1. THE FILE

- The file is the container for storing information.
- Neither a file's size nor its name is stored in file.
- All file attributes such as file type, permissions, links, owner, group owner etc are kept in a separate area of the hard disk, not directly accessible to humans, but only to kernel.

- The UNIX has divided files into three categories:

1. **Ordinary file** – also called as regular file. It contains only data as a stream of characters.
2. **Directory file** – it contains files and other sub-directories.
3. **Device file** – all devices and peripherals are represented by files.

Ordinary File - ordinary file itself can be divided into two types-

1. **Text File** – it contains only printable characters, and you can often view the contents and make sense out of them.
2. Binary file – it contains both printable and unprintable characters that cover entire ASCII range.
   Examples- Most Unix commands, executable files, pictures, sound and video files are binary.

Directory File - a directory contains no data but keeps some details of the files and subdirectories that it contains. A directory file contains an entry for every file and subdirectories that it houses. If you have 20 files in a directory, there will be 20 entries in the directory. Each entry has two components-

- the filename
- a unique identification number for the file or directory (called as inode number).

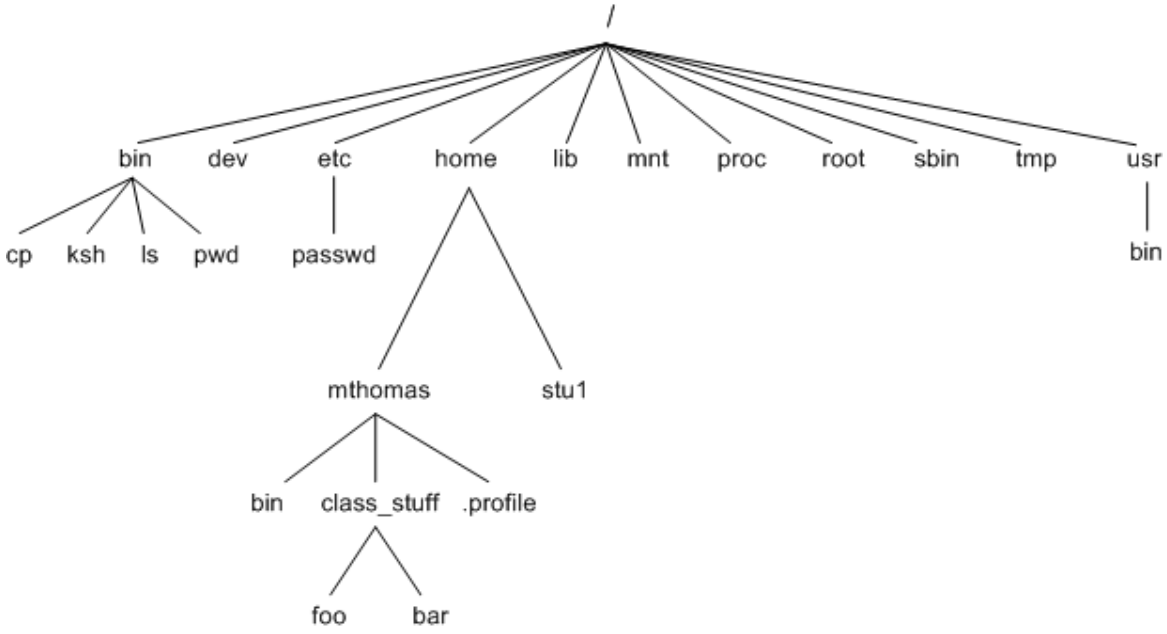
Device File - Installing software from CD-ROM, printing files and backing up data files to tape. All of these activities are performed by reading or writing the file representing the device. Advantage of device file is that some of the commands used to access an ordinary file also work with device file. Device filenames are generally found in a single directory structure, /dev.

2. WHAT'S IN A (FILE) NAME?

1. A filename can consist up to 255 characters.
2. File may or may not have extensions, and consist of any ASCII character expect the / & NULL character.
3. Users are permitted to use control characters or other unprintable characters in a filename.
4. Examples - .last_time list. @#$%*abcd a.b.c.d.e
5. But, it is recommended that only the following characters be used in filenames-
   - Alphabetic characters and numerals
   - the period(.), hyphen(-) and underscore(_).

3. THE PARENT-CHILD RELATIONSHIP

- The files in UNIX are related to one another.
- The file system in UNIX is a collection of all of these files (ordinary, directory and device files) organized in a hierarchical (an inverted tree) structure as shown in below figure.



- The feature of UNIX file system is that there is a top, which serves as the reference point for all files.
- This top is called root and is represented by a / (Front slash).
- The root is actually a directory.
- The root directory (/) has a number of subdirectories under it.
- The subdirectories in turn have more subdirectories and other files under them.
- Every file apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root.
- In parent-child relationship, the parent is always a directory.

4. The HOME VARIABLE: HOME DIRECTORY

- When you logon to the system, UNIX places you in a directory called home directory.
- It is created by the system when the user account is created.
- If a user login using the login name kumar, user will land up in a directory that could have the path name /home/kumar.

- The shell variable HOME knows the home directory.

$echo $HOME
/home/kumar

5. pwd: CHECKING YOUR CURRENT DIRECTORY

- Any time user can know the current working directory using pwd command.

  $ pwd
  /home/kumar
- Like HOME it displays the absolute path.

6. cd: CHANGING THE CURRENT DIRECTORY

- User can move around the UNIX file system using cd (change directory) command.
- When used with the argument, it changes the current directory to the directory specified as argument, progs:

  $ pwd
  /home/kumar
  $cd progs
  $ pwd
  /home/kumar/progs
- Here we are using the relative pathname of progs directory. The same can be done with the absolute pathname also.

  $cd /home/kumar/progs
  $ pwd
  /home/kumar/progs

  $cd /bin
  $ pwd
  /bin

- cd can also be used without arguments:

  $ pwd
  /home/kumar/progs
  $cd
  $ pwd
  /home/kumar

- cd without argument changes the working directory to home directory.

  $cd /home/sharma
  $ pwd
  /home/sharma
  $cd
  /home/kumar

7. mkdir: MAKING DIRECTORIES

- Directories are created with **mkdir** (make directory) command. The command is followed by names of the directories to be created. A directory patch is created under current directory like this:

  $mkdir patch

- You can create a number of subdirectories with one **mkdir** command:
  $mkdir patch dba doc

- For instance the following command creates a directory tree:
  $mkdir progs progs/cprogs progs/javaprogs

- This creates three subdirectories – progs, cprogs and javaprogs under progs.

- The order of specifying arguments is important. You cannot create subdirectories before creation of parent directory.
- For instance following command doesn't work

  $mkdir progs/cprogs progs/javaprogs progs
  mkdir: Failed to make directory "progs/cprogs"; No such directory
  mkdir: Failed to make directory "progs/javaprogs"; No such directory

- **System refuses to create a directory due to fallowing reasons:**
- The directory is already exists.
- There may be ordinary file by that name in the current directory.
- User doesn't have permission to create directory

8. rmdir: REMOVING A DIRECTORY

- The **rmdir** (remove directory) command removes the directories. You have to do this to remove progs:

  $rmdir progs

- If **progs** is empty directory then it will be removed form system.

- **rmdir** expect the arguments reverse of mkdir.

- Following command used with **mkdir** fails with **rmdir**

$mkdir progs progs/cprogs progs/javaprogs

rmdir: directory "progs": Directory not empty

- First subdirectories need to be removed from the system then parent.
- Following command works with **rmdir**

$mkdir progs/cprogs progs/javaprogs progs

- First it removes **cprogs** and **javaprogs** form **progs** directory and then it removes **progs** fro system.

- **rmdir : Things to remember**
- You can't remove a directory which is not empty
- You can't remove a directory which doesn't exist in system.
- You can't remove a directory if you don't have permission to do so.

9. ABSOLUTE PATHNAME

- Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.
- Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.
- Following are some examples of absolute filenames.
  /etc/passwd
  /users/kumar/progs/cprogs
  /dev/rdsk/Os3

Example

- date command can executed in two ways as
  $date **// Relative path**
  Thu Sep 7 10:20:29 IST 2017

  $/bin/date **// Absolute path**
  Thu Sep 7 10:20:29 IST 2017

10. RELATIVE PATHNAME

- A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this –

  progs/cprogs
  rdsk/Os3

Using . and .. in relative path name

- User can move from working directory /home/kumar/progs/cprogs to home directory /home/kumar using cd command like

  $pwd
  /home/kumar/progs/cprogs
  $cd /home/kumar
  $pwd
  /home/kumar

- Navigation becomes easy by using common ancestor.
- **. (a single dot)** - This represents the current directory
- **.. (two dots)** - This represents the parent directory

- Assume user is currently placed in /home/kumar/progs/cprogs

  $pwd
  /home/kumar/progs/cprogs
  $cd ..
  $pwd
  /home/kumar/progs

- This method is compact and easy when ascending the directory hierarchy. The command **cd ..** Translates to this "change your current directory to parent of current directory".

- The relative paths can also be used as:

  $pwd
  /home/kumar/progs
  $cd ../..
  $pwd
  /home

- The following command copies the file prog1.java present in javaprogs, which is present is parent of current directory to current directory.

  $pwd
  /home/kumar/progs/cprogs
  $cp ../javaprogs/prog1.java .
- Now prog1.java is copied to cprogs under progs directory.

FILE RELATED COMMANDS

11. cat: DISPLAYING AND CREATING FILES

cat command is used to display the contents of a small file on the terminal.

```
$ cat cprogram.c
# include <stdioh>
void main ()
{
        Printf("hello");
}
```

As like other files cat accepts more than one filename as arguments

```
$ cat ch1 ch2
It contains the contents of chapter1
It contains the contents of chapter2
```

In this the contents of the second files are shown immediately after the first file without any header information. So cat concatenates two files- hence its name.

cat OPTIONS

- **Displaying Nonprinting Characters (-v)**
  cat without any option it will display text files. Nonprinting ASCII characters can be displayed with –v option.
- **Numbering Lines (-n)**
  -n option numbers lines. This numbering option helps programmer in debugging programs.

Using cat to create a file

cat is also useful for creating a file. Enter the command **cat**, followed by > character and the filename.

```
$ cat > new
This is a new file which contains some text, just to
Add some contents to the file new
[ctrl-d]
$_
```

When the command line is terminated with **[Enter]**, the prompt vanishes. Cat now waits to take input from the user. Enter few lines; press **[ctrl-d]** to signify the end of input to the system
To display the file contents of new use file name with **cat** command.

$ cat new
This is a new file which contains some text, just to
Add some contents to the file new

12. cp: COPYING A File

The **cp** command copies a file or a group of files. It creates an exact image of the file on the disk with a different name. The **syntax takes two filename** to be specified in the command line.

- When both are ordinary files, **first file is copied to second.**
  $ cp csa csb
- If the destination file (csb) doesn't exist, **it will first be created before copying takes place.** If not it will simply be overwritten without any warning from the system.
- Example to **show two ways of copying files to the cs directory:**

  $ cp ch1 cs/module1 ch1 copied to module1 under cs
  $ cp ch1 cs ch1 retains its name under cs

- cp can also be used with the shorthand notation, .(dot), to signify the current directory as the destination. To copy a file **'new'** from /home/user1 to your current directory, use the following command:
  $cp /home/user1/new new ***destination is a file***
  $cp /home/user1/new . ***destination is the current directory***

- cp command can be used **to copy more than one file with a single invocation of the command.** In this case the last filename must be a directory.
- **Ex: To copy the file ch1,chh2,ch3 to the module , use cp as**
  $ cp ch1 ch2 ch3 module

- The files will have the same name in **module**. If the files are already resident in **module**, they will be overwritten. In the above diagram module directory should already exist and cp doesn't able create a directory.
- UNIX system uses * as a shorthand for multiple filenames.
- **Ex:**
  $ cp ch* usp Copies all the files beginning with ch

cp options

- **Interactive Copying(-i) :** The **–i** option warns the user before overwriting the destination file, If unit 1 exists, cp prompts for response
  $ cp -i ch1 unit1
  $ cp: overwrite unit1 (yes/no)? Y
- A y at this prompt overwrites the file, any other response leaves it uncopied.

Copying directory structure (-R) :

- It performs recursive behavior command can descend a directory and examine all files in its subdirectories.
- **-R : behaves recursively to copy an entire directory structure**
  $ cp -R usp newusp
  $ cp -R class newclass

- If the **newclass/newusp** doesn't exist, **cp** creates it along with the associated subdirectories.

13. rm: DELETING FILES

- The rm command deletes one or more files.
- Ex: Following command deletes three files:

  $ rm mod1 mod2 mod3

- Can remove two chapters from usp directory without having to cd
- **Ex:**
  $rm usp/marks ds/marks

- To remove all file in a directory use *****
  $ rm *
- Removes all files from that directory

rm options

- **Interactive Deletion (-i) :** Ask the user confirmation before removing each file:
  $ rm -i ch1 ch2
  rm: remove ch1 (yes/no)? ? y
  rm: remove ch1 (yes/no)? ? n [Enter]

- A 'y' removes the file (ch1) any other response like n or any other key leave the file undeleted.
- **Recursive deletion (-r or -R):** It performs a recursive search for all directories and files within these subdirectories. At each stage it deletes everything it finds.

  $ rm -r * ***Works as rmdir***

- It deletes all files in the current directory and all its subdirectories.

- **Forcing Removal (-f): rm** prompts for removal if a file is **write-protected.** The **-f** option overrides this minor protection and forces removal.

  rm -rf* **Deletes everything in the current directory and below**

14. mv: RENAMING FILES

- **The mv command renames (moves) files. The main two functions are:**
  1. **It renames a file(or directory)**
  2. **It moves a group of files to different directory**
- It doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming.

- **Ex: To rename the file csb as csa we can use the following command**

  $ mv csb csa

- If the destination file doesn't exist in the current directory, it will be created. Or else it will just rename the specified file in mv command.
- A group of files can be moved to a directory.
- Ex: Moves three files ch1,ch2,ch3 to the directory module

  $ mv ch1 ch2 ch3 module

- Can also used to rename directory

  $ mv rename newname

- mv replaces the filename in the existing directory entry with the new name. It doesn't create a copy of the file; it renames it
- Group of files can be moved to a directory
- mv chp1 chap2 chap3 **unix**

15. more : PAGING OUTPUT

- To view the file ch1, we can use more command along with the filename, it is used for display

  $ more odfile press q to exit
- this file is an example for od command ^d used as an interrupt key ^e indicates the end of file.
- It displays the contents of ch1 on the screen, one page at a time. If the file contents is more it will show the filename and percentage of the file that has been viewed:

  ----More--- (15%)

Navigation

- f or Spacebar: to scroll forward a page at a time
- b to move back one page

Using more in pipeline

- The **ls** outpu won't fit on the screen if there are too many files, So the command can be used like this:

  ls | more

- The pipeline of two commands where the output of two commands, where the output of one is used as the input of the other.

16. wc: COUNTING LINES,WORDS AND CHARACTERS

- wc command performs Word counting including counting of lines and characters in a specified file. It takes one or more filename as arguments and displays a four columnar output.

  $ wc ofile
  4 20 97 ofile

- Line: Any group of characters not containing a newline
- Word: group of characters not containing a space, tab or newline
- Character: smallest unit of information, and includes a space, tab and newline

- **wc** offers 3 options to make a specific count. –l option counts only number of lines, -w and –c options count words and characters, respectively.

  $ wc -l ofile
  4 ofile
  $ wc -w ofile
  20 ofile

- Multiple filenames, **wc** produces a line for each file, as well as a total count.

  $ wc -c ofile file
  97 ofile
  15 file
  112 total

17. od: DISPLAYING DATA IN OCTAL

- **od** command displays the contents of executable files in a **ASCII octal value.**

  $ more ofile
  this file is an example for od command
  ^d used as an interrupt key

- -b option displays this value for each character separately.
- Each line displays 16 bytes of data in octal, preceded by the **offset in the file of the first byte in the line.**

  $ od –b file

  0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
  0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
  0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
  0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
  0000100 145 171

- **-c character option**
- Now it shows the printable characters and its corresponding ASCII octal representation

  $ od –bc file

  od -bc ofile
  0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
  T h i s f i l e i s a n
  0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
  e x a m p l e f o r o d c
  0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
  o m m a n d \n ^ d u s e d a
  0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
  s a n i n t e r r u p t k
  0000100 145 171
  e y

- Some of the representation:
- The tab character, [ctrl-i], is shown as \t and the octal vlaue 011
- The bell character , [ctrl-g] is shown as 007, some system show it as \a
- The form feed character,[ctrl-l], is shown as \f and 014
- The LF character, [ctrl-j], is shown as \n and 012
- Od makes the newline character visible too.

BASIC FILE ATTRIBUTES

The UNIX file system allows the user to access other files not belonging to them and without infringing on security. A file has a number of attributes (properties) that are stored in the inode. In this chapter, we discuss,

- ls –l to display file attributes (properties)
- Listing of a specific directory
- Ownership and group ownership
- Different file permissions

18. LISTING FILE ATTRIBUTES

ls command is used to obtain a list of all filenames in the current directory. The output in UNIX lingo is often referred to as the listing. Sometimes we combine this option with other options for displaying other attributes, or ordering the list in a different sequence. ls look up the file's inode to fetch its attributes. **It lists seven attributes of all files in the current directory and they are:**

- **File type and Permissions**
  - The file type and its permissions are associated with each file.
- **Links**
  - Links indicate the number of file names maintained by the system. This does not mean that there are so many copies of the file.
- **Ownership**
  - File is created by the owner. The one who creates the file is the owner of that file.
- **Group ownership**
  - Every user is attached to a group owner. Every member of that group can access the file depending on the permission assigned.
- **File size**
  - File size in bytes is displayed. It is the number of character in the file rather than the actual size occupied on disk.
- **Last Modification date and time**
  - Last modification time is the next field. If you change only the permissions or ownership of the file, the modification time remains unchanged. If at least one character is added or removed from the file then this field will be updated.
- **File name**
  - In the last field, it displays the file name.

For example,

```
$ ls -l
total 72
-rw-r--r-- 1 kumar metal 19514 may 10 13:45 chap01
-rw-r--r-- 2 kumar metal 19555 may 10 15:45 chap02
drwxr-xr-x 2 kumar metal 512 may 09 12:55 helpdir
drwxr-xr-x 3 kumar metal 512 may 09 11:05 progs
```

Listing Directory Attributes

$ls -d

This command will not list all subdirectories in the current directory .

For example,

$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 09:57 progs

- Directories are easily identified in the listing by the first character of the first column, which here shows a d.
- The significance of the attributes of a directory differs a good deal from an ordinary file.
- To see the attributes of a directory rather than the files contained in it, use ls –ld with the directory name. Note that simply using ls –d will not list all subdirectories in the current directory. Strange though it may seem, ls has no option to list only directories.

File Ownership

- When you create a file, you become its owner. Every owner is attached to a group owner. Several users may belong to a single group, but the privileges of the group are set by the owner of the file and not by the group members. When the system administrator creates a user account, he has to assign these parameters to the user:

  The user-id (UID) – both its name and numeric representation
  The group-id (GID) – both its name and numeric representation

File Permissions

UNIX follows a three-tiered file protection system that determines a file's access rights. It is displayed in the following format: Filetype owner (rwx) groupowner (rwx) others (rwx)

For Example:

-rwxr-xr-- 1 kumar metal 20500 may 10 19:21 chap02

rwx r-x r--

owner/user group owner others

- The first group has all three permissions. The file is readable, writable and executable by the owner of the file.
- The second group has a hyphen in the middle slot, which indicates the absence of write permission by the group owner of the file.
- The third group has the write and execute bits absent. This set of permissions is applicable to others.

- You can set different permissions for the three categories of users – owner, group and others. It's important that you understand them because a little learning here can be a dangerous thing. Faulty file permission is a sure recipe for disaster.

19. CHANGING FILE PERMISSIONS

A file or a directory is created with a default set of permissions, which can be determined by umask. Let us assume that the file permission for the created file is -rw-r-- r--. Using chmod command, we can change the file permissions and allow the owner to execute his file.

The command can be used in two ways:

- In a **relative** manner by specifying the changes to the current permissions
- In an **absolute** manner by specifying the final permissions

Relative Permissions

- chmod only changes the permissions specified in the command line and leaves the other permissions unchanged.
- Its **syntax** is:
  chmod category operation permission filename(s)
- chmod takes an expression as its argument which contains:
- user category (user, group, others)
- operation to be performed (assign or remove a permission)
- type of permission (read, write, execute)

- **Category : u – user g – group o – others a - all (ugo)**
- **operations : + assign - remove = absolute**
- **permissions: r – read w – write x - execute**

- Let us discuss some examples:

- Initially,
  -rw-r—r-- 1 kumar metal 1906 sep 23:38 xstart
  $chmod u+x xstart
  -rwxr—r-- 1 kumar metal 1906 sep 23:38 xstart

- The command assigns (+) execute (x) permission to the user (u), other permissions remain unchanged.
  $chmod ugo+x xstart or chmod a+x xstart or chmod +x xstart
  $ls –l xstart
  -rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart

- chmod accepts multiple file names in command line
  $chmod u+x note note1 note3

- Let initially,

  -rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
  $chmod go-r xstart

- Then, it becomes

$ls –l xstart
-rwx—x--x 1 kumar metal 1906 sep 23:38 xstart

Absolute Permissions

- Here, we need not to know the current file permissions. We can set all nine permissions explicitly. A string of three octal digits is used as an expression. The permission can be represented by one octal digit for each category. For each category, we add octal digits. If we represent the permissions of each category by one octal digit, this is how the permission can be represented:

Read permission – 4 (octal 100)
Write permission – 2 (octal 010)
Execute permission – 1 (octal 001)

Octal	Permissions	Significance
0	---	no permissions
1	--x	execute only
2	-w-	write only
3	-wx	write and execute
4	r--	read only
5	r-x	read and execute
6	rw-	read and write
7	rwx	read, write and execute

- We have three categories and three permissions for each category, so three octal digits can describe a file's permissions completely. The most significant digit represents user and the least one represents others. chmod can use this three-digit string as the expression.
- Using relative permission, we have,
  $chmod a+rw xstart
- Using absolute permission, we have,
  $chmod 666 xstart
  $chmod 644 xstart
  $chmod 761 xstart

- will assign all permissions to the owner, read and write permissions for the group and only execute permission to the others.

- 777 signify all permissions for all categories, but still we can prevent a file from being deleted.

- 000 signifies absence of all permissions for all categories, but still we can delete a file.
- It is the directory permissions that determine whether a file can be deleted or not.
- Only owner can change the file permissions. User cannot change other user's file's permissions.
- But the system administrator can do anything.

The Security Implications

- Let the default permission for the file xstart is

  -rw-r—r- -

  $chmod u-rw, go-r xstart or chmod 000 xstart

- This is simply useless but still the user can delete this file.
- On the other hand,

  $chmod a+rwx xstart or chmod 777 xstart

  -rwxrwxrwx

- The UNIX system by default, never allows this situation as you can never have a secure system. Hence, directory permissions also play a very vital role here .

We can use chmod Recursively.

$chmod -R a+x shell_scripts

- This makes all the files and subdirectories found in the shell_scripts directory, executable by all users. When you know the shell meta characters well, you will appreciate that the * doesn't match filenames beginning with a dot. The dot is generally a safer but note that both commands change the permissions of directories also.

Directory Permissions

- It is possible that a file cannot be accessed even though it has read permission, and can be removed even when it is write protected. The default permissions of a directory are,

  rwxr-xr-x (755)

- A directory must never be writable by group and others .
- Example:

  $mkdir c_progs

  $ls –ld c_progs

  drwxr-xr-x 2 kumar metal 512 may 9 09:57 c_progs

- If a directory has write permission for group and others also, be assured that every user can remove every file in the directory. As a rule, you must not make directories universally writable unless you have definite reasons to do so.

20. Changing File Ownership

- Usually, on BSD and AT&T systems, there are two commands meant to change the ownership of a file or directory. Let kumar be the owner and metal be the group owner. If sharma copies a file of kumar, then sharma will become its owner and he can manipulate the attributes.
- chown changing file owner and chgrp changing group owner
- On BSD, only system administrator can use chown
- On other systems, only the owner can change both

chown

- Changing ownership requires super user permission, so use su command

  $ls -l note
  -rwxr----x 1 kumar metal 347 may 10 20:30 note

  $chown sharma note; ls -l note
  -rwxr----x 1 sharma metal 347 may 10 20:30 note

- Once ownership of the file has been given away to sharma, the user file permissions that previously applied to Kumar now apply to sharma. Thus, Kumar can no longer edit note since there is no write privilege for group and others. He cannot get back the ownership either. But he can copy the file to his own directory, in which case he becomes the owner of the copy.

chgrp

- This command changes the file's group owner. No super user permission is required.

  #ls –l dept.lst

  -rw-r—r-- 1 kumar metal 139 jun 8 16:43 dept.lst

  #chgrp dba dept.lst; ls –l dept.lst
  -rw-r—r-- 1 kumar dba 139 Jun 8 16:43 dept.lst

- In this chapter we considered two important file attributes – permissions and ownership. After we complete the first round of discussions related to files, we will take up the other file attributes.