Unix

Unix is an operating System 1969 [1971] by (originally made) Kenn Thompson & Dennis Ritchie.

- Written in C, Assembly language

First unics is written in Assembly level then it is written in C and named unix.

Feature -

(i) Multi user (ii) Multitasking (iii) Multi process (iv) file structure like tree (v) Open source (vi) Portability (vii) Programming facility

(*) In unix 'man' command is used to view help content on any command.

Similarities between MS dos & unix :

- only one user
- command line interface
- Pipes & I/O redirection concept
- Hierarchical directory
- Wildcard character concept

Unix & Windows

- both Multitasking
- built in TCP/ip Protocol

Unix	MS Dos
have GUI	no GUI
multi user	Single user
/- Separate director.	\ to Separate director.
used in Server	is used in embedded system
has shell Script	has batch files
has concept like process priority	no concept like process priority
case Sensitive	not case Sensitive

Commands :=>

(1) ls : list the content of directory

ls → list the name of files of current directory

ls -l → Gives a long list file in your directory. Beside file name there is also file owner, number of character etc detail available.

ls -a → cause all file listed including hidden one that begin with period(.)

- F → executable files (*)
- x - multiple column
- R → Recursive manner
- r → Sort file in reverse

(2) pwd : stands for "print working directory"

pwd -L : Print the symbolic path

pwd -P : Print the actual path

If only type pwd then pwd -P is assumed

(3) PS : The PS means Process

It report the information on current running processes

PS

↓

The result contain four column of informatio

PID – pplid	TTY	TIME	CMD
1422	pts/0	00.00.00	bash
1470	pts/0	00.00.00	ps

PID → Process id

TTY → The name of console user is logged in

TIME → Amount of CPU process has been running in minute & seconds

CMD → command that launches the process

(4) time : It prints the summary of real-time, user CPU time and System CPU time spent by the command

- time command

Ex- time cal

⇒ cal will show

real 0m0.002s

user 0m0.002s

time -p command

It prints in posix format

3.00

time -p sleep

real 3.00 sys 0.00

user 0.00

(5) cal : cal means calender

cal → Prints the current month

cal -y → To display complete year

cal -m monthno (1-12) → display month of current year

cal -[illegible]h → Dont highlight the current date

cal -y 2000 or cal 2000 } — display calender of 2000

ncal — vertical calender Representati.

cal -m month year → Display the month of that year

(cal -m 12 2000)

cal 12 2006

cal -w → will print week no

(6) touch : This command used to creat change or modify timestamps of a file

- touch filename → used to create empty file

ex- touch ari.txt
or
touch ari.txt anisha.txt // we can create multiple file also.

- touch -a filename — changes access time of the file to current time
like- touch -a ari.txt

Stat ari.txt
↓
will give all the properties of file.

- touch -m filename — changes modification time only like- touch -m ari.txt

- touch -c filename — is used to check whether file is created or not. if not then don't create it. (it changes access & modify time also)

- touch -r file1 file2 — replace the time stamp of file 2 with file 1

⑦ Cat :

Cat means Concatenate is used to read data from file. Creat, view & concatenate files also

- `Cat filename` — It will show the content of given file
- `Cat file1 file2` — This will show content of both file
- `Cat -n filename` — This will show contents Proceedding with line numbers.
- `Cat >filename` — Creat a file with the given name Then we can write anything and after finishing Press ctrl+D to exit
- `Cat file1 file2 ... filen >filenewname` — Concatenate two or more text file to a new file.
- `Cat file1 > file2` — Copy the contents of file 1 to file 2 (may be existing / if not exist then automatically Created)

- `cat file1 >> file2` - Appends the contents of file1 to file2

This also create file for non-exist

(8) mv :=> mv stands for move.

- It renames a file or folder
- It moves a group of files to a different directory

- `mv source_name_file destination_name_file` - If the destination file doesn't exist it will be created
- If it is exist then it will be overrighted.

```
mv a.txt b.txt
```

- mv -f → force move by overwriting destinat. without prompt
- mv -i → interactive Prompt
- mv -b → taking backup before overwritten with tilde (~) character

⑨ CP => Cp stands for copy files or group of files or directory.

Cp source destination

Cp source Directory

Cp source1 source2 ... Directory

- Cp srcfile Desfile — it copy content of 1st file to 2nd file. If 2nd file does not exist then it will create then overright without waring.

- Cp src1 src2 src3 Directory

Copy all files to the directory

- Cp Dir1 Dir2 — copy all files of dir 1 to dir 2

-f → force -n - existing should not overwrite.

-b → backup -R - copy entire directory

-i → interactive -u - update copy never

-r → recursive -v - information message

-p → Presure the properties (access, Modified time)

(10) kill : kill command is used to terminate the process manually.

- kill -l — display all the available signals (number, SIG Prefix, without SIG Prefix)

* negative PID indicates it is a group of process

- kill PID — To kill a specific process.

You can get PID by running command PS (process ID)

(11) nice :⇒ nice command helps execution of a process with modified scheduling priority. It launches a process with a user defined scheduling priority. If we give a process higher Priority kernal will allocate more CPU time.

- Default Priority is always 0 if you write nice it will give 0

Niceness value — -20 (most favourable Process high Priority) to 19 (Least favourable / low Priority)

- `ps -el | grep terminal` — to check the nice value of a process

- `nice -10 gnome-terminal` — Set the priority of the process
  - `-n` n is between -20 to 19
  - `--10` — negative Priority Set

(12) renice ⇒ Renice command allows you to change or modify the scheduling priority of an already running process

- `sudo renice -n 15 -p 77982`
  - optional

15 is new priority — p 77982 means process id 77982

- `renice -n 10 -g 4` — change priority of all process in a group
  - 4 is the group id.

(13) Sleep :- ⇒ Delay for a specific amount of time

Syntax Sleep number[suffix]

- Sleep 3 — it will sleep for 3 second if no suffix mention then second is default
- Sleep 3m — It will delay the executing of current instruction for 3 minute

(14) bc :- ⇒ bc stands for basic calculator. is an arbitary-precision calculator Language with syntax similar to C.

It is used for command line calculator

- echo 12+5 will print 12+5

but if we write echo 12+5 | bc then it will give ↓ 17

- We can also use bc to compute numerous mathematical operation that are contained in a text file

- bc < filename.

(15) sort :⇒ It is used to sort the contents of a file.

By default [sort filename.txt] sort according to ascii Value line by line

number will first
Uppercase starting in line
Lowercase

Sort Axis.txt

1089
I love you
saranghae anisha

I love you
Lo89
saranghae anisha

note— It does not replace the sorted data in existing file.

options —

- -o : to write the sorted output in a new file

[Sort -o file.txt input file]

- -r : Sorting in reverse order (z-a) (Z-A) (9-0)
- -n : Sort the numerically

12
13
19
120
...

ascending

- -nr : reverse numerically descending
- -k = Sorting a table on the basis of column number. (primary key sorting)

- -c : to check whether given file is sorted or not
  if sorted no o/p else o/p

- -M : Sort by month

- -u : sort and remove duplicates sorted file will be stored

uniq - duplicate data removed | sort file1.txt > file sorted.txt
↳ redirection symbol

diff :

cmp : Compare two files and show the first mismatch on which line and which # byte
~~Hide folder~~ no difference - no o/p

(1b) diff :→ comparing two text files and display exactly where the ~~match~~ mismatch it

A	A	c - change
B	B	< first file
<	M	> - second file
D	N	d - delete
		a - add

file1 file2

diff file1 file2

↓ 3,4 < 3,4

3 & 4 line of first file to change with 3, 4 line of file2

< C
< D
< M
< N

A A
B B
C
D
1 2

3, 4 add 2 < equal to 2nd line
3rd line
< C
< D

(X) A A
B B
C

2 & 3 add 3rd line file 2 to 2nd line of file 1
> C

(17) Comm :⇒ Compare two file

- used to compare two sorted file
- provide o/p in 3 columns
- First column display unic line of first file.
- Second column display unic line of 2nd file
- Third column display common line in 2 files

(XX) must sorted both file

(X) Arijit
Banu
Chandan
Zebra

Arijit
Bisakha
Chandan
Sayan

file 1

file 2

Comm file1.txt file2.txt

Arijit

Babu Bisakha Chandan

Zebra Sayan

1st column 2nd column 3rd column

(*) if not sorted it will give Comment that file is not sorted.

(18) wc : word count lines, chars to count total no of words in a file

O/P number of lines number of words total characters

wc filename

wc filename1 filename2

file 1

Arijit Maity
Anisha Mondal
Sayan Chanda
priyanka Jana

wc file1.txt

4 8 55 filename

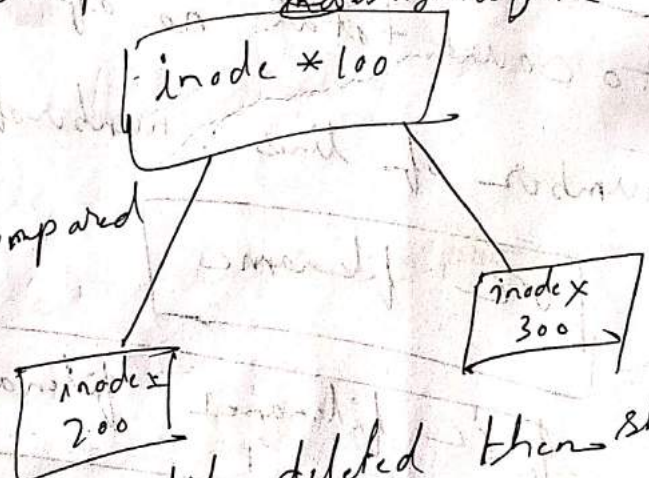
wc - it takes input from user then show result

-l → only lines will be displayed
-w → only words
-c → only character
-lw — only lines & words

(15) ln :=> Make links between files

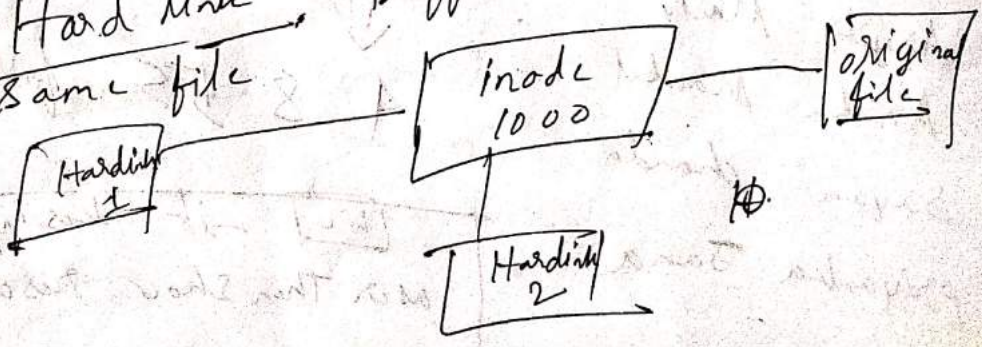
Every file in the system has an inode (index node)

Soft link: It is like shortcut. it is a pointer which points to another file



if original file deleted then shortcut will not open file.

Hard link: Different name of the same file



Hardlinked size is same as original file.
If original file deleted still hardlink file will be exist.

- Ln command used to create hardlink and softlink.

original file
hardlink created file
`ln file1 file2` — will create hard link
if you edit any file then every hardlink & original will be edited

original file
softlink created file
`ln -s file2` — will create soft link
if original file deleted then softlink will not be accessible

Same

---

20. `du` :⇒ disk usage
is used to estimate file space usage by given files or directories.

`du` — prints all the size of current directory files then at last show the total size.

`du -h` → h means human readable
It shows same with byte, GB etc

`du -s` – display only the summarize version

`du -a` – will, not just directories but all files.

`du directory_name` – will show data about specific directory.

---

(4) chmod → change mode of file
- Can take multiple files as arguments

3 types of roles →
- owners / users (u)
- groups (g)
- others (o)

3 types of permission:
- read (r)
- write (w)
- execute (x)

When we give `ls -l` the
it will give a lot of information
magic number group
-rw-r--r--
user others
it is a file
drwxrwxrw-
directory

ugo = a – means all

Project - XYZ

Module 1	Module 2	Module 3
A B C	D E F	G H I

A created a file sample.txt

A - user/owner DEFGHI - others

B, C - group (working in same module) (working in other Module)

chmod command used in 2 method

(1) Symbolic/Text mode.

touch sample.txt
ls -l sample.txt
- rw- r-- r--
file user group other

+ → Add specified modes

- → Remove specified modes

= → Set the permission and remove others existing permission

Previous permission - rw- r-- r-- Adding only

chmod u+x sample.txt execute permiss. to user

chmod u+x, g+rw, o+rw Sample.txt
must comma no space allowed
↓ Same
chmod u+x, go=rw Sample.txt
adding execute to user & read, write to user group.

(ii) numeric way:

read (r) — 4
write (w) — 2
execute (x) — 1

read write exe
r w x
4 2 1

chmod 0 0 0 sample.txt
owner, groups, others

chmod 6 5 4 sample.txt
read+write, read+execute, read

(x) chmod file1 file2 — possible

(22) chown ⇒ change owner

is used to change the file owner or group.

chown owner file — to change owner of the file

chown ani file.txt
ownername — filename

-c : Reports about file change mode

same { -v

chown -c owner file

chown :group_name file_name → To change group name

chown :group2 sample.txt

chown owner:group_name filename → To change owner as well as group

chown --from=previous_owner newowner file name

change ownership from current owner to new owner where current owner must be same as we write. chown --from ari root sample.txt

chown --from=:privious group newgroup file

for group

chown --reference=oldowner newowner

copy ownership of oldowner to new owner

chown owner:group * → changing user and group to all.

-R- Recursively change group ownership of a directory and any of its subdirectories all of contents inside

(23) chgrp :⇒ change group ownership of a file or directory

chgrp groupname file name / folder

↓

change group ownership of file.

-R → Recursively all subdirectory and file

-c → Describe action

-v

chgrp --reference=abc.txt sample.txt

↑ change group

(24) ~~20~~ Top :⇒

is used to show the linux process running status in details

Top half portion contain statistic of process and resource usage

Lower half contain a list of currently running process.

It will update frequently, untill you press q it will not terminate.

`top -n 10` — automatically exit after 10 seconds of refreshing (number can vary)

`top -u username` — Display specific user processes

Press Z — display running process in color.

Press L — display absolute path of running process

Press K — to kill a process

tail also used for cutting byte

(25) cut ⇒ it is used to cutting out sections from each of lines of file.
it can be used to cut parts of a line by byte position, character & field

`cut filename` will give error (you must specify a list of byte/ char/ field.

`cut -b 1,2,3 file.txt` or `cut -b 1-3 file.txt` — cutting 1st 3 byte of each line

`cut -b 1- file.txt` — cut from 1st byte to end byte of a line

`cut -b -3 file.txt` — cut from 1st byte to 3rd byte of every line

(X) tabs & backspace is considered as character of 1 byte

`cut -c -3 file.txt` — cutting by character first 3 character

(X) cut uses [tab] as a default field delimiter

`cut -d "delimiter" -f field_number file.txt`

Andhra Pradesh
Bihar
gujrat
Madhya Pradesh

`cut -d " " -f 1 state.txt`

Andhra.
Bihar
gujrat
Madhya.

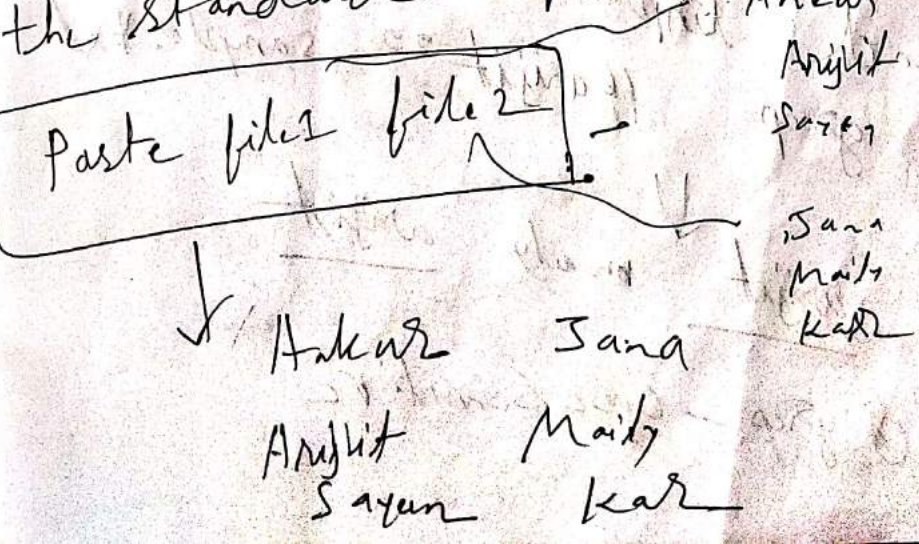
- complement :=> it complement the output like we say to print 1st field instead it will print all other field than 1st field

- output-delimiter — by default output delimeter same as input delimeter but we can modify it also

cut -d "," -f 1,2 state.txt --output-delimiter="|"

-c : specific column
-f : cutting fields -d : specified delimiter

(26) paste :=> It is used to join files horizontally (parrarel merging) by outputting lines consisting of lines from each file specified separated by tab as delimeter (by default) to the standard output.



-d : It uses tab delimeter by default for merging.
The delimeter can be changed with any other character by -d

```
paste -d "|" file1 file2
```

Arijit | Maity
Sayo. | Kar

-s : for joining lines in paste

(27) grep ⇒ Global search for Regular expression and printout.
. Searches file for particular pattern of characters and displays the matched pattern only

```
grep [option] Pattern [filename]
```

It display the matched Pattern and print the entire line

grep "arijit" sample.txt

arijit ↓ maity arijit

grep - case Sensitive

-c : print count of the lines that matches a pattern

-h : Display the matched lines but do not display filename

-i : Ignore case sensitivity

-l : display list of file name only

-n : Display matched line numbers

-v : print out lines that do not matched pattern / delete

-e exp : Specifies expression → can use multiple times

-f file - takes pattern from a file

-E - treat pattern as an extended regular expression

-w - matched the whole word only

-o → prints only matched part of matching line

-A n : print search line & n line after it

-B n : print search line & n line before that

-C n : print search line & after, before n line.

grep does not search ~~s~~ subdirectories
to search subdirectories
-R → ~~reverse search~~ recursive search

grep -winR "arijit" ./ — current directory

grep -win "arijit" ./* — all files in directory

./*.txt all text files

(*) When pattern does not found grep command silently return

(*) We can search in more than one file

(*) posix identifier — regular, extended regular

^ — for matching begin of file

$ — for matching at end of line

multiple words — single quote / double
special character — must double quote

(28) file :⇒ used to determine the type of a file

file [options] filename

file j1.txt

⇓

j1.txt : text file

-b : This is use to display only file type

* : display all filed time in current directory (file *)

-s : for special file

file -m file filename

this is written by ass

this will be checked

there is Regular expressions

Magic file checking

according to this whole file is

(29) Whereis ⇒ is used to locate binary, source, manual pages for a command. This command searches in restricted set of location.
- usually used to find executable program

man pages
configuration file

Whereis commandname

-b → only binary file
-m → only manual file
-s - only source

✱ It doesnot need root Privillage

(30) Which ⇒ location of executable file of a command

Which command

✱ It can take multiple command as argument

3 return type

0 - all command found
1 - one or more command does not exist
2 - invalid option

-a - print all matching pathname of argument

(31) echo :=> Display text or variables value on screen

it is used in scripting

| echo "strings" |

echo arijit ↓ same "arijit"

arijit

like

-n → Do not output the trailing newline

-e → enable interpretation of the following backslash escaped character

↓

\a — alert

\b — remove spaces between text

\c — supress trailing newline (\c ari → not print ari)

jit ari ...

\n — new line

\r — carriage return jit \r ari

ari (will be printed)

\t — horizontal tab

\v — vertical tab space ari jit mar

echo * print all files similar to ls command

-n - omit echoing trailing newline

myvar = 5

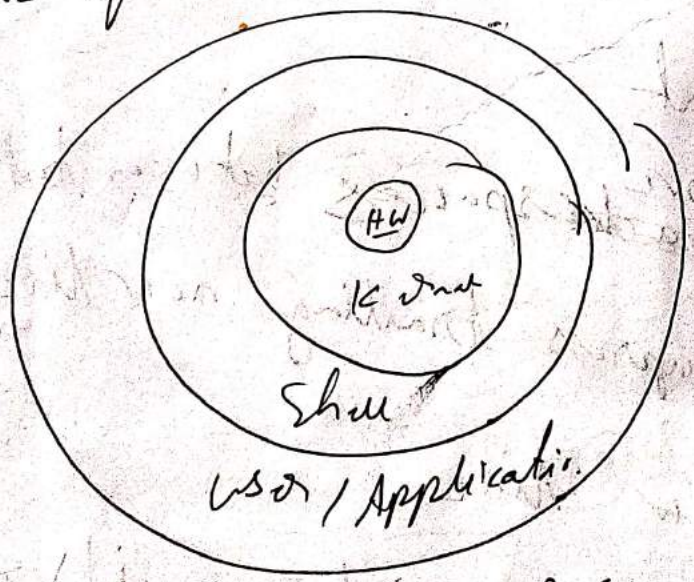
echo $myvar → 5

(32) env :⇒ Unix has two type of environment variable

- System defined (usually Capital letter)
- user defined (usually lower case)

env to display current environment variable.

Value of environment variable — echo $HOME



Shell takes input & pass to Kernal

Shell is also a process

(X) When we enter any command it goes to path address (echo $PATH) if it is found it execute or it give error

(33) PATH :=> PATH is a environment variable

- it shows all the directory that is being searched when you enter any command.

```
echo $PATH
```

(X) We can also create our own path

(34) Classpath :=> it is a list of the class libraries that are needed by the JVM & other Java application to run your program.

you must have to set the classpath variable

```
export CLASSPATH=/root/java
```

To check it

```
echo ${CLASSPATH}
```

(35) find =>

- is a command line utility for walking a file hierarchy.
- It can be used to find files and directories and perform subsequent operation on them.
- It supports searching by file, folder, creation date, modified date, owner

find [Where to search] [-options] What to find

!!

find . -name testfile.txt — find a file named testfile.txt in current and subdirectory

find /home -name *.jpg — all jpg file in home directory

options :-

-name : Search for file name.

-i : not case sensitive

-empty : Search empty file

-perm octal Search file with entered permissio
(-perm 664)

-user name — find all file of user
'name'

-print — display the path name
of files

-newer file — Search for file that
were modified after
file

-type — Specifying file type

-inum — by inode number

-atime — access time

-mtime — modify time

-365 less than 365 days

+365 more than 365 days

-a — and condition

-o — or

-ok — Seek confirmation

Vi editor ⇒ When we connect a server through window we cant use gui (gedit-data) so we have to do via commands

The first editor in unix System is 'ed'. It can only edit files in current directory and can not execute shell commands.

That's why It has been replaced by Vi editor (Visual editor)

- used to create a file
- used to read a file
- used to edit a file.

Advantage :- • available on all the flavour of unix system
• user friendly than ed also

modes :⇒ There are mainly 3 mode

(i) Command mode :

When Vi starts up it is in command mode
• it takes command from keyboard but does not display in window.

It helps us
- move through file
- copy text
- paste text
- Delete text

(ii) Entering text in input mode data

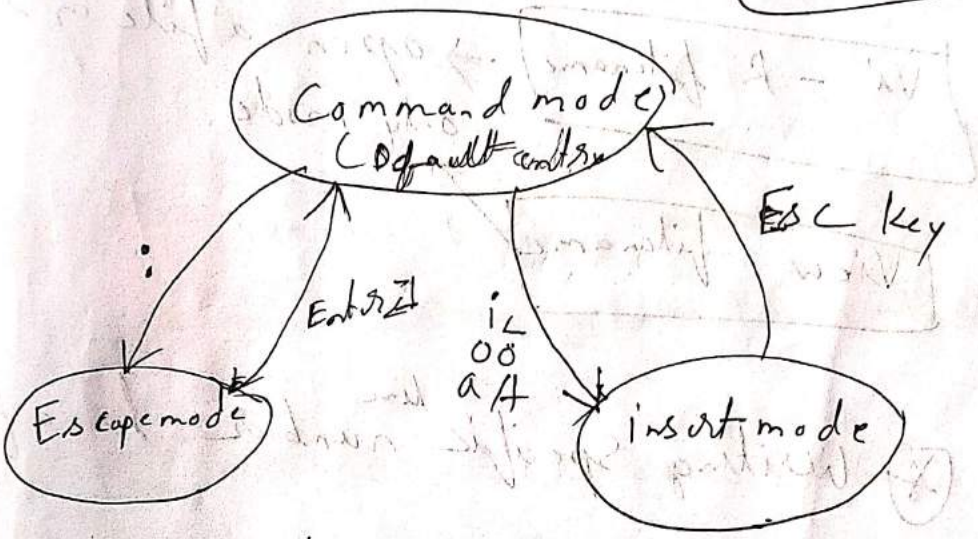
(ii) Insert mode: This mode help to insert text into file.

(ii) Escape mode (ex mode): You first have to be in command mode. Escape mode can be invoked by typing ":"

As soon as you type : cursor will jump to last line of the screen and wait for a command

- Saving a file
- exiting the vi editor

file handling + Substitution



Save & exit command:

First you have to be in command mode then press ":" you will be in escape mode

q → quit (you have to save first else it will not quit give warning)

q! - quit without saving

w - save

wq - save & quit (ZZ is also best used in command mode)

x - by

no need to be in escape mode

W filename → Write to file to filename (Saving)

W! filename — overwrite to file called filename (forcefully)

R filename — Read data from file called filename and print on screen

Basic command for starting

Vi filename: create a new file if not exist else just open the file.

Vi -R filename → open a file in read only mode.

View filename

(X) Writing specific line number to a file

:1,4w filename — 1 to 4 line save as in filename

:1,$w filename — first to last line

:.w filename — current line only

:.,$w filename — current line to last line

:$w filename — last line only

Moving Within a file :⇒

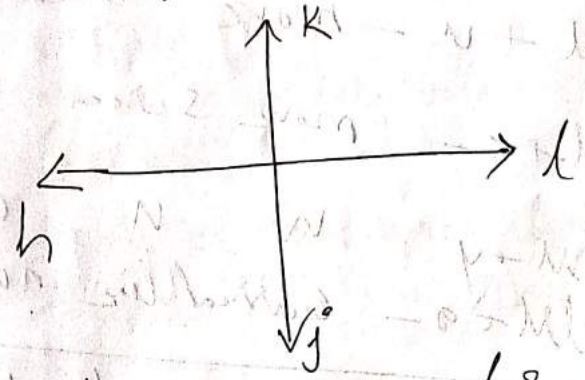
You must have to be in command mode to use below commands

k- moves the cursor up one line

j- moves the cursor down one line

l- moves the cursor to the right one character positi..

h- moves the cursor to the left one character position



We can also combine number like

5k means 5 line up from the cursor

0 or ^ → Position cursor at beginning of the line

$ → Position cursor at end of line

W → position cursor to the next Word

b → position cursor to the previous word

(→ position cursor beginning of the current sentence.

) → beginning of next sentence

H → move to top of screen
M – move to middle of the screen
L – move to bottom of the screen
e – position cursor to the end of ward

Control command (scrolling)

ctrl + d – move forward $\frac{1}{2}$ screen
ctrl + f – move " one full screen (one page)
ctrl + u – move backward $\frac{1}{2}$ screen
ctrl + e – move screen up one line
ctrl + y " " down " "
ctrl + g – current line number

G – moves the last line of file
absolute movement

ctrl + l → clear the screen

e – move forward to end of the word

b – move back to beginning of the word

Editing files :=>

i – insert text before current cursor location

I – insert text before the beginning of current line

a – insert text after the current cursor location.

A – Insert text at the end of current line.

O – create a new line below cursor location

O – create a new line above cursor location.

Deleting character

x – delete the character under the cursor location

X – Delete the character before the cursor location

dw – delete current cursor – to the next word

d^ – delete current cursor – beginning of line

d$ – delete current cursor – end of line

D

dd - Delete the line cursor is on

Change Command

cc - Removes the content of line leaving you in insert mode.

cw - chang the word the curs on is on to the end of the word

r - Replace the character under the cursor. Returns command mode.

R - overwrite multiple char currently under the cursor → infinite.

s - Replace the current character from the character you type. You left in insert mode

S - Delete the line cursor is on and replace it with new text.

Copy paste command :-

yy — copies the current line (whole)
(yanked)

yw — copies the current word from the cursor

p — put copied text after the cursor

P — put copied text before the cursor

Set command

:set ic — ignore case while searching

:set ai — set autoindent

:set nu — Display lines with line numbers

:set ro — change file type to Read only

Running Command

While working on vi you can also run any command

:! ls — it will show o/p

Word & character Searching:-

/ or ? used for searching
/ forward direction
? backward direction

. — matches single character

* — Matches zero or more previous char

$ — end of line

Searching & Replacing (ex mode)

:s/pattern/replacewordas char
— for replacing only that word

:s/pattern/replace/g — global
allover line

n — Repeating Search on same direction

N — Reverse the direction

o If there is no replace / to find pattern it will simple delete all the matching

1,$ = %

g — interactive Substitute

Wildcard :⇒ A wildcard is a symbol that takes a place of unknown character or set of characters.

Commonly used is * & ?

? connot match by

(i) The asterisk (*) :⇒ * represent any number of character.

Ari* will return Arijit, Arindam, Arima, Arisen.txt — any numbers of character

*sha will return: anisha, Anisha, Disha, Dalisha etc

(ii) The question mark (?) :⇒ ? represent only one character.

like ari?.txt will return

ari1.txt

ari2.txt

arij.txt

arijit.txt ✗

$ is not wildcard

Some command to get into insert mode automatically

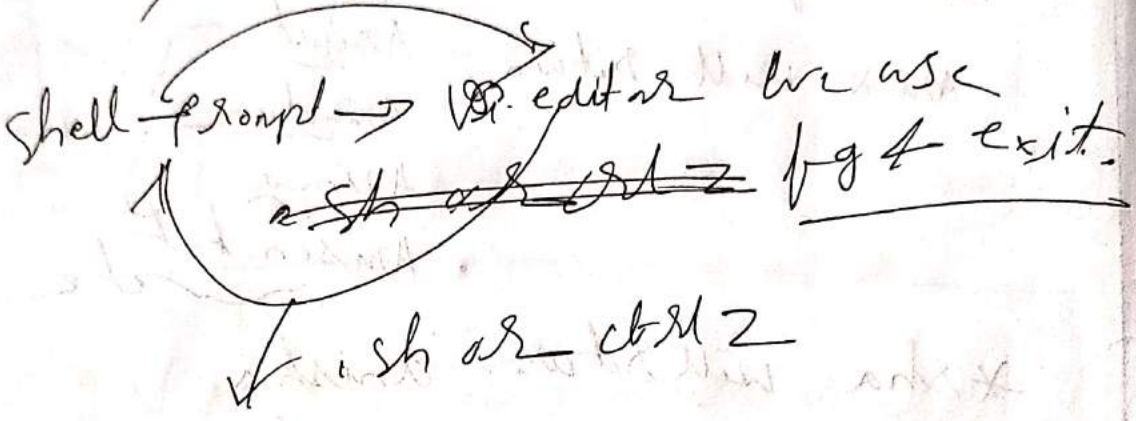
r — Replace single char (arijit → Srijit)

R — Replace multiple char (arijit → nikhil)

s — Replace single char with many (arijit → manarijit)

S — entire line replace

. → current line in ex mode



J — joining lines

u — undo most recent single editing change

U — undo all the changes of the line (cursor is on)

ctrl-R — Redo the changes by undo

. → Repeating last command

Unix Developed in 1969

Shell: ⇒ Shell is a component which user communicate with operating System (Kernel)

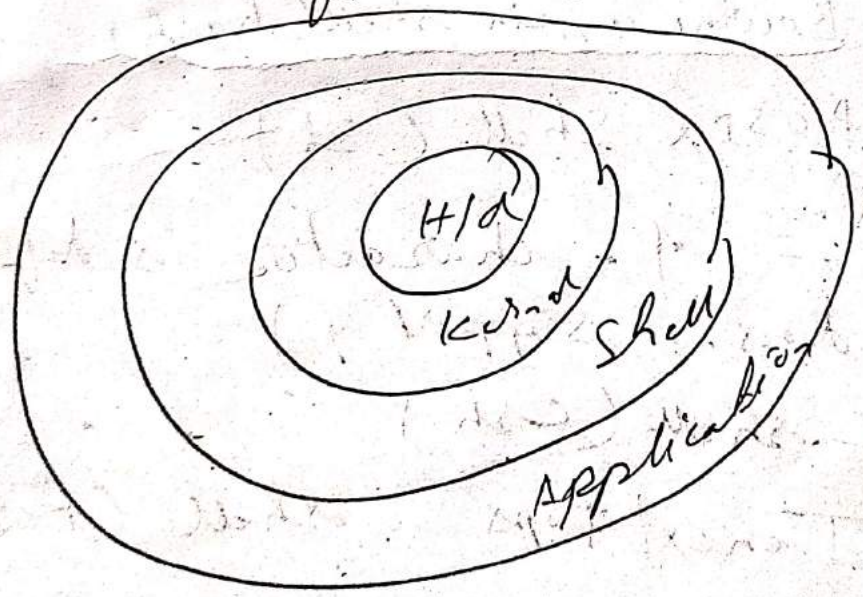
. Shell is a command line interpreter

. It translate command provided by the user and converts into a language that is understood by Kernel

ex - C shell, Bourne shell, Korn shell. again (Bash) etc

Kernel: . It interacts with the hardware.

o Memory Management, task Scheduling file management



Shell Programming

Saving with .sh is optional

A shell gathers input from user and execute program based on input.

- Shell read input after we press enter

Why need
- to avoid repetitive work
- System monitoring
- System administration backup

Types of Shell

(i) Bourne shell :⇒ $ character is default prompt
- bourne shell (sh)
- korn shell (ksh) — Most used
- Bourne again shell (bash)
- POSIX Shell (sh)

(ii) C Shell :⇒ % character is default prompt. (does not support !)
- C shell (csh)
- Tenex/Tops cshell (tcsh)

Shell script :⇒ It is a list of command which are listed in the order of execution

- A good shell script will have comments Preceded by # sign describing each step
- Shell script and function are both interpreted (not compiled)

- Start creating Shell script

(i) create test.sh using Vi editor

(ii) write below as first line

#!/bin/sh

- this tells the command that below are to be executed by the bourne shell.

It's called shebang (#- hash !- bang)

- # This is a test shell
- this is commed line

echo `pwd` — this is command — not single quote beside 1

echo "date" : this is command

(iii) Save file using wq echo $(date)

(iv) To execute script us ./test.sh

if execute permission is not granted use

chmod 777 test.sh

(v) To get user input use Read command

#!/bin/sh

echo "Whats your name"

Read name

echo "Saranghae $name"

Shell variable :⇒

A variable is nothing but a pointer to the actual data.

names : can contain (letters a-z or A to Z) numbers (0-9) or underscore (_)

- starting with number not allowed ✗
- no special char allowed ✗

_ANI ✓ 2_Anisha ✗

Sayan ✓ $ani ✗

sayan_1 ✓ var1 ✗

- Unix Shell var should be uppercase but not mandatory

Defination :⇒

```
Var_name = Value
```

name = "Arijit" // name=arijit

name is a var which contain value Arijit

Var=100

The above type of variable is called Scalar variable as it can only hold one value at a time

Accessing Value : To access value put the dollar sign($) before var name.

```
echo $name
```

- Reading variable from user :=>

To take input from user we use read

```
read name
```

input will be stored in variable named name.

- Later we can change the value of name also

- Read only Variable : Its like constant once value is given there is no way to change

```
name = "Arijit"
readonly name
name = "Anisha"  // error
```

Unsetting Var : => Deleting variable is done by

```
unset var_name
```

- We can't unset readonly variable

Variable types

(i) Local :=> Present only current instance of shell

(ii) Environment :=> is available to any child process of shell

(iii) Shell – Special variable that is set by the shell in order to function correctly.

Special Var :=> positional parameter / Harish Arijit Maity Command line argument

echo
- $$ → process id number
- $0 → the filename of current script
- $n → like $1 will give first argument, $2 " " 2nd argu
- $# → total no of argument
- $* – All the argument are double quoted
- $@ → all the argument individually double quoted
- $? → The exist status of last command execute
- $! – process no of last background command

Ex

```
echo $$
echo $# " The number of argument is $#"
echo "The first arg is $1"
echo "The 2nd arg is $2"
echo "The file name is $0"
```

./test.sh arijit mark

Arijit mark

Creating array ⇒ "0" ⇒ element

```
#!/bin/bash
name=(arijit Sayan Priyanka Anisha)
echo ${name[*]}   // will print all value
echo ${name[0]}   // will print 1st value
```

Operator ⇒ (i) arithmatic
(ii) Relational
(iii) Boolean
(iv) String
(v) File test

Bourne shell didn't have any mechanism to perform simple arithmatic operation but it uses external program like: expr / awk

(i) Arithmatic :⇒

Space

+ (addition) — `expr $a + $b`

- (subtraction) — `expr 20 - 10`

/ (division) — `expr $a / $b`

Space

* (multiplication) — `expr $a * $b`

% (modulas) — `expr $a % $b`

= (Assignment) a=$b

Space must in Condition expression.

== (equality) [$a == $b]

Return true as false

!= (not equal) [$a != $b]

(ii) Relational :⇒

Space

-eq (equal) [$a -eq $b]

-ne (not equal) [$a -ne $b]

-gt (greater than)

-lt (less than)

-ge (greater than or equal to)

-le (less than or equal to)

Space

(ii) [$a==$b] ✗ [$a == $b] ✓

(iii) Boolean :->

! — logical negation [! false] is true

-o — logical or. One true [$a -lt 20 -o $b -gt 100]
so whole true

-a — logical and. both true then true [$a -lt 20 -a $b -gt 10.]
space

(iv) String operators :-> a= "abc" b= "efg"

= — check if both string same [$a = $b]
false

!= — checks if not equal then true [$a != $b]

-z — check if given string is zero length [-z $a]

-n — check if given string length not zero [-n $a]
null

str — check if str is not empty string [$a]

(V) Relational operator :- (test)

File operator

-e → check if file exist even if is directory [-e $file]

-r → check if readable [-r $file]

-w - writable

-x - executable

-s - size greater than zero

-b - if block special file

-c - character special file

-d - if it is directory

-p - if name pip

-U - Set user id bit

-k - sticky bit setted

-ot - older than

-ef - linked or not

Decision making :=>

(i) if fi statement
(ii) if else fi statement
(iii) if elif else fi statement

(1) if fi :=>

Syntax = if [expression]
then statements execute if expr true
fi

Ex- a = 10
b = 20
if [$a == $b]
then
echo "a is equal to b"
fi
if [$a != $b]
then
echo "a isnot equal to b"
fi

(ii) if else fi Statement

```
if [ expression ]
then
        statement to be executed if true
else
        statement to be executed if
                                false
fi
```

(iii) if elif fi :=>

```
if [ expr1 ]
then
    Statement execute if expr1 true
elif [ expr2 ]
  then  statement execute if expr2 true
elif [ expr3 ]
then
        statement execute if expr3 true
else
        statement execute if no expr true
fi
```

(iv) Case Statement :=>

```
Case word in
    pattern 1 ) statement to be executed
        ;;
    pattern 2 ) statement to be executed
        ;;
    pattern 3 ) statement to be executed
        ;;
    * ) default condition
        ;;
esac
```

;; - indicates the program flow should jump to the entire case statement (break)

* case also use wildcard, support multipattern

Loop

① While loop

```
While command
  do
     Statement if command
                 True
  done
```

command

Eg- a=0

```
While [ $a -lt 10 ]
  do
     echo $a
     a=`expr $a + 1`
  done
```

Statement must

```
0
1
2
3
4
5
6
7
8
9
```

(ii) for loop :

for loop operate on list of item

Syn

```
for var in word .. wordn
    do
        statement
    done
```

Ex

```
for var in 0 1 2 3 4 5
    do
        echo "welcome $var time"
    done
```

welcome 0 time

" 1 time

" 5 time

var is name of variable, each time for loop execute the value of variable is set to the next word.

(iii) until → loop execute until condition ~~true~~ false

```
until command
    do
        statement
    done
```

(iv) The select - Select loop provide an easy way to create a numbered menu from which user can select option

```
Select var in word - words
    do
        Statement
    done
```

Ex Select Drink in tea coffee juice none

```
do
    case $drink in
        tea|coffee) echo "go to counter"
        ;;
        juice) echo "at home"
        ;;
        none) break
        ;;
```

*) echo "invalid selection"
;;
esac
done

(*) We can also nesting loops
use break & continue.

Substitution :⇒

Shell perform substitution when it encounter an expression that contain one or more special character.

(*) echo -e : use for interpretation of backslash escape

\a — alert bell \b - backspace
\n — new line
\c — suppress trailing newline
\r — carriage return
\t — tab (horizontal)
\v — vertical tab

Command substitution

a=`Command` → echo $a

also
$(var) — substitute the value of var

Metacharacters

There are some characters which have special meaning like

* ? [ ] ' " \ $ ( ) & ; | 200$dollars

So in order to use this (we have to destroy it special meaning

We use \ before it

Function :=> For doing repetitive work

- function name()
  {
  list of command
  }

like

```
Hello ()
{ echo "Hello word $1 $2"
return 10
}

Hello Anisha sarangkar // invoke function.
ret=$? // capture value return by last command
echo "Return value is $ret"
```

- Shell always running until we log out
- Shell first scan meta characters like * of ?
- Shell
  - multi faced
  - command interpreter
  - Provide environment
- chsh - change login shell
- $$ - it will print $ as $ = turn off special meaning
- "$var" - Print value

- Saving with .sh is optional
- Shell script execute in a separate child process
- Read firstname lastname ✓

[ ] — shorthand for test

---

expr (string handling)

determine length of the string

extract, locate the position of the string

expr "argit" : '.*'

⇒ 6 — length of string

| (:1) — extracting substring

is an external command

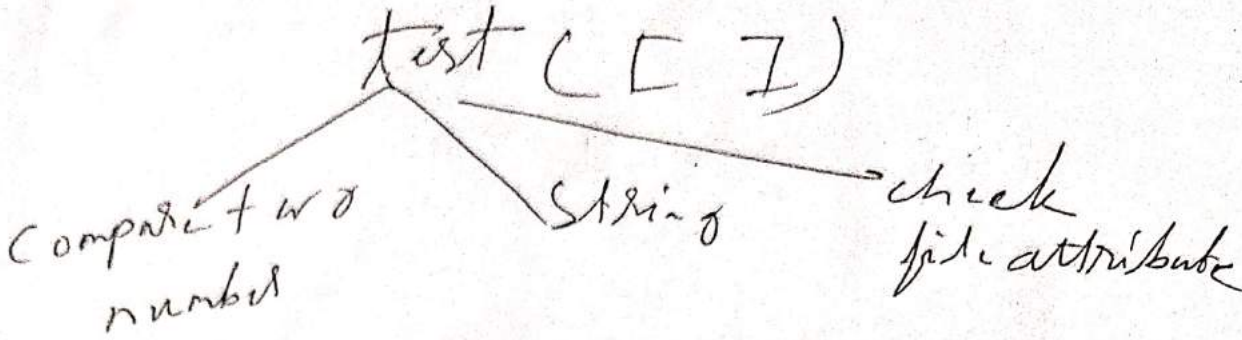
Set :- is an internal command which gives value to positional argu ($1, $2 ..)

also used for command substitution.

— used this

- << - reading data from same script file

test ([ ])



Prompt for root user is # or $.
but normal user is % or $

wall → used for communicating between root & normal users

pswd → Password.