What is Unix?

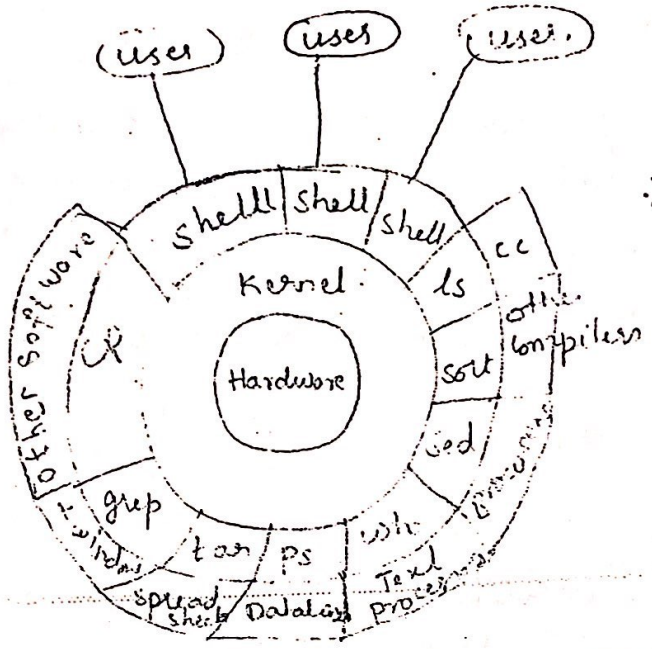
1) Unix is a Computer O.S.
2) Unix is a multi-user, multi-tasking O.S. networked system.
multiple users may have multiple tasks running simultaneously :: stability, portability & powerful N/w capabilities
3). Unix is a m/c independent O.S.
4). Unix is a s/w development environment.

Why Unix?

1) H/w independence.
   i) The O.S code is written in C lang rather than a specific assembly lang.
   ii) The O.S s/w can be easily moved from one h/w system to another.
   iii) Unix appln can be easily moved to other Unix machines.
2) Productive environment for s/w development
   i) rich set of tools
   ii) Versatile command lang.

Unix is a multiuser, multiprocessing, portable O.S. It is designed to facilitate prgmg, text processing & Comm.

# Unix Architecture






The kernel-shell relationship

Unix architecture is divided into two components.

i) Kernel ii) Shell.

Kernel → Interacts with the m/c's h/w

Shell → " " " User.

Unix is a layered O.S. The innermost layer is the h/w that provides the services for the O.S. The O.S, referred to in Unix as kernel, interacts directly with the h/w & provides the services to the user pgms.

These user pgms don't need to know anything about the h/w. They just need to know how to interact with the kernel & its up to the kernel to provide the desired service.

User pgms interact with the kernel through a set of standard system calls. These system calls request services to be provided by the kernel. Such services would include accessing a file : open close read ...

link or execute a file.

Unix is a multi-user, multi-tasking O.S. You can have many users logged into a system simultaneously, each running many pgms. It's the Kernel's job to keep each process & user separate & to regulate access to system h/w including CPU, memory, disk & other I/o device

Kernel does the following jobs
manages the
i) System's memory.
ii) schedules processes.
iii) Decides their priorities.

Shell.

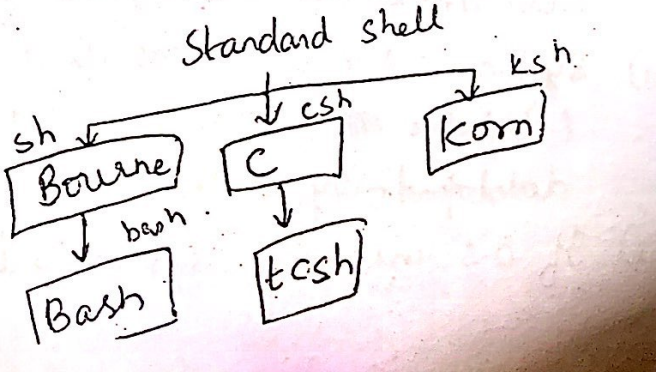
1) Whenever you login to a Unix system you are placed in a shell pgm. The shell's prompt is usually visible at the cursor's position on your screen. To get your work done, you enter Commands at this prompt.

2) The shell is a Command interpreter, it takes each Command & passes it to the O.S Kernel to be acted upon. It then displays the result of this operation on your screen.

3) Several shells are usually available on any Unix system, each with its own strengths & weaknesses.

4) Different users may use different shells. The most Commonly available shells are.

i) Bourne shell (sh)
ii) C shell (csh)
iii) Korn shell (ksh)
iv) Tc shell (Tcsh)
v) [illegible] shell (bash)



## The file & Process:

Two simple entities support the Unix system.

i) File ii) Process.

Files have places & processes have life".

A file is just an array of bytes & can contain virtually anything.

Unix doesn't really care to know the type of file you are using. It considers even directories & devices as members of the file system, dominant file type is text.

Process is the name given to a file when it is executed as a pgm. It is the time image of an executable file.

name of .c pgm → file

execution of .c pgm → process

$ cat → file

$ cat file.c & run → process

## System Calls:

1) The Unix system comprising the kernel, shell & appln - is written in C.
2) Though there are thousand of commands in the system, they all use a handful functions, called system calls, to communicate with the kernel.
3) All Unix use the same system calls.
4) System calls are described in the POSIX specification (Portable O.S Interface for Computer Environments) developed by IEEE
5) If O.S uses diff system calls, then it won't be Unix

system call like write, open etc.

6) Same system call can accen both the file & a device.

7) System calls are built into the kernel.

Features of Unix :

Unix is an O.S, so it has all the features an O.S is expected to have.

i) Unix : A Multiuser system.

Unix is a multiprg system. It permits multiple program to run & compete for the attention of the CPU.

- Multiple Users can run seperate jobs.
- Single user can also run multiple jobs.

ii) Unix : A Multitasking System

1) A single user can also run multiple tasks concurrently. It is usual for a user to edit a file, print another one on the printer, send mail to friend etc.

2) In the multitasking environment, a user sees one job running in the foreground, the rest run in the background. Switch jobs b/w fore & back ground.

iii) The Building Block approach.

2 Commands can be used together ls | wc
↓ pipe.

Commands connected in this way is called filters. o/p of one Command is given as I/p to next Command.

4) Unix toolkit

1) In Unix there are general purpose tools, text manipulation Utilities (filters), Compilers & Interpreters networked appln & system administration tools.
2) In Unix new tools are added & the older ones are being removed & modified.

5). Pattern matching

1) Unix has pattern matching features.

ls Command is used to list

chap 1, chap 2, chap(*) → metacharacter.

) Programming facility

) Documentation

There is a online help facility available. i'e is the man Command.

2) Unix resources are available on the Internet.

1) Cal: The Calendar.

This cal command is used to see the calendar of any specific month or a complete year.

$ cal

$ cal 03 2007.

2) date: displaying the system date.

The unix system maintains an internal clock meant to run. When the system is shut down, a battery backup keeps the clock ticking.

$ date.

o/p

The command can also be used with suitable format specifiers as arguments. It is preceded by the + symbol followed by the % operator.

$ date +%m

o/p 8

$ date +%h → month name in short

Aug.

$ date +"%h %m"

o/p Aug 08.

$ date "Today's date is +%D"

There are many format specifiers & the useful one are listed below.

d - The day of the month.

Y - The last 2 digits of the year.

H, M & S - The hour, minute & second respectively.

D - The date in the format mm/dd/yy.

T - the Time in the format hh:mm:ss.

). Echo: displaying a message.

It is used to display a message.

An escape sequence is generally a two character string beginning with a \ (backslash).

Ex: \c → It is used to place the cursor & prompt in the same line that displays the o/p.

Ex: $echo "enter filename : \c"

o/p: Enter filename : $-

\t → A tab which pushes text to the right by eight character positions

\n → A newline which creates the effect of pressing [enter].

) Printf: An alternative to echo.

Printf command is available on most modern Unix systems. It is the one you should use instead of echo.

:) $ printf "No filename entered \n"
No filename entered.
$-

:) $ printf "My current shell is %s\n" $SHELL
My current shell is /usr/bin/bash.

%s → format string acts as a placeholder for the value of $SHELL. printf replaces %s with the value of $SHELL. It is the standard format for printing strings.

%s → string.
%30s → printed in a space of 30 characters wide.
%d → Decimal integer.
%6d → printed in a space of 6 characters wide.
%o → octal integer.
%x → Hexadecimal integer.
%f → Floating point no.

ex: printf "The Value of 255 is %o in octal and %x in hexadecimal\n" 255, 255 → in hexadecimal (base 16).
↳ in base 8/in octal.

5). bc: The calculator.

Unix provides two types of calculators
i) graphical object (uses xcalc Command)
ii) Text-based (bc Command).

$ bc

cursor keeps on blinking & nothing seems to happen.

1) $ bc
12+5
17
press ctrl-d
$ -

2) $ bc
12*12 ; 2^32 ^ → power of
o/p 144
4294967296

bc performs only integer computation & truncates the decimal portion.

9/5
= 1. (truncated).

To enable floating point computation, you have to set scale

scale = 2. (truncates to 2 decimal places).

17/7 :
= 2.42.

6) who: Who are the users?

Unix maintains an account of all users who are logged on to the system. The who Command displays an information listing of these users.

$ who
root console Aug 1 07:51 (:0)
sharma pts/10 Aug 1 07:56 (pc123.heavens.com)
sharma pts/6 Aug 1 02:10 (pc125.heavens.com)
↓ ↓ ↓
date & time

↳ Header option.

| NAME | LINE | TIME | IDLE | PID | COMMENTS |
|---|---|---|---|---|---|
| root | Console | Aug 1 07:51 | 0:48 | 11040 | (:0) |
| Kumar | pts/10 | Aug 1 07:56 | 0:33 | 11200 | (PC123.heavens.com) |
| Sachin | pts/14 | Aug 1 08:36 | . | 13678 | (mercury.") |

↓ identifies Process

Activity has occurred in the last one min

idle for 33 min

$ who am i

Kumar pts/10 Aug 1 07:56 (PC123:heavens.com)

6) Cat → takes I/P from the keyboard & sends O/P to the Monitor

$ cat > file1

Type the Content here.

Press ctrl d

$ cat file1

Displays.

7) $ x=5

$ echo $x.

5

8) cmp: Comparing 2 files.

$ cmp chap01 chap02

chap01 chap02 differ:

Char 9, line 1

i) CP: Copying a file

cp command copies a file or a group of files.

$ cp chap01 until

2) rm: Deleting files

$ rm chap01 chap02 chap03

rm won't remove a directory, but it can remove only files.

$ rm * (all files gone).

3) mv: Renaming files.

it renames a file (or directory)

$ mv chap01 nano1

1) tput clear.: clearing the screen.

$tput clear.

2) PS: Viewing process.

$ps → process.

o/p

| PID | TTY | TIME | CMD |
|---|---|---|---|
| 364 | console | 0:00 | ksh. |

when you logged out your process is killed.

3) ls: listing files.

$ ls

$ ls chap*

4) Directing o/p to a file.

$ ls > list.

$ cat list.

chap01
chap02

5) wc: counting no: of lines in a file.

$ wc list.

6 6 42 list.

-l → no: of lines.
-w → no: of words
-c → no: of characters.

LOCATING COMMANDS.

) The Unix system is Command based.
) Unix Commands are seldom more than 4 characters
Unix Commands are single Commands like ls, cat, who.
These are all in lowercase.

ex: $ LS
bash: LS: Command not found.

) This message is from the shell. There's obviously no Command LS on the Unix system.
) It suggests that there's a predetermined list of such Commands that the shell first searches before it gives message.
) These Commands are essentially files Containing pgms mainly written in C.
) Files are stored in directories.
) ls → Command → is a file (or pgm) found in the directory /bin.
) The easiest way of knowing the location of an executable pgm is to use the type Command.

$ type ls
ls is /bin/ls

PATH

1) The sequence of directories that the shell searches to look for a Command is specified in its own PATH Variable.
2) Use Echo to evaluate this Variable & you'll see a directory list separated by Colons:

$ echo $ PATH.
/bin: /usr/bin: /usr/local/bin: /usr/ccs/bin: /usr/local/java
/bin.

3) There are Six directories in this Colon-separated list. to Consider the 2nd one, /usr/bin represents a hierarchy of 3 directory names

/ → root directory (top most)
↓
usr
↓
bin.

4) When you issue a Command, the shell searches this list in the sequence specified to locate & execute it.

$ netscape.
ksh: netscape: not found.

This doesn't in any way Confirm that netscape doesn't exist on this system, it could reside in a different directory.

$ /usr/local/bin/netscape.

INTERNAL & EXTERNAL COMMANDS.

The Commands that are present in the directories are referred to us as External Command. ls is a pgm or file having an independent existence in the /bin directory it is branded as an External Command.

ex: ls, cat, who.

The Commands that are not present in the PATH directories are referred to us as Internal Commands.

ex: $ type echo. id, source, fg.

When you type echo, the shell won't look in its PATH to locate it. it will execute it from its own set of built-in Commands that are not stored as separate files.

Shell → External Command → possesses its own

## Command Structure.

) Commands & arguments have to be separated by spaces or tabs to enable the system to interpret them as words.

spaces + tabs → whitespace.

Cat Readme.

) Unix arguments consist of options, expressions, Instrn, filenames etc.

## options

.) There's a special type of argument that's mostly used with a – sign.

Ex. $ ls -l note.

). ls -z note

ls: illegal option -- z.

The above mess has been generated by the command & not by the shell.

ii) $ ls-l

bash: ls-l: Command not found (why).

iii) ls -l -a -t or ls -lat.

## Filename Arguments.

. Many Unix Commands use a filename as argument so the command can take I/p from the file.

$ ls -lat chap01 chap02 chap03

cp chap01 chap02 progs → directory.

mv chap01 chap02.

) Command with arguments & options is known as Command line. It is complete only after the User has hit [enter]

3) The [illegible] is then fed to the shell as its I/p for interpretation & execution

Combining Commands.

1) $: wc note ; ls -l note.

2) Redirect the o/p of these commands, you may even like to group them together within parentheses

(wc note; ls -l note) > newlist.

man: Browsing the manual pages on line

1) Unix offers an online help facility in the man command.

2) Man displays the documentation - often called the man documentation

3) Ex:. To seek help on the wc command

$ man wc

4) The entire man page for wc is dumped on the screen.

5) Man presents the first page & pauses. It does this by sending its o/p to a pager pgm, which displays this o/p one page at a time.

6) Unix systems currently use these pager pgms

- more
- less → std pgr on linux systems

--more--(26%)

7) To quit the pager, press q. It returns to shell's prompt

Navigation & search.

2) 2 commands which should work on all systems.

- f or spacebar, which advances the display by one screen of text at a time.
- b, moves back one screen.

ex: you can call up the page containing the word clobber by using the string with the / (frontslash)

/clobber [enter].

Understanding a man page.

1) A man page is % into a no: of compulsory & optional sections.
2) Every command doesn't have all sections, but the first 3 [NAME, SYNOPSIS & DESCRIPTION) are generally seen.
3) Name - one-line Introduction to the command.

Synopsis :- syntax used by the command.

Description : detailed explanation.

Synopsis section is one that we need to examine closely. manpage to wc command.

wc [-c | -m | -C] [-lw] [file ...]

1) Command argument enclosed in [ ] → it is optional otherwise argument is req. The wc man page shows 3 [.]. This means that wc can be used without argu.
2) The ellipsis (a set of 3 dots) implies that there can be more instances of the preceding word. The exp [file ...] signifies that wc. can be used with more than 1 filename as argument.

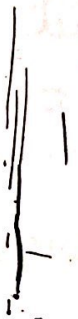
- If you find the | characters in any of these areas, it means that only one of the options shown on either side of the pipe can be used. Here, only one of the options -c, -m & -C can be used.

## options in Man (-k)

$ man -k → man searches a summary database & prints a one-line descriptions of the command. ~~For example~~

$ man -k awk.

awk - pattern scanning & processing lang.



# The file system

## FILE

- The file is a container for storing information. it is simply as a sequence of characters.

ex: file foo & write three characters a, b, c
foo contains only string abc & nothing else.

i) Unix file doesn't contain the eof (end-of-file) mark.

ii) Unix treats directories & devices as files.

Directory → folder to store filenames & other directories

Devices → physical devices like hard disk, memory, CD-Rom Printer & modem are treated as files.
Shell & even memory is also a file.

iii) Files are divided into three categories.

FILE

- ORDINARY
- DIRECTORY
- DEVICE
- FIFO

i) ORDINARY FILE OR REGULAR FILE

All programs you write belong to this type.

ORDINARY FILE

- Text file
  - i) Contains only printable characters.
  - ii) All C, Java, shell & perl Scripts are text files.
- Binary file.
  - i) Contain both printable & unprintable characters that cover the entire ASCII range (0 to 255)
  - ii) Most Unix Commands are binary files.
  - iii) object code & executables that you produce by Compiling C pgms are B.F.
  - iv) Picture Sound & Video files

2) DIRECTORY FILE

1) A directory contains no data, but keeps some details of the files & Subdirectories

2) File system organized → no: of directories
                         → Sub-directories.

3) Two or more files in seperate directories can have the same filename.

4). Directory file contains entry for every file & subdirectory.

20 files → directory → 20 entries.

Each entry

↓ Filename

↓ Unique identification no. (inode no)

5) A directory contains the filename & not the file's contents.

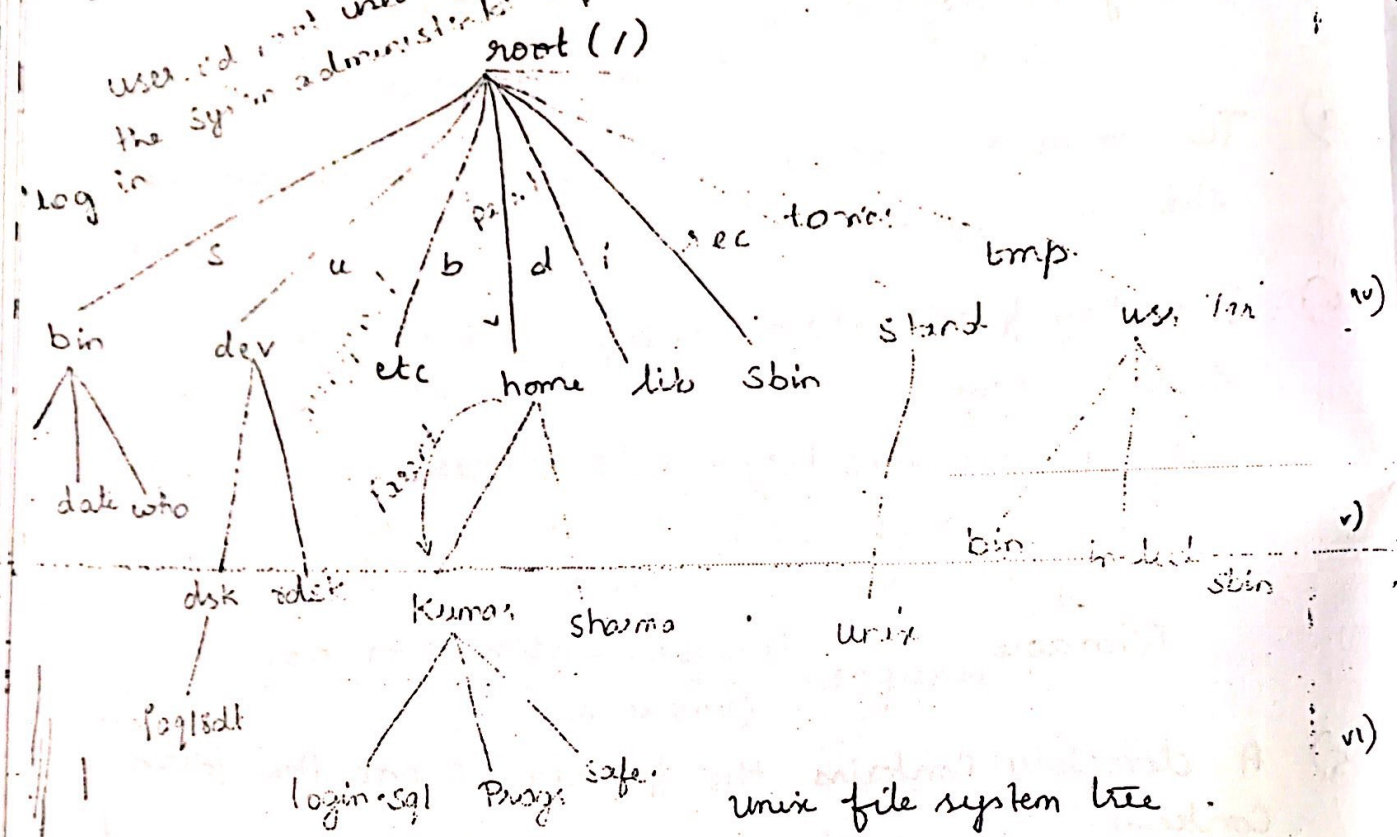
3) DEVICE FILE (refer last page)

1) We will also be printing files, installing s/w from CD-ROM. or backing up files to tape. All of these activities are performed by writing or reading the file representing the device.

Ex: When you restore files from tape, you read the file associated with the tape drive.

2) Device filenames are generally found inside a single directory structure /dev.

The file system in UNIX is a collection of all of these related files (ordinary, directory & device files) organized in a hierarchical structure.




Unix file system tree.

Root is actually a directory.

In these parent-child relationships, the parent is always a directory.

home & Kumar are both directories as they are both parents of at least one file or directory.

login.sql is simply an ordinary file & it can't have any directory under it.

) /bin & /usr/bin :- These are the directories where all the commonly used Unix commands are found. The PATH variable always shows these directories in its list. most of the command files are binary files, the directory is called bin.

) /sbin & /usr/sbin :- If there's a command that you can't execute but the system administrator can, then it would probably be in one of these directories. ) directories which contain

you won't be able to execute most commands in these directories. Only the system administrator's PATH shows these directories.

iii) /etc : (This directory contains the configuration files of the system) You can change a very important aspect of system functioning by editing a text file in this directory. (your login name & password are stored in files /etc/passwd & /etc/shadow)

iv) /dev :- This directory contains all device files. These files don't occupy space on disk. There could be more subdirectories like pts, dsk & rdsk in this directory.

v) /lib & /usr/lib :- Contain all library files in 'binary' form you'll need to link your C pgms with files in these directories

vi) /usr/include :- Contains the std header files used by C pgms. The statement #include<stdio.h> used in most C pgms refers to the file stdio.h in this directory.

vii) /usr/share/man - These is where the man pages are stored.

viii) /tmp : The directories where users are allowed to create temporary files. These files are wiped away regularly by the system.

ix) /Var - The Variable part of the file system. Contains all your print jobs & your outgoing & incoming mail.

x) /home - On many systems users are housed here. Kumar would have his home directory in /home/kumar. Your system may use a different location for

## The Home Variable : The Home Directory.

1) When you log on to the system, Unix automatically places you in a directory called the home directory.
2) It is created by the system when a user account is opened.
   If you login using the login name kumar.

   pathname /home/kumar.

3) The shell variable HOME knows your home directory.

   $ echo $HOME

   /home/kumar. → absolute Pathname.

   sequence of directory names separated by /'s.

4) Absolute pathname shows a file's location with reference to the top ie root.
5) Foo file located in your home directory can be given as $HOME/foo.
6) Most shells (except Bourne) also uses the ~ symbols for this purpose.

   $HOME/foo ⟺ ~/foo.

   Shows any user's home directory.

user sharma has the file foo in his home directory then kumar can access it as ~sharma/foo.

" A tilde followed by / (~/) refers to one's own home directory, but when followed by a string (~sharma) refers to the home directory of that user represented by the string"

pwd: checking your current directory

you can move around from one directory to another but at any point of time, you are located in only one directory. This directory is known as your current directory.

$ pwd (print working directory)

/home/kumar.

cd: changing the current directory

$ pwd

/home/kumar

$ cd progs

$ pwd

/home/kumar/progs.

cd /home/kumar/progs → same.

when you need to switch to the /bin directory where most of the commonly used unix commands are kept,

$ pwd

/home/kumar/progs.

$ cd /bin

$ pwd

/bin.

cd can also be used without any arguments.

$ pwd

/home/kumar/progs.

$ cd

$ pwd

/home/kumar.

## mkdir : Making Directories

1) Directories are created with the mkdir Command.

$ mkdir patch.

$ mkdir patch dbs doc  Three directories

2) The following Commands creates a directory tree.

$ mkdir pis pis/progs pis/data.

$ mkdir pis/data pis/progs pis

mkdir: failed to make directory "Pis/data": No such file

$ mkdir test

mkdir: failed to make directory "test": Permission denied

This can happen due to the reasons:

1) The directory test may already exist
2) There may be an ordinary file by that name in the current directory.
3) The permissions set for the current directory don't permit the creation of files & directories by the User.

## rmdir : Removing directories

1) The rmdir Command removes directories.

$ rmdir pis  Directory must be empty
↳ no files

2) rmdir pis/data pis/progs pis

Note: When you delete a directory & its subdirectories a reverse logic has to be applied.

3) rmdir pis pis/progs pis/data

rmdir: directory "pis": Directory not empty.

This error message leads to two important rules that you should remember when deleting directories

- you can't delete a directory with rmdir unless it is empty. pis directory couldn't be removed because of the existence of the subdirectories, progs & data under

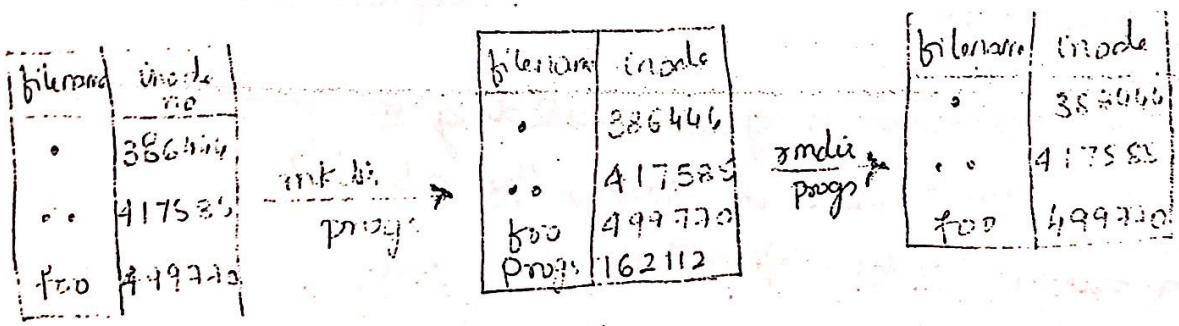
Ex: $cd progs
$ pwd
/home/kumar/pis/progs.

$rmdir /home/kumar/pis/progs
rmdir: directory 'home/kumar/pis/progs': Directory does not exist

To remove this directory, you must position yourself in the directory above progs, ie pis & then remove it from there.

$ cd /home/kumar/pis
$ pwd
/home/kumar/pis
$ rmdir progs

How files & Directories are created & Removed.



Absolute pathnames:

) Many Unix commands use file & directory names as arguments.

$ cat login.sql. work only if the file login.sql exists in your current directory.

/usr and want to access login.sql in /home/kumar.

$ cat /home/kumar/login.sql.

→ absolute pathname.

As stated before, if the first character of a pathname s /, the file's location must be determined with respect = root. Such a pathname is called absolute pathname

No two files in a Unix system can have identical absolute pathnames. You can have 2 files with the same name, but in different directories : their pathnames will also be different. Thus, the file

/home/kumar/progs/caf.pl can co-exist with the le /home/kumar/safe/caf.pl.

Using the absolute pathnames for a Command.

, If you know where the date Command exist you can also use this absolute pathnames

$/bin/date

Then sep.

If you execute programs residing in some other directory that isn't in path, the absolute pathname then needs to be specified.

## Relative pathname.

A relative pathname is the path from the working directory to the file. The pathname relative to the directory. A relative pathname should not be started with a slash.

ex: cd progs.
cat login.sql. } we are assuming both are in current directory

If progs also contains a directory scripts under it, you still won't need a absolute pathname to change to that directory.

cd progs/scripts

Here we have a pathname that has a /, but it is not an absolute pathname because it doesn't begin with a /

## Using . & .. in Relative pathnames.

In the preceding example, you changed your director from /home/kumar/pis/progs to its parent directory (/home/kumar/pis) by using cd with an absolute P.N.

cd /home/kumar/pis.

Unix offers a shortcut - the relative pathname - that uses either the current or parent directory as reference & specifies the path relative to it. A relative pathname uses one of these cryptic symbols.

- . (a single dot) - This represents the current directory.
- .. (two dots) - This represents the parent directory.

Assume you are placed in /home/kumar/progs/data/text

```
$ pwd
/home/kumar/progs/data/text.
$ cd ..
$ pwd.
/home/kumar/progs/data.
```



cd .. translates ~~to~~ to this.

"Change your directory to the parent of the current directory".

Ex: $ pwd

/home/kumar/pis.

$ cd ../..

$ pwd → /home

Any command which uses the current directory as argument can also work with a single dot (.)

cp ../sharma/.profile . (A filename can begin with a dot)

This copies the file .profile to the current directory (.).

## ls: listing directory contents.

$ ls

### ls options.

ls has a large number of options.

i) O/P in multiple columns (-x)

$ ls -x.

08-packets.html  Toc.sh  "  "
dept.lst  "  "  "
usdsk06x  "  "  "



ii) Identifying Directories & Executables (-F)

The output ls that you have seen so far showed the filenames. To identify directories & executable files, the -F options should be used. Combining -Fip option with -x produces a multicolumnar o/p.

$ ls -Fx.

| | | | |
|---|---|---|---|
| 08-packets.html | Toc.sh* | Alendar* | cptodos.sh* |
| dept.lst | emp.lst | helpdir/ | progs/ |
| usdsk06x. | usdsk07x | usrdsk08x | ux2nd06, |

* & / are used as type indicators.

* → file contains executable code.

/ → refers to a directory.

Showing hidden files also (-a)

ls doesn't show all files in a directory. There are certain hidden files often found in the home directory.

$ ls -axF.

./  ../  .exrc  .kshrc
.sh_history  .xdtsupcheck

Recursive listing (-R).

The -R (recursive) option lists all files & subdirectories in a directory tree.

```
$ ls -xR
08-packets.html   Toc.sh      calendar    uptodos.sh
dept.lst          emp.lst     helpdir     progs
usdsk06x          usdsk07x    usdsk08x    ux2ndo6

./helpdir:
forms.hlp   graphics.hlp   report.hlp

./progs
array.pl   cent2fah.pl   n2words.pl   name.pl
```

The list shows the filenames in 3 sections.
1) One under the home directory & those under the subdirectories helpdir & progs.

options

| | |
|---|---|
| -x | multicolumnar o/p |
| -F | marks executables with *, directories with / |
| -a | shows all filenames beginning with a dot including . & .. |
| -R | recursive list. |
| -r | sorts filenames in reverse order. |
| -l | long listing in ASCII collating seq showing seven attributes of a file. |
| -d dirname | lists only dirname if dirname is a dir |
| -t | Sorts filenames by last modification time |
| -lt | Sorts listing by last " " |
| -u | Sorts filenames by last access time |
| -lu | Sorts by Ascii collating seq but listing shows last access time |

-lut  Sorted by last accentine.

-i  Displays inode no.

## Basic File Attributes.

### ls -l (long)

This option displays most attributes of a file - like its permissions, size & ownership details.

```
$ ls -l
total 72
-rw-r--r--  1 kumar metal 19514 May 10 13:45 chapc
-rw-r--r--  1 kumar metal  4174 May 10 15:01 chp
-rw-rw-rw-  1  ..     ..    94 Feb 12 12:30 dept
-rw-r--r--  1  "      "   9156 Mar 12 1999 genie
drwxr-xr-x  2  "      "    512 May 9 10:31 help
drwxr-xr-x  2  "      "
```

Which of these commands will work? Explain with reasons.

i) mkdir a/b/c

ii) mkdir a a/b

iii) rmdir a/b/c.

iv) rmdir a a/b

v) mkdir /bin/foo

## tty Comand (teletype)

1) The tty utility is used to show the name of the terminal you are using.

2) Unix treats ~~even unix~~ terminals as a file.

$ tty → pts directory
/dev/pts/10 → filename
↳ device directory.

Solaris m/c.

dev/tty01.

## 2) Uname:

1) Uname command displays certain features of the operating system running on your machine.

ex: i) $ uname
Linux.

ii) $ uname -r
2.6.23.1-42.fc8

iii) $ uname -n
~~localhost localhost~~.
localhost.localdomain.

## 3) Passwd:

changing your password
changing password for user user1
changing password for user1
(current) unix password.

Script :

Script is used to take a copy of everything that is output to the terminal & place it in a log file. It should be followed by name of the file to place the login, & the exit Command should be used to stop logging & close the file.

$ script
    Script started, file is typescript.

$ exit
    Script done, file is typescript.

Script

1) A script is typically a screen capture of Commands being executed in Unix operating environment.

$ script
    Script Command is started. File is typescript

$ exit
    script done, file is typescript

$ a2ps typescript (to print the Script file).
$ more typescript (to view the Contents of the script file on the Screen)

1) cp file1 file2 — Copy file1 to a file called file2.

2) cp file1 /tmp — Copy file1 to the /tmp directory

3) cp file1 ~smith — Copy file1 to the home directory of Smith

4) cp */tmp — Copy everything in the directory to the /tmp directory.

## mv Command

1) It is like to the cp Command, except it copies the file & deletes the original.

2) The mv Command is what you would use to rename a file

# Unix Commands



## 1) cp: Copying a file

The cp command copies a file or group of files.

$ cp chap01 Unit1

$ cp chap01 progs/unit1 — chap01 copied to unit1 under prog

$ cp chap01 progs — chap01 retains its name under progs.

$ cp chap01 chap02 chap03 progs

Copy the files chap01, 02, 03 to the progs directory.

$ cp chap* progs.



### cp options

#### 1) Interactive Copying (-i)

The -i option warns the user before overwriting the destination file.

$ cp -i chap01 Unit1
cp: overwrite 'Unit1'?

#### 2) Copying Directory Structures (-R)

$ cp -R progs newprogs

This command copies an entire directory structure progs to newprogs.

2) rm: Deleting files.

i) The rm(remove) Command deletes one or more files.

$ rm chap01 chap02 chap03

$ rm progs/chap01 progs/chap02

$ rm * (all files deleted).

rm options

i) rm -i chap01 chap02 chap03

rm: remove "chap01" ?

" " "chap02" ?

" " "chap03" ?

ii) recursive deletion. (-r or -R)

$ rm -r * all files in the Current directory & all its subdirectories.

3) mv: Renaming files

i) The mv Command renames (moves) files. It has 2 functions

ii) It renames a file (or directory)

iii) It moves a group of files to a different directory.

$ mv chap01 man01.

$ mv chap01 chap02 chap03 progs

The following Command moves three files to the progs directory.

mv can also be used to rename a directory

```
$mv pis perdir
   dir  dir
```

## more: Paging output

```
$ more chap01
```

Contents of chap01 on the screen will be displayed one page at a time.

--More--(17%)

Internal commands for more & less

| more | less | Action |
|---|---|---|
| i) Spacebar or f | spacebar or for z | one page forward |
| ii) 20f | — | 20 pages forward |
| iii) b | b | one page back |
| | etc. | |

## lp: Printing a file

1) No user is allowed direct access to the printer.
2) One has to line up a job along with others in a print queue.
3) $ lp filename (default printer should be set)
4) $ lp -dlaser chap01.ps. (more than one printer)
   ↓ ptrname

-t(option) followed by the title string, f
the first page.

$lp -t "Firstchapter" chapo1.ps.

-n (multiple copies)

$ lp -n3 chapo1.ps.

$ lpstat (print queue)

$ cancel laser (Cancels current job on printer laser)

$ cancel prl-320 (Cancels jobs with request-id prl-320)

## wc : Counting lines, words & characters

$ cat infile
Iam the wc_Command
I Count characters, words and lines
With options I can also make a selective Count

$ wc infile
3 20 103 infile
↳ lines ↳ Words ↳ characters ↳ filename.

$ wc -l infile (Number of lines)
$ wc -w infile (Number of words)
$ wc -c infile (Number of characters)

$wc chapo1 chapo2 chapo3

## Knowing the file types

Chandra's Date: / / Pg:

Unix provides the file command to determine the type the file.

```
$ file archive.zip
archive.zip  ZIP archive
$ file *        (all files will be displayed)
User-guide.ps    Postscript document.
Createuser.sh    Commands text.
fork1.c          C pgm text.
oslect.pdf       Adobe Portable document.
```

## Displaying data in octal

The od command makes the invisible non printing characters visible by displaying the ascii octal value of its input.

```
$ od -b odfile
0000000  127 150 151 164 145 040 163 160 141 145 040 ---
$ od -bc odfile
0000000  127 150 151 164 145 040 163 160 ---> octal numbers
           w   h   i   t   e       s   p ---> printable characters
```

W → 127

| | | | |
|---|---|---|---|
| [ctrl_i] | tab character | It | octal Value 011 |
| [ctrl_g] | bell | \a | " 007 |
| [ctrl_l] | formfeed | \f | " 014 |
| [ctrl_j] | linefeed | \n | " 012 |

## cmp: Comparing two files

there are 5 unix commands which can be used to compare the contents of 2 files Compare (cmp), difference (diff) common (comm)

```
$ cmp chap01 chap02
chap01 chap02 differ: char 9, line 1
```

Two files are compared byte by byte, & the location of the 1st mismatch (9th character of the first line) is echoed on the screen.

If two files are identical, cmp displays no message

```
$ cmp chap01 chap01
$
```

Ex:
```
$ cat cmpfile1
123456
7890

$ cat cmpfile1.cpy
123456
7890

$ cmp cmpfile1 cmpfile1.cpy

$ cat cmpfile2
123456
as9u

$ cmp cmpfile1 cmpfile2
cmpfile1 cmpfile2 differ: char 8, line 2
```

## cmp with list option (-l)

The list option displays all the differences found in the files, byte by byte.

Ex:
```
$ cmp -l cmpfile1 cmpfile2
  8   67  141
  9   70  163
```

1st 2nd 3rd 4th 5th 6th 7th
123456 newline
7890
8th 9th 10th 11

only differences are in byte 8 (7 v/s a), byte 9 (8 v/s s) 11/0 v/s u). Each difference is displayed separate line followed by the octal value of the characters. The character from the 1st file is listed Looking at the 1st line of o/p, the 8 indicates the th character. newline is a character.

∴ Seventh character is the newline at the end of the st line & the 8th character is the 1st character on the cond line. The 67 is the octal value for the digit 7, 141 is the octal value for the character a.

## with supress list option (-s)

The Supress list option is similar to the default cept that no o/p is displayed. When no o/p is displayed, the results can be determined by testing e exit status.

exit status; is 0, the two files are identical.
" " 1, atleast one byte is different.

ex: $cmp cmpfile1 cmpfile1.cpy
$ echo $?
0
$cmp cmpfile1 cmpfile2
$ echo $?
1

# Compressing & Decompressing files : gzip & gunzip.

```
$ wc -c libc.html.
3875302   libc.html.
$ gzip libc.html.
$ wc -c libc.html.gz
788096  libc.html.gz
```

## postscript file

```
$ wc -c user_guide.ps;
$ gzip user_guide.ps;
$ wc -c User-guide.ps.gz.
372267  User-guide.ps
128341  User-guide.ps.gz
```

Before Compression
After —"—

```
$ gzip -l libc.html.gz  User.guide.ps.gz.
Compressed  uncompressed  ratio  Uncompressed name
788096      3875302       79.6%  libc.html
128341      372267        65.5%  User_guide.ps.
```

## Uncompressing a gzipped file (-d)

1) To restore the original & uncompressed file, you have two options.

   i) gzip -d or gunzip.

   i) gunzip libc.html.gz
   ii) gzip -d libc.html.
   iii) gunzip libc.html.gz

Retrieves libc.html.

User-guide.ps.gz.

## Recursive Compression (-r)

It descends a directory structure and compress all files found in subdirectories.

gzip -r progs — Compresses all files in progs

gunzip -r progs
or
gzip -dr progs.

} decompress all files in this directory

## tar: Archival program

For creating a disk archive that contains a group of files or an entire directory structure, we need to use tar.

- -c Create an archive.
- -x extract files from archive.
- -t Display files in archive
- -f arch Specify the archive arch.

## Creating tar Archive (-c)

To create an archive, we need to specify the name of the archive (-f), the copy or write operations (-v) and the filenames as arguments.

$ tar -cvf archive.tar libc.html User-guide.ps

a libc.html 3785k.
a User-guide.ps 364k. (a indicates append)

tar -cvf progs.tar C-progs java-progs shell-scripts

## Using gzip with tar

If the created archive is very big, you may like to compress it with gzip.

gzip archive.tar

## Extracting files from Archive (-x)

tar uses the -x option to extract files from an archive.

tar -xvf progs.tar. Extracts 3 directories

To extract files first gunzip to decompress the archive & then run tar.

$ gunzip archive.tar.gz. Retrieves archive.tar

$ tar -xvf archive.tar. Extracts files.

x libc.html, 3875302 bytes, 7569 tape blocks.
x User-guide.ps, 372267 bytes, 728 tape blocks.

$ tar -xvf archive.tar User-guide.ps (Extracts only user-guide.ps

## Viewing the archive (-t)

To view the contents of the archive, use the -t option.

```
$ tar -tvf archive.tar.
```

## Zip & Unzip : Compressing & Archiving together

gzip & tar using these two commands to compress a directory structure. you can use only one zip.

1) zip requires the first argument to be compressed filename, the remaining arguments are interpreted as files & directories to be compressed.

```
$ zip archive.zip libc.html user_guide.ps.
```

comp filename  files & directories

archive.zip exists, filenames can be either be updated or appended to the archive depending on whether they already exist in the archive.

## Recursive Compression (-r)

zip uses -r option, It descends the tree structure in the same way tar does except that it also compresses files.

```
cd; zip -r Sumit_home.zip.
```

## Using Unzip

Files are restored with the unzip command, which in its simplest form, uses the compressed filename as argument.

$ unzip archive.zip
Archive: archive.zip

Date: / / Pg:

## Viewing the archive (-v)

You can view the Compressed archive with the -v option. The list shows both the Compressed & uncompressed size of each file in the archive.

$ unzip -v archive.zip
Archive: archive.zip

| Length | Method | Size | Ratio | Date | Time | CRC-32 | Name |
|---|---|---|---|---|---|---|---|
| 3875302 | Defln:N | | | | | | |

# Shell programming

1) A shell can be used in one of two ways.
   i) Command Interpreter, used interactively.
   ii) A prgmg lang, to write shell scripts (your own Custom Commands)

2) Shell scripts is just a file containing shell Commands, but with a few extras.
   1st line of a shell script should be a Comment of the following form
   #!/bin/sh.
   Bourne shell script is the most Commonly used.

3) A shell program must be readable & executable.
   chmod u+rx Scriptname.

4) As with any Command, a shell script has to be "in your path" to be executed.
   If '.' is not in your path, you must specify
   ./ scriptname.sh.

Ex: $ hello.sh

```
#!/bin/sh
echo "Hello world"
```

$ Helloword.

*gedit scriptfile.sh

```
[#! /bin/bash]
ls
=-
$ sh scriptfile.sh.
```

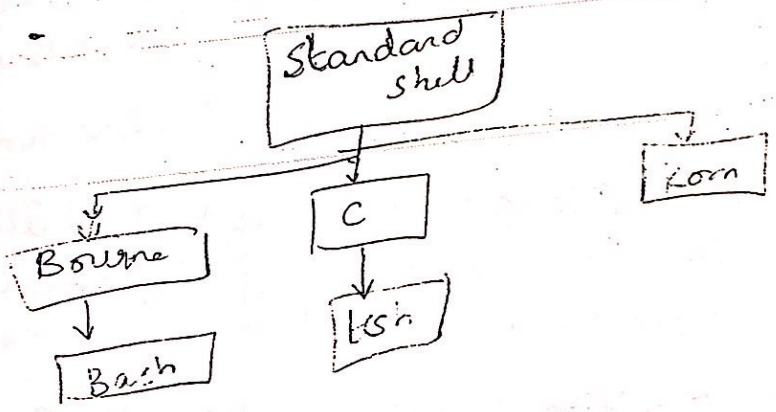
:) Shell scripts are files in which we write the sequence of commands that we need to perform & is executed using shell utility.

shell script begins with

#! /bin/bash.

#! : is a line for which #! is prefixed to interpret a path.

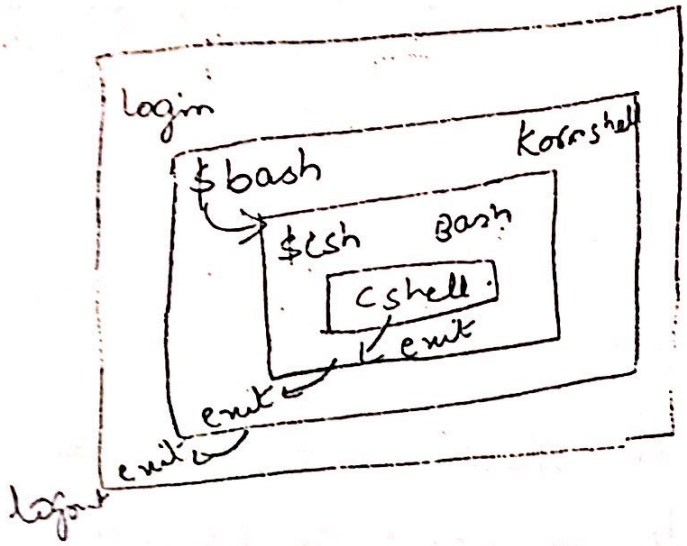
/bin/bash is interpreter command path for shell bash.




Two major parts to a shell.

1st → Interpreter → reads your commands & works with the kernel to execute them.

2nd shell is a programming capability that allows




$ ls chap?.

chapx chapy chapz

$ ls chap??

chap01 . chap02

$ ls .???* (hidden files) (matches atleast 3 characters after the dot).

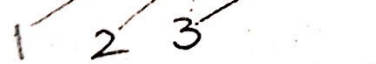
.bash_profile

$ ls emp*lst

emp.lst emp1.lst emp222lst

## Read.

```
$echo "enter the Values of a, b & c"
$read a, b, c
$echo "$a" "$b" "$c"
```




1, 2 c: null
a:1. b:2 Variable

1, 2 3 4
a b c

read ~~Command~~ statements starts assigning Values to Variables on a 1-to-1 basis & moment it finds it has Come across the last Variable & there is more than one Value left, it assigns rest of the Values to the last Variable in the list.

$ echo -e "I like work---(\r) I can sit & watch it for hours"

I like work---
↓ new line character.
I can sit & watch it for hours.

\r sequence is called carriage return. It causes the cursor to be positioned at the beginning of the current line

$ echo -e "I like work---\r I can sit and watch it for hours"

I can sit and watch it for hours.

## Exit and exit status of Command

exit status has two values.

. Program is runned succesfully

exit status = 0

exit status = | | unsuccesfull.

; Ex: $ cat foo

cat: Can't open file foo

Returns Nonzero exit status.

The shell Variable $? stores this status

Parameter $? → It stores the exit status of the last Command. It has the Value 0 if the Command succeeds and a nonzero Value if it fails. It is set by exit argument.

```
$ grep director emp.lst > /dev/null;
  echo $?                          Succes 0.

$ grep manager emp.lst > /dev/null;
  echo $?                          failure
```