

EXPERIMENT :-3

AIM:- FILE SYSTEM HEIRARCHY & FILE PERMISSIONS IN LINUX

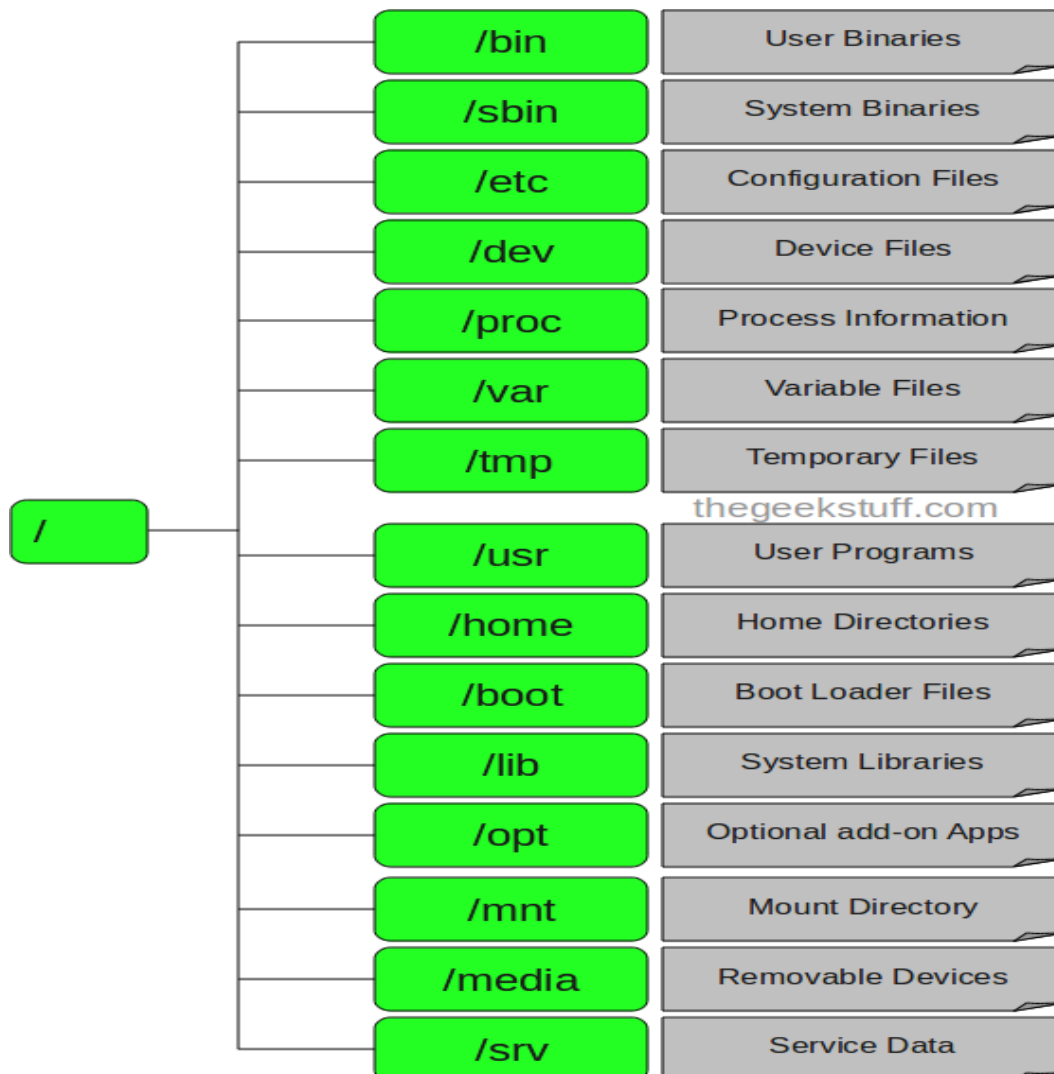
Linux File Hierarchy Structure

The Linux File Hierarchy Structure or the Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems. It is maintained by the Linux Foundation.

In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.

Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.

Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS and are not considered authoritative for platforms other than Linux.



1. / – Root

Every single file and directory starts from the root directory.

Only root user has write privilege under this directory.

Please note that /root is root user's home directory, which is not same as /.

2. /bin – User Binaries

Contains binary executables.

Common linux commands you need to use in single-user modes are located under this directory.

Commands used by all the users of the system are located here.

For example: ps, ls, ping, grep, cp.

3. /sbin – System Binaries

Just like /bin, /sbin also contains binary executables.

But, the linux commands located under this directory are used typically by system administrator, for system maintenance purpose.

For example: iptables, reboot, fdisk, ifconfig, swapon

4. /etc – Configuration Files

Contains configuration files required by all programs.

This also contains startup and shutdown shell scripts used to start/stop individual programs.

For example: /etc/resolv.conf, /etc/logrotate.conf

5. /dev – Device Files

Contains device files.

These include terminal devices, usb, or any device attached to the system.

For example: /dev/tty1, /dev/usbmon0

6. /proc – Process Information

Contains information about system process.

This is a pseudo filesystem contains information about running process. For example: `/proc/{pid}` directory contains information about the process with that particular pid.

This is a virtual filesystem with text information about system resources. For example: `/proc/uptime`

7. `/var` – Variable Files

`var` stands for variable files.

Content of the files that are expected to grow can be found under this directory.

This includes — system log files (`/var/log`); packages and database files (`/var/lib`); emails (`/var/mail`); print queues (`/var/spool`); lock files (`/var/lock`); temp files needed across reboots (`/var/tmp`);

8. `/tmp` – Temporary Files

Directory that contains temporary files created by system and users.

Files under this directory are deleted when system is rebooted.

9. `/usr` – User Programs

Contains binaries, libraries, documentation, and source-code for second level programs.

`/usr/bin` contains binary files for user programs. If you can't find a user binary under `/bin`, look under `/usr/bin`. For example: `at`, `awk`, `cc`, `less`, `scp`

`/usr/sbin` contains binary files for system administrators. If you can't find a system binary under `/sbin`, look under `/usr/sbin`. For example: `atd`, `cron`, `sshd`, `useradd`, `userdel`

`/usr/lib` contains libraries for `/usr/bin` and `/usr/sbin`

`/usr/local` contains users programs that you install from source. For example, when you install apache from source, it goes under `/usr/local/apache2`

10. `/home` – Home Directories

Home directories for all users to store their personal files.

For example: `/home/john`, `/home/nikita`

11. `/boot` – Boot Loader Files

Contains boot loader related files.

Kernel initrd, vmlinuz, grub files are located under /boot

For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

12. /lib – System Libraries

Contains library files that supports the binaries located under /bin and /sbin

Library filenames are either ld* or lib*.so.*

For example: ld-2.11.1.so, libncurses.so.5.7

13. /opt – Optional add-on Applications

opt stands for optional.

Contains add-on applications from individual vendors.

add-on applications should be installed under either /opt/ or /opt/ sub-directory.

14. /mnt – Mount Directory

Temporary mount directory where sysadmins can mount filesystems.

15. /media – Removable Media Devices

Temporary mount directory for removable devices.

For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

16. /srv – Service Data

srv stands for service.

Contains server specific services related data.

For example, /srv/cvs contains CVS related data.

FILE PERMISSIONS IN LINUX

Linux file permissions, attributes, and ownership control the access level that the system processes and users have to files. This ensures that only authorized users and processes can access specific files and directories.

The basic Linux permissions model works by associating each system file with an owner and a group and assigning permission access rights for three different classes of users:

- The file owner.
- The group members.
- Others (everybody else)

User/Owner

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user- group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else.

Hence, when you set the permission for others, it is also referred as set permissions for the world.

Permissions

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- Read: This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

- **Write:** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

Changing file/directory permissions with 'chmod' command

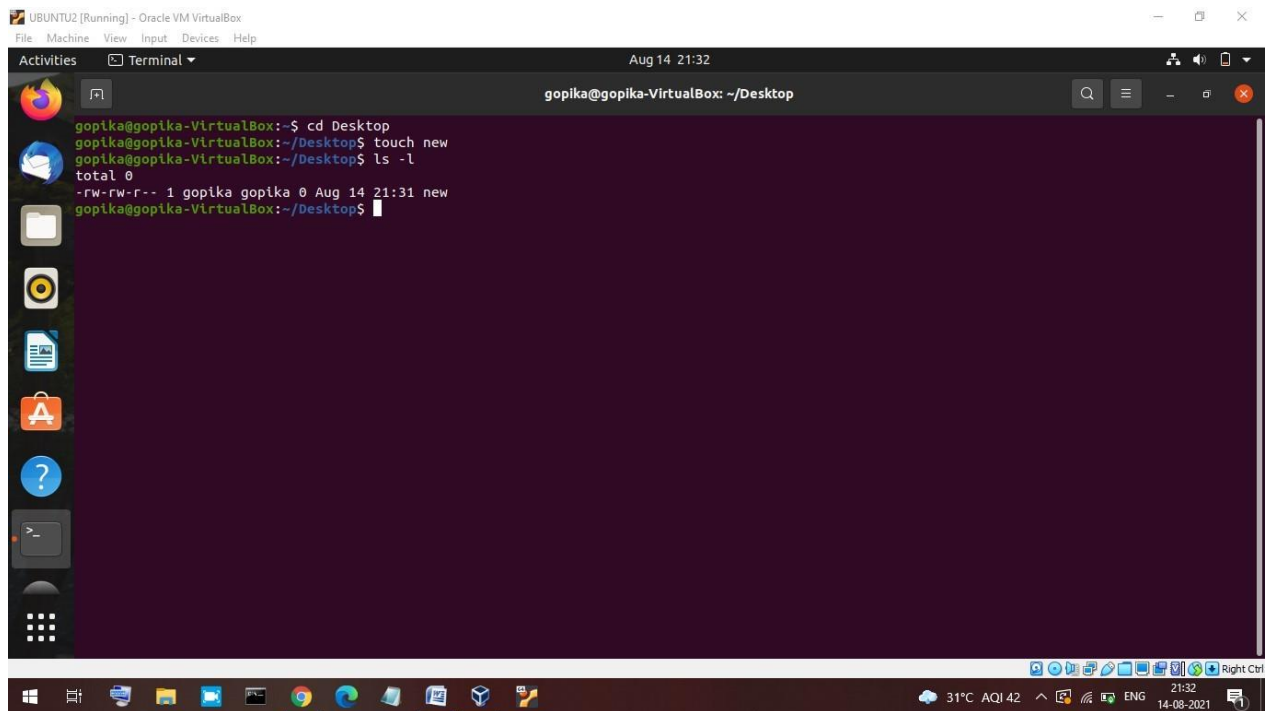
Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the Unix file permissions.

OPERATOR	DESCRIPTION
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permission set earlier

The various owners are represented as –

User Denotations	
u	User /owner
g	group
o	others
a	all



The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~/Desktop". The user has executed the following commands:

```
gopika@gopika-VirtualBox:~$ cd Desktop
gopika@gopika-VirtualBox:~/Desktop$ touch new
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
-rw-rw-r-- 1 gopika gopika 0 Aug 14 21:31 new
gopika@gopika-VirtualBox:~/Desktop$
```

The output shows a file named "new" with permissions "-rw-rw-r--", owned by "gopika" and "gopika", with a size of 0 bytes, created on Aug 14 at 21:31.

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, **-rw-rw-r--** represents that the owner has read (r) and write (w) permission, but no execute permission.

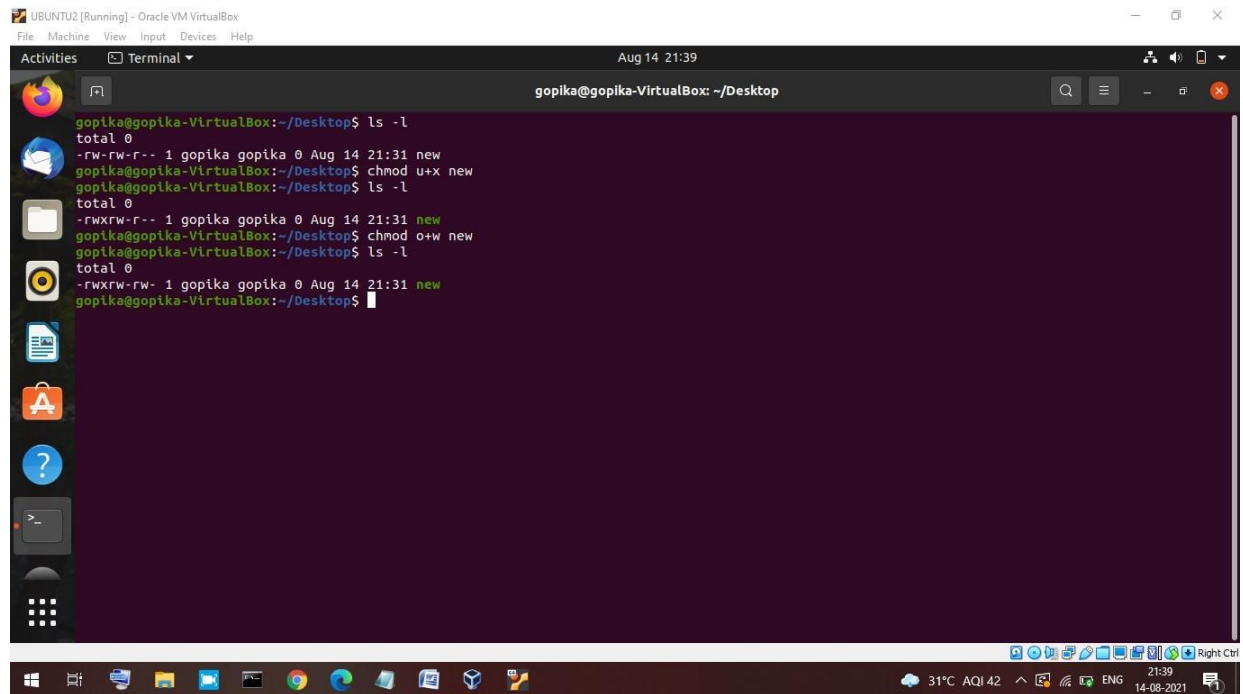
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rw-rw-r--** represents that the group has read (r) and write(w) permission but no execute (x) permission.
- The last group of three characters (8-10) represents the permissions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

To change directory permissions in Linux, use the following:

- **chmod + rwx filename** to add permissions.
- **chmod - rwx directoryname** to remove permissions.
- **chmod + x filename** to allow executable permissions.
- **chmod - wx filename** to take out write and executable permissions.

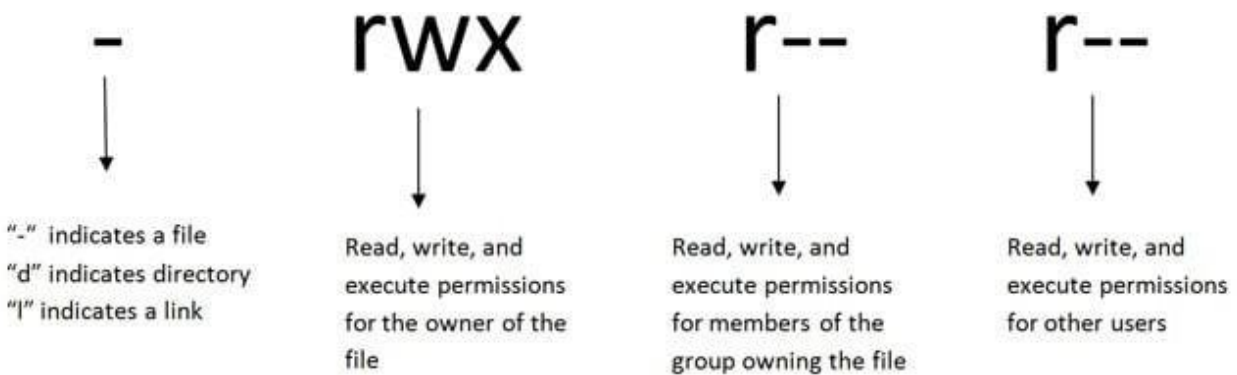
Here “r” is for read, “w” is for write, and “x” is for execute. This

only changes the permissions for the owner of the file.



The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~/Desktop" with a dark purple background. The user has executed a series of commands to create a file named 'new' and modify its permissions. The output of the 'ls -l' command shows the file's permissions as -rw-rw-r--. The user then runs 'chmod u+x new' to add execute permission for the owner, and 'chmod o+w new' to add write permission for others. The final 'ls -l' output shows the permissions as -rwxrw-rw-.

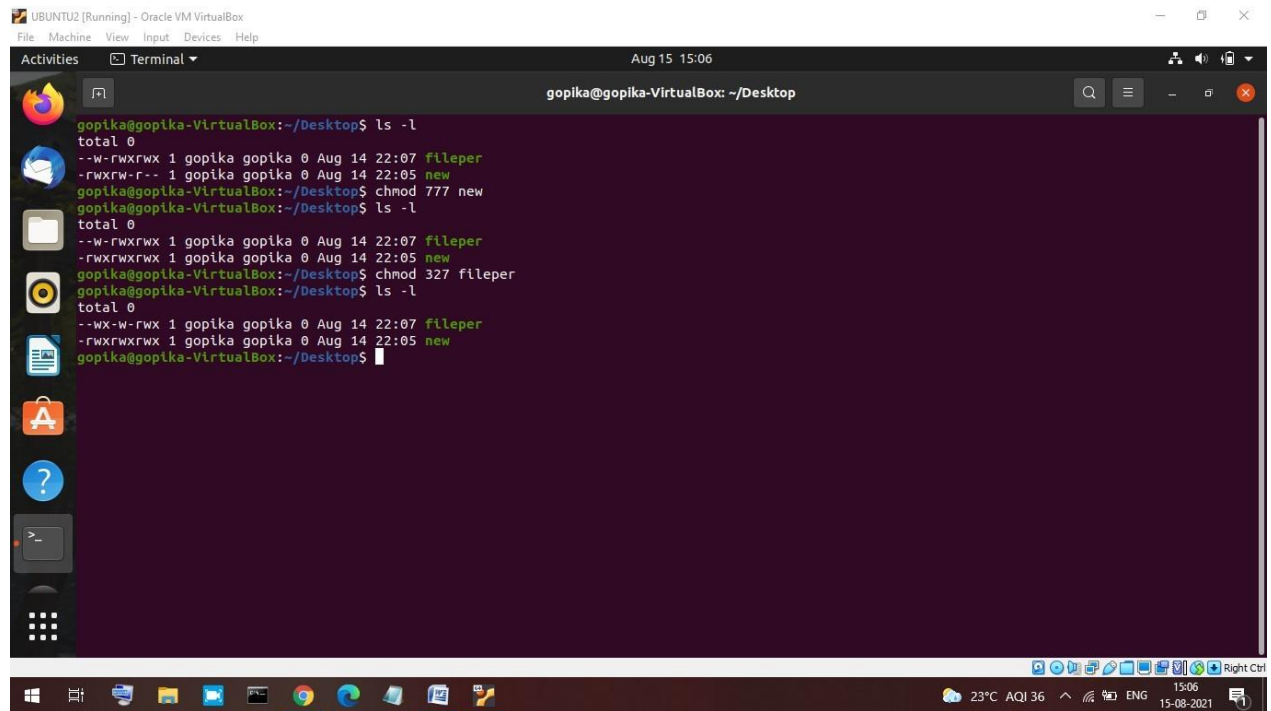
```
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
-rw-rw-r-- 1 gopika gopika 0 Aug 14 21:31 new
gopika@gopika-VirtualBox:~/Desktop$ chmod u+x new
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
-rwxrw-r-- 1 gopika gopika 0 Aug 14 21:31 new
gopika@gopika-VirtualBox:~/Desktop$ chmod o+w new
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
-rwxrw-rw- 1 gopika gopika 0 Aug 14 21:31 new
gopika@gopika-VirtualBox:~/Desktop$
```



Numeric Mode

Permission numbers are:

- **0 = ---** no permission
- **1 = --x** execute permission only
- **2 = -w-** permission to write only
- **3 = -wx** permission for write and execute
- **4 = r-** permission for read
- **5 = r-x** permission for read and execute
- **6 = rw-** permission for read and write
- **7 = rwx** permission for read ,write and execute



```
gopika@gopika-VirtualBox: ~/Desktop
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
--w-rwxrwx 1 gopika gopika 0 Aug 14 22:07 fileper
-rwxrwx-r-- 1 gopika gopika 0 Aug 14 22:05 new
gopika@gopika-VirtualBox:~/Desktop$ chmod 777 new
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
--w-rwxrwx 1 gopika gopika 0 Aug 14 22:07 fileper
-rwxrwxrwx 1 gopika gopika 0 Aug 14 22:05 new
gopika@gopika-VirtualBox:~/Desktop$ chmod 327 fileper
gopika@gopika-VirtualBox:~/Desktop$ ls -l
total 0
--wx-w-rwx 1 gopika gopika 0 Aug 14 22:07 fileper
-rwxrwxrwx 1 gopika gopika 0 Aug 14 22:05 new
gopika@gopika-VirtualBox:~/Desktop$
```

CHANGING FILE OWNERSHIPS

The chown command allows us to change the user and/or group ownership of a given file, directory, or symbolic link.

In Linux, all files are associated with an owner and a group and assigned with permission access rights for the file owner, the group members, and others.

These commands will give ownership to someone, but all sub files and directories still belong to the original owner.

Chown <name> <file name>

Or

chown [OPTIONS] USER[:GROUP] FILE(s)

USER is the user name or the user ID (UID) of the new owner. GROUP is the name of the new group or the group ID (GID). FILE(s) is the name of one or more files, directories or links. Numeric IDs should be prefixed with the + symbol.

- **USER** - If only the user is specified, the specified user will become the owner of the given files, the group ownership is not changed.
- **USER:** - When the username is followed by a colon :, and the group name is not given, the user will become the owner of the files, and the files group ownership is changed to user's login group.
- **USER:GROUP** - If both the user and the group are specified (with no space between them), the user ownership of the files is changed to the given user and the group ownership is changed to the given group.
- **:GROUP** - If the User is omitted and the group is prefixed with a colon :, only the group ownership of the files is changed to the given group.
- **:** If only a colon : is given, without specifying the user and the group, no change is made.

By default, on success, chown doesn't produce any output and returns

zero. Superuser permissions are necessary to execute the chown

command.