UNIT 2

STANDARD INPUT AND OUTPUT

Linux and other Unix-like operating systems feature the concept of standard streams of data. Each command, and therefore each process (i.e., running instance of a program), is automatically initialized with three data streams: one input stream, called standard input, and two output streams, called standard output and standard error. These streams consist of data in plain text form and are considered to be special types of files.

Programs that run from a console (i.e., an all-text display mode) or from a terminal window can receive input data in several ways. One is through command line arguments, which are names of files entered on the command line after the name of the program and any options. In the following example, file1 and file2 are arguments for the wc command:

```
wc -w file1 file2
```

That is, file1 and file2 provide the input data for wc, which by default counts the number of lines, words and characters in any given text. The -w option customizes wc by telling it to count only the number of words in each file and ignore the number of lines and characters.

STANDARD INPUT

Standard input, often abbreviated stdin, is the input data for a program in the absence of any command line arguments. It is by default any text entered from the keyboard. Thus if wc is typed in at the command line and the ENTER key is pressed without providing any arguments, any text typed in on all subsequent lines will be stored in memory until the wc command is executed by simultaneously pressing the CONTROL and d keys on a new, blank line.

INPUT REDIRECTION

The most common use of redirection is to redirect the input to be typed at the keyboard. The general use of input redirection is when you have some kind of file, which you have ready and now you would like to use some command on that file.

You can use input redirection by typing the '<' operator. An excellent example of input redirection has been shown below:

```
$ mail cousin < my_typed_letter
```

The above command would start the mail program with the contents of the file named 'my_typed_letter' as the input since the input redirection operator was used.

Standard input can be redirected to come from any text file in place of the keyboard by using the input redirection operator, which is a leftward pointing angular bracket. Thus, to redirect the standard input for the command sort (which sorts lines of text in alphabetic order) from the keyboard to the file named file3, type:

```
sort < file3
```

The result is the same as using file3 as an argument, although the mechanism is different, i.e.,

```
sort file3
```

A program can alternatively obtain its input from another command through the use of a pipe, which can be considered a type of redirection operator and is represented by the vertical bar character. In the following example, the output of the head command, which by default reads the first ten lines of a file, becomes the standard input for the sort command:

```
head file3 | sort
```

STANDARD OUTPUT

Command line programs write to standard output, sometimes abbreviated stdout. Its default destination is generally the display screen, but it can be redirected (e.g., to a file, where it will be saved, or to a printer) or piped to another program to serve as its standard input.

For example, the command file writes the names of files and their types (e.g., text, HTML, GIF or directory) to standard output, which is, as usual, the monitor screen. Thus the following command will display on the screen a list showing the name and type of each file in the current directory.

```
file *
```

In this example, wildcard represented by the star character. A wildcard is a character than can represent some set of characters.

The standard output for this command can easily be redirected to another file using the output redirection operator, which is a rightward facing angular bracket. In the example below, it is redirected to a file named file4:

```
file * > file4
```

Or it can be redirected to become the standard input of another program using the pipe operator, such as:

```
file * | grep directory
```

Here the filter grep is used to search for any lines in its standard input (which, of course, is the standard output of file *) that contain the sequence of characters directory.

OUTPUT REDIRECTION

The most common use of redirection is to redirect the output (that normally goes to the terminal) from a command to a file instead. This is known as output redirection. This is generally used when you get a lot of output when you execute your program.

The way to redirect the output is by using '>' operator in shell command you enter. This is shown below. The '>' symbol is known as the output redirection operator.

```
$ ls > listing
```

The ls command would normally give you a directory listing. Since you have the '>' operator after the 'ls' command redirection would take place.

PIPES

The most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called 'pipes'. With pipes, the standard output of one command is fed into the standard input of another. Here is the absolute use of this command.

```
$ ls –l | less
```

In this example, the output of the ls command is fed into less. By using this "| less" trick, you can make any command have scrolling output.

Some examples of commands used together with pipes: –

COMMAND	WHAT IT DOES

ls- lt| head	Display the 10 newest files in the current directory
du| sort –nr	Display a list of directories and how much space they consume, sorted from the largest to the smallest.
find –type f –print| wc-l	Displays the total no. of files in the current working directory and all of its subdirectories.

Pipes can be used in Linux feature which enable different processes to communicate. One of the fundamental features that makes Linux and other Unix useful is the "pipe". Pipe allows separate process to communicate without having been design explicitly to work together. A simple example of using a pipe is the command: –

```
ls | grep x
```

Print the list of directory and prints only those lines containing the letter x.

TEE COMMAND

Tee command is a little tool which is very useful but often overlooked. The function of tee does not sound very exciting. Tee command is used to store and view (both at the same time) the output of any other command. It writes to the STDOUT, and to a file at a time as shown in the examples below.

Example 1: Write output to stdout, and also to a file: –

The following command displays output only on the screen (stdout).

```
$ ls
```

The following command writes the output only to the file and not to the screen.

```
$ ls > file
```

The following command (with the help of tee command) writes the output both to the screen (stdout) and to the file.

```
$ ls | tee file
```

Example 2: Write the output to two commands: –

You can also use tee command to store the output of a command to a file and redirect the same output as an input to another command.

The following command will take a backup of the crontab entries, and pass the crontab entries as an input to sed command which will do the substituion.

```
$ crontab -l | tee crontab-backup.txt | sed 's/old/new/' | crontab –
```

Read from an input and write to a standard output and file.

LINUX FILE SECURITY

The Linux security model is based on the one used on UNIX systems, and is as rigid as the UNIX security model, which is already quite robust.

Directory and the files which are stored in them are arranged in a hierarchical tree structure. Access can be controlled for both the files and the directories allowing a very flexible level of access.

In Linux, every file and every directory owned by a single user on that system. Each file and directory also has a security group associated with it that has access rights to the file or directory. Is a user is not the directory or file owner nor assigned to the security group for the file that user is classified as other and may still have certain rights to access the file.

Each of the three file access categories owner, group and other has a set of three access permissions associated with it.

	1	2	3	4	5	6	7
i.	drwxrwxr-x	24	root	users	1024	Aug 19 00:05	
ii.	drwxr-xr-x	22	root	Root	1024	Jun 11 22:04	Mail
iii.	-rw-rw-r–	3	root	Root	43244	Jul 20 14:11	REDAME

Each of these fields provides useful information to the security administration. First, a description of each field, then a more in-depth explanation of most important ones:

Field 1	Permission for this file and directory

Field 2	No. of hard links to this file or directory
Field 3	Owner of the file or directory
Field 4	The group to which the file belongs
Field 5	Size of file
Field 6	Modification time
Field 7	File name

PERMISSION TYPES

Code	**Permission**	**Meaning**
0 or –	Access denied	The access right that is supposed to be on this place is not granted.
4 or r	Read	read access is granted to the user category defined in this place
2 or w	Write	write permission is granted to the user category defined in this place
1 or x	Execute	execute permission is granted to the user category defined in this place
s	Special term	Setuid or setgid permission

USER GROUP CODES

Code	Meaning
u	user permissions
g	group permissions
o	permissions for others

CHMOD COMMAND: –

A normal consequence of applying strict file permissions, and sometimes a nuisance, is that access rights will need to be changed for all kinds of reasons. We use the chmod command to do this, and eventually to chmod has become an almost acceptable English verb, meaning the changing of the access mode of a file. The chmod command can be used with alphanumeric or numeric options.

```
$ cat hello
$ ls -l hello
$ chmod u+x hello
```

When using chmod with numeric arguments, the values for each granted access right have to be counted together per group.

FILE PROTECTION WITH CHMOD

Command	Meaning
chmod 400 file	To protect a file against accidental overwriting.
chmod 500 directory	To protect yourself from accidentally removing, renaming or moving files from this directory.

chmod 600 file	A private file only changeable by the user who entered this command.
chmod 644 file	A publicly readable file that can only be changed by the issuing user.
chmod 660 file	Users belonging to your group can change this file, others don't have any access to it at all.
chmod 700 file	Protects a file against any access from other users, while the issuing user still has full access.
chmod 755 directory	For files that should be readable and executable by others, but only changeable by the issuing user.
chmod 775 file	Standard file sharing mode for a group.
chmod 777 file	Everybody can do everything to this file.

LOGGING ON TO ANOTHER GROUP

When you type id on the command line, you get a list of all the groups that you can possibly belong to, preceded by your user name and ID and the group name and ID that you are currently connected with.

Changing Group:

The chgrp command is used to change the group with which a file is associated.

```
$ ls -i
```

```
$ chgrp presidents abc.txt
```

where, presidents: – old user is author can change to president.

Change the directory:

```
chgrp authors examples
```

Changing owner:

```
$ ls -l myfile
-r—r—r 1 fred sysadmin
$ chown root: root myfile
chmod 400 mydoc.txt
```

0	No permission
1	Execute
2	Write
3	Write/execute
4	Read
5	Read/execute
6	Read/write
7	Read/write/execute

<u>DEFAULT PERMISSION AND UMASK</u>: –

By default, Linux and Unix permissions for new directories are typically set to 755 allowing read, write and execute permission to user and only read and execute to group and other users. 644 is also used to set default permission in Linux.

A user or administration may want to change the Linux default permissions by using the umask command in a login script.

The umask command can be used without specifying any arguments to determine what the current default permission are.

The -s option ca be used to see the current default permissions displayed in the alpha symbolic format.

Default permissions can be changed by specifying the mode argument to umask within the user's shell profile.

Using umask to set default permissions.

```
$ umask

022

(User = 0, group = 2, other = 2)

$ umask -s

(User = rwx, group = rx, other = rx)

$ umask 033

$ umask

0033

$ umask -s

(User = rwx, group = r, other = r)
```

LIMITATION OF THE UMASK:

1. The umask command can restricts permissions.
2. The umask command can't grant extra permissions beyond what is specified by the program that creates the file or directory.

UMASK AND LEVEL OF SECURITY:

UMASK VALUE	SECURITY LEVEL	EFFECTIVE PERMISSION
022	Permissive	755
026	Moderate	751

027	Moderate	750
077	moderate	700

<u>Vi EDITOR BASICS</u>

There is a no. of different Unix file editors that you can use to create and modify files. Although not always the easiest to learn, the vi editor is one of the most versatile. This document provides you with the basics information to help you get started with the vi-text editor. A text editor is simply a program used to edit files which contain plain text.

While using vi, at any one time you are is one of three modes of operation. These modes are known as “command mode”, “insert mode”, and “last line mode”.

For command mode, enter vi < filename.

For insert mode, press i.

For last line mode, press esc, shift key: , wq press ENTER

When you start up vi, you are in “command mode”. This mode allows you to use certain commands to edit files or to change to other modes. For example, typing “x” while in command mode deletes the character underneath the cursor. The arrow keys move the cursor around the file which you are editing.

You may actually insert or edit text within “insert mode”. When using vi, you will probably spend most of your time within this mode. You start insert mode by using a command such as “i” (for insert) from command mode. While in insert mode, you are inserting text into the document from your current cursor location. To end insert mode and return to command mode press esc.

“Last line mode” is a special mode used to give certain extended commands to vi. While typing these commands, they appear on the last line of the screen (hence the name). For example, when you type “:” from the command mode, you jump into the last line mode and can use commands such as “wq” (to write the file and quit vi) or “q!” (to quit vi without saving changes). Last line mode is generally used for vi commands which are longer than one character. In last line mode you enter a single-line command and press enter to execute it.

To create or edit a file:

vi [filename]

```
$ vi test
```

vi opens the file you specify in filename unless it does not exist. If the file does not exist, vi opens a new file which will be called filename. The column of "~" characters indicates that you are at the end of the file.

There are many ways to edit files in Unix. Now a day you would find an improved version of vi editor which is called vim. Here vim stands for vi improved.

The vi is generally considered the standard in Unix editors because:

1. It is usually available on all the flavours of Unix system.
2. It is implementations are very similar across the board.
3. It is requiring very few resources.
4. It is more user friendly than any other editors like ed or ex.

INODE

An inode is a data structure on a file system on Linux and other Unix like operating system that stores all the information about a file excepts its name and its actual data.

The inode (index node) is a fundamental concept in the Linux and Unix file system. Each object in the file system is represented by an inode. Some attributes are:

1. File type (executable, block special, etc.)
2. Permissions (read, write, execute)
3. Owner
4. Group
5. File size
6. File access, change and modification time
7. File deletion time
8. of links (soft or hard)
9. Extended attributes such as append only or no one can delete file including root user
10. Access Control List (ACLs)

An inode is a data structure on a traditional Unix style file system such as UFS or ext3. An inode stores basic information about a regular file, directory or other file system.

How to see inode no.?

You use ls – i command to see inode no. of file.

```
$ ls – i /etc/passwd
```

Output:

```
32820 /etc/passwd
```

COMMAND TO ACCESS INODE NO.

Some command to access the inode no. for files:

- df – i: – it displays the inode information of the file system.

```
$ df – i
```

- stat command: – it is used to display file statistics that also displays inode no. of a file.

```
stat a (file)
```

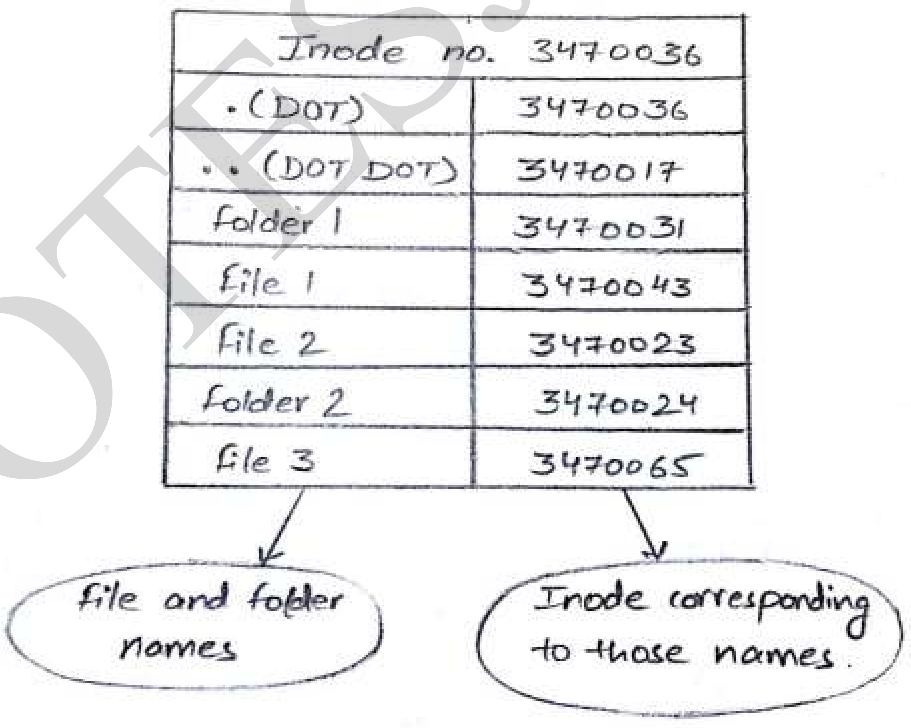
In a filesystem, inodes consist roughly of 1% of the total disk space, whether it is a whole storage unit (hard disk, thumb drive, etc) or a partition on a storage unit. The inode space is used to "track" the files stored on the hard disk. The inode entries store metadata about each file, directory or object but only points to there structures rather than storing the data. Each entry is 128 bytes in size. The metadata contained about each structure can include the following:

1. Inode number
2. Access Control List (ACL)
3. Extended attribute
4. Direct/indirect disk blocks
5. of blocks
6. File access, change and modification time
7. File deletion time
8. File generation no.
9. File size

10. File type
11. Group
12. of links
13. Owner
14. Permissions
15. Status flag

STRUCTURE OF AN INODE

Inode structure of a directory just consists of name to inode mapping of files and directory.



Inode no. 3470036	
.(DOT)	3470036
..(DOT DOT)	3470017
folder 1	3470031
file 1	3470043
file 2	3470023
folder 2	3470024
file 3	3470065

INODE STRUCTURE OF A DIRECTORY

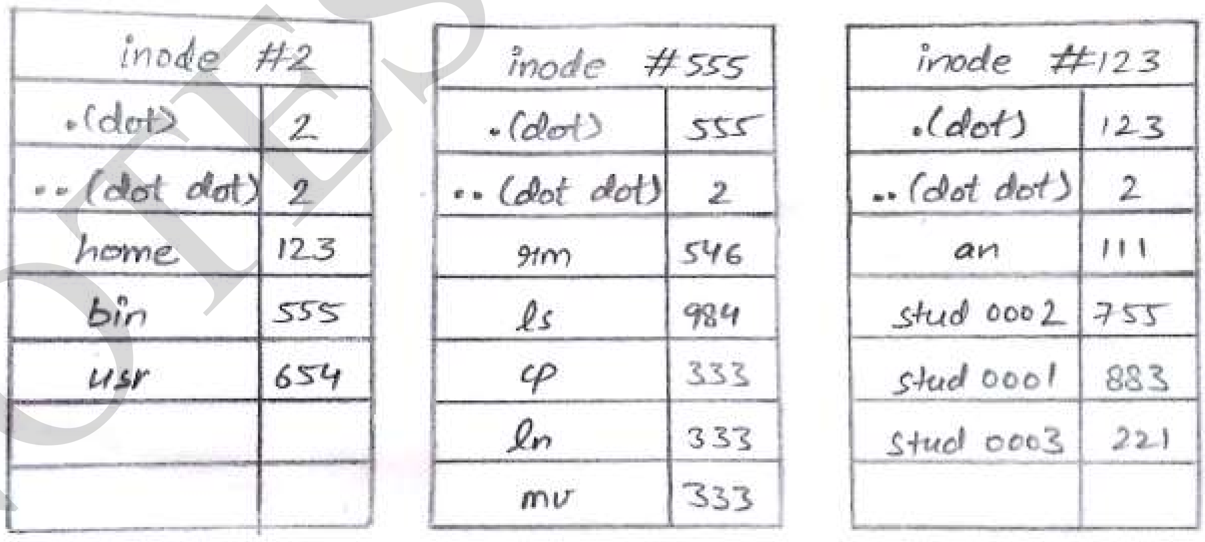
So here in the above shown diagram you can see the first two entries of (.) dot and (..) dot dot. You might have seen them whenever you list the contents of a directory.

A Unix or Linux directory is simply a list of pairs of names and associated inode no. that is all – the disk blocks of Unix directories contain only names and inode no.s.. Files have disk blocks containing file data; directories also have disk blocks; but the blocks contain lists of names and inode numbers.

Like most other inodes, directory inodes contain attribute information about the inode (permission, owner, etc.) and one or more disk block pointers in which to store data; but what is stored in the disk blocks of a directory is not file data but directory data (names and inode no.).

To find out attribute information of some file system object, which is stored with the inode, not in the directory. You must first use the inode number associated with the object to find the inode of the item and look at the item's attributes. This is why ls or ls – i are much faster than ls – l since ls – l has to do a look up on every inode in the directory to find out the inode attributes:

Suppose the root directory has inode number #2. Here is a small part of a unix file system tree, showing hypothetical inode no. of some directory and files:



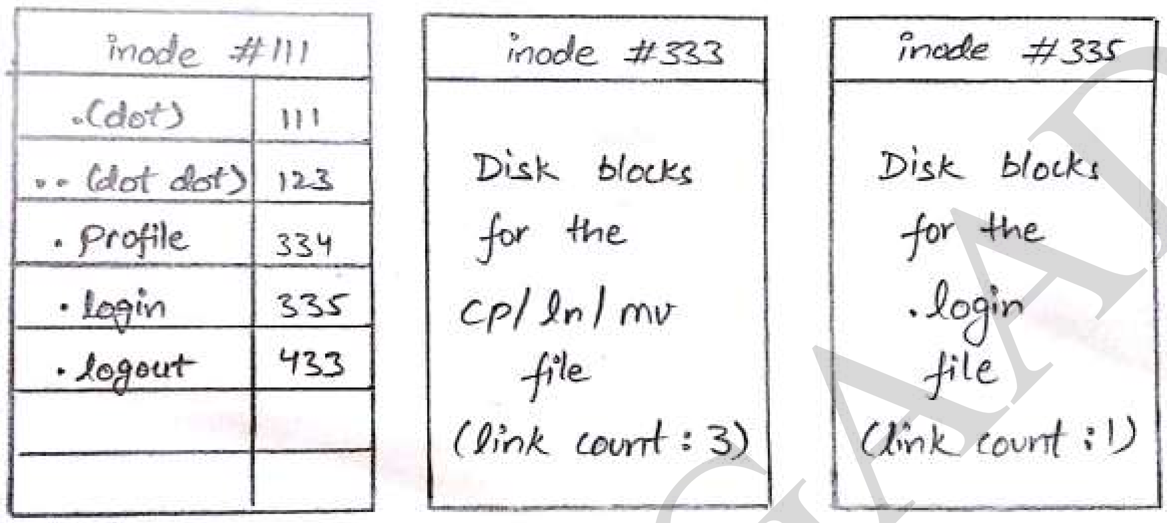
inode #2	
.(dot)	2
..(dot dot)	2
home	123
bin	555
usr	654

inode #555	
.(dot)	555
..(dot dot)	2
9im	546
ls	984
cp	333
ln	333
mv	333

inode #123	
.(dot)	123
..(dot dot)	2
an	111
stud 0002	755
stud 0001	883
stud 0003	221

UNIX FILE SYSTEM TREE

How one directory named bin (#555) has three names to no. maps for the same node (#333). All three names (cp, ln, mv) refer to the same node no. in this case a file containing an executable program (this program is one that looks at its name and behaves differently depending on which name you use to call it).


inode #111
.(dot) 111
.. (dot dot) 123
.profile 334
.login 335
.logout 433
inode #333
Disk blocks for the cp/ln/mv file (link count : 3)
inode #335
Disk blocks for the .login file (link count : 1)

INODE

HARD LINKS AND SOFT/SYMBOLIC LINKS

Every file has a data structure (record) known as an inode that stores information about the file, and the filename is simply used as a reference to that data structure. A link is simply a way to refer to the contents of a file. There are two types of links:

HARD LINKS

A hard link is a pointer to the file's inode. For example, suppose that we have a file a-file.txt that contains the string "The file a-file.txt".

```
$ cat a-file.txt
The file a-file.txt
```

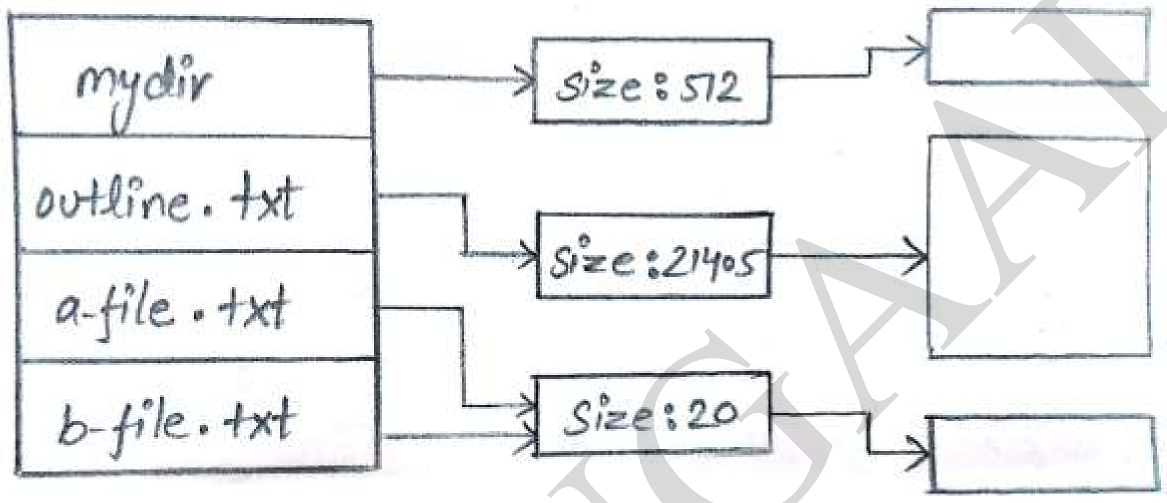
Now we use the ln command to create a link to a-file.txt called b-file.txt:

```
$ ls
./../ a-file.txt
```

```
$ ln a-file.txt b-file.txt
```

```
$ ls
```

```
./../ a-file.txt b-file.txt
```



HARD LINK

The two names a-file.txt and b-file.txt now refer to the same data.

```
$ cat b-file.txt
```

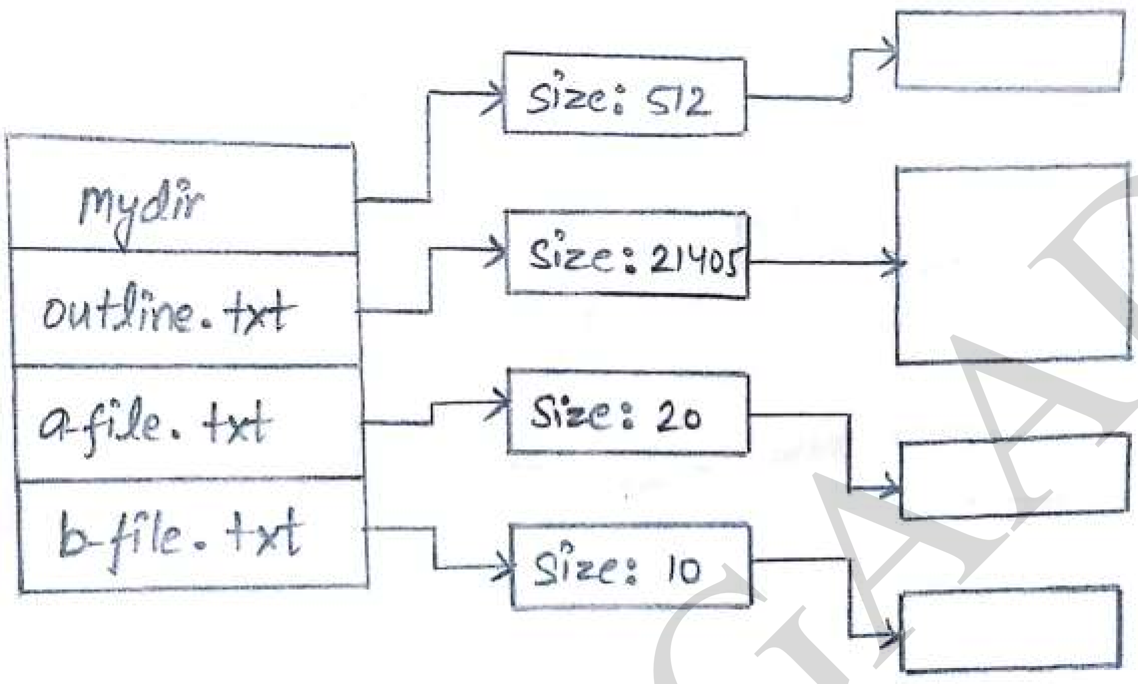
```
The file a-file.txt
```

SOFT LINK (SYMBOLIC LINKS)

A soft link, also called symbolic link. It is a file that contains the name of another file. We can then access the contents of the other file through that name. that is, a symbolic link is like a pointer to the pointer to the file's contents. For instance, supposed that in the previous example we had used the -s option of the ln to create a soft link:

```
$ ln -s a-file.txt b-file.txt
```

On disk, the file system would look like the following picture:



SOFT LINKS

$ ls

./../ a-file.txt b-file.txt

<u>**HARD LINK VS SOFT LINK IN LINUX**</u>

HARD LINK:

1. Hard link can't link directories.
2. Can't cross file system boundaries.
3. It can only be used on files (not directories) and is basically a way of giving multiple names to same physical file.
4. The files are hard links if they have the same Inode no.s.

SOFT LINK (SYMBOLIC LINK):

1. Symbolic links are not updated.
2. To create links between directories.
3. Can cross file system boundaries.
4. The soft link can exist on a different disk partition than the target.

<u>How to create a symbolic link?</u>

You can create symbolic link with in command:

```
$ ln -s /path/to/file1.txt    /path/to/file2.txt
$ ls -ali
```

Above command will create a symbolic link to file1.txt.

<u>Symbolic link creating and deletion:</u>

Let us create a directory called foo:

```
$ mkdir foo
$ cd foo
```

copy /etc/resolv.conf, enter:

```
$ cp /etc/resolv.conf
```

View inode no. enter:

```
$ ls -ali
```

<u>OUTPUT:</u>

```
total 152

1048600   drwxr-xr-x   2   vivek   vivek   4096   2008-12-09   20:19

1048601   -rwxr-xr-x   1   vivek   vivek   129    2008-12-09   20:19   Resolve.conf
```

Now create soft link to resolv.conf enter:

```
$ ln -s resolv.conf alink.conf
$ ls -ali
```

<u>OUTPUT:</u>

```
1048602   lrwxrwxrwx   1   vivek   vivek   11   2008-12-   20:24   Alink.conf->
                                                09                 resolv.conf
```

If we delete the text file resolv.conf., alink.conf becomes a broken link and our data is lost:

```
$ rm resolv.conf
$ ls -ali
```

MOUNT COMMAND

All files accessible in a Linux/Unix system are arranged in one big tree. The file hierarchy rooted at /. These files can be spread out over special devices. The mount command serves to attach the file system found on some devices to the big file tree. The standard form of the mount command is:

```
mount -t type device dir
```

This tells the kernel to attach the file system found on device (which is of *type* type) at the directory dir. The previous content (if any) and owner and mode of dir become invisible as long as this file system remains mounted. The path name dir refers to the root of the file system on device.

Listing help: –

mount -h (print a help message)

mount -v (print a version)

mount -l [-t type]

Lists all mounted file system (of *type*) the options -l adds the labels in this listing.

mount/dev/cdrom/mnt/cdrom

This command will connect the device "/dev/cdrom" (usually cdrom drive) to the directory "/mnt/cdrom" so that you can access the files and directories on the CDROM disk in the CDROM drive under the "mnt/cdrom"directory, which must already exists when this command is executed.

$ sudo fdisk -l (describe the list of devices/ directories or partition)

EXAMPLES: –

```
$ mkdir mount
$ mkdir mount/thumb

$ ls mount/thumb

$ sudo mount -t auto /dev/sdb/home/zoya/mount/thumb

$ ls mount/thumb (mount all files in my pen drive).
```

UNMOUNT COMMAND

The unmount command detaches the file system mentioned from the file hierarchy. A file system is specified by giving the directory where it has been mounted.

EXAMPLE: –

```
unmount /mnt/usb
```

A usb key device, assuming that it had been mounted in the directory /mnt/usb.

COMMANDS

CREATING ARCHIVES

The archiver also known simply as "ar", is a Unix/Linux utility that maintain group of files as a single archive file. Today ar is generally used only to create and update static library files that the link editor or linker uses: it can be used to create archives for any purpose but has been largely replaced by tar for purposes other than static libraries. In the Linux standard base, ar has been deprecated and is expected to disappear in a future release of that standard. The rational provide was that "the LSB does not include software development utilities nor does it specify .o and .a file formats".

TAR ARCHIVES

The tar (tape archive) command is used to convert a group of files into an archive.

An archive is a single file that contains any no. of individual files plus information to allow them to be restored to their original from by one or more extraction programs.

Archives are convenient for storing files as well as for transmitting data and distributing programs.

Tar was originally designed for backups on magnetic tapes, it can now be used to create archive files anywhere on a file system.

Tar command is used to create archive and extract the archive files.

SYNTAX: –

tar [options] [archive-file] [file or directory to be archived]

OPTIONS: –

-c	Creates archive
-x	Extract the archive
-f	Creates archive with given file names
-t	Display or lists files in archives file
-v	Archives and adds to an existing archive file
-A	Concatenates the archive files

EXAMPLES: –

1. To archive a directory or file:

```
tar -cvf backup.tar/etc
```

This command creates a tar file called backup.tar which is the archive of /etc directory, where,

Backup.tar – is a tar file created

/etc – is a directory archived

2. To archive a directory or file and store it in a storage device:

```
tar -cvf /dev/fdo  /home/user1/HGB
```

This command will archive /etc directory and store it in floppy-disk, where,

/dev/fdo – is a floppy-disk name where the archive is stored.

/home/user1/HGB – is a directory archived.

3. To extract the archive:

tar -xvf backup.tar

This command will extract the backup.tar file.

4. To list the file in an archive.

tar -tvf backup.tar

The above command will display the files and directories that archived in backup.tar.

GZIP

Using gzip is bit easier than zip in some ways. With zip you need to specify the name of the newly created zip file or zip won't work; gzip reduces the size of named files using Lempel-Zip Coding (LZ77). Whenever possible, each file is replaced by one with the extension .gz, while keeping the same ownership modes, access and modification times. gzip will only attempt to compress regular files.

EXAMPLE:

1. gzip /home/z1/documents/f1.doc – 14kb

In this example we compress the f1.doc to f1.doc.gz – 3kb.

gz is the extension of gzip file format.

2. gzip -v /home/z1/documents/editor.doc

This command is used to show the percentage or compression rate of file and completely replaced the doc to gz file.

GUNZIP

Compress or expand files. gunzip takes a list of files on its command line and replaces each files whose name ends with .gz, -gz, .z, -z, -Z or .Z and which begins with the correct magic no. with an uncompressed file without the original extension. gunzip also recognized the special extension .tgz and .taz. when compressing .gzip uses the .tgz extension if necessary instead of truncating a file with a .tar extension. Uncompressed files compressed with gzip.

EXAMPLES:

```
$ ls -l
-rw-r—r–        scott        scott        224425        p-L.txt.gz

$ gunzip p-L.txt.gz
$ ls -l
-rw-r—r–        scott        scott        508925        P-L.txt
```

BZIP2

bzip2 features a high rate of compression together with reasonably fast speed. Most files can be compressed to a smaller file size than is possible with the more traditional gzip and zip programs.

Moreover, like those programs, the compression is lossless, meaning that no data is lost during compression and thus the original files can be exactly regenerated. The only disadvantage of bzip2 is that it is some what slower than gzip and zip.

SYNTAX:

```
bzip2 [option] file-name
```

bzip2 is commonly used without any options. Any no. of files can be compressed simultaneously by merely listing their named as arguments. The following would compress the three files named file1, file2 and file3.

EXAMPLE:

```
bzip2 file1 file2 file3
```

```
$ ls -l

-rw-r—r–     scott     scott     1234     md.txt

$ bzip2 md.txt

$ ls -l

-rw-r—r–     scott     scott     123      md.txt.bz2
```

BUNZIP2

Decompress the specified filename using a sophisticated algorithm that is up to 30 percent faster that that used by gzip. bunzip2 does not overwrite existing output files.

EXAMPLES:

```
bunzip2 a1.tar.bz2

$ ls -l

$ bunzip2 md.txt.bz2

$ ls -l

md.txt

av
```