Linux Assignement-5

Name- Abinaya.P

Class/Section- 3-C

Usn- ENG24CY0075

Roll.no- 44

1) What is a shell in Linux OS? Categories and why Bash is popular

In Linux, the shell is basically the middleman between you and the system. Whenever we type something in the terminal, the shell takes our command, figures out what needs to be done, and tells the operating system to carry it out. Without a shell, we would have to deal directly with complex system calls, which is almost impossible for day-to-day use.

There are mainly two families of shells:

- **Bourne-like shells** (sh, bash, dash, ksh): These are scripting-friendly and widely used in automation.
- **C-like shells** (csh, tcsh): Their syntax looks a bit like C programming language.
  And then there are **modern shells** such as zsh and fish, which come with more interactive features.

Bash (Bourne Again Shell) is the most popular for a few reasons. First, most Linux distributions set it as the default for many years, so users and scripts everywhere assume bash features will be present. Second, it balances well between being POSIX-compatible (so scripts run everywhere) and giving extra features like command history, tab completion, brace expansion, and arrays.

Finally, it has a huge community, so if you ever get stuck, there's always help online.

2) What does ls -Z display?

Normally ls just lists files. When you add -Z, it shows the **security context** of each file if your system is using SELinux. The output includes labels that tell you what user, role, and type are associated with the file in terms of security policies. For example:

```
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 myfile.txt
```

Here, that long string after the owner and group is the SELinux context. This helps administrators control and audit which processes can access which files.

3) Command to list hidden files in the current directory

In Linux, files that start with a dot (.) are hidden. To list them, we can use:

```
ls -a
```

This shows **all** files, including hidden ones. But it also shows . (current directory) and .. (parent directory). If you only want the actual hidden files without those two, you could do:

```
ls -d .[^.]* ..?*
```

That way you get only the real hidden files.

4) Difference between hard links and soft (symbolic) links

A **hard link** is like giving another name to the same file. Both names point to the exact same data on disk (same inode). If you delete the original name, the

other still works, because the data isn't gone until all links are removed. The drawback is that hard links only work within the same filesystem and usually can't be created for directories.

A **soft link** (or symbolic link) is more like a shortcut. It just stores the path to another file. If the original file is removed or moved, the soft link breaks. The advantage is that soft links can cross filesystems and also point to directories.

Example:

```
ln file.txt file.hard
```

```
ln -s file.txt file.soft
```

5) Explain -rwxr-x--x

This string shows the permissions on a file. Let's break it:

- Owner: rwx → owner can read, write, and execute.
- Group: r-x → group can read and execute, but not write.
- Others: --x → everyone else can only execute, no read or write.

So if it's a script, the owner can open and edit it. Group members can run it and also read its contents. Everyone else can only run it without seeing what's inside.

6) Change group of data.txt to staff

The command is:

```
chgrp staff data.txt
```

Another way is:

```
chown :staff data.txt
```

Both achieve the same thing, which is changing the group ownership.

7) Why is 777 dangerous?

Permission 777 means everyone can read, write, and execute. That sounds convenient but is actually risky. If you give 777 to a script, **anyone** on the system can modify it. Imagine you have a script that runs automatically with admin rights. If someone edits it and inserts malicious code, the system could be compromised.

Example:

```
chmod 777 /var/www/html/index.php
```

Now every user can edit your web server's main file. An attacker could add backdoors, and you wouldn't even notice. That's why 777 should be avoided unless there is a very controlled use case (like temporary public directories with sticky bit set).

8) Difference between apropos and whatis

Both commands look into the manual page database but serve slightly different purposes:

- whatis gives a **one-line description** for an exact command. For example:

  ```
  whatis ls
  ```

- apropos searches for a **keyword** in man page descriptions. This is useful when you don't know the exact command name. For example:

  ```
  apropos calendar
  ```

So, think of whatis when you know the command, and apropos when you only know what you're looking for.

9) Redirect error output to error.log

Normally, the terminal shows two types of output: standard output (fd 1) and standard error (fd 2). To redirect just the errors to a file:

```
mycommand 2> error.log
```

This will keep the normal output on screen but send the error messages to error.log.

10) Append with tee instead of overwriting

The tee command is often used when you want to save output to a file but also see it on screen. By default, it overwrites the file. If you want to add new output at the end of the file instead, you use the -a flag:

```
mycommand | tee -a logfile.txt
```

So every time the command runs, the results just get appended to logfile.txt without wiping old content.