Experiment-6

Aim: Creating and Manipulating files: cat, cp, mv, rm, ls and its options, touch and their options, which is, where is, what is

Cat command in Linux with examples

Cat (concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, and concatenate files. So let us see some frequently used cat commands.

1) To view a single file

Command:

```
$cat filename
```

Output

```
It will show content of given filename
```

2) To view multiple files

Command:

```
$cat file1 file2
```

Output

```
This will show the content of file1 and file2.
```

3) To view contents of a file preceding with line numbers.

Command:

```
$cat -n filename
```

Output

```
It will show content with line number

example:-cat-n  geeks.txt

1)This is geeks

2)A unique array
```

4) Create a file

Command:

```
$ cat >newfile
```

Output

```
Will create and a file named newfile
```

5) Copy the contents of one file to another file.

Command:

```
$cat [filename-whose-contents-is-to-be-copied] > [destination-filename]
```

Output

```
The content will be copied in destination file
```

6) **Cat command can suppress repeated empty lines in output**

Command:

```
$cat -s geeks.txt
```

Output

```
Will suppress repeated empty lines in output
```

7) Cat command can append the contents of one file to the end of another file.

Command:

```
$cat file1 >> file2
```

Output

```
Will append the contents of one file to the end of another file
```

8) Cat command can display content in reverse order using tac command.

Command:

```
 $tac filename
```

Output

```
Will display content in reverse order
```

9) Cat command can highlight the end of line.

Command:

```
$cat -E "filename"
```

Output

```
Will highlight the end of line
```

touch command in Linux with Examples

The ***touch*** command is a standard command used in UNIX/Linux operating system which is used to create, change and modify timestamps of a file. Basically, there are two different commands to create a file in the Linux system which is as follows:

- **cat command**: It is used to create the file with content.
- **touch command:** It is used to create a file without any content. The file created using touch command is empty. This command can be used when the user doesn't have data to store at the time of file creation.

Using touch Command

Initially, we are in home directory and this can be checked using the *pwd* command. Checking the existing files using command **ls** and then long listing command (ll) is used to gather more details about existing files.

Touch command Syntax to create a new file: You can create a single file at a time using touch command.

Syntax:
```
touch file_name
```

The file which is created can be viewed by **ls** command and to get more details about the file you can use **long listing command ll or ls -l** command.

Touch command to create multiple files: Touch command can be used to create the multiple numbers of files at the same time. These files would be empty while creation.

Syntax:
```
touch File1_name File2_name File3_name
```

Multiple files with name *Doc1*, *Doc2*, *Doc3* are created at the same time using touch command here.

touch Command Options

Like all other command **Touch** command have various options. These options are very useful for various purpose.

touch -a: This command is used to change access time only. To change or update the last access or modification times of a file touch -a command is used.

Syntax:

```
touch -a fileName
```

Here *touch -a* command changes access time of the file named *Doc1*.

touch -c : This command is used to check whether a file is created or not. If not created then don't create it. This command avoids creating files.

Syntax:

```
touch -c fileName
```

touch -m : This is used to change the modification time only. It only updates last modification time.

Syntax:

```
touch -m fileName
```

touch -t : This is used to create a file using a specified time.

Syntax:

```
touch -t YYMMDDHHMM fileName
```

cp command in Linux with examples

cp stands for **copy**. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. *cp* command require at least two filenames in its arguments.

Syntax:

```
cp [OPTION] Source Destination
cp [OPTION] Source Directory
cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory
```

First and second syntax is used to copy Source file to Destination file or Directory. Third syntax is used to copy multiple Sources(files) to Directory.

cp command works on three principal modes of operation and these operations depend upon number and type of arguments passed in cp command :

1. **Two file names:** If the command contains two file names, then it copy the contents of **1st file** to the **2nd file**. If the 2nd file doesn't exist, then first it creates one and content is copied to it. But if it existed then it is simply overwritten without any warning. So be careful when you choose destination file name.

```
cp Src_file Dest_file
```

Suppose there is a directory named *geeksforgeeks* having a text file **a.txt**.

Example:

```
$ ls
a.txt
$ cp a.txt b.txt
$ ls
a.txt  b.txt
```

2. **One or more arguments :** If the command has one or more arguments, specifying file names and following those arguments, an argument specifying directory name then this command copies each source file to the destination directory with the same name, created if not existed but if already existed then it will be overwritten, so be careful !!

```
cp Src_file1 Src_file2 Src_file3 Dest_directory
```

Suppose there is a directory named *geeksforgeeks* having a text file **a.txt, b.txt** and a directory name **new** in which we are going to copy all files.

Example:

```
$ ls
a.txt  b.txt  new
Initially new is empty
$ ls new
$ cp a.txt b.txt new
$ ls new
a.txt  b.txt
```

Note: For this case last argument *must* be a **directory** name. For the above command to work, *Dest_directory* must exist because **cp** command won't create it.

3. **Two directory names:** If the command contains two directory names, **cp** copies all files of the source directory to the destination directory, creating any files or directories needed. This mode of operation requires an additional option, typically **R**, to indicate the recursive copying of directories.

```
cp -R Src_directory Dest_directory
```

In the above command, **cp** behavior depends upon whether *Dest_directory* exists or not. If the *Dest_directory* doesn't exist, cp creates it and copies content of *Src_directory* recursively as it is. But if Dest_directory exists then copy of *Src_directory* becomes sub-directory under *Dest_directory*.

Options:
There are many options of **cp** command, here we will discuss some of the useful options:

Suppose a directory named **geeksforgeeks** contains two files having some content named as **a.txt** and **b.txt**. This scenario is useful in understanding the following options.

```
$ ls geeksforgeeks
a.txt  b.txt

$ cat a.txt
GFG

$ cat b.txt
GeeksforGeeks
```

1. -i(interactive): i stands for Interactive copying. With this option system first warns the user before overwriting the destination file. **cp** prompts for a response, if you press **y** then it overwrites the file and with any other option leave it uncopied.

```
$ cp -i a.txt b.txt
cp: overwrite 'b.txt'? y

$ cat b.txt
GFG
```

2. –b (backup): With this option **cp** command creates the backup of the destination file in the same folder with the different name and in different format.

```
$ ls
a.txt  b.txt

$ cp -b a.txt b.txt

$ ls
a.txt  b.txt  b.txt~
```

3. -f(force): If the system is unable to open destination file for writing operation because the user doesn't have writing permission for this file then by using **-f** option with **cp** command, destination file is deleted first and then copying of content is done from source to destination file.

```
$ ls -l b.txt
-r-xr-xr-x+ 1 User User 3 Nov 24 08:45 b.txt
```

```
User, group and others doesn't have writing permission.

Without -f option, command not executed
$ cp a.txt b.txt
cp: cannot create regular file 'b.txt': Permission denied

With -f option, command executed successfully
$ cp -f a.txt b.txt
```

4. -r or -R: Copying directory structure. With this option **cp** command shows its recursive behavior by copying the entire directory structure recursively. Suppose we want to copy **geeksforgeeks** directory containing many files, directories into **gfg** directory (not exist).

```
$ ls geeksforgeeks/
a.txt  b.txt  b.txt~  Folder1  Folder2

Without -r option, error
$ cp geeksforgeeks gfg
cp: -r not specified; omitting directory 'geeksforgeeks'

With -r, execute successfully
$ cp -r geeksforgeeks gfg

$ ls gfg/
a.txt  b.txt  b.txt~  Folder1  Folder2
```

mv command in Linux with examples

mv stands for **move**. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:

(i) It rename a file or folder.
(ii) It moves group of files to different directory.

No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

Syntax:

```
mv [Option] source destination
```

Let us consider 5 files having name **a.txt, b.txt** and so on till **e.txt**.

To rename the file **a.txt** to **geek.txt (not exist)**:

```
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt
```

```
$ mv a.txt geek.txt
$ ls
b.txt  c.txt  d.txt  geek.txt
```

If the destination file **doesn't exist**, it will be created. In the above command **mv** simply replaces the source filename in the directory with the destination filename (new name). If the destination file **exists**, then it will be **overwrite** and the source file will be deleted. By default, **mv** doesn't prompt for overwriting the existing file, so be careful!!

Let's try to understand with example, moving **geeks.txt** to **b.txt (exist)**:

```
$ ls
b.txt  c.txt  d.txt  geek.txt

$ cat geek.txt
India

$ cat b.txt
geeksforgeeks

$ mv geek.txt b.txt

$ ls
b.txt c.txt d.txt


$ cat b.txt
India
```

Options:

1. -i (Interactive): Like in cp, the -i option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press **y** for confirm moving, any other key leaves the file as it is. This option doesn't work if the file doesn't exist, it simply rename it or move it to new location.

```
$ ls
b.txt  c.txt  d.txt  geek.txt

$ cat geek.txt
India

$ cat b.txt
geeksforgeeks
```

```
$ mv -i geek.txt b.txt
mv: overwrite 'b.txt'? y

$ ls
b.txt c.txt d.txt

$ cat b.txt
India
```

2. -f (Force): mv prompts for confirmation overwriting the destination file if a file is **write protected**. The **-f** option overrides this minor protection and overwrites the destination file forcefully and deletes the source file.

```
$ ls
b.txt c.txt d.txt geek.txt

$ cat b.txt
Geeksforgeeks

$ ls -l b.txt
-r--r--r--+ 1 User User 13 Jan  9 13:37 b.txt

$ mv geek.txt b.txt
mv: replace 'b.txt', overriding mode 0444 (r--r--r--)? n

$ ls
b.txt  c.txt  d.txt  geek.txt

$ mv -f geek.txt b.txt

$ ls
b.txt  c.txt  d.txt

$ cat b.txt
India
```

rm command in Linux with examples

rm stands for **remove** here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names). **By default, it does not remove directories.**

This command normally works silently and you should be very careful while running **rm** command because once you delete the files then you are not able to recover the contents of files and directories.

Syntax:

```
rm [OPTION]... FILE...
```

Let us consider 5 files having name **a.txt, b.txt** and so on till **e.txt**.

```
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt
Removing one file at a time

$ rm a.txt

$ ls
b.txt  c.txt  d.txt  e.txt

Removing more than one file at a time
$ rm b.txt c.txt

$ ls
d.txt  e.txt
```

Note: No output is produced by **rm**, since it typically only generates messages in the case of an error.

Options:

1. **-i (Interactive Deletion):** Like in cp, the -i option makes the command ask the user for confirmation before removing each file, you have to press **y** for confirm deletion, any other key leaves the file un-deleted.

```
$ rm -i d.txt
rm: remove regular empty file 'd.txt'? y

$ ls
e.txt
```

2. **-f (Force Deletion): rm** prompts for confirmation removal if a file is **write protected**. The **-f** option overrides this minor protection and removes the file forcefully.

```
$ ls -l
total 0
-r--r--r--+ 1 User User 0 Jan  2 22:56 e.txt

$ rm e.txt
rm: remove write-protected regular empty file 'e.txt'? n

$ ls
e.txt
```

```
$ rm -f e.txt

$ ls
```

Note: -f option of rm command will not work for write-protect directories.

3. -r (Recursive Deletion): With **-r(or -R)** option rm command performs a tree-walk and will delete all the files and sub-directories recursively of the parent directory. At each stage it deletes everything it finds. Normally, **rm** wouldn't delete the directories but when used with this option, it will delete.

Below is the tree of directories and files:

```
$ ls
A

$ cd A

$ ls
B  C

$ ls B
a.txt  b.txt

$ ls C
c.txt  d.txt
```

Now, deletion from **A** directory(as parent directory) will be done as:

```
$ rm *
rm: cannot remove 'B': Is a directory
rm: cannot remove 'C': Is a directory

$ rm -r *

$ ls
```

Every directory and file inside **A** directory is deleted.

'ls' command in Linux

ls is a Linux shell command that lists directory contents of files and directories.

The syntax for the ls command is as follows:

```
ls [OPTIONS] [FILES]
```

When used with no arguments, the ls command will list the names of all files in the current working directory:

```
ls
```

The files are listed in alphabetical order:

```
cache  db  empty  games  lib  local  lock  log  mail  opt  run  spool  tmp
```

To list files in a specific directory, pass the path to the directory to the ls command. For example, to list the contents of the /etc directory, type:

```
ls /etc
```

You can also pass multiple directories and files to the ls command separated by space:

```
ls /etc /var /etc/passwd
```

If the user you are logged in doesn't have read permissions to the directory you will get a message saying that the ls command can't open the directory:

```
ls /root
ls: cannot open directory '/root': Permission denied
```

The ls command has a number of options. In the sections below, we will explore the most commonly used options.

Display All Information About Files/Directories Using ls -l

$ ls -l : To show long listing information about the file/directory.

-rw-rw-r– 1 maverick maverick 1176 Feb 16 00:19 1.c

1st Character – File Type: First character specifies the type of the file. In the example above the hyphen (-) in the 1st character indicates that this is a normal file. Following are the possible file type options in the 1st character of the ls -l output.

Field Explanation

- - - Regular file
- b - Block special file
- c - Character special file
- d - Directory
- l - Symbolic link
- n - Network file
- p - FIFO

- s - Socket

- **Field 1 – File Permissions**: Next 9 character specifies the files permission. The every 3 characters specifies read, write, execute permissions for user(root), group and others respectively in order. Taking above example, -rw-rw-r– indicates read-write permission for user(root) , read permission for group, and no permission for others respectively. If all three permissions are given to user(root), group and others, the format looks like -rwxrwxrwx
- **Field 2 – Number of links**: Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file.
- **Field 3 – Owner**: Third field specifies owner of the file. In this example, this file is owned by username 'maverick'.
- **Field 4 – Group**: Fourth field specifies the group of the file. In this example, this file belongs to "maverick' group.
- **Field 5 – Size**: Fifth field specifies the size of file in bytes. In this example, '1176' indicates the file size in bytes.
- **Field 6 – Last modified date and time**: Sixth field specifies the date and time of the last modification of the file. In this example, 'Feb 16 00:19' specifies the last modification time of the file.
- **Field 7 – File name**: The last field is the name of the file. In this example, the file name is 1.c.

Show Hidden Files

By default, the ls command will not show hidden files. In Linux, a hidden file is any file that begins with a dot (`.`).

To display all files including the hidden files use the `-a` option:

```
ls -la ~/
```

```
drwxr-x--- 10 linuxize  linuxize  4096 Feb 12 16:28 .
drwxr-xr-x 18 linuxize  linuxize  4096 Dec 26 09:21 ..
-rw-------  1 linuxize  linuxize  1630 Nov 18  2017 .bash_history
drwxr-xr-x  2 linuxize  linuxize  4096 Jul 20  2018  bin
drwxr-xr-x  2 linuxize  linuxize  4096 Jul 20  2018  Desktop
drwxr-xr-x  4 linuxize  linuxize  4096 Dec 12  2017 .npm
drwx------  2 linuxize  linuxize  4096 Mar  4  2018 .ssh
```

We are also using the `-l` option to show the details of all of the files.

How to Sort the ls Output

As we already mentioned, by default the ls command is listing the files in alphabetical order.

- `--sort=extension` **(or `-X`)** - sort alphabetically by entry extension
- `--sort=size` **(or `-S`)** - sort by file size
- `--sort=time` **(or `-t`)** - sort by modification time

If you want to get the results in the reverse sort order use the `-r` option.

For example, to sort the files in the `/var` directory by modification time in the reverse sort order you would use:

```
ls -ltr /var
```

It's worth mentioning that the ls command is not showing the total space occupied by the contents of the directory.

List Subdirectories Recursively

Use the `-R` argument to tell the ls command to display the contents of the subdirectories:

```
ls -R
```

Display File Inode Number Using ls -i

Sometimes you may want to know the inode number of a file for internal maintenance. Use -i option as shown below to display inode number. Using inode number you can remove files that has special characters in it's name.

$ ls –i

which Command in Linux

Which command is very small and simple command to locate executables in the system. It allows user to pass several command names as arguments to get their paths in the system. "which" commands searches the path of executable in system paths set in $PATH environment variable.

Syntax:

```
$ which [-option]
```

For example,

```
$ which ls gdb open grep

/bin/ls

/usr/bin/gdb

/bin/open

/bin/grep
```

It locates command names – "ls", "gdb", "open" and "grep" specified as arguments to "which" command and displays paths of each executable where it exists in the system.

Display all the paths using -a option

"which" command gives option "-a" that displays all paths of executable matching to argument.

```
$ which echo
/usr/sbin/echo
```

Above will search display the executable "echo" from all paths set in $PATH environment variable and displays the first path where echo executable is found. It may be case that executable is placed at other paths of $PATH environment variable as well. To get all paths where executable is present in the system, "-a" option can be used.

```
$ which -a  echo
/usr/sbin/echo

/bin/echo
```

whatis Example

Display single line information about a command
Use whatis command to display single line information about all the keyword matches of passwd.

```
$ whatis ifconfig

ifconfig            (8)  - configure a network interface
```

When there are more than one man page available for a particular command, it will displays single line information from all those man pages as shown below.

```
$ whatis passwd

passwd              (1)  - update a user's authentication tokens(s)

passwd              (5)  - password file

passwd [sslpasswd]   (1ssl)  - compute password hashes
```

whereis command in Linux with Examples

whereis command is used to find the location of source/binary file of a command and manuals sections for a specified file in Linux system.

Syntax
The basic syntax is as follows:

```
whereis command
whereis program
whereis [options] program
whereis -BMS directory  -f command
```

How do I only search for binary files?

Try passing the -b option as follows:

```
$ whereis -b date
```

How do I only search for manual sections files?

Use the -m option as follows:

```
$ whereis -m date
```

How do I only search for source code files?

Pass the -s option as follows:

```
$ whereis -s date
```

Assignment

1. Identify the types of files in Linux?
2. Maximum length for the name of file?
3. Which characters are not allowed in the name of file?