

2. Lab# 02 File and Directory Management

In the last lab, we learn Linux commands. In this lab, I will explain more Linux commands like **man**, **date**, **who**, **sort**, wild cards, etc. You will also learn about file and directory security and access permission and how to grant and remove access permission from a user who reads, writes, and executes permission.

2.1 Learning Outcome

After the completion of the lab, the student will be able:

- To set different permissions to a file and a directory (Read, Write and Execute). and also set different permissions for different users (Owner, group, and others)
- To list all the users on the system and display the user ID
- To use the manual page (**man**) and
- To use wild cards

2.2 Some more important commands

<u>Command</u>	<u>Description</u>
date	The date command displays the current date and time on the screen. The system administrator sets the dates users cannot change them.

Example

\$date

Wed Feb 7 10:35:41 PKT 2022

There are several options in which the date can be displayed. If you want to see only the date, you can do like this:

\$ date +" %d"

07

for time only

\$date +" %r"

10:38:11 AM

%Y = Year. %H = Hour(00..23), %I = Hour (01..12), %m = Month

<u>Command</u>	<u>Description</u>
clear	Clears the screen.
echo	Echoes back whatever you type on the command line after echo . Options -n doesn't begin a new line after echoing the information.

Examples

echo Hello there

echo -n Hello

Command

sort

Description

Sorts a column in a file in alphabetical order. The output is default displayed on your terminal, but you can specify a filename as the argument or redirect the output to a file.

Option

- r** Sorts in reverse order
- b** Ignores leading blanks
- f** Ignores the distinction between lowercase and upper case letters
- o** Sorts the output in the specified file
- n** Numbers are sorted by their arithmetic values

2.2.1 More on Sorting

The sort command sorts the specified file on a line-by-line basis. If the first characters on two lines are the same, it compares the second characters to determine the order of the sort. If the second characters are the same, it compares to the third characters, and this process continues until the two characters differ or the line ends. If two lines are identical, it does not matter which one is placed first.

Example

Following is a file whose name is **sortfile**. Please observe the following sort operation: (**Note:** Create the file **sortfile** using **nano** and put the following content in that file.)

```
file to be sorted
File To be Sorted
25 years old
End of file
apple on the table
6 apples
```

\$ sort -f sortfile

```
25 years old
6 apples
apple on the table
End of file
file to be sorted
File To be Sorted
```

\$ sort -fn sortfile

```
apple on the table
End of file
file to be sorted
File To be Sorted
6 apples
25 years old
```

\$ sort -f -r sortfile

```
File To be Sorted
file to be sorted
End of file
```

apple on the table
6 apples
25 years old

<u>Command</u>	<u>Description</u>
wc	('word count') Counts the number of words, lines, and characters in a file. Options -c Display only the number of characters in the file. -l Display only the number of lines in the file. -w Display only the number of words in the file.
who	who command lists the login names, terminal lines, and login times of the users who are currently logged on to the system.

Example

```
$ who
mumtaz pts/3 Feb 7 14:54
qamar pts/0 Feb 7 09:21
yousuf pts/2 Feb 7 10:12
```

If you type **whoami**, Linux displays who the system thinks you are:

```
$ whoami
mumtaz pts/2 Feb 7 09:25
```

<u>Command</u>	<u>Description</u>
head	The head command will output the first part of the file

The syntax of the **head** command is pretty straightforward:

```
$head [option]... [file]...
```

Example

```
$head file_name
```

This will print the first ten lines of the file if it contains more than ten lines.

```
$ head -n 4 sortfile
```

The output of this command is the first 4 lines of the file sortfile.

<u>Command</u>	<u>Description</u>
tail	The tail command will output the last part of the file

The syntax of the **head** command is pretty straightforward:

```
$ tail [option]... [file]...
```

The **tail** command will, by default, write the last ten lines of the input file to the standard output:

```
$tail sortfile
```

To output the last seven-line of the file, we use the command **tail**:

Stail -n 7 sortfile

2.3 File and Directory Security

Files/Directories can be created by setting permissions, allowing people to **read, write, or execute your file**. Each file on the machine divides the users of the machine into three categories:

- The file's **owner** (who creates the file)
- A **group** of users
- **Other** users

There is one more type of user; the **superuser**. The system administrator may be the only superuser, but often several people have access to the superuser's password. Anyone logged in as the superuser has total access to every file directory in the system.

2.3.1 Types of Access

There are three types of access:

- **read**
- **write**
- **execute**

2.3.2 File Permissions

If a file has **read** permissions, it can be examined at a terminal, printed, viewed by an editor, and so on. If it has **write** permissions, the file's contents can be changed (for example, by an editor), and the file can be overwritten or deleted. That program can be run if it has to execute permissions and is a binary program or a shell script. Having a type of access is referred to as having **access permission**. You can change the access permission for your own file. For example, if you do not want anyone else to access a file, you can remove read access for anyone but you. If you want other users in your group to be able to write to a group of files, you can extend write permission to the group. Each user type (the owner, the group, and others) can have any combination of the three access types for each file or Directory.

2.3 Directory Permissions

Directories have permissions modes that work similarly to file permission modes. However, the directory access permissions have different meanings:

Read: The read (**r**) permission in a directory means you can use the **ls** command to the filenames.

Write: The write (**w**) permission in a directory means you can add or remove files from that Directory.

Execute: The execute (**x**) permission in a directory means you can use the **cd** command to change to that Directory.

Note:- Your installation has a default setup for all newly created files and directories. You can check your default access through the **ls -l** command.

2.4 Access Specification

When you create a file or directory, it comes into existence with default access specifications. It may give all access permissions to the owner, just read and write permissions to the group, and just read permission to everyone, or there may be any situation. The following figure shows how different groups and characters represent access permissions.

drwx	rw-	r--
user	group	others

The first character indicates whether the file is a directory (d) or not (-) (- is for file). The next nine places are divided into three sets, each length 3. The first set of three indicates the **owner access**, the next set of three suggests the **group access**, and the final set of three shows the access for everybody else. The maximum access is represented by **rwX**, indicating **read**, **write**, and **execute**. Whenever a dash (-) appears, access permission is not given.

2.4.1 Checking Access

A command can check the access privileges in given files and directories.

```
$ ls -l
-rw-r--r-- 1 saleem stud 700 June 19 08:00 data.file
access     links owner group size last change name
```

In this example, the first character is “-”, “indicating that the object is a file. The owner's group suggests that the owner, i.e., Saleem, can read and write but cannot execute. Any group member can read the file but not execute it. Also, someone can read this file without writing it or executing it. It is the authority of the administrator to create a group of users. Each group has some users and is given a unique name. Each group also has a unique number (group id), and each user has a unique number, called **user-id or UID**.

<u>Command</u>	<u>Description</u>
id	It gives the user's name, the groups they are a member of, both names and numbers, and the user's user-id and current group-id.

Example

```
$ id (enter)
```

```
uid=275(john1), group=50(staff)
```

```
$id chris
```

```
uid=145(charis) gid=12(ugrads) groups=12(ugrads), 417(proj)
```

This shows that **chris** is a member of groups **ugrads**, and **proj**, with **GID** numbers 12 and **417**, respectively. Currently, chris is allotted to group ugrads.

The access privileges can be changed. A command **chmod** is used to change the access privileges.

<u>Command</u>	<u>Description</u>
chmod	Change access permission for one or more files.

chmod user-type [operations][permissions] filelist

user-type

u User or owner of a file.
g Group that owns the file.
o Other.
a All three user types.

operations

+ Add the permission.
- Remove the permission.
= Set permission; all other permission resets.

Permissions

r Read permission.
w Write permission.
x Execute(run) permission.

Examples

\$chmod u-w result.comp

Write permission for owner (user) removed from file result.comp.

\$chmod go+r stock

Group and other users get read access for file stock.

\$chmod g=r+x myfile

It will set group access for reading and executing but not writing to it. However, you can specify the new mode as a three-digit number that is computed by adding together the numeric equivalents of the desired permission. The following table shows the numeric value assigned to each permission letter.

owner			Group			Other		
r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1

Example

Change access permission to allow the read, write, and execute permission to all users.

\$ chmod 777 final.

2.5 Manual Page

The man command is a built-in manual for using Linux commands. It allows users to view the reference manuals of a command or utility run in the terminal. The man page (short for manual page) includes a command description, suitable options, flags, examples, and other informative sections. The basic **man** command syntax is:

man [option] ... [Command name] ...

2.5.1 Options and Examples

2.5.1.1 No Option

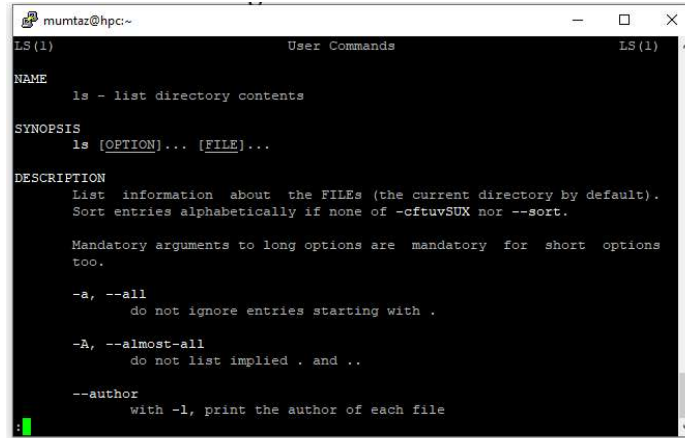
It displays the complete manual of the command.

Syntax:

\$man [command name]

Example

\$man ls

A terminal window titled 'mumtaz@hpc:~' with a black background and white text. It displays the manual page for the 'ls' command. The output includes sections for NAME, SYNOPSIS, and DESCRIPTION. The DESCRIPTION section provides details about listing files and directories, including options like -a, -A, and --author.

```
mumtaz@hpc:~  
LS(1) User Commands LS(1)  
NAME  
ls - list directory contents  
SYNOPSIS  
ls [OPTION]... [FILE]...  
DESCRIPTION  
List information about the FILES (the current directory by default).  
Sort entries alphabetically if none of -cftuvSUX nor --sort.  
Mandatory arguments to long options are mandatory for short options too.  
-a, --all  
do not ignore entries starting with .  
-A, --almost-all  
do not list implied . and ..  
--author  
with -l, print the author of each file
```

The command 'ls' manual pages are returned in this example.

2.5.1.2 Section number

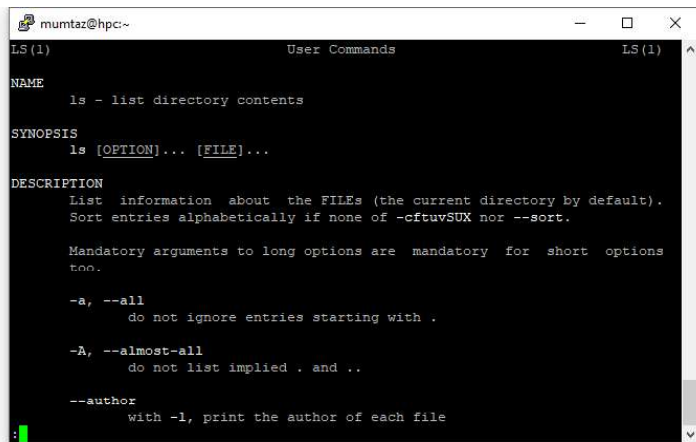
Since a manual is divided into multiple sections, this option displays only a specific section of a manual.

Syntax:

\$man [section number] [command name]

Example

\$man 1 ls

A terminal window titled 'mumtaz@hpc:~' with a black background and white text. It displays the manual page for the 'ls' command, specifically section 1. The output is identical to the previous example, showing the NAME, SYNOPSIS, and DESCRIPTION sections.

```
mumtaz@hpc:~  
LS(1) User Commands LS(1)  
NAME  
ls - list directory contents  
SYNOPSIS  
ls [OPTION]... [FILE]...  
DESCRIPTION  
List information about the FILES (the current directory by default).  
Sort entries alphabetically if none of -cftuvSUX nor --sort.  
Mandatory arguments to long options are mandatory for short options too.  
-a, --all  
do not ignore entries starting with .  
-A, --almost-all  
do not list implied . and ..  
--author  
with -l, print the author of each file
```

In this example, the manual pages of command 'ls' are returned in section 1.

2.5.1.3 -f Option

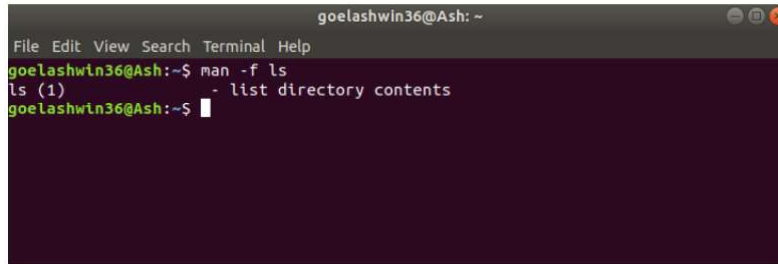
One may be unable to remember the sections in which a command is present. So this option gives the section in which the given command is present.

Syntax:

\$man -f [command name]

Example

\$man -f ls



The manual is generally split into eight numbered sections, organized as follows:

Section	Description
1	General commands
2	System calls
3	Library functions , covering in particular the C standard library
4	Special files (usually devices, those found in /dev) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellaneous
8	System administration commands and daemons

Just press the Q key on the keyboard to exit the man command.

2.5.1.4 Other Help Command

There are other ways, apart from "**man**," to get help with Linux commands that you do not know. Explore the workings of the commands "**help**," "**info**," "**whatis**," and "**apropos**" yourself.

2.6 Wildcards

Wildcards are special characters (metacharacters) applied with a command. These characters operate on a group of files/directories. Most of the time, these commands have wild cards and operate on a group of files/directories. These have some standard features in their names. When a wild card(s) and other characters are grouped together, they form a "**pattern**."

For example, in case there is a group of files in the current directory, in this group following files have a common feature in their names that start with the character "t."

--- t t1 tile1 tile2 ---

In order to select these files from the group of files, a command like '**ls**' must be operated with a suitable wildcard.

The symbols used for wild cards are the following.

Symbol

Meaning

*	It matches zero or any number of characters
?	It matches any single character in a file/directory
[]	It matches any one character in the bracket.

Example

Following are the files in a given directory; the results returned by different wildcards when used with '**ls**' is shown below.

fan fat fest foo loo on p1 p2 p7 p9 t t1 test tile1 tt2

Note: You don't have to create all the above files using **nano**, as that would be cumbersome. Instead, you can use the **touch** command and specify all the filenames as arguments. It would create empty files for you with just one command, as in:

touch file1 file2 myfile

2.6.1 Wildcard *

ls *t returns **fat fest t** (the last character should be 't' with any number of preceding characters)

ls t* returns **t t1 tile tt2** (the first character should be 't' with any number of following characters)

ls t*t returns **test** (first and last characters should be 't' with any number of characters between them)

ls f*t returns **fat fest** (first and last characters should be 'f' and 't' respectively, with any number of characters between them)

ls *oo returns **foo loo zoo** (the last two characters should be 'oo' with any number of preceding characters.)

2.6.2 Wildcard ?

ls t? returns **t1** (the first character should be 't' with **only one** character following it)

Note:- file named 't' has not been selected.

ls t?t returns **tat** (first and last character should be 't' with **only one** character between them)

2.6.3 Wildcard []

ls p[12] returns **p1 p2** (starting character should be 'p' and ending character should be '1' or '2')

\$ls p[1-9] returns **p1 p2 p7 p9** (starting character should be 'p', and ending character could be anything between '1' to '9')

???t Four-character file name. The first three characters may be any, but the last character should be 't.'

??[a-c] Three-character filename beginning with any two characters, but the last character should be 'a', 'b,' or 'c.'

[a-c][1-9] Two character file name starting with 'a', 'b', or 'c' and ending with any character between '1' and '9'

2.6.4 Wildcard !

\$ls p[!1-7] returns **p9** (starting character should be 'p' and ending character should not be '1' to '7')