Linux

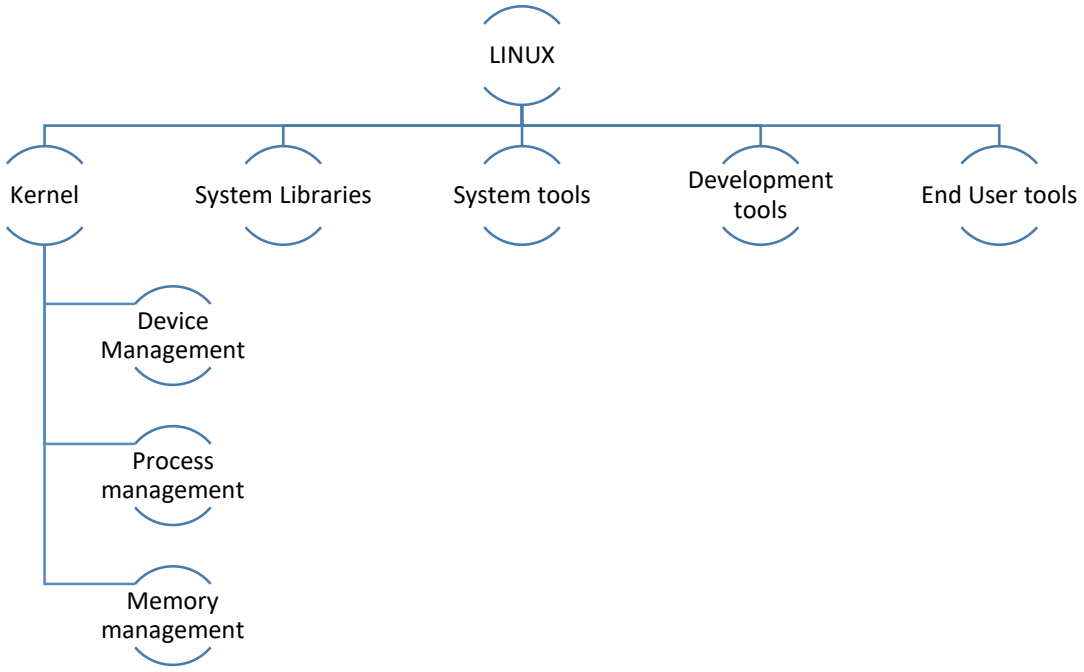
In the simple language Linux is an operating system (OS). We all are familiar with other operating systems like Microsoft windows, Apple Mac OS, iOS, Google android, etc, just like them linux is also an operating system.

An operating system is a software that enables communication between computer hardware and software. It conveys input to get processed by the processor and brings output to the hardware to display it. This is the basic function of an operating system. Although, it performs many other important tasks.

Linux is around us since mid of 90's. It can be used from wristwatches to supercomputers. It is everywhere in our phones, laptops, PCs, cars and even in refrigerators. It is very much famous among the developers and normal computer users.

Structure of Linux Operating System:

Linux OS has following components:



1) Kernel

kernel is the core of the operating system. It establishes communication between devices and software. Moreover, it manages the system resources. Basically it has four responsibilities:

Device management: A system has many devices connected to it like CPU, memory device, sound cards, graphic cards, etc. A kernel stores all the data related to all the devices in device driver (without this kernel won't be able to control the devices). Thus kernel knows what a device can do and how to manipulate it to bring out the best performance. It also manages communication between all the devices. Kernel has certain rules that has to be followed by all the devices.

Memory management: Another function that kernel has to manage is the memory management. Kernel keeps a track of used and unused memory and make sure that processes shouldn't manipulate data of each other using virtual memory address.

Process management: In process management kernel assign enough time and gives priorities to processes before handling CPU to other process. It also deals with security and ownership information.

Handling system calls: Handling system calls means a programmer can write a query or ask the kernel to perform a task.

2) System Libraries

System libraries are special programs that helps in accessing the kernel's features. A kernel has to be triggered to perform a task and this triggering is done by the applications. But applications must know how to place a system call because each kernel has a different set of system calls. Programmers have developed standard library of procedures to communicate with kernel. Each operating system supports these standards and then these are transferred to system calls for that operating system.

Example the system library for Linux is glibc (GNU C library).

3) System Tools

Linux OS has a set of utility tools which are usually simple commands. It is a software which GNU project has written and publish under their open source license so that software is freely available to everyone.

With the help of commands we can access wer files, edit and manipulate data in wer directories or files, change location of files or anything.

4) Development Tools

With the above three components wer OS is running and working. But to update wer system we have additional tools and libraries. These additional tools and libraries are written by the programmers and are called tool chain. A tool chain is a vital development tool used by the developers to produce a working application.

5) End User Tools

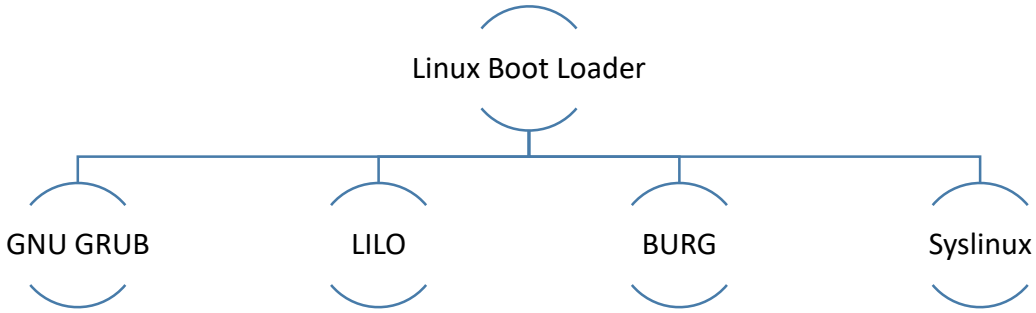
These end tools make a system unique for a user. End tools are not required for the operating system but are necessary for a user.

Some examples of end tools are graphic design tools, office suites, browsers, multimedia players, etc.

Boot Loader

A boot loader is a small program stored in the MBR or GUID partition table that helps to load an operating system into memory. Without a boot loader, wer operating system cannot be loaded into memory.

There are several boot loaders we can install together with Linux on our systems and in this tutorial, we shall briefly talk about a handful of the best Linux boot loaders to work with.



GNU GRUB:

GNU GRUB is a popular and probably the most used multi-boot Linux boot loader available, based on the original GRUB (GRand Unified Bootloader) which was created by Eirch Stefan Broleyn. It comes with several improvements, new features and bug fixes as enhancements of the original GRUB program.

GRUB has the following prominent features:

1. Supports multi-boot

2. Supports multiple hardware architectures and operating systems such as Linux and Windows
3. Offers a Bash-like interactive command line interface for users to run GRUB commands as well interact with configuration files
4. Enables access to GRUB editor
5. Supports setting of passwords with encryption for security
6. Supports booting from a network combined with several other minor features

LILO:

LILO is a simple yet powerful and stable Linux boot loader. With the growing popularity and use of GRUB, which has come with numerous improvements and powerful features, LILO has become less popular among Linux users.

It has a number of features as listed below:

1. Does not offer an interactive command line interface
2. Supports several error codes
3. Offers no support for booting from a network
4. All its files are stored in the first 1024 cylinders of a drive
5. Faces limitation with BTFS, GPT and RAID plus many more.

BURG:

Based on GRUB, BURG is a relatively new Linux boot loader. Because it is derived from GRUB, it ships in with some of the primary GRUB features, nonetheless, it also offers remarkable features such as a new object format to support multiple platforms including Linux, Windows, Mac-OS and beyond.

Additionally, it supports a highly configurable text and graphical mode boot menu, stream plus planned future improvements for it to work with various input/output devices.

SYSLINUX:

Syslinux is an assortment of light weight boot loaders that enable booting from CD-ROMs, from a network and so on. It supports filesystems such as FAT for MS-DOS, and ext2, ext3, ext4 for Linux. It as well supports uncompressed single-device Btrfs.

Note that Syslinux only accesses files in its own partition, therefore, it does not offer multi-filesystem boot capabilities.

Basic Commands

Linux Directory Commands

1. pwd :

   Linux pwd (print working directory) command displays wer location currently we are working on. It will give the whole path starting from the root ending to the directory.

   Syntax:

   ```
   pwd
   ```

   ```
   root@linux:/home/ubuntu# pwd
   /home/ubuntu
   root@linux:/home/ubuntu#
   ```

2. ls :

   The ls is the list command in Linux. It will show the full list or content of wer directory. Just type ls and press enter key. The whole content will be shown.

   Syntax:

   ```
   ls
   ```

   ```
   root@linux:/home/ubuntu# ls
   abc.txt  abcd  sample.txt
   root@linux:/home/ubuntu#
   ```

3. cd :

   The "cd" stands for 'change directory' and this command is used to change the current directory i.e; the directory in which the user is currently working.

   Syntax:

   ```
   cd <dir name>
   ```

   Here, we are in /home/ubuntu/ dir. Now, we will change our pwd using ‘cd’ command to dir. ‘abcd’ and we will check our present working dir using pwd.

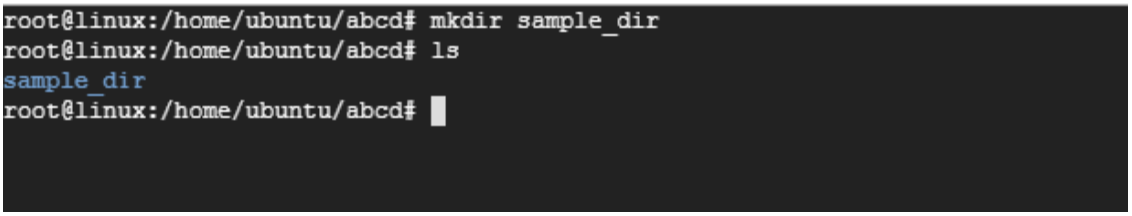
   ```
   root@linux:/home/ubuntu# cd abcd/
   root@linux:/home/ubuntu/abcd# pwd
   /home/ubuntu/abcd
   root@linux:/home/ubuntu/abcd#
   ```

4. mkdir :

   The mkdir stands for 'make directory'. With the help of mkdir command, we can create a new directory wherever we want in wer system. Just type "mkdir <dir name> , in place of <dir name> type the name of new directory, we want to create and then press enter.

   Syntax:

   ```
   mkdir <dirname>
   ```

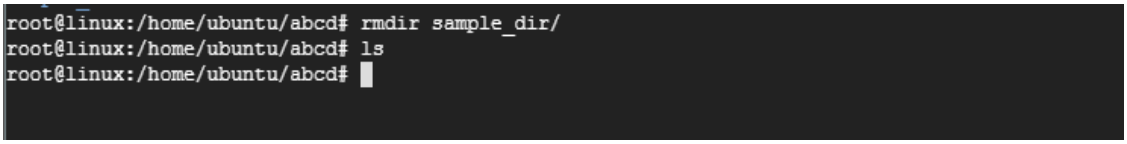


5. rmdir :

   This command is used to delete a directory. But will not be able to delete a directory including a sub-directory. It means, a directory has to be empty to be deleted.

   Syntax:

   ```
   rmdir <dirname>
   ```



Linux File Commands:

1. touch :

   Touch command is a way to create empty files (there are some other mehtods also). We can update the modification and access time of each file with the help of touch command.

   Syntax:

   ```
   touch <filename>
   ```

```
root@linux:/home/ubuntu/abcd# ls
root@linux:/home/ubuntu/abcd# touch abclearning.txt
root@linux:/home/ubuntu/abcd# ls
abclearning.txt
root@linux:/home/ubuntu/abcd#
```

2. rm :

The 'rm' means remove. This command is used to remove a file. The command line doesn't have a recycle bin or trash unlike other GUI's to recover the files. Hence, be very much careful while using this command. Once we have deleted a file, it is removed permanently.

Syntax:

```
rm <filename>
```

```
root@linux:/home/ubuntu/abcd# rm abclearning.txt
root@linux:/home/ubuntu/abcd# ls
root@linux:/home/ubuntu/abcd#
```

3. cp :

'cp' means copy. 'cp' command is used to copy a file or a directory.

Syntax:

```
cp <existing filename> <new filename>
```

Here, we will copy sample_linux.txt file in dir '/home/ubuntu/ABCD' from dir '/home/Ubuntu/abcd'

```
root@linux:/home/ubuntu/abcd# ls
root@linux:/home/ubuntu/abcd# touch sample_linux.txt
root@linux:/home/ubuntu/abcd# ls
sample_linux.txt
root@linux:/home/ubuntu/abcd# cp sample_linux.txt /home/ubuntu/ABCD/
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD/
sample_linux.txt
root@linux:/home/ubuntu/abcd#
```

4. mv :

Linux mv command is used to move existing file or directory from one location to another. It is also used to rename a file or directory. If we want to rename a single directory or file then 'mv' option will be better to use.

Syntax:

```
mv <>
```

Here, we will move a file sample_linux.txt in dir 'home/Ubuntu/ABCD1'.

```
root@linux:/home/ubuntu/abcd# ls
sample_linux.txt
root@linux:/home/ubuntu/abcd# mkdir /home/ubuntu/ABCD1
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD1
root@linux:/home/ubuntu/abcd# mv sample_linux.txt /home/ubuntu/ABCD1/
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD1
sample_linux.txt
root@linux:/home/ubuntu/abcd#
```

A) mv –i

If we want to copy a file using 'mv' option and if that file already exists then it will silently over write the existing file. But if we'll use 'i' option then it will first ask for wer permission to over write it.

Here, we will add some content in file 'sample_linux.txt' and move it in same dir where 'sample_linux.txt' is already present. Using command 'mv –i ' will overwrite older file.

Syntax:

mv –i <source_path> <target_path>

```
root@linux:/home/ubuntu/abcd# ls
sample_linux.txt
root@linux:/home/ubuntu/abcd# nano sample_linux.txt
root@linux:/home/ubuntu/abcd# cat sample_linux.txt
hi linux
root@linux:/home/ubuntu/abcd# mv -i sample_linux.txt  /home/ubuntu/ABCD1/
mv: overwrite '/home/ubuntu/ABCD1/sample_linux.txt'? y
root@linux:/home/ubuntu/abcd# cat  /home/ubuntu/ABCD1/sample_linux.txt
hi linux
root@linux:/home/ubuntu/abcd#
```

B) mv *

This command is used to move all the files from the current directory to another specified directory at once.

Syntax:

mv * <target_path>

```
root@linux:/home/ubuntu/abcd# ls
a.txt  a1.txt  a2.txt
root@linux:/home/ubuntu/abcd# mv * /home/ubuntu/ABCD1/
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD1/
a.txt  a1.txt  a2.txt  sample_linux.txt
root@linux:/home/ubuntu/abcd#
```

C) mv - - suffix

This option helps we to have a backup of the destination file before over writing it with the moving file. The original destination file will be replaced by the extension provided with the command.

Syntax:

mv - -suffix file1 file2

```
total 0
-rw-r--r-- 1 root root 0 Sep 13 08:55 file1.txt
-rw-r--r-- 1 root root 0 Sep 13 08:55 file2.txt
root@linux:/home/ubuntu/abcd# mv --suffix=.txt file1.txt file2.txt
root@linux:/home/ubuntu/abcd# ls
file2.txt  file2.txt.txt
root@linux:/home/ubuntu/abcd# ls -l
total 0
-rw-r--r-- 1 root root 0 Sep 13 08:55 file2.txt
-rw-r--r-- 1 root root 0 Sep 13 08:55 file2.txt.txt
root@linux:/home/ubuntu/abcd#
```

In above example, we have moved 'file1.txt' into 'file2.txt'. After passing the command 'mv –suffix=.txt file1.txt file2.txt', 'file1.txt' will be replaced by 'file2.txt'. While, already existing file 'file2.txt' will be replaced with the extension 'file2.txt.txt'.

D) mv –u

When we use 'u' option, files that already exist in the destination directory will be automatically skipped and all the other files will be copied.

```
root@linux:/home/ubuntu/abcd# ls
sample_linux.txt  sample_linux1.txt  sample_linux2.txt
root@linux:/home/ubuntu/abcd# cat sample_linux.txt
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD1
a.txt  a1.txt  a2.txt  sample_linux.txt
root@linux:/home/ubuntu/abcd# mv -u * /home/ubuntu/ABCD1/
root@linux:/home/ubuntu/abcd# ls /home/ubuntu/ABCD1
a.txt  a1.txt  a2.txt  sample_linux.txt  sample_linux1.txt  sample_linux2.txt
```

Linux Filters

Linux Filter commands accept input data from stdin (standard input) and produce output on stdout (standard output). It transforms plain-text data into a meaningful way and can be used with pipes to perform higher operations.

These filters are very small programs that are designed for a specific function which can be used as building blocks.

Linux filter commands:

1. cat

   When cat command is used inside pipes, it does nothing except moving stdin to stout.

```
anay@linux:~$ ls
sample.txt
anay@linux:~$ cat sample.txt
sample file for testing linux commands
anay@linux:~$
```

2. Cut

   The 'cut' command is useful in selecting a specific column of a file. After (-d), delimiter (from where we want to separate the columns) comes. Delimiters can be a space (' '), a hyphen (-), a slash (/) or anything else. After (-f), column number is mentioned.

   Syntax:

   $ cut -d(delimiter) -f(columnNumber) <fileName>

```
anay@linux:~$ cat marks.txt
anay-66
suman-44
abhi-2323
sai-323
anay@linux:~$ cut -d- -f2 marks.txt
66
44
2323
323
anay@linux:~$
```

3. Grep

   The 'grep' command stands for "global regular expression print". grep command filters the content of a file which makes our search easy.

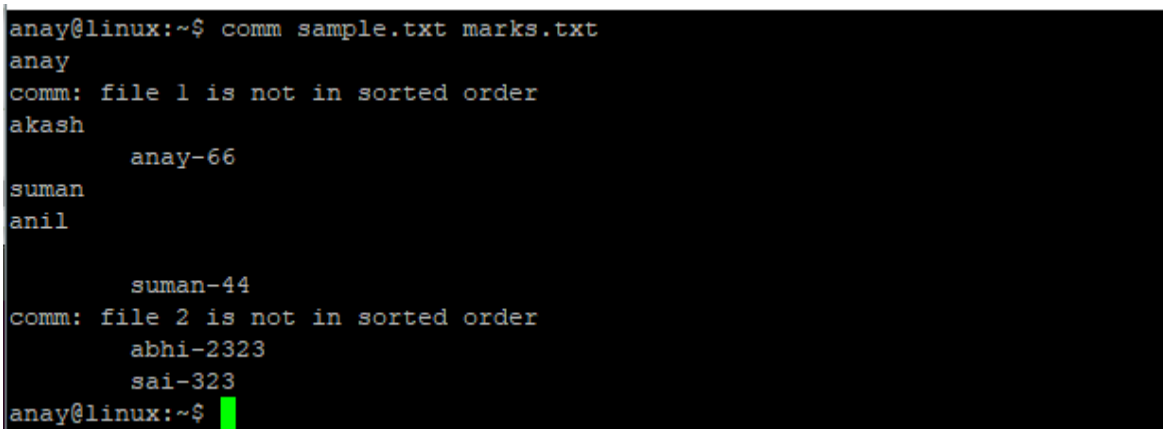
   grep with pipe

   The 'grep' command is generally used with pipe (|).

```
anay@linux:~$ cat marks.txt | grep an
anay-66
suman-44
anay@linux:~$
```

4. Comm

The 'comm' command compares two files or streams. By default, 'comm' will always display three columns. First column indicates non-matching items of first file, second column indicates non-matching items of second file, and third column indicates matching items of both the files. Both the files has to be in sorted order for 'comm' command to be executed.

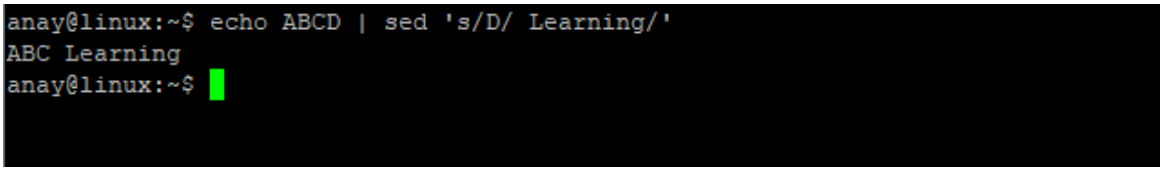


5. sed

Command 'sed' stands for stream editor. We can use this command to edit streams (files) using regular expressions. But this editing is not permanent. It remains only in display, but in actual, file content remains same.

Syntax:

```
$ command | sed 's/<oldWord>/<newWord>/'
```



6. tee

The 'tee' command is similar to 'cat' command with only one difference. It puts stdin on stdout and also put them into a file.

Syntax:

```
$ command | sed 's/<oldWord>/<newWord>/'
```

```
anay@linux:~$ ls
marks.txt  sample.txt
anay@linux:~$ cat sample.txt | tee sample1.txt | cat
anay
akash
suman
anil

anay@linux:~$ cat sample1.txt
anay
akash
suman
anil

anay@linux:~$
```

7. tr

   The command 'tr' stands for 'translate'. It is used to translate, like from lowercase to uppercase and vice versa or new lines into spaces.

```
anay@linux:~$ cat sample1.txt | tr 'na' 'NA'
ANAy
AkAsh
sumAN
ANil

anay@linux:~$
```

8. uniq

   With the help of uniq command we can form a sorted list in which every word will occur only once.

```
anay@linux:~$ cat sample.txt
anay
anay
akash
suman
anil
suman
anay@linux:~$ sort sample.txt | uniq
akash
anay
anil
suman
anay@linux:~$
```

9. wc

   The 'wc' command helps in counting the lines, words and characters in a file.

```
anay@linux:~$ wc sample.txt
 6  6 33 sample.txt
anay@linux:~$ wc -l sample.txt
6 sample.txt
anay@linux:~$ wc -w sample.txt
6 sample.txt
anay@linux:~$ wc -c sample.txt
33 sample.txt
anay@linux:~$
```

10. od

The 'od' term stands for octal dump. It displays content of a file in different human-readable formats like hexadecimal, octal and ASCII characters

```
anay@linux:~$ od -b sample.txt
0000000 141 156 141 171 012 141 156 141 171 012 141 153 141 163 150 012
0000020 163 165 155 141 156 012 141 156 151 154 012 163 165 155 141 156
0000040 012
0000041
anay@linux:~$ od -t xl sample.txt
0000000 61 6e 61 79 0a 61 6e 61 79 0a 61 6b 61 73 68 0a
0000020 73 75 6d 61 6e 0a 61 6e 69 6c 0a 73 75 6d 61 6e
0000040 0a
0000041
```

11. sort

The 'sort' command sorts the file content in an alphabetical order.

```
anay@linux:~$ sort sample.txt
akash
anay
anay
anil
suman
suman
anay@linux:~$ sort sample1.txt

akash
anay
anil
suman
anay@linux:~$
```

12. gzip

Gzip (GNU zip) is a compressing tool, which is used to truncate the file size. By default original file will be replaced by the compressed file ending with extension (.gz).

To decompress a file we can use gunzip command and wer original file will be back.

Syntax:

$ gzip <file1> <file2> <file3>. . .
$ gunzip <file1> <file2> <file3>. . .

Linux Regular Expression

Regular expression is also called regex or regexp. It is a very powerful tool in Linux. Regular expression is a pattern for a matching string that follows some pattern.

Regex can be used in a variety of programs like grep, sed, vi, bash, rename and many more.

Regular Expression Meta-characters

A regular expression may have one or several repeating meta-characters.

Meta-character	Description
.	Replaces any character.
^	Matches start of string and represents characters not in the string.
$	Matches end of string.
*	Matches zero or more times the preceding character.
\	Represents the group of characters.
()	Groups regular expressions.
?	Matches exactly one character.
+	Matches one or more times the preceding character.
{N}	Preceding character is matched exactly N times.
{N,}	Preceding character is matched exactly N times or more.
{N,M}	Preceding character is matched exactly N times, but not more than N times.
-	Represents the range.
\b	Matches empty string at the edge of a word.
\B	Matches empty string if it is not at the edge of a word.
\<	Matches empty string at the beginning of a word.
\>	Matches empty string at the end of a word.

We can use these meta-char with cat commands.

Eg. If we want to check that the character 'p' appears exactly 2 times in a string one after the other. For this the syntax would be:

```
$ cat sample | grep -E p\{2}
```

Linux File Ownership

Every Linux system have three types of owner:

User: A user is the one who created the file. By default, whosoever, creates the file becomes the owner of the file. A user can create, delete, or modify the file.

Group: A group can contain multiple users. All the users belonging to a group have same access permission for a file.

Other: Any one who has access to the file other than user and group comes in the category of other. Other has neither created the file nor is a group member.

Users and groups can be locally managed in /etc/psswd or /etc/group.

Syntax:

```
$ ls -lh
```

```
root@linux:/home# ls -lh
total 12K
drwxr-xr-x 5 anay          anay          4.0K Sep 14 08:48 anay
drwxr-xr-x 5 anay_valuemom anay_valuemom 4.0K Sep 13 09:26 anay_valuemom
drwxr-xr-x 6 ubuntu        ubuntu        4.0K Sep 13 08:42 ubuntu
root@linux:/home#
```

Look at the above snapshot, all the listed files and directories have the same user and group that is anay, anay_valuemom, ubuntu. First column denotes the user and second column denotes the group.

Listing User Accounts

To know the local users account, following command can be used. It list out all the local users from the system.

Syntax:

```
$ cut -d: -f1 /etc/passwd | column
```

```
root@linux:/home# cut -d: -f1 /etc/passwd | column
root            lp              list            messagebus      pollinate
daemon          mail            irc             _apt            _chrony
bin             news            gnats           lxd             ubuntu
sys             uucp            nobody          uuidd           anay_valuemom
sync            proxy           systemd-network dnsmasq         anay
games           www-data        systemd-resolve landscape
man             backup          syslog          sshd
root@linux:/home#
```

Linux chgrp: change group

The chgrp command can be abbreviated as change group. We can change the group owner of the file using chgrp command.

Syntax:

```
$ chgrp <newGroup> <fileName>
```

```
root@linux:/home/ubuntu# ls -l
total 12
drwxr-xr-x 2 root root 4096 Sep 13 08:33 ABCD
drwxr-xr-x 2 root root 4096 Sep 13 09:06 ABCD1
-rw-r--r-- 1 root root    0 Sep 13 08:25 abc.txt
drwxr-xr-x 2 root root 4096 Sep 13 09:09 abcd
-rw-r--r-- 1 root root    0 Sep 13 08:25 sample.txt
root@linux:/home/ubuntu# chgrp anay ABCD
root@linux:/home/ubuntu# ls -l
total 12
drwxr-xr-x 2 root anay 4096 Sep 13 08:33 ABCD
drwxr-xr-x 2 root root 4096 Sep 13 09:06 ABCD1
-rw-r--r-- 1 root root    0 Sep 13 08:25 abc.txt
drwxr-xr-x 2 root root 4096 Sep 13 09:09 abcd
-rw-r--r-- 1 root root    0 Sep 13 08:25 sample.txt
root@linux:/home/ubuntu#
```

In the above example, the dir. ‘ABCD’ (highlighted lines) was of root group and after executing the command ‘chgrp anay ABCD’, we can see that the group name is ‘anay’ .

Note: Only root user have the permission to change the owner or group of the files in the system.

Linux chown: change owner

Command chown is used to change the owner of the file.

Syntax:

```
$ chown <newowner> <fileName>
```

In the example given below, Dir ‘ABCD’’s owner was root and after executing the command ‘chown anay ABCD’, the owner is ‘anay’.

Note: Only root user have the permission to change the owner or group of the files in the system.

```
root@linux:/home/ubuntu# ls -l
total 12
drwxr-xr-x 2 root anay 4096 Sep 13 08:33 ABCD
drwxr-xr-x 2 root root 4096 Sep 13 09:06 ABCD1
-rw-r--r-- 1 root root    0 Sep 13 08:25 abc.txt
drwxr-xr-x 2 root root 4096 Sep 13 09:09 abcd
-rw-r--r-- 1 root root    0 Sep 13 08:25 sample.txt
root@linux:/home/ubuntu# chown anay ABCD
root@linux:/home/ubuntu# ls -l
total 12
drwxr-xr-x 2 anay anay 4096 Sep 13 08:33 ABCD
drwxr-xr-x 2 root root 4096 Sep 13 09:06 ABCD1
-rw-r--r-- 1 root root    0 Sep 13 08:25 abc.txt
drwxr-xr-x 2 root root 4096 Sep 13 09:09 abcd
-rw-r--r-- 1 root root    0 Sep 13 08:25 sample.txt
root@linux:/home/ubuntu#
```

List of Special Files

When we type ls -l command, ten characters are displayed before user owner and group. First character tells us about the type of the file.

Following are the file types:

First Character	File Type
-	Normal file
d	Directory
l	Symbolic link
p	Named pipe
b	Blocked device
c	Character device
s	Socket

```
root@linux:/home/ubuntu# ls -ld /dev/sda
brw-rw---- 1 root disk 8, 0 Sep 14 08:43 /dev/sda
root@linux:/home/ubuntu#
```

Look at the above snapshot, first latter ‘b’ denotes blocked device.

```
root@linux:/home/ubuntu# ls
ABCD  ABCD1  abc.txt  abcd  sample.txt
root@linux:/home/ubuntu# ls -ld sample.txt
-rw-r--r-- 1 root root 0 Sep 13 08:25 sample.txt
root@linux:/home/ubuntu#
```

And here in above snapshot '-' denotes the normal file.

File Permissions

All the three owners (user owner, group, others) in the Linux system have three types of permissions defined. Nine characters denotes the three types of permissions.

Read (r) : The read permission allows we to open and read the content of a file. But we can't do any editing or modification in the file.

Write (w) : The write permission allows we to edit, remove or rename a file. For instance, if a file is present in a directory, and write permission is set on the file but not on the directory, then we can edit the content of the file but can't remove, or rename it.

Execute (x): In Unix type system, we can't run or execute a program unless execute permission is set. But in Windows, there is no such permission available.

Permissions are listed below:

permission	on a file	on a directory
r (read)	read file content (cat)	read directory content (ls)
w (write)	change file content (vi)	create file in directory (touch)
x (execute)	execute the file	enter the directory (cd)

```
root@linux:/home/ubuntu# ls -ld ABCD
drwxr-xr-x 2 anay anay 4096 Sep 13 08:33 ABCD
root@linux:/home/ubuntu# ls
```

File permissions for (-rw-rw-r--)

position	characters	ownership
1	-	denotes file type
2-4	rw-	permission for user
5-7	rw-	permission for group
8-10	r--	permission for other

When we are the User owner, then the user owner permission applies to we. Other permissions are not relevant to we.

When we are the Group then the group permission applies to we. Other permissions are not relevant to we.

When we are the Other, then the other permission applies to we. User and group permissions are not relevant to we.

Setting Permissions With chmod

We can change the permissions with chmod command accordingly to wer need. Below are some examples to change the permissions for different groups.

To add permissions to a group

Syntax:

```
$ chmod <groupName>+<permissionName> <fileName>
```

```
root@linux:/home/ubuntu# ls -l Sample.java
-rw-r--r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod u+x Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rwxr--r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

Look at the above snapshot, permission to execute is added to the user owner group.

To remove permissions from a group

Syntax:

```
$ chmod <groupName>-<permissionName> <fileName>
```

```
root@linux:/home/ubuntu# ls -l Sample.java
-rwxr--r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod u-x Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rw-r--r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

To add permission to all the groups together

Syntax:

```
$ chmod a+<permissionName> <fileName>
```

```
root@linux:/home/ubuntu# ls -l Sample.java
-rw-r--r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod a+w Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rw-rw-rw- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

To add permission to all the groups without typing a

Syntax:

```
$ chmod +<permissionName> <fileName>
```

```
root@linux:/home/ubuntu# ls -l Sample.java
-rw-rw-rw- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod +x Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rwxrwxrwx 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

To set explicit permission

Syntax:

```
$ chmod <groupName>=<permissions> <fileName>
```

To set explicit permissions for different groups

Syntax:

```
$ chmod <groupName>=<permissions> <fileName>
```

```
root@linux:/home/ubuntu# ls -l Sample.java
-rwxrwxrwx 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod u=rwx,g=rw,o=r Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rwxrw-r-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

Setting Octal Permissions

Octal permissions can also be set for the groups.

For example, to set r octal will be 4, to set w octal will be 2, to set x octal will be 1.

Octal Table:

binary	octal	permissions
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--

101	5	r-x
110	6	rw-
111	7	rwx

From this we can conclude that,

777 = rwxrwxrwx

765 = rwxrw-r-x

654 = rw-r-xr--

```
root@linux:/home/ubuntu# chmod 777 Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
-rwxrwxrwx 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod 274 Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
--w-rwxr-- 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu# chmod 111 Sample.java
root@linux:/home/ubuntu# ls -l Sample.java
---x--x--x 1 root root 0 Sep 14 10:10 Sample.java
root@linux:/home/ubuntu#
```

Look at the above snapshot, we have shown some random octal examples with the numbers 777, 274 and 111.

umask

While creating a file or directory, by default a set of permissions are applied. These default permissions are viewed by umaskcommand.

For safety reasons all Unix systems doesn't provide execution permission to newly created files.

Adding execution permission is upto we.

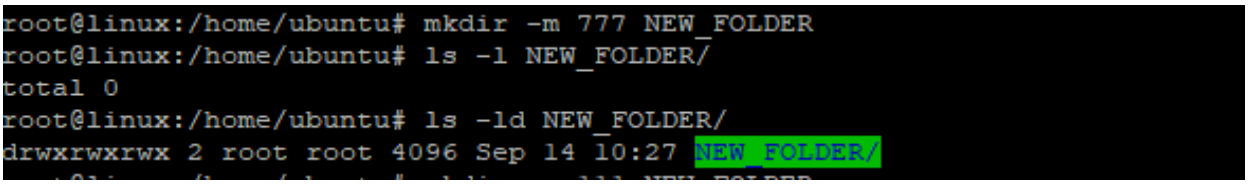
```
root@linux:/home/ubuntu# umask
0022
root@linux:/home/ubuntu# touch Sample_file.txt
root@linux:/home/ubuntu# ls -l Sample_file.txt
-rw-r--r-- 1 root root 0 Sep 14 10:26 Sample_file.txt
root@linux:/home/ubuntu#
```

mkdir -m

The 'mkdir -m' command can be used to set the mode.

Syntax:

```
$ mkdir -m <mode> <fileName>
```



cp -p

The 'cp -p' command preserves the permissions and time stamps from source files.

Syntax:

```
$ cp -p <sourceFile> <destinationFile>
```

Processes in Linux/Unix

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

Whenever a command is issued in unix/linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.

Through a 5 digit ID number unix/linux keeps account of the processes, this number is call process id or pid. Each process in the system has a unique pid.

Used up pid's can be used in again for a newer process since all the possible combinations are used.

At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

Initializing a process

A process can be run in two ways:

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

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 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.

```
anay@linux:~$ ls
marks.txt  sample.txt  sample1.txt
anay@linux:~$ pwd &
[2] 2011
/home/anay
anay@linux:~$
```

Since pwd does not wants any input from the keyboard, it goes to the stop state until moved to the foreground and given any data input.

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

Tracking ongoing processes

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

```
anay@linux:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
anay        1643  1642  0 13:52 pts/0    00:00:00 -bash
anay        1836  1643  0 14:11 pts/0    00:00:00 nano sample
anay        2033  1643  0 14:34 pts/0    00:00:00 ps -f
anay@linux:~$
```

Stopping a process

When running in foreground, hitting Ctrl + c (interrupt character) will exit the command. For processes running in background kill command can be used if it's pid is known.

```
anay@linux:~$ ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
anay        1643  1642  0 13:52 pts/0    00:00:00 -bash
anay        1836  1643  0 14:11 pts/0    00:00:00 nano sample
anay        2055  1643  0 14:36 pts/0    00:00:00 ps -f
anay@linux:~$ kill 1643
anay@linux:~$
```

Other process commands:

bg: A job control command that resumes suspended jobs while keeping them running in the background
Syntax:

```
$ bf [job]
```

fg: It continues a stopped job by running it in the foreground.
Syntax:

```
$ fg [ %job_id ]
```

top: This command is used to show all the running processes within the working environment of Linux.
Syntax:

```
$ top
```

nice: It starts a new process (job) and assigns it a priority (nice) value at the same time.
Syntax:

```
$ nice [-nice value]
```

renice : To change the priority of an already running process renice is used.
Syntax:

```
$ renice [-nice value] [process id]
```

df: It shows the amount of available disk space being used by file systems
Syntax:

```
$ df
```

free: It shows the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel
Syntax:

```
$ free
```

Types of Processes

Parent and Child process: The 2nd and 3rd column of the ps –f command shows process id and parent's process id number. For each user process there's a parent process in the system, with most of the commands having shell as their parent.

Zombie and Orphan process: After completing its execution a child process is terminated or killed and SIGCHLD updates the parent process about the termination and thus can continue the task assigned to it. But at times when the parent process is killed before the termination of the child process, the child processes becomes orphan processes, with the parent of all processes "init" process, becomes their new ppid.
A process which is killed but still shows its entry in the process status or the process table is called a zombie process, they are dead and are not used.

Daemon process: They are system-related background processes that often run with the permissions of root and services requests from other processes, they most of the time run in the background and wait for processes it can work along with for ex print daemon.
When ps –ef is executed, the process with ? in the tty field are daemon processes

Linux Iptables:

Firewall decides fate of packets incoming and outgoing in system. IPTables is a rule based firewall and it is pre-installed on most of Linux operating system. By default it runs without any rules. IPTables was included in Kernel 2.4, prior it was called ipchains or ipfwadm. IPTables is a front-end tool to talk to the kernel and decides the packets to filter. This guide may help we to rough idea and basic commands of IPTables where we are going to describe practical iptables rules which we may refer and customized as per wer need.

Different services is used for different protocols as:

- iptables applies to IPv4.
- ip6tables applies to IPv6.
- arptables applies to ARP.

ebtables applies to Ethernet frames..

There are at present three tables.

- Filter
- NAT
- Mangle

At present, there are total four chains:

- INPUT: Default chain originating to system.
- OUTPUT: Default chain generating from system.
- FORWARD: Default chain packets are send through another interface.
- RH-Firewall-1-INPUT: The user-defined custom chain.

Example 1: Checking the status of IPTables / Firewall. Options "-L" (List ruleset), "-v" (Verbose) and "-n" (Displays in numeric format).

```
root@linux:~# iptables -L -n -v
Chain INPUT (policy ACCEPT 4477 packets, 2474K bytes)
 pkts bytes target     prot opt in     out     source               destination
 4477 2474K sshguard   all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 3561 packets, 369K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain sshguard (1 references)
 pkts bytes target     prot opt in     out     source               destination
root@linux:~#
```

Example2: Flushing or deleting IPTables rules.

Below command will remove all the rules from tables. Take rulesets backup before executing above command.

```
root@linux:~# iptables -F
root@linux:~# iptables -L -n -v --line-numbers
Chain INPUT (policy ACCEPT 6 packets, 432 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 4 packets, 464 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain sshguard (0 references)
num   pkts bytes target     prot opt in     out     source               destination
root@linux:~#
```

Example3: Block an Ip address:

```
$ iptables –A INPUT –s <ip_address> -j DROP
```

Example4: Block connection to a network interface :

```
$ iptables –A INPUT –i  eth0 –s <ip_address> -j DROP
```

Example5: Allow incoming HTTP

```
$ iptables –A INPUT –p tcp –dport 80 –m conntrack –ctstate NEW,ESTABLISHED –j ACCEPT
```

```
$ iptables –A OUTPUT –p tcp –sport 80 –m conntrack –ctstate ESTABLISHED –j ACCEPT
```

The second command, which allows the outgoing traffic of **established** HTTP connections, is only necessary if the OUTPUT policy is not set to ACCEPT.

VI Text 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 –

- It is available in almost all Linux Distributions
- It works the same across different platforms and Distributions
- It is user-friendly. Hence, millions of Linux users love it and use it for their editing needs

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

To work on VI editor, we need to understand its operation modes. They can be divided into two main parts.

Command mode:

- The vi editor opens in this mode, and it only understands commands
- In this mode, we can, move the cursor and cut, copy, paste the text
- This mode also saves the changes we have made to the file
- Commands are case sensitive. We should use the right letter case.

Insert mode:

- This mode is for inserting text in the file.
- We can switch to the Insert mode from the command mode by pressing 'i' on the keyboard
- Once we are in Insert mode, any key would be taken as an input for the file on which we are currently working.
- To return to the command mode and save the changes we have made we need to press the Esc key

Starting the vi editor

To launch the VI Editor -Open the Terminal (CLI) and type

```
$ vi <filename_NEW> or <filename_EXISTING>
```

&If we specify an existing file, then the editor would open it for we to edit. Else, we can create a new file.

vi Editing commands

Note: We should be in the "command mode" to execute these commands. VI editor is case-sensitive so make sure we type the commands in the right letter-case.

Keystrokes	Action
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 3dd	Delete line 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 4dw	Delete word 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

Make sure we press the right command otherwise we will end up making undesirable changes to the file. We can also enter the insert mode by pressing a, A, o, as required.

Moving within a file

We need to be in the command mode to move within a file. The default keys for navigation are mentioned below else; We can also use the arrow keys on the keyboard.

Keystroke	Use
k	Move cursor up
j	Move cursor down
h	Move cursor left
l	Move cursor right

Saving and Closing the file

We should be in the command mode to exit the editor and save changes to the file.

Keystroke	Use
Shift+zz	Save the file and quit
:w	Save the file but keep it open
:q	Quit without saving
:wq	Save the file and quit