Chapter 4

Basic Linux Commands

Linux commands are text-based instructions given to a Linux operating system to perform specific tasks or operations. They are executed on the command line interface (CLI) or terminal.

Linux commands are typically composed of a command name and options, followed by arguments. The command name is the actual instruction that the user wants the operating system to execute, and options provide additional functionality or modify the behavior of the command. Arguments are the parameters passed to the command that specify the targets or data on which the command should operate.

1. Basic structure of linux command's

The command line interface (CLI) in Linux is a text-based interface that allows users to interact with the operating system by typing commands on a terminal emulator. The structure of a command in Linux follows a general pattern that consists of the command name, followed by options and arguments, separated by spaces. Here's an example of a typical command in Linux:

```
command_name [option(s)] [argument(s)]
```

Let's break down each element of the command structure:

- command_name: This is the name of the command that the user wants to execute.
- [option(s)]: These are optional flags or switches that modify the behavior of the command. Options are preceded by a hyphen (-) or two hyphens (--), and they can be combined together. Some options may require additional arguments.
- [argument(s)]: These are the parameters that the command needs to execute. Arguments can be files, directories, device names, IP addresses, or any other data that the command requires.

Six Basic Rules of Linux Command Notation

1. Any text standing by itself, and not within [], <>, or {}, must be typed exactly as shown.

2. Any text within square brackets ([]) is optional. You can type it or not type it. For instance, the syntax ls [-l] means you must type ls (per the first rule), while adding -l is optional but not necessary. Do not type the square brackets themselves! In our example, type ls or ls -l. Do not type ls [-l] .

3. Angle brackets (<>) and the text within them must be replaced by appropriate text (usually a name or value). The text within the brackets usually indicates the nature of the replacement.

For instance, the syntax more <filename> means that you should replace <filename> with the name of the file you want to examine using more. If you want to look at the file output, type more output. Remember, do not use the angle brackets when you actually type the command!

4. Curly braces ({}) indicate that you must choose one of the values given within the braces. The values are separated by | (which in this case means or, not pipe!). For example, the syntax command -{a|b} means you must enter either command -a or command -b.

5. An ellipsis (...) means "and so on." They are normally used with parameters such as filenames, which is described later.

6. The sixth basic rule states that the brackets can be combined as necessary. For instance, you don't have to type a filename with the more command which would be indicated as more [<filename>]. The outer set of square brackets makes the entire parameter optional. If you do decide to use the parameter, replace the inner set of angle brackets with the appropriate value. Because the more command enables one or more filenames to be specified, the syntax becomes more [<filename> ...]. The ellipsis means you can have as many <filenames> as you want, such as more output1 output2 output3.

2. Terminal

Before discussing the commands, first we need to know what is terminal in Linux system? Terminal is console window through which user can give the input and after processing get output. There are number of commands to perform different task with specific syntax. The Linux terminal refers to a command-line interface (CLI) provided by the Linux operating system. It allows users to interact with the system by typing commands rather than using a graphical user interface (GUI). The terminal provides a text-based environment where users can execute commands, run programs, manage files and directories, configure system settings, and perform various administrative tasks.

The Linux terminal typically presents a prompt, which is a text string indicating that it is ready to accept a command. Users can then type commands, followed by pressing the Enter key to execute them. The terminal interprets the commands and communicates with the underlying operating system to carry out the requested actions.

Linux terminals offer a powerful and flexible way to control and manipulate the system. They provide access to a wide range of utilities and tools that can be combined and scripted to automate tasks, manage processes, network connections, install software, and perform system maintenance. The terminal is often favored by advanced users, system administrators, developers, and those who prefer a more efficient and direct way to interact with their Linux systems.

Virtual terminals

As we know Linux is multi user operating system it means more than one user can login on the system and perform different task at a same time. Not all users have more than one keyboard and display networked with system for multiple user logins. So, user still can login with different names on Linux system at same time using single keyboard and display such terminals known as virtual terminals.

user can open virtual terminal by pressing alt+ctlr+f1..f6 that is hit alt+ctlr+f2 or alt+ctlr+f5 to open terminal 2 or 5 and login with username and password. Now open terminal 1 and execute who command to check user currently logged in.

User can continue upto tty6 i.e. default number of allowed tty consoles are 6(Not in all flavours of linux).

User can switch from tty1 to tty6 using Ctrl+Alt+F[1-6] on the console. By conventional tty1 is normal main console which appears after Linux started.

```
[Bsingh@localhost ~]$ who
Bsingh   tty1         2020-01-22 12:29 (:0)
Bsingh   pts/0        2020-01-22 12:29 (:0)
Bsingh   pts/1        2020-01-22 12:48 (:0)
root     tty2         2020-01-22 12:51
Bsingh   tty5         2020-01-22 12:52
```

Note

if you are on terminal 2 and then use **logout** command to logout the current user. To return by default terminal press Ctrl+Alt+F1.

3. Linux Basic Commands

Let's start with some simple commands.

1) pwd command (print working directory)

This 'pwd' command prints the whole path starting from root directory to current working directory. pwd stands for "print working directory." When you run the pwd command, it displays the current directory or the full path of the current working directory.

The current working directory refers to the directory in the file system that you are currently located in. It is important to know the current working directory because it affects how you navigate and reference files and directories on the command line.

When you execute the pwd command, it prints the absolute path of the current directory to the terminal. For example, if your current working directory is "/home/user/documents," running the pwd command will display "/home/user/documents" as the output.

The pwd command is useful when you need to confirm the exact location within the file system hierarchy where you are working. It helps you understand your current context and can be particularly helpful when writing scripts or navigating through complex directory structures.

Syntax

```
$pwd [options]
```

For example

```
$ pwd
```

```
/home/Bsingh
```

2) cal command (calendar)

This command displays the calendar of the current month. By default, when you run the cal command without any arguments, it displays the current month's calendar. For example, if you run cal in May 2023, it will show the calendar for May 2023.

Syntax

```
$ cal [[month] year]
```

For example

1. To display current month of current year

```
$ cal
```

Jan 2020
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

2. To display calendar for the specified month and year.

Syntax $cal month year

```
$ cal 08 1991
```

August 1991
Su Mo Tu We Th Fr Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

3. To display the calendar of specific year

```
$ cal 2022
```

4. To display previous, current and next month of current year

```
$ cal -3
```

3) echo command

This command will echo whatever you provide it that is it displays a line of text or string on standard output

Syntax

```
$ echo [options] [strings]
```

Options

-e	enable the interpretation of escaped characters
\n	new line
\t	horizontal tab
\v	vertical tab
\r	carriage return with backspace
\b	backspace

For example

1. To display a string

```
$ echo "Bsingh"
Bsingh
```

2. To display the value of a variable

The ‘echo’ command is used to display the values of a variable. One such variable is ‘HOME’. To check the value of a variable, precede the variable with a $ sign.

```
$ echo $HOME
/home/Bsingh
```

3. Echo command with \n option

```
root@BSingh-14ITL6:~# echo -e "this \nis \na \nnew \nproject"
this
is
a
new
project
```

4. Echo command with \t option

```
root@BSingh-14ITL6:~# echo -e "this \tis \ta \tnew \tproject"
this    is      a       new     project
```

4) date command

This command is used to displays current time and date. This also print the different format of previous and future date and time. When you run the date command without any arguments, it will simply display the current date and time in the default format specified by your system's

settings. The output typically includes information such as the day of the week, month, day, time, and time zone.

Syntax

```
$ date [options] [+format]
```

Options

-d, --date=STRING: It is used to display time described by STRING.
%a: It is used to display the abbreviated weekday name (e.g., Sun)
%A: It is used to display the full weekday name (e.g., Sunday)
%b:it is used to display the abbreviated month name (e.g., Jan)
%B: It is used to display the full month name (e.g., January)
%c: It is used to display the date and time (e.g., Thu Mar 3 23:05:25 2005)
%C: It is used to display the century; like %Y, except omit last two digits (e.g., 20)
%d: It is used to display the day of the month (e.g., 01)
%D: It is used to display date; same as %m/%d/%y
%F: It is used to display the full date; same as %Y-%m-%d
%T:it is used to display the time; same as %H:%M:%S
%u: It is used for the day of the week (1..7); 1 is Monday
%y: It is used for the last two digits of the year (00..99)
%Y: It is used for a year

For example

1. To display current date and time

```
$ date
Fri Jan 24 09:07:09 IST 2020
```

2. If you are interested only in time, you can use 'date +%T' (in hh:mm:ss):

```
$ date +%T
01:13:14
```

3. Displacing date in different format

```
root@BSingh-14ITL6:~# date +'%A %d-%m-%y'
Thursday 29-09-22
root@BSingh-14ITL6:~# date +'%a %d-%m-%y'
Thu 29-09-22
root@BSingh-14ITL6:~# date +'%b %d-%m-%y'
Sep 29-09-22
root@BSingh-14ITL6:~# date +'%B %d-%m-%y'
September 29-09-22
root@BSingh-14ITL6:~# date +'%c %d-%m-%y'
Thu Sep 29 06:19:57 2022 29-09-22
```

```
root@BSingh-14ITL6:~# date +'%C %d-%m-%y'
20 29-09-22
```

4. Display date using -d option

```
root@BSingh-14ITL6:~# date -d now
Thu Sep 29 06:03:07 IST 2022
root@BSingh-14ITL6:~# date -d tomorrow
Fri Sep 30 06:03:57 IST 2022
root@BSingh-14ITL6:~# date -d yesterday
Wed Sep 28 06:04:25 IST 2022
root@BSingh-14ITL6:~# date -d "last monday"
Mon Sep 26 00:00:00 IST 2022
root@BSingh-14ITL6:~# date -d "next monday"
```

5) tty command

It is used to displays current terminal connected to system as standard input. The **tty** stands for teletype known as terminal.

The `tty` command is a command-line utility in Unix-like operating systems, including Linux. It is used to print the file name of the terminal connected to the standard input.

When you run the `tty` command, it returns the file name of the terminal device associated with your current session. The terminal device is the device or file that handles input and output for the terminal. The `tty` command allows you to determine the terminal device you are currently using.

The output of the `tty` command is typically a file name that represents the terminal device, such as "/dev/tty1" or "/dev/pts/0". The actual file name may vary depending on your system configuration and the specific terminal you are using.

The `tty` command is often used in shell scripts or command pipelines to determine the current terminal device and perform actions or output information accordingly. It can be useful in scenarios where you need to interact with or manipulate the terminal device directly.

```
$ tty
/dev/pts/1
```

6) whoami command

This command is used to display the name of user who is currently logged in. It is used to display the username of the currently logged-in user. When you run the `whoami` command, it returns the username associated with the current user session. This is particularly useful when working in a multi-user environment or when you need to quickly identify the user you are logged in as.

The output of the `whoami` command is typically a single line containing the username of the current user, such as "john" or "alice". It does not display any additional information beyond the username.

The `whoami` command is often used in shell scripts or command pipelines to retrieve the username and perform actions or enforce certain permissions based on the user's identity. It provides a simple and convenient way to determine the current user in a command-line environment.

Syntax

```
$ whoami
```

For example

```
profbsingh@BSingh-14ITL6:/$ whoami
profbsingh
```

7) id command

It is used to display information about the current user or a specified user, such as their user ID (UID), group ID (GID), and group membership. When you run the `id` command without any arguments, it displays information about the current user. The output typically includes the user's UID, GID, and a list of supplementary group IDs they belong to.

Here are some common ways to use the `id` command:

- `id`: Display information about the current user.

- `id [username]`: Display information about the specified user. For example, `id bsingh` will show information about the user named "bsingh".

- `id -u`: Display only the user's UID.

- `id -g`: Display only the user's primary GID.

- `id -G`: Display a comma-separated list of the user's supplementary group IDs.

The `id` command is useful for retrieving information about user accounts and their group affiliations. It can be used to verify user identities, determine group memberships, and manage permissions and access control in a Unix-like system.

Syntax

```
$ id [options] [user]
```

For example

1. To display the information of currently logged in user

```
profbsingh@BSingh-14ITL6:/$ id

uid=1000(profbsingh)gid=1000(profbsingh)groups=1000(profbsingh),4(adm),20(dialout),24(
cdrom)
```

By default, information about the current user is displayed. If another username is provided as an argument, information about that user will be printed.

2. To display the information about specified user

```
profbsingh@BSingh-14ITL6:/$ id root
uid=0(root) gid=0(root) groups=0(root)
```

3. To display only user id

```
profbsingh@BSingh-14ITL6:/$ id -u
1000
```

4. To display only group id

```
profbsingh@BSingh-14ITL6:/$ id -g
1000
```

8) clear command

When you run the clear command, it will clear the entire terminal screen, removing all previous output and commands. This can be useful when the terminal screen becomes cluttered with information and you want to start with a clean slate.

By clearing the terminal screen, the clear command makes it easier to read and work with new commands or output that follows. It helps maintain a visually organized and uncluttered working environment.

To use the clear command, simply type clear at the command prompt and press the Enter key. The terminal screen will be cleared, and you'll have a blank screen to continue your work.

It's important to note that running the clear command does not delete any previously executed commands or their output. It only clears the screen visually, allowing you to have a fresh start without scrolling through previous information.

Syntax

```
$ clear
```

9) help command and help option

As the name suggest help command is used to learn about built in commands. Nobody can remember all the commands all the time so user can use help command or help option with almost all the commands. We can use help option from command like

1. With almost every command, '--help' option shows usage summary for that command.

```
$ date --help
Usage: date [OPTION]... [+FORMAT] or: date [-u|--utc|--universal]
[MMDDhhmm[[CC]YY][.ss]] Display the current time in the given FORMAT, or set
the system date.
```

2. Help command

```
$ help pwd
```
```
pwd: pwd [-LP]
    Print the name of the current working directory.

    Options:
      -L    print the value of $PWD if it names the current working
            directory
      -P    print the physical directory, without any symbolic links

    By default, `pwd' behaves as if `-L' were specified.

    Exit Status:
    Returns 0 unless an invalid option is given or the current directory
cannot be read.
```

10) whatis command

This command gives a one-line description about the command. It can be used as a quick reference for any command.

Syntax

```
$ whatis [options]
```

For example:-

```
$ whatis date
```
```
date (1) - print or set the system date and time
```

```
[root@localhost Bsingh]# whatis who
```
```
who (1)              - show who is logged on
```

```
profbsingh@BSingh-14ITL6:~$ whatis -v who
```

```
who (1)              - show who is logged on
```

```
profbsingh@BSingh-14ITL6:~$ whatis -r who
```

```
who (1)              - show who is logged on

whoami (1)           - print effective userid
```

```
profbsingh@BSingh-14ITL6:~$ whatis -l who
```

```
who (1)              - show who is logged on
```

11) w command

w command is used to check which users are logged in to the system, and what command they are executing at that particular time:

Syntax

```
$ w [options] [username]
```

For example

```
[Bsingh@localhost ~]$ w
```

```
 20:52:02 up 1 day, 10:55,  3 users,  load average: 0.32, 0.35, 0.25
 USER     TTY      LOGIN@   IDLE   JCPU   PCPU WHAT
 Bsingh   tty1     Thu09          34:54m  3:31   0.07s  /usr/bin/startplasma-x11
 Bsingh   pts/0    Thu09          34:54m  0.00s  6.00s  kded5 [kdeinit5]
 Bsingh   pts/1    11:19           1.00s   0.19s  0.06s  /bin/bash
```

The first line of the output shows system information:

- **System time:**The current system time.
- **Up time:**How long the system has logged in.
- **Number of users:**The number of users currently logged in.
- **Average system load:**The average number of jobs running on the system in the last 1, 5, and 15 minutes, respectively.

The second line shows user and process information:

- **USER:**The names of currently logged in users.
- **TTY:**The name of the terminal the user is logging in from.
- **FROM:**The name or IP address of the terminal or host the user is logging in from.
- **LOGIN@:**The time the user logged in, in a 24-hour format.
- **IDLE:**The time since the user last used the terminal; displays**?xdm?**if the user is currently active.
- **JCPU:** The total run time of all system processes attached to the user's terminal.
- **PCPU:** Elapsed time for the user's current process.
- **WHAT:** The name of the user's current process.

12) history command

History command is used to display the previously executed commands and these commands are saved in a history file. When you run the history command in a terminal or command prompt, it displays a list of previously executed commands, typically showing the command number and the command itself.

The history command allows you to view and recall commands you have entered during your current session. It is useful for keeping track of your command history and quickly reusing or modifying previously executed commands.

Syntax

```
$ history
```

For example

```
$ history
```

```
 1  systemctl disable dnf -makecache.service
 2  su
 3  sudo dnf install ./rpmfusion-
 4  sudo dnf install ./rpmfusion-free-release-31.noarch.rpm
 5  su
 6  hostname
 7  hostname fedora31
 8  sudo hostname fedora31
 9  sudoedit
10  sudo hostname fedora31
```

13) man command (Manual Pages)

'--help' option and 'whatis' command do not provide thorough information about the command. For more detailed information, Linux provides manual pages and info pages. To see a command's manual page, man command is used. It is used to display the manual pages for various commands, system calls, and library functions. It provides detailed information, usage examples, and descriptions of command options.

Syntax

```
$ man [options]
```

For example

```
$ man date
```

DATE(1) User Commands DATE(1)

NAME

date - print or set the system date and time

SYNOPSIS

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION

Display the current time in the given FORMAT, or set the system date.

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

-d, --date=STRING

display time described by STRING, not 'now'

--debug

annotate the parsed date, and warn about questionable usage to stderr

14) bc command (basic calculator)

The bc command in Linux is a calculator that provides advanced mathematical capabilities. It stands for "basic calculator" and is an arbitrary-precision calculator language. It supports a wide range of mathematical functions and operations, making it useful for both simple and complex calculations.
Syntax

```
$ bc
```

For example

```
[Bsingh@localhost ~]$ bc
```

bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
5+6
11
5*6
30
30/2
15
quit
[Bsingh@localhost ~]$

Note: - user need to type 5+6 then press enter key, output will be displayed in the basic calculator. If user want to quit from calculator, then type quit and press enter key.

Exercise

I. Write a command to store a manual of any command into file under home directory.

II. Suppose file1 and file2 contains its own data, if you execute following command then what will be output?

    $cat <file1>file2

III. What will be output of following commands $expr 8 + 8 and $expr 8 * 8? How expr command different from bc command?

IV. How $whoami is different from $who am i ?

V. Is this command **ls > file** different from **ls | tee file** ? If yes then how?(file is name of any file)

VI. How BASH is different from DOS?

VII. Write a command to set system clock at 9:00am?

VIII. How clear command is different from press ‘Ctrl+l’ to clear terminal screen ?

IX. Write a command to count number of character, words and lines in given file X?

X. Write a command to list USB ports and device connected to it?