1. What's the difference between ' ' and " " in shell?

Single Quotes ' '

- Literal quoting: everything inside is taken *as-is*.
- No variable, command, or escape expansion happens.

Example:

'Today is $HOME and date is $(date)'

Output:

Today is $HOME and date is $(date)

Double Quotes " "

- Weak quoting: protects most characters but still allows expansions.
- Things that still work inside " ":
  - **Variables** ($HOME, $USER)
  - **Command substitution** ($(date))
  - **Arithmetic expansion** ($((2+3)))

Example:

"Today is $HOME and date is $(date)"

Output:

Today is /home/ahmed and date is Tue Sep 2 23:05:12 EET 2025

2. Explain [-f filename] vs [-d dirname].

The expressions [-f filename] and [-d dirname] are used in shell scripting (like Bash) to test the type of a filesystem object. Both are conditional expressions that return a true (0) or false (1) exit status, and are most commonly used within an if statement.

[-f filename]

The -f flag tests whether the given filename exists and is a regular file.

- What it checks for: A regular file is a standard file on the filesystem that stores data, such as a text file, an image, or an executable program. It will return false for directories, symbolic links, pipes, or other special file types.

[-d dirname]

The -d flag tests whether the given dirname exists and is a directory.

- What it checks for: A directory is a special type of file that contains references to other files and directories. It will return false for regular files, symbolic links, or other special file types.

3. Explain stdout/stderr redirection, appending vs overwrite. How can you confirm redirection using file descriptors?

Redirection is a core feature of shell environments that changes where a command's output goes. In Linux and other Unix-like systems, every process has at least three standard I/O (input/output) streams, each associated with a file descriptor (a number).

- File descriptor 0: Standard Input (stdin)—The default source of input for a command, typically the keyboard.
- File descriptor 1: Standard Output (stdout)—The default stream for a command's normal output, typically the terminal screen.
- File descriptor 2: Standard Error (stderr)—The default stream for a command's error and diagnostic messages, also typically sent to the terminal screen.

stdout vs stderr redirection

Redirecting stdout captures the normal output of a command, while redirecting stderr captures error messages. This allows you to separate normal results from errors.

Overwriting with > and 2>

- Redirect stdout: The > operator sends a command's standard output to a file. If the file already exists, it is completely overwritten → command > file.txt
- Redirect stderr: The 2> operator specifically redirects standard error to a file. The 2 is the file descriptor for stderr → command 2> error.txt
- Redirect both to separate files → command > output.txt 2> error.txt
- Redirect both to the same file (classic method): By redirecting stderr to the same location as stdout, both streams can be consolidated. The order matters: stderr must be redirected after stdout has been set → command > combined.txt 2>&1

Appending with >> and 2>>

- Append stdout: The >> operator appends a command's standard output to the end of a file without overwriting its existing contents → command >> file.txt
- Append stderr: The 2>> operator appends standard error to the end of a file → command 2>> error.txt
- Append both to the same file → command >> combined.txt 2>&1
- Append both (modern Bash shorthand) → command &>> combined.txt

To confirm redirection using file descriptors

You can inspect a running process's file descriptors in the /proc filesystem. Each process has a directory, /proc/PID, where PID is the process ID. Inside this directory, the /proc/PID/fd subdirectory contains symbolic links to the files or devices a process has open.

1. Run a command and send it to the background to keep the parent shell free. For demonstration, we will redirect stdout to a file.
2. Find the PID of the background process using the $! shell variable, which holds the PID of the last background command.
3. Inspect the file descriptors of the process by listing the contents of /proc/$PID/fd/.
4. Confirm the content of the file. After the background process finishes, you can check that the output was indeed sent to the file.
5. Alternatively, check the process with lsof. The lsof (list open files) command can also show which files a process has open, including the file descriptors.

4. Show an example of a for loop in bash. Then, write a simple bash calculator that does add/subtract.