**LAB 1 Objective:** Practice Linux terminal commands to manage files and directories and document the process

**Step 1: Create Directory and Text File**

1. Open terminal and go to Desktop:  
   cd ~/Desktop
2. Create new directory:  
   mkdir myLab
3. Enter the directory:  
   cd myLab
4. Create and edit text file:  
   nano helloWorld.txt
5. Type hello world, then save with Ctrl+O, Enter, and exit with Ctrl+X.

**Step 2: Move and Organize Files** 6. Create another directory:

mkdir library

1. Move text file into new directory:  
   mv helloWorld.txt library/

Return to Desktop and remove old directory:  
cd ~/Desktop

1. rm -r myLab

**Step 3: Document Commands** 9. Go to the library directory:

cd ~/Desktop/library

1. Create documentation file:  
   nano lab\_commands.txt

Write the commands used and describe each. Example:  
cd ~/Desktop # Navigate to Desktop

mkdir myLab # Create 'myLab' directory

1. ...
2. Save and exit using Ctrl+O, Enter, then Ctrl+X.

**Expected Output:**

* A folder library on the Desktop
  + Contains helloWorld.txt and lab\_commands.txt

**Optional Challenge:** Try using these alternatives:

echo "hello world" > helloWorld.txt

cat > helloWorld.txt

**Notes:**

* **Use Tab for auto-complete.**
* **Use Ctrl+L to clear terminal screen.**
* **Use man [command] to see manual for a command.**

### **Linux Command Line Cheat Sheet**

| **Command** | **Description** |
| --- | --- |
| **cd [dir]** | **Change directory** |
| **ls** | **List contents of directory** |
| **mkdir [name]** | **Create a new directory** |
| **nano [file]** | **Open a file in nano text editor** |
| **echo [text] > file** | **Write text to a file** |
| **mv [src] [dest]** | **Move or rename a file or directory** |
| **rm -r [dir]** | **Remove directory and contents** |
| **cat [file]** | **Display file content in terminal** |
| **touch [file]** | **Create an empty file** |
| **pwd** | **Print current directory** |
| **cp [src] [dest]** | **Copy files or directories** |
| **grep '[pattern]' [file]** | **Search text in file** |
| **chmod [mode] [file]** | **Change file permissions** |
| **chown [user]:[group]** | **Change file owner** |
| **df -h** | **Show disk space usage** |
| **top** | **Show real-time system processes** |
| **ifconfig or ip a** | **Display network interface configuration** |
| **sudo [command]** | **Run command with administrative privileges** |

**Notes:**

* **Use Tab to auto-complete file and folder names.**
* **Use Ctrl+L to clear the terminal screen.**
* **Use man [command] to view the manual/help for any command.**

**Lab 2 Instructions**

**Step 1: Create Your First Bash Script**

1. Open a terminal and navigate to your Desktop:  
   cd ~/Desktop
2. Create a new directory for your scripts:  
   mkdir bashLab && cd bashLab
3. Create a new script file:  
   nano hello\_script.sh

Type the following in the editor:  
#!/bin/bash

1. echo "Hello, Raspberry Pi!"
2. Save with Ctrl+O, press Enter, then exit with Ctrl+X.
3. Make the script executable:  
   chmod +x hello\_script.sh
4. Run your script:  
   ./hello\_script.sh

**Step 2: Add User Input and Variables**

1. Create another script:  
   nano greet\_user.sh

Enter the following:  
#!/bin/bash

echo "What is your name?"

read name

1. echo "Hello, $name! Welcome to Bash scripting."

Save, make executable, and run:  
chmod +x greet\_user.sh

1. ./greet\_user.sh

**Step 3: Use Conditionals and Loops**

1. Create a new script:  
   nano number\_check.sh

Type the script below:  
#!/bin/bash

echo "Enter a number:"

read num

if [ $num -gt 10 ]; then

echo "That's a big number!"

else

echo "That's a small number."

1. fi

**Step 4: Loop Practice**

1. Create a loop script:  
   nano loop\_example.sh

Enter the following code:  
#!/bin/bash

for i in {1..5}

do

echo "Loop number $i"

1. done

**Expected Output:**

* A folder bashLab with 4 scripts:
  + hello\_script.sh
  + greet\_user.sh
  + number\_check.sh
  + loop\_example.sh

**Bash Scripting Cheat Sheet**

| **Command/Structure** | **Description** |
| --- | --- |
| #!/bin/bash | Declares file as a bash script |
| echo [text] | Prints text to the screen |
| read [var] | Gets user input and stores in var |
| if [ condition ]; then ... fi | Conditional execution |
| [ $var -gt num ] | Checks if var is greater than num |
| for i in {1..5} | Loop from 1 to 5 |
| chmod +x file.sh | Make file executable |
| ./file.sh | Run a bash script |

**Tips:**

* Always start your script with #!/bin/bash
* Use bash -x script.sh to debug line by line
* Use comments (#) to document your code

| **Function** | **Example** |
| --- | --- |
| Output Pin | Pin(15, Pin.OUT) |
| Input Pin | Pin(14, Pin.IN) |
| Pull-up | Pin(14, Pin.IN, Pin.PULL\_UP) |
| Toggle Pin | pin.toggle() |
| Read Pin | pin.value() |
| Write Pin | pin.value(1 or 0) |
| PWM Output | PWM(Pin(15)) |
| Analog Input | ADC(26) |

#### **1. Variables & Data Types**

python

CopyEdit

x = 10 # Integer

pi = 3.14 # Float

name = "Alice" # String

is\_active = True # Boolean

**2. Basic Data Structures**

# List (ordered, mutable)

fruits = ["apple", "banana", "cherry"]

# Tuple (ordered, immutable)

coords = (10, 20)

# Dictionary (key-value pairs)

person = {"name": "Alice", "age": 30}

# Set (unordered, unique elements)

unique\_numbers = {1, 2, 3}

**3. Control Flow**

python

CopyEdit

# If-else(else if =elif)

if x > 5:

print("Greater than 5")

else:

print("5 or less")

# For loop

for fruit in fruits:

print(fruit)

# While loop

count = 0

while count < 5:

print(count)

count += 1

**4. Functions**

python

CopyEdit

def greet(name):

return f"Hello, {name}!"

print(greet("Alice"))

**5. Input & Output**

python

CopyEdit

name = input("Enter your name: ") # User input as string

print("Hello,", name)

**6. File Operations**

python

CopyEdit

# Write to file

with open("file.txt", "w") as f:

f.write("Hello, file!")

# Read from file

with open("file.txt", "r") as f:

content = f.read()

print(content)

**8. Comments**

# This is a single line comment

"""

This is a

multi-line comment or docstring

"""

**9. Common Built-in Functions**

| **Function** | **Description** |
| --- | --- |
| len(obj) | Length of a list, string, etc. |
| type(obj) | Returns the type of an object |
| int(), str(), float() | Type casting |
| range(start, stop, step) | Generate sequence of numbers |
| print() | Print to console |

**10. Import Modules**

python

CopyEdit

import math

print(math.sqrt(16)) # Outputs: 4.0

##############################################

# Lab: Implementing a Continuous Monitoring Loop in Python

# Duration: ~10 minutes

# Objective: Use loops and conditionals to simulate a real-time monitoring system.

##############################################

# 🚀 Task

# Your goal is to write a function called `monitor(threshold)` that continuously

# simulates a monitoring system. If a security breach is detected, the system

# should respond to varying threat levels.

# 🧠 Key Concepts

# - while loops for continuous monitoring

# - if statements for conditional checks

# - for loops for responding to conditions

# - time delays for realism

# 🛠 Step 1: Import required modules

# Hint: You need time delays and random events

import time

import random

# 🛠 Step 2: Define the monitor function

# Hint: Use a `while True` loop to simulate an ongoing system

def monitor(threshold):

print("Monitoring system started... Press Ctrl+C to stop.")

while True:

# 🔄 Simulate whether an alert is active

# Hint: Randomly assign True or False to alert\_active

alert\_active = random.choice([True, False])

print("Alert status:", "ACTIVE" if alert\_active else "Normal")

# 🚨 If the alert is active, respond to threat levels

# Hint: Use a for loop to go from 1 to threshold

if alert\_active:

print("Security breach detected!")

for level in range(1, threshold + 1):

print(f" ➤ Responding to threat level {level}")

time.sleep(0.5) # Pause to simulate processing time

# 🕒 Wait before the next check

time.sleep(1)

# 🧪 Step 3: Run the monitor with a sample threshold

# Hint: Call the function with an example value like 3 or 5

if \_\_name\_\_ == "\_\_main\_\_":

monitor(threshold=3)

# 🛑 Note: This loop runs forever. Use Ctrl+C to stop it manually.

##############################################

# LAB: LED Blinking (First GPIO Project)

# Platform: Raspberry Pi (any model with GPIO)

# Objective: Blink an LED using Python and GPIO

# Duration: ~15–20 minutes

##############################################

# 🧰 Materials Needed

# - 1 x LED (any color)

# - 1 x 330Ω resistor

# - Breadboard

# - 2 x jumper wires

# - Raspberry Pi with Raspbian OS (or Raspberry Pi OS) installed

# ⚙️ Circuit Setup

# Connect the LED as follows:

# 1. Connect the \*\*long leg (anode)\*\* of the LED to GPIO pin \*\*17\*\* through a \*\*330Ω resistor\*\*.

# 2. Connect the \*\*short leg (cathode)\*\* of the LED to a \*\*GND pin\*\* on the Pi.

# (Use GPIO17 = physical pin 11, and GND = pin 6 or 9)

# 🧠 Tip: Always use a resistor with an LED to avoid damaging the GPIO.

# 🛠️ Step 1: Install GPIO Library (if needed)

# If using Raspberry Pi OS, RPi.GPIO is usually preinstalled. But if it’s missing, install it with:

# Run this in terminal:

sudo apt update

sudo apt install python3-rpi.gpio

# 🧪 Step 2: Python Code to Blink the LED

# Save this as led\_blink.py

import RPi.GPIO as GPIO

import time

# Set the GPIO mode

GPIO.setmode(GPIO.BCM) # BCM pin-numbering

GPIO.setwarnings(False)

# Set GPIO 17 as output

LED\_PIN = 17

GPIO.setup(LED\_PIN, GPIO.OUT)

# Blink the LED

print("Starting LED blink. Press Ctrl+C to stop.")

try:

while True:

GPIO.output(LED\_PIN, GPIO.HIGH) # LED ON

time.sleep(1) # Wait 1 second

GPIO.output(LED\_PIN, GPIO.LOW) # LED OFF

time.sleep(1) # Wait 1 second

except KeyboardInterrupt:

print("\nExiting...")

# Clean up GPIO to avoid warnings next time

finally:

GPIO.cleanup()