**Write a script to change the permission of the group to RWX for all the file in the current director starting with T**

Code - #!/bin/bash

# Change group permissions to RWX for all files starting with "T"

chmod g+rwx T\*

echo "Successfully changed group permissions for all files starting with 'T' to RWX."

**Write a program to make the other users readonly for all the file in the director whose path will be provided by the user .**

#!/bin/bash

# Get the directory path from the user

read -p "Enter the directory path: " directory

# Check if the directory exists

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

echo "Error: Directory '$directory' does not exist."

exit 1

fi

# Change group permissions to read-only (r--) for others using chmod

chmod go-w "$directory" # Apply to directory itself

chmod og-w "$directory/\*" # Apply to all files within the directory

echo "Successfully made all files in '$directory' read-only for other users."

**Countdown Timer: Write a script that uses a while loop to countdown from 10 to 1, printing each number on a new line, and then displays "Boom!" when the countdown reaches zero.**

#!/bin/bash

# Set the starting countdown value

countdown=10

# Loop while countdown is greater than 0

while [[ $countdown -gt 0 ]]

do

# Print the current countdown value

echo $countdown

# Decrement countdown by 1

countdown=$((countdown-1))

# Sleep for 1 second (optional for visual delay)

sleep 1

done

# Print "Boom!" at the end

echo "Boom!"

**User Input Validation: Create a script that prompts the user to enter a positive number. Use a while loop to continuously prompt the user until a positive number is entered.**

while true; do

read -p "Enter a positive number: " number

# Check if input is a number and positive

if [[ "$number" -gt 0 && -z "${number##\*[0-9]\*}" ]]; then

echo "You entered a positive number: $number"

break

else

echo "Please enter a positive number (greater than 0)."

 fi

done

**File Processing: Write a script that reads each line from a file named data.txt and displays each line preceded by a line number, using a while loop.**

#!/bin/bash

# Check if the file exists

if [ ! -f "data.txt" ]; then

echo "The file 'data.txt' does not exist."

exit 1

fi

# Initialize the line number

line\_number=1

# Read the file line by line

while read -r line; do

echo "$line\_number: $line"

line\_number=$((line\_number + 1))

done < "data.txt"

**Directory Traversal: Create a script that traverses through all files in a directory and its subdirectories, displaying the name and size of each file.**

#!/bin/bash

# Check if a directory is provided as an argument

if [ -z "$1" ]; then

echo "Usage: $0 directory\_path"

exit 1

fi

# Traverse the directory and its subdirectories

find "$1" -type f | while read -r file; do

# Get the file size

file\_size=$(stat -c%s "$file")

# Print the file name and size

echo "File: $file, Size: $file\_size bytes"

done

**Menu System: Implement a simple menu system using a while loop, where the user can select options such as "Add", "Delete", "View", and "Exit". The loop should continue until the user chooses to exit.**

#!/bin/bash

# Function to display the menu

display\_menu() {

echo "Menu:"

echo "1. Add"

echo "2. Delete"

echo "3. View"

echo "4. Exit"

echo -n "Choose an option: "

}

# Function to handle the Add option

add\_item() {

echo "You chose to Add an item."

# Add your add item logic here

}

# Function to handle the Delete option

delete\_item() {

echo "You chose to Delete an item."

# Add your delete item logic here

}

# Function to handle the View option

view\_items() {

echo "You chose to View items."

# Add your view items logic here

}

# Main loop

while true; do

display\_menu

read -r choice

case $choice in

1)

add\_item

;;

2)

delete\_item

;;

3)

view\_items

;;

4)

echo "Exiting..."

break

;;

\*)

echo "Invalid option, please try again."

;;

esac

echo

done

**Password Validation: Write a script that prompts the user to enter a password. Use a while loop to repeatedly prompt the user until a valid password (e.g., at least 8 characters with a mix of letters, numbers, and special characters) is entered.**

#!/bin/bash

is\_valid\_password() {

  local password=$1

 if [[ ${#password} -lt 8 ]]; then

 return 1

 fi

if ! [[ "$password" =~ [A-Za-z] ]]; then

return 1

 fi

 if ! [[ "$password" =~ [\!\@\#\$\%\^\&\\*\(\)\,\.\?\":\{\}\|\<\>\] ]]; then

 return 1

fi

 return 0

}

while true; do

read -sp "Enter your password: " password

echo

if is\_valid\_password "$password"; then

echo "Password is valid."

Break

else

echo "Invalid password. It must be at least 8 characters long and contain a mix of letters, numbers, and special characters."

fi

done

**Process Monitoring: Create a script that continuously monitors a specific process (identified by its name or PID) using a while loop. Display a message if the process stops running.**

#!/bin/bash

# Process name or PID (replace with your desired process identifier)

process\_identifier="myprocess" # Can be a process name or PID

# Loop continuously

while true; do

# Check if process is running using ps

if ! ps -e | grep -q "$process\_identifier"; then

echo "Process '$process\_identifier' is not running!"

# Optionally, you can add commands to restart the process here

fi

# Sleep for a few seconds before checking again

sleep 5 # Adjust the sleep time as needed (in seconds)

done

**Downloading Files: Write a script that downloads multiple files from a list of URLs stored in a file named urls.txt. Use a while loop to read each URL from the file and download the corresponding file using wget or curl.**

#!/bin/bash

# Download directory (replace if needed)

download\_dir="downloads"

# Check if download directory exists (optional)

if [[ ! -d "$download\_dir" ]]; then

echo "Creating download directory: '$download\_dir'"

mkdir -p "$download\_dir"

fi

# URL list file

url\_list="urls.txt"

# Loop through URLs in the file

while IFS= read -r url; do

# Extract filename from URL (optional)

filename=$(basename "$url")

# Download the file with wget (adjust options as needed)

wget -q -O "$download\_dir/$filename" "$url"

# Check download status (optional)

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

echo "Downloaded: $url"

else

echo "Error downloading: $url"

fi

done < "$url\_list"

# Explanation:

# - IFS= : Sets Internal Field Separator to avoid splitting on spaces within URLs

# - read -r url : Reads a URL line with raw input handling (-r)

# - basename "$url" : Extracts filename from URL using parameter expansion

# - wget -q -O : Quiet mode (-q), specify output filename (-O)

# - $? : Captures the exit code of the previous command (wget)

**Input File Existence Check: Implement a script that prompts the user to enter the name of a file. Use a while loop to check if the file exists in the current directory. If the file does not exist, prompt the user to enter the filename again.**

#!/bin/bash

# Loop until a valid file is entered

while true; do

# Prompt for filename

read -p "Enter a filename: " filename

# Check if file exists

if [[ -f "$filename" ]]; then

echo "File '$filename' exists."

break # Exit the loop on successful file existence

fi

# File not found message

echo "File '$filename' does not exist. Please try again."

Done

**Backup Script: Create a backup script that continuously monitors a directory for changes using a while loop. Whenever a new file is added to the directory, the script should automatically create a backup of that file in a separate directory.**

#!/bin/bash

# Directory to monitor (replace with your actual directory)

source\_dir="/path/to/source/directory"

# Backup directory (replace with your desired location)

backup\_dir="/path/to/backup/directory"

# Function to create a backup

create\_backup() {

local filename="$1"

local backup\_file="$backup\_dir/$filename"

# Check if backup directory exists (optional)

if [[ ! -d "$backup\_dir" ]]; then

echo "Creating backup directory: '$backup\_dir'"

mkdir -p "$backup\_dir"

fi

# Copy the file to the backup directory

cp "$source\_dir/$filename" "$backup\_file"

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

echo "Created backup: $backup\_file"

else

echo "Error creating backup of '$filename'"

fi

}

# Get all files in the source directory initially

source\_files=( "$source\_dir/"\* )

# Loop continuously

while true; do

# Get all files in the source directory after the initial check

new\_source\_files=( "$source\_dir/"\* )

# Check for new files (set difference)

new\_files=("${new\_source\_files[@]}" "${source\_files[@]}")

new\_files=("${new\_files[@]/"$source\_dir"\//}") # Remove directory prefix

new\_files=($(comm -zdiff <(echo "${new\_source\_files[@]}") <(echo "${source\_files[@]}")))

# Process new files

for filename in "${new\_files[@]}"; do

create\_backup "$filename"

done

# Update source files list for next iteration

source\_files=("${new\_source\_files[@]}")

# Sleep for a few seconds before checking again

sleep 5 # Adjust the sleep time as needed (in seconds)

done