Linux Notes for Reference

07 February 2023 09:36

https://www.geeksforgeeks.org/linux-from-the-beginning-history-and-evolution/

https://www.baeldung.com/linux/boot-process

https://www.linuxfoundation.org/blog/blog/the-linux-foundation-its-not-just-the-linux-operating-system

https://www.digitalocean.com/community/tutorials/create-a-partition-in-linux

https://www.digitalocean.com/community/tutorials/how-to-add-swap-space-on-ubuntu-22-04

https://www.codingninjas.com/codestudio/library/features-of-the-linux-operating-system

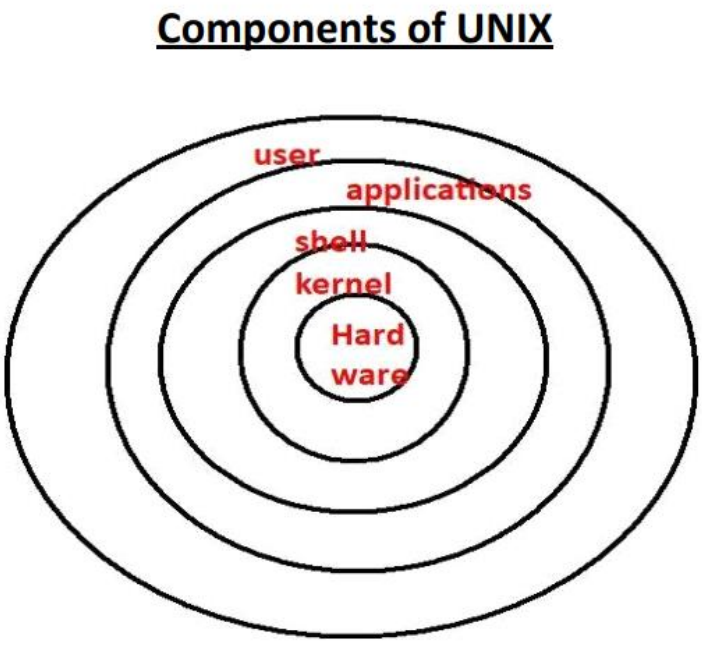
https://www.javatpoint.com/linux-distributions

https://www.javatpoint.com/architecture-of-linux

https://www.geeksforgeeks.org/the-linux-kernel/

https://www.geeksforgeeks.org/difference-between-linux-and-windows/

https://www.zdnet.com/article/what-are-virtualbox-guest-snapshots-and-how-do-you-take-them/



Shell:

✻ It is the outer layer of UNIX operating System.

✻ It reads our command, verify syntax and check whether the corresponding command related application is available or not.

✻ If everything is proper, then shell interprets our command into kernal understandable form and handover to the kernal.

✻ Shell acts as interface between user and kernal.

Kernal:

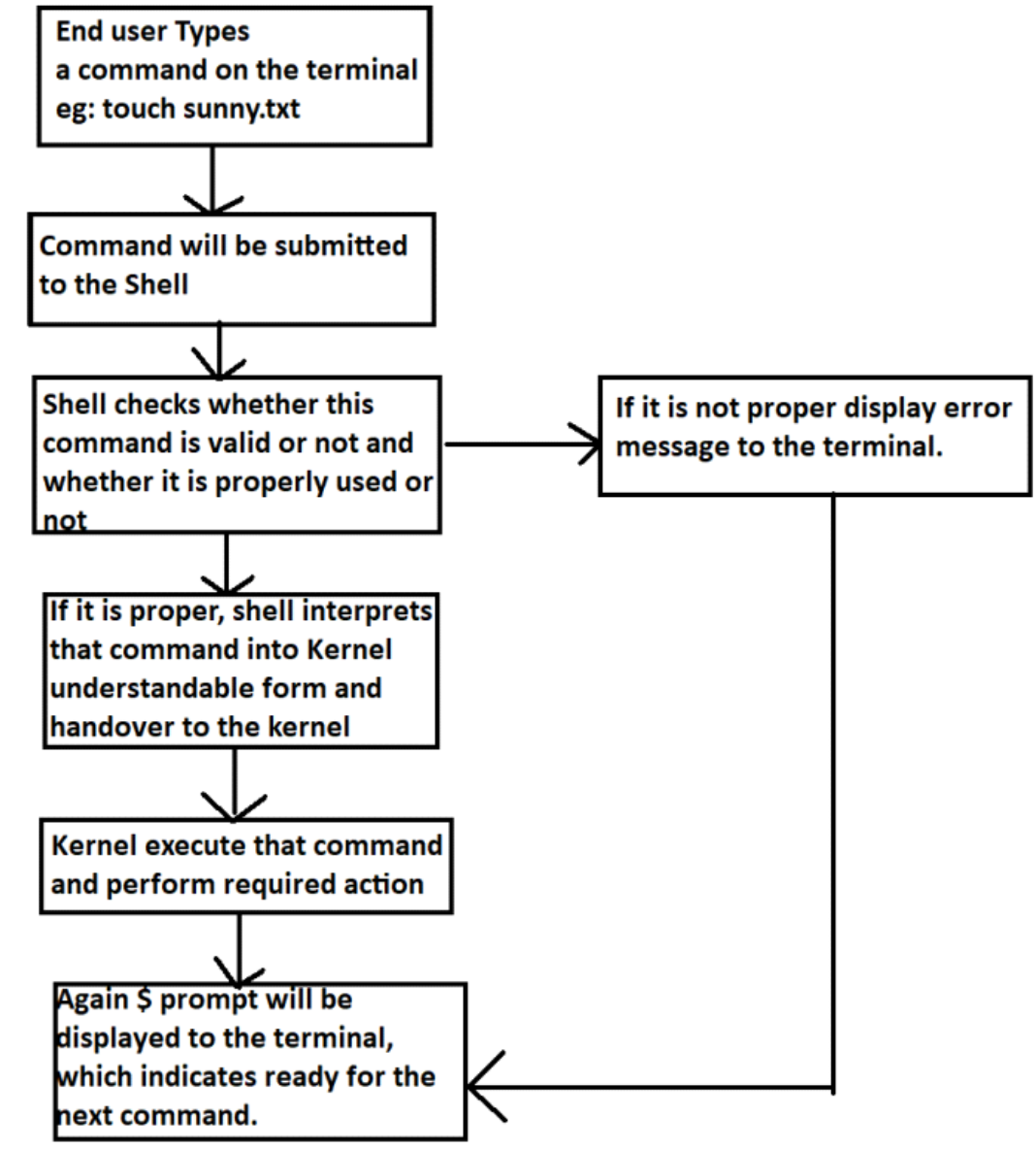
✻ It is the core component of UNIX operating system.

✻ It is responsible to execute our commands.

✻ It is responsible to interact with hardware components.

✻ Memory allocation and processor allocation will takes care by kernal

Command Execution Flow



Linux File System Structure :-

1. /bin :- bin - stands for binary.s

.

Binary is a file which contains the compiled source code.

We can also call it as executable, because it can be executed on the computer.

/bin is a sub-directory of the root directory in Unix/Linux OS.

this directory contains basic commands which is enough for the minimal system function
ex :- ls, cat, cp

2. /sbin : system binaries or super user binaries. This folder contains commands which are required for changing system properties.
Ex:- adduser,reboot,shutdown

3. /boot : The contents are mostly Linux kernel files and bootloader files(files needed to start up the operating system)

4. /dev : This contains device files
This file represents your speaker device,keyboard

5. /etc :- it contains all system related configuration files in here or in its sub-directories
A "configuration file" is defined as a local file used to control the operation of a program;it cannot be an executable binary.
Ex:- adduser.conf, theme config

6. /cdrom :- directory is a standard practice to mount cd, but not necessary. We use media and mnt to mount anything these days

/home :- The home directory can be said as a personal working space for all the users except root. There is a separate directory for every user. For example, two users ‘jitendra' and ‘jack' will have directories like "/home/jitendra“ and "/home/jack“

/lib :-
directory contains those shared library files needed to boot the system and run the commands in the filesystem,
ie. by binaries in /bin and /sbin

Only the shared libraries required to run binaries in /bin and /sbin will be here
Difference between lib, lib32, lib64, libx32

lib :- architecture independent files.
lib32 :- for 32 bit architecture libraries
lib64 :- for 64 bit architecture libraries
libx32 :- for 64 bit architecture libraries but the pointer size is 32 bit,
Normally software using the x86-64 instruction set uses 64-bit pointer size.

/media :- When you connect a removable media such as USB disk, SD card or DVD, a directory is automatically created under the /media directory for them. You can access the content of the removable media from this directory.

/mnt – Mount directory
This is similar to the /media directory but instead of automatically mounting the removable media, mnt is used by system administrators to manually mount a filesystem

/opt – Optional software
Traditionally, the /opt directory is used for installing/storing the files of third-party applications that are not available from the distribution's repository.

In the old days, "/opt" was used by UNIX vendors like AT&T, Sun, DEC and 3rd-party vendors to hold "Option" packages; i.e. packages that you might have paid extra money for.

/proc :- It contains useful information about the processes that are currently running.
It could be used for obtaining information about a system, we can also edit the config files related to kernel here.

/root :- it works as the home directory of the root user. So instead of /home/root, the home of root is located at /root. root directory (/) is different from root user directory.

/tmp :- this directory holds temporary files. Many applications use this directory to store temporary files. Even you can use directory to store temporary files.

the contains of the /tmp directories are deleted when your system restarts. Some Linux system also delete files old files automatically so don' store anything important here.

/var :- stores system-generated variable data files. This includes spool directories and files, administrative and logging data, cache, transient and temporary files.
Ex :- /var/spool contains data which is awaiting some kind of later processing

/run :- runtime variable data. The purposes of this directory were once served by /var/run, system may use both

/srv :- This directory gives users the location of data files for a particular service.
For example, if you run a HTTP or FTP server, it's a good practice to store the website data in the /srv

directory.

/usr :- User System Resources. in '/usr' go all the executable files, libraries, source of most of the system programs. For this reason, most of the files contained therein is read only (for the normal user).

why /usr/bin , /usr/sbin ?

/sys :- allows you to get information about the system and its components (mostly attached and installed hardware) in a structured way.
Ex:- device, kernel, firmware

/snap :- The /snap directory is, by default, where the files and folders from installed snap packages appear on your system.

Types of Files in Linux:-

In Linux everything is treated as File.

All files are divided into 3 types

1) Normal or Ordinary files:

These files contain data. It can be either text files (like abc.txt) OR binary files (like images, videos etc).

2) Directory Files:

🕐 These files represent directories.

🕐 In windows, we can use folder terminology where as in linux we can use directory terminology.

🕐 Directory can contains files and sub directories.

3) Device Files:

In Linux, every device is represented as a file. By using this file we can communicate with that device.

Note: short-cut commands to open and close terminal

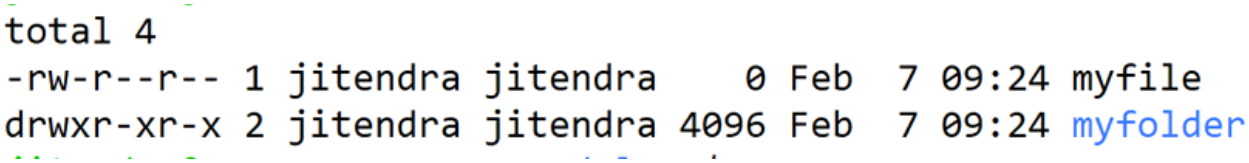
ctrl+alt+t ➔ To open terminal

ctrl+d ➔ To close terminal

How to check File Type:

In Ubuntu, blue color files represents directories and all remaining are considered as normal files. This color conventions are varied from flavour to flavour. Hence it is not standard way to check file type.

We have to use 'ls -l' command



The first character represents the type of file.

d = Directory File

- = Normal File

l = Link File

c = Character Special File

b = Block Special File

s = Socket File

Note: c, b, s are representing system files and mostly used by super user (also known as root user or admin user)

1. ls :-

We can use ls command to listout all files and directories present in the given directory.

We can get manual documentation for any command by using man.

man ls

It provides complete information about ls command.

Various options of ls Command:

1) ls

It will display all files and directories according to alphabetical order of names.

2) ls -r

It will display all files and directories in reverse of aplhabetical order.

3) ls | more

To display content line by line

(To come out we have to use q)

4)ls -l

To display long listing of files

```
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 4
-rw-r--r-- 1 jitendra jitendra    0 Feb  7 09:24 myfile
drwxr-xr-x 2 jitendra jitendra 4096 Feb  7 09:24 myfolder
```

5) ls -t

To display all files based on last modified date and time. Most recent is at top and old are at bottom.

6) ls -rt

To display all files based on reverse of last modified date and time. Old files are at top and recent files are at bottom.

7) ls -a

a means all

To display all files including hidden files. Here . and .. also will be displayed.

8) ls -A

A means almost all

To display all files including hidden files except . and ..

9)) ls -h

display in human readable format

10) ls -R

🕐 R means Recursive.

🕐 It will list all files and directories including sub directory contents also. By default ls will display only direct contents but not sub directory contents.

Eg: All the following commands are equal

$ ls -l -t -r

$ ls -t -r -l

$ ls -l -r -t

$ ls -ltr

$ ls -trl

Which Command will make a Long listing of all the Files in our System including Hidden Files,
sorted by Modification Date (Oldest First)?
ls -lat

5. date :-

We can use date command to display date and time of system.

Various Format Options:
1) date +%D
To display only date in the form: mm/dd/yy

2) date +%T
To display only time in the form: hh:mm:ss

3) date +%d
To display only day value

4) date +%m
To display only month value

5) date +%y
To display only year value in yy form

6) date +%Y
To display only year value in yyyy form.

7) date +%H
To display only Hours value (in 24 hours scale format)

8) date +%M
To display only Minutes value

9) date +%S
To display only Seconds value

10) date +%A
To display full day name ex:- Sunday

11) date +%a

abbreviated weekday name ex:- sun

12) date +%B
to full month name ex:- January

13) date +%b
abbreviated month name ex :- jan

14) date +%w

To display day of the week (0..6), 0 is Sunday

15) date +%W

To display week number of year

16) date +%q

To display quarter of year

17) date +%j

To display day of year (1----> 366)

18) date +%I

to display hour in 12 hour format (01..12)

19) date +%r

12-hour clock format ex :- 11:11:04 PM

Date Flags :-

1) date -u ----> to display UTC time (universal standard)
2) date -s "string" ----> to set date as given string

```
jitendra@pc:~$ sudo date -s "2023-03-10 12:12:12"
[sudo] password for jitendra:
Fri Mar 10 12:12:12 IST 2023
jitendra@pc:~$ date
Fri Mar 10 12:12:14 IST 2023
jitendra@pc:~$
```

3) date --date="string" ----> to display past and future date time

```
jitendra@pc:~$ date --date="next friday"
Fri Mar 17 00:00:00 IST 2023
jitendra@pc:~$ date --date="next hour"
Fri Mar 10 13:36:13 IST 2023
jitendra@pc:~$ date --date="next year"
Sun Mar 10 12:36:22 IST 2024
jitendra@pc:~$ date --date="last friday"
Fri Mar  3 00:00:00 IST 2023
jitendra@pc:~$ date --date="last month"
Fri Feb 10 12:36:37 IST 2023
jitendra@pc:~$ date --date="2 days ago"
Wed Mar  8 12:36:47 IST 2023
jitendra@pc:~$ date --date="yesterday"
Thu Mar  9 12:37:09 IST 2023
jitendra@pc:~$ date --date="tomorrow"
Sat Mar 11 12:37:22 IST 2023
jitendra@pc:~$ date --date="1 year"
Sun Mar 10 12:37:41 IST 2024
```

3. cal :-

$ cal To display current month calendar.

$ cal 2020 To display total year calendar.

$ cal 1 To display 1st year calendar.

$ cal 9999 To display 9999th year calendar.

$ cal 10000 cal: year '10000' not in range 1..9999

$ cal 08 2019 To display august 2019th calendar

cal -j ----> show julian calender

cal -3 ----> past current next month calender

cal -m 5 ---> 5th month calender

cal -y 2024 ---> year 2024 calender

cal -1 ---> current month calender (this is default)

```
jitendra@pc:~$ cal -j
        March 2023
 Su  Mo  Tu  We  Th  Fr  Sa
             60  61  62  63
 64  65  66  67  68  69  70
 71  72  73  74  75  76  77
 78  79  80  81  82  83  84
 85  86  87  88  89  90

jitendra@pc:~$ cal -3
   February 2023          March 2023           April 2023
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4            1  2  3  4                  1
 5  6  7  8  9 10 11   5  6  7  8  9 10 11   2  3  4  5  6  7  8
12 13 14 15 16 17 18  12 13 14 15 16 17 18   9 10 11 12 13 14 15
19 20 21 22 23 24 25  19 20 21 22 23 24 25  16 17 18 19 20 21 22
26 27 28              26 27 28 29 30 31     23 24 25 26 27 28 29
                                            30
```

cal -d string ------> string will be a date which support many formats including below

```
jitendra@pc:~$ cal -d 2 july 2023
      July 2023
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
jitendra@pc:~$ cal -d 2023-03
      March 2023
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
```

6. mkdir:-

We can create directories by using mkdir command.

1) mkdir dir1

To create a directory

2) mkdir dir1 dir2 dir3

To create multiple directories

3) mkdir dir1/dir2/dir3

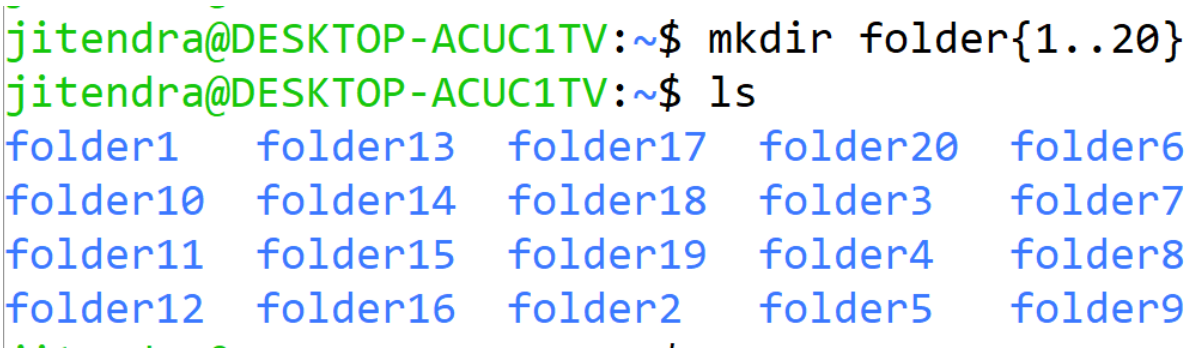
To create dir3. But make sure dir1 and in that dir2 should be available already.

4) mkdir -p dir1/dir2/dir3

 -p means path of directories.

 All directories in the specified path will be created.

 First dir1 will be created and in that dir2 will be created and within that dir3 will be created



7. rmdir :-

We can remove directories by using rmdir command.

1) $ rmdir dir1

To remove empty directory dir1

2) 2. $ rmdir dir1 dir2 dir3

To remove multiple empty directories

Note: rmdir command will work only for empty directories. If the directory is not empty then we will get error. We cannot use rmdir for files. Hence the most useless (waste) command in linux is rmdir.

8. rm :-

to delete files

rm file1 file2

To delete non empty folders

$ rm -r mydir
$ rm -R folder
Note: In Linux operating system, there is no way to perform undo operation. Once we delete a file or directory, it is impossible to retrieve that. Hence while using rm command we have to take special care.
The following command is the most dangerous command in linux, because it removes total file system.
Various options with rm Command:
1) interactive Option(-i)
While removing files and directories, if we want confirmation then we have to use -i

2)verbose Option(-v):
If we want to know the sequence of removals on the screen we should go for -v option.

9. Copy (cp)

1) To Copy from File1 to File2 (File to File)
 $ cp source_file destination_file
 $ cp file1 file2
 Total content fo file1 will be copied to file2.
 If file2 is not already available, then this command will create that file.
 If file2 is already available and contains some data, then this data will be over write with file1 content.

2) To Copy File to Directory:
 $ cp file1 file2 output
 file1 and file2 will be copied to output directory.
 Here we can specify any number of files, but last argument should be directory.
 output directory should be available already.

3) To Copy all Files of One Directory to another Directory:
 $ cp dir1/* dir2
 All files of dir1 will be copied to dir2
 But dir2 should be available already.

4) To Copy Total Directory to another Directory:
 $ cp dir1 dir2
 cp: -r not specified; omitting directory 'dir1'
Whenever we are copying one directory to another directory,
compulsory we should use -r option. $ cp -r dir1 dir2 total dir1 will be copied to dir2

5) To Copy Multiple Directories into a Directories:
 $ cp -r dir1 dir2 dir3 dir4 dir5
 dir1,dir2,dir3 and dir4 will be copied to dir5

10. mv :-

Moving and Renaming Directories:
Both moving and renaming activities can be performed by using single command: mv

1) Renaming of files:
$ mv oldname newname
Eg: $ file1.txt file2.txt
file1.txt will be renamed to file2.txt

2) Renaming of Directories:
$ mv dir1 dir2
dir1 will be renamed to dir2

3) Moving files to directory:
$ mv a.txt b.txt c.txt output
a.txt,b.txt and c.txt will be moved to output directory.

4) Moving of all files from one directory to another directory:
$ mv dir1/* dir2
All files of dir1 will be moved to dir2. After executing this command dir1 will become empty.

5) Moving total directory to another directory:
$ mv dir1 dir2

Note: If dir2 is already available then dir1 will be moved to dir2.
If dir1 is not already available then dir1 will be renamed to dir2.

--------Creation of files ----------

Creation of Files:
In Linux, we can create files in the following ways:
1) By using touch command (to create empty file)
2) By using cat command
3) By using editors like gedit, vi, nano etc

9. cat :-

cat > file1.txt

Eg:
$ cat > file1.txt
Hello Friends
Listen Carefully

Otherwise Linux will give Left and Right
ctrl+d To save and exit

If file1.txt is not already available, then file1.txt will be created
with our provided data.
If file1.txt is already available with some content,
then old data will be over written with
our provided new data.
Instead of overwriting, if we want append operation
then we should use >> with cat
command.
cat >> file1.txt
extra content
ctrl+d

>> for appending

) If we are using Touch Comamnd, but the File is already available then
what will happend?
The content of the file won't be changed. But last modified date and time
(i.e., timestamp) will be updated

10. Touch :-

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

creating files using touch

```
jitendra@DESKTOP-ACUC1TV:~/class$ touch file1
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1
jitendra@DESKTOP-ACUC1TV:~/class$ touch file2 file3 file4
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1  file2  file3  file4
jitendra@DESKTOP-ACUC1TV:~/class$ touch file{5..10}
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1  file10  file2  file3  file4  file5  file6  file7  file8  file9
jitendra@DESKTOP-ACUC1TV:~/class$ touch student{1..5}.txt
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1   file2  file4  file6  file8  student1.txt  student3.txt  student5.txt
file10  file3  file5  file7  file9  student2.txt  student4.txt
jitendra@DESKTOP-ACUC1TV:~/class$
```

touch file1 ------> change timestamp (both access and modify time)

touch -a file1 -----> change access file of file1

touch -m file1 ------> change modify time of file1

touch -r file1 file2 -----> use file1's timestamp as reference and change timestamp of file2

(now file2 timestamp will change and become same as file1)

try :- touch -r file2 -a file1 and observe using stat command what happens

---------View Content of the Files ----------

We can view content of the file by using the following commands
1) cat
2) tac
3) rev
4) head
5) tail
6) less
7) more

11. cat :-

$ cat < file1.txt
OR
$ cat file1.txt < is optional

```
jitendra@DESKTOP-ACUC1TV:~$ cat file1.txt
this is line 1
this is line 2
end of the file
```

-n option to give line numbering to file content
-b to give numbering to all lines apart from blank lines

```
jitendra@DESKTOP-ACUC1TV:~$ cat -b file1.txt
     1  this is line 1
     2  this is line 2
     3  end of the file


     4  this is appended line
```

We can view multiple files content at a time by using cat command.
$ cat file1.txt file2.txt file3.txt

Various utilities of cat Command:
1) To create new file with some content
$ cat > filename
 data
 ctrl+d

2) To append some extra data to existing file
$ cat >> filename
 extra data
 ctrl+d

3) To view content of file
 $ cat < filename or $ cat filename

4) Copy content of one file to another file
 $ cat input.txt > output.txt

5) To copy content of multiple files to a single file
 $ cat file1.txt file2.txt file3.txt > file4.txt

6) Merging/appending of one file content to another file
 $ cat file1.txt >> file2.txt

12. tac :-

It is the reverse of cat.
It will display file content in reverse order of lines. i.e first line will become last line and last line will become first line.
This is vertical reversal.

```
jitendra@DESKTOP-ACUC1TV:~$ cat file1.txt
this is line 1
this is line 2
end of the file


this is appended line
jitendra@DESKTOP-ACUC1TV:~$ tac file1.txt
this is appended line


end of the file
this is line 2
this is line 1
```

13. rev :

rev means reverse. Here each line content will be reversed. It is horizontal reversal.

```
jitendra@DESKTOP-ACUC1TV:~$ cat file1.txt
this is line 1
this is line 2
end of the file


this is appended line
jitendra@DESKTOP-ACUC1TV:~$ rev file1.txt
1 enil si siht
2 enil si siht
elif eht fo dne


enil dedneppa si siht
```

cat command will display total file content at a time. It is best suitable for small files. If the file contains huge lines then it is not recommended to use cat command. We should go for head, tail, less and more commands.

14. head :

We can use head command to view top few lines of content.

✻ head file1.txt

 It will display top 10 lines of file1.txt.

 10 is the default value of number of lines.

✻ head -n 30 file1.txt OR head -30 file1.txt

 To display top 30 lines of the file.

 Instead of 30 we can specify any number.

✻ head -n -20 file1.txt

To display all lines of file1.txt except last 20 lines.

✻ head -c 100 file1.txt

To display first 100 bytes of file content

15. tail :

 We can use tail command to view few lines from bottom of the file.

 It is opposite to head command.

✻ tail file1.txt

Last 10 lines will be displayed.

✻ tail -n 30 file1.txt OR tail -30 file1.txt OR tail -n -30 file1.txt

It will display last 30 lines.

✻ tail -n +4 file1.txt

It will display from 4th line to last line

✻ tail -c 200 file1.txt

It will display 200 bytes of content from bottom of the file.

16. more :

We can use more command to view file content page by page.

✻ more file1.txt

 It will display first page.

 Enter To view next line

 Space Bar To view next page

 q To quit/exit

✻ more -d file1.txt

-d option meant for providing details like
--More--(5%)[Press space to continue, 'q' to quit.]

17. less :

 By using more command, we can view file content page by page only in forward direction.
 If we want to move either in forward direction or in backward direction then we should go for less command.

d To go to next page.(d means down) **b** To go to previous page. (b means backward)

Creation of Hidden Files and Directories:-

 If any file starts with '.' , such type of file is called hidden file.

 If we don't want to display the files then we have to go for hidden files.

 Hidden files meant for hiding data. All system files which are internally required by kernal are hidden files.

 We can create hidden files just like normal files, only difference is file name should starts with dot.

touch .securefile1.txt
cat > .securefile1.txt

Even by using editors also we can create hidden files.

We can create hidden directories also just like normal directories.
mkdir .db_info

Note: By using hidden files and directories we may not get full security. To make more secure we have to use proper permissions. For this we should use 'chmod' command.

Interconversion of Normal Files and Hidden Files:

Based on our requirement, we can convert normal file as hidden file and viceversa.
mv a.txt .a.txt
We are converting normal file a.txt as hidden file.
mv .a.txt a.txt
Similarly directories also
mv dir1 .dir1
mv .dir1 dir1

----------Comparing Files----------

18. cmp :-

It will compare byte by byte.
cmp file1.txt file2.txt
If content is same then we won't get any output.
If the content is different, then it provides information about only first difference. byte number and line number will be provided.
$ cmp a.txt c.txt
a.txt c.txt differ: byte 7, line 2
Note: cmp command won't show all differences and show only first difference.

19. diff:-

It will show all differences in the content.
diff file1.txt file2.txt If the content is the same then no output.
If the content is different then it will show all differences

-q shows message when files are different.
-s shows message when files are same | identical
-y shows comparison line by line (parallel comparison)
-i ignore the uppercase and lowercase

< file1
> file2

c = change
d = delete
a = add

3,4d2 → delete line 3 and line 4 in order to reduce the file to 2 lines equal.

3,6d2 → delete line 3 and line 6 in order to reduce the file to 2 lines and make equal.

4a5,7 → after line 4 add lines 5 to 7 in first file.

20. sdiff :-

We can use sdiff command for side by side comparison (parallel comparison)

```
$ sdiff a.txt b.txt
Sunny                                   Sunny
Bunny                                   Bunny
Chinny                                  Chinny
Vinny                                   Vinny
Pinny                                   Pinny
$ sdiff a.txt c.txt
Sunny                                   Sunny
Bunny                                 | bunny
Chinny                                | chinny
Vinny                                   Vinny
Pinny                                   Pinny
```

Note: sdiff comamnd and diff command with -y option are same.

21. comm

By using this command we can compare data of two sorted files.
comm file1.txt file2.txt
It display results in 3 columns
column-1: Data present only in file1.txt but not in file2.txt
column-2: Data present only in file2.txt but not in file1.txt
column-3: Common data of both files.

With comm command we can use the following options
-1 If we don't want to display column-1
-2 If we don't want to display column-2
-3 If we don't want to display column-3
-12 If we don't want to display columns 1 and 2

```
jitendra@DESKTOP-ACUC1TV:~$ comm file1.txt file2.txt
a
b
c
                d
                e
                f
        g
        h
        i
```

22. Word Count (WC)

We can use wc command to count number of lines, words and characters present in the given file.
wc filename
 no_of_lines no_of_words no_of_characters filename
Eg:
$ wc a.txt
 4 26 166 a.txt
 4 □ Number of Lines
 26 □Number of words
 166 □ Number of characters (File size in bytes)

We can use the following options with wc Command

-l □ To print only number of lines
-w □ To print only number of words
-c □ To print only number of characters
-lw □ To print only number of lines and words
-lc □ To print only number of lines and characters
-wc □ To print only number of words and characters
-L □ To print number of characters present in Longest Line.

We can use wc command for multiple files simultaneously.
$ wc a.txt b.txt c.txt
 4 26 166 a.txt
 3 4 27 b.txt
 4 4 112 c.txt
11 34 305 total

23. sort :-

We can sort data of the file by using sort command.

sort filename
Here sorting is based on alphabetical order

$ cat a.txt
Sunny
Bunny
Chinny
Vinny
Pinny

$ sort a.txt
Bunny
Chinny
Pinny
Sunny

Vinny

If we want to sort based on reverse of alphabetical order, then we should use -r option.
$ sort -r a.txt
Vinny
Sunny
Pinny
Chinny
Bunny

If the file contains alphanumeric data, then first numbers will be considered and then alphabet symbols.
$ cat a.txt
7
Sunny
8
Bunny
1

If the file contains only numbers, then the sorting is not based on numeric value and it is just based on digits.
$ cat > a.txt
11
2
7
2222222
9
$ sort a.txt
11
2
2222222
7
9

f we want to sort based on numeric value then we have to use -n option.
-n means numeric value
$ sort -n a.txt
2
7
9
11
2222222

Bydefault sort command will display duplicate lines. If we want only unique lines then we

have to use -u option.
-u meant for unique lines.

```
$ cat a.txt
1
1
2
2
Sunny
Sunny
Bunny
$ sort a.txt
1
1
2
2
Bunny
Sunny
Sunny
$ sort -u a.txt
1
2
Bunny
Sunny
```

24. uniq

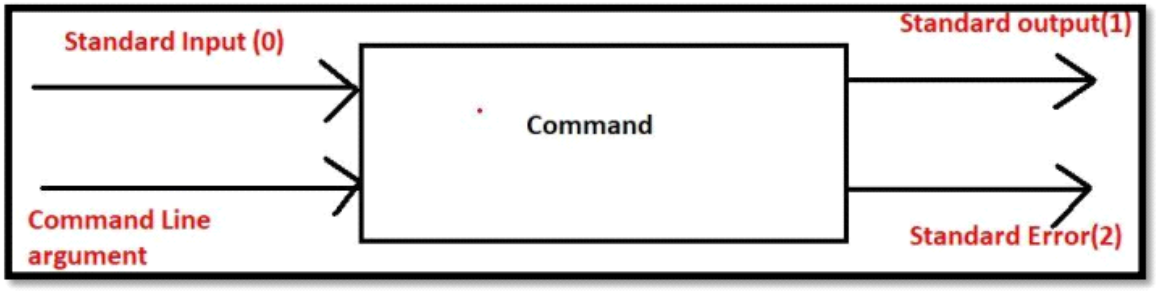
We can use uniq command to display unique content in the file.
But to use uniq command, compulsory the file should be sorted, otherwise it won't work properly

```
$ cat a.txt
Sunny
sunny
Bunny
Chinny
Sunny
Bunny
Chinny
```

```
$ uniq a.txt
Sunny
sunny
Bunny
Chinny
Sunny
Bunny
Chinny
```

With uniq command we can use multiple options:
-d To display only duplicate lines
-c To display number of occurrences of each line
-i Ignore case while comparing
-u To display only unique lines i.e the lines which are not duplicated

---Input and Output of Commands and Redirection------



Commands can take input, perform required operation and produces some output.
While executing command if anything goes wrong then we will get error message.

Command can take input either from standard Input or from command line arguments.
Command will produce results to either Standard output or Standard Error.

Standard Input, Standard Output and Standard Error are Data Streams and can flow from one place to another place. Hence redirection and piping are possible.

Command Line arguments are static and these are not streams. Hence redirection and piping concepts are not applicable to command line arguments.

These data streams are associated with some numbers.
Standard Input associated with 0.
Standard Output associated with 1.
Standard Error associated with 2.

Bydefault Standard input connected with keyboard, Standard output and Standard Error connected with Terminal. But we can redirect.

Standard Input from the keyboard and output to Standard Ouput
Device:
$cat
read required input from the keyboard
this data will be displayed to the standard output.
ctrl+d

Note: For the cat command if we are not providing any arguments, then the input will be taken from standard input device (keyboard) and display the output to the standard output device (Terminal).

$ cat
This is data provided from Standard Input
This is data provided from Standard Input

Input from command line arguments and error messages

Standard Error:
$ rm file100
rm: cannot remove 'file100': No such file or directory

We are providing filename as command line argument to the rm command.
Specified file not available and hence this command will produces error message to the Standard Error device (Terminal).

Note: Some commands may accept Standard Input and Some commands may accept command line arguments.
1) rm command will always accept command line arguments only.
rm file1 file2
2) echo command will always aceept command line arguments only.
echo "jitendra"
3) cat command can accept input either from standard input or from comamnd line arguments.

Redirection :-

As Standard Input, Standard Ouput and Standard Error are Data streams, we can redirect these streams.

Redirecting Standard Ouput:

We can redirect standard output by using > and >> symbols.
> will perform overwriting of existing data
>> will perform appending to existing data

Eg 1: To redirect the standard output of cat command from terminal to output.txt
$cat 1> output.txt
sample data
ctrl+d
sample data won't be displayed to the terminal and will write to output.txt
Redirection symbol > is always associated with 1 by default. Hence we are not required to specify 1 explicilty.

Redirecting Standard Error:
We can redirect error messages from the terminal to our own file by using > and >> symbols.
$ cal 34 w3892384208342 2>> error.txt

Now error message won't be displayed to the console and will written to error.txt.
For error redirection 2 is mandatory.

Redirecting Standard Input:
We can redirect standard input from keyboard to our required file.
We can perform input redirection by using < symbol.

$cat 0< a.txt 1>>output.txt 2>>error.txt

< symbol is always associated with 0 by default. Hence we can remove.

$cat < a.txt >>output.txt 2>>error.txt

***Note: To redirect both standard output and standard error to the same destination we can use shortcut as follows
$ cat < a.txt &> output.txt
&> means both standard output and standard error.

Q1) In How Many Ways Command can get Input?
2 ways. Either from Standard Input or from command line arguments.

----------Piping----------

Sometimes we can use output of one command as input to another command. This concept is called piping.
By using piping, multiple commands will work together to fulfill our requirement.

We can implement piping by using vertical bar (|).

$ ls -l /etc | wc
215 1940 11872

First ls got executed and the output of this command will become input to wc command.

Eg 2: $ ls -l /etc | more

Eg 3: $ ls -l /etc | wc |wc -l
The output is: 1

Eg 4: $ ls -l /etc | head -5

ex:- given a text file sort it , remove duplicates, give line numbering

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat>file1.txt
abc
def
ghi
abc
def
jitendra@DESKTOP-ACUC1TV:~/class$ cat<file1.txt | sort | uniq | nl
     1  abc
     2  def
     3  ghi
```

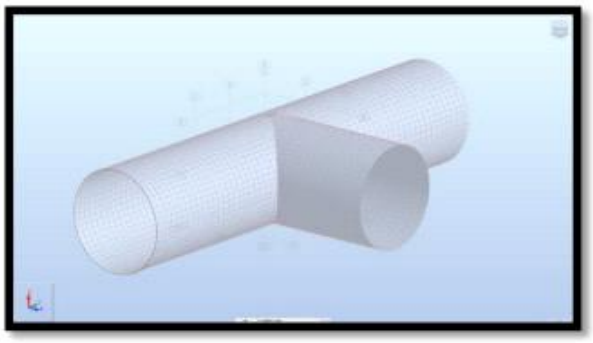
25. tee :-

Requirement:
The output of the ls command should be saved to output.txt and should be provided as input to wc command:
ls -l 1>output.txt | wc

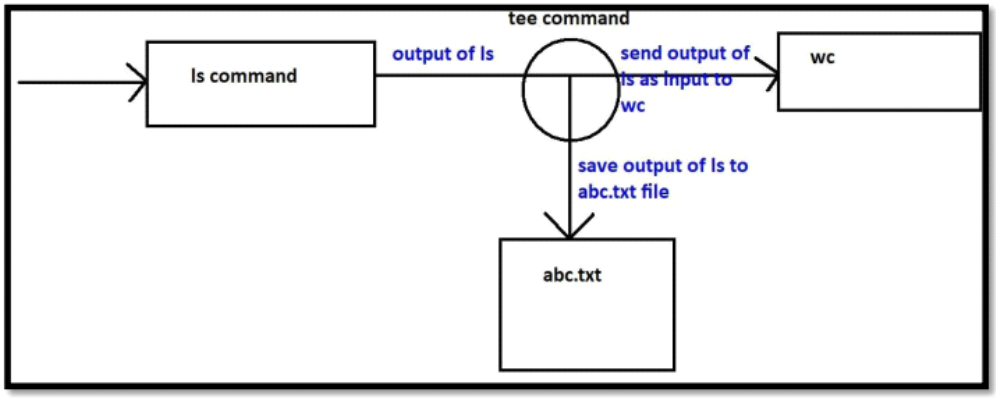
This command won't work because if we are using redirection in the middle of piping, it will break piping concept.

In piping, if we want to save the output of one command to a file and if we want to pass that output as input to next command simultaneously, then we should go for tee command

tee command is just like T-Junction or T-Pipe. It will take one input but provides two outputs.



Eg 1: To save the output of ls command to a file and to display to the terminal simultaneously.
$ ls –l It will display to the terminal
$ ls -l > abc.txt It will save to the abc.txt but won't display to the terminal.
$ ls -l | tee abc.txt



26. xargs

xargs is a Unix command which can be used to build and execute commands from standard input.

```
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file  file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ xargs rm
file
file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ ls
jitendra@DESKTOP-ACUC1TV:~/class$
```

rm command does not take standard input it neads arguments
so inorder to rm command with standard input we need to convert standard input into command line arguments
xargs command will do exactly that

Eg 1: Assume input.txt contains file names. Each file contains some data.
Read file names from the input.txt, write total content to output.txt and display the total number of lines present in output.txt.

$ cat input.txt | xargs cat | tee output.txt | wc -l

Eg 2: Assume input.txt contains file names. Read file names from the input.txt and remove all these files.

$ cat input.txt | xargs rm

----Regular Expressions and Wildcard Characters---

If we want to represent a group of strings according to a particular pattern, then we should go for regular expressions.
By using wildcard characters, we can build regular expressions.
A wildcard character can be used as a substitute for required sequence of characters in the
regular expression.

1) * Represents zero or more characters

2) ? Represents only one character

3) [] Range of characters

4) [abc] Either a or b or c

5) [!abc] Any character except a,b and c

6) [a-z] Any lower case alphabet symbol

7) [A-Z] Any upper case alphabet symbol

8) [a-zA-Z] Any alphabet symbol

9) [0-9] Any digit from 0 to 9

10) [a-zA-Z0-9] Any alphanumeric character

11) [!a-zA-Z0-9] Except alpha numeric character (i.e special symbol)

12) [[:lower:]] Any lower case alphabet symbol

13) [[:upper:]] Any upper case alphabet symbol

14) [[:alpha:]] Any alphabet symbol

15) [[:digit:]] Any digit from 0 to 9

16) [[:alnum:]] Any alpha numeric character

17) [![:digit:]] Any character except digit

18){} List of files with comma separator

examples

1) To list out all files present in current working directory $ ls *

2) To list out all files with some extension $ ls *.*

3) To list out all files starts with a $ ls a*

4) To list out all files starts with a and ends with t $ ls a*t

5) To list out all .java files $ ls *.java

6) To list out all files where file name contains only 2 characters and first character

should be 'a' $ ls a?

7) To list out all files where file name contains only 3 characters $ ls ???

8) To list out all files where file name contains atleast 3 characters $ ls ???*

9) To list out all files where file name starts with a or b or c $ ls [abc]*

10) To list out all files where file name should not starts with a, b and c $ ls [!abc]*

11) To list out all files starts with lower case aphabet symbol
$ ls [a-z]* OR $ls [[:lower:]]*

12) To list out all files starts with upper case aphabet symbol
$ ls [A-Z]* OR $ls [[:upper:]]*

13) To list out all files starts with digit.
$ ls [0-9]* OR $ls [[:digit:]]*

14) To list out all files where first letter should be upper case alphabet symbol, second letter should be digit and third letter should be lower case alphabet symbol.
$ ls [[:upper:]][[:digit:]][[:lower:]]

15) To list out all files starts with special symbol
$ls [![:alnum:]]*

16) To list out all files with .java and .py extension
$ ls {*.java, *.py}

Note: We can use these wildcard characters with the following commands also.
cp, mv, rm

17) To copy all files starts with digit to dir1 directory.
$cp [[:digit:]]* dir1
$cp [0-9]* dir1

18) To move all files starts with alphabet symbol and with .txt extension to dir2 directory?
$mv [[:alpha:]]*.txt dir2

19) Remove all files starts with a or b or c and ends with e or t.
$rm [abc]*[et]

--File Archieving and Compression commands--

it is very common requirement to pack and compress a
group of files. The main advanatages are:
1) It improves memory utilization
2) Transportation will become very easy
3) It reduces download times
etc

This process involves the following 2 activities:
1) Creation of Archive file
2) Apply compression algorithms on that archive file

Creation of Archive File

27. tar :-

We can group multiple files and directories into a single archive file by using tar command.
tar tape archive

tar [options] [archive-file] [file or directory to be archived]

-c : Creates Archive
-x : Extract the archive
-f : creates archive with given filename
-t : displays or lists files in archived file
-v : Displays Verbose Information
-z : zip, tells tar command that creates tar file using gzip
-j : filter archive tar file using bzip2
-r : update or add file or directory in already existed .tar file

A) To create tar file
tar -cvf demo.tar file1.txt file2.txt file3.txt
tar -cvf demo.tar *

B) To display table of contents of tar file
tar -tvf demo.tar

C) To Extract contents of tar file
tar -xvf demo.tar

here v means verbose it will just display files which are compressed on terminal
we can ignore it

compressing any file

There are multiple compression and decompression algorithms.

28. gzip It is very fast but less compression power

1) To Compress a file
$ gzip demo.tar
demo.tar.gz This file got created

we can compress any file not only tar file

2) To uncompress gz file:
$ gzip -d demo.tar.gz OR $ **gunzip** demo.tar.gz
 This command will provide our original tar file

29. bzip2 It is a bit slow but more compression power

1. To compress tar file:

$ bzip2 demo.tar
demo.tar.bz2 this file will be created.

2. To uncompress bz2 file:
$ **bunzip2** demo.tar.bz2

How to create tar File and compress in a Single Command:

1. By using **gzip compression** algorithm
To create tar and then compress
$ tar -czf demo.tar *.txt
z option will do compression
demo.tar will be created and it is already compressed
To uncompress and extract tar file
$ tar -xvzf demo.tar

2. By using **bzip2 compression** algorithm
Instead of 'z', we have to use 'j'
To create tar and then compress
$ tar -cjf demo.tar *.txt
j option will do compression
demo.tar will be created and it is already compressed
To uncompress and extract tar file
$ tar -xvjf demo.tar

```
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1   file11  file13  file15  file17  file19  file20  file4  file6  file8
file10  file12  file14  file16  file18  file2   file3   file5  file7  file9
jitendra@DESKTOP-ACUC1TV:~/class$ tar -zcf files.zip *
jitendra@DESKTOP-ACUC1TV:~/class$ ls
file1   file11  file13  file15  file17  file19  file20  file4  file6  file8  files.zip
file10  file12  file14  file16  file18  file2   file3   file5  file7  file9
jitendra@DESKTOP-ACUC1TV:~/class$
```

Use Case: Backup of Total User Home Directory

$ pwd
/home/jitendra
$ tar -cvzf backup.tar *

$mkdir new

home
$mv backup.tar newhome

$cd newhome
$ tar -xvzf backup.tar

28. grep :-

grep stands for

globally search a regular expression and print it
global regular expression print.
global regular expression parser

We can use grep command to search the given pattern in a single or multiple files.
grep <pattern> filename
It prints all matched lines.

-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 (uppercase lowercase)

-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.

-w : Match whole word

-A n : Prints searched line and nlines after the result.
-B n : Prints searched line and n line before the result.
-C n : Prints searched line and n lines after before the result.

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat>file1.txt
this is a sample
text file with
few lines of the text
data into it
jitendra@DESKTOP-ACUC1TV:~/class$ grep the file1.txt
few lines of the text
jitendra@DESKTOP-ACUC1TV:~/class$ grep -c the file1.txt
1
jitendra@DESKTOP-ACUC1TV:~/class$ grep -v the file1.txt
this is a sample
text file with
data into it
jitendra@DESKTOP-ACUC1TV:~/class$ grep -n the file1.txt
3:few lines of the text
jitendra@DESKTOP-ACUC1TV:~/class$ grep -l the file1.txt
file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ grep -l the *
file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$
```

to search in multiple files
grep linux file1.txt file2.txt

this will search for pattern linux in file1.txt and file2.txt

```
jitendra@DESKTOP-ACUC1TV:~/class$ grep -e file -e text file1.txt
text file with
few lines of the text
```

search all the mobile numbers in a given file

```
jitendra@DESKTOP-ACUC1TV:~/class$ grep  [6-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
 file1.txt
8095321456
is a new file 9876512340
```

29. egrep:-

Instead of using -e option, we can use egrep command directly.
It is extended grep.
It interprets patterns as an extended regular expression.

```
jitendra@DESKTOP-ACUC1TV:~/class$ egrep "(file|text)" file1.txt
text file with
few lines of the text
```

30. fgrep :-

grep with -F Option OR fgrep:

fgrep "Fixed String Global Regular Expression Print"

The grep filter searches a file for a particular pattern of characters and displays all lines that contain that pattern.
The fgrep filter searches for fixed-character strings in a file or files.

fgrep command interprets the PATTERN as a list of fixed string
grep always interpreted as regular expressions.

```
jitendra@DESKTOP-ACUC1TV:~/class$ grep ^few file1.txt
few lines of the text
jitendra@DESKTOP-ACUC1TV:~/class$ fgrep ^few file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ fgrep few file1.txt
few lines of the text
```

fgrep doesn't support the regular expressions like grep

All Regular expression patterns are divided into 3 types.

1) Character Patterns
2) Word Patterns
3) Line Patterns

1) Character Patterns:-

1) $ grep 'd*' demo.txt
 It display all lines which contains d followed by any number of characters.
 ubuntu not providing support for this.

2) $ grep 'c[aeiou]ll' demo.txt
It will search for call, cell, cill, coll, cull

3) $ grep 'b..l' demo.txt
. means any character. It will search for all 4 letter words where first letter should b and last letter should be l.

2) Word Patterns:-

\<word\> It will always searches for the given word
It is **exactly same** as

grep -w word demo.txt

\<xyz It will search for the word starts with xyz
xyz\> It will search for the word ends with xyz

```
jitendra@DESKTOP-ACUC1TV:~/class$ grep '\<line\>' file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ grep '\<lines\>' file1.txt
few lines of the text
jitendra@DESKTOP-ACUC1TV:~/class$
```

line is not a exact work in file1.txt
while lines is a pattern in file since we are matching exact work first command doesn't give any output

3) Line Patterns (Anchors):

^ Line starts with
$ Line ends with

1) $ grep '^d' demo.txt
It will display all lines starts with d

2) $ grep '^the' demo.txt
It will display all lines starts with the

3) $ grep '^\<the\>' demo.txt
It will display all lines starts with the word 'the'

4) $ grep '^[aeiou]' demo.txt
It will display all lines starts with vowel.

5) $ grep '^[^aeiou]' demo.txt
It will display all lines not starts with vowel.

6) $ grep 't$' demo.txt
It will display all lines ends with t

7) $ grep '[aeiou]$' demo.txt
It will display all lines ends with vowel

8) $ grep '[0-9]$' demo.txt
It will display all lines ends with digit.

9) $ grep '^unix$' demo.txt
It will display all lines where total line content should be unix

10) $ grep '^....$' demo.txt
It will display all lines where line contains exactly 4 characters.

11) $ grep '^\.' demo.txt
It will display all lines starts with .

12) $ grep '\$$' demo.txt
It will display all lines ends with $

13) $ grep '^$' demo.txt
It will display all blank lines

14) $ grep -v '^$' demo.txt
It will display all lines except blank lines

Q) How to Delete Blank Lines Present in the given File?

$ grep -v '^$' demo.txt > temp.txt
$ mv temp.txt demo.txt

Additional Patterns supported by only egrep but not grep:
1. (|) It matches any of the string in the given list
$ egrep '(unix|java|oracle)' demo.txt

2. {m} It matches exact number of preceding character.

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file1.txt
this is a sample
text file with
few lines of the text
data into it
8095321456
this
is a new file 9876512340
great
jitendra@DESKTOP-ACUC1TV:~/class$ grep [0-9]{10} file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$ egrep [0-9]{10} file1.txt
8095321456
is a new file 9876512340
jitendra@DESKTOP-ACUC1TV:~/class$ egrep [0-9]{11} file1.txt
jitendra@DESKTOP-ACUC1TV:~/class$
```

Same as grep [0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9] fil1.txt

grep doesn't support {} pattern ,

3. {m,n} The preceding character should match minimum m times and maximum n times.
$ egrep '[0-9]{1,5}' demo.txt
It will search for lines which contains minimum 1 digit and maximum 5 digit numbers.

4. {m,} minimum m number of times but no restriction on maximum number
$ egrep '[0-9]{3,}' demo.txt
minimum 3 digits but no restriction on maximum number.

Q. write Commands to Count the Number of Directories Present in the Current Working Directory?

$ ls -l | grep '^d' | wc -l

```
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 8
-rw-r--r-- 1 jitendra jitendra    0 Feb 11 15:22 file1
-rw-r--r-- 1 jitendra jitendra    0 Feb 11 15:22 file2
-rw-r--r-- 1 jitendra jitendra    0 Feb 11 15:22 file3
drwxr-xr-x 2 jitendra jitendra 4096 Feb 11 15:22 folder1
drwxr-xr-x 2 jitendra jitendra 4096 Feb 11 15:22 folder2
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l | grep ^d | wc -l
2
```

31. Cut :-

We can use cut command to extract data from the file.
The file should contain column formatted data, ie tabular data.

Create a file like below named emp.dat

eno|ename|esal|eaddr|dept|gender
100|sunny|1000|mumbai|admin|female
200|bunny|2000|chennai|sales|male
300|chinny|3000|delhi|accounting|female
400|vinny|4000|hyderabad|admin|male
500|pinny|5000|mumbai|sales|female

1) Display Character on specific Position in every Record:
$ cut -c 9 emp.dat
-c meant for specific character
e
y
y
n
y
Y

2) Display Range of Characters in every Record:
 $cut -c 5-9 emp.dat

It will display 5th to 9th characters in every record
 $cut -c 5- emp.dat

It will display 5th character to last character in every record
 $ cut -c -3 emp.dat

It will display from 1st character to 3rd character in every record.
 $ cut -c 3-5,7-10 emp.dat
It will display 3rd to 5th character and 7th to 10th character in every record.

3) Display Specific Column Data:
 $ cut -d '|' -f 3 emp.dat
-d means delimiter (separator). The default delimiter is tab.
-f means field
 It will display 3rd Field (OR Column) data.

4) Display Range of Columns:
 $ cut -d '|' -f 2-3 emp.dat
It will display 2nd and 3rd Columns data.

 $ cut -d '|' -f 2- emp.dat
It will display from 2nd column to last column data.

 $ cut -d '|' -f -3 emp.dat
It will display from 1st column to 3rd column data.

 $ cut -d '|' -f 1,3,5 emp.dat
It will display 1st, 3rd and 5th column data.

32. Paste

We can use paste command to join two or more files horizontally by using some delimiter. Default delimiter is tab.

Syntax: $ paste file1 file2 …

$ paste subjects.txt fee.txt
CoreJava 1000
AdvJava 2000
Python 2500
Django 1500
UNIX 250

It will show content side by side of both file subjects.txt and fee.txt

We can specify delimiter explicitly by using -d option.
$ paste -d '-' subjects.txt fee.txt
Core Java-1000
Adv Java-2000
Python -2500
Django-1500
UNIX-250
Note: Delimiter should be only one character. If we are providing more than one character, then it will consider only first character.

33. tr :-

tr means translate.
This command translates character by character.

$ cat > demo.txt
While learning unix not required to eat.

1) $ tr 'aeiou' 'AEIOU' < demo.txt
It will replace lower case vowels with upper case vowels in demo.txt
WhIlE lEArnIng UnIx nOt rEqUIrEd tO EAt.

2) $ tr '[a-z]' '[A-Z]' < demo.txt
Every lower case alphabet symbol will be replaced with upper case alphabet symbol.
WHILE LEARNING UNIX NOT REQUIRED TO EAT

3) $ tr '[a-zA-Z]' '[A-Za-z]' < demo.txt
Every lower case character will be replaced with upper case character and every upper case character will be replaced with lower case character.
tr '[a-zA-Z]' '[A-Za-z]' < demo.txt > temp.txt

4) $ tr 'aeiou' '7' < demo.txt
To replace every lower case vowel with digit 7.

5) $ tr '|' '\t' < emp.dat
Replace | symbol with tab in emp.dat
$ tr '|' ':' < emp.dat

6) $ tr -d 'a' < demo.txt
-d means delete
It will delete all occurrence of 'a' in demo.txt
$ tr -d 'aeiou' < demo.txt
It will delete all lower case vowels in demo.txt

7) $ tr -s 'a' < demo.txt
It replaces sequence of a's with a single a.
-s means squeeze-repeats

---------File Permissions----------

File Permissions describe the allowed operations by various users.
With respect to file permissions, all users are categorized into the following 4 types.
User Categories:
user/owner Represented by 'u'
group Represented by 'g'
others Represented by 'o'
all Represented by 'a'

Permission Types:

Number	Permission
0	No permission
1	Execute
2	Write
3	Execute and Write
4	Read
5	Read and Execute
6	Read and Write
7	Read, Write and Execute

Table_FilePermissions

Permission Type	For Files	For Directories
1. Read (r)	Read permission for the file means we can view content of the file.	Read permission for the directory means we can list out contents of that directory. i.e we can use ls command.
2. Write(w)	Write permission for the file means we can modify the content of the file.	Write permission for the directory means we can modify contents of that directory. i.e we can create a new file and we can delete an existing file etc
3. Execute	Execute permission for the file means we can execute the file just like a normal program.	Execute permission to the directory means we can enter into that directory. i.e we can use cd command.

Operations related to permissions:

We can perform the following 3 operations.
+ → Add a particular permission to user|group|other|all
- → Remove a particular permission to user|group|other|all
= → Assignment a particular permission to user|group|other|all

34. chmod Command:

chmod means change mode.
We can use chmod command to change file or directory permissions.

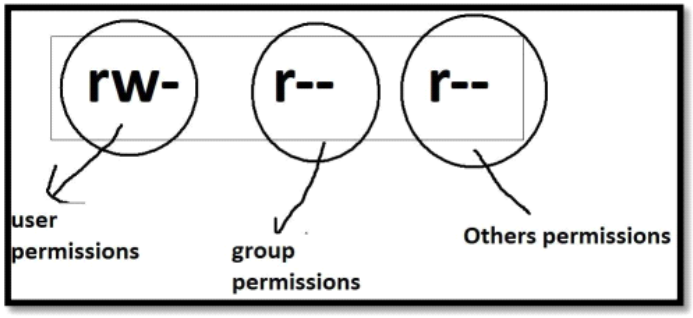
Syntax: $ chmod <user_category><operation><permission> file_name/directory_name

Eg: For user add execute permission,for group add write permission,for others remove read permission
$ chmod u+x,g+w,o-r demo.txt

Note: Only owner and super user (root) can change file permissions.

How to check Permissions of existing File:
By using ls -l command:

$ ls -l
total 0
-rw-r--r-- 1 jitendra jitendra 0 Nov 27 21:19 demo.txt
The file permissions are



Total 9 permissions. First 3 are user permissions, next 3 are group permissions and next 3 are others permissions.

user permissions: rw-
user can perform both read and write operations but not execute operation

group permissions: r--
group members can perform only read operation and cannot perform write and execute operations

others permissions: r--
other members can perform only read operation and cannot perform write and execute operations.

User Permissions + Group Permissions + Others Permissions
order is important

Eg 1: $ chmod u+x demo.txt
adding execute permission to the user

Eg 2: $ chmod u+w,g+rw,o+r demo.txt
adding write permission to the user
adding read and write permissions to the group
adding read permission to the others

Eg 3: $chmod u+x,g-w,o+w demo.txt
adding execute permission to the user
removing write permission from the group
adding write permission to the others

Eg 4: $ chmod u=rw,g=rw,o=r demo.txt
Now user permissions: rw-
group permission: rw-
others permission: r--

Eg 5: $ chmod a=- demo.txt
Now user permissions: ---
group permission: ---
others permission: ---

Eg 6: $ chmod a=rwx demo.txt
Now user permissions: rwx
group permission: rwx
others permission: rwx

Read Permission to the File:-
If the file not having read permission then we are not allowed to view content of the file. Hence cat, head, tail, more, less commands won't work.

Write Permission to the File:-
If the file not having write permission, then we cannot modify the content of the file.

Execute Permission to the File:-
If the user not has executed permission on any file, then he cannot execute that file as a program.

Read Permission to the Directory:-
If the user has read permission on any directory, then he can list out the contents of that directory. i.e he can use ls command.

Write Permission on the Directory:-
If the user has write permission on any directory, then he is allowed to modify the content of that directory. i.e he can add new files and remove existing files.

Execute Permission to the Directory:-
If the user not has executed permission on any directory, then he is not allowed to enter into that directory. i.e he cannot use cd command.

Note:- If the user not having read permission on any file, then he cannot execute that file even though he has executed permission.

Linux vs Security:

The virus files usually created by others.
others are not having execute permission on our directories. Hence others are not allowed to add virus files to our directories.
Hackers are not having executed permission on our directories. Hence they cannot read our file data.

Because of this, Linux is considered as more secured operating system.
Linux follows 2 levels of security.
1st level: login with credentials
2nd level: File and Directory permissions

Note: We are using permission types as r,w,x and these are considered as symbolic permissions. But we can also specify permissions by using octal number, such type of permissions are called numeric permissions.

Numeric Permissions:-

We can specify permissions by using octal number.
Octal means base-8 and allowed digits are 0 to 7.
0 → 000 → No Permission
1 → 001 → Execute Permission
2 → 010 → Write Permission
3 → 011 → Write and execute Permissions
4 → 100 → Read Permission
5 → 101 → Read and execute Permissions
6 → 110 → Read and write Permission
7 → 111 → Read, Write and execute Permissions

Note:
4 → Read Permission
2 → Write Permission
1 → Execute Permission
It is more easy to remember

5 → 4+1 → r-x
3 → 2+1 → -wx
6 → 4+2 → rw

35. umask :-

umask means user mask. Hiding permissions.
Based on umask value,default permissions will be there for files and directories.
The default umask value:022

$umask
0022

Igonre first 0 last 3 digits are umask value

Default permissions to the file: 666 - umask value = 666 - 022 = 644
(user r&w, group read, others read)

Default permissions to the directory= 777 - umask value =777-022=755
(user r&w&x, group r&x, others r&x)

Now whenever you create new files or directories the permissions will be based upon umask value

Q) For Newly created Files Default Permissions should be 444.
Then what should be umask Value?
Ans: 222
Everyone has only read access

36. chown :-

chown means change owner.
Only root user can perform this activity.
chown root demo.txt

Now the owner of demo.txt is root.

```
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 0
-rw-r--r-- 1 jitendra jitendra 0 Feb 11 15:22 file1
jitendra@DESKTOP-ACUC1TV:~/class$ sudo chown joker file1
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 0
-rw-r--r-- 1 joker jitendra 0 Feb 11 15:22 file1
```

37. Chgrp:-

chgrp means change group.
Only root user can perform this activity.

chgrp root demo.txt
Now the demo.txt belongs to root group.

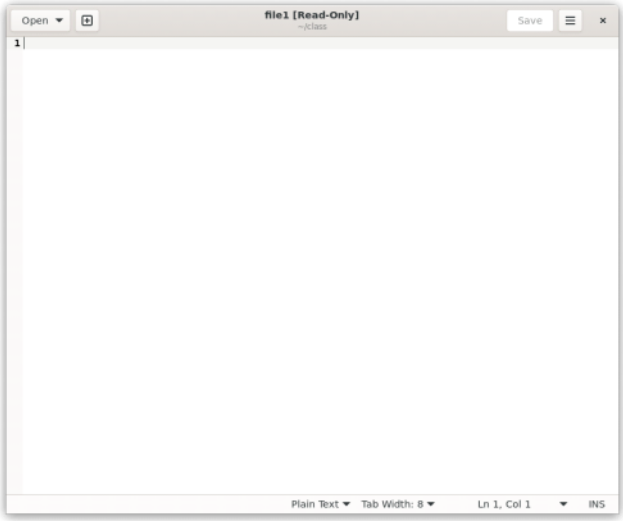
```
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 0
-rw-r--r-- 1 joker jitendra 0 Feb 11 15:22 file1
jitendra@DESKTOP-ACUC1TV:~/class$ sudo chgrp joker file1
jitendra@DESKTOP-ACUC1TV:~/class$ ls -l
total 0
-rw-r--r-- 1 joker joker 0 Feb 11 15:22 file1
```

----------Working with editors ----------

38. Gedit :-

It is graphical editor.
It is simply same as window's notepad. $ gedit file1.txt $ gedit first.sh

```
jitendra@DESKTOP-ACUC1TV:~/class$ gedit file1
```



39. Vi :-

vi means visual editor.
We can use vi to create new files and to edit content of the existing files.
$ vi file1.txt
If file1.txt is not available, then a new file will be created and opened that file for editing purpose.

To save this empty file we should use :wq (w means save and q means exit)
If the file already contains some data then editor will be opened that file and ready for edit.

How to edit the File:

There are 3 types of modes in file editing

1) Command Mode:

It is the default mode.
Here we can use any vi command.
From command mode, we can enter into insert mode by using multiple ways, but mostly by using i

2) Insert Mode OR Input Mode:

In this mode, we can modify file data. We can insert/append new data.
From insert mode, we can enter into command mode by using <Esc> key.

3) Exit Mode:

To quit from the editor.
From the command mode we have to press: then we can enter into exit mode.

How to Insert and Append Data:

A To Append data at the end of the line
I To Insert data at the beginning of the line
a To append data to the right side of the cursor position (Just after the cursor position)
i To insert data to the left side of the cursor position (Just before the cursor position

How to Delete Data:

We can delete data either by character wise or by word wise or by line wise.
deletion character wise:
x To delete a single character (del key)
3x To delete 3 characters. Instead of 3, we can pass any number.
X To delete previous character (backspace key)
3X To delete last 3 previous characters

Deletion Words wise:

dw To delete current word.
3dw To delete 3 words

Deletion Line wise:

dd To delete current line
3dd To delete 3 lines
d$ Deletes from current position to end of line.
d^ Deletes from current position to beginning of the line.
dgg Deletes from beginning of the file to current cursor position.
Dg Deletes from current position to end of file.

How to Replace Data:

r Replace current character.
R To replace multiple characters from the current position.
S OR cc To replace a single line

Opening New Lines to Insert Data:

O To open a line above the cursor position.(i.e before current line)
o To open a line below the cursor position (i.e after current line)

Copy and Paste Data:

yy To Copy a Line (yanking)
3yy To copy 3 lines
yw To copy a word
3yw To copy 3 words
y$ To copy from current cursor position to end of line.
y^ To copy from beginning of the line to current cursor position.
p Paste above the cursor position
P Paste below the cursor position

Cursor Navigation Commands:

k Top Arrow
j Bottom Arrow
l Right Arrow
h Left Arrow
3k 3 Times Top Arrow
3j 3 Times Bottom Arrow
3l 3 Times Right Arrow
3h 3 Times Left Arrow
$ End of the Current Line (End Key)
^ Beginning of the Current Line (Home Key)
H Beginning of the Current Page
M Middle of the Current Page
L End of the Current Page

G Move to last line (ie end of the file)
at exit mode
:1 To go to 1
st line
:7 To go to 7th line
ctrl+f One Page Forward (Page Down)
ctrl+b One Page Backward(Page Up)
Note: If we want to perform undo previous operation then we should use 'u'
u Means undo previous operation

Exit Mode Commands:

:w Save File Data
:wq Save and Quit from the Editor
:q Quit Editor

:q! → Force Quit. If we perform any changes those will be discarded.
:set nu → To set line numbers in the editor
:set nonu → To remove line numbers
:n → Place the cursor to the nth line
:$ → Place the cursor to the last line
:!<unix_command> → To execute any command

In command mode
:r!command

```
execute and save command output
inside the file

/home/jitendra/class
```

~
~
~
~

---------- classroom ppt content same as above ----------

Vi (visual editor) editor

- It is default editor that comes with the UNIX OS.
- Using vi editor, we can edit an existing file or create a new file from scratch and read a file.
- **Syntax : $vi filename**
- Eg :

- **Modes of Operation in vi editor**

- There are three modes of operation in vi:

- Command mode

- Insert mode

- Last Line Mode(Escape Mode)

1. Command Mode:-

When vi starts up, it is in Command Mode.
This mode is where vi interprets any characters we type as commands
and thus does not display them in the window.

This mode allows us to move through a file, and to delete, copy, or paste a piece of text.

To enter into Command Mode from any other mode, it requires pressing the [Esc] key.

If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

2. Insert mode:-

This mode enables you to insert text into the file.

Everything that's typed in this mode is interpreted as input and finally, it is put in the file.

To enter text, you must be in insert mode.

To come in insert mode you simply type i.

To get out of insert mode, press the Esc key, which will put you back into command mode.

3. Last Line Mode(Escape Mode):-

Line Mode is invoked by typing a colon [:], while vi is in Command Mode.

The cursor will jump to the last line of the screen and vi will wait for a command.

This mode enables you to perform tasks such as saving files, executing commands.

cmd : $vi filename
Eg :
Opens an existing file in read only mode.

cmd : $vi -R filename
Eg :
Opens an existing file in read only mode

cmd : $view filename
Eg :

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 switch from command to insert mode in vi**

Command	Action
i	Start typing before the current character
I	Start typing at the start of current line
a	Start typing after the current character
A	Start typing at the end of current line
o	Start typing on a new line after the current line
O	Start typing on a new line before the current line

- **To move around a file in vi**

Commands	Action
j	To move down
k	To move up
h	To move left
l	To move right

- **To jump lines in vi**

Commands	Action
G	Will direct you at the last line of the file
``	Will direct you to your last position in the file

To repeat and undo vi

Commands	Action
u	Undo the last command
.	Repeat the last command

- **To delete in vi**

Commands	Action
x	Delete the current character
X	Delete the character before the cursor
r	Replace the current character
xp	Switch two characters
dd	Delete the current line
D	Delete the current line from current character to the end of the line
dG	delete from the current line to the end of the file

- **Command to cut, copy and paste in vi**

Commands	Action
dd	Delete a line
yy	(yank yank) copy a line
p	Paste after the current line
P	Paste before the current line

Command to cut, copy and paste in vi

Commands	Action
<n>dd	Delete the specified n number of lines
<n>yy	Copy the specified n number of lines

- **Start and end of line in vi**

Commands	Action
0	Bring at the start of the current line
^	Bring at the start of the current line
$	Bring at the end of the current line
d0	Delete till start of a line
d$	Delete till end of a line

Joining lines in vi

Commands	Action
J	Join two lines
yyp	Repeat the current line
ddp	Swap two lines

- **Move forward or backward in vi**

Commands	Action
w	Move one word forward
b	Move one word backward
<n>w	Move specified number of words forward
dw	Delete one word
yw	Copy one word
<n>dw	Delete specified number of words

- **Search a string in vi**

Commands	Action
/string	Forward search for given string
?string	Backward search for given string
/^string	Forward search string at beginning of a line
/string$	Forward search string at end of a line
n	Go to next occurrence of searched string
/\<he\>	Search for the word he (and not for there, here, etc.)
/pl[abc]ce	Search for place, plbce, and plcce

- To write into file in vim – Press **i**
- To save and Exit – Press **Esc** key then **:wq!**
- To exit without saving the file - **:q!**
- To To quit and save changes - Press the **Esc** key then **:wq**

40. nano :-

It is command line editor.
It can be used to create new files and edit content of existing files

ctrl+g (F1) Display this help text

ctrl+x (F2) Close the current file buffer / Exit from nano

ctrl+o (F3) Write the current file to disk

ctrl+r (F5) Insert another file into the current one

ctrl+w (F6) Search forward for a string or a regular expression

ctrl+\ (M-R) Replace a string or a regular expression

ctrl+k (F9) Cut the current line and store it in the cutbuffer

ctrl+u (F10) Uncut from the cutbuffer into the current line

But main important options:
ctrl+o To save content
ctrl+x To quit from the editor

---------- commands ----------

41. Cd :-

1) Every directory implicitly contains 2 directories . and ..
. Represents Current Directory
.. Represents Parent Directory

2) $ cd .
Changes to Current Directory (Useless)

3) $ cd ..
Changes to Parent Directory

4) $cd
If we are not passing any argument, then changes to user home directory.

5) $ cd ~
~ Means User Home Directory.
It will Changes to User Home Directory.

6) $ cd -
- Means Previous Working Directory.
It will Changes to Previous Working Directory.

42. Pwd :-

pwd stands for Print Working Directory.
It prints the path of the working directory, starting from the root.

43. History :-

history command is used to view the previously executed command

To show the limited number of commands that executed previously as follows:
$ history 5

```
jitendra@DESKTOP-ACUC1TV:~/class$ history 5
 2003  vi file1
 2004  ls
 2005  nano file1
 2006  history
 2007  history 5
```

- The command can be executed using event number also.
  $!1997

- To print command before executing so that a wrong command
- does not get executed use :p after event number of command.
  Example:

  !1997:p

- History can also be removed using history -d event_number
  Example:

  history -d 1996

- The most recent command can be viewed using !!
  Example:

  !!

- The whole history can be removed using history -c option.
  Example:

  history -c

44. uname :-

The command ‘uname‘ displays the information about the system.

```
jitendra@DESKTOP-ACUC1TV:~$ uname
Linux
jitendra@DESKTOP-ACUC1TV:~$ uname -a
Linux DESKTOP-ACUC1TV 5.15.79.1-microsoft-st
andard-WSL2 #1 SMP Wed Nov 23 01:01:46 UTC 2
022 x86_64 x86_64 x86_64 GNU/Linux
jitendra@DESKTOP-ACUC1TV:~$ uname -s
Linux
jitendra@DESKTOP-ACUC1TV:~$ uname -o
GNU/Linux
jitendra@DESKTOP-ACUC1TV:~$ uname -p
x86_64
jitendra@DESKTOP-ACUC1TV:~$ uname -v
#1 SMP Wed Nov 23 01:01:46 UTC 2022
jitendra@DESKTOP-ACUC1TV:~$ uname -n
DESKTOP-ACUC1TV
jitendra@DESKTOP-ACUC1TV:~$
```

Options :-

-v = current kernel version
-n = hostname
-s = kernel name
-r = release version
-o = os name

-i = hardware platform
-m = machine hardware name
-p = processor type

```
jitendra@pc:~$ uname -i
x86_64
jitendra@pc:~$ uname -m
x86_64
jitendra@pc:~$ uname -p
x86_64
```

45. logname :-

logname - print user's login name

Hostname :-

to display or set the system's hostname
$hostname
myhostname

hostname -i -----> show ip address of hostname (only primay network interface)

hostname -I -----> list all IP addresses associated with the system

hostname -s -----> short hostname

ex:- for myhost.example.com short hostname will be **myhost**

hostname -f -----> to display the fully qualified domain name (FQDN)
ex :- myhostname.example.com

hostname -d -----> to display the domain name of system
ex:- example.com

hostname -a -----> to display all the alias of current hostname

which :-

The which command allows users to search the list of paths in the $PATH environment variable and outputs the full path of the command specified as an argument. The command works by locating the executable file matching the given command

```
jitendra@pc:~$ which cat
/usr/bin/cat
jitendra@pc:~$ which -a cat
/usr/bin/cat
/bin/cat
jitendra@pc:~$ which cat touch tr
/usr/bin/cat
/usr/bin/touch
/usr/bin/tr
```

cat command is actually is in /usr/bin/cat
/bin/cat is a link to previous folder

46. banner :-

prints the given string in banner form

```
jitendra@DESKTOP-ACUC1TV:~/class$ banner Ajgar
    #
   # #           #    ####      ##     #####
  #    #         #   #    #    #  #    #    #
 #      #        #   #        #    #   #    #
 #######         #   #  ###   ######   #####
 #      #   #    #   #    #   #    #   #   #
 #      #    ####     ####    #    #   #    #
```

```
jitendra@DESKTOP-ACUC1TV:~/class$ banner c,linux
  ####           #           #    #    #  #    #  #    #
 #    #          #           #    ##   #  #    #   #  #
 #        ###    #           #    # #  #  #    #    ##
 #        ###    #           #    #  # #  #    #    ##
 #    #    #     #           #    #   ##  #    #   #  #
  ####    #      ######      #    #    #   ####   #    #
```

47. free :-

The free command gives information about used and unused memory
usage and swap memory of a system. By default, it displays memory in kb (kilobytes).

Memory mainly consists of RAM (random access memory) and swap memory.

Swap memory is a part of hard disk drive that acts like a virtual RAM.

```
jitendra@DESKTOP-ACUC1TV:~/class$ free
               total        used        free      shared  buff/cache   available
Mem:         3446760      335864     2926364        2520      184532     2960868
Swap:        1048576           0     1048576
```

free -b ------> display information in Bytes
free -m ------> display information in Megabytes
free -g ------> display information in Gigabytes
free -h ------> human readable form
free -t ------> It adds an additional line in the output showing the column totals.

48. df :-

The df command stands for **disk free**, and it shows you the amount of space taken up by different drives. By default, df displays values in 1-kilobyte blocks.

df -h -----> human readable form
df -Bg ------> block size in gb
df -Bm -----> block size in mb

```
Filesystem      Size      Used     Avail    Use%    Mounted on

udev            210M         0       210M     0%       /dev

tmpfs           49M       1004K      48M      3%       /run

/dev/sda2       7.9G       4.3G      3.2G     58%      /
```

Your output may have more entries. The columns should be self-explanatory:

- **Filesystem** – This is the name of each particular drive. This includes physical hard drives, logical (partitioned) drives, and virtual or temporary drives.
- **Size** – The size of the filesystem.
- **Used –** Amount of space used on each filesystem.
- **Avail –** The amount of unused (free) space on the filesystem.
- **Use% –** Shows the percent of the disk used.
- **Mounted on –** This is the directory where the file system is located. This is also sometimes called a **mount point**.

```
jitendra@DESKTOP-ACUC1TV:~/class$ df -Bg
Filesystem     1G-blocks  Used Available Use% Mounted on
none                2G    1G         2G   1% /mnt/wsl
drivers           224G  138G        87G  62% /usr/lib/wsl/drivers
none                2G    0G         2G   0% /usr/lib/wsl/lib
/dev/sdc          251G    3G       236G   1% /
none                2G    1G         2G   1% /mnt/wslg
rootfs              2G    1G         2G   1% /init
none                2G    0G         2G   0% /run
none                2G    0G         2G   0% /run/lock
none                2G    0G         2G   0% /run/shm
none                2G    0G         2G   0% /run/user
tmpfs               2G    0G         2G   0% /sys/fs/cgroup
none                2G    1G         2G   1% /mnt/wslg/versions.txt
none                2G    1G         2G   1% /mnt/wslg/doc
drvfs             224G  138G        87G  62% /mnt/c
drvfs             931G  538G       394G  58% /mnt/d
```

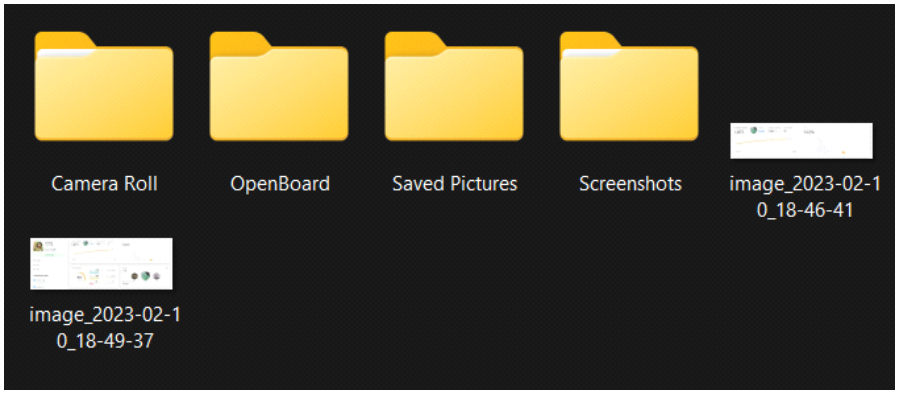
49. du :-

Command du stands for Disk Usage. It is used to check the information of disk usage of files and directories on a system.

Command du display a list of all the files along with their respective sizes. By default, size given is in kilobytes.

File names are used as arguments to get the file size.

du ---> show disk usage of current directory
du file/directory ---> show disk usage of perticular file or folder
du -a -----> for all files and directory
du -h -----> human readable form
du -Bg ------> block size in gb
du -Bm -----> block size in mb



```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du
3088    ./Camera Roll
0       ./OpenBoard
0       ./Saved Pictures
2092    ./Screenshots
5344    .
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du image2
du: cannot access 'image2': No such file or directory
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du image2.jpg
du: cannot access 'image2.jpg': No such file or directory
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ clear -x
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du
3088    ./Camera Roll
0       ./OpenBoard
0       ./Saved Pictures
2092    ./Screenshots
5344    .
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du image2.png
136     image2.png
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du -h
3.1M    ./Camera Roll
0       ./OpenBoard
0       ./Saved Pictures
2.1M    ./Screenshots
5.3M    .
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du -Bg
1G      ./Camera Roll
0G      ./OpenBoard
0G      ./Saved Pictures
1G      ./Screenshots
1G      .
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ du -a
0       ./Camera Roll/desktop.ini
3088    ./Camera Roll/WIN_20221107_10_35_09_Pro.mp4
3088    ./Camera Roll
0       ./desktop.ini
136     ./image2.png
28      ./image_2023-02-10_18-46-41.png
0       ./OpenBoard
0       ./Saved Pictures/desktop.ini
0       ./Saved Pictures
0       ./Screenshots/desktop.ini
160     ./Screenshots/photo.png
168     ./Screenshots/Screenshot (2).png
156     ./Screenshots/Screenshot (3).png
112     ./Screenshots/Screenshot (4).png
164     ./Screenshots/Screenshot (5).png
848     ./Screenshots/Screenshot (6).png
392     ./Screenshots/Screenshot_20221128_150313.png
28      ./Screenshots/Screenshot_20221203_160610.png
```

50. bc :-

basic calculator, will do calculation on basic math expression
it will take standard input and give calculation output on standard output

```
jitendra@DESKTOP-ACUC1TV:~$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004,
 2006, 2008, 2012-2017 Free Software Foundat
ion, Inc.
This is free software with ABSOLUTELY NO WAR
RANTY.
For details type `warranty'.
1+2
3
1*4+10/4
6

jitendra@DESKTOP-ACUC1TV:~$ cat>file
1+3*56
jitendra@DESKTOP-ACUC1TV:~$ bc<file
169
```

-q ----> will not print welcome message

-l, --mathlib
Define the standard math library more accurate better options

```
jitendra@PC:~$ bc -q
sqrt(5)
2

jitendra@PC:~$ bc -q -l
sqrt(5)
2.23606797749978969640
```

-i ----> to convert base

```
jitendra@PC:~$ bc -i -q
ibase=2
111
7
101
5
11111
31
```

using scale command with bc

```
scale=2
sqrt(5)
2.23
scale=10
sqrt(5)
2.2360679774
```

to print at certain decimal values

51. stat :-

The stat is a command which gives information about the file and filesystem.
Stat command gives information such as the size of the file,
access permissions and the user ID and group ID,
birth time access time of the file.

```
jitendra@DESKTOP-ACUC1TV:~$ stat file
  File: file
  Size: 7               Blocks: 8          IO Block: 4096   regular file
Device: 820h/2080d      Inode: 29284       Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/jitendra)   Gid: ( 1000/jitendra)
Access: 2023-02-12 14:16:12.297575096 +0530
Modify: 2023-02-12 14:16:07.827576670 +0530
Change: 2023-02-12 14:16:07.827576670 +0530
 Birth: 2023-02-12 14:16:02.547578543 +0530
```

```
jitendra@DESKTOP-ACUC1TV:~/class$ touch -t 201212121212 file1
jitendra@DESKTOP-ACUC1TV:~/class$ stat file1  File: file1
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 820h/2080d      Inode: 27014       Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/jitendra)   Gid: ( 1000/jitendra)
Access: 2012-12-12 12:12:00.000000000 +0530
Modify: 2012-12-12 12:12:00.000000000 +0530
Change: 2023-02-12 14:42:17.847017531 +0530
 Birth: 2023-02-12 14:26:28.677353409 +0530
```

52. Whereis :-

.whereis is used to find the path of the Linux binary /executable files,
source files and man page files.There are many Linux distributions.
Not every Linux distribution keeps the binary/executable files, source files and man page files in the same location.
So, to find out the path of these files when needed, the whereis command is used.

-m ---> only mannual location
-b ----> only binary location

```
jitendra@DESKTOP-ACUC1TV:/home$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
jitendra@DESKTOP-ACUC1TV:/home$ whereis -b ls
ls: /usr/bin/ls
jitendra@DESKTOP-ACUC1TV:/home$ whereis -m ls
ls: /usr/share/man/man1/ls.1.gz
```

53. Whatis :-

. The whatis command is used to get brief information about Linux commands or functions.
It displays the manual page description in a single line

```
jitendra@DESKTOP-ACUC1TV:/home$ whatis ls
ls (1)               - list directory contents
jitendra@DESKTOP-ACUC1TV:/home$ whatis touch
touch (1)            - change file timestamps
jitendra@DESKTOP-ACUC1TV:/home$ whatis rm
rm (1)               - remove files or directories
jitendra@DESKTOP-ACUC1TV:/home$ whatis ping
ping (8)             - send ICMP ECHO_REQUEST to network hosts
jitendra@DESKTOP-ACUC1TV:/home$ whatis lscpu
lscpu (1)            - display information about the CPU architecture
jitendra@DESKTOP-ACUC1TV:/home$
```

-w ----> if we want to search for **wildcard**

-r ------> if we want to search for regex

```
jitendra@DESKTOP-ACUC1TV:/home$ whatis -w rm*
rm (1)               - remove files or directories
rmdir (1)            - remove empty directories
rmdir (2)            - delete a directory
rmmod (8)            - Simple program to remove a module from the Linux Kernel
rmt (8)              - remote magnetic tape server
rmt-tar (8)          - remote magnetic tape server
jitendra@DESKTOP-ACUC1TV:/home$ whatis -r ^rm
rm (1)               - remove files or directories
rmdir (1)            - remove empty directories
rmdir (2)            - delete a directory
rmmod (8)            - Simple program to remove a module from the Linux Kernel
rmt (8)              - remote magnetic tape server
rmt-tar (8)          - remote magnetic tape server
```

-l ----> output in long form , don't trim

```
jitendra@DESKTOP-ACUC1TV:/home$ whatis -l ssh-import-id
ssh-import-id (1)    - retrieve one or more public keys from a public keyserver and append
 them to the current user's authorized_keys file (or some other specified file)
jitendra@DESKTOP-ACUC1TV:/home$ whatis  ssh-import-id
ssh-import-id (1)    - retrieve one or more public keys from a public keyserver and app...
```

By default linux trim's the output if it is long

54. reboot :-

This command will reboot your system
Same as shutdown -r

55. shutdown :-

shutdown 05:00 --> schedule shutdown at 5 am

Shutdown +10 "system upgrade" ---> The following command will
shut down the system
in 10 minutes from now and notify the users with message “System upgrade”

sudo shutdown -c "Canceling the reboot" ----> When canceling a scheduled
shutdown,
you cannot specify a time argument, but you can still broadcast a
message that will be sent to all users.

56. uptime:-

Uptime is a command that returns information about how long your
system has been running
together with the current time,

number of users with running sessions,
and the system load averages for the past 1, 5, and 15 minutes.

-p -----> pretty format
-s ------> since

15:18:28 ----> current time of day
up 1:54 ----> the system has been up for 1 hour 54 minutes
0 users ----> number of logged in users (0 due to wsl)
load average 0.00 0.00 0.00 -----> The system load average over the
last 1, 5, and 15 minutes, respectively.

```
jitendra@DESKTOP-ACUC1TV:/home$ uptime
 15:18:04 up  1:54,  0 users,  load average: 0.00, 0.00, 0.00
jitendra@DESKTOP-ACUC1TV:/home$ uptime -s
2023-02-12 13:23:18
jitendra@DESKTOP-ACUC1TV:/home$ uptime -p
up 1 hour, 54 minutes
```

57. su :-

to switch user in linux

```
jitendra@DESKTOP-ACUC1TV:/home$ su
Password:
root@DESKTOP-ACUC1TV:/home# su joker
joker@DESKTOP-ACUC1TV:/home$ su jitendra
Password:
jitendra@DESKTOP-ACUC1TV:/home$
```

if no user is given it will switch to root (root password must be set for that)

su username ----> to switch to any other user

58. sudo :-

sudo (Super User DO) command in Linux is generally used as a prefix
of some command that only superuser are allowed to run.
If you prefix “sudo” with any command, it will run that command
with elevated privileges or in other words allow a user with proper
permissions to execute a command as another user, such as the superuser.

```
jitendra@DESKTOP-ACUC1TV:~$ useradd kk
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
jitendra@DESKTOP-ACUC1TV:~$ sudo useradd kk
jitendra@DESKTOP-ACUC1TV:~$ cat /etc/passwd
```

This is the equivalent of “run as administrator” option in Windows.
The option of sudo lets us have multiple administrators.

These users who can use the sudo command need to have an entry in the sudoers file located at “/etc/sudoers”.

Remember that to edit or view the sudoers file you have to use sudo command. To edit the sudoers file it is recommended to use “visudo” command.

By default, sudo requires that users authenticate themselves with a password which is the user’s password, not the root password itself.

If we want to add user for sudoers permissions
Run following command :- sudo -aG sudo joker (user joker will be added to sudo group and now have permission to use sudo command)

59. Lscpu :-

lscpu - display information about the CPU architecture

```
jitendra@DESKTOP-ACUC1TV:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx
    CPU family:          23
    Model:               17
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            0
    BogoMIPS:            3992.49
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mt
                          pse36 clflush mmx fxsr sse sse2 ht syscall nx
                         pe1gb rdtscp lm constant_tsc rep_good nopl tsc
                         sc cpuid extd_apicid pni pclmulqdq ssse3 fma c
                         ovbe popcnt aes xsave avx f16c rdrand hypervis
                         cy svm cr8_legacy abm sse4a misalignsse 3dnowp
```

60. tty :-

Linux operating system represents everything in a file system,
the hardware devices that we attach are also represented as a file.
The terminal is also represented as a file.
There a command exists called tty which displays information related to terminal.
The tty command of terminal basically prints the file name of the terminal

```
jitendra@DESKTOP-ACUC1TV:~$ tty
/dev/pts/0
```

61. Ulimit :- this command is used to see and set the limit for various resource usage

Of the current user in the system

```
jitendra@DESKTOP-ACUC1TV:~$ ulimit
unlimited
jitendra@DESKTOP-ACUC1TV:~$ ulimit -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 13438
max locked memory           (kbytes, -l) 65536
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 8192
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 13438
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
jitendra@DESKTOP-ACUC1TV:~$ ulimit -s 3333
```

```
jitendra@DESKTOP-ACUC1TV:~$ ulimit -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 13438
max locked memory           (kbytes, -l) 65536
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 3333
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 13438
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
jitendra@DESKTOP-ACUC1TV:~$
```

here we are limiting the stack size using ulimit command

62. Ping :-

PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host.

--->This command takes as input
the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.

---->Fast ping
low latency means faster connection. Ping uses ICMP(Internet Control Message Protocol) to send an ICMP echo message to the specified host if that host is available then it sends ICMP reply message.

----> Ping is generally measured in millisecond every modern operating system has this ping pre-installed.

```
jitendra@DESKTOP-ACUC1TV:~$ ping google.com
PING google.com (216.239.32.10) 56(84) bytes of data.
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=1 ttl=108 time=87.2 ms
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=2 ttl=108 time=88.2 ms
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=3 ttl=108 time=98.8 ms
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=4 ttl=108 time=89.7 ms
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=5 ttl=108 time=158 ms
64 bytes from ns1.google.com (216.239.32.10): icmp_seq=6 ttl=108 time=137 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 87.220/109.826/157.737/27.571 ms
```

here **rtt ---> round trip time**

ping 0: It is one of the quickest option to ping a localhost.
The terminal will resolve determine the IP address and gives a response once we enter this command.

ping localhost: We can use the ping localhost name.
This name will refer to our system and when we enter this command, we will say "ping this system".

ping -c ----> specifies the number of ICMP echo requests to send to the target host before stopping
the ping process.
ex:- ping -c 5 google.com will send 5 ICMP requests and then stop

```
jitendra@PC:~$ ping -c 4 bing.com
PING bing.com (13.107.21.200) 56(84) bytes of data.
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=1 ttl=111 time=173 ms
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=2 ttl=111 time=159 ms
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=3 ttl=111 time=166 ms
64 bytes from 13.107.21.200: icmp_seq=4 ttl=111 time=377 ms

--- bing.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3848ms
rtt min/avg/max/mdev = 158.823/218.647/376.550/91.309 ms
jitendra@PC:~$
```

this time we don't need to mannualy cancel the requests (crtl+c) it automatically stopped after 4 requests

ping -s -----> specifies the size of the ICMP echo request packet in bytes
. For example, ping -s 100 google.com will send an ICMP echo request packet of 100 bytes to google.com. default is 56 bytes

```
jitendra@PC:~$ ping -s 100 google.com
PING google.com (142.250.183.174) 100(128) bytes of data.
76 bytes from bom07s32-in-f14.1e100.net (142.250.183.174): icmp_seq=1 ttl=54 (truncated)
76 bytes from bom07s32-in-f14.1e100.net (142.250.183.174): icmp_seq=2 ttl=54 (truncated)
76 bytes from bom07s32-in-f14.1e100.net (142.250.183.174): icmp_seq=3 ttl=54 (truncated)
76 bytes from bom07s32-in-f14.1e100.net (142.250.183.174): icmp_seq=4 ttl=54 (truncated)
76 bytes from bom07s32-in-f14.1e100.net (142.250.183.174): icmp_seq=5 ttl=54 (truncated)
^C
--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 97.182/106.365/123.133/9.426 ms
```

"truncated" means that the ICMP echo request packet has been cut off or shortened due to some network configuration or issue

ping -i -----> This option sets the time interval between sending packets. By default , it's set to one second. For example, the following command will send packets every five seconds
ping -i 5 google.com

ping -t ------> This option sets the Time-to-Live (TTL) value of the packet, which determines the number of network hops that the packet can take before being discarded. By default, the TTL value is set to 64. For example, the following command will set the TTL value to 128:
ex:- ping -t 128 google.com

```
jitendra@PC:~$ ping -t 128 bing.com
PING bing.com (13.107.21.200) 56(84) bytes of data.
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=1 ttl=111 time=438 ms
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=2 ttl=111 time=477 ms
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=3 ttl=111 time=172 ms
64 bytes from 13.107.21.200 (13.107.21.200): icmp_seq=4 ttl=111 time=171 ms
^C64 bytes from 13.107.21.200: icmp_seq=5 ttl=111 time=392 ms

--- bing.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 5438ms
rtt min/avg/max/mdev = 171.392/330.168/476.828/131.950 ms
```

the actual ttl value may be greater or lesser than what you specified, ttl value decrease with each hop
and intermediate router can add additional ttl values

63. nl :-

To add numbering to the lines in a file
-b a ----> add numbering to empty lines also
-I 2 ----> increment is now 2
-s "..." ----> custom separator between number and text

```
jitendra@DESKTOP-ACUC1TV:~$ nl file
     1  ab
     2  cd
     3  ef

     4  gh
jitendra@DESKTOP-ACUC1TV:~$ nl -b a file
     1  ab
     2  cd
     3  ef
     4
     5  gh
jitendra@DESKTOP-ACUC1TV:~$ nl -i 5 file
     1  ab
     6  cd
    11  ef

    16  gh
jitendra@DESKTOP-ACUC1TV:~$ nl -s "--->" file
     1--->ab
     2--->cd
```

64. id :-

id command in Linux is used to find out user and group names and
numeric ID's (UID or group ID) of the current user or any other user in the server
-g : Print only the effective group id.
-G : Print all Group ID's.
-n : Prints name instead of number.
-r : Prints real ID instead of numbers.
-u : Prints only the effective user ID.

```
jitendra@DESKTOP-ACUC1TV:~$ id
uid=1000(jitendra) gid=1000(jitendra) groups=1000(jitendra),4(adm),20(dialout),24(cdrom),2
5(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),116(netdev)
jitendra@DESKTOP-ACUC1TV:~$ id -u
1000
jitendra@DESKTOP-ACUC1TV:~$ id -un
jitendra
jitendra@DESKTOP-ACUC1TV:~$ id -g
1000
jitendra@DESKTOP-ACUC1TV:~$ id -gn
jitendra
jitendra@DESKTOP-ACUC1TV:~$ id joker
uid=1003(joker) gid=1018(joker) groups=1018(joker),27(sudo),2223(dc),3333(Marval)
jitendra@DESKTOP-ACUC1TV:~$
```

65. file :-

file command is used to determine the type of a file

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Desktop/Code/C Programming Chitkara/Lecture$ file image2.png
image2.png: PNG image data, 1622 x 584, 8-bit/color RGB, non-interlaced
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Desktop/Code/C Programming Chitkara/Lecture$ file hello.c
hello.c: C source, ASCII text, with CRLF line terminators
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Desktop/Code/C Programming Chitkara/Lecture$ ls
Lecture.code-workspace  a.exe  a.out  hello  hello.c  hello.exe  image2.png  temp.c  temp.exe  tempCodeRunnerFile.c
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Desktop/Code/C Programming Chitkara/Lecture$
```

66. sed :-

We can use sed command to retrieve and edit data present in the given file.
We can perform operations based on line like delete 2nd line etc.
We can also perform operations based on context like delete lines where a specific word is present etc

Sed option expression filename

Options :-

i ------ > in place (make actual changes in file instead of printing)
e ------ > for multiple expressions

a. print nth line 2 times

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed '3p' file1
enum|ename|esal|eaddr|dept|genter
100|jitendra|15l|bikaner|cse|male
102|kunal|12l|delhi|cse|male
102|kunal|12l|delhi|cse|male
129|shivani|6l|delhi|arts|female
```

b) Display only specific Line:

$ sed -n '3p' file1
It will display only 3rd line

c) Display Last Line:
$ sed -n '$p' file1
$ means last line

d) Display Multiple Lines in Range:
$ sed -n '2,4p' file1
it will display from 2nd to 4th lines.

e) display all line except specific line:

$ sed -n '2p' file1
It will display only 2nd line

$sed -n '2!p' file1
It will display all lines except 2nd line

f) Display all Lines except Range of Lines:

$sed -n '2,4p' file1
It will display all lines from 2nd to 4th.

$sed -n '2,4!p' file1
It will display all lines except 2 to 4.

g)all lines having specific word
$ sed -n '/kunal/p' file1
It will display all lines which contains admin.

h) deleting all lines with specific word
$ sed '3d' file1
d means deletion
While displaying records, it will delete 3rd record. But this line won't be deleted in the file

i) to delete permanently in the file
$ sed -i '5d' file

-i meant for deleting record permanently in the file.
This command won't display anything to the console but 5th record in the file will be deleted.

j) To Delete Range of Lines:
$ sed -i '1,$d' file1
It will delete all records from 1st to last in the file. Now file will become empty. (To delete range of records)

k) to replace a word in a file

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed 's/bikaner/rajasthan/' file1
enum|ename|esal|eaddr|dept|genter
100|jitendra|15l|rajasthan|cse|male
102|kunal|12l|delhi|cse|male
129|shivani|6l|delhi|arts|female
```

we have to use s option
s means **substitute**

l) ignore case while replacing

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed 's/MALE/female/' file1
enum|ename|esal|eaddr|dept|genter
100|jitendra|15l|bikaner|cse|male
102|kunal|12l|delhi|cse|male
129|shivani|6l|delhi|arts|female
jitendra@DESKTOP-ACUC1TV:~/class$ sed 's/MALE/female/i' file1
enum|ename|esal|eaddr|dept|genter
100|jitendra|15l|bikaner|cse|female
102|kunal|12l|delhi|cse|female
129|shivani|6l|delhi|arts|fefemale
```

m) Using multiple expression in sed using -e option

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed -n -e '/kunal/p' -e '/jitendra/p' file
jitendra dsa cse cb
kunal c ece gfg
jitendra@DESKTOP-ACUC1TV:~/class$
```

n) Printing two more lines after 2nd line or after matching the pattern

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed -n '2,+2p' file
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
jitendra@DESKTOP-ACUC1TV:~/class$ sed -n ' '/jitendra/',+2p' file
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
```

o) Deleting n number of lines after matching the pattern

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed  ' '/jitendra/',+2d' file
name sub dept origin
vikas java cse scaler
```

p) Changing the subject from c to c++ where origin in cn

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed  '/cn/ s/c/c++/' file
name sub dept origin
jitendra dsa cse cb
rajat c++ it cn
kunal c ece gfg
vikas java cse scaler
```

q) Show 2 more lines after matching pattern

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed -n '/cn/,+2p' file
rajat c it cn
kunal c ece gfg
vikas java cse scaler
```

r) Add a new file after a perticular pattern or line number

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed '/rajat/a vishal c++ ee cb' file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
vishal c++ ee cb
kunal c ece gfg
vikas java cse scaler
```

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed '3a vishal c++ ee cb' file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
vishal c++ ee cb
kunal c ece gfg
vikas java cse scaler
```

s) Inserting line before a pattern

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed '/rajat/i vishal c++ ee cb' file
name sub dept origin
jitendra dsa cse cb
vishal c++ ee cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
```

t) Using regular expression with sed

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file
name sub dept origin
jitendra dsa cse cb
rajat c it cn
kunal c ece gfg
vikas java cse scaler
jitendra@DESKTOP-ACUC1TV:~/class$ sed -n '/gfg$/p' file
kunal c ece gfg
```

some examples

$ sed -i 's/^$/this is a blank line/' demo.txt
It replaces every blank line with 'this was a blank line'

```
jitendra@DESKTOP-ACUC1TV:~/class$ sed 's/^$/this was a blank line/' file1
this is a text
this was a blank line
file
this was a blank line
i
this was a blank line
hate this
jitendra@DESKTOP-ACUC1TV:~/class$ cat file1
this is a text

file

i

hate this
```

67. awk :-

We can use awk commands to search in the files and print results in our required format.
AWK provides more convinient way for file processing than our regular linux commands.
Most commonly used in shell scripting if we want to handle very large files.

Structure of awk Command:

awk {BEGIN BLOCK}{ACTION BLOCK}{END BLOCK} filenames
BEGIN block will be executed only once before processing the file.
ACTION block will be executed for every line/record present in the file.
END block will be executed only once after completing all lines/records processing.

Predefined Variables of awk:

FS Field Separator
NR Row Number / Number of Rows
NF Number of Fields

RS ? Record Separator

a) to print any column

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat file1
name  rollno   age
jitendra   2323 24
kunal 333 25
rajat 234 43
kartik 211 434
vikas 231 26
jitendra@DESKTOP-ACUC1TV:~/class$ cut -f1 file1
name  rollno   age
jitendra   2323 24
kunal 333 25
rajat 234 43
kartik 211 434
vikas 231 26
jitendra@DESKTOP-ACUC1TV:~/class$ awk '{print $1 }' file1
name
jitendra
kunal
rajat
kartik
vikas
jitendra@DESKTOP-ACUC1TV:~/class$
```

b) to print multiple columns with custom seperator

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk '{print $1"-->"$2 }' file1
name-->rollno
jitendra-->2323
kunal-->333
rajat-->234
kartik-->211
vikas-->231
```

c) print all columns

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk '{print $0 }' file1
name  rollno   age
jitendra   2323 24
kunal 333 25
rajat 234 43
kartik 211 434
vikas 231 26
```

d) custom deliminator

using -F option

```
jitendra@DESKTOP-ACUC1TV:~/class$ cat>file1
name|rollno|age
jitendra|233|23
kunal|222|44
shivank|555|11
satyam|289|23
rajat|23|344
jitendra@DESKTOP-ACUC1TV:~/class$ awk -F'|' '{print $1"--->"$2}' file1
name--->rollno
jitendra--->233
kunal--->222
shivank--->555
satyam--->289
rajat--->23
```

using FS option

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk  '{print $1"--->"$2}' FS='|' file1
name--->rollno
jitendra--->233
kunal--->222
shivank--->555
satyam--->289
rajat--->23
```

e) all rows except first

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk -F'|'  ' NR!=1{print $1"--->"$2}' file1
jitendra--->233
kunal--->222
shivank--->555
satyam--->289
rajat--->23
```

f) showing only 1st row

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk -F'|'  ' NR==1{print $1"--->"$2}' file1
name--->rollno
```

g) row number is greater than

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk -F'|'  ' NR>2{print $1"--->"$2}' file1
kunal--->222
shivank--->555
satyam--->289
rajat--->23
```

we can use all relational and logical operators with NR

awk 'NR==2 || NR==4 {print $0}' FS="," file1.txt

It will display only 2nd and 4th records

awk 'NR==2 || NR==4 {print $1}' FS="," file1.txt
It will display only 2nd and 4th records first field value.

awk 'NR>2 && NR<5 {print $0}' FS="," file1.txt
It will display only 3rd and 4th records

awk 'NR>1 && NR<5 {print $0}' FS="," file1.txt
It will display from 2nd record to 4th record.

awk 'NR==2,NR==4{print $0}' FS=',' file1.txt
It will display from 2nd record to 4th record.

Note: awk will always treat every file as tabular file only.

h) print all employees who are older than 23

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk '$3>23{print $0}' FS='|' file1
name|rollno|age
kunal|222|44
rajat|23|344
```

to not display the header

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk '$3>23 && NR!=1 {print $0}' FS='|' file1
kunal|222|44
rajat|23|344
jitendra@DESKTOP-ACUC1TV:~/class$
```

i) search pattern in file

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk '/s/{print $0}' FS='|' file1
shivank|555|11
satyam|289|23
jitendra@DESKTOP-ACUC1TV:~/class$
```

j) using if inside awk

```
jitendra@DESKTOP-ACUC1TV:~/class$ awk -F'|' '{if($3>50){$3=$3+800;print $0}}' file1
name rollno 800
rajat 23 1144
jitendra@DESKTOP-ACUC1TV:~/class$
```

k) Print line numbers of data using awk

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
emp       role       dept    salary
jitendra  trainer    cse     100
rajat     dev        cse     120
vishal    manager    it      50
vikas     manager    it      200
piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk '{ print NR   $0}' file
1emp       role       dept    salary
2jitendra  trainer    cse     100
3rajat     dev        cse     120
4vishal    manager    it      50
5vikas     manager    it      200
6piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk '{ print NR " "   $0}' file
1 emp       role       dept    salary
2 jitendra  trainer    cse     100
3 rajat     dev        cse     120
4 vishal    manager    it      50
5 vikas     manager    it      200
6 piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$
```

l) Count lines using awk

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
emp       role       dept    salary
jitendra  trainer    cse     100
rajat     dev        cse     120
vishal    manager    it      50
vikas     manager    it      200
piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk 'END{print}' file
piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk 'END{print NR}' file
6
```

End will print last line NR will give numbering to lines

m) Loops in awk

```
jitendra@DESKTOP-ACUC1TV:~$ awk 'BEGIN {for(i=1;i<=10;i++)
print "square of ",i," is ",i*i;}'
square of  1  is  1
square of  2  is  4
square of  3  is  9
square of  4  is  16
square of  5  is  25
square of  6  is  36
square of  7  is  49
square of  8  is  64
square of  9  is  81
square of  10  is  100
jitendra@DESKTOP-ACUC1TV:~$
```

n) Pattern searching in awk

```
jitendra@DESKTOP-ACUC1TV:~$ awk '{print}' file
emp        role        dept    salary
jitendra   trainer     cse     100
rajat      dev         cse     120
vishal     manager     it      50
vikas      manager     it      200
piyush     trainer     cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk ' /manager/{print}' file
vishal     manager     it      50
vikas      manager     it      200
jitendra@DESKTOP-ACUC1TV:~$
```

o) Output field separator

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
emp       role       dept    salary
jitendra  trainer    cse     100
rajat     dev        cse     120
vishal    manager    it      50
vikas     manager    it      200
piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk '{print$1,$2,$3}'  file
emp role dept
jitendra trainer cse
rajat dev cse
vishal manager it
vikas manager it
piyush trainer cse
jitendra@DESKTOP-ACUC1TV:~$ awk 'BEGIN{OFS="--->"} {print$1,$2,$3}' file
emp--->role--->dept
jitendra--->trainer--->cse
rajat--->dev--->cse
vishal--->manager--->it
vikas--->manager--->it
piyush--->trainer--->cse
jitendra@DESKTOP-ACUC1TV:~$
```

p) Output record separator

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
emp       role       dept    salary
jitendra  trainer    cse     100
rajat     dev        cse     120
vishal    manager    it      50
vikas     manager    it      200
piyush    trainer    cse     300
jitendra@DESKTOP-ACUC1TV:~$ awk 'BEGIN{ORS="::::"} {print$1,$2,$3}' file
emp role dept::::jitendra trainer cse::::rajat dev cse::::vishal manager it::::vikas manag
er it::::piyush trainer cse::::jitendra@DESKTOP-ACUC1TV:~$
```

q) Script in a file

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
emp        role        dept    salary
jitendra   trainer     cse     100
rajat      dev         cse     120
vishal     manager     it      50
vikas      manager     it      200
piyush     trainer     cse     300
jitendra@DESKTOP-ACUC1TV:~$ cat script
{print $1 "----->" $3}
jitendra@DESKTOP-ACUC1TV:~$ awk -f script file
emp----->dept
jitendra----->cse
rajat----->cse
vishal----->it
vikas----->it
piyush----->cse
jitendra@DESKTOP-ACUC1TV:~$
```

r) Find number of employees whose salary is greater than 120

```
jitendra@DESKTOP-ACUC1TV:~$ cat file
name,dept,salary
jitendra,cse,100
vikas,ece,200
vijay,it,120
vikram,eee,300
pritam,cse,50
```

```
jitendra@DESKTOP-ACUC1TV:~$ cat script
BEGIN{
FS=",";
c=0;
}
NR!=1{
if($3>100)
{
c=c+1;
}
}
END{
print "The total number of employees where salary >100 : "c
}
```

```
jitendra@DESKTOP-ACUC1TV:~$ awk -f script file
The total number of employees where salary >100 : 3
jitendra@DESKTOP-ACUC1TV:~$
```

68. find:-

a) find a file in perticular directory

```
jitendra@LinuxBatch:~$ find . -name mycode.py
./mycode.py
jitendra@LinuxBatch:~$ find . -iname MYCODe.py
./mycode.py
jitendra@LinuxBatch:~$
```

b) -i ingnore case

c) to find a directory

```
jitendra@LinuxBatch:~$ find . -name folder
./class/folder
./folder
jitendra@LinuxBatch:~$ find . -type d -name folder
./class/folder
jitendra@LinuxBatch:~$
```

d) to specify a file or directory in find command

```
jitendra@LinuxBatch:~$ find . -type f -name fold*
./folder
jitendra@LinuxBatch:~$ find . -type d -name fold*
./class/folder
```

e) to find files with a perticular permission

```
jitendra@LinuxBatch:~/folder$ ls -l
total 0
-rwxrwxrwx 1 jitendra jitendra 0 Feb 22 11:24 file1
-rw-rw-r-- 1 jitendra jitendra 0 Feb 22 11:24 file2
jitendra@LinuxBatch:~/folder$ find . -perm 777
./file1
jitendra@LinuxBatch:~/folder$ find . -perm 664
./file2
```

f) to find files which does not have a perticular permission

```
jitendra@LinuxBatch:~/folder$ ls -l
total 0
-rwxrwxrwx 1 jitendra jitendra 0 Feb 22 11:24 file1
-rw-rw-r-- 1 jitendra jitendra 0 Feb 22 11:24 file2
jitendra@LinuxBatch:~/folder$ find . ! -perm 664
.
./file1
jitendra@LinuxBatch:~/folder$ find  . -type f ! -perm 664
./file1
jitendra@LinuxBatch:~/folder$
```

g) find files where user has read permission

```
jitendra@LinuxBatch:~/folder$ ls -l
total 0
-rwxrwxrwx 1 jitendra jitendra 0 Feb 22 11:24 file1
-rw-rw-r-- 1 jitendra jitendra 0 Feb 22 11:24 file2
jitendra@LinuxBatch:~/folder$ find . -type f -perm /u=r
./file2
./file1
```

h) find empty files

```
jitendra@LinuxBatch:~/folder$ find . -type f -empty
./file2
./file1
jitendra@LinuxBatch:~/folder$
```

i) find files with more specific permission

```
jitendra@LinuxBatch:~/folder$ ls -l
total 0
-rwxrwxrwx 1 jitendra jitendra 0 Feb 22 11:24 file1
-rw-rw-r-- 1 jitendra jitendra 0 Feb 22 11:24 file2
jitendra@LinuxBatch:~/folder$ find . -type f -perm -g=w,o=x
./file1
jitendra@LinuxBatch:~/folder$
```

69. locate :-

The locate command and find command is used to search a file by name. But, the difference between both commands is that locate command is a background process and searches the file in the database whereas, find command searches in the filesystem. The locate command is much faster than find command.

If you are unable to find a file with locate command, then it means that your database is out of date, and you can update your database with the "updatedb" command.

70. fdisk :-

fdisk is used for view, create, delete, change , resize, copy and move partitions on a hard drive

A partition is a logical division of a hard disk that is treated as a separate unit by operating systems (OSes) and file systems. The OSes and file systems can manage information on each partition as if it were a distinct hard drive

-l: Lists the partition table of the specified device(s) without making any changes.
-n: Creates a new partition.
-d: Deletes a partition.
-p: Prints the partition table of the specified device(s).
-t: Changes the system ID (type) of a partition.
-u: Changes the display units to sectors.
-v: Shows the version of the fdisk command.
-h or --help: Shows the help message.

----Users and groups permissions---

1) How to add a new group?
2) How to add a new user?

3) Switching from one user to another user
4) How to get information about a particular user?
5) How to delete user?
6) How to delete group ?
7) How to modify user info
8) How to modify group info
7) How to change ownership of a file?
8) How to change group membership of a file?
9) How to change group of a user ?
10) Add a User to Multiple Groups
11) How to check available groups?
12) How to change password of a user?
13) What is the difference between adduser and useradd in Linux?
14)sudo command
15)sudoers file
16)sudo command
17)sudoers file
18) How to check the allowed commands by sudo for a particular user ?

1) How to add a new group?

we have to user addgroup command to add new group. for this sudo permission is required.

```
pc@Jitendra:~$ sudo addgroup dceu
Adding group `dceu' (GID 1001) ...
Done.
pc@Jitendra:~$ cat /etc/group
```

info about group will be in /etc/group file

```
netdev:x:116:pc
pc:x:1000:
dceu:x:1001:
pc@Jitendra:~$
```

2) How to add a new user?

using useradd command

```
pc@Jitendra:~$ sudo adduser batman
Adding user `batman' ...
Adding new group `batman' (1001) ...
Adding new user `batman' (1001) with group
batman' ...
Creating home directory `/home/batman' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for batman
Enter the new value, or press ENTER for the
default
        Full Name []: bruce wayne
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
pc@Jitendra:~$ cat /etc/passwd
```

/etc/passwd file will have info about all the users

```
tcpdump:x:107:113::/nonexistent:/usr/sbin/nologin
pc:x:1000:1000:,,,:/home/pc:/bin/bash
batman:x:1001:1001:bruce wayne,,,:/home/batman:/bin/bash
pc@Jitendra:~$
```

And by default if you don't specify the gorupname then group of same name(username) will be created nd assigned to the user

Useradd command :- to add user need superuser access to do this

1. –m : if you want users home directory also
2. –g : if you want to give other group other than default
3. -s : to give shell otherwise default shell will be dash shell

3) Switching from one user to another user ?

```
pc@Jitendra:~$ su batman
Password:
batman@Jitendra:/home/pc$
```

Note: If we use su command without any argument, then it will switch to root user.
Hence the following 2 commands are equal.
$ su root
we can also use sudo -i if above command is not working
durga@durga-VirtualBox:~$ sudo -i

4) How to get information about a particular user?

5) How to delete user?

```
pc@Jitendra:~$ sudo deluser batman
[sudo] password for pc:
Removing user `batman' ...
Warning: group `batman' has no more members.
Done.
pc@Jitendra:~$
```

Note : we can check whether user deleted or not by
checking /etc/passwd file

6) How to delete group ?

```
netdev:x:116:pc
pc:x:1000:
dceu:x:1001:
pc@Jitendra:~$ sudo delgroup dceu
Removing group `dceu' ...
Done.
pc@Jitendra:~$
```

7) How to Modify user info ?

We can modify users info using usermod command

a) To add comment for a user
sudo usermod -c "This is new comment" username

b) To change user's home directory
sudo usermod -d /home/jitendra2 jitendra

c) To change expiry date of a user
sudo usermod -e 2023-03-31 username

d) To change group of user
sudo usermod -g jitendra newgroup

e) To change user's login name
sudo usermod -l oldname newname

f) To change shell for the user
sudo usermod -s /bin/sh username

g) To change userid of the user
sudo usermod -u 1234 username

8) How to Modify group info ?

groupmod command in Linux is used to modify or change the existing group on Linux system

a) To change group id
groupmod -g 1000 dceu

b) To change group name
Groupmod -n dceu dc

Gpasswd command

-a to ad
-d delete
-A to to add admin

9) How to change ownership of a file?

We can change by using chown command.
But for this sudo permission required.

```
pc@Jitendra:~$ ls -l
total 4
-rwxr--r-- 1 pc pc 3700 Mar 24 13:58 code.sh
pc@Jitendra:~$
```

```
pc@Jitendra:~$ sudo chown root code.sh
pc@Jitendra:~$ ls -l
total 4
-rwxr--r-- 1 root pc 3700 Mar 24 13:58 code.sh
pc@Jitendra:~$
```

10) How to change group membership of a file?

```
pc@Jitendra:~$ sudo chgrp root code.sh
pc@Jitendra:~$ ls -l
total 4
-rwxr--r-- 1 root root 3700 Mar 24 13:58 code.sh
pc@Jitendra:~$
```

11) How to change group of a user ?

We have to use usermod command.
Syntax: $ usermod -g groupname username

Note: We can find groups associated with particular user by using groups command as follows
$ groups username

```
pc@Jitendra:~$ groups pc
pc : pc adm dialout cdrom floppy sudo audio dip video plugdev netdev
pc@Jitendra:~$ sudo usermod -g dceu jitendra
usermod: user 'jitendra' does not exist
pc@Jitendra:~$ sudo usermod -g dceu pc
pc@Jitendra:~$ groups pc
pc : dceu adm dialout cdrom floppy sudo audio dip video plugdev netdev
pc@Jitendra:~$
```

12) Add a User to Multiple Groups

While assigning the secondary groups to a user account, we can easily assign multiple groups at once by separating the list with a comma.

$ usermod -a -G group1,group2,group3 username

13) How to check available groups?

To check available groups:
$ groups
This command lists all the groups that you belong to.
To check which group a certain user belongs to:
$ groups [username]

14) How to change password of a user?

To change a password for user named durga:
$ sudo passwd durga
To change a password for root user:
$ sudo passwd root
To change your own password:
$ passwd

-d ---- > delete the password and make account passwordless
-e ----- > expire the password and ask to change
--mindays ----- > minimum number of days after we can to change the password.

--maxdays ----- > maximum days after which your password will expire and you need to change need to change
--warndays ------ > will warn user to change password

```
pc@Jitendra:~$ sudo passwd pc
New password:
Retype new password:
passwd: password updated successfully
pc@Jitendra:~$
```

15) What is the difference between adduser and useradd in Linux?

Useradd is built-in Linux command that can be found on any Linux system. However, creating new users with this low-level is a difficult and lengthy task. First we have to creater user and then we have to provide password and extra information separately.
Adduser is not a standard Linux command. It is essentially a Perl script that uses the useradd command in the background. This high-level utility is more efficient in properly creating new users on Linux. Default parameters for all new users can also be set through the adduser command.

16)sudo command

sudo means Super User Do.
We can use sudo command to execute commands as another user, mostly root user.

```
pc@Jitendra:~$ sudo -u kala whoami
kala
pc@Jitendra:~$
```

Note: If we want to use sudo command for any user other than root then we have to use -u option.
The following two commands are equal.
$ sudo command
$ sudo -u root command

17)sudoers file

Which commands can be executed by using sudo of a particular user, information is configured in sudoers file, which is present in /etc directory.
System Administrators are responsible to configure this file.

```
pc@Jitendra:~$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
pc@Jitendra:~$ sudo cat /etc/sudoers
#
# This file MUST be edited with the 'visudo'
 command as root.
#
# Please consider adding local content in /e
tc/sudoers.d/ instead of
# directly modifying this file.
```

18) How to check the allowed commands by sudo for a particular user ?

$ sudo -l
It will display allowed commands of current user by sudo.
$ sudo -l -u username
It will display allowed commands of provided user by sudo.

```
pc@Jitendra:~$ sudo -l
Matching Defaults entries for pc on
    Jitendra:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/
bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap
/bin,
    use_pty

User pc may run the following commands on
        Jitendra:
    (ALL : ALL) ALL
```

-----process management-----

first process is init/systemd and it does not have any parent
demon ----> process which are started with system and keep running demon never dies

68. ps :-

ps -------- > means process status

a) **if we don't pass any argument then it shows processes related to current session**

```
jitendra@LinuxBatch:~$ ps
    PID TTY          TIME CMD
   5855 pts/0    00:00:00 bash
   5912 pts/0    00:00:00 ps
```

b) **ps -e -------> everything**
it shows all the process

```
jitendra@LinuxBatch:~$ ps -e
    PID TTY          TIME CMD
      1 ?        00:00:03 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      5 ?        00:00:00 slub_flushwq
      6 ?        00:00:00 netns
      8 ?        00:00:00 kworker/0:0H-events_highpri
     10 ?        00:00:00 mm_percpu_wq
     11 ?        00:00:00 rcu_tasks_rude_
     12 ?        00:00:00 rcu_tasks_trace
     13 ?        00:00:00 ksoftirqd/0
     14 ?        00:00:02 rcu_sched
     15 ?        00:00:00 migration/0
     16 ?        00:00:00 idle_inject/0
     18 ?        00:00:00 cpuhp/0
     19 ?        00:00:00 cpuhp/1
```

a process that is not associated with terminal will show ? in tty field

c) ps -f -------> full form

```
jitendra@LinuxBatch:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
jitendra    5855    5829  0 19:18 pts/0    00:00:00 bash
jitendra    5917    5855  0 19:30 pts/0    00:00:00 ps -f
```

UID UserId/ User Name
PID Process ID (Every Process has Unique ID)
PPID Parent Process ID
Time --> CPU Utilization Time
Stime ---> start time
c ----> cpu usage %

d) processes related to a perticular user

ps -u username

e) process with name

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps
  PID TTY          TIME CMD
  152 pts/0    00:00:00 bash
  176 pts/0    00:00:00 tar
  177 pts/0    00:00:00 sh
  178 pts/0    00:00:02 bzip2
  186 pts/0    00:00:00 ps
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps -C tar
  PID TTY          TIME CMD
  176 pts/0    00:00:00 tar
```

I want to know process of id of perticular command

f) ps -aux

all user extra , Shows a detailed listing of all processes running on the system, including those started by other users, and additional information about each process.

g) ps -t terminal

to show all the processes related to a specific terminal

```
jitendra@PC:~$ tty
/dev/pts/0
jitendra@PC:~$ ps -t pts/0
  PID TTY          TIME CMD
   15 pts/0    00:00:00 bash
   97 pts/0    00:00:00 ps
```

h) **ps -G groupId** ----> to show all process related to groupId

i) **ps -C commandname** -----> to show all process related to a perticular command

j) **ps -l ----->** longform shows additional info about each process

```
jitendra@PC:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000    15    14  1  80   0 -  1591 do_wai pts/0    00:00:00 bash
0 R  1000    24    15  0  80   0 -  1870 -      pts/0    00:00:00 ps
jitendra@PC:~$
```

F ------> The process's flags or state.
UID -----> The user ID of the process owner.
PID -----> The process ID of the process.
PPID ---->The parent process ID of the process.
C: ---->The CPU utilization of the process.
PRI: ---->The priority of the process.
NI: ----->The nice value of the process.
ADDR: ---> The memory address of the process.
SZ: ---> The size of the process in memory (in kilobytes).
WCHAN: --> The name of the kernel function that the process is waiting on.
TTY: ---> The terminal that the process is running on.
TIME: ----->The CPU time used by the process.
CMD: ----->The command that started the process.

k) **ps -r** ----> view all the running processes

l) ps --ppid PPID ------> process related to parent process id

1. sleep :-

the sleep command will tell our computer to don't do anythong for a while
 and then start
taking commands or run the previous command

ex :- sleep
sleep 5
sleep 5m
sleep 5d

sleep 10 ; pwd

2. jobs :-

jobs command will give information about the jobs which are in background or foreground

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]-  Running                 tar -jcf backup1.tar * &
[2]+  Stopped                 tar -jcf backup2.tar *
[3]   Stopped                 tar -jcf backup3.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]-  Done                    tar -jcf backup1.tar *
[2]+  Stopped                 tar -jcf backup2.tar *
[3]   Stopped                 tar -jcf backup3.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

+ -----> last job
- -----> second last job
& -----> indicates job that is running in background

options with jobs

-p -----> will also give process id of the job
-r -----> only running jobs
-s -----> only stopped jobs
-l -----> long form (it will also display process id)

ctrl+c ----> kill foreground job
ctrl+z -----> suspend foreground job

3. bg :-

bg command in linux is used to place foreground jobs in background,
or resume any job in background

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ls
'Camera Roll'                          Screenshots
 Notion-Enhanced-Setup-2.0.18-1.exe    androidhd.com_pikashow_10.8.2.apk
 Obsidian.1.1.9.exe                    desktop.ini
 OpenBoard                             image2.png
'Saved Pictures'                       image_2023-02-10_18-46-41.png
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ tar -jcf backup.tar *
^Z
[1]+  Stopped                 tar -jcf backup.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Stopped                 tar -jcf backup.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ bg %1
[1]+ tar -jcf backup.tar * &
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Running                 tar -jcf backup.tar * &
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

here I am creating a compressed archieve of all files and folders inside
pictures folder using bzip2 compression algo

then suspending or stopping the job
then moving that job in background

bg -----> will move last job in background
bg %+ -----> will move last job in background
bg % - -----> will move last to last job in background
bg %n -----> will move nth job in background

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Running                 tar -jcf backup1.tar * &
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ kill -STOP %1

[1]+  Stopped                 tar -jcf backup1.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Stopped                 tar -jcf backup1.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

How to Kill Background Job:
kill -9 processid
kill -9 %jobid

How to suspend Background job:
kill -STOP %jobid
kill -19 processid

How to Resume Background Job:
bg %jobid
kill -CONT %jobid
kill -CONT processid

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs -l
[1]+   168 Stopped (signal)        tar -jcf backup1.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ kill -9 168
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Killed                    tar -jcf backup1.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

4. fg :-

fg command in linux used to put a background job in foreground

How to Kill Foreground Job:
ctrl+c

How to suspend foreground job:
ctrl+z

How to Resume Foreground Job:
$fg

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ tar -jcf backup1.tar * &
[1] 173
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
[1]+  Running                 tar -jcf backup1.tar * &
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ fg
tar -jcf backup1.tar *
^Z
[1]+  Stopped                 tar -jcf backup1.tar *
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ fg
tar -jcf backup1.tar *
^C
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ jobs
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

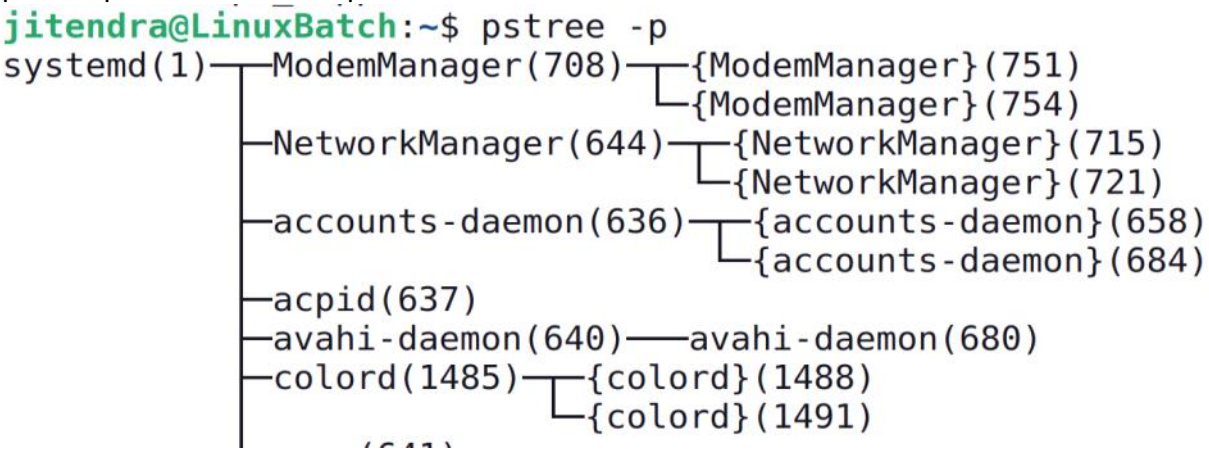
started a task in background then moved that into foreground
then stoped, resumed it , killed it

fg -----> will move last job in foreground
fg %+ -----> will move last job in foreground
fg % - -----> will move last to last job in foreground
fg %n -----> will move nth job in foreground

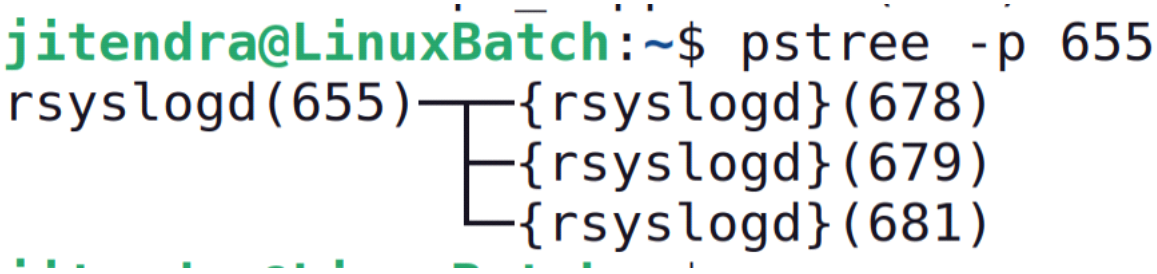
5. pstree :-

to see all the process in form of tree

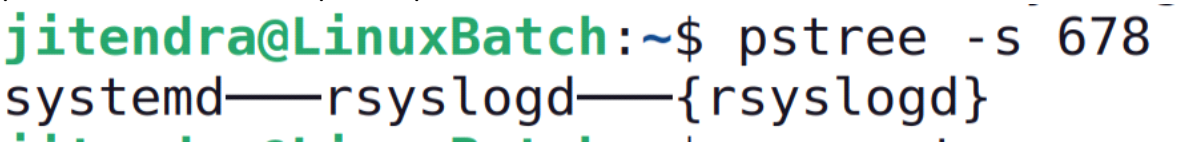
pstree -p -----> will show process id also



pstree -u -----> will show all process of current user,
pstree -u jitendra -----> will show all process of user jitendra



pstree -s ----> will show parent process



6. nice :-

if there is a process A, which detects fraud with
input data and there is another process B, which makes hourly
backups of some data,

then the priority(A) > priority(B)
This ensures that if both A and B are running at the same time, A
would be allocated more processing bandwidth.

Only root user can set the nice value from -20 to 19. Other
users can only set nice values from 0 to 19.

the higher this value, the lower priority

to run a command with with custom priority

The "niceness" scale goes from -20 to 19,

whereas -20 it's the highest priority and 19 the lowest priority.

```
jitendra@DESKTOP-ACUC1TV:~$ nice -5 sleep 100 &
[3] 293
jitendra@DESKTOP-ACUC1TV:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000   273   272  0  80   0 -  1551 do_wai pts/2    00:00:00 bash
0 S  1000   293   273  0  85   5 -   801 hrtime pts/2    00:00:00 sleep
0 R  1000   294   273  0  80   0 -  1869 -      pts/2    00:00:00 ps
jitendra@DESKTOP-ACUC1TV:~$
```

nice -n -5 ------> to assign negative nice value.

7. renice :-

to change the scheduling priority of already running process

---> running a process in background
---> changing it's nice value using PID

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ tar -jcf backup.tar * &
[1] 381
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000   366   365  0  80   0 -  1552 do_wai pts/2    00:00:00 bash
0 S  1000   381   366  6  80   0 -  1123 pipe_w pts/2    00:00:00 tar
0 S  1000   382   381  0  80   0 -   721 do_wai pts/2    00:00:00 sh
0 R  1000   383   382 47  80   0 -  2541 -      pts/2    00:00:00 bzip2
0 R  1000   384   366  0  80   0 -  1869 -      pts/2    00:00:00 ps
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ sudo renice -5 -p 381
[sudo] password for jitendra:
381 (process ID) old priority 0, new priority -5
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000   366   365  0  80   0 -  1552 do_wai pts/2    00:00:00 bash
0 S  1000   381   366  1  75  -5 -  1123 pipe_w pts/2    00:00:00 tar
0 S  1000   382   381  0  80   0 -   721 do_wai pts/2    00:00:00 sh
0 D  1000   383   382 33  80   0 -  2541 p9_cli pts/2    00:00:11 bzip2
0 R  1000   388   366  0  80   0 -  1869 -      pts/2    00:00:00 ps
```

renice -n 10 -u username ------> to change nice value of all the process of given user

```
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000   366   365  0  80   0 -  1617 do_wai pts/2    00:00:00 bash
0 S  1000   390   366  2  80   0 -  1123 pipe_w pts/2    00:00:00 tar
0 S  1000   391   390  0  80   0 -   721 do_wai pts/2    00:00:00 sh
0 D  1000   392   391 42  80   0 -  2541 p9_vir pts/2    00:00:00 bzip2
0 R  1000   393   366  0  80   0 -  1869 -      pts/2    00:00:00 ps
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ sudo renice -5 -u jitendra
1000 (user ID) old priority 0, new priority -5
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000   366   365  0  75  -5 -  1617 do_wai pts/2    00:00:00 bash
0 S  1000   390   366  1  75  -5 -  1123 pipe_w pts/2    00:00:00 tar
0 S  1000   391   390  0  75  -5 -   721 do_wai pts/2    00:00:00 sh
0 D  1000   392   391 33  75  -5 -  2541 p9_cli pts/2    00:00:13 bzip2
0 R  1000   397   366  0  75  -5 -  1869 -      pts/2    00:00:00 ps
jitendra@DESKTOP-ACUC1TV:/mnt/c/Users/jitendra/Pictures$
```

renice -n 10 -g groupname ------> to change nice value of all the process of given group

8. kill :-

kill pid ----> kill process

kill -9 pid ----> focefully terminate

kill -STOP pid -----> to stop/pause a process
kill -CONT pid -----> to start/resume a process

9. killall :-

using killall command we can kill a process by name

-r ----> we can use regular expression
-v ----> verbose
-i ----> intrective
-I ----> ignore case (capital i)

```
jitendra@LinuxBatch:~$ killall -v ping
Killed ping(3491) with signal 15
jitendra@LinuxBatch:~$ killall -i ping
Kill ping(3493) ? (y/N) y
jitendra@LinuxBatch:~$
```

10. Top :-

```
jitendra@LinuxBatch:~$ top

top - 00:19:33 up 1 min,  1 user,  load average: 2.10, 0.92, 0.34
Tasks: 199 total,   1 running, 198 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  0.7 sy,  0.0 ni, 97.9 id,  0.0 wa,  0.0 hi,  0.9 si,  0.0
MiB Mem :   1975.8 total,    496.1 free,    714.1 used,    765.6 buff/cache
MiB Swap:   2680.0 total,   2680.0 free,      0.0 used.   1081.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
1015	jitendra	20	0	3989244	345624	135812	S	5.0	17.1	0:20.17
432	systemd+	20	0	14828	6152	5360	S	0.7	0.3	0:00.30
1607	jitendra	20	0	641832	57920	45136	S	0.7	2.9	0:01.55
22	root	20	0	0	0	0	S	0.3	0.0	0:00.64
109	root	20	0	0	0	0	I	0.3	0.0	0:00.21
353	root	20	0	0	0	0	I	0.3	0.0	0:00.12
1643	**jitendra**	**20**	**0**	**21728**	**4056**	**3360**	**R**	**0.3**	**0.2**	**0:00.15**
1	root	20	0	101192	11844	8284	S	0.0	0.6	0:02.91
2	root	20	0	0	0	0	S	0.0	0.0	0:00.03
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00

1 -------> Uptime , user count , load average ,
2 -------> cpu count (press 1 to check other cpus), user process, system process, nice value, idle,
hardware intrupt, system interrupt, stolen time
3 -------> free

Res ----> reserved memory
Virt -----> virtual memory
Shr -----> shared memory
S ----> running , stopped , idle etc

Press u ----> enter username
K ----> kill

- ‘M’ to sort by memory usage
- ‘P’ to sort by CPU usage
- ‘N’ to sort by process ID
- ‘T’ to sort by the running time

By default, top displays all results in descending order. However,
you can switch to ascending order by pressing ‘R’.

--------Networking commands --------

While working on internet , we may come across many problems. To overcome these problems
We have some tools or we can say networking commands which can be used to
Sort out the issue and detect the issue.

1. Traceroute :-

This command is used to view the route using which you will be transmitted from
Your computer to the said website. It is similar to PING except that is identifies the
Pathway rather than time.

```
jitendra@PC:~$ traceroute google.com
traceroute to google.com (142.250.183.174), 30 hops max, 60 byte packets
 1  DESKTOP-ACUC1TV.mshome.net (192.168.176.1)  0.421 ms  0.903 ms  0.873 ms
 2  192.168.106.137 (192.168.106.137)  14.338 ms  17.027 ms  15.564 ms
 3  10.8.255.254 (10.8.255.254)  140.743 ms  140.733 ms  140.722 ms
 4  * * *
 5  10.174.171.81 (10.174.171.81)  73.090 ms  73.080 ms  73.070 ms
 6  * * *
 7  192.168.100.25 (192.168.100.25)  49.247 ms  58.473 ms  65.443 ms
 8  * * 192.168.100.66 (192.168.100.66)  84.960 ms
```

--->Some organization set their system to not respond on traceroute command
--->some routers does not respond sometime due to overload

traceroute -I google.com -----> to use ICMP echo request instead of UDP packets.
traceroute -T google.com -----> This command will use TCP SYN (synchronize) packets instead of UDP packets.
traceroute -p 80 google.com -----> this command will set the destination port number to 80.
traceroute -m 20 google.com ------> set the maximum number of hops

2. Nslookup :-

NSLookup (short for "Name Server Lookup") is a command-line tool used to quer
y Domain Name System (DNS) servers to obtain DNS information, such as IP
addresses, hostnames, and other DNS records. In general, the NSLookup
command is used to troubleshoot DNS-related issues, verify DNS
configurations, and obtain information about DNS servers.

nslookup google.com

```
jitendra@PC:~$ nslookup facebook.com
Server:         192.168.176.1
Address:        192.168.176.1#53

Non-authoritative answer:
Name:   facebook.com
Address: 157.240.16.35
Name:   facebook.com
Address: 2a03:2880:f12f:83:face:b00c:0:25de
```

nslookup -x 157.240.16.35 ----> reverse lookup using ip address

nslookup debug

```
jitendra@PC:~$ nslookup debug
Server:         192.168.176.1
Address:        192.168.176.1#53

** server can't find debug: NXDOMAIN
```

3. ipconfig :-

This command is used to display detailed information about the network
We are connected to.
Ipconfig/all will display detailed information

4. ifconfig :-

Interface configure ----> this command is used to configure and display network interfaces.
syntax :-

ifconfig [interface] [options]

The **interface** argument specifies the network interface to configure or display.
If no interface is specified, ifconfig displays information for **all interfaces**.

-a : This option is used to display all the interfaces available, even if they are down.
-s : To display short list instead of full details

up : This option is used to activate the driver for the given interface.

Syntax:
ex:- ifconfig interface up

down : This option is used to deactivate the driver for the given interface.
Syntax:
ex:- ifconfig interface down

changing ip/netmask/broadcast address

ifconfig interface addresstype address

```
jitendra@PC:~$ sudo ifconfig eth0 netmask 255.255.0.0
jitendra@PC:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.177.40  netmask 255.255.0.0  broadcast 192.168.177.41
        inet6 fe80::215:5dff:fee0:1da6  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:e0:1d:a6  txqueuelen 1000  (Ethernet)
        RX packets 26  bytes 3560 (3.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1352 (1.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4. netstat :-

netstat -at ---> all tcp connection
netstat -au ----> all udp connection
netstat -l -----> all listnening states
netstat -lt -----> only tcp listening
netstat -s -----> statstics protocalwise
netstat -st -----> tcp
netstat -su ----> for udp
netstat -p -----> process id process name
netstat -n -----> numeric port
netstat -c ----> continuous statistic s
netstat -I ----> interface/ ifconfig command output
netstat -ie -----> extended interface
netstat -r ------> to print routing table

---------Shell Scripting----------

1) Shell is responsible to read commands/applications provided by user.

2) Shell will check whether command is valid or not and whether it is properly used or not. If everything is proper then shell interpretes (converts) that command into kernel understandable form. That interpreted command will be handover to kernel.

3) Kernel is responsible to execute that command with the help of hardware.

Shell acts as interface between user and kernel.

shell+kernel is nothing but operating system.

1) Bourne Shell:

--> It is developed by Stephen Bourne.
--> This is the first shell which is developed for UNIX.
--> By using sh command we can access this shell.

2) BASH Shell:

--> BASH Bourne Again SHell.
--> It is advanced version of Bourne Shell.
--> This is the default shell for most of the linux flavours.
--> By using bash command we can access this shell.

3) Korn Shell:

--> It is developed by David Korn.
--> Mostly this shell used in IBM AIX operating system.
--> By using ksh command, we can access this shell.

4) CShell:

--> Developed by Bill Joy.
--> C meant for California University.
--> It is also by default available with UNIX.
--> By using csh command, we can access this shell.

5) TShell:

--> T means Terminal.
--> It is the advanced version of CShell.
--> This is the most commonly used shell in HP UNIX.
--> By using tcsh command, we can access this shell.

6) Z Shell:

--> Developed by Paul.
--> By using zsh command we can access this shell.

Note: The most commonly used shell in linux environment is BASH. It is more powerful than remaining shells.

How to Check Default Shell in our System?

```
jitendra@DESKTOP-ACUC1TV:~$ echo $0
-bash
jitendra@DESKTOP-ACUC1TV:~$ echo $SHELL
/bin/bash
jitendra@DESKTOP-ACUC1TV:~$ cat /etc/passwd
```

3rd way is to check passwd file where we have information about the Users

How to check all available Shells in our System?

/etc/shells file contains all available shells information.

```
jitendra@DESKTOP-ACUC1TV:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/sh
/bin/dash
/usr/bin/dash
/usr/bin/tmux
/usr/bin/screen
```

How to Switch to other Shells?

Based on our requirement we can switch from one shell to another shell.

```
jitendra@DESKTOP-ACUC1TV:~$ sh
$ echo $0
sh
$ exit
jitendra@DESKTOP-ACUC1TV:~$ dash
$ echo $0
dash
$ exit
jitendra@DESKTOP-ACUC1TV:~$ sh
$ bash
jitendra@DESKTOP-ACUC1TV:~$
```

What is Shell Script:

A sequence of commands saved to a file and this file is nothing but shell script.

Inside shell script, we can also use programming features like conditional statements, loops, functions etc. Hence we can write scripts very easily for complex requirements also.

Best suitable for automated tasks.

How to write and run Shell Script:

Step - 1: Write script:
We can write our script using cat command, gedit, vi or any other editors like Vscode etc

Step - 2: Provide execute permissions to the script:-
Using chmod command we can change permissions

Step - 3: Run the script:-
We can run the script in multiple ways

```
jitendra@PC:~$ #step 1
jitendra@PC:~$ cat>script
echo "hello world"
jitendra@PC:~$ #step 2
jitendra@PC:~$ chmod u+x script
jitendra@PC:~$ #step3
jitendra@PC:~$ ./script
hello world
jitendra@PC:~$ bash script
hello world
jitendra@PC:~$ sh script
hello world
jitendra@PC:~$
```

Note: The default shell is bash. Hence bash is responsible to execute our script. Instead of bash, if we want to use Bourne shell then we need to execute the Script using sh

Importance of Sha-Bang:

By using sha-bang, we can specify the interpreter which is responsible to execute the script.

□ Sharp
! □ Bang

#! □ Sharp Bang or Shabang or Shebang

#! /bin/bash □ It means the script should be executed by bash

#! /bin/sh □ It means the script should be executed by Bourne Shell

#! /usr/bin/python3 □ It means the script should be executed by Python3 interpreter

If we write shabang in our script at the time of execution, we are not required to provide command to execute and we can execute script directly.

```
jitendra@PC:~/class$ cat code.py
#!/bin/python3
n = int(input("enter a number  "))
if n%2==0:
    print("even")
else:
    print("odd")
jitendra@PC:~/class$ ./code.py
enter a number  4
even
jitendra@PC:~/class$
```

Shell Variables :-

Variables are place holders to hold values.
Variables are key-value pairs.
In Shell programming, there are no data types. Every value is treated as text type/ String type.

All variables are divided into 2 types
1) Environment variables / predefined variables
2) User defined variables

1)Environment Variables:

These are predefined variables and mostly used internally by the system. Hence these variables also known as System variables.
But based on our requirement, we can use these variables in our scripts.
We can get all environment variables information by using either env command or set command.

```
jitendra@PC:~$ env
SHELL=/bin/bash
WSL2_GUI_APPS_ENABLED=1
WSL_DISTRO_NAME=Ubuntu
NAME=PC
PWD=/home/jitendra
LOGNAME=jitendra
HOME=/home/jitendra
LANG=C.UTF-8
```

A simple shell script that uses environment variables

```
jitendra@PC:~$ cat script
#! /bin/bash
echo "User Name: $USER"
echo "User Home Directory: $HOME"
echo "Default Shell Name: $SHELL"
jitendra@PC:~$ ./script
User Name: jitendra
User Home Directory: /home/jitendra
Default Shell Name: /bin/bash
jitendra@PC:~$
```

2)User defined Variables:

Based on our programming requirement, we can define our own variables also.

```
jitendra@PC:~$ cat script
#! /bin/bash
name="jitendra singh"
echo $name

jitendra@PC:~$ ./script
jitendra singh
jitendra@PC:~$
```

Rules to define Variables:

1) It is recommended to use UPPER CASE characters.
2) If variable contains multiple words, then these words should be separated with _ symbol.
3) Variable names should not starts with digit.
 $ 123A=40
 123A=40: command not found
4) We should not use special symbols like -,@,# etc

How to define Readonly Variables:

We can define read only variables by using readonly keyword.
$A=100
$readonly A
$A=300
bash: A: readonly variable
If the variable is readonly then we cannot perform reassignment for that variable.It will become constant

Variable Substitution:

Accessing the value of a variable by using $ symbol is called variable substitution.
Syntax:
$variablename
${variablename}
Recommended to use ${variablename}

```
jitendra@PC:~$ cat script
#!/bin/bash
a=10
b=20
COURSE="linux"
ACTION="SLEEP"
echo "Values of a and b are: $a and $b"
echo "My Course is: ${COURSE}"
echo "Your Favourite Action: $ACTIONING"
echo "Your Favourite Action: ${ACTION}ING"
jitendra@PC:~$ ./script
Values of a and b are: 10 and 20
My Course is: linux
Your Favourite Action:
Your Favourite Action: SLEEPING
jitendra@PC:~$
```

Command Substitution:

We can execute command and we can substitute its result based on our requirement by using command substitution.

Syntax:

Old style: `command ` These are backquotes butnot single quotes

New Style: $(command)

Ex 1:-

```
jitendra@PC:~$ echo "Today date is: `date +%D`"
Today date is: 02/26/23
jitendra@PC:~$ echo "Today date is: $(date +%D)"
Today date is: 02/26/23
jitendra@PC:~$
```

Ex 2:-

```
jitendra@PC:~$ echo "my current working directory is $(pwd)"
my current working directory is /home/jitendra
jitendra@PC:~$ echo "my current working directory is 'pwd'"
my current working directory is 'pwd'
jitendra@PC:~$ echo "my current working directory is `pwd`"
my current working directory is /home/jitendra
jitendra@PC:~$
```

Do not get confused between single quotes and backquotes

Command Line Arguments :-

The arguments which are passing from the command prompt at the time of executing our script, are called command line arguments.

$ ./test.sh learning linux is very easy

The command line arguments are learning, linux, is, very, easy.
Inside script we can access command line arguments as follows:

$# → Number of Arguments (5)

$0 → Script Name (./test.sh)

$1 → 1st Argument (learning)
$2 → 2nd Argument (linux)
$3 → 3rd Argument (is)
$4 → 4th Argument (very)
$5 → 5th Argument (easy)
$* → All Arguments (learning Linux is very easy)
$@ →All Arguments (learning Linux is very easy)

$? → Represents exit code of previously executed command or script.

```
jitendra@PC:~$ cat test.sh
#!/bin/bash
echo "number of arguments : $#"
echo "Script name : $0"
echo "first argument : $1"
echo "second argument : $2"
echo "third argument : $3"
echo "fourth argument : $4"
echo "fifth argument : $5"
echo "Total arguments : $*"
jitendra@PC:~$ ./test.sh My Name is Jitendra Singh
number of arguments : 5
Script name : ./test.sh
first argument : My
second argument : Name
third argument : is
fourth argument : Jitendra
fifth argument : Singh
Total arguments : My Name is Jitendra Singh
```

Difference between $@ and $*:

$@ → All command line arguments with space separator
"$1" "$2" "$3" ...
$* → All command line arguments as single string
"$1c$2c$3c.."

Where c is the first character of the Internal Field Separator (IFS).
The default first character is space.

How to Check Default IFS:
$ set | grep "IFS"
IFS=$' \t\n'

We can set IFS during script

```
#!/bin/bash
IFS="-"
echo "number of arguments : $#"
echo "Script name : $0"
echo "first argument : $1"
echo "second argument : $2"
echo "third argument : $3"
echo "fourth argument : $4"
echo "fifth argument : $5"
echo "Total arguments seperated by IFS: $*"
echo "Total arguments seperated by space : $@"
jitendra@PC:~$ ./test.sh My Name is Jitendra Singh
number of arguments : 5
Script name : ./test.sh
first argument : My
second argument : Name
third argument : is
fourth argument : Jitendra
fifth argument : Singh
Total arguments seperated by IFS: My-Name-is-Jitendra-Singh
Total arguments seperated by space : My Name is Jitendra Singh
jitendra@PC:~$
```

Q1. shell script to check length of given command line argument

```
jitendra@PC:~$ cat test.sh
#!/bin/bash

len=$(echo -n "$1" | wc -c)
echo "The length of given string is $1 : $len"
jitendra@PC:~$ ./test.sh "jitendra"
The length of given string is jitendra : 8
jitendra@PC:~$ ./test.sh jitendra
The length of given string is jitendra : 8
jitendra@PC:~$
```

Another method

```
jitendra@PC:~$ cat test.sh
#!/bin/bash

first=$1
len=${#1}
echo "The length of given string is $1 : $len"
jitendra@PC:~$ ./test.sh jitendra
The length of given string is jitendra : 8
jitendra@PC:~$
```

#var ------> will give length of the variable value
#1 ------> will give length of first command line argument

Doing all the tasks in a single line using variable substitution

```
jitendra@PC:~$ cat test.sh
#!/bin/bash
echo "The length of given string is $1 : ${#1}"
jitendra@PC:~$ ./test.sh jitendra
The length of given string is jitendra : 8
jitendra@PC:~$
```

How to Read Dynamic Data from the Use

Using read keyword we can read the dynamic data from the user

```
jitendra@PC:~$ read a b
100 200
jitendra@PC:~$ echo "a = $a, b = $b"
a = 100, b = 200
jitendra@PC:~$
```

What if I give more than 2 values

```
jitendra@PC:~$ read a b
100 200 300 400
jitendra@PC:~$ echo "a = $a, b = $b"
a = 100, b = 200 300 400
jitendra@PC:~$
```

With Prompt Message:

Approach-1

```
jitendra@PC:~$ cat test.sh
#! /bin/bash
echo "Enter A Value:"
read A
echo "Enter B Value:"
read B
echo "A Value: $A"
echo "B Value: $B"
jitendra@PC:~$ ./test.sh
Enter A Value:
100
Enter B Value:
200
A Value: 100
B Value: 200
jitendra@PC:~$
```

Approach-2

Using read -p

p----> prompt

```
jitendra@PC:~$ cat test.sh
#! /bin/bash
read -p "Enter A Value:" A
read -p "Enter B Value:" B
echo "A Value: $A"
echo "B Value: $B"
jitendra@PC:~$ ./test.sh
Enter A Value:100
Enter B Value:200
A Value: 100
B Value: 200
jitendra@PC:~$
```

read -s ------> it hides the input on screen which is provided by end user

```
jitendra@PC:~$ cat test.sh
#! /bin/bash
read -p "Enter username:" username
read -s -p "Enter password:" password
echo ""
echo "username is: $username"
echo "password is : $password"
jitendra@PC:~$ ./test.sh
Enter username:jitendra
Enter password:
username is: jitendra
password is : 12345
jitendra@PC:~$
```

Q. Write a shell script which will take input data of an employee
R. and add that to a file
eid , ename, esal, erole

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter employee id :" eid
read -p "Enter employee name :" ename
read -p "Enter employee salary :" esal
read -p "Enter employee role :" erole

echo "Below details are saved to the file"
echo "$eid:$ename:$esal:$erole"
echo "$eid:$ename:$esal:$erole">>empsdata.txt
```

Q. Write a Script to Read File Name from the End User anddisplay its Content?

```
jitendra@PC:~$ cat code.sh
#!/bin/bash
read -p "Enter filename to display content: " fname
echo "------------------------------"
cat $fname
echo "------------------------------"

jitendra@PC:~$ ./code.sh
Enter filename to display content: empsdata.txt
------------------------------
1:jitendra:10k:trainer
2:kunal:20k:sde
------------------------------
jitendra@PC:~$
```

Q Write a Script to Read File Name from the End User and Remove Blank Lines Present in that File?

```
#! /bin/bash
 read -p "Enter any File name to remove blank lines:" fname
 grep -v "^$" $fname > temp.txt
```

```
mv temp.txt $fname
echo "In $fname all blank lines deleted"
```

Write a Script to Read File Name from the End User and Remove Duplicate Lines Present in that File?

```
#! /bin/bash
read -p "Enter any File name to remove duplicate lines:" fname
sort -u $fname > temp.txt
mv temp.txt $fname
echo "In $fname all duplicate lines deleted"
```

----------Operators----------

1)Arithmetic Operators

+ □ Addition
- □ Substraction
* □ Multiplication
/ □ Division
% □ Modulo Operator

2)Relational Operators: (Numeric Comparison Operators)

-gt □Greater than
-ge □Greater than or equal to
-lt □Less than
-le □Less than or equal to
-eq □ Is equal to
-ne □Not equal to

3)Logical Operators:

-a □ Logical AND
-o □Logical OR
! □Logical Not

4)Assignment operator =

Note: Except assignment operator, for all operators we have to provide space before and after operator.

How to perform Mathematical Operations:

There are multiple ways
1) By using expr keyword

2) By using let keyword
3) By using (())
4) By using []

1) By using expr Keyword:

expr means expression

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter First Number:" a
read -p "Enter First Number:" b
sum=$(expr $a + $b)
echo $sum
jitendra@PC:~$ ./code.sh
Enter First Number:3
Enter First Number:5
8
```

Note: While using expr keyword, $ symbol is mandatory.
Space must be required before and after + symbol.

2) By using let Keyword:

```
let sum=a+b
echo "The Sum: $sum"
let sum=$a+$b
echo "The Sum: $sum"
```

Here $ symbol is optional.
But we should not provide space before and after +

3) By using (())

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter First Number:" a
read -p "Enter First Number:" b
sum=$((a+b))
echo $sum
jitendra@PC:~$ ./code.sh
Enter First Number:3
Enter First Number:4
7
jitendra@PC:~$
```

Here space and $ symbol, both are optional.

4)By using []:

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter First Number:" a
read -p "Enter First Number:" b
sum=$[a+b]
echo "The Sum: $sum"
sum=$[$a+$b]
echo "The Sum: $sum"

jitendra@PC:~$ ./code.sh
Enter First Number:3
Enter First Number:5
The Sum: 8
The Sum: 8
jitendra@PC:~$
```

Here also space and $ symbol, both are optional.

Note:- All the above 4 approaches will work only for integral arithmetic

(only for integer numbers).
If we want to perform floating point arithmetic then we should use **bc** command.

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter First Number:" a
read -p "Enter First Number:" b
sum=$(echo $a+$b | bc)
echo "The Sum: $sum"
echo "The difference is $(echo $a-$b | bc)"
echo "The Product is $(echo $a*$b | bc)"
jitendra@PC:~$ ./code.sh
Enter First Number:3.7
Enter First Number:43.2
The Sum: 46.9
The difference is -39.5
The Product is 159.8
jitendra@PC:~$
```

Q. Write a Script to Read 4 Digit Integer Number and Print the Sum of Digits Present in that Number?

```
jitendra@PC:~$ cat code.sh
#! /bin/bash
read -p "Enter First Number:" n
a=$(echo $n | cut -c 1)
b=$(echo $n | cut -c 2)
c=$(echo $n | cut -c 3)
d=$(echo $n | cut -c 4)

echo "sum of digits is : $[a+b+c+d]"
jitendra@PC:~$ ./code.sh
Enter First Number:5464
sum of digits is : 19
jitendra@PC:~$
```

Q Write a Script to Read Employee Monthly Salary and Print his Bonus. The Bonus should be 20% of Annual Salary

```
jitendra@PC:~$ cat code.sh
#!/bin/bash
read -p "Enter your monthly salary" sal
yearly=$(echo "12*$sal" | bc)
bonus=$(echo "${yearly}*20/100" | bc)
echo "your bonus is $yearly"
```

--------Control Statements---------

1) if statement
2) case statement
3) while loop
4) for loop
5) until loop
6) break
7) continue
8) exit

1)if Statement:

There are 4 types of if statements
a) simple if
b) if-else
c) nested if
d) ladder if

a) simple if:-

```
if [ condition ]
   then
      action
 fi
```

If condition is true then only action will be executed.

Q1. write a script to read name from the end user and if name is "jitendra" then display a special message

b) if -else:

```
if [ condition]
then
action if condition is true
else
action if condition is false
fi
```

c) Nested if:

```
if [ condition ]
      then
            .........
            .........
            if [ condition ]
                  then
                  .........
                  .........
            else
                  ........
            fi
                  .....
else
     .....
fi
```

d) ladder -if:

```
if [condition]
then
 action-1
elif [ condition ]
then
 action-2
elif [ condition ]
then
 action-3
else
 default action
fi
```

Q. WAS to find greater of 2 numbers

```
#!/bin/bash
read -p "Enter first number " a
read -p "Enter Second number " b

if [ $a -gt $b ]
then
echo "$a is greater"
else
echo "$b is greater"
fi
```

Q. Write a Script to Check whether Numbers OR Equal OR Not. If the Numbers are not Equal then Print Greater Number?

```
#!/bin/bash
read -p "Enter first number " a
read -p "Enter Second number " b
if [ $a -eq $b ]
then
echo "both are equal"
elif [ $a -gt $b ]
then
echo "$a is greater"
else
echo "$b is greater"
fi
```

Q. write a script to find greater of 3 numbers

first method

```
#!/bin/bash
read -p "Enter first number " a
read -p "Enter Second number " b
read -p "Enter third number " c
if [ $a -gt $b ]
then
    if [ $a -gt $c ]
    then
    echo "$a is greater"
    else
    echo "$c is greater"
    fi
elif [ $b -gt $c ]
then
echo "$b is greater number"
else
echo "$c is greater number"
fi
```

second method

```
1 #!/bin/bash
2 read -p "Enter first number " a
3 read -p "Enter Second number " b
4 read -p "Enter third number " c
5 if [ $a -eq $b -a $a -gt $c ]
6 then
7 echo "$a is greater number"
8 elif [ $b -gt $c ]
9 then
10 echo "$b is greater number"
11 else
12 echo "$c is greater number"
13 fi
```

Q. write a script to read marks of 3 subjects if any of subject
R. have marks less than 33 then print student failed

method 1

```
1 #!/bin/bash
2 read -p "Enter first number " a
3 read -p "Enter Second number " b
4 read -p "Enter third number " c
5
6 if [ $a -gt 33 -a $b -gt 33 -a $c -gt 33 ]
7 then
8 echo "passed"
9 else
10 echo "failed"
11 fi
```

It should be -ge

method 2

```
#!/bin/bash
read -p "Enter first number " a
read -p "Enter Second number " b
read -p "Enter third number " c

if [ $a -lt 33 ]
then
echo "failed"
elif [ $b -lt 33 ]
then
echo "failed"
elif [ $c -lt 33 ]
then
echo "failed"
else
echo "passed"
fi
```

Exit Codes/ Status Codes:

Every command and script return some value after execution, which indicates that whether it is successfully executed or not. This return value is called exit code or status code.

We can find exit code by using "$?".

zero means command/script executed successfully.

non-zero means command/scrit not executed successfully.

```
jitendra@PC:~$ echo $0
-bash
jitendra@PC:~$ echo $?
0
jitendra@PC:~$ echopico
echopico: command not found
jitendra@PC:~$
jitendra@PC:~$ echo $?
127
jitendra@PC:~$ echo $?
0
jitendra@PC:~$
```

Q. Write a Script that takes 2 Integer Numbers as Command Line Arguments and Prints Sum
If Number of Arguments is not 2, then we have to get Message saying "You should provide only 2 Arguments"
If the Arguments are not Integer Numbers then we have to get Message saying "You should provide Integer Numbers only"

```
#! /bin/bash
if [ $# -ne 2 ]
then
echo "you should provide exactly two args"
exit 1
fi
x=$1
y=$2
sum=$( expr $x + $y )
if [ $? -ne 0 ]
then
echo "you should provide ints only"
exit 2
else
echo "the sum is $sum"
fi
```

Write a Script that Reads an Integer Number and Check whether the given Number is +ve Number OR -ve Number?

Write a Script that Reads an Integer Number and Checks whether it is Even Number OR not?

Write a Script that Reads an Integer Number and Checks whether it is 3 Digit Number OR not?

File Test Options :-

-e Returns true if file/directory exists
-s Returns true if the file is non-empty
-f Returns true if the file is a regular file
-d Returns true if the file is a directory

-l Returns true if the file is link file
-b Returns true if the file is block special file
-c Returns true if the file is character special file

-r Returns true if current user has read permission on the file
-w Returns true if current user has write permission on the file
-x Returns true if current user has execute permission on the file
-o Returns true if current user is owner of the file.
file1 -ot file2 Returns true if file1 is older than file2 (last modified time)
file1 -nt file2 Returns true if file1 is newer than file2(last modified time)

Q. Script to Test whether the given File is Regular File OR Directory?

```
1 #! /bin/bash
2 read -p "Enter File Name to test:" fname
3 if [ -e $fname ]; then
4  if [ -f $fname ]; then
5  echo "It is regular file"
6  elif [ -d $fname ]; then
7  echo "It is Directory file"
8  else
9  echo "It is special file"
10 fi
11else
12 echo "$fname does not exist"
13fi
14
```

Write a Script that Reads a File Name and Display its Content to the Terminal

```
1 #! /bin/bash
2 read -p "Enter File Name to test:" fname
3 if [ -e $fname ]; then
4  if [ -f $fname ]; then
5  if [ -r $fname ]; then
6  cat $fname
7  else
8  echo "User not having Read permission"
9  fi
10 else
11 echo "It is not a regular file"
12 fi
13else
14 echo "$fname does not exist"
15fi
16
```

q. Write a Script that Reads File Name and Check whether it is Empty File OR not?

```
#!/bin/bash
 read -p "Enter File Name to test:" fname
 if [ -e $fname ]; then
 if [ -f $fname ]; then
 if [ -s $fname ]; then
 echo "$fname is not empty file"
 else
 echo "$fname is empty file"
 fi
else
echo "It is not a regular file"
fi
else
echo "$fname does not exist"
fi

```

Q write a script to check wheather 2 files are same or not

```
#!/bin/bash
 read -p "Enter file1: " file1
 read -p "Enter file2: " file2

 result=$(cmp $file1 $file2)
if [ -z "$result" ]
then
echo "The given 2 files have same content"
else
echo "given 2 files have different content"
fi

```

q Write a Script that Accepts a File Name and Display User Permissions?

```
#! /bin/bash
 read -p "Enter First File Name: " fname
 READ=NO
 WRITE=NO
 EXECUTE=NO
 if [ -r $fname ]; then
 READ=YES
 fi
 if [ -w $fname ]; then
WRITE=YES
fi
if [ -x $fname ]; then
EXECUTE=YES
fi
echo "User Permissions Summary"
echo "————————————————————————"
echo "Read Permission: $READ"
echo "Write Permission: $WRITE"
echo "Execute Permission: $EXECUTE"

```

Write a Scirpt that Reads File Name and Remove the specified File?

```
#! /bin/bash
read -p "Enter file/directory name to delete:" fname
if [ -e $fname ]; then
 rm -r $fname
 echo "$fname removed successfully"
else
 echo "$fname does not exist"
fi

```

Mini Application:

Copy all files and directories present in the first directory to the second directory. We should create compressed tar file and have to move that tar file.

After moving tar file to the second directory, extract all files and directories and remove that tar file.

Tests to Perform:
1) The number of command line arguments should be 2
2) The source and destination directories should be available already
3) The source and destination arguments should be directories
4) The user should has read and execute permissions on source directory
5) The user should has write and execute permissions on destination directory
6) All error messages should be send to error file and the file name should contain timestamp.
7) All intermediate steps should be displayed to the terminal

String Test Options:

1) str1 = str2 Returns true if both strings are same

2) str1 != str2 Returns true if both strings are different

3) -z str Returns true if the string is empty

4) -n str Returns true if the string is not empty

5) str1 > str2 Returns true if the str1 is alphabetically greater than str2

5) str1 < str2 Returns true if the str1 is alphabetically less than str2

2)case Statement:

If multiple options are available then it is not recommended to use nested if-else statements. It increases length of the code and reduces readability.
To handle such type of requirements we should go for case statement

Syntax:

```
case $variable in
 option1 )
 action-1
 ;;
 option2 )
 action-2
 ;;
 option3 )
 action-3
 ;;
 * )
 default action
 ;;
 esac
```

Note:

1) space is optional while defining options.
2) ;; can be used to come out of case statement.
3) ;; is mandatory for all options except for default option.
4) If we take default option at the beginning, then for any input the same default option will be executed.

Q. take a digit from 0 to 9 and print the digit number

```
#! /bin/bash
read -p "Enter any digit from 0 to 9 : " n
case $n in
0)
echo "Zero"
echo "Zero again"
;;
1) echo "One" ;;
2) echo "Two" ;;
3) echo "Three";;
4) echo "Four" ;;
5) echo "Five" ;;
6) echo "Six";;
7) echo "Seven" ;;
8) echo "Eight" ;;
9) echo "Nine" ;;
*) echo "Please enter a digit from 0 to 9 only"
esac
```

Q read a charactor from user input then check wheater it is digit , alphabet or special symbol

```
#! /bin/bash
read -p "Enter any charactor to check " n
case $n in
[A-Za-z])
echo "It is an Alphabet symbol"
;;
[0-9])
echo "it is a digit"
;;
# [^a-zA-Z0-9])
*)
echo "It is a Special Symbol"
;;
esac
```

^ symbol is used for exclude

Q. Write a Script that Accepts a Single Characte r and Checks whether it is Digit OR Special Character OR Vowel OR Consonent?

```
#! /bin/bash
read -p "Enter Any Character to check: " ch
case $ch in
 [^a-zA-Z0-9])
 echo "It is a Special Character"
 ;;
 [0-9]) echo "It is a Digit"; echo "digit"s
 ;;
 [aeiouAEIOU])
 echo "It is a Vowel"
 ;;
 [^aeiouAEIOU])
 echo "It is a Consonent"
 ;;
 *)
 echo "Enter only one character"
esac
```

if we write multiple comands in same line by seperating using ;

Iterative Statements:

If we want to execute a group of commands multiple times then we should go for iterative statements.
There are 3 types of iterative statements
1) while loop
2) until loop
3) for loo

1) while Loop:

If we don't know the number of iterations in advance, then we should go for while

loop.
Syntax:
while [condition]
do
body
done
As long as condition is true,
then body will be executed. Once condition fails then only
loop will be terminated

Q. Write a Script to Print Numbers from 1 to 10

```
js@Jitendra:~$ cat code.sh
#!/bin/bash
i=1
while [ $i -ne 10 ]
do
echo $i
i=$[i+1]
done

js@Jitendra:~$ ./code.sh
1
2
3
4
5
6
7
8
9
```

Use echo -n if you want to print in same line

Write a Script to generate Numbers until Limit which is provided by End User?

```
1 #!/bin/bash
2  read -p "Enter Limit:" n
3  i=1
4  while [ $i -le $n ]
5  do
6  echo $i
7  sleep 2
8  let i++
9  done
```

Q3) Write a Script to find the Sum of First n Integers, where n is provided by End User

```
1 #! /bin/bash
2 read -p "Enter n value:" n
3 i=1
4 sum=0
5 while [ $i -le $n ]
6 do
7  let sum=sum+i
8  let i++
9 done
10 echo "The Sum of first $n numbers: $sum"
11
```

Write a Script to Display Timer (Digital Timer)

```
#! /bin/bash
while [ true ]
do
 clear
 printf "\n\n\n\n\n\n\t\t\t\t$(date +%H:%M:%S)"
 sleep 1
done


```

Note: To use escape characters like \n and \t, we should not use echo and we should use printf command

Note: true and false are keywords which represents boolean values.

break Statement

Based on some condition, if we want to break loop execution (i.e to come out of loop) then we should go for break statement

```
#! /bin/bash
i=1
while [ $i -le 10 ]
do
 if [ $i -eq 5 ]; then
 break
 fi
 echo $i
 let i++
done

```

Write a script to run timer till given hour and minute

```
#! /bin/bash
while [ true ]
do
 clear
 printf "\n\n\n\n\n\n\t\t\t\t$(date +%H:%M:%S)"
 sleep 1
 h=$(date +%H)
 m=$(date +%M)
 if [ $h -eq 8 -a $m -eq 54 ]; then
 break
 fi
done

```

continue Statement:

We can use continue statement to skip current iteration and continue for the next iteration.

```
#! /bin/bash
i=0
while [ $i -lt 10 ]
do
 let i++
 if [ $[i%2] -eq 0 ]; then
 continue
 fi
 echo $i
done

```

```
js@Jitendra:~$ ./code.sh
1
3
5
7
9
js@Jitendra:~$
```

Write a Script to Read File Name and Display its Content?

```bash
#! /bin/bash

while [ true ]
do

read -p "enter file name to display data : " fname

if [ -f $fname ]
then
echo "————————————————————"
echo "The content of $fname"
echo "————————————————————"
cat $fname
else
echo "$fname does not exist"
fi

read -p "do you want to display another file [Yes|No]: " option
echo $option
case $option in
[Yy][eE][sS])
echo $option
;;

[nN][oO])
break
;;
esac

done
echo "thanks for using app"

```

Take a string as an input from user and print it's reverse

```
#! /bin/bash

read -p "Enter the string to reverse: " str

len=$(echo -n $str | wc -c)
ans=""

while [ $len -gt 0 ]
do
ch=$(echo -n $str | cut -c $len)
ans=$ans$ch
len=$[len-1]
done

echo "The original string  $str"
echo "the reversed string  $ans"

if [ $str = $ans ]
then
echo "palindrome"
fi

```

```
#! /bin/bash

read -p "Enter the string to reverse: " str

if [ $str = $(echo $str | rev) ]
then
echo "palindrome"
fi

```

2) Until loop :-

It is opposite to while loop.
Syntax:

```
until [ condition ]
do
 body
done
```

The body will be executed as long as condition returns false. Once condition returns true, then loop will be terminated.

Q1. print 1 to 5 using until loop

```
js@Jitendra:~$ cat code.sh
#!/bin/bash

i=1
until [ $i -gt 5 ]
do
echo $i
let i++
done
js@Jitendra:~$ ./code.sh
1
2
3
4
5
```

3) For loop:-

If we want to perform some action for every item in the given list, then we should go for for loop

Syntax:-

```
for variable in item1 item2 item3... itemN
do
 action
done
```

Q. Print q 1 to 5 using for loop

```
1 #!/bin/bash
2
3 for i in 1 2 3 4 5
4 do
5 echo $i
6 done
7
```

Ex : -

```
js@Jitendra:~$ cat code.sh
#!/bin/bash

for course in java linux python c++ dsa
do
echo $course
done
js@Jitendra:~$ ./code.sh
java
linux
python
c++
dsa
```

Q. Write a script that display numbers from 1 to 100 , which are divisible by 10

```
1 #!/bin/bash
2
3 for i in {1..100}
4 do
5 if [ $[i%10] -eq 0 ]
6 then
7 echo $i
8 fi
9 done
```

Q. Write a script to display all file names present in current working directory

```
1 #!/bin/bash
2
3 for name in *
4 do
5 if [ -f $name ]
6 then
7 echo $name
8 fi
9 done
```

Q. Write a script to append current date and time in every .txt file present in current Working directory

```
1 #!/bin/bash
2
3 for file in *.txt
4 do
5 date>>$file
6 done
```

Q. Write a script to print all command line arguments

```
#!/bin/bash

if [ $# -ne 0 ]
then

for arg in $@
do
echo "$arg"
done

else
echo "no command line arguments passed"
fi
```

```
js@Jitendra:~$ ./code.sh
no command line arguments passed
js@Jitendra:~$ ./code.sh c++ dsa linux
c++
dsa
linux
js@Jitendra:~$
```

Q. Write a script to display multiple files content to the terminal and all the File names Passed as command line arguments ?

```bash
#!/bin/bash

if [ $# -ne 0 ]; then

    for fname in $@; do
        if [ -f $fname ]; then
            echo "------File Content----------"
            cat $fname
            echo "--------------------------------"
        else
            echo "$fname is not regular file"
        fi
    done

else
    echo "no command line arguments passed"
fi

```

Q. Write a script to append multiple files content to a single file result.txt file names are passed as command Line arguments ?

```bash
#! /bin/bash
if [ $# -ne 0 ]; then
    for fname in $@; do
        if [ -f $fname ]; then
            cat $fname >>result.txt
        else
            echo "$fname does not exist or it is not a regular file"
        fi
    done
else
    echo "Please pass atleast one file name"
fi

```

Q. Write a Script to Display all Employees Information where Salary is Greater than 2500

emp.txt

```
102:Jitendra:100000:Bengaluru
121:Vikas:150000:Kanpur
133:Rajesh:50000:Delhi
111:Kartik:200000:Hyderabad
67:Muthu:120000:Chennei
```

```
#!/bin/bash
for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ':' -f 3)
    if [ $sal -gt 100000 ]; then
        echo $record
    fi
done

```

Q. Write a script to save all employees info where salary is greater than 2500 ans city is hyderabad to hyd.txt

```
#!/bin/bash
for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ':' -f 3)
    city=$(echo $record | cut -d ":" -f 4)
    if [ $sal -gt 100000 -a $city = "Hyderabad" ];
     then
        echo $record
    fi
done

```

Q. Write a script to display minimum and maximum salaries

```
#!/bin/bash
max=$(cat emp.txt | head -1 | cut -d ":" -f 3)
min=$(cat emp.txt | head -1 | cut -d ":" -f 3)
max_record=$(cat emp.txt | head -1)
min_record=$(cat emp.txt | head -1)

for record in $(cat emp.txt)
do
    sal=$(echo $record | cut -d ':' -f 3)
    if [ $sal -gt $max ];
     then
        max=$sal
        max_record=$record
    fi

    if [ $sal -lt $min ];
     then
        min=$sal
        min_record=$record
    fi
done

echo "maximum salary " $max
echo "Employee data " $max_record

echo "minimum salary " $min
echo "Employee data " $min_record

```

Alternative Syntax of for loop:

```
for((i=1;i<10;i++))
do

done
```

eg :-print using for loop

```
#!/bin/bash
read -p "enter a number n " n
for((i=0;i<n;i++))
do
echo $i
done
```

eg :- display table using loop

```
#!/bin/bash
read -p "enter a number n " n
for((i=0;i<=10;i++))
do
echo "$n * $i = $[n*i]"
done
```

Q. check wheather a given number is prime or not

```bash
#!/bin/bash
read -p "enter a number n " n
prime=true
for ((i = 2; i <= $n / 2; i++)); do
    if [ $((n % i)) -eq 0 ]; then
        prime=false
        break
    fi
done

echo $prime
```

Q. print triangle pattern

```bash
#!/bin/bash
read -p "enter a number n " n
for ((i = 0; i < $n; i++)); do
    for ((j = 0; j <= $i; j++)); do
        echo -n "*"
    done
    echo ""
done
```

----------Arrays----------

IF we want to represent a group of values with a single name then we should go for arrays

1. **Declaring arrays**
   declare -a arrayname
2. **Initializing arrays**
   courses=(java c++ linux python)
3. **Assigning values**
   courses[0]=java

courses[1]=python
courses[2]=linux

4. Accessing Elements : -

${courses[i])} ----------> ith element
${courses[@]} ----------> All elements present inside array
${courses[@]} ----------> All elements present inside array separated by IFS
${!courses[@]} ----------> All elements where elements are available
${#courses[@]} ----------> It returns the number of elements present inside array.
${#courses[0]} ----------> It returns the length of first element.

```
#!/bin/bash

course[0]=java
course[1]=python
course[2]=c++
course[3]=Django
course[6]=Node

echo "first element : ${course[0]}"
echo "all elements using @ : ${course[@]}"
echo "all elements using * : ${course[*]}"
echo "all indices where elements are avaialble : ${!course[@]}"
echo "total number of elements : ${#course[@]} "
echo "total length of first element : ${#course[0]}"

#finding variable value length using #
d="good book"
len=${#d}
echo $len
```

```
pc@Jitendra:~$ ./code.sh
first element : java
all elements using @ : java python c++ Django Node
all elements using * : java python c++ Django Node
all indices where elements are avaialble : 0 1 2 3 6
total number of elements : 5
total length of first element : 4
9
pc@Jitendra:~$
```

Q. print array elements using while loop, for loop

```bash
#!/bin/bash
fruits=("Apple" "Orange" "Banana" "Mango")
size=${#fruits[@]}
i=0

echo "All elements by using while loop:"
echo "————————————————————————————————"
while [ $i -lt $size ]
do
echo ${fruits[$i]}
let i++
done

echo "All elements by using for loop:"
echo "————————————————————————————————"

for fruit in ${fruits[@]}
do
echo $fruit
done

echo "All elements by using advanced for loop:"
echo "————————————————————————————————"

for((i=0;i<${#fruits[@]};i++))
do
echo ${fruits[$i]}
done
```

```
pc@Jitendra:~$ ./code.sh
All elements by using while loop:
---------------------------------
Apple
Orange
Banana
Mango
All elements by using for loop:
---------------------------------
Apple
Orange
Banana
Mango
All elements by using advanced for loop:
---------------------------------
Apple
Orange
Banana
Mango
pc@Jitendra:~$
```

Q2 Write a script for accessing Array elements by using for loop indices are random

```bash
#!/bin/bash

# declare -a fruits
fruits[1]="mango"
fruits[23]="banana"
fruits[34]="apple"

echo "----loop on elements----"
for fruit in ${fruits[@]}
do
echo $fruit
done

echo "----loop on index----"
for idx in ${!fruits[@]}
do
#echo $fruits[$idx] will not work
#here
echo ${fruits[$idx]}
done
```

```
pc@Jitendra:~$ ./code.sh
----loop on elements----
mango
banana
apple
----loop on index----
mango
banana
apple
pc@Jitendra:~$
```

5. Removing array elements

Yes we can remove array elements using unset command

unset arr[10]

6. reading the values into array

```
read -p "Enter the number of values: " n

for ((i = 0, j = 1; i < n; i++)); do
    read -p "Enter the number $((j++)): " nums[${i}]
done

for ((i = 0; i < n; i++)); do
    echo ${nums[${i}]}
done
```

Q. read n numbers and then print sum of even and odd numbers

```
read -p "enter size of array " n
for ((i = 0; i < n; i++)); do
    read nums[$i]
done

oddsum=0
evensum=0

for ((i = 0; i < n; i++)); do
    if [ $((nums[$i] % 2)) -ne 0 ]; then
        oddsum=$((oddsum + nums[$i]))
    else
        evensum=$((evensum + nums[$i]))
    fi
done

echo "odd numbers sum = $oddsum"
echo "even numbers sum = $evensum"
```

Q. write a script to store all .txt file names present in current directory in to an array

```
# store name of all text file
# in our current directory
# to array
i=0
for file in *.txt;
do
    list[i++]=$file
done

for ((i = 0; i < ${#list[@]}; i++));
do
    echo ${list[$i]}
done
```

----------Functions----------

If any group of commands are repeatedly required, then it is not recommended to write these commands separately everytime. It increases length of the code and reduces readability.
Such type of repeated code we have to define inside a block and we can call that block where ever required. This block of commands is nothing but function.
Hence function is nothing but a block of repeatable commands.

Advantages of Functions:
1) It reduces length of the code.
2) It improves readability.
3) It improves maintainability.
4) It promotes DRY principle.
DRY Don't Repeat Yourself

1. Defining a Function

1st Way:

```
function function_name()
{
 commands
}
```

2nd Way:

```
function_name()
{
 commands
}
```

2. Calling a Function

function_name param1 param2 param3 ....

```bash
#!/bin/bash

#defining function 1
wish1()
{
    echo "Hello how are you doing ?"
}
#defining function 2
function wish2()
{
    echo "Hellow how are you ?, again."
}

#calling functions
wish1

wish2

```

eg 2:-

```
#!/bin/bash

f1()
{
    echo "I am in function 1 1"
}

f2()
{
    echo "I an in function 2"
    f1
}

f2

```

```
pc@Jitendra:~$ ./code.sh
I an in function 2
I am in function 1 1
pc@Jitendra:~$
```

3. functions with parameters :

function can accept parameters also. with in the function we can access parameters as follows :

$1 ------> first parameter
$2 ------> second parameter
$@ ------> All parameters
$* ------> All parameters
$# ------> Total number of parameters
$0 ------> It is script name but not function name

```bash
#!/bin/bash

demo()
{
    echo "first parameter : $1"
    echo "second parameter : $2"
    echo "total numuber of parameters : $#"
    echo "all parameters with @ :  $@"
    echo "script name : $0"

}

demo 10 20 30 40

```

```
pc@Jitendra:~$ ./code.sh
first parameter : 10
second parameter : 20
total numuber of parameters : 4
all parameters with @ :  10 20 30 40
script name : ./code.sh
pc@Jitendra:~$
```

eg :-

```bash
#!/bin/bash

wish()
{
    if [ $# -eq 0 ]
    then
    echo "Hello Guest Good Evening"
    else
    echo "Hello $1 good evening"
    fi
}

wish
wish "Jitendra Singh"

```

```
pc@Jitendra:~$ ./code.sh
Hello Guest Good Evening
Hello Jitendra Singh good evening
pc@Jitendra:~$
```

Q. **write a function to do arithmetic operators of two operators**

```bash
#!/bin/bash

calc() {
    if [ $# -ne 2 ]; then
        echo "You should pass exactly 2 arguments"
    else
        a=$1
        b=$2
        echo "$a+$b = $((a + b))"
        echo "$a-$b = $((a - b))"
        echo "$a*$b = $((a * b))"
        echo "$a/$b = $((a / b))"
        echo
    fi
}
calc 10
calc 20 10
calc 200 100
calc 2000 1000

```

Q. write a function to print all parameters of a function

```bash
#!/bin/bash

parameter_printing() {
    if [ $# -eq 0 ]; then
        echo "No parameters passed to this function"
    else
        echo "All Passed Parameters are:"
        echo "........................."
        for p in $@; do
            echo $p
        done
    fi
}

```

Q. Find maximum of 2 using function

```bash
#!/bin/bash

max()
{
    if [ $1 -gt $2 ]
    then
    echo "The max of $1 and $2 is " $1
    else
    echo "The max of $1 and $2 is " $2
    fi
}

max 10 20
max 200 100

```

Q. write a function to find factorial

```bash
#! /bin/bash
factorial() {
    original=$n
    fact=1
    while [ $n -gt 1 ]; do
        let fact=fact*n
        let n--
    done
    echo "The Factorial of $original is: $fact"
}
read -p "Enter a number to find factorial:" n
factorial $n

```

Q. Find factorial of 1st 10 natural number

```bash
#! /bin/bash
factorial() {
    original=$1
    n=$1
    fact=1
    while [ $n -gt 1 ]; do
        let fact=fact*n
        let n--
    done
    echo "The Factorial of $original: $fact"
}
for i in {1..10}; do
    factorial $i
done

```

Q. write a program to generate all prime numbers till given number

```bash
#! /bin/bash
prime_numbers() {
    for ((n = 2; n <= $1; n++)); do
        is_prime="Yes"

        for ((i = 2; i < n; i++)); do
            if [ $((n % i)) -eq 0 ]; then
                is_prime="No"
                break
            fi
        done

        if [ $is_prime = Yes ]; then
            echo $n
        fi
    done
}

read -p "Enter N value : " n
prime_numbers $n

```

```
pc@Jitendra:~$ ./code.sh
Enter N value : 10
2
3
5
7
pc@Jitendra:~$
```

Variable Scope :-

Bydefault every variable in shell script is global. i.e we can access anywhere in our script. But before using a variable, it should be declared already.

Eg 1:

```
#! /bin/bash
f1()
{
 echo "x value : $x"
}
x=10
f1
```

Output:

```
 x value : 10
```

Eg 2:

```
#! /bin/bash
f1()
{
 x=20
 echo "x value : $x"
}
x=10
f1
echo "After f1 execution x value: $x"
```

Output:

```
x value : 20
```

After f1 execution x value: 20

Eg 3:

```
#! /bin/bash
f1()
{
 echo "x value : $x"
}
f1
x=10
f1
```

Output:

```
x value :
x value : 10
```

--->The variables which are declared inside a function, can be accessed outside of that function, because every variable has global scope by default.

---->If we want a variable only within the function and should not be available outside of that function, then we have to use local keyword for the variable.
local variables can be accessed only inside function and cannot be accessed outside of that function.

```bash
#! /bin/bash

f1()
{
    local x=10
    echo "inside function x is " $x
}

f1

echo "outside function x is " $x

```

```
pc@Jitendra:~$ ./code.sh
inside function x is  10
outside function x is
pc@Jitendra:~$
```

Return Statement in Functions:

Every function in shell scripting returns some value. The default returned value is the exit code of the last command inside function.
But based on our requirement we can return values explicitly by using return statement.
return <exitcode>
The allowed values for exitcode are 0 to 255.
0 means successful
non-zero means unsuccessful.
We can get return value of function by using $?.

eg:-

```
#! /bin/bash
sum()
{
 if [ $# -ne 2 ]; then
 echo "You should pass exactly two numbers"
 return 1
 else
 echo "The SUM:$(($1+$2))"
 fi
}
sum 10 20
echo "The Return value of this function:$?"
echo
sum 10
echo "The Return value of this function:$?"

```

```
pc@Jitendra:~$ ./code.sh
The SUM:30
The Return value of this function:0

You should pass exactly two numbers
The Return value of this function:1
pc@Jitendra:~$
```

Use Case:

```
backup()
{
 commands to take backup
}

backup
if [ $? != 0 ]; then
 something goes wrong backup failed
else
 backup successful
```

fi

break vs exit vs return:

1) break:

We can use break only inside loops to break loop exeuction. we will come out of the loop and remaining statements after loop will be executed.

2) exit:

We can use anywhere exit statement to terminate script execution.The script will be terminated. No chance of executing any other statement.

3) return:

We can use return statement only inside function to stop function execution. After return statement, the remaining statements inside function won't be exeuction. But after function call the remaining statements will be executed.

How to call functions present in another Script:

Script file 1 with functions defined

mycode.sh

```
#! /bin/bash
x=888
y=999
add() {
    echo "$1 + $2 = $(($1 + $2))"
}
multiply() {
    echo "$1 * $2 = $(($1 * $2))"
}
subtract() {
    echo "$1 - $2 = $(($1 - $2))"
}
divide() {
    echo "$1 / $2 = $(($1 / $2))"
}
```

code.sh

```
#! /bin/bash
# syntax for including another script
. ./mycode.sh

add $x $y
add 10 20
subtract 20 10
multiply 10 20
divide 20 10

echo "The value of x : $x"
echo "The value of y : $y"
```

GCC compiler :-

GCC, or GNU Compiler Collection, is a free and open-source compiler system developed by the GNU Project. It is widely used for compiling code written in a variety of programming languages, including C, C++, Objective-C, Fortran, Ada, and others. GCC is the default compiler for most Linux distributions.

GCC is a collection of compilers, each of which supports one or more programming languages. These compilers are invoked by the command line interface. The basic syntax for compiling a C program with GCC is:

gcc [options] filename.c -o executable_name

In this command, [options] represents any additional compiler options that you may want to use, filename.c is the name of the C source file, and executable_name is the desired name of the output executable file.

GCC supports a wide variety of options that allow you to control the behavior of the compiler, optimize your code, and generate debugging information. Some of the most commonly used options include:

-c: This option tells GCC to compile the source file(s) without linking them to produce an object file. This is useful when you want to compile multiple source files separately and then link them together later. For example, if you have two source files main.c and helper.c, you can compile them separately using gcc -c main.c and gcc -c helper.c, and then link them together using gcc main.o helper.o -o program.

-O: This option enables optimization of the code generated by the compiler. The level of optimization can be specified by appending a number to the option, such as -O1, -O2, or -O3. The higher the optimization level, the more aggressive the optimizations performed by the compiler, but also the longer the compilation time. Optimization can result in faster and more efficient code, but can also make debugging more difficult.

While optimization can improve program performance and efficiency, there are several reasons why it may not always be desirable to use it:

Increased compilation time: Enabling optimization can significantly increase the time it takes to compile your code. This is because the compiler has to perform additional analysis and transformations on your code, which can be time-consuming. In some cases, the increased compilation time may be unacceptable, especially if you are developing a large project or need to iterate quickly.

Increased code complexity: Optimization can make your code more complex and harder to understand. This is because the optimizations performed by the compiler can change the structure and behavior of your code, making it more difficult to reason about. This can make debugging and maintenance more difficult, especially for inexperienced developers.

Increased memory usage: Some optimizations, such as loop unrolling and function inlining, can increase the amount of memory used by your program. This can be a problem if you are developing for a resource-constrained environment or if your program needs to run on a large number of machines.

Possible loss of portability: Some optimizations may be specific to a particular CPU architecture or operating system, and may not work correctly on other platforms. This can lead to bugs and compatibility issues if you need to port your code to other systems.

Potential introduction of bugs: Optimization can sometimes introduce subtle bugs into your code, especially if you are using non-standard or undefined behavior. This is because some optimizations may rely on assumptions about the behavior of your code that are not guaranteed by the language specification. As a result, it is important to thoroughly test your optimized code to ensure that it behaves correctly.

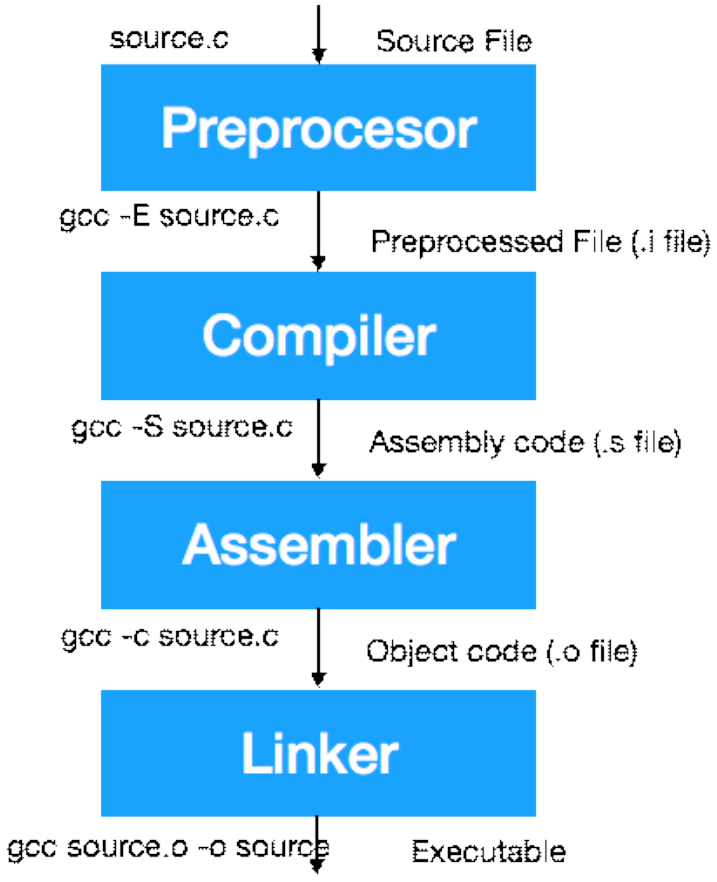
For these reasons, it is often a good idea to use optimization selectively and to test your code thoroughly before deploying it. You should also be aware of the trade-offs involved in optimization and weigh the benefits against the potential costs.

-g: This option generates debugging information in the output file. This information can be used by debugging tools to help you trace the execution of your program and find bugs. The level of debugging

information can be specified by appending a number to the option, such as -g1, -g2, or -g3. The higher the level, the more detailed the debugging information, but also the larger the output file.

-Wall: This option enables all warning messages produced by the compiler. Warnings are messages that indicate potential problems in your code, such as uninitialized variables, unused parameters, or type mismatches. Enabling warnings can help you catch errors early and improve the quality of your code.

-std=<standard>: This option specifies the version of the programming language standard that the compiler should use to compile the code. For example, -std=c11 specifies that the compiler should use the C11 standard, while -std=c++17 specifies that the compiler should use the C++17 standard. Using the correct standard can help ensure that your code is portable and compatible with different compilers and platforms.



What is Swap?

Swap is a portion of hard drive storage that has been set aside for the operating system to temporarily store data that it can no longer hold in RAM. This lets you increase the amount of information that your server can keep in its working memory, with some caveats. The swap space on the hard drive will be used mainly when there is no longer sufficient space in RAM to hold in-use application data.

The information written to disk will be significantly slower than information kept in RAM, but the operating system will prefer to keep running application data in memory and use swap for the older data. Overall, having swap space as a fallback for when your system's RAM is depleted can be a good safety net against out-of-memory exceptions on systems with non-SSD storage available.

step 1 check already available swap information

```
pc@Jitendra:/$ sudo swapon --show
NAME      TYPE       SIZE USED PRIO
/dev/sdb  partition    1G   0B   -2
/swapfile file      1024M   0B   -3
pc@Jitendra:/$
```

step 2 check available space on hard disk

```
pc@Jitendra:/$ df -h
Filesystem      Size  Used Avail Use% Mounted on
none            1.7G  4.0K  1.7G   1% /mnt/wsl
none            224G   63G  161G  28% /usr/lib/wsl/drivers
none            1.7G     0  1.7G   0% /usr/lib/wsl/lib
/dev/sdc        251G  2.8G  236G   2% /
none            1.7G   80K  1.7G   1% /mnt/wslg
rootfs          1.7G  1.9M  1.7G   1% /init
none            1.7G  4.0K  1.7G   1% /run
none            1.7G     0  1.7G   0% /run/lock
none            1.7G     0  1.7G   0% /run/shm
none            1.7G     0  1.7G   0% /run/user
tmpfs           1.7G     0  1.7G   0% /sys/fs/cgroup
none            1.7G   88K  1.7G   1% /mnt/wslg/versions.txt
none            1.7G   88K  1.7G   1% /mnt/wslg/doc
drvfs           224G   63G  161G  28% /mnt/c
drvfs           931G  654G  277G  71% /mnt/d
pc@Jitendra:/$
```

now see where is our root directory is located
in our case it is at /dev/sdc
and we have plenty of available space

step 3 create a swap file

The best way of creating a swap file is with the fallocate program. This command instantly creates a file of the specified size.

```
pc@Jitendra:/$ sudo fallocate -l 2G swapfile2
```

```
pc@Jitendra:/$ ls -l swap*
-rw------- 1 root root 1073741824 Mar 27 21:49 swapfile
-rw-r--r-- 1 root root 2147483648 Mar 27 22:08 swapfile2
```

step 4 change permission

Now that we have a file of the correct size available, we need to actually turn this into swap space.

----> First, we need to lock down the permissions of the file so that only users with root privileges can read the contents. This prevents normal users from being able to access the file, which would have significant security implications.

```
pc@Jitendra:/$ sudo chmod 600 swapfile2
```

step 5 now mark the file as swap space

```
pc@Jitendra:/$ sudo mkswap swapfile2
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=500be28b-559c-449f-81a0-b9d91632268d
```

step 6 after marking the file as swap space file we can enable the swap file

```
pc@Jitendra:/$ sudo swapon swapfile2
pc@Jitendra:/$ sudo swapon --show
NAME       TYPE       SIZE USED PRIO
/dev/sdb   partition    1G   0B   -2
/swapfile  file      1024M   0B   -3
/swapfile2 file         2G   0B   -4
pc@Jitendra:/$
```

step 7 we check if swap space is added or not using free -h

```
pc@Jitendra:/$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.3Gi       336Mi       2.8Gi       2.0Mi       162Mi       2.8Gi
Swap:          4.0Gi          0B       4.0Gi
pc@Jitendra:/$
```

step 8 making the swap file permanent

Our recent changes have enabled the swap file for the current session. However, if we reboot, the server will not retain the swap settings automatically. We can change this by adding the swap file to our /etc/fstab file.

```
pc@Jitendra:~$ echo '/swapfile2 none swap sw 0 0' | sudo tee -a /etc/fstab
```

deleting swap space :-

```
pc@Jitendra:/$ sudo swapon /swapfile2
pc@Jitendra:/$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.3Gi       314Mi       2.8Gi       2.0Mi       187Mi       2.8Gi
Swap:          3.0Gi          0B       3.0Gi
pc@Jitendra:/$ sudo swapoff /swapfile2
pc@Jitendra:/$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.3Gi       312Mi       2.8Gi       2.0Mi       187Mi       2.8Gi
Swap:          1.0Gi          0B       1.0Gi
pc@Jitendra:/$
```

step 2 :- remove the file
rm swapfilename

step 3 :- remove entry from /etc/fstab

---

https://www.digitalocean.com/community/tutorials/create-a-partition-in-linux

https://www.digitalocean.com/community/tutorials/how-to-add-swap-space-on-ubuntu-22-04

Soft links , hardlinks

ln command – make links between files
Syntax :
ln [options] [-T] Target Link_Name : create a link to Target with name linkname
ln [options] Target : creates a link to the target in current dir.
ln [options] Target Directory : creates link to each Target in directory
ln [option] –t Directory Target : creates link to each Target in directory
By default it creates hardlinks, and each target must exist and by default each destination must not exist(linkname)

-s : symbolic links
-t : specifies the target directory in which to create links
-T : no target directory, treat linkname as normal file always.
Eg. : ln filename linkname
Ln –s filename linkname

// Alternate for creating a hard link is link command
Link filename linkname

Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.
ls -l command shows all the links with the link column shows number of links.

Links have actual file contents
Removing any link, just reduces the link count, but doesn't affect other links.
Even if we change the filename of the original file then also the hard links properly work.
We cannot create a hard link for a directory to avoid recursive loops.
If original file is removed then the link will still show the content of the file.
The size of any of the hard link file is same as the original file and if we change the content in any of the hard links then size of all hard link files are updated.
The disadvantage of hard links is that it cannot be created for files on different file systems and it cannot be created for special files or directories.

A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
ls -l command shows all links with first column value l? and the link points to original file.
Soft Link contains the path for original file and not the contents.
Removing soft link doesn't affect anything but removing original file, the link becomes "dangling" link which points to nonexistent file.
A soft link can link to a directory.
The size of the soft link is equal to the length of the path of the original file we gave. E.g if we link a file like ln -s /tmp/hello.txt /tmp/link.txt then the size of the file will be 14bytes which is equal to the length of the "/tmp/hello.txt".
If we change the name of the original file then all the soft links for that file become dangling i.e. they are worthless now.
Link across file systems: If you want to link files across the file systems, you can only use symlinks/soft links.

5 types of soft links

Absolute : they are given as absolute path from the root directory
Dangling: links for which target doesnot exist.
Messy: they contain unnecessary slashes or dots in path
Lengthy : uses "../" more than necessary in the path
Other_fs: links , whose target currently reside on a different file system.

Run level

Target Purpose
graphical.target: multiple user, GUI and text based login
Multi-user.target: multiple user and text based login no GUI
rescue.target: sulogin prompt, basis system initialization completed
emergency.target: sulogin prompt, initramfs pivot complete and system root mount on / read only

Runlevel	Target	Units	Description
0	runlevel0.target,	poweroff.target	Shut down and power off the system.
1	runlevel1.target,	rescue.target	Set up a rescue shell.
2	runlevel2.target,	multi-user.target	Set up a non-graphical multi-user system.
3	runlevel3.target,	multi-user.target	Set up a non-graphical multi-user system.
4	runlevel4.target,	multi-user.target	Set up a non-graphical multi-user system.
5	runlevel5.target,	graphical.target	Set up a graphical multi-user system.
6	runlevel6.target,	reboot.target	Shut down and reboot the system.

Commands to check runlevel
Runlevel : returns current runlevel
Command to show targets on current rn level
Systemctl list-units --type target