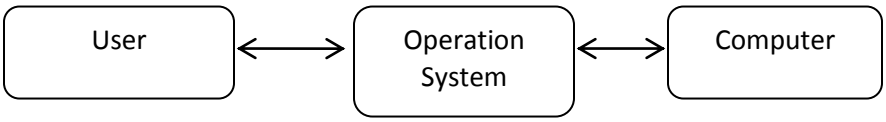
UNIX Introduction

What is UNIX?

UNIX is an operating system which is a set of programs that act as a link between the computer and the user.



Operating System is classified in two types:

1) **CUI:** Character user Interface e.g. DOS, UNIX etc. (Not User friendly)
2) **GUI:** Graphical User Interface e.g. Windows etc. (User friendly)

- ☞ UNIX was originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Douglas McElroy, and Joe Hosanna.
- ☞ 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 called multitasking.

Different Types of Operating Systems:

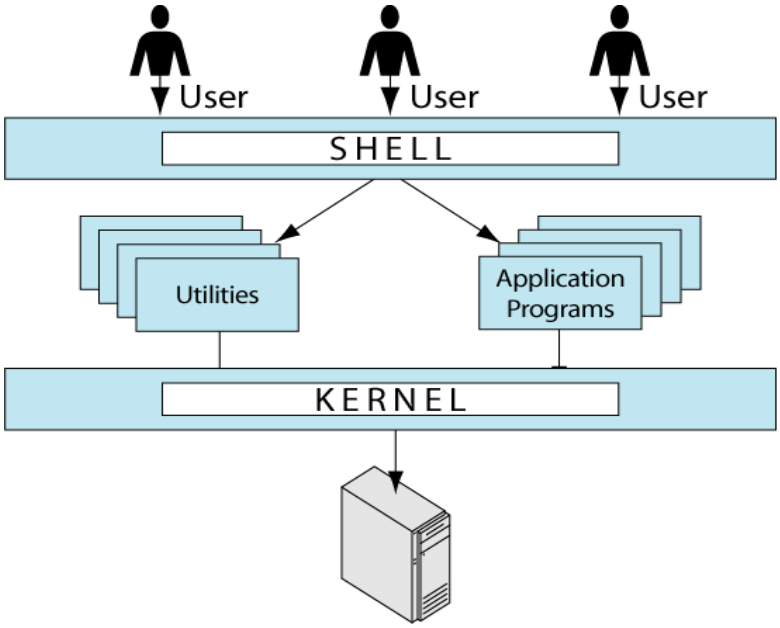
- ☞ ***Single-user, single-process operating systems:*** Allow only one user at a time to use the computer system. The user can execute/run only one process at a time.
  Examples: DOS, Windows 3.1

- ☞ ***Single-user, multi-process operating systems:*** Allow a single user to use the computer system; however, the user can run multiple processes at the same time.
  Example: OS/2

- ☞ ***Multi-user, multi-process operating systems*:** Allow multiple users to use the computer system simultaneously. Each user can run multiple processes at the same time.
  Examples: UNIX, Windows XP

UNIX Architecture:

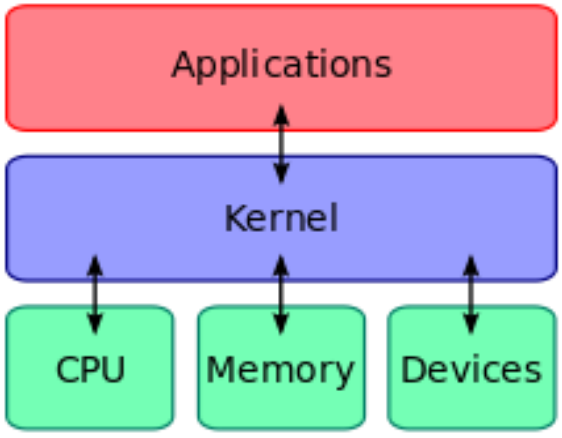
The UNIX operating system is made up of three parts; the kernel, the shell and the programs.



Kernel:

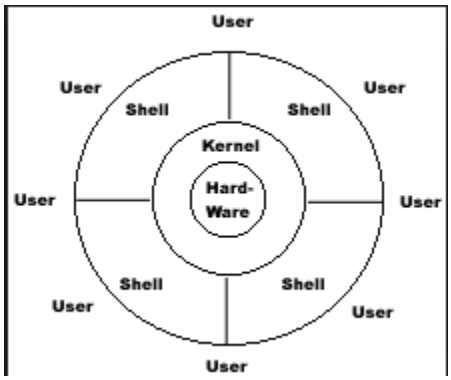
The Kernel is a core part of UNIX as well as the heart of the operating system. It interacts with hardware and most of the tasks like memory management, access to computers, scheduling and file management.

- To manage computer memory
- To control access to the computer
- To perform input and output services (which allows computers to interact with terminals storage devices and printers)
- To allocate the resources of the computer (such as CPU and input/output devices) among users.



Shell:

1) The shell acts as an interface between the user and the kernel.
2) When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI).
3) It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt (% on our systems).



History- The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list or type history for a list of previous commands.

Features of UNIX:

1) **Multi-User:** More than one user can share same system resources at the same time known as Multi-User. System resources means Disk, CPU, Applications, Printers etc.
2) **Multi-Tasking:** Execution of more than one task or application simultaneaoulsy known as Multi-Tasking. Main motive of Multi-Tasking is maximum utlization of CPU Resouces.
3) **Open System:** Unix is an Open Source Code.

Redhat + New Features = **LINUX**
Sun Micro Systems + New Features = **Sun Solaris**
IBM + New Features = **IBM – AIX**
HP + New Features = **HP – UX**
Santa Cruz + New Features = **Sco – UNIX**
Silicon Graphics + New Features = **IRIX**

→ **Flavors of UNIX:** Any O/S developed based on UNIX Open Source is known as flavors of UNIX.

Note: UNIX does not have O/S s/w in market. We need to install any one of the above UNIX flavor.

Features of LINUX:

- Free Software
- GUI (User friendly)
- Minimum H/W Specifications
- Inbuilt S/W (C, C++, Perl, PHP, Python, MySQL, Apache Web Server etc)

Flavors of LINUX:

- Fedora LINUX
- Ubuntu LINUX
- Open Suse LINUX
- Cent OS LINUX
- Orace Enterprise Linux (OEL)

4) **Programming Facility:** With Shell Scripting you can use group of UNIX commands & Shell keywords.
Main Concept of Shell Scripting is:

a) To handle text files
b) To create new UNIX commands
c) To customize work environment

5) **Security:** UNIX has given two level of security, i.e. ***System Level & File Level***.

a) **System Level Security:** Controlled by System Administrator
b) **File Leve Security:** Controlled by Owner of the file

Files & Processes:

1) Everything in UNIX is either a file or a process.
2) A process is an executing program identified by a Unique PID (Process Identifier).
3) A file is a collection of data. They are created by users using text editors, running compilers etc.

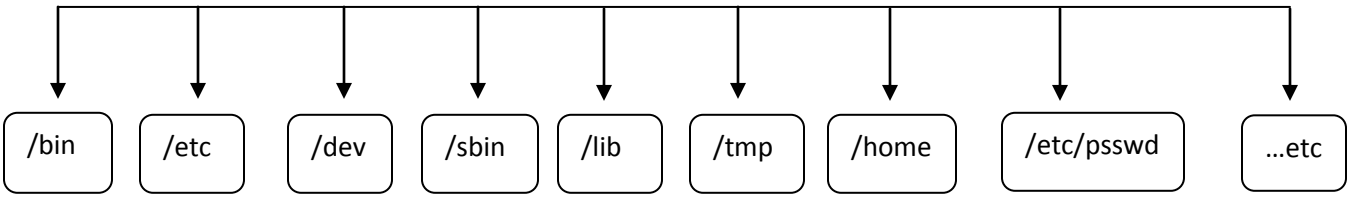
There are 3 main types of files:

1) **Regular files:** It is a collection of data or group of records
2) **Directory files:** It is a group of files & sub-directories
3) **Device files:**
   a) **Bocked Special files or Unreadable files:**Floppy, CD-ROMS, Printers etc
   b) **Charctel Special File:**
      Stdin (I/P device→Keyboard)
      Stdout & Stderr (O/P device→Monitor,i.e.readable format)

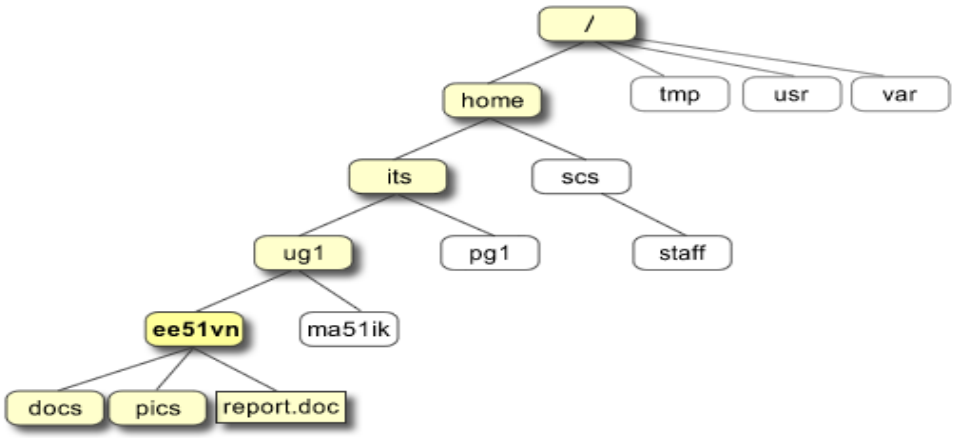
The Directory Structure:

All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called **root** (written as a slash /). It is Administrators home directory.



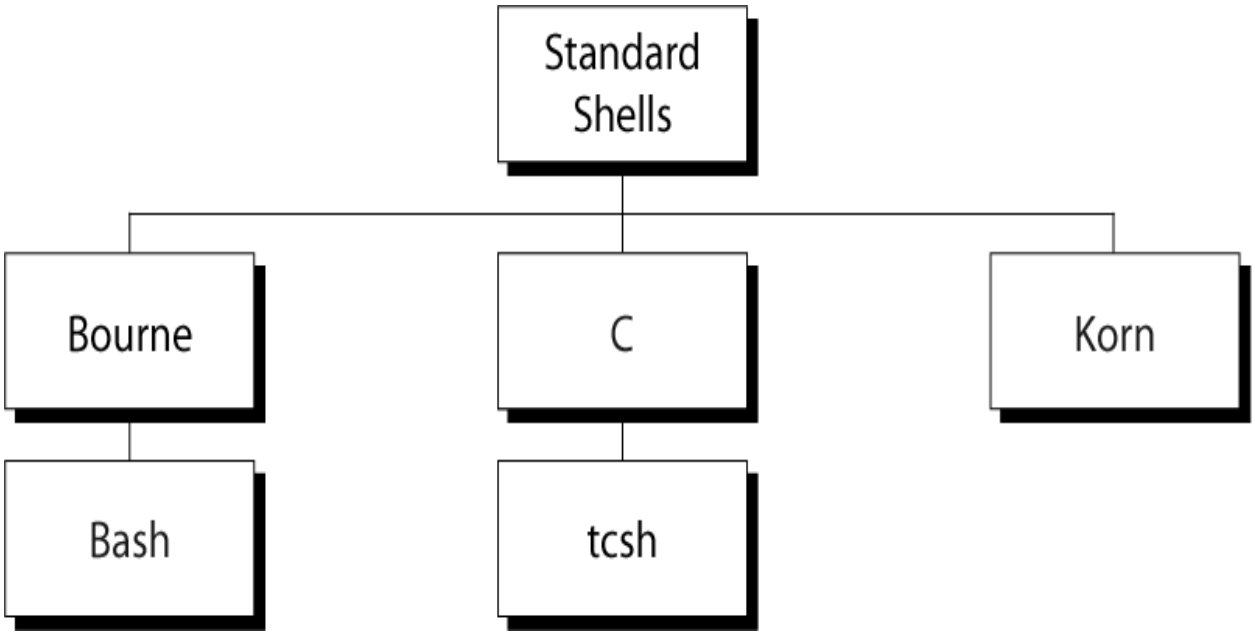
1) **/bin:** It contains all user executable commands
2) **/etc:** It contains all system configuration files
3) **/dev:** It contains all device file information
4) **/sbin:** It contains all administrators executable commands
5) **/lib:** It contains all library file information
6) **/tmp:** It contains all library file information
7) **/home** OR**/usr:** It contains all temporary file information (These 2 are Users Home Directory)
8) **/ect/passwd:** This file contains each & every user information with 7 fields
   [*Username : Passwd : UID : GID : Comment : home : Shell*]



In the diagram above, we see that the home directory of the undergraduate student **"ee51vn"** contains two sub-directories (**docs** and **pics**) and a file called **report.doc**.
The full path to the file **report.doc** is **"/home/its/ug1/ee51vn/report.doc"**

Note: Mountpoint is the UNIX terminology for Partitioning.

Some Standard UNIX Shells:



Basic Command

$ logname: - It displays user login name.
$ pwd: - It displays present working directory path.
$ date:- It displays systems date & time.
$ clear:- Clears the screen.
$ cal:- Displays current month calendar
$ cal 2016 :- Displays 2016 year calendar
$ uname:- It displays operating system name
$ uname –r:- It displays kernel version
$ hostname:- It displays server name
$ hostname –i:- It displays server ip address.
$ who:- It displays list of users connected to server.
$ finger username:- It displays given user information
$ who am I :- It displays current user information
$ whoami: It display current user
$ tty:- it displays current terminal name.
$ uptime:- it displays how long server is up & running, no of users connected, average load on the server.
$ su:- it is used to switch from one user account to another user account
E.g. $su Saif
$ which/where is :- It display location of the given command
E.g. $ which date
E.g. $ where is date
$ man command_name:- It displays help pages of given command.
$ exit:- It is used to logout from current user account.

1) How to create a user?
$ useradd user_name

2) How to delete a user?
$ userdel user_name
Note: You may want to delete the home directory for the deleted user account:
$ rm -r /home/user_name

3) To list all users you can use?
$ cut -d : -f1 /etc/passwd

4) To modify the username of a user?
$ usermod -l new_username old_username

5) To change the password for a user?
$ passwd user_name

Working with files

$ cat :- It is used for creating new files, (or) to open existing file (or) to append/change data to existing file.

Creating a new file:

$ cat > File_Name
E.g. $ cat > Saif
(Type same data)
(Ctrl+d) [to save data & close file, called as eof character used by UNIX systems]

Open an existing a file:

$ cat < File_Name
E.g. $ cat < Saif
$ cat File_Name

Append/Change data to a file:-

$ cat >> File_Name

[Ctrl+d]

Note: Creating & appending (>) or (>>) is mandatory, whereas for opening a file (<) not mandatory.

$ cat -n file_name → Number all Output Lines
$ cat -b file_name → Number Non-Blank Output Lines

How to open multiple files:-

$ cat file1 file2 ……file_n

E.g.- $ cat emp_1 emp_2 emp_3…emp_n

Note:
1) It displays emp_1 table contents first, followed by emp_2, emp_3…emp_n and so on.
2) Can't create multiple files using cat command.

$ touch:- It is used to create empty files. i.e. Zero bytes files
$ touch file_name

Create multiple files:-
$ touch file_1 file_2….file_n

Deleting files:-
$ rm: - The rm command deletes one or more files. It normally operates silently and should be used with caution. A file once deleted can't be recovered.
E.g. $ rm file_name
$ rm * → removes all files in the current directory.

Note: Whether or not you are able to remove a file depends, not on the file's permissions but on the permissions you have for the directory.

$ rm –i :- To delete a file with permission/confirmation.
E.g. $ rm -i file_name

Recursive Deletion (-r or -R): With the –r or –R option, rm performs a treewalk – a thorough recursive search for all subdirectories and files within these subdirectories. At each stage it deletes everything it finds. rm won't normally remove directories, but when used with this option, but when used with this option it will.
$ rm -r * → Behaves partially like rmdir
You'll delete all files in the current directory and all its subdirectories. If you don't have a backup, then these files will be lost forever.

Forcing Removal (-f): rm prompts for removal if a file is write-protected. The -f option overrides this minor protection and forces removal. When you combine it with -r option, it could be the most risky thing to do.
$ rm -rf * → Deletes everything in the current directory & below.

Caution: Make sure you are doing the right thing before you use **rm *.** Be doubly sure before you use **rm -rf *.** The first command removes only ordinary files in the current directory. The second one removes everything – files & directories alike. If the root user (the super user) invokes **rm -rf *** in the / directory, the entire UNIX system will be wiped out from the hard disk.

Deleting multiple files:-
$ rm file_1 file_2 ……file_n

Note: Typing **cd** with no argument always returns you to your home directory. This is very useful if you are lost in the file system. And, **cd /** takes directly to Root Directory.

Working with Directories:-

$ mkdir: Directories are created with mkdir command. The command is followed by the names of the directories to be created.

$mkdir dir_name
$ mkdir Unix1
$ mkdir Unix1 Unix1/d1 Unix1/d2 → Creates two subdirectories under Unix1 directory
$ mkdir d1 d2 d3 → Creates multiple directories

$ rmdir dir_name
$ rmdir d1 d2 d3 → Remove multiple directories
Note:
1) You can't delete a directory with **rmdir** unless it's empty.
2) You can't remove a subdirectory unless you are placed in a directory which is hierarchically above the one you have chosen to remove.

$ rm – r dir_name:-It deletes recursively entire directory structure
$ rm – ri dir_name:- It deletes recursively entire directory structure with confirmation
$ rm – rf dir_name:- It deletes recursively entire directory structure with forcibly

Copying files:-

$ cp :- To copy a file.

$ cp Source_File Target_File

Note: Source file must be existing file and target file may be a new file/existing file.

Moving/Renaming files:

$mv: To Rename/Move a file or directory.

$ mv old_file new_file (old file is renamed/moved to new file)

$ mv old_directory new_directory (old directory is renamed/moved to new directory)

Note:-

$ cp	→ Copy & Paste
$ mv	→ Cut & Paste

Creating Hidden files: Any file_name/directory_name starts with (.) is hidden file/directory.

$ cat > .file_name

Ctrl+d [To save & close file]

$ mkdir .directory_name

$ mv emp .emp	→Hide existing file
$ mv .emp emp	→ Unhide existing file
$ mv abc .abc	→ Hide directory
$ mv .abc abc	→ Unhide directory

Viewing list of files:-

$ ls	→It list current directory all files & sub – directories in ascending order based on ASCII Values.
$ ls -a	→It list all files along with hidden files.
$ ls -r	→It list all files in reverse order (Descending)
$ ls -R	→It list all files recursively.
$ ls -t	→It list all files based on date & time of creation.
$ ls -l	→It list all files in long list format. i.e.9 fields.

Word Count: UNIX features a universal word-counting program that also counts lines & characters. It takes one or more filenames as an arguments and displays a four-columnar output.

$ wc File_Name → (2 lines 10 words 45 character File_name)

1) A **Line** is any group of characters not containing a newline.
2) A **Word** is a group of characters not containing a space, tab or newline.
3) A **Character** is the smallest unit of information, and includes a space, tab and a newline.

$ wc -l File_Name →It counts number of lines.
$ wc -w File_Name →It counts number of words.
$ wc -c File_Name →It counts number of characters.
$ wc -lw File_Name →It counts number of lines and words.
$ wc -wc File_Name →It counts number of words and characters.
$ wc -lc File_Name →It counts number of lines and characters.

Displaying Data in Octal: Many files (especially executable) contain nonprinting characters, and most UNIX commands don't display them properly.

$ od -b file_name
$ od -c file_name
$ od -bc file_name

Note: Each line is now replaced with two. The octal representations are shown in the first line. The printable characters and escape sequences are shown in the second line.

1) The tab character, [Ctrl-i], is shown as \t and the octal value is 011.
2) The bell character, [Ctrl-g], is shown as 007. Some systems show it as \a.
3) The formfeed character,[Ctrl-l], is shown as \f and 014.
4) The LF (linefeed or newline) character, [Ctrl-j], is shown as \n and 012.

Note: od makes the newline character visible too.

Compare:

1) You may often need to know whether two files are identical so one of them can be deleted.
2) The two files are compared byte by byte and the location of the first mismatch character is echoed to the screen.
3) If two files are identical, **cmp** displays no message, but simply returns the prompt.

$ cmp file_1 file_2
$ cmp file_2 file_3

Difference:- diff is the third command that can be used to display file differences. It also tells you which lines in one file have to be changed to make the two files identical.

$ diff file_1 file_2
$ diff file_2 file_3

Filter Commands:

a) In a file data entered by using delimiter is known as flat file
b) The default delimiter is Tab key
c) Delimiter means field separator
d) Enter key means record separator

1) cut: It is used to extract specific fields and characters from a given file.

a) $ cut -f 2,4 Student	→It displays 2nd, 4th fields from Student file
b) $ cut -f 2-4 Student	→It displays 2nd field to 4th field from Student file
c) $ cut -d ',' -f 2,4EMP	→It displays 2nd, 4th fields from EMP file
d) $ cut -d ',' -f 2-4 EMP	→It displays 2nd to 4th fields from EMP file
e) $ cut -c 1-10 Student	→It displays every line 1st to 10th character from Student file
f) $ cut -c 5-10, 15-24 Student	→It displays every line 5th to 10th& 15th to 20th character from Student file.cat

2) paste: It is used to join two or more files horizontally by using delimiter (default is Tab).

$cat > States
MH
HYD
GOA
J&K
Ctrl+d

$ cat > Cities
Pune
Mumbai
Nasik
Wakad
Ctrl+d

a) $ paste States Cities
b) $ paste -d '|' States Cities > temp
c) $ paste -s file_name
d) $ paste -s -d '||\n' file_name

Note: The -s option joins lines in the same way vi's J does. Using this option on this file will Join all of these lines in a single line. Hence, we can use -d option with single or multiple delimiter to join lines.

If we specify the delimiter sting as ||\n with -d, the delimiter are used in circular manner. The first and second line would be joined with a | as a delimiter, and the same would be true for the second and the third line. The third and fourth line would be separated by a \n newline. After the list is exhausted it is reused.

3) tr: The tr (translate) filter manipulates individual characters in a line. More specifically, it translates characters using one or two compact expressions.

$ tr options exp_1 exp_2 standard_input

Note: tr takes input from standard_input; it doesn't take a filename as argument. By default, it translates each character in exp_1 to its mapped counterpart in exp_2. The first character in the first expression is replaced by the first character in the second expression, and similarly for the other characters.

Length of the two expressions should be equal. If they are not, the longer expression will have unmapped characters (not in Linux).

$ cat > file_tr
I am learning unix
Ctrl+d

a) $ tr 'aeiou' 'AEIOU' < file_tr → It replaces Lower Case characters with Upper Case
b) $ tr '[a-z]' '[A-Z]' < file_tr → It replaces all Lower Case a-z characters with Upper Case
c) $ tr ',' '\t' < Emp → It replaces Comma Delimiter with Tab Delimiter in Emp file
d) $ tr -d 'a' < file_tr → It deletes 'a' character from Sample file
e) $ tr -d 'aeiou' < file_tr → It deletes Lower Case 'aeiou' characters from Sample file
f) $ tr -s ' ' < file_tr → It compress multiple Consecutive characters (Squeezes)
g) $ tr -cd ',' < student → It deletes all characters except the delimiter

Banner: Banner command prints a message in large letters which look like banner.
$ banner Hello
Note: Banner can accommodate only 10 characters in one line.

4) sort: Sorting is the ordering of data in ascending or descending order. It sorts the given file contents in Ascending Order based on ASCII Values, i.e. Whitespace first, then numerals, Upper case & finally Lower case letters.

ASCII Values:
0-9 → 48-57
A-Z → 65-90
a-z→ 97-122

a) $ sort file_sort
b) $ sort -r file_sort →It sorts in Descending order
c) $ sort -u file_sort →It displays only Unique lines, i.e. it eliminates duplicate lines
d) $ sort file_nos_sort →It sorts as per ASCII Values
e) $ sort -n file_nos_sort →It sorts in numerically
f) $ sort -t ',' -k 2 emp → Sorting on PK (-k option) on the second column. (-t) delimiter with sort option.

Sorting on Secondary Key: You can sort on more than one key, i.e. you can provide a secondary key to **sort.** If the primary key is fourth field and the secondary field is the second field, then you need to specify for every -k option, where the sort fields end.

$ sort -t ',' -k 4 -k 2 emp → It sorts 4th filed, then 2nd field
$ sort -t ',' -k 4,4 -k 2,2r emp → It sorts 4th filed, then 2nd field in Descending order.
Note: -k 4,4 indicates that sorting starts on the fourth field and ends on the same field.

Sorting on Columns: You can also specify a character position within a field to be the beginning of sort. If you are to sort the file according to the year of birth, then you need to sort on the seventh & eighth position within the fifth field.

$ sort -t ',' -k 5.7,5.8 emp → It sorts on year of the fifth field from D.O.B.
$ sort -t ',' -k 5.7,5.8r emp → It sorts reverse on year of the fifth field from D.O.B.

Note: The -k option also uses the form -k m.n, where n is the character position in the mth field. So, 5.7,5.8 mean that sorting starts on column 7 of the fifth field and ends on column 8.

$ sort -u file_name → The -u option, lets you show only unique lines.
$ sort -c file_name → It checks whether the file is sorted or not.

5) uniq: It displays unique lines in the given file but the file contents must be in Sorted Order.

a) $ uniq file_uniq
b) $ uniq -u file_uniq →It displays Non-Duplicate lines
c) $ uniq –d Unique_File →It displays only duplicated lines
d) $ uniq –c Unique_File →It counts how many times the lines are repeated in the file.

6) Date command:

$ date
$ date '+%m-%h'
$ date '+Date: %d-%m-%y%nTime:%H-%M-%S'

%m →Month of the year (1-12)
%d →Day of month (1-31)
%y →Year (last two digits)
%D →Date as mm/dd/yy
%H →Hour (OO to 23)
%M →Minutes (OO to 59)
% T →Time as HH:MM:SS

Process: A process is simply an instance of a running program. A program is said to be born, when the program starts execution and remains alive as long as the program is active. After execution is complete, the process is said to die.

$ echo $$ → To know the PID of your current shell.
$ ps

PID	TTY	TIME	CMD
2855	pts/1	00:00:00	bash
8080	pts/1	00:00:00	sort
9372	pts/1	00:00:00	ps

$ ps -a: It is used to find out which processes are running for the other users who have logged in.
$ ps -u user_name: To see what a particular user is doing.
$ ps -t: It is the command to find out the processes that have been launched from a particular terminal.
$ ps -f: It gives additional information about process such as Parent Process ID, start time, name of process running.
$ ps -e: It is the command to know about every process running at that instance.

Kill:
$ kill PID: When invoked kill command sends a termination signal to the process being killed. We can employ sure kill signal to forcibly terminate a process. Signal number for sure kill is 9.

$ Kill -s kill 2398 → Recommended way of using KILL
$ kill -9 2398 → Same as above but not recommended

Grep [Globally search a regular expression and print it]: It is used to search a String and Regular Expression in a given file or files.

Syntax:
grep string/pattern filename(s)

a) $ grep 'smidsy' file_grep →It prints the line containing smidsy
b) $ grep 'Smidsy' file_grep →It prints the line containing Smidsy
c) $ grep 'smidsy' file1 file2 file3 →It searches smidsy string in file1, file2 & file3 files and print those lines
e) $grep smidsy * → It searches smidsy string in current directory all files

Note: If string contains more than one word then it should be enclosed in double quotes

Grep command option:

a) $ grep -i 'smidsy' file_grep →It ignores case sensitive
b) $ grep -c 'smidsy' file_grep →It counts number of lines having smidsy
c) $ grep -n 'smidsy' file_grep →It prints lines with line numbers containing smidsy
d) $ grep -v 'smidsy' file_grep →It displays lines that do not contain smidsy
e) $ grep -l 'smidsy' file_grep →It prints only filenames containing smidsy
f) $ grep -o 'smidsy' file_grep →It prints pattern containing only smidsy
g) $ grep - - color 'smidsy' file_grep →It highlights the pattern with color

Note: If you want to find out Matching Multiple Patterns (-e) option.
h) $ grep -e 'Exp' -e 'nos' file_grep → It matches multiple strings and prints the lines

- E: Treats Pattern as an extended regular expression.
- F: Matches multiple fixed strings (in **fgrep** style).

Regular Expression: Any string containing wild character is known as Regular Expression or Pattern.
These patterns are classified in 3 types:
1) Character pattern
2) Word pattern
3) Line pattern

1) Character pattern:

1) * → Zero or more occurrence of previous character
2) . → A Single character
3) .* → Nothing or any number of characters
4) [pqr] → A single character p, q or r
5) [^pqr] → A single character which is not p, q or r
6) [^a-Za-z] → A non-alphabetic character
7) [1-3] → A digit between 1 and 3

a) $ grep 'smi*' File_grep → It prints line having smi pattern
b) $ grep 'i[ts]' File_grep → It prints lines starting with i and any pattern matching in []
c) $ grep 'N..e' File_grep → It prints (.) any single character

2) Word Pattern:

a) \< \> OR -w → Word Boundary
b) \< → Start of the Word
c) \> → End of the Word

1) grep '\<smidsy\>' file_grep →It Prints line containing the word smidsy
2) grep -w "smidsy" file_grep →It Prints line containing the word smidsy
3) grep '\<smidsy' file_grep → It Prints line starting with the word smidsy
4) grep 'smidsy\>' file_grep → It Prints line ending with the word smidsy
5) grep '\<[0-9]' file_grep →It Prints line starting with number from [0-9] and any length
6) grep '\<[0-9][0-9]\>' file_grep → It Prints line starting with any number from [0-9] and length upto 2
7) grep -n '\<[0-9][0-9]\>' file_grep →It Prints number of lines and the matching pattern
8) grep -c '\<[0-9][0-9]\>' file_grep →It Prints the count of matching pattern
9) grep -E '\<[0-9]{1,2}\>' file_grep → It Prints upto 2 digits from a line
10) grep -Eo '^\<[0-9]{1,2}\>' file_grep → It Prints only 2 digits starting from a line

3) Line Pattern:

a) ^ → Start of the Line
b) $ → End of the Line
c) ^$ → Lines containing nothing (Empty Lines)

1) grep "^P" File_grep → It Prints line staring with P
2) grep "s$" File_grep → It Prints line ending with s
3) grep "^Th" File_grep → It Prints line starting with the 'Th' characters
4) grep "^[TH]" File_grep → It Prints line starting with T or H characters
5)grep "^[^TH]" File_grep → It Prints line which do not start with T or H characters
6) grep "^\<The\>" File_grep → It Prints line with Starting with 'The' word
7) grep "^[0-9]" File_grep → It Prints line staring with number from 0-9
8) grep "[0-9]$" File_grep → It Prints line ending with number from 0-9
9) grep "^Smidsy$" File_grep → It Prints line having only word as 'Smidsy'
10) grep "\<Smidsy\>" File_grep → It Prints line having word as 'Smidsy'
11)grep "^....$" File_grep → It Prints line having exactly 4 characters
12) grep "^&" File_grep → It Prints line Starting with '&' character
13) grep "&$" File_grep → It Prints line Ending with '&' character
14) grep -n "^$" File_grep → It Prints Empty Lines
15) grep -v "^$" File_grep → It Prints Non-Empty Lines
16) grep -c "^$" File_grep → It Counts Empty Lines

fgrep: It is used to search multiple strings, but it doesn't allow searching regular expressions. It searches the strings faster that the grep.

$ fgrep 'Unix
Oracle
dba'' Student → It prints line containing Unix, Oracle or dba from
Student file

$ fgrep "Hari → It prints line containing Hari, Oracle or 103 from
Student file
Oracle
103" Student

egrep: It is combination of grep & fgrep plus some additional wild card characters.

1) + → Matches one or more occurrence of previous char
2) ? → Matches Zero or One occurrence of previous char
3) expr1 | expr2 → Matches expr1 or expr2
4) (x1|x2)x3 → Matches x1x3 or x2x3
5) (m, n) → It matches minimum 'm' occurrences and maximum 'n' occurrences of its preceding character
6) (m,) →It matches minimum 'm' occurrences of its preceding character

1) egrep 'gg+' file_egrep → Matches one or more occurrence of character gg
2) egrep 'gg?' file_egrep → Matches zero or one occurrence of character gg
3) egrep 'sengupta|dasgupta' file_egrep → It prints line containing either of the pattern
4) egrep '(sen|das)gupta' file_egrep → It matches either of the pattern
5) egrep 'ab{2}c' file_grep → It matches exact occurrence of its preceding char
6) egrep 'ab{2,3}c' file_grep → It matches min 2 & max 3 occurrence of its preceding character
7) $ egrep "\<[0-9]{2}\>" Efile_grep → It matches exact 2 digit numbers
8) $ egrep '\<[0-9]{2,5}\>' Efile_grep → It matches 2 or 3 or 4 or 5 digit numbers
9) $ egrep "ab{3,}c" Efile_grep → It matches minimum 3 & maximum 'n' of 'b'character
10) $ egrep '\<[0-9]{3,}\>' Efile_grep → It matches minimum 3 digit numbers & maximum 'n' digits

bc: bc command use to open the Unix calculator and perform the arithmetic operations.

```
$ bc
6+10
16
Ctrl+d

$ bc
12*12; 10*10
Ctrl+d
```

Note: bc shows the output of the computation in the next line. Start **bc** again and then make multiple calculations in the same line, using the ; as delimiter. The output of each computation is, however, shown in a separate line.

sed [Stream Editor]: It is used to search and replace a string and it is multi-purpose filter string.

Syntax:

sed options 'address action' file
sed 's/old string/new string/g'

s → means substitution
g → means every line all occurrences
without g → means every line 1st occurrences

1) sed 's/smidsy/Saif/g' file_sed
2) sed 's/smidsy/Saif/gi' file_sed
3) sed 's/smidsy/Saif/' file_sed
4) sed 's/smidsy/Saif/i' file_sed
5) sed 's/^tech/Saif/g' file_sed → It replaces 'tech' with Saif
6) sed "s/^tech$/Saif/g" file_sed → It replaces exactly 'tech' with Saif
7) sed "s/\<tech\>/Saif/g" file_sed → It replaces 'tech' with Saif anywhere in file
8) sed "s/^$/***/g" file_sed →It replaces all Empty Lines with '***'
9) sed "s/\<smidsy\>//gi" file_sed → It deletes 'smidsy' word from the file
10) sed "s/[0-9][0-9]/*/g" file_sed → It replace 2 digit number with '*'

11) sed '3q' file_sed → It quits from the 3rd line
12) sed -n '1p' file_sed → It prints 1st Line
13) sed -n '3p' file_sed → It Prints 3rd line
14) sed -n '3,5p' file_sed → It Prints 3rd to 5th lines
15) sed -n '1,2p
    7,9p
    $p' file_sed → It prints first two lines, 7-9 & last line
15A) sed -n '3,$!p' file_sed → Don't print lines 3 to end (! means Negating the action)
16) sed -n '$p' file_sed →It Prints Last Record from a file
17) $sed -n '1,$p' file_sed → It Prints 1st to Last Record from a file
18) $ sed -n '1p
    $p' file_sed → It Prints 1st& Last Record from a file
19) $ sed -n '1p;$p' file_sed →It Prints 1st & Last Record from a file
20) $ sed '1d' file_sed →It Deletes first line from a file
21) $ sed '4d' file_sed →It Deletes 4th line from a file
22) $ sed '3,5d' file_sed → It Deletes 3rd to 5th Lines from a file
23) $ sed '$d' file_sed → It Deletes Last Line from a file

24) $ sed '3,5w file_temp' file_sed → It Writes/Copies 3rd to 5th line from Sed_File to file_temp
25) $ sed 's/\<[0-9]\{2\}\>/*/g' file_sed → It replaces exact 2 digits with '*'
26) $ sed '/\<[0-9]\{2\}\>/s//*/g' file_sed → Same as Above

Piping (|): It is used to combine two or more commands as pipe always execute left side command & O/P goes to right side as I/P.

1) Count number of users connected to the server?
$ who | wc -l

2) Count number of files in current directory?
$ ls | wc -l

3) Count total number of sub-directories in current directory?
$ ls -l | grep '^d' | wc -l
$ ls -l | grep -c '^d'

4) Display today's date day?
$ date | cut -c 1-3

5) echo "There are `ls | wc -l` files in the current directory"
Note: Command substitution is enabled when backquotes are used within double quotes. If you use single quotes, it's not.

tee: tee is an external command and not a feature of the shell. It handles a character stream by duplicating its input. It saves one copy in a file and writes the other to standard output.

1) $ cat Saif | tee Temp → It copies Saif file data to Temp and also displays data to the screen
2) $ cat Saif | tee Temp Temp1 → It copies Saif file data to Temp, Temp1 and also displays data to the screen

Head: It displays FIRST 'n' lines from a file

$ head -1 Saif →It displays FIRST line from the file.
$ head -3 Saif →It displays FIRST 3 lines from the file.

Tail: It displays LAST 'n' lines from a file

$ tail -1 Saif →It displays LAST line from the file.
$ tail -3 Saif →It displays LAST 3 lines from the file.

Note: By Default, head & tail displays 10 lines of a given file.

File Permissions:

1) Default Permissions for Regular File:

Open File → Read
Write or Modify → Write
Execute → Read & Execute

Default:

rw -	r w -	r w -
User	Group	Others

2) Default Permissions for Directory File:

ls → Read
Create or Delete Files → Write
cd → Read & Execute

Default:

rwx	r w x	r w x
User	Group	Others

$ ls –l → To check permissions on files

chmod: It is used to change file permissions

Syntax:

chmod who/[+,-,=]/Permission File_Name/Directory_Name

Who:

User/Owner →u
Group →g
Other →o

(+) →Add Permission
(-) →Deny Permission
(=) →Assign Permission

Permissions:

Read → r
Write →w
Execute →x

Examples:

1) $ chmod u+x file_name → It adds Execute Permission to User Members
$ ls -l file_name → See the change in permissions

2) chmod u-wx, g-x, o-x file_name → It removes Permissions of Write, Execute for User and Execute for Group & Others from file_name
$ ls -l file_name → See the change in permissions

3) $ chmod u=w file_name →It adds only Write Permission to User members and it deletes Read and Execute Permission from User members.
$ ls -l file_name → See the change in permissions

Absolute Permissions:

Octal Code (2nd Method)	
Read	→4
Write	→2
Execute	→1

0	→No Permissions	4	→r
1	→x	5	→r, x
2	→w	6	→r, w
3	→w, x	7	→r, w, x

1) chmod 477 file_name →It gives Read Permission to User, Read/Write/Execute Permissions to Group & Others
$ ls -l file_name → See the change in permissions

2) chmod 700 file_name →It gives Read/Write/Execute Permission to User and No Permissions to Group & Others
$ ls -l file_name → See the change in permissions

3) chmod777Permission_File →It gives Read/Write/Execute Permission to User, Group & Others
$ ls -l file_name → See the change in permissions

4) chown: To change Owner of the File/Directory
Syntax:
$ chown Owner_Name File/Directory

5) chgrp: To change Group of the File/Directory
Syntax:
$ chgrp Group_Name File/Directory

UMASK:
When you create files and directories, the permissions assigned to them depend on the system's default settings. The UNIX system has the following default permissions for all files and directories.

1) rw-rw-rw- → (Octal 666) for Regular files.
2) rwxrwxrwx → (Octal 777) for Directories.

first 0 indicate the value is octal; umask can be assigned new value by:
$ umask 022

This become 644 (666 - 022) for files & 755 (777 - 022) for directories.

Tar:
The tar command used to rip a collection of files and directories into highly compressed archive file commonly called tarball or tar, gzip and bzip in Linux. The tar is most widely used command to create compressed archive files and that can be moved easily from one disk to anther disk or machine to machine.

Tar Usage and Options:
c – Creates a new .tar archive file.
v – Verbosely show the .tar file progress.
f – File name type of the archive file.
x – Extract an Archive file
t – Viewing content of Archive file
j – another compressing option tells tar command to use bzip2 for compression
z – zip, tells tar command that creates tar file using gzip.
r – Append or Update files/directories to existing archive file
W - Verify an archive file
Wildcards – Specify patterns in Unix tar command

1) Create tar Archive File
The below example command will create a tar archive file test.tar for a directory /home/user/test.tar in current working directory.
$ tar -cvf test.tar /home/user

2. Create tar.gz Archive File
To create a compressed gzip archive file we use the option as z. For example, the below command will create a compressed MyImages-14-09-12.tar.gz file for the directory /home/user. (Note: tar.gz and tgz both are similar).
$ tar cvfz test.tar.gz /home/user

3. Create tar.bz2 Archive File
The bz2 feature compress and create archive file less than the size of the gzip.
The bz2 compression takes more time to compress and decompress files as compared to gzip which takes less time. To create highly compressed tar file, we use option as j. The following example of command will create a test.tar.bz2 file for a directory /home/user. (Note: tar.bz2 and tbz is similar as tb2).

$ tar cvfj test.tar.bz2 /home/user
OR
$ tar cvfj test.tar.tbz /home/user
OR
$ tar cvfj test.tar.tb2 /home/user

4. Untar tar Archive File
To Untar or extract a tar file, just issue following command using option x (extract). For example, the below command will Untar the file test.tar in present working directory. If you want Untar in a different directory then use option as -C (specified directory).

Untar file in Current Directory:
$ tar -xvf test.tar

Untar file in Specified Directory:
$tar -xvf test.tar -C /home/user

5. Uncompress tar.gz Archive File
To Uncompress tar.gz archive file, just run following command. If would like to Untar in different directory just use option -C and the path of the directory, like we shown in the above example.
$ tar -xvf test.tar.gz

6. Uncompress tar.bz2 Archive File
To Uncompress highly compressed tar.bz2 file, just use the following command. The below example command will Untar all the files from the archive file.
$ tar -xvf test.tar.bz2

7. List Content of tar Archive File
To list the content of tar archive file, just run the following command with option t (list content). The below command will list the content of test.tar file.
$ tar -tvf test.tar

8. List Content tar.gz Archive File
$ tar -tvf test.tar.gz

9. List Content tar.bz2 Archive File
$ tar -tvf test.tar.bz2

10. Untar Single file from tar File
To extract a single file called File1.sh from Files.sh.tar use the following command.
$ tar -xvf Files.sh.tar File1.sh
OR
$ tar –extract –file=Files.sh.tar File1.sh

11. Untar Single file from tar.gz File
To extract a single file File1.xml from Files.tar.gz archive file, use the command as follows.
$ tar -zxvf Files.sh.tar.gz File1.sh
OR
$ tar –extract –file=Files.sh.tar.gz File1.sh

12. Untar Single file from tar.bz2 File
To extract a single file called File1.php from the file Files.tar.bz2 use the following option.
$ tar -jxvf Files.sh.tar.bz2 File1.sh
OR
$ tar –extract –file=Files.sh.tar.bz2 File1.sh

13. Untar Multiple files from tar, tar.gz and tar.bz2 File
To extract or Untar multiple files from the tar, tar.gz and tar.bz2 archive file. For example, the below command will extract “file 1” “file 2” from the archive files.
$ tar -xvf test.tar “File1” “File2”
$ tar -zxvf test.tar.gz “File1” “File2”
$ tar -jxvf test.tar.bz2 “File1” “File2”

14. Extract Group of Files using Wildcard

To extract a group of files we use wildcard based extracting. For example, to extract a group of all files whose pattern begins with .php from a tar, tar.gz and tar.bz2 archive file.

$ tar -xvf test.tar -- wildcards '*.php'
$ tar -zxvf test.tar.gz -- wildcards '*.php'
$ tar -jxvf test.tar.bz2 -- wildcards '*.php'

15. Add Files or Directories to tar Archive File

To add files or directories to existing tar archived file we use the option r (append). For example, we add file xyz.txt and directory php to existing test.tar archive file.

$ tar -rvf test.tar xyz.txt -- Adding a File
$ tar -rvf test.tar php -- Adding a Directory

16. Add Files or Directories to tar.gz and tar.bz2 files

The tar command doesn't have an option to add files or directories to an existing compressed tar.gz and tar.bz2 archive file. If we do try will get tbe following error.

$ tar -rvf test.tar.gz xyz.txt -- Adding a File
$ tar -rvf test.tar.bz2 php -- Adding a Directory
tar: This does not look like a tar archive
tar: Skipping to next header
xyz.txt
tar: Error exit delayed from previous errors

17. How to Verify tar, tar.gz and tar.bz2 Archive File

To verify any tar or compressed archived file we use option as W (verify). To do, just use the following examples of command. (Note: You cannot do verification on a compressed (*.tar.gz, *.tar.bz2) archive file).

$ tar tvfw test.tar --- did not understood

18. Check the Size of the tar, tar.gz and tar.bz2 Archive File

To check the size of any tar, tar.gz and tar.bz2 archive file, use the following command. For example, the below command will display the size of archive file in Kilobytes (KB).

$ tar -czf - test.tar | wc -c
$ tar -czf - test.tar.gz | wc -c
$ tar -czf - test.tar.bz2 | wc -c

Vi editor:

It is used to create files or to modify existing files. It is classified into 3 types.

1) Command mode: The default mode of the editor where every key pressed is interpreted as a command to run on text. Unnecessary pressing of Esc in this mode sounds a beep but also confirms that you are in this mode.

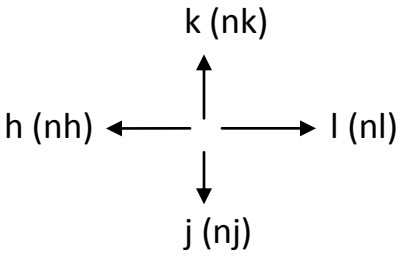
2) Input/Insert mode: Every key pressed after switching to this mode actually shows up as a text.

3) Ex Mode (Last Line Mode): This mode is used to handle files like saving & perform substitution. Pressing (:) in Command Mode invokes this mode.

The following are the commands to shift from Command Mode to Insert Mode.

A →It places cursor at end of the current line
a →It places cursor at right side of the cursor line
I → It places cursor at beginning of the current line
i →It places cursor at left side of the cursor line
o → Opens new line below current line
O → Opens new line above current line

Command mode commands:



(n) means any number

1) w (nw) →Next Word Starting
   e(ne) →Word Ending
   b(ne) →Previous Word Beginning
2) $ →End of the current line (End Key)
   ^ →Beginning of the current line (Home Key)
3) H →Beginning of the Current Page
   M →Middle of the Current Page
   L →End of the Current Page
4) Ctrl+f →Forward on Page (Page Down)
   Ctrl+b → End of the Page (Page Up)

5) x (nx) → Delete Current Character (Del Key)
6) X → Delete Previous Character(Backspace Key)
7) dw(ndw) →Delete Current Word
8) dd(ndd) →Delete Current Line
9) d$ → Delete Current Position to End of the line.
10) d^ → Delete Current Position to Beginning of the line.
11) yw(nyw) → To Copy a Word
12) yy(nyy) → To Copy a Line
13) y$ → It Copies Current Position to End of the line
14) y^ →It Copies Current Position to Beginning of the line
15) p →Paste Below the Cursor
16) P → Paste Above the Cursor
17) J →To Join a line
18) cc → To Clear a line
19) u → undo
20) c^ →Cut Current Position to Beginning of the line
21) c$ → Cut Current Position to End of the line

Ex Mode Commands:

:w →Saves file & remains in editing mode
:x →Saves file & quits editing mode
:wq →Same as above
:q →Quits editing mode when no changes are made to file
:q! →Quits editing mode but after abandoning changes
:n →Places Cursor At Nth Line.
:$ →Places Cursor At Last Line In The File
: Set nu →Set Line Numbers
: Set nonu →To Remove Line Numbers
:! <UNIX command> → Execute UNIX Command
:/string/ →Top To Bottom Search [n → next occurrence]
: ?string? → Bottom to Top Search [N → Previous occurrence]