INTRODUCTION TO SCRIPTING

SCRIPTING

- **Scripting** refers to writing small programs, or scripts
- It is used automate tasks, manipulate data, or control software behavior.
- Scripts are often interpreted rather than compiled, making them easier to write and execute quickly.

TYPES OF SCRIPTING LANGUAGES

Shell Scripting (Bash, PowerShell)	Used for automating command-line tasks.
	Linux shell scripts (.sh), Windows PowerShell scripts (.ps1).
Web Scripting (JavaScript, PHP)	JavaScript for frontend interactions, PHP for backend processing.
	JavaScript manipulates DOM elements dynamically.
Automation Scripting (Python, Perl)	Used for system automation, file handling, and data manipulation.
	Python scripts for automating file renaming.
Data Processing Scripting (Python, R, SQL)	Used for data analysis, machine learning, and database queries.
	SQL scripts for extracting reports.
Game Scripting (Lua, JavaScript)	Embedded in game engines for defining behaviors.
	Lua scripts in Unity or Roblox.

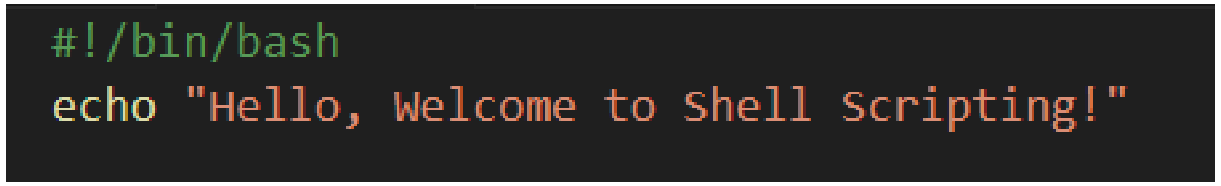
SHELL SCRIPTING

- Shell scripting in **Bash (Bourne Again Shell)** is used to automate tasks in Linux/Unix.
- A script is a sequence of commands stored in a file with a **.sh** extension.

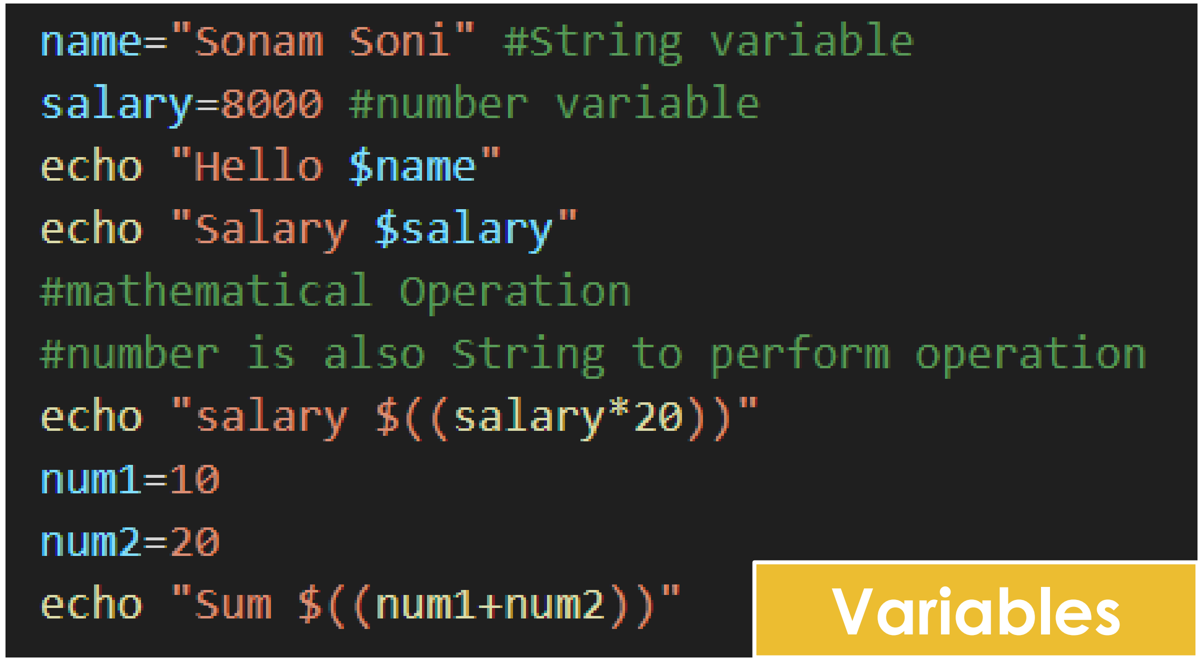
BASIC COMMANDS

Command	Description
echo	Prints output to the terminal
pwd	Prints the current working directory
ls	Lists files and directories
cd	Changes directories
mkdir	Creates a new directory
touch	Creates an empty file
rm	Removes files or directories
cp	Copies files or directories
mv	Moves or renames files
cat	Displays the contents of a file

CREATE SCRIPTS



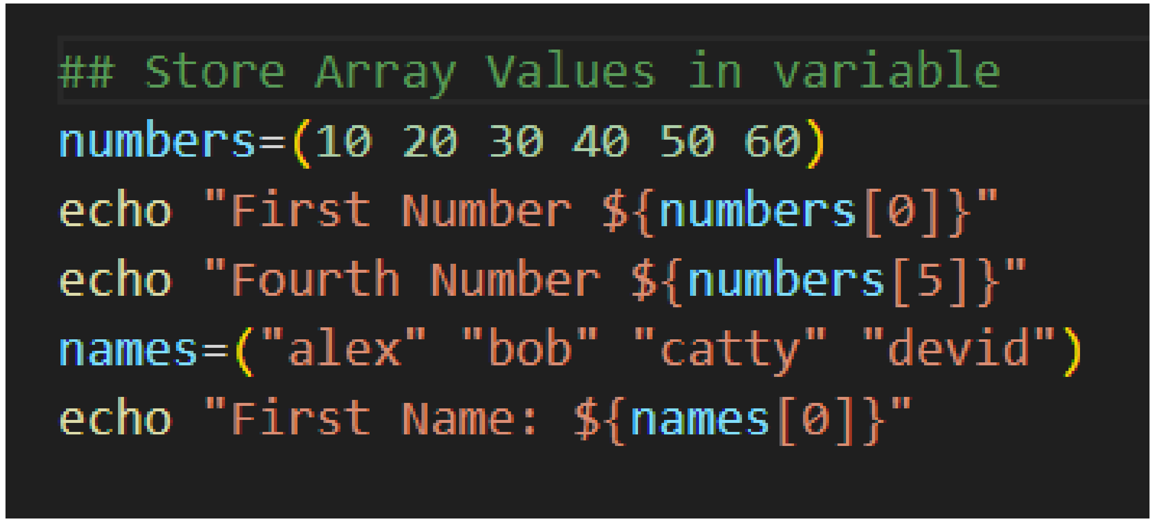
```
#!/bin/bash
echo "Hello, Welcome to Shell Scripting!"
```



```
name="Sonam Soni" #String variable
salary=8000 #number variable
echo "Hello $name"
echo "Salary $salary"
#mathematical Operation
#number is also String to perform operation
echo "salary $((salary*20))"
num1=10
num2=20
echo "Sum $((num1+num2))"
```

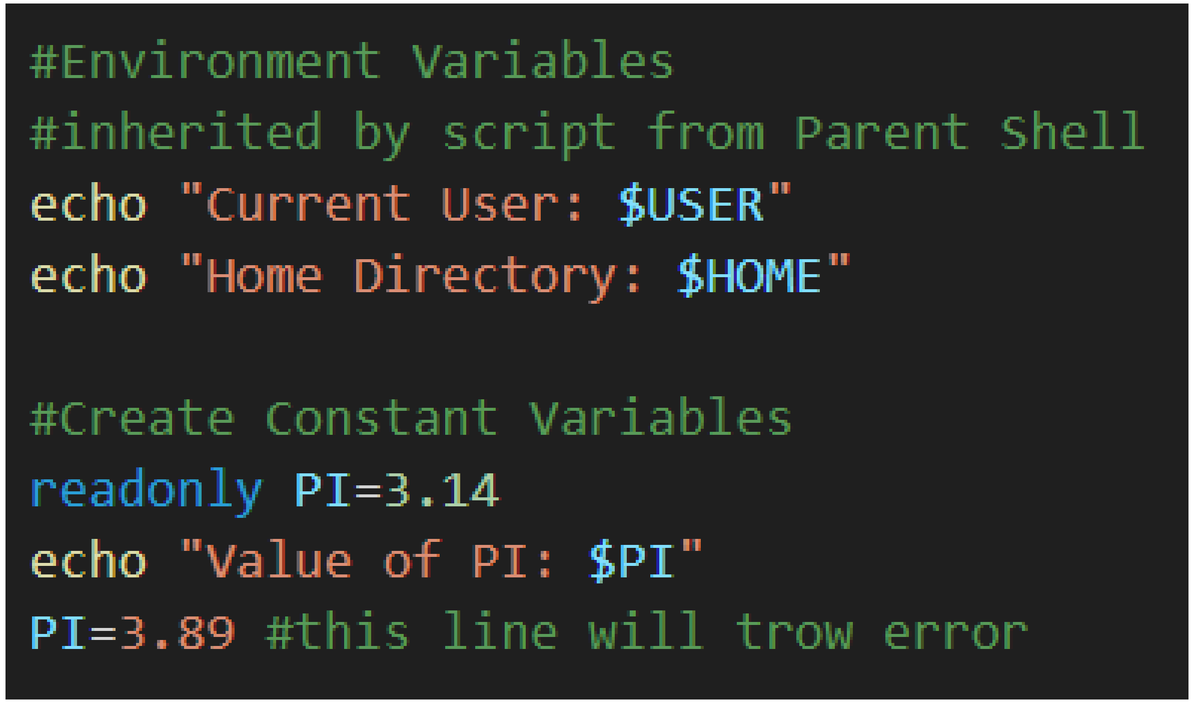
Variables

Arrays

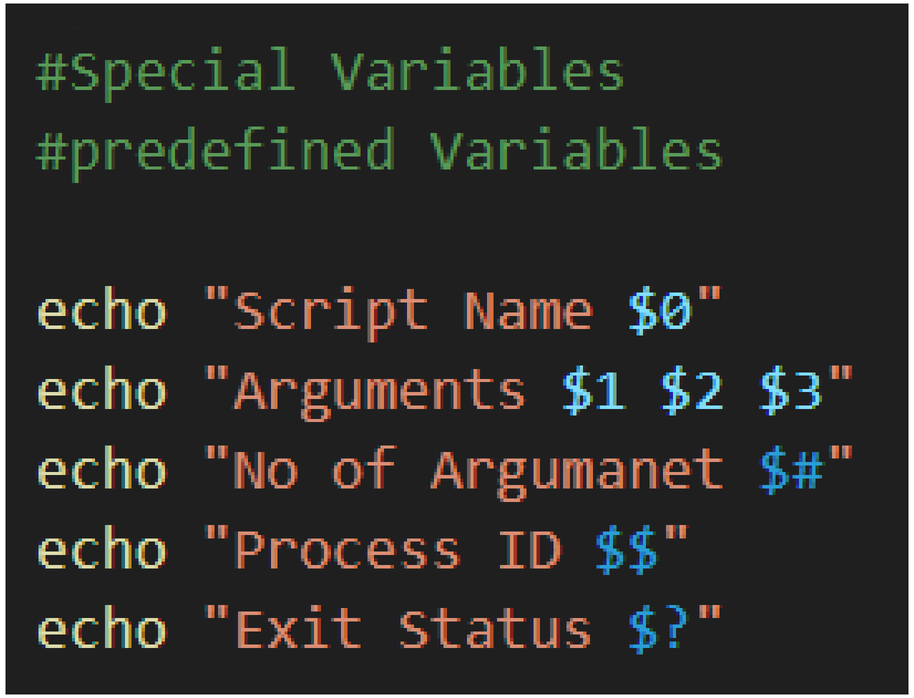


```
## Store Array Values in variable
numbers=(10 20 30 40 50 60)
echo "First Number ${numbers[0]}"
echo "Fourth Number ${numbers[5]}"
names=("alex" "bob" "catty" "devid")
echo "First Name: ${names[0]}"
```

VARIABLES

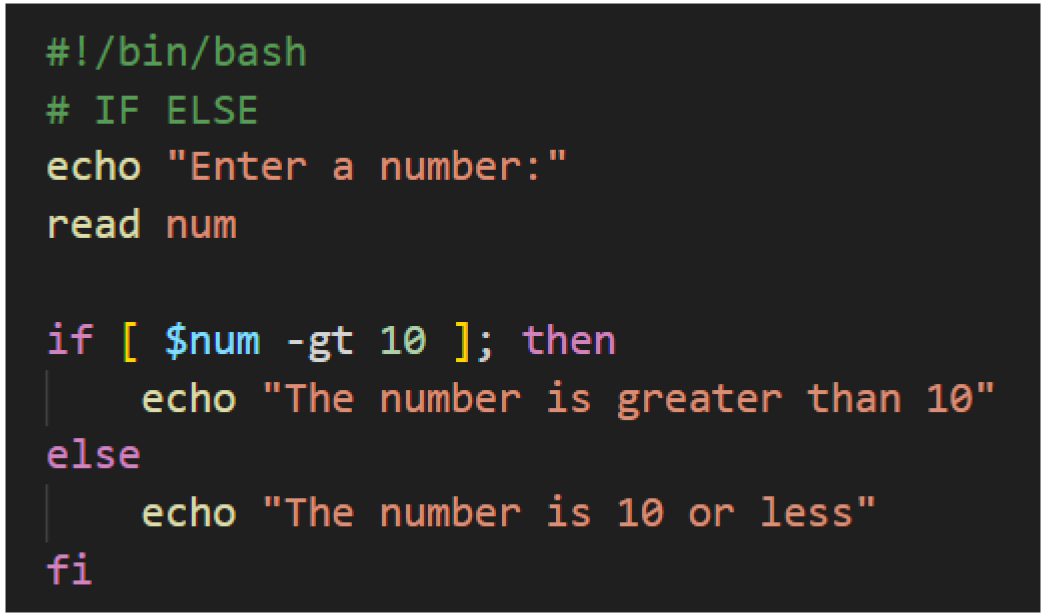


```
#Environment Variables
#inherited by script from Parent Shell
echo "Current User: $USER"
echo "Home Directory: $HOME"

#Create Constant Variables
readonly PI=3.14
echo "Value of PI: $PI"
PI=3.89 #this line will trow error
```



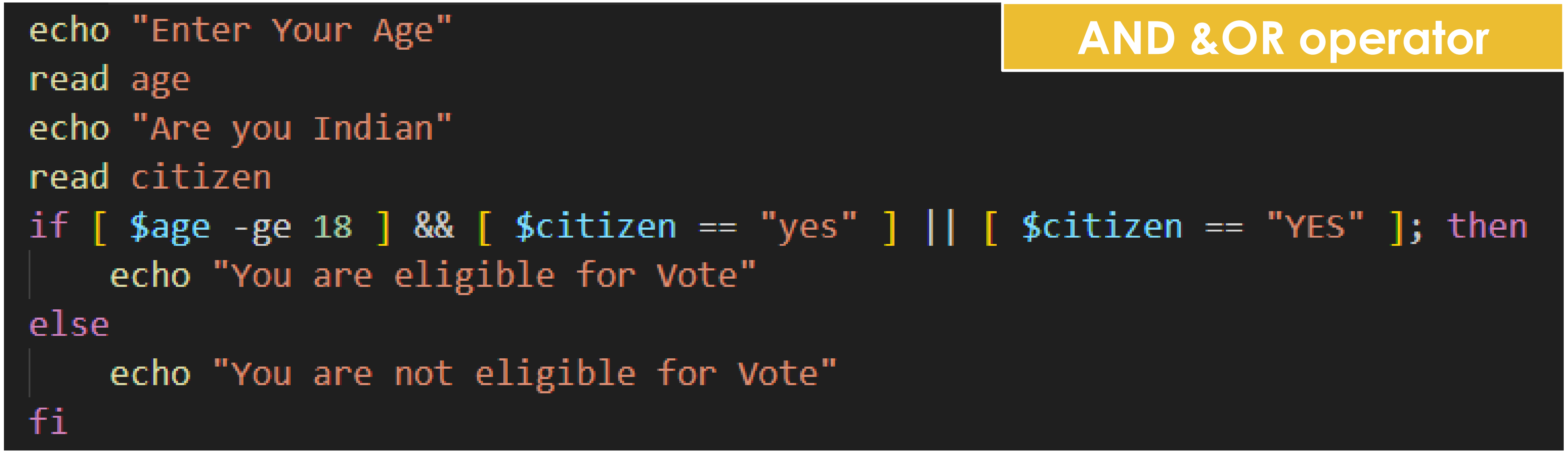
```
#Special Variables
#predefined Variables

echo "Script Name $0"
echo "Arguments $1 $2 $3"
echo "No of Argumanet $#"
echo "Process ID $$"
echo "Exit Status $?"
```

CONDITIONAL SCRIPTING



```bash
#!/bin/bash
# IF ELSE
echo "Enter a number:"
read num

if [ $num -gt 10 ]; then
    echo "The number is greater than 10"
else
    echo "The number is 10 or less"
fi
```

AND &OR operator

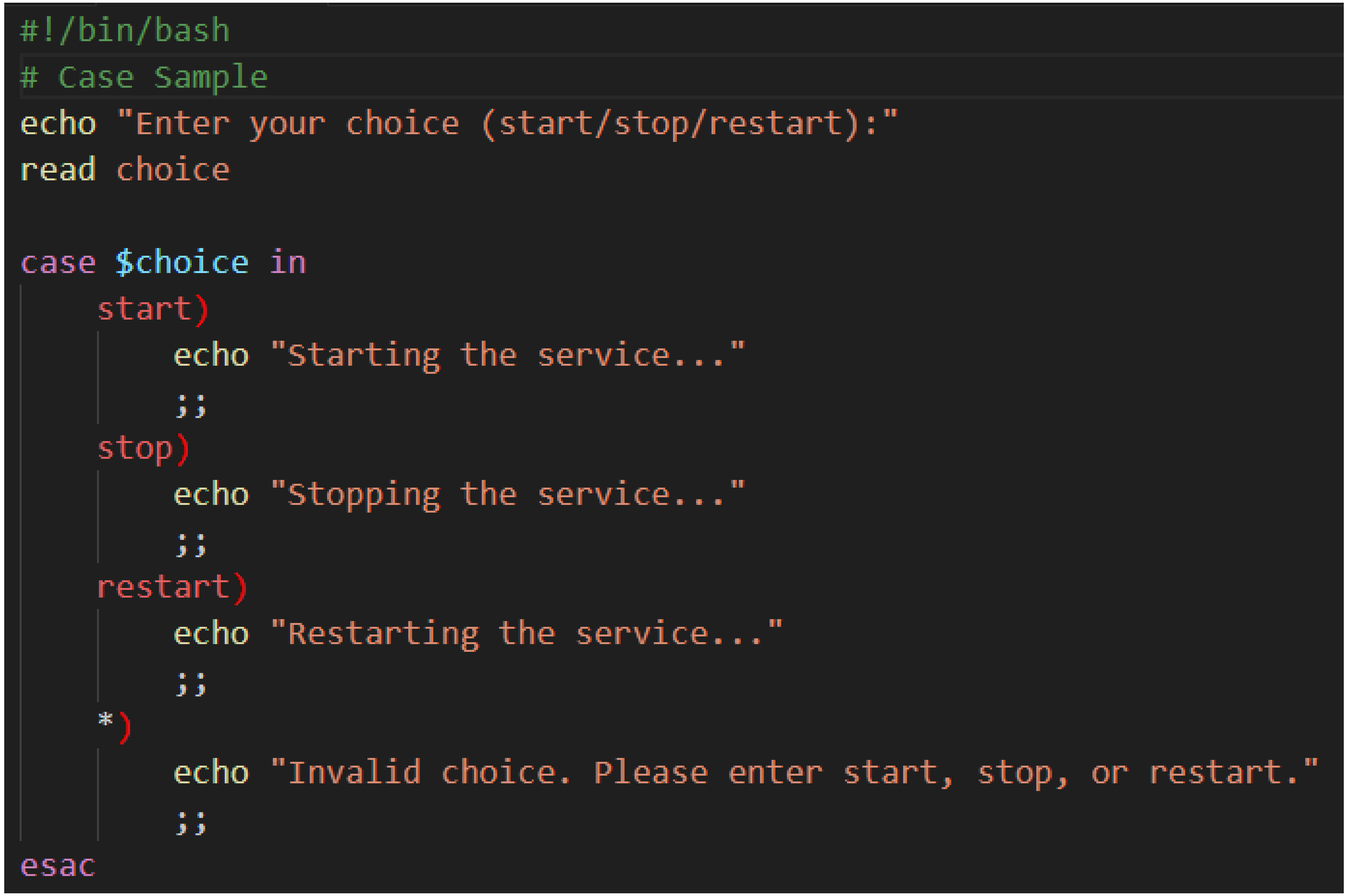


```bash
echo "Enter Your Age"
read age
echo "Are you Indian"
read citizen
if [ $age -ge 18 ] && [ $citizen == "yes" ] || [ $citizen == "YES" ]; then
    echo "You are eligible for Vote"
else
    echo "You are not eligible for Vote"
fi
```

CASE COMMAND UNDERSTANDING

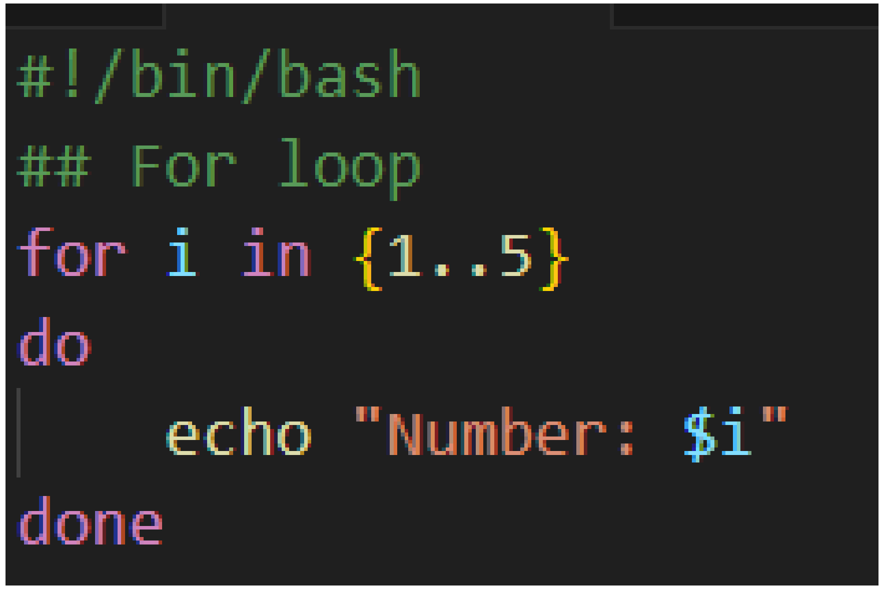
Symbol	Meaning
case variable in	Starts the case statement by checking the value of variable.
pattern1)	Defines a pattern (condition). The) closes the pattern.
;;	Ends a block of commands for a case.
*)	Represents the **default case** (if no other patterns match).
esac	Ends the case statement (reverse of case).

CASE EXAMPLE

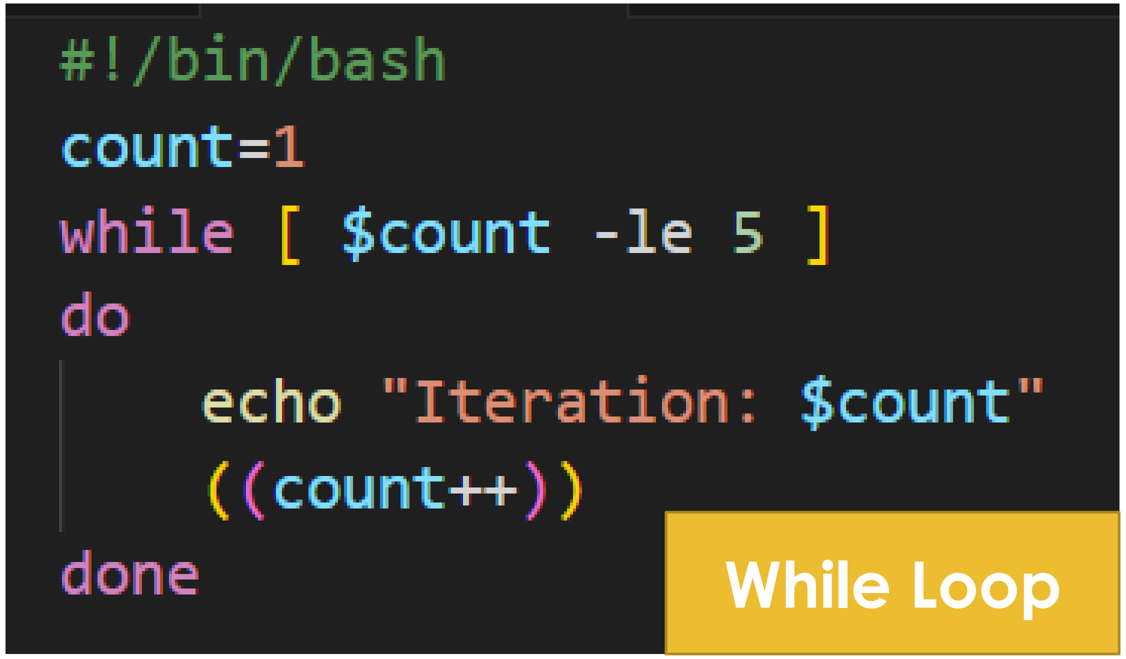


```
#!/bin/bash
# Case Sample
echo "Enter your choice (start/stop/restart):"
read choice

case $choice in
    start)
        echo "Starting the service..."
        ;;
    stop)
        echo "Stopping the service..."
        ;;
    restart)
        echo "Restarting the service..."
        ;;
    *)
        echo "Invalid choice. Please enter start, stop, or restart."
        ;;
esac
```

LOOPS



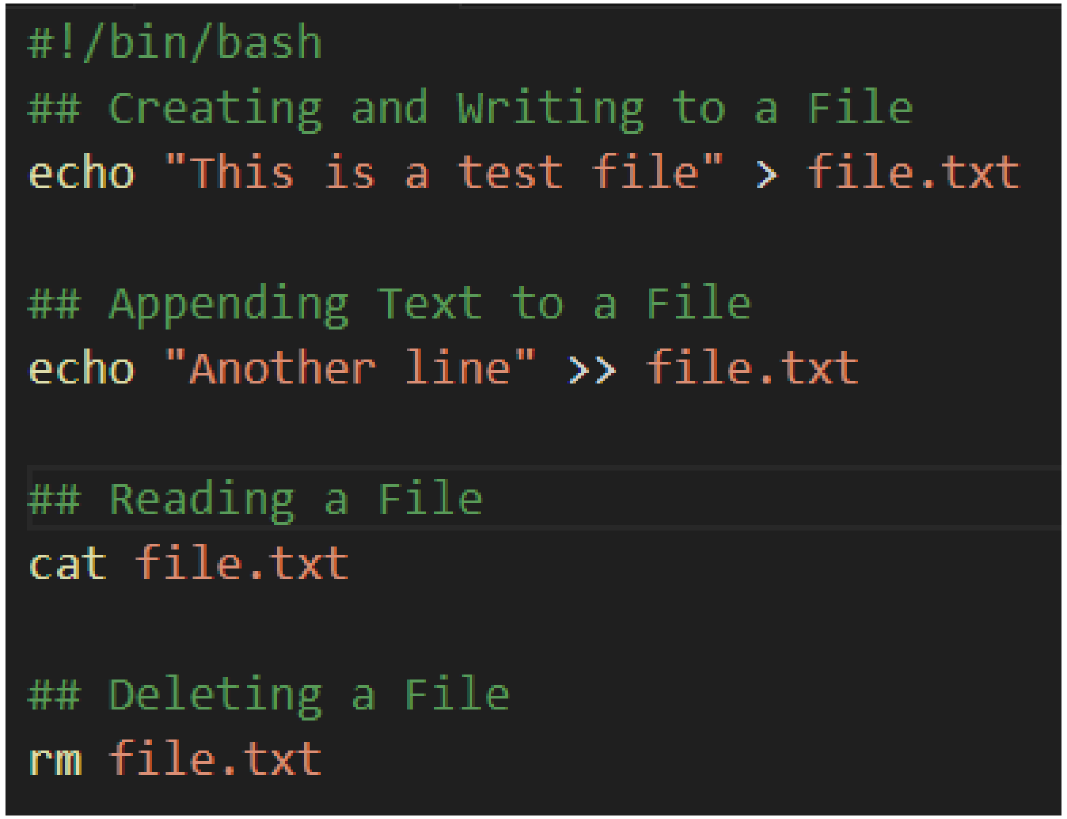
```bash
#!/bin/bash
## For loop
for i in {1..5}
do
    echo "Number: $i"
done
```



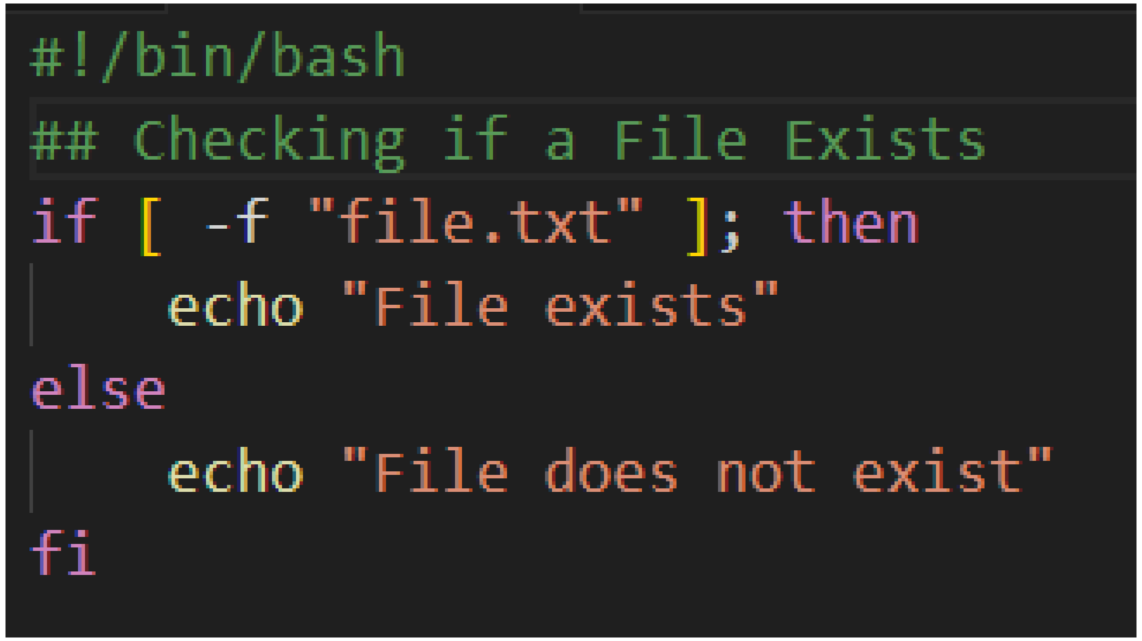
```bash
#!/bin/bash
count=1
while [ $count -le 5 ]
do
    echo "Iteration: $count"
    ((count++))
done
```

While Loop

FILE OPERATIONS

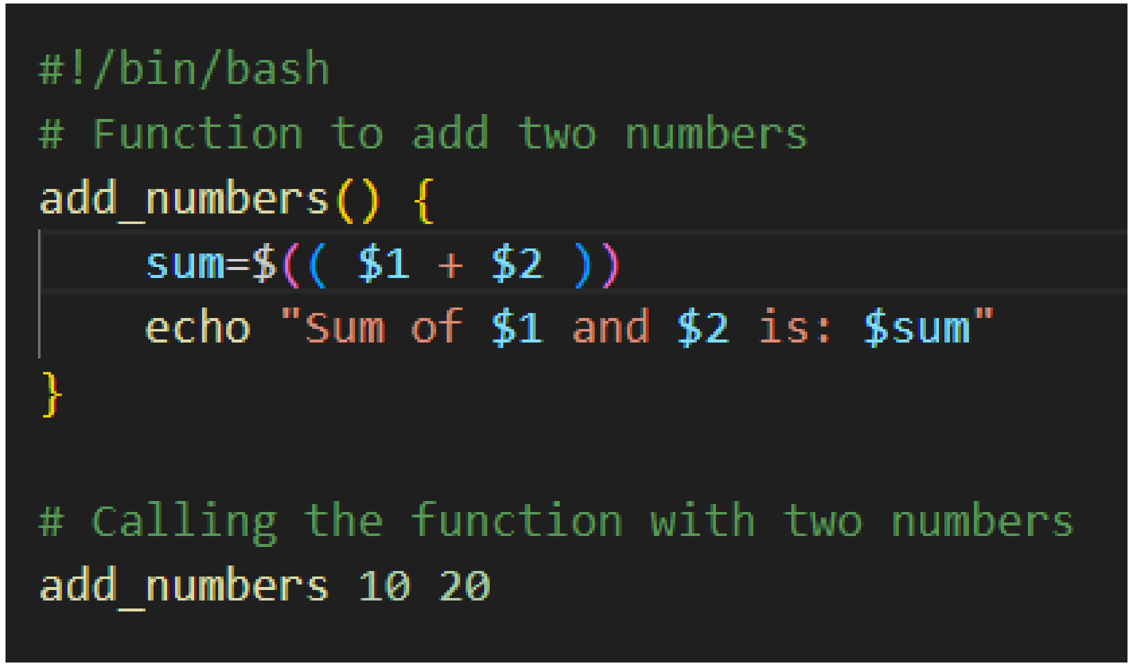


```bash
#!/bin/bash
## Creating and Writing to a File
echo "This is a test file" > file.txt

## Appending Text to a File
echo "Another line" >> file.txt

## Reading a File
cat file.txt

## Deleting a File
rm file.txt
```

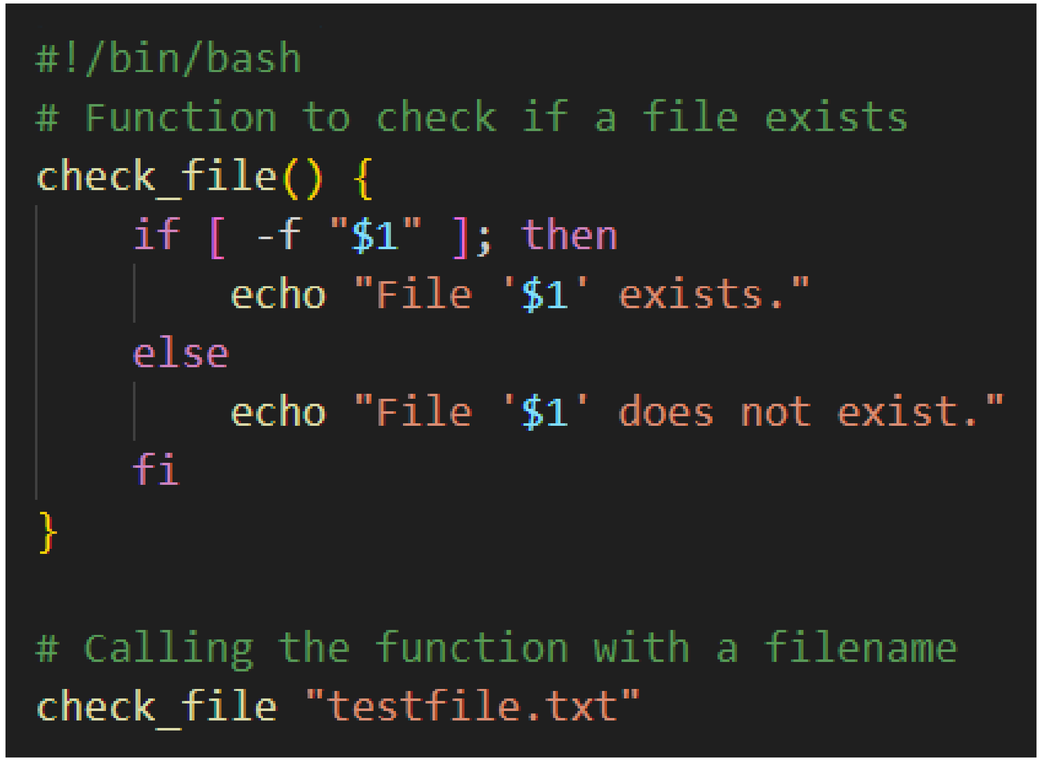


```bash
#!/bin/bash
## Checking if a File Exists
if [ -f "file.txt" ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
```

FUNCTIONS

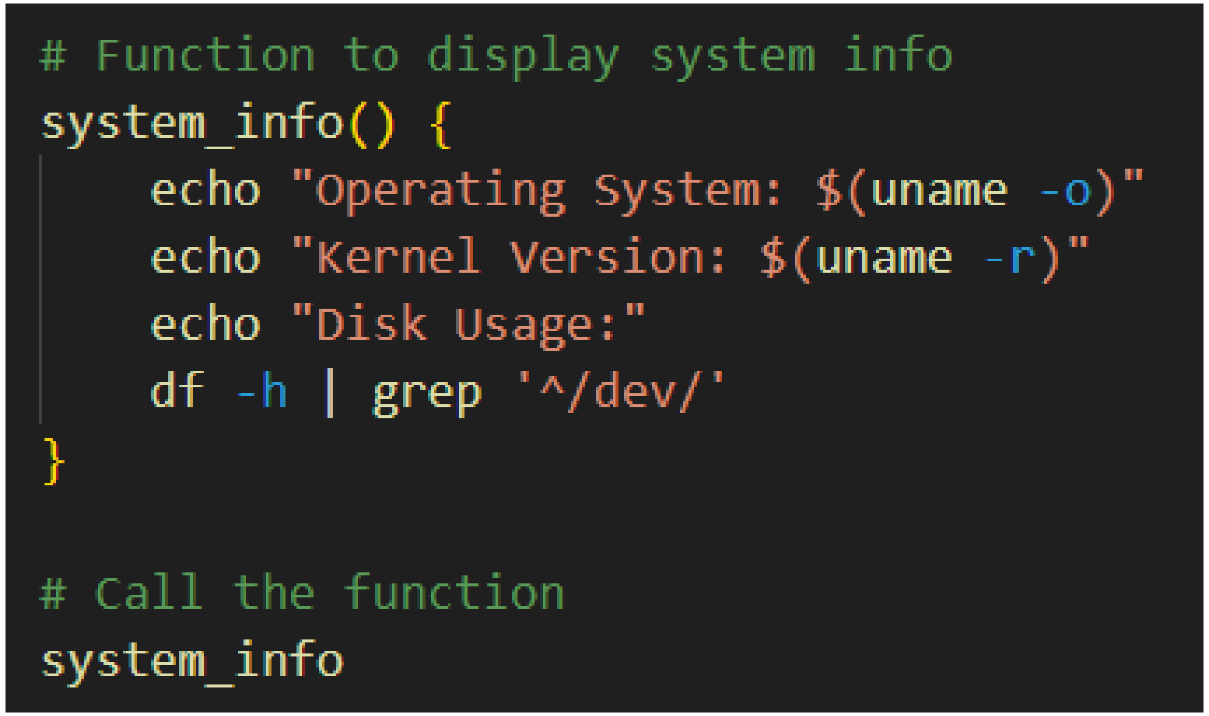


```bash
#!/bin/bash
# Function to add two numbers
add_numbers() {
    sum=$(( $1 + $2 ))
    echo "Sum of $1 and $2 is: $sum"
}

# Calling the function with two numbers
add_numbers 10 20
```



```bash
#!/bin/bash
# Function to check if a file exists
check_file() {
    if [ -f "$1" ]; then
        echo "File '$1' exists."
    else
        echo "File '$1' does not exist."
    fi
}

# Calling the function with a filename
check_file "testfile.txt"
```

FUNCTIONS



```
# Function to display system info
system_info() {
    echo "Operating System: $(uname -o)"
    echo "Kernel Version: $(uname -r)"
    echo "Disk Usage:"
    df -h | grep '^/dev/'
}

# Call the function
system_info
```

- **OStype** using uname -o
- **Kernel version** using uname -r
- **Disk usage** using df -h
- The grep '^/dev/' filters mounted disk partitions.

POWERSHELL

- PowerShell is a task automation and configuration management framework from Microsoft, primarily used for managing Windows systems.
- PowerShell works with objects, meaning commands return structured data (objects) instead of plain text.
- PowerShell is built on .NET, allowing it to leverage .NET libraries and perform complex tasks easily.
- PowerShell uses cmdlets (e.g., Get-Process, Set-ExecutionPolicy), which follow a consistent Verb-Noun naming convention.
- Older versions of PowerShell were Windows-exclusive, but PowerShell Core (now called PowerShell 7) is cross-platform (Windows, Linux, macOS).
- PowerShell scripts (.ps1 files) are heavily used for system administration, automation, and managing cloud environments.

COMMANDS

- **Displaying Output:**
  - Write-Host "Hello, PowerShell!"
- **Variables:**
  - $greeting = "Hello, World!"
  - Write-Host $greeting
- **Getting System Information:**
  - Get-ComputerInfo
- **Listing Files in a Directory:**
  - Get-ChildItem C:\Users
- **Reading User Input:**
  - $name = Read-Host "Enter your name"
  - Write-Host "Hello, $name!"

COMMANDS

- **Looping with ForEach-Object:**
  - $names = @("Alice", "Bob", "Charlie")
  - $names | ForEach-Object { Write-Host "Hello, $_" }
- **If-Else Condition:**

```
$age = Read-Host "Enter your age"
if ($age -ge 18) {
    Write-Host "You are an adult."
} else {
    Write-Host "You are a minor."
}
```

Creating and Using a Function:

```
function GreetUser {
param ($name)
Write-Host "Welcome, $name!"
}

GreetUser "John"
```

- **Checking Running Processes**
  - Get-Process
- **Stopping a Process (e.g., Notepad):**
  - Stop-Process -Name "notepad" -Force

WORKING WITH FILES

- **Create a New File**
  - New-Item -Path "C:\Users\Public\example.txt" -ItemType File
- **Write Text to a File**
  - "Hello, PowerShell File Handling!" | Out-File -FilePath "C:\Users\Public\example.txt“
- **Append Text to a File**
  - "Appending another line." | Add-Content -Path "C:\Users\Public\example.txt"
- **Read a file:**
  - Get-Content "C:\Users\Public\example.txt“
- **Delete a file:**
  - Remove-Item "C:\Users\Public\example.txt" -Force

WORKING WITH FILES

- Check the file is exist or not:

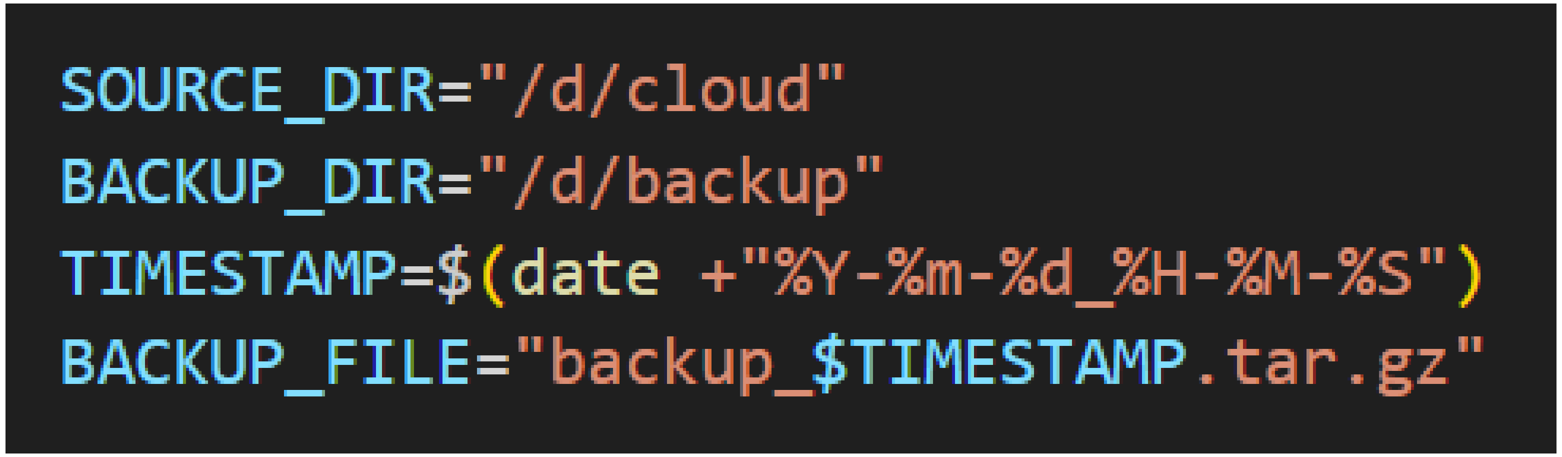
```
if (Test-Path "C:\Users\Public\example.txt") {
        Write-Host "File exists."
} else {
        Write-Host "File does not exist."
}
```

WORKING WITH FILES

- Rename File:
  - Rename-Item -Path "C:\Users\Public\example.txt" -NewName "new_example.txt"
- Copy File:
  - Copy-Item -Path "C:\Users\Public\new_example.txt" -Destination "C:\Users\Public\backup_example.txt“
- Move file:
  - Move-Item -Path "C:\Users\Public\backup_example.txt" -Destination "C:\Users\Public\Documents\"
- Loop through file:
  - Get-ChildItem "C:\Users\Public\" | ForEach-Object { Write-Host $_.Name }

ACTIVITY 1

- Automate the process of backing up important files or directories.
- Step 1: Define the source directory and backup destination



```
SOURCE_DIR="/d/cloud"
BACKUP_DIR="/d/backup"
TIMESTAMP=$(date +"%Y-%m-%d_%H-%M-%S")
BACKUP_FILE="backup_$TIMESTAMP.tar.gz"
```

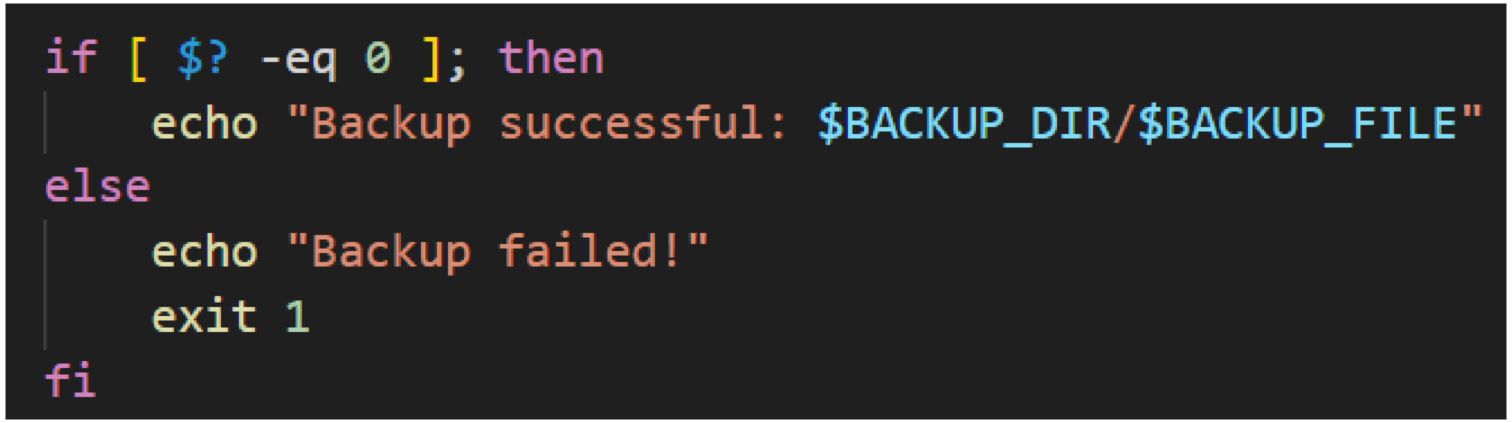
- Step 2: Ensure backup directory exists

```
mkdir -p "$BACKUP_DIR"
```

- Step 3: Create a compressed backup of the source directory

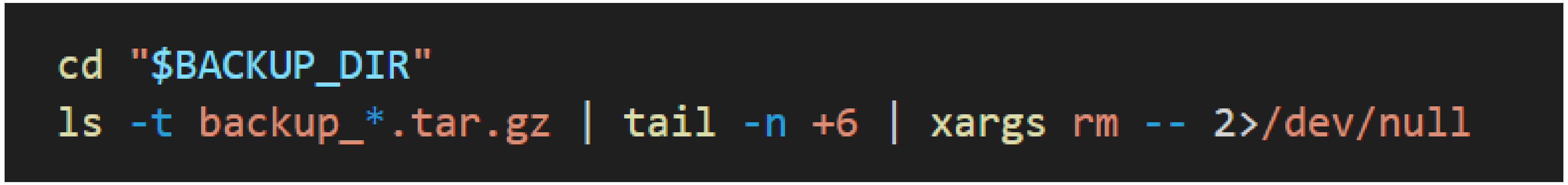
```
tar -czf "$BACKUP_DIR/$BACKUP_FILE" "$SOURCE_DIR"
```

- Step 4: Verify if the backup was created successfully



```
if [ $? -eq 0 ]; then
    echo "Backup successful: $BACKUP_DIR/$BACKUP_FILE"
else
    echo "Backup failed!"
    exit 1
fi
```

- Step 5: Delete old backups (Keep only last 5 backups)



```
cd "$BACKUP_DIR"
ls -t backup_*.tar.gz | tail -n +6 | xargs rm -- 2>/dev/null
```

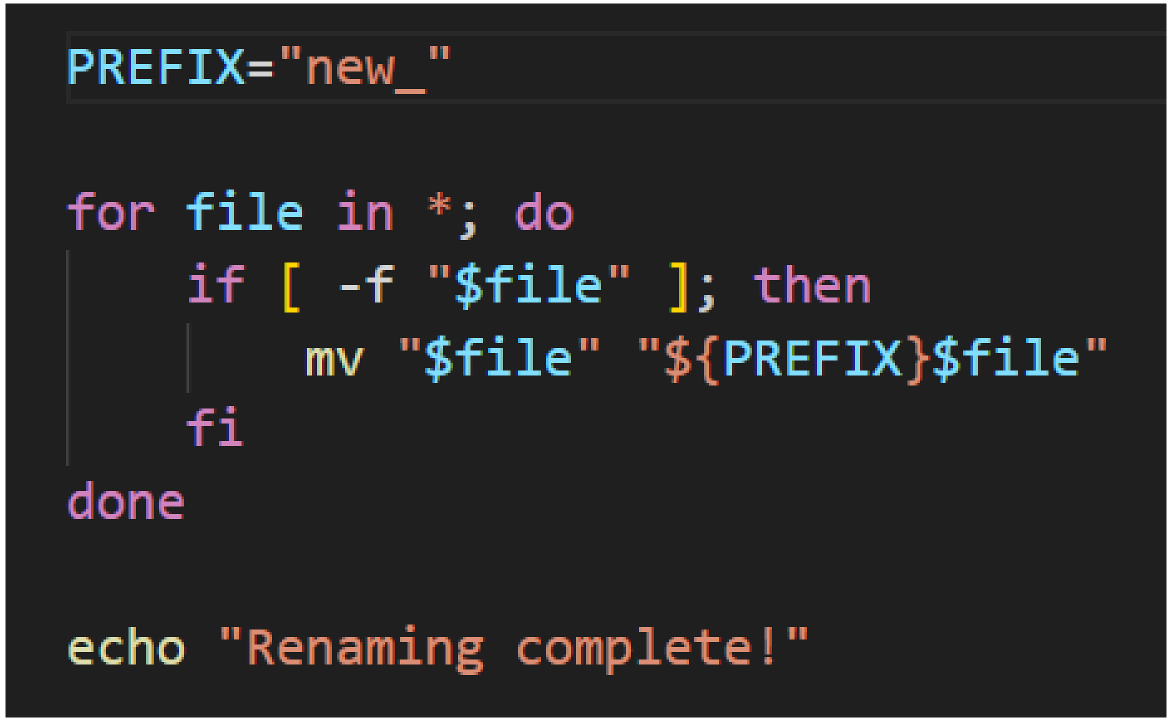
EXPLANATION

- **ls -t backup_*.tar.gz**
  - **ls** lists all backup files in the current directory.
  - **-t** sorts files by modification time -newest first - oldest last.
- **tail -n +6** skips the first 5 lines and give everything else like older backups.
- **xargs rm --** takes the list of old backups and passes them to rm for deletion.
- **--** prevents issues if filenames start with –
- **2>/dev/null**: Redirects any errors to /dev/null, **hiding errors** if no old files exist.

- To run this task daily we can set using **Task Scheduler**.

ACTIVITY 2

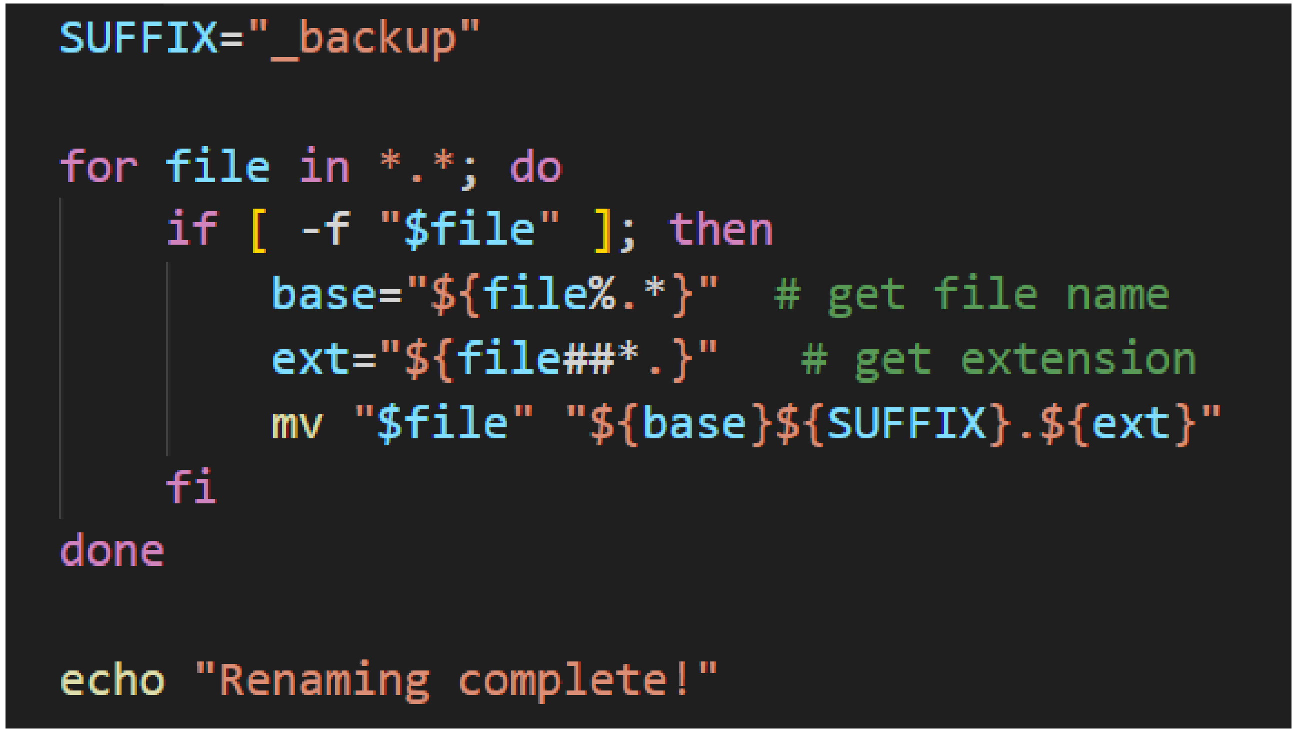
- Automate File Renaming



```
PREFIX="new_"

for file in *; do
    if [ -f "$file" ]; then
        mv "$file" "${PREFIX}$file"
    fi
done

echo "Renaming complete!"
```

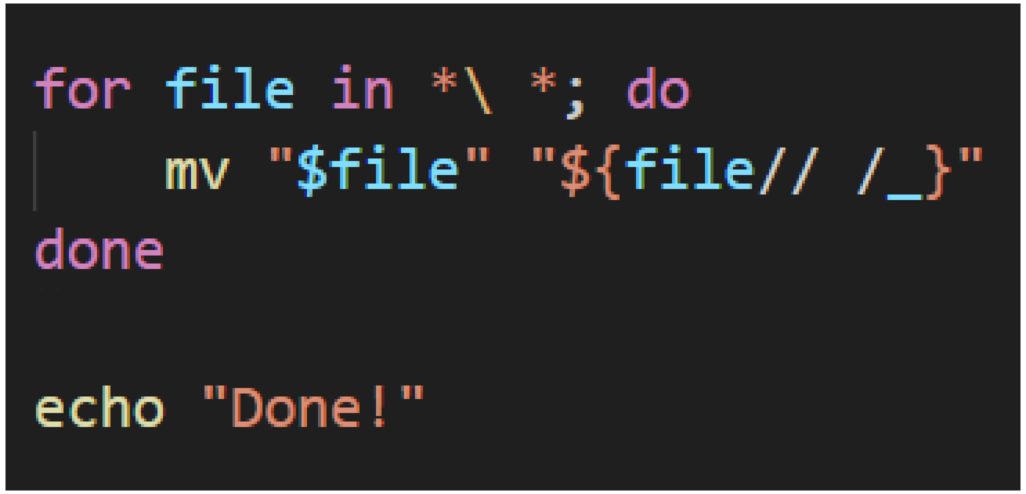
ACTIVITY 3

- Add a Suffix to File Names



```bash
SUFFIX="_backup"

for file in *.*; do
    if [ -f "$file" ]; then
        base="${file%.*}"  # get file name
        ext="${file##*.}"   # get extension
        mv "$file" "${base}${SUFFIX}.${ext}"
    fi
done

echo "Renaming complete!"
```

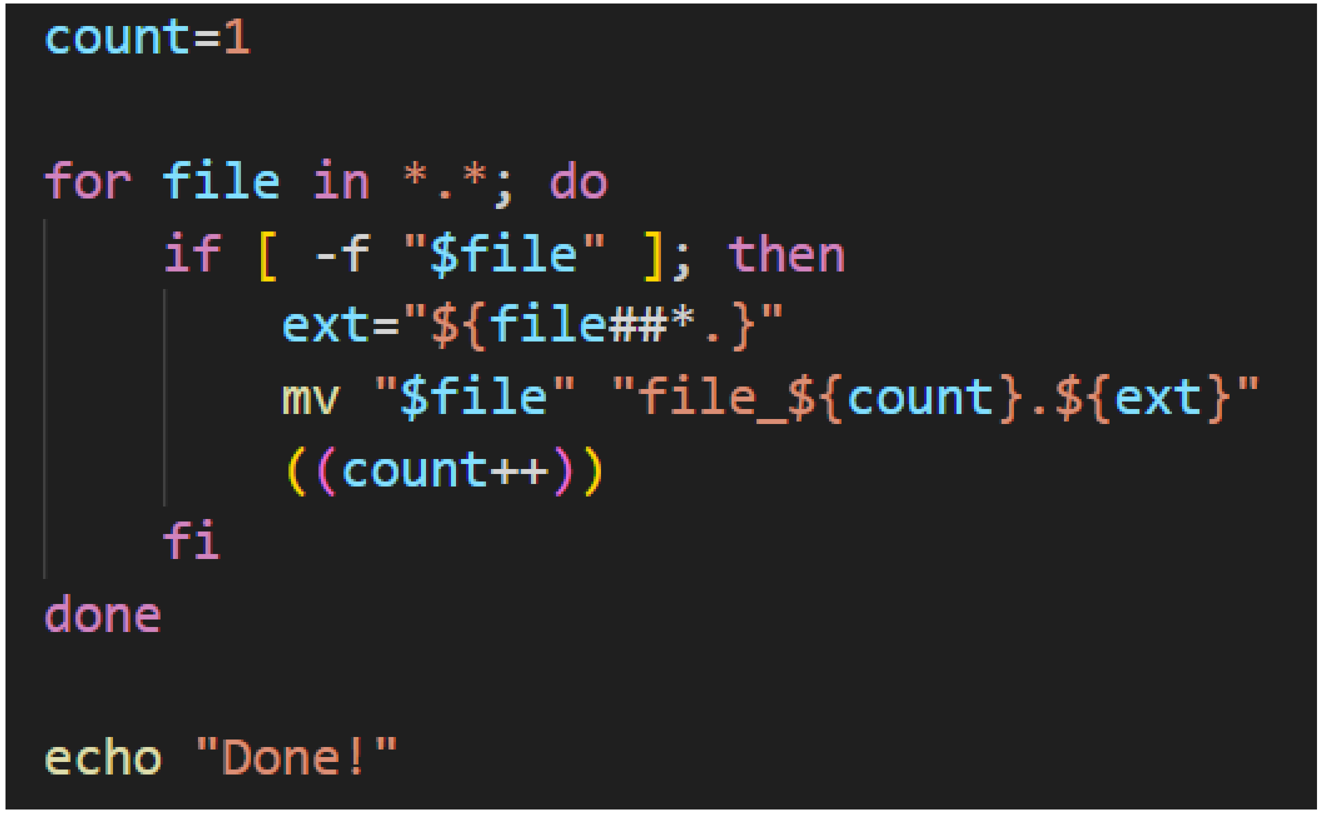
- Replace Spaces with Underscores



```bash
for file in *\ *; do
    mv "$file" "${file// /_}"
done

echo "Done!"
```

ACTIVITY 4

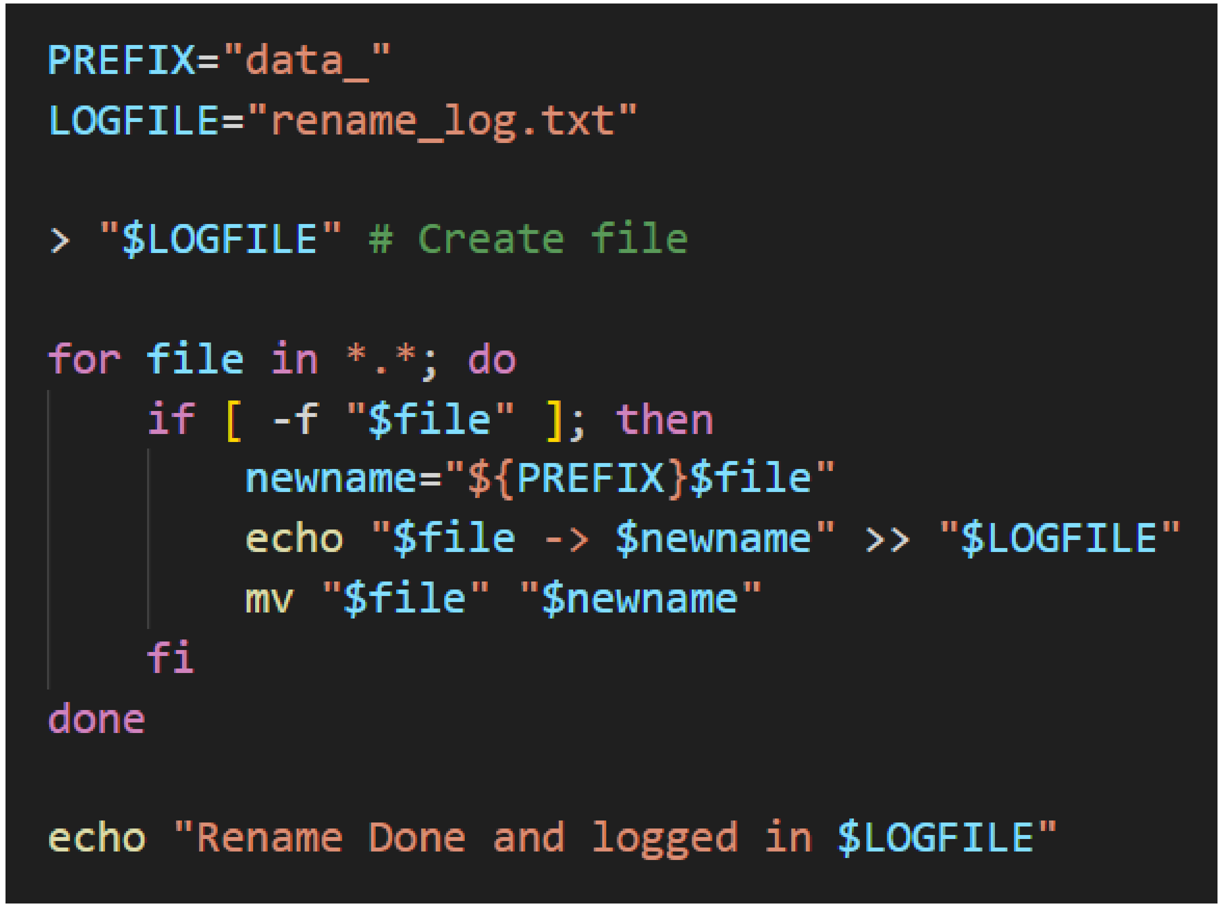
- Number Files in Sequential Order



```bash
count=1

for file in *.*; do
    if [ -f "$file" ]; then
        ext="${file##*.}"
        mv "$file" "file_${count}.${ext}"
        ((count++))
    fi
done

echo "Done!"
```

ACTIVITY 5

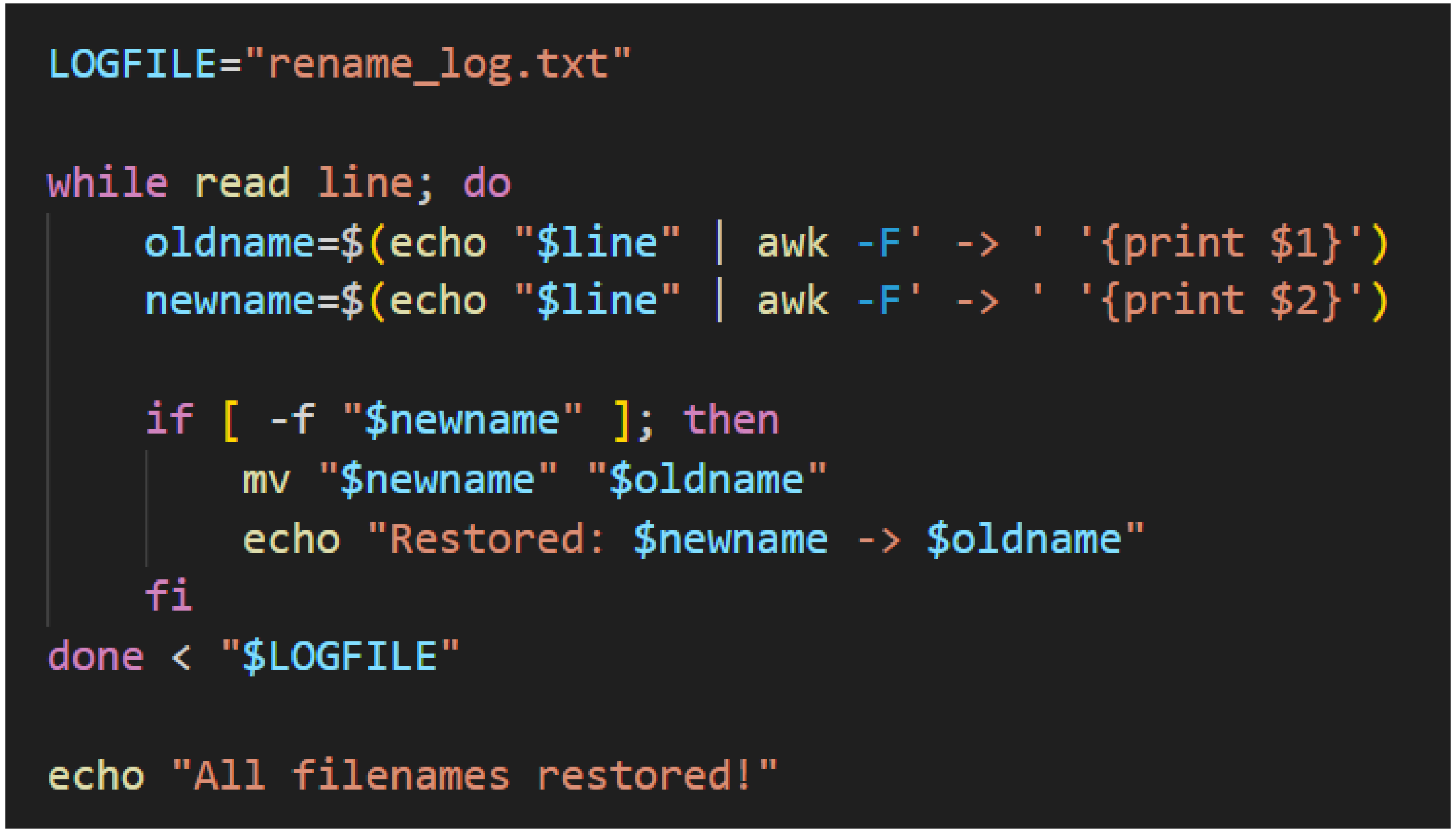
- Rename Files and Generate a Log



```bash
PREFIX="data_"
LOGFILE="rename_log.txt"

> "$LOGFILE" # Create file

for file in *.*; do
    if [ -f "$file" ]; then
        newname="${PREFIX}$file"
        echo "$file -> $newname" >> "$LOGFILE"
        mv "$file" "$newname"
    fi
done

echo "Rename Done and logged in $LOGFILE"
```

ACTIVITY 6

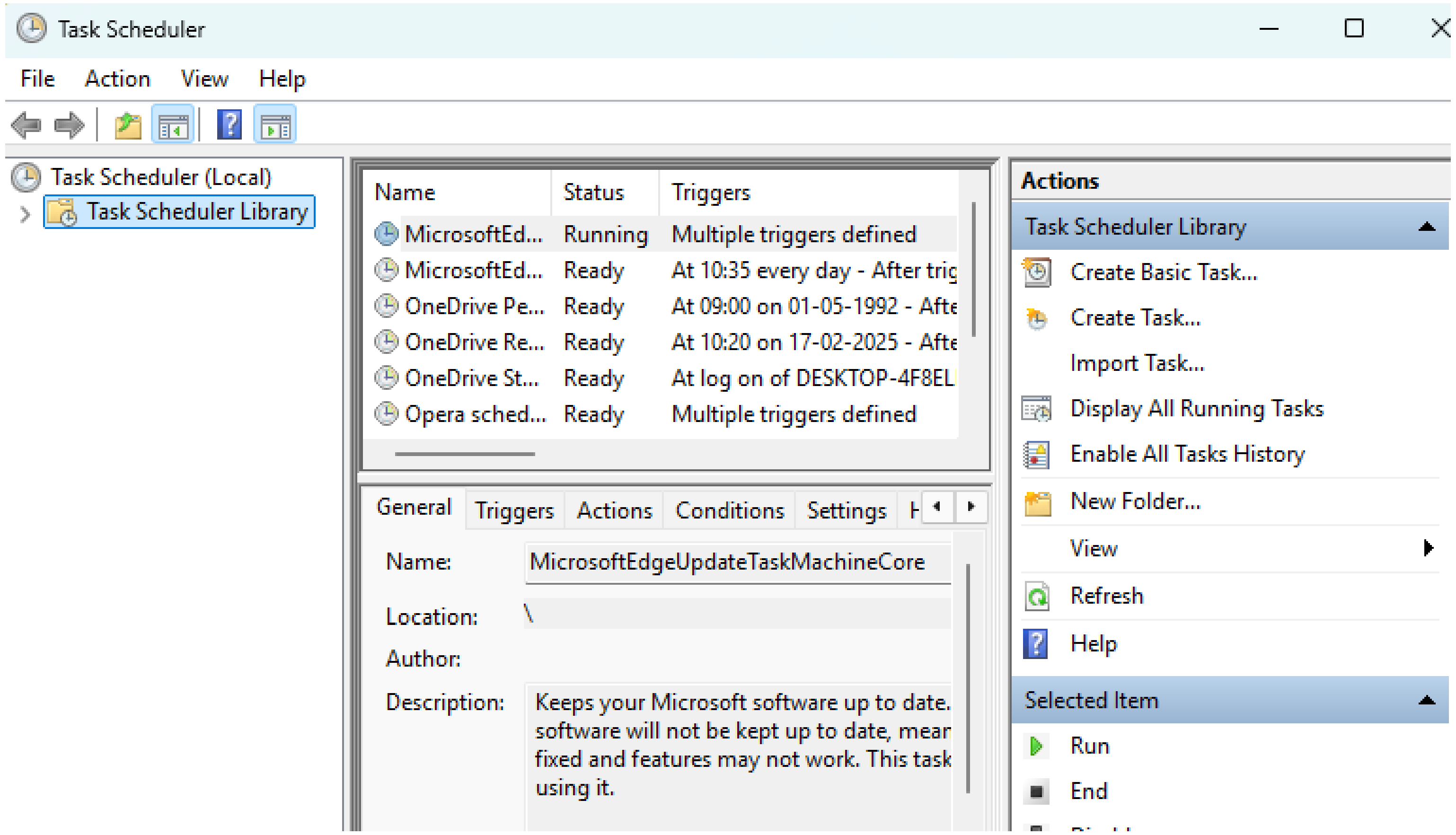
- Restore Original Filenames Using the Log



```
LOGFILE="rename_log.txt"

while read line; do
    oldname=$(echo "$line" | awk -F' -> ' '{print $1}')
    newname=$(echo "$line" | awk -F' -> ' '{print $2}')

    if [ -f "$newname" ]; then
        mv "$newname" "$oldname"
        echo "Restored: $newname -> $oldname"
    fi
done < "$LOGFILE"

echo "All filenames restored!"
```

RUN ANY SCRIPT AS SCHEDULER

- Open Task Scheduler → Click on Create Basic Task



Task Scheduler

File Action View Help

Task Scheduler (Local)
Task Scheduler Library

Name	Status	Triggers
MicrosoftEd...	Running	Multiple triggers defined
MicrosoftEd...	Ready	At 10:35 every day - After trig
OneDrive Pe...	Ready	At 09:00 on 01-05-1992 - Afte
OneDrive Re...	Ready	At 10:20 on 17-02-2025 - Afte
OneDrive St...	Ready	At log on of DESKTOP-4F8EL
Opera sched...	Ready	Multiple triggers defined

General Triggers Actions Conditions Settings H

Name: MicrosoftEdgeUpdateTaskMachineCore

Location: \

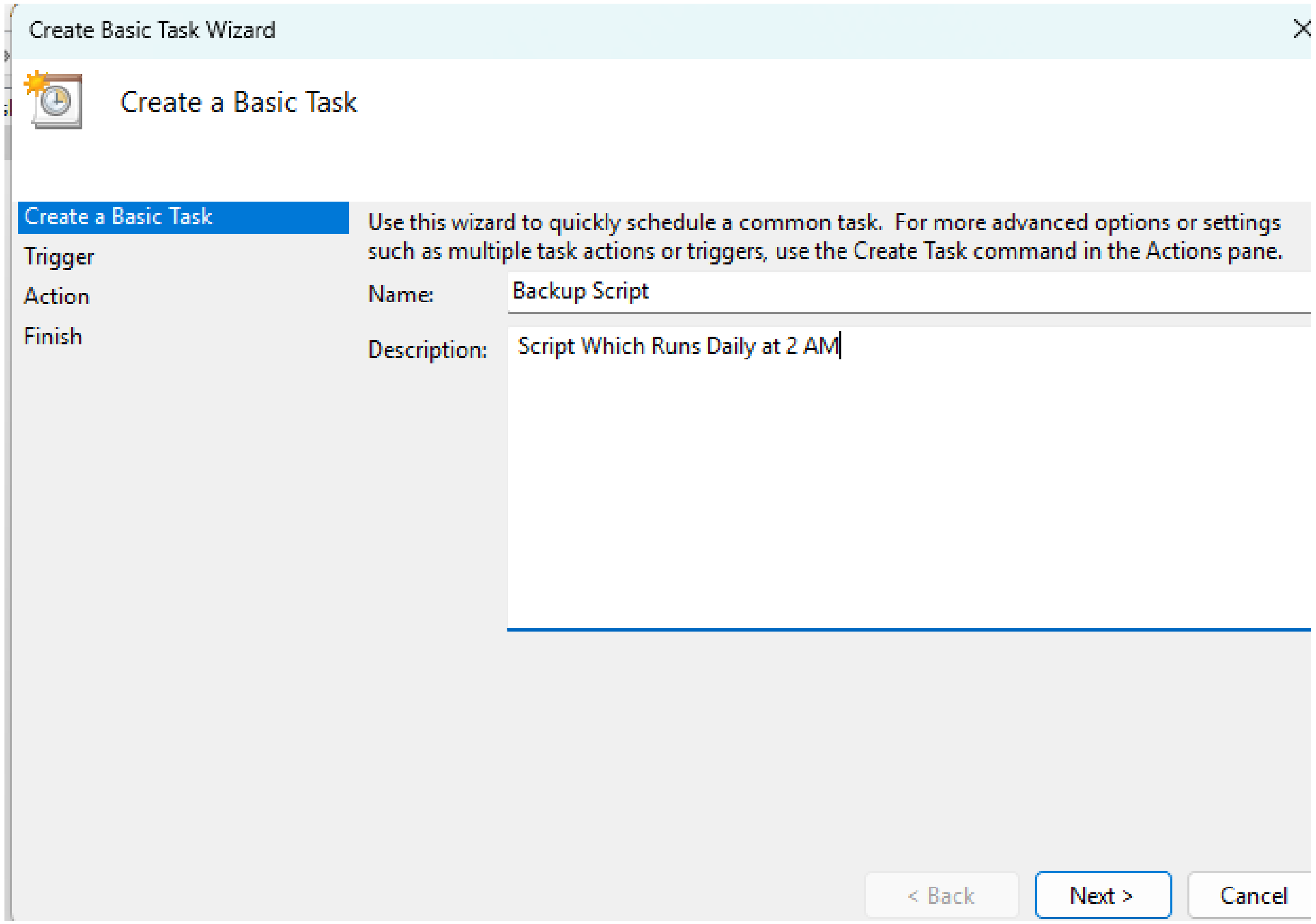
Author:

Description: Keeps your Microsoft software up to date. software will not be kept up to date, mear fixed and features may not work. This task using it.

Actions

Task Scheduler Library
- Create Basic Task...
- Create Task...
- Import Task...
- Display All Running Tasks
- Enable All Tasks History
- New Folder...
- View
- Refresh
- Help

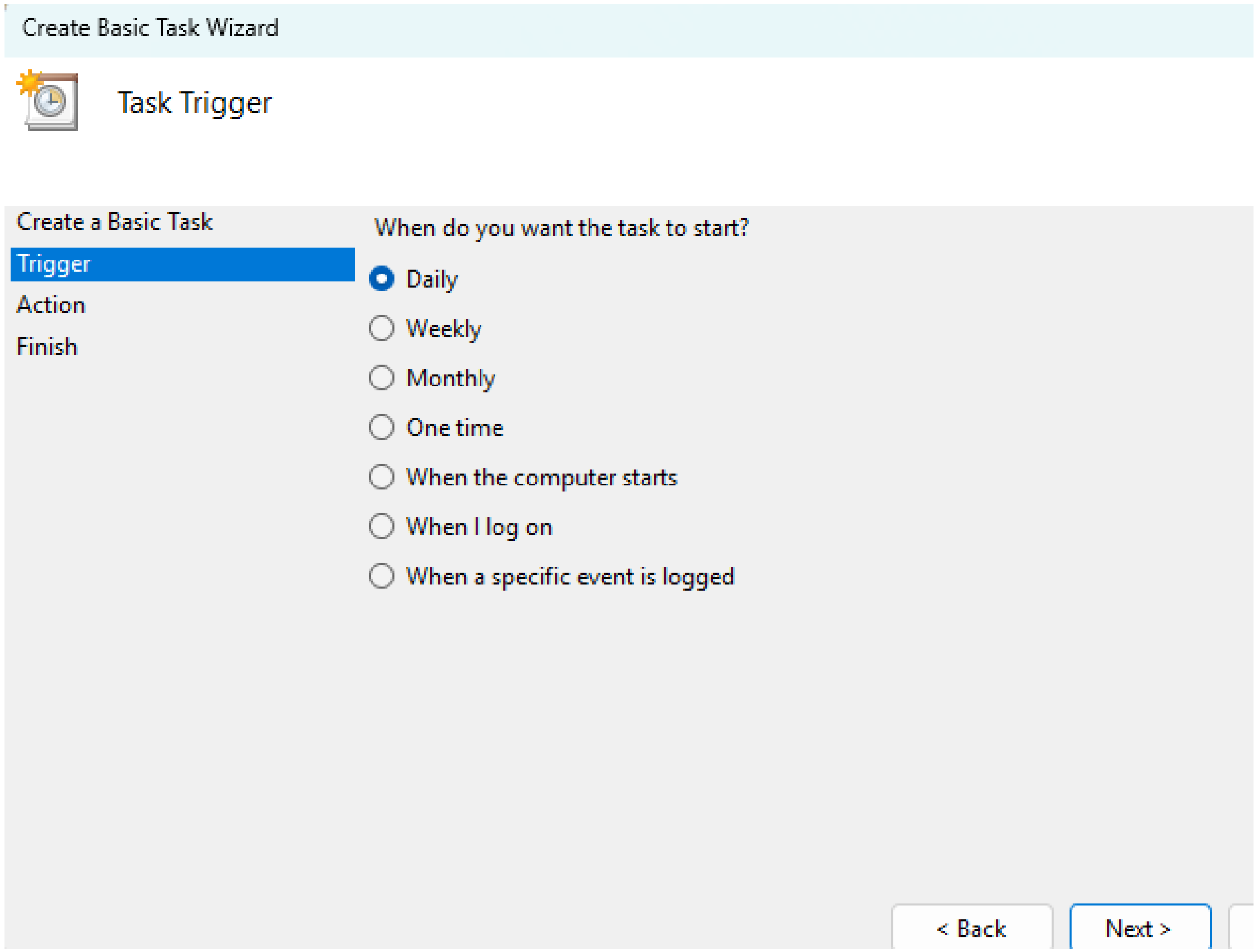
Selected Item
- Run
- End



Use this wizard to quickly schedule a common task. For more advanced options or settings such as multiple task actions or triggers, use the Create Task command in the Actions pane.

Name: Backup Script

Description: Script Which Runs Daily at 2 AM

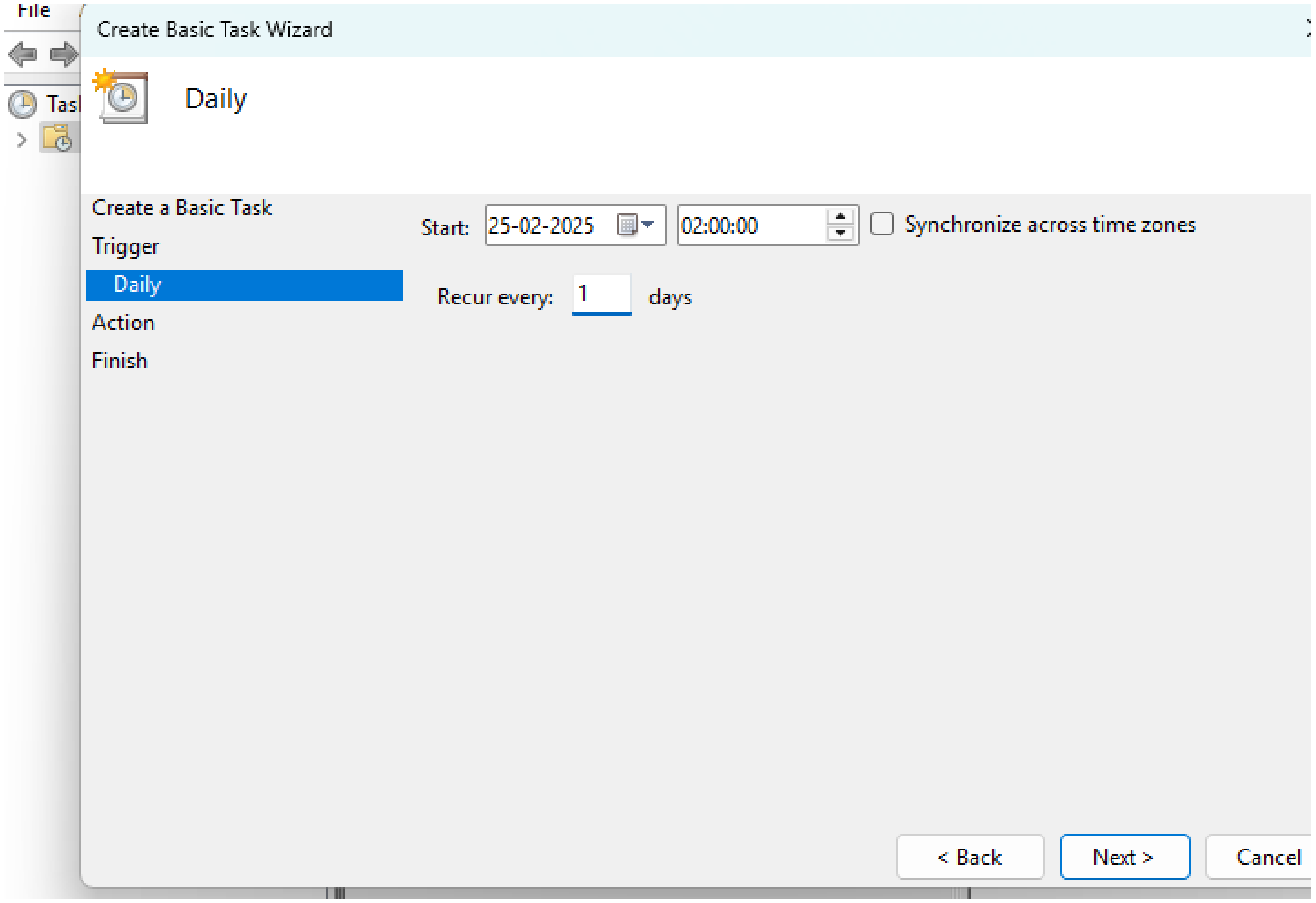

Create Basic Task Wizard

Task Trigger

Create a Basic Task
Trigger
Action
Finish

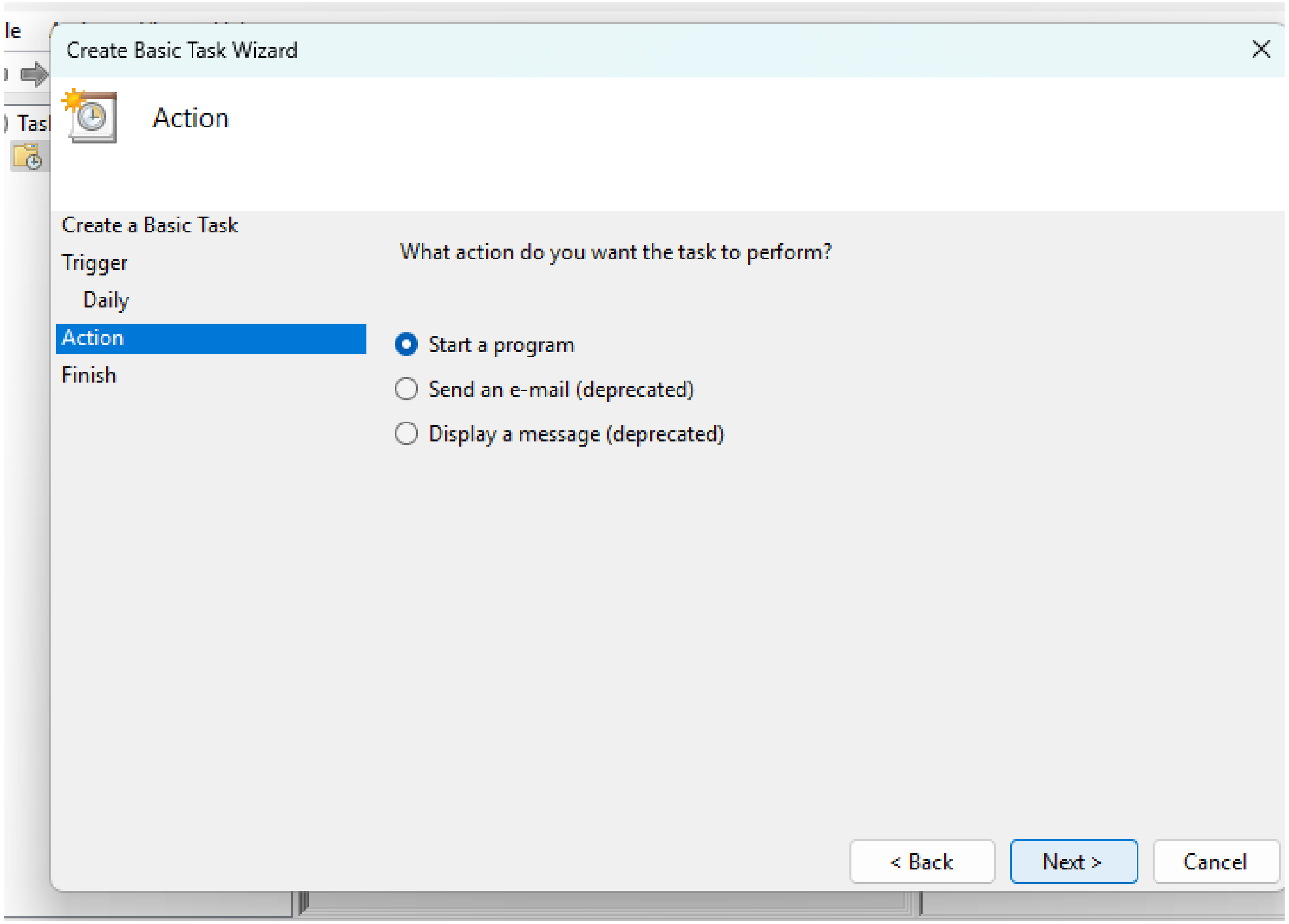
When do you want the task to start?

- [✓ Daily]
- Weekly
- Monthly
- One time
- When the computer starts
- When I log on
- When a specific event is logged



Start: 25-02-2025 02:00:00 ☐ Synchronize across time zones

Recur every: 1 days



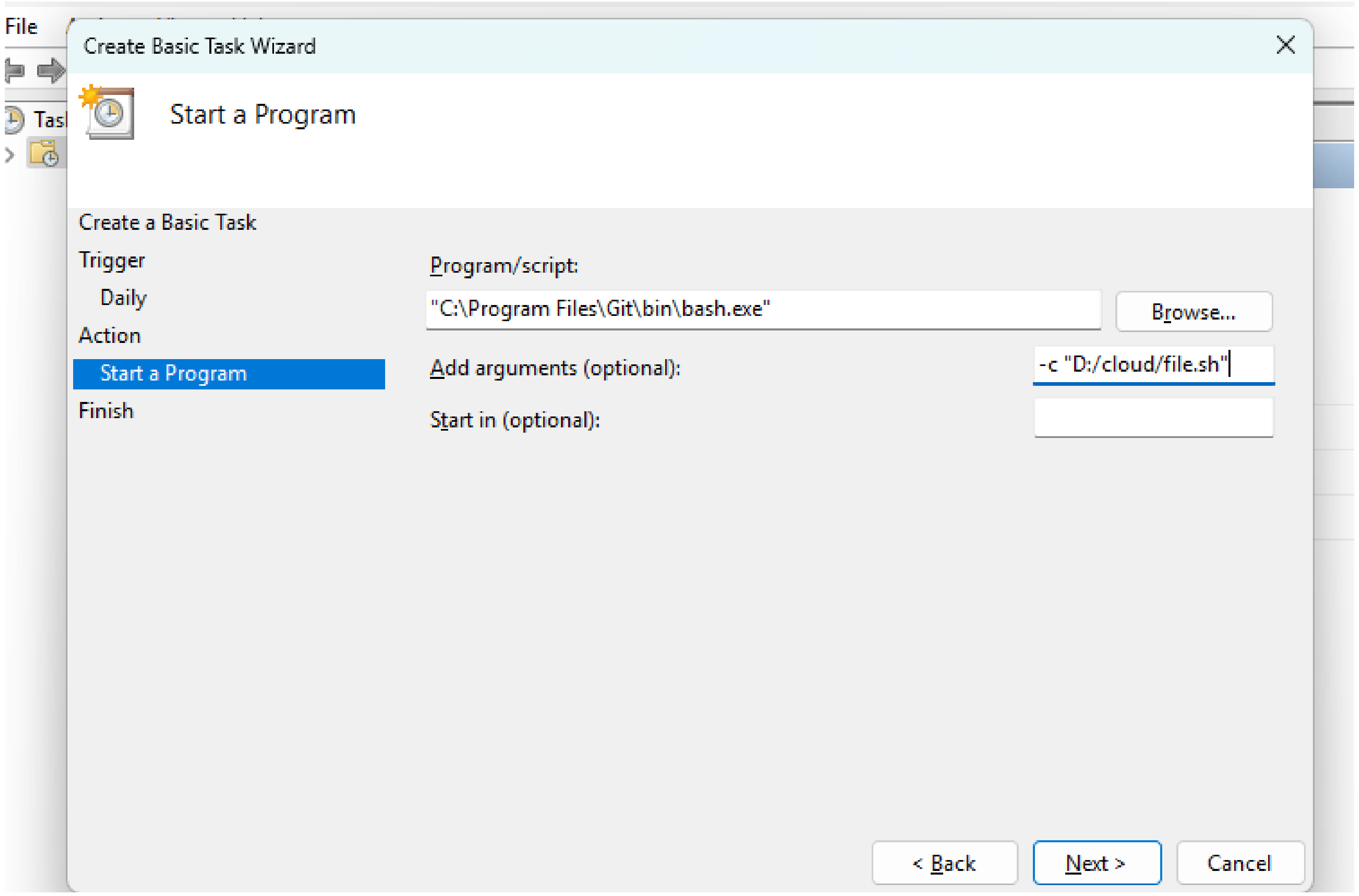
Create Basic Task Wizard

Action

Create a Basic Task
Trigger
Daily
Action
Finish

What action do you want the task to perform?

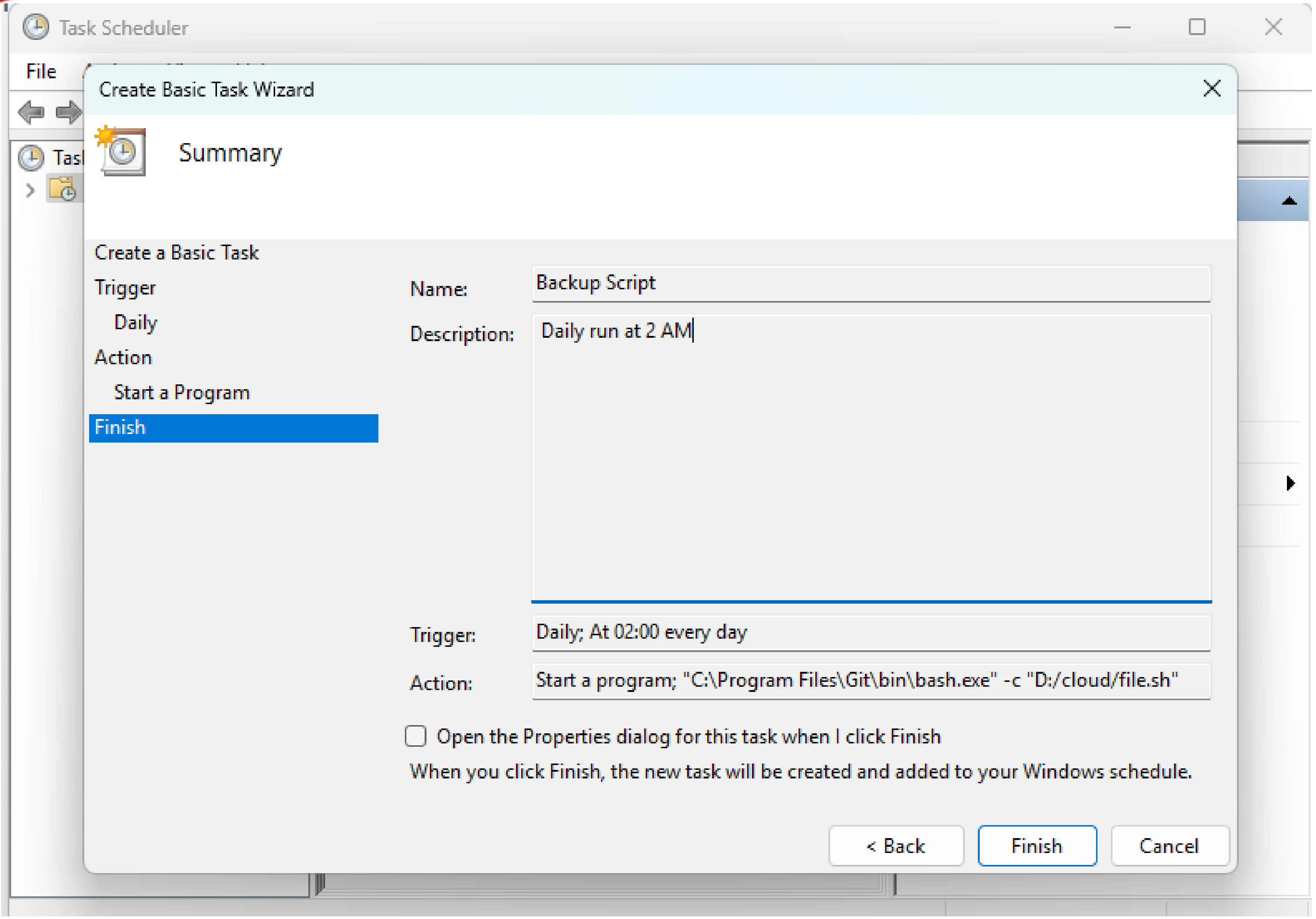
[✓ Start a program]
Send an e-mail (deprecated)
Display a message (deprecated)



Program/script: "C:\Program Files\Git\bin\bash.exe"

Add arguments (optional): -c "D:/cloud/file.sh"

Start in (optional):



Name: Backup Script

Description: Daily run at 2 AM

Trigger: Daily; At 02:00 every day

Action: Start a program; "C:\Program Files\Git\bin\bash.exe" -c "D:/cloud/file.sh"

☐ Open the Properties dialog for this task when I click Finish

When you click Finish, the new task will be created and added to your Windows schedule.

- This Task will Run daily at given time.
- For temporary test you can select task and click on run
- It will run in background and you can see the backup created.