Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

SYSTEM PERFORMANCE ANALYSIS AND TUNING IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO SYSTEM PERFORMANCE ANALYSIS AND TUNING

What is System Performance Analysis and Tuning?

System performance analysis and tuning is the process of **evaluating, monitoring, and optimizing a UNIX/Linux system** to improve its efficiency, responsiveness, and stability. Performance tuning ensures that the system **operates smoothly under varying workloads**, preventing bottlenecks and failures.

System performance analysis focuses on:

- **Monitoring CPU, memory, disk, and network usage**.
- **Identifying performance bottlenecks and resource constraints**.
- **Optimizing processes, disk I/O, and network performance**.
- **Tuning system parameters to improve efficiency**.

Performance tuning involves **adjusting system configurations, modifying kernel parameters, and optimizing application settings** to enhance overall system responsiveness.

Example: Checking System Performance with top Command

```
top
```

This command provides real-time information about **CPU usage, running processes, memory consumption, and system load**.

Exercise

1. Use top or htop to identify the **highest CPU-consuming process** on your system.
2. Check system uptime and load averages using uptime.

Case Study: Performance Tuning for a Cloud Hosting Provider

A cloud provider experiences **slow application response times** due to high CPU utilization. By **analyzing CPU load, optimizing process scheduling, and adjusting system limits**, they reduce **server response time by 50%**, improving customer experience.

Chapter 2: CPU Performance Analysis and Optimization

Understanding CPU Usage in UNIX/Linux

The **CPU (Central Processing Unit)** is the brain of the system, executing processes and handling computations. High CPU usage can lead to **sluggish performance, slow application response, and system crashes**.

Monitoring CPU Performance

Use the mpstat command (from sysstat package) to analyze CPU usage:

```
mpstat -P ALL 5
```

- %user – CPU time spent executing user processes.

- %system – CPU time used by system/kernel processes.
- %idle – CPU idle percentage (low idle indicates high CPU usage).

Identifying CPU-Intensive Processes

The top or htop command lists running processes and CPU utilization:

htop

To identify the **most CPU-intensive process**:

ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -10

Optimizing CPU Usage

- **Limit CPU Usage of Processes**:
- cpulimit -p 1234 -l 50
- **Change Process Priority (nice and renice)**:
- renice -n 10 -p 1234
- **Optimize Kernel Scheduler Settings** in /etc/sysctl.conf:
- vm.swappiness=10
- kernel.sched_min_granularity_ns=10000000

Exercise

1. Identify the **top CPU-consuming process** and reduce its priority.
2. Configure **kernel CPU scheduling** for better performance.

Case Study: Reducing CPU Bottlenecks in a Financial Trading System

A stock trading system requires **low-latency processing**. By **reducing CPU contention, tuning process priorities, and optimizing kernel scheduling**, the company achieves **faster trade executions** with minimal lag.

Chapter 3: Memory Performance Tuning

Understanding Memory Usage in UNIX/Linux

Memory (RAM) is crucial for **fast application execution**. If memory is insufficient, the system uses **swap space**, leading to slowdowns.

Checking Memory Usage

Use the free command to view memory statistics:

```
free -m
```

- **Mem:** Total, used, and free physical memory.
- **Swap:** Virtual memory used when RAM is full.

Use vmstat to monitor memory usage dynamically:

```
vmstat 5
```

Optimizing Memory Usage

- **Adjust Swap Settings** to prevent excessive disk swapping:
- sudo sysctl vm.swappiness=10
- **Clear Cached Memory** to free up RAM:
- echo 3 | sudo tee /proc/sys/vm/drop_caches
- **Use ulimit to Limit Memory for Processes**:

- ulimit -m 1048576

Exercise

1. Identify memory-intensive processes using ps aux --sort=-%mem | head -10.
2. Reduce swap usage by adjusting vm.swappiness.

Case Study: Optimizing RAM Usage for a Database Server

A database system **suffers from slow queries** due to high memory usage. By **adjusting buffer cache settings, optimizing queries, and limiting swap dependency**, performance improves significantly.

Chapter 4: Disk I/O Performance Optimization

Monitoring Disk Usage and Performance

Disk Input/Output (I/O) performance affects **application response times and system stability**.

Use iostat to check disk activity:

iostat -x 5

- **%util** – Disk usage percentage (above 80% indicates a bottleneck).
- **await** – Time spent waiting for disk operations.

Optimizing Disk Performance

- **Use fsck to Check and Repair Filesystem Issues**:
- sudo fsck -f /dev/sda1
- **Enable Disk Write Caching**:

- sudo hdparm -W1 /dev/sda
- **Optimize Filesystem for Performance** (ext4 tuning):
- sudo tune2fs -o journal_data_writeback /dev/sda1

Exercise

1. Identify **high disk utilization** using iostat.
2. Optimize filesystem performance using tune2fs.

Case Study: Improving File Read/Write Speed on a Web Server

A web hosting provider **experiences slow page loads** due to disk I/O bottlenecks. By **enabling write caching, using SSDs, and optimizing filesystem journaling**, file access speeds increase by **40%**.

Chapter 5: Network Performance Tuning

Analyzing Network Performance

Network performance is crucial for **fast data transfers and low-latency communications**.

Use ifstat to monitor network bandwidth usage:

ifstat -i etho 5

Use netstat to check active connections:

netstat -tunlp

Optimizing Network Settings

- **Increase TCP Buffer Size** in /etc/sysctl.conf:

- net.core.rmem_max=16777216
- net.core.wmem_max=16777216
- **Enable BBR TCP Congestion Control**:
- sudo sysctl -w net.ipv4.tcp_congestion_control=bbr
- **Reduce Network Latency** using ethtool:
- sudo ethtool -G etho rx 4096 tx 4096

Exercise

1. Identify **network latency issues** using ping and traceroute.
2. Optimize **TCP buffer size** to improve data transfer speeds.

Case Study: Reducing Latency for a Video Streaming Platform

A video streaming service **suffers from buffering issues**. By **tuning network buffers, enabling BBR congestion control, and optimizing TCP settings**, latency decreases, providing a **smoother viewing experience**.

CONCLUSION

This guide covered:

✅ **CPU, memory, disk, and network performance analysis**.
✅ **Optimizing system resources for improved performance**.
✅ **Implementing tuning techniques to prevent bottlenecks**.

CPU, MEMORY, AND DISK I/O OPTIMIZATION IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO SYSTEM OPTIMIZATION

What is System Optimization?

System optimization in UNIX/Linux is the process of **fine-tuning CPU, memory, and disk I/O performance** to ensure efficient resource utilization, prevent bottlenecks, and improve system responsiveness. Optimizing these components is essential for:

- **Faster application execution** and reduced system lag.
- **Efficient utilization of hardware resources**.
- **Minimizing system crashes and performance degradation**.
- **Ensuring smooth multitasking and process execution**.

System optimization techniques involve **monitoring, analyzing, and adjusting system parameters** to improve performance without compromising stability.

Example: Checking System Performance Using top

```
top
```

This command provides a real-time view of **CPU, memory, and process utilization**, helping administrators identify performance issues.

Exercise

1. Run top and identify the **top CPU-consuming process**.
2. Check system uptime and load averages using uptime.

Case Study: Performance Optimization for a Cloud Server

A cloud-based application experiences **slow response times** due to high CPU and memory usage. By **identifying bottlenecks, optimizing process scheduling, and tuning memory allocation**, the company improves system performance by **40%**.

Chapter 2: CPU Optimization in UNIX/Linux

Understanding CPU Performance

The **CPU (Central Processing Unit)** is responsible for executing instructions and managing system tasks. High CPU utilization can slow down system performance and lead to **delays in process execution**.

1. Monitoring CPU Usage

Use mpstat to analyze CPU usage across cores:

```
mpstat -P ALL 5
```

- %usr – CPU time spent on user processes.
- %sys – CPU time spent on system/kernel processes.
- %idle – Unused CPU time (low idle means high CPU usage).

Use top to display real-time CPU usage:

```
top -o %CPU
```

To find the **top CPU-consuming process**, use:

```
ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -10
```

2. Optimizing CPU Performance

A. Adjusting Process Priorities Using nice and renice

- **Lower priority (useful for background processes)**
- nice -n 10 command_name
- **Change priority of a running process**
- renice -n -5 -p 1234

B. Limiting CPU Usage for Specific Processes

Use cpulimit to restrict CPU usage:

```
cpulimit -p 1234 -l 50
```

This limits process **1234** to **50% CPU usage**.

C. Optimizing CPU Scheduling Settings

Edit /etc/sysctl.conf to fine-tune CPU scheduling:

```
kernel.sched_latency_ns=6000000

kernel.sched_min_granularity_ns=750000
```

Apply changes:

```
sudo sysctl -p
```

Exercise

1. Identify and **lower the priority** of a high CPU-consuming process.
2. Adjust CPU scheduling settings in /etc/sysctl.conf.

Case Study: Reducing CPU Bottlenecks in a Financial Trading System

A stock trading system requires **fast trade execution**. By **adjusting process priorities and optimizing CPU scheduling,** the company reduces latency and improves **real-time trade processing**.

Chapter 3: Memory Optimization in UNIX/Linux

Understanding Memory Usage

Memory (RAM) is essential for **fast data access and application execution**. When memory is insufficient, the system **uses swap space**, which slows performance.

1. Monitoring Memory Usage

Check real-time memory usage:

```
free -m
```

Analyze memory consumption using vmstat:

```
vmstat 5
```

Find the **top memory-consuming processes**:

```
ps aux --sort=-%mem | head -10
```

2. Optimizing Memory Usage

A. Clearing Cached Memory

To free up memory without affecting running applications:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

B. Adjusting Swappiness (Swap Usage Tuning)

Modify /etc/sysctl.conf to reduce swap dependency:

```
vm.swappiness=10
```

Apply the change:

```
sudo sysctl -p
```

A **lower value (e.g., 10-20)** reduces swap usage, improving speed.

C. Limiting Memory Usage for Specific Processes

Use ulimit to set memory limits:

```
ulimit -m 1048576  # Limit memory usage to 1GB
```

Exercise

1. Adjust vm.swappiness to reduce **swap usage** and improve performance.
2. Identify **memory-intensive applications** and clear cached memory.

Case Study: Optimizing Memory Allocation for a Database Server

A MySQL database **frequently runs out of memory**, causing slow query execution. By **tuning buffer sizes, adjusting swappiness, and optimizing caching**, database performance improves by **30%**.

Chapter 4: Disk I/O Optimization in UNIX/Linux

Understanding Disk I/O Performance

Disk Input/Output (I/O) speed affects **file operations, application performance, and system responsiveness**. Slow disk performance can lead to:

- **High application load times**.

- **Slow database queries**.
- **System lag and unresponsiveness**.

1. Monitoring Disk Performance

Use iostat to analyze disk activity:

```
iostat -x 5
```

- **%util** – Disk usage percentage (above 80% indicates a bottleneck).
- **await** – Average wait time for disk operations.

Find **top disk-consuming processes**:

```
iotop -o
```

2. Optimizing Disk I/O Performance

A. Using fsck to Check and Repair Filesystem Issues

```
sudo fsck -f /dev/sda1
```

B. Enabling Disk Write Caching for Faster Performance

```
sudo hdparm -W1 /dev/sda
```

C. Optimizing Filesystem Performance (Ext4 Tuning)

To enable writeback mode for faster disk writes:

```
sudo tune2fs -o journal_data_writeback /dev/sda1
```

D. Using noatime to Reduce Disk Reads

Modify /etc/fstab to include the noatime option:

```
UUID=xxxx-xxxx  /  ext4  defaults,noatime  0  1
```

This prevents frequent disk updates for file access times.

Exercise

1. Identify **disk I/O bottlenecks** using iostat and iotop.
2. Enable noatime to improve disk performance.

Case Study: Improving Storage Performance for a Web Hosting Provider

A hosting company experiences **slow website load times** due to high disk I/O usage. By **optimizing disk caching, enabling noatime, and using SSDs**, file access speed increases by **50%**.

CONCLUSION

This guide covered:
✅ **Monitoring and optimizing CPU performance using mpstat, nice, and process scheduling.**
✅ **Memory optimization using vm.swappiness, clearing caches, and memory limits.**
✅ **Disk I/O improvements with iostat, write caching, and filesystem tuning.**

KERNEL TUNING PARAMETERS IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO KERNEL TUNING

What is Kernel Tuning?

Kernel tuning in UNIX/Linux refers to the process of **modifying kernel parameters** to improve **system performance, security, and stability**. The kernel acts as the core of the operating system, managing hardware resources, scheduling processes, and handling memory, CPU, and network operations.

Why is Kernel Tuning Important?

- **Optimizes system performance** by adjusting CPU, memory, and network settings.
- **Enhances security** by restricting unauthorized access and modifying resource limits.
- **Prevents bottlenecks** in high-load environments.
- **Customizes system behavior** for specific workloads (databases, web servers, cloud computing, etc.).

Kernel tuning parameters are primarily managed using the **sysctl** command and stored in the configuration file /etc/sysctl.conf.

Example: Checking Current Kernel Parameters

To view all active kernel parameters:

```
sudo sysctl -a
```

To check a specific parameter (e.g., TCP buffer size):

sudo sysctl net.core.rmem_max

Exercise

1. Use sysctl -a to list **all available kernel parameters** on your system.
2. Identify **network-related kernel parameters** using sysctl -a | grep net.

Case Study: Optimizing Kernel Parameters for a High-Traffic Web Server

A web hosting company experiences **slow response times** due to excessive TCP connection timeouts. By **tuning network buffer sizes and reducing connection timeouts**, they improve performance and handle more simultaneous users.

Chapter 2: Managing Kernel Parameters with sysctl

Understanding sysctl Command

The sysctl command allows administrators to **view, modify, and apply kernel parameters** dynamically without rebooting.

Basic sysctl Commands

- **List all kernel parameters:**
- sudo sysctl -a
- **Modify a kernel parameter temporarily:**
- sudo sysctl -w kernel.threads-max=200000
- **Apply changes permanently (via /etc/sysctl.conf):**

- echo "kernel.threads-max=200000" | sudo tee -a /etc/sysctl.conf
- sudo sysctl -p
- **Reload sysctl.conf without rebooting:**
- sudo sysctl --system

Exercise

1. Modify a kernel parameter temporarily using sysctl -w.
2. Persist a kernel setting using /etc/sysctl.conf and reload it.

Case Study: Preventing Fork Bomb Attacks in a Multi-User Environment

A university server crashes frequently due to students **running fork bombs (infinite process creation loops)**. By **limiting the maximum number of processes per user (kernel.pid_max)**, administrators prevent such attacks and stabilize the system.

CHAPTER 3: CPU PERFORMANCE TUNING PARAMETERS

Optimizing CPU Scheduling

The kernel **schedules processes** based on priority and system load. Fine-tuning CPU parameters helps **improve performance** for different workloads.

1. Adjusting Process Scheduling for Performance

Modify CPU scheduling to prioritize real-time applications:

```
sudo sysctl -w kernel.sched_latency_ns=6000000
sudo sysctl -w kernel.sched_min_granularity_ns=750000
```

Make changes persistent:

```
echo "kernel.sched_latency_ns=6000000" | sudo tee -a /etc/sysctl.conf
```

```
echo "kernel.sched_min_granularity_ns=750000" | sudo tee -a /etc/sysctl.conf
```

```
sudo sysctl -p
```

2. Increasing Maximum Threads and Processes

To increase the **maximum number of threads** allowed by the system:

```
sudo sysctl -w kernel.threads-max=250000
```

Exercise

1. Modify **CPU scheduling parameters** and observe system performance changes.
2. Increase **maximum threads** and verify using sysctl kernel.threads-max.

Case Study: Optimizing CPU Scheduling for a Real-Time Video Processing Server

A media company runs a **real-time video streaming service**. By **reducing process scheduling latency**, they minimize buffering and improve user experience.

Chapter 4: Memory Management Tuning Parameters

Optimizing Memory and Swap Usage

Memory management parameters control how the kernel **allocates and optimizes RAM and swap usage**.

1. Adjusting Swappiness (Controls Swap Usage)

A low value reduces swap usage, keeping processes in RAM:

```
sudo sysctl -w vm.swappiness=10
```

To make it permanent:

```
echo "vm.swappiness=10" | sudo tee -a /etc/sysctl.conf
```

2. Increasing Virtual Memory Limits

Increase the maximum memory a process can allocate:

```
sudo sysctl -w vm.max_map_count=262144
```

3. Clearing Cached Memory Periodically

To prevent excessive caching, clear memory periodically:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

Exercise

1. Modify **swappiness** to optimize swap usage.
2. Increase **virtual memory limits** and verify using sysctl vm.max_map_count.

Case Study: Improving Database Performance by Reducing Swap Usage

A database server **experiences slow query execution** due to excessive swapping. By **reducing swappiness and tuning memory allocation**, database query times improve by **30%**.

CHAPTER 5: DISK I/O PERFORMANCE TUNING PARAMETERS

Enhancing Disk Performance

Disk I/O parameters improve **file read/write speeds, reduce latency, and optimize caching mechanisms**.

1. Increasing File Descriptors for High I/O Workloads

Increase the **maximum number of open files**:

```
sudo sysctl -w fs.file-max=2097152

echo "fs.file-max=2097152" | sudo tee -a /etc/sysctl.conf
```

2. Optimizing Disk Read/Write Performance

Enable disk writeback caching for **faster writes**:

```
sudo sysctl -w vm.dirty_ratio=20

sudo sysctl -w vm.dirty_background_ratio=10
```

Exercise

1. Increase **file descriptors** and check the new value using sysctl fs.file-max.
2. Adjust **disk writeback settings** for better performance.

Case Study: Optimizing Disk I/O for a Cloud Storage Provider

A cloud storage provider **suffers from slow file access speeds**. By **optimizing file descriptors and disk write caching**, they improve data retrieval speed, reducing user wait times.

CHAPTER 6: NETWORK PERFORMANCE TUNING PARAMETERS

Improving Network Throughput and Latency

Networking parameters affect **data transfer speeds, TCP connections, and network stability**.

1. Increasing TCP Buffer Size for High-Speed Data Transfers

```
sudo sysctl -w net.core.rmem_max=16777216
```

```
sudo sysctl -w net.core.wmem_max=16777216
```

2. Enabling TCP BBR for Faster Data Transfer

BBR improves TCP congestion control for high-speed networks:

```
sudo sysctl -w net.ipv4.tcp_congestion_control=bbr
```

3. Reducing TIME_WAIT State for Faster Connection Reuse

```
sudo sysctl -w net.ipv4.tcp_fin_timeout=30
```

Exercise

1. Increase **TCP buffer sizes** and verify using sysctl net.core.rmem_max.
2. Enable **TCP BBR** and test network speed improvements.

Case Study: Reducing Latency for a Global CDN Provider

A CDN provider **experiences high latency for global data transfers**. By **enabling TCP BBR and increasing buffer sizes**, they achieve **faster content delivery** across their network.

Conclusion

This guide covered:

✅ Managing **kernel parameters using sysctl**.

- ✅ Optimizing **CPU, memory, disk, and network performance**.
- ✅ Enhancing **security and system stability** through kernel tuning.

TROUBLESHOOTING SYSTEM ISSUES IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO SYSTEM TROUBLESHOOTING

What is System Troubleshooting?

System troubleshooting in UNIX/Linux is the **process of diagnosing and resolving system issues** that affect performance, functionality, and security. Troubleshooting involves analyzing system logs, monitoring resource usage, and applying corrective measures to restore normal operations.

Common System Issues in UNIX/Linux

- **High CPU usage** leading to slow performance.
- **Memory leaks and excessive swap usage**.
- **Disk space running out or I/O bottlenecks**.
- **Network connectivity issues** such as packet loss.
- **Service failures (web servers, databases, SSH, etc.)**.

Effective troubleshooting requires a structured approach:

1. **Identify symptoms** (slow performance, crashes, errors).
2. **Check logs** for warnings and critical alerts.
3. **Analyze system metrics** (CPU, memory, disk, network usage).
4. **Apply fixes and test** the system behavior.
5. **Document the resolution** for future reference.

Example: Checking System Logs for Errors

To view system logs for errors:

```
sudo journalctl -p err -b
```

This lists all errors from the current boot session.

Exercise

1. Identify the **last 10 error messages** in /var/log/syslog.
2. Use journalctl to check for **critical system warnings**.

Case Study: Resolving Performance Issues in a Web Server

A company experiences **slow website loading times**. By analyzing CPU usage and disk I/O, they identify **high memory consumption by Apache**. Restarting the service and optimizing configurations **reduces response time by 60%**.

Chapter 2: Diagnosing and Fixing CPU-Related Issues

Identifying High CPU Usage

High CPU usage can slow down the system, causing delays in application execution.

Use top to check CPU usage in real-time:

```
top -o %CPU
```

Find the top **CPU-consuming process**:

```
ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -10
```

Resolving High CPU Usage Issues

A. Reducing Process Priority (nice and renice)

Lower priority of a CPU-intensive process:

```
sudo renice +10 -p 1234
```

B. Killing Unresponsive or Malicious Processes

To terminate a high-CPU-consuming process:

```
sudo kill -9 1234
```

C. Checking CPU Load Averages

Use uptime to check the system load:

```
uptime
```

A high load average indicates **CPU overload**.

Exercise

1. Identify **high CPU-consuming processes** using top.
2. Reduce CPU usage of a process using renice.

Case Study: Fixing CPU Bottlenecks in a Data Processing Server

A data analytics company notices **CPU spikes** during peak hours. By **scheduling resource-intensive tasks at off-peak hours and optimizing queries**, they **reduce CPU load by 40%**.

Chapter 3: Troubleshooting Memory Issues

Identifying Memory Usage Problems

If a system runs out of RAM, it starts using swap, **leading to slow performance**.

Check memory usage:

```
free -m
```

Analyze processes consuming memory:

```
ps aux --sort=-%mem | head -10
```

Resolving High Memory Usage

A. Clearing Cached Memory

To free up cached memory:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

B. Adjusting Swap Usage (vm.swappiness)

To reduce swap dependency:

```
sudo sysctl -w vm.swappiness=10
```

C. Killing Memory-Intensive Processes

Find and terminate the highest memory-consuming process:

```
sudo kill -9 5678
```

Exercise

1. Identify **top memory-consuming processes** using ps aux.
2. Clear cached memory using drop_caches.

Case Study: Optimizing Memory Usage for a Large Database Server

A MySQL server **slows down due to excessive swap usage**. By **tuning memory buffer settings and reducing swappiness,** database queries execute **30% faster**.

CHAPTER 4: TROUBLESHOOTING DISK SPACE AND I/O ISSUES

Checking Disk Space Usage

If the disk is full, applications may fail to run.

Check available disk space:

```
df -h
```

Find **large files consuming disk space**:

```
sudo du -ah /var | sort -rh | head -10
```

Resolving Disk Space Issues

A. Deleting Unused Log Files

```
sudo rm -rf /var/log/*.log
```

B. Finding and Removing Old Files

To delete files older than **30 days**:

```
sudo find /var/log -type f -mtime +30 -delete
```

C. Checking Disk I/O Performance (iostat)

```
iostat -x 5
```

Exercise

1. Identify the **top 10 largest files** on the system.
2. Delete log files older than **30 days**.

Case Study: Fixing Slow Web Server Performance Due to Full Disk

A hosting provider experiences **downtime due to full disk space**. By **automating log rotation and clearing old backups**, they **free up 50GB of storage**, ensuring smooth operations.

Chapter 5: Troubleshooting Network Connectivity Issues

Checking Network Connection

Verify if the system is connected to the network:

```
ping -c 4 google.com
```

Check active network interfaces:

```
ip addr show
```

Identifying and Resolving Network Issues

A. Checking DNS Resolution Issues

If a domain fails to resolve:

```
nslookup google.com
```

Use Google's DNS server (8.8.8.8):

```
echo "nameserver 8.8.8.8" | sudo tee /etc/resolv.conf
```

B. Restarting Network Services

For Ubuntu/Debian:

```
sudo systemctl restart networking
```

For RHEL/CentOS:

```
sudo systemctl restart NetworkManager
```

C. Checking Firewall Rules

If connections are blocked:

```
sudo ufw status
sudo iptables -L -v
```

Exercise

1. Check if **Google's website is reachable** using ping.
2. Restart network services and verify connectivity.

Case Study: Fixing a Server with No Internet Access

A cloud server **loses internet access** after a configuration change. By **resetting DNS settings and restarting the network service,** engineers **restore connectivity within 5 minutes**.

Chapter 6: Troubleshooting Service Failures

Checking Running Services

List all active services:

```
sudo systemctl list-units --type=service --state=running
```

Check the status of a specific service (e.g., Apache):

```
sudo systemctl status apache2
```

Restarting Failed Services

Restart a service:

```
sudo systemctl restart apache2
```

Check service logs:

```
sudo journalctl -u apache2 --since "1 hour ago"
```

Exercise

1. Identify **failed services** using systemctl.

2. Restart a service and verify logs for errors.

Case Study: Fixing a Crashed Database Server

A database service **crashes due to a corrupted configuration file**. By **restoring the last known good configuration and restarting the service**, engineers bring the system back online.

CONCLUSION

This guide covered:

✅ Diagnosing **CPU, memory, disk, network, and service issues**.
✅ Using **sysctl, top, df, ping, and systemctl** for troubleshooting.
✅ Applying **fixes to restore system stability**.

BACKUP AND RECOVERY STRATEGIES IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO BACKUP AND RECOVERY

What is Backup and Recovery?

Backup and recovery are essential processes in UNIX/Linux system administration that **protect data from loss, corruption, or accidental deletion**. Backups ensure that data can be restored in the event of hardware failures, security breaches, or human errors.

Why Backup and Recovery are Important?

- **Prevents data loss due to system crashes, hardware failures, or cyber-attacks.**
- **Ensures business continuity by minimizing downtime.**
- **Provides a recovery mechanism in case of accidental file deletions.**
- **Meets compliance and regulatory requirements for data protection.**

Backup strategies involve:
✅ **Full backups** – A complete copy of all files and system data.
✅ **Incremental backups** – Copies only the files changed since the last backup.
✅ **Differential backups** – Copies all files changed since the last full backup.

Example: Checking Available Backup Storage Space

```
df -h
```

This command displays **disk space usage** to ensure there is enough storage for backups.

Exercise

1. List all mounted storage devices using df -Th.
2. Identify large files that should be backed up using du -ah /home | sort -rh | head -10.

Case Study: Implementing a Backup Strategy for an E-commerce Website

A web hosting company experienced **data loss due to a ransomware attack**. By implementing **daily incremental and weekly full backups**, they reduced data loss risk and restored their systems quickly.

CHAPTER 2: TYPES OF BACKUP STRATEGIES

1. Full Backup

A **full backup** creates a complete copy of all system files, applications, and databases. It is the most **comprehensive but requires the most storage**.

Example: Creating a Full Backup Using tar

```
tar -cvpzf /backup/full_backup.tar.gz --exclude=/backup --exclude=/proc --exclude=/sys --exclude=/tmp --exclude=/dev /
```

This compresses all system files into **one archive**, excluding unnecessary directories.

2. Incremental Backup

An **incremental backup** only copies files that have changed since the last backup. It is **faster and uses less storage**.

Example: Using rsync for Incremental Backup

```
rsync -av --delete /home/ /backup/incremental/
```

This synchronizes only **new and modified files**.

3. Differential Backup

A **differential backup** copies all files changed since the last **full backup**. It grows larger over time but is faster to restore than incremental backups.

Example: Creating a Differential Backup Using rsync

```
rsync -av --compare-dest=/backup/full/ /home/ /backup/differential/
```

4. Snapshot Backup

Snapshots create **point-in-time copies of the filesystem**. They are commonly used in **LVM (Logical Volume Manager) and cloud environments**.

Example: Creating an LVM Snapshot

```
lvcreate --size 5G --snapshot --name snap_backup /dev/vg0/root
```

This creates a **5GB snapshot** of the root volume.

Exercise

1. Create a full backup of /etc using tar.
2. Use rsync to create an incremental backup of /var/log.

Case Study: Optimizing Backup Storage for a Financial Institution

A financial company **reduces backup storage costs** by switching from **daily full backups to weekly full + daily incremental backups,** saving **70% of storage space** while maintaining data security.

CHAPTER 3: AUTOMATING BACKUPS

Using Cron Jobs for Scheduled Backups

Backups should be scheduled to **run automatically** to ensure data consistency.

Example: Automating a Daily Backup with cron

1. Open the cron scheduler:
2. crontab -e
3. Add the following line for a **daily backup at 2 AM**:
4. 0 2 * * * tar -czf /backup/daily_backup.tar.gz /home

Automating Backups with rsync and Cron

To sync /var/www daily to a backup server:

```
0 3 * * * rsync -avz /var/www/ user@backup-server:/backup/
```

Exercise

1. Schedule an automatic backup of /home using crontab.
2. Automate an incremental backup using rsync.

Case Study: Implementing Automated Backups for a Cloud Storage Service

A cloud storage provider **schedules daily backups using cron and rsync** to replicate user data across multiple locations, ensuring redundancy and data security.

Chapter 4: Data Recovery Techniques

Restoring Files from a Tar Backup

To restore a **full backup**:

```
tar -xvzf /backup/full_backup.tar.gz -C /
```

To restore a **specific file**:

```
tar -xvzf /backup/full_backup.tar.gz -C /home/user
documents/file.txt
```

Restoring Files from rsync Backup

```
rsync -av /backup/incremental/ /home/
```

Recovering Deleted Files Using extundelete (for ext4 Filesystems)

```
sudo extundelete /dev/sda1 --restore-file /home/user/deleted_file.txt
```

Recovering Data from LVM Snapshots

```
lvconvert --merge /dev/vg0/snap_backup
```

This merges the snapshot back into the main volume.

Exercise

1. Restore a specific file from a tar backup.
2. Use rsync to recover files from an incremental backup.

Case Study: Disaster Recovery in an Enterprise Data Center

A power failure **corrupts the primary database server** of a multinational corporation. Using **LVM snapshots**, administrators **restore the system in under an hour**, preventing major financial losses.

Chapter 5: Cloud and Remote Backup Solutions

Using Cloud Storage for Backups

Cloud storage provides **offsite backup solutions** that prevent data loss due to local hardware failures.

Backing Up Files to AWS S3

```
aws s3 sync /backup s3://mybackup-bucket --storage-class STANDARD_IA
```

Using scp to Transfer Backups to a Remote Server

```
scp /backup/full_backup.tar.gz user@backup-server:/remote_backup/
```

Using rclone for Cloud Backups (Google Drive, Dropbox, S3, etc.)

```
rclone sync /backup remote:backup-folder
```

Exercise

1. Transfer a backup file to a remote server using scp.
2. Sync a local directory to Google Drive using rclone.

Case Study: Implementing Cloud Backups for a Healthcare Provider

A healthcare company **switches from local disk backups to AWS S3** for offsite redundancy, ensuring compliance with **HIPAA regulations** and reducing data recovery time.

Conclusion

This guide covered:

✅ **Different backup strategies (full, incremental, differential, snapshots).**

✅ **Automating backups using cron and rsync.**

✅ **Data recovery techniques using tar, rsync, and LVM snapshots.**

✅ **Using cloud storage and remote backups for disaster recovery.**

CLUSTERING AND LOAD BALANCING IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO CLUSTERING AND LOAD BALANCING

What is Clustering and Load Balancing?

Clustering and load balancing are **techniques used to improve system performance, availability, and scalability** by distributing workloads across multiple servers. These methods ensure that no single server becomes overwhelmed, preventing downtime and improving response times.

Difference Between Clustering and Load Balancing

Feature	**Clustering**	**Load Balancing**
Purpose	Provides redundancy and failover	Distributes traffic across multiple servers
Mechanism	Multiple nodes work as a single system	Requests are routed to different servers
Use Cases	High availability, database clustering	Web servers, application servers
Example Technologies	Pacemaker, Corosync, Kubernetes	HAProxy, Nginx, Apache Load Balancer

Both methods are commonly used in **high-traffic websites, cloud computing, and database management** to enhance reliability and performance.

Example: Checking System Load Before Implementing Load Balancing

```
uptime
```

This command shows the **average system load**, helping administrators decide when to scale.

Exercise

1. Use uptime to check your system's load average.
2. Identify active network connections using netstat -tunlp.

Case Study: Improving Website Performance with Load Balancing

An e-commerce website experiences **high traffic spikes** during sales. By implementing **HAProxy for load balancing**, they distribute user requests across multiple servers, reducing downtime and improving customer experience.

CHAPTER 2: UNDERSTANDING CLUSTERING IN UNIX/LINUX

What is Clustering?

Clustering is a technique where **multiple servers (nodes) work together** as a single system. It provides **high availability, fault tolerance, and scalability**.

Types of Clusters

1. **High Availability (HA) Clusters** – Ensure service availability by automatically switching to another node in case of failure.
   - Tools: **Pacemaker, Corosync, Keepalived**
2. **Load Balancing Clusters** – Distribute traffic evenly across multiple nodes.
   - Tools: **HAProxy, Nginx, LVS**

3. **Storage Clusters** – Allow multiple servers to share the same storage.
   - Tools: **GlusterFS, Ceph**
4. **Computational Clusters** – Used for scientific computing and data processing.
   - Tools: **Kubernetes, Apache Hadoop**

Setting Up a Basic HA Cluster Using Pacemaker and Corosync

Step 1: Install Required Packages (Debian/Ubuntu)

```
sudo apt update

sudo apt install pacemaker corosync pcs -y
```

Step 2: Enable Cluster Services

```
sudo systemctl enable corosync pacemaker

sudo systemctl start corosync pacemaker
```

Step 3: Configure Cluster Nodes

Define cluster nodes:

```
sudo pcs cluster setup --name my_cluster node1 node2

sudo pcs cluster start --all
```

Exercise

1. Install Pacemaker and Corosync on two nodes.
2. Configure a simple **HA cluster** and start the service.

Case Study: Preventing Downtime in a Banking System

A bank deploys **a high-availability cluster for online transactions**. When the **primary server fails**, traffic automatically switches to a backup server, preventing disruptions.

CHAPTER 3: LOAD BALANCING IN UNIX/LINUX

What is Load Balancing?

Load balancing **distributes incoming network traffic across multiple servers**, ensuring efficient resource utilization, improved response times, and fault tolerance.

Types of Load Balancing

Load Balancing Type	**Description**	**Example Tools**
Round Robin	Requests are distributed sequentially among servers.	HAProxy, Nginx
Least Connections	New requests go to the server with the fewest active connections.	HAProxy, Nginx
IP Hash	Requests from the same client go to the same backend server.	Nginx, Apache
Weighted Load Balancing	Traffic is distributed based on server capacity.	HAProxy

Setting Up a Load Balancer with HAProxy

Step 1: Install HAProxy

sudo apt update

```
sudo apt install haproxy -y
```

Step 2: Configure HAProxy as a Load Balancer

Edit the HAProxy configuration file:

```
sudo nano /etc/haproxy/haproxy.cfg
```

Add the following configuration:

```
frontend http_front
   bind *:80
   default_backend web_servers


backend web_servers
   balance roundrobin
   server web1 192.168.1.101:80 check
   server web2 192.168.1.102:80 check
```

Step 3: Restart HAProxy

```
sudo systemctl restart haproxy
```

Step 4: Verify Load Balancing

Use curl to check response from different backend servers:

```
curl http://localhost
```

Exercise

1. Install and configure HAProxy for **round-robin load balancing**.
2. Verify load balancing by accessing the website multiple times.

Case Study: Handling Traffic Spikes for a Video Streaming Platform

A video streaming service uses **Nginx load balancing** to distribute traffic across multiple media servers. This prevents **server crashes during peak hours** and ensures a **smooth viewing experience**.

CHAPTER 4: ADVANCED LOAD BALANCING TECHNIQUES

1. Load Balancing with Nginx

Nginx can act as a **reverse proxy and load balancer** for web servers.

Configuring Nginx for Load Balancing

1. Install Nginx:
2. sudo apt install nginx -y
3. Edit the Nginx configuration file:
4. sudo nano /etc/nginx/nginx.conf
5. Add the following load balancing settings:
6. upstream backend {
7. server 192.168.1.101;
8. server 192.168.1.102;
9. }
10.
11. server {
12. listen 80;

13. location / {

14. proxy_pass http://backend;

15. }

16. }

17. Restart Nginx:

18. sudo systemctl restart nginx

2. Load Balancing with Keepalived (Failover Mechanism)

Keepalived provides **redundancy** by assigning a **Virtual IP Address (VIP)** to the active load balancer.

Configuring Keepalived for High Availability

Edit /etc/keepalived/keepalived.conf:

```
vrrp_instance VI_1 {
  state MASTER
  interface eth0
  virtual_router_id 51
  priority 100
  virtual_ipaddress {
    192.168.1.200
  }
}
```

Start the service:

```
sudo systemctl restart keepalived
```

Exercise

1. Configure **Nginx load balancing** for web servers.
2. Set up **Keepalived** to provide a failover solution.

Case Study: Ensuring 99.99% Uptime for a Cloud Storage Provider

A cloud provider **uses Keepalived for high availability**. If the **primary server goes down**, the backup server takes over **without disrupting users**.

CONCLUSION

This guide covered:

✅ **Clustering techniques using Pacemaker and Corosync.**

✅ **Load balancing strategies using HAProxy, Nginx, and Keepalived.**

✅ **Failover mechanisms to ensure high availability.**

✅ **Real-world case studies on traffic handling and redundancy.**

Assignment Solution: Analyze and Optimize System Performance in UNIX/Linux

Objective

This assignment provides a step-by-step guide to **analyzing and optimizing system performance** in UNIX/Linux. By following this guide, you will learn how to:

✅ Monitor **CPU, memory, disk I/O, and network performance**.

✅ Identify **bottlenecks** and performance issues.

✅ Apply **optimization techniques** to improve system efficiency.

Step 1: Checking System Performance Overview

The **first step** in analyzing system performance is getting an overall view of system resource usage.

1. Check System Uptime and Load Averages

Run the following command to check system uptime and load averages:

```
uptime
```

Understanding the Output:

```
11:30:45 up 10 days, 2:30,  2 users,  load average: 1.23, 0.95, 0.70
```

- The **first** number (1.23) represents the system load in the last **1 minute**.
- The **second** number (0.95) represents the system load in the last **5 minutes**.

- The **third** number (0.70) represents the system load in the last **15 minutes**.
- A load average **higher than the number of CPU cores** indicates high system usage.

2. Identify Running Processes and Resource Usage

Run the top command to check real-time system performance:

```
top
```

Press q to exit.

Alternatively, use htop (if installed):

```
htop
```

This provides a **graphical display** of CPU and memory usage.

Exercise

1. Run uptime and note the **load average**.
2. Use top to identify the **highest CPU-consuming process**.

Case Study: Diagnosing High Load on a Web Server

An e-commerce website experiences **slow response times**. By checking top, administrators find that **Apache web server processes** are consuming excessive CPU. Restarting the service and optimizing configuration reduces load by **40%**.

Step 2: Analyzing and Optimizing CPU Performance

1. Identifying CPU Bottlenecks

To list the top **CPU-consuming processes**, use:

```
ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -10
```

2. Adjusting Process Priorities (nice and renice)

If a process is using too much CPU, lower its priority:

```
sudo renice +10 -p <PID>
```

To **limit CPU usage for a process**, install and use cpulimit:

```
sudo apt install cpulimit -y  # Ubuntu/Debian

sudo yum install cpulimit -y  # RHEL/CentOS

cpulimit -p <PID> -l 50
```

This limits the process to **50% CPU usage**.

3. Optimize CPU Scheduling Settings

Modify CPU scheduling parameters in /etc/sysctl.conf:

```
echo "kernel.sched_min_granularity_ns=5000000" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p
```

Exercise

1. Identify the top **CPU-consuming process** using ps.
2. Adjust its priority using renice.

Case Study: Reducing CPU Overload in a Financial Trading Server

A financial system needs **low-latency trading execution**. By **reducing process scheduling granularity**, they **improve trade execution speed by 20%**.

STEP 3: ANALYZING AND OPTIMIZING MEMORY USAGE

1. Checking Memory Usage

To check memory usage:

```
free -m
```

This shows:

- **Total memory available**.
- **Used and free memory**.
- **Swap memory usage**.

2. Finding Memory-Intensive Processes

To list the **top memory-consuming processes**:

```
ps aux --sort=-%mem | head -10
```

3. Reducing Swap Usage (Improving Performance)

Modify the swappiness value to **reduce swap usage**:

```
echo "vm.swappiness=10" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p
```

A lower value (e.g., 10) ensures that the system **prefers RAM over swap**, reducing disk I/O and improving speed.

4. Freeing Cached Memory

To clear cached memory:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

Exercise

1. Identify **top memory-consuming processes** using ps aux --sort=-%mem.
2. Adjust **swappiness** to **reduce swap usage**.

Case Study: Improving Database Performance by Reducing Swap Usage

A MySQL database server suffers **slow queries due to excessive swapping**. By **reducing swappiness**, performance improves, reducing query response time by **30%**.

Step 4: Analyzing and Optimizing Disk I/O Performance

1. Checking Disk Usage

To find **disk usage and free space**:

```
df -h
```

To check **which directories use the most space**:

```
du -ah / | sort -rh | head -10
```

2. Monitoring Disk I/O Performance

To check real-time disk activity:

```
iostat -x 5
```

3. Optimizing Disk Performance

A. Enabling Disk Write Caching

```
sudo hdparm -W1 /dev/sda
```

B. Tuning Filesystem Performance

Enable noatime (reduces unnecessary disk writes):
Edit /etc/fstab and modify:

```
UUID=xxxx-xxxx  /  ext4  defaults,noatime  0  1
```

Exercise

1. Identify **top disk-consuming directories** using du.
2. Enable noatime for faster disk read/write.

Case Study: Increasing Storage Speed for a Cloud Backup System

A cloud storage provider **experiences slow file access speeds**. By **enabling disk write caching and using SSDs**, file access speeds increase by **50%**.

Step 5: Analyzing and Optimizing Network Performance

1. Checking Network Usage

To monitor real-time network traffic:

```
ifstat -i eth0 5
```

2. Checking Open Connections

```
netstat -tunlp
```

3. Optimizing Network Performance

Increase TCP buffer sizes for better performance:

```
echo "net.core.rmem_max=16777216" | sudo tee -a /etc/sysctl.conf

echo "net.core.wmem_max=16777216" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p
```

Enable TCP BBR congestion control:

```
echo "net.ipv4.tcp_congestion_control=bbr" | sudo tee -a /etc/sysctl.conf
```

```
sudo sysctl -p
```

Exercise

1. Monitor **network traffic** using ifstat.
2. Enable **TCP BBR for improved network performance**.

Case Study: Reducing Latency for a Video Streaming Service

A streaming provider **suffers from buffering issues**. By **optimizing TCP buffer sizes and enabling BBR**, latency decreases by **40%**, improving user experience.

Conclusion

This guide covered:

✅ **CPU, memory, disk, and network performance analysis**.

✅ **Using top, ps, iostat, and ifstat to monitor system performance**.

✅ **Applying optimizations to improve system efficiency**.

Assignment Solution: Set Up and Test System Backup Strategies in UNIX/Linux

Objective

This assignment provides a **step-by-step guide** to setting up and testing **backup strategies** in UNIX/Linux. By the end of this guide, you will be able to:

- ✅ Configure **full, incremental, and differential backups**.
- ✅ Automate backups using **cron jobs**.
- ✅ Test and restore backups to ensure data recovery.

Step 1: Understanding Backup Strategies

Before setting up backups, understand the **three main types**:

Backup Type	**Description**	**Storage Usage**	**Recovery Time**
Full Backup	A complete copy of all files	High	Fast
Incremental Backup	Backs up only files changed since the last backup	Low	Slow
Differential Backup	Backs up files changed since the last **full backup**	Medium	Moderate

Example Use Case:

- **Daily incremental backups** (fast and efficient).
- **Weekly full backups** (ensures complete data recovery).

Exercise

1. Identify important directories that need backup (e.g., /home, /etc, /var).
2. Check available disk space using df -h.

STEP 2: CREATING A FULL BACKUP USING TAR

1. Install Required Tools

Ensure tar (archiving tool) is installed:

```
sudo apt install tar -y  # Ubuntu/Debian
```

```
sudo yum install tar -y  # RHEL/CentOS
```

2. Perform a Full Backup

Run the following command to create a full backup:

```
sudo tar -cvpzf /backup/full_backup.tar.gz --exclude=/backup --exclude=/proc --exclude=/sys --exclude=/tmp --exclude=/dev /
```

- -c → Create an archive.
- -v → Show progress.
- -p → Preserve permissions.
- -z → Compress the archive using gzip.
- -f → Specify the backup file name.

3. Verify Backup

List the contents of the backup:

```
tar -tzf /backup/full_backup.tar.gz | head -10
```

Exercise

1. Create a full backup of the /home directory.
2. Verify the backup using tar -tzf.

Step 3: Setting Up Incremental and Differential Backups Using rsync

1. Install rsync (If Not Installed)

```
sudo apt install rsync -y  # Ubuntu/Debian
sudo yum install rsync -y  # RHEL/CentOS
```

2. Create an Incremental Backup

Sync only changed files from /home to /backup/incremental/:

```
rsync -av --delete /home/ /backup/incremental/
```

- -a → Archive mode (preserves permissions and timestamps).
- -v → Verbose output.
- --delete → Remove deleted files from the backup.

3. Create a Differential Backup

Sync files modified **since the last full backup**:

```
rsync -av --compare-dest=/backup/full/ /home/ /backup/differential/
```

4. Verify Backup Contents

ls -lah /backup/incremental/

ls -lah /backup/differential/

Exercise

1. Perform an incremental backup of /var/log.
2. Verify backup files using ls -lah.

Step 4: Automating Backups with Cron Jobs

1. Edit Crontab

Run the following command to schedule backups:

crontab -e

2. Schedule Automatic Backups

Add the following lines to automate backups:

- **Daily incremental backup at 2 AM**:
- 0 2 * * * rsync -av --delete /home/ /backup/incremental/
- **Weekly full backup every Sunday at 3 AM**:
- 0 3 * * 0 tar -cvpzf /backup/full_backup_$(date +\%F).tar.gz /home
- **Monthly differential backup at 4 AM on the 1st of each month**:
- 0 4 1 * * rsync -av --compare-dest=/backup/full/ /home/ /backup/differential/

3. List All Cron Jobs

```
crontab -l
```

Exercise

1. Schedule an **automatic daily incremental backup** using crontab.
2. Verify that the scheduled backup is listed with crontab -l.

Step 5: Testing Backup and Recovery

1. Restore a Full Backup

To restore a full backup:

```
sudo tar -xvzf /backup/full_backup.tar.gz -C /
```

2. Restore a Specific File from Backup

To restore /home/user/documents/file.txt:

```
tar -xvzf /backup/full_backup.tar.gz -C / home/user/documents/file.txt
```

3. Restore an Incremental Backup

To sync files from incremental backup to /home:

```
rsync -av /backup/incremental/ /home/
```

4. Test Backup Integrity

Verify backup integrity using diff:

```
diff -r /home /backup/incremental/
```

Exercise

1. Restore a **specific directory** from the full backup.

2. Verify that the restored files match the original data using diff.

Step 6: Setting Up Remote and Cloud Backups

1. Transfer Backup to a Remote Server Using scp

```
scp /backup/full_backup.tar.gz user@backup-server:/remote_backup/
```

2. Use rclone for Cloud Backups

Install rclone:

```
sudo apt install rclone -y
```

Configure cloud storage:

```
rclone config
```

Sync backup to Google Drive, S3, or Dropbox:

```
rclone sync /backup remote:backup-folder
```

Exercise

1. Transfer a backup file to a **remote server** using scp.
2. Configure rclone and sync backups to a **cloud storage service**.

Step 7: Monitoring and Managing Backups

1. Check Backup Logs

```
cat /var/log/syslog | grep backup
```

2. Set Up Email Alerts for Failed Backups

Modify crontab to send email alerts:

```
0 2 * * * rsync -av --delete /home/ /backup/incremental/ || echo "Backup failed" | mail -s "Backup Alert" admin@example.com
```

3. Verify Backup Storage Usage

```
du -sh /backup/
```

Exercise

1. Set up **email notifications** for failed backups.
2. Check **backup log files** to ensure successful operations.

Conclusion

This guide covered:

✅ **Setting up full, incremental, and differential backups using tar and rsync.**

✅ **Automating backups with cron and verifying integrity with diff.**

✅ **Configuring remote and cloud backups using scp and rclone.**

✅ **Monitoring backup logs and setting up failure alerts.**