My Server Performance Monitoring Script - Assignment Report

Introduction

For this assignment, I was tasked with creating a server monitoring script that shows important system performance information. After some trial and error (especially with choosing the right environment), I successfully created a bash script that monitors CPU usage, memory, disk space, running processes, and system information.

What I Wanted to Achieve

My goal was to create a simple script that could give me a quick overview of my system's health. I wanted it to show:

- How much CPU is being used
- Memory usage and availability
- Disk space information
- Which processes are using the most resources
- Basic system information like uptime and logged-in users

My Script Development Process

Initial Challenges with Git Bash

At first, I tried to develop and run my script using Git Bash because I thought it would work the same as a Linux terminal. This turned out to be a mistake! I ran into several issues:

- Commands like `free` and `ps` with Linux options didn't work properly
- Some system monitoring commands were completely missing
- The output format was different than expected
- I kept getting "command not found" errors

I spent quite a bit of time trying to figure out why my script wasn't working, thinking I had made syntax errors.

Switching to WSL - The Game Changer

After struggling with Git Bash, I decided to try WSL (Windows Subsystem for Linux). This made all the difference! Suddenly all my commands worked perfectly. WSL provided a real Linux environment where all the system monitoring tools were available and worked as expected.

My Final Script Explained

Here's the script I created and what each part does:

The Script Header

```bash
#!/bin/bash
echo "===== Server Performance Stats ====="
```

I started with the shebang line to tell the system to use bash, and added a nice header to make the output look organized.

CPU Usage Monitoring

```bash
echo "CPU Usage:"
top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4 "%"}'
```

This part was tricky to figure out. I used `top` command to get process information, then used `grep` to find the CPU line, and `awk` to calculate the total CPU usage by adding user and system usage percentages.

Memory Usage Section

```bash
echo "Memory Usage:"
free -m | grep "Mem" | awk '{printf "Used: %sMB (%.2f%%) | Free: %sMB (%.2f%%)\n", $3, $3*100/$2, $4, $4*100/$2 }'
```

For memory monitoring, I used the `free` command with `-m` flag to show results in megabytes. The `awk` command here calculates percentages and formats the output nicely.

Disk Usage Information

```bash
echo "Disk Usage:"
df --total -h | grep 'total' | awk '{print "Used: " $3 " | Free: " $4 " | Usage: " $5}'
```

The `df` command shows disk usage, and I used `--total` flag to get overall disk usage instead of individual partitions.

Top Process Lists

```bash
echo "Top 5 Processes by CPU:"
ps -eo pid,comm,pcpu --sort=-pcpu | head -n 6

echo "Top 5 Processes by Memory:"
ps -eo pid,comm,pmem --sort=-pmem | head -n 6
```

These sections show which programs are using the most resources. The `ps` command with custom formatting and sorting was really useful here.

System Information

```bash
echo "Uptime:"
uptime -p

echo "OS Version:"
lsb_release -d

echo "Load Average:"
uptime | awk -F'load average:' '{ print $2 }'

echo "Logged in Users:"
who

echo "Failed Login Attempts:"
grep "Failed password" /var/log/auth.log | wc -l
```

I added these sections to give more context about the system status, including security information with failed login attempts.

How I Created and Tested the Script

Step 1: Writing the Script

I used the `nano` text editor in WSL to create my script file:

```bash
nano server_monitor.sh
```

Step 2: Making it Executable

I had to give the script execute permissions:

```bash
chmod +x server_monitor.sh
```

Step 3: Testing and Running

I ran the script using:

```bash
./server_monitor.sh
```

What I Learned from My Mistakes

Why Git Bash Failed Me

- Git Bash is primarily designed for Git operations, not full Linux system administration
- Many Linux commands either don't exist or work differently in Git Bash
- It doesn't have access to the same system resources as a real Linux environment
- The silly mistakes I made were actually environment-related, not coding errors

Why WSL Was the Right Choice

- WSL provides a complete Linux environment on Windows
- All standard Linux commands work as expected
- Better access to system resources and log files
- More authentic Linux experience for learning

Challenges I Faced and Solutions

Problem 1: Command Compatibility

Issue: Many commands didn't work in Git Bash **Solution**: Switched to WSL for proper Linux environment

Problem 2: Permission Issues

Issue: Couldn't access some log files **Solution**: Learned about file permissions and when to use `sudo`

Problem 3: Output Formatting

Issue: Getting clean, readable output **Solution**: Used `awk` and `printf` for better formatting

Testing Results

When I run my script, it provides a comprehensive overview of system performance:

- Shows current CPU usage percentage
- Displays memory usage in both MB and percentages
- Reports disk space usage
- Lists top resource-consuming processes
- Provides system uptime and user information
- Shows security-related information

Improvements I Could Make

Looking back at my script, I can think of several improvements:

1. Add error handling for missing commands
2. Save output to a log file with timestamps
3. Add color coding for different alert levels
4. Create a loop to update information in real-time
5. Add network usage monitoring

Conclusion

This assignment taught me valuable lessons about both scripting and choosing the right development environment. My initial struggles with Git Bash led to an important learning experience about the differences between various command-line environments.

The final script successfully meets all the requirements and provides useful system monitoring capabilities. Most importantly, I learned that sometimes what seems like a coding problem is actually an environment issue - switching from Git Bash to WSL solved most of my problems immediately.

This project gave me hands-on experience with bash scripting, Linux commands, text processing with awk and grep, and system administration concepts. I'm now more confident in creating similar monitoring tools and understand the importance of using the right tools for the job.

What I'd Do Differently Next Time

If I were to start this project again, I would:

1. Research the best environment for Linux scripting before starting
2. Test individual commands before combining them in a script
3. Start with a simpler version and gradually add features
4. Add proper error handling from the beginning
5. Document my code better as I write it

This experience showed me that making mistakes is part of the learning process, and sometimes the "silly mistakes" teach us the most valuable lessons!