Introduction to Unix.

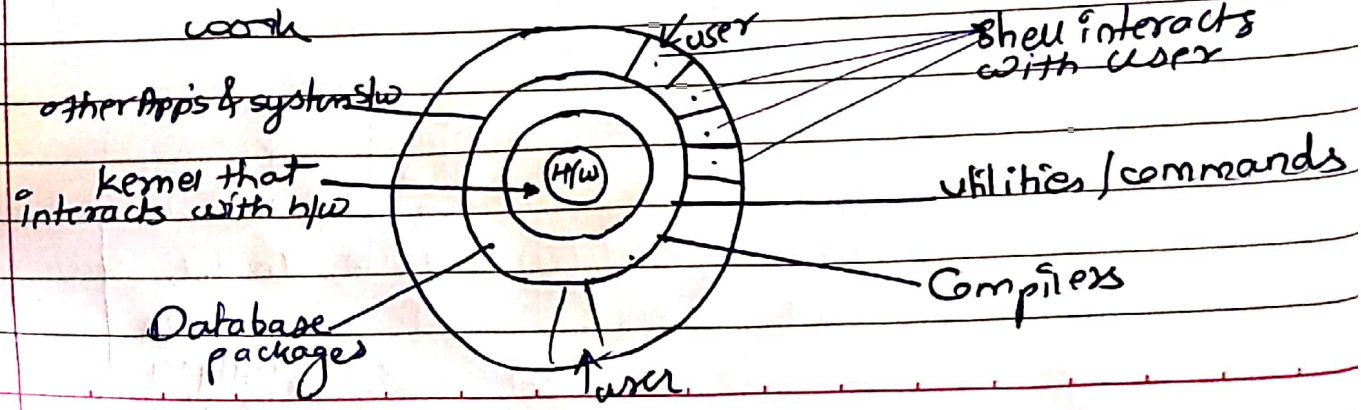
→ It is a time sharing, multi-user OS with elegant, powerful file system, command interpreter (shell) & set of utilities.
→ It is an OS for programmers.
→ Types : SCO Unix, Sun Unix, Xenix, Linux
→ Security is built-in through username & password, combined with access rights for files.

→ Principles of Unix

- multi user & multi Tasking
- Everything is file
- Small, single purpose programs
- Configuration data stored in text
- Ability to chain programs together to perform complex tasks

→ Structure of Unix System

- Shell & Kernel together make Unix System work



Unix Kernel

- Kernel is collection of programs written in C that runs directly on hw
- Kernel is directly loaded into memory when system is booted. low level jobs are its basic services
  - → System Initialization
  - → Process / memory / file / IO management
  - → Programming Interface
  - → Communication facilities

Unix Utilities

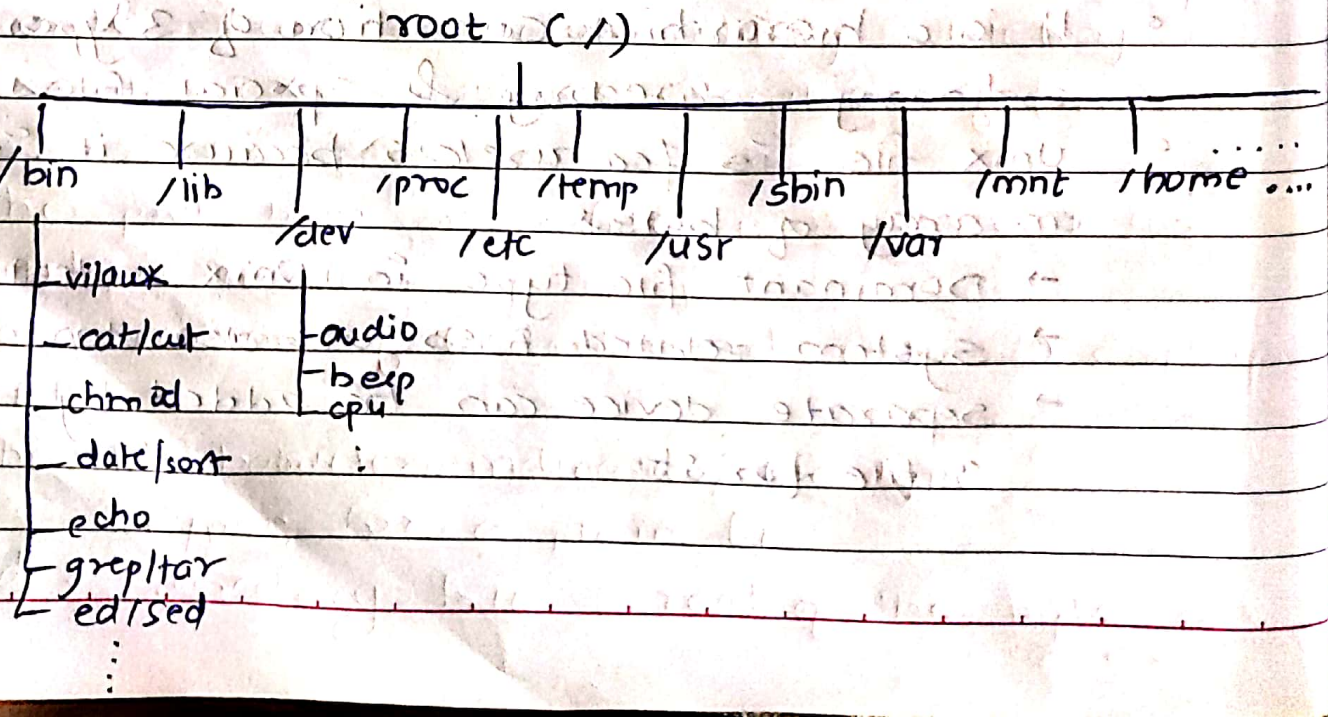
- Unix utilities / commands are collection of about 200 programs (ed, sed, awk, grep, etc) that service day-to-day processing.
- They are invoked through shell.

Unix file System (UFS)

- It is a hierarchical collection of 3 types of files ordinary, directory & special files.
- Unix file is featureless because it is simply an array of bytes
  - → Dominant file type in Unix is text file.
  - → System related files are also stored in text form.
  - → Separate device can be added by creating a file for it

UFS

- Root is the Supremo & is represented by '/'. Every subdirectory must have a parent (Root doesnot have parent)
- File names can be upto 14 characters long. Can contain both upper & lower case alphabets, digits, a dot, hyphen (-), underscore (_)
- Path names are a sequence of directory names separated by '/'. They are used to access files.
  - Absolute pathname → file location is determined wrt root
  - Relative pathname → file location is determined wrt current directory.
- Though UFS looks hierarchial, it is actually a directed cyclic graph because files can be shared



Imp directories in linux file system

1) /home → It holds users home directories. In other unix systems it can be /usr directory

2) /bin → It hold many of the basic linux progra bin stands for binaries, files that are executable

3) /usr → It holds many user oriented directories
   - bin
   - sbin → holds sys administration files
   - docs
   - man
   - games
   - spool.

4) /etc → It & its subdirectories hold many of linux config files.

5) /dev → It holds device files. All info sent to /dev/null is thrown in trash. your terminal is one of the /dev/tty files.

Internal file maintainance

- For each file created in the system, an inode is also created. Inode is a disk file record of 64 bytes that maintains the permanent attributes of file.
- An inode has following attributes.

- owner & group identifiers.
- file type & file size.
- No. of links for this file.

- Times of file creation, last file access & modification & last inode modification
- list of access rights - r/w/x permissions
- reference count showing no of times file is opened.
- physical address of file on the disk: array of 13 pointers for data storage.

• whenever a file is opened, its inode is brought into main memory. The active inode is kept there until the file is closed & is used to locate beginning of an open file on disk

File Access Permissions (FAP)

FAP refers to the permissions associated with a file with respect to following:
File owner, Group owner, other users.

Read r allows displaying / copying
write w allows editing / deleting
Execute x allows execution of file

The protection on a file is referred to as its file mode.
long version of ls command (ls -l) displays FAP.
The 1st character indicates file type,
⇒ - : hyper for a plain file
⇒ d : directory file, rest of the bytes may be r/w/x

Byte 0 is 'b' (blocks), 'c' (character) or 'p' (device) for special files

Unix process

- process in unix is program in execution
- kernel maintains a process table to manage all processes.
- 2 data structures per process are user structure & process structure

Environment Variables. (predefined / default / builtin variables)

- HOME variable.
  $ echo $${HOME} <Enter>
- PATH variable
  contains list of all full path names (separated by colon)
- PS1 variable (prompt symbol 1)
  The system prompt may be changed by setting value of this var to desired prompt.
  $ PS1 = "Hello>" <Enter>
  Hello> # can be changed only at unix command line, not within shell script.
- PS 2 variable
  prompt string for continued command line (default '>')
- LOGNAME
- etc.

cd / ⇒ root
cd ~ ⇒ home

SDS Page No
Date / /

Unix shell.

- shell is the command interpreter for various users

Shell commands

- Getting help → check manual pages.

man man → help on man command
man who → help with who command
info is
ls --help

- pwd → prints present working
- cd → change directory
- mkdir → make directory
- rmdir → remove directory
- ls → list contents of directory
- cat → display contents of file
- rm → removes files
- mv → rename / move file from one directory to another
- date → system date & time
- who → logged in user name
- echo → display string
- ln → create link

{ # → comment in unix }

wildcard characters

* → matches a string of none or any no. of chars

? → matches exactly 1 char

[abcd] } → matches exactly 1 of specified set
[a-d] } of chars or range of chars

[!abd] } → matches exactly 1 char that is not
[!a-d] } in the specified set of chars or range of chars

Vi Editor

- It is a visual editor used to enter & edit text files
- Invoking vi with / without filename puts it in command mode

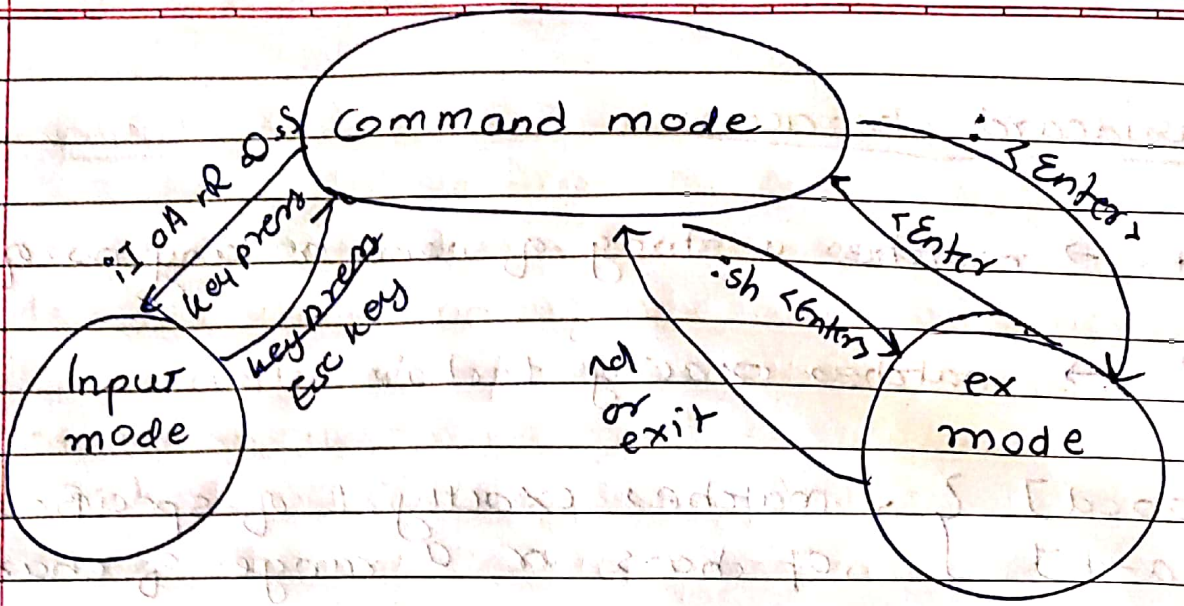
$ vi [<filename>]

vi command mode syntax

[<repeatcount>] <command>

vi works in 3 modes

1) Input → where any key is entered as text
2) Command → where keys are used as commands to act on text.
3) Ex → where ex commands can be entered in last line to act on text



Save & exit from vi

Command	Action.
ZZ	Saves file only if vi started with a filename

Save & exit command of ex mode.

:w	Save file & remain in edit mode
:wq	save file & quit edit mode
:w <filename>	write buffer to file name
:q	quit editing mode when no changes are saved
:q!	quit editing mode after abandoning changes.
:x	Save file & quit editing mode

Filters & pipes

- filter is a program that takes its input from standard input file, processes it & send its output to standard output file

Simple filters ⇒ head, tail, cut, paste, sort, uniq, etc
advanced filters ⇒ grep, pg, wc, tr, awk

grep (global search for regular expression)
- It searches for particular pattern
- grep [options] reg-exp filename

Options
→ -n : prints each line matching the pattern with line no.
→ -c : prints only count of lines matching "
→ -v : prints those lines that do not "

Specifying grep regular exp.

→ \c Backslash turns off any special meaning of character c
→ ^ pattern following it, must occur at beginning of line
→ $ pattern preceding it must occur at end of line
→ . matches a single character
→ [...] matches any one of char [ ]
→ [^...] matches any char not in [ ]
→ x|rz Concatenation of reg exp : x followed by z

SDS Page No
Date

pipes (1)

- Combine filters & commands such that standard op of one can be sent as standard ip to another.

— Commands —

SDS | Page No | Date

- command syntax.

Command	option(s)	Argument(s)
ls	-lh (-l -h)	/usr/bin
sort	-u	users.txt
grep	-i "needle"	file.txt

Command → It is the program you are running

Option(s) → options tell the program how to operate
→ start with dash
→ usually 1 letter
→ Commands can have more than 1 opt

Argument(s) → It tells the command what to operate on.

list of different Commands

① $ ls (list content of a directory)

ex home/desktop/Documents / $ ls
op list of directories in Documents

a) ls -l (long listing)
ex. home/desktop/Documents / $ ls -l
op. drwxr-xr-x 3 rosh 4096 Jun 11 10:56 hr
⋮

b) ls -a (do not ignore entries starting with .)
(ls -a or ls -all)

c) ls -A / ls -almost-all (do not list implied . and ..)

② file

- file myfile.txt (Determines file type)

- stat myfile.txt (Display ownership, modification info, etc)

③ cd (change directory)

ex • desktop/ cd Documents
• desktop/Documents/ cd Exercise files
(Backslash space means space (so no error))

• desktop/Documents/cd Exercise files
→ op: cd: too many arguments

• desktop/Documents/Exercise files/ ls
departments myfile.txt

• desktop/ / / ls -R departments (Recursive)
departments/hr
departments/hr/emp
department/sales
departments/sales/affiliates
(ls -R recursively lists our contents of a folder)

• desktop:/d / Exercise file/ cd departments
d./d../e f/dept/ cd..
(takes us to the parent directory of current directory)

d../d../e f

• d /d /e f/ cd dept/hr/emp

• d/d/e f/dept/hr/emp

• dept/hr/emp $ cd../../sales/affiliates
cd../ → parent dir of emp i.e hr
cd../../ → parent dir of hr i.e dept
cd../../ → dept/

• dept/sales/affiliates $

• dept/sales/affiliates $ cd -
(takes you back to prev used folder)
i.e
dept/hr/emp $

dept/sales/affiliates $ cd
(takes you back to home folder from wherever you are)

④ mkdir (make directory)

documents / exercise files / mkdir new-folder

- documents / exercise files / ls
  → department new folder
- documents / exercise files / mkdir department / new folder
  documents / exercise files / cd dept
  /dept $ ls
  → hr sales new-folder
- cd / cd exercise files / mkdir dept / mkdir new folder / f1
  dept / new folder / f2

⑤ rmdir (remove directory)

- rmdir can remove a dir only if it is empty.

✓ dept / new folder / rmdir f1

dept / new folder / rmdir f2

⑥ cp (copy)

→ desktop / doc $ ls
→ f1.txt f2.txt f3.txt
desktop / doc $ cp f1.txt f4.txt (copies content of f1.txt into f4.txt)
desktop / doc $ ls
f1.txt f2.txt f3.txt f4.txt

→ desktop / doc $ cp f2.txt exercise files / dept / hr / emp
(copies f2.txt into emp)
desktop / doc $ ls exercise files / dept / hr / emp
→ f2.txt

⑦ mv (moving & renaming)

→ desktop / doc $ mv f1.txt exercise files / dept / sales
(moves f1.txt from doc to sales.)

→ desktop / doc / cd exercise files
desktop / doc / exercise files / mv dept / hr / emp dept / hr / customers
(renames emp to customers.)

cd. ⇒ current folder

Q move all txt files from docs to dept/sales

../docs $ mv *.txt Ef /dept/sales

Q move everything from dept/sales to current folder.

../docs $ mv Ef/dept/sales/ * .
(* = current folder)

(8) rm (remove a file)

/docs $ rm f3.txt ← removes f3.txt from docs.

Q. remove all files named file_.txt from docs

/docs $ rm file?.txt

rmdir → deletes only empty dir
rm → deletes only files.

Q. remove a folder dept that is not empty.

→ ../docs $ rm -r Ef/dept

It recursively deletes all files/folders inside dept then deletes dept too

(1) find

$ find . -name "f*.txt"
(. = current folder)

find any file starting with f in current folder

User Roles

linux / Unix is a multi user env
Switch b/w users with 'su' command

User Roles
→ Normal User → modify their own files, cannot make system changes
→ Super User (root) → modify any file, make system changes.

/doc/exercise file $ ls /root
→ Permission denied

/doc/ef $ sudo ls /root
(↑ works)
anything written after sudo is executed using superuser privileges

/d/ef $ sudo -k
(↑ give up privilege)

File permissions

→ rwx rwx rwx file
User Group others

→ chmod → change permissions on a file

→ 2 methods to represent permissions

① Octal (ex 755, 644 & 777)

② Symbolic (ex a=r, g+w & o-x)

Octal →

	Read(4)	write(2)	Execute(1)	Result
User	r	w	x	7
Group	r	-	x	5
Others	r	-	-	4

→

octal value	mode
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

Symbolic

	Read(r)	write(w)	Execute(x)	Result
User(u)	+	+	+	u+rwx
Group(g)	=	-	-	g=r
others(o)	-	-	-	o-rwx
All(a)				

→ comparing octal & Symbolic values

octal value	Symbolic Value	Result
777	a+rwx	rwxrwxrwx
755	u+rwx, g+rx, o=rx	rwxr-xr-x
644	u+rw, g=r, o=r	rw-r--r--
700	u+rwx, g-rwx, o-rwx	rwx------

terminal

```
$ ls -l fi.txt
-rwxr-xr-x
$ chmod 644 fi.txt
$ chmod u+rw g=r o=r fi.txt

$ chmod u-r fi.txt
$ chmod 244 fi.txt
```

⇒ $ touch newfile (creates a file)
$ ls -l newfile
-rw-r--r-- .....
↑ default file permissions when you create a file

hard & Symbolic links

links → a file that acts as reference to another file

Types: Hard link → Points to data on disk (inode)

Soft link (or symlink: symbolic link) → Points to file on disk (relative path)

⇒ $ ln -s poems.txt writing.txt
creates symlink in poems.txt to writing.txt

⇒ $ ln poems.txt words.txt
hard link from poems.txt to words.txt

⇒ $ ln -s poems.txt doc/ef/writing.txt
relative path

⇒ words ✓ (-s)

(Every file on our system is a hardlink to underlying data)

(Even if file moves, hardlinks exist / do not get affected)

pipe (|)

pipe → used to connect multiple commands.
→ It gives the output of 1st command to the 2nd & so on.

⇒ $ echo "hello"
hello

⇒ $ echo "hello" | wc
1 1 6
↳ 1 line → 1 word → 6 characters

⇒ $ echo "hello world" | wc
1 6 10

Text tools

cat (Text tool)

cat → Concatenate text (used to output text files to screen or to another tool)

⇒ $ cat poems.txt
Johny Johny
Yes papa
.....
.....

cat > f1.txt → creates f1.txt & allows you to write in f1.txt until you press ctrl+d

cat f1.txt > f2.txt → puts content of f1.txt in f2.txt

cat f1.txt >> f2.txt → appends content of f1.txt & f2.txt

head

⇒ $ head poems.txt
displays first 10 lines by default

⇒ $ head -n5 poems.txt
displays first 5 lines

tail

⇒ $ tail poems.txt
displays last 10 lines by default

⇒ $ tail -n3 poems.txt
displays last 3 lines

cat, head/tail using pipe

⇒ $ cat poems.txt | cat -n
displays all the content of poems.txt with line nos.

⇒ $ cat poems.txt | cat -n | tail -n5

51 ...abcd
52 ...efgh
53 ...ijkl
54 ...mnop
55 ...qrst

⇒ cat poems.txt | tail -n5 | cat -n

1 abcd
2 ...efgh
3 ...ijkl
4 ...mnop
5 ...qrst

less

less → paginates text & provides navigation controls

$ less poems.txt
press (h → help q → quit)

Searching for text

grep

grep → This tool searches text or files for given string or pattern of text

⇒ $ grep "the" poems.txt
the lines having word "the" are displayed

⇒ $ grep -n "the" poems.txt.
the lines having word "the" are displayed along with line number

* grep is case sensitive *

$ grep "The" poems.txt
→ diff op than (grep "the" poems.txt)

* to make grep case insensitive use -i.

$ grep -i "the" poems.txt
→ lines having the / The / THE ... are displayed

$ grep -vi "the" poems.txt
→ displays all those lines that does not contain the because of -v

— xgex —

- grep supports regular exp too. using -E

$ grep -E "[hijk]" poems.txt
→ it displays all those lines having letters h / i / j / k. ([/ → or)

$ grep -E "\w{6,}" poems.txt
→ all those lines having words with 6 or more characters.

manipulating txt

1. awk

→ $ cat data.txt

name	ID	item
ABC	142	Apple
XYZ	293	Ball
EFG	642	car
HIJ	998	Dog
KLM	184	Elephant
NOT	916	Fish

→ $ awk '{print $2}' data.txt
→ prints / displays 2nd field in data.txt

ID
142
293
642
998
184
916

→ $ awk '{print $2 "\t" $1}' data.txt
→ prints field 2 & field 1 with a tab space b/w them

ID	name
142	AB
293	XYZ
:	:
NOT	916

→ $ awk '{ print $2 "\t" $1 }' data.txt | sort -n

↓ numeric sort

ID	name
142	ABC
184	klm
293	
642	
916	
998	

→ ② sed (helpful for changing the data)

$ sed "s/Apple/Anaconda" data.txt

→ changes/replaces every ocurrence of Apple to Anaconda

name	ID	team
ABC	142	Anaconda
XYZ	293	Ball
:	:	cat
		:
		fish

Sort

⇒ $ sort data.txt

→ sorts first field Alphabetically (Accto first char of 1st field)

ABC	142	Apple
EFG	642	cat
HIJ	998	Dog
klm	184	Elephant
name	ID	Team
NOT	916	Fish
XYZ	293	Ball

⇒ $ sort -k2 data.txt

→ sorts 2nd field (Acc to the first char of 2nd field)

ABC	142	Apple
klm	184	Elephant
XYZ	293	Ball
EFG	642	cat
Not	916	Fish
HIJ	998	Dog
name	ID	Team

⇒ $ sort -k2 -n data.txt

→ numeric sort

Name	ID	team
ABC	142	Apple
:	184	:
	293	:
	642	:
	916	
	998	

SDS | Page No | Date

— unique. (-u) —

$ cat dupes.txt
Hello baby
nice to meet you
hello baby
great day It is !
right ?
right ?
$ cat -u dupes.txt
→ hello baby
nice to meet you
great day it is !
right ?

other commands

1. rev → prints text in reverse

$ rev file.txt
elppa
llab
tac

$ echo "one two three" | rev
three two one.