**1.Understanding Exit Status**:

very command returns an **exit status**:

- 0 means success
- Any **non-zero** value means failure

**Example:**

Bash:

```
ls /nonexistent_directory

echo "Exit status: $?"
```

-The ls command will fail because the directory does not exist.
- $? stores the exit status of the **last executed command**.

**2.Using if Statements for Error Checking**

**Example:**

Bash:

```
mkdir my_folder

if [ $? -eq 0 ]; then

   echo "Directory created successfully!"

else

   echo "Failed to create directory."

fi
```

-$? checks if mkdir my_folder succeeded.
- If **exit status = 0**, it prints success; otherwise, it prints an error message.

**3.Using trap for Cleanup:**

The trap command in Bash is used to **catch and handle signals** that a script receives during execution. It allows you to execute specific commands when a particular signal is detected, ensuring proper cleanup or custom behavior before the script exits.

**Common Signals in Bash**

| **Signal** | **Number** | **Description** |
|---|---|---|
| EXIT | 0 | Runs when the script exits (normally or forcefully) |

| Signal | Number | Description |
|---|---|---|
| SIGINT | 2 | Sent when you press Ctrl+C |
| SIGTERM | 15 | Sent when a process is requested to terminate (kill command) |
| SIGHUP | 1 | Sent when a terminal session ends (logout or disconnect) |
| SIGKILL | 9 | Forces a process to stop immediately (cannot be trapped) |

**Example: Cleaning Up a Temporary File**

```
#!/bin/bash

trap 'echo "Cleaning up..."; rm -f temp.txt; exit' SIGINT SIGTERM

touch temp.txt

echo "Temporary file created. Press Ctrl+C to stop."

while true; do sleep 1; done
```

**- Explanation:**

- trap catches SIGINT (Ctrl+C) and SIGTERM signals.
- When the script is interrupted, it:
    1. Prints "Cleaning up...".
    2. Deletes temp.txt.
    3. Exits gracefully.

**4.Redirecting Errors**

**Example:**

Bash:

```
ls /wrong_folder 2> errors.log
```

-2> redirects **error messages** (stderr) to errors.log.

To ignore errors:

bash

```
ls /wrong_folder 2> /dev/null
```

- 2> /dev/null **hides errors completely**.

**5.Creating Custom Error Messages**

**Example:**

Bash:

```
#!/bin/bash

if ! cd /wrong_directory; then

    echo "Error: Cannot change directory. Please check the path." >&2

    exit 1

fi
```

- If cd fails, a **custom error message** is printed to stderr (>&2).
- exit 1 ensures the script exits with an error.