Unit-5 Linux Basics

❖ Linux Introduction

Linux was first developed by Linus torvads .
Linux is the most famous free and open source OS.
Open source Os are available in source code format rather than as compiled binary code
Free software means not by money point of view. But software that is free means it is distributed along with its source code.
Anyone who receives it is free to make change and redistribute it.
There are many versions of Linux.famous versions are,
Red hat Linux → Fedora ,Centos
Debian → Ubuntu,Linux mint,Linspire,PCLinux
Linux can be found on wide range of devices-from personal computers,mobiles,tablets and embedded systems to main frame computers and supercomputers

❖ Basic architecture of Unix/ Linux

- It is also known as the layered structure of Linux.
- Linux is a UNIX-like OS, its architecture resembles to that of UNIX.

1. Hardware:

- Bottom layer is hard ware.
- It consist of physical devices such as CPU ,memory ,disks ,monitors ,printers etc.
- These devices provides various services.

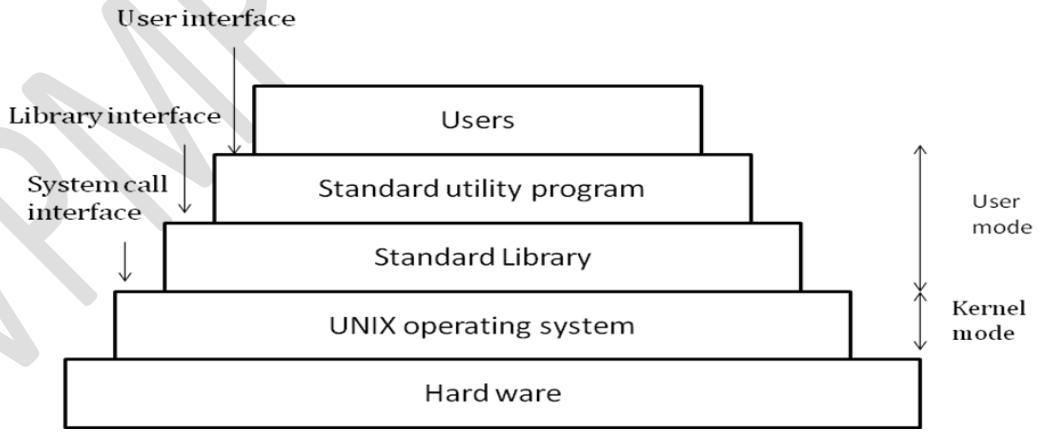


Figure : Linux architecture

2. Linux operating system:

- Next higher layer in Linux OS
- It is called system kernel.
- Kernel manages all the underlying hardware
- It directly interacts with the H/W and provides user programs required services
- It hides complex details of H/W
- It provides interface between user program and H/W
- Main service of OS :
    - process management , memory management ,file system management ,I/O management

3. Standard library

- Contains a set of procedures, one procedure per system call
- These procedures are written in assembly language and used to invoke various system calls from user program.

4. Standard utility program

- All UNIX versions supply OS , system call , large no. of standard program
- Program include command processor (shell) ,compilers ,editors ,text processing program, file manipulation utility a various commands…
- Program makes user tasks simpler
- User interacts with program, program interacts with OS to get services from OS

5. users

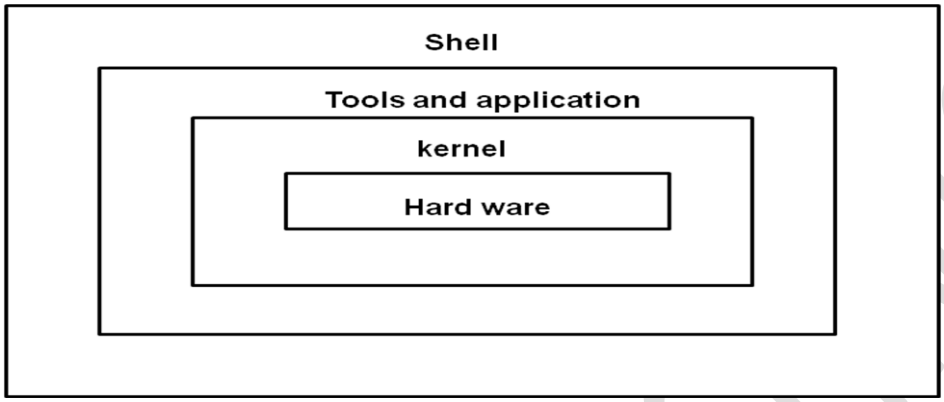
- Top most layers
- User program come in this layer
- They interacts with system either by library procedure to invoke system call or by using utility program such as shell

❖ Introduction to shell and commands

Kernel

- Kernel is core part of any OS.
- Kernel is program, which is loaded in memory ,when system is turned on and provides various services until system is turned off.
- Kernel manages all the underlying hardware
- It directly interacts with the H/W , It hides complex details of H/W
- When user needs to use any H/W ,it has to use services (system call) provided by kernel.
- System call request kernel and kernel performs the job behalf on user process.

- Main service of OS :
  - process management , memory management ,file system management ,I/O management
- Only one kernel in the system



The Shell (command interpreter)

- A **Unix shell**, also called "the command line "
- It is interface between user programs and the kernel.
- Users direct the operation of the computer entering command input as text for a shell to execute.
- When user logs in to system processor shell starts execution
- It terminates when user logs out from system
- Shell works as command interpreter .it accepts command from user and translates them into which the kernel can understand
- More than one shell can be in system

❖ Commands: pwd, cd, mkdir, rmdir, ls, cat, cp, rm, mv, wc, split, cmp, comm, diff, head, tail, grep, sort

1. pwd Command

The pwd command is used to display the location of the current working directory.

Syntax: pwd

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ pwd
/home/javatpoint
```

2. mkdir Command

The mkdir command is used to create a new directory under any directory.

Syntax: mkdir <**directory** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ mkdir new_directory
javatpoint@javatpoint-Inspiron-3542:~$
```

3. rmdir Command

The rmdir command is used to delete a directory.

Syntax: rmdir <**directory** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ rmdir new_directory
javatpoint@javatpoint-Inspiron-3542:~$
```

4. ls Command

The ls command is used to display a list of content of a directory.

Syntax: ls

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ ls
a               Desktop             examples.desktop  Music           sample
Akash           Directory           hello.c           pico            snap
a.out           Documents           hello.i           Pictures        Templates
composer.phar   Downloads           hello.o           project         Test.txt
Demo.sh         eclipse             hello.s           Public          Videos
Demo.txt        eclipse-installer   index.html        Python
Demo.txt~       eclipse-workspace   mail              Python-3.8.0
```

5. cd Command

The cd command is used to change the current directory.

Syntax: cd <**directory** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ cd Desktop
javatpoint@javatpoint-Inspiron-3542:~/Desktop$
```

6. cat Command

The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

Syntax: cat [OPTION]... [FILE]..

To create a file, execute it as follows:

cat > <**file** name>
// Enter file content

Press "**CTRL+ D**" keys to save the file. To display the content of the file, execute it as follows:

cat <**file** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~/Newfolder$ cat > Demo.txt
This is a text file.
javatpoint@javatpoint-Inspiron-3542:~/Newfolder$ cat Demo.txt
This is a text file.
```

7. rm Command

The rm command is used to remove a file.

Syntax: rm <file name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~/Newfolder$ rm Demo.txt
javatpoint@javatpoint-Inspiron-3542:~/Newfolder$ rm Demo1.txt Demo2.txt
```

8. cp Command

The cp command is used to copy a file or directory.

Syntax: To copy in the same directory:

cp <**existing** file name> <**new** file name>

To copy in a different directory:

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ cp demo.txt demo1.txt
javatpoint@javatpoint-Inspiron-3542:~$ cp demo.txt Documents
```

9. mv Command

The mv command is used to move a file or a directory form one location to another location.

Syntax: mv <**file** name> <**directory** path>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ mv demo.txt Directory
```

10. head Command

The head command is used to display the content of a file. It displays the first 10 lines of a file.

Syntax: head <**file** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ head Demo.txt
1
2
3
4
5
6
7
8
9
10
```

11. tail Command

The tail command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.

Syntax: tail <**file** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ tail Demo.txt
2
3
4
5
6
7
8
9
10
11
```

12.grep Command

The grep is the most powerful and used filter in a Linux system. The 'grep' stands for "**global regular expression print**." It is useful for searching the content from a file. Generally, it is used with the pipe.

Syntax: command | grep **<searchWord>**

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ cat marks.txt | grep 9
celena-90
```

13. comm Command

The 'comm' command is used to compare two files or streams. By default, it displays three columns, first displays non-matching items of the first file, second indicates the non-matching item of the second file, and the third column displays the matching items of both files.

Syntax: comm **<file1>** **<file2>**

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ comm Demo.txt Demo1.txt
                1
2
                3
comm: file 2 is not in sorted order
        11
                4
                5
        22
        33
6
7
8
9
comm: file 1 is not in sorted order
10
11
```

13. tr Command

The tr command is used to translate the file content like from lower case to upper case.

Syntax: command | tr <'old'> <'new'>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ cat marks.txt | tr 'prcu' 'PRCU'
alex-50
alen-70
jon-75
CaRRy-85
Celena-90
jUstin-80
```

14. wc Command

The wc command is used to count the lines, words, and characters in a file.

Syntax: wc <**file** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ wc marks.txt
 6  6 52 marks.txt
```

15. sort Command

The sort command is used to sort files in alphabetical order.

Syntax: sort <**file** name>

Output:

```
javatpoint@javatpoint-Inspiron-3542:~$ sort marks.txt
alen-70
alex-50
carry-85
celena-90
jon-75
justin-80
```

- **Editing files with "vi", "vim", "gedit", "gcc"**

The vi editor is elaborated as **vi**sual editor. It is installed in every Unix system. In other words, it is available in all Linux distros. It is user-friendly and works same on different distros and platforms. It is a very powerful application. An improved version of vi editor is **vim**.

The vi editor has two modes:

- **Command Mode:** In command mode, actions are taken on the file. The vi editor starts in command mode. Here, the typed words will act as commands in vi editor. To pass a command, you need to be in command mode.
- **Insert Mode:** In insert mode, entered text will be inserted into the file. The **Esc** key will take you to the command mode from insert mode.

By default, the vi editor starts in command mode. To enter text, you have to be in insert mode, just type **'i'** and you'll be in insert mode. Although, after typing **i** nothing will appear on the screen but you'll be in insert mode. Now you can type anything.

To exit from insert mode press **Esc** key, you'll be directed to command mode.

Commands	Action
:wq	Save and quit
:w	Save
:q	Quit
:w fname	Save as fname
ZZ	Save and quit
:q!	Quit discarding changes made
:w!	Save (and write to non-writable file)

To exit from vi, first ensure that you are in command mode. Now, type :wq and press enter. It will save and quit vi.

❖ Read Command-line Arguments in Shell Scripts

Now we have developed an understanding of the command-line arguments in Linux. Now it’s time to use this knowledge for practical application of the netstat command.

For this tutorial, we will go over an example to learn how to use the command-line arguments in your shell script.

First, we will create a shell script to demonstrate the working of all the reserved variables which we discussed in the previous section. Use nano or any preferred editor of your choice and copy the following.

This is the shell script which we plan to use for this purpose.

```
#!/bin/sh
echo "Script Name: $0"
echo "First Parameter of the script is $1"
echo "The second Parameter is $2"
echo "The complete list of arguments is $@"
echo "Total Number of Parameters: $#"
echo "The process ID is $$"
echo "Exit code for the script: $?"
```

Copy

Once we are done, we will save the script as PositionalParameters.sh and exit our text editor.

Now, we will open the command line on our system and run the shell script with the following arguments.

```
./PositionalParameters.sh learning command line arguments
```

Copy

The script will run with our specified arguments and use positional parameters to generate an output. If you followed the step correctly, you should see the following screen.

```
~/Desktop$ sudo sh ./PositionalParameters.sh learning command line arguments
Script Name: ./PositionalParameters.sh
First Parameter of the script is learning
The second Parameter is command
The complete list of arguments is learning command line arguments
Total Number of Parameters: 4
The process ID is 14974
Exit code for the script: 0
```

Reading Arguments

Our output shows the correct output by substituting the reserved variables with the appropriate argument when we called it.

The process was run under the process ID 14974 and quit with the exit code 0.

❖ The logical operators

1. **&&** :command 2 is executed only when command1 is successful in execution.
   Syntax :
   Command1 **&&** commnad2
   Example : if [$a –gt $b && $a -gt $c]
   echo " a is maximum"
2. **||** **:**command 2 is executed only when command1 is fails in its execution.command2 will not be executed at all.

Syntax : Command1 || command2
if [ans –eq 'y' || ans –eq 'Y']
echo " continue"

❖ Evaluate expression using test and [..]

- **test** and **[…..]** is used to evaluate conditional expressions with integers ,strings and file attributes.
  test expression
  [expression]
- **Integer related test operators:**

Expression	Result
exp1 **-eq** exp2	It returns true if exp1 is equal to exp2
exp1 **-ne** exp2	It returns true if exp1 is not equal to exp2
exp1 **-le** exp2	It returns true if exp1 is less than or equal to exp2
exp1 **-lt** exp2	It returns true if exp1 is less than to exp2
exp1 **-ge** exp2	It returns true if exp1 is greater or equal to exp2
exp1 **-gt** exp2	It returns true if exp1 is greater than exp2

- **File related test operators**

Expression	Result
-b file	True if file exists and is a block special file
-c file	True if file exists and is a character special file

-d file	True if file exists and is a directory
-f file	True if file exists and is a regular file
-r file	True if file exists and is a readable
-w file	True if file exists and is a writable
-x file	True if file exists and is a executable
-k file	True if file exists and is a sticky bit is set
-l file	True if file exists and is a symbolic link
-s file	True if file exists and its size is greater than zero
f1 -nt f2	True if f1 is newer than f2
f1 -ot f2	True if f1 is older than f2
f1 -ef f2	True if f1 is linked to f2

- **String related test expression**

Expression	**Result**
-n	True if length of string is not zero
-z	True if length of string is zero
String	True if string is not set to null
string1 = string2	True if string1 is equal to string2
string1 != string2	True if sting1 is not equal to sting2
string = pattern	True if string matches pattern
String != pattern	True if string does not match pattern

❖ Branching : if, case

1. if

Syntax: **if** command1
then
commands
fi

2. if ….else

```
Syntax :        if command1
                then
                        command2
                else
                        command3
                fi
```

3. else …if ladder :

```
Syntax :
                if command-1
                then
                        commands
                elif command-2
                then
                        commands
                elif command-3
                then
                        commands
                else
                        commands
                fi
```

4. case

```
        case expression in
        pattern1)   commands1    ;;
        pattern2)   commands2    ;;
        pattern3)   commands3    ;;
        …..
        …..
        esac
```

❖ Basic of Looping : while, for

1. while

```
Syntax :        while command1
                do
                        commands
                done
```

2. until :

```
Syntax :        until command1
```

do

commands

done

3. for :

Syntax :

for i **in** list

do

commands

done