UNIT - I

Features of UNIX

* UNIX is a OS so it has all the features an OS expected to have. UNIX also looks at a few things differently & possesses features unique to itself.

1) Multiuser System.

↳ UNIX is multiprogramming system, it permits multiple programs to run & compete for the attention of CPU. This can happen in two ways

* multiple users can run seperate jobs
* A single user can run multiple jobs.

↳ In UNIX the resources are actually shared between all users. UNIX is also multiuser sim.

↳ For creating illusory effect computer breaks up a unit of time into several segments & each user is allotted a segment. So at any time the machine will be doing job of single user.

2) multitasking System too.

A single user can also run multiple tasks concurrently: UNIX is a multitasking sim.

It is usual for the user that to edit a file, print a doc on printer, send email to friend & browsing.

3) The Building Block approach

↳ The designer never attempt to pack too many features in few tool instead they fill "small beautiful ~~beautiful~~" & developed a few hundred commands each of which performed one simple job.

↳ For ex:- (ls & wc) were used with | (pipe) to count number of files in your directory. no seperate command was designed to perform this tasks.

↳ using pipes & filter - UNIX implements small-is-beautiful philosophy.

4) UNIX Toolkit

↳ UNIX represents the kernel but kernel by itself doesnot do much. that can benefit users.

↳ To properly exploit power of UNIX you need to use most applications that are shipped with every UNIX sim

↳ These applications are quite diverse in scope - There are general purpose tools, text manipulating utility compiler & interpreter, network appl & sim admin tools.

5) pattern matching

↳ UNIX features very sophisticated pattern matching feature

↳ you listed chapter of text by using ls command with an unusual argument (chap*). Instead of explicitly specifying all filenames.

ls chap*.

↳ The '*' is special character used by the Shell to indicate that it can match a number of filenam

6) programming facility -

↳ UNIX shell is also programming language; it was designed for a programmer not a casual user.

↳ It has all necessary ingredients like control structure loop & variables that establish it as powerful progra -mming language in its own right.

↳ These features are used to design shell script - programs that can invoke UNIX commands

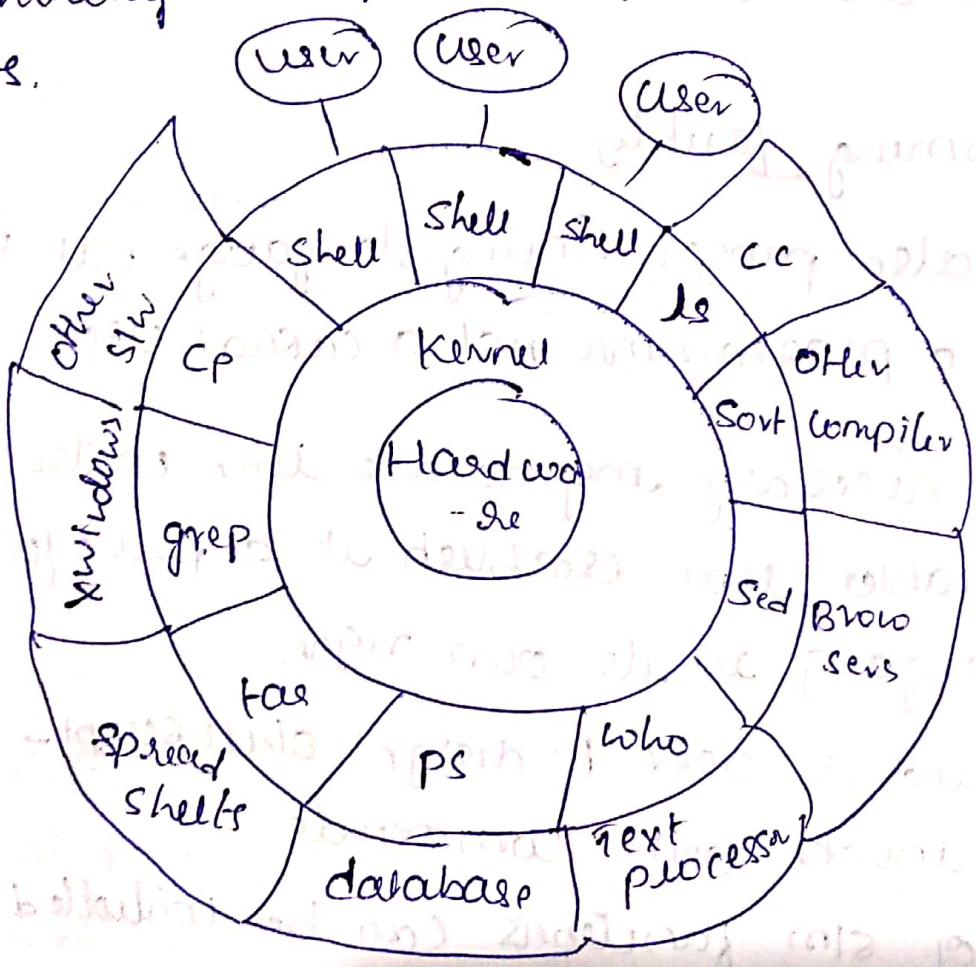
↳ many of Shell functions can be controlled & auto -mated using shell scripts.

7) Documentation

The principal online help facility available in the 'man' command which remains most important reference for commands & their configuration files.

UNIX Architecture

The entire UNIX system is supported by a handful of essentially a simple though somewhat abstract concepts.



The Kernel - Shell relationship

* division of labor : kernel & shell.

↳ Among these "fertile idea" is the division of labor between two agencies - kernel & shell.

↳ The kernel interact with machine's hardware

↳ The shell interacts with the user.

↳ kernel is core of the OS - a collection of routines mostly written in C. it is loaded into memory when the sys is booted & communicate directly with Hlw.

↳ user programs (applications) which needs access to the Hlw (like hard disk or terminal) uses the services of kernel

↳ These programs can access the kernel through set of functions called System calls.

↳ Apart from providing support to user programs kernel also manages sys memory, shedules processes decides their priority

↳ It is often called the operating system - a program -ers gateway to computer!

↳ computer don't have any inherent capability to translate command in action. That requires a command interpreter - a job that is handled by the outer part of operating system - the shell.

↳ shell is actually a interface b/n user & kernel

↳ Even though there is only one kernel running on the system there are several shells in action one for each user who is logged in.

↳ when you enter a command through keyboard the shell thoroughly examines keyboard input for special character. If it finds any, it rebuild a simplified command line & finally communicate with the kernel to see the command is executed.

2) The files & process.

Two simple entities support UNIX sim - the file & process.

↳ kaare Crishian detects two powerful illusion in them - file have place & process have & life

↳ Files is just a array of bytes and contains virtually anything

↳ It is also related to another file. being part of single hierarchial structure

↳ process which is name given to any file when is executed as a program

↳ you can say that process is simply the "time image" of an executable file.

↳ like files processes also belongs to a seperate hierarchial structure.

3> System Calls

UNIX s/m comprising the kernel, shell & applications is written in C.

↳ Though there are over a thousands of commands in the s/m they all use handfull of functions called System calls to communicate with kernel.

↳ example: A typical UNIX command writes a file with 'write' System call without going into the innards that actually achieves write operation.

Locating commands

↳ UNIX s/m is command based things happen because of command you type in

↳ UNIX commands are seldom more than 4 characters long & all commands are single words & lowercase

The path: The sequence of directories that the shell searches to look for a command is specified in its own PATH variable

↳ use 'echo' to evaluate this variable & you'll see directory list seperated by colon

```
$ echo $PATH
/bin:/user/bin:/user/bin)usr/ccs/bin:)user)
  local/java/bin:.
```

↳ dot at the end specifies the current directory

Internal & external commands

external commands: These are not built-in commands Rather stored in seperate directories

↳ Since it is a program or file having indepen- dent existance in /bin directory

internal commands: are built in commands

↳ echo is an internal command it will be executed from its own set of built in commands.

Command structure

Syntax to write commands

↳ First word is command, additional words are called arguments.

↳ commands and arguments can be seperated by space

options

↳ special type of argument that is used with - sign

ex:- ls -l note

↳ -l is argument to ls by defination it is known as options

↳ options can be combined together with single - sign

ls -l -a -t

ls -lat

File name arguments

↳ UNIX commands use filename as argument so that the command take input from the file.

ls -lat chap01 chap02 chap03

cp chap01 chap02 progs cp copies file

3> Exceptions

There are some commands which does not accept arguments

ex: pwd, who,

Flexibility of command usage.

1> combining commands

↳ UNIX allows you to specify more than one comma -nd in command line.

↳ each command has to be seperated from one another by a ;

ex; wc note ; ls -l note

2> command line can be overflow or be split into multiple lines.

↳ Terminal is restricted to only 80 character. that doesn't prevent you to enter lengthy command

↳ command simply overflow to the next line through it is still in single logical line., shell issues Secondry prompt (>) to indicate to you that the command isn't complete

```
$ echo "This is
> a three line
> text message"
```

3) entering command before previous command has finished

→ UNIX provides full duplex terminal which lets you type a command at any time & rest assured that the shell will interpret it.

2. FILE SYSTEM

1) The File

* The file is a container for storing information
* UNIX treats directory and device as file as well.

files are divided into 3 categories.

1) ordinary file: It contains only ~~stream~~ Stream of ~~data~~ characters.

ordinary file itself is divided into two

a) Text file: contains only printable characters. you can also view the content & make sense out of it.

b) Binary file: contains both printable & unprintable characters that cover the entire ASCII range.

Most of UNIX commands are binary file.

2) Directory File: contains no data but keeps some details of file & subdirectories that it contains.

* Directory file contains an entry for every file or directory that it houses. ex:- if you have 20 file in a directory then there will be 20 entries each entry has two components
  ↳ file name
  ↳ A unique identification numbers for the file or dir (called inode numbers)

* you can't write a directory file but you can perform some action that makes the kernel to write directory

3> Device File.
  ↳ Device filenames are generally found inside a single directory structure, /dev.
  ↳ A device file is a indeed special, it is not really a stream of characters. In fact it doesn't contain anything at all.

2.2 What's in a File name.

  ↳ File name can contains upto 255 characters
  ↳ File may or may not have extensions, and can consists of particularly any ASCII characters except / & NULL character
  ↳ File can have any number dots embedded in its name, a.b.c.d:
  ↳ UNIX is case sensitive.

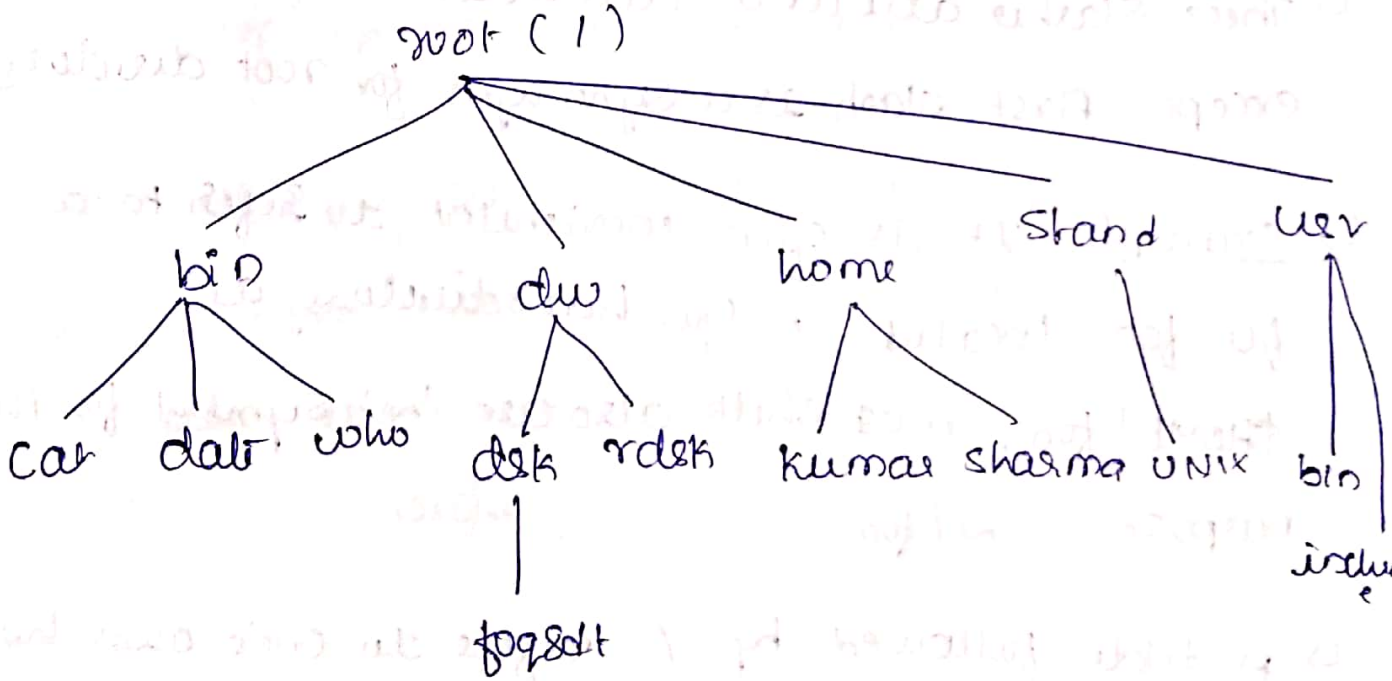
2.3 Parent child Relationship

↳ All files in UNIX are related to one another
The file sys in UNIX is a collection of all of these Related files

↳ The implicit feature of UNIX is that there is top which serves as reference point for all files. This top is called 'Root' & is Represented by '/', root is actually directory

↳ The Root directory will have number of subdirectory under it.

↳ These sub directories inturn will have subdirectory & other files



* every file apart from root must have a parent & it should be possible to trace the ultimate parentage of file to root.

2.4 The HOME variable : The HOME directory

↳ when you log onto sim UNIX automatically place you in home directory.

↳ It is created by sim when a user account is created

↳ shell variable HOME knows your home directory

```
$ echo $HOME
  /home/kumar
```

↳ /home/kumar is absolute pathname - which is simply sequence of directory names seperated by slash

↳ These slashes acts as a delimeter to fix a directory except first slash is a synonym for root directory

↳ Example : It is often convenient to refer to a file foo located in your home directory as. $HOME/foo. most shells also use '~' symbol for this purpose ~/foo.

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

2.5 pwd : Knowing your current directory

↳ user is placed in one directory of the file s/m on logging in

↳ you can move around file s/m ~~also~~ from one directory to another but at any point of time you are located in only one directory is known as current directory

↳ at any time, you should be able to know what your current directory is. The pwd (print working directory) command tells you that.

```
$pwd
/home/kumar
```

↳ pwd displays absolute pathname.

2.6 cd : Changing the current directory

↳ you can move around the file s/m using cd command

↳ when used with argument, it changes the current directory to directory specified by the argument.

```
$pwd
/home/kumar
$cd pgms
/home/kumar/pgms
```

cd pgms : means change your subdirectory to ~~pgg~~ pgms under current directory

using pathname causes no harm

cd /home/kumar/pgms

↳ when you need to switch to bin directory where most of commonly used UNIX commands are kept you should use the absolute pathname.

```
$pwd
/home/kumar/pgms
$cd /bin                absolute pathname required
$pwd                    here because /bin is not in a
/bin                    current directory.
```

↳ cd can also be used without arguments

```
$pwd
/home/kumar/pgms
$cd                     cd used without arguments
$pwd                    reverts to the home directory
/home/kumar
```

↳ when cd is used without argument it simply reverts to its home directory. it doesn't show you the current directory

2.7 mkdir : Making Directory

↳ directories are created with mkdir command.
↳ command is followed by name of directory to be created
↳ directory path is created under current directory

mkdir path

↳ you can creat any number of subdirectories using one mkdir command.

mkdir pis pis/progs pis/data ; creates directory tree.

↳ The order of specifying the argument is important.

* Sometimes o/m refuses to creat directory
This can happen due to these reasons:
1) The directory may already exist
2) There may be an ordinary file with that name in the current directory
3) permision set for the current directory don't permit creation of directory.

2.8 rmdir : Removing Directory

↳ The rmdir command removes directories

rmdir pis Directory must be empty

↳ like mkdir rmdir can delete more than one directory

rmdir pis/data pis/progs pis

↳ you can't delete the directory until it is empty

↳ you can't remove directory unless you are placed in a directory which is hierarchially above the one you have chosen to remove

Ex2> $ cd progs.

$ pwd

/home/kumar/pis/progs

rmdir : directory "home/kumar/pis/progs" directory does not exist.

To Remove this directory you must position yourself in directory above pis. progs

↳ mkdir & rmdir commands work only in directory owned by user

2.9 Absoluti pathname

cat /home/kumar/login.sql

If the first character of pathname is / the file location must be determined with respect to root (/) Such a pathname as the one above is called absoluti pathname.

using absoluti pathname for a command.

↳ when you specify date command, the sh has to locate file date from the list of directories specified in the PATH variable & then execute it

↳ However if you know the location of a peticular command you can preceed its name with complete path

2.10 Relative pathnames

cd progs

cat login.sql

cd progs/scripts

above commands uses pathnames which are not absoluti pathnames because they did't start with (/) .they are form of relative path

using . and .. in Relative pathnames

pwd
/home/kumar/pis/progs
cd /home/kumar/pis

↳ changed directory from /home/kumar/pis/progs to its parent directory using cd command

↳ Navigation often easies by using ancestor as reference

↳ UNIX offers shortcut - the relative pathname - that uses either the current or parent directory as reference + specify path relative to it

- . (Single dot) this represents current directory
- .. (two dots) this represents parent directory

Example: Assuming that you are placed in /home/kumar/progs/data/text you can use .. as an argument to cd to move to parent directory

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

↳ you can combine any such sets

```
$ cd ../..
$ pwd
  /home
```

2.11 ls : Listing directory contents

↳ ls command can be used to obtain a list of all filenames in the current directory

```
$ ls
```

will display list of filenames in the current directory

↳ directories often contain many files & you may simply be interested in only knowing whether a particular file is available in that case just use ls with filename

```
$ ls calender
calender

$ ls perl
perl: No such file or directory
```

↳ ls can also be used with multiple filename & options that list most of file attributes.

2.11.1 ls options

* output in multiple columns (-x) : when you have several files its better to display filenames in multiple column.

```
$ ls -x
```

Identifying directories & executables (-F) : To identify directories & executable files -F options should be used

$ ls -Fx

Packets.html	TOC.sh	calendar
dept.lst	emp.lst	helpdisk/

Showing hidden files(-a) : ls does not normally shows all files in a directory there are some hidden files -a option lists all hidden files as well

Handling ordinary files

3.1 cat : Displaying and creating files

↳ cat command is used to display the content of small file on the terminal

ex:- $ cat dept.lst

01| accounts |2123
02| progs| 5423
031 marketing | 6521

↳ like other commands cat also accepts more than one filenames as arguments

cat chap01 chap02

The content of 2nd file are shown immediately after first file without header information

cat options (-v and -n)

↳ displaying nonprinting characters (-v) : cat normally displays text file. Executables, when seen with cat : simply displays junk values.

* If you have nonprinting ASCII in your file then you can use [-v] option display these characters

↳ Numbering Lines (-n) : The -n option numbers lines

* C compiler indicate line numbers where errors are detected & it helps a programmer to debug program

* your vi editor can show line numbers too, if your version of cat ~~does~~ support -n options.

Using cat for creating file.

↳ cat is also useful for creating a file.

$ cat > foo

> Symbol following command means that the o/p goes to the filename following it.

3.2 CP: Copying Files

↳ the cp (copy) command copies file or a group of file

↳ Syntax

cp chapo1 unit1

when both are ordinary files it copies first file content to second file.

↳ If the destination file doesn't exists it will be created ~~be created~~ before copying takes place. If not it simply be overwritten without any warning from s/m

cp chapo1 progs/unit1 — chapo1 copied to unit1 under progs

cp chapo1 progs — chapo1 retains its name under progs

↳ cp often used with shorthand notation . (dot) to signify the current directory.

To copy file .profile from /home/shasma to current directory you can use two commands either of

cp /home/shasma/.profile .profile

/home/shasma/.profile .

↳ cp can be used to copy more than one file. The last name must be directory

cp chap01 chap02 chap03 progs

For the above command to work progs directory must exist because cp can't create it.

cp options :

1) Interactive copying (-i): This options warns user before overwriting the distination file.

If unix1 exists, cp prompts for response

$ cp -i chap01 unix1

cp: overwrite unix1 (yes/No)? y

2) Copying directory structure (-R): UNIX commands are capable of Recursive behavior - This means that the command can descend a directory & examine all the files in subdirectory.

$ cp -R progs newprogs newprogs must not exists

cp -R command behaves recursively to copy an entire directory structure say, progs to newprogs

3.3 rm: Deleting Files

↳ The rm (remove) command deletes one or more files

rm chapo1 chapo2 chapo3

rm chap* could be dangerous

↳ you may sometime need to delete all files in directory as a part of cleanup operation.

$ rm * all files gone.

rm options

1> Interactive deletion (-i) : This option makes the comm -and ask the user for confirmation before removing each file.

$ rm -i chapo1
rm: remove chapo1 (yes|no)? yes

5.4 mv: Renaming file

↳ The mv command renames (moves) file. It has two distinct functions.

1) It renames a file (or directory)
2) It moves group of files to a different directory

↳ mv command doesn't create a copy of a file; it merely renames it No additional space is consumed on disk. during renaming

↳ If destination file is not exist it will be created

↳ like cp group of files can be moved to a directory

$ mv chap01 chap02 chap03 progs

↳ mv can also be used to rename directory for instance

pis to pesdis

mv pis pesdir

↳ -i, -R options are available with mv also works exactly ~~similar~~ in the same way as with cp

3.5 The lp Subsystem: printing a file

↳ No user allowed direct access to the printer one has to spool (line up) a job along with the others in a print queue. Spooling ensures orderly printing of jobs & relieves the user from the necessity of administrating the printing resource

↳ Spooling facility in System V is provided by the lp (line printing) command.

```
$ lp rfc822.ps
  request id is prd-320 (1 file)
```

lp options

↳ lp accepts above request because a default printer has been defined by administrator.

↳ If it is not or if there is more than one printer in the system, you have to use -d option with the printer name

```
$ lp -d laser chap01.ps
```

↳ -t option followed by title string, prints title on the first page

```
$ lp -t "First chapter" chap01.ps
```

↳ after the file has been printed you can notify the user with -m option (mail)

↳ you can also print multiple copies -n option

```
lp -n3 -m chapo1.ps
```

other commands in lp subsystem

↳ print queue is viewed with lpstat command

↳ you can use delete command to cancel any jobs submitted by you. cancel command uses requestid or printer name as arguments

```
$ cancel laser
$ cancel prl-320
```

3.8 wc: counting lines, words & characters

↳ UNIX features universal word counting program that also counts lines and characters

↳ It takes one or more characters & displays output in four columnar table

↳ before using wc use cat command to view its content

$ cat infile

I am the ~~word~~ wc command

I count character words & lines

You can now use wc without option to make a word count of data in file

$ wc infile

3 20 103 infile

3lines 20 words & 103 characters

↳ wc offers 3 options to make specific counts

-l option counts only the number of lines

-w options counts only the words

-c option counts only the characters

3.9 od : displaying data in octal

↳ many files contains non printing characters & most UNIX commands don't display them properly

↳ $ more odfile

white space includes a

The \a character rings a bell

The \f character skips a page

↳ od command displays data in octal form

$ od -b odfile

127 150 151 164 145 040 163
141 143 145 040 151 156 143
. . . .
:

↳ -b options displays values for each character seperate
↳ -b & -c will display octal value of each character along with a character

3.10 cmp: compare two files

↳ cmp commands can be used to compare two files.
↳ $ cmp chap01 chap02
↳ The two files are compared to each other byte by byte & the location of first mismatch is echoed to the screen.
↳ If two files are identical cmp displays no message, but simply returns the prompt

3.11 comm: what is common?

↳ suppose you have two lists of people & you asked to find out the names available in one & not in other or even those common in both

$ cat file1

C.K. Shukla
Chanchal Singhvi
S.N dasgupta
Sumit Chakrobarty

$cat file2

anil agarwa
basun sengapta
C.K Shukla
lalit chowdry
S.N dasgupta

$ comm file[12]

Chanchal
Sumit Chakrubarthy

anil
basun sengupta
lalit

C K shukla
SN dasgupta

↳ 1st column displays unique names in 1st file 2nd column - " - in 2nd file 3rd column display common names in both files

3.12 diff : converting one file to another

↳ diff is a third command that can be used to display file differences, it also tells you which line in file have to be changed to make two files identical.

$ diff file1 file2

0a1,2 Append after line 0 of first file
> anil aggarwa this line
> basun dasgupta & this line

2c 4 change line 2 of first file.
< chanchal singui Replacing this line
with
> lalit chawdey this line

Ud 5 Delete line 4 of first file.
< Sumi Chakrabarthy containing this line

Basic File Attributes

1> ls -l: listing file attributes

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

```
$ ls -l
total 72
-rw-r--r--  1 kumar metal 1951 may 10 13:45 chapo1
-rw-r--r--  1 kumar metal 4174 may 10 15:01 chap
drwxr-xr-x  2 kumar metal  512 may 9 9:37 progs
```

↳ total 70 indicates that total of 70 blocks are occupied by these file on disk.

File type & permissions: First column shows file-type & permission. First character in the column specifies whether it is a file or directory

- for ordinary files.

d for directory

↳ Series of characters that contain the values r, w, x & -. In UNIX system files have three types of permissions - read, write & execute.

↳ Links : Second column indicates the number of links associated with the file. This is actually number of filenames maintained by the system of that file. UNIX lets a file have as many names as you want it to have, even though there is a single file on disk.

↳ ownership: when you create a file you automatically become its owner. The owner has full authority to tamper with a file content & permissions - a privilage not available with others except root user.

↳ group ownership: when opening user accounts system administrator also assigns the user to some group.

↳ * The concept of group of users also owning a file has acquired importance today as group mates members often needs to work on the same file.

File Size: Fifth column shows size of the file in bytes i.e the amount of data it contains.

* The important things to remember is that it is only a character count of the file & not a measure of disk space that it occupies
* dipt.lst contains 84 bytes it would occupy 1024 bytes on disk on sim that use 1024 byte block size.

Last modification date: The sixth, seventh & eighth columns indicate last modification time of file.

Filename: Last column displays the filename.

2) The -d option: Listing directory attributes

↳ ls when used with directory names lists file in the directory rather than directory itself. To force ls to list the attribute of directory rather than its contents you need to use -d (directory) option.

```
$ ls -ld helpdir progs
drwxr-xr-x 2 kumar metal 512 may 9 10:31 helpdir
```

drwxr-xr-x 2 kumar metal 512 may 9 9:5
prog

3) File accessship

↳ when you create a file your username will display in the third column of file's listing you are the owner of the file

↳ several user may belongs to single group. People working on a project are generally assigned to a common group. & all files created by group members will have the same group owner.

↳ pevilage of group are set by the owner of the file not by group members

↳ when sys administrator create user account he has to assign these parameters to the user.

* The user id (UID): both its name & numeric representation

* group id (GID): - " -

↳ The file /etc/passwd contains UID & GID (Only nu

~~/etc/ptsgroup~~

/etc/group contains GID (both numbers & name).

↳ $id uid = 655537(kumar) gid = 65537(metal)

4> File permissions

↳ The initial - represents ordinary file & its left out string is file permission.

rwx r-x r--

↳ each group here represents category & contains 3 slots representing read, write & execute permission of file

↳ r indicate read permission which means cat can display content of file

↳ w indicate write permission: you can edit such file using editor.

↳ x indicate execute permission: the file can be executed as a program.

5> chmod: change file permissions

↳ we will refer to owner as user because that's how the chmod command refer to the owner.

$ cat /usr/bin/staex >x staex copies the file staex

$ ls -l xstaex

-rw x--r-- 1 kumar mtech 1906 Sep 5 23:38 xstaex

↳ by dfault a file does not have execute permission to do that file permission needs to be changed

↳ The chmod (change mode) command is used to set the permissions of one more file for all three categories of users (user, group and other).

↳ It can be run by the user & superuser.

The command can be used in 2 ways.

a> In a relative manner by specifying the current permissions

b> In a absolute manner by specifying final permission

5.1 Relative permissions

↳ when changing permission in relative manner chmod only changes the permission specified in the command line and leaves other permission unchanged.

Chmod category operation permission filename(s)

↳ chmod takes as its argument an expression comprising letters & symbols that completely describe category. & type of permission being assigned or removed

Expression contains 3 components

* user category (user, group, others)
* the operation to be performed (assign or remove)
* type of permission (r, w, x)

+ chmod u+x xstat

$ ls -l xstat

-rwxr--r-- 1 kumar mtai

↳ the above command assigns (+) execute (x) permission to user (u). both other permissions unchanged

you can now execute file if you are owner of the file.

but the other categories still can't

↳ To enable all of them to execute a file you have to use multiple characters to represent the user category (ugo)

$ chmod ugo+x xstat ; ls -l xstat

↳ string ugo combines all the category user, group

4 owners UNIX allows shorthand for this as 'a'

$ chmod a+x xstat

$ chmod +x xstat.

↳ chmod accepts multiple filenames in the command line when you need to assign the same permission to group of files

$ chmod u+r note1 note2

↳ permissions are removed with (-) operator.

$ chmod go-r xstart

above command removes read permission for group & other users

↳ ~~dem~~

↳ chmod also accepts multiple expression delimited by commas.

$chmod. a-x, go+r xstart.

↳ more than one permission can also be set

$chmod o+wx xstart

5.2 Absoluti permissions

↳ Sometimes you don't need to know what a file current permission are but want to set all permision bits explicitly. The expression used by chmod here is string of three octal numbers

↳ Read permission - 4 (100)

↳ write permission - 2 (010)

↳ Execute permision - 1 (001)

↳ you can use this method to assign read & write permision to all three categories as follows

```
& chmod 666 xstab ; ls -l xstab
-rw -rw -rw- 1 kumar mtech
```

↳ 6 indicate read & write permision (4+2).

using chmod Recursively (-R)

↳ It's possible make chmod descend hisarchy & apply the expression to every file & subdirectory it finds

```
chmod -R a+x shell-scripts
```

This make all directories found in the tree walk ex-ecutable by all users

Directory permission

↳ directory also have their own permission & significance of these permission differs a great deal from those of ordinary file.

↳ Read & write permission of files are also influenced by the permissions of the directory housing them

$ mkdir c-progs ; ls -ld c-progs

drwxr-xr-x 2 kumar metal 512 may 9 9:57 c-progs.

↳ The default permissions of a directories on this Sim are rwxr-xr-x. A directory must never be writable by groups others

Hard links

↳ the link count is displayed in the 2nd column of the listing. This count is normally 1. But some of files have 2 links

↳ All attributes seems to be identical but the files could still be copies But this can only be confirmed by using the -i option with ls command

$ ls -li backup.sh restore.sh

U78274 -rwxr-xr-- 2 kumar mtech ... backup
U78274 — " — ... restore.sh

ln: create a hard links

↳ A file is linked with ln (link) command which takes two filenames as arguments. command can create soft & hard link.

$ ln emp.lst employee employee must not exists

Q9518 -rwxr-xr-x 3 kumar mtech ... empdata
Q9518 — " — employee

→ you can link multiple files.

where to use hard links

1) let's consider that you have written

The Directory

↳ Directory has its own permissions, owners & links.

↳ However significance of file attribute changes a great deal with when applied to a directory

↳ permission too acquire different meaning when the term is applied to directory

Read permission

↳ Read permission for a directory means that list of file names stored in that directory are accessable

↳ Since ls reads the directory to display filenames if a directory's read permission is removed then ls won't work.

```
$ ls -ld progs
drwxr-xr-x  2  kumar metal . . .
$ chmod -r progs ; ls progs
progs: permission denied
```

write permission

↳ Be aware you can't write to the directory file only kernel can do that. If that were possible then any user could destroy the intigrity of the file s/m.

↳ write permission for a directory implies that you are permitted to create or remove files in it

$ chmod 555 progs : ls -ld progs

dr-xr-xr-x 2 kumar mca

$ cp emp.lst progs

cp: cannot create progs/emp.lst : permission denied

Execute permission

↳ Executing a directory doesn't make any sense so what does its execute privilage mean?

↳ It only means that user can "pass through" the directory in searching for subdirectories when you use pathname with any command

$ cat /home/kumar/prog/emp.sh

$ chmod 666 progs : ls -ld progs

drw-rw-rw- 2 kumar mca

$ cd progs

bash: progs: permission denied

umask : default file & directory permissions

↳ when you creat a file or directory the permission assigned to them depends on sim's default settings

↳ UNIX sim has following default ~~setting~~ permissions for all files & directories

* rw-rw-rw- (666) for regulas files
* rwxrwxrwx (777) for directories

↳ you can't see these permissions when you creat a file or directory. Actually this default is transform -d by substracting user mask from it to remove one or more permissions.

```
$umask
022
```

↳ This is octal number which has to be substract -d from default to obtain actual default

This becomes 644 (666-022) for files & 755 (777-022) for directories. when you creat a file on this sim it will have permission

rw-r--r--