

## FILE TYPES IN LINUX

### Identifying Linux File types

There is only 1 command you need to know, which will help you to identify and categorize all the seven different file types found on the Linux system.

```
$ ls -ld <file name>
```

Here is an example output of the above command.

```
$ ls -ld /etc/services
-rw-r--r-- 1 root root 19281 Feb 14 2012 /etc/services
```

**ls** [command](#) will show the file type as an encoded symbol found as the first character of the file permission part. In this case it is "-", which means "regular file". It is important to point out that Linux file types are not to be mistaken with file extensions. Let us have a look at a short summary of all the seven different types of Linux file types and **ls** command identifiers:

1. **-** : regular file
2. **d** : directory
3. **c** : character device file
4. **b** : block device file
5. **s** : local socket file
6. **p** : named pipe
7. **l** : symbolic link

### 8. Regular file

9. The regular file is a most common file type found on the Linux system. It governs all different files such as text files, images, binary files, shared libraries, etc. You can create a regular file with the **touch** command:

```
10. $ touch linuxcareer.com
    $ ls -ld linuxcareer.com
    -rw-rw-r-- 1 lubos lubos 0 Jan 10 12:52 linuxcareer.com
```

11. The first character of the **ls** command, in this case "-", denotes the identification code for the regular file. To remove a regular file you can use the **rm** command:

```
12. $ rm linuxcareer.com
    $
```

### 13. Directory

14. Directory is second most common file type found in Linux. Directory can be created with the **mkdir** command:

15. `$ mkdir FileTypes`

`$ ls -ld FileTypes/`

`drwxrwxr-x 2 lubos lubos 4096 Jan 10 13:14 FileTypes/`

16. As explained earlier, directory can be identified by "d" symbol from the `ls` command output. To remove empty directory use the `rmdir` command.

```
17.$ rmdir FileTypes
```

18. When trying to remove directory with the `rmdir` command, which contains additional files you will get an error message:

```
19.rmdir: failed to remove `FileTypes/': Directory not empty
```

20. In this case you need to use a command:

```
21.$ rm -r FileTypes/
```

## 22. Character device

23. Character and block device files allow users and programs to communicate with hardware peripheral devices. For example:

```
24.$ ls -ld /dev/vmmon
```

```
crw----- 1 root root 10, 165 Jan  4 10:13 /dev/vmmon
```

25. In this case the character device is the vmware module device.

## 26. Block Device

27. Block devices are similar to character devices. They mostly govern hardware as hard drives, memory, etc.

```
28.$ ls -ld /dev/sda
```

```
brw-rw---- 1 root disk 8, 0 Jan  4 10:12 /dev/sda
```

## Local domain sockets

Local domain sockets are used for communication between processes. Generally, they are used by services such as X windows, syslog and etc.

```
$ ls -ld /dev/log
```

```
srw-rw-rw- 1 root root 0 Jan  4 10:13 /dev/log
```

Sockets can be created by socket system call and removed by the `unlink` or `rm` commands.

## Named Pipes

Similarly as Local sockets, named pipes allow communication between two local processes. They can be created by the `mknod` command and removed with the `rm` command.

## Symbolic Links

With symbolic links an administrator can assign a file or directory multiple identities. Symbolic link can be thought of as a pointer to an original file. There are two types of symbolic links:

- hard links
- soft links

The difference between hard and soft links is that soft links use file name as reference and hard links use direct reference to the original file. Furthermore, hard links cannot cross file systems and partitions. To create symbolic soft link we can use **ln -s** command:

```
$ echo file1 > file1
$ ln -s file1 file2
$ cat file2
file1
$ ls -ld file2
lrwxrwxrwx 1 lubos lubos 5 Jan 10 14:42 file2 -> file1
```

To remove symbolic link we can use **unlink** or **rm** command.

In Unix, there are three basic types of files –

- **Ordinary Files** – An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
- **Directories** – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
- **Special Files** – Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

## Listing Files

To list the files and directories stored in the current directory, use the following command –

```
$ls
```

Here is the sample output of the above command –

```
$ls
```

```
bin          hosts  lib      res.03
ch07         hw1    pub      test_results
ch07.bak     hw2    res.01   users
docs        hw3    res.02   work
```

The command **ls** supports the **-l** option which would help you to get more information about the listed files –

```
$ls -l
```

```
total 1962188
```

```
drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
```

```

-rw-rw-r-- 1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x 2 amrood amrood      4096 Feb 15 2006 univ
drwxr-xr-x 2 root   root        4096 Dec  9 2007 urlspedia
-rw-r--r-- 1 root   root       276480 Dec  9 2007 urlspedia.tar
drwxr-xr-x 8 root   root        4096 Nov 25 2007 usr
drwxr-xr-x 2      200      300    4096 Nov 25 2007 webthumb-1.01
-rwxr-xr-x 1 root   root        3192 Nov 25 2007 webthumb.php
-rw-rw-r-- 1 amrood amrood     20480 Nov 25 2007 webthumb.tar
-rw-rw-r-- 1 amrood amrood      5654 Aug  9 2007 yourfile.mid
-rw-rw-r-- 1 amrood amrood    166255 Aug  9 2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29 2007 zlib-1.2.3
$

```

Here is the information about all the listed columns –

- **First Column** – Represents the file type and the permission given on the file. Below is the description of all type of files.
- **Second Column** – Represents the number of memory blocks taken by the file or directory.
- **Third Column** – Represents the owner of the file. This is the Unix user who created this file.
- **Fourth Column** – Represents the group of the owner. Every Unix user will have an associated group.
- **Fifth Column** – Represents the file size in bytes.
- **Sixth Column** – Represents the date and the time when this file was created or modified for the last time.
- **Seventh Column** – Represents the file or the directory name.

In the **ls -l** listing example, every file line begins with a **d**, **-**, or **l**. These characters indicate the type of the file that's listed.

Sr.No.	Prefix & Description
1	<p><b>-</b></p> <p>Regular file, such as an ASCII text file, binary executable, or hard link.</p>
2	<p><b>b</b></p> <p>Block special file. Block input/output device file such as a physical hard drive.</p>
3	<p><b>c</b></p> <p>Character special file. Raw input/output device file such as a physical hard drive.</p>
4	<p><b>d</b></p> <p>Directory file that contains a listing of other files and directories.</p>

5	<b>l</b> Symbolic link file. Links on any regular file.
6	<b>p</b> Named pipe. A mechanism for interprocess communications.
7	<b>s</b> Socket used for interprocess communication.

## Metacharacters

Metacharacters have a special meaning in Unix. For example, \* and ? are metacharacters. We use \* to match 0 or more characters, a question mark (?) matches with a single character.

For Example –

```
$ls ch*.doc
```

Displays all the files, the names of which start with **ch** and end with **.doc** –

```
ch01-1.doc  ch010.doc  ch02.doc  ch03-2.doc
ch04-1.doc  ch040.doc  ch05.doc  ch06-2.doc
ch01-2.doc  ch02-1.doc  c
```

Here, \* works as meta character which matches with any character. If you want to display all the files ending with just **.doc**, then you can use the following command –

```
$ls *.doc
```

## Hidden Files

An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.

Some common examples of the hidden files include the files –

- **.profile** – The Bourne shell ( sh) initialization script
- **.kshrc** – The Korn shell ( ksh) initialization script
- **.cshrc** – The C shell ( csh) initialization script
- **.rhosts** – The remote shell configuration file

To list the invisible files, specify the **-a** option to **ls** –

```
$ ls -a
```

```
.          .profile    docs        lib         test_results
..         .rhosts     hosts       pub         users
.emacs     bin         hw1         res.01      work
.exrc      ch07        hw2         res.02
```

```
.kshrc      ch07.bak      hw3      res.03
$
```

- **Single dot (.)** – This represents the current directory.
- **Double dot (..)** – This represents the parent directory.

## Creating Files

You can use the **vi** editor to create ordinary files on any Unix system. You simply need to give the following command –

```
$ vi filename
```

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode. Once you are in the edit mode, you can start writing your content in the file as in the following program –

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

Once you are done with the program, follow these steps –

- Press the key **esc** to come out of the edit mode.
- Press two keys **Shift + ZZ** together to come out of the file completely.

You will now have a file created with **filename** in the current directory.

```
$ vi filename
$
```

## Editing Files

You can edit an existing file using the **vi** editor. We will discuss in short how to open an existing file –

```
$ vi filename
```

Once the file is opened, you can come in the edit mode by pressing the key **i** and then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file –

- **l** key to move to the right side.
- **h** key to move to the left side.
- **k** key to move upside in the file.
- **j** key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

## Display Content of a File

You can use the **cat** command to see the content of a file. Following is a simple example to see the content of the above created file –

```
$ cat filename
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
$
```

You can display the line numbers by using the **-b** option along with the **cat** command as follows –

```
$ cat -b filename
1  This is unix file....I created it for the first time.....
2  I'm going to save this content in this file.
$
```

## Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above –

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns –

- **First Column** – Represents the total number of lines in the file.
- **Second Column** – Represents the total number of words in the file.
- **Third Column** – Represents the total number of bytes in the file. This is the actual size of the file.
- **Fourth Column** – Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax –

```
$ wc filename1 filename2 filename3
```

## Copying Files

To make a copy of a file use the **cp** command. The basic syntax of the command is –

```
$ cp source_file destination_file
```

Following is the example to create a copy of the existing file **filename**.

```
$ cp filename copyfile
$
```

You will now find one more file **copyfile** in your current directory. This file will exactly be the same as the original file **filename**.

## Renaming Files

To change the name of a file, use the **mv** command. Following is the basic syntax –

```
$ mv old_file new_file
```

The following program will rename the existing file **filename** to **newfile**.

```
$ mv filename newfile
$
```

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

## Deleting Files

To delete an existing file, use the **rm** command. Following is the basic syntax –

```
$ rm filename
```

**Caution** – A file may contain useful information. It is always recommended to be careful while using this **Delete** command. It is better to use the **-i** option along with **rm** command.

Following is the example which shows how to completely remove the existing file **filename**.

```
$ rm filename
$
```

You can remove multiple files at a time with the command given below –

```
$ rm filename1 filename2 filename3
$
```

## Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for it when it starts up –

- **stdin** – This is referred to as the *standard input* and the associated file descriptor is 0. This is also represented as STDIN. The Unix program will read the default input from STDIN.
- **stdout** – This is referred to as the *standard output* and the associated file descriptor is 1. This is also represented as STDOUT. The Unix program will write the default output at STDOUT
- **stderr** – This is referred to as the *standard error* and the associated file descriptor is 2. This is also represented as STDERR. The Unix program will write all the error messages at STDERR.
- Unix uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character (*/*), and all other directories are contained below it.
- **Home Directory**
  - The directory in which you find yourself when you first login is called your home directory.
  - You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.
  - You can go in your home directory anytime using the following command –

```
$ cd ~
```
  - \$
  - Here ~ indicates the home directory. Suppose you have to go in any other user's home directory, use the following command –



- `$cd ~username`
- `$`
- To go in your last directory, you can use the following command –
- `$cd -`
- `$`

## ● Absolute/Relative Pathnames

- Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.
- Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.
- Following are some examples of absolute filenames.
- `/etc/passwd`
- `/users/sjones/chem/notes`
- `/dev/rdisk/Os3`
- A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this –
- `chem/notes`
- `personal/res`
- To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory –
- `$pwd`
- `/user0/home/amrood`
- `$`

## ● Listing Directories

- To list the files in a directory, you can use the following syntax –
- `$ls dirname`
- Following is the example to list all the files contained in **/usr/local** directory –
- `$ls /usr/local`

```

• X11      bin      gimp      jikes      sbin
• ace      doc      include   lib        share
• atalk    etc      info      man        ami

```

## ● Creating Directories

- We will now understand how to create directories. Directories are created by the following command –
- `$mkdir dirname`
- Here, **dirname** is the absolute or relative pathname of the directory you want to create. For example, the command –
- `$mkdir mydir`
- `$`
- Creates the directory **mydir** in the current directory. Here is another example –
- `$mkdir /tmp/test-dir`
- `$`
- This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory.
- If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, –
- `$mkdir docs pub`
- `$`

- Creates the directories docs and pub under the current directory.
- **Creating Parent Directories**
- We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, **mkdir** issues an error message as follows –
- `$mkdir /tmp/amrood/test`
- `mkdir: Failed to make directory "/tmp/amrood/test";`
- `No such file or directory`
- `$`
- In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example –
- `$mkdir -p /tmp/amrood/test`
- `$`
- The above command creates all the required parent directories.
- **Removing Directories**
- Directories can be deleted using the **rmdir** command as follows –
- `$rmdir dirname`
- `$`
- **Note** – To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.
- You can remove multiple directories at a time as follows –
- `$rmdir dirname1 dirname2 dirname3`
- `$`
- The above command removes the directories `dirname1`, `dirname2`, and `dirname3`, if they are empty. The **rmdir** command produces no output if it is successful.
- **Changing Directories**
- You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below –
- `$cd dirname`
- `$`
- Here, **dirname** is the name of the directory that you want to change to. For example, the command –
- `$cd /usr/local/bin`
- `$`
- Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path –
- `$cd ../../home/amrood`
- `$`
- **Renaming Directories**
- The **mv (move)** command can also be used to rename a directory. The syntax is as follows –
- `$mv olddir newdir`
- `$`
- You can rename a directory **mydir** to **yourdir** as follows –
- `$mv mydir yourdir`
- `$`
- The directories **.** (dot) and **..** (dot dot)

- The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory, often referred to as the parent directory.
- If we enter the command to show a listing of the current working directories/files and use the **-a option** to list all the files and the **-l option** to provide the long listing, we will receive the following result.
- ```
$ls -la
```
- ```
drwxrwxr-x    4  teacher   class    2048   Jul 16 17:56 .
```
- ```
drwxr-xr-x    60   root      1536   Jul 13 14:18 ..
```
- ```
-----      1  teacher   class    4210   May 1 08:27 .profile
```
- ```
-rwxr-xr-x    1  teacher   class    1948   May 12 13:42 memo
```
- ```
$
```

In this chapter, we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes –

- **Owner permissions** – The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** – The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** – The permissions for others indicate what action all other users can perform on the file.

## The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows –

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024   Nov 2 00:10  myfile
drwxr-xr--  1 amrood   users 1024   Nov 2 00:10  mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) –

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

## File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below –

## Read

Grants the capability to read, i.e., view the contents of the file.

## Write

Grants the capability to modify, or remove the content of the file.

## Execute

User with execute permissions can run a file as a program.

## Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned –

## Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

## Write

Access means that the user can add or delete files from the directory.

## Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

## Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

## Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Sr.No.	Chmod operator & Description
1	<b>+</b> Adds the designated permission(s) to a file or directory.
2	<b>-</b> Removes the designated permission(s) from a file or directory.

3	= Sets the designated permission(s).
---	---

Here's an example using **testfile**. Running **ls -l** on the testfile shows that the file's permissions are as follows –

```
$ls -l testfile
-rwxrwxr-- 1 amrood  users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls -l**, so you can see the permission changes –

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx 1 amrood  users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx 1 amrood  users 1024  Nov 2 00:10  testfile
$chmod g = rx testfile
$ls -l testfile
-rw-r-xrwx 1 amrood  users 1024  Nov 2 00:10  testfile
```

Here's how you can combine these commands on a single line –

```
$chmod o+wx,u-x,g = rx testfile
$ls -l testfile
-rw-r-xrwx 1 amrood  users 1024  Nov 2 00:10  testfile
```

## Using chmod with Absolute Permissions

The second way to modify permissions with the **chmod** command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--

<b>5</b>	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
<b>6</b>	Read and write permission: 4 (read) + 2 (write) = 6	rw-
<b>7</b>	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwX

Here's an example using the testfile. Running **ls -l** on the testfile shows that the file's permissions are as follows –

```
$ls -l testfile
-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls -l**, so you can see the permission changes –

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx 1 amrood users 1024 Nov 2 00:10 testfile
$chmod 043 testfile
$ls -l testfile
----r---wx 1 amrood users 1024 Nov 2 00:10 testfile
```

## Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files –

- **chown** – The **chown** command stands for "**change owner**" and is used to change the owner of a file.
- **chgrp** – The **chgrp** command stands for "**change group**" and is used to change the group of a file.

## Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows –

```
$ chown user filelist
```

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept –

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user **amrood**.

**NOTE** – The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

## Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows –

```
$ chgrp group filelist
```

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept –

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

## SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners' **execute** permission normally resides.

For example, the command –

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd*
$
```

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users –

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following command –

```
$ chmod ug+s dirname
$ ls -l
drwsr-sr-x 2 root root 4096 Jun 19 06:45 dirname
$
```

In this chapter, we will discuss in detail about the Unix environment. An important Unix concept is the **environment**, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

For example, first we set a variable TEST and then we access its value using the **echo** command –

```
$TEST="Unix Programming"
$echo $TEST
```

It produces the following result.

```
Unix Programming
```

Note that the environment variables are set without using the \$ sign but while accessing them we use the \$ sign as prefix. These variables retain their values until we come out of the shell.

When you log in to the system, the shell undergoes a phase called **initialization** to set up the environment. This is usually a two-step process that involves the shell reading the following files –

- /etc/profile
- profile

The process is as follows –

- The shell checks to see whether the file **/etc/profile** exists.
- If it exists, the shell reads it. Otherwise, this file is skipped. No error message is displayed.
- The shell checks to see whether the file **.profile** exists in your home directory. Your home directory is the directory that you start out in after you log in.
- If it exists, the shell reads it; otherwise, the shell skips it. No error message is displayed.

As soon as both of these files have been read, the shell displays a prompt –

```
$
```

This is the prompt where you can enter commands in order to have them executed.

**Note** – The shell initialization process detailed here applies to all **Bourne** type shells, but some additional files are used by **bash** and **ksh**.

## The .profile File

The file **/etc/profile** is maintained by the system administrator of your Unix machine and contains shell initialization information required by all users on a system.

The file **.profile** is under your control. You can add as much shell customization information as you want to this file. The minimum set of information that you need to configure includes –

- The type of terminal you are using.



- A list of directories in which to locate the commands.
- A list of variables affecting the look and feel of your terminal.

You can check your **.profile** available in your home directory. Open it using the vi editor and check all the variables set for your environment.

## Setting the Terminal Type

Usually, the type of terminal you are using is automatically configured by either the **login** or **getty** programs. Sometimes, the auto configuration process guesses your terminal incorrectly.

If your terminal is set incorrectly, the output of the commands might look strange, or you might not be able to interact with the shell properly.

To make sure that this is not the case, most users set their terminal to the lowest common denominator in the following way –

```
$TERM=vt100
$
```

## Setting the PATH

When you type any command on the command prompt, the shell has to locate the command before it can be executed.

The PATH variable specifies the locations in which the shell should look for commands. Usually the Path variable is set as follows –

```
$PATH=/bin:/usr/bin
$
```

Here, each of the individual entries separated by the colon character (:) are directories. If you request the shell to execute a command and it cannot find it in any of the directories given in the PATH variable, a message similar to the following appears –

```
$hello
hello: not found
$
```

There are variables like PS1 and PS2 which are discussed in the next section.

## PS1 and PS2 Variables

The characters that the shell displays as your command prompt are stored in the variable PS1. You can change this variable to be anything you want. As soon as you change it, it'll be used by the shell from that point on.

For example, if you issued the command –

```
$PS1='=>'
=>
=>
=>
```

Your prompt will become =>. To set the value of **PS1** so that it shows the working directory, issue the command –

```
=>PS1="[\u@\h \w]\$"  
[root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$  
[root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$
```

The result of this command is that the prompt displays the user's username, the machine's name (hostname), and the working directory.

There are quite a few **escape sequences** that can be used as value arguments for PS1; try to limit yourself to the most critical so that the prompt does not overwhelm you with information.

Sr.No .	Escape Sequence & Description
1	<b>\t</b> Current time, expressed as HH:MM:SS
2	<b>\d</b> Current date, expressed as Weekday Month Date
3	<b>\n</b> Newline
4	<b>\s</b> Current shell environment
5	<b>\W</b> Working directory
6	<b>\w</b> Full path of the working directory
7	<b>\u</b> Current user's username
8	<b>\h</b> Hostname of the current machine

9	<code>\#</code> Command number of the current command. Increases when a new command is entered
10	<code>\\$</code> If the effective UID is 0 (that is, if you are logged in as root), end the prompt with the # character; otherwise, use the \$ sign

You can make the change yourself every time you log in, or you can have the change made automatically in PS1 by adding it to your **.profile** file.

When you issue a command that is incomplete, the shell will display a secondary prompt and wait for you to complete the command and hit **Enter** again.

The default secondary prompt is > (the greater than sign), but can be changed by re-defining the **PS2** shell variable –

Following is the example which uses the default secondary prompt –

```
$ echo "this is a
> test"
this is a
test
$
```

The example given below re-defines PS2 with a customized prompt –

```
$ PS2="secondary prompt->"
$ echo "this is a
secondary prompt->test"
this is a
test
$
```

## Environment Variables

Following is the partial list of important environment variables. These variables are set and accessed as mentioned below –

Sr.No	Variable & Description
1	<b>DISPLAY</b> Contains the identifier for the display that <b>X11</b> programs should use by default.
2	<b>HOME</b>

	Indicates the home directory of the current user: the default argument for the <b>cd built-in</b> command.
3	<b>IFS</b> Indicates the <b>Internal Field Separator</b> that is used by the parser for word splitting after expansion.
4	<b>LANG</b> LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is <b>pt_BR</b> , then the language is set to (Brazilian) Portuguese and the locale to Brazil.
5	<b>LD_LIBRARY_PATH</b> A Unix system with a dynamic linker, contains a colon-separated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
6	<b>PATH</b> Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
7	<b>PWD</b> Indicates the current working directory as set by the cd command.
8	<b>RANDOM</b> Generates a random integer between 0 and 32,767 each time it is referenced.
9	<b>SHLVL</b> Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.
10	<b>TERM</b> Refers to the display type.
11	<b>TZ</b> Refers to Time zone. It can take values like GMT, AST, etc.

12

## UID

Expands to the numeric user ID of the current user, initialized at the shell startup.

Following is the sample example showing few environment variables –

```
$ echo $HOME
/root
]$ echo $DISPLAY

$ echo $TERM
xterm
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin
$
```

To create an empty file named file1 in the /tmp directory, enter the following commands:

```
$ cd /tmp
$ touch space
$ ls -l file1
$ ls -l file
-rw-r--r-- 1 geek wheel 0 Dec 13 22:05 file
```

To create multiple empty files use the touch commands with the file names in one line as shown below.

```
$ touch file1 file2 file3

$ ls -lrt file*
-rw-r--r-- 1 geek wheel 0 Dec 13 22:19 file2
-rw-r--r-- 1 geek wheel 0 Dec 13 22:19 file1
-rw-r--r-- 1 geek wheel 0 Dec 13 22:19 file3
```

## Creating Directories

The mkdir command creates new directories.

```
$ mkdir directory_name
```

and/or

```
$ mkdir -p directory_names
```

Include the `-p` option if the directory name includes a path name. The command used with the `-p` option creates all of the non-existing parent directories that do not yet exist in the path to the new directory. You can use absolute or relative path names on the command line when creating new directories.

For example, create a new directory, named `dir1`, within the `/tmp` directory.

```
$ cd /tmp
$ mkdir dir1
```

You can use the command `'ls -ld'` to view the created directory.

```
$ ls -ld dir1
drwxr-xr-x  2 geek  wheel  64 Dec 13 22:26 dir1
```

To create a new directory named `dir_in` located inside a directory named `dir_out`, use the `mkdir` command with the `-p` option. The `dir_out` directory does not yet exist.

```
$ mkdir -p dir_out/dir_in
```

To create the `dir1`, `dir2`, and `dir3` directories, enter the `mkdir` command with all the directory names in one line as shown below.

```
$ mkdir dir1 dir2 dir3

$ ls -ld dir*
drwxr-xr-x  2 sandy  wheel  64 Dec 13 22:26 dir1
drwxr-xr-x  2 sandy  wheel  64 Dec 13 22:28 dir2
drwxr-xr-x  2 sandy  wheel  64 Dec 13 22:28 dir3
```

## Removing Files

You can permanently remove files from the directory hierarchy with the `rm` command.

```
$ rm -option filename
```

The `rm` command is a destructive command if not used with the correct option. The table describes the options that you can use with the `rm` command when removing files and directories.

Option	Description
<code>-r</code>	Includes the contents of a directory and the contents of all subdirectories when you remove a directory

-i	Prevents the accidental removal of existing files or directories
----	--

The **-r** option allows you to remove directories that contain files and subdirectories. The **-i** option prompts you for confirmation before removing any file.

- A **yes** response completes the removal of the file.
- A **no** response aborts the removal of the file.

For example, remove the file named file1 from the /tmp directory.

```
$ cd /tmp
$ rm file1
```

Lets see an example of using the -i option to delete the files.

```
$ rm -i file2
remove file2? y
```

## Removing Directories

You can use the rm command with the **-r** option to remove directories that contain files and subdirectories.

```
$ rm -options directories
```

For example, remove the dir1 directory and its content by using the rm -r command.

```
$ cd /tmp
$ rm -r dir1

$ ls -ld dir1
ls: dir1: No such file or directory
```

If you do not use the -r option with the rm command while removing directories, the following error message appears:

```
rm: directoryname: is a directory.
```

To interactively remove a directory and its contents, use the -i option along with the rm -r command. For example,

```
$ rm -ir dir2
examine files in directory dir2? y
remove dir2/file2? y
remove dir2/file1? y
remove dir2? y
```

The **rmdir** command removes empty directories.

```
$ rmdir directories
```

For example to remove the empty directory dir3, use the command below.

```
$ cd /tmp  
$ rmdir dir3
```

To remove a directory in which you are currently working in, you must first change to its parent directory.

## Create, Copy, Rename, and Remove Unix Files and Directories

This document lists commands for creating, copying, renaming and removing Unix files and directories. It assumes you are using Unix on the ITS Login Service (login.itd.umich.edu). The instructions here apply to many other Unix machines; however, you may notice different behavior if you are not using the ITS Login Service.

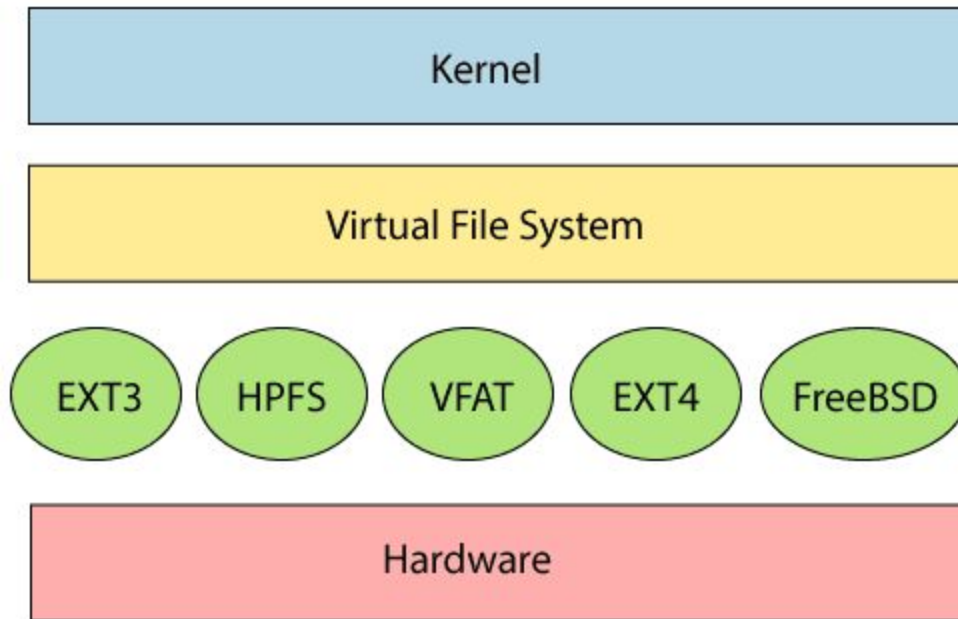
## Linux File System

The [Linux](#) file system contains the following sections: A Linux file system is a structured collection of files on a disk drive or a partition. A partition is a segment of memory and contains some specific data. In our machine, there can be various partitions of the memory. Generally, every partition contains a file system.

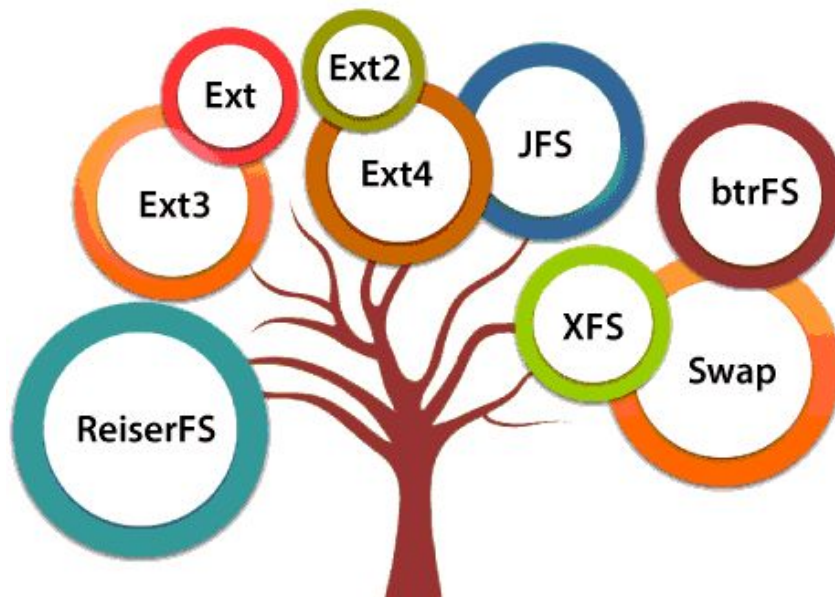
The general-purpose computer system needs to store data systematically so that we can easily access the files in less time. It stores the data on hard disks (HDD) or some equivalent storage type. There may be below reasons for maintaining the file system:

- o Primarily the computer saves data to the RAM storage; it may lose the data if it gets turned off. However, there is non-volatile RAM (Flash RAM and SSD) that is available to maintain the data after the power interruption.
- o Data storage is preferred on hard drives as compared to standard RAM as RAM costs more than disk space. The hard disks costs are dropping gradually comparatively the RAM.





## Types of Linux File System



Linux Create

## File

Linux file system considers everything as a file in Linux; whether it is text file images, partitions, compiled programs, directories, or hardware devices. If it is not a file, then it must be a process. To manage the data, it forms a tree structure.

Linux files are case sensitive, so **test.txt** and **Test.txt** will be considered as two different files. There are multiple ways to create a file in Linux. Some conventional methods are as follows:

- o [using cat command](#)
- o [using touch command](#)
- o [using redirect '>' symbol](#)
- o [using echo command](#)
- o [using printf command](#)
- o [using a different text editor like vim, nano, vi](#)

Apart from all of the above methods, we can also create a file from the desktop file manager. Let's understand the above methods in detail:

## 1. Using cat command

The cat command is one of the most used [commands in Linux](#). It is used to **create a file, display the content of the file, concatenate the contents of multiple files, display the line numbers**, and more.

Here, we will see how to create files and add content to them using [cat command](#).

First of all, create a directory and named it as **New\_directory**, execute the **mkdir** command as follows:

1. mkdir New\_directory  
Change directory to it:

1. cd New\_directory

**Output:**

```
javatpoint@javatpoint-GB-BXBT-2807:~$ mkdir New_directory
javatpoint@javatpoint-GB-BXBT-2807:~$ cd New_directory
```

Now execute the cat command to create a file:

1. cat > test.txt

The above command will create a text file and will enter in the editor mode. Now, enter the desired text and press **CTRL + D** key to save and exit the file and it will return to the command line.

To display the content of the file, execute the cat command as follows:

1. cat test.txt

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~/New_directory$ cat > test.txt
This is a text file
created using cat command
javatpoint@javatpoint-GB-BXBT-2807:~/New_directory$ cat test.txt
This is a text file
created using cat command
```

## 2. Using the touch command

The **touch** command is also one of the popular commands in Linux. It is used to **create a new file, update the time stamp on existing files and directories**. It can also create empty files in Linux.

The [touch command](#) is the simplest way to create a new file from the command line. We can create multiple files by executing this command at once.

To create a file, execute the touch command followed by the file name as given below:

1. touch test1.txt

To list the information of the created file, execute the below command:

1. ls -l test1.txt

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ touch test1.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test1.txt
-rw-r--r-- 1 javatpoint javatpoint 0 Feb 21 14:17 test1.txt
```

To create multiple files at once, specify files and their extensions after the touch command along with single space. Execute the below command to create three files at once:

1. touch test1.txt test2.txt test3.txt

To create two different types of file, execute the command as follows:

1. touch test4.txt test.odt

The above command will create two different files named as **test4.txt** and **test.odt**.

To display the file and its timestamp, execute the **ls** command as follows:

1. ls -l

Consider the below output:

```

javatpoint@javatpoint-GB-BXBT-2807:~$ touch test1.txt test2.txt test3.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls
Desktop    Downloads  New_directory  Public    test1.txt  test3.txt
Documents  Music      Pictures        Templates test2.txt  Videos
javatpoint@javatpoint-GB-BXBT-2807:~$ touch test4.txt test.odt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l
total 36
drwxr-xr-x 3 javatpoint javatpoint 4096 Feb  6 15:12 Desktop
drwxr-xr-x 2 javatpoint javatpoint 4096 Feb 21 15:50 Documents
drwxr-xr-x 3 javatpoint javatpoint 4096 Feb 21 10:51 Downloads
drwxr-xr-x 4 javatpoint javatpoint 4096 Jan 28 09:48 Music
drwxr-xr-x 2 javatpoint javatpoint 4096 Feb 20 17:02 New_directory
drwxr-xr-x 4 javatpoint javatpoint 4096 Feb 21 15:32 Pictures
drwxr-xr-x 2 javatpoint javatpoint 4096 Jan  9 13:10 Public
drwxr-xr-x 2 javatpoint javatpoint 4096 Jan  9 13:10 Templates
-rw-r--r-- 1 javatpoint javatpoint  0 Feb 21 15:31 test1.txt
-rw-r--r-- 1 javatpoint javatpoint  0 Feb 21 15:31 test2.txt
-rw-r--r-- 1 javatpoint javatpoint  0 Feb 21 15:31 test3.txt
-rw-r--r-- 1 javatpoint javatpoint  0 Feb 21 15:51 test4.txt
-rw-r--r-- 1 javatpoint javatpoint  0 Feb 21 15:51 test.odt
drwxr-xr-x 2 javatpoint javatpoint 4096 Jan  9 13:10 Videos

```

If we pass the name of an existing file, it will change the timestamp of that file.

*Note: The significant difference between touch and cat command is that using cat command, we can specify the content of the file from the command prompt comparatively the touch command creates a blank file.*

### 3. Using the redirect (>) symbol

We can also create a file using the redirect symbol (>) on the command line. To create a file, we just have to type a redirect symbol (>) followed by the file name. This symbol is mostly used to redirect the output. There are two ways to redirect the output. If we use > **operator**, it will overwrite the existing file, and >> operator will append the output.

To create a file with redirect (>) operator, execute the command as follows:

1. > test5.txt

The above command will create a file, to display the existence of the created file, execute the below command:

1. ls -l test5.txt

Consider the below output:

```

javatpoint@javatpoint-GB-BXBT-2807:~$ > test5.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test5.txt
-rw-r--r-- 1 javatpoint javatpoint 0 Feb 24 10:01 test5.txt

```



## 4. Using echo command

The **echo** command is used to create a file, but we should specify the file content on the command line.

To create the file with the echo command, execute the command as follows:

1. `echo " File content" > test6.txt`

The above command will create the **test6** file. To display the existence of the file, execute the below command:

1. `ls -l test6.txt`  
consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ echo " This is test6 file created by echo command" > test6.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test6.txt
-rw-r--r-- 1 javatpoint javatpoint 44 Feb 24 11:33 test6.txt
javatpoint@javatpoint-GB-BXBT-2807:~$
```

## 5. Using printf command

We can also create a file using **printf** command. For this we need to specify the file content on the command line.

To create a file with the printf command, execute the command as follows:

1. `printf " File content" > test7.txt`

To display the file details, execute the ls command as follows:

1. `ls -l test7.txt`

To display the file content, execute the cat command as follows:

1. `cat test7.txt`

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ printf " This is test7 file created by printf command" > test7.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test7.txt
-rw-r--r-- 1 javatpoint javatpoint 45 Feb 24 11:46 test7.txt
```

## 6. Using Text Editor

We can also create a file using the different text editors like **vim**, **nano**, **vi**, and more.

### o Using Vim text editor

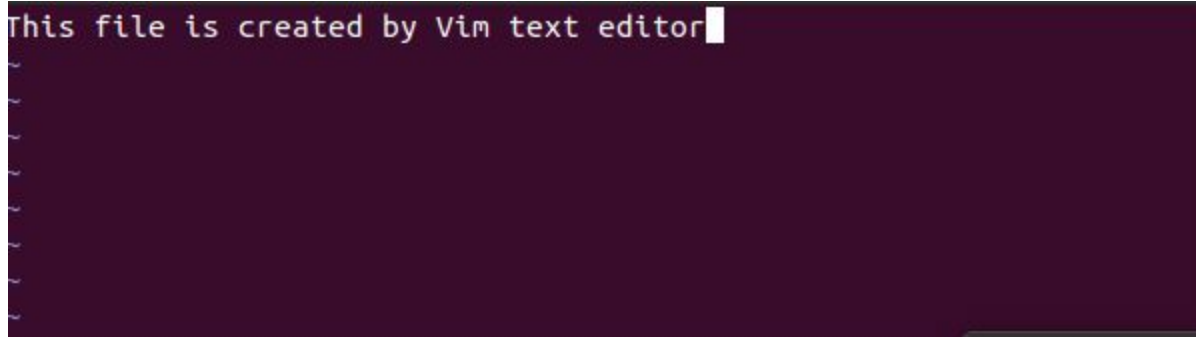
We can create a file using the **Vim text editor**. If you do not have the vim editor installed on your machine, execute the below command:

To create a file using the vim text editor, execute the below command:

1. `vim test8.txt`

The above command will open the text editor, press `i` key to go to the insert mode of the editor.

Enter the file content, press **Esc key** preceded by `:wq` to save and exit the file. The text editor looks like as follows:



To display the file information, execute the **ls** command as follows:

1. `ls -l test8.txt`

To view the file content, run the cat command as follows:

1. `cat test8.txt`

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ vim test8.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test8.txt
-rw-r--r-- 1 javatpoint javatpoint 40 Feb 24 12:21 test8.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ cat test8.txt
This file is created by Vim text editor
```

## 7. Using Nano editor

We can create a file using the **nano** text editor. To create a file, execute the below command:

1. `nano test9.txt`

The above command will open the nano text editor. Enter the desired text and press **CTRL + X** then type `y` for confirmation of the file changes. Press **Enter key** to exit from the editor.

The nano text editor looks like the below image:

```
GNU nano 4.3                                test9.txt                                Modified
This file is created using nano text editor
```

To display the file information, execute the below command:

1. `ls -l test9.txt`

To view the file content, execute the below command:

1. `cat test9.txt`

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ nano test9.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test9.txt
-rw-r--r-- 1 javatpoint javatpoint 44 Feb 24 14:15 test9.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ cat test9.txt
This file is created using nano text editor
```

## Using Vi editor

To create a file with Vi editor, execute the below command:

1. `vi test10.txt`

The above command will open the Vi editor. Press `i` key for the insert mode and enter the file content. Press `Esc` key and `:wq` to save and exit the file from the editor.

To display the file information, execute the below command:

1. `ls -l test10.txt`

To display the file content, execute the below command:

1. `cat test10.txt`

Consider the below output:

```
javatpoint@javatpoint-GB-BXBT-2807:~$ vi test10.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ ls -l test10.txt
-rw-r--r-- 1 javatpoint javatpoint 43 Feb 24 15:52 test10.txt
javatpoint@javatpoint-GB-BXBT-2807:~$ cat test10.txt
This file is created using Vi text editor.
```

## Linux file command

`file` command is used to determine the file type. It does not care about the extension used for file. It simply uses `file` command and tell us the file type. It has several options.

### Syntax:

1. `file <filename>`

### Example:

1. file 1.png

```
sssit@JavaTpoint: ~/Desktop
sssit@JavaTpoint:~/Desktop$ file jdk-8u91-linux-i586.rpm
jdk-8u91-linux-i586.rpm: RPM v3.0 bin i386/x86_64
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ file 1.png
1.png: PNG image data, 724 x 463, 8-bit/color RGBA, non-interlaced
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ file linux.docx
linux.docx: Microsoft Word 2007+
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ file linuxfun.pdf
linuxfun.pdf: PDF document, version 1.4
sssit@JavaTpoint:~/Desktop$ file usr
usr: directory
sssit@JavaTpoint:~/Desktop$
```

In above snapshot, you can see file command along with different arguments, specifying their file types.

**Note:** File command tell us the file type with the help of a magic file that contains all the patterns to recognize a file type. Path of magic file is /usr/share/file/magic. For more information enter the command 'man 5 magic'.

## Linux File Command Options

Option	Function
<a href="#">file -s</a>	Used for special files.
<a href="#">file *</a>	Used to list types of all the files.
<a href="#">file /directory name/*</a>	Used to list types of all the files from mentioned directory.
<a href="#">file [range]*</a>	It will list out all the files starting from the alphabet present within the given range.

## Linux touch command

touch command is a way to create empty files (there are some other mehtods also). You can update the modification and access time of each file with the help of touch command.

### Syntax:

1. touch <filename>

### Example:

1. touch myfile1



```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music  Pictures  Templates  
Desktop    Documents  examples.desktop  office  Public    Videos  
sssit@JavaTpoint:~$ touch myfile1  
sssit@JavaTpoint:~$ touch myfile2  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music  myfile2  Pictures  Templates  
Desktop    Documents  examples.desktop  myfile1  office    Public    Videos  
sssit@JavaTpoint:~$
```

Look above, we have created two files namely 'myfile1' and 'myfile2' through touch command. To create multiple files just type all the file names with a single touch command followed by enter key. For example, if you would like to create 'myfile1' and 'myfile2' simultaneously, then your command will be:

1. touch myfile1 myfile2

## touch Options

Option	Function
<a href="#">touch -a</a>	To change file access and modification time.
<a href="#">touch -m</a>	It is used to only modify time of a file.
<a href="#">touch -r</a>	To update time of one file with reference to the other file.
<a href="#">touch -t</a>	To create a file by specifying the time.
<a href="#">touch -c</a>	It doesn't create an empty file.

## Linux touch -a command

touch command with option 'a' is used to change the access time of a file. By default, it will take the current time of your system.

### Syntax:

1. touch -a <filename>

### Example:

1. touch -a usr

To see the access and change time of your file, you need to use **stat** command.

```
sssit@JavaTpoint: ~/Desktop
sssit@JavaTpoint:~/Desktop$ stat usr
  File: `usr'
  Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 807h/2055d    Inode: 655963       Links: 3
Access: (0755/drwxr-xr-x)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 11:03:31.579478467 +0530
Modify: 2016-05-11 17:55:13.834427646 +0530
Change: 2016-05-24 10:39:45.873020113 +0530
 Birth: -
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ touch -a usr
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ stat usr
  File: `usr'
  Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 807h/2055d    Inode: 655963       Links: 3
Access: (0755/drwxr-xr-x)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 16:50:14.663712815 +0530
Modify: 2016-05-11 17:55:13.834427646 +0530
Change: 2016-05-25 16:50:14.663712815 +0530
 Birth: -
sssit@JavaTpoint:~/Desktop$
```

In above snapshot we have used 'stat' command (which we'll learn in later tutorial) just to check the status of our directory (usr). So don't get confused with that. Now you can match the access time of directory (usr) before and after passing the command 'touch -a usr'. It has taken the default access time of our system.

## Linux touch -m command

The touch '-m' option will help you to change only the modification time of a file.

### Syntax:

1. touch -m <filename>

### Example:

1. touch -m usr

```

sssit@JavaTpoint: ~/Desktop
sssit@JavaTpoint:~/Desktop$ stat linuxfun.pdf
  File: `linuxfun.pdf'
  Size: 7022363      Blocks: 13720      IO Block: 4096   regular file
Device: 807h/2055d   Inode: 658072      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 11:07:36.806939775 +0530
Modify: 2016-05-11 18:40:09.000000000 +0530
Change: 2016-05-25 11:07:35.790939820 +0530
 Birth: -
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ touch -m linuxfun.pdf
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ stat linuxfun.pdf
  File: `linuxfun.pdf'
  Size: 7022363      Blocks: 13720      IO Block: 4096   regular file
Device: 807h/2055d   Inode: 658072      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 11:24:49.268540162 +0530
Modify: 2016-05-25 11:24:48.256540203 +0530
Change: 2016-05-25 11:24:48.256540203 +0530
 Birth: -
sssit@JavaTpoint:~/Desktop$

```

Notice carefully in the above snapshot, only modification time has been changed.

## Linux touch -r option

This command will update time with reference to the other mentioned command. There are two ways to use this command. Both works the same.

In below example, we want to change time-stamp of '**Demo.txt**' with reference to '**demo.txt**'. Firstyou can write it as,

- 1.
  2. touch -r demo.txt Demo.txt
- or

- 1.
2. touch Demo.txt -r demo.txt

First we'll see the status of both the files before using touch comand.



```
sssit@JavaTpoint: ~/Downloads
sssit@JavaTpoint:~/Downloads$ stat demo.txt
  File: `demo.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 807h/2055d Inode: 658414       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 10:45:00.276624267 +0530
Modify: 2016-05-24 10:37:35.265025647 +0530
Change: 2016-05-25 10:44:59.264624225 +0530
 Birth: -
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ stat Demo.txt
  File: `Demo.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 807h/2055d Inode: 658511       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 10:43:52.356621370 +0530
Modify: 2016-05-24 10:37:01.761027070 +0530
Change: 2016-05-24 10:37:22.945026171 +0530
 Birth: -
sssit@JavaTpoint:~/Downloads$
```

Now after using **touch -r demo.txt Demo.txt** command, time of **Demo.txt** has been changed with reference to time of **demo.txt**

```
sssit@JavaTpoint: ~/Downloads
sssit@JavaTpoint:~/Downloads$ touch -r demo.txt Demo.txt
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ stat Demo.txt
  File: `Demo.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 807h/2055d Inode: 658511       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 10:45:00.276624267 +0530
Modify: 2016-05-24 10:37:35.265025647 +0530
Change: 2016-05-25 11:51:51.924471400 +0530
 Birth: -
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ stat demo.txt
  File: `demo.txt'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 807h/2055d Inode: 658414       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 10:45:00.276624267 +0530
Modify: 2016-05-24 10:37:35.265025647 +0530
Change: 2016-05-25 10:44:59.264624225 +0530
 Birth: -
sssit@JavaTpoint:~/Downloads$
```

Linux touch -t command

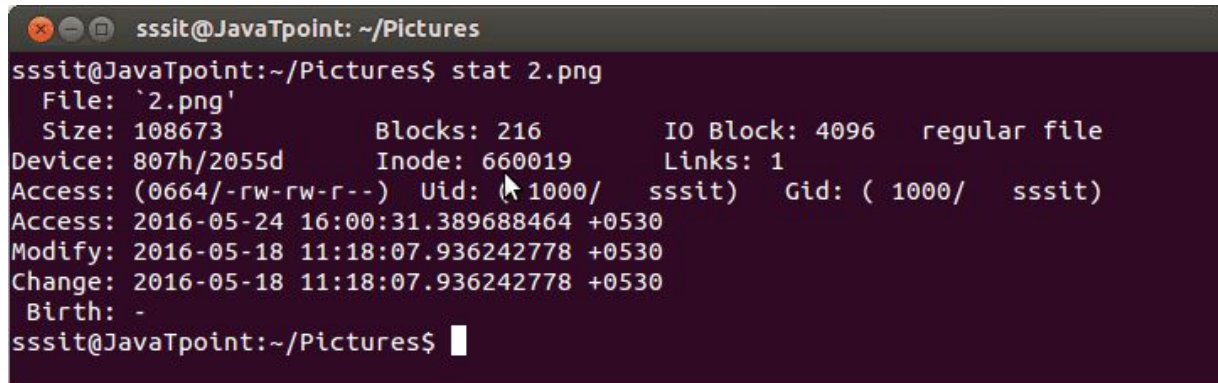
with this command, you can change the access time of a file by determining a specified time to it.

It will modify the time by specified time instead of default time.

Format of time will be:

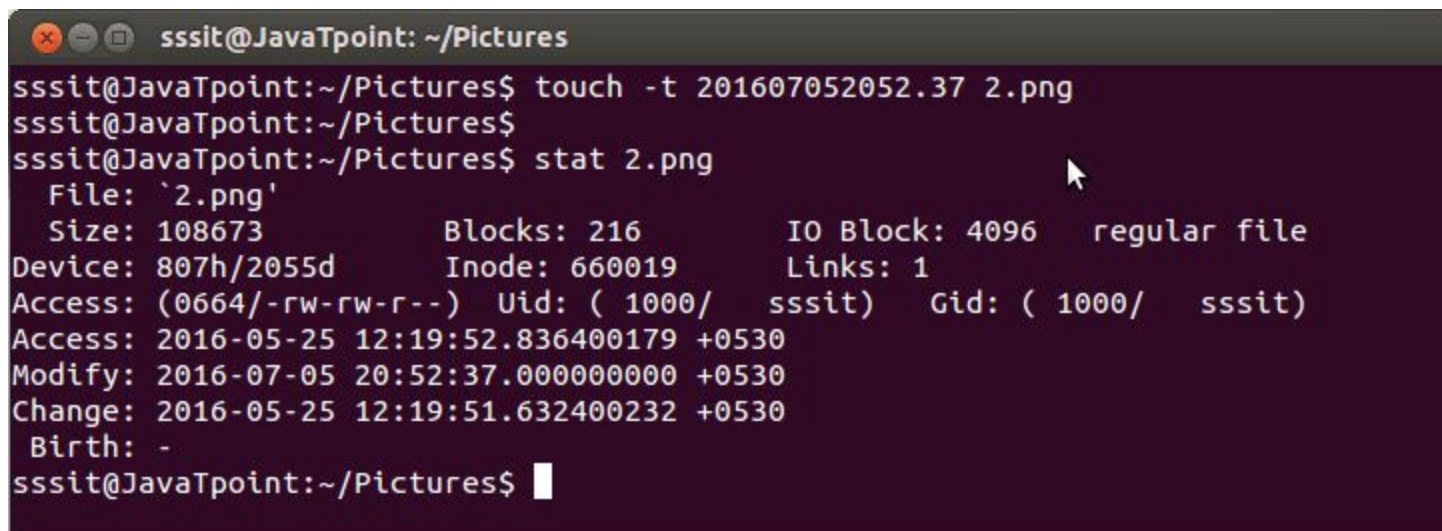
1. touch -t YYYYMMDDhhmm.ss

Below screenshot shows status of file 2.png before the touch command,



```
sssit@JavaTpoint: ~/Pictures
sssit@JavaTpoint:~/Pictures$ stat 2.png
  File: `2.png'
  Size: 108673      Blocks: 216      IO Block: 4096   regular file
Device: 807h/2055d  Inode: 660019    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-24 16:00:31.389688464 +0530
Modify: 2016-05-18 11:18:07.936242778 +0530
Change: 2016-05-18 11:18:07.936242778 +0530
 Birth: -
sssit@JavaTpoint:~/Pictures$
```

This screenshot shows that time of file 2.png has been modified by our specified time.



```
sssit@JavaTpoint: ~/Pictures
sssit@JavaTpoint:~/Pictures$ touch -t 201607052052.37 2.png
sssit@JavaTpoint:~/Pictures$
sssit@JavaTpoint:~/Pictures$ stat 2.png
  File: `2.png'
  Size: 108673      Blocks: 216      IO Block: 4096   regular file
Device: 807h/2055d  Inode: 660019    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   sssit)   Gid: ( 1000/   sssit)
Access: 2016-05-25 12:19:52.836400179 +0530
Modify: 2016-07-05 20:52:37.000000000 +0530
Change: 2016-05-25 12:19:51.632400232 +0530
 Birth: -
sssit@JavaTpoint:~/Pictures$
```

## Linux touch -c command

Using 'c' option with touch command will not create an empty file, if that file doesn't exist.

### Syntax:

1. touch -c <filename>

### Example:

1. touch -c movie

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music      myfile2    Pictures   Templates  
Desktop    Documents  examples.desktop  myfile1    office     Public     Videos  
sssit@JavaTpoint:~$  
sssit@JavaTpoint:~$ touch -c movie  
sssit@JavaTpoint:~$  
sssit@JavaTpoint:~$ ls  
cretecler  Disk1      Downloads  Music      myfile2    Pictures   Templates  
Desktop    Documents  examples.desktop  myfile1    office     Public     Videos  
sssit@JavaTpoint:~$
```

In above figure, we wanted to create file 'movie' but with 'c' option no file has been created.

## What Are Unix Files and Directories?

A file is a "container" for data. Unix makes no distinction among file types—a file may contain the text of a document, data for a program or the program itself.

Directories provide a way to organize files, allowing you to group related files together. Directories may contain files and/or other directories. Directories are analogous to Macintosh and Windows folders.

## Naming Unix Files and Directories

Each file and directory has a name. Within a directory, each item (that is, each file or directory) must have a unique name, but items with the same name may exist in more than one directory. A directory may have the same name as one of the items it contains.

File and directory names may be up to 256 characters long. Names may use almost any character (except a space). You can divide a multi-word file name using either an underscore or a period (for example, **chapter\_one** or **chapter.two**).

Some characters have special meanings to Unix. It is best to avoid using these characters in file names:

```
/ \ ' ' * | ! ? ~ $ < >
```

Unix is case-sensitive. Each of these is a unique file: **myfile**, **Myfile**, **myFile**, and **MYFILE**.

## Creating a File

Many people create files using a text editor, but you can use the command **cat** to create files without using/learning to use a text editor. To create a practice file (called **firstfile**) and enter one line of text in it, type the following at the % prompt:

```
cat > firstfile
```

(Press the **Enter/Return** key.)

```
This is just a test.
```

(Press the **Enter/Return** key.)

Terminate file entry by typing **Control-d** on a line by itself. (Hold down the Control key and type d.) On your screen, you will see:

```
% cat > firstfile
```

```
This is just a test.
```

```
^D
```

To examine the contents of a file you have just created, enter this at the % prompt:

```
cat firstfile
```

## Copying a File

To make a duplicate copy of a file, use the command **cp**. For example, to create an exact copy of the file called **firstfile**, you would type:

```
cp firstfile secondfile
```

This results in two files with different names, each containing the same information.

The **cp** command works by overwriting information. If you create a different file called **thirdfile** and then type the following command:

```
cp thirdfile firstfile
```

you will find that the original contents of **firstfile** are gone, replaced by the contents of **thirdfile**.

## Renaming a File

Unix does not have a command specifically for renaming files. Instead, the **mv** command is used both to change the name of a file and to move a file into a different directory.

To change the name of a file, use the following command format (where **thirdfile** and **file3** are sample file names):

```
mv thirdfile file3
```

This command results in the complete removal of **thirdfile**, but a new file called **file3** contains the previous contents of **thirdfile**.



Like **cp**, the **mv** command also overwrites existing files. For example, if you have two files, **fourthfile** and **secondfile**, and you type the command

```
mv fourthfile secondfile
```

**mv** will remove the original contents of **secondfile** and replace them with the contents of **fourthfile**. As a result, **fourthfile** is renamed **secondfile**, but in the process **secondfile** is deleted.

## Removing a File

Use the **rm** command to remove a file. For example,

```
rm file3
```

deletes **file3** and its contents. You may remove more than one file at a time by specifying a list of files to be deleted. For example,

```
rm firstfile secondfile
```

You will be prompted to confirm whether you really want to remove the files:

```
rm: remove firstfile (y/n)? y
```

```
rm: remove secondfile (y/n)? n
```

Type **y** or **yes** to remove a file; type **n** or **no** to leave it intact.

## Creating a Directory

Creating directories permits you to organize your files. The command

```
mkdir project1
```

creates a directory called **project1**, where you can store files related to a particular project. The directory that you create will be a subdirectory within your current directory. For details on how to navigate directories and display the files and directories they contain, see [List Contents and Navigate Unix Directories](#).

## Moving and Copying Files Into a Directory

The **mv** and **cp** commands can be used to put files into a directory. Assume that you want to place some files from your current directory into a newly created directory called **project1**. The command

```
mv bibliography project1
```

will move the file **bibliography** into the directory **project1**. The command

```
cp chapter1 project1
```

will place a copy of the file **chapter1** in the directory **project1**, but leave **chapter1** intact in the current directory. There will now be two copies of **chapter1**, one in the current directory and one in **project1**.



## Renaming a Directory

You can also use the **mv** command to rename and move directories. When you type the command

```
mv project1 project2
```

the directory called **project1** will be given the new name **project2** as long as a directory called **project2** did not previously exist. If directory **project2** already existed before the **mv** command was issued,

```
mv project1 project2
```

would move the directory **project1** and its files into the directory **project2**.

## Copying a Directory

You can use the **cp** command to create a duplicate copy of a directory and its contents. To copy directory **project1** to directory **proj1copy**, for example, you would type

```
cp -r project1 proj1copy
```

If directory **proj1copy** already exists, this command will put a duplicate copy of **directory project1** into directory **proj1copy\**.

## Removing a Directory

Use the command **rmdir** to remove an empty directory. Multiple empty directories may be removed by listing them after the command:

```
rmdir testdir1 testdir2
```

If you try to remove a directory that is not empty, you will see

```
rmdir: testdir3: Directory not empty
```

If you are sure that you want to remove the directory and all the files it contains, use the command

```
rm -r testdir3
```

## Summary of Commands

### Working With Files

- **mv file1 file2**  
Renames **file1** to **file2** (if **file2** existed previously, overwrites original contents of **file2**).
- **cp file1 file2**  
Copies **file1** as **file2** (if **file2** existed previously, overwrites original contents of **file2**).

- **rm file3 file4**  
Removes **file3** and **file4**, requesting confirmation for each removal.

## Working With Directories

- **mkdir dir1**  
Creates a new directory called **dir1**.
- **mv dir1 dir2**  
If **dir2** does not exist, renames **dir1** to **dir2**.  
If **dir2** exists, moves **dir1** inside **dir2**.
- **cp -r dir1 dir2**  
If **dir2** does not exist, copies **dir1** as **dir2**.  
If **dir2** does exist, copies **dir1** inside **dir2**.
- **rmdir dir1**  
Removes **dir1**, if **dir1** contains no files.
- **rm -r dir1**  
Removes **dir1** and any files it contains. Use with caution.

## Working With Files and Directories

- **cp file1 dir1**  
Copies file **file1** into existing directory **dir1**.
- **mv file2 dir2**  
Moves file **file2** into existing directory **dir2**.