Independent Skill Development Mission

Independent Skill Development NxT Mission
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

SYSTEM MONITORING TOOLS (TOP, HTOP, IOSTAT, VMSTAT)

CHAPTER 1: INTRODUCTION TO SYSTEM MONITORING IN LINUX

System monitoring is a crucial aspect of **Linux administration** that allows users to **analyze system performance, diagnose issues, and optimize resource utilization**. Linux provides several powerful command-line tools for **real-time system monitoring**, including **top, htop, iostat, and vmstat**. These tools help users track **CPU usage, memory consumption, disk performance, and system load**, ensuring the system runs efficiently.

System monitoring is essential for:

- **Performance Optimization** – Identifying CPU and memory bottlenecks.
- **Troubleshooting Issues** – Detecting high resource-consuming processes.
- **Preventing System Failures** – Monitoring disk activity and preventing over-utilization.
- **Capacity Planning** – Understanding resource consumption trends.

This chapter provides an in-depth overview of **Linux system monitoring tools** such as **top, htop, iostat, and vmstat**, their usage, commands, examples, and real-world case studies.

Chapter 2: Process and Resource Monitoring with top

Understanding top Command

The top command is a built-in Linux tool that provides a **real-time dynamic overview** of system processes and resource usage. It displays **CPU utilization, memory consumption, running processes, and system load averages**, making it a fundamental tool for system administrators.

Features of top

- Displays system uptime and load average.
- Lists running processes along with their CPU and memory usage.
- Allows users to **sort, filter, and terminate processes** interactively.
- Provides a summary of **total memory and swap usage**.

Using top Command

To launch top, simply type:

```
top
```

Sample output:

```
top - 10:30:01 up 2:00,  1 user,  load average: 0.15, 0.10, 0.05

Tasks: 120 total,   1 running, 119 sleeping,   0 stopped,   0 zombie
```

```
%Cpu(s):  2.0 us,  1.0 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

KiB Mem :  8000000 total,  4000000 free,  2000000 used,  2000000 buff/cache
```

This output provides **real-time system status**, including CPU usage, memory consumption, and system load.

Common top Commands

Command	Function
q	Exit top
k	Kill a process
M	Sort processes by memory usage
P	Sort processes by CPU usage
1	Display CPU usage per core

Example: Sorting by CPU Usage
To sort processes by **highest CPU usage**, press P.

Case Study: Diagnosing High CPU Usage with top

Scenario: A Linux server is experiencing **slow performance**. The administrator uses top and finds a process consuming **90% CPU**.

Solution:

1. Run top and identify the process ID (PID).
2. Use the following command to terminate the high CPU-consuming process:
3. sudo kill -9 <PID>

4. Monitor the system again using top to confirm CPU load reduction.

This case study demonstrates how top helps diagnose and fix **high CPU utilization issues** in real-time.

Chapter 3: Enhanced Process Monitoring with htop

Understanding htop

htop is an **interactive and visually appealing** alternative to top. It provides a user-friendly interface for monitoring **CPU, memory, and process activity**. Unlike top, htop offers:

- **Graphical representation** of CPU and memory usage.
- **Easier process navigation and sorting**.
- **Mouse support** for selecting and killing processes.
- **Tree view** of parent and child processes.

Installing htop

Most Linux distributions do not include htop by default. To install it:

```
sudo apt install htop  # Ubuntu/Debian
```

```
sudo yum install htop  # RHEL/CentOS
```

To launch htop:

```
htop
```

Key Features of htop

Command	Function
F6	Sort processes
F9	Kill a process
F5	Show process tree
F2	Open settings

Example: Identifying High Memory Usage with htop

To identify **processes consuming excessive memory**, use:

1. Open htop
2. Press F6 and select **Memory**
3. Identify and terminate problematic processes

Case Study: Optimizing Server Performance with htop

Scenario: A web server is running multiple background tasks, causing **slow response times**.

Solution:

1. Open htop to **monitor CPU and memory usage**.
2. Identify high-memory processes and use F9 to **terminate unresponsive tasks**.
3. Adjust the system **niceness values** (priority levels) to improve performance.

By using htop, the administrator **optimized system performance** without restarting the server.

CHAPTER 4: DISK PERFORMANCE MONITORING WITH IOSTAT

Understanding iostat

The iostat command **monitors disk I/O performance**, helping users detect slow storage devices and disk bottlenecks.

Installing iostat

iostat is part of the **sysstat package**, which must be installed first:

```
sudo apt install sysstat  # Ubuntu/Debian
```

```
sudo yum install sysstat  # RHEL/CentOS
```

To run iostat:

```
iostat
```

Example: Checking Disk I/O Statistics

```
iostat -x 1 5
```

This command **refreshes disk I/O stats every second for 5 iterations**.

Key Metrics in iostat Output

Metric	Description
tps	Transactions per second
kB_read/s	Kilobytes read per second
kB_wrtn/s	Kilobytes written per second
%util	Percentage of disk utilization

Case Study: Troubleshooting Slow Disk Performance

Scenario: A database server is experiencing **slow query performance**.

Solution:

1. Run iostat -x 5 to **analyze disk usage trends**.
2. If %util exceeds **80%**, identify the process causing high disk I/O.
3. Optimize disk usage by moving logs and cache to a separate disk.

By monitoring **disk activity with iostat**, the administrator **identified and resolved disk bottlenecks**.

CHAPTER 5: SYSTEM STATISTICS WITH VMSTAT

Understanding vmstat

The vmstat command provides a summary of **CPU, memory, and swap usage**, making it useful for **detecting system slowdowns**.

Using vmstat

To display system performance every **2 seconds**:

```
vmstat 2 10
```

This runs vmstat **10 times**, refreshing every **2 seconds**.

Interpreting vmstat Output

Column	Meaning
r	Number of running processes
swpd	Swap memory usage

free	Free memory available
bi	Blocks read from disk
bo	Blocks written to disk

Case Study: Diagnosing High Memory Usage with vmstat

Scenario: A developer reports **slow application performance** due to insufficient RAM.

Solution:

1. Run vmstat 5 10 to monitor **swap usage**.
2. If swpd (swap usage) is high, **increase RAM or optimize applications**.
3. Restart memory-intensive services if necessary.

With vmstat, the administrator **prevented system slowdowns** by **analyzing memory and swap usage trends**.

Chapter 6: Exercise

1. **Use top to find the process with the highest CPU usage and terminate it.**
2. **Install htop and list all running processes with a tree view.**
3. **Use iostat to check disk read/write speeds on your system.**
4. **Run vmstat every 2 seconds and interpret swap memory usage.**
5. **Analyze system load using top and adjust process priority using nice.**

CONCLUSION

Understanding **Linux system monitoring tools** (top, htop, iostat, vmstat) helps administrators **diagnose performance issues, optimize system resources, and maintain server stability**.

Crontab & Scheduled Jobs

Chapter 1: Introduction to Crontab and Scheduled Jobs

What is Crontab?

Crontab (**Cron Table**) is a feature in Linux that allows users to **schedule repetitive tasks** automatically. Instead of manually executing commands at specific times, Crontab helps users and system administrators automate **system maintenance, backups, software updates, and other tasks**.

The tool behind Crontab is called **Cron**, a background process (daemon) that checks a list of scheduled jobs and executes them at the predefined times.

Why Use Crontab?

- **Automates routine tasks** such as backups and log cleaning.
- **Reduces manual intervention** for scheduling tasks.
- **Improves efficiency** by running maintenance scripts at non-peak hours.
- **Allows precise scheduling** down to the minute, hour, day, month, or specific weekdays.

This chapter covers **how to use Crontab, syntax, scheduling jobs, editing Crontab, and troubleshooting scheduled tasks**, along with examples and a real-world case study.

Chapter 2: Understanding Crontab and its Syntax

1. How Crontab Works

Each user can have their own **crontab file**, where they define scheduled tasks. The Cron daemon reads this file and executes jobs as specified.

To view **system-wide Cron jobs**, check the global crontab file:

```
cat /etc/crontab
```

2. Viewing Existing Crontab Jobs

To display the current user's scheduled jobs, use:

```
crontab -l
```

3. Editing Crontab Jobs

To create or edit a crontab file, use:

```
crontab -e
```

This opens the **default text editor**, allowing users to define scheduled jobs.

---

4. Understanding Crontab Syntax

Crontab jobs follow a **specific syntax**:

```
* * * * * command_to_execute

|||||

|||| +---- Day of the week (0-6, Sunday=0)

||| +------ Month (1-12)

|| +-------- Day of the month (1-31)

| +---------- Hour (0-23)
```

```
+------------ Minute (0-59)
```

Example: Schedule a job to run every day at **3:30 AM**

```
30 3 * * * /home/user/backup.sh
```

5. Common Crontab Time Expressions

Expression	Meaning
* * * * *	Every minute
0 * * * *	Every hour
0 0 * * *	Every day at midnight
0 12 * * 1	Every Monday at noon
*/5 * * * *	Every 5 minutes

Chapter 3: Scheduling Jobs with Crontab

1. Scheduling a Simple Task

To schedule a **disk usage check every hour**, add this to Crontab:

```
0 * * * * df -h > /home/user/disk_usage.log
```

This command runs **df -h** every hour and saves the output to a log file.

2. Running a Script at a Specific Time

To run a backup script every night at **2:00 AM**:

```
0 2 * * * /home/user/backup.sh
```

3. Automating System Updates

To update system packages **every Sunday at 4 AM**:

```
0 4 * * 0 sudo apt update && sudo apt upgrade -y
```

4. Running a Job Every 10 Minutes

```
*/10 * * * * /home/user/check_logs.sh
```

This checks logs every **10 minutes**.

Chapter 4: Advanced Crontab Usage

1. Redirecting Output to Log Files

To log output and errors for debugging:

```
0 3 * * * /home/user/script.sh >> /home/user/script.log 2>&1
```

2. Using Special Strings in Crontab

Crontab supports shortcuts for common schedules:

Special String	**Equivalent**
@reboot	Runs once at startup
@hourly	Runs every hour (0 * * * *)
@daily	Runs every day at midnight (0 0 * * *)
@weekly	Runs every week (0 0 * * 0)
@monthly	Runs every month (0 0 1 * *)

Example: Run a script at system startup:

```
@reboot /home/user/startup_script.sh
```

3. Running Cron Jobs as Another User

To schedule a job for another user (john):

```
sudo crontab -u john -e
```

4. Preventing Multiple Instances of a Script

To prevent a script from running multiple times:

```
*/5 * * * * flock -n /tmp/backup.lock /home/user/backup.sh
```

CHAPTER 5: MANAGING AND DEBUGGING CRON JOBS

1. Listing Scheduled Jobs for a Specific User

```
crontab -l -u username
```

2. Removing All Crontab Jobs

```
crontab -r
```

3. Debugging Crontab Jobs

If a cron job is not working, check logs:

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

4. Checking If Cron Service is Running

```
sudo systemctl status cron
```

If it's not running, start the service:

```
sudo systemctl start cron
```

CHAPTER 6: CASE STUDY – AUTOMATING DAILY DATABASE BACKUPS

Scenario:

A company needs to automate **MySQL database backups** to prevent data loss.

Solution:

1. **Create a Backup Script (db_backup.sh)**
2. #!/bin/bash
3. TIMESTAMP=$(date +"%Y-%m-%d_%H-%M-%S")
4. BACKUP_DIR="/backups"
5. mkdir -p $BACKUP_DIR
6. mysqldump -u root -p mydatabase > $BACKUP_DIR/db_backup_$TIMESTAMP.sql
7. **Make the Script Executable**
8. chmod +x /home/user/db_backup.sh
9. **Schedule the Backup Every Night at 1 AM**
10. 0 1 * * * /home/user/db_backup.sh

Outcome:

- The database is automatically backed up daily.
- No manual intervention is needed.
- Backups are stored with timestamps for easy recovery.

CHAPTER 7: EXERCISE

1. **Schedule a script (cleanup.sh) to delete old log files every week.**
2. **Use @reboot to run a script on system startup.**
3. **Configure a cron job to check free disk space every hour.**
4. **Schedule a cron job that sends an email report every day.**
5. **Debug a failed cron job by checking /var/log/syslog.**

CONCLUSION

Crontab is an essential tool for **automating tasks** in Linux. By mastering **scheduled jobs, logging, debugging, and advanced cron features**, users can enhance **system administration, maintenance, and performance optimization**.

LINUX FIREWALL & SECURITY BASICS

CHAPTER 1: INTRODUCTION TO LINUX FIREWALL AND SECURITY

Why is Security Important in Linux?

Linux is known for its **robust security architecture**, but securing a Linux system requires **proper configuration of firewalls, user permissions, and system hardening techniques**. A firewall is a key component that controls **incoming and outgoing network traffic** based on predefined security rules.

Key Aspects of Linux Security

- **Firewall Management:** Controlling traffic using iptables or ufw.
- **User and Permission Management:** Implementing the **principle of least privilege**.
- **System Updates and Patching:** Regularly updating software to fix vulnerabilities.
- **Monitoring and Logging:** Using fail2ban, auditd, and log analysis to detect intrusions.

By configuring security measures properly, Linux users can **prevent unauthorized access, protect sensitive data, and mitigate cyber threats**.

CHAPTER 2: UNDERSTANDING LINUX FIREWALLS

1. What is a Firewall?

A firewall is a **network security system** that filters traffic based on predefined rules. It can:

- **Allow or block connections** based on security policies.
- **Protect the system from unauthorized access and attacks.**
- **Restrict incoming and outgoing traffic** to specific ports and IP addresses.

2. Types of Firewalls in Linux

Firewall	**Description**
iptables	A powerful command-line firewall tool (low-level packet filtering).
ufw (Uncomplicated Firewall)	A user-friendly firewall interface (for Ubuntu & Debian).
firewalld	A dynamic firewall used in **RHEL & CentOS** systems.

CHAPTER 3: CONFIGURING FIREWALLS IN LINUX

1. Using iptables for Firewall Management

iptables is a **rule-based firewall** that filters packets based on IP addresses, protocols, and ports.

View Current Firewall Rules

```
sudo iptables -L -v
```

Blocking an IP Address

```
sudo iptables -A INPUT -s 192.168.1.100 -j DROP
```

This blocks all traffic from **192.168.1.100**.

Allow SSH Connections (Port 22)

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

Save Firewall Rules Permanently

```
sudo iptables-save > /etc/iptables.rules
```

---

2. Managing Firewalls with ufw (Uncomplicated Firewall)

ufw (Uncomplicated Firewall) simplifies firewall management and is commonly used in **Ubuntu** systems.

Enable ufw Firewall

```
sudo ufw enable
```

Allow SSH, HTTP, and HTTPS Traffic

```
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
```

Deny Incoming Traffic from an IP Address

```
sudo ufw deny from 192.168.1.100
```

View Active Firewall Rules

```
sudo ufw status verbose
```

Disable Firewall Temporarily

```
sudo ufw disable
```

---

3. Using firewalld (For RHEL & CentOS Systems)

firewalld provides **dynamic firewall management** in **RHEL and CentOS**.

Start and Enable firewalld

```
sudo systemctl start firewalld
```

```
sudo systemctl enable firewalld
```

List Available Firewall Zones

```
sudo firewall-cmd --get-active-zones
```

Allow HTTP & HTTPS Traffic

```
sudo firewall-cmd --zone=public --add-service=http --permanent
```

```
sudo firewall-cmd --zone=public --add-service=https --permanent
```

```
sudo firewall-cmd --reload
```

Check Active Firewall Rules

```
sudo firewall-cmd --list-all
```

Chapter 4: Securing Linux with System Hardening Techniques

1. Disabling Unused Services

Unnecessary services **increase the attack surface**. Disable unwanted services using:

```
sudo systemctl disable apache2
sudo systemctl disable ftp
```

2. Configuring SSH Security

- Disable root login in SSH:
- sudo nano /etc/ssh/sshd_config

Change:

```
PermitRootLogin no
```

- Restart SSH service:
- sudo systemctl restart sshd

3. Enforcing Password Policies

To require strong passwords, edit /etc/login.defs:

```
PASS_MIN_LEN 12
PASS_WARN_AGE 7
```

4. Enabling Automatic Security Updates

For Debian-based systems:

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure unattended-upgrades
```

Chapter 5: Monitoring and Logging for Security

1. Using fail2ban to Prevent Brute Force Attacks

fail2ban detects repeated failed login attempts and **blocks IPs temporarily**.

Install fail2ban

```
sudo apt install fail2ban
```

Enable Fail2Ban Service

```
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

Check Banned IP Addresses

```
sudo fail2ban-client status sshd
```

---

2. Checking System Logs for Security Events

Log File	Description
/var/log/auth.log	Stores authentication logs (logins, SSH access).
/var/log/syslog	Contains general system logs.
/var/log/secure	Security logs (for RHEL/CentOS).

Example: Checking Failed SSH Login Attempts

```
sudo cat /var/log/auth.log | grep "Failed password"
```

CHAPTER 6: CASE STUDY – PROTECTING A LINUX WEB SERVER FROM ATTACKS

Scenario:

A company runs a **web server** that is frequently targeted by brute-force login attempts and port scans.

Solution:

1. **Enable Firewall Protection:**
2. sudo ufw enable
3. sudo ufw allow http
4. sudo ufw allow https
5. sudo ufw allow ssh
6. **Limit SSH Access to a Specific IP:**
7. sudo ufw allow from 192.168.1.10 to any port 22
8. **Install fail2ban to Block Repeated Login Attempts:**
9. sudo apt install fail2ban
10. sudo systemctl start fail2ban
11. **Disable Unused Services to Reduce Attack Surface:**
12. sudo systemctl disable telnet
13. sudo systemctl disable ftp

Outcome:

- **Unwanted SSH attempts are blocked.**
- **Only trusted IPs can access SSH.**
- **Firewall rules prevent unauthorized network access.**
- **System security is significantly improved.**

CHAPTER 7: EXERCISE

1. **Configure iptables to allow SSH and block all other incoming traffic.**

2. **Use ufw to allow only HTTP (port 80) and HTTPS (port 443) traffic.**

3. **Set up fail2ban to ban an IP after 3 failed SSH login attempts.**

4. **Check /var/log/auth.log to identify unauthorized login attempts.**

5. **Harden SSH security by disabling root login and changing the SSH port.**

CONCLUSION

Linux security relies on **firewall management, system hardening, and monitoring techniques** to protect against attacks. By configuring **iptables, ufw, firewalld, and fail2ban**, users can **secure their Linux environment, prevent unauthorized access, and reduce system vulnerabilities**.

MANAGING SYSTEM LOGS

CHAPTER 1: INTRODUCTION TO SYSTEM LOGGING IN LINUX

What Are System Logs?

System logs are **records of system activities, errors, security events, and application events** in a Linux system. They help administrators:

- **Monitor system performance and security**
- **Troubleshoot issues** related to hardware, software, and users
- **Track system health and uptime**
- **Comply with security and auditing policies**

Types of System Logs

Linux categorizes logs into different types:

- **System logs** (kernel, boot, hardware events)
- **Security logs** (authentication, firewall events)
- **Application logs** (web servers, databases)
- **Event logs** (scheduled tasks, system crashes)

This chapter explores **how to manage, filter, analyze, and automate log maintenance** using tools like journalctl, rsyslog, logrotate, and dmesg.

CHAPTER 2: UNDERSTANDING LOG FILES AND THEIR LOCATIONS

1. Common System Log Files in Linux

Linux logs are stored in the /var/log/ directory.

Log File	**Description**
/var/log/syslog	General system messages (Ubuntu, Debian).
/var/log/messages	System logs (RHEL, CentOS).
/var/log/auth.log	Authentication logs (login attempts, SSH access).
/var/log/kern.log	Kernel messages.
/var/log/boot.log	System boot messages.
/var/log/dmesg	Hardware and boot messages.
/var/log/secure	Security-related logs (RHEL-based systems).
/var/log/apache2/access.log	Apache web server access logs.
/var/log/mysql.log	MySQL database logs.

---

2. Viewing System Logs

Using cat and less to Read Logs

cat /var/log/syslog

less /var/log/auth.log

Use less to scroll through large log files.

Filtering Logs with grep

To find **SSH login attempts**:

grep "sshd" /var/log/auth.log

Checking Kernel Logs

dmesg | less

This displays **hardware-related messages** during system boot.

Chapter 3: Managing Logs with journalctl (For systemd Systems)

1. What is journalctl?

journalctl is used for viewing and filtering logs in **systemd-based** distributions (Ubuntu, RHEL, CentOS).

2. Viewing System Logs

To see the full system log:

journalctl

To view logs from the last boot:

journalctl -b

3. Filtering Logs

- **By Service:**
- journalctl -u sshd
- **By Time Range:**
- journalctl --since "1 hour ago"

4. Checking Failed Login Attempts

journalctl -u sshd | grep "Failed password"

5. Viewing Logs in Real-Time

To monitor logs **as they are generated**:

```
journalctl -f
```

CHAPTER 4: CONFIGURING RSYSLOG FOR LOG MANAGEMENT

1. What is rsyslog?

rsyslog is a **powerful logging system** that allows users to:

- **Collect logs from multiple sources**
- **Filter and route logs based on priority**
- **Send logs to remote servers**

2. Configuring rsyslog

Edit the rsyslog configuration file:

```
sudo nano /etc/rsyslog.conf
```

To **enable remote logging**, add:

```
*.* @192.168.1.100:514
```

Then restart the service:

```
sudo systemctl restart rsyslog
```

3. Checking rsyslog Status

```
sudo systemctl status rsyslog
```

CHAPTER 5: LOG ROTATION USING LOGROTATE

1. What is logrotate?

logrotate automatically **compresses, renames, and deletes old logs**, preventing logs from consuming excessive disk space.

2. Checking Default Log Rotation Settings

To view log rotation rules:

```
cat /etc/logrotate.conf
```

To check settings for specific services:

```
cat /etc/logrotate.d/apache2
```

3. Custom Log Rotation Configuration

To **rotate Apache logs daily and keep 7 days of logs**, create a new configuration file:

```
sudo nano /etc/logrotate.d/apache_custom
```

Add the following:

```
/var/log/apache2/*.log {
   daily
   rotate 7
   compress
   missingok
   notifempty
   create 0640 root adm
}
```

Save and restart logrotate:

```
sudo logrotate -f /etc/logrotate.conf
```

4. Testing Log Rotation

```
sudo logrotate -d /etc/logrotate.conf
```

Chapter 6: Securing Logs with Permissions and Remote Logging

1. Securing Log Files

To prevent unauthorized access, set proper **file permissions**:

```
sudo chmod 640 /var/log/auth.log
sudo chown root:adm /var/log/auth.log
```

2. Configuring Remote Log Storage

For centralized logging, send logs to a **remote syslog server**:

```
*.* @logserver.example.com:514
```

On the remote server, ensure rsyslog is configured to accept logs:

```
sudo nano /etc/rsyslog.conf
```

Enable:

```
$ModLoad imudp
$UDPServerRun 514
```

Restart rsyslog:

```
sudo systemctl restart rsyslog
```

Chapter 7: Case Study – Troubleshooting a Server Crash Using Logs

Scenario:

A web server **crashed unexpectedly**, and the administrator needs to diagnose the issue.

Solution:

1. **Check System Logs for Errors**
2. journalctl -b -1

This shows logs from **the last boot** before the crash.

3. **Check Kernel Logs for Hardware Issues**
4. dmesg | grep -i "error"
5. **Analyze Apache Logs for Web Server Issues**
6. tail -n 50 /var/log/apache2/error.log
7. **Check Authentication Logs for Unauthorized Access**
8. grep "Failed password" /var/log/auth.log

Outcome:

- Logs revealed **a failing disk drive** (dmesg showed I/O errors).
- **Immediate disk replacement prevented further downtime.**

Chapter 8: Exercise

1. **View the system logs using journalctl and filter logs from the last hour.**

2. **Check /var/log/auth.log for failed SSH login attempts.**
3. **Configure logrotate to archive system logs weekly and keep backups for 30 days.**
4. **Set up remote logging using rsyslog and send logs to another Linux system.**
5. **Secure log files by changing ownership and restricting access.**

CONCLUSION

Effective **log management** is essential for **system monitoring, troubleshooting, and security auditing**. By using tools like journalctl, rsyslog, and logrotate, administrators can **analyze logs, automate log rotation, and secure log files** to ensure system stability and security.

Networking in Linux (IP Addressing, Subnetting, DNS, DHCP)

Chapter 1: Introduction to Networking in Linux

What is Networking in Linux?

Networking in Linux enables systems to **communicate with other devices** over local networks (LAN) or the internet. Linux provides robust tools for managing **IP addressing, subnetting, DNS, and DHCP**, ensuring efficient data transmission and network configuration.

Why is Networking Important in Linux?

- **Facilitates communication** between devices in a network.
- **Allows access to the internet** and external services.
- **Enables remote administration** using SSH and other protocols.
- **Supports security and traffic control** via firewalls and routing rules.

This chapter will cover **Linux networking fundamentals**, including **IP addressing, subnetting, DNS configuration, and DHCP services**.

Chapter 2: IP Addressing in Linux

1. What is an IP Address?

An **IP address (Internet Protocol Address)** is a **unique identifier** assigned to a device in a network. Linux supports both:

- **IPv4 (e.g., 192.168.1.1)**
- **IPv6 (e.g., 2001:db8::ff00:42:8329)**

2. Viewing IP Address in Linux

To display the IP address of a system:

```
ip addr show
```

or

```
ifconfig  # (Deprecated but still used)
```

Sample output:

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
  inet 192.168.1.10/24 brd 192.168.1.255 scope global eth0
```

- inet 192.168.1.10/24 → The system's IP address with a **subnet mask**.

---

3. Assigning a Static IP Address

To manually configure a **static IP address** (Ubuntu/Debian-based systems):

Edit the network configuration file:

```
sudo nano /etc/netplan/01-network-manager-all.yaml
```

Add the following configuration:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      addresses:
        - 192.168.1.100/24
      gateway4: 192.168.1.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

Apply the changes:

```
sudo netplan apply
```

For **CentOS/RHEL**, modify:

```
sudo nano /etc/sysconfig/network-scripts/ifcfg-eth0
```

Set:

```
BOOTPROTO=none
IPADDR=192.168.1.100
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
```

DNS1=8.8.8.8

Then restart networking:

```
sudo systemctl restart network
```

Chapter 3: Subnetting in Linux

1. What is Subnetting?

Subnetting divides a large network into **smaller, manageable networks** to improve performance and security.

2. Understanding Subnet Masks

A **subnet mask** determines which part of the IP address belongs to the **network** and which belongs to the **host**.

CIDR	Subnet Mask	Hosts per Subnet
/24	255.255.255.0	254
/26	255.255.255.192	62
/30	255.255.255.252	2

3. Calculating Network and Broadcast Addresses

For **192.168.1.0/24**:

- **Network Address:** 192.168.1.0
- **Broadcast Address:** 192.168.1.255
- **Valid Hosts:** 192.168.1.1 – 192.168.1.254

To find subnet details in Linux:

```
ipcalc 192.168.1.0/24
```

or

```
ip route show
```

Chapter 4: Domain Name System (DNS) in Linux

1. What is DNS?

DNS (Domain Name System) translates **human-readable domain names** into **IP addresses**.

2. Checking DNS Configuration

To see the system's current DNS servers:

```
cat /etc/resolv.conf
```

Sample output:

```
nameserver 8.8.8.8

nameserver 8.8.4.4
```

3. Changing DNS Servers

To use Google's DNS (8.8.8.8 and 8.8.4.4):

```
sudo nano /etc/resolv.conf
```

Modify:

```
nameserver 8.8.8.8

nameserver 8.8.4.4
```

For persistent changes (Ubuntu/Debian), edit:

```
sudo nano /etc/systemd/resolved.conf
```

Add:

```
DNS=8.8.8.8 8.8.4.4
```

Restart DNS services:

```
sudo systemctl restart systemd-resolved
```

4. Testing DNS Resolution

To verify DNS lookup:

```
nslookup google.com
```

or

```
dig google.com
```

Chapter 5: Dynamic Host Configuration Protocol (DHCP)

1. What is DHCP?

DHCP **automatically assigns** IP addresses to devices in a network. It eliminates the need for **manual IP assignment**.

2. Checking DHCP Configuration

To check if an interface is using DHCP:

```
nmcli device show eth0 | grep IP4.DHCP
```

or

```
cat /var/lib/dhcp/dhclient.leases
```

3. Setting Up a DHCP Client

Ensure the network interface is configured for DHCP in **Ubuntu/Debian**:

```
network:
  ethernets:
    eth0:
      dhcp4: yes
```

Apply changes:

```
sudo netplan apply
```

For **RHEL/CentOS**, edit:

```
sudo nano /etc/sysconfig/network-scripts/ifcfg-eth0
```

Set:

```
BOOTPROTO=dhcp
```

Restart networking:

```
sudo systemctl restart network
```

4. Setting Up a DHCP Server

To install and configure a **DHCP server**:

```
sudo apt install isc-dhcp-server
```

Edit the configuration file:

```
sudo nano /etc/dhcp/dhcpd.conf
```

Define a DHCP range:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.100 192.168.1.200;
  option routers 192.168.1.1;
  option domain-name-servers 8.8.8.8;
}
```

Restart the DHCP service:

```
sudo systemctl restart isc-dhcp-server
```

Chapter 6: Case Study – Setting Up a Local Network in Linux

Scenario:

A small company needs to set up a **local network** with the following requirements:

- **DHCP assigns IPs** automatically.
- **DNS translates domain names** into IPs.
- **Subnetting is used** to divide departments.

Solution:

1. **Configure the DHCP server** to assign IPs in the 192.168.1.0/24 range.
2. **Set up a DNS server** using bind9 for internal name resolution.
3. **Divide the network** into /26 subnets for different teams.

Outcome:

- **All devices receive automatic IPs** via DHCP.
- **Users can access servers via domain names** instead of IPs.
- **Efficient subnetting** ensures organized network management.

CHAPTER 7: EXERCISE

1. **Find your system's IP address and subnet mask using ip addr.**
2. **Configure a static IP address for a network interface.**
3. **Change the DNS server to 1.1.1.1 and test resolution using nslookup.**
4. **Set up a DHCP server to assign IPs dynamically.**
5. **Subnet a network and calculate valid host ranges for /27 and /29.**

CONCLUSION

Understanding **Linux networking** enables users to **configure and troubleshoot IP addressing, DNS, and DHCP** efficiently.

SSH, FTP, SCP, AND SFTP CONFIGURATION

CHAPTER 1: INTRODUCTION TO SECURE REMOTE ACCESS AND FILE TRANSFER IN LINUX

Why Secure Remote Access and File Transfer Matter

Linux provides multiple methods for **secure remote access and file transfers**, including **SSH (Secure Shell), FTP (File Transfer Protocol), SCP (Secure Copy Protocol), and SFTP (Secure File Transfer Protocol)**. These protocols allow system administrators and users to:

- **Remotely manage Linux servers** from anywhere.
- **Transfer files securely** between systems.
- **Automate system administration tasks** using SSH scripts.
- **Enable secure file sharing** within a network.

Overview of Protocols

Protocol	**Description**	**Security**
SSH (Secure Shell)	Remote login and command execution over a secure channel	Encrypted
FTP (File Transfer Protocol)	Traditional file transfer method	Not encrypted (unless FTPS is used)
SCP (Secure Copy Protocol)	Transfers files securely using SSH	Encrypted

SFTP (Secure FTP)	FTP over SSH, providing secure file transfers	Encrypted

This chapter covers **installation, configuration, and usage of SSH, FTP, SCP, and SFTP**, with examples, exercises, and case studies.

Chapter 2: SSH Configuration and Secure Remote Access

1. What is SSH?

SSH (**Secure Shell**) is a protocol for **secure remote login** and command execution between two machines. It encrypts data, ensuring **confidentiality and security**.

2. Installing SSH Server

Most Linux distributions include **OpenSSH** by default. If it's not installed, install it using:

- **Debian/Ubuntu**:
- sudo apt install openssh-server -y
- **CentOS/RHEL**:
- sudo yum install openssh-server -y

3. Starting and Enabling SSH Service

To start SSH:

```
sudo systemctl start ssh
```

To enable SSH at boot:

```
sudo systemctl enable ssh
```

To check the status:

sudo systemctl status ssh

4. Connecting to an SSH Server

To connect to a remote server using SSH:

ssh username@server_ip

Example:

ssh alice@192.168.1.100

5. Configuring SSH for Security

To improve SSH security, modify its configuration file:

sudo nano /etc/ssh/sshd_config

- **Disable Root Login**:
- PermitRootLogin no
- **Change SSH Port** (default is 22):
- Port 2222
- **Allow Only Specific Users**:
- AllowUsers alice bob

Restart SSH to apply changes:

sudo systemctl restart ssh

6. Setting Up SSH Key-Based Authentication

For more security, use SSH keys instead of passwords:

ssh-keygen -t rsa

ssh-copy-id alice@192.168.1.100

Now, you can SSH into the server **without a password**.

CHAPTER 3: FILE TRANSFER USING FTP (FILE TRANSFER PROTOCOL)

1. What is FTP?

FTP is a protocol used for **transferring files between computers** over a network. However, standard FTP **does not encrypt data**, making it **less secure** than SFTP.

2. Installing an FTP Server

To set up an FTP server on Linux:

- **Debian/Ubuntu**:
- sudo apt install vsftpd -y
- **CentOS/RHEL**:
- sudo yum install vsftpd -y

3. Configuring FTP Server (vsftpd)

Edit the configuration file:

sudo nano /etc/vsftpd.conf

- **Enable anonymous access (not recommended for security)**
- anonymous_enable=NO
- **Enable local user login**
- local_enable=YES
- **Restrict users to their home directories**

- chroot_local_user=YES

Restart the FTP server:

sudo systemctl restart vsftpd

sudo systemctl enable vsftpd

4. Connecting to FTP Server

From a client machine:

ftp server_ip

Example:

ftp 192.168.1.100

Login using your **Linux credentials**, then use FTP commands like:

- ls – List files
- put file.txt – Upload file
- get file.txt – Download file
- bye – Exit FTP session

CHAPTER 4: SECURE FILE TRANSFER USING SCP (SECURE COPY PROTOCOL)

1. What is SCP?

SCP is a secure way to transfer files **over SSH**. It encrypts both **data and authentication credentials**.

2. Copying Files from Local to Remote Machine

scp file.txt alice@192.168.1.100:/home/alice/

This copies file.txt to the remote user's home directory.

3. Copying Files from Remote to Local Machine

```
scp alice@192.168.1.100:/home/alice/file.txt /local/destination/
```

4. Copying a Directory Recursively

```
scp -r /local/directory alice@192.168.1.100:/home/alice/
```

CHAPTER 5: SECURE FILE TRANSFER USING SFTP (SECURE FTP)

1. What is SFTP?

SFTP (**Secure File Transfer Protocol**) is a **secure alternative to FTP**, as it runs over SSH and encrypts all transfers.

2. Connecting to SFTP Server

```
sftp alice@192.168.1.100
```

After connecting, use commands like:

- ls – List files
- cd folder – Navigate directories
- get file.txt – Download a file
- put file.txt – Upload a file
- bye – Exit SFTP

3. Restricting SFTP Users to Home Directory

Modify SSH configuration:

```
sudo nano /etc/ssh/sshd_config
```

Add:

```
Match User alice
  ForceCommand internal-sftp
  ChrootDirectory /home/alice
  AllowTcpForwarding no
  X11Forwarding no
```

Restart SSH:

```
sudo systemctl restart ssh
```

Chapter 6: Case Study – Secure File Transfers in a Corporate Network

Scenario:

A company needs a **secure method for employees to upload and download files** from a central server while restricting access to sensitive system files.

Solution:

1. **Install and configure an SFTP server** with restricted user access.
2. **Disable FTP and enforce SFTP over SSH** for security.
3. **Set up SSH key authentication for admin access.**
4. **Monitor file transfers using system logs (/var/log/auth.log).**

Outcome:

- **Employees can securely transfer files using SFTP.**
- **Unnecessary FTP access is disabled, preventing potential security risks.**
- **Logs are monitored for suspicious activities.**

Chapter 7: Exercise

1. **Install and configure an SSH server on your Linux machine.**
2. **Set up an FTP server and allow only local user logins.**
3. **Use SCP to transfer a file from your local machine to a remote server.**
4. **Restrict SFTP users to their home directories using SSH configurations.**
5. **Monitor SSH login attempts using /var/log/auth.log.**

Conclusion

SSH, FTP, SCP, and SFTP are **essential tools** for **remote access and secure file transfer** in Linux. While **SSH and SFTP** provide encrypted communication, **FTP** should be used only when necessary, preferably with **FTPS (FTP Secure)**. By mastering these tools, system administrators can **enhance security, efficiency, and remote management capabilities**.

Configuring Web Servers (Apache, Nginx)

Chapter 1: Introduction to Web Servers in Linux

What is a Web Server?

A web server is a **software application** that processes **HTTP/HTTPS requests** and serves web content, such as **HTML pages, images, videos, and dynamic applications**. The two most commonly used web servers in Linux are:

- **Apache (HTTPD):** A highly customizable, widely used web server.
- **Nginx:** A lightweight, high-performance web server often used for reverse proxying and load balancing.

Why Use a Web Server?

- **Hosts websites and web applications.**
- **Handles client requests efficiently.**
- **Supports dynamic content (PHP, Python, Node.js).**
- **Implements security measures like SSL/TLS.**

This chapter covers **installation, configuration, security, and optimization of Apache and Nginx** web servers with hands-on examples and case studies.

Chapter 2: Installing and Configuring Apache Web Server

1. Installing Apache

Apache is available in most Linux distributions:

On Debian/Ubuntu:

```
sudo apt update

sudo apt install apache2 -y
```

On CentOS/RHEL:

```
sudo yum install httpd -y

sudo systemctl enable httpd
```

2. Starting and Enabling Apache

To start the Apache service:

```
sudo systemctl start apache2  # Ubuntu/Debian

sudo systemctl start httpd    # CentOS/RHEL
```

To enable it at boot:

```
sudo systemctl enable apache2  # Ubuntu/Debian

sudo systemctl enable httpd    # CentOS/RHEL
```

To check its status:

```
sudo systemctl status apache2
```

3. Verifying Apache Installation

Open a web browser and enter your server's IP address:

```
http://your-server-ip
```

If Apache is running, you should see the **Apache default web page**.

To find your server's IP:

```
ip addr show
```

---

4. Configuring Virtual Hosts in Apache

A **Virtual Host** allows Apache to serve multiple websites from a single server.

Step 1: Create a Directory for the Website

```
sudo mkdir -p /var/www/example.com/html

sudo chown -R $USER:$USER /var/www/example.com/html

sudo chmod -R 755 /var/www
```

Step 2: Create an Index File

```
sudo nano /var/www/example.com/html/index.html
```

Add the following:

```
<html>

  <head><title>Welcome to Example.com</title></head>

  <body><h1>Example.com is working!</h1></body>

</html>
```

Step 3: Create the Virtual Host Configuration File

```
sudo nano /etc/apache2/sites-available/example.com.conf  # Ubuntu/Debian

sudo nano /etc/httpd/conf.d/example.com.conf  # CentOS/RHEL
```

Add the following:

```
<VirtualHost *:80>
```

```
    ServerAdmin admin@example.com

    ServerName example.com

    ServerAlias www.example.com

    DocumentRoot /var/www/example.com/html

    ErrorLog ${APACHE_LOG_DIR}/error.log

    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Step 4: Enable the Virtual Host

```
sudo a2ensite example.com.conf  # Ubuntu/Debian

sudo systemctl reload apache2    # Ubuntu/Debian
```

For CentOS/RHEL, restart Apache:

```
sudo systemctl restart httpd
```

5. Allow Firewall Access

```
sudo ufw allow 80/tcp   # Ubuntu/Debian

sudo firewall-cmd --permanent --add-service=http  # CentOS/RHEL

sudo firewall-cmd --reload
```

Chapter 3: Installing and Configuring Nginx Web Server

1. Installing Nginx

On Debian/Ubuntu:

```
sudo apt update
sudo apt install nginx -y
```

On CentOS/RHEL:

```
sudo yum install epel-release -y
```

```
sudo yum install nginx -y
```

2. Starting and Enabling Nginx

```
sudo systemctl start nginx
```

```
sudo systemctl enable nginx
```

To check its status:

```
sudo systemctl status nginx
```

3. Verifying Nginx Installation

Open a browser and enter:

http://your-server-ip

If successful, the **default Nginx welcome page** appears.

---

4. Configuring Virtual Hosts in Nginx

Similar to Apache, Nginx uses **server blocks** for hosting multiple sites.

Step 1: Create a Directory for the Website

```
sudo mkdir -p /var/www/example.com/html
```

```
sudo chown -R $USER:$USER /var/www/example.com/html
```

```
sudo chmod -R 755 /var/www
```

Step 2: Create an Index File

```
sudo nano /var/www/example.com/html/index.html
```

Add:

```
<html>
  <head><title>Welcome to Example.com</title></head>
  <body><h1>Nginx is running Example.com!</h1></body>
</html>
```

Step 3: Create a Server Block Configuration

```
sudo nano /etc/nginx/sites-available/example.com
```

Add:

```
server {
    listen 80;
    server_name example.com www.example.com;
    root /var/www/example.com/html;
    index index.html;

    access_log /var/log/nginx/example.com_access.log;
    error_log /var/log/nginx/example.com_error.log;
}
```

Step 4: Enable the Site and Reload Nginx

```
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/
```

```
sudo systemctl restart nginx
```

5. Allow Firewall Access

```
sudo ufw allow 80/tcp   # Ubuntu/Debian
```

```
sudo firewall-cmd --permanent --add-service=http  # CentOS/RHEL
```

```
sudo firewall-cmd --reload
```

Chapter 4: Enabling HTTPS with SSL Certificates

1. Installing Let's Encrypt SSL for Apache/Nginx

```
sudo apt install certbot python3-certbot-apache  # Apache
```

```
sudo apt install certbot python3-certbot-nginx   # Nginx
```

2. Obtain and Install an SSL Certificate

For Apache:

```
sudo certbot --apache -d example.com -d www.example.com
```

For Nginx:

```
sudo certbot --nginx -d example.com -d www.example.com
```

3. Automatically Renew SSL Certificates

```
sudo certbot renew --dry-run
```

Chapter 5: Case Study – Hosting a Website with Apache and Nginx

Scenario:

A company wants to host its website using **Apache** and set up a **reverse proxy with Nginx** for better performance.

Solution:

1. **Apache hosts the main website** on port 8080.
2. **Nginx is configured as a reverse proxy** to forward requests to Apache.

Nginx Reverse Proxy Configuration

```
sudo nano /etc/nginx/sites-available/reverse-proxy
```

Add:

```
server {
  listen 80;
  server_name example.com;

  location / {
    proxy_pass http://127.0.0.1:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

Enable the site and restart Nginx:

```
sudo ln -s /etc/nginx/sites-available/reverse-proxy /etc/nginx/sites-enabled/
```

```
sudo systemctl restart nginx
```

Outcome:

- **Nginx handles incoming requests** and forwards them to Apache.
- **Performance is improved** with Nginx caching.
- **HTTPS is enabled** for security.

CHAPTER 6: EXERCISE

1. **Install and configure an Apache web server with a virtual host.**
2. **Set up an Nginx server block for a website.**
3. **Enable SSL with Let's Encrypt on Apache or Nginx.**
4. **Configure Nginx as a reverse proxy for Apache.**
5. **Monitor web server logs for errors and access patterns.**

CONCLUSION

Configuring Apache and Nginx allows you to **host, secure, and optimize websites** efficiently. By mastering **virtual hosts, SSL, reverse proxy, and performance tuning**

MANAGING SERVICES & DAEMONS IN LINUX

CHAPTER 1: INTRODUCTION TO SERVICES AND DAEMONS IN LINUX

What Are Services and Daemons?

In Linux, **services** and **daemons** are background processes that run without direct user interaction. They handle essential tasks such as:

- **Networking (e.g., SSH, DNS, DHCP)**
- **Web hosting (e.g., Apache, Nginx)**
- **Logging and monitoring (e.g., syslog, cron)**
- **Security (e.g., firewall, Fail2Ban)**

Differences Between Services and Daemons

Feature	Service	Daemon
Purpose	Manages system functionalities	Background processes for specific tasks
Runs On	System startup or manually	On demand or continuously
Example	Apache (httpd), MySQL (mysqld)	cron, syslogd, dbus

Linux systems manage these processes using **systemd, SysVinit, or Upstart**, depending on the distribution.

This chapter covers **starting, stopping, enabling, disabling, and monitoring services and daemons** using systemctl, service, and chkconfig.

Chapter 2: Managing Services with systemd and systemctl

Most modern Linux distributions use **systemd**, which provides **systemctl** for service management.

1. Checking Service Status

To check if a service is running:

```
sudo systemctl status apache2  # Ubuntu/Debian

sudo systemctl status httpd    # CentOS/RHEL
```

Example output:

```
● apache2.service - The Apache HTTP Server

  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)

  Active: active (running) since Wed 2023-02-01 12:30:00 UTC; 10min ago
```

2. Starting and Stopping Services

Action	Command
Start a service	sudo systemctl start <service>
Stop a service	sudo systemctl stop <service>
Restart a service	sudo systemctl restart <service>
Reload service config	sudo systemctl reload <service>

Example:

```
sudo systemctl restart nginx
```

3. Enabling and Disabling Services

To **start a service automatically at boot**:

```
sudo systemctl enable ssh
```

To **disable it from starting at boot**:

```
sudo systemctl disable ssh
```

To check if a service is enabled:

```
sudo systemctl is-enabled apache2
```

Chapter 3: Managing Services with SysVinit (service and chkconfig)

Older Linux distributions (before systemd) use **SysVinit** for service management.

1. Checking Service Status (SysVinit)

```
sudo service apache2 status
```

2. Starting and Stopping Services

```
sudo service ssh start
```

```
sudo service ssh stop
```

3. Enabling and Disabling Services Using chkconfig

To list all startup services:

```
chkconfig --list
```

To enable a service at boot:

```
chkconfig httpd on
```

To disable it:

```
chkconfig httpd off
```

Chapter 4: Monitoring Running Daemons

1. Listing All Running Services

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

To list **failed services**:

```
sudo systemctl --failed
```

2. Checking System Logs for Service Failures

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

3. Viewing Active Daemons

To list all active daemons:

```
ps aux | grep daemon
```

To track real-time resource usage of daemons:

```
top
```

Chapter 5: Creating and Managing Custom Services

System administrators often need to create **custom services** for running scripts or applications in the background.

1. Creating a Custom Systemd Service

Create a service file:

```
sudo nano /etc/systemd/system/customscript.service
```

Add the following configuration:

```
[Unit]
Description=My Custom Script
After=network.target

[Service]
ExecStart=/usr/local/bin/myscript.sh
Restart=always
User=root

[Install]
WantedBy=multi-user.target
```

2. Enabling the Custom Service

Reload systemd and enable the service:

```
sudo systemctl daemon-reload
sudo systemctl enable customscript
sudo systemctl start customscript
```

To check its status:

```
sudo systemctl status customscript
```

Chapter 6: Case Study – Automating a Backup Service

Scenario:

A company needs to **automate daily backups** of critical files using a **background service**.

Solution:

1. **Create a backup script (/usr/local/bin/backup.sh)**:
2. #!/bin/bash
3. tar -czf /backups/home_backup_$(date +%F).tar.gz /home/
4. **Make it executable**:
5. chmod +x /usr/local/bin/backup.sh
6. **Create a systemd service file (/etc/systemd/system/backup.service)**:
7. [Unit]
8. Description=Daily Backup Service
9. After=network.target
10.
11. [Service]
12. ExecStart=/usr/local/bin/backup.sh
13. Restart=always
14. User=root
15.
16. [Install]
17. WantedBy=multi-user.target

18. **Enable and start the service**:
19. sudo systemctl daemon-reload
20. sudo systemctl enable backup
21. sudo systemctl start backup
22. **Verify the service is running**:
23. sudo systemctl status backup

Outcome:

- The system now **automatically backs up home directories daily**.
- **Data loss risk is minimized** with regular backups.

CHAPTER 7: EXERCISE

1. **Start, stop, and restart the SSH service using systemctl.**
2. **List all enabled services at system startup.**
3. **Create a systemd service that runs a script every 10 minutes.**
4. **Use journalctl to check logs for a failed service.**
5. **Disable an unused service (e.g., FTP) and verify it's no longer active.**

CONCLUSION

Managing services and daemons efficiently allows **smooth system operation and automation**. By mastering **systemd, SysVinit, and monitoring tools**, administrators can ensure **high availability, performance, and security** of Linux services.

Assignment Solution: Setting Up an Apache or Nginx Web Server

Objective

This assignment provides a **step-by-step guide** to installing and configuring an **Apache or Nginx web server** on a Linux system. By the end of this guide, you will have a working web server serving a **basic webpage** with proper firewall settings and a secure configuration.

Step 1: Choose and Install the Web Server

You can install either **Apache (httpd)** or **Nginx** based on your preference.

For Apache Web Server

1. Install Apache

- **On Debian/Ubuntu:**
- sudo apt update
- sudo apt install apache2 -y
- **On CentOS/RHEL:**
- sudo yum install httpd -y

2. Start and Enable Apache

sudo systemctl start apache2   # Ubuntu/Debian

sudo systemctl enable apache2

```
sudo systemctl start httpd     # CentOS/RHEL

sudo systemctl enable httpd
```

3. Verify Apache Installation

Open a web browser and enter your server's IP address:

http://your-server-ip

You should see the **default Apache web page**.

For Nginx Web Server

1. Install Nginx

- **On Debian/Ubuntu:**
- sudo apt update
- sudo apt install nginx -y
- **On CentOS/RHEL:**
- sudo yum install epel-release -y
- sudo yum install nginx -y

2. Start and Enable Nginx

```
sudo systemctl start nginx

sudo systemctl enable nginx
```

3. Verify Nginx Installation

Open a browser and enter:

http://your-server-ip

If successful, you will see the **default Nginx welcome page**.

Step 2: Configure Virtual Hosts (Apache) or Server Blocks (Nginx)

For Apache - Setting Up a Virtual Host

1. Create a Directory for Your Website

```
sudo mkdir -p /var/www/example.com/html

sudo chown -R $USER:$USER /var/www/example.com/html

sudo chmod -R 755 /var/www
```

2. Create an Index Page

```
sudo nano /var/www/example.com/html/index.html
```

Add the following:

```
<html>

  <head><title>Welcome to Example.com</title></head>

  <body><h1>Apache Virtual Host is Working!</h1></body>

</html>
```

3. Configure the Virtual Host File

```
sudo nano /etc/apache2/sites-available/example.com.conf  # Ubuntu/Debian

sudo nano /etc/httpd/conf.d/example.com.conf  # CentOS/RHEL
```

Add:

```
<VirtualHost *:80>
   ServerAdmin admin@example.com
   ServerName example.com
   ServerAlias www.example.com
   DocumentRoot /var/www/example.com/html
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

4. Enable the Virtual Host and Restart Apache

```
sudo a2ensite example.com.conf  # Ubuntu/Debian
sudo systemctl reload apache2    # Ubuntu/Debian
sudo systemctl restart httpd     # CentOS/RHEL
```

For Nginx - Setting Up a Server Block

1. Create a Directory for Your Website

```
sudo mkdir -p /var/www/example.com/html
sudo chown -R $USER:$USER /var/www/example.com/html
sudo chmod -R 755 /var/www
```

2. Create an Index Page

```
sudo nano /var/www/example.com/html/index.html
```

Add:

```
<html>
  <head><title>Welcome to Example.com</title></head>
  <body><h1>Nginx Server Block is Working!</h1></body>
</html>
```

3. Configure the Nginx Server Block

```
sudo nano /etc/nginx/sites-available/example.com
```

Add:

```
server {
   listen 80;
   server_name example.com www.example.com;
   root /var/www/example.com/html;
   index index.html;

   access_log /var/log/nginx/example.com_access.log;
   error_log /var/log/nginx/example.com_error.log;
}
```

4. Enable the Site and Restart Nginx

```
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/

sudo systemctl restart nginx
```

Step 3: Configure the Firewall

To allow HTTP and HTTPS traffic through the firewall:

On Ubuntu/Debian (UFW Firewall)

```
sudo ufw allow 80/tcp

sudo ufw allow 443/tcp

sudo ufw reload
```

On CentOS/RHEL (firewalld)

```
sudo firewall-cmd --permanent --add-service=http

sudo firewall-cmd --permanent --add-service=https

sudo firewall-cmd --reload
```

Step 4: Enable HTTPS with SSL Certificates

1. Install Let's Encrypt SSL for Apache or Nginx

```
sudo apt install certbot python3-certbot-apache  # Apache

sudo apt install certbot python3-certbot-nginx   # Nginx
```

2. Obtain and Install an SSL Certificate

For Apache:

```
sudo certbot --apache -d example.com -d www.example.com
```

For Nginx:

```
sudo certbot --nginx -d example.com -d www.example.com
```

3. Enable Automatic SSL Renewal

```
sudo certbot renew --dry-run
```

STEP 5: VERIFY AND MONITOR THE WEB SERVER

1. Check Server Logs for Errors

- **For Apache:**
- sudo tail -f /var/log/apache2/error.log
- **For Nginx:**
- sudo tail -f /var/log/nginx/error.log

2. Check If the Server Is Listening on Port 80 and 443

```
sudo netstat -tulnp | grep LISTEN
```

3. Test the Website from a Browser

Enter:

http://example.com

https://example.com

STEP 6: CASE STUDY – DEPLOYING A WEBSITE WITH APACHE AND NGINX

Scenario:

A startup wants to **host their company website** on a Linux server using **Apache** for static pages and **Nginx as a reverse proxy**.

Solution:

1. **Apache serves the main website** on port 8080.
2. **Nginx acts as a reverse proxy**, forwarding requests from users to Apache.

Nginx Reverse Proxy Configuration

```
sudo nano /etc/nginx/sites-available/reverse-proxy
```

Add:

```
server {
   listen 80;
   server_name example.com;

   location / {
      proxy_pass http://127.0.0.1:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
   }
}
```

Enable and restart:

```
sudo ln -s /etc/nginx/sites-available/reverse-proxy /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

Outcome:

- **Nginx handles incoming requests** and forwards them to Apache.
- **Performance is improved** with caching.
- **HTTPS is enabled** for security.

CONCLUSION

By following this guide, you successfully:

✅ **Installed and configured Apache or Nginx**
✅ **Created virtual hosts or server blocks**
✅ **Enabled firewall rules for security**
✅ **Implemented SSL for HTTPS support**

ASSIGNMENT SOLUTION: CONFIGURING SSH FOR SECURE REMOTE ACCESS

Objective

This assignment provides a **step-by-step guide** to configuring **Secure Shell (SSH)** for remote access in Linux. By the end of this guide, you will have a **securely configured SSH server**, allowing encrypted remote login and disabling unnecessary security risks.

STEP 1: INSTALL AND ENABLE THE SSH SERVER

1. Install OpenSSH Server

Most Linux distributions come with **OpenSSH** pre-installed. If not, install it using the following commands:

- **On Debian/Ubuntu:**
- sudo apt update
- sudo apt install openssh-server -y
- **On CentOS/RHEL:**
- sudo yum install openssh-server -y

2. Start and Enable SSH Service

Once installed, start the SSH service:

sudo systemctl start ssh   # Ubuntu/Debian

sudo systemctl start sshd   # CentOS/RHEL

To ensure SSH starts automatically on system boot:

```
sudo systemctl enable ssh     # Ubuntu/Debian
sudo systemctl enable sshd    # CentOS/RHEL
```

To verify the SSH service is running:

```
sudo systemctl status ssh
```

Example output:

```
● ssh.service - OpenSSH server daemon
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-02-01 12:30:00 UTC; 5min ago
```

Step 2: Configure Firewall for SSH Access

To allow SSH connections through the firewall, run:

- **For Ubuntu/Debian (UFW Firewall):**
- `sudo ufw allow 22/tcp`
- `sudo ufw reload`
- **For CentOS/RHEL (firewalld):**
- `sudo firewall-cmd --permanent --add-service=ssh`
- `sudo firewall-cmd --reload`

To check if the firewall rules are applied:

```
sudo ufw status   # Ubuntu/Debian
sudo firewall-cmd --list-all  # CentOS/RHEL
```

Step 3: Connecting to the SSH Server

From another machine, use the SSH command to connect:

```
ssh username@server_ip
```

Example:

```
ssh alice@192.168.1.100
```

If prompted with a fingerprint confirmation, type yes and press **Enter**.

To exit the SSH session, type:

```
exit
```

Step 4: Secure SSH Configuration

The SSH default settings can be modified to **increase security**.

1. Edit the SSH Configuration File

```
sudo nano /etc/ssh/sshd_config
```

2. Change the SSH Port (Optional)

By default, SSH listens on **port 22**. Change this to **a non-standard port** (e.g., 2222) to reduce automated attacks.

Find and change:

```
#Port 22
```

To:

```
Port 2222
```

Then, **allow the new port** in the firewall:

- **For UFW (Ubuntu/Debian):**
- sudo ufw allow 2222/tcp
- sudo ufw reload
- **For firewalld (CentOS/RHEL):**
- sudo firewall-cmd --permanent --add-port=2222/tcp
- sudo firewall-cmd --reload

Restart SSH to apply changes:

sudo systemctl restart ssh

To connect with the new port:

ssh -p 2222 alice@192.168.1.100

---

3. Disable Root Login

For security, disable direct **root login** via SSH.

Find:

PermitRootLogin yes

Change it to:

PermitRootLogin no

Save the file and restart SSH:

sudo systemctl restart ssh

---

4. Allow Only Specific Users

Limit SSH access to specific users to prevent unauthorized logins.

Find and add:

```
AllowUsers alice bob
```

Restart SSH:

```
sudo systemctl restart ssh
```

---

5. Limit Failed Login Attempts

To **prevent brute-force attacks**, set a limit for incorrect login attempts.

Find:

```
MaxAuthTries 6
```

Change it to:

```
MaxAuthTries 3
```

Restart SSH:

```
sudo systemctl restart ssh
```

STEP 5: SET UP SSH KEY-BASED AUTHENTICATION (PASSWORDLESS LOGIN)

Using SSH keys enhances security by **eliminating the need for passwords**.

1. Generate an SSH Key Pair

On the **client machine**, run:

```
ssh-keygen -t rsa -b 4096
```

Press **Enter** three times to accept the default location (~/.ssh/id_rsa).

2. Copy the Public Key to the Server

```
ssh-copy-id alice@192.168.1.100
```

Alternatively, manually copy it:

```
scp ~/.ssh/id_rsa.pub alice@192.168.1.100:~/
```

```
ssh alice@192.168.1.100
```

```
mkdir -p ~/.ssh && cat id_rsa.pub >> ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys
```

3. Disable Password Authentication

On the server, edit:

```
sudo nano /etc/ssh/sshd_config
```

Find:

```
PasswordAuthentication yes
```

Change it to:

```
PasswordAuthentication no
```

Restart SSH:

```
sudo systemctl restart ssh
```

Now, you can log in without entering a password.

STEP 6: IMPLEMENTING FAIL2BAN FOR SSH PROTECTION

To block repeated failed SSH login attempts, install **Fail2Ban**.

- **On Ubuntu/Debian:**
- sudo apt install fail2ban -y
- **On CentOS/RHEL:**
- sudo yum install fail2ban -y

Create a custom Fail2Ban rule for SSH:

```
sudo nano /etc/fail2ban/jail.local
```

Add:

```
[sshd]

enabled = true

port = 2222

maxretry = 3

bantime = 600
```

Restart Fail2Ban:

```
sudo systemctl restart fail2ban
```

To check banned IPs:

```
sudo fail2ban-client status sshd
```

STEP 7: MONITORING SSH ACCESS AND LOGS

To **check SSH login attempts**:

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

or

```
sudo cat /var/log/auth.log | grep "sshd"
```

To see **active SSH connections**:

```
who
```

To **view failed login attempts**:

```
sudo grep "Failed password" /var/log/auth.log
```

Case Study – Securing Remote SSH Access for a Company

Scenario:

A company wants to **securely allow remote SSH access** for system administrators while **preventing brute-force attacks and unauthorized logins**.

Solution:

1. **Enable SSH key authentication** instead of passwords.
2. **Change the SSH port** to a non-standard port (e.g., 2222).
3. **Restrict SSH access to specific users** (admin1, admin2).
4. **Disable root login** and enforce **max 3 login attempts**.
5. **Enable Fail2Ban** to block repeated login failures.

Outcome:

- **Secure remote access for authorized users.**
- **Reduced risk of brute-force attacks.**

- **Improved logging and monitoring of SSH activity.**

CONCLUSION

By following this guide, you have successfully:

✅ **Installed and configured SSH for remote access**

✅ **Enhanced security by changing ports and restricting access**

✅ **Enabled key-based authentication for passwordless login**

✅ **Set up Fail2Ban to prevent brute-force attacks**