Welcome to the Bash Scripting Challenge - Day 4! This challenge is designed to test our Bash scripting skills and problem-solving abilities in the context of process monitoring and management.

**Scenario 1: Monitoring a specific process**

We have to write a Bash script that efficiently monitors a specific process on a Linux system. The script's primary goal is to ensure the process is always running, and if it unexpectedly stops, it should be restarted automatically.

**Tasks -**

**1: Process Selection:**

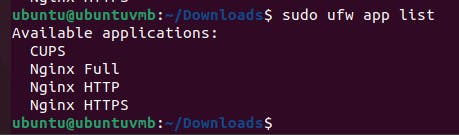
The script should accept a command-line argument to specify the target process to monitor. For example: ./monitor\_process.sh <process\_name>.

Let's use the "nginx" process for this task. So to install nginx in your Linux system, follow these steps -

sudo apt update

sudo apt install nginx

sudo ufw app list

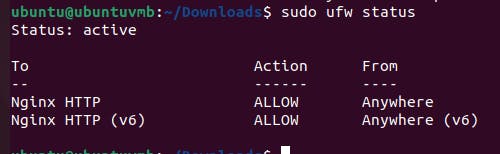


sudo ufw allow "Nginx HTTP"

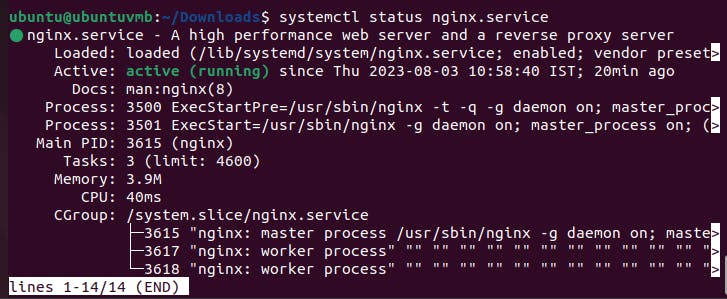
sudo ufw enable



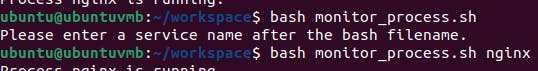
sudo ufw status



systemctl status nginx.service



Now, we have nginx service active in our system, So let's proceed with the task.



We will tell user to enter a service name if he hasn't entered anything along with the script name.

**2: Process Existence Check:**

* Implement a function that checks if the specified process is currently running on the system.
* If the process is running, print a message indicating its presence.



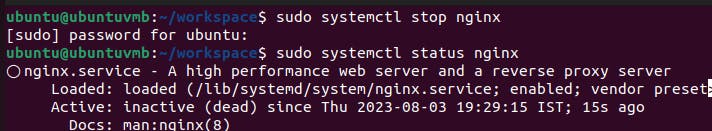


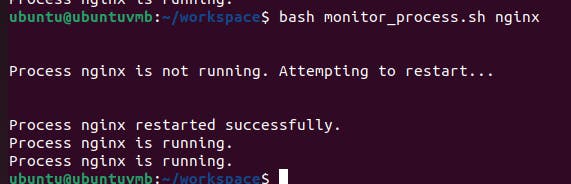


**3: Restarting the Process:**

* If the process is not running, implement a function that attempts to restart the process automatically.
* Print a message indicating the attempt to restart the process.
* Ensure the script does not enter an infinite loop while restarting the process. Limit the number of restart attempts.

Let's stop the process manually -





**4: Documentation:**

* Include clear and concise comments in the script to explain its logic and functionality.
* Write a separate document describing the purpose of the script, how to use it, and any specific considerations.

**Scenario 2: Monitoring Metrics Script with Sleep Mechanism**

This project aims to create a Bash script that monitors system metrics like CPU usage, memory usage, and disk space usage. The script will provide a user-friendly interface, allow continuous monitoring with a specified sleep interval, and extend its capabilities to monitor specific services like Nginx.

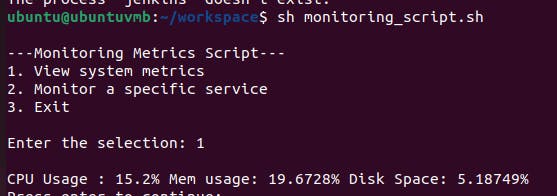
**1: Implementing Basic Metrics Monitoring**

Write a Bash script that monitors the CPU usage, memory usage, and disk space usage of the system. The script should display these metrics in a clear and organized manner, allowing users to interpret the data easily. The script should use the top, free, and df commands to fetch the metrics.



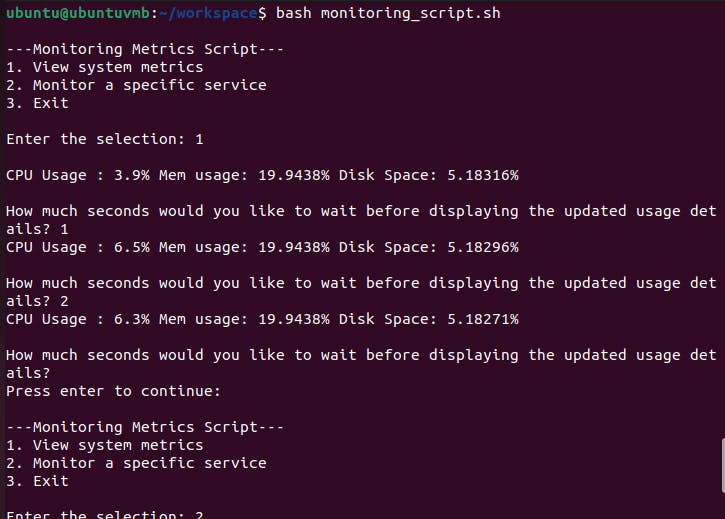
**2: User-Friendly Interface**

Enhance the script by providing a user-friendly interface that allows users to interact with the script through the terminal. Display a simple menu with options to view the metrics and an option to exit the script.



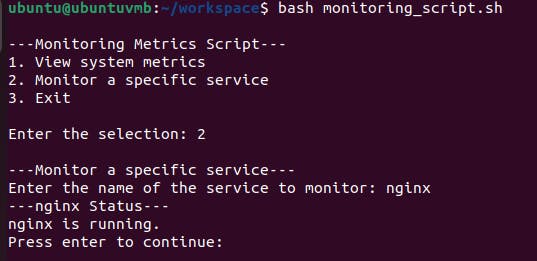
**3: Continuous Monitoring with Sleep**

Introduce a loop in the script to allow continuous monitoring until the user chooses to exit. After displaying the metrics, add a "sleep" mechanism to pause the monitoring for a specified interval before displaying the metrics again. Allow the user to specify the sleep interval.

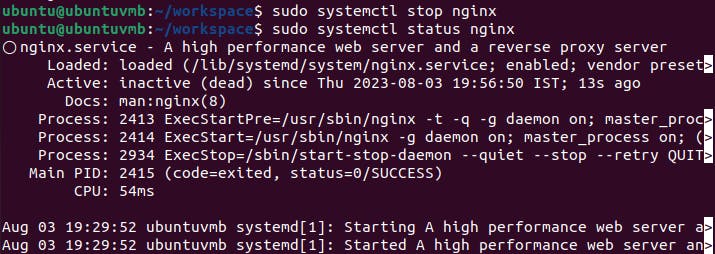


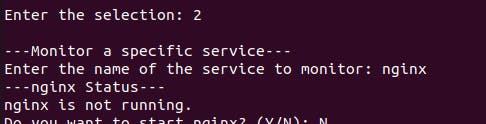
**4: Monitoring a Specific Service (e.g., Nginx)**

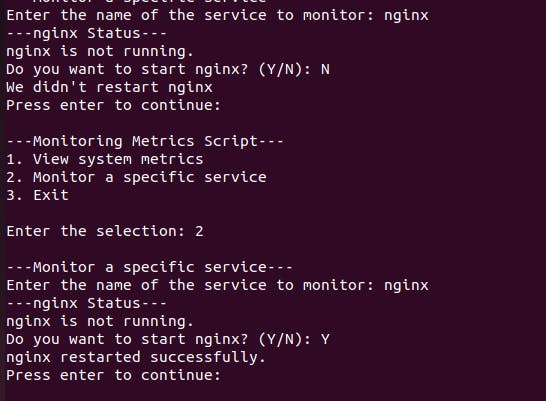
Extend the script to monitor a specific service like Nginx. Check if the service is running and display its status. If it is not running, provide an option for the user to start the service. Use the systemctl or appropriate command to check and control the service.



Now, let's stop it manually-







**5: Allow User to Choose a Different Service**

Modify the script to give the user the option to monitor a different service of their choice. Prompt the user to enter the name of the service they want to monitor, and display its status accordingly.

A computer screen shot of a program

Description automatically generated



**6: Error Handling**

Implement error handling in the script to handle scenarios where commands fail or inputs are invalid. Display meaningful error messages to guide users on what went wrong and how to fix it.

