**visualShell Scripting Questions for Practice**

**Beginner Level**

1. What is a shell script, and why is it used?

## **What is a Shell Script?**

A **shell script** is a text file that contains a sequence of **commands** written in a shell scripting language (such as **Bash**, **Zsh**, or **Sh**). These commands are executed **sequentially** by the shell interpreter.

### ****Why is Shell Scripting Used?****

Shell scripts **automate repetitive tasks** and **simplify system administration**. Instead of typing commands manually, you can write them in a script and run them **whenever needed**.

## **📌 Key Benefits of Shell Scripting**

| **Feature** | **Why It’s Useful** |
| --- | --- |
| **Automation** | Reduces manual effort by automating tasks like backups, software installation, and system monitoring. |
| **Efficiency** | Executes multiple commands in one go, saving time. |
| **Customization** | Helps create customized workflows and configurations. |
| **System Administration** | Used for managing users, permissions, and monitoring system health. |
| **Task Scheduling** | Can be used with cron jobs to run scripts at specific times. |
| **Error Handling** | Includes logic to handle failures and send alerts. |

## **📌 Real-World Example of a Shell Script**

### ****Example: Backup a Directory****

This script creates a backup of the /home/user/docs directory every time it runs.

#!/bin/bash

# Define variables

SOURCE\_DIR="/home/user/docs"

BACKUP\_DIR="/home/user/backup"

TIMESTAMP=$(date +"%Y-%m-%d\_%H-%M-%S")

BACKUP\_FILE="backup\_$TIMESTAMP.tar.gz"

# Create a backup

tar -czf $BACKUP\_DIR/$BACKUP\_FILE $SOURCE\_DIR

# Print success message

echo "Backup created: $BACKUP\_DIR/$BACKUP\_FILE"

### ****How It Works:****

1. **Defines variables** (SOURCE\_DIR, BACKUP\_DIR, TIMESTAMP).
2. **Uses tar command** to compress the directory.
3. **Stores backup with a timestamp**.
4. **Prints a success message**.

## **📌 How to Run a Shell Script**

1. **Create a script file**:

bash

nano myscript.sh

1. **Add commands inside the file** (starting with #!/bin/bash).
2. **Make the script executable**:

bash

chmod +x myscript.sh

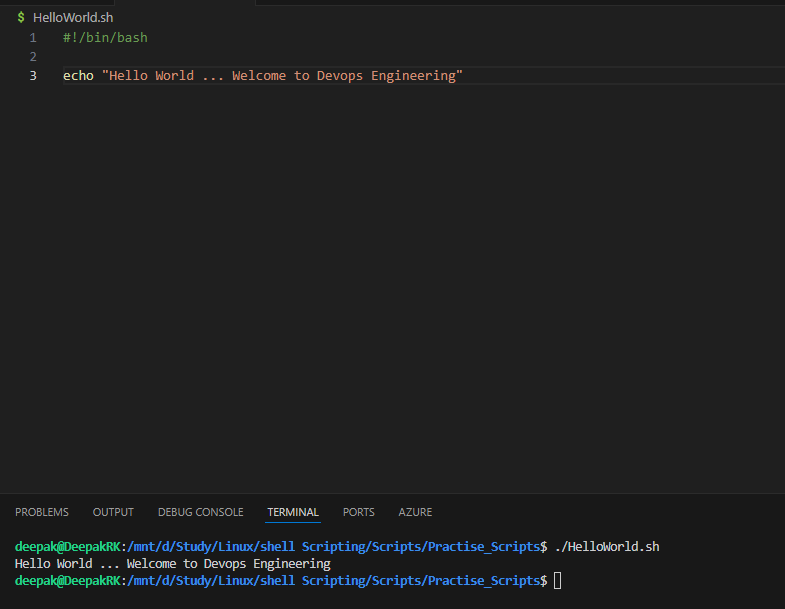
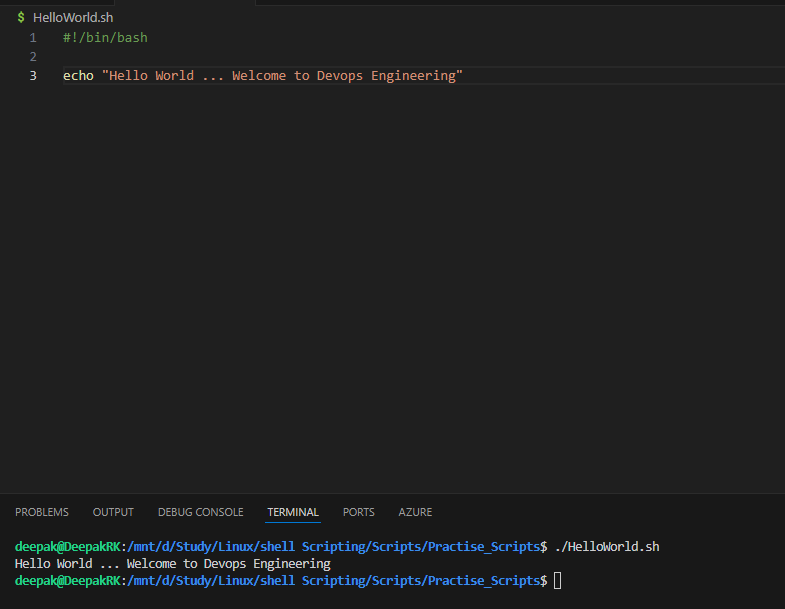
1. **Run the script**:

bash

./myscript.sh

## **📌 Common Use Cases for Shell Scripts**

* Automating **server setup** (installing packages, configuring services).
* **Managing users** (creating, deleting, modifying users).
* **Monitoring system performance** (checking CPU, memory usage).
* **Automated deployments** (pulling updates, restarting services).
* **Log file analysis** (filtering and summarizing logs).

1. Write a script to print "Hello, World!".  
     
     
   
2. How do you take user input in a shell script?

#!/bin/bash

echo "What is your name ?"

read name

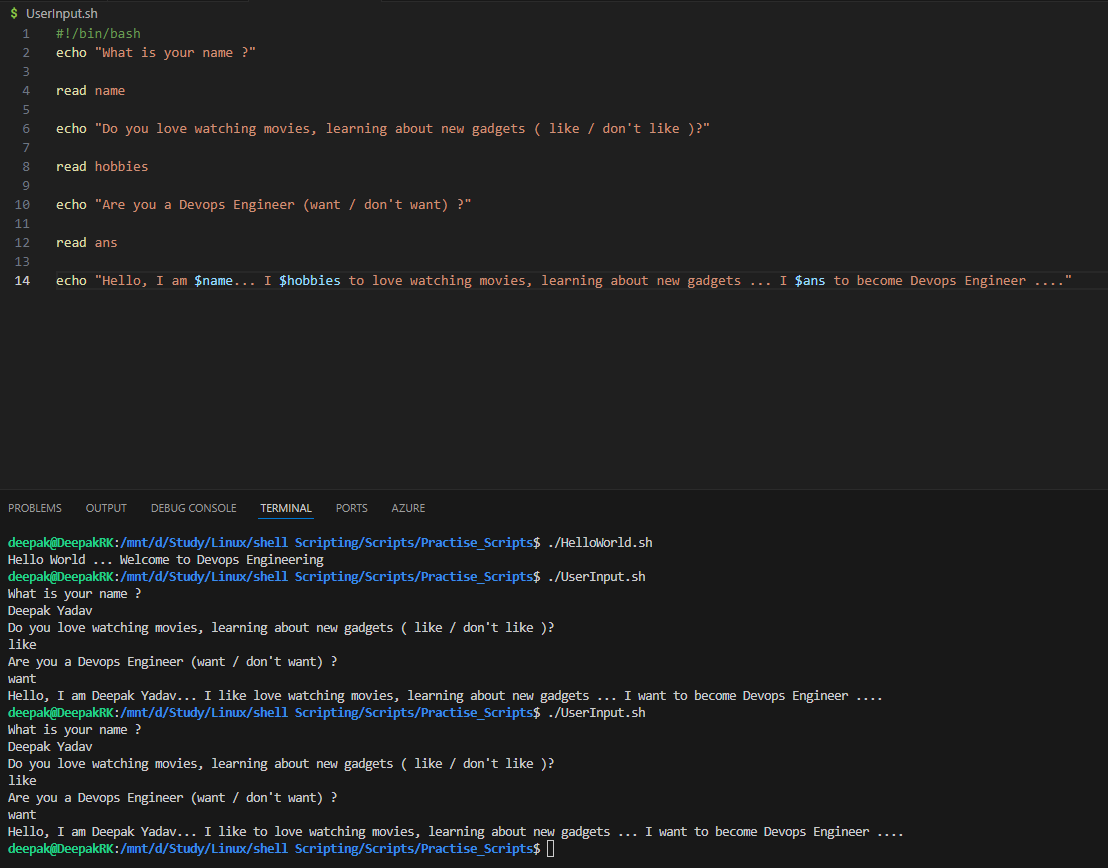
echo "Do you love watching movies, learning about new gadgets ( like / don't like )?"

read hobbies

echo "Are you a Devops Engineer (want / don't want) ?"

read ans

echo "Hello, I am $name... I $hobbies to love watching movies, learning about new gadgets ... I $ans to become Devops Engineer ...."



1. Write a script to display the current date and time.

#!/bin/bash

# Get the current date and time

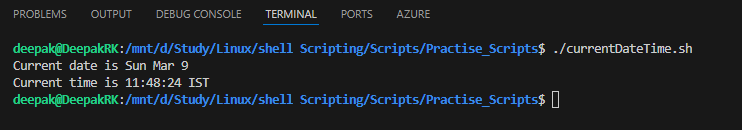
todayDate=$(date | awk '{print $1, $2, $3}')

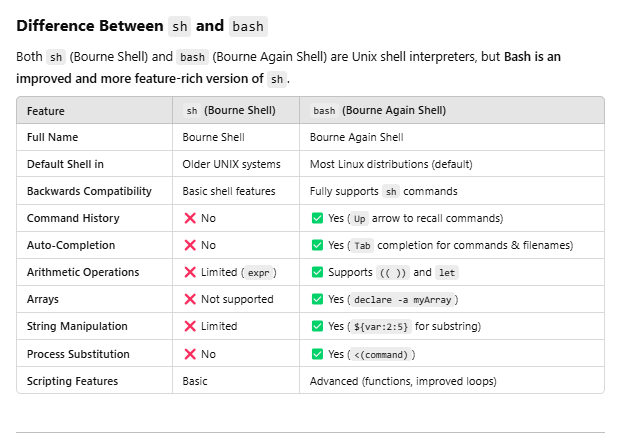
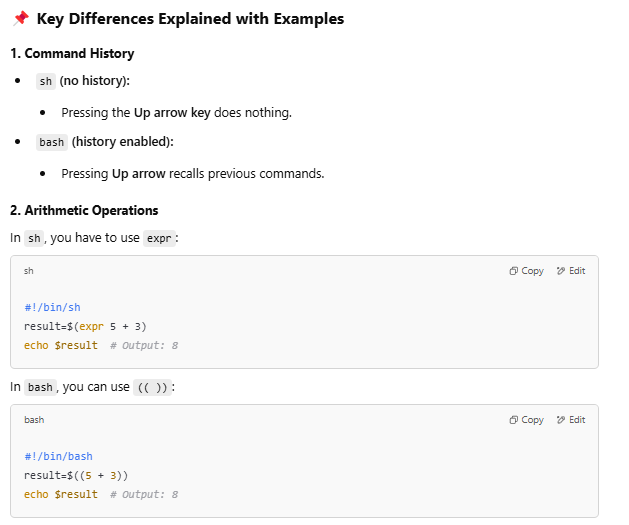
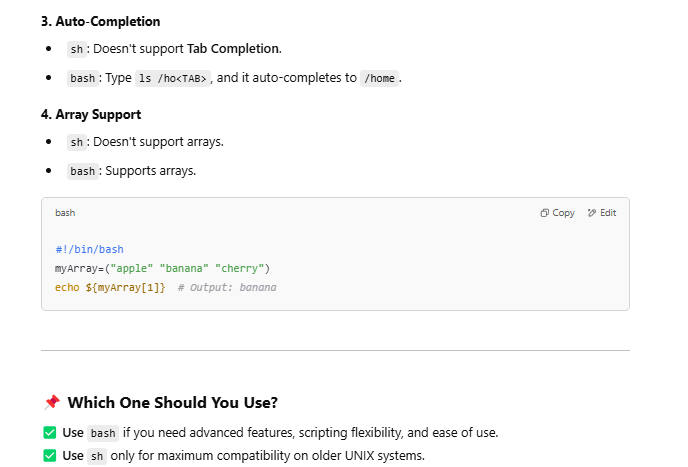
todayTime=$(date | awk '{print $4, $5}')

# Output the results

echo "Current date is $todayDate"

echo "Current time is $todayTime"



1. What is the difference between sh and bash?  
     
     
     
     
   
2. Write a script to check if a file exists in the current directory.

#!/bin/bash

FILE="/mnt/d/Study/Linux/shell Scripting/Scripts/Practise\_Scripts/HelloWorld.sh"

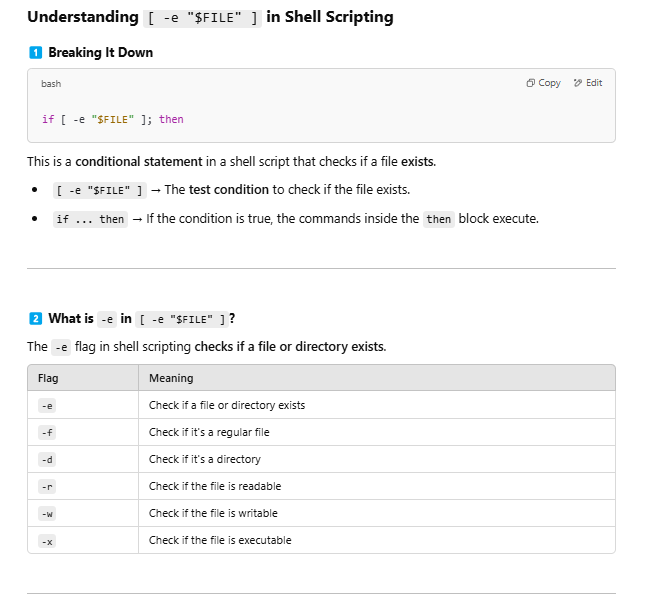
if [ -e "$FILE" ]; then

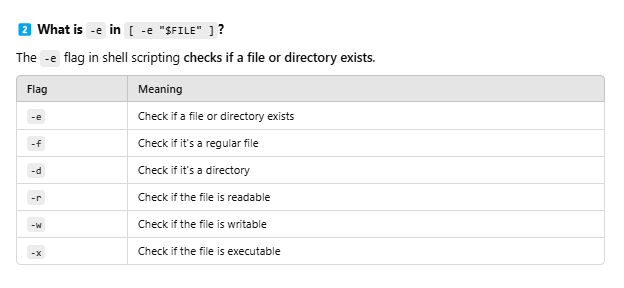
    echo "File exists."

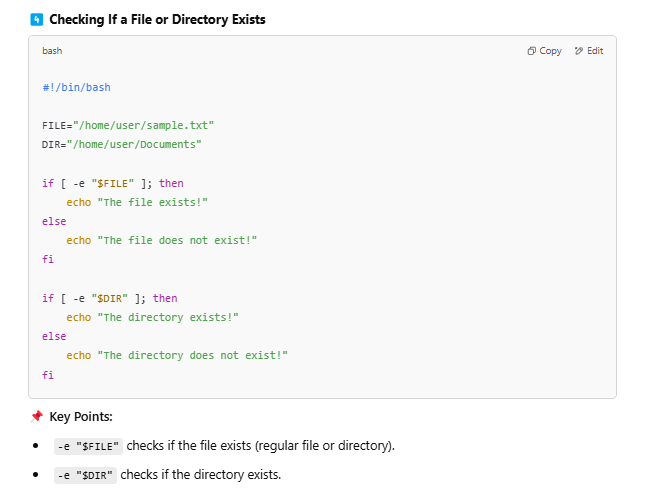
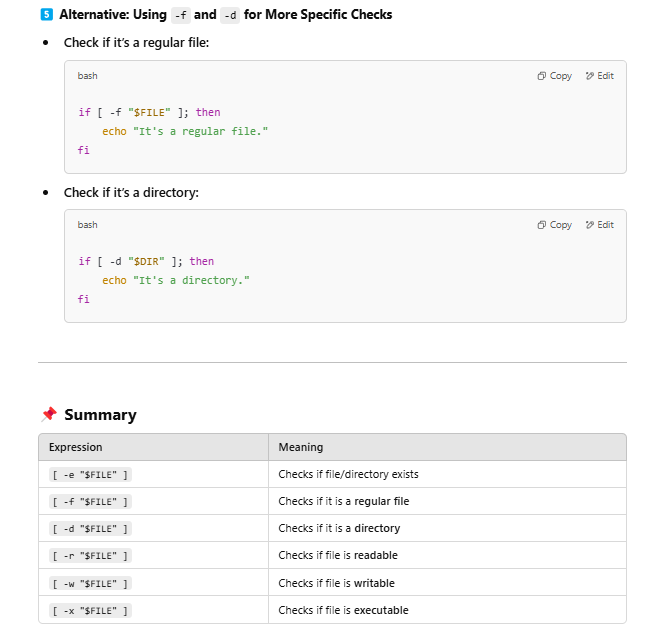
else

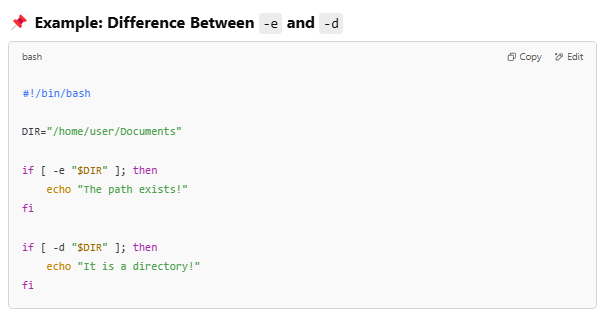
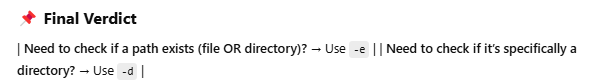
    echo "File does not exist."

fi





1. Explain the use of chmod in shell scripting.  
     
   Refer notes
2. Write a script to print numbers from 1 to 10 using a loop.

#!/bin/bash

for i in {1..100}

do

    echo "Number $i"

done

# -------------------------------------

for ((i=1;i<=20;i++))

    do

        if (( i % 2 == 0 )); then

            echo "Even Number -- $i"

        fi

    done

# -------------------------------------

for ((i=1; i<=20; i++))

do

    if [ $((i % 2)) -eq 0 ]; then

        echo "Even Number -- $i"

    fi

done

### for ****Loop in Bash Scripting – Use Cases & Examples****

A **for loop** in Bash allows you to **iterate over a list of values or a sequence of numbers**. It is commonly used for **automation, file processing, user management, and system monitoring**.

## **1️⃣ Basic** for **Loop Syntax**

for variable in list

do

command

done

✅ The loop **iterates through each item in list** and executes the commands inside the do ... done block.

# **📌 Use Cases of** for **Loop in Bash**

## **1. Iterating Over a List of Words**

🔹 **Use Case:** Print a list of fruit names.

#!/bin/bash

for fruit in apple banana mango orange

do

echo "I like $fruit"

done

📌 **Output:**

I like apple

I like banana

I like mango

I like orange

## **2. Looping Through a Range of Numbers**

🔹 **Use Case:** Print numbers from 1 to 5.

#!/bin/bash

for i in {1..5}

do

echo "Number: $i"

done

📌 **Output:**

Number: 1

Number: 2

Number: 3

Number: 4

Number: 5

## **3. Using C-Style** for **Loop**

🔹 **Use Case:** Print even numbers from 2 to 10.

bash

#!/bin/bash

for ((i=2; i<=10; i+=2))

do

echo "Even number: $i"

done

📌 **Output:**

yaml

Even number: 2

Even number: 4

Even number: 6

Even number: 8

Even number: 10

## **4. Looping Through Files in a Directory**

🔹 **Use Case:** List all .txt files in a directory.

bash

#!/bin/bash

for file in /home/user/documents/\*.txt

do

echo "Found file: $file"

done

📌 **Output:**

swift

Found file: /home/user/documents/file1.txt

Found file: /home/user/documents/file2.txt

## **5. Renaming Multiple Files**

🔹 **Use Case:** Add a .backup extension to all .log files.

bash

#!/bin/bash

for file in \*.log

do

mv "$file" "$file.backup"

echo "Renamed: $file → $file.backup"

done

📌 **Before:**

lua

error.log, access.log

📌 **After:**

lua

error.log.backup, access.log.backup

## **6. Looping Over Command Output**

🔹 **Use Case:** Get a list of running processes for a specific user.

bash

#!/bin/bash

for process in $(ps -u $USER -o comm=)

do

echo "Running process: $process"

done

📌 **Output:**

arduino

Running process: bash

Running process: firefox

Running process: code

## **7. Checking Disk Space for Multiple Directories**

🔹 **Use Case:** Monitor disk usage of multiple directories.

bash

#!/bin/bash

for dir in /home /var /etc

do

du -sh "$dir"

done

📌 **Output:**

bash

2.3G /home

1.2G /var

512M /etc

## **8. Pinging Multiple Servers**

🔹 **Use Case:** Check connectivity to multiple servers.

bash

#!/bin/bash

for server in google.com github.com amazon.com

do

ping -c 1 "$server" &> /dev/null && echo "$server is reachable" || echo "$server is down"

done

📌 **Output:**

csharp

google.com is reachable

github.com is reachable

amazon.com is down

## **9. Creating Multiple Users**

🔹 **Use Case:** Add multiple Linux users.

bash

#!/bin/bash

for user in alice bob charlie

do

sudo useradd "$user"

echo "User $user created"

done

📌 **Output:**

sql

User alice created

User bob created

User charlie created

## **10. Automating Software Installation**

🔹 **Use Case:** Install multiple packages.

bash

#!/bin/bash

for package in nginx mysql-server git

do

sudo apt install -y "$package"

echo "$package installed"

done

📌 **Output:**

nginx

nginx installed

mysql-server installed

git installed

## **📌 Summary Table:** for **Loop Use Cases**

| **Use Case** | **Example Code** |
| --- | --- |
| **Looping through words** | for fruit in apple banana mango; do echo $fruit; done |
| **Looping through numbers** | for i in {1..5}; do echo $i; done |
| **C-style for loop** | for ((i=1; i<=5; i++)); do echo $i; done |
| **Looping through files** | for file in \*.txt; do echo $file; done |
| **Renaming files** | for file in \*.log; do mv "$file" "$file.bak"; done |
| **Looping over command output** | for user in $(cat users.txt); do echo $user; done |
| **Checking disk space** | for dir in /home /var; do du -sh "$dir"; done |
| **Pinging multiple servers** | for site in google.com yahoo.com; do ping -c 1 "$site"; done |
| **Creating multiple users** | for user in alice bob; do sudo useradd "$user"; done |
| **Installing software** | for pkg in nginx git; do sudo apt install -y "$pkg"; done |

## **Conclusion**

* for loops are **powerful** and help automate **repetitive tasks**.
* Use them for **iterating over files, numbers, command outputs, and more**.
* You can combine for loops with **conditionals (if), functions, and user input** for advanced scripting.

### ****When to Use**** [ ]****,**** [[ ]]****, and**** (( )) ****in Bash?****

In Bash scripting, **square brackets [ ], double square brackets [[ ]], and double parentheses (( ))** serve different purposes.

## **📌 1. Single Square Brackets** [ ] **(POSIX** test **Command)**

🔹 Used for **string comparisons, file checks, and simple arithmetic**.  
🔹 Works in **all shells** (POSIX-compliant).  
🔹 **Does not support &&, ||, or < > without escape characters.**

### ✅ ****Example: Checking If a File Exists****

bash

if [ -f "/etc/passwd" ]; then

echo "File exists"

fi

### ✅ ****Example: String Comparison****

bash

if [ "$name" = "Deepak" ]; then

echo "Hello, Deepak!"

fi

🔹 **Note:** Must use = instead of ==.

### ✅ ****Example: Arithmetic Comparison****

bash

if [ $((10 % 2)) -eq 0 ]; then

echo "10 is even"

fi

🔹 **Note:** -eq, -ne, -lt, -gt, -le, -ge are required for numbers.

## **📌 2. Double Square Brackets** [[ ]] **(Bash-Only Enhanced Test)**

🔹 **More advanced version** of [ ], available **only in Bash**.  
🔹 Supports **logical operators (&&, ||) without escaping**.  
🔹 Supports **regex matching (=~)**.  
🔹 Safer for string comparisons.

### ✅ ****Example: String Comparison (With**** ==****)****

if [[ "$name" == "Deepak" ]]; then

echo "Hello, Deepak!"

fi

🔹 **Note:** == works in [[ ]], but not in [ ].

### ✅ ****Example: Using**** && ****and**** || ****Without Escaping****

bash

if [[ -f "/etc/passwd" && -r "/etc/passwd" ]]; then

echo "File exists and is readable"

fi

🔹 **Note:** This would **fail** with [ ] unless you escape && as \[ -f ... \] && \[ -r ... \].

### ✅ ****Example: Regex Matching****

bash

if [[ "hello123" =~ ^hello[0-9]+$ ]]; then

echo "Matches regex!"

fi

🔹 **Note:** Regex does **not work** with [ ].

## **📌 3. Double Parentheses** (( )) **(Arithmetic Evaluation)**

🔹 Used **only for numbers** (arithmetic operations).  
🔹 **Faster** and **simpler** for math operations.  
🔹 **Does not require -eq, -lt, etc.**  
🔹 Supports **increment ++ and --**.

### ✅ ****Example: Arithmetic Condition****

bash

if (( 10 > 5 )); then

echo "10 is greater than 5"

fi

🔹 **Note:** No need for -gt, just > works.

### ✅ ****Example: Looping with**** (( ))

bash

for ((i=1; i<=5; i++))

do

echo "Count: $i"

done

### ✅ ****Example: Incrementing a Variable****

bash

count=5

(( count++ ))

echo $count # Output: 6

## **📌 Summary Table**

| **Bracket Type** | **Usage** | **Supports Logical Operators?** | **Supports Regex?** | **Best For** |
| --- | --- | --- | --- | --- |
| [ ] (Single) | **POSIX-compliant test** | ❌ No (&& and ` |  | ` require escaping) |
| [[ ]] (Double) | **Bash-only improved test** | ✅ Yes | ✅ Yes (=~ operator) | String comparisons, regex matching |
| (( )) (Double Parentheses) | **Arithmetic operations** | ✅ Yes (&&, ` |  | `) |

### ****📌 When to Use Which One?****

✅ **Use [ ]** → When writing scripts that **must work in all Unix shells (POSIX-compliant)**.  
✅ **Use [[ ]]** → When writing **Bash-specific scripts** that involve **string comparisons or regex**.  
✅ **Use (( ))** → When performing **mathematical operations** (arithmetic calculations).

### ****📌 Bash Script Using**** [ ]****,**** [[ ]]****, and**** (( )) ****Together****

This script:  
✅ Checks if a file exists ([ ])  
✅ Validates user input using regex ([[ ]])  
✅ Performs arithmetic calculations ((( )))

### ****🔹 Example Script: File Check, String Validation, and Math Operations****

bash

#!/bin/bash

# Define variables

FILE="/etc/passwd"

USER\_INPUT="Deepak123"

NUM1=10

NUM2=5

### ✅ 1. Using [ ] for File Check (POSIX-compliant)

if [ -f "$FILE" ]; then

echo "[ ] File '$FILE' exists!"

else

echo "[ ] File '$FILE' does not exist!"

fi

### ✅ 2. Using [[ ]] for String Comparison & Regex Matching

if [[ "$USER\_INPUT" =~ ^[A-Za-z]+[0-9]+$ ]]; then

echo "[[ ]] User input '$USER\_INPUT' is valid (contains letters followed by numbers)!"

else

echo "[[ ]] Invalid user input!"

fi

### ✅ 3. Using (( )) for Arithmetic Operations

if (( NUM1 > NUM2 )); then

echo "(( )) $NUM1 is greater than $NUM2"

fi

# Incrementing a variable using (( ))

(( NUM1++ ))

echo "(( )) NUM1 after increment: $NUM1"

# Using (( )) in a loop

echo "(( )) Printing even numbers from 2 to 10:"

for ((i=2; i<=10; i+=2)); do

echo "$i"

done

### ****📌 Explanation****

1️⃣ **[ -f "$FILE" ]** → Checks if the file /etc/passwd exists (**POSIX-compliant**).  
2️⃣ **[[ "$USER\_INPUT" =~ ^[A-Za-z]+[0-9]+$ ]]** → Uses regex to check if input has **letters followed by numbers**.  
3️⃣ **(( NUM1 > NUM2 ))** → Directly compares two numbers (**no need for -gt**).  
4️⃣ **(( NUM1++ ))** → Increments NUM1 using (( )).  
5️⃣ **Loop using (( ))** → Prints even numbers from 2 to 10.

### ****📌 Sample Output****

pgsql

[ ] File '/etc/passwd' exists!

[[ ]] User input 'Deepak123' is valid (contains letters followed by numbers)!

(( )) 10 is greater than 5

(( )) NUM1 after increment: 11

(( )) Printing even numbers from 2 to 10:

2

4

6

8

10

### ****📌 Summary****

| **Feature** | **Syntax Used** | **Purpose** |
| --- | --- | --- |
| **File Existence Check** | [ -f "$FILE" ] | Checks if file exists |
| **String Comparison & Regex** | [[ "$USER\_INPUT" =~ regex ]] | Validates user input format |
| **Number Comparison** | (( NUM1 > NUM2 )) | Compares two numbers |
| **Variable Increment** | (( NUM1++ )) | Increments a variable |
| **Looping with Numbers** | for ((i=2; i<=10; i+=2)) | Prints even numbers |

### ****🚀 Key Takeaways****

✅ Use [ ] for **file checks and basic conditions**.  
✅ Use [[ ]] for **advanced string operations and regex matching**.  
✅ Use (( )) for **math calculations and loops**.

1. How do you comment lines in a shell script?  
     
   refer notes   
     
   # --SLC  
   /\* -----------  
     
    -------------- \*/ --🡪 MLC
2. Write a script to check if a number is even or odd.

#!/bin/bash

echo "Give the number which you want to check is even or odd ?"

read n

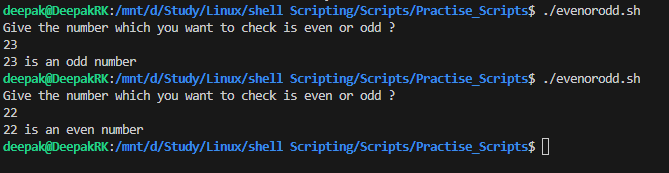
if (( $n % 2 == 0 )); then

    echo "$n is an even number"

else

    echo "$n is an odd number"

fi



**Intermediate Level**

Write a script to find the factorial of a number.

#!/bin/bash

a=0

b=1

echo "Enter the number n :"

read n

echo -n "Fibonacci series of $n is : $a $b "

for(( i=2;i<=n;i++ ))

do

    sum=$(( a+b ))

    echo -n "$sum "

    a=$b

    b=$sum

done

echo ''

The -n option in echo prevents a **newline** from being added at the end of the output.

**Example Without -n (Default Behavior)**

echo "Hello"

echo "World"

**Output:**

Hello

World

Each echo command prints a newline (\n) automatically.

**Example With -n**

echo -n "Hello "

echo "World"

**Output:**

Hello World

🔹 **Explanation:**

* The first echo -n "Hello " prints "Hello " **without a newline**.
* The second echo "World" prints "World" on the same line.

**When to Use -n?**

✅ To print output on the same line.  
✅ Useful when printing dynamic content without unwanted line breaks.

1. Create a script to count the number of files in a directory.

#!/bin/bash

# Ask the user for the directory path

echo "Enter the path of the directory where files need to be counted:"

read dirpath

# Check if the directory exists

if [ -d "$dirpath" ]; then

    file\_count=$(ls "$dirpath" | wc -l)

    echo "Number of files in the directory: $file\_count"

else

    echo "Error: Directory does not exist!"

fi

# ---------------------------------------------------------------

# Ask the user for the directory path

echo "Enter the path of the directory where files need to be counted:"

read dirpath

# Check if the directory exists

if [ -d "$dirpath" ]; then

    file\_count=$(find "$dirpath" -maxdepth 1 -type f | wc -l)

    echo "Number of files in the directory: $file\_count"

else

    echo "Error: Directory does not exist!"

fi

### 📌 Explanation of Fixes

✅ **find "$dirpath" -maxdepth 1 -type f**

* find is used to search for **files only (-type f)**.
* -maxdepth 1 ensures **only the top-level files are counted** (ignores subdirectories).
* wc -l counts the number of found files.

✅ **Ignores Directories**

* Unlike ls, which lists everything, find ... -type f only considers files.

1. Write a script to display the first 10 lines of a file.

#!/bin/bash

echo "Enter the path of the file :"

read filename

head -10 $filename

#!/bin/bash

echo "Enter the filename : "

read filename

filepath=$( sudo find /opt -type f -name "$filename" 2>/dev/null | head -n 2 )

if [[ -n $filepath ]]; then

    echo "File found: $filepath"

    echo "First 10 lines of $filename"

    head -n 10 $filename

else

    echo "File not found"

fi

1. How do you schedule a shell script to run every day at a specific time?  
     
   Cronjob
2. Write a script to copy all .txt files from one directory to another.  
     
   cp \*.txt /path/to/another/dir

#!/bin/bash

# Prompt for source directory

echo "Enter the source directory:"

read src

# Prompt for destination directory

echo "Enter the destination directory:"

read dest

# Check if source directory exists

if [[ ! -d "$src" ]]; then

    echo "Error: Source directory does not exist!"

    exit 1

fi

# Check if destination directory exists, if not, create it

if [[ ! -d "$dest" ]]; then

    echo "Destination directory does not exist. Creating it..."

    mkdir -p "$dest"

fi

# Copy all .txt files from source to destination

cp -v "$src"/\*.txt "$dest" 2>/dev/null

# Check if any .txt files were copied

if [[ $? -eq 0 ]]; then

    echo "All .txt files have been copied successfully from $src to $dest."

else

    echo "No .txt files found in $src or an error occurred."

fi

### Explanation in Shell Scripting

$? is a special variable that **stores the exit status** of the last executed command.

* If $? -eq 0 → ✅ The last command **ran successfully**
* If $? -ne 0 → ❌ The last command **failed**

1. **Explain the difference between $@ and $\* in shell scripting.**

Both $@ and $\* are used to **represent all positional parameters (arguments)** passed to a shell script **or function**. However, they behave differently when quoted.

## **1️⃣** $\* **– Expands as a Single String**

* Treats all arguments as **one single string** when quoted ("$\*").
* The arguments are joined into a **single string** using **the first character of $IFS (default is space)**.

### ****Example:****

bash

CopyEdit

#!/bin/bash

echo "Using \$\*:"

for arg in "$\*"; do

echo "$arg"

done

#### ****Run the script with multiple arguments:****

sh

CopyEdit

./script.sh apple banana "cherry pie"

#### ****Output:****

makefile

CopyEdit

Using $\*:

apple banana cherry pie

💡 **Notice:** Everything is treated as a **single string**.

## **2️⃣** $@ **– Expands as Separate Strings**

* Treats each argument **individually**, preserving quotes when quoted ("$@").
* Each argument remains **separate** in loops.

### ****Example:****

bash

CopyEdit

#!/bin/bash

echo "Using \$@:"

for arg in "$@"; do

echo "$arg"

done

#### ****Run the script with multiple arguments:****

sh

CopyEdit

./script.sh apple banana "cherry pie"

#### ****Output:****

bash

CopyEdit

Using $@:

apple

banana

cherry pie

💡 **Notice:** "cherry pie" is correctly preserved as one argument.

## **📌 Key Differences**

| **Feature** | **$\* (quoted)** | **$@ (quoted)** |
| --- | --- | --- |
| Expansion | Expands into **one single string** | Expands into **separate arguments** |
| Loop Behavior | Iterates **once** over the full string | Iterates **individually** for each argument |
| Preserves Quotes | ❌ No (arguments get merged) | ✅ Yes (arguments stay separate) |
| Separator | Uses $IFS (default: space) | Keeps original argument structure |

1. Write a script to check if a directory exists. If not, create it.

if [[ -d “$dirpath” ]];

Write a script to find the largest of three numbers entered by the user.

#!/bin/bash

echo "Enter the path to be checked"

read path

if [[ -d "$path" ]]; then

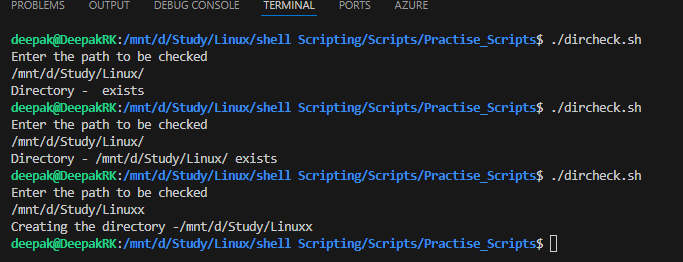
     echo "Directory - $path exists"

else

    echo "Creating the directory -$path"

    mkdir -p $path

fi



1. Write a script to reverse a string entered by the user.

#!/bin/bash

# Function to reverse a string

reverse\_string() {

    local str="$1"

    echo "$str" | rev

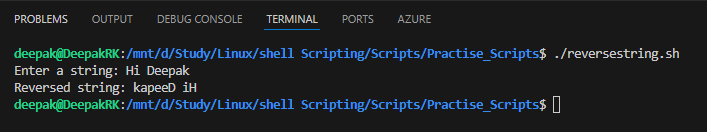
}

# Get user input

read -p "Enter a string: " user\_input

reversed\_string=$(reverse\_string "$user\_input")

echo "Reversed string: $reversed\_string"



1. How would you handle errors in a shell script?

Handling errors in a shell script is crucial for ensuring that your script behaves predictably and can recover from unexpected situations. Here are several strategies to effectively manage errors in shell scripts:

### 1. ****Using Exit Status****

Every command in a shell script returns an exit status (also known as return code). A status of 0 indicates success, while any non-zero value indicates an error. You can check the exit status of commands using the special variable $?.

bash

**#!/bin/bash**

*# Example command*

cp source.txt destination.txt

*# Check if the command was successful*

if [ $? -ne 0 ]; then

echo "Error: Failed to copy file."

exit 1

fi

### 2. ****Using**** set -e

You can use set -e at the beginning of your script to make the script exit immediately if any command returns a non-zero status. This is useful for catching errors early.

bash

**#!/bin/bash**

set -e

*# This command will cause the script to exit if it fails*

cp source.txt destination.txt

echo "File copied successfully."

### 3. ****Using**** trap

The trap command allows you to specify commands to be executed when the script receives certain signals or exits. This can be useful for cleanup tasks or logging errors.

bash

**#!/bin/bash**

*# Function to handle errors*

error\_handler() {

echo "An error occurred in the script."

exit 1

}

*# Trap errors and call the error\_handler function*

trap 'error\_handler' ERR

*# Example command*

cp source.txt destination.txt

### echo "File copied successfully." ****🛠️ What This Script Does?****

1. **Defines an error\_handler function** that prints an error message and exits.
2. **Uses trap 'error\_handler' ERR** → If any command fails, it calls error\_handler.
3. **Copies source.txt to destination.txt**.
4. **Prints "File copied successfully."** only if cp succeeds.

## **🔍 Example 1: Successful Execution**

### ****✅ Scenario:**** source.txt ****exists****

bash

CopyEdit

touch source.txt # Creating the source file

bash script.sh

🔹 **Output:**

arduino

CopyEdit

File copied successfully.

🚀 The script runs smoothly since cp succeeds.

## **❌ Example 2: Handling Errors (File Not Found)**

### ****🛑 Scenario:**** source.txt ****does NOT exist****

bash

CopyEdit

rm source.txt # Ensure the file doesn't exist

bash script.sh

🔹 **Output:**

nginx

CopyEdit

An error occurred in the script.

🚨 Since cp source.txt destination.txt fails, the trap calls error\_handler(), which prints an error message and exits.

## **🔄 Example 3: Enhanced Error Handling (With Line Numbers)**

To **debug errors easily**, modify the script to **show the exact line number** where the error occurred.

### ****📝 Improved Script****

bash

CopyEdit

#!/bin/bash

# Enhanced error handler with line number

error\_handler() {

echo "An error occurred on line $1."

exit 1

}

# Trap errors and pass the line number

trap 'error\_handler $LINENO' ERR

# Example command

cp source.txt destination.txt

echo "File copied successfully."

### ****🔍 Example Output When Error Occurs****

bash

CopyEdit

An error occurred on line 11.

🔥 **Now you know exactly where the script failed!**

## **✅ Best Practices for Bash Script Error Handling**

1. **Use trap to handle errors and cleanup**:

bash

CopyEdit

trap 'echo "Something went wrong!"' ERR

1. **Show line numbers in error messages**:

bash

CopyEdit

trap 'echo "Error at line $LINENO"' ERR

1. **Use set -e to exit immediately if any command fails**:

bash

CopyEdit

set -e # Stops execution on error

1. **Log errors to a file for debugging**:

bash

CopyEdit

error\_handler() {

echo "An error occurred on line $1" | tee error.log

}

trap 'error\_handler $LINENO' ERR

## **🎯 Summary**

| **Feature** | **Basic Script** | **Improved Script** |
| --- | --- | --- |
| Handles Errors? | ✅ Yes | ✅ Yes |
| Shows Line Number? | ❌ No | ✅ Yes |
| Logs Errors? | ❌ No | ✅ Yes |
| Stops on Error? | ✅ Yes | ✅ Yes |

Would you like help with adding logging or email alerts on errors? 🚀🔥  
  
In your Bash script, ERR is a **special signal** used with the trap command.

## **🔹 What is** ERR **in Bash?**

* ERR is a **trap signal** that triggers when **any command fails** (i.e., returns a **non-zero exit status**).
* trap 'error\_handler' ERR tells the script:  
  📌 **"If any command fails, run error\_handler."**

### 4. ****Using Conditional Statements****

You can use conditional statements to handle errors gracefully and provide meaningful messages to the user.

bash

**#!/bin/bash**

*# Example command*

if cp source.txt destination.txt; then

echo "File copied successfully."

else

echo "Error: Failed to copy file."

exit 1

fi

### 5. ****Logging Errors****

You can redirect error messages to a log file for later review. This is especially useful for long-running scripts.

bash

**#!/bin/bash**

LOGFILE="script.log"

*# Example command with error logging*

if ! cp source.txt destination.txt; then

echo "Error: Failed to copy file." >> "$LOGFILE"

exit 1

fi

### 6. ****Using Functions for Error Handling****

You can create functions to encapsulate error handling logic, making your script cleaner and more modular.

bash

**#!/bin/bash**

log\_error() {

echo "Error: $1" >> script.log

}

copy\_file() {

cp "$1" "$2" || log\_error "Failed to copy $1 to $2"

}

*# Example usage*

copy\_file source.txt destination.txt

In shell scripting, $? is a special variable that holds the exit status of the last command executed. The exit status is a numeric value returned by a command to indicate whether it was successful or if an error occurred. Here’s a breakdown of what $? signifies and a list of other special variables and commands commonly used in shell scripting.

**$? - Exit Status**

* **Significance**: The value of $? is 0 if the last command was successful (i.e., it completed without errors). A non-zero value indicates that an error occurred. The specific non-zero value can provide information about the type of error.

**Example**:

bash

Copy Code

ls /nonexistent\_directory

echo $? *# This will output a non-zero value indicating an error*

**Common Special Variables in Shell Scripting**

Here’s a list of commonly used special variables and commands in shell scripting:

1. **$?**: Exit status of the last command executed.
2. **$0**: The name of the script itself.
3. **$1, $2, ..., $N**: Positional parameters representing the arguments passed to the script. $1 is the first argument, $2 is the second, and so on.
4. **$#**: The number of positional parameters (arguments) passed to the script.
5. **$@**: All positional parameters as separate words. It treats each argument as a separate entity.
6. **$\***: All positional parameters as a single word. It treats all arguments as a single string.
7. **$**: The process ID (PID) of the current shell or script.
8. **$!**: The process ID of the last background command executed.
9. **$-**: The current options set for the shell.
10. **$?**: Exit status of the last command executed (as mentioned above).
11. **$IFS**: Internal Field Separator, which defines how the shell recognizes word boundaries (default is space, tab, and newline).
12. **$RANDOM**: Returns a random number between 0 and 32767.
13. **$UID**: The user ID of the current user.
14. **$HOME**: The home directory of the current user.
15. **$PATH**: The directories the shell searches for commands.

**Example Usage**

Here’s a simple script that demonstrates the use of some of these special variables:

bash

Copy Code

**#!/bin/bash**

echo "Script name: $0"

echo "First argument: $1"

echo "Number of arguments: $#"

echo "All arguments: $@"

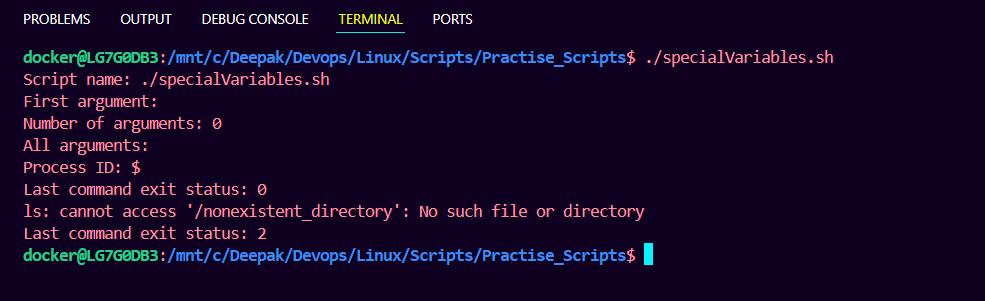
echo "Process ID: $"

echo "Last command exit status: $?"

*# Example command*

ls /nonexistent\_directory

echo "Last command exit status: $?"



**Advanced Level**

1. Write a script to monitor disk usage and send an alert if usage exceeds 80%.

### ****📧 Setting Up Mail on Ubuntu to Send Email Alerts****

To send email alerts from your Ubuntu server using **Bash scripts** (like the **disk usage alert** script), you need to set up an **SMTP mail service**.

## **🔹 Step 1: Install** mailutils **(for Sending Emails)**

bash

CopyEdit

sudo apt update && sudo apt install -y mailutils

📌 **Why?**

* mailutils allows you to use the mail command to send emails from the terminal.
* It supports **SMTP**, which is required to send emails via services like **Gmail, Outlook, or SMTP servers**.

## **🔹 Step 2: Configure an SMTP Server**

There are **two main ways** to send emails from Ubuntu:

1. **Using a public SMTP provider** (e.g., Gmail, Outlook, SMTP relay).
2. **Using postfix (a local mail server)**.

### ****Option 1: Configure Gmail SMTP (Recommended)****

✅ **Best for sending alerts without setting up a full mail server**

1️⃣ **Edit /etc/msmtprc to configure SMTP**

bash

CopyEdit

sudo nano /etc/msmtprc

Paste this:

account default

host smtp.gmail.com

port 587

auth on

user your-email@gmail.com

password your-app-password

tls on

tls\_starttls on

from your-email@gmail.com

logfile /var/log/msmtp.log

📌 **Important**

* Replace your-email@gmail.com with your **Gmail address**.
* Replace your-app-password with a **Gmail App Password** (if using Gmail with 2FA).

2️⃣ **Set Correct Permissions**

bash

CopyEdit

sudo chmod 600 /etc/msmtprc # Protects password file

3️⃣ **Test Email Sending**

bash

CopyEdit

echo "Test email from Ubuntu" | mail -s "Test Subject" recipient@example.com

📌 **If it works, your SMTP setup is correct!**

### ****Option 2: Install**** postfix ****for a Local Mail Server****

✅ **Best for sending emails within a local network**

1️⃣ Install Postfix:

bash

CopyEdit

sudo apt install -y postfix

2️⃣ Configure Postfix to use SMTP relay:

bash

CopyEdit

sudo nano /etc/postfix/main.cf

Modify or add:

ini

CopyEdit

relayhost = [smtp.gmail.com]:587

3️⃣ Restart Postfix:

bash

CopyEdit

sudo systemctl restart postfix

4️⃣ Test email sending:

bash

CopyEdit

echo "Postfix Email Test" | mail -s "Test Email" recipient@example.com

## **🔍 Verifying Email Logs**

If emails don’t send, check logs:

bash

CopyEdit

cat /var/log/mail.log # Check mail errors

tail -f /var/log/msmtp.log # If using msmtp

## **🚀 Summary**

| **Method** | **Best For** | **Setup Time** | **Recommended** |
| --- | --- | --- | --- |
| **Gmail SMTP (msmtp)** | Sending alerts from scripts | Fast (5 min) | ✅ Yes (Easy & Secure) |
| **Postfix Mail Server** | Internal mail services | Medium (10-15 min) | ✅ Yes (For Local Networks) |

*#!/bin/bash*

threshold=80

usage=$(df -kh /mnt/c/ | awk 'NR==2 {print $5}' | sed 's/%//g')

send\_alert (){

    echo " Warning - Disk Space has exceeded $threshold and the current usage is $usage "

    echo "$date -- Disk Space has exceeded -- $usage" >> log\_usage.txt

*# Optional: Send an email alert (Requires 'mail' command to be installed)*

*# echo "Disk usage is critical: ${USAGE}%" | mail -s "Disk Space Alert" admin@example.com*

}

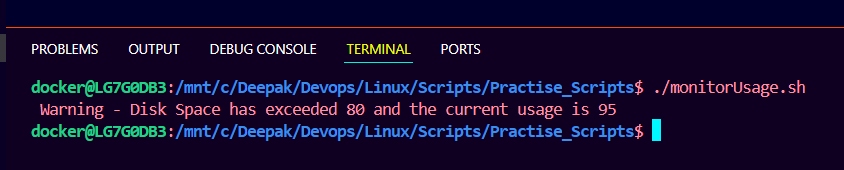
if [[ "$usage" -ge "$threshold" ]]; then

    send\_alert

else

    echo "Disk Space is normal : $usage %"

fi



1. Create a script to parse a log file and extract lines containing the word "ERROR".

*#!/bin/bash*

logfile="/mnt/c/Deepak/Devops/Linux/Scripts/Practise\_Scripts/server.log"

outputfile="parse\_error.log"

if [[ ! -f "$logfile" ]]; then

    echo "File does not exist...Exiting"

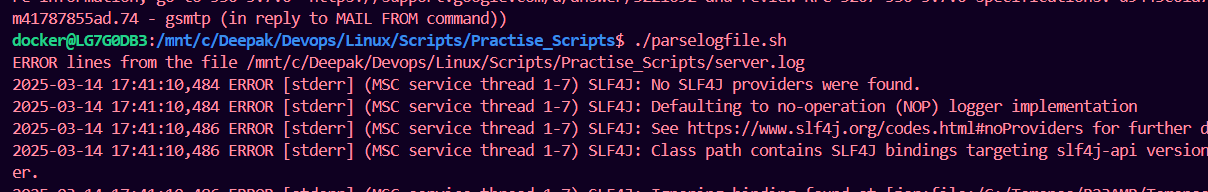
    exit 1

fi

grep "ERROR" $logfile >> $outputfile

echo "ERROR lines from the file $logfile"

cat "$outputfile"



1. Write a script to find all empty files in a directory and delete them.

*#!/bin/bash*

echo "Enter the dir in which we need to find the empty files"

read emptyfilesdir

find "$emptyfilesdir" -type f -empty

if [[ *$?* -eq 0 ]]; then

    echo "Delete the empty files (y/n) :"

    read ans

    if [[ "$ans" = "y" ]]; then

        echo "Deleting the empty files"

        find "$emptyfilesdir" -type f -empty -delete

    else

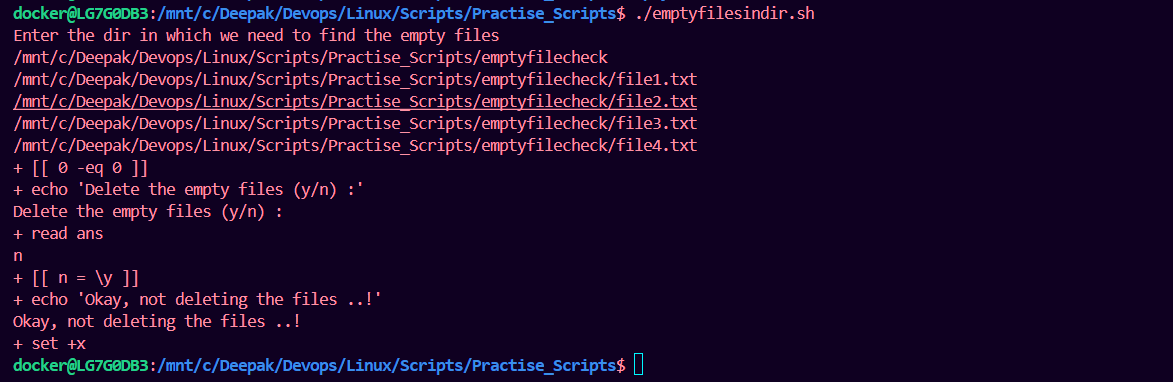
        echo "Okay, not deleting the files ..!"

    fi

else

    echo "No Empty files"

fi



1. How do you handle command-line arguments in a shell script? Write an example.

In a Bash script, you can handle command-line arguments using **positional parameters** ($1, $2, etc.) or using **flags/options** (getopts).

## **1️⃣ Basic Example: Using Positional Parameters**

### ****📜**** greet.sh ****(Script to Handle Command-Line Arguments)****

bash

CopyEdit

#!/bin/bash

# Check if the user provided two arguments

if [ $# -lt 2 ]; then

echo "Usage: $0 <name> <age>"

exit 1

fi

# Assign arguments to variables

NAME=$1

AGE=$2

# Print a greeting message

echo "Hello, $NAME! You are $AGE years old."

### ****🔍 Running the Script****

1️⃣ **Give execute permission**

bash

CopyEdit

chmod +x greet.sh

2️⃣ **Run with arguments**

bash

CopyEdit

./greet.sh Deepak 25

3️⃣ **Output**

sql

CopyEdit

Hello, Deepak! You are 25 years old.

## **2️⃣ Advanced Example: Using Named Options (**getopts**)**

If you want to use \*\*flags like -n for name and -a for age\*\*, use getopts`.

### ****📜**** greet\_options.sh ****(Script with Named Arguments)****

bash

CopyEdit

#!/bin/bash

# Default values

NAME="Guest"

AGE="Unknown"

# Parse command-line options

while getopts "n:a:" opt; do

case "$opt" in

n) NAME=$OPTARG ;; # -n option for Name

a) AGE=$OPTARG ;; # -a option for Age

\*) echo "Usage: $0 -n <name> -a <age>"; exit 1 ;;

esac

done

# Print the greeting

echo "Hello, $NAME! You are $AGE years old."

### ****🔍 Running the Script with Flags****

bash

CopyEdit

chmod +x greet\_options.sh

./greet\_options.sh -n Deepak -a 25

**Output:**

sql

CopyEdit

Hello, Deepak! You are 25 years old.

## **🎯 Summary: Which Method to Use?**

| **Method** | **Use Case** |
| --- | --- |
| **Positional Parameters ($1, $2)** | Simple scripts with fixed argument order |
| **getopts (Named Options: -n, -a)** | More user-friendly and flexible |

1. Write a script to compress all files in a directory into a .tar.gz file.

*#!/bin/bash*

echo "Enter the dir need to be archived ..!"

read dirarchive

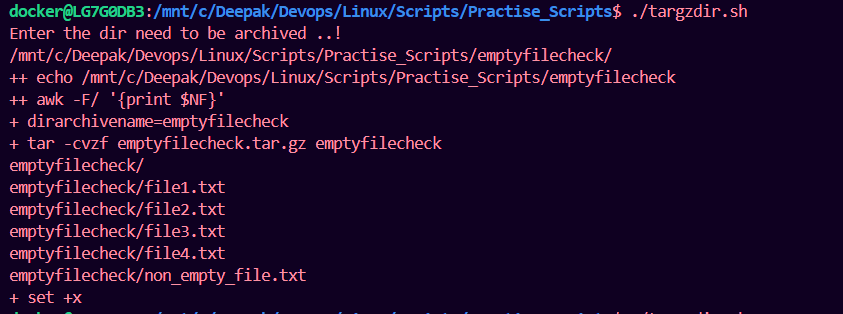
dirarchive=${dirarchive%/}

set -x

dirarchivename=$(echo $dirarchive | awk -F'/' '{print $NF}')

tar -cvzf "$dirarchivename".tar.gz "$dirarchivename"

set +x



1. Write a script to check if a process is running and restart it if it's not.

*#!/bin/bash*

*# Define the process name*

echo "Enter the PROCESS to be checked :"

read process

PROCESS\_NAME="$process"

echo "Process : $PROCESS\_NAME"

set -x

LOG\_FILE="/var/log/process\_monitor.log"

*# Check if the process is running*

if pgrep -x "$PROCESS\_NAME" > /dev/null; then

    echo "$(date) - $PROCESS\_NAME is running." >> "$LOG\_FILE"

else

    echo "$(date) - $PROCESS\_NAME is NOT running. Restarting..." >> "$LOG\_FILE"

*# Restart the process using systemctl (for services)*

    sudo systemctl restart "$PROCESS\_NAME"

*# Alternative: If it's not a systemd service, use nohup to restart manually*

*# nohup /path/to/$PROCESS\_NAME > /dev/null 2>&1 &*

fi

set +x

1. Explain the difference between > and >> in shell scripting.

Both > and >> are **output redirection operators** in shell scripting, but they behave differently when handling files.

## **1️⃣** > **(Overwrite Redirect)**

✅ **Redirects output to a file, overwriting its content**.  
✅ **Creates a new file if it doesn’t exist**.

### ****Example****

bash

CopyEdit

echo "Hello, Deepak!" > output.txt

🔹 **If output.txt exists**, it is **overwritten**.  
🔹 **If output.txt does not exist**, it is **created**.

#### ****Example Output (****output.txt ****contents)****

CopyEdit

Hello, Deepak!

## **2️⃣** >> **(Append Redirect)**

✅ **Redirects output to a file, but appends to the existing content**.  
✅ **Preserves existing data** instead of overwriting.

### ****Example****

bash

CopyEdit

echo "Welcome to DevOps!" >> output.txt

🔹 **If output.txt exists**, it **adds the new line** to the end of the file.  
🔹 **If output.txt does not exist**, it **creates it**.

#### ****Example Output (****output.txt ****contents after appending)****

css

CopyEdit

Hello, Deepak!

Welcome to DevOps!

🚀 **The previous content remains!**

## **🔍 Comparison Table**

| **Operator** | **Behavior** |
| --- | --- |
| > | **Overwrites** the file (creates if missing) |
| >> | **Appends** to the file (creates if missing) |

## **📌 Practical Use Cases**

| **Scenario** | **Use > or >>?** | **Example** |
| --- | --- | --- |
| **Clear & write new logs** | > | echo "New log entry" > logfile.txt |
| **Append new log entries** | >> | echo "Error: Something failed" >> logfile.txt |
| **Overwrite a report file** | > | ls -l > report.txt |
| **Add more data to a report** | >> | ls -l >> report.txt |

## **🔥 Summary**

✔ **Use > when you want to replace the file contents.**  
✔ **Use >> when you want to add new data without losing old content.**

1. Write a script to generate a Fibonacci sequence up to n terms.

*#!/bin/bash*

a=0

b=1

echo "Enter the number n :"

read n

echo -n "Fibonacci series of $n is : $a $b "

for(( i=2;i<=n;i++ ))

do

    sum=$(( a+b ))

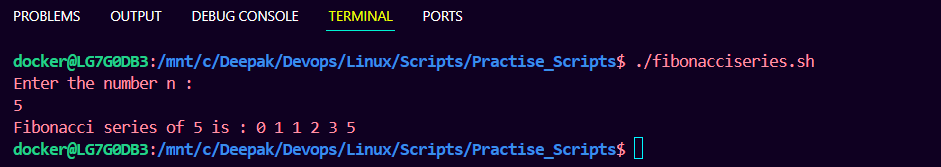
    echo -n "$sum "

    a=$b

    b=$sum

done

echo ''



1. Create a script to check if a website is reachable using ping.

*#!/bin/bash*

*# Check if a website URL is provided*

if [ *$#* -ne 1 ]; then

    echo "Usage: $0 <website\_url>"

    exit 1

fi

WEBSITE=$1

LOG\_FILE="ping\_log.txt"

*# Ping the website (send 2 packets, wait 2 seconds)*

ping -c 2 -W 2 "$WEBSITE" > /dev/null 2>&1

*# Check the exit status of ping command*

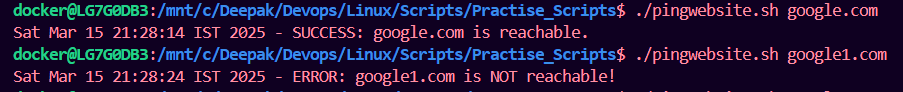
if [ *$?* -eq 0 ]; then

    echo "$(date) - SUCCESS: $WEBSITE is reachable." | tee -a "$LOG\_FILE"

else

    echo "$(date) - ERROR: $WEBSITE is NOT reachable!" | tee -a "$LOG\_FILE"

fi



1. Write a script to clean up temporary files older than 7 days in /tmp.

*#!/bin/bash*

*# Directory to clean*

TEMP\_DIR="/tmp"

LOG\_FILE="/var/log/tmp\_cleanup.log"

DAYS\_OLD=7

*# Check if the directory exists*

if [ ! -d "$TEMP\_DIR" ]; then

    echo "Error: Directory $TEMP\_DIR not found!"

    exit 1

fi

*# Find and delete files older than 7 days*

echo "Cleaning up files older than $DAYS\_OLD days in $TEMP\_DIR..."

find "$TEMP\_DIR" -type f -mtime +$DAYS\_OLD -exec rm -f {} \; -print >> "$LOG\_FILE"

*# Print completion message*

echo "Cleanup completed. Deleted files are logged in $LOG\_FILE"

**Expert Level**

1. Write a script to create a backup of a directory, including a timestamp in the backup file name.

*#!/bin/bash*

*# Directory to back up (Change this to your target directory)*

SOURCE\_DIR="/mnt/c/Deepak/Devops/Linux/Scripts/Practise\_Scripts/emptyfilecheck/"

SOURCE\_DIR=${SOURCE\_DIR%/}

echo "Source dir : $SOURCE\_DIR"

*# Backup storage location (Change as needed)*

BACKUP\_DIR="/home/docker/backups"

SOURCE\_DIR\_NAME=$(echo "$SOURCE\_DIR" | awk -F'/' '{print $NF}')

echo "SOURCE\_DIR\_NAME : $SOURCE\_DIR\_NAME"

*# Create backup directory if it doesn't exist*

mkdir -p "$BACKUP\_DIR"

*# Generate timestamp*

TIMESTAMP=$(date +"%Y%m%d\_%H%M%S")

*# Backup file name*

BACKUP\_FILE="$BACKUP\_DIR/${SOURCE\_DIR\_NAME}\_$TIMESTAMP.tar.gz"

echo "BACKUP\_FILE : $BACKUP\_FILE"

*# Create the backup*

tar -czf "$BACKUP\_FILE" "$SOURCE\_DIR"

*# Verify if backup was created successfully*

if [ *$?* -eq 0 ]; then

    echo "Backup successful: $BACKUP\_FILE"

else

    echo "Backup failed!"

    exit 1

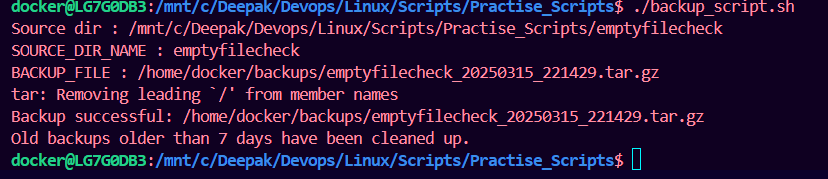
fi

*# Optional: Delete backups older than 7 days*

find "$BACKUP\_DIR" -type f -name "backup\_\*.tar.gz" -mtime +7 -exec rm -f {} \;

*# Completion message*

echo "Old backups older than 7 days have been cleaned up."



1. How would you automate a task using a cron job? Describe with an example.
2. Write a script to send an email notification when a specific event occurs.
3. Write a script to extract specific columns from a CSV file.
4. Create a script to simulate a progress bar while performing a long-running task.
5. Write a script to find duplicate files in a directory.
6. Explain how you would debug a shell script with errors.
7. Write a script to calculate CPU and memory usage of a process.
8. Create a menu-driven shell script with options like "Display Date", "List Files", and "Exit".
9. Write a script to compare two directories and list the files that are only present in one of them.

**Shell Scripting Questions for Practice (Extended)**

**Beginner Level**

1. Write a script to print the name of the script and the number of arguments passed to it.
2. How do you create and execute a shell script in Linux?
3. Write a script to calculate the sum of two numbers.
4. Create a script to fetch the hostname of the system.
5. Write a script to display the list of files in a directory.
6. What are environment variables in shell scripting? Write a script to print all environment variables.
7. Write a script to check if a given string is a palindrome.
8. Write a script to count the number of words in a file.
9. Write a script to check whether a file is readable, writable, and executable.
10. Create a script to display your system's username and home directory.

**Intermediate Level**

1. Write a script to rename all .txt files in a directory to .bak.
2. Write a script to find the sum of digits of a number.
3. Create a script to print the multiplication table of a number.
4. Write a script to find all files larger than 1MB in a directory.
5. Write a script to check if a number is prime.
6. Create a script to display the 5 most recently modified files in a directory.
7. Write a script to merge two files line by line.
8. Write a script to display the permissions of all files in a directory.
9. Write a script to replace all occurrences of a word in a file with another word.
10. Create a script to display the reverse of a number entered by the user.

**Advanced Level**

1. Write a script to monitor a directory and log changes (file creation, deletion, modifications).
2. Create a script to find the number of unique lines in a file.
3. Write a script to find and replace a string in all files in a directory.
4. How do you trap signals in shell scripting? Write a script to handle Ctrl+C.
5. Write a script to generate a random password of 10 characters.
6. Write a script to calculate the total size of all files in a directory.
7. Write a script to check if a process is running and kill it if necessary.
8. Create a script to sort the lines in a file in ascending and descending order.
9. Write a script to extract the IP address of your system.
10. Write a script to count the number of directories, files, and symbolic links in a directory.

**Expert Level**

1. Write a script to rotate log files (archive old logs and create a new one).
2. Create a script to monitor CPU usage and send an alert if it exceeds 90%.
3. Write a script to split a large file into smaller chunks.
4. Create a script to check if the system is running low on disk space and delete old files to free up space.
5. Write a script to backup all .log files in a directory to a compressed .tar.gz file.
6. Write a script to automate SSH login without a password using public/private key authentication.
7. Create a script to calculate the uptime of the system.
8. Write a script to parse a JSON file and extract specific values (use tools like jq).
9. Write a script to compare two files line by line and highlight the differences.
10. Create a script to validate an email address format.

**Bonus Questions for Practice**

1. Write a script to create a user account and set a default password.
2. Write a script to calculate the difference between two dates.
3. Create a script to send a notification when a specific port is in use.
4. Write a script to extract email addresses from a text file.
5. Create a script to find the longest word in a file.
6. Write a script to check if a file contains a specific string.
7. Create a script to calculate the average of numbers entered by the user.
8. Write a script to find the top 10 most frequently used commands in the shell history.
9. Write a script to find and delete duplicate lines in a file.
10. Create a script to display a colorful banner or ASCII art using the figlet or toilet command.

**Specialized Use Cases**

These questions focus on real-world tasks often encountered in Linux administration or DevOps roles: 51. Write a script to fetch the current weather information using an API (e.g., OpenWeatherMap, using curl).

52. Create a script to automate the installation of a software package (e.g., nginx or apache).

53. Write a script to extract and summarize system logs (e.g., /var/log/syslog).

54. Create a script to monitor a specific service (e.g., nginx) and restart it if it stops.

55. Write a script to create a cron job that runs a script every hour.

56. Create a script to compare the contents of two directories and list differences.

57. Write a script to manage permissions for a group of files (e.g., set all .sh files as executable).

58. Create a script to automate database backups (e.g., mysqldump for MySQL).

59. Write a script to monitor user login/logout activity in real time.

60. Create a script to fetch the top 5 processes consuming the most memory or CPU.

**Tips for Using These Questions**

* **Practice writing scripts**: Don’t just read the questions—implement the solutions! Test each script in a Linux/Unix environment.
* **Understand the logic**: Focus on problem-solving rather than memorizing scripts. Employers value a good understanding of scripting logic and syntax.
* **Explore advanced tools**: Learn and use tools like awk, sed, and grep to handle complex text processing tasks.
* **Debugging**: Practice debugging your scripts using set -x or bash -x to trace execution.

Here are some script questions categorized into beginner, intermediate, and advanced levels, involving awk, sed, and grep. Each question includes a brief description of what the script should accomplish.

**Beginner Level**

1. **Using grep to Search for a Pattern**:
   * **Question**: Write a script that takes a filename and a search pattern as arguments and uses grep to find and display all lines in the file that contain the pattern.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 2 ]; then

echo "Usage: $0 filename pattern"

exit 1

fi

grep "$2" "$1"

1. **Using sed to Replace Text**:
   * **Question**: Write a script that takes a filename and two strings as arguments. Use sed to replace all occurrences of the first string with the second string in the file.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 3 ]; then

echo "Usage: $0 filename old\_string new\_string"

exit 1

fi

sed -i "s/$2/$3/g" "$1"

1. **Using awk to Print Specific Columns**:
   * **Question**: Write a script that takes a filename as an argument and uses awk to print the first and third columns of a space-separated file.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 1 ]; then

echo "Usage: $0 filename"

exit 1

fi

awk '{print $1, $3}' "$1"

**Intermediate Level**

1. **Using grep with Regular Expressions**:
   * **Question**: Write a script that takes a filename and a regular expression as arguments and uses grep to find and display all lines that match the regular expression.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 2 ]; then

echo "Usage: $0 filename regex"

exit 1

fi

grep -E "$2" "$1"

1. **Using sed to Delete Lines**:
   * **Question**: Write a script that takes a filename and a line number as arguments and uses sed to delete the specified line from the file.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 2 ]; then

echo "Usage: $0 filename line\_number"

exit 1

fi

sed -i "${2}d" "$1"

1. **Using awk to Calculate Sum**:
   * **Question**: Write a script that takes a filename as an argument and uses awk to calculate and print the sum of the values in the second column.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 1 ]; then

echo "Usage: $0 filename"

exit 1

fi

awk '{sum += $2} END {print sum}' "$1"

**Advanced Level**

1. **Using grep to Count Matches**:
   * **Question**: Write a script that takes a filename and a search pattern as arguments and uses grep to count the number of lines that contain the pattern, displaying the count.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 2 ]; then

echo "Usage: $0 filename pattern"

exit 1

fi

count=$(grep -c "$2" "$1")

echo "Number of lines containing '$2': $count"

1. **Using sed for Complex Substitution**:
   * **Question**: Write a script that takes a filename and a string as arguments. Use sed to replace all occurrences of the string with "REPLACED" only if it appears at the beginning of a line.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 2 ]; then

echo "Usage: $0 filename string"

exit 1

fi

sed -i "s/^$2/REPLACED/g" "$1"

1. **Using awk for Conditional Processing**:
   * **Question**: Write a script that takes a filename as an argument and uses awk to print lines where the value in the second column is greater than 100.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $#

pls continue

.

GPT-4o mini

Certainly! Here’s the continuation of the advanced level question using awk for conditional processing, along with the complete example:

**Advanced Level (Continued)**

1. **Using awk for Conditional Processing**:
   * **Question**: Write a script that takes a filename as an argument and uses awk to print lines where the value in the second column is greater than 100.
   * **Example**:

bash

Copy Code

**#!/bin/bash**

if [ $# -ne 1 ]; then

echo "Usage: $0 filename"

exit 1

fi

awk '$2 > 100' "$1"

**Summary of Questions**

Here’s a summary of the questions categorized by difficulty level:

Beginner Level

1. **Using grep to Search for a Pattern**: Find and display lines containing a specific pattern in a file.
2. **Using sed to Replace Text**: Replace occurrences of a string in a file with another string.
3. **Using awk to Print Specific Columns**: Print the first and third columns of a space-separated file.

Intermediate Level

1. **Using grep with Regular Expressions**: Find and display lines matching a regular expression in a file.
2. **Using sed to Delete Lines**: Delete a specified line from a file.
3. **Using awk to Calculate Sum**: Calculate and print the sum of values in a specific column.

Advanced Level

1. **Using grep to Count Matches**: Count the number of lines containing a specific pattern in a file.
2. **Using sed for Complex Substitution**: Replace occurrences of a string at the beginning of a line with "REPLACED".
3. **Using awk for Conditional Processing**: Print lines where the value in a specific column is greater than a given number.

Here are **60 practice questions** on **AWK, SED, GREP, and other command-line text-processing tools** like cut, sort, uniq, tr, and find.

I've divided them into three levels:

* **Beginner (Basic commands & syntax)**
* **Intermediate (Complex patterns & pipelines)**
* **Advanced (Scripting & real-world applications)**

## **Beginner Level (Basic Commands & Syntax)**

1. Use grep to find all lines containing the word "error" in a file called log.txt.
2. Find all lines in data.txt that do **not** contain the word "success" using grep.
3. Use grep with a case-insensitive search to find "warning" in system.log.
4. Extract only the first column from users.csv using cut.
5. Print the second and fourth columns of students.csv, using cut, where fields are separated by commas.
6. Replace all occurrences of "foo" with "bar" in input.txt using sed.
7. Delete all blank lines from notes.txt using sed.
8. Use sed to insert the text "**Start:** " at the beginning of every line in tasks.txt.
9. Print all lines from words.txt that start with "A" using grep.
10. Count the number of occurrences of "ERROR" in server.log using grep.
11. Print all unique lines from names.txt using uniq.
12. Sort numbers.txt in ascending order using sort.
13. Convert all lowercase letters to uppercase in message.txt using tr.
14. Remove all digits from data.txt using tr.
15. Use find to list all .txt files in the current directory.
16. Use find to locate all files larger than 1MB in the /var/log directory.
17. Print the first 10 lines of bigfile.txt using head.
18. Print the last 5 lines of results.log using tail.
19. Use wc to count the number of words in article.txt.
20. Extract all lines from students.txt that contain numbers using grep.

## **Intermediate Level (Complex Patterns & Pipelines)**

1. Use grep to find all lines in logfile.txt that contain "failed" but not "success".
2. Extract only the email addresses from contacts.txt using grep with a regex pattern.
3. Use sed to replace all occurrences of multiple spaces with a single space in text.txt.
4. Print all lines from data.csv where the third column contains "passed" using awk.
5. Extract lines where the second field is greater than 50 in scores.csv using awk.
6. Find and display only the IP addresses from access.log using grep and regex.
7. Use awk to print the sum of all numbers in the second column of expenses.csv.
8. Replace all occurrences of "localhost" with "127.0.0.1" in config.txt using sed.
9. Count how many .log files exist in the logs/ directory using find and wc.
10. Extract the first 5 characters from every line in codes.txt using cut.
11. Use sed to delete lines that contain only whitespace in document.txt.
12. Use awk to print the last column of data.txt regardless of column count.
13. Find all lines in urls.txt that contain a valid URL using grep with a regex pattern.
14. Extract all words from notes.txt that are exactly 5 letters long using grep.
15. Convert a space-separated file into a comma-separated file using tr.
16. Find and replace all instances of "apple" with "orange" only on lines 3-10 in fruits.txt using sed.
17. Use find to delete all .tmp files in /home/user/ that are older than 7 days.
18. Use awk to print all rows from marks.csv where the fourth column is empty.
19. Print only the odd-numbered lines from list.txt using sed.
20. Sort products.txt by the second column numerically using sort and cut.

## **Advanced Level (Scripting & Real-World Applications)**

1. Write an awk script to find the **average value** of the second column in data.csv.
2. Write a sed command to replace "2023" with "2024" **only in lines containing "January"** in dates.txt.
3. Extract only the domain names from email addresses in emails.txt using awk.
4. Count how many times each unique word appears in book.txt using sort, uniq, and awk.
5. Write a pipeline using grep, awk, and sed to extract phone numbers from contacts.txt.
6. Use find and xargs to search for .log files and delete them **after confirmation**.
7. Write a one-liner using awk to print only the second and last column of table.txt.
8. Write a sed script to wrap each line in html.txt inside <p> tags.
9. Extract all valid IP addresses from network.log using grep and regex.
10. Write a pipeline using grep and awk to find the **longest line** in essay.txt.
11. Use sed to reverse the order of words in each line of sentence.txt.
12. Create a script using awk to find and print the **top 3 highest numbers** from the third column of scores.csv.
13. Write a command using find and grep to search for a keyword inside all .txt files in /docs.
14. Remove duplicate consecutive lines from logs.txt using uniq.
15. Use awk to filter rows where the second column contains a date within the last 30 days in records.csv.
16. Write a sed command to delete everything after the first space on each line in names.txt.
17. Write a script using awk to categorize values in sales.csv as **low, medium, or high** based on thresholds.
18. Create a find command to locate all symbolic links in /home/user/.
19. Extract all hashtags (#example) from tweets.txt using grep with a regex pattern.
20. Use awk to print the second and third fields **if the first field matches a user-provided value** in records.csv.