.

A text editor plays an important role while coding. So, it is important to select the best text editor. A text editor should not only be simple but also functional and should be good to work with.

A *text editor with IDE features* is considered as a good text editor.

## 1.Vi/VIM editor

Vim editor is one of the most used and powerful command-line based editor of the Linux system. By default, it is supported by most Linux distros. It has enhanced functionalities of the old Unix Vi editor. It is a user-friendly editor and provides the same environment for all the Linux distros. It is also termed as **programmer's editor** because most programmers prefer Vi editor.

## What is the VI editor?

The VI editor is the most popular and classic text editor in the Linux family. Below, are some reasons which make it a widely used editor –

1) It is available in almost all Linux Distributions

2) It works the same across different platforms and Distributions

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

Nowadays, there are advanced versions of the vi editor available, and the most popular one is **VIM** which is **V**i **Im**proved. Some of the other ones are Elvis, Nvi, Nano, and Vile. It is wise to learn vi because it is feature-rich and offers endless possibilities to edit a file.

Vi editor has some special features such as Vi modes and syntax highlighting that makes it powerful than other text editors. Generally, it has two modes:

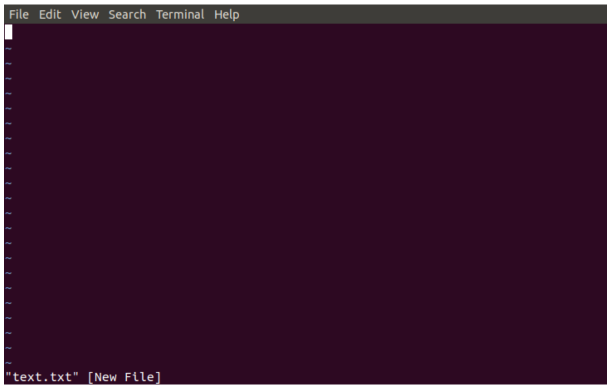
**Command Mode:** The command mode allows us to perform actions on files. By default, it starts in command mode. In this mode, all types of words are considered as commands. We can execute commands in this mode.

**Insert Mode:** The insert mode allows to insert text on files. To switch from command mode to insert mode, press the **Esc** key to exit from active mode and **'i'** key.

To invoke the vi editor, execute the vi command with the file name as follows:

1. vi <file name>

   It will look like below image:



```
"text.txt" [New File]
```

## VI Editing commands

- i – Insert at cursor (goes into insert mode)
- a – Write after cursor (goes into insert mode)
- A – Write at the end of line (goes into insert mode)
- ESC – Terminate insert mode
- u – Undo last change
- U – Undo all changes to the entire line
- o – Open a new line (goes into insert mode)
- dd – Delete line
- 3dd – Delete 3 lines.
- D – Delete contents of line after the cursor
- C – Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.
- dw – Delete word
- 4dw – Delete 4 words
- cw – Change word
- x – Delete character at the cursor
- r – Replace character
- R – Overwrite characters from cursor onward
- s – Substitute one character under cursor continue to insert
- S – Substitute entire line and begin to insert at the beginning of the line
- ~ – Change case of individual character

## Moving within a file

- k – Move cursor up
- j – Move cursor down
- h – Move cursor left
- l – Move cursor right

## Saving and Closing the file

- Shift+zz – Save the file and quit
- :w – Save the file but keep it open
- :q! – Quit vi and do not save changes
- :wq – Save the file and quit

### 2. Nano editor

Nano is a straight forward editor. It is designed for both beginners and advanced users. It has many customization features.

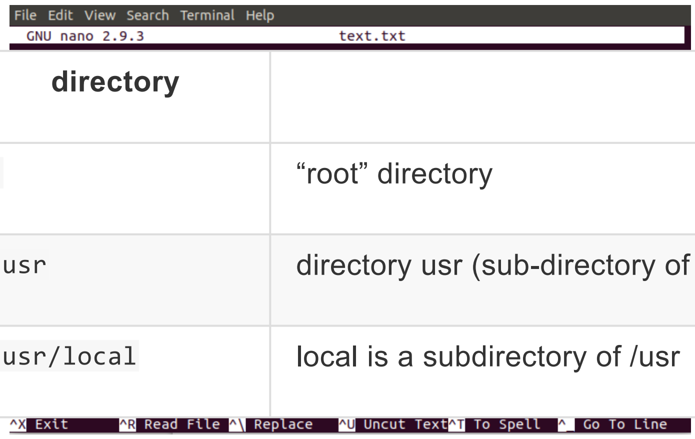
Some advanced features of a nano text editor are as following:

- It has highly customizable key bindings
- It supports syntax highlighting
- It has undo and redo options
- It provides full line display on the standard output
- It has pager support to read from standard input

To open file with nano editor, execute the command as follows:

```
1. nano <file name>
```

The nano editor looks like:



```
  GNU nano 2.9.3                    text.txt
```

| directory | explanation |
|---|---|
| `/` | “root” directory |
| `/usr` | directory usr (sub-directory of / “root” directory) |
| `/usr/local` | local is a subdirectory of /usr |

```
^X Exit      ^R Read File ^\ Replace    ^U Uncut Text^T To Spell   ^_ Go To Line
```

In the nano editor, the useful options are given at the bottom, use the **CTRL+ option** to perform an operation. For example, to exit from the editor, use **CTRL +X** keys.

### File management and directories

## Directories

File and directory paths in UNIX use the forward slash “/” to separate directory names in a path. If we are using a server our shell will start from /home/yourUserName/ directory.

Here are a few examples of the directory strcucture:

### *Creating a directory*

**mkdir** command creates a new directory. The command below creates a new directory named “newDir” under the current directory.

```
$ mkdir newDir
```

This command creates a new directory in user’s home directory.

```
$ mkdir ~/newDir
```

The next command creates a the target directory and all the non-existing directories in the path. The command will create samtools directory, and will create “opt” directory if it does not exist. All of this will be done in user’s home directory as indicated by “~/” in that path.

```
$ mkdir -p ~/opt/samtools
```

## Moving around the file system with cd

***cd*** command stands for “change directory”. Here are a few examples of the ***cd*** command and ***pwd***.

Type variants of these to your shell to move around your file system.

| **Command + arguments** | **explanation** |
|---|---|
| `pwd` | Show the “present working directory”, or current directory. |
| `cd` | Change current directory to your HOME directory. |
| `cd /usr/local` | Change current directory to /usr/local |
| `cd INIT` | change current directory to INIT which is a sub-directory of the current |
| `cd ..` | Change current directory to the parent directory of the current |
| `cd ~akalin` | Change the current directory to the user akalin’s home directory (if you have permission). |

## Listing directory contents

***ls*** command lists the contets of a directory. It can take multiple options, some of those are explained below.

| **commands** | **explanation** |
|---|---|
| `ls` | list a directory |
| `ls -l` | list a directory in detailed format including **file sizes** and **permissions** |
| `ls -a` | List the current directory including hidden files. Hidden files start with “.” |
| `ls -ld *` | List all the file and directory names in the current directory using long format. Without the “d” option, ls would list the contents of any sub-directory of the current. With the “d” option, ls just lists them like |

| | |
|---|---|
| | regular files. |
| `ls -lh` | list detailed format this time file sizes are human readable not in bytes |

## Moving, renaming and copying files

***cp*** command **copies** the files and ***mv*** command **moves** the files. They are generally used with two main arguments. `cp target_file destination_file` or `mv target_file destination_file`.

| commands | explanation |
|---|---|
| `cp file1 file2` | copy file1 as file2 |
| `cp /data/seq_data/file1 ~/` | copy file1 at /data/seq_data to your home directory. |
| `mv file1 newname` | move or rename a file |
| `mv file1 ~/opt/` | move file1 into sub-directory opt in your home directory. |

## Finding files

There are a couple of ways we can find files in our file system. The find command works in the following syntax `find directory -name targetfile`. It is useful when we have a rough idea about file location.
The following finds all files ending in ".html" under /home/user directory.

```
$ find /home/user -name "*.html"
```

find can also do more than just finding files. It also execute commands on the files we find via -exec option. The following command finds all files in the current directory with ".txt" ending and counts the number of lines in every text file. The '{}' is replaced by the name of each file found and the ';' ends the -exec clause.

```
$ find . -name "*.txt" -exec wc -l '{}' ';'
```

Another command that can find files is **locate**. The locate command provides a faster way of locating all files whose names match a particular search string. For example:

```
 $ locate ".txt"
```

will find all filenames in the filesystem that contain ".txt" anywhere in their full paths.

A disadvantage of locate is that it stores all filenames on the system in an index that is usually updated only once a day. This means locate will not find files that have been created very recently.

## Searching the contents of a text file

Often times we would need search a file for existence of certain characters or words. If we need to find gene ids in a text file containing some scores and gene ids, we would like to get the line(s) that contains our gene id of interest. This is similar to “find” functions in modern text processors such as MS Word. This can be achieved via **grep commmand**. Syntax of the command is: `grep options pattern files`

| Command | Explanation |
|---|---|
| `grep id1 genes.txt` | searches and prints lines matching “id1” in “genes.txt” |
| `grep id1 *.txt` | searches and prints lines matching “id1” in files ending with “.txt” |
| `grep -vi id1 *.txt` | similar to above, but -i option ignores the case (Id1,ID1,iD1 and id1 treated equally), -v option prints lines that don’t match the pattern |

### *using grep and find together*

You can search all files in an entire directory tree for a particular pattern by combining **grep** and **find**. The following command prints lines containing “genes” string, from the files ’find . -name "*.txt" -print’ found.

```
$ grep genes `find . -name "*.txt" -print`
```

The search patterns that grep uses are a special named **regular expressions**. We can have more comlicated searches using regular expressions, but that is more of an advanced application.

### File Management

In Linux, most of the operations are performed on files. And to handle these files Linux has directories also known as folders which are maintained in a tree-like structure. Though, these directories are also a type of file themselves. Linux has 3 types of files:

1. **Regular Files:** It is the common file type in Linux. it includes files like – text files, images, binary files, etc. Such files can be created using the touch command. They consist of the majority of files in the Linux/UNIX system. The regular file contains ASCII or Human Readable text, executable program binaries, program data and much more.

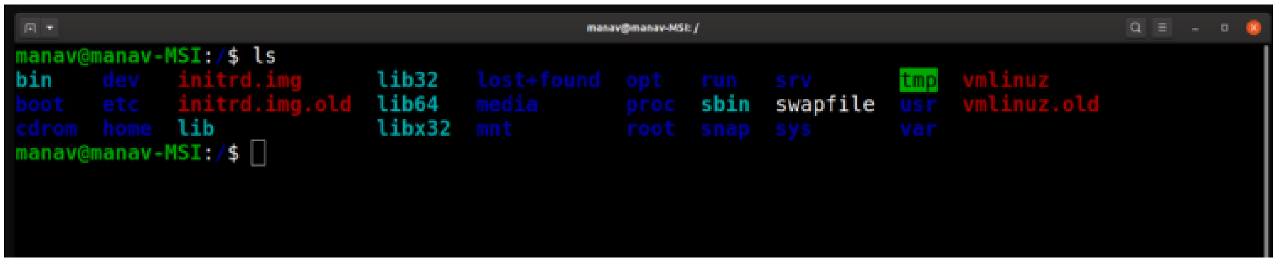
2. **Directories:** Windows call these directories as folders. These are the files that store the list of file names and the related information. The root directory(/) is the base of the system, /home/ is the default location for user's home directories, /bin for Essential User Binaries, /boot – Static Boot Files, etc. We could create new directories with mkdir command.
3. **Special Files:** Represents a real physical device such as a printer which is used for IO operations. Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. You can see them in a file system like an ordinary directory or file.

In Unix systems, there are two types of special files for each device, i.e. character special files and block special files. For more details, read the article **Unix file system**.

## 1. Files Listing

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

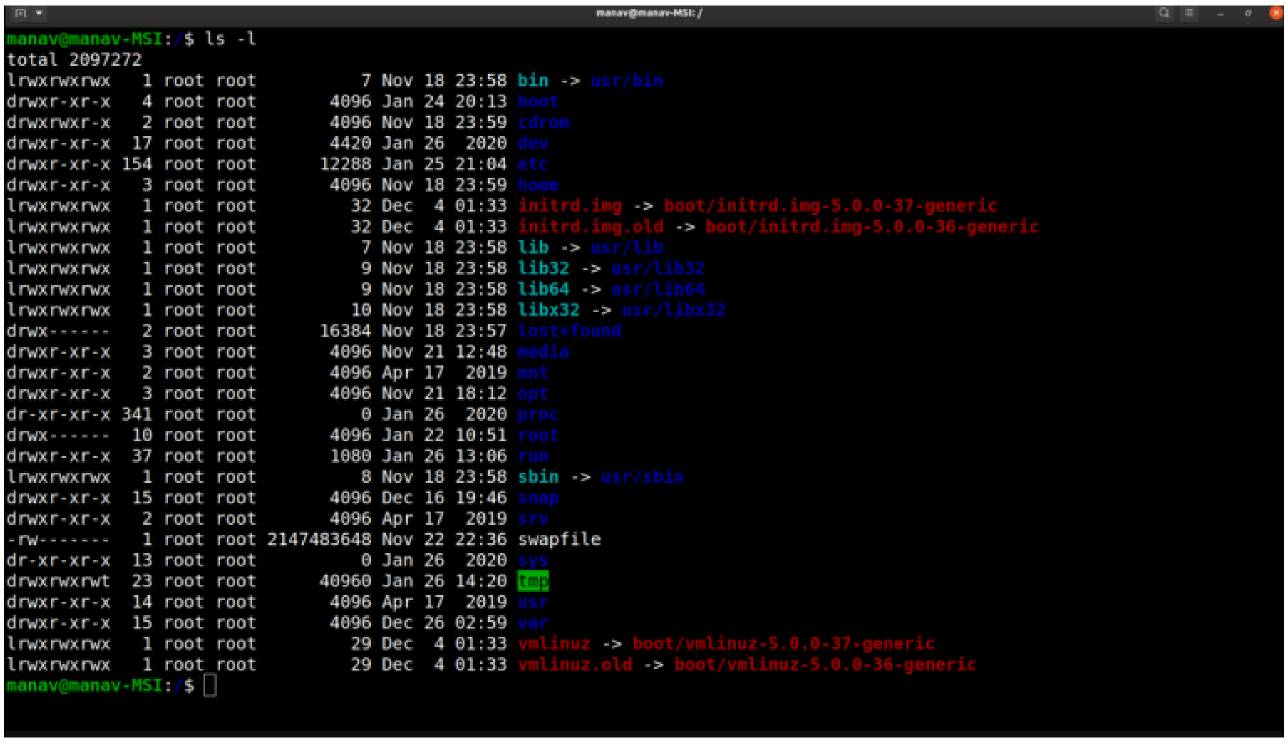
```
$ls
```



```
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
```



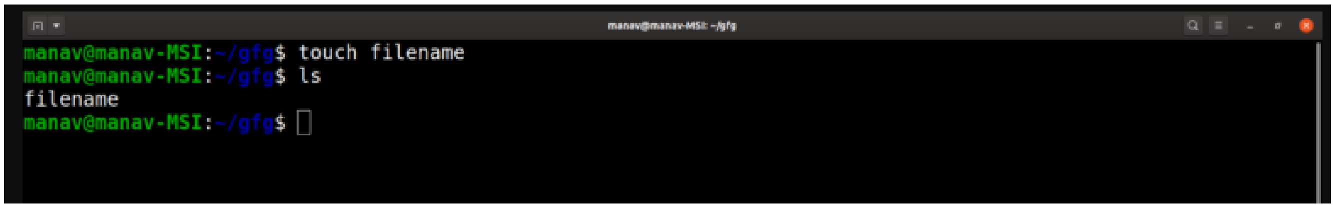
```
manav@manav-MSI:/$ ls -l
total 2097272
lrwxrwxrwx   1 root root          7 Nov 18 23:58 bin -> usr/bin
drwxr-xr-x   4 root root       4096 Jan 24 20:13 boot
drwxrwxr-x   2 root root       4096 Nov 18 23:59 cdrom
drwxr-xr-x  17 root root       4420 Jan 26  2020 dev
drwxr-xr-x 154 root root      12288 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
```

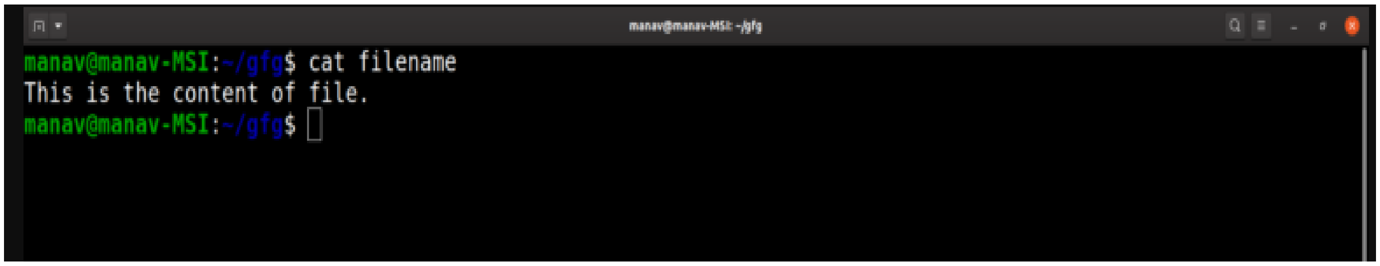


```
manav@manav-MSI:~/gfg$ touch filename
manav@manav-MSI:~/gfg$ ls
filename
manav@manav-MSI:~/gfg$ 
```

## 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
```

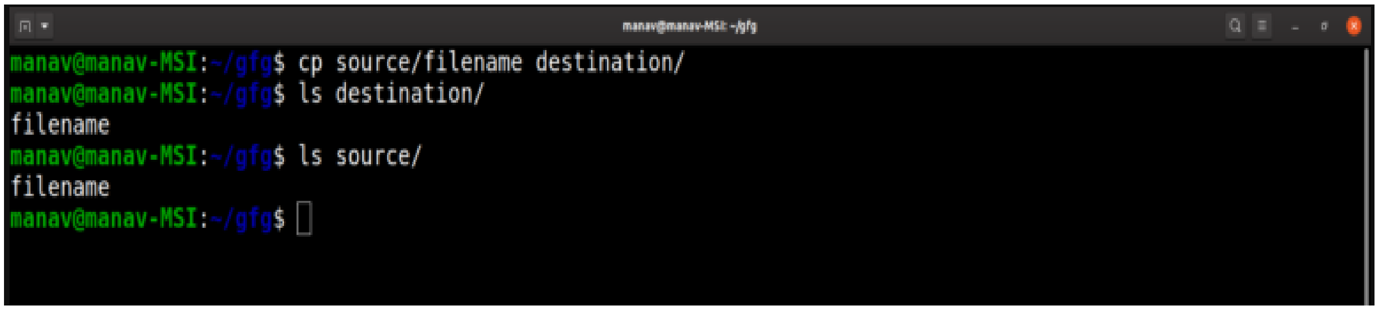


```
manav@manav-MSI:~/gfg$ cat filename
This is the content of file.
manav@manav-MSI:~/gfg$ 
```

## 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/
```

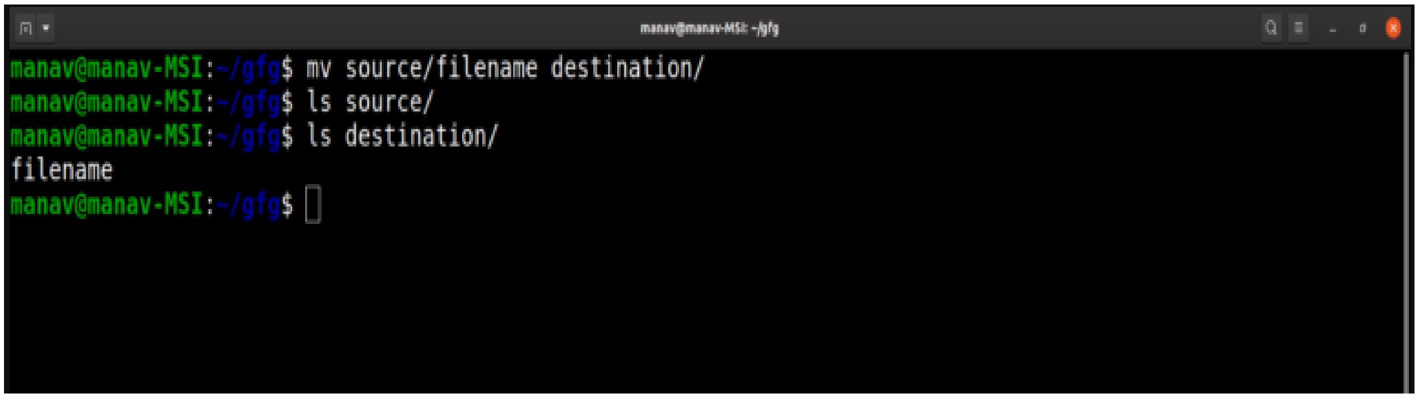


```
manav@manav-MSI:~/gfg$ cp source/filename destination/
manav@manav-MSI:~/gfg$ ls destination/
filename
manav@manav-MSI:~/gfg$ ls source/
filename
manav@manav-MSI:~/gfg$ 
```

## 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.

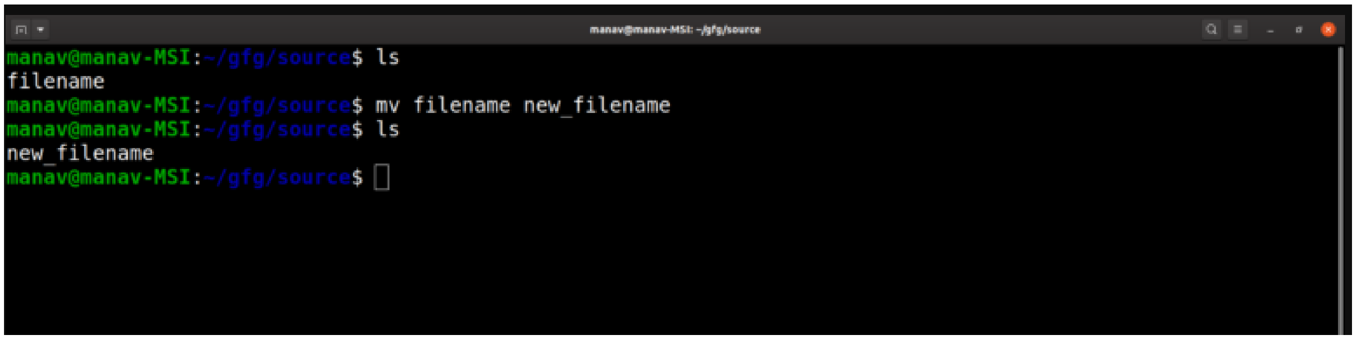
```
$mv source/filename destination/
```



```
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

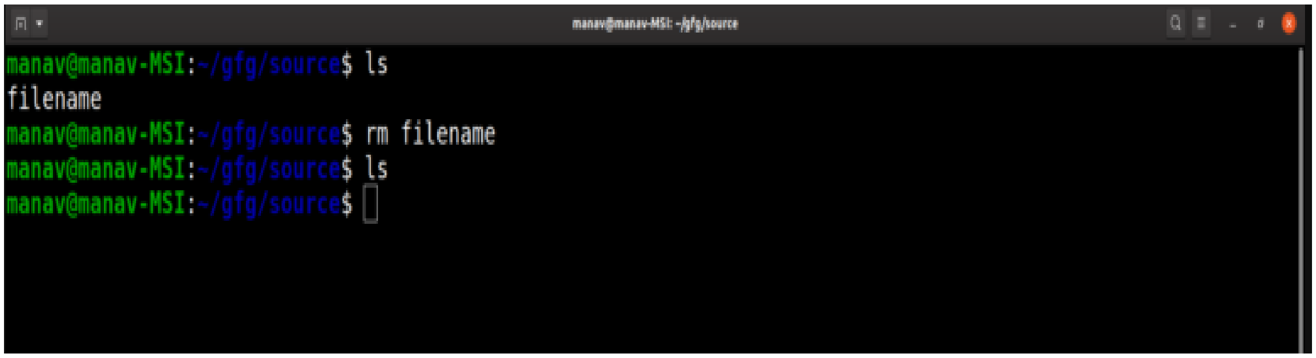
mv command could be used to rename a file. It will rename the filename to new_filename or in other words, it will remove the filename file and would be creating a new file with the new_filename with the same content and name as that of the filename file.
`$mv filename new_filename`



```
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$ 
```