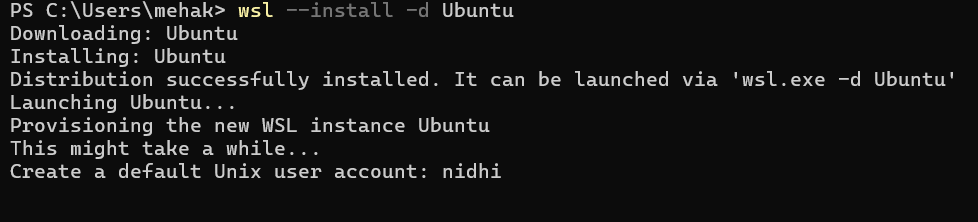
Hands-on Practice

**Prerequisites:**

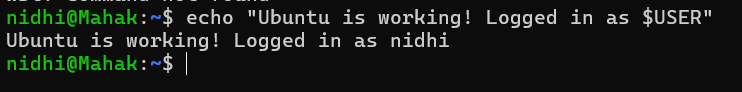
Install Ubuntu for shell scripting.

Here are the steps to install Ubuntu in Windows:

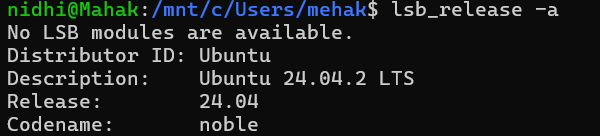
* Open your terminal
* Run the command: wsl --install -d Ubuntu



* Now, create your default user account and set up a new password.
* Ubuntu is successfully installed on your computer.
* To check if Ubuntu is working, you can write this command:



* To check the version, you can execute the command (lsb\_release -a)



1. Basic script for automation

Create a simple Bash automation script that gathers and displays essential system information using basic shell scripting techniques. The goal is to automate the process of retrieving data such as hostname, current date/time, uptime, memory usage, and disk space without using loops, conditionals, or advanced logic, making it ideal for beginners learning shell scripting.

Steps:

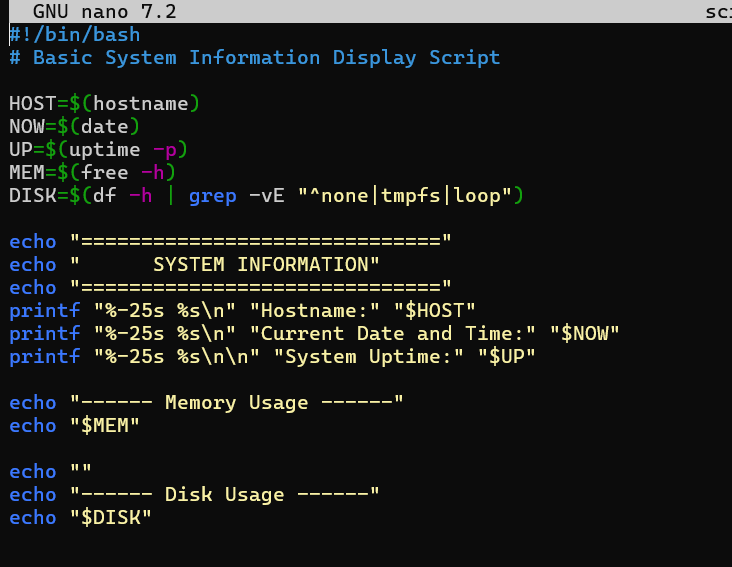
1. Open the Ubuntu terminal.

Press **Ctrl + Alt + T** to launch the Ubuntu terminal.

1. Create a file named script.sh



1. Write the commands inside the script.



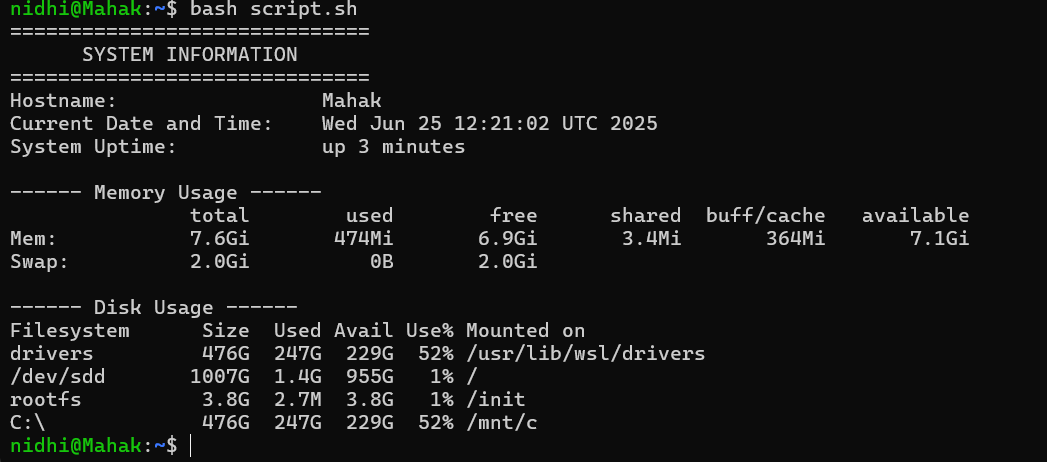
After writing the script, click CTRL + O and ENTER.

And then, to exit the script, click CTRL + X.

1. To make the file script.sh executable, run this command:



1. Run the script



Topics covered:

* Shebang declaration (#!/bin/bash)
* Variable assignment using VAR=value
* Command substitution using $(...)
* Use of basic system commands:
  + hostname—get system hostname
  + date – get current date and time
  + uptime—check system uptime
  + free -h – view memory usage
  + df -h – check disk space usage

2. Script for system health monitoring

Problem statement:

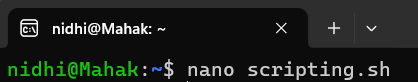
Design a system health monitoring script in Bash that checks real-time CPU and memory usage against predefined threshold limits. The script should calculate usage using standard Linux tools (top, free, awk, bc) and trigger alerts when resource usage exceeds configurable thresholds. It must implement conditional logic to compare current usage with thresholds and provide clear, formatted output to help users detect potential performance issues promptly.

Steps:

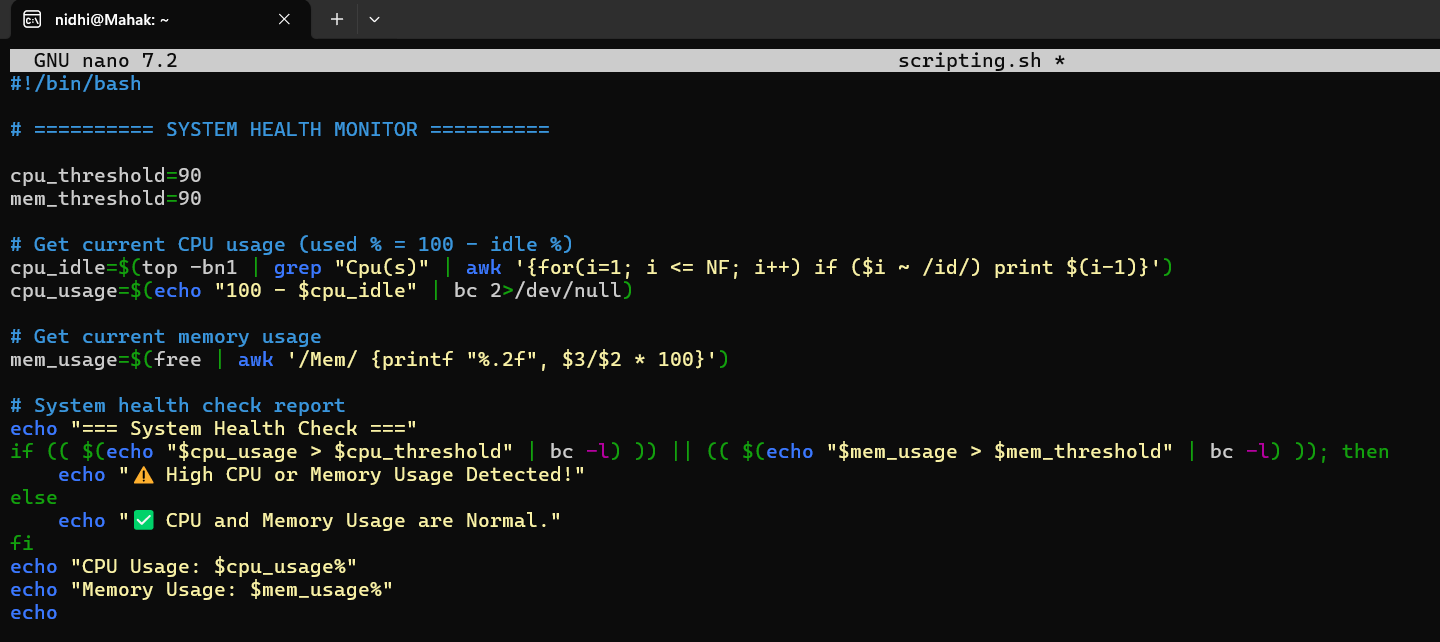
1. Open the Ubuntu terminal.

Press **Ctrl + Alt + T** to launch the Ubuntu terminal.

1. Create the Script File



1. Inside, add your script content.



When done, save nano with CTRL + O and press ENTER, then exit with CTRL + X.

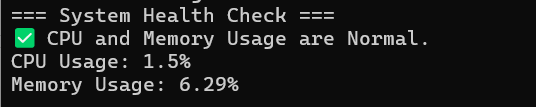
1. Make the Script Executable



1. Run the Script



OUTPUT



Topics covered:

* It defines threshold variables (cpu\_threshold, mem\_threshold) for flexible alert limits.
* It calculates CPU usage using top, grep, awk, and bc to determine active usage.
* It computes memory usage using free piped into awk, formatted to two decimal places.
* It implements threshold-based conditional logic via Bash if and bc, issuing alerts when either CPU or memory exceeds the limit.

3. Backup automation script

Problem Statement:

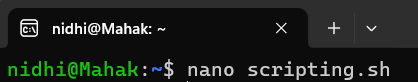
Create a backup automation script in Bash that reliably compresses and stores data from a specified source directory into a timestamped archive. The script should use configurable variables for flexibility, ensure safe creation of required directories, and organize the logic into a modular function for reusability. It must generate compressed backups using tar -czf and support unique file naming using timestamps to prevent overwrites and enable historical tracking.

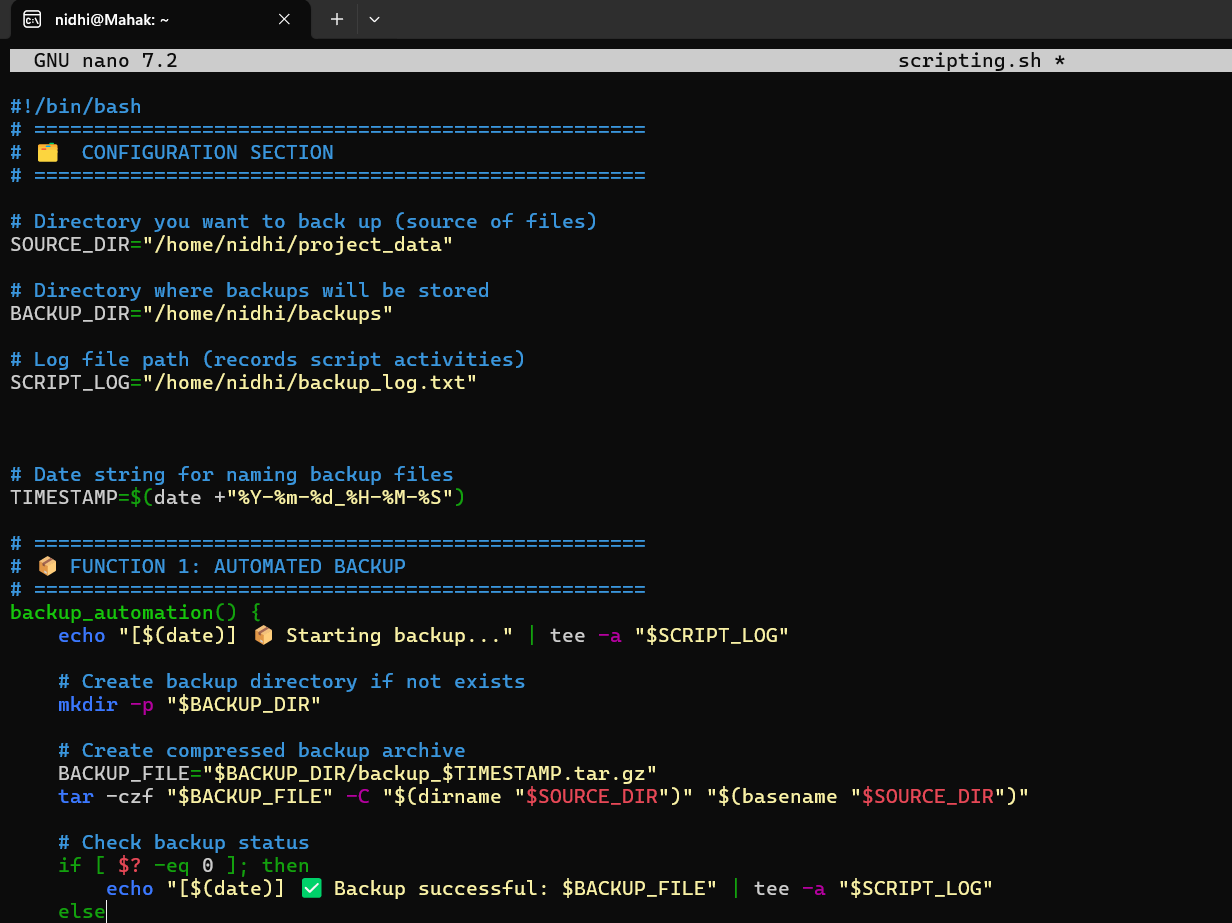
Steps:

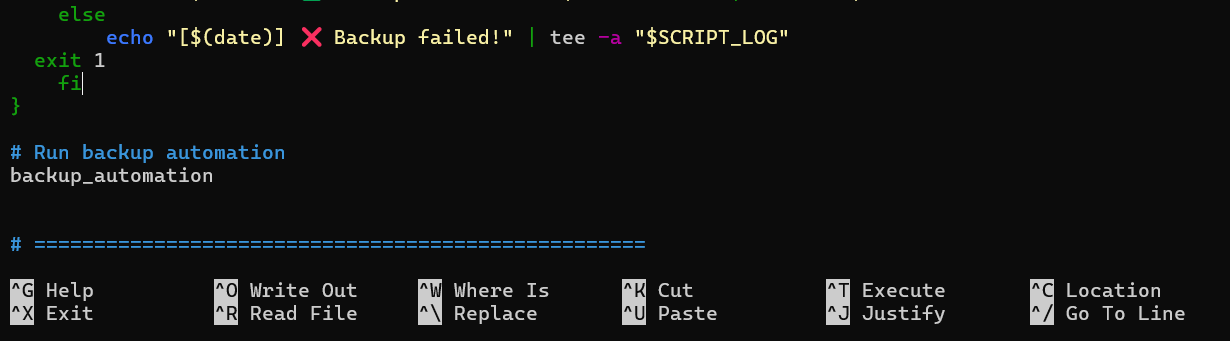
1. Open the Ubuntu terminal.

Press **Ctrl + Alt + T** to launch the Ubuntu terminal.

1. Edit the existing file



1. Inside, add your backup automation script.



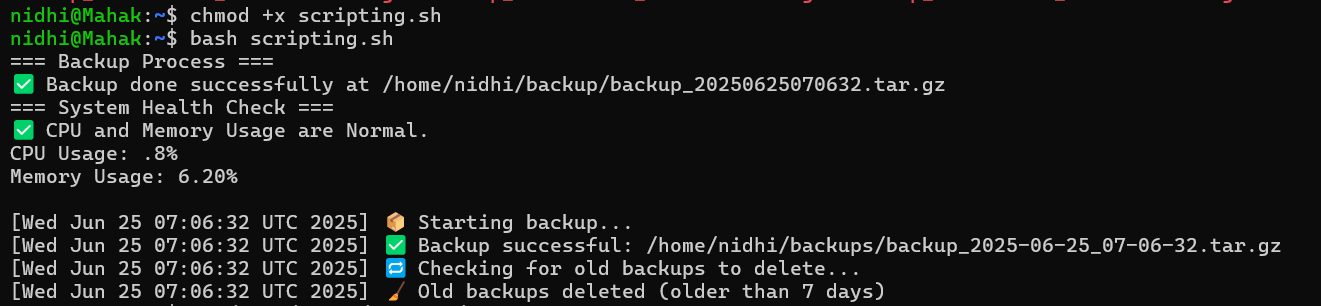
When done, save nano with CTRL + O and press ENTER, then exit with CTRL + X.

1. Make the Script Executable:

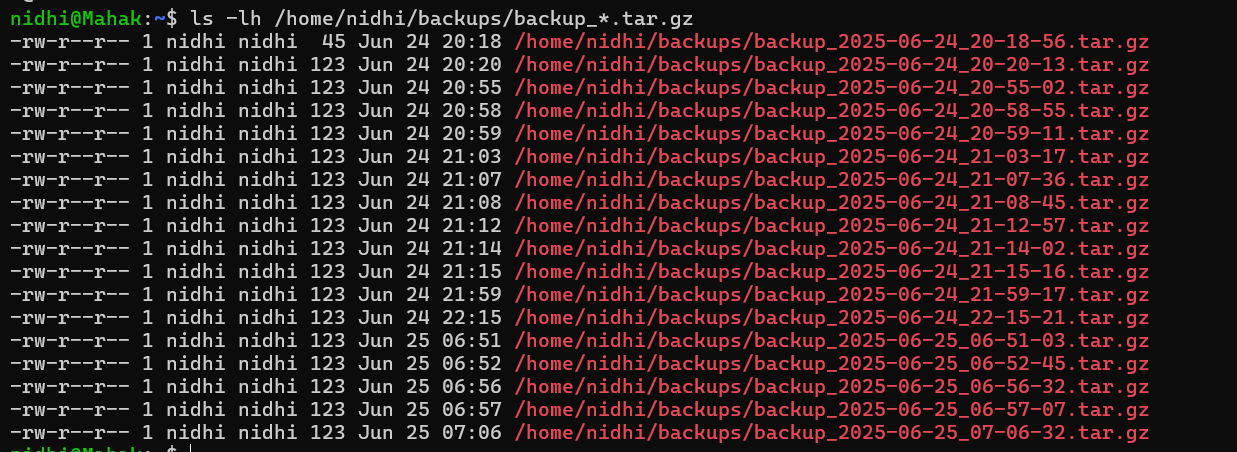


1. Run the Script:

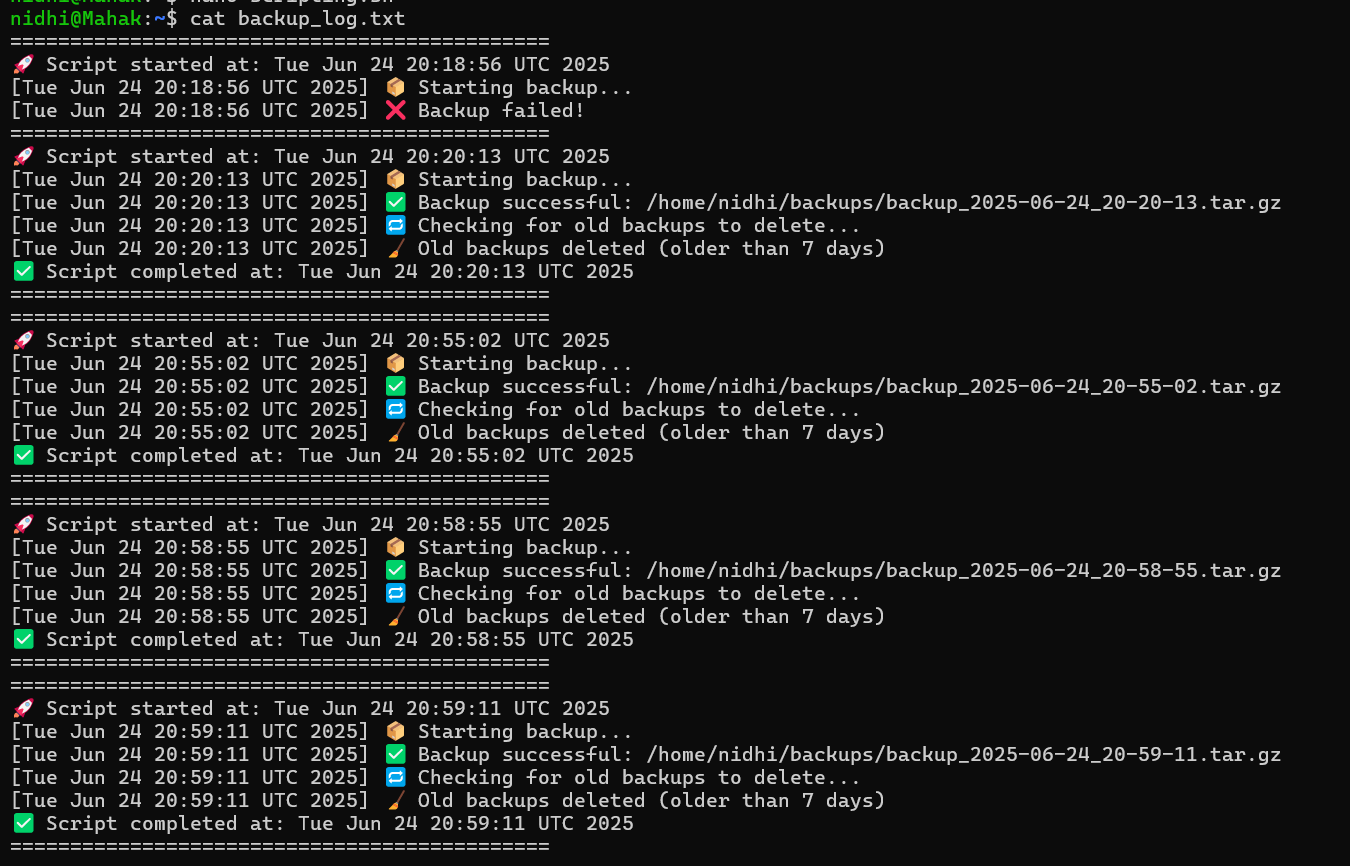


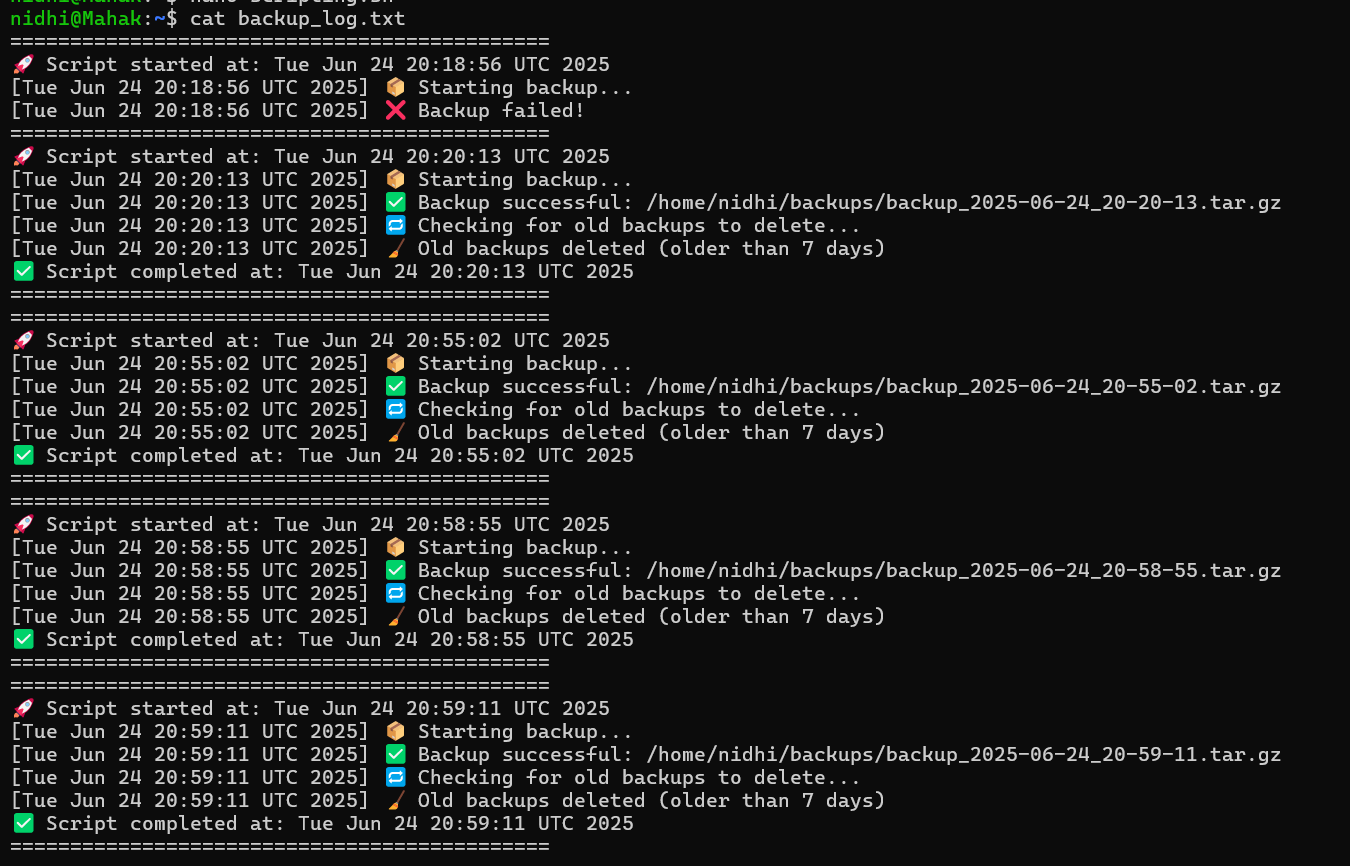


1. Verify that backups are being created as expected and see their timestamps at a glance. I had run the script many times, so it is showing all the backups created.



1. To display the entire file at once:





Topic covered:

* Configurable variables (SOURCE\_DIR, BACKUP\_DIR, SCRIPT\_LOG, TIMESTAMP) for flexibility and unique file naming
* Modular function (backup\_automation()) for clean structure and reusability
* Safe directory creation using mkdir -p to avoid errors
* Compressed archive generation with tar -czf using dirname/basename for accurate path handling
* Dual output logging via tee, writing to both console and log file.
* Exit-status check ($?) to detect success or failure of the backup

4. Log rotation utility

Problem Statement:

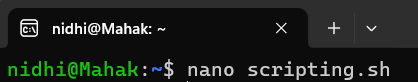
Develop a log rotation utility script in Bash that automatically deletes old backup or log files based on a configurable retention period. The script should use a modular function to encapsulate the cleanup logic, ensure safe deletion with proper error checking, and provide dual logging (to both the console and a log file) with timestamps to ensure transparency, traceability, and maintainability.

Steps:

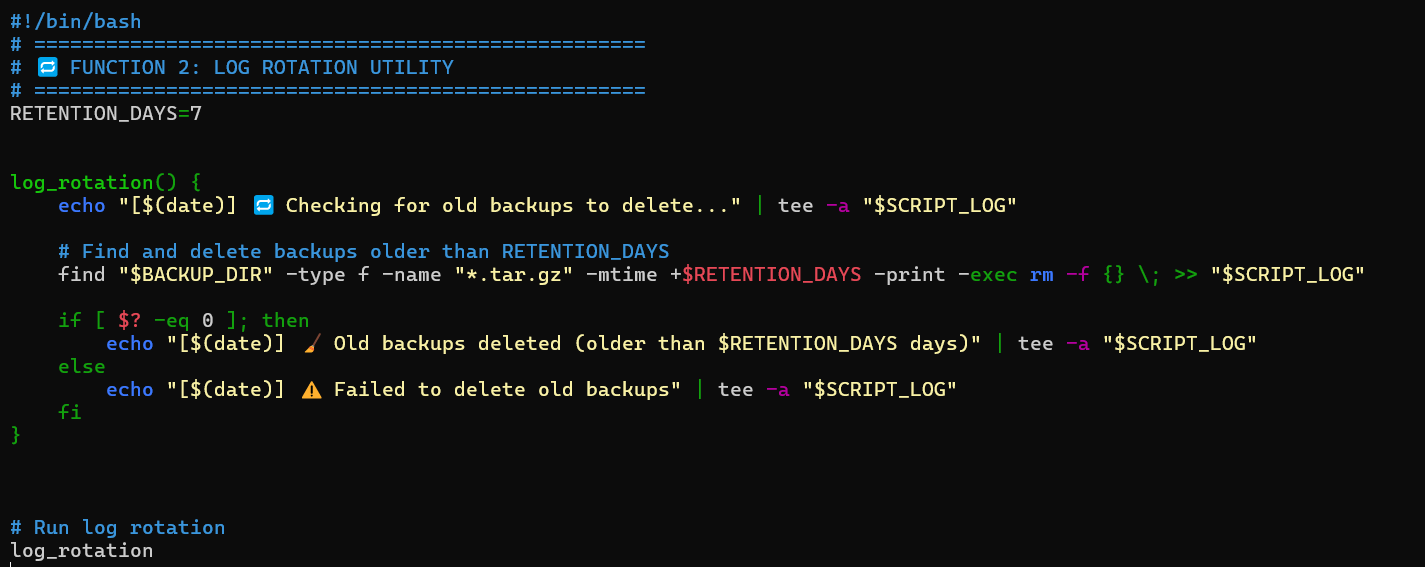
1. Open the Ubuntu terminal.

Press **Ctrl + Alt + T** to launch the Ubuntu terminal.

1. Edit the existing file



1. Inside, add your script

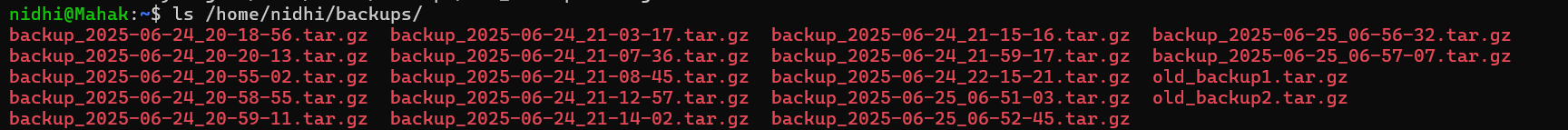


When done, save nano with CTRL + O and press ENTER then exit with CTRL + X.

1. Make the Script Executable:



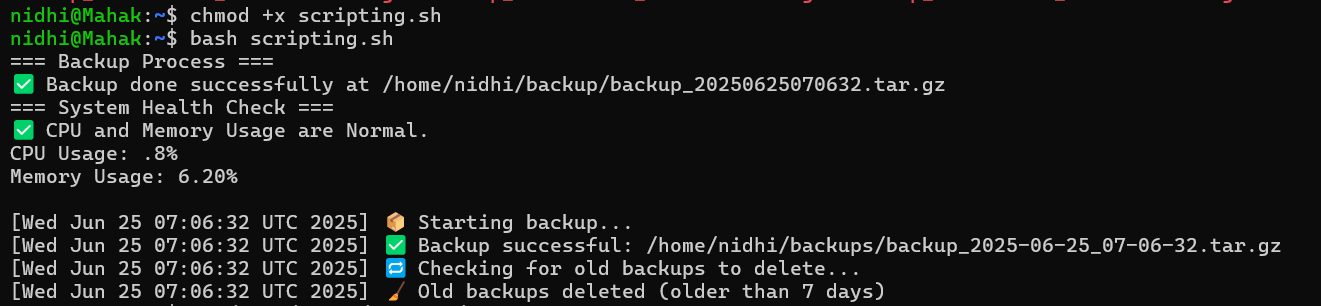
1. Check the file present inside backups:



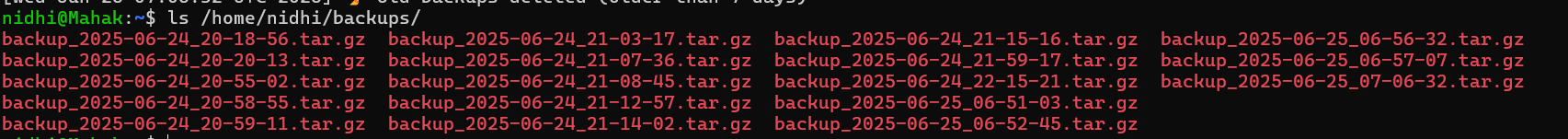
1. Run the Script:



OUTPUT:



1. To verify if the old backups are deleted, list the files in backups



The files that are older than 7 days are deleted.

Topics covered:

* Configurable RETENTION\_DAYS=7 for how long backups/logs are kept
* In the modular log\_rotation() function, cleanup logic is encapsulated.
* To find and remove old backup files that are older than the retention threshold, use find "$BACKUP\_DIR" -type f -name "\*.tar.gz" -mtime +$RETENTION\_DAYS -exec rm -f {} \;
* Safe cleanup using an exit-status check ($?) to record a successful or unsuccessful tee
* For transparency, dual logging output (both console and log file) with timestamps

Debugging and optimizing the existing script

Problem statement:

In modern computing environments, managing system performance and ensuring data backups are vital for system reliability and disaster recovery. Manually monitoring CPU and memory usage or creating backups can be error-prone and inefficient.

This Bash script automates and optimizes the existing script for

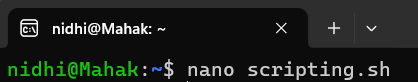
* Real-time system health monitoring (CPU and memory)
* Scheduled directory backups with timestamped archives
* Log management and old file cleanup using log rotation

Steps:

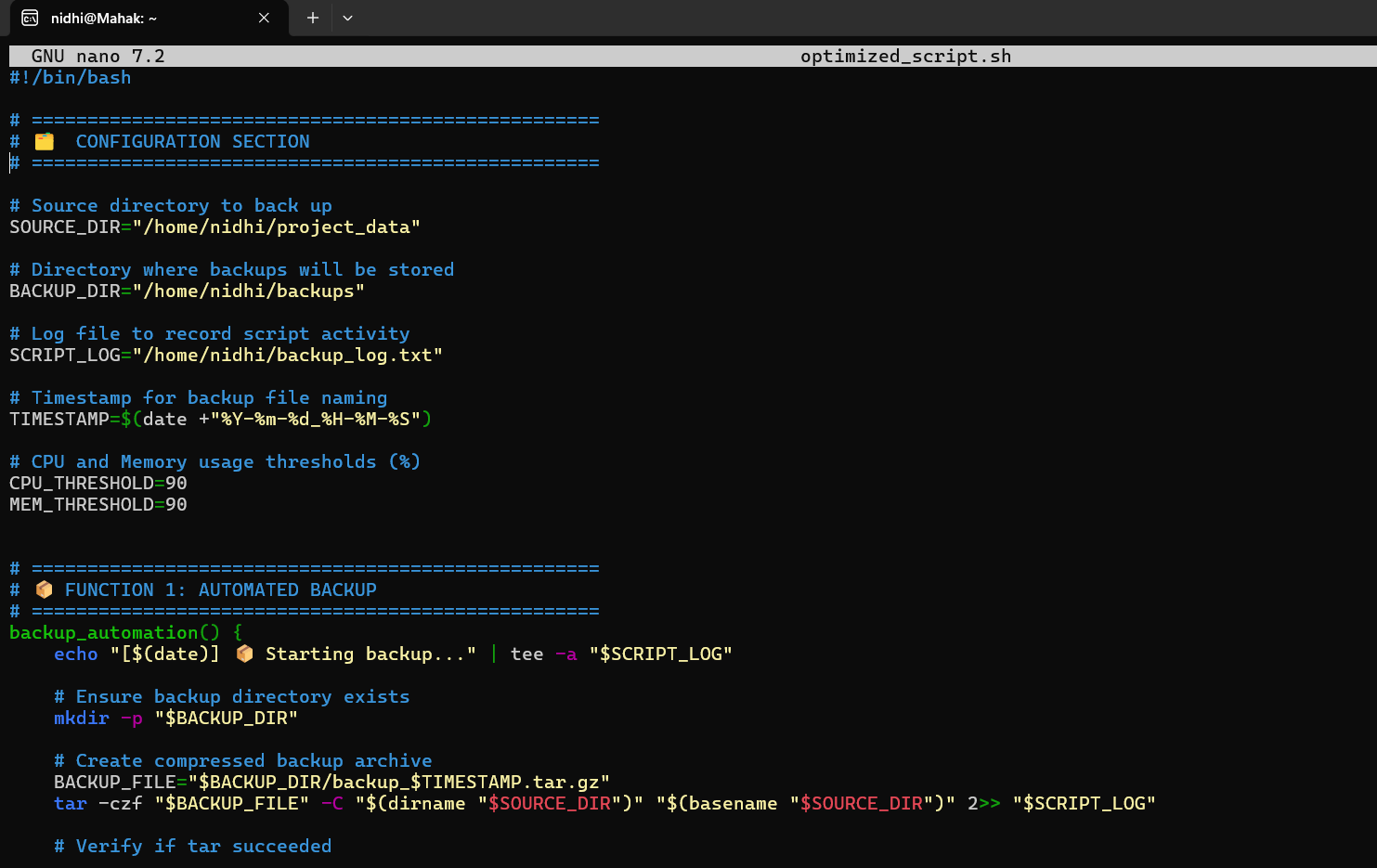
1. Open the Ubuntu terminal.

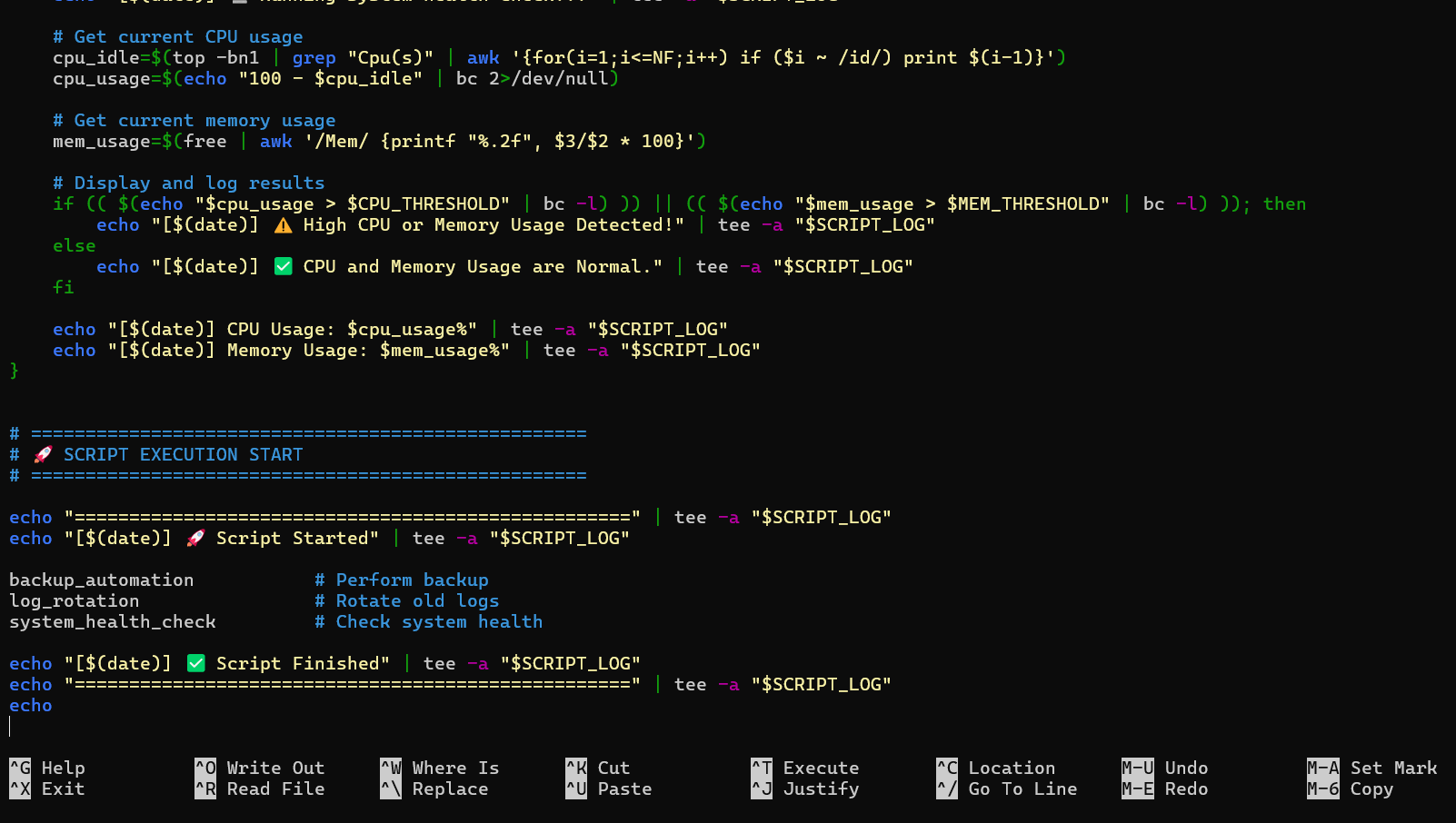
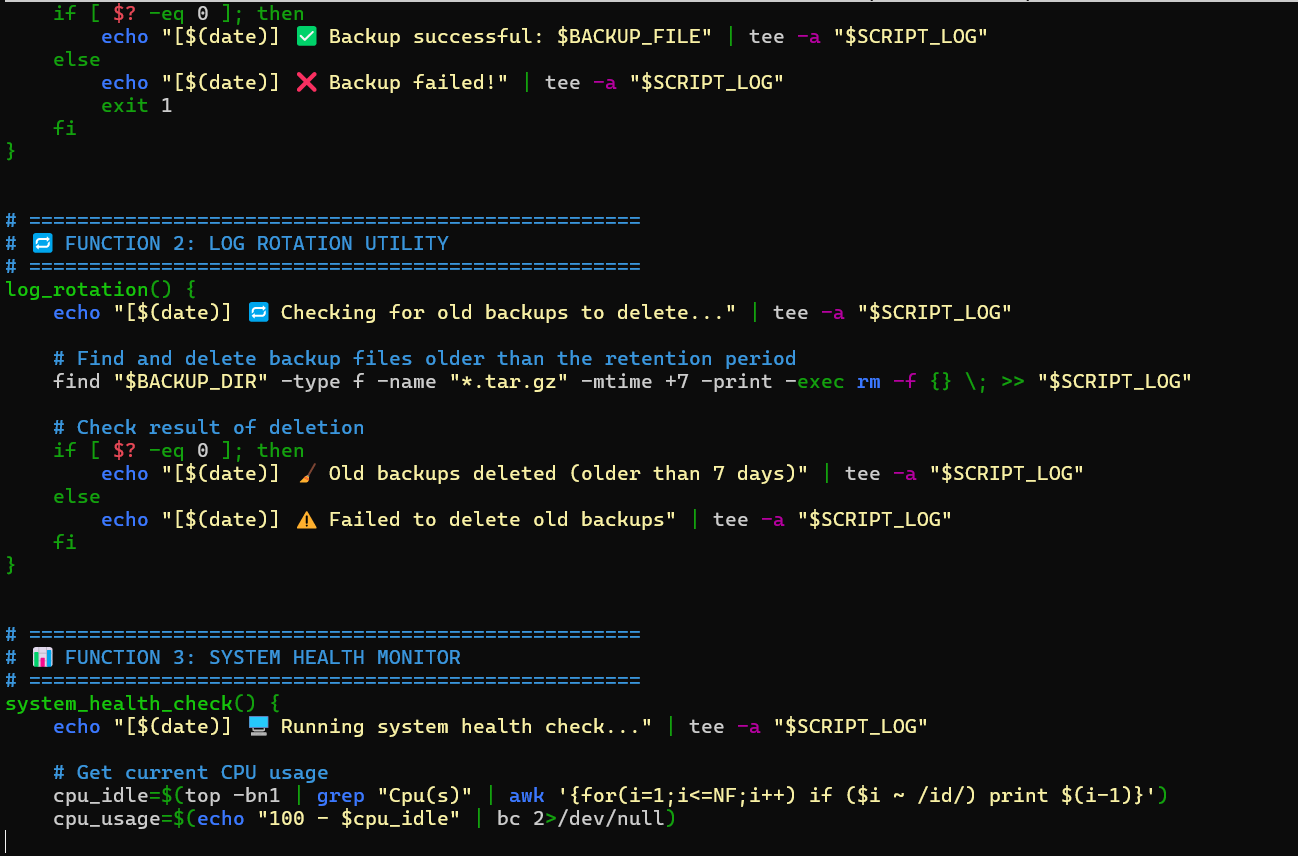
Press **Ctrl + Alt + T** to launch the Ubuntu terminal.

1. Go to the existing script ([scripting.sh](http://scripting.sh))



1. Debug and optimize the existing script.





After finishing, use CTRL + O to save Nano, ENTER, and CTRL + X to exit.

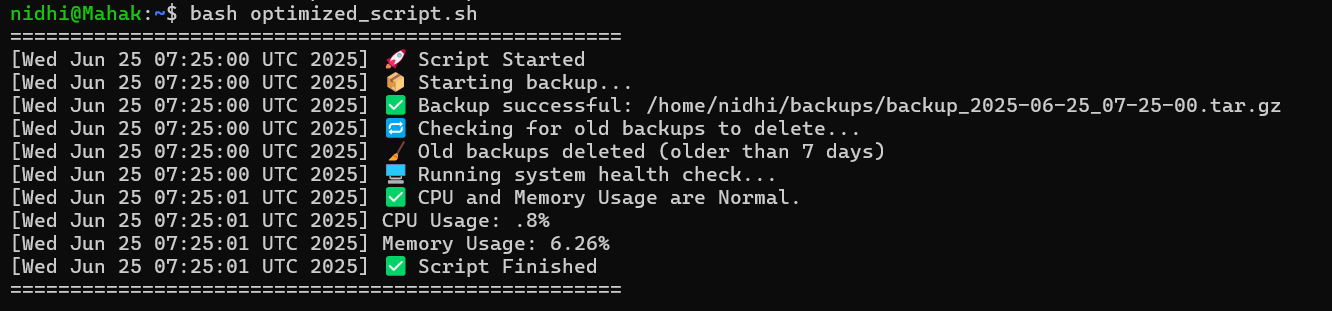
1. Make the Script Executable:



5. Run the Script:



OUTPUT:



Topic covered:

* Shell Scripting: Used variables, functions, conditionals, and formatted output with echo and printf.
* System Monitoring: Retrieved CPU and memory usage using top, free, and awk.
* Automation: Automated backup creation using tar and log rotation using find.
* Backup Management: Created timestamped .tar.gz archives and added success/failure status checks.
* Log Management: Logged all actions using tee and implemented auto-deletion of old backups with find -mtime.
* File Operations: Handled directory creation with mkdir -p, archive generation with tar, and cleanup using rm -f.
* Timestamping: Generated dynamic filenames using date +"%Y-%m-%d\_%H-%M-%S" to avoid overwriting.
* Error Handling: Implemented conditional checks, exit codes, and float comparisons with bc -l to ensure reliability.

**Conclusion:**

This documentation has been developed to provide a clear, hands-on understanding of shell scripting fundamentals and their practical applications. Through the creation of basic scripts, a system health monitor, a backup automation tool, and a log rotation utility, learners gain exposure to essential scripting concepts used in real-world Linux environments. Additionally, the process of debugging and optimizing an existing script reinforces best practices in script design, error handling, and resource management.

Overall, this project serves as a comprehensive learning module that bridges theoretical knowledge with practical implementation—equipping students with the skills required to automate routine tasks, enhance system efficiency, and apply shell scripting confidently in professional settings.