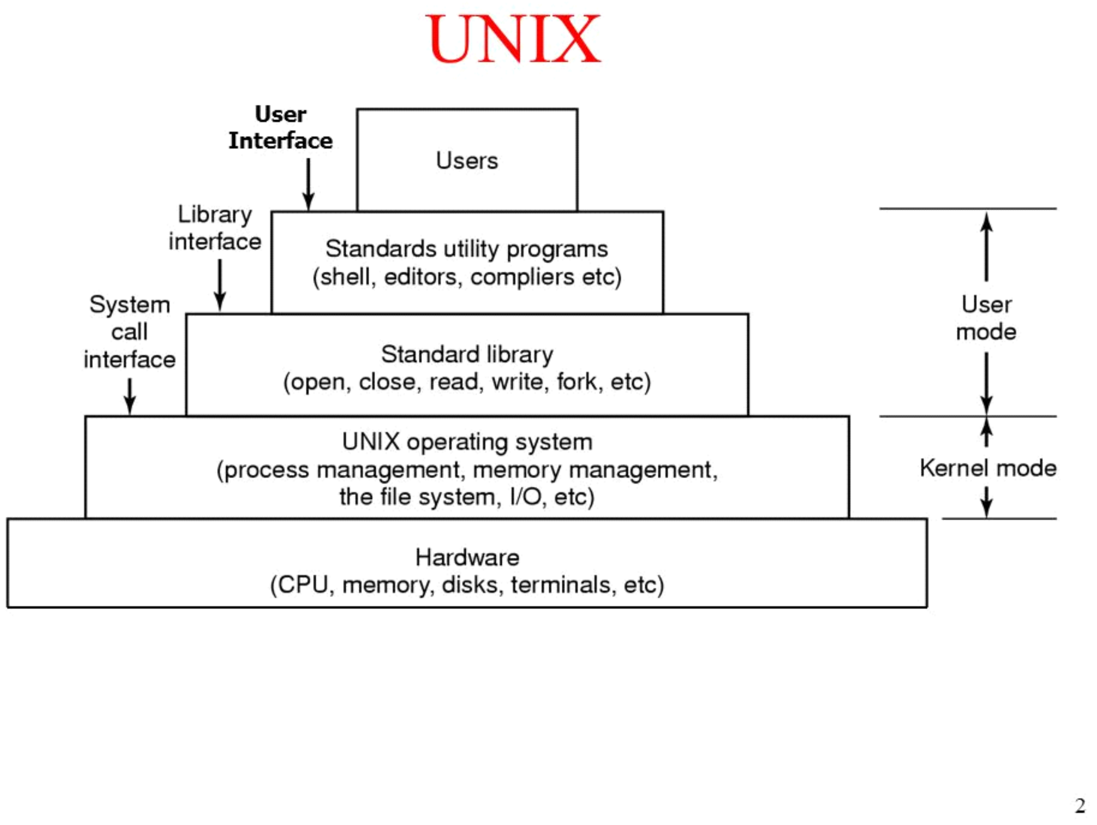
<u>UNIT-1</u>

<u>HISTORY AND INTRODUCTION OF UNIX</u>

The Unix operating system found its beginnings in MULTICS, which stands for Multiplexed Operating and Computing System. The MULTICS project began in the mid 1960s as a joint effort by General Electric, Massachusetts Institute for Technology and Bell Laboratories. In 1969 Bell Laboratories pulled out of the project.

One of Bell Laboratories people involved in the project was Ken Thompson. He liked the potential MULTICS had, but felt it was too complex and that the same thing could be done in simpler way. In 1969 he wrote the first version of Unix, called UNICS. UNICS stood for Uniplexed Operating and Computing System. Although the operating system has changed, the name stuck and was eventually shortened to Unix.

Ken Thompson teamed up with Dennis Ritchie, who wrote the first C compiler. In 1973 they rewrote the Unix kernel in C. The following year a version of Unix known as the Fifth Edition was first licensed to universities. The Seventh Edition, released in 1978, served as a dividing point for two divergent lines of Unix development. These two branches are known as SVR4 (System V) and BSD.


UNIX
User Interface
Users
Library interface
Standards utility programs (shell, editors, compliers etc)
System call interface
Standard library (open, close, read, write, fork, etc)
UNIX operating system (process management, memory management, the file system, I/O, etc)
Hardware (CPU, memory, disks, terminals, etc)
User mode
Kernel mode
2

Why Use Unix?

One of the biggest reasons for using Unix is networking capability. With other operating systems, additional software must be purchased for networking. With Unix, networking capability is simply part of the operating system. Unix is ideal for such things as world wide e-mail and connecting to the Internet.

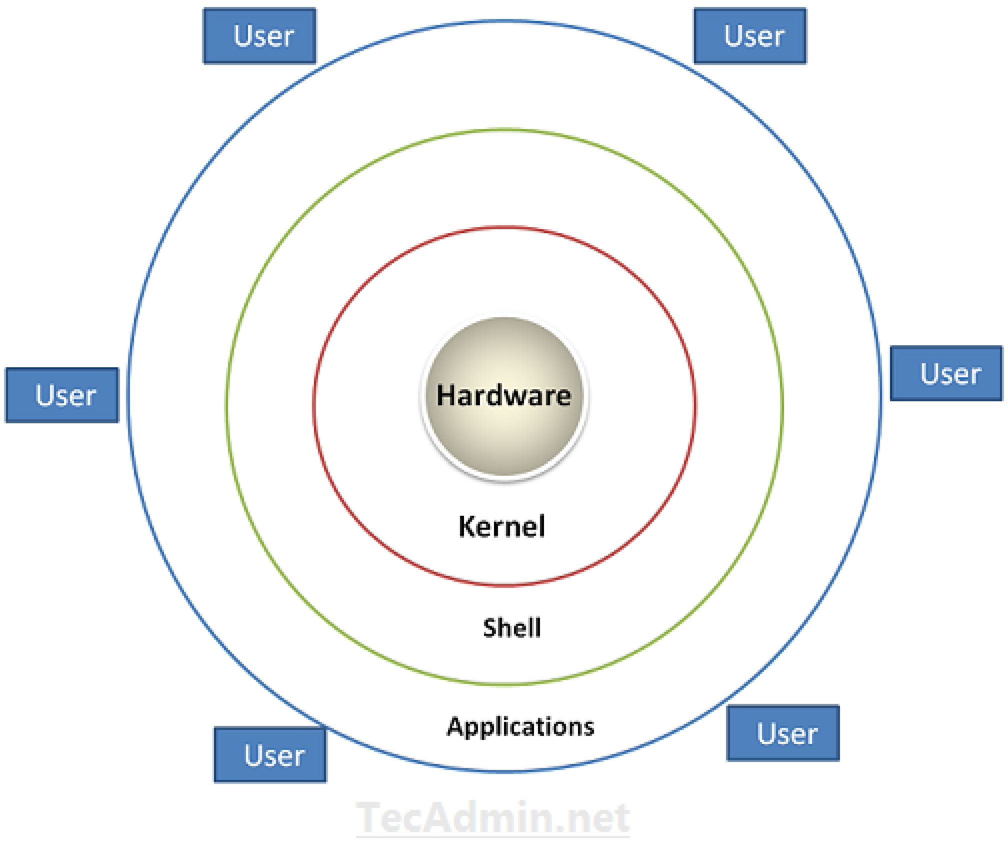
Unix was written in a machine independent language. So Unix and unix-like operating systems can run on a variety of hardware. These systems are available from many different sources, some of them at no cost. Because of this diversity and the ability to utilize the same "user-interface" on many different systems, Unix is said to be an open system.

<u>HISTORY AND INTRODUCTION OF LINUX:</u>

In 1991 Linus Torvalds began developing an operating system kernel, which he named “Linux” [Torvalds 1999]. This kernel could be combined with the FSF material and other components (in particular some of the BSD components and MIT’s X-windows software) to produce a freely-modifiable and very useful operating system. This book will term the kernel itself the “Linux kernel” and an entire combination as “Linux”. Note that many use the term “GNU/Linux” instead for this combination.

In the Linux community, different organizations have combined the available components differently. Each combination is called a “distribution”, and the organizations that develop distributions are called “distributors”. Common distributions include Red Hat, Mandrake, SuSE, Caldera, Corel, and Debian. There are differences between the various distributions, but all distributions are based on the same foundation: the Linux kernel and the GNU glibc libraries.

Architecture of Linux:



Linux System Architecture

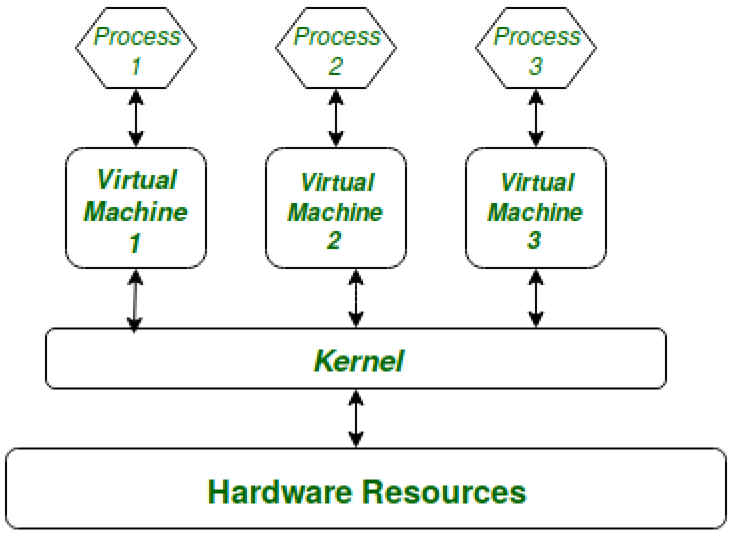
Hardware – Hardware consists of all physical devices attached to the System. For example: Hard disk drive, RAM, Motherboard, CPU etc.

Kernel – Kernel is the core component for any (Linux) operating system which directly interacts with the hardware.

Shell – Shell is the interface which takes input from Users and sends instructions to the Kernel, Also takes the output from Kernel and send the result back to output shell.

Applications – These are the utility programs which runs on Shell. This can be any application like Your web browser, media player, text editor etc.

In a General Purpose Computer running many processes simulteneously, we need a middle layer to manage the distribution of the hardware resources of the computer efficiently and fairly among all the various processes running on the computer. This middle layer is referred to as the **kernel**. Basically the kernel virtualizes the common hardware resources of the computer to provide each process with its own virtual resources. This makes the process seem as it is the sole process running on the machine. The kernel is also responsible for preventing and mitigating conflicts between different processes.
This schematically represented below:



Per-Process Hardware Virtualization

The **Core Subsystems** of the **Linux Kernel** are as follows:

1. The Process Scheduler
2. The Memory Management Unit (MMU)
3. The Virtual File System (VFS)
4. The Networking Unit
5. Inter-Process Communication Unit

<u>Types of Shell</u>

1. In addition to graphical user interfaces like Gnome, KDE and MATE, the Linux operating system also offers several shells.

2. These command-line interfaces provide powerful environments for software development and system maintenance.

3. Some of the shells available in Linux are: sh, bash, csh, tcsh, etc.

4. sh: The Bourne shell, called "sh," is one of the original shells, developed for Unix computers by Stephen Bourne at AT&T's Bell Labs in 1977.

5. It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.

6. bash: "Bash" -- the "Bourne-again Shell," based on sh -- has become the new default standard.

7. One attractive feature of bash is its ability to run sh shell scripts unchanged. Conveniences include command completion and a command history.
8. zsh is improved version of bash.

9. csh and tcsh: Using C syntax as a model, Bill Joy at Berkeley University developed the "C-shell," csh, in 1978.

10. Ken Greer, working at Carnegie-Mellon University, developed tcsh. Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type.

11. ksh: David Korn developed the Korn shell, or ksh. Ksh is compatible with sh and bash.

12. Ksh improves on the Bourne shell by adding floating-point arithmetic, job control, command aliasing and command completion.

Benefits of shell:

- To automate the frequently performed operations
- To run sequence of commands as a single command
- Easy to use
- Portable (It can be executed in any Unix-like operating systems without any modifications)

Linux I/O Redirection

Redirection can be defined as changing the way from where commands read input to where commands sends output. You can redirect input and output of a command.

For redirection, meta characters are used. Redirection can be into a **file** (shell meta characters are angle **brackets** '<', '>') or a **program** (shell meta characters are **pipe**symbol '|').

Standard Streams In I/O Redirection

The bash shell has three standard streams in I/O redirection:

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

Redirection Into A File

Each stream uses redirection commands. Single bracket '>' or double bracket '>>' can be used to redirect standard output. If the target file doesn't exist, a new file with the same name will be created.

Overwrite

Commands with a single bracket **'>' overwrite** existing file content.

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

Note: Writing **'1>'** or **'>'** and **'0<'** or **'<'** is same thing. But for stderr you have to write **'2>'**.

Syntax:

```
cat > <fileName>
```

Example:

```
cat > sample.txt
```

Piping in Unix or Linux

A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing. The Unix/Linux systems allow stdout of a command to be connected to stdin of another command. You can make it do so by using the pipe character '|'.
Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands/ programs/ processes. The command line programs that do the further processing are referred to as filters.

This direct connection between commands/ programs/ processes allows them to operate simultaneously and permits data to be transferred between them continuously rather than having to pass it through temporary text files or through the display screen.
Pipes are unidirectional **i.e data flows from left to right through the pipeline.**
Syntax :

```
command_1 | command_2 | command_3 | .... | command_N
```

Example :
1. Listing all files and directories and give it as input to more command.

```
$ ls -l | more
```

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
2. cut
3. grep
4. comm
5. sed
6. tee
7. tr
8. uniq
9. wc
10. od
11. sort
12. gzip

Text Editors for Linux :

1. VIM
If you are bored of using the default "**vi**" editor in linux and want to edit your text in an advanced text editor that is packed with powerful performance and lots of options, then vim is your best choice. As per the name suggests, VIM means "**vi improved**" as it is just an advanced version of the default linux text editor. It is specially designed keeping in mind the needs of a developer. It is also called as a programmers editor for its highly configurable options. Similar to the Vi editor, it can be used as a command line utility or also as a standalone GUI application.
Some of the unique features of VIM includes:

- Syntax Coloring
- Tag System
- Tab expansion
- Session Screen
- Split screen
- Digraph input
- Automatic commands

2. Geany
Geany is one of the most popular text editors for Linux desktop environment that comes with an integrated **GTK+** toolkit. It also serves as an excellent development environment for programmers and developers. If you are looking for text editor that also doubles up as a development environment, then Geany is your best bet. It is lightweight and supports almost all major programming languages and doesn't have many dependencies from other packages.
Some of the unique features of Geany includes:

- Easy to use and clean interface
- Syntax highlighting for easy development
- Lots of customizable options
- Line numbering for easy tracking of code
- Easy pluggable interface

3. Sublime Text Editor
Sublime text editor is another popular text editor for the linux environment. It is packed with a lot of features and is specially designed to be used as a text editor and also as a development environment. It supports a lot of programming along with many markup languages. With the numerous plugins available, you can take the text editor to the next level by extending its functionality to a great extent. One of the unique features of the text editor is the "Goto Anything" feature that helps you to easily go to any section of the code or navigate to any file in your system.

Some of the other unique features of Sublime text editor includes:

- Excellent Command Palette

- Python-based plugin API
- Parallel editing of Code
- Project specific preferences

4. Brackets

Adobe launched a text editor way back in 2014 called the Brackets for the linux environment. It is an open source text editor that comes packed with a lot of exciting features that makes working with this text editor a lot of fun. It is also easy and simple to use with a clean interface. It is specially designed to act as both as text editor as well as a code editor to help web designers and programmers. It's completely developed using HTML, CSS and JavaScript. It is lightweight, but still has all the qualities to beat some of the best text editors with its advanced features.

Some of the unique features of Brackets text editor includes:

- Live Preview
- Inline Editing
- Focused visual tools Pre processor support

5. Gedit

If you are working in a GNOME desktop environment, then by default it comes loaded with a text editor called Gedit. Similar to the objective of GNOME to always provide functionalities that are clean and straightforward, Gedit also follows the same objective as it is lightweight and comes with clean and simple user interface. It first got released to the public in 2000 with the GNOME desktop environment. It is completed developed using C language and supports completely for internationalized text.

Some of the unique features of Gedit includes:

- Syntax highlighting
- Supports internationalized text
- Supports various programming languages

6. Kate

If you are familiar with the Kubuntu desktop environment then you would have definitely know about Kate text editor that comes as a default editor loaded with the Kubuntu environment. It is a lightweight and easy to use text editor. You can work with multiple files simultaneously. It also can be used a powerful IDE.

Some of the unique features of Kate includes:

- A powerful IDE
- Supports many languages
- Auto-detects languages
- Sets indentation for documents automatically

7. Eclipse

Front end developers and designers looking for a robust and advanced text/code editor can definitely go for the Eclipse editor. It is popular among many java developers as it is completely developed in JAVA and is also contains a lot of features that supports writing and developing Java application easily. If you need additional language support, then you need to install extra plugins to achieve this. The Eclipse IDE becomes even more powerful with the help of additional plugins as you can insert a lot of advanced functionalities to the editor. It can also be used to develop programs for PHP, Python, C, C++, Ruby on Rails, COBOL etc.

Some of the unique features of Eclipse includes:

- Free and open source text editor
- Includes Java Development tools for Java developers
- Plugin Support

8. Kwrite

Kwrite text editor is developed by KDE and first released to the public in 2000. It is entirely based on the Kate text editor along with the KParts technology from KDE. With the help of additional plugin installation, you can extend the functionality of Kwrite to a great extent and make it a more powerful development environment. It can also be used to edit remote file along with encoding your files.

Some of the unique features of Kwrite includes:

- Word Completion
- Auto Indentation
- Syntax highlighting
- vi input mode

9. Nano

Nano is another popular text editor that is also used in the UNIX operating systems. It is similar to the Pico text editor and first got released in 2000. It also comes packed with a lot of additional functionalities to make this as a powerful and advanced text editor. It can be run in a command line interface only.

Some of the unique features of Nano includes:

- Case sensitive search
- Auto Indentation
- Tab Completion
- Autoconf support

10. GNU Emacs

GNU Emacs is one of the oldest text editor for the linux environment that has been here for a long time. It is developed by Richard Stallman, the project founder of GNU. It is being used by thousands of linux programmers all around the world and GNU Emacs is one of their favorite and preferred text editors. It is entirely developed using LISP and C.

Some of the unique features of GNU Emacs includes:

- Mail and News options
- Debugger interface extension
- Extensive documentation and support

Unit-2

Rule of creating files

There are many text editors like (vim, nano, vi) and many commands like (cat, echo, printf, touch) to create a file in the Linux operating system via command line. This article will contain on the following tools:

1. Touch command
2. Cat command
3. Echo command
4. Printf command
5. Nano text editor
6. Vi text editor
7. Vim text editor

1) Create a file with touch command

we will use `touch` command with any extension to create file, this command will create an empty file `touch.txt` in your current directory as an example below.

```
$ sudo touch touch.txt

$ sudo touch touch.docx
```

To see the file type command below.

```
$ ls -l

output

-rw-r--r--. 1 root root          0 Sep    4 08:08 touch.docx

-rw-r--r--. 1 root root          0 Sep    4 08:06 touch.txt
```

2) Create a file with cat command

We will use `cat` command to create file, this command will create an empty file `cat.txt` in your current directory as an example below, but you must add text in the file.

```
$ cat > cat.txt
```

Add the text below.

```
This file has been created with cat command
```

To save the file hit `Ctrl + d`, and to see the file type command below.

```
$ ls -l cat.txt

output

-rw-r--r--. 1 root root         0 Sep  4 08:06 cat.txt
```

To open the file, we will use `cat` command to open it.

```
$ cat cat.txt

output

This file has been created with echo command
```

3) Create a file with echo command

We will use `echo` command to create file, this command will create a file `echo.txt` in your current directory as an example below, but you should add text in the line command.

```
$ echo "This file has been created with echo command" > echo.txt
```

To see the file,type command below.

```
$ ls -l echo.txt

output
```

```
-rw-r--r--. 1 root root           0 Sep   4 08:06 echo.txt
```

To open the file, we will use `cat` command to open it.

```
$ cat echo.txt

output

This file has been created with echo command
```

4) Create a file with printf command

We will use `printf` command to create file, this command will create a file `printf.txt` in your current directory as an example below, but you should add text in the line command.

```
$ printf "This file has been created with printf command" > printf.txt
```

To see the file type command below.

```
$ ls -l printf.txt

output

-rw-r--r--. 1 root root           0 Sep   4 08:06 printf.txt
```

To open the file, we will use `cat` command to open it.

```
$ cat printf.txt

output

This file has been created with printf command
```

5) Create a file with nano text editor

To create a file using nano text editor, first install it, after that type command below and the text editor will be opened to adding text.

```
$ nano nano.txt
```

Add the text below.

```
This file has been created with nano text editor
```

To save the file type `Ctrl + x` and type `y`, to see the file type command below.

```
$ ls -l nano.txt

output

-rw-r--r--. 1 root root          0 Sep   4 08:06 nano.txt
```

To open the file, We will use `nano` command to open it.

```
$ nano nano.txt

output

This file has been created with nano command
```

6) Create a file with vi text editor

To create a file using vi text editor, type command below and the text editor will open the file, but you can't add any text before converting it to insert mode by typing `i` character.

```
$ vi vi.txt
```

Add the text below.

```
This file has been created with vi text editor
```

To save the file and exit hit `Esc` after that `:wq`, To see the file type command below.

```
$ ls -l vi.txt

output

-rw-r--r--. 1 root root          0 Sep   4 08:06 vi.txt
```

To open the file, we will use `vi` command to open it.

```
$ vi vi.txt

output

This file has been created with vi command
```

7) Create a file with vim text editor

To create a file using vim text editor, type command below and the text editor will open the file, but you can't add any text before converting it to insert mode by typing `i` character.

```
$ vim vim.txt
```

Add the text below.

```
This file has been created with vim text editor
```

To save the file and exit hit `Esc` after that `:wq`, to see the file type command below.

```
$ ls -l vim.txt

output

-rw-r--r--. 1 root root          0 Sep   4 08:06 vim.txt
```

To open the file, we will use `vim` command to open it.

```
$ vim vim.txt

output

This file has been created with vim command
```

Linux file system-

A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.

Your hard drive can have various partitions which usually contain only one file system, such as one file system housing the **/file system** or another containing the **/home file system**.

One file system per partition allows for the logical maintenance and management of differing file systems.

Everything in Unix is considered to be a file, including physical devices such as DVD-ROMs, USB devices, and floppy drives.

Directory Structure

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A Unix filesystem is a collection of files and directories that has the following properties −

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an **inode**.
- By convention, the root directory has an **inode** number of **2** and the **lost+found** directory has an **inode** number of **3**. Inode numbers **0** and **1** are not used. File inode numbers can be seen by specifying the **-i option** to **ls command**.
- It is self-contained. There are no dependencies between one filesystem and another.

The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix −

Sr.No.	Directory & Description
1	/ This is the root directory which should contain only the directories needed at the top level of the file structure

2	**/bin** This is where the executable files are located. These files are available to all users
3	**/dev** These are device drivers
4	**/etc** Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages
5	**/lib** Contains shared library files and sometimes other kernel-related files
6	**/boot** Contains files for booting the system
7	**/home** Contains the home directory for users and other accounts
8	**/mnt** Used to mount other temporary file systems, such as **cdrom** and **floppy** for the **CD-ROM** drive and **floppy diskette drive**, respectively
9	**/proc** Contains all processes marked as a file by **process number** or other information that is dynamic to the system
10	**/tmp** Holds temporary files used between system boots

11	**/usr** Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
12	**/var** Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
13	**/sbin** Contains binary (executable) files, usually for system administration. For example, ***fdisk*** and ***ifconfig*** utlities
14	**/kernel** Contains kernel files

Navigating the File System

Now that you understand the basics of the file system, you can begin navigating to the files you need. The following commands are used to navigate the system −

Sr.No.	Command & Description
1	**cat filename** Displays a filename
2	**cd dirname** Moves you to the identified directory
3	**cp file1 file2** Copies one file/directory to the specified location

4	**file filename** Identifies the file type (binary, text, etc)
5	**find filename dir** Finds a file/directory
6	**head filename** Shows the beginning of a file
7	**less filename** Browses through a file from the end or the beginning
8	**ls dirname** Shows the contents of the directory specified
9	**mkdir dirname** Creates the specified directory
10	**more filename** Browses through a file from the beginning to the end
11	**mv file1 file2** Moves the location of, or renames a file/directory
12	**pwd** Shows the current directory the user is in
13	**rm filename**

	Removes a file
14	**rmdir dirname** Removes a directory
15	**tail filename** Shows the end of a file
16	**touch filename** Creates a blank file or modifies an existing file or its attributes
17	**whereis filename** Shows the location of a file
18	**which filename** Shows the location of a file if it is in your PATH

Printing amd searching file using grep

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).
Syntax:
grep [options] pattern [files]
Options Description
-c : This prints only a count of the lines that match a pattern
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching
-l : Displays list of a filenames only.
-n : Display the matched lines and their line numbers.
-v : This prints out all the lines that do not matches the pattern
-e exp : Specifies expression with this option. Can use multiple times.
-f file : Takes patterns from file, one per line.
-E : Treats pattern as an extended regular expression (ERE)
-w : Match whole word

```
-o : Print only the matched parts of a matching line,
  with each such part on a separate output line.
```

Sample Commands
Consider the below file as an input.

```
$cat > geekfile.txt
unix is great os. unix is opensource. unix is free os.

learn operating system.

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

1. Case insensitive search : The -i option enables to search for a string case insensitively in the give file. It matches the words like “UNIX”, “Unix”, “unix”.

```
$grep -i "UNix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

```
$grep -c "unix" geekfile.txt
```

Output:

```
2
```

3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

```
$grep -l "unix" *

or

$grep -l "unix" f1.txt f2.txt f3.xt f4.txt
```

Output:

```
geekfile.txt
```

4. Checking for the whole words in a file : By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

```
$ grep -w "unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
$ grep -o "unix" geekfile.txt
```

Output:

```
unix

unix

unix

unix

unix

unix
```

6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

```
$ grep -n "unix" geekfile.txt
```

Output:

```
1:unix is great os. unix is opensource. unix is free os.

4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

7. Inverting the pattern match : You can display the lines that are not matched with the specified search sting pattern using the -v option.

```
$ grep -v "unix" geekfile.txt
```

Output:

```
learn operating system.

Unix linux which one you choose.
```

8. Matching the lines that start with a string : The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
$ grep "^unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

9. Matching the lines that end with a string : The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
$ grep "os$" geekfile.txt
```

10.Specifies expression with -e option. Can use multiple times :

```
$grep –e "Agarwal" –e "Aggarwal" –e "Agrawal" geekfile.txt
```

11. -f file option Takes patterns from file, one per line.

```
$cat pattern.txt

Agarwal
Aggarwal
Agrawal
$grep –f pattern.txt    geekfile.txt
```

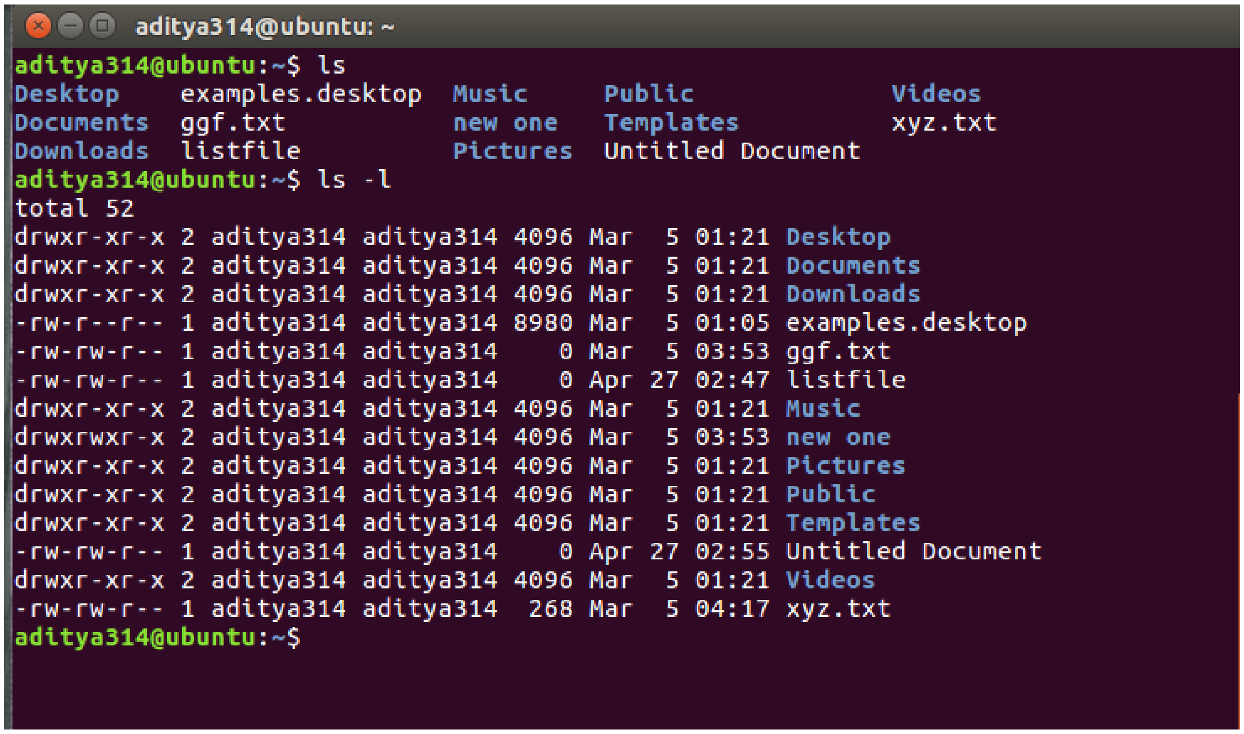
Permission to Set files and Change Owner of Files

Linux is a multi-user operating system, so it has security to prevent people from accessing each other's confidential files.

Introduction

When you execute an "ls" command, you are not given any information about the security of the files, because by default "ls" only lists the names of files. You can get more information by using an "option" with the "ls" command. All options start with a '-'. For example, to execute "ls" with the "long listing" option, you would type ls -l

When you do so, each file will be listed on a separate line in long format. There is an example in the window below.



```
aditya314@ubuntu:~$ ls
Desktop    examples.desktop  Music     Public            Videos
Documents  ggf.txt           new one   Templates         xyz.txt
Downloads  listfile          Pictures  Untitled Document
aditya314@ubuntu:~$ ls -l
total 52
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Desktop
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Documents
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Downloads
-rw-r--r-- 1 aditya314 aditya314 8980 Mar  5 01:05 examples.desktop
-rw-rw-r-- 1 aditya314 aditya314    0 Mar  5 03:53 ggf.txt
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:47 listfile
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Music
drwxrwxr-x 2 aditya314 aditya314 4096 Mar  5 03:53 new one
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Pictures
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Public
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Templates
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:55 Untitled Document
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Videos
-rw-rw-r-- 1 aditya314 aditya314  268 Mar  5 04:17 xyz.txt
aditya314@ubuntu:~$
```

There's a lot of information in those lines.

1. The first character will almost always be either a '-', which means it's a file, or a 'd', which means it's a directory.
2. The next nine characters (rw-r–r–) show the security; we'll talk about them later.
3. The next column shows the owner of the file. In this case it is me, my userID is "aditya314".
4. The next column shows the group owner of the file. In my case I want to give the "aditya314" group of people special access to these files.
5. The next column shows the size of the file in bytes.
6. The next column shows the date and time the file was last modified.
7. And, of course, the final column gives the filename.

Deciphering the security characters will take a bit more work.

Understanding the security permissions

First, you must think of those nine characters as three sets of three characters (see the box at the bottom). Each of the three "rwx" characters refers to a different operation you can perform on the file.

```
---      ---      ---

rwx      rwx      rwx

user     group    other
```

Read, write, execute and –

The 'r' means you can "read" the file's contents.
The 'w' means you can "write", or modify, the file's contents.
The 'x' means you can "execute" the file. This permission is given only if the file is a program.
If any of the "rwx" characters is replaced by a '-', then that permission has been revoked.

User, group and others

user – The user permissions apply only the owner of the file or directory, they will not impact the actions of other users.
group – The group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
others – The others permissions apply to all other users on the system, this is the permission group that you want to watch the most.

For example, consider that the user's permissions for some files is "rw-" as the first three characters. This means that the owner of the file ("aditya314", i.e. me) can "read" it (look at its contents) and "write" it (modify its contents). I cannot execute it because it is not a program; it is a text file.

If "r-x" is the second set of 3 characters it means that the members of the group "aditya314" can only read and execute the files.

The final three characters show the permissions allowed to anyone who has a UserID on this Linux system. Let us say we have the permission ("r–"). This means anyone in our Linux world can read, but they cannot modify the contents of the files or execute it.

Changing security permissions

The command you use to change the security permissions on files is called “chmod”, which stands for “change mode”, because the nine security characters are collectively called the security “mode” of the file.

1. The first argument you give to the “chmod” command is ‘u’, ‘g’, ‘o’. We use:
   u for user
   g for group
   o for others,
   you can also use a combination of them (u,g,o).
   This specifies which of the three groups you want to modify.
2. After this use
   a ‘+’ for adding
   a ‘-‘ for removing
   and a “=” for assigning a permission.
3. Then specify the permission r,w or x you want to change.
   Here also you can use a combination of r,w,x.
   This specifies which of the three permissions “rwx” you want to modify
4. use can use commas to modify more permissions
5. Finally, the name of the file whose permission you are changing

An example will make this clearer.
For example, if you want to give “execute” permission to the world (“other”) for file “xyz.txt”, you would start by typing

```
chmod o
```

Now you would type a ‘+’ to say that you are “adding” a permission.

```
chmod o+
```

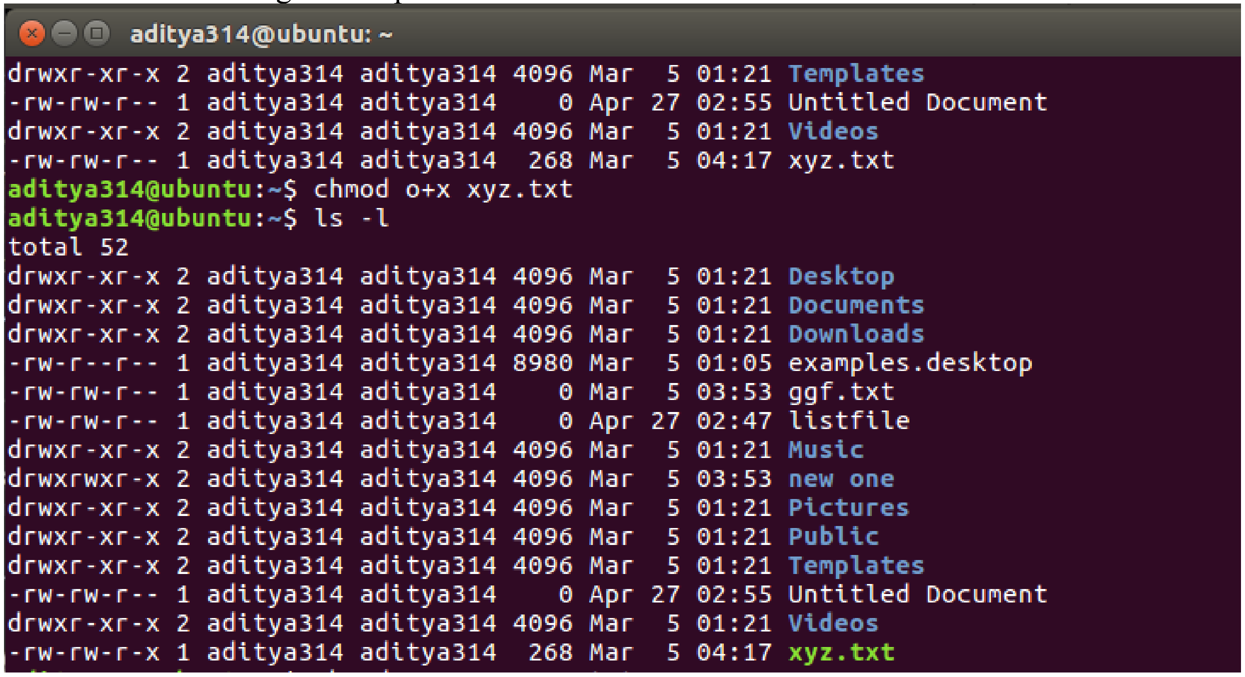
Then you would type an ‘x’ to say that you are adding “execute” permission.

```
chmod o+x
```

Finally, specify which file you are changing.

```
chmod o+x xyz.txt
```

You can see the change in the picture below.

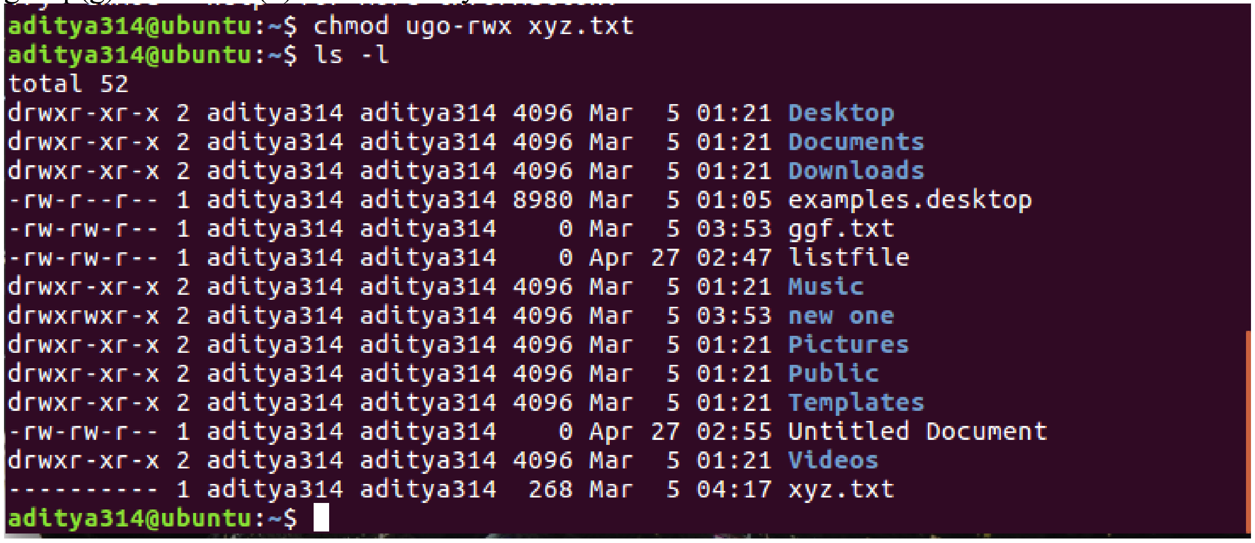


```
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Templates
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:55 Untitled Document
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Videos
-rw-rw-r-- 1 aditya314 aditya314  268 Mar  5 04:17 xyz.txt
aditya314@ubuntu:~$ chmod o+x xyz.txt
aditya314@ubuntu:~$ ls -l
total 52
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Desktop
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Documents
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Downloads
-rw-r--r-- 1 aditya314 aditya314 8980 Mar  5 01:05 examples.desktop
-rw-rw-r-- 1 aditya314 aditya314    0 Mar  5 03:53 ggf.txt
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:47 listfile
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Music
drwxrwxr-x 2 aditya314 aditya314 4096 Mar  5 03:53 new one
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Pictures
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Public
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Templates
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:55 Untitled Document
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Videos
-rw-rw-r-x 1 aditya314 aditya314  268 Mar  5 04:17 xyz.txt
```

You can also change multiple permissions at once. For example, if you want to take all permissions away from everyone, you would type

```
chmod ugo-rwx xyz.txt
```

The code above revokes all the read(r), write(w) and execute(x) permission from all user(u), group(g) and others(o) for the file xyz.txt which results to this.



```
aditya314@ubuntu:~$ chmod ugo-rwx xyz.txt
aditya314@ubuntu:~$ ls -l
total 52
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Desktop
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Documents
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Downloads
-rw-r--r-- 1 aditya314 aditya314 8980 Mar  5 01:05 examples.desktop
-rw-rw-r-- 1 aditya314 aditya314    0 Mar  5 03:53 ggf.txt
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:47 listfile
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Music
drwxrwxr-x 2 aditya314 aditya314 4096 Mar  5 03:53 new one
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Pictures
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Public
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Templates
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:55 Untitled Document
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Videos
---------- 1 aditya314 aditya314  268 Mar  5 04:17 xyz.txt
aditya314@ubuntu:~$
```

Another example can be this:

```
chmod ug+rw,o-x abc.mp4
```

The code above adds read(r) and write(w) permission to both user(u) and group(g) and revoke execute(x) permission from others(o) for the file abc.mp4.

Something like this:

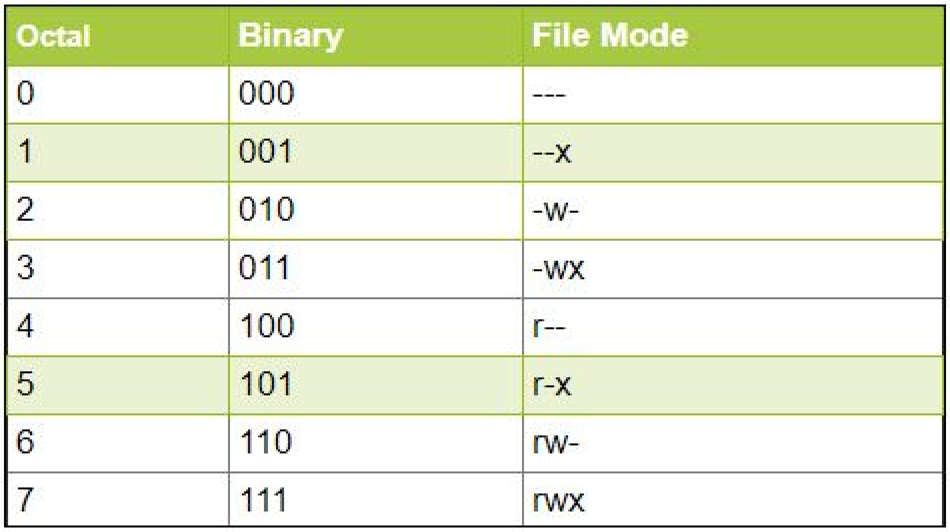
```
chmod ug=rx,o+r abc.c
```

assigns read(r) and execute(x) permission to both user(u) and group(g) and add read permission to others for the file abc.c.

There can be numerous combinations of file permissions you can invoke, revoke and assign. You can try some in your linux system.

The octal notations

You can also use octal notations like this.



Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Using the octal notations table instead of ‘r’, ‘w’ and ‘x’. Each digit octal notiation can be used of either of the group ‘u’,’g’,’o’.

So, the following work the same.

```
chmod ugo+rwx [file_name]

chmod 777 [file_name]
```

Both of them provides full read write and execute permission (code=7) to all the group.

Same is the case with this..

```
chmod u=r,g=wx,o=rx [file_name]

chmod 435 [file_name]
```

Both the codes give read (code=4) permission to user, write and execute (code=3) for group and read and execute (code=5) for others.

And even this…

```
chmod 775 [file_name]
chmod ug+rwx,o=rx [file_name]
```

Both the commands give all permissions (code=7) to user and group, read and execute (code=5) for others.

Process state command in linux

Linux is a multitasking and multi-user systems. So, it allows multiple processes to operate simultaneously without interfering with each other. Process is one of the important fundamental concept of the Linux OS. A process is an executing instance of a program and carry out different tasks within the operating system.

Linux provides us a utility called **ps** for viewing information related with the processes on a system which stands as abbreviation for **"Process Status".** ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in **/proc** file-system. /proc contains virtual files, this is the reason it's referred as a virtual file system.
ps provides numerous options for manipulating the output according to our need.

Syntax –

```
ps [options]
```

Options for ps Command :

1. **Simple process selection :** Shows the processes for the current shell –

```
2.    [root@rhel7 ~]# ps
3.       PID TTY          TIME CMD
4.    12330 pts/0    00:00:00 bash
5.    21621 pts/0    00:00:00 ps
```

Result contains four columns of information.
Where,
PID – the unique process ID
TTY – terminal type that the user is logged into
TIME – amount of CPU in minutes and seconds that the process has been running
CMD – name of the command that launched the process.
Note – Sometimes when we execute **ps** command, it shows TIME as 00:00:00. It is nothing but the total accumulated CPU utilization time for any process and 00:00:00 indicates no CPU time has been given by the kernel till now. In above example we found that, for bash no CPU time has been given. This is because bash is just a parent process for different processes which needs bash for their execution and bash itself is not utilizing any CPU time till now.

6. **View Processes :** View all the running processes use either of the following option with ps –

```
7.    [root@rhel7 ~]# ps -A
8.    [root@rhel7 ~]# ps -e
```

9. **View Processes not associated with a terminal :** View all processes except both session leaders and processes not associated with a terminal.

```
10.    [root@rhel7 ~]# ps -a
11.      PID TTY          TIME CMD
12.    27011 pts/0    00:00:00 man
13.    27016 pts/0    00:00:00 less
14.    27499 pts/1    00:00:00 ps
```

Note – You may be thinking that what is session leader? A unique session is assing to evry process group. So, session leader is a process which kicks off other processes. The process ID of first process of any session is similar as the session ID.

15. **View all the processes except session leaders :**

```
16.    [root@rhel7 ~]# ps -d
```

17. **View all processes except those that fulfill the specified conditions (negates the selection) :**
Example – If you want to see only session leader and processes not associated with a terminal. Then, run

```
18.    [root@rhel7 ~]# ps -a -N
19.    OR
20.    [root@rhel7 ~]# ps -a --deselect
```

21. **View all processes associated with this terminal :**

```
22.    [root@rhel7 ~]# ps -T
```

23. **View all the running processes :**

```
24.    [root@rhel7 ~]# ps -r
```

25. **View all processes owned by you :** Processes i.e same EUID as ps which means runner of the ps command, root in this case –

```
26.    [root@rhel7 ~]# ps -x
```

Process selection by list
Here we will discuss how to get the specific processes list with the help of ps command. These options accept a single argument in the form of a blank-separated or comma-separated list. They can be used multiple times.
For example: ps -p "1 2" -p 3,4

1. Select the process by the command name. This selects the processes whose executable name is given in cmdlist. There may be a chance you won't know the process ID and with this command it is easier to search.
Syntax : ps -C command_name

```
2.    Syntax :
3.    ps -C command_name
4.
```

```
Example :
[root@rhel7 ~]# ps -C dhclient
  PID TTY          TIME CMD
19805 ?        00:00:00 dhclient
```

Select by group ID or name. The group ID identifies the group of the user who created the process.

```
Syntax :
ps -G group_name
ps --Group group_name

Example :
[root@rhel7 ~]# ps -G root
```

View by group id :

```
Syntax :
ps -g group_id
ps -group group_id

Example :
[root@rhel7 ~]# ps -g 1
  PID TTY          TIME CMD
    1 ?        00:00:13 systemd
```

View process by process ID.

```
Syntax :
ps p process_id
ps -p process_id
ps --pid process_id

Example :
[root@rhel7 ~]#   ps p 27223
  PID TTY      STAT   TIME COMMAND
27223 ?        Ss     0:01 sshd: root@pts/2

```

```
[root@rhel7 ~]#    ps -p 27223
   PID TTY                  TIME CMD
27223 ?               00:00:01 sshd

[root@rhel7 ~]#    ps --pid 27223
   PID TTY                  TIME CMD
27223 ?               00:00:01 sshd
```

You can view multiple processes by specifying multiple process IDs separated by blank or comma –

Example :

```
[root@rhel7 ~]#    ps -p 1 904 27223
   PID TTY           STAT      TIME COMMAND
      1 ?               Ss         0:13 /usr/lib/systemd/systemd --switched-root --system --d
   904 tty1          Ssl+      1:02 /usr/bin/X -core -noreset :0 -seat seat0 -auth /var/r
27223 ?               Ss         0:01 sshd: root@pts/2
```

Here, we mentioned three process IDs – 1, 904 and 27223 which are separated by blank.

43. Select by parent process ID. By using this command we can view all the processes owned by parent process except the parent process.

```
[root@rhel7 ~]# ps -p 766
   PID TTY                  TIME CMD
   766 ?               00:00:06 NetworkManager

[root@rhel7 ~]# ps --ppid 766
   PID TTY                  TIME CMD
19805 ?               00:00:00 dhclient
```

In above example process ID **766** is assigned to NetworkManager and this is the parent process for dhclient with process ID 19805.

51. View all the processes belongs to any session ID.

```
Syntax :
ps -s session_id
ps --sid session_id

Example :
```

```
[root@rhel7 ~]# ps -s 1248
    PID TTY          TIME CMD
   1248 ?        00:00:00 dbus-daemon
   1276 ?        00:00:00 dconf-service
   1302 ?        00:00:00 gvfsd
   1310 ?        00:00:00 gvfsd-fuse
   1369 ?        00:00:00 gvfs-udisks2-vo
   1400 ?        00:00:00 gvfsd-trash
   1418 ?        00:00:00 gvfs-mtp-volume
   1432 ?        00:00:00 gvfs-gphoto2-vo
   1437 ?        00:00:00 gvfs-afc-volume
   1447 ?        00:00:00 wnck-applet
   1453 ?        00:00:00 notification-ar
   1454 ?        00:00:02 clock-applet
```

Select by tty. This selects the processes associated with the mentioned tty :

```
Syntax :
ps t tty
ps -t tty
ps --tty tty

Example :
[root@rhel7 ~]# ps -t pts/0
    PID TTY          TIME CMD
  31199 pts/0    00:00:00 bash
  31275 pts/0    00:00:00 man
  31280 pts/0    00:00:00 less
```

Select by effective user ID or name.

Syntax :

```
ps U user_name/ID
ps -U user_name/ID
ps -u user_name/ID
ps –User user_name/ID
ps –user user_name/ID
```

Output Format control

These options are used to choose the information displayed by ps. There are multiple options to control output format. These option can be combined with any other options like **e, u, p, G, g** etc, depends on our need.

1. Use **-f** to view full-format listing.

```
2.      [tux@rhel7 ~]$ ps -af
3.      tux         17327 17326    0 12:42 pts/0        00:00:00 -bash
4.      tux         17918 17327    0 12:50 pts/0        00:00:00 ps -af
```

5. Use **-F** to view Extra full format.

```
6.      [tux@rhel7 ~]$ ps -F
7.      UID              PID    PPID    C       SZ      RSS PSR STIME TTY                  TIME
    CMD
8.      tux         17327 17326    0 28848    2040     0 12:42 pts/0        00:00:00 -bash
9.      tux         17942 17327    0 37766    1784     0 12:50 pts/0        00:00:00 ps -F
```

10. To view process according to user-defined format.

```
11.     Syntax :
12.     [root@rhel7 ~]#    ps --formate column_name
13.     [root@rhel7 ~]#    ps -o column_name
14.     [root@rhel7 ~]#    ps o column_name
15.
16.     Example :
17.     [root@rhel7 ~]#    ps -aN --format cmd,pid,user,ppid
18.     CMD                                      PID USER          PPID
19.     /usr/lib/systemd/systemd --         1 root               0
20.     [kthreadd]                                2 root               0
21.     [ksoftirqd/0]                             3 root               2
22.     [kworker/0:0H]                            5 root               2
23.     [migration/0]                             7 root               2
24.     [rcu_bh]                                  8 root               2
25.     [rcu_sched]                               9 root               2
26.     [watchdog/0]                             10 root               2
```

In this example I wish to see command, process ID, username and parent process ID, so I pass the arguments cmd, pid, user and ppid respectively.

27. View in BSD job control format :

```
28.     [root@rhel7 ~]# ps -j
29.       PID  PGID   SID TTY          TIME CMD
30.     16373 16373 16373 pts/0    00:00:00 bash
31.     19734 19734 16373 pts/0    00:00:00 ps
```

32. Display BSD long format :

```
33.     [root@rhel7 ~]# ps l
34.     F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY
    TIME COMMAND
35.     4     0   904   826  20   0 306560 51456 ep_pol Ssl+ tty1       1:32
    /usr/bin/X -core -noreset :0 -seat seat0 -auth /var/run/lightdm/root/:0 -noli
36.     4     0 11692 11680  20   0 115524  2132 do_wai Ss   pts/2      0:00 -bash
```

37. Add a column of security data.

```
38.     [root@rhel7 ~]# ps -aM
39.     LABEL                                                PID TTY
    TIME     CMD
40.     unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 19534 pts/2 00:00:00 man
41.     unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 19543 pts/2 00:00:00 less
42.     unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 20469 pts/0 00:00:00 ps
```

43. View command with signal format.

```
44.     [root@rhel7 ~]# ps s 766
```

45. Display user-oriented format

```
46.     [root@rhel7 ~]# ps u 1
47.     USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START
    TIME COMMAND
48.     root         1  0.0  0.6 128168  6844 ?        Ss   Apr08   0:16
    /usr/lib/systemd/systemd --switched-root --system --deserialize 21
```

49. Display virtual memory format

```
50.     [root@rhel7 ~]# ps v 1
51.       PID TTY      STAT   TIME  MAJFL   TRS   DRS   RSS %MEM
    COMMAND
52.         1 ?        Ss     0:16     62  1317 126850 6844  0.6
    /usr/lib/systemd/systemd --switched-root --system --deserialize 21
```

53. If you want to see environment of any command. Then use option **e** –

```
54.     [root@rhel7 ~]# ps ev 766
```

55.
```
   PID TTY      STAT   TIME  MAJFL   TRS   DRS   RSS %MEM COMMAND
```
56.
```
   766 ?        Ssl    0:08     47  2441 545694 10448  1.0 /usr/sbin/NetworkManager --no-daemon LANG=en_US.UTF-8 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
```
57. View processes using highest memory.
58.
```
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem
```

12 – print a process tree

```
[root@rhel7 ~]# ps --forest -C sshd

  PID TTY          TIME CMD

  797 ?        00:00:00 sshd

11680 ?        00:00:03  \_ sshd

16361 ?        00:00:02  \_ sshd
```

59. List all threads for a particular process. Use either the **-T or -L** option to display threads of a process.

60.
```
[root@rhel7 ~]# ps -C sshd -L
```
61.
```
  PID   LWP TTY          TIME CMD
```
62.
```
  797   797 ?        00:00:00 sshd
```
63.
```
11680 11680 ?        00:00:03 sshd
```

```
16361 16361 ?        00:00:02 sshd
```