# Linux tar Command

the tar command is one of the essential commands as far as file management is concerned. It’s short for Tape Archive, and it’s used for creating & extracting archive files. An archive file is a compressed file containing one or multiple files bundled together for more accessible storage and portability.

**$ tar [options][archive-file] [file or dir to be archived]**

**tar Command Options**

The tar command provides the following options:

**-c**: This creates an archive file.

**-x**: The option extracts the archive file.

**-f**: Specifies the filename of the archive file.

**-v**: This prints verbose information for any tar operation on the terminal.

**-t**: This lists all the files inside an archive file.

**-u**: This archives a file and then adds it to an existing archive file.

**-r**: This updates a file or directory located inside a .tar file

**-z**: Creates a tar file using gzip compression

**-j**: Create an archive file using the bzip2 compression

**-W**: The -w option verifies an archive file.

**Create an archive file** when using the gzip compression (using the -z option), the file should bear the .tar.gz suffix. For example, the command below creates an archive file called sales.tar.gz from the three PDF files.

```
$ tar -czvf sales.tar.gz  sales1.pdf sales2.pdf sales3.pdf
```

Suppose you want to create an archive file using the bzip2 compression ( using the -j option ) algorithm. The archive file should be suffixed with a .tar.bz2 extension. Using our first example, we can compress the three PDF files using the bzip2 algorithm as follows.

```
$ tar -cjvf sales.tar.bz2 sales1.pdf sales2.pdf sales3.pdf
```

you can also compress directories. For example, the command below creates a simple tar archive file of the home directory.

```
$ tar -cvf home.tar /home/james
```

**List the contents of an archive file**

Using the -t option, you can have a peek or list the contents of an archive file without extracting it as shown.

```
$ tar -tf sales.tar.gz
```

**Extract an archive file in the current directory**

To extract an archive file in the current working directory, use the -x option as shown below. In the example below, we are uncompressing or extracting the documents.tar.gz archive, which contains three text files.

```
$ tar -xvf documents.tar.gz
```

**Extract an archive file in a separate directory**

To extract an archive file to a different directory, the -C option is followed by the destination path, as shown in the example below.

```
$ tar -xvf documents.tar.gz -C /tmp/files
```

**Extract specific files from an archive**

You can extract certain specified files by listing them one by one on the command line. In the example below, we are extracting the files file1.txt and file2.txt from the   documents.tar.gz archive.

$ tar -xvf documents.tar.gz ~~file1.txt file2.txt~~

## Add a file to a .tar archive

To add or append **a .tar archive file**, use the **-r** option as shown. Here, we are adding the file **file3.txt** to the **archives.tar** archive.

```
$ tar -rvf archives.tar file3.txt
```

## Remove a file from a .tar archive

To remove a file from **a .tar archive**, use the **–delete option** as shown. Here, we are doing the complete opposite and removing the file **file3.txt** instead.

```
$ tar --delete -f archives.tar file3.txt
```

## Making an uncompressed tar archive with -cvf option

This option makes a tar file known as **file.tar**. It is the archive of every .txt file inside ***mydir*** directory.

The command is as follows:

```
$ tar cvf file.tar *.txt
```

**Extracting files through the archive with -xvf option**

This option can extract files through archives.

```
$ tar xvf file.tar
```

**gzip compression over tar archive with -z option**

This option makes a tar file known as **file.tar.gz.** It is the archive of every .txt file.

The command is as follows:

```
$ tar cvzf file.tar.gz *.txt
```

**Untar multiple .tar.tbz, .tar.gz, .tar files**

This option will help us to **untar** or extract more than one file from tar.bz2, tar.gz, and a tar archive file.

```
$ tar -jxvf file.tar.tbz "hello1.txt"
$ tar -zxvf file.tar.gz "hello1.txt" "hello2.txt"
```

## Check the size of the existing tar.tbz, tar.gz, tar file

The command will help us to show the archive file's size in kilobytes (KB) which is mentioned above.

**The command is as follows:**

```
$ tar -czf - file2.tar.tbz | wc -c
```

**Or,**

```
$ tar -czf - file1.tar.gz | wc -c
```

**Or,**

```
$ tar -czf - file.tar | wc -c
```

# Touch Command

In **Linux** every single file is associated with timestamps, and every file stores the information of last access time, last modification time and last change time. So, whenever we create new file, access or modify an existing file, the timestamps of that file automatically updated.

**Touch Command Options**

**-a**, change the access time only

**-c**, if the file does not exist, do not create it

**-d**, update the access and modification times

**-m**, change the modification time only

**-r**, use the access and modification times of file

**-t**, creates a file using a specified time

## 1. How to Create an Empty File

The following touch command creates an empty (zero byte) new file called **sheena**.

```
# touch sheena
```

## 2. How to Create Multiple Files

By using touch command, you can also create more than one single file. For example the following command will create 3 files named, **sheena**, **meena** and **leena**.

```
# touch sheena meena leena
```

## 3. How to Change File Access and Modification Time

To change or update the last access and modification times of a file called **leena**, use the **-a** option as follows. The following command sets the current time and date on a file. If the **leena** file does not exist, it will create the new empty file with the name.

```
# touch -a leena
```

## 4. How to Avoid Creating New File

Using **-c** option with touch command avoids creating new files. For example the following command will not create a file called **leena** if it does not exists.

```
# touch -c leena
```

## 5. How to Change File Modification Time

If you like to change the only modification time of a file called **leena**, then use the **-m** option with touch command. Please note it will only updates the last modification times (not the access times) of the file.

```
# touch -m leena
```

## 6. Explicitly Set the Access and Modification times

You can explicitly set the time using **-c** and **-t** option with touch command. The format would be as follows.

```
# touch -c -t YYDDHHMM leena
```

```
# touch -c -t 22201730 leena.
```

the access and modification time of file **leena**, with **ls -l** command.

## 7. How to Use the time stamp of another File

The following touch command with **-r** option, will update the time-stamp of file **meena** with the time-stamp of **leena** file. So, both the file holds the same time stamp.

```
# touch -r leena meena
```

## . Create a File using a specified time

If you would like to create a file with specified time other than the current time, then the format should be.

```
# touch -t YYMMDDHHMM.SS tec
```

For example the below command touch command with **-t** option will gives the **tec** file a time stamp of **18:30:55 p.m**. on Mar **20**, **2022**.

```
# touch -t 202003201830.55 tec
```

# Find Command

The find command is used to search and locate the list of files and directories based on conditions you specify for files that match the arguments. **find command** can be used in a variety of conditions like you can find files by **permissions**, **users**, **groups**, **file types**, **date**, **size**, and other possible criteria.

**1. Find Files Using Name in Current Directory**

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

```
# find . -name tec.txt
./tec.txt
```

**2. Find Files Under Home Directory**

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

```
# find /home -name tec.txt
 /home/tecmint.txt
```

## 3. Find Files Using Name and Ignoring Case

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

```
# find /home -iname tec.txt
./tec.txt
./Tec.txt
```

## 4. Find Directories Using Name

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

```
# find / -type d -name Tec
/Tecmint
```

## 5. Find PHP Files Using Name

Find all **php** files whose name is **tec.php** in a current working directory.

```
# find . -type f -name tec.php
./tecmint.php
```

## 6. Find all PHP Files in the Directory

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

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

```
./tec.php ./login.php ./index.php
```

## Part II – Find Files Based on their Permissions

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

```
# find . -type f -perm 0777 -print8.
```

**Find Files Without 777 Permissions**

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

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

**Find Read-Only Files**

Find all **Read-Only** files.

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

**Find Executable Files**

Find all **Executable** files.

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

## Find Files with 777 Permissions and Chmod to 644

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

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

## Find Directories with 777 Permissions and Chmod to 755

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

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

## Find and remove single File

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

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

## Find and remove Multiple File

To find and remove multiple files such as **.mp3** or **.txt**, then use.

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

OR

```
# find . -type f -name "*.mp3" -exec rm -f {} \;
```

**Find all Empty Files**

To find all empty files under a certain path.

```
# find /tmp -type f -empty20.
```

**Find all Empty Directories**

To file all empty directories under a certain path.

```
# find /tmp -type d -empty21.
```

**File all Hidden Files**

To find all hidden files, use the below command.

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

## Part III – Search Files Based On Owners and Groups

**Find Single File Based on User**

To find all or single files called **tec.txt** under **/** root directory of owner root.

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

**Find all Files Based on User**

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

```
# find /home -user tecmint24.
```

**Find all Files Based on Group**

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

```
# find /home -group developer25.
```

**Find Particular Files of User**

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

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

## Part IV – Find Files and Directories Based on Date and Time

**Find Last 50 Days Modified Files**

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

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

## Find Last 50 Days Accessed Files

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

```
# find / -atime 5028.
```

## Find Last 50-100 Days Modified Files

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

```
# find / -mtime +50 –mtime -10029.
```

## Find Changed Files in Last 1 Hour

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

```
# find / -cmin -6030.
```

## Find Modified Files in Last 1 Hour

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

```
# find / -mmin -6031.
```

## Find Accessed Files in Last 1 Hour

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

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

## Part V – Find Files and Directories Based on Size

**Find 50MB Files**

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

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

**Find Size between 50MB – 100MB**

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

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

**Find and Delete 100MB Files**

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

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

**Find Specific Files and Delete**

Find all **.mp3** files with more than **10MB** and delete them using one single command.

```
# find / -type f -name *.mp3 -size +10M -exec rm {} \;
```

## echo’ command

The echo command is one of the most commonly and widely used built-in commands for Linux bash and C shells, that typically used in a scripting language and batch files to display a line of text/string on standard output or a file.

The syntax for the echo command is:

```
echo [option(s)] [string(s)]
```

1. Input a line of text and display it on standard output

```
$ echo Tec is a community of Linux Nerds
```

Outputs the following text:

```
Tec is a community of Linux Nerds
```

**2.** Declare a variable and echo its value. For example, Declare a variable of **x** and assign its value=**10**.

```
$ x=10
```

echo its value:

```
$ echo The value of variable x = $x
The value of variable x = 10
```

**Note**: The ‘**-e**‘ option in Linux acts as an interpretation of escaped characters that are backslashed.

**3.** Using option ‘**\b**‘ – backspace with backslash interpretor ‘**-e**‘ which removes all the spaces in between.

```
$ echo -e "Tec \bis \ba \bcommunity \bof \bLinux \bNerds"
TecisacommunityofLinuxNerds
```

**4.** Using option ‘**\n**‘ – New line with backspace interpretor ‘**-e**‘ treats new line from where it is used.

```
$ echo -e "Tec \nis \na \ncommunity \nof \nLinux \nNerds"
Tec is a community of Linux Nerds
```

**5.** Using option ‘**\t**‘ – horizontal tab with backspace interpretor ‘**-e**‘ to have horizontal tab spaces.

```
$ echo -e "Tecmint \tis \ta \tcommunity \tof \tLinux \tNerds"
```

```
Tecmint is a community of Linux Nerds
```

**6.** How about using option new Line ‘**\n**‘ and horizontal tab ‘**\t**‘ simultaneously.

```
$ echo -e "\n\tTecmint \n\tis \n\ta \n\tcommunity \n\tof \n\tLinux \n\tNerds"
```

**7.** Using option ‘**\v**‘ – vertical tab with backspace interpretor ‘**-e**‘ to have vertical tab spaces.

```
$ echo -e "\vTecmint \vis \va \vcommunity \vof \vLinux \vNerds"
```

**8.** How about using option new Line ‘**\n**‘ and vertical tab ‘**\v**‘ simultaneously.

```
$ echo -e "\n\vTecmint \n\vis \n\va \n\vcommunity \n\vof \n\vLinux 
\n\vNerds"
```

The echo can be used with a redirect operator to output to a file and not standard output.

```
$ echo "Test Page" > testpage
```

## ‘who’ Command

**who command** which displays users who are currently logged on to a Linux system, including the terminals they are connecting from.

```
$ who who [OPTION]... [ FILE | ARG1 ARG2 ]
```

**1.** If you run **who** command without any arguments, it will display account information (user login name, user’s terminal, time of login as well as the host the user is logged in from) on your system similar to the one shown in the following output.

```
$ who
```

**2.** To print the heading of the columns displayed, use the -H flag as shown.

```
$ who –H
```

**3.** To print the login names and total number of logged on users, use the -q flag.

```
$ who -q
```

**4.** In case you want to show only **hostname** and user associated with **stdin**, use the -m switch.

```
$ who –m
```

**5.** Next, to add user’s message status as +, - or ?, use the -T option.

```
$ who –T
```

he who command also helps you to view some useful system information such as **last boot time**, **current runlevel** (**target** under **systemd**), print dead processes as well as processes spawned by **init**.

**6.** To view the time of last system boot, use the -b flag and adding the -u option allows for listing of logged on users in the same output.

```
$ who –b
```

```
$ who -bu
```

**7.** You can check the current runlevel with the -r option.

```
$ who -r
```

**8.** The following command will print dead processes.

```
$ who –d
```

**9.** Furthermore, to see active processes spawned by init, use the -p option.

```
$ who –p
```

**10.** Last but not least, the -a flag allows for printing of default output combined with information from some of the options we have covered.

```
$ who –a
```

**you can find more options by consulting the who man page.**

```
$ man who
```

## whoami Command in Linux

whoami is an basic Unix/Linux command used to find username associated with current effective user id. This is generally used to identify the current logged in user in a shell. This command is also useful in shell scripts to identify the user id from which the script is running.

**Syntax:**

```
whoami [OPTION]...
```

Simply type the ‘whoami’ on command prompt to find logged in user in current shell.

```
whoami
```

```
root
```

- Let’s write a small bash shell script and check if script is running as root user or not. This is very useful to warn user that the script is running as root user.
-

```
#!/bin/bash

if [ `whoami` == 'root' ]; then
      echo "Warning! You are running this script as root user"
fi
```

**Difference Between whoami and who am i Command**

Both the commands whoami and who am i are used to get logged in username in Linux system. The username is defined in passwd file associate with effective user id.

When a user login as root on the network, then both whoami and who am I commands will return root.

**whoami**

Output:> root

**Who am I**

Output:> root pts/14 …..

But, when you logged in as another user (eg: rahul) and switched to root user (su – root). The whoami will show root but who am i will show the originally logged in user ‘rahul’.

whoami

Output:> root

**who am i**
**Output:> rahul pts/14 …..**

## Linux wc command

The Linux **wc** command is used to count the number of lines, words, and byte (character) in a file or input stream.

**Syntax:**

wc [OPITON] [FILE} **Example:**

Use **-l** to count the number of lines in a file

**wc -l myfile.txt** Use **-w** to count the number of words in a file

**wc -w myfile.txt** Use **-c** to count the number of bytes in a file. You can use this to count character in a file

**wc -c myfile.txt**

**Use wc with Piped Input**

You can also use wc with piped input in Linux and count the lines, words, and characters in an input data.

**cat myfile.txt | wc -lwc**

Another example of wc command to count the number of lines in the output of the previous command.

**find . -name "*.log" | wc -l**

# Linux ps command

The ps command, short for Process Status, is a command line utility that is used to **display or view information related to the processes running in a Linux system**. The ps command lists current running processes alongside their PIDs and other attributes.

**ps command without arguments**

The ps command without arguments lists the running processes in the current shell

**ps**

The output consists of four columns

PID – This is the unique process ID

TTY – This is the typeof terminal that the user is logged in to

TIME – This is the time in minutes and seconds that the process has been running

CMD – The command that launched the process

**Viewing all the running processes in different formats**

To have a glance at all the running processes, execute the command below

**ps -A**

ps –e

**View processes associated with the terminal**

To view processes associated with the terminal run

ps -T

**View processes not associated with terminal**

To view all processes with the exception of processes associated with the terminal and session leaders execute

ps -a

**Show all current running processes**

To view all current processes execute

ps -ax

-a flag stands for all processes
-x will display all processes even those not associated with the current tty

**Display all processes in BSD format**

If you wish to display processes in BSD format, execute

**ps au OR**

**ps aux**

**To perform full format listing**

To view a full format listing run

**ps -ef OR**

**ps –eF**

**Filter processes according to the user**

If you wish to list processes associated with a specific user, use the -u flag as shown

**ps -u user**

**For example**

**ps -u jamie**

**Filter process by thread process**

If you wish to know the thread of a particular process, make use of the -Lflag followed by the PID

For example

```
ps -L 4264
```

**Show every process running as root**

Sometimes, you may want to reveal all processes run by the root user. To achieve this run

```
ps -U root -u root
```

**Display group processes**

If you wish to list all processes associated by a certain group run

ps -fG group_nameOr

ps -fG groupID

For example

```
ps -fG root
```

## Kill Linux

- Linux Operating System comes with a **kill command** to terminate a process. The command makes it possible to continue running the server without the need to reboot after a major change/update. Here comes the great power of Linux and this is one of the reasons, why Linux is running on **96.4%** of servers, on the planet.
- **Kill Command Usage**
- The common syntax for **kill command** is:
- **# kill [signal or option] PID(s)**
  **List All Running Linux Processes**
- To know all the processes and correspondingly their assigned **pid**, run the following ps command.
- **# ps –A**
- How about Customising the above output using syntax as '**pidof process**'.
- **# pidof mysqld.** 1684

Another way to achieve the above goal is to follow the below syntax.

```
# ps aux | grep mysqld
```

## How to Kill a Process in Linux

Before we step ahead and execute a **kill command**, some important points to be noted:

A user can kill all his processes.

A user can not kill another user’s process.

A user can not kill processes the System is using.

A root user can kill System-level-process and the process of any user.

To kill the above process **PID**, use the kill command as shown.

```
kill -9 3139
```

The above command will kill the process having **pid=3139**, where **PID** is a **Numerical Value** of the process.

Another way to perform the same function can be rewritten as.

```
# kill -SIGTERM 3139
```

Similarly ‘**kill -9 PID**‘ is similar to ‘**kill -SIGKILL PID**‘ and vice-versa.

**How to Kill a Process in Linux Using Process Name**

You must be aware of the process name, before killing and entering a wrong process name may screw you.

```
# pkill mysqld
```

Kill more than one process at a time.

```
# kill PID1 PID2 PID3 or

# kill -9 PID1 PID2 PID3 or

# kill -SIGKILL PID1 PID2 PID3
```

## chmod command

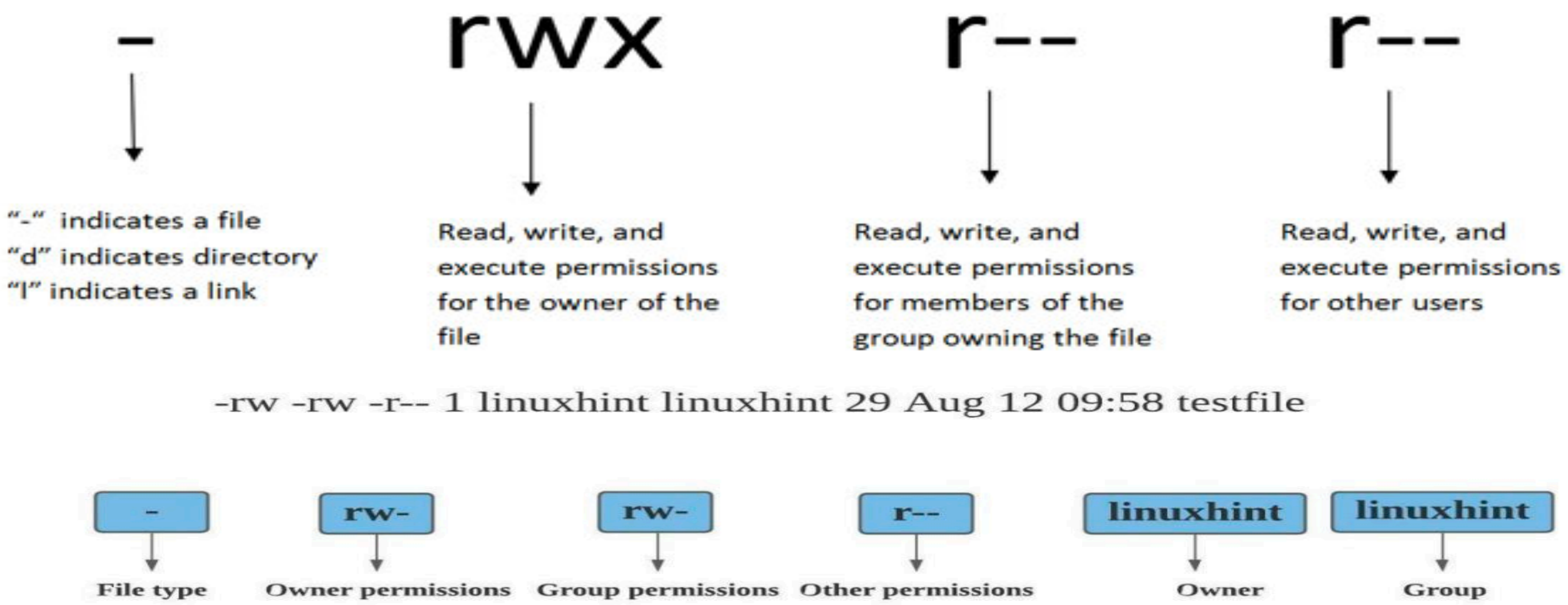
The “**chmod**” command in **Linux** enables you to control the access of scripts, directories, and your system files. This command is utilized to change the Linux file permissions, which seems a complicated method but is simple once you understand its functionality. Before discussing the **chmod** command, let’s go through the fundamentals of Linux file permission.

File permission is the type of access associated with a file. Each file in Linux has its owner, a group, and permission access for **three main types of users**: the **file owner**, the **group** members, **and others**. Each of these user classes has **three types of file permissions**: **read**, **write**, and **execute** permissions. Knowing about the file permission helps you specify which users can execute, read, or write the file.

**How to check file permission in Linux**

The “**ls**” is used to check the permission of the files present on your system. To view permission of a single file, add its name to the “**ls**” command. For instance, we will execute the below-given command to check the file permissions of the “**testfile**”:

- **ls -l testfile**
- Here, the “**-l**” option is added to get the content of the “**testfile**” including its file permissions:


-
rwx
r--
r--
“-“ indicates a file
“d” indicates directory
“l” indicates a link
Read, write, and execute permissions for the owner of the file
Read, write, and execute permissions for members of the group owning the file
Read, write, and execute permissions for other users
-rw -rw -r-- 1 linuxhint linuxhint 29 Aug 12 09:58 testfile
-
rw-
rw-
r--
linuxhint
linuxhint
File type
Owner permissions
Group permissions
Other permissions
Owner
Group

The **first character** in the output specifies the **entry type** where "**–**" represents a "**file**", and "**d**" indicates a "**directory**". Then we have, **three sets of nine characters,** where **the first three characters set** represent file **owner permissions**, **the next characters set** represent **group permissions**, and **the last set** represents permissions for **other** users that are not considered in the first two categories:

Each **set** of permissions contains **three characters**. If the character is a dash "**–**", it indicates that access permissions are denied. Permission has been given to a user if the character is "**r**", "**w**", or "**x**". The "**r**" in a permission set indicates that the user only has **read permission** which means that the file can be only **opened** and **viewed**, with "**w**," you will have **write permission** for the specific file, and you will be able to **edit**, **modify** and **delete** the file. Lastly, the "**x**" letter represents the **execute permissions**; for instance, if your file is a C++ program or script, it will have the execute permission with the letter "**x**".

If the "**ls**" command shows you "**—**" for any set, this means that **no permission** is granted. In another case, "**rwx**" indicates that all types of permission are granted, including reading, writing, and executing.

Now you have some background related to file permissions, and it will help you understand the **chmod command** working more efficiently.

## What is chmod command in Linux

The “**chmod**” is an acronym for “**change mode**”. It modifies the access of your system directories, files, and scripts. The “**chmod**” command has specific **modes** that determine the **permission** for modification. These modes are represented by **numerical form (letters)** or **symbolic form (octal numbers)**. When you use the chmod command with numerical form, it follows the below-given syntax:

**$ chmod [Options] [Filename]**

- In numerical representation, we have these **options**:
- “**0**” represents “**no permission**”.
- “**1**” represents “**execute permission**”. Enter the directory and access files
- “**2**” represents “**write permission**”. Delete or modify the files in a directory
- “**4**” represents “**read permission**”. List the files within the directory
- If you want to use the symbolic representation, then **chmod** command syntax will be written as follows:

| | Read | Write | Execute |
|---|---|---|---|
| Octal | 4 | 2 | 1 |
| Symbolic | r | w | x |

When assigning permissions using the **octal** method, use a 3 byte number such as: 760.

The number 760 translates into: Owner: rwx; Group: rw; Other (or world) no permissions.

Another scenario: 733 would translate to: Owner: rwx; Group: wx; Other: wx.

```
$ chmod [Option1] [Operator] [Option2] [Filename]
```

We have the following **options** in the symbolic form:

“**u**” indicates file **owner**.

“**g**” indicates **groups**.

“**o**” indicates **others**.

“**a**” indicates **all users** as owner, group, and others (ugo).

Whereas the **chmod** command accepts the following **operators**:

“**+**”: This operator is utilized to **add specified permissions**.

“**–**”: This operator is utilized to **remove specified permissions**.

“**=**”: This operator is utilized to define the **exact file permission** for any user.

**Example 1: Setting "read by owner only" file permission using chmod command**

In this example, we will change the file permissions of "**testfile**" so that only the owner can read it. Other than this permission, no other group or user can read, write or execute this file. Even the owner will not have the access to execute and write something in the file. To do so, use "**4**" as a numerical representation of "**read-only**" and place it at the start of three character set, and adding "**0**" for the "**groups**" and "**others**" mode will not grant any permissions to those users:

- **$ chmod 400 testfile**
- $ ls -l testfile

- **Example 2: Setting "read by group only" file permission using chmod command**
- Place the "**4**" as "**group**" mode between the zeroes of "**owner**" and "**others**" mode. This sequence will associate "**ready by group only**" permission to the file:
- **$ chmod 040 testfile**
- $ ls -l testfile

you can allow the "**read by others only**" file permission by defining the "**004**" mode in the chmod command.

**Example 3: Setting "write by owner only" file permission using chmod command**

In numerical representation of the modes, "**2**" indicates the "**write**" permissions. Place the "**2**" at the start of the permission set, and add two zeros after that:

```
$ chmod 200 testfile
```

- **Example 5: Setting "read by everyone" file permission using chmod command**
- Using symbolic links, if you change the file permission to "**read by everyone**," then execute the below-given command in your terminal:
- **$ chmod a+r testfile**
- Here "**a**" represents "**all users**", "**r**" indicates "**read**" permissions, and the "**+**" operator is used to add the read permission to the specified users:

Example 1 :
Let’s change the assgn1_client.c permission so that the owner cannot write(w) in the file but can only read it.

BEFORE: -rw-rw-r-- mik mik assgn1_client.c

**COMMAND: chmod u=r assgn1_client.c**

AFTER: -r--rw-r-- mik mik assgn1_client.c

Example 2 :

Let’s restrict the permission such that the user cannot search the directory EXAM.

BEFORE: drwxrwxr-x mik mik EXAM

**COMMAND: chmod u=rw EXAM**

AFTER: drw-rwxr-x mik mik EXAM