Week - 1 Automation Using Shell Scripting & Python &RedHat Linux Administration

Shell Scripting

- Basics of Shell Scripting 15
- Real-Time Scenarios We have in-shell Scripting. 15

Red Hat Linux Administration

- Introduction to Red Hat Linux 20
- File System Management 20
- User and Group Administration 20
- Package Management with Yum 20-30
- System Services and systemd 30-40
- Networking Configuration 30-40
- Networking Concepts - SNAT, DNAT, IP, Netmask 30-40
- Security and Permissions 20-30
- System Performance Monitoring 10-15
- Storage Management 15-20
- Backup and Restore 15-20
- Kernel and Module Management 15
- Remote Access with SSH 15
- CPU Scheduling, Job Scheduling 20

Detailed Booklet - WEEK 1

1) Shell Scripting

- Introduction to Shell Scripting
- Writing and Executing Shell Scripts
- Variables and Data Types
- Control Structures (if, else, loops)
- Functions in Shell Scripts

What is Terminal?

While running Ubuntu, Linux Mint, or any other Linux distribution, we interact with the shell by using the terminal.

What Is Shell and Shell Scripting?

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something that the kernel can understand. It is a command language interpreter that executes commands read from input devices such as

keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Usually, shells are interactive, which means they accept commands as input from users and execute them. However, sometimes we want to execute a bunch of commands routinely, so we have to type in all commands each time in the terminal.

As a shell can also take commands as input from a file, we can write these commands in a file and execute them in the shell to avoid this repetitive work. These files are called **Shell Scripts** or Shell Programs.

Why do we need shell scripts?

There are many reasons to write shell scripts:

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups.
- System monitoring
- Adding new functionality to the shell etc.

Variable: A shell variable is a character string in a shell that stores some value.

```
#!/bin/bash

VAR_1="Devil"
VAR_2="OWL"

echo "$VAR_1$VAR_2"
```

Output:

```
DevilOWL
```

Environment Variable:
These variables are commonly used to configure the behavior of script and programs that are run by the shell. Environment variables are only created once, after which they can be used by any user.

For example:

`export PATH=/usr/local/bin:$PATH` would add `/usr/local/bin` to the beginning of the shell's search path for executable programs.

`for` statement in Shell Script in Linux
The for loop operates on lists of items. It repeats a set of commands for every item in a list. Here **var** is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Example 1:

```
#/bin/bash
for <var> in <value1 value2 ... valuen>
do
   <command 1>
   <command 2>
   <etc>
done
```

#/bin/bash

Example 2:

```
for a in 1 2 3 4 5 6 7 8 9 10
do

# if a is equal to 5 break the loop
if [ $a == 5 ]
then
break
fi

# Print the value
echo “Iteration no $a”
done
```

Function

A function is a collection of statements that execute a specified task. Its main goal is to break down a complicated procedure into simpler subroutines that can subsequently be used to accomplish the more complex routine. For the following reasons, functions are popular:

- Assist with code reuse.
- Enhance the program’s readability.
- Modularize the software.
- Allow for easy maintenance.

Example:

```
function_name(){
   // body of the function
}
```

2) Real-time scenarios in Shell Scripting

- Practical examples of Shell Scripting in real-world scenarios
- Automation tasks with Shell Scripts
- Error handling and debugging in Shell Scripts
- Create some scripts for demonstration

Automating Backup Tasks:
Scenario: For doing regularly backing up important files or databases.

System Monitoring and Reporting:
Scenario: For monitoring the system resources and generating periodic reports.

User Account Management:
Scenario: To automate the creation and deletion of user accounts.

Log Rotation:
Scenario: For managing and rotating log files to prevent them from consuming too much disk space.

Website Availability Monitoring:
Scenario: To automate the process of checking website availability.

—----------Red Hat Linux Administration—----------

3) Introduction to Red Hat Linux

- Overview of Red Hat Linux
- Installation and Configuration
- Basic Commands and Navigation
- File System Hierarchy

What Is Redhat Linux?

Red Hat Linux is an enterprise-grade Linux distribution developed and maintained by Red Hat, Inc. It is designed to provide a stable, secure, and well-supported operating system for

businesses, organizations, and individual users. Red Hat Linux incorporates open-source software and features a subscription-based support model, ensuring timely updates, security patches, and access to a wealth of software packages through the Red Hat package management system. Red Hat Linux is widely used in data centers, cloud environments, and mission-critical enterprise applications.

What Is an Operating System?

An operating system (OS) is essential software that bridges hardware and user applications, providing vital services for efficient resource utilization. Its primary function is to enable the execution of software, including hosting servers. Without an OS, running servers, software, or programs would be impossible, highlighting its indispensable role in computing.

What is Kernel?

The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system. It manages the following resources of the Linux system –

- File management
- Process management
- I/O management
- Memory management
- Device management etc.

Redhat Linux Installation

We can install RedHat Linux directly on bare metal or via virtualization or in the cloud, In this training we will be Installing our Linux OS on top of AWS Cloud, We can go to AWS EC2 Service and there we can launch an Instance(OS), In this training we would be using both Redhat Linux and Amazon Linux, Amazon Linux work exactly like Redhat, almost all the Redhat commands we can run on Amazon Linux, So Let's launch a amazon linux in EC2, and then run the below commands. While showing the Linux commands

Some Basic Linux commands:

- #pwd: Print the current working directory.
- #ls: List files and directories.
- #cd: Change directory.
- #mkdir: Create an empty folder
- #su: to switch users, for eg, #sudo su root
  (sudo here is used for privilege escalation(It is like the "run as administrator option of Windows) and here with this command, we will switch to the root user

4) File System Management

- Working with Files and Directories in Linux

Here are some useful commands to work with Linux File System:

#mkdir: Create a new directory.
Example: mkdir new_directory
#mkdir dir1 dir2 dir3: create multiple dir in current loc
#mkdir -p /a/b/c/: Create a dir with multiple levels of nested dir
#mkdir -m <mode> dir_name: Create dir with specific permissions

#ls: List files and directories in the current working directory
#ls -l: display detailed info in long format
#ls -a: list file and directories along with hidden files
#ls -S: sorts files by size, with the largest first

#cd ~: move to a home directory
#cd <path>: move to a specified path

#rmdir dir_name: Remove an empty dir
#rmdir -r dir_name: Remove a dir and its contents recursively

#touch: Create an empty file.
Example: touch new_file.txt

#cp: Copy files or directories.
Example: cp file.txt /path/to/destination

#mv: Move or rename files or directories.
Example: mv file.txt new_location/file.txt

#rm: Remove/delete files or directories.
Example: rm file.txt
#rm -rf: Remove directories and it's contents forcefully

#cat: Display the contents of a file.
Example: cat file.txt
#cat > file_name: Create or append data to a file using user input

#head file_name: Display the first 10 lines of a file
#tail file_name: Display the last 10 lines of a file

#vim: Text editors for creating or modifying files.
Example: vim file.txt

- *To save the file content press - esc + : + w*

- *To come out from the file - esc + : + q*
- *To save and exit the file - esc + : + wq*
- *To insert the data in the file - i*

#cp source dest: copy files and directories from one directory to another directory
Example: cp ./a.py /root/

#mv source_file dest_dir: Move a file from one location to another
#mv old_filename new_filename: Rename a file
#mv source_dir dest_dir: Move a directory
#mv old_dir new_dir: Rename a directory

#grep: Search for patterns in files.
Example: grep pattern file.txt

5) User and Group Administration

- User and Group Management
- Adding and Removing Users
- Password Policies
- Group Permissions

Why We need User and Group Administration?

Access Control:
User and group administration forms the foundation for access control mechanisms, determining who can access what resources on the system.

Security:
Proper management of user accounts and groups enhances system security by enforcing authentication policies and controlling user privileges.

Resource Management:
Group administration simplifies the assignment of permissions to multiple users simultaneously, facilitating efficient resource management.

Accountability:
User administration allows for tracking individual user activities on the system, enhancing accountability and auditability.

User & Group Management Commands:

- *#whoami: Displays the current username.*

- *#groupadd teamA: Creates a new group named "teamA."*
- *#cat /etc/group | grep teamA: Displays information about the "teamA" group from the /etc/group file, /etc/group file has the list of all groups and associated user with them,*
- *#useradd -G teamA userX: Adds a new user named "userX" to the "teamA" group.*
- *#useradd -G teamA userY: Adds another user named "userY" to the "teamA" group.*
- *#cat /etc/group | grep teamA: Displays updated information about the "teamA" group.*
- *#passwd userX: Sets a password for the "userX" user.*
- *#passwd userY: Sets a password for the "userY" user.*
- *#mkdir /home/teamA: Creates a directory named "teamA" in the /home directory.*
- *#ls -l -a: Lists all files*
- *#chown :teamA /home/teamA/: Changes the group ownership of the "teamA" directory, initially root was the group owner, but now “teamA” is the group owner*
- *#chmod g+w /home/teamA/: Adds write permission for the group to the "teamA" directory.*
- *#chmod o-rx teamA: Removes read and execute permissions for others on the "teamA" directory.*
- *#su - userX: Switches to the "userX" user.*
- *#whoami: Displays the current username (now "userX").*
- *#cd /home/teamA: Changes the current directory to "teamA."*
- *#touch userX-file.txt: Creates an empty file named "userX-file.txt."*
- *#ls -lrt: Lists files in long format, sorted by modification time (includes the new file).*
- *#chown :teamA userX-file.txt: Changes the group ownership of "userX-file.txt" to "teamA."*
- *ls -lrt: Lists files in long format, sorted by modification time (includes updated ownership).*
- *#exit: Exits the "userX" user session.*
- *#su - userY: Switches to the "userY" user.*

- *#cd /home/teamA: Changes the current directory to "teamA."*
- *#ls -l | grep userX-file.txt: Lists details of the file "userX-file.txt" in the current directory.*
- *#echo "This is userY's comment" > userX-file.txt: Adds a comment to "userX-file.txt."*
- *#cat userX-file.txt: Displays the content of "userX-file.txt."*
- *#exit: Exits the "userY" user session.*

6) Package Management with Yum

- Introduction to Yum Package Manager
- Installing, Updating, and Removing Packages
- Repository Configuration
- Dependency Management

What is Yum?

YUM is a package management utility for RPM (Red Hat Package Manager) based systems, such as Red Hat Enterprise Linux, CentOS, Fedora, and AWS Linux. It simplifies the process of installing, updating, and removing software packages on Linux systems. Here are some common YUM commands:

Configure yum repository

- vi /etc/yum.repos.d/myrepo.repo

```
[myrepo]
name=My Custom Repository
baseurl=https://example.com/repo/
enabled=1
gpgcheck=1
gpgkey=https://example.com/repo/RPM-GPG-KEY
```

Save and Exit - Esc + : + wq

name: A descriptive name for the repository.
baseurl: The URL where the repository is hosted.
enabled: Set to 1 to enable the repository.
gpgcheck: Set to 1 to enable GPG signature checking.
gpgkey: URL to the GPG key for package verification.

Now do try installing the packages with **#yum install**

- **#yum list:** List all available packages.
- **#yum search package_name:** Search for a specific package.
- **#yum info package_name:** Display detailed information about a package.
- **#yum install package_name**: Install a package.
- **#yum update:** Update all installed packages to the latest version.
- **#yum update package_name:** Update a specific package to the latest version.
- **#yum remove package_name:** Remove or uninstall a package
- **#yum list installed:** List all installed packages
- **#yum clean all:** Clean the package cache, removing downloaded packages.
- **#yum repolist:** Display the repositories and their status.
- **#yum history:** Display a transaction history of package installations, updates, and removals.

Example:
Install some of the below packages

- httpd
- vim
- wget

7) System Services and systemd

- Understanding systemd
- Managing Services
- systemctl commands
- Troubleshooting Services

Introduction to systemd

systemd is a system and service manager for Linux operating systems. It is responsible for initializing and managing system services, controlling the startup process, and handling various aspects of system management.

systemctl command, is the central management tool for controlling the init system. We will cover how to manage services, check statuses, change system states, and work with the configuration files.

Service Management
The fundamental purpose of an init system is to initialize the components that must be started after the Linux kernel is booted (traditionally known as “userland” components). The init system is also used to manage services and daemons for the server at any point while

the system is running. With that in mind, we will start with some basic service management operations.

In systemd, the target of most actions are “units”, which are resources that systemd knows how to manage. Units are categorized by the type of resource they represent and they are defined with files known as unit files. The type of each unit can be inferred from the suffix on the end of the file.

For service management tasks, the target unit will be **service units**, which have unit files with a suffix of *.service*. However, for most service management commands, you can actually leave off the .service suffix, as systemd is smart enough to know that you probably want to operate on a service when using service management commands.

Starting and Stopping Services

To start a systemd service, executing instructions in the service’s unit file, use the **start** command. If you are running as a non-root user, you will have to use sudo since this will affect the state of the operating system:

#sudo systemctl start application.service

To stop a currently running service, you can use the stop command instead:
#sudo systemctl stop application.service

Restarting and Reloading
To restart a running service, you can use the restart command:

#sudo systemctl restart application.service

If the application in question is able to reload its configuration files (without restarting), you can issue the reload command to initiate that process:
#sudo systemctl reload application.service

If you are unsure whether the service has the functionality to reload its configuration, you can issue the reload-or-restart command. This will reload the configuration in-place if available. Otherwise, it will restart the service so the new configuration is picked up:

#sudo systemctl reload-or-restart application.service

Enabling and Disabling Services

The above commands are useful for starting or stopping services during the current session. To tell systemd to start services automatically at boot, We have to enable them.

To start a service at boot, use the enable command:

#sudo systemctl enable application.service

This will create a symbolic link from the system's copy of the service file (usually in */lib/systemd/system or /etc/systemd/system*) into the location on disk where systemd looks for autostart files (usually /etc/systemd/system/some_target.target.wants.)

To disable the service from starting automatically:
#sudo systemctl disable application.service

This will remove the symbolic link that indicates that the service should be started automatically.

Checking the Status of Services
To check the status of a service:

#systemctl status application.service

This gives you a nice overview of the current status of the application, notifying you of any problems and any actions that may be required.

There are also methods for checking for specific states. For instance, to check to see if a unit is currently active (running), we can use the is-active command:
#systemctl is-active application.service

This will return the current unit state, which is usually active or inactive. The exit code will be "0" if it is active, making the result simpler to parse in shell scripts.

To see if the unit is enabled:
#systemctl is-enabled application.service

This will output whether the service is enabled or disabled and will again set the exit code to "0" or "1" depending on the answer to the command question.

A third check is whether the unit is in a failed state. This indicates that there was a problem starting the unit in question:
#systemctl is-failed application.service

This will return active if it is running properly or failed if an error occurred. If the unit was intentionally stopped, it may return unknown or inactive. An exit status of "0" indicates that a failure occurred and an exit status of "1" indicates any other status.

8) Networking Configuration

- Network Configuration Basics
- IP Addressing and Subnetting
- Configuring Network Interfaces
- Routing and Firewall Configuration

IP Addressing and Subnetting:

An IP address is a unique numerical label assigned to each device on a network. The true meaning of an IP is *"any number which is of 32-bit size is a valid IP address*" For example, 192.168.1.1 is an IPv4 address. #ifconfig

The **netmask** is used in conjunction with an IP address to define the network and host portions of the address. In this example, 255.255.255.0 indicates a subnet mask where the first 24 bits are network bits, and the last 8 bits are host bits. #ifconfig

There are 4 rules for having network connectivity between two systems

A) Both systems should have a valid IP Address
B) Both systems should have a physical connection, Which could be via wire or wireless
C) Both systems should be in the same network
D) A Private IP can only connect with another Private IP, Similarly a Public IP can connect with another Public IP only

How to find two systems are on the same network - Just convert the IP address of system 1 and netmask of system 1 in binary and do the & operation between them, And similarly do for system2 If they both return the same value, it means they are in same network

Assigning IP addresses to network interfaces:

#ifconfig <interface_name> <ip_address>
This command sets the IP address for a specific network interface. For example, if you want to assign the IP address 192.168.1.2 to the interface eth0, you would use ifconfig eth0 192.168.1.2.

Subnetting:
#ifconfig <interface_name> <ip_address> netmask <subnet_mask>

Subnetting involves dividing a network into smaller sub-networks. The netmask determines the size of the subnet. For example, if you want to assign the IP address 192.168.1.2 to the interface eth0 with a subnet mask of 255.255.255.0, you would use ifconfig eth0 192.168.1.2 netmask 255.255.255.0.

Local Area Network (LAN):

A Local Area Network (LAN) is a network that connects computers and devices within a limited geographical area, such as a home, office, or campus. LANs facilitate local communication and resource-sharing among connected devices.

Router:

A router is a networking device that connects multiple computer networks together and directs data traffic between them. It operates at the network layer (Layer 3) of the OSI model.
Key Functions:

Routing: Determines the optimal path for data packets to travel between networks.
Network Address Translation (NAT): Translates private IP addresses to a public IP address, allowing multiple devices in a local network to share a single public IP.
Firewall: Filters and controls incoming and outgoing network traffic based on an organization's previously established security policies.

Switch:

A switch is a networking device that connects devices within the same local network and uses MAC addresses to forward data to the correct destination. It operates at the data link layer (Layer 2) of the OSI model.

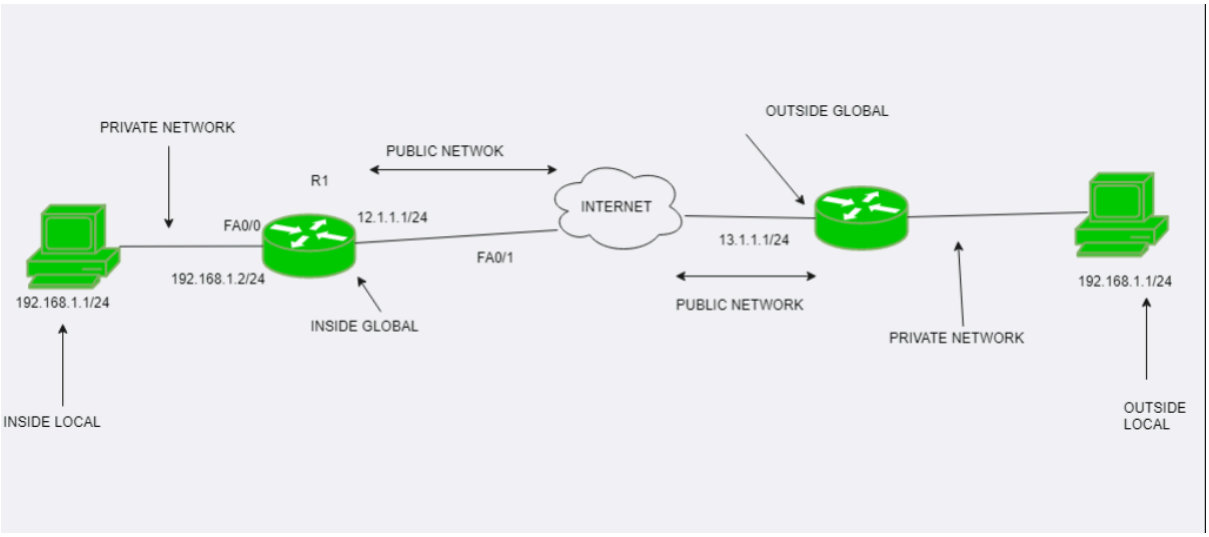
Network Address Translation (NAT):

NAT is a process that modifies network address information in packet headers while in transit, typically in a router. It enables multiple devices within a local network to share a single public IP address.

SNAT- SNAT is a technique that translates source IP addresses generally when connecting from a private IP address to a public IP address. It maps the source client IP address in a network package request to a translation defined on NAT. It is the most common form of NAT that is used when an internal host needs to initiate a session with an external host or public host.

DNAT- Destination NAT is used to modify the destination IP address and port of incoming packets. This is often used for port forwarding, redirecting traffic from one external port to an internal IP and port.

Network Diagram:



Explain, Routers, and switches, and also explain how network packages travel from a local system via router to the internet

9) Security and Permissions

- Security Best Practices
- User Authentication
- SELinux Overview
- Firewalls and Security Policies

DevOps engineers play a key role in implementing and managing security practices. In this comprehensive guide, we will delve into key security best practices, user authentication, SELinux, and firewall configurations specific to Red Hat Linux.

File Permissions:

#chmod: Change file permissions.
Example: chmod 755 file.txt
#chmod 755 file_name: Change permissions using octal notation
#chmod -R 755 directory_name: Change permissions recursively (including subdirectories)
#chmod u=rw,go=r file_name: Change permissions using symbolic notation
#chown new_owner file_name: Change the owner of a file
#chown new_owner:new_group file_name: Change the owner and group owner of a file
#chown :new_group file_name: Change the group owner of a file or directory

Security Best Practices:

User and Group Management:
Principle of Least Privilege (PoLP) dictates granting users only the permissions necessary for their tasks:

Creating a new user
#sudo useradd -m newuser

Granting sudo privileges
#sudo usermod -aG wheel newuser

Password Policies:
Enforce strong password policies:

Install and configure Pluggable Authentication Modules (PAM)
#sudo yum install -y pam_pwquality
Modify /etc/security/pwquality.conf for password complexity.

SSH Security:
Secure SSH configurations by disallowing root login, using key-based authentication, and changing the default SSH port:

Disable root login- for this go to /etc/ssh/sshd_config file
PermitRootLogin no

Change default SSH port
Port 2222

Disallow password authentication, And use ssh keys only for authentication
PasswordAuthentication no

SELinux Overview
1. Checking SELinux Status:
#sestatus

2. Changing SELinux Mode:

Change to Permissive mode
#sudo setenforce 0

Change to Enforcing mode
#sudo setenforce 1

Firewall and Security Policies

1. Firewalld Basics:

Install firewalld
#sudo yum install -y firewalld

Start and enable firewalld
#sudo systemctl start firewalld
#sudo systemctl enable firewalld

2. Opening Ports:

Allow incoming traffic on port 80
sudo firewall-cmd --add-port=80/tcp --permanent
sudo firewall-cmd --reload

3. Creating Zones:

Create a new zone
#sudo firewall-cmd --new-zone=myzone --permanent
Configure specific settings for the zone in /etc/firewalld/zones/myzone.xml.

10) System Performance Monitoring

- Monitoring System Resources
- Performance Metrics and Tools
- Troubleshooting Performance Issues
- Performance Tuning

System Performance Monitoring involves continuous tracking and analysis of a computer system's resources to ensure optimal performance and identify potential issues.

Importance
Monitoring is critical for proactive issue identification, capacity planning, and ensuring a seamless user experience in a dynamic environment.

System Resource Usage:

Command: *#top* or *#htop*

Description: Display real-time information about system resource usage, including CPU, memory, and processes.

CPU Usage:

Command: *#mpstat*, *#sar*, *#vmsta*t
Description: Monitor CPU usage, identify bottlenecks, and analyze trends over time.

Memory Usage:

Command: *#free, #vmstat*, *#sar*
Description: Check free and used memory, including swap usage. Identify potential memory leaks.

Disk I/O:

Command: *#iostat*, *#iotop*
Description: Monitor disk I/O operations, identify disk bottlenecks, and analyze read/write performance.

Network Activity:

Command: *#iftop, #nload, #netstat*
Description: Analyze network usage, identify high traffic, and monitor bandwidth.

Process Monitoring:

Command: *#ps, #pstree*
Description: View running processes, their resource consumption, and their relationships.

System Logs:

Command: *#journalctl, #dmesg*
Description: Check system logs for errors, warnings, and other messages. Use journalctl for systemd-based systems.

Performance Tuning:

Command: *#sysctl*
Description: Adjust kernel parameters to optimize system performance. Changes are usually made in the */etc/sysctl.conf* file.

Monitoring Tools:

Command: *#sar*, *#sysstat*
Description: The sysstat package includes tools like sar for collecting, reporting, and saving system activity information.

Hardware Information:

Command: *#lscpu,# lshw*
Description: Retrieve information about the system's hardware, including CPU, memory, and devices.

Security and Performance Auditing:

Command: *#auditd*
Description: Use auditd to collect and report security-related events and monitor system changes.

Custom Scripts and Alerts:

Command: Bash/Python scripts
Description: We can Develop custom scripts to automate monitoring tasks and trigger alerts based on predefined thresholds.

Container Orchestration Monitoring:

Command: *#docker stats, #kubectl top*
Description: For containerized environments, monitor resource usage and performance metrics for containers.

Graphical Tools:

Command: *gnome-system-monito*r, #htop
Description: Graphical tools for monitoring system performance in a user-friendly interface.
Third-Party Monitoring Solutions:

Tools: *Prometheus, Grafana, Nagios*
Description: We can also Integrate third-party monitoring solutions for advanced visualization, alerting, and historical data analysis.

11) Storage Management

Managing Disks and Partitions
Logical Volume Management (LVM)
RAID Configuration
Filesystem Management

1**. Managing Disks and Partitions**

Partitioning Basics:
Partitions are logical divisions on a disk that allow for the organization and separation of data. There are primary, extended, and logical partitions. Primary partitions are standalone partitions, while an extended partition can be subdivided into logical partitions.

Practical
The fdisk and parted commands are commonly used for disk management. For instance, to create a partition using fdisk:

#fdisk /dev/sdX
Where /dev/sdX is the target disk. The command enters the interactive partitioning tool, allowing you to create, delete, and modify partitions.

Once a partition is created, it needs to be formatted with a filesystem. The mkfs command is used for this purpose. For example:

#mkfs.ext4 /dev/sdXY
This command creates an ext4 filesystem on the specified partition (/dev/sdXY).

Introduction to LVM:
Logical Volume Management (LVM) provides a layer of abstraction between the operating system and physical storage. It allows for dynamic resizing of storage elements, making it a flexible solution for managing disk space.

Advantages of LVM:
LVM offers benefits such as the ability to resize volumes on-the-fly, create snapshots for backups, and manage storage across multiple disks more efficiently.

Configuring LVM:

Creating Physical Volumes:

#pvcreate /dev/sdXY
Building Volume Groups:
#vgcreate my_volume_group /dev/sdXY
Creating Logical Volumes:

#lvcreate -n my_logical_volume -L 20G my_volume_group
This example creates a 20GB logical volume named my_logical_volume within the volume group my_volume_group.

LVM Operations:

Resizing Logical Volumes:

#lvextend -L +10G /dev/my_volume_group/my_logical_volume
This command increases the size of the logical volume by 10GB.

Filesystem Management

A filesystem is a structure that organizes and stores data on storage devices. It includes files, directories, and metadata. Filesystems facilitate data access and storage organization.

Mounting and unmounting filesystems:

#mount /dev/sdXY /mnt/my_mount_point
This command mounts a filesystem located on /dev/sdXY to the specified mount point.

Configuring automatic mounts (/etc/fstab):
Edit the /etc/fstab file to include entries for automatic mounts during system boot.

12) Backup and Restore

- Data Backup Strategies
- Backup Tools and Utilities
- Restoring from Backups
- Backup Automation

What is Backup?

Backup and Restore involve the process of creating copies of important data to protect against data loss and facilitate recovery in case of system failures or disasters.

Importance Of Backup And Restore?

Data loss can have severe consequences, and having robust backup and restore mechanisms is crucial for maintaining data integrity and availability.

Data Backup Strategies

1) **Full Backup:** A complete copy of all selected data, It will create a tar file by archiving the data.

   # Perform a full backup of a directory
   #tar -cvzf backup.tar.gz /path/to/data

2) **Incremental Backup:** Backing up only the data that has changed since the last backup.

   # Perform an incremental backup, Here in the below example
   `***--listed-incremental=backup.snar***: Specifies a snapshot file (backup.snar) that keeps track of the state of files in previous backups. This allows tar to determine which files have changed since the last backup.

   #tar --listed-incremental=backup.snar -cvzf backup_incremental.tar.gz /path/to/data

3) **Differential Backup:** Backing up the data that has changed since the last full backup.
   # Create a differential backup -
   #find /path/to/source -type f -newer /path/to/full_backup.timestamp -exec cp --parents {} /path/to/differential_backup/ \;

   Here are the descriptions of all the options

   - find /path/to/source: Searches for files in the specified source directory.
   - -type f: Specifies that only regular files should be considered.
   - -newer /path/to/full_backup.timestamp: Limits the search to files that are newer than the timestamp of the last full backup. This assumes you have a file named full_backup.timestamp in the directory of the last full backup.
   - -exec cp --parents {} /path/to/differential_backup/ \;: Executes the cp command for each file found by find. The --parents option preserves the directory structure, and {} is replaced by the current file. The files are copied to the specified differential backup directory.

Backup Tools and Utilities

a) **tar Command**
   It is A versatile command-line tool for archiving and compressing files.
   #tar -czvf backup.tar.gz /path/to/source_directory

b) **rsync Command:** Efficient file-copying tool that synchronizes files and directories.

   # Sync files to a backup directory
   #rsync -av --delete /path/to/source/ /path/to/backup/

c) **dump and restore Commands:** Used for file system-level backups and restores.

   # Create a backup using dump
   #dump -0uf backup.dump /dev/sdX

   # Restore using restore

#restore -rf backup.dump

Restoring from Backups

a) **Full Restore:** Restoring all data from a full backup.

Restore from a full backup
#tar -xvzf backup.tar.gz -C /path/to/restore

b) **Incremental Restore:** Applying incremental backups to restore data.

Restore from an incremental backup
#tar --incremental -xvzf backup_incremental.tar.gz -C /path/to/restore

Backup Automation

a) **Cron Jobs**: Scheduled tasks for automation.

Schedule a daily backup at 2 AM
0 2 * * * tar -cvzf /backup/daily_backup.tar.gz /path/to/data

b) Ansible for Backup Automation: We will learn about ansible in future sessions

13) Kernel and Module Management

- Kernel Overview
- Managing Kernel Modules
- Kernel Configuration
- Kernel Upgrades

Kernel Information:

Command: *#uname -a, #cat /proc/version*
Description: Display information about the kernel version, architecture, and build information.

Kernel Parameters:

Command: *#sysctl -a*
Description: View and modify kernel parameters to tune system behavior. Changes can be made using the sysctl command or by editing /etc/sysctl.conf.
Load and Unload Kernel Modules:

Commands: *#lsmod, #modprobe, #rmmod*

Description: List loaded kernel modules, load a new module, and remove an existing module. The /etc/modules file can be used to specify modules that should be loaded at boot.

Kernel Module Information:

Command: *#modinfo <module_name>*
Description: Display detailed information about a specific kernel module, including its parameters and dependencies.

Kernel Logs:

Command: *#dmesg*
Description: View kernel messages and logs, including information about hardware, device discovery, and kernel module loading.

Kernel Panic Analysis:

Command: *#kdump*
Description: Set up and configure kdump for kernel crash analysis. This involves capturing kernel core dumps when a kernel panic occurs.

Kernel Module Auto-loading:

Directory: /etc/modules-load.d/
Description: Create files in the /etc/modules-load.d/ directory to specify modules that should be loaded at boot time. Each file contains module names.

Kernel Security and Hardening:

Command: *#grubby*
Description: Use tools like grubby to manage kernel boot parameters and configurations, ensuring a secure and optimized boot process.

Kernel Upgrade:

Commands: *#yum update kernel, #dnf update kernel*
Description: Use package management tools (yum or dnf) to update the kernel to the latest version available in the repositories.

Kernel Header Files:

Command: *#yum install kernel-devel*
Description: Install kernel header files to build external kernel modules, often required for third-party drivers.

Interrupt Handling:

Command: *#cat /proc/interrupts*

Description: Display information about interrupt usage, helping identify potential hardware or driver-related issues.

Kernel Tuning:

Files: /etc/sysctl.conf, /etc/sysctl.d/
Description: Adjust kernel parameters in configuration files to optimize system performance. Changes take effect upon reboot or by using sysctl -p.

14) Remote Access with SSH

- Secure Shell (SSH) Introduction
- SSH Configuration
- Key-based Authentication
- Tunneling and Port Forwarding

What is SSH?
Secure Shell (SSH) is a cryptographic network protocol used for secure communication over an unsecured network. It provides a secure channel for remote login, command execution, and other network services between two computers.

Key Features of SSH:

- **Encryption**: All communication between the client and server is encrypted.
- **Authentication**: Users can authenticate using passwords or public-key cryptography.
- **Secure File Transfer**: SSH includes tools like scp and sftp for secure file transfer.

SSH Configuration

The main configuration file for the SSH server is typically located at **/etc/ssh/sshd_config**.

Key configuration options include:

Port: Specifies the port on which the SSH server listens.
PermitRootLogin: Controls whether the root user can log in directly.
PasswordAuthentication: Enables or disables password authentication.

Now by default Password Authentication could be disabled, So We can go to the SSH configuration file and allow PasswordAuthentication and aslo PermitRootLogin, So that we can login to the root user.

Key-Based Authentication

Step 1: Generate SSH Key Pair

#ssh-keygen -t rsa -b 2048
This command generates a new SSH key pair using the RSA algorithm with a key size of 2048 bits. Press Enter to accept the default file location (~/.ssh/id_rsa) and an optional passphrase for added security.

Step 2: Copy the Public Key to the Server
Assuming you already have SSH access to the server using a password
#ssh-copy-id username@server_ip
Replace username with your actual username on the server, and server_ip with the actual IP address or hostname of your server. This command copies the public key to the ~/.ssh/authorized_keys file on the server.

If ssh-copy-id is not available on your system, you can manually copy the public key:
#cat ~/.ssh/id_rsa.pub | ssh username@server_ip 'cat >> ~/.ssh/authorized_keys'

Step 3: Verify Key-Based Authentication
Try logging in to the server:

#ssh username@server_ip
You should be able to log in without entering a password.

Additional Tips:

Permissions: Ensure that the ~/.ssh directory on the server has the correct permissions (700) and the ~/.ssh/authorized_keys file has the correct permissions (600).
Bash

#chmod 700 ~/.ssh
#chmod 600 ~/.ssh/authorized_keys

SSH Agent:

You can use the SSH agent to manage your private keys and avoid entering passphrases repeatedly.
With SSH key-based authentication, you enhance the security of your server by replacing password-based logins with cryptographic keys, making it more resistant to brute-force attacks.

15) CPU Scheduling, Job Scheduling

- Understanding CPU Scheduling
- Managing Processes
- Job Scheduling with cron
- Anacron for System Automation

Introduction to CPU Scheduling

CPU scheduling is a vital aspect of operating system management, involving the allocation of the CPU to processes. Efficient scheduling ensures optimal system performance by minimizing waiting times and maximizing resource utilization.

Scheduling Algorithms

Various scheduling algorithms, such as *First-Come-First-Serve (FCFS)*, *Shortest Job Next (SJN)*, and *Round Robin*, govern how processes are prioritized and executed. Each algorithm has its strengths and weaknesses, influencing system responsiveness and throughput.

Priority Scheduling

Priority-based scheduling assigns priorities to processes, allowing the CPU to favor those with higher priority. While effective, this approach requires careful balancing to prevent resource starvation and ensure fairness among tasks.

Managing Processes
Process Lifecycle
Processes undergo a lifecycle comprising creation, execution, suspension, and termination. Understanding these stages is crucial for efficient resource management.

Display CPU Scheduling Information:

#ps: Display information about active processes.
Example: #ps aux
Adjusting Process Priority:
nice: Adjust the priority of a process.
Example: *#nice -n 10 command*

Process States
Processes can exist in various states, including running, waiting, and ready. Transitioning between these states is orchestrated by the operating system scheduler, ensuring effective utilization of system resources.

View Process Status:
#top: Display real-time system statistics and a list of processes.
Example: #top

Kill a Process:
#kill: Terminate a process by sending a signal.
Example: *#kill -9 PID*

Display Process Information:

#pstree: Display a tree of processes.
Example: #pstree

Job Scheduling with cron

Introduction to cron
Cron is a time-based job scheduler in Unix-like operating systems. It enables users to schedule tasks at specified intervals, automating repetitive processes without manual intervention.

Edit User's crontab:
#crontab -e: Edit the user's crontab file.
Example: *#crontab -e*

View User's crontab:
crontab -l: Display the user's crontab entries.
Example: #crontab -l

Cron Syntax
Cron jobs are defined by a syntax that specifies when a task should run. This syntax includes minute, hour, day of the month, month, and day of the week fields, offering precise control over scheduling.

Examples of Cron Jobs
Users can schedule various tasks using cron, such as data backups, log rotation, and system maintenance. Cron jobs are initiated silently in the background, enhancing system efficiency.