cmd\_name [option(s)] [argument(s)]

*where,*

*cmd\_name* is the name of the command,

*options(s)* are used to provide extra functionality to the command and

*argument(s)* are parameters or data provided to the command, which it uses as input.

Remember, *options(s)* are optional to be used with commands but *argument(s)* are optional for some commands and compulsory for others.

List Files and Directory

To see the list of files use the command: ls

You can change the directory using the cd command. It's syntax is:

cd directory

*where,*

directory is the name or path of the directory. It can be either relative or absolute.

We can also use special symbols in place of directory such as cd ~ to traverse to home directory, cd / to traverse to root directory, and cd .. to traverse to parent directory of the current directory.

Every user is given a separate home directory.

Home Directory & Current Directory

Inside the console, you are always in a directory. On login, by default, you land in your home directory.

To see the present working directory use: pwd

To change the directory to your home directory use only cd command without any arguments.

**List Files and Directory**

To see the list of files use the command: ls

You can change the directory using the cd command. It's syntax is:

cd directory

where,

directory is the name or path of the directory. It can be either relative or absolute.

We can also use special symbols in place of directory such as cd ~ to traverse to home directory, cd / to traverse to root directory, and cd .. to traverse to parent directory of the current directory.

Every user is given a separate home directory.

**Home Directory & Current Directory**

Inside the console, you are always in a directory. On login, by default, you land in your home directory.

To see the present working directory use: pwd

To change the directory to your home directory use only cd command without any arguments.

**User manual of commands**

To get information about any linux command, we can use man command. We can use it as

man command\_name

It displays user manual of the command which contains all information about it.

**Create Directory**

You can create the directory using mkdir command in the following way:

mkdir <directoryname(s)>

Here (s) means that we can also create multiple directories at once by giving several names (separated by space) to the command.

**Create Directory - nested**

We can use the -p option with the mkdir command to create a directory inside another directory.

We can create nested directories without using this option too, but suppose we want to create a directory **dataset** inside a directory **project** and if there's no directory named **project**, then an error will occur.

mkdir -p will create directory **project** instead of showing an error and if **project** will be present beforehand then no change will occur.

**Delete a file**

To delete a file you can use rm command like:

rm file\_name(s)

For deleting directories we have to use -r option with the command.

Also, to delete all files of a particular extension, we can replace file\_name with \*extension. So, to delete all txt files in a directory, we can use rm \*.txt

**Create File**

A file is a sequence of bytes and represents data. It is found in a directory. A file could contain any kind of data: an executable program, data representing movies, music, pictures or plain text.

You can create an empty file using touch command. Please note that the extension of the file doesn't matter much in Unix.

**Create Text File Using Nano**

You can use a text editor to edit or create a text file. There are many editors in Linux such as nano, pico, vi, emacs. Let's try to use nano for creating a file

Launch nano to edit:

nano myfirstfile.txt

Type the following text into it:

He walked into his exile

Press Control+x, you will get a prompt at the bottom of the screen asking you to "Save modified buffer (Answering No will DESTROY CHANGES)". Press y as we want to save the changes, and then Enter to Save Changes and exit the nano editor.

Alternatively you can create the file with this command

echo "He walked into his exile" > ~/myfirstfile.txt

**Copy File**

To copy files and directories we use the cp command. It's syntax is:

cp source destination

where,

source can contain single or multiple files and directories and

destination can contain a single file or directory.

**INSTRUCTIONS**

Please make the copy of myfirstfile.txt to myfirstfile\_copy.txt by using the command

cp myfirstfile.txt myfirstfile\_copy.txt

**Use cases of cp command**

The operation performed by cp source destination depends on what source and destination are.

Its use cases are:

When source and destination arguments are both files, the first file gets copied to the second one. If the second file is not present, then it is created first.

When source and destination arguments are both directories, the source directory gets copied into the destination directory.

When the source argument has single or multiple files or directories, they are moved to the destination directory. Remember, in this case, the destination must be a directory.

**Copy a file into another directory**

Create a directory with a name myproject using the command:

mkdir myproject

Copy the file myfirstfile.txt created in the previous exercise to myproject directory

cp myfirstfile.txt myproject

# **Copy files of a particular extension**

We can also copy all files of a particular extension to a directory by replacing source-name with \*.extension.

# **Copy a directory into another directory**

We have to use -r or -R option with cp to copy directories. Remember, the directory gets copied along with it's sub-directories.

1. Create a directory "src"
2. mkdir src
3. Create a file in "src"
4. touch src/myfile.txt
5. Create a directory "proj"
6. mkdir proj
7. Copy the src into "proj"

cp -r src proj

# **Move Files & Directories**

mv command is used to move or rename files and directories. It's syntax is same as cp, which is:

mv source destination

mv myfirstfile.txt firstfile.txt

# **Use cases of mv command**

Use cases of mv source destination are:

1. For renaming, both source and destination must be a single file.
2. When the source argument contains a single file and destination is an existing directory, then the source file gets moved to the destination directory.
3. When the source argument contains multiple files or directories, then the destination must be a directory. The files and directories are moved to the destination directory.

# **Delete Files & Directories**

To delete a file or a folder, use rm command

# **Remove a directory**

To delete a folder, use rm command command with -r option.

# **Seeing Inside File - cat, tail, head**

To see what is inside a text file, you can use either cat, tail or head command.

Using cat you can see the whole content of the file:

cat myfirstfile\_copy.txt

Do not use this command to look inside a huge file. For the huge file, you can use the more command which would display the content of a file in a paginated way:

more myfirstfile\_copy.txt

tail shows you the last few lines of a file

tail myfirstfile\_copy.txt

By default tail shows you only last 10 lines, you can change it using the command line option. For example, to see the last 20 lines, you can use

tail -20 myfirstfile\_copy.txt

We can use the -f option with tail for displaying newly appended lines of a continuously growing file. This option is mainly used to monitor the growth of log files. So, the console will keep on displaying the new lines as soon as there will be an update in the file.

tail -f access.log

To terminate it, press Ctrl + C.

If you are interested in the first few lines, you can use the head command. By default head shows you only first 10 lines, you can change it using the command line option. For example, to see the first 20 lines, you can use

head -20 myfirstfile\_copy.txt

# **Use find command**

Sometimes you might want to search for a particular file based on various attributes of a file such as size or name etc. The find command comes very handily for such use cases. It's syntax is-

find [path...] [expression]

where,

path contains the directory from where to start searching or directories in which find will search and

expression is made up of search patterns and actions. It determines what to find.

# **Searching files by name**

To search by a particular file name, we use the -name option with the command. We can also use \*extension instead of a name to search for all files of a particular extension.

For example, we can find all the txt files in the current directory by-

find . -name '\*.txt'

where, . indicates the current directory. So, the search will start from the current directory and will be performed in all it's sub-directories too.

# **Searching files by size**

We can use the -size option with the find command to search files by size. So, to search all files of size less than 1 MB, we can use:

find . -size -1M

where, - sign before 1 tells us that we want to search for files less than the mentioned size. We can use + to search for files greater than the particular size. If we don't specify any sign, then find will search for all files having a size equal to the mentioned size.

M denotes megabyte here. We can use b for 512 byte blocks which is also by default, c for bytes, k for kilobytes and G for gigabytes.

# **Extra functions of find command**

We can also search files of a particular type by using -type option with find command. We can search all directories present in the current directory by-

find . -type d

where d indicates a directory. We can use f for searching for files.

We can use the -exec option to execute a command after finding the respective file. So, we can remove all files of size greater than 1 megabyte by-

find . -size +1M -exec rm {} \;

where, {} \; tells exec to execute command with the result.

# **Use grep command**

If you want to locate a word in files, you can use the grep command. Grep lists all the lines from files in which a particular word exists. Examples of grep

grep myword file1 file2

If you want to search in files recursively - inside every subdirectory of a directory, use the following command

grep -r myword directory

If you want to search case insensitive, use -i switch

grep -i myword file1 file2

### Practical Use Cases

grep is a very powerful tool. It can somewhat behave like where clause in SQL queries.

Few Examples are:

1. On a web server, you could filter only the errors from a log file
2. Say you have a directory containing the temperature of various cities and you are looking for temperatures of the city having the name as nyc, you could easily do: grep nyc tempdirectory/\*

# **Use wc command**

To find the number of lines, words, and characters, use wc command. It's syntax is:

wc file\_name

We can use the -l option to just count the number of lines, -w for words and -m for characters. We can also use -L to display the length of the longest line.

Using the following wc command, which output did you get:

wc /cxldata/big.txt

Top of Form

128457 1095695 6488666 /cxldata/big.txt

Line word caharacters file name

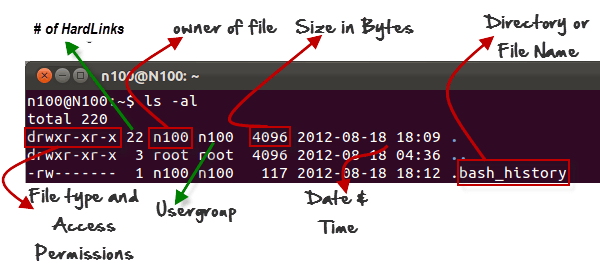
# **Permissions - Overview**

Permissions in Unix are an integral part of the operating system.

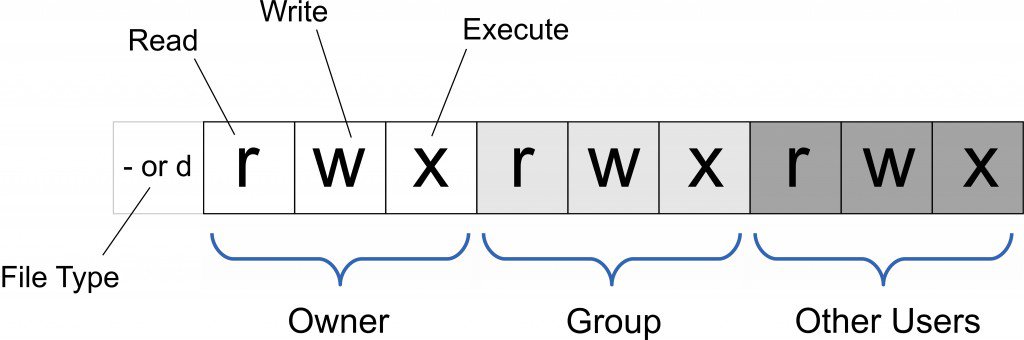
You can see the details of all files in the current directory by using options -l and -a combinedly with ls command:

ls -al

Every file has an owner (also referred to as the user), a group and permissions. A user can be part of many groups. A group can have many users.



The permission attributes of a file signify who can do what.



In file type, - represents file and d represents directory.

# **List of files**

To see the list of files and directories, we use the ls command. It's syntax is:

ls file\_name

where, file\_name can contain zero or multiple files or directories. If no file\_name is specified, then it just returns all the files and directories in the current directory.

Generally used options with ls command are:

1. -l : It will show the result in a long listing format. It displays all the information about a file or a directory.
2. -a : It will display all files including the hidden files.
3. -t : It sorts the list by modification time, showing the last modified file first.

# **Permissions - Using chmod To Change**

You can change the permissions of a file using chmod command: chmod permission\_cmd myfile

You can allow or disallow the user (u), group (g) or other(o) the following actions: read (r), write (w) and execute (x).

So, if you want to allow the user (the person who owns it) to execute the file:

chmod u+x myfile

And to disallow the user (the person who owns it) to execute the file:

chmod u-x myfile

Say you want to give the rw (read & write) permissions to owner and group of a file, you will have to use the following command:

chmod u+r,u+w,g+r,g+w myfile

or

chmod u+rw,g+rw myfile

To give members of a group ability to modify a file, use:

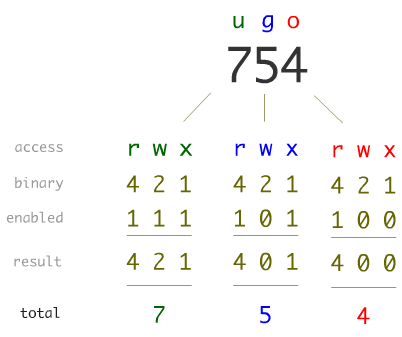
chmod g+w myfile

# **Permissions - Numeric**

You can also use numbers to represent the permissions.

4 is for Read, 2 is for Write and 1 is for Execute permission.

The following diagram shows if the user has all three permissions rwx and group has only read and execute permission and others have only read permissions, the rwxr-xr-- can be represented by 754.



So, for a file with permissions rwxr-xr-x, the owner has rwx permission which 4+2+1=7 and group and others have r-x which means 4+0+1 = 5. Thus the command you would use to set such permission would be: chmod 755 myfile.

# **Permissions - Advanced**

There is a special user called root on Unix systems which has all the privileges which can permit any user on any file.

Some users who are allowed to act on behalf of the root are called sudoers. This list of sudoers can be edited using sudoedit command. Such users are allowed to run commands as if they are root using: sudo .

You can change the owner of a file using chown and change the group of a file using chgrp command. Please note that changing the owner of the file is possible only with an administrative account.

A user can delete a file if

Top of Form

User owns the file

If user is a member of file's group and group has write permissions

If user's login is 'root'

If users is a member of sudoers

If the file has the following permissions: -rwxrwxrwx

# **Process**

Every program or command is a sequence of instructions stored as a file. You run it on the shell or by any other means such as double clicking it from user interface.  
When it is run, the content of the program is read from file and loaded into the memory and then instructions are executed by operating system.  
All the commands that we have been using like ls, cat are program.

# **Find information about processes**

While a program is running, it is called a process. A process is uniquely identified by PID - process id. To find out the information about all the processes, you can use the following commands: pstree, top, ps.

ps lists the processes of the system. Without any arguments, it lists only your processes. To see all of the processes by all users, run

ps aux

where,

option a displays the processes of all users and not just our own,

option u displays information in user-oriented format and

option x displays the processes running without a controlling terminal. When you run the above command, you can see many ? sign in column TTY(Terminal Type). So, x displays these processes. These processes are mainly background processes.

The first column mentions the user id who has started the process.

To continuously monitor all processes, use the top command. This command is usually used to monitor which processes are running, taking up most CPU or Memory.

# **BSD syntax of commands**

We use ps aux to display the processes of all users. But, we notice here that we don't use the - sign with options that we were using till now. So, using options without - is known as BSD syntax to write commands. The other one (one with -), which we use regularly, is known as Standard syntax to write commands.

ps command has been in use since the early versions of Unix. When it was written, BSD syntax was in use to write commands. So, when standard syntax came into the scene, we started to use - with options. But people who used BSD syntax didn't like that they had to modify their programs and habits due to the new syntax.

So, the commands were re-defined with the standard syntax without removing their compatibility with the BSD syntax. So, the commands which are very old (which were written when BSD syntax was in use) like ps command supports both BSD syntax and standard syntax.

Note that ps -aux is distinct from ps aux. Feel free to use man ps to know more about it.

# **Background Processes**

So far whenever we ran a command, if it is taking time, we had to wait till it finishes before we could type another command. If a process is taking the time and does not require input from you, you would like to run it in the background.

To run a program in the background, put an & at the end of the command:

mycmd &

# **Send a process in background**

There is a command called sleep which can make you wait or suspend your terminal for a specified time. So, it pauses the execution of next command for a given time.

So, you can suspend your terminal for 5 seconds like:

sleep 5

We can also use s suffix with number for seconds(which is by default), m for minutes, h for hours and d for days. So you can use sleep 2m to suspend terminal for 2 minutes.

Please send sleep command in background using:

sleep 1000 &

Note - You will get a process ID on the execution of the above command

To kill any running process, please use the command:

kill processid

Basically, kill lets you send a signal to the running process. For example, to send the terminate signal, you can use

kill -9 processid

You can use kill -l to get list of all available signals.

In Unix, a job is a group of one or more processes. One of the ways of creating job is when we put a process in the background.

You can see the list of jobs by using the command:

jobs

The number that you see in square brackets [] is the job id. This is different from the process id. To bring a process to foreground, you can use

fg %jobid

or simply

fg jobid

If jobs command lists following:

[1]- Stopped tail -f mycmd.sh

[2]+ Stopped top

[3] Running sleep 1000 &

# **More - Interacting with processes**

To send a running process to background, first press Ctrl+z to suspend it and then type bg to send the suspended process to background.

When you exit the shell or disconnect, the processes you were running get killed sometimes. To keep a process running in the background even if the shell has been disconnected, use

nohup

or

screen

# **Process hierarchy**

A process (parent) can execute another process (child). When a parent process is killed all the child processes are automatically assigned to the main system process called **init**. The init process is the first process that is started on your computer and is numbered as **1**. If this process is killed the system will shutdown.

To see the tree of processes, you can use the command

pstree

A user can kill only the processes that the user has created not the processes of other users. Only root can kill the processes of other users.

# **Writing first shell script**

A shell script is a file containing a list of shell commands which would otherwise could be run individually on the OS shell prompt.

**Note:** In Unix, the extension doesn't dictate the program to be used while executing a script. It is the first line of the script that would dictate which program to use. The first line starts with the sequence of two characters **#!** that is called **shebang**. This is followed by the program that tells the operating system which interpreter to use to parse the rest of the file.

The Shebang interpreter directive takes the following form:

#!interpreter [arguments]

In the example below, the program is /bin/bash which is a Unix shell.

Let's create a simple command that prints two words:

1. Open a text editor to create a file myfirstscript.sh:
2. nano myfirstscript.sh
3. Write the following into the editor:
4. #!/bin/bash
5. name=linux
6. echo "hello $name world"

If you want that every variable must be declared before it can be used, then add **set -u** at the top of the script after the shebang line.

#!/bin/bash

set -u

name=linux

echo "hello $name world"

echo $surname

1. Press Ctrl+X to save and then Y
2. It will ask for a filename, check if its myfirstscript.sh. If not enter the same filename.
3. Hit Enter key to exit
4. Now, by default, it would not have executable permission. You can make it executable like this:
5. chmod +x myfirstscript.sh
6. To run the script, use:

./myfirstscript.sh

# **Every Program Returns a Value**

Every program returns a value to the operating system. It is also referred to as the exit status. In Linux, a program returns 0 if successful. Otherwise a non-zero error number.

To check the return value of the previous command you can check the value of a special variable ?

echo $?

# **Networking: Sockets and ports**

There are some programs such as a web server that need to keep running all the time. Such programs are called services.

You can communicate with such programs even from another computer. These programs or processes bind themselves to a port. In other words, such programs listen on a port. No two programs can listen on the same port. Every computer is a given an IP address which is unique in the network so that other computers or users on other computers can connect to such systems.

For example, a web server listens on 80 port and an email server listens on 25 port.

You can connect and talk to any service using nc command in the following way:

nc computer\_name(or ip\_address) port\_number

The way a person refers to himself as "I", a computer refers itself as localhost or IP Address 127.0.0.1. it is also called as a loopback address.

# **Files & Directories: linking**

If you want to give multiple names to a single file without copying the content, you can create the link.

1. Create a file:
2. nano orig\_text
3. Put the following text in it:
4. This is my valuable data
5. Save and Exit: Press CTRL+x and y
6. Check if the file has right data:
7. cat orig\_text
8. Create Link:
9. ln orig\_text mylink
10. Check if it is there:

ls -l mylink

# **Hard link and soft link**

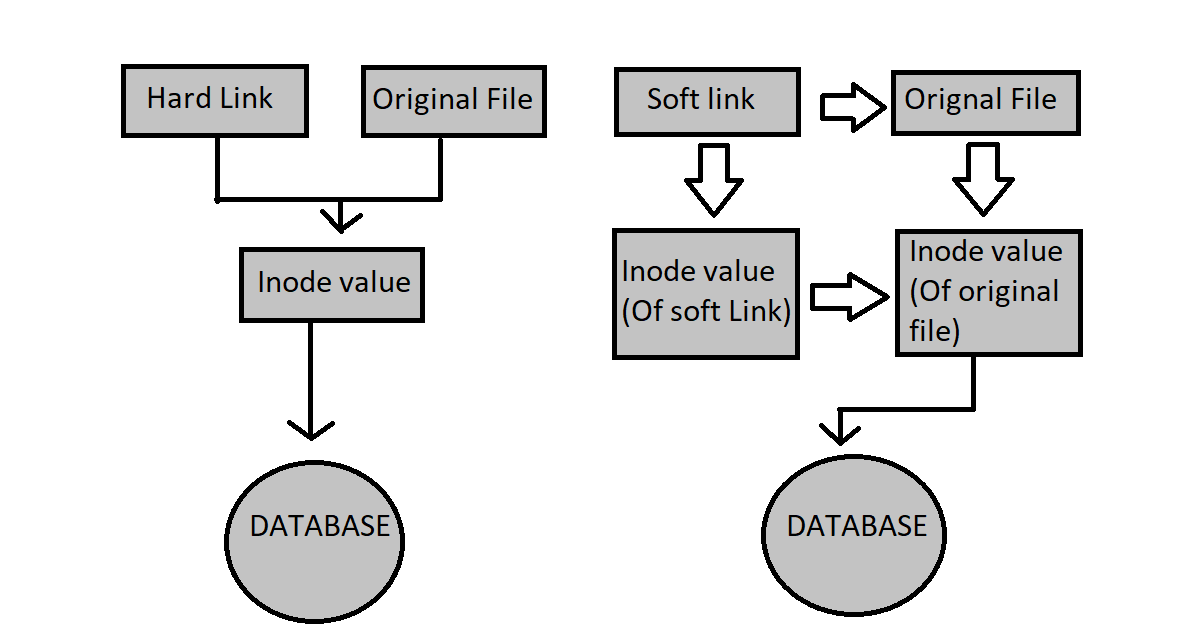
A link in UNIX is a pointer to the file. It is a connection between a file name and the actual data on the disk. It is of two types:

1. Hard links
2. Soft links (Symbolic links)

Hard link is the mirror copy to the original file. It means that even if the file is moved or deleted, the hard link will still contain the data of the original file. This is because the hard link is assigned the same inode value as the original file.

**Inode value-** Whenever a file is created in linux, an inode value is assigned to it. It is a unique index number assigned to each file in Linux/Unix system. When we access the file by its filename, internally, it is accessed through its inode value.

On the other hand, a soft link is just a reference to another file or directory. It just contains the path to the original file and not the content. If the original file is moved or deleted, the link will point to a non-existent file. This is because soft links are assigned a different inode number.



So, as we can see in the above diagram that even if the original file is deleted, hard link will still point to the same inode value. On the other hand, soft link will be broken as it doesn't have any direct connection with the inode value of the original file.

We use ln command to make link. It by default creates hard link. For symbolic link, we use ln -s.

# **Files & Directories - Symbolic links**

A symbolic link points to a file. In case, the original file is deleted, the symbolic link would be pointing to non-existing file.

You can create a symbolic link to a directory too.

1. Create a file:
2. nano orig\_text1
3. Put the following text in it:
4. This is my not-so-valuable data
5. Save and Exit: Press Ctrl+x and y
6. Check if the file has right data:
7. cat orig\_text1
8. Create Link:
9. ln -s orig\_text1 myslink
10. Check if it is there:
11. ls -l myslink

You should see something like this

lrwxrwxrwx 1 sandeepgiri9034 sandeepgiri9034 10 Dec 14 11:45 myslink -> orig\_text1

1. Check the contents of myslink using:

cat myslink

hard link is not allowed on directory but soft is allowed

# **Chaining Unix Commands**

Say, you want to execute a command only if another command is successful then you use &&.

And if you want a command to be executed if another command has failed, you use ||.

1. Create a file tea\_ready:
2. touch tea\_ready
3. The following command would print "Tea is ready" if tea\_ready file exists:
4. ls -l tea\_ready && echo "Tea is ready"
5. Delete the file tea\_ready:
6. rm tea\_ready
7. Check the command from #2 again:
8. ls -l tea\_ready && echo "Tea is ready"
9. The following will print "Tea is not ready":
10. ls -l tea\_ready || echo "Tea is not ready"
11. You can use brackets to avoid confusion:
12. (ls -l tea\_ready && echo "Tea is ready") || echo "Tea is not ready"

Mark as Completed

# **Redirecting the output of a program**

The output of a program can be saved to a file:

myprogram > myfile

If myfile does not exist, it will be created. If it exists it will be overwritten.

1. un the following command to save the output of echo to a file:
2. echo "hello" > hello.out
3. Append world to it:

echo "world" >> hello.out

# **Pipes - Introduction**

If we want to send the output of one program to another, we can use the pipe. A pipe is denoted by |.

echo command prints on the standard output whatever argument is passed to it.

echo "Hi"

wc command prints the number of characters, words, and lines out of whatever you type on standard input. Start wc command, type some text and press Ctrl+d to end the input:

wc

hi

how are you

[CTRL + d]

**Output-**

2 4 15

**Note-** wc command also counts new line (**\n** ) character.

Now, if we want to count the number of words, characters in the output of any program, we can pipe the output in the following way:

echo "Hello, World" | wc

You can also save the results using redirection, in the following way:

echo "Hello, World" | wc > wc\_results

Please execute the above command to succeed in the assessment.

# **Filters**

Pipes are very powerful. They can be chained to solve many problems. So, most of the commands are built in a way that they can be used with pipes as long as they read from standard input (keyboard or pipe) and write to standard output (screen or pipe )

Such programs that can be used with pipes are generally called filters. Command examples of filters are:

wc - for counting the letters, words, and lines in the input

grep - displays only the lines from the input in which keyword (which is passed as argument) is found.

sort - sorts/orders the input lines lexically (alphabetically) by default but can be changed

more - displays the input in a page-wise manner

cat - displays the content of the file passed as an argument

sed - substitute a word with another word: sed 's/word/another\_word/'

tr - translate character ranges. For example to lowercase characters in input you can use:

tr 'A-Z' 'a-z'

uniq - Display the unique input lines. The input lines need to be sorted. If you want to display frequency, use:

uniq -c