CHAPTER 4: **File attributes**

File and directory attributes listing and very brief idea about the attributes, File ownership, File permissions, Changing file permissions – relative/symbolic permission & absolute permission, Changing file ownership, Changing group ownership,

File system and inodes, Hard link, Soft link, Significance of file attribute for directory, Default

permissions of file and directory and using umask, Listing of modification and access time, Time stamp changing (touch), File locating (find)

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.

COMMAND to display file attributes:

,•ls–l to display file attributes (properties)

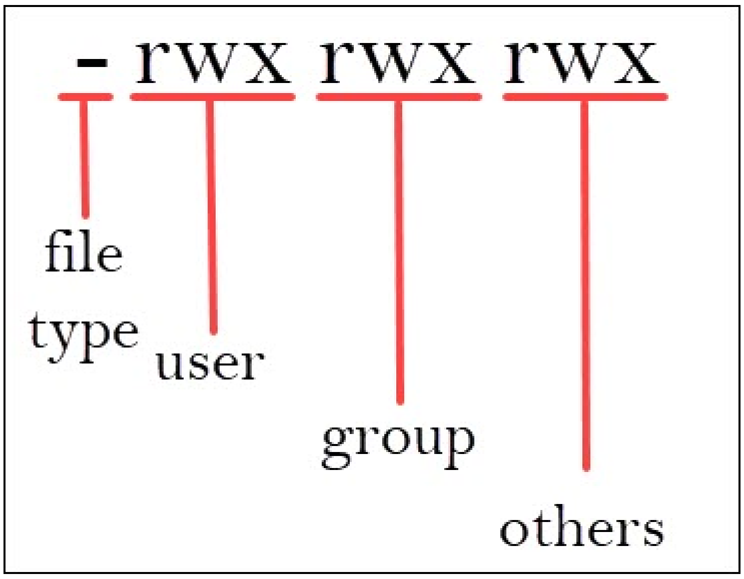
•Listing of a specific directory

•Ownership and group ownership

•Different file permissions

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:

•(1)File type and Permissions



File type. There are three possibilities for the type. It can either be a regular file (**–**), a directory (**d**) or a link (**i**).

•(ii)Links

•(iii)Ownership

•(iv)Group ownership

•(v)File size

•(vi)Last Modification date and time

•(vii)File name

The file type and its permissions are associated with each file.

Links indicate the number of file names maintained by the system. This does not mean that there are so many copies of the file.

File is created by the owner.

Every user is attached to a group owner.

File size in bytes is displayed.

Last modification time is the next field. If you change only the permissions or ownership of the file, the modification time remains unchanged.

In the last field, it displays the file name.

COMMAND NAME: ls

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

-a will list all the **hidden files** as well as the normal ones

-i, --inode

print the **index number** of each file

-r, --reverse

reverse order while sorting

-R, --recursive

```
            list subdirectories recursively

-s, --size      print the allocated size of each file, in blocks

-S     sort by file size, largest first

-t     sort by modification time, newest first

-1     list one file per line.

--block-size=SIZE

            with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M';

The  SIZE argument is an integer and optional unit (example: 10K is 10*1024).  Units

 are K,M,G,T,P,E,Z,Y (powers of 1024) or KB,MB,... (powers of 1000).
```

Kilo(K), mega(M), giga(G), tera(T), peta(P), exa(E), zetta(Z),yolta(Y).

Example:To create a **hidden file** use . with filename

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ vi .a.txt
```

To display hidden files use ls -a or ls -la command:

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ ls -la

total 12

-rw--w--w- 1 debasis debasis    15 Jan 15 20:39 .a.txt

-rw-r--r-- 1 debasis debasis 10240 Jan 13 12:14 archiveEx.tar
```

File Permissions:

Chmod (change mode): permission sets

Chmod (change mode) is one of the most frequently used commands in unix or linux operating system. The chmod command is used to change the file or directory access permissions.
To know about the access permissions of a file or directory, use the ls -l command as shown below:

```
$ ls -l  filename.sh
-rwx-rw-r—1 student student 94 Oct  4 03:12 filename.sh
```

The **syntax** of chmod command is:

chmod [options] mode/permissions filename

permissions defines the permissions for the **owner of the file (the "user"), members of the group who owns the file (the "group")**, and **anyone else ("others").**

Changing File Permissions:

The **chmod** command enables you to change the permissions on a file. You must be superuser or the owner of a file or directory to change its permissions.

You can use the **chmod** command to set permissions in either of two modes:

- **Relative/Symbolic permissions** – Use combinations of letters and symbols to add or remove permissions(with symbols (alphanumeric characters)).In relative permissions chmod only changes the permissions in the command line and leaves the **other permissions unchanged**.

- **Absolute permissions** – Use numbers to represent file permissions (the method most commonly used to set permissions). When you change permissions by using the absolute mode, you represent permissions for each triplet by an octal mode number(with octal numbers (the digits **0** through **7**).

Relative/Symbolic Mode Representation of Permissions:

The following symbols are used to represent the users, groups and others:

- u : User
- g : Group
- o : Others
- a : All (user, group and others)

The following symbols represent the permissions:

- r : read

- w : write
- x : execute

The following symbols represent the permissions grant or revoke:

- + : Additional permissions. Selected permissions are added.
- - : Revoke the permissions. Selected permissions are revoked.
- = : Specific permissions. Only selected permissions are assigned.

Let's say you are the owner of a file named **filename.txt**, and you want to set its permissions so that:

1. the **u**ser can **r**ead, **w**rite, and e**x**ecute it;
2. members of your **g**roup can **r**ead and e**x**ecute it; and
3. **o**thers may only **r**ead it.

This command will do the trick:

$ Chmod ugo+rwx filename.txt

$ Chmod ug-wx filename.txt

```
chmod u=rwx,g=rx,o=r filename.txt
```

This example uses symbolic permissions notation. The letters **u**, **g**, and **o** stand for "**user**", "**group**", and "**other**". The equals sign ("**=**") means "set the permissions exactly like this," and the letters "**r**", "**w**", and "**x**" stand for "read", "write", and "execute", respectively. The commas separate the different classes of permissions, and there are no spaces in between them.

Absolute Mode Representation of Permissions:

Here is the equivalent command using octal permissions notation:

```
chmod 754 filename.txt
```

Here the digits **7**, **5**, and **4** each individually represent the permissions for the user, group, and others, in that order. Each digit is a combination of the numbers **4**, **2**, **1**, and **0**:

- **7** stands for "read,write and

  execute"(1 1 1 =7)
- **6 stands for(read,write)**

  (110=6)
- **4 stands for "read"(100=4)**
- **2** stands for "write",(010=2)
- **1** stands for "execute"(001=1),

  and
- **0** stands for "no

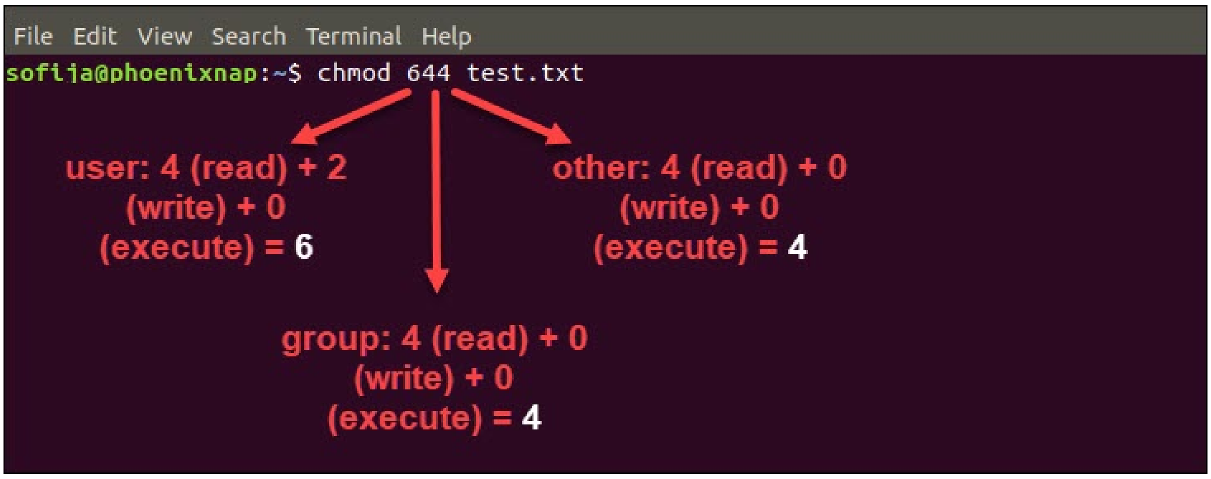
  permission."(000=0)

So **7** is the combination of permissions **4+2+1** (read, write, and execute), **5** is **4+0+1** (read, no write, and execute), and **4** is **4+0+0** (read, no write, and no execute).

To make a file and run it, use the following commands :

$chmod u+x filename (set file as exectable)

$./filename (To execute shell program or Run)


File Edit View Search Terminal Help

```
sofija@phoenixnap:~$ chmod 644 test.txt
```

user: 4 (read) + 2 (write) + 0 (execute) = 6
other: 4 (read) + 0 (write) + 0 (execute) = 4
group: 4 (read) + 0 (write) + 0 (execute) = 4

UNIX/Linux File Ownership:

Every Linux system have three types of owner:

1. **User:** A user is the one who created the file. By default, whosoever, creates the file becomes the owner of the file. A user can create, delete, or modify the file.
2. **Group:** A group can contain multiple users. All the users belonging to a group have same access permission for a file.
3. **Other:** Any one who has access to the file other than **user** and **group** comes in the category of **other**. Other has neither created the file nor is a group member.

COMMAND NAME : id

id - print real and effective user and group IDs

```
debasis@LAPTOP-H3N6JCNE:~$ id

output:

uid=1000(debasis) gid=1000(debasis)
groups=1000(debasis),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),
29(audio),30(dip),44(video),46(plugdev),117(netdev)
```

UID:

A **UID (user identifier)** is a number assigned by Linux to each user on the system. This number is used to identify the user to the system and to determine which system resources the user can access.

- UID 0 (zero) is reserved for the root.
- UIDs 1–99 are reserved for other predefined accounts.
- UID 100–999 are reserved by system for administrative and system accounts/groups.
- UID 1000–10000 are occupied by applications account.
- UID 10000+ are used for user accounts.

GID:

Groups in Linux are defined by GIDs (group IDs).

- GID 0 (zero) is reserved for the root group.
- GID 1–99 are reserved for the system and application use.
- GID 100+ allocated for the user's group.

Users and groups can be locally managed in **/etc/psswd** or **/etc/group.**

Listing User Accounts

To know the local users account, following command can be used. It list out all the local users from the system.

Example:

```
$ cat /etc/passwd | cut -d ":" -f1 | more
```

UNIX/Linux chgrp: change group:

The chgrp command can be abbreviated as **change group**. You can change the group owner of the file using chgrp command.

NAME

chgrp - change group ownership

Syntax:

chgrp **<newGroupname> <fileName>**

OR: To change the **group ownership** type in the following command:

```
chgrp [group_name] [file_name]
```

Example:

```
#change the group ownership from student to games:

student# chgrp games m1.txt
```

Display/List all group name:

COMMAND NAME:

groups - print the groups a user is in

debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ **groups**

output:

```
debasis adm dialout cdrom floppy sudo audio dip video plugdev netdev
```

```
#change the group name of the file:
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ sudo chgrp floppy m1.txt
[sudo] password for debasis:
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l m1.txt
ouput:
-rw-r--r-- 1 debasis floppy 21 Jan 14 17:58 m1.txt
```

Linux chown: change owner:

Command chown is used to change the owner of the file.

NAME

chown - change file owner and group

Syntax:

chown **<newOwnername> <fileName>**

OR: To change the **file ownership** use the chown command:

```
chown [user_name] [file_name]
```

```
Student# chown student2 m1.txt
```

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ sudo chown games m1.txt
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l
total 32
```

```
drwxr-xr-x 1 debasis debasis  4096 Jan 13 12:15 Example

-rw-r--r--  1 debasis debasis 30720 Jan 14 11:17 archiveEx.tar

-rw-r--r--  1 games   floppy     21 Jan 14 17:58 m1.txt

-rw-r--r--  1 debasis debasis     0 Jan 13 11:56 m2.txt
```

Command chown can also be used to change both user owner and group.

Syntax:

chown **<newOwner:newGroup> <fileName>**

```
#change both user and group from the existing :
$ sudo chown games:cdrom m2.txt
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l *.txt
-rw-r--r-- 1 games   floppy  21 Jan 14 17:58 m1.txt
-rw-r--r-- 1 games   cdrom    0 Jan 13 11:56 m2.txt
-rw-r--r-- 1 debasis debasis  0 Jan 13 11:56 m3.txt
-rw-r--r-- 1 debasis debasis  0 Jan 13 11:56 m4.txt
-rw-r--r-- 1 debasis debasis  0 Jan 13 11:56 m5.txt
```

Change the group ownership of a folder:

To change the group ownership of a folder, execute the 'chgrp' command with the folder name as follows:

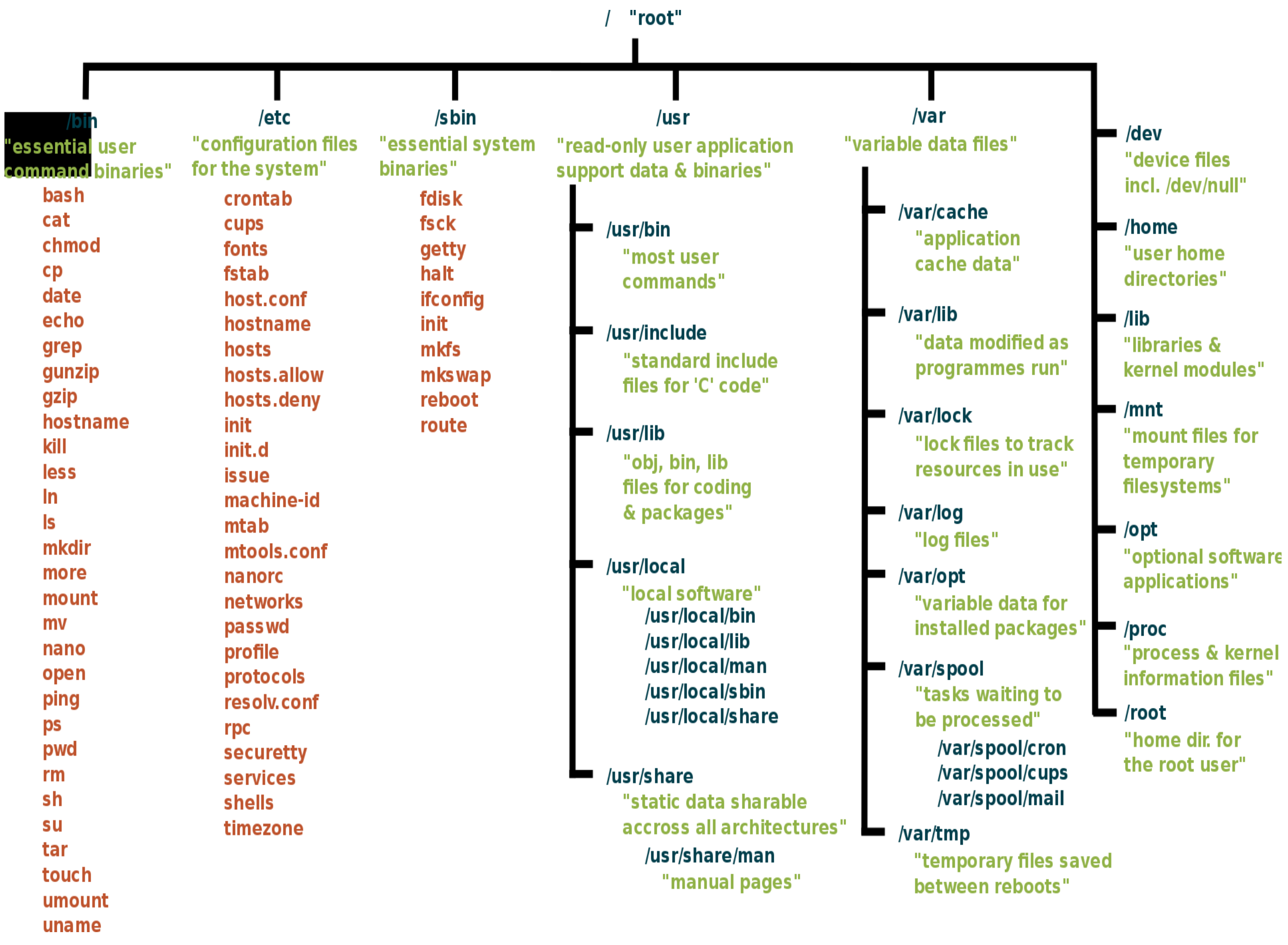
Syntax:
sudo chgrp groupnme Directoryname

```
#Change the group name of the directory and all others files and subdirectories:

debasis@LAPTOP-H3N6JCNE:~/CSE$ sudo chgrp -R cdrom Batch3
```

UNIX FILE SYSTEM:

/ (root) : bin,etc,sbin,usr,var,dev,home,lib,mnt,opt,proc,root



UNIX/Linux File Systems LAYOUT:

Layout of the file system:

- Each **physical drive** can be divided into several **partitions**
- Each partition can contain one **file system**
- Each file system contains:
  1. **boot block(s)**;
  2. **superblock;**
  3. **inode list;**
  4. **data blocks.**

1.Boot Block(s) :

A boot block may contain the bootstrap code that is read into the machine upon booting.

Blocks on a Linux (and often a Unix) filesystem are **1024 bytes** in length, but may be longer or shorter. The blocks are normally a power of 2 in size (**1024** is 2 to the 10th power). Some systems use **512 bytes** (2 to the 9th) but 2048 and 4096 are also seen.

The first few blocks on any partition may hold a boot program, a short program for loading the kernel of the operating system and launching it. Often, on a Linux system, it will be controlled by LILO(**LILO is the LInux LOader**) or Grub, allowing booting of multiple operating systems. It's quite simple (and common) to have a multiple boot environment for both Linux and a flavour of Windows.

NOTE:

GNU GRUB (short for **GNU GRand Unified Bootloader**, commonly referred to as **GRUB**) is a boot loader package from the GNU Project

2.Superblock:

The boot blocks are followed by the superblock, which contains information about the geometry of the physical disk, the layout of the partition, number of inodes and data blocks, and much more.

- A superblock describes the state of the file system:
  - how large it is;
  - how many files it can store;
  - where to find free space on the file system;
  - who has ownership of it;
  - and more.

3. The inode list is an array of "**i**nformation **node**s" analogous to the FAT (File Allocation Table) system in MS-DOS.

4. data blocks start at the end of the inode list and contain file data and directory blocks.

The term file system can mean a single disk, or it can mean the entire collection of devices on a system. It's held together in this second sense by the directory structure.

The directory "tree" usually spans many disks and/or partitions by means of mount points. For example, in Red Hat Linux, there are pre-defined mount points for floppy disks and CD-ROMs at **`floppy`** and **`cdrom`** in **`/mnt`**.

UNIX / Linux Inodes:

Linux Inodes:

The **inode** (index node) is a data structure in a Unix-style file system that describes a file-system object such as a file or a directory.

An Inode number is a uniquely existing number for all the files in Linux and all Unix type systems.

When a file is created on a system, a file name and Inode number is assigned to it.

Generally, to access a file, a user uses the file name but internally file name is first mapped with respective Inode number stored in a table.

Inode Contents:

An Inode is a data structure containing metadata about the files.

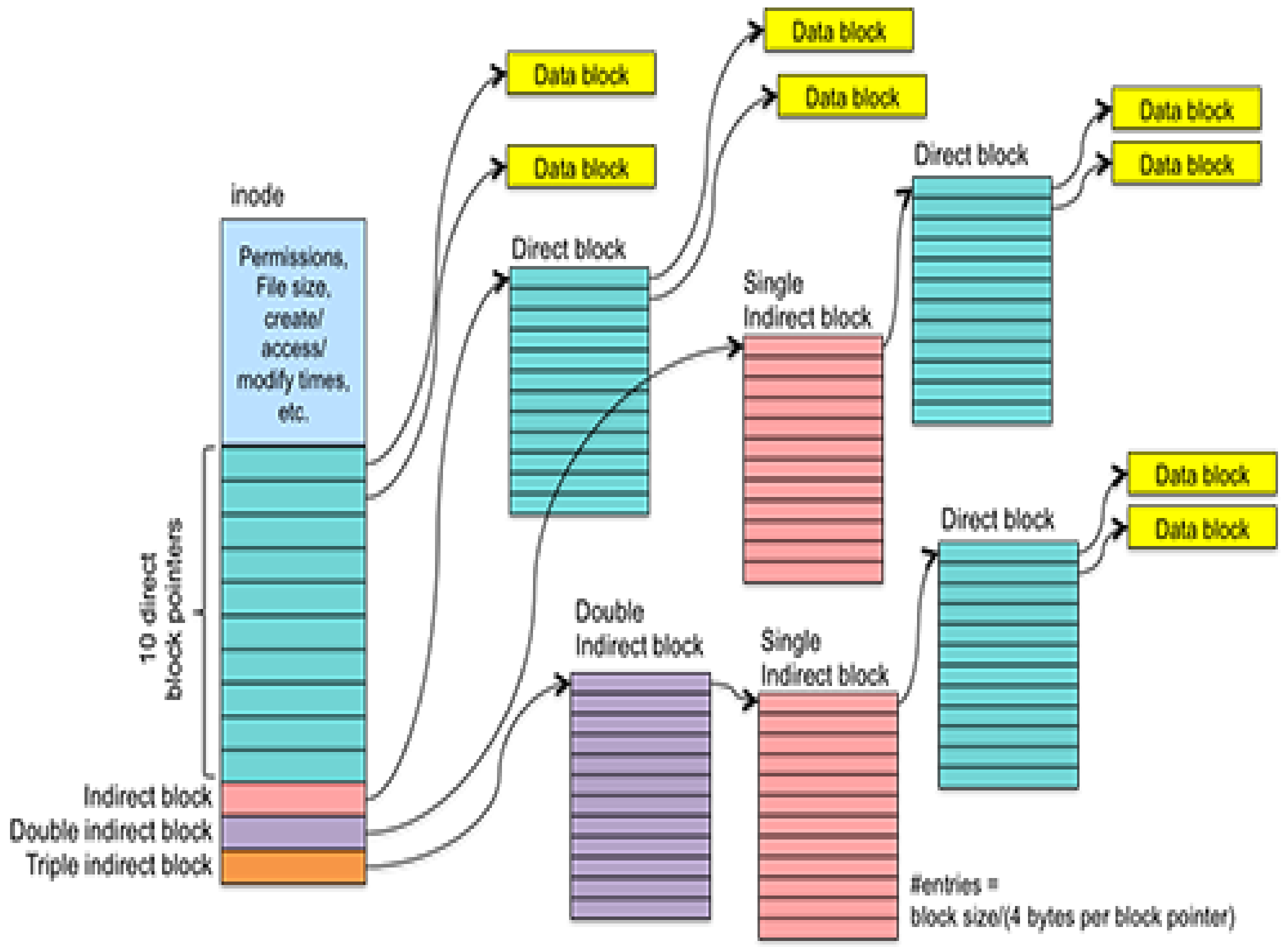
Inodes contain the following information:

- **File types** - file, folder, executable program, block special etc.
- **UID(owner)**
- **GID(Group)**
- **File size** (in bytes, sometimes also in blocks)
- **Time stamps** - creation, last access, modification times

[Timestamps telling when the inode itself was last modified (`ctime`, *inode change time*), the file content last modified (`mtime`, *modification time*), and last accessed (`atime`, *access time*).]

- **File permissions** - read, write, execute
- **Access control list** - permissions for special users/groups
- **File protection flags**
- **Location of file on hard disk** - directory path where the file is stored
- **Number of links** – (soft/hard link)
- **Additional file metadata**
- **File pointers (**Array of pointers to data blocks on disk) - addresses of the storage blocks that store the file contents

NOTE: An inode does not contain the filename or the actual data. When a file is created in the UNIX file system, it is assigned an inode number and a filename. This linked pair allows the filename to be changed without affecting the file ID in the system.



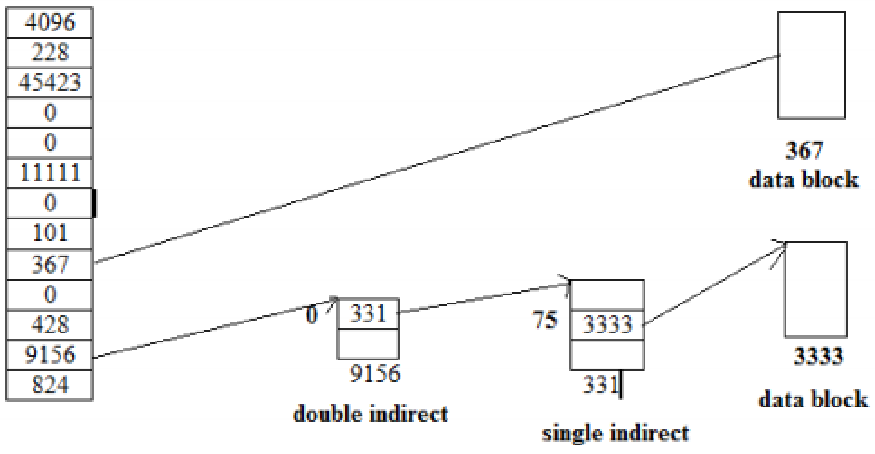
Inode Number:

Each Inode has a unique number and Inode number can be seen with the help of **ls -li** command.

```
$ ls –li

total 1

 844424930564508 -rwxrwxrwx 1 Debasis Goswami  None  0 Aug 12 11:32 aa.txt
12947848928690251 -rw-r--r-- 1    Debasis Goswami None 16 Aug 31 16:52 file.txt
```



Block Layout of a sample File and its Inode

Several blocks entries in the inode are 0,meaning that the logical block entries contain no data. This happens if no process ever wrote data into the file at any byte offsets corresponding to those blocks and hence the block numbers remain at their initial value 0. No disk space is wasted for such blocks.

Linking files

In Linux and Unix, a data file is a bunch of data blocks on a disk, managed by an inode. Its name is stored only in the directory. Or in many directories. Both **"soft" (symbolic) links** and **"hard" links** can be made using the `ln(1)` command or the `link(2)` and `symlink(2)` system calls.

COMMAND NAME: ln

ln - make links between files (**hard link)**

-s, --symbolic : make symbolic links instead of hard links

```
#hard link

debasis@LAPTOP-H3N6JCNE:~/link$ ln  source_hardlink.txt hardlink.txt

output:

debasis@LAPTOP-H3N6JCNE:~/link$ cat hardlink.txt
this is my hard link test file.
```

```
debasis@LAPTOP-H3N6JCNE:~/link$ cat source_hardlink.txt
this is my hard link test file.
```

Q:**What is Soft Link And Hard Link In Linux?**

Answer:
A **symbolic** or **soft link** is an actual link to the original file, whereas a **hard link** is a mirror copy of the original file.
If you delete the original file, the soft link has no value, because it points to a non-existent file.
But in the case of hard link, it is entirely opposite. Even if you delete the original file, the hard link will still has the data of the original file. Because hard link acts as a mirror copy of the original file

A hard Link

- can't cross the file system boundaries (i.e. A hardlink can only work on the same filesystem),
- can't link directories,
- has the same inode number and permissions of original file,
- permissions will be updated if we change the permissions of source file,
- has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed.
- A hard link acts as a copy (mirrored) of the selected file. It accesses the data available in the original file.

A soft link

- can cross the file system,
- allows you to link between directories,
- has different inode number and file permissions than original file,
- permissions will not be updated,
- has only the path of the original file, not the contents.
- A soft link (also known as Symbolic link) acts as a pointer or a reference to the file name.

To Check inode number:

```
debasis@LAPTOP-H3N6JCNE:~/link$ ls -li
total 0
```

```
▪ 12103423998608328 -rw-r--r-- 2 debasis debasis 32 Jan 16 19:38 hardlink.txt
▪ 12103423998608328 -rw-r--r-- 2 debasis debasis 32 Jan16 19:38 source_hardlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ vi source_hardlink.txt



debasis@LAPTOP-H3N6JCNE:~/link$ cat source_hardlink.txt
▪ this is my hard link test file.
▪ add the text in a file.
▪ debasis@LAPTOP-H3N6JCNE:~/link$ cat hardlink.txt
▪ this is my hard link test file.
▪ add the text in a file.

debasis@LAPTOP-H3N6JCNE:~/link$ vi hardlink.txt


debasis@LAPTOP-H3N6JCNE:~/link$ cat hardlink.txt
▪ this is my hard link test file.
▪ add the text in a file.
▪ add again.
▪ debasis@LAPTOP-H3N6JCNE:~/link$ cat source_hardlink.txt
▪ this is my hard link test file.
▪ add the text in a file.
▪ add again.
```

```
debasis@LAPTOP-H3N6JCNE:~/link$ ln source_hardlink.txt hardlink2.txt

debasis@LAPTOP-H3N6JCNE:~/link$ ls -li

total 0

12103423998608328 -rw-r--r-- 3 debasis debasis 67 Jan 16 19:40 hardlink.txt

12103423998608328 -rw-r--r-- 3 debasis debasis 67 Jan 16 19:40
hardlink2.txt

12103423998608328 -rw-r--r—3 debasis debasis 67 Jan 16 19:40
source_hardlink.txt
```

```
debasis@LAPTOP-H3N6JCNE:~/link$ rm source_hardlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ ls -li

total 0
```

```
12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16
19:40 hardlink.txt

12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16
19:40 hardlink2.txt

debasis@LAPTOP-H3N6JCNE:~/link$ cat hardlink2.txt
this is my hard link test file.
add the text in a file.
add again.

NOTE: NO effect though original/source file deleted.
```

```
#Example of soft or Symbolic link:

debasis@LAPTOP-H3N6JCNE:~/link$ ln -s source_softlink.txt softlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ ls -li

total 0

12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16 19:40
hardlink.txt

12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16 19:40
hardlink2.txt

28991922601209363 lrwxrwxrwx 1 debasis debasis 19 Jan 16 20:13
softlink.txt -> source_softlink.txt

 5348024557552600 -rw-r--r-- 1 debasis debasis 49 Jan 16 20:12
source_softlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ cat softlink.txt

This is my source file for soft or symbolic link

.

debasis@LAPTOP-H3N6JCNE:~/link$ rm source_softlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ ls -li

total 0

12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16 19:40
hardlink.txt

12103423998608328 -rw-r--r-- 2 debasis debasis 67 Jan 16 19:40
hardlink2.txt

28991922601209363 lrwxrwxrwx 1 debasis debasis 19 Jan 16 20:13
```

```
softlink.txt -> source_softlink.txt

debasis@LAPTOP-H3N6JCNE:~/link$ cat softlink.txt

cat: softlink.txt: No such file or directory

NOTE: After deleting source soft link file, soft link file
also be deleted.
   • Original file be deleted, not accessible of shortcut
     file.
```

Q: What is the difference between Hard link and the normal copied file?

Answer:

You might be wondering why would we create a hard link while we can easily copy/paste the original file? Creating a hard link to a file is different than copying it.

If you copy a file, it will just duplicate the content. So if you modify the content of a one file (either original or hard link), it has no effect on the other one. However if you create a hard link to a file and change the content of either of the files, the change will be seen on both.

File descriptor:

In Unix and related computer operating systems, a **file descriptor** (**FD**, less frequently **fildes**) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket. File descriptors form part of the POSIX application programming interface.

A file descriptor is a non-negative integer, generally represented in the C programming language as the type `int` (negative values being reserved to indicate "no value" or an error condition).

Each Unix process (except perhaps a daemon) should expect to have three standard POSIX file descriptors, corresponding to the three standard streams:

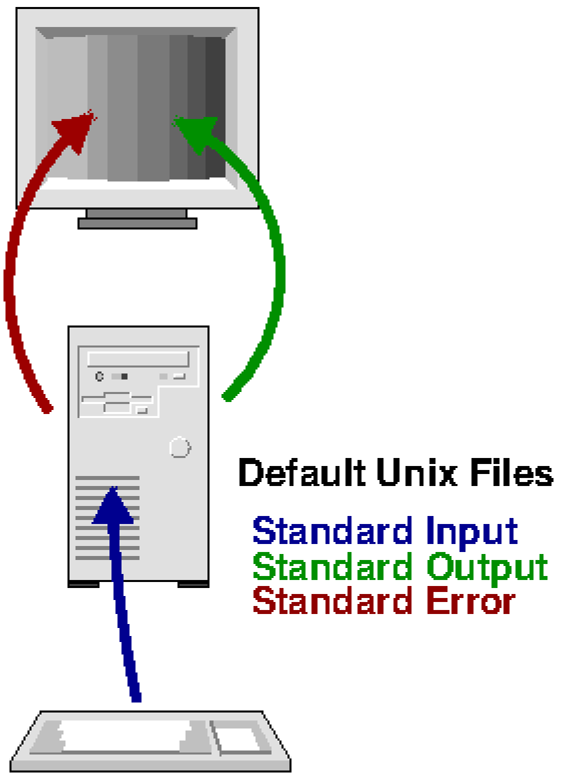
Integer value	Name	<`unistd.h`> symbolic constant[1]	<`stdio.h`> file stream[2]
0	Standard input	`STDIN_FILENO`	`stdin`
1	Standard output	`STDOUT_FILENO`	`stdout`

2	Standard error	STDERR_FILENO	stderr

Table 1.1. Standard Files Provided by Unix

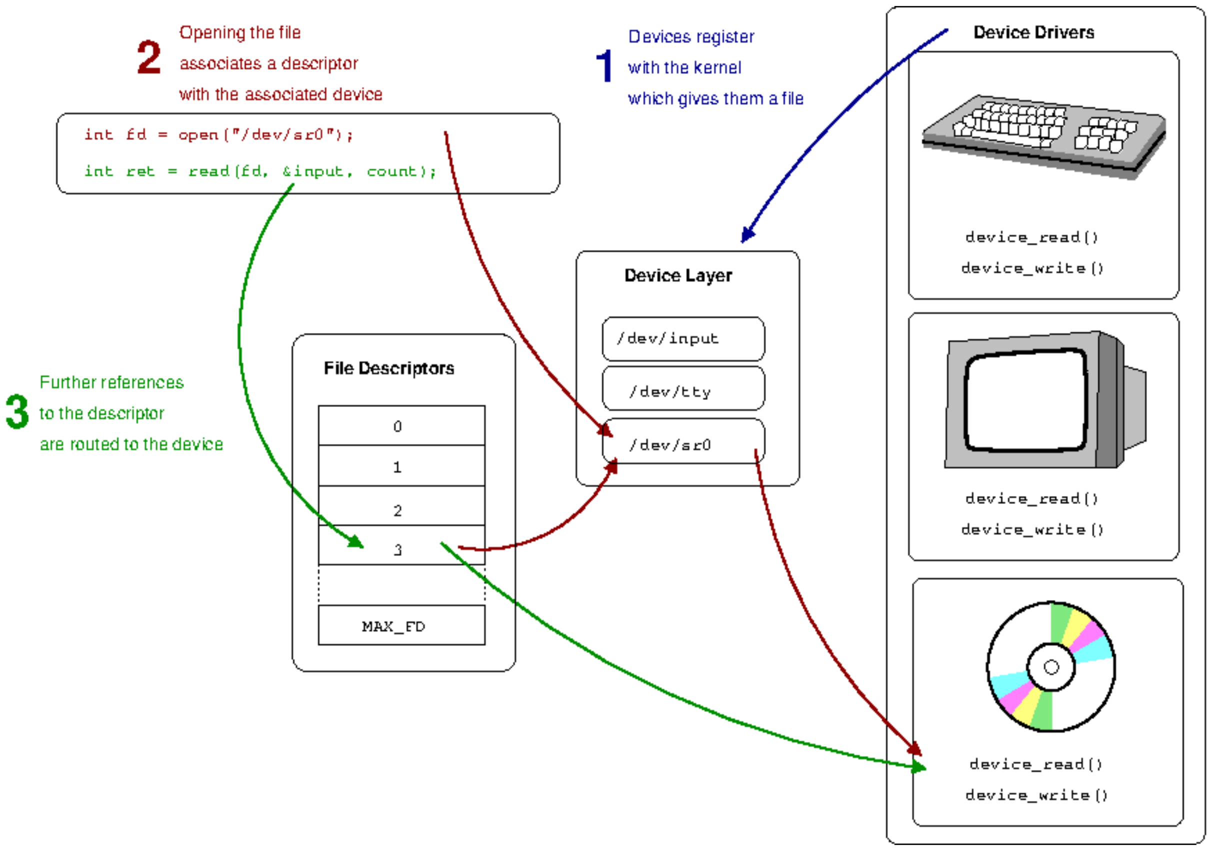
Descriptive Name	Short Name	File Number	Description
Standard In	stdin	0	Input from the keyboard
Standard Out	stdout	1	Output to the console
Standard Error	stderr	2	Error output to the console

Figure 1.2. Default Unix Files



This raises the question of what an *open file* represents. The value returned by an `open` call is termed a *file descriptor* and is essentially an index into an array of open files kept by the kernel.

Figure 1.3. Abstraction


2 Opening the file associates a descriptor with the associated device
int fd = open("/dev/sr0");
int ret = read(fd, &input, count);
1 Devices register with the kernel which gives them a file
Device Drivers
device_read()
device_write()
Device Layer
/dev/input
/dev/tty
/dev/sr0
File Descriptors
0
1
2
3
MAX_FD
3 Further references to the descriptor are routed to the device
device_read()
device_write()
device_read()
device_write()

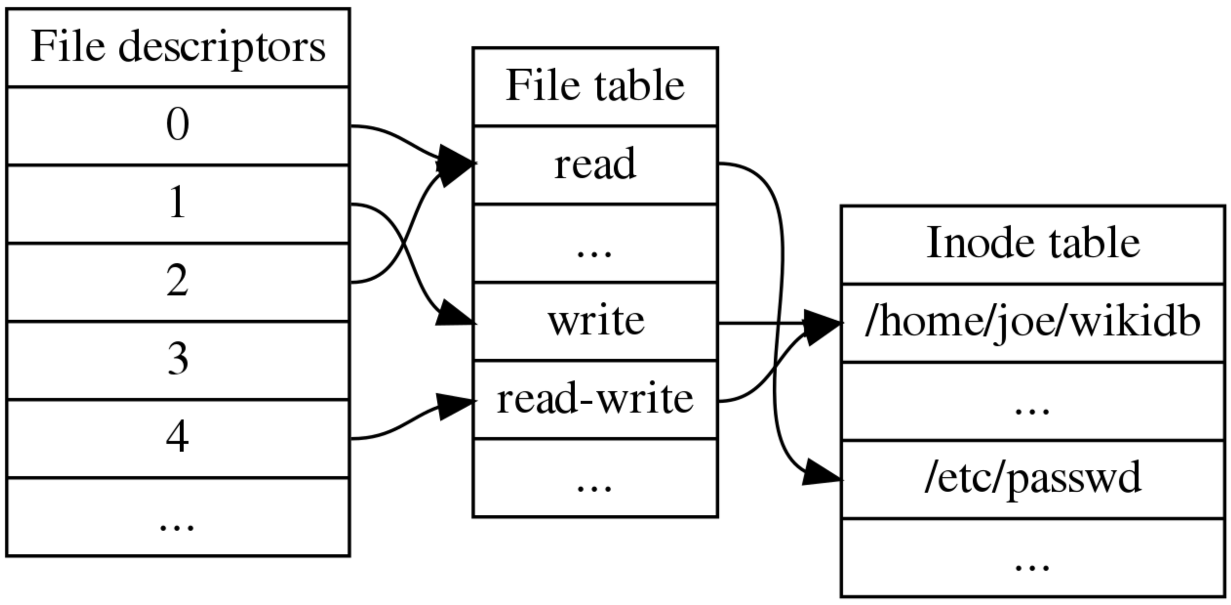
File descriptors are an index into a file descriptor table stored by the kernel. The kernel creates a file descriptor in response to an `open` call and associates the file descriptor with some abstraction of an underlying file-like object, be that an actual hardware device, or a file system or something else entirely.

Consequently a process's `read` or `write` calls that reference that file descriptor are routed to the correct place by the kernel to ultimately do something useful.

To display file descriptor details:

```
$ cd /proc
$echo $$
590 (PID of Shell/process)
$cd 590
$ls
Output:
attr    comm    fd    mountinfo  ns        schedstat  statm auxv   cwd    gid_map  mounts
oom_adj      setgroups  status cgroup  environ  limits  mountstats  oom_score_adj  smaps    task
cmdline  exe     maps    net      root       stat     uid_map

debasis@LAPTOP-H3N6JCNE:/proc/34$ cd fd
debasis@LAPTOP-H3N6JCNE:/proc/34/fd$ ls
0  1  2  255
```



File descriptors for a single process, file table and inode table. Note that multiple file descriptors can refer to the same file table entry (e.g., as a result of the dup system call and that multiple file table entries can in turn refer to the same inode (if it has been opened multiple times; the table is still simplified because it represents inodes by file names, even though an inode can have multiple names).
File descriptor 3 does not refer to anything in the file table, signifying that it has been closed.

Q:What is inode table?
Answer: An **inode** is an entry in **Inode table**, containing information (the metadata) about a regular file and directory. An **inode** is a data structure on a traditional Unix-style file system such as ext3 or ext4.

Q: What is ext3 and ext4 file system?
Answer: ext4 stands for fourth extended **file system**. It was introduced in 2008. ... You can also mount an existing **ext3** fs as **ext4** fs (without having to upgrade it). Several other new features are introduced in **ext4**: multiblock allocation, delayed allocation, journal checksum. fast fsck, etc.

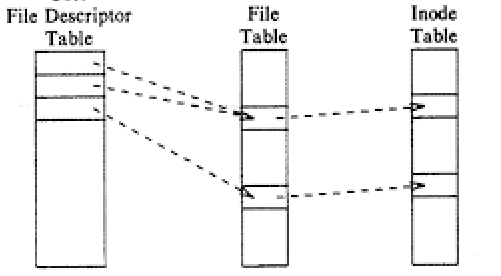
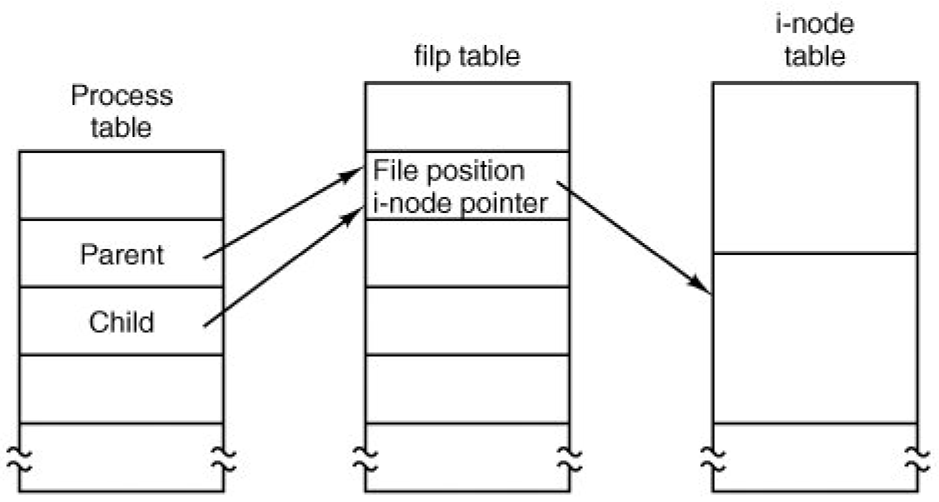
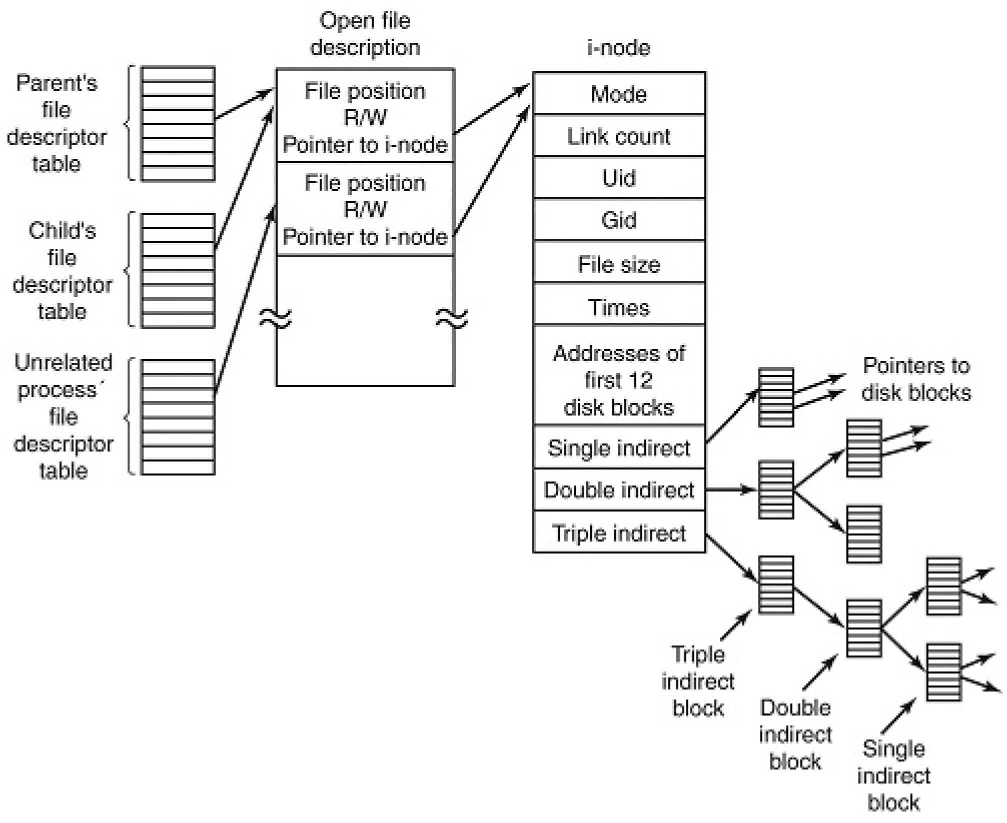


Figure 2.2. File Descriptors, File Table, and Inode Table





<u>Default permissions of file and directory and using umask:</u>

Q:What is the default umask?

By **default**, the system sets the permissions on a text file to 666, which grants read and write permission to user, group, and others, and to 777 on a directory or executable file. The value assigned by the **umask** command is subtracted from the **default**. You can use the **umask** (stands for user mask) command to **determine** the **default permissions** for newly created **files**.

Q:What is the typical default umask value?

Answer: Typical default umask value is 022.

What is the typical default umask value? For Linux files, the **default** permission is 666, and the **default umask** is 022. When a new file is created, it will be assigned 644 (rw-r--r--) permissions.

How to change Default Umask(**user mask**) Permission in Linux:

When we create a new file or directory, shell automatically assigns the default permission to it. Default permission is the subtraction of umask permission and pre-defined initial permission.

```
Default permission = pre-defined initial permission - umask permission
```

- The pre-defined initial permissions for **files and directories are 666 and 777** respectively.
- The default umask permissions for root user and remaining users are 0022 and 0002 respectively.
- The pre-defined initial permissions are fixed and cannot be changed. The default umask permissions are flexible and can be updated as per requirement.
- Umask permissions are also known as umask values or umask setting. All these words (umask permissions, umask values and umask setting) are used to represent the four numeric variables which are used to calculate the default permissions.

```
To check default umask:

debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ umask

output:

0022
```

Note:

For Files: 666-022=644(**user 6**-wr, **Group 4-**r, **other 4**-r).

For Directories: 777-022=755(**User 7**drwx, **Group 5** r-x, **other 5** r-x)

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l
```

```
total 32

drwxr-xr-x 1 debasis debasis  4096 Jan 13 12:15 Example(755)
```

For Files: 666-022=644(**user 6-**wr, **Group 4-**r, **other 4-**r).

110->6-> rw-

100->4->r--

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l

total 32

 -rw-r--r-- 1 debasis debasis     0 Jan 13 11:56 m5.txt(644)
```

#default permission change:

To change umask and set it as 044

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ umask 044

debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ umask

0044
```

Output:

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l

total 32

drwx-wx-wx 1 debasis debasis  4096 Jan 15 19:30 dg(733)
```

For FILE:

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ cat>new.txt

this is my test filedebasis@LAPTOP-H3N6JCNE:~/BCA/bca$ ls -l

total 1

-rw--w--w- 1 debasis debasis    20 Jan 15 19:34 new.txt(622)
```

Why above output:

For directory: 777-044=733 (111 drwx, 011 -wx, 011 -wx

For File: 644-044=622

010 :2 “ -w-

Q: What is SUID , SGID and sticky bit?

Answer: SUID means set user ID and **SGID** means set group ID. **SUID** have a value of 4 or use u+s. **SGID** has value of 2 or use g+s similarly **sticky bit** has a value of 1 or use +t to apply the value.

Q:What does chmod 1777 mean?

Answer: Chmod 1777 (**chmod** a+rwx,ug+s,+t,u-s,g-s) sets permissions so that, (U)ser / owner can read, can write and can execute. (G)roup can read, can write and can execute.

Q:What is the use of Sticky bit?

Answer: A Sticky bit is a permission bit that is set on a file or a directory that lets **only the owner of the file/directory or the root user to delete or rename the file**. No other user is given privileges to delete or rename the file created by some other user.

1. Set the sticky bit on Directory

The example below enables the sticky bit on a directory.

Use chmod command to set the sticky bit. If you are using the octal numbers in chmod, give 1 before you specify other numbered privileges, as shown below. The example below, gives rwx permission to user, group and others (and also adds the sticky bit to the directory).

```
$ chmod 1777 dir
```

Or, you can assign only sticky bit to an existing directory (without touching any other user, group and other privileges) using chmod command as shown below.

```
$ chmod +t dir
```

To remove the sticky bit from a directory, do the following.

```
$ chmod -t dir
```

Significance of file attribute for directory:

Directories:

- Always contain at least two entries: "`.`" (self) and "`..`" (parent)
- Have at least as many links as the total number of subdirectories
- The root directory is different in that self and parent are identical

Listing of modification and access time:

Check file modification time in LINUX:

Before going into how to use the touch command, let's start by reviewing the file timestamps in Linux.

A file in Linux has three timestamps:

- **atime (access time)** - The last time the file was accessed/opened by some command or application such as cat , vim or grep .
- **mtime (modify time)** - The last time the file's content was modified.
- **ctime (change time)** - The last time the file's attribute or content was changed. The attribute includes file permissions, file ownership or file location.

You can use -mtime option. It returns list of file if the file was last accessed N*24 hours ago. For example to find file in last 2 months (60 days) you need to use -mtime +60 option.

- **-mtime +60** means you are looking for a file modified 60 days ago.

- **-mtime -60** means less than 60 days.
- **-mtime 60** If you skip + or – it means exactly 60 days.

To display the file status including the timestamps, use the `stat` command.

```
stat file_name
```

Creating a new file requires write permissions on the parent directory. Otherwise, you will receive a permission denied error.

```
COMMAND Name : stat
stat - display file or file system status file1
```

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ stat m5.txt

  File: m5.txt

  Size: 18              Blocks: 0          IO Block: 4096   regular file

Device: 2h/2d   Inode: 14636698789000959  Links: 1

Access: (0644/-rw-r--r--)  Uid: ( 1000/ debasis)   Gid: ( 1000/ debasis)

Access: 2021-01-15 20:04:35.096559800 +0530

Modify: 2021-01-15 20:04:35.096559800 +0530

Change: 2021-01-15 20:04:35.096559800 +0530

 Birth: -
```

Example:

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ touch -c m4.txt

debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ stat m4.txt

  File: m4.txt

Size: 0                Blocks: 0          IO Block: 4096   regular empty file

Device: 2h/2d   Inode: 10696049115051774  Links: 1

Access: (0644/-rw-r--r--)  Uid: ( 1000/ debasis)   Gid: ( 1000/ debasis)

Access: 2021-01-15 20:15:12.771789800 +0530
```

```
Modify: 2021-01-15 20:15:12.771789800 +0530

Change: 2021-01-15 20:15:12.771789800 +0530

 Birth: -
```

Time stamp changing (touch):

Linux Touch Command Examples (How to Change File Timestamp):

Linux touch Command

Every file in Linux is associated with timestamps, which specifies the last access time, last modification time and last change time.

Whenever we create a new file, or modify an existing file or its attributes, these timestamps will be updated automatically.

Touch command is used to change these timestamps (access time, modification time, and change time of a file).

Create an Empty File using touch

You can create an empty file using touch command. The following example will create a zero byte new file named m1.txt.

```
$ touch m1.txt
```

For example, if the file `file1` doesn't exist the following command will create it otherwise, it will change its timestamps:

```
touch file1
```

To create or modify multiple files at once, specify the file names as arguments:

```
touch file1 file2 file3
```

If you don't want the touch command to create new files, use the `-c` (`--no-create`) option.

For example, if the file `file1.txt` exist the following command will change the file timestamps otherwise, it will do nothing:

```
touch -c file1.txt
```

File locating (find):

Find command is used to search and locate the list of files and directories based on conditions you specify for files that match the arguments.

Find can be used in a variety of conditions like you can find files by **permissions**, **users**, **groups**, **file type**, **date**, **size**, and other possible criteria.

COMMAND NAME:

find - search for files in a directory hierarchy

Example:

1. Find Files Using Name in Current Directory

Find all the files whose name is **test.txt** in a current working directory.

```
# find . -name “test.txt”
```

2. Find Files Under Home Directory

Find all the files under **/home** directory with name **test.txt**.

```
# find /home -name test.txt

/home/test.txt
```

3. Find Files Using Name and Ignoring Case

Find all the files whose name is **test.txt** and contains both capital and small letters in **/home** directory.

```
# find /home -iname test.txt

./test.txt
./Test.txt
```

4. Find Directories Using Name

Find all directories whose name is **student** in **/** directory.

```
# find / -type d -name student

/student
```

5. Find Python Files Using Name

Find all **python** files whose name is **test.py** in a current working directory.

```
# find . -type f -name test.py

./test.py
```

NOTE: **-type f** – Only search for files and not directories

6. Find all Python Files in Directory

Find all **python** files in a directory.

```
# find . -type f -name "*.py"

./test.py
./login.py
./index.py
```

7. Find Files With 777 Permissions

Find all the files whose permissions are **777**.

```
# find . -type f -perm 0777 -print
```

8. Find Files Without 777 Permissions

Find all the files without permission **777**.

```
# find / -type f ! -perm 777
```

9. Find SGID Files with 644 Permissions

Find all the **SGID bit** files whose permissions set to **644**.

```
# find / -perm 2644
```

10. Find Sticky Bit Files with 551 Permissions

Find all the **Sticky Bit** set files whose permission are **551**.

```
# find / -perm 1551
```

11. Find SUID Files

Find all **SUID** set files.

```
# find / -perm /u=s
```

12. Find SGID Files

Find all **SGID** set files.

```
# find / -perm /g=s
```

13. Find Read Only Files

Find all **Read Only** files.

```
# find / -perm /u=r
```

14. Find Executable Files

Find all **Executable** files.

```
# find / -perm /a=x
```

15. Find Files with 777 Permissions and Chmod to 644

Find all **777** permission files and use **chmod** command to set permissions to **644**.

```
# find / -type f -perm 0777 -print -exec chmod 644 {} \;
```

16. Find Directories with 777 Permissions and Chmod to 755

Find all **777** permission directories and use **chmod** command to set permissions to **755**.

```
# find / -type d -perm 777 -print -exec chmod 755 {} \;
```

17. Find and remove single File

To find a single file called **test.txt** and remove it.

```
# find . -type f -name "test.txt" -exec rm -f {} \;
```

18. Find and remove Multiple Files

To find and remove multiple files such as **.jpeg** or **.txt**, then use.

```
# find . -type f -name "*.txt" -exec rm -f {} \;

OR

# find . -type f -name "*.jpeg" -exec rm -f {} \;
```

19. Find all Empty Files

To find all empty files under a certain path.

```
# find /tmp -type f -empty
```

20. Find all Empty Directories

To file all empty directories under a certain path.

```
# find /student -type d -empty
```

#Find all empty directories from user debasis:

```
debasis@LAPTOP-H3N6JCNE:~$ find /home/debasis -type d -empty

/home/debasis/.config/procps

/home/debasis/BCA/bca/dg

/home/debasis/BCA/bca/testEx/kolkata

/home/debasis/BCA/BCA1/BCABATCH

/home/debasis/CSE10/Batch2

/home/debasis/CSE2/test1/test2/tes3

/home/debasis/ECE/Network/BatchECE

/home/debasis/MCA/Batch1

/home/debasis/MCA/Batch2

/home/debasis/student1

/home/debasis/test/india
```

21. File all Hidden Files

To find all hidden files, use the below command.

```
# find /tmp -type f -name ".*"
```

find /home -type f -name “.*.txt”

find /home -type f -name “.*.*”

Search Files Based On Owners and Groups:

22. Find Single File Based on User

To find all or single file called **test.txt** under **/** root directory of owner root.

```
# find / -user root -name test.txt
```

23. Find all Files Based on User

To find all files that belong to user **student** under **/home** directory.

```
# find /home -user student
```

24. Find all Files Based on Group

To find all files that belong to the group **Developer** under **/home** directory.

```
# find /home -group developer
```

25. Find Particular Files of User

To find all **.txt** files of user **student** under **/home** directory.

```
# find /home -user student -iname "*.txt"
```

26. Find Last 50 Days Modified Files

To find all the files which are modified **50** days back.

```
# find / -mtime 50
```

27. Find Last 50 Days Accessed Files

To find all the files which are accessed **50** days back.

```
# find / -atime 50
```

28. Find Last 50-100 Days Modified Files

To find all the files which are modified more than **50** days back and less than **100** days.

```
# find / -mtime +50 –mtime -100
```

29. Find Changed Files in Last 1 Hour

To find all the files which are changed in the last **1 hour**.

```
# find / -cmin -60
```

30. Find Modified Files in Last 1 Hour

To find all the files which are modified in the last **1 hour**.

```
# find / -mmin -60
```

31. Find Accessed Files in Last 1 Hour

To find all the files which are accessed in the last **1 hour**.

```
# find / -amin -60
```

Find Files and Directories Based on Size:

32. Find 50MB Files

To find all **50MB** files, use.

```
# find / -size 50M
```

33. Find Size between 50MB – 100MB

To find all the files which are greater than **50MB** and less than **100MB**.

```
# find / -size +50M -size -100M
```

34. Find and Delete 100MB Files

To find all **100MB** files and delete them using one single command.

```
# find / -type f -size +100M -exec rm -f {} \;
```

35. Find Specific Files and Delete

Find all **.jpeg** files with more than **10MB** and delete them using one single command.

```
# find / -type f -name *.jpeg -size +10M -exec rm {} \;
```

#Find all text files from user name debasis:

```
debasis@LAPTOP-H3N6JCNE:~$ find /home -user debasis  -type f -name "*.txt"    | more
```

#Find and count total directories from user debasis:

```
debasis@LAPTOP-H3N6JCNE:~$ find /home -user debasis  -type d  | more | wc -l

39
```

#Find empty directory from current path:

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/testEx$ find . -type d -empty
```

#Find all empty directory from parent or previous directory(..):

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca/Example$ find .. -type d -empty
```

```
output:
../dg
../emtydir
../testEx/kolkata
```

#To find text files that were last modified 60 days ago, use

```
$ find /home/student -iname "*.txt" -mtime -60 -print
```

#List all mp3s that were accessed **exactly** 10 days ago:

```
$ find /home/student -iname "*.mp3" -atime 10 -type f
```

#List all text files that were accessed **more than** 10 days ago:

```
debasis@LAPTOP-H3N6JCNE:~$ find . -iname "*.txt" -atime +10 -type f
```

#List all text files that were accessed **Less than** 10 days ago:

```
debasis@LAPTOP-H3N6JCNE:~$ find . -iname "*.txt" -atime -10 -type f
```

#Display all files of size > 0 bytes and Less than 2k

```
debasis@LAPTOP-H3N6JCNE:~$ find /home/debasis -type f -ctime -60 -iname "*"  -size +1k -size -
10k
```

#Display all files of size > 0 byetes and Less than 2k

```
debasis@LAPTOP-H3N6JCNE:~/BCA/bca$ find . -atime -10 -size  +0b -size -2k -print
```

```
debasis@LAPTOP-H3N6JCNE:~$ find . -type f -name "*.*"  -size +0b -size -10k | more
```