

**Name= shaik Raakin**  
**Class =3rd sem**  
**Branch = cybersecurity**  
**usn=ENG24CY0192**  
**Roll no:64**  
**section:A**

**1Q)What does the command pwd, whoami, and hostname display?**

**sol)**

pwd= known as “print working directory” command shows the current directory path where the user is located.

whoami= command displays the username of the current user.

hostname= it displays the name of the computer or system in the network.

**2Q)Write the command to create a directory named “project” inside the /home/student folder and keep three .txt file into it. Give output snapshot.**

**sol)**

The command “mkdir” is used to create the directory , and the command “cd” is used to change the directory and then the command “touch” is used to create the text files.

```
raakin@ubuntu-25:~$ mkdir home
```

```
raakin@ubuntu-25:~$ cd home
```

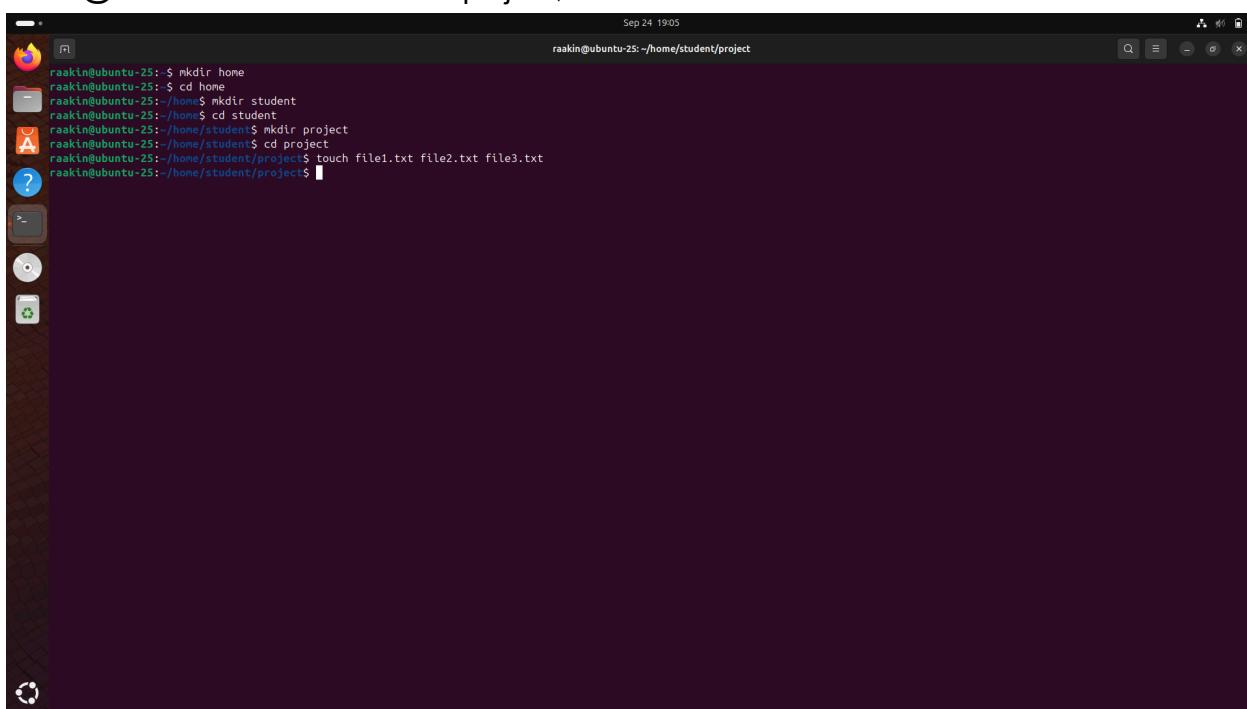
```
raakin@ubuntu-25:~/home$ mkdir student
```

```
raakin@ubuntu-25:~/home$ cd student
```

```
raakin@ubuntu-25:~/home/student$ mkdir project
```

```
raakin@ubuntu-25:~/home/student$ cd project
```

```
raakin@ubuntu-25:~/home/student/project$ touch file1.txt file2.txt file3.txt
```



The screenshot shows a terminal window on an Ubuntu desktop environment. The terminal window title is "raakin@ubuntu-25: ~/home/student/project". The terminal content shows the following command history:

```
raakin@ubuntu-25:~$ mkdir home
raakin@ubuntu-25:~$ cd home
raakin@ubuntu-25:~/home$ mkdir student
raakin@ubuntu-25:~/home$ cd student
raakin@ubuntu-25:~/home/student$ mkdir project
raakin@ubuntu-25:~/home/student$ cd project
raakin@ubuntu-25:~/home/student/project$ touch file1.txt file2.txt file3.txt
raakin@ubuntu-25:~/home/student/project$
```

**3Q)Explain the difference between absolute path and relative path with proper examples.  
sol)**

**absolute path** : provides the full path to a file or directory, starting from the root (/) and going through every directory until it reaches the specified file or directory.

eg:/home/raakin/home/student/project

**relative path**: provides a file or directory location in consideration to the current working directory.

eg:file1.txt

Absolute paths are always unambiguous, while relative paths are shorter and more convenient" when working exclusively inside one directory structure.

**4Q)What command will give you the already executed command traces in the terminal.**

**Give output snapshot.**

**Sol:**

In a Linux , the history command provides a list of all commands that have been run in the terminal session. Each command is numbered, and the number can be reused by simply entering “!number” to run the command again.

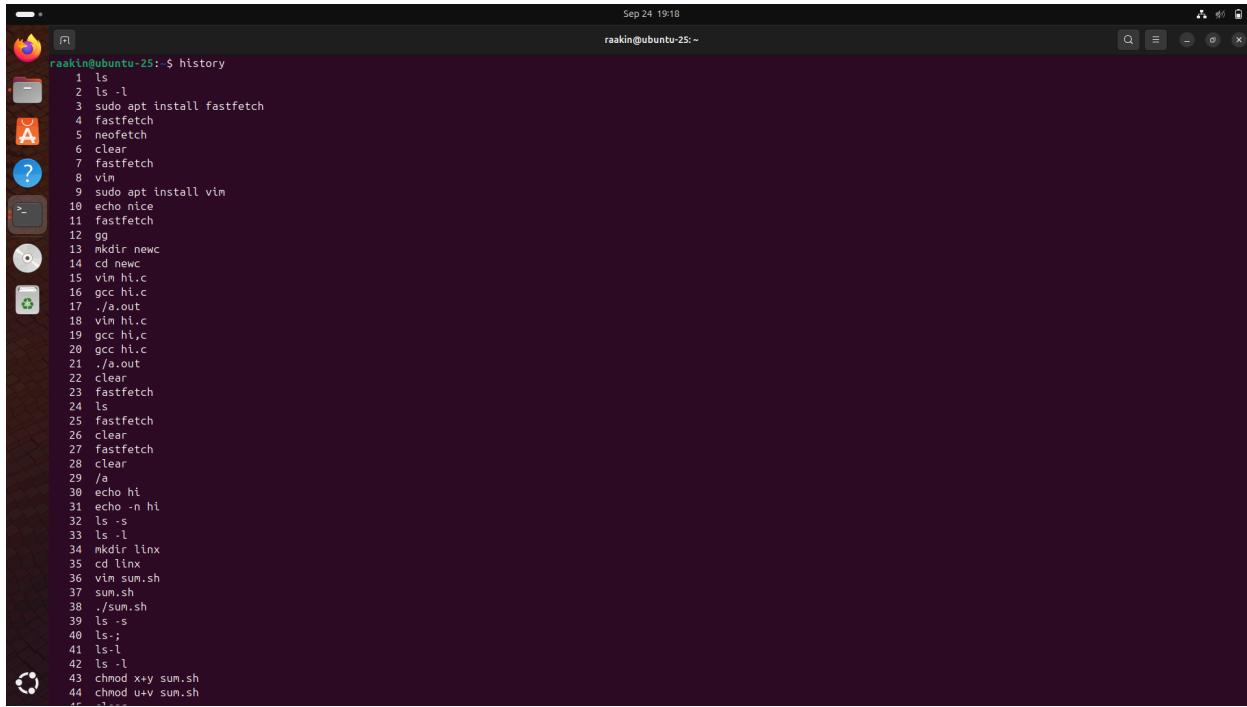
Eg: typing !5 will run the 5th command in the history.

This is especially useful because it allows you to recall the entire command rather than re-typing.

Eg:

raakin@ubuntu-25:~\$ history

```
1 ls
2 ls -l
3 sudo apt install fastfetch
4 fastfetch
5 neofetch
6 clear
7 fastfetch
8 vim
9 sudo apt install vim
10 echo nice
```



The screenshot shows a terminal window on an Ubuntu 25.04 desktop environment. The title bar indicates the session is 'raakin@ubuntu-25:~-' and the date and time are 'Sep 24 19:18'. The terminal displays the output of the 'history' command, which lists a series of command entries numbered from 1 to 44. The commands include basic file operations like 'ls', 'cd', and 'mkdir', system administration tasks like 'sudo apt install' and 'neofetch', and file editing with 'vim'. Some commands involve compilation with 'gcc' and running programs like 'sum.sh'. The terminal has a dark theme with light-colored text.

```
raakin@ubuntu-25: ~$ history
1 ls
2 ls -l
3 sudo apt install fastfetch
4 fastfetch
5 neofetch
6 clear
7 fastfetch
8 vim
9 sudo apt install vim
10 echo nice
11 fastfetch
12 gg
13 mkdir newc
14 cd newc
15 vim hi.c
16 gcc hi.c
17 ./a.out
18 vim hi.c
19 gcc hi.c
20 gcc hi.c
21 ./a.out
22 clear
23 fastfetch
24 ls
25 fastfetch
26 clear
27 fastfetch
28 clear
29 /a
30 echo hi
31 echo -n hi
32 ls -s
33 ls -l
34 mkdir linx
35 cd linx
36 vim sum.sh
37 sum.sh
38 ./sum.sh
39 ls -s
40 ls -;
41 ls -l
42 ls -l
43 chmod x+y sum.sh
44 chmod u+v sum.sh
45 ~
```

**5Q)Compare the working functionality of find and locate command. Which one is faster and why?**

**sol)**

find command checks for files and directories by searching the filesystem in real-time. It can search based on criteria like name, type, size, or modification time.

The locate command searches a previously built database (mlocate.db) of existing file paths rather than searching the live filesystem. Because locate uses an existing database to perform the search, it is faster than find, but it could show stale results if the database has not been updated via updatedb.

**6Q)Which command is used to modify file permissions in Linux? Give an example.**

**sol)**

The chmod (change mode) command is used to modify file permissions.

Eg:

```
raakin@ubuntu-25:~/linx$ gedit hi.zh
raakin@ubuntu-25:~/linx$ chmod +x hi.zh
raakin@ubuntu-25:~/linx$ ./hi.zh
hi
```

**7Q) A file has permissions -rw -r- r--. What does this mean?**

**sol**

This permission string is broken into parts:

- `-` → it is a regular file (not a directory).
- `rw-` → the owner can read and write.
- `r--` → the group can only read.
- `r--` → all other users can only read.

It means that everyone can read the file but only the owner can read it.

**8Q) Explain the difference between chown and chgrp with an example.**

**sol)**

**chown:**

The command chown modifies a file or directory's ownership to that of another user. By way of example, the command chown alice file1.txt would change file1.txt's ownership to so that the new owner is Alice.

**chgrp:**

The command chgrp, on the other hand, changes the group ownership of the file only. An example of this would be chgrp developers file1.txt where the file's group ownership would be changed to developers.

So in summary, chown changes both user and group ownership and chgrp focuses solely on group ownership.

**9Q) A file needs to be accessible by multiple users but only writable by the owner. How will you set permissions?**

**sol)**

To achieve this, you can assign read permissions to group and others, but keep write permission only for the owner.

eg:

```
raakin@ubuntu-25:~/home/student/project$ chmod 644 file1.txt
raakin@ubuntu-25:~/home/student/project$ ls -l
total 0
-rw-r--r-- 1 raakin raakin 0 Sep 24 19:03 file1.txt
-rw-rw-r-- 1 raakin raakin 0 Sep 24 19:03 file2.txt
-rw-rw-r-- 1 raakin raakin 0 Sep 24 19:03 file3.txt
raakin@ubuntu-25:~/home/student/project$
```

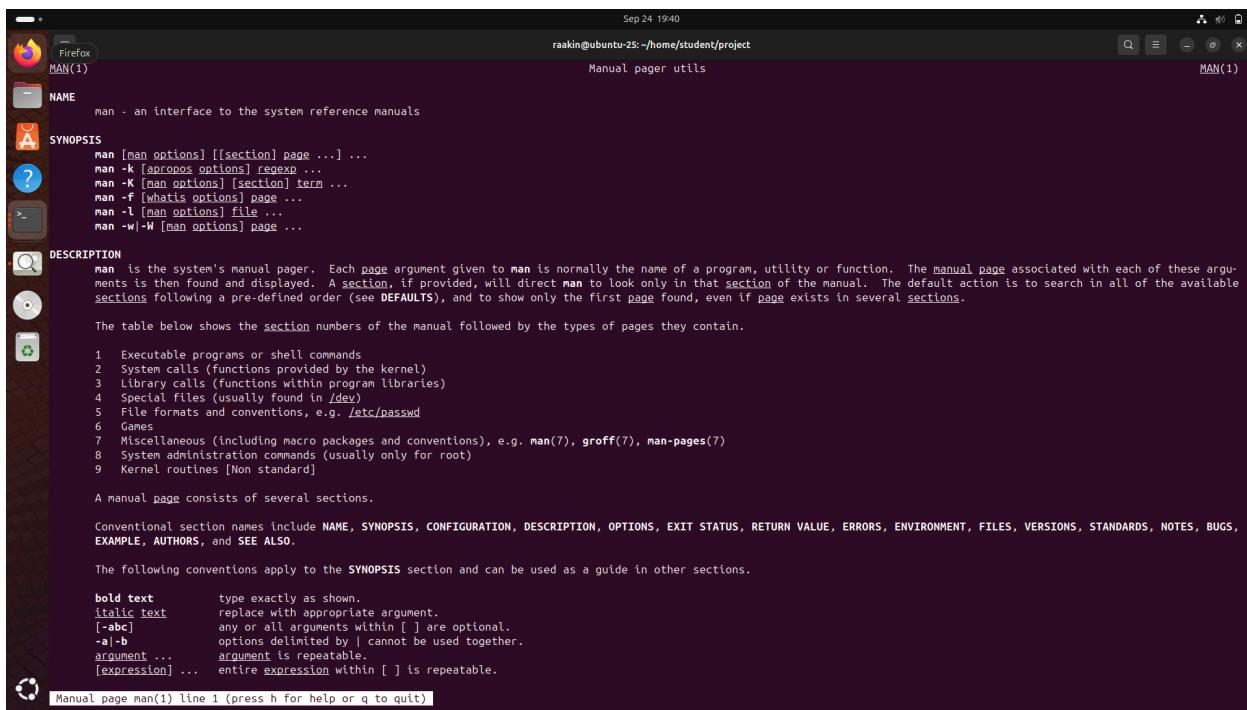
This sets `rw-` for the owner and `r--` for both group and others. As a result, anyone can open and read the file, but only the owner can modify it.

## 10Q)How do you check the manual page for any Linux commands? sol)

The `man` command is used to display the manual page of any Linux command.

This provides an overview of the `ls` command, including syntax, arguments and options, and examples of use. This is a full manual reference guide in sections (user commands, system calls, etc.). After you reference the manual, press `q` to quit.

Example:



Sep 24 19:40  
raakin@ubuntu-25:~/home/student/project  
Manual pager utils  
MAN(1)

**NAME**  
`man` - an interface to the system reference manuals

**SYNOPSIS**  
`man [man options] [[section] page ...] ...`  
`man -k [apropos options] regexp ...`  
`man -K [man options] [section] term ...`  
`man -f [whatis options] page ...`  
`man -l [man options] file ...`  
`man -W [man options] page ...`

**DESCRIPTION**  
`man` is the system's manual pager. Each page argument given to `man` is normally the name of a program, utility or function. The `manual page` associated with each of these arguments is then found and displayed. A `section`, if provided, will direct `man` to look only in that `section` of the manual. The default action is to search in all of the available `sections` following a pre-defined order (see `DEFAULTS`), and to show only the first `page` found, even if `page` exists in several `sections`.

The table below shows the `section` numbers of the manual followed by the types of pages they contain.

1	Executable programs or shell commands
2	System calls (functions provided by the kernel)
3	Library calls (functions within program libraries)
4	Special files (usually found in <code>/dev</code> )
5	File formats and conventions, e.g. <code>/etc/passwd</code>
6	Games
7	Miscellaneous (including macro packages and conventions), e.g. <code>man(7)</code> , <code>groff(7)</code> , <code>man-pages(7)</code>
8	System administration commands (usually only for root)
9	Kernel routines [Non standard]

A `manual page` consists of several sections.

Conventional section names include `NAME`, `SYNOPSIS`, `CONFIGURATION`, `DESCRIPTION`, `OPTIONS`, `EXIT STATUS`, `RETURN VALUE`, `ERRORS`, `ENVIRONMENT`, `FILES`, `VERSIONS`, `STANDARDS`, `NOTES`, `BUGS`, `EXAMPLE`, `AUTHORS`, and `SEE ALSO`.

The following conventions apply to the `SYNOPSIS` section and can be used as a guide in other sections.

<b>bold text</b>	type exactly as shown.
<b>italic text</b>	replace with appropriate argument.
<code>[abc]</code>	any or all arguments within [ ] are optional.
<code>-a -b</code>	options delimited by   cannot be used together.
<code>argument ...</code>	<code>argument</code> is repeatable.
<code>[expression] ...</code>	entire expression within [ ] is repeatable.

Manual page `man(1)` line 1 (press `h` for help or `q` to quit)