**System Monitoring with Bash Script and Cron Jobs**

**1. Introduction**

**Bash Scripting**

Bash (Bourne Again Shell) is one of the most widely used command-line shells on Linux. Bash scripting allows automation of repetitive tasks, execution of system commands, monitoring, and handling files or processes. It is especially useful for system administrators and DevOps engineers to manage daily operations efficiently.

**Cron Jobs**

Cron is a time-based job scheduler in Unix/Linux systems. It enables users to run scripts or commands at scheduled intervals automatically. Typical use cases include system monitoring, backups, log rotation, cleanup, and regular maintenance tasks.

**2. Objective**

The goal of this project was to:

1. Create a Bash script (system\_usage.sh) to monitor CPU usage, disk usage, and memory usage.
2. Store the monitoring data into a log file named system\_report.log.
3. Automate the script execution every 2 minutes using cron.
4. Set up another cron job to clear the log file every hour to prevent it from growing too large.

**3. Steps Performed**

### Step 1: Create the Monitoring Script

* A file named system\_usage.sh was created.
* The script contains commands to check CPU, Disk, and Memory usage.
* Example commands inside the script:

#!/bin/bash  
echo "----- System Report: $(date) -----" >> /path/to/system\_report.log  
echo "CPU Usage:" >> /path/to/system\_report.log  
top -bn1 | grep "Cpu(s)" >> /path/to/system\_report.log  
echo "Memory Usage:" >> /path/to/system\_report.log  
free -h >> /path/to/system\_report.log  
echo "Disk Usage:" >> /path/to/system\_report.log  
df -h >> /path/to/system\_report.log  
echo "-----------------------------------" >> /path/to/system\_report.log

**Step 2: Make the Script Executable**

chmod +x system\_usage.sh

**Step 3: Configure Cron Jobs**

1. **Run Monitoring Script Every 2 Minutes**

Edit the crontab using:

crontab -e

Add the line:

\*/2 \* \* \* \* /bin/bash /path/to/system\_usage.sh

1. **Clear Log File Every Hour**

0 \* \* \* \* > /path/to/system\_report.log

5. Sample Log Output  
  
Below is an example of how the log file (system\_report.log) looks after execution:  
  
----- System Report: Wed Sep 10 09:15:01 2025 -----  
CPU Usage:  
%Cpu(s): 3.0 us, 1.0 sy, 0.0 ni, 95.5 id, 0.5 wa, 0.0 hi, 0.0 si, 0.0 st  
Memory Usage:  
 total used free shared buff/cache available  
Mem: 7.7G 2.5G 3.1G 150M 2.1G 5.0G  
Swap: 2.0G 0B 2.0G  
Disk Usage:  
Filesystem Size Used Avail Use% Mounted on  
/dev/sda1 50G 20G 28G 42% /

**4. Issues Faced During Implementation**

1. **File Permission Errors**

* Sometimes the script was not executable due to missing chmod +x.

1. **Path Issues**

* Cron jobs run in a limited environment, so absolute paths (/usr/bin/top, /usr/bin/free, /usr/bin/df) had to be used instead of just command names.

1. **Log File Overwriting**

* Initially, using > instead of >> in the script replaced the entire log file instead of appending.

1. **Cron Job Not Running**

* The cron service was not active in some cases and had to be started with:
* sudo systemctl enable cron
* sudo systemctl start cron

1. **Time Formatting**

* Ensuring each log entry had a timestamp was necessary to differentiate logs.

# 6. Conclusion

By combining Bash scripting with cron jobs, we automated the process of system monitoring. The script provides real-time insights into CPU, memory, and disk usage, while cron ensures it runs at regular intervals without manual effort. Automating log cleanup prevents uncontrolled file growth, ensuring system performance remains unaffected. This approach saves time, improves efficiency, and demonstrates the importance of scripting and scheduling in system administration.