Assignment 5
Linux Programming

Name : Yathish N R

Class: 3B CY

Roll No: 66

USN : ENG25CY1007

1. **What is a shell in Linux OS? How many categories of shell is currently exists in**

Linux? Why bash shell is very popular in Linux distribution?
What is a Shell in Linux OS?

A shell is an **interface program** that acts as a bridge between the user and the Linux **kernel** (the core of the OS). It accepts human-readable commands (typed into a terminal or written in a script), interprets them, and passes them to the kernel for execution.

Categories of Shell

Most operating system shells fall into two primary categories:

- **Command-Line Interface (CLI) Shells:** These use text input (like commands) for user interaction. Examples include Bash, Zsh, and Ksh.
- **Graphical User Interface (GUI) Shells:** These use graphical elements (windows, icons, menus) for user interaction. Examples include GNOME Shell and KDE Plasma.

Why is Bash Shell Popular?

Bash (Bourne-Again SHell) is popular due to several key factors:

- **Default Shell:** It is the **default login shell** for the majority of modern Linux distributions (like Ubuntu, Fedora, and RHEL).
- **Compatibility:** It is a superset of the original Bourne Shell (sh), meaning most older shell scripts work in Bash without modification.
- **Features:** It offers enhanced interactive features (like command history, command-line editing, and tab completion) and advanced scripting capabilities (better control flow, functions, and arrays).
- **Free Software:** It was developed by the Free Software Foundation (FSF) as a free alternative to proprietary Unix shells, aligning with the core principles of Linux.

2. **What does the ls -Z command display?**

The ls -Z command displays the **Security Context** (or SELinux security labels) of files and directories alongside their names.

This label is crucial in systems using **SELinux** (**Security-Enhanced Linux**), which uses a mandatory access control (MAC) model to restrict access beyond standard user permissions.
The output typically looks like this:
unconfined_u:object_r:user_home_t:s0 file.txt
The security context informs the system about what processes or users are allowed to interact with the file.

3. **Write a command to list all hidden files in the current directory.**

In Linux, a hidden file is any file or directory whose name starts with a dot (.). The command to list all files, including hidden ones, is:
ls -a

4. **Explain the difference between hard links and soft links** (**symbolic links**) **in Linux**

The difference between links lies in **what they point to** and what happens when the original file is deleted.

Feature	Hard Link	Soft Link (Symbolic Link)
What it Points to	The **Inode** (the data structure containing file content and metadata).	The **Path/Filename** of the original file.
Data Integrity	If the original file is deleted, the data still exists and is accessible through the hard link until **all** hard links are removed.	If the original file is deleted or moved, the soft link becomes a **dangling link** (broken) and is useless.
File System/Volume	Must reside on the **same file system** and volume as the original file.	Can link to files or directories **across different file systems** or volumes.
Type	Acts as a second, identical name for the original file.	Acts as a **shortcut** or pointer to the file name.

5. **A file has permissions -rwxr-x—x. Explain who can read, write, and execute it.**

The file permission string -rwxr-x--x is broken down into three sets of access rights for three different entities:

Category	Permission String	Numeric Value	Interpretation (Read, Write, Execute)
Owner (**User**)	rwx	4+2+1=7	**Read, Write, and Execute** (Full Control)

Group	r-x	4+0+1=5	**Read and Execute** (Cannot modify/write)
Others (World)	--x	0+0+1=1	**Execute only** (Cannot read content or modify)

6. **Write the command to change the group ownership of a file data.txt to group staff**

The command used to change the group ownership of a file is chgrp. chgrp
staff data.txt

7. **Why is it dangerous to give 777 permissions to a file? Explain with an example.**

Giving a file **777** permissions (-rwxrwxrwx) means granting **Read, Write, and Execute** permissions to **EVERYONE** (Owner, Group, and Others).
Danger Explanation: When a file has 777 permissions, any user (including non-logged-in users via certain services) can **execute, read, or modify** the file.

- **Example:** If a script used by the web server (like update_db.sh) is set to 777, a malicious user could potentially **overwrite the script** with their own code. The web server would then execute the attacker's code, leading to system compromise, data theft, or corruption.
- **Principle:** This violates the **Principle of Least Privilege**, as it grants far more access than is necessary, creating a major security vulnerability.

8. **What is the difference between apropos (i.e., man -k) and whatis (i.e., man -f)?**

Both commands search the database of manual pages (man pages), but they differ in their search method:

Command	Equivalent	Search Method	Purpose
whatis	man -f	Performs an **exact match** on the command name only.	Provides a **quick, one-line description** for a command whose name you already know precisely.
apropos	man -k	Searches for the keyword **anywhere** within the command's name and its short description.	Finds **relevant commands** when you know what you want to do (e.g., "rename") but not the exact command name (mv).

9. **Write a command to redirect the error output of a command to a file named**

Error.log.

Linux, standard error (stderr) is stream number **2**. To redirect it to a file, you use the redirection operator (>) preceded by the stream number.

```
# General command structure:
<command> 2> error.log
```

Example:

```
cat /non/existent/file 2> error.log
```

This command attempts to read a file that doesn't exist. The error message is written to error.log and not displayed on the screen.

10. How can you use the tee command to append output to a file instead of Overwriting it?

The tee command reads from standard input and writes to both standard output (the screen) and one or more files.
To **append** the output instead of overwriting the file (which is the default behavior), you must use the **-a** (or --append) option.

```
# General command structure:
<command> | tee -a <filename>
```

Example:

```
# This command displays the date AND appends it to log.txt date | tee -a log.txt
```