UNIT - 2
CONCEPT OF FILE & FILE SYSTEMS

- **Introduction to File system**
- **The Basics of Files**
- **What's in a File**
- **Directories and File Names**
- **The Directory Hierarchy**
- **Permissions**
- **I -Nodes**
- **File Attributes and Permissions**
- **The file Command knowing the File Type**
- **The chmod command for Changing File Permissions**
- **The chown command for Changing the Owner of a File**
- **The chgrp command for Changing the GroupOwner of a File.**
- **Important & Frequently asked questions**

1.INTRODUCTION TO UNIX FILE SYSTEM:

Computers can store information on various storage media, such as Magnetic disks, Magnetic tapes, and Optical disks. A file is a named collection of related information that is recorded on secondary storage. File is a memory block of secondary storage device like disk, magnetic tape that logically stores related records permanently. Operating system provides various components to manage the data on a disk system permanently. Such as..

File manager: It manages the allocation of disk-space for the storage of user files.

Buffer manager: It is responsible for fetching data from disk storage into main memory buffers for processing, and then writing the updated data back onto the disk.

Disk Space manager: It is responsible for managing disk space on disk.

File system – provides a powerful & flexible way to organize and manage user information. File system is a Group of files and relevant information are organized in a well defined order and stored on a Hard Disk . Operating System defines a File System on Devices which is usually Hierarchical file system including UNIX.

Directory- It is a **File** that keeps a **list** of other files and sub directories on the File System .

UNIX has single File system. Devices are mounted into this File System. The disk space allotted to a linux/unix file system is made up of "**blocks**" , each of which are **512** bytes.

To find out block size on your file system, use the **cmchk** command.

$cmchk

BSIZE=2048

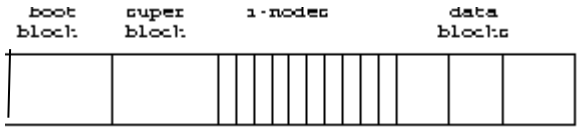


Fig: Disk File Block Structure

1.1.Types of File Blocks on a disk:

All blocks belonging to the file system are logically divided into :

1)Boot block (Master Boot Record)

2)Super block

3)Inode block

4)Data block

Boot block: It is the **first** block in the storage disk, numbered 0 , is called the **boot** block. It is normally unused by the filesystem and is set aside for booting procedure. It represents the beginning of the file system. It contains a program called "**Boost Strap Loader**". This program is executed during '**booting'** process to load the file system into the main memory. (Booting - means starting process of system when computer is turned ON.

Note: All file system contain only **one Boot block**.

Super block: It is the **second** block of every file system and is numbered 1. It describes the **entire file system** and its **state.** It is used to control the allocation of blocks . It contents of super block are:
a)The size of file and status of file name
b)Details of free blocks and their addresses. A file occupies a minimum of 1 Block, even it has one character in it. A block size can be of 512 bytes or multiple of 512 bytes.
c)Size of inode section of the file system.
d)It specifies "how large the file system is , how many maximum files it can accommodate, how many more files it can accommodate, how many more files can create etc."

Inode block: It contains most information regarding the **files**. Every file will have an entry in this area, identified by a **64**-bit structure referred to as I-node. The information related to all the files (not the contents) is stored in an **inode table** on the disk.
For each file, there is an inode entry in the table. Each entry is made up of **64** bytes in the table. These details are..
1) Owner of the file
2) Group to which the owner belongs
3) Type of a file
4) File access permission
5) Date and time of last access
6) Date and time of last modifications
7) No of links to the file
8) Size of the file
9) Address of blocks where the file is physically present.

Data blocks: Contains the actual file contents. An allocated block can belong to only one file in the file system.This block can't be used for storing any other file's contents unless the file which it originally belonged is deleted.

1.2.TYPES OF FILES:

A file in a UNIX system having the following types:
- Regular file / Ordinary file (-)
- Directory file (d)
- Device or Special file (c or b)
- Hidden files or Dot files (.)
- FIFO file (f)
- Link file (l)
- Socket file (s)

1. Ordinary files – An ordinary file is a file on the system that contains data, text, or program instructions.
- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.
- Do not contain other files.

2. Directories – Directories store both special and ordinary files. UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has **two** components.

i)The Filename
ii) A unique identification number for the file or directory (called the inode number)

Features:
- Branching points in the hierarchical tree.
- Used to organize groups of files.

- May contain ordinary files, special files or other directories.
- Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, (named /) located at the top of the tree.
- In long-format output of ls –l , this type of file is specified by the "d" symbol.

3. Device or Special Files – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Ouput (I/O) operations. On UNIX systems there are **two** types of special files for each device, **character** special files and **block** special files.

- **Character special file:** A file related to I/O and used to model serial I/O devices like Terminals ,Printers & Networks.
- **Block Special file:** A file related to memory and used to model devices like Disk drives and Magnetic tapes.

4.**Hidden** files or **Dot** files : A dot(.) character also used to construct a file system. Any file name beginning with a .(dot) character is called a hidden file a dot file. These are used to store some specific information regarding configuration of system. Ex: .profile, .bashrc etc.
5.**FIFO** file : used for making communication between two or more application programs.

2.THE BASICS OF A FILE:

FILE : A File is a sequence of Records or Bytes . A byte is a small chunk of information, typically 8 - bits long. Here, a **byte** is equivalent to a **character**. Data is usually stored in the form of records.

RECORD: It is a collection of Fields. Each record consists of a collection of related data values or items, where each value is formed of one or more bytes.

Files are the **building blocks** of any operating system. Every file has some properties .

The UNIX File Attributes are
1) File type - specifies what type of file it is.
2) Access permission - the file access permission for owner, group and others.
3) Hard link count - number of hard link of the file
4) UID - The file owner/ User ID.
5) GID -The file Group ID .
6) File size - The file size in bytes.
7) I-node no - The system I-node (Information Node) number of the file.
8) File system id - The file system id where the file is stored.
9) Last access time - the time, the file was last accessed.
10) Last modified time - the file, the file was last modified.
11) Last change time - the time, the file was last changed.

Creating A Small File:

The Unix environment allows the user to create a file to maintain their information. The creation of a file requires some editors (such as ed , vi editor) or commands (cat , touch etc.)
// creation of files with cat command
The cat (short for "concatenate") command is one of the most frequently used command in Linux/Unix like operating systems. Cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.
Syntax: cat [options] name
> Redirect the output of the cat command to a file rather than standard output. The destination file will be created if it doesn't exist and will be overwritten if it does.
>> Append the output of the cat command to the end of an existing file. The destination file will be created if it doesn't exist.
| Send (or pipe) the output of the cat command into another command for further processing.

//creation of a file named as 'myfile'

$ **cat** > myfile ....here **myfile** is a file with 15 bytes.

hi how are you.

ctrl+d

// knowing file content

$ **cat** myfile

hi how are you.

//Describing the structure of a file

ls -l myfile

-rwxrwxrwx 1 aliet **engg** 16 Jun 26 16:19 myfile

Data visualization in files:

OD COMMNAD: od is a program/command for displaying ("dumping") data in various human-readable output formats. The name is an acronym for "octal dump" , default it is printed in the octal data format. It can also display output in a variety of other formats, including hexadecimal, decimal, and ASCII. It is useful for visualizing data that is not in a human readable format, like the executable code of a program.

To work on this command, First create one file with numbers before using **od** command.

$ cat numbers

1
2
3
4
5
6
7
8
9
10

Required options are **-b and -c**

-b ----shows the bytes as characters

-c ----shows the bytes as octal numbers

Example 1:

aliet@aliet-cse:~$ **od -c numbers**

output:

0000000 1 \n 2 \n 3 \n 4 \n 5 \n 6 \n 7 \n 8 \n

0000020 9 \n 1 0 \n

0000025

Here, The left side 7 -digit numbers are the **positions** in the file.

$ **od -cb numbers**

output:

0000000 1 \n 2 \n 3 \n 4 \n 5 \n 6 \n 7 \n 8 \n

061 012 062 012 063 012 064 012 065 012 066 012 067 012 070 012

0000020 9 \n 1 0 \n

071 012 061 060 012

0000025

So, we see that output was produced in octal format. The first column in the output of **od** represents the **positions** (7-bit in length) in a file. That is the ordinal number of the next character shown in octal.
Note :od command prints a **visual representation** of all bytes of a file. Possible octal values are..

Character	**octal value**
Backspace	010
\t	011
\n	012
Blank space	140
a to z	141 to 170
A to Z	101 to 126
1	061
2	062
3	063
:	:
:	:

Example 2: Text file
$ od -c myfile

0000000 h i h o w a r e y o u . \n

0000020

$ od -**cb** myfile

0000000 h i h o w a r e y o u . \n

150 151 040 150 157 167 040 141 162 145 040 171 157 165 056 012

0000020

Note: Display the byte offsets in different formats such as hexadecimal, Octal, & decimal using **-A** option
The byte offset can be displayed in any of the following formats :

Hexadecimal (using -x along with -A) . Ex: $ od -Ax -c myfile
Octal (using -o along with -A). Ex: $ od -Ad -c myfile
Decimal (using -d along with -A) Ex: $ od -Ao -c input

Note: when user not to display offset information i.e file positions in a file by using '**-An'** option
$ od -An -c myfile

h i h o w a r e y o u . \n

Note: **od** command with **no option** dumps the file 16-bit , or 2-byte words and makes the magic number visible:
Ex: od /bin/ed ....ed is pure executable file in octal representation
output:
0000000 000410 025000 000462 011444 000000 000000 000000 000001
0000020 170011 016600 000002 005060 177776 010600 162706 000004
0000040 016616 000004 005720 010066 000002 005720 001376 020076
.....

Note: It is possible to copy the data from one file to another with > (output redirection operator)
$od –c myfile > myfile2

$cat myfile //contents of myfile

0000000 h i h o w a r e y o u . \n
0000020
$cat myfile2 //contents of myfile2
0000000 h i h o w a r e y o u . \n
0000020

Buffered output from cat: cat command reads the full file in sequence and store it in buffer and display the buffer 1's its done.
with -u option you can avoid it.
Note: cat command maintains internally a **buffer** to store data .
$ cat
hi
hi
hello
hello

$ cat
hai hai hello hello

Note: here **ctrl+d** is used to stop.
Un-Buffered cat: it uses option **–u.**

$ cat **-u**

123123

456456

3.WHATS'S IN A FILE?

Files are always used for storing data and information. Every file has its own format depends on type of a file. The format of a file is determined by the programs only. There is a wide variety of file types as well programs. Here, file types are not determined by the file system, even the Kernel also can't tell the type of a file is used. It doesn't know it. So, here "**file**" command is used to know it.

Syntax: $file filename

Examples:
$ file myfile **output**: myfile: ASCII text

$ file /bin **output**: /bin: directory

$ file /home **output**: /home: directory

$ file Desktop **output**: Desktop: directory

$ file Documents **output**: Documents: directory

$ file / **output**: /: directory

$ file /root **output** : /root: directory

$ file /etc/passwd output: /etc/passwd: ASCII text

$ file /usr/src output: /usr/src: directory

$file /bin /bin/ed /usr/src/cmd/ed.c /usr/man/man1/ed.1 ...multiple inputs are given here

output:

/bin: directory

/bin/ed: pure executable ..../bin directory maintains all executable files (in binary format)

/usr/src/cmd/ed.c: c program text

/usr/amn/man1ed.1: manual text file

4.Directories and File Names

A filename can be any sequence of ASCII characters without special characters like < , > . A file name starts with a period(.) is a hidden file, generally used by a UNIX utility.

Ex: .profile, .bashrc, .mailrc and .cshrc.

Simple rules:

1)Start your names with an alphabetical order.

2)Use dividers to separate parts of the name. (use underscore,period, and the hyphen)

3)Use an extension at the end of the filename, even though UNIX doesn't recognize extensions.

4)Never start a filename with a period. File names that start with a period are known as "hidden files" in UNIX.Generally , hidden files are created and used by the system .

5)All files have unambiguous (different) names ,starting with proper order , for example /usr/mydir/myfile.txt. because more than one user may use the same name to a file.

If you want to know the files in your current directory, use **ls commad**.

$ls

myfile

$

Each running program/file (i.e a process) has a current directory. All file names are implicitly assumed to start with the name of that directory ,unless they begin directly with a slash.

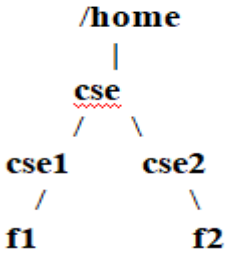
Here, user login shell & ls have a current directory. To get the current directory, use **pwd** (present/print working directory) command is used. pwd identifies the current directory.

$ pwd

/home/aliet

The current directory is an attribute of a process, not a person or a program. People have login directories, processes have current directories. If a child process creates a child process , the child inherits the current directory of its parent. But if the child then changes to a new directory , the parent is unaffected .its current directory remains the same .

Consider a **simple** directory structure and its **creation**.



$pwd

$ mkdir cse //creation of new directory

$ cd cse //changed to newly created directory.

...../cse$ **mkdir** cse1 cse2 //Creation of two sub/child directories.

.....cse$ **cd** cse1 //changed to newly created directory

$pwd

/home/cse/cse1

...../cse/cse1$ **cat >f1**
hii this is directory structure
ctrl+d
...../cse/cse1$ cat f1
hii this is directory structure
...../cse/cse1$ ls
f1

...../cse/cse1$ cd .. //back to its parent directory
...../cse/$cd cse2
...../cse/cse2/cat > f2
This file for CSE2
...../cse/cse2/ls
f2
...../cse/cse2$ cd .. //back to its parent directory
...../cse$
$ ls cse //Listing of two subdirectories in CSE directory.
cse1 cse2

$ ls cse/cse1 //Listing of files in cse/cse1 directory.

f1

DU COMMAND : disk usage command reports usage of disk by a recursive examination of the directory time. i.e consumption of disk space by both directory & files.

$ cd cse

.../cse$ cd cse1

../cse/cse1$ du

8 .

....cse/cse1$ cd ..

.../cse$ **du**

8 ./cse1

4 ./cse2

16 .

....cse$ du –a // to get all files & its used block size.

4 ./cse1/f1

8 ./cse1

4 ./cse2

16 .

usage of **du** with **grep** command:

$du –a | grep myfile
4

grep command: Global Regular Expression Print .

It is used to **finding patterns** in files. This command searches one or more files for text , matches a pattern in the given file. It is also called as **Filter** command.
-used to extract information from files.
-To search the output of a command for lines related to a particular item.
-To locate files containing a particular keyword.
Syntax: grep "pattern" filename
Ex: $ grep "hi" myfile

hi how are you.

$ grep -i "Hi" myfile //ignoring the case

hi how are you.

$ **du -a | grep** myfile // disk usage

4 ./myfile.bak

4 ./myfile

$ **grep** cse **/etc/passwd** // searching for cse in /etc/passwd file.

cse:x:1001:1002::/home/cse:

cse1:x:1002:1003::/home/cse1:

DF COMMAND(DISK FREE) : df command (disk free) tells the amount of free space available on the disk. The 'df' command also stands for "disk filesystem", it is used to get full summary of available and used disk space usage of file system on Linux system.

$df //for entire filesystem
output:

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/cciss/c0d0p2	78361192	23185840	51130588	32%	/
/dev/cciss/c0d0p5	24797380	22273432	1243972	95%	/home
/dev/cciss/c0d0p3	29753588	25503792	2713984	91%	/data
/dev/cciss/c0d0p1	295561	21531	258770	8%	/boot
tmpfs	257476	0	257476	0%	/dev/shm

$ df -h //human readable format

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/cciss/c0d0p2	75G	23G	49G	32%	/
/dev/cciss/c0d0p5	24G	22G	1.2G	95%	/home
/dev/cciss/c0d0p3	29G	25G	2.6G	91%	/data
/dev/cciss/c0d0p1	289M	22M	253M	8%	/boot
tmpfs	252M	0	252M	0%	/dev/shm

$ df -t ex3 //for specific file

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/cciss/c0d0p2	78361192	23190072	51126356	32%	/
/dev/cciss/c0d0p5	24797380	22273432	1243972	95%	/home
/dev/cciss/c0d0p3	29753588	25503792	2713984	91%	/data
/dev/cciss/c0d0p1	295561	21531	258770	8%	/boot

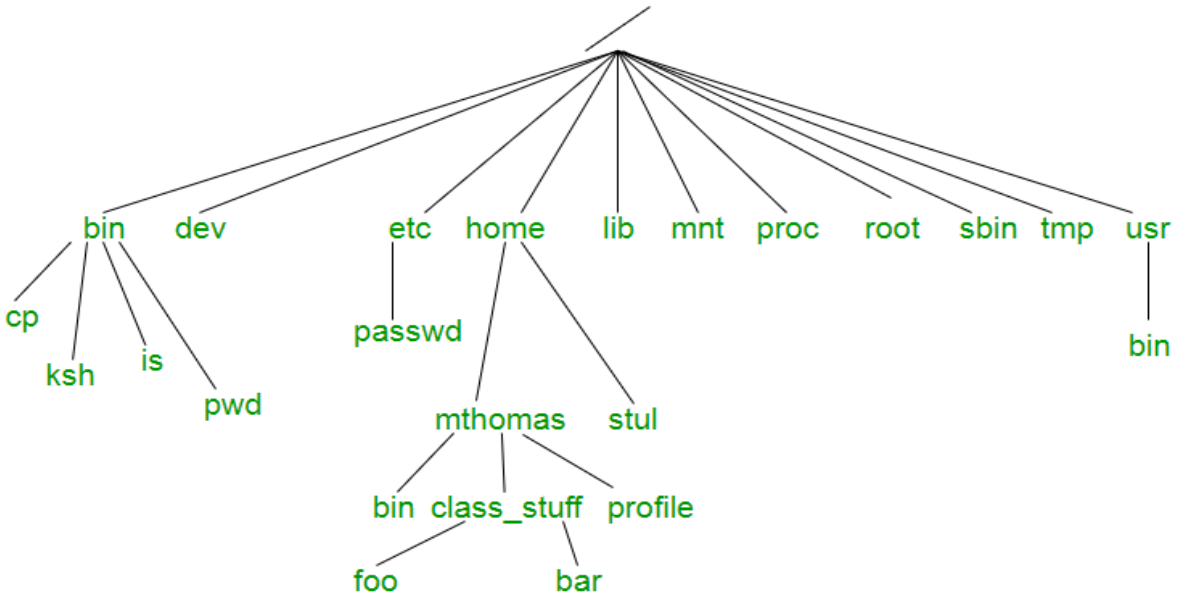
5.UNIX FILE SYSTEM / DIRECTORY STRUCURE:

The Unix file system is essentially composed of files and **directories**. Directories are special files that may contain other files.

Note: Everything in the unix system is called a file.

Unix file system is a methodology (or) technique (or) logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.



The following system files (i.e. directories) are present in most Unix filesystems:

- ***/bin*** - short for binaries, this is the directory where many commonly used executable commands reside.(such as cat,rm cp etc.)
- ***/dev*** - contains device specific files
- ***/etc*** - contains system configuration files

  -Stores system administrative files and programs.

  /etc/passwd – Stores all user information.
  /etc/shadow – Stores user passwords
  /etc/group – Stores all group information
- ***/home*** - contains user directories and files. It is for all of the system's users.
- ***/lib*** - contains all library files. Essential system library files used by tools in '/bin'
- ***/mnt*** - contains device files related to mounted devices

- ***/proc*** - contains files related to system processes
- ***/root*** - the root users' home directory (note this is different than /)
- ***/sbin*** - system binary files reside here. If there is no sbin directory on your system, these files most likely reside in 'etc'
- ***/tmp*** - storage for temporary files created by programs , which are periodically removed from the filesystem
- ***/usr*** - also contains executable commands. Sub directories with files related to user tools & applications.

  /usr/include – stores standard header files .

  /usr/lib- stores standard library files .

Files are indentified by **path names**

• Files must be created before they can be used.

Path name:

In unix file system structure , each node is either a file or a directory of files, where we can maintain files and directories. It specifies a file or directory by its path name. There are TWO Path names are there.

1) *FULL, or ABSOLUTE path name*
2) *RELATIVE path name*

Full /absolute path name : It starts with the root(/), and follows the branches of the file system, each separated by /, until you reach the desired file, Example:

/home/usr/cse/sec/std.txt

Relative path name : it specifies the path relative to another, usually the current working directory to a specified file ,Example:
cse/sec/std.txt

Two special directories used in file system for moving from one level of directories to another level of directories.

. the current directory
.. the parent of the current directory .

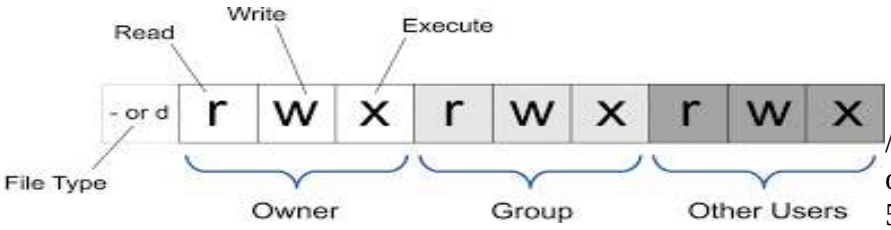
Note: To get the directory hierarchy , use
$ ls /
bin
boot
dev
etc
lib
tmp
usr
Note:To get hierarchical structure of file system use
$man hier

6.Permissions:

The UNIX file system is designed to support multiple users.When many users are sharing one system,it is imortant to be able restrict access to certain foiles.The system administarator wants to prevent other users from changing important **system** files.

Permissions for files:

there are three of file permissions for the three classes of users: The owner(or user) of the file , The group the file belongs to and all other users of the system.To know the permissions of a file, use ls -l command .



Ex: ls -l /etc/passwd only
..... for **aliet** user
-rw-r--r-- 1 **aliet** 5115 jun 25 9:40 /etc/passwd
Ex:ls -lg /etc/passwd ..... for **group admin** user
-rw-r--r-- 1 **adm** 5115 jun 25 9.45 /etc/passwd
Ex:ls -ld . ..... for **directory** file
drwxrwxr-x 3 cse 80 jun 25 9:45

Permissions for Directories:
Similar to files, Directories also have permissions. For directories, **read** permission allows users to list the contents of the directory. **Write** permission allows users to create or remove files or directories inside that directory , and **execute** permission allows users to change to this directory using the **cd** command .

PERMISSION	**FILE**	**DIRECTORY**
READ	User can open and Read the content of a file	User can list files present in directory and cannot read files of directory
WRITE	User can modify contents of file	User can add or delete files to directory
EXECUTE	User can run executable files	User can use **cd** command and can go through the directory

7. I-node:

The **inode** is a data structure in a Unix-style file system that describes a filesystem object such as a file or a directory.Every file have a unique **inode** to maintain information about the file except name & its data. Each inode stores the attributes and disk block location(s) of the object's data. File system object attributes may include metadata (times of last change, access, modification), as well as owner and permission data.

Each file is associated with an *inode*, which is identified by an integer number, often referred to as an *i-number* or *inode number*.

Inodes store information about files and directories (folders), such as file ownership, access mode (read, write, execute permissions), and file type.

The inode number indexes a table of inodes in a known location on the device. From the inode number, the kernel's file system driver can access the inode contents, including the location of the file - thus allowing access to the file.

A file's inode number can be found using the `ls -i` command. The `ls -i` command prints the i-node number in the **first column** of the report.

To get I node of a file named f5 is

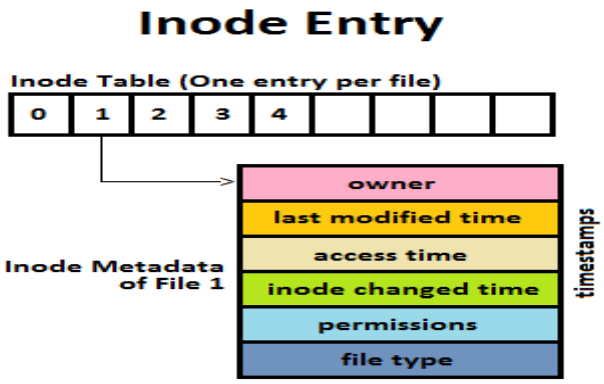
$ ls – i f5

2100210 -rw-rw-r-- 1 aliet aliet 25 Jun 26 17:04 f5

An inode of a file contains....

1) Owner of the file
2) Group to which the owner belongs
3) Type of a file
4) File access permission

5) Date and time of last access
6) Date and time of last modifications
7) No of links to the file
8) Size of the file
9) Address of blocks where the file is physically present.



Ln(link) command: To create a link between two files. This command is used to creates a new name for an existing file.
Syntax: ln [options] file1 file2
Here , changes to one file are also reflected in the other file.
Types of link methods: 1)Hard link
2)Soft link / Symbolic link
Hard link: It is default link to any file. It is a UNIX path name for a file

Syntax: $ **ln** file1 file2 //here file2 should not exist in filesysytem.i.e new file.
Ex: $**ln** f2 f5
To know the **status of a link** (along with a **long list** of information) whether it is created or not. Use ls -li
$ ls **-li** f2 f5

2100210 -rw-rw-r-- 1 aliet aliet 25 Jun 26 17:04 f2

2100210 lrwxrwxrwx 2 aliet aliet 2 Jun 26 17:09 f5

When we work on **ln** command, two linked files must have the same inode number. Otherwise ,those two files are not have a link to each other.
Soft link or symbolic link:
To create symbolic link, ln command is used with option –s
$ ln -s f2 f7

$ ls -li f2 f7

2100210 -rw-rw-r-- 2 aliet aliet 25 Jun 26 17:04 f2

2100214 lrwxrwxrwx 1 aliet aliet 2 Jun 26 17:09 **f7-> f2**

Unlink : it removes or eliminates a link in between files(i.e existing link)

Syntax: unlink file1 file2
Ex: unlink f2 f7

Difference : hard link and symbolic link

Hard link	**Symbolic link**
Does not create a new inode	Create a new inode

Cannot link directories unless it is done by root	Can link directories
Cannot link files across file systems	Can link files across file system
Increase hard link count of the linked inode	Does not change hard link count of the linked inode

Note: ln command works like **cp** command, but there is a difference in that.
Difference : cp and ln command
Cp command creates a duplicated copy of file to another file with a different path name.
Where as, **ln** command saves space by not duplicating the copy here the new file will have same inode number as original file.

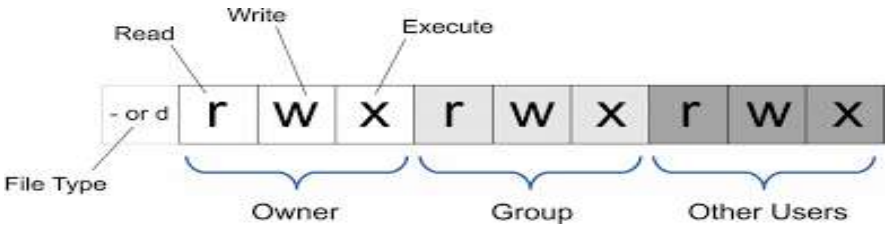
8.File Attributes and Permissions:

File Attributes:
Files are the **building blocks** of any operating system. Every file has some properties . Properties are nothing but attributes of a file.

The UNIX File Attributes are
1) File type - specifies what type of file it is.
2) Access permission - the file access permission for owner, group and others.
3) Hard link count - number of hard link of the file
4) UID - The file owner/ User ID.
5) GID -The file Group ID .
6) File size - The file size in bytes.
7) I-node no - The system I-node (Information Node) number of the file.
8) File system id - The file system id where the file is stored.
9) Last access time - the time, the file was last accessed.
10) Last modified time - the file, the file was last modified.
11) Last change time - the time, the file was last changed.

Permissions:

The UNIX file system is designed to support multiple users. When many users are sharing one system, it is important to be able restrict access to certain foiles. The system administrator wants to prevent other users from changing important **system** files.
Permissions for files:
there are three of file permissions for the three classes of users: The owner(or user) of the file , The group the file belongs to and all other users of the system.To know the permissions of a file, use ls -l command .



Ex: ls -l /etc/passwd ..... for **aliet** user only
-rw-r--r-- 1 **aliet** 5115 jun 25 9:40 /etc/passwd
Ex:ls -lg /etc/passwd ..... for **group admin** user
-rw-r--r-- 1 **adm** 5115 jun 25 9.45 /etc/passwd
Ex:ls -ld . ..... for **directory** file
drwxrwxr-x 3 cse 80 jun 25 9:45

Permissions for Directories:
Similar to files, Directories also have permissions. For directories, **read** permission allows users to list the contents of the directory. **Write** permission allows users to create or remove files or directories inside that

directory , and **execute** permission allows users to change to this directory using the **cd** command .

PERMISSION	FILE	DIRECTORY
READ	User can open and Read the content of a file	User can list files present in directory and cannot read files of directory
WRITE	User can modify contents of file	User can add or delete files to directory
EXECUTE	User can run executable files	User can use **cd** command and can go through the directory

9.The File Command knowing the File Type:

In Unix, there is a wide variety of file types. Here, file types are not determined by the file system, even the Kernel also can't tell the type of a file is used. It doesn't know it. So, here "**file**" command is used to know it.

Syntax: $file filename

Examples:
$ file myfile **output**: myfile: ASCII text

$ file /bin **output**: /bin: directory

$ file /home **output**: /home: directory

$ file Desktop **output**: Desktop: directory

$ file Documents **output**: Documents: directory

$ file / **output**: /: directory

$ file /root **output** : /root: directory

$ file /etc/passwd **output**: /etc/passwd: ASCII text

$ file /usr/src **output**: /usr/src: directory
$file /bin /bin/ed /usr/src/cmd/ed.c /usr/man/man1/ed.1 ...multiple inputs here
output:
/bin: directory
/bin/ed: pure executable ..../bin directory maintains all executable files (in binary format)
/usr/src/cmd/ed.c: c program text
/usr/amn/man1ed.1: manual text file

type command **:**

In Linux/unix system, **type** command is used for displaying information about command type. It displays if command is an alias,shell function, shell builtin, disk file, or shell reserved word.

You can use type command with other command names also.

Note: The type command will show output only when the command/executable name is available or installed in Linux System.

type [options] name [name ...]

Description:

-t option :
It will show a single word or string.It tells a command or name is an alias, shell reserved word, function,

builtin, or disk file(external command).

$ **type -t ls** **output:** alias
$ **type -t date** **output:** file
$ **type -t declare** **output:**builtin

-p option :

The command of the disk file that would be executed. In other words, absolute path of command.

$ **type -p date** **output:** /bin/date

-P option :

Force a PATH search for each command/NAME, even if it is an alias,builtin, or function, and returns the name of the disk file that would be executed.
(We can get PATH environment value by using command echo $PATH)

$ **type -P date** **output:**/bin/date

NOTE: In below example, date command was searched in all directories listed in PATH(Environment set). And found the date command in /bin

-a option :

It display all locations have command or NAME .
It includes aliases, builtins, and functions (only in case , if -p option is not used)

$ **type -a date** **output:**date is /bin/date
$ **type -a echo** **output:** echo is a shell builtin

f option :

The -f option, suppress shell function lookup.

$ **type -f echo** **output:** echo is a shell builtin
$ **type -f date** **output:** date is /bin/date

10.The Chmod Command for Changing File Permissions:

Chmod command: **CHANGING MODE OF A FILE**

Chmod is used to change permissions on files and directories. The command chmod may be used with either letters(also known as symbolic) or numbers (also known as octal) to set the permissions. The letters used with chmod are in the table below:

Letter	permission/usage
r	Read
w	Write
x	Execute
X	Execute (only if a directory)
u	User or Owner of the file
g	Group User
o	Other user
a	All users
+	Grant permissions
-	Revoke permissions
=	Set permissions

Read, Write & Execute Permissions:

Permissions are the “rights” to act on a file or directory. The basic rights are read, write, and execute.

- **Read** - a readable permission allows the contents of the file to be viewed. A read permission on a directory allows you to list the contents of a directory.

- **Write** - a write permission on a file allows you to modify the contents of that file. For a directory, the write permission allows you to edit the contents of a directory (e.g. add/delete files).

- **Execute** - for a file, the executable permission allows you to run the file and execute a program or script. For a directory, the execute permission allows you to change to a different directory and make it your current working directory. Users usually have a default group, but they may belong to several additional groups.

Viewing File Permissions :

To view the permissions on a file or directory, issue the command ls -l <directory/file>. Remember to replace the information in the < > with the actual file or directory name. Below is sample output for the ls command:

-rw-r--r-- 1 root root 1031 Nov 18 09:22 /etc/passwd

The **first ten** characters show the access permissions. The first dash (-) indicates the type of file (d for directory, s for special file, and - for a regular file). The next three characters (**rw-**) define the owner's permission to the file. In this example, the file owner has read and write permissions only. The next three characters (**r--**) are the permissions for the members of the same group as the file owner (which in this example is read only). The last three characters (**r--**) show the permissions for all other users and in this example it is read only.

It is important to remember that the first character of the first column of a file listing denotes whether it is a directory or a file. The other nine characters are the permissions for the file/directory.
The example **drwxrw-r–** is broken down as follows:
d is a directory
rwx the user has read, write, and execute permissions
rw- the group has read and write permissions
r– all others have read only permissions
Note that the dash (-) denotes permissions are removed. Therefore, with the "all others" group, r– translates to read permission only, the write and execute permissions were removed.

Conversely, the plus sign (+) is equivalent to granting permissions: chmod u+r , g+x <filename>
In other words, the user was given read permission and the group was given execute permission for the file. Note, when setting multiple permissions for a set, a comma is required between sets.

Chmod Octal Format:

To use the octal format, you have to calculate the permissions for each portion of the file or directory. The first ten characters mentioned above will correspond to a four digit numbers in octal. The execute permission is equal to the number one (1), the write permission is equal to the number two (2), and the read permission is equal to the number four (4). Therefore, when you use the octal format, you will need to calculate a number between 0 and 7 for each portion of the permission. A table has been provided below for clarification.

syntax: chmod <permissions> <file/directory name>

Example: chmod 777 myfile ------ all permissions to all users

```
$ ls –l myfile      (or)
-rwxrwxrwx  1  aliet aliet 40  2018-06-18 14:09 myfile

$ chmod   a+rwx  myfile
$ ls –l myfile      (or)
-rwxrwxrwx  1  aliet aliet 40  2018-06-18 14:09 myfile

$ chmod   ugo+rwx  myfile        (or  ugo=r+w+x  or  ugo=rwx)
$ ls –l myfile
```

-rwxrwxrwx 1 aliet aliet 40 2018-06-18 14:09 myfile

chmod 541 myfile ------ read& execute for user, read only for group , execute for others.

$ ls –l myfile
-r-xr----x 1 aliet aliet 40 2018-06-18 14:09 myfile (or)

chmod u+rx g+r u+x myfile ------ read& execute for user, read only for group , execute for others.

$ ls –l myfile
-r-xr----x 1 aliet aliet 40 2018-06-18 14:09 myfile (or)

chmod u-w g-wx u-rw myfile ------ read& execute for user, read only for group , execute for others.

$ ls –l myfile
-r-xr----x 1 aliet aliet 40 2018-06-18 14:09 myfile

An octal table showing the numeric equivalent for permissions is provided below.

Permissions	Binary	Octal	Description
---	000	0	No permissions
--x	001	1	Execute-only permission
-w-	010	2	Write-only permission
-wx	011	3	Write and execute permissions
r--	100	4	Read-only permission
r-x	101	5	Read and execute permissions
rw-	110	6	Read and write permissions
rwx	111	7	Read, write, and execute permissions

Permission string	Octal code	Meaning
rwxrwxrwx	777	Read, write, and execute permissions for all users.
rwxr-xr-x	755	Read and execute permission for all users. The file's owner also has write permission.
rwxr-x---	750	Read and execute permission for the owner and group. The file's owner also has write permission. Users who aren't the file's owner or members of the group have no access to the file.
rwx------	700	Read, write, and execute permissions for the file's owner only; all others have no access.
rw-rw-rw-	666	Read and write permissions for all users. No execute permissions for anybody.
rw-rw-r--	664	Read and write permissions for the owner and group. Read-only permission for all others.
rw-rw----	660	Read and write permissions for the owner and group. No world permissions.
rw-r--r--	644	Read and write permissions for the owner. Read-only permission for all others.
rw-r-----	640	Read and write permissions for the owner, and read-only permission for the group. No permission for others.
rw-------	600	Read and write permissions for the owner. No permission for anybody else.
r--------	400	Read permission for the owner. No permission for anybody else.

Example: To enable execution permission for the file “myfile” by all is

Chmod ugo+x myfile (or)
Chmod +x myfile (or)
Chmod a+x myfile

Important commands:
1)dir Command:
It works like Unix/Linux **ls** command, it lists the contents of a directory.

$ dir
508
cse

2)groups command :
groups command displays all the names of groups a user is a part of like this.
$ groups

aliet adm cdrom sudo dip plugdev lpadmin sambashare

3)id command : shows user and group information for the current user or specified username .
$ id

uid=1000(aliet) gid=1000(aliet)
groups=1000(aliet),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare)

4)grep:

It is a search tool on Unix-like systems which can be used to search for anything whether it be a file, or a line or multiple lines in file is grep utility. It is very vast in functionality which can be attributed to the large number of options it supports like: searching using string pattern, or regular expression pattern .
Syntax: grep “pattern” filename
$ grep "hi" myfile

hi how are you.

$ grep -i "Hi" myfile

hi how are you.

$ sudo du -a | grep myfile

[sudo] password for aliet:

4 ./myfile.bak

4 ./myfile

$ grep cse /etc/passwd //to search for cse in /etc/passwd

cse:x:1001:1002::/home/cse:

cse1:x:1002:1003::/home/cse1:

ls -l /etc/passwd

-rw-r--r-- 1 root root 2392 Jun 26 15:56 /etc/passwd

11.The Chown Command Changing the Owner of a File i.e CHANGE OF OWNERSHIP OF A FILE.

chown command : changes/updates the user and group ownership of a file/directory
Before that ,we need to know "how to add new users in to a system?"
1) Adding new users
syntax: $ **sudo useradd** username
Example:
$ **sudo useradd user1** .....add a new user
[sudo] password for aliet: .....enter system admin password
aliet@aliet-desktop:~$ **sudo passwd user1**
.....setting a password , here username and password are the same
Enter new UNIX password:
****
Retype new UNIX password:
****
passwd: password updated successfully

After adding of new user, you need to change that account by using **su** command.
su : Switch to user
$ **su user1**
.....switch to user1 using su command
Password:
****
user1@aliet-desktop:/home/aliet$ //now the user in user1 mode

Now , again go back to aliet user account, where you can apply **chown** command
Syntax: chown newusername filename

Ex:$ **sudo chown user1 modefile**

[sudo] password for aliet:
****
Now, the file has been changed from aliet to user1 account. Now , again login into user1 account to change users permissions. by using chmod command.

$ **su user1**

Password:
****
user1@aliet-desktop:/home/aliet$ **cat modefile** **//now the user in user1 node.**

hi how are you...

user1@aliet-desktop:/home/aliet$ **chmod 711 modefile**

user1@aliet-desktop:/home/aliet$ **ls -l modefile**

-rwx--x--x 1 user1 aliet 18 Jun 21 11:16 modefile

user1@aliet-desktop:/home/aliet$ **cat >> modefile**

new text is added here

user1@aliet-desktop:/home/aliet$ **cat modefile**

hi how are you...

new text is added here

12.The Chgrp Command Changing the Group of a File:

chgrp command: Like chown, this command changes the group ownership of a file.
It can be used by the owner of the file, and the user belongs to the same group can only change the group ownership of a file to another group.

This command provide the **new group name** as its **first** argument and the **name of file** as the **second** argument.

Groups command : groups command displays all the names of groups a user is a part of like this.

The **adduser** and **addgroup** commands are used to add a user and group to the system respectively according to the default configuration specified in **/etc/adduser.conf** file.

Syntax: groupadd groupname

$ **sudo** groupadd engg //creation of group named as "engg"

[sudo] password for aliet:

$ sudo useradd -m cse1
$ sudo passwd cse1

Enter new UNIX password:
****
Retype new UNIX password:
****
passwd: password updated successfully

$ sudo **userdmod** -a -G engg cse1 // add a user to agroup
$ cat >myfile

hi this is for change group user

After creation of new group, you need to create a file and chanage the permissions.

$cat myfile
hi this is for change group user

$ ls -l myfile

-rw-rw-r-- 1 aliet aliet 34 Jun 26 16:05 myfile

$ chmod 777 myfile

$ ls -l myfile
-rwxrwxrwx 1 aliet aliet 34 Jun 26 16:05 myfile

Now, user need to use **chgrp** command to the change the group owner of the file.
Syntax: chgrp newgroupname filename
Ex: $ sudo **chgrp** engg myfile

$ ls -l myfile

-rwxrwxrwx 1 aliet engg 34 Jun 26 16:05 myfile

$ cat >myfile

hi how are you.

$cat myfile

hi how are you.

$ ls -l myfile

-rwxrwxrwx 1 aliet **engg** 16 Jun 26 16:19 myfile

****

13.Important & frequently asked questions

1) What is a file system? Explain Unix file system with neat diagram

2) What is a file ? Explain its attributes and permissions with examples

3) Write short notes on Inodes.

4) Explain briefly about file permissions with examples

5) Mention and Explain the commands used for changing the owner of a file as well as the group of a file.

6)Illustrate file directories with neat diagram and explain **Directory handling commands** supported by Unix. (pwd, mkdir, cd, rmdir)

7) Explain the basics of file, directories and file names ?

8)Give a short note on file and mention various types of file and explain **File handling commands** (cat, rm, cp, mv, wc)

9)Briefly explain how to change file permissions, Changing the owner of a file ,Changing the group of a file.

10)Briefly explain about **ls** command with all options and examples.

11)Explain about the file command-knowing the file type.