What is Unix ?

The Unix operating system is a set of programs that act as a link between the computer and the user.

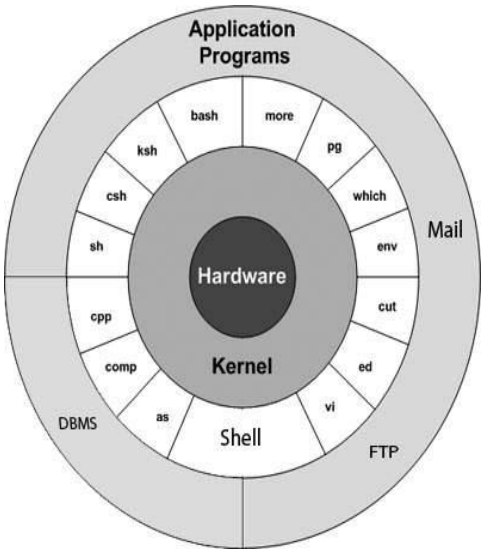
The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture

Here is a basic block diagram of a Unix system −



The main concept that unites all the versions of Unix is the following four basics −

- **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell** − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.

- **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous

others provided through 3rd party software. All the commands come along with various options.

- **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

System Information Commands And General Commands

1. Uname- / users
    - To know only the system name, you can use the **uname** command
    - Uname -n - To view your network hostname, use the **'-n'** switch with the uname command
    - Uname -v – kernel version
    - Uname -r – kernel release
    - Uname -m – machine hardware name
    - Uname -a – all
2. Hostname – To view Network hostname
3. Echo – to print the message on screen.
   Echo $SHELL
4. Cal –
   In Linux and Unix-like systems, the "**cal**" command is a command-line utility used to print calendars on the terminal. "**Cal**" is an abbreviated form of Calendar.
   Cal -m 3 – specific month display
   Cal -y 2019 – specific year display
   Cal - 3 – display current , previous and next year
   cal month year
   cal 07 2020
   cal 2020

   ncal -w – week number display

5. Man – manual for any command
   Man cal

6. Who - The who command is used **to get information about currently logged in user on to system**
   Whoami – name associated with current user.
   user login name, user's terminal, time of login as well as the host the user is logged in from

7. Ls – ls command is used to list the contents of the current working directory.
    - Ls [dir] - List files in another directory
    - Ls / - List files in the root directory
    - Ls .. - to list the contents of the parent directory one
    - ls ../.. for contents two levels above
    - ls ~ command to list the contents in the users's home directory
    - ls -d */ command to list only directories:
    - ls * command to list the contents of the directory with it's subdirectories
    - ls -R command to list all files and directories with their corresponding subdirectories down to the last file:

- ls -s command (the **s** is lowercase) to list files or directories with their sizes:
- ls -l command to list the contents of the directory in a table format with columns including:
    - content permissions
    - number of links to the content
    - owner of the content
    - group owner of the content
    - size of the content in bytes
    - last modified date / time of the content
    - file or directory name
- ls -lh command to list the files or directories in the same table format above, but with another column representing the size of each file/directory:
- ls -a command to list files or directories including hidden files or directories. In Linux, anything that begins with a . is considered a hidden file:
- ls -t command to list files or directories and sort by last modified date in descending order (biggest to smallest).
    - To reverse the sorting order like so: ls -tr
- ls -S (the **S** is uppercase) command to list files or directories and sort by size in descending order (biggest to smallest).
    - To reverse the sorting order like so: ls -Sr:
- ls > output.txt command to print the output of the preceding command into an output.txt file.

8. wc –

   It is used to find out number of **newline count**, **word count**, **byte and characters** count in a files specified by the file arguments.
   EG: The syntax of **wc** command as shown below.
   Wc tecmint.txt
   The '**wc**' command without passing any parameter will display three numbers **number of lines**, **number of words** and **112 number of bytes** of the file.
   - Wc file1 file2
   - Wc – l - To count number of newlines in a file use the option '**-l**', which prints the number of lines from a given file.
   - Wc -w - the following command to count the words in a file.
   - Wc -c or -m - **number of bytes** and **characters** respectively in a file.
   - wc -L - print out the length of longest (**number of characters**) line in a file
   - wc file1 file2\
   - ls | wc -l

9. head - **head** - display the beginning of a text file or piped data

   It prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.
   - Head file1
   - Head file1 file2
   - -n – specified number of lines from beginning
   - Head -2 file1
   - -c - specified bytes from begining

- Head -c 10 file1
- -v - To display the file name before outputting the first 10 lines,
- Head -v file1
- **head** can be piped to one or more commands to modify the output:
- ls /etc | head
- For example, to list files in the */etc* directory using the **ls** command and print 10 entries.

10. Tail - Linux tail command **prints the last ten lines of the specified file**.
    - -n - Limit output to the last n lines
      Tail -3 file1
    - -c - Limit output to the last n bytes
      Tail -c 5 file1
    - -v Force output of file names
    - + - using this it will print the data starting from that line number
    - Tail +5 file1
11. Chmod command –

    The **chmod** command is used to change the access mode of a file.
    chmod [reference][operator][mode] file.
    The references are used to distinguish the users to whom the permissions apply i.e. they are list of letters that specifies whom to give permissions.
    The references are represented by one or more of the following letters:

- Reference Class - Description

u-owner - file's owner

g-group-users who are members of the file's group

o-others- users who are neither the file's owner nor members of the file's group

a-all - All three of the above, same as ugo

- The operator is used to specify how the modes of a file should be adjusted.
- The following operators are accepted:

Operator Description

+ Adds the specified modes to the specified classes

- Removes the specified modes from the specified classes

= The modes specified are to be made the exact modes for the specified classes

- There are three basic modes which correspond to the basic permissions:

r Permission to read the file.

w Permission to write (or delete) the file.

x Permission to execute the file, or, in the case of a directory, search it.

BEFORE: -rw-rw-r-- mik mik assgn1_client.c

COMMAND: chmod u=r assgn1_client.c

AFTER: -r--rw-r-- mik mik assgn1_client.c

Chmod u+x file1
Chmod u -x file1

- You can also assign permission using octal bits –
  Read -4
  Write -2
  Execute -1
  Eg Chmod 556 file1

12. Chown - **chown** command is used to change the file Owner or group. Whenever you want to change ownership you can use chown command.

chown owner_name file_name

chown master file1**.**txt

13. **chgrp command** in Linux is used to change the group ownership of a file or directory. All files in Linux belong to an owner and a group.

We can change the group of any specific file. To change the group ownership of a file, execute the
command as follows:

sudo chgrp javatpoint Demo1.txt

14. rmdir

remove empty directories. If any specified directory is not empty, **rmdir** will not remove it, and will proceed to try and remove any other directories you specified.

If you want to remove a directory that is not empty (and also remove everything it contains), you can use the **rm** command with the **-r** (recursive) option. See the **rm** command for more information.

15. Rm

rm removes each specified file. By default, it does not remove directories.

-i – prompts the user

-r – removes directories recursively.

-f – removes file forcefully

-r – remove directory with content in it

16. cp

This command is used to copy the source file to destination file or a group of files to destination directory. If dest file does not exist, it will create the destination file.

cp [*options*] *source dest*

cp file1 f1

cp main.c def.h /home/usr/rapid/

cp * dir1

cp *.c bak

cp src /home/usr/rapid/ - dir to dir

-i – Ask before overwriting the file

-R - recursive copy (including hidden files)

cp -R dev bak

-f - force copy by removing the destination file if needed

Forcefully copy of not have the write permission

-u - update - copy when source is newer than dest

17. mv

mv command is used to move files and directories

it is also used to rename a file

mv [*options*] *source dest*

-i – Ask before overwriting the file

-f - force move by overwriting destination file without prompt

u - update - move when source is newer than dest

Move all C files in current directory to subdirectory *bak* :

mv *.c bak

Move all files in subdirectory *bak* to current directory

mv bak/* .

Rename file *main.c* to *main.bak*:

mv main.c main.bak

Rename directory *bak* to *bak2*:

mv bak bak2

Update - move when *main.c* is newer:

mv -u main.c bak

18. Date command

It shows the date and time in the form used on the Internet.

$date -- System Date & Time

$date + %m -- Current Month (05)

$date + %d -- Current day (25)

$date + %y -- Current year (1986)

$date + %D -- Date in format mm/dd/yy

$date + %T -- Time in format HH:MM:SS

$date + %H -- Hour

$date + %M -- Minute

$date + %S -- Second

$date + "%H%M%S" -- Hour Minute Second

$echo "Today date is: `date` " -- Today date is: 25-05-1986

19. Ps

ps displays information about a selection of the active processes.

PID	TTY	STAT	TIME	CMD
5140	pts/4	Ss	00:00:00	bash
61244 pts/4 R+ 00:00:00 ps				

PID: Every process is assigned a PID (Process Identifier) which is a unique identifier that is associated with a running process in the system.

TTY: Controlling terminal associated with the process.

STAT: Process State Code

TIME: Total time of CPU Usage

CMD: The command that is executed by the process.

20. Cut

Cut command is used to cut or pick up a given number of character or fields from the specified file.

This command is support you to see specified filed.

Like you have large database information but out of them you want to see some selected fields than cut command is useful in UNIX.

Syntax

$cut [Main Options] Field or Character List [Options] File Name

Options

-f This option is used to specify field list separated by TAB

-b This option is used to specify bytes list single character taken as bytes
-c This option is used to specify character list single character taken as character

It is must that you will have to used one main option with cut command.

-d, --delimiter=DELIM use DELIM instead of TAB for field delimiter

-f, --fields=LIST select only these fields; also print any line that contains no delimiter character, unless the -s option is specified

-n (ignored)

--complement the set of selected bytes, characters or fields.

-d This option is used to specify the character as field separator
-s This option is used to skip the lines which does not have field separator

Example 1:

$cat >> abc.txt

1. one two three four five six
2. one two three four five six
3. one two three four five six

(1) $cut –b 1,2 abc.txt (This will show you given byte)

1.
2.
3.

(2) $cut –b 1-4 abc.txt (This will show you range 1 to 4 byte)

1. o
2. o
3. o

(3) $cut –c 1,2 abc.txt (This will show you given character)

1.
2.
3.

(4) $cut –c 1-4 abc.txt (This will show you range 1 to 4 character)

4. o 5.
   o
6. o

Example 2:

$cat >>abc1.txt
1:one:two:three:four:five:six
2:one:two:three:four:five:six
3:one:two:three:four:five:six

$cut –f 1-3 –d":" abc1.txt
1:one:two
2:one:two
3:one:two

Example 3:

(1) $cat >> abc.txt
1. one two three four five six
2. one two three four five six
3. one two three four five six
abcdefghijklmnopqrstuvwxyz

(2) $cut –f 1-4 abc.txt
$cat >> abc.txt
1. one two three
2. one two three
3. one two three
abcdefghijklmnopqrstuvwxyz

(3) $cut –f 1-4 –s abc.txt
1. one two three 2.
one two three
3. one two three

Show usernames (in the first colon-delimited field) from /etc/passwd:
$ cut -d: -f1 /etc/passwd
Show first column of /etc/passwd:
$ cut -c 1 /etc/passwd

21. Paste Command

- This command is used to merges lines of files.
- This command prints lines consisting of sequentially corresponding lines of each given files, separated by tabs, terminated by a new line.

- If no files are given , the standard input is used.
- A filename of -means standard input.

Syntax

$Paste [Option] File Name1 File Name2

Options

-s Serial Paste the lines of one file at a time rather then one line from each file. -d Delimiters tow files

Example 1

$cat >> 1.txt

1. aaa bbb ccc
2. aaa bbb ccc
3. aaa bbb ccc

$cat >> 2.txt

A) AAA BBB CCC

B) AAA BBB CCC

C) AAA BBB CCC

(1) $paste 1.txt 2.txt

1. aaa bbb ccc A) AAA BBB CCC 2. aaa bbb ccc B) AAA BBB CCC
3. aaa bbb ccc C) AAA BBB CCC

(2) $paste -d"|" 1.txt 2.txt

1. aaa bbb ccc |A) AAA BBB CCC 2. aaa bbb ccc |B) AAA BBB CCC
3. aaa bbb ccc |C) AAA BBB CCC

(3) $paste -s 1.txt 2.txt

1. aaa bbb ccc 2. aaa bbb ccc 3. aaa bbb ccc

A) AAA BBB CCC B) AAA BBB CCC C) AAA BBB CCC

(4) $paste -s -d"|" 1.txt 2.txt

1. aaa bbb ccc |2. aaa bbb ccc |3. aaa bbb ccc

A) AAA BBB CCC |B) AAA BBB CCC |C) AAA BBB CCC

22. Diff command:

- Reports the differences between file1 and file2.
- Prints file1 text flagged (<) and file2(>).
- Displays line-by-line differences between pairs of text files.

Syntax

diff *file1 file2*

Options

-b: ignore blank spaces.
-e: produce a script of commands to recreate file2 from file1, using ed editor.
-D: merge file1 and file2 into a single file containing conditional C preprocessors (#ifdef). Defining symbol and then compiling yelds file2. Not defining symbol, yields file1.
-i: --ignore-case Ignore case differences in file contents.

--ignore-file-name-case Ignore case when comparing file names.

--no: -ignore-file-name-case Consider case when comparing file names.

-w: --ignore-all-space Ignore all white space.

-B: --ignore-blank-lines Ignore changes whose lines are all blank.

-a: --text Treat all files as text.

--label: LABEL Use LABEL instead of file name.

-q: --brief Output only whether files differ.

-y: --side-by-side Output in two columns.

--left -column Output only the left column of common lines.

--suppress-common-lines Do not output common lines.

-r: --recursive Recursively compare any subdirectories found.

-N: --new-file Treat absent files as empty.

E.g. :

Diff file1 file2

2,5c2,3

<

<

<

>

>

>

OUTPUT :

2,4c5,6 --- remove lines 2-4, and insert updated lines 5-6

4a2,4 --- starting at 4, add updated lines 2-4 *(i.e. 2,4 means 2, 3 and 4)*

2,4d1 --- remove lines 2-4.

23. Sort command

Sorts the lines in a text file.

Syntax sort [option] [+fields] filename [-o output file]

Options

-b: Ignores spaces at beginning of the line.

-d: Uses dictionary sort order and ignores the punctuation.

-f: Ignores caps

-i: Ignores nonprinting control characters.

-m: Merges two or more input files into one sorted output.

-M: Treats the first three letters in the line as a month (such as may.)

-n: Sorts by the beginning of the number at the beginning of the line.

-r: Sorts in reverse order

-u : If line is duplicated only display once

-o – output to a new file

24. **cmp** - Compare two files, and if they differ, tells the first byte and line number where they differ.

Cmp f1 f2

-i – skip particular number of initial bytes from both files.

Cmp -i 2 f1 f2

-n – it is used to compare first n bytes of the file.

25. Comm :-

The comm command compares two sorted files line by line and writes three columns to standard output.

These columns show lines that are unique to files one, lines that are unique to file two and lines that are shared by both files.

1st column – lines present in file1

2nd column – lines present in file2

3rd column – lines common in both the files.

26. Uniq command:

Report or filter out repeated lines in a file

Syntax

uniq [options] [-f fields] [-s char] [-n] [+m] [input_file [output_file]]

sort the file first

Options

-c: Precede each output line with a count of the number of times the line occurred in the input.

-d: Suppress the writing of lines that are not repeated in the input.

-u: Suppress the writing of lines that are repeated in the input.

-f: fields Ignore the first fields fields on each input line when doing comparisons, where fields is a positive decimal integer. A field is the maximal string matched by the basic regular expression: [[:blank:]]*[^[:blank:]]* If fields specifies more fields than appear on an input line, a null string will be used
for comparison.

-s char: Ignore the first chars characters when doing comparisons, where chars is a positive decimal integer. If specified in conjunction with the -f option, the first chars characters after the first fields fields will be ignored. If chars specifies more characters than remain on an input line, a null string will be used for comparison.

-n: Equivalent to -f fields with fields set to n.

+m: Equivalent to -s chars with chars set to m.

-b, --ignore-leading-blanks : ignore leading blanks

-d, --dictionary-order: consider only blanks and alphanumeric characters

-f, --ignore-case: fold lower case to upper case characters

-g, --general-numeric-sort compare

according to general numerical value

-i, --ignore-nonprinting consider

only printable characters

27. FIND

- used to locate files in the file system
- it recursively examines a directory tree to look for files either by name or by matching one or file attribute.

Syntax

find [options] [paths] [expression]

The options for this command are used to specify how symbolic links should be treated. This is followed by the set of paths to search in. If no paths are specified, then the current directory is used. The given expression is then run on each of the files found in the paths.

The expression consists of a series of options, tests, and actions, each returning a boolean. The expression is evaluated left to right for each file in the path until the result is determined i.e. the result is known to be true or false.

Test expressions are used to evaluate specific properties of the files and return true or false accordingly.

- -atime n: Returns true if the file was accessed n days ago.
- -ctime n: Returns true if the file's status was changed n days ago.
- -mtime n: Returns true if the file's contents were modified n days ago.
- -name pattern: Returns true if the file's name matches the provided shell pattern.
- -iname pattern: Returns true if the file's name matches the provided shell pattern. The matching here is case insensitive.
- -path pattern: Returns true if the file's name with the path matches the shell pattern.
- -regex pattern: Returns true if the file's name with the path matches the regular expression.
- -size n: Returns true if the file size is n blocks.
- -perm – mode: Returns true if all the permission bits for mode are set for the file.
- -type c: Returns true if the file is of type c (e.g. 'b' for block device file, 'd' for directory etc.).
- -username: Returns true if the file is owned by username 'name'.

The action expressions are used to define actions that have side effects and may return true or false. If not actions are specified, the '-print' action is performed for all matching files.

-
- -delete: Delete the matched file, and return true if successful.
- -exec command: Execute the given command for each matching file, and return true if the return value is 0.
- -ok command: Like the 'exec' expression, but confirms with the user first.
- -ls: List the matching file as the per 'ls -dils' format.
- -print: Print the name of the matching file.

Options for selection criteria.

 -name : find with file name

1. find . –name “*.sh” it will print all files conatins extension .sh
2. find / -name Chapter1 searches through the root filesystem ("/") for the file named "Chapter1"
3. find ./ -name abc
4. find . -name Chapter1 -type f

To search in the current directory, and all subdirectories, just use the . character to reference the current directory in your find commands,

-type : find files or directory as per type

find . –type d –print

f -> files d -> directories

l -> symbolic links

Search for a file by the name abc.txt below the current directory, and prompt the user to delete each match.

Note that the “{}” string is substituted by the actual file name while running and that the “\;” string is used to terminate the command to be executed.

$ find ./ -name abc.txt -exec rm -i {} \;

Search for files that were modified in the last 7 days below the current directory

$ find ./ -mtime -7

Search for files that have all permissions set in the current hierarchy

$ find ./ -perm 777

Find ./ -size 5M – find specific size

Find ./ -empty -type f; - find empty files

Find -empty -type f -delete – delete empty file

28. TR COMMAND

The tr command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace. It can be used with UNIX pipes to support more complex translation. **tr stands for translate.**

tr [OPTION] SET1 [SET2]

Options

-c : complements the set of characters in string.i.e., operations apply to characters not in the given set

-d : delete characters in the first set from the output.

-s : replaces repeated characters listed in the set1 with single occurrence

-t : truncates set1

- **How to convert lower case to upper case**
  To convert from lower case to upper case the predefined sets in tr can be used

  $cat greekfile | tr “[a-z]” “[A-Z]”
  OR

  $cat geekfile | tr “[:lower:]” “[:upper:]”
- **How to translate white-space to tabs**
  The following command will translate all the white-space to tabs
  $ echo "Welcome To GeeksforGeeks" | tr [:space:] '\t'
- **How to translate braces into parenthesis**
  You can also translate from and to a file. In this example we will translate braces in a file with parenthesis.

  $ tr '{}' '()' newfile.txt
- **How to use squeeze repetition of characters using -s**
  To squeeze repeat occurrences of characters specified in a set use the -s option. This removes repeated instances of a character.
  OR we can say that,you can convert multiple continuous spaces with a single space
  $ echo "Welcome To GeeksforGeeks" | tr -s [:space:] ' '
- **How to delete specified characters using -d option**
  To delete specific characters use the -d option.This option deletes characters in the first set specified.
  $ echo "Welcome To GeeksforGeeks" | tr -d 'w'
- **To remove all the digits from the string, use**
  $ echo "my ID is 73535" | tr -d [:digit:]
- **How to complement the sets using -c option**
  You can complement the SET1 using -c option. For example, to remove all characters except digits, you can use the following.
  $ echo "my ID is 73535" | tr -cd [:digit:]

29. grep command

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

The grep command is used to read from the specified file on the command line.

Syntex

```
grep [options] pattern [files]
```

option

```
Options Description
-c : This prints only a count of the lines that match a pattern
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching
-l : Displays list of a filenames only.
-n : Display the matched lines and their line numbers.
-v : This prints out all the lines that do not matches the pattern
```

-e exp : Specifies expression with this option. Can use multiple times.
-f file : Takes patterns from file, one per line.
-E : Treats pattern as an extended regular expression (ERE)
-w : Match whole word
-o : Print only the matched parts of a matching line,
with each such part on a separate output line.

-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.

Examples:

```
Cat geekfile.txt

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

learn operating system.

Unix linux which one you choose.

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

- **Case insensitive search :** The -i option enables to search for a string case insensitively in the given file. It matches the words like “UNIX”, “Unix”, “unix”.
  grep -i "UNix" geekfile.txt

- **Displaying the count of number of matches :** We can find the number of lines that matches the given string/pattern
  $grep -c "unix" geekfile.txt

- **Display the file names that matches the pattern :** We can just display the files that contains the given string/pattern
  $grep -l "unix" *
  or
  $grep -l "unix" f1.txt f2.txt f3.xt f4.txt
- **Checking for the whole words in a file :** By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.
  grep -w "unix" geekfile.txt

- **Displaying only the matched pattern :** By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.
  grep -o "unix" geekfile.txt
- **Show line number while displaying the output using grep -n :** To show the line number of file with the line matched.
  grep -n "unix" geekfile.txt
- **Inverting the pattern match :** You can display the lines that are not matched with the specified search string pattern using the -v option.
  grep -v "unix" geekfile.txt
- **Matching the lines that start with a string :** The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.
  grep "^unix" geekfile.txt
- **Matching the lines that end with a string :** The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern
  grep "os$" geekfile.txt
- **Specifies expression with -e option. Can use multiple times :**

  $grep –e "Agarwal" –e "Aggarwal" –e "Agrawal" geekfile.txt
- **-f file option Takes patterns from file, one per line.**
  cat pattern.txt

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

- **Print n specific lines from a file:** -A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.
  grep -A[NumberOfLines(n)] [search] [file]
  $grep -B[NumberOfLines(n)] [search] [file]
  $grep -C[NumberOfLines(n)] [search] [file]

  EG. grep -A1 learn geekfile.txt

- **Search recursively for a pattern in the directory: -R** prints the searched pattern in the given directory recursively in all the files.
  grep -R [Search] [directory]
  grep -iR geeks /home/geeks

```
OUTPUT - ./geeks2.txt:Well Hello Geeks

./geeks1.txt:I am a big time geek
```

Regular Expression

In UNIX, regular expressions are defined using the following Meta symbols and extensions to the regular expression notation.

Special characters are used to define the matching rules and positions.

#1) Anchor Characters: '^' and '$' at the beginning and end of the pattern are used to anchor the pattern to the start of the line, and to the end of the line respectively.
Example: "^Name" matches all lines that start with the string "Name". The strings "\<" and "\>" are used to anchor the pattern to the start and end of a word respectively.
#2) Wildcard Character: '.' Is used to match any character.
Example:"^.$" will match all lines with any single character.
#3) Escaped Characters: Any of the special characters can be matched as a regular character by escaping them with a '\'.
Example: "\$*" will match the lines that contain the string "$*"
#4) Character Range: A set of characters enclosed in a '[' and ']' pair specify a range of characters to be matched.
Example: "[aeiou]" will match all lines that contain a vowel. A hyphen can be used while specifying a range to shorten a set of consecutive characters.
E.g. "[0-9]" will match all lines that contain a digit. A carat can be used at the beginning of the range to specify a negative range.
E.g. "[^xyz]" will match all lines that do not contain x, y or z.
#5) Repetition Modifier: A '*' after a character or group of characters is used to allow matching zero or more instances of the preceding pattern.
The grep command supports a number of options for additional controls on the matching:

- -i: performs a case-insensitive search.
- -n: displays the lines containing the pattern along with the line numbers.
- -v: displays the lines not containing the specified pattern.
- -c: displays the count of the matching patterns.

Examples:

- Match all lines that start with 'hello'. **E.g:** "hello there"

```
$ grep "^hello" file1
```

- Match all lines that end with 'done'. **E.g:** "well done"

```
$ grep "done$" file1
```

- Match all lines that contain any of the letters 'a', 'b', 'c', 'd' or 'e'.

```
$ grep "[a-e]" file1
```

- Match all lines that do not contain a vowel

```
$ grep "[^aeiou]" file1
```

- Match all lines that start with a digit following zero or more spaces. **E.g:** " 1." or "2."

```
$ grep " *[0-9]" file1
```

- Match all lines that contain the word hello in upper-case or lower-case

$ grep -i “hello”

30. Egrep command :-

egrep is a pattern searching command which belongs to the family of grep functions. It works the same way as ***grep -E*** does. It treats the pattern as an extended regular expression and prints out the lines that match the pattern. If there are several files with the matching pattern, it also displays the file names for each line

```
egrep [ options ] 'PATTERN' files
```

The *egrep* command used mainly due to the fact that it is faster than the grep command. The egrep command treats the meta-characters as they are and do not require to be escaped as is the case with grep. This allows reducing the overhead of replacing these characters while pattern matching making egrep faster than *grep* or *fgrep*. **Options:** Most of the options for this command are same as ***grep***.

31. sed command:

SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace.

By using SED you can edit files even without opening them, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
- SED command in unix supports regular expression which allows it perform complex pattern matching.

- **`sed OPTIONS... [SCRIPT] [INPUTFILE...]`**

a. **Replacing or substituting string :** Sed command is mostly used to replace the text in a file.
The below simple sed command replaces the word “unix” with “linux” in the file.

```
$sed 's/unix/linux/' geekfile.txt
```

Here the “s” specifies the substitution operation. The “/” are delimiters

b. **Replacing the nth occurrence of a pattern in a line :** Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below

command replaces the second occurrence of the word “unix” with “linux” in a line.

```
$sed 's/unix/linux/2' geekfile.txt
```

c. **Replacing all the occurrence of the pattern in a line :** The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.
```
$sed 's/unix/linux/g' geekfile.txt
```
d. **Replacing from nth occurrence to all occurrences in a line :** Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth… “unix” word with “linux” word in a line.
```
$sed 's/unix/linux/3g' geekfile.txt
```

e. **Parenthesize first character of each word :** This sed example prints the first character of every word in parenthesis.
```
$ echo "Welcome To The Geek Stuff" | sed 's/\(\b[A-Z]\)/\(\1\)/g
```
f. **Replacing string on a specific line number :** You can restrict the sed command to replace the string on a specific line number. An example is
```
$sed '3 s/unix/linux/' geekfile.txt
```
g. **Duplicating the replaced line with /p flag :** The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.
```
$sed 's/unix/linux/p' geekfile.txt
```
h. **Printing only the replaced lines :** Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.
```
$sed -n 's/unix/linux/p' geekfile.txt
```
i. **Replacing string on a range of lines :** You can specify a range of line numbers to the sed command for replacing a string.
```
$sed '1,3 s/unix/linux/' geekfile.txt
```

```
$sed '2,$ s/unix/linux/' geekfile.txt
```

j. **Deleting lines from a particular file :** SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file Examples:

- **To Delete a particular line say n in this example**

  Synax:

  ```
  $ sed 'nd' filename.txt
  ```

Example:

```
$ sed '5d' filename.txt
```

- **To Delete a last line**

```
Syntax:
$ sed '$d' filename.txt
```

- **To Delete line from range x to y**

```
Syntax:
$ sed 'x,yd' filename.txt
Example:
$ sed '3,6d' filename.txt
```

- **To Delete from nth to last line**

```
Syntax:
$ sed 'nth,$d' filename.txt
Example:
$ sed '12,$d' filename.txt
```

- **To Delete pattern matching line**

```
Syntax:
$ sed '/pattern/d' filename.txt
Example:
$ sed '/abc/d' filename.txt
```

1) sed '1d' EMP.txt // delete first line

102 SmithManager 40000
103 ChalseProgrammer 30000
104 JamsSearching 25000
105 Prats Coder 40000
106 PiterAnalysis 25000
107 shah Tester 20000
108 Bav Programmar 40000

2) sed '/Anjal/d' EMP.txt

101 John Analysis 25000
102 SmithManager 40000
104 JamsSearching 25000
105 Prats Coder 40000
106 PiterAnalysis 25000
107 shah Tester 20000
108 Bav Programmar 40000

3) sed '/^$/d' EMP.txt //delete blank lines
4) sed -n '2~2p' EMP.txt //display even line
sed -n '/John/p' EMP.txt //display John record
101 John Analysis 25000
102 SmithManager 40000
103 ChalseProgrammer 30000
104 JamsSearching 25000
105 Prats Coder 40000
106 PiterAnalysis 25000
107 shah Tester 20000
108 Bav Programmar 40000

5) sed -n '2~2p' EMP.txt //display even line

102 SmithManager 40000
104 JamsSearching 25000
106 PiterAnalysis 25000
108 Bav Programmar 40000

5) sed -n '/John/p' EMP.txt //display John record

101 John Analysis 25000

Options

-n: --quiet, --silent suppress automatic printing of pattern space

-e: script, --expression=script add the script to the commands to be executed

-f: script-file, --file=script-file add the contents of script-file to the commands to be executed

-i[suffix], --in-place[=suffix] : edit files in place (makes backup if extension supplied)

-l N: --line-length=N specify the desired line-wrap length for the `l' command

-r: --regexp-extended use extended regular expressions in the script.

-s: --separate consider files as separate rather than as a single continuous long stream.

-u: --unbuffered load minimal amounts of data from the input files and flush the output buffers more often

32. Awk commnd

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling and allows the user to use variables, numeric functions, string functions, and logical operators.

Awk is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line. Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform the associated actions.

Awk is abbreviated from the names of the developers – Aho, Weinberger, and Kernighan.

1. AWK Operations:
(a) Scans a file line by line
(b) Splits each input line into fields
(c) Compares input line/fields to pattern
(d) Performs action(s) on matched lines
2. Useful For:
(a) Transform data files
(b) Produce formatted reports
3. Programming Constructs:
(a) Format output lines
(b) Arithmetic and string operations
(c) Conditionals and loops

Syntax:

```
awk options 'selection _criteria {action }' input-file > output-file

-f program-file : Reads the AWK program source from the file
                  program-file, instead of from the
                  first command line argument.
-F fs            : Use fs for the input field separator
```

Consider the following text file as the input file for all cases below:

```
$cat > employee.txt
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

- **Default behavior of Awk:** By default Awk prints every line of data from the specified file.

```
awk '{print}' employee.txt
```

- **Print the lines which match the given pattern.**

```
$ awk '/manager/ {print}' employee.txt
```

- **Splitting a Line Into Fields :** For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.

```
$ awk '{print $1,$4}' employee.txt
```

- **Built-In Variables In Awk**

Awk's built-in variables include the field variables—$1, $2, $3, and so on ($0 is the entire line) — that break a line of text into individual words or pieces called fields.

- **NR:** NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
- **NF:** NF command keeps a count of the number of fields within the current input record.
- **FS:** FS command contains the field separator character which is used to divide fields on the input line. The default is "white space", meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.
- **RS:** RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
- **OFS:** OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.

- **ORS:** ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

- **Use of NR built-in variables (Display Line Number)**
  ```
  $ awk '{print NR,$0}' employee.txt
  ```
- **Use of NF built-in variables (Display Last Field)**
  ```
  $ awk '{print $1,$NF}' employee.txt
  ```
- **Another use of NR built-in variables (Display Line From 3 to 6)**
  ```
  $ awk 'NR==3, NR==6 {print NR,$0}' employee.txt
  ```

```
cat > geeksforgeeks.txt


A    B    C

Tarun    A12    1

Man    B6    2

Praveen    M42    3
```

- **To print the first item along with the row number(NR) separated with ” – “ from each line in geeksforgeeks.txt:**

```
$ awk '{print NR "- " $1 }' geeksforgeeks.txt
```

- **To return the second column/item from geeksforgeeks.txt:**

The question should be:- To return the second column/item from geeksforgeeks.txt:

```
$ awk '{print $2}' geeksforgeeks.txt
```

- **To print any non empty line if present**

  awk ‘NF == 0 {print NR}’ geeksforgeeks.txt

  OR

  awk ‘NF <= 0 {print NR}’ geeksforgeeks.txt

- **To find the length of the longest line present in the file:**

```
awk '{ if (length($0) > max) max = length($0) } END { print max }'
geeksforgeeks.txt
```

- **To count the lines in a file:**
  ```
  $ awk 'END { print NR }' geeksforgeeks.txt
  ```
- **Printing lines with more than 10 characters:**
  ```
  $ awk 'length($0) > 10' geeksforgeeks.txt
  ```
- **To find/check for any string in any specific column:**
  ```
  $ awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt
  ```

- **To print the squares of first numbers from 1 to n say 6:**

```
$ awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i*i; }'
```