**Week 6:**

**1. Introduction to Local Area Networks (LAN)**

- \*\*Planning and Setup\*\*: Involves selecting a network topology, purchasing necessary equipment, and installing hardware such as network cards and antennas.

- \*\*Configuring TCP/IP\*\*: Essential for using most networking applications in Linux, allowing communication within the LAN and with external networks.

**2. Network Interfaces**

- \*\*Naming Conventions\*\*: Interfaces are typically named sequentially (e.g., eth0 for ethernet, wlan0 for wireless).

- \*\*Key Interfaces\*\*:

- `eth0, eth1, ...`: Ethernet interfaces.

- `wlan0, wlan1, ...`: Wireless interfaces.

- `ppp0, ppp1, ...`: Dial-up interfaces.

- `bond0`: First bonded network device.

- `virbr0`: Internal bridge for virtual hosts.

- `lo`: Loopback interface.

**3. Command Line Tools for Network Management**

- \*\*ifconfig\*\*: Used to configure network interfaces.

- Display network configuration (`ifconfig`), manage IP addresses, and troubleshoot networks.

- \*\*ip command\*\*: Modern replacement for ifconfig, used to show and manipulate routing and network interfaces.

- Examples include `ip route` (to display routing information) and `ip -4 addr` (to display IPv4 addresses).

- \*\*netstat and ss commands\*\*: For monitoring network connections and performance.

- `netstat` displays active connections and routing table information.

- `ss` is used for viewing socket statistics and is faster than netstat.

**4. Network Configuration Files**

- Located in `/etc/sysconfig/network-scripts`.

- Files named `ifcfg-<device\_name>` are used to configure individual network interfaces.

**5. nmcli (Network Manager Command Line Interface)**

- Used for managing networking in CentOS 7 and newer versions.

- Commands like `nmcli con show` display all network connections, and `nmcli dev show` provides device details.

**6. tcpdump**

- Command-line utility used to capture and analyze network traffic.

- Useful for network troubleshooting.

**7. Managing Host Files**

- The host file (`/etc/hosts`) acts like a local DNS server, mapping hostnames to IP addresses.

- Editing the host file involves adding entries for specific IP addresses and domain names.

**8. Troubleshooting Commands**

- \*\*traceroute\*\*: Identifies the path packets take to a network host, useful for diagnosing connectivity issues.

- \*\*ping\*\*: Tests connectivity to another network host, providing information about the reachability and round-trip time.

**9. Practical Tips**

- Always ensure to update the network scripts and restart the network service after changes (`sudo /etc/init.d/network restart`).

- Use verbose and debugging options with commands to get detailed output when troubleshooting complex network issues.

**Week 11**

**1. Overview of Software Package Management**

- \*\*Purpose\*\*: To manage the installation, updating, and removal of software on Linux systems.

- \*\*Significance\*\*: Acts as a critical function within all Linux distributions, with package management quality being a major determinant of a distribution's overall quality.

**2. Package Management Systems**

- \*\*Types of Packages\*\*:

- `.deb`: Used by Debian, Ubuntu, Linux Mint, Raspbian.

- `.rpm`: Used by Fedora, CentOS, Red Hat Enterprise Linux, OpenSUSE.

- \*\*Package Files\*\*: Compressed collections containing software and metadata, including pre and post-installation scripts.

**3. Package Repositories**

- \*\*Function\*\*: Central hubs where packages are stored and made available to users.

- \*\*Types\*\*:

- Main repositories: Officially supported software.

- Third-party repositories: Additional software not included in the main repositories.

**4. Dependencies**

- \*\*Importance\*\*: Necessary to ensure that a software package functions correctly by having required components or libraries.

- \*\*Management\*\*: Package managers handle dependencies to avoid issues during installation.

**5. YUM (Yellowdog Updater, Modified)**

- \*\*Functionality\*\*:

- `yum update`: Updates all packages, with options to exclude or specify certain packages.

- `yum install`: Installs new software packages.

- `yum remove`: Removes installed packages.

- `yum check-update`: Lists packages that have updates available.

- \*\*Information Commands\*\*:

- `yum info`: Displays detailed information about packages.

- `yum list`: Lists available or installed packages.

**6. RPM Package Manager**

- \*\*Usage\*\*:

- `rpm -i`: Installs a package.

- `rpm -U`: Upgrades a package.

- `rpm -qa`: Displays all installed packages.

- `rpm -q`: Checks if a specific package is installed.

**7. Managing Package Caches**

- \*\*Commands\*\*:

- `yum clean packages`: Cleans out the package cache.

- `yum clean all`: Performs a complete clean of caches and metadata.

- `yum repolist`: Lists all active repositories.

**8. Practical Examples**

- \*\*YUM Usage\*\*: Demonstrated with installing Emacs and managing its dependencies.

### Exam Tips:

- \*\*Understand Command Functions\*\*: Know what each command does and when to use it.

- \*\*Dependency Management\*\*: Be aware of how package managers handle dependencies.

- \*\*Differences in Package Systems\*\*: Be able to distinguish between `.deb` and `.rpm` packages and their respective distributions.

**Week 12:**

**1. Basics of File and Directory Permissions**

- \*\*Permissions Types\*\*:

- `r`: Read - Allows opening and viewing a file.

- `w`: Write - Allows editing of the file.

- `x`: Execute - Allows execution of the file, not necessarily viewing or editing.

**2. Types of Users and Permissions**

- \*\*Root User\*\*: Has complete control over the system.

- \*\*Other Users\*\*: Typically have more limited capabilities and are often organized into groups with similar functions.

- \*\*Permission Levels\*\*: Root user assigns permissions based on the needs for file access.

**3. Granting Ownership and Group Management**

- \*\*Changing Ownership\*\*: Use the `chown <user> <file path>` command to change the ownership of a file to a different user.

- \*\*Changing Group\*\*: Use the `chgrp <group> <file path>` command to change the file's group.

**4. Understanding File and Directory Permission Notations**

- \*\*File Listing Indicators\*\*:

- `d`: Directory

- `-`: Regular file

- `L`: Link

- \*\*Permission Representation\*\*:

- `rwx`: Read, write, and execute

- `rw-`: Read and write

- `r-x`: Read and execute

- `---`: No permissions

**5. CHMOD Command: Changing Permissions**

- \*\*Octal Notation\*\*:

- Each set of permissions is represented as a single digit.

- `7` (rwx), `6` (rw-), `5` (r-x), etc.

- \*\*Common Settings\*\*:

- `777`: Full permissions for everyone. Generally not recommended.

- `755`: Owner can read, write, and execute. Others can read and execute. Common for programs.

- `700`: Owner can read, write, and execute. No permissions for others. Used for private directories.

- `644`: Owner can read and write; others can only read. Common for data files.

- `600`: Owner can read and write; others have no permissions. Used for private data files.

**6. Applying Permissions**

- \*\*Changing Permissions Using CHMOD\*\*:

- Using symbolic method: `chmod u+rwx,g+rx,o-rwx <file>` adds full permissions to user, read and execute to group, and removes all permissions from others.

- Using octal method: `chmod 755 <file>` sets permissions to `rwxr-xr-x`.

**7. Practical Examples and Tips**

- \*\*Applying Permissions\*\*:

- For a file to be executable as a program by all users: `chmod 755 <file>`

- For a personal file that should be readable and writable by the owner only: `chmod 600 <file>`

**Week 13:**

**1. Overview of Device Files**

- \*\*Location\*\*: All device files are located in the `/dev` directory.

- \*\*Types of Device Files\*\*:

- `b`: Block device - Allows random access to fixed-size blocks of data.

- `c`: Character device - Handles data as a stream, suitable for devices that produce or consume data serially.

- `p`: Pipe device - Used for passing information from one process to another.

- `s`: Socket device - Used for network communication and inter-process communication.

**2. Using `dd` for Device Management**

- \*\*Function\*\*: The `dd` command is used to copy a specified amount of data, performing any necessary conversions.

- \*\*Options\*\*:

- `if=file`: Specifies the input file.

- `of=file`: Specifies the output file.

- `bs=size`: Block size for reading and writing.

- `count=num`: Number of blocks to copy.

**3. Filesystem Management Commands**

- \*\*Viewing Partitions\*\*: Use `fdisk -l` to list all partitions on all disks.

- \*\*Creating a Partition\*\*:

- Use `n` for a new partition, specify type (`p` for primary, `l` for logical), and define the sectors.

- \*\*Deleting a Partition\*\*:

- Use `d` to delete a specific partition based on its number.

- \*\*Changing Partition Type\*\*:

- Use `t` to specify the partition type.

**4. Filesystem Formatting**

- \*\*Using `mkfs`\*\*:

- Format new partitions with `sudo mkfs.ext4 /dev/sda1` for an ext4 filesystem on the first partition of the first disk.

**5. Logical Volume and RAID Management**

- \*\*Logical Volumes\*\*: Allow more flexible management of disk space.

- \*\*RAID Levels\*\*:

- \*\*RAID 0 (Striping)\*\*: Spreads data across multiple disks for high performance but no redundancy.

- \*\*RAID 1 (Mirroring)\*\*: Copies the same data on two disks; provides redundancy.

- \*\*RAID 5 (Single Disk Parity)\*\*: Distributes parity along with the data, providing a balance of good performance and data safety.

- \*\*RAID 6 (Double Disk Parity)\*\*: Provides two parity blocks per disk allowing more safety for data.

- \*\*RAID 10 (Nested)\*\*: Combines mirroring and striping for high performance and redundancy.

**6. Hardware vs. Software RAID**

- \*\*Software RAID\*\*: Utilizes system resources for managing RAID without needing a physical RAID controller.

- \*\*Hardware RAID\*\*: Uses a dedicated controller which can significantly improve performance and reliability.

### Exam Preparation Tips

- Understand the types of device files and their specific uses.

- Familiarize yourself with `dd` command options and scenarios for use.

- Practice filesystem management commands like `fdisk`, `mkfs`, and managing RAID configurations.

- Differentiate between software and hardware RAID, including their advantages and appropriate use cases.

**Week 14:**

**1. Monitoring Disk Usage**

- \*\*Commands\*\*:

- `$df`: Reports the amount of disk space used and available on filesystems.

- `-h`: Human-readable format.

- `-m`: Displays disk space in megabytes.

- `-T`: Shows the file system type.

- `$du`: Provides information on disk usage of files and directories.

- `-h`: Human-readable format.

- `-a`: Lists all files and directories.

- `-c`: Gives the total size of the directory.

**2. Monitoring Memory Usage**

- \*\*Key Commands\*\*:

- `$free`: Displays memory usage, including total, used, free, shared, buffers, and cached memory.

- `$top`: Provides a dynamic, real-time view of the system’s running processes including CPU and memory usage.

- `$vmstat`: Reports information on memory, paging, block IO, traps, and CPU activity.

**3. Managing System Services**

- \*\*Service Management Tools\*\*:

- `$systemctl`: Manages the state of "systemd" system and service manager.

- `start <service>`: Starts a service.

- `stop <service>`: Stops a service.

- `enable <service>`: Enables a service to start at boot.

- `disable <service>`: Disables a service from starting at boot.

- `$journalctl`: Views logs managed by systemd’s logging service, `journald`.

- `-r`: Displays logs in reverse order.

- `-n`: Shows a specific number of recent log entries.

- `-p`: Filters log entries based on priority level.

**4. Memory Management Commands**

- \*\*Handling Unresponsive Processes\*\*:

- `$kill`: Used to send signals to processes.

- `1 (HUP)`: Reloads the process.

- `9 (KILL)`: Forcefully kills the process.

- `15 (TERM)`: Gracefully stops the process.

**5. System and Service Management with `systemd`**

- \*\*Key Functionalities\*\*:

- Executes tasks in parallel, resulting in faster boot times.

- Compatible with SysV init scripts.

- Manages on-demand activation of services.

### Exam Tips:

- \*\*Understand Memory and Disk Commands\*\*: Familiarize with commands like `$free`, `$top`, `$df`, and `$du` for monitoring system resources.

- \*\*Service Management\*\*: Be able to manage services using `systemctl` and understand the logging capabilities of `journalctl`.

- \*\*Practical Application\*\*: Practice using these commands in a Linux environment to gain hands-on experience with monitoring and managing system resources.

**Week 15: Basic command lines:**

**1. Overview of CRON**

- \*\*Definition\*\*: Cron is a time-based job scheduler in Unix-like operating systems used to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals.

- \*\*Crontab\*\*: The configuration file that specifies shell commands to run periodically on a given schedule.

**2. Crontab Syntax**

- \*\*Fields\*\*:

- \*\*MIN\*\*: Minute field (0 to 59)

- \*\*HOUR\*\*: Hour field (0 to 23)

- \*\*DOM\*\*: Day of Month (1 to 31)

- \*\*MON\*\*: Month field (1 to 12)

- \*\*DOW\*\*: Day of Week (0 to 6, where Sunday can be 0 or 7)

- \*\*CMD\*\*: Command to be executed

- \*\*Format\*\*: MIN HOUR DOM MON DOW CMD

**3. Editing and Managing Crontab**

- \*\*Commands\*\*:

- `crontab -e`: Edit your crontab file, or create one if it doesn’t already exist.

- `crontab -l`: Display your crontab file.

- `crontab -r`: Remove your crontab file.

**4. Cron Permissions**

- \*\*Files\*\*:

- `/etc/cron.allow`: If this file exists, it must contain your username for you to use cron jobs.

- `/etc/cron.deny`: If the cron.allow file does not exist but the cron.deny file does, you must not be listed in the cron.deny file to use cron jobs.

**5. Examples of Scheduling Cron Jobs**

- \*\*Specific Time\*\*: `30 08 10 06 \* /home/victor/full-backup` (Run at 8:30 AM on June 10th)

- \*\*Every Minute\*\*: `\* \* \* \* \* CMD` (Run CMD every minute)

- \*\*Multiple Times a Day\*\*: `00 11,16 \* \* \* /home/victor/bin/incremental-backup` (Run at 11:00 AM and 4:00 PM every day)

- \*\*Range of Times\*\*: `00 09-18 \* \* \* /home/victor/bin/check-db-status` (Run hourly from 9 AM to 6 PM every day)

- \*\*Weekdays Only\*\*: `00 09-18 \* \* 1-5 /home/victor/bin/check-db-status` (Run hourly from 9 AM to 6 PM, Monday through Friday)

- \*\*Every 10 Minutes\*\*: `\*/10 \* \* \* \* /home/victor/bin/check-disk-space` (Run every 10 minutes)

**6. Cron Special Keywords**

- \*\*@yearly (or @annually)\*\*: `@yearly /home/victor/bin/annual-maintenance` (Run at midnight on January 1st every year)

- \*\*@monthly\*\*: `@monthly /home/victor/bin/tape-backup` (Run at midnight on the first day of each month)

- \*\*@daily (or @midnight)\*\*: `@daily /home/victor/bin/cleanup-logs` (Run at midnight every day)

- \*\*@hourly\*\*: `@hourly CMD` (Run at the top of every hour)

- \*\*@reboot\*\*: `@reboot CMD` (Run once at startup)

### Exam Tips:

- \*\*Understand Crontab Syntax\*\*: Familiarize yourself with how to specify the scheduling of jobs using the crontab fields.

- \*\*Know How to Edit and Manage Crontabs\*\*: Be able to edit, list, and remove crontab entries.

- \*\*Practical Usage\*\*: Practice writing cron jobs for various scheduling needs to reinforce the concepts.

**Week 16: script**

**1. Introduction to BASH Scripting**

- \*\*BASH (Bourne Again Shell)\*\*: A popular command-line interface (CLI) shell used in Linux and Unix-based operating systems.

- \*\*Shell Script\*\*: A file containing a series of commands. The shell reads these commands as if they were typed directly into the terminal.

**2. Basics of Shell Scripting**

- \*\*Script Format\*\*:

- Starts with a shebang (`#!`) followed by the path to the interpreter (e.g., `#!/bin/bash`).

- The shebang line tells the operating system to treat the file as an executable script rather than a plain text file.

**3. Writing a Bash Script**

- \*\*Permissions\*\*: Use `chmod +x scriptname.sh` to make the script executable.

- \*\*Basic Commands in Script\*\*:

- `echo`: Used to display messages or output variables.

- `sleep`: Pauses the script for a specified number of seconds (e.g., `sleep 3`).

**4. Variables in Scripts**

- \*\*Declaration\*\*: Variables are declared without a prefix, but are accessed with a dollar sign (e.g., `variable="value"`, then used as `$variable`).

- \*\*Purpose\*\*: Variables save time by storing data that can be reused throughout the script.

**5. User Interaction**

- \*\*User Prompts\*\*: Scripts can prompt users for input, which can then be stored in variables for later use.

- \*\*Positional Arguments\*\*: Scripts can accept inputs directly from the command line, referred to as positional parameters (e.g., `$1`, `$2`).

**6. Integrating Commands**

- \*\*Commonly Used Commands\*\*:

- `whoami`: Displays the current user.

- `pwd`: Prints the current working directory.

- `date`: Shows the current date.

**7. Example of a Simple Bash Script**

```bash

#!/bin/bash

# This script displays user information and waits for user input.

echo "Hello, $(whoami). You are in the directory $(pwd)."

echo "What is your favorite color?"

read color

echo "You said $color!"

```

Exam Preparation Tips:

- \*\*Understand Shebang\*\*: Know why the shebang line is critical for script execution.

- \*\*Syntax and Permissions\*\*: Be able to write basic scripts, set execution permissions, and understand how to use variables and user input.

- \*\*Practical Application\*\*: Practice writing scripts that use common commands and manage user input to reinforce understanding and improve proficiency.