Linux Study Guide

---

1. Overview of Linux

Understanding Linux Distributions

- **Linux** is an open-source operating system kernel created by Linus Torvalds in 1991.
- A **Linux distribution (distro)** is an operating system based on the Linux kernel. It includes a variety of software packages and tools for managing and using the system.
- **Popular Distributions**:
    - **Ubuntu**: User-friendly, popular for desktops and servers.
    - **Debian**: Stability-focused, often used for servers.
    - **CentOS / Red Hat Enterprise Linux (RHEL)**: Focused on enterprise environments, CentOS is a free version of RHEL.
    - **Fedora**: Cutting-edge software, sponsored by Red Hat.
    - **Arch Linux**: Rolling release distro, minimalistic, suitable for experienced users.
    - **Linux Mint**: Based on Ubuntu, designed for ease of use.

Basic Architecture of Linux

- **Kernel**: The core component that manages hardware resources (CPU, memory, I/O).
- **Shell**: Command-line interface (CLI) that allows interaction with the kernel (e.g., Bash).
- **System Libraries**: Functions that allow programs to interact with the kernel.
- **System Utilities**: Tools and programs that perform specific tasks (e.g., file management, networking).
- **User Space**: All components outside of the kernel that users interact with, such as applications and files.

---

2. Basic Commands

File Management Commands

- **ls**: List directory contents.
    - Usage: `ls [options] [directory]`
    - Example: `ls -l` (long format with details), `ls -a` (include hidden files).
- **cp**: Copy files or directories.
    - Usage: `cp [source] [destination]`
    - Example: `cp file1.txt /path/to/destination/`

- **mv**: Move or rename files or directories.
    - Usage: `mv [source] [destination]`
    - Example: `mv file1.txt /new/location/` or `mv oldname.txt newname.txt`.
- **rm**: Remove files or directories.
    - Usage: `rm [options] [file]`
    - Example: `rm file.txt` (deletes the file), `rm -r dir/` (deletes a directory recursively).
- **mkdir**: Create a new directory.
    - Usage: `mkdir [options] directory_name`
    - Example: `mkdir new_folder`.
- **rmdir**: Remove an empty directory.
    - Usage: `rmdir directory_name`
    - Example: `rmdir empty_folder`.

Viewing and Manipulating File Content

- **cat**: Concatenate and display file contents.
    - Usage: `cat [file]`
    - Example: `cat file.txt` (displays file content).
- **more**: View file content one page at a time.
    - Usage: `more [file]`
    - Example: `more file.txt` (scroll through the file page by page).
- **less**: View file content with advanced navigation (scroll forward and backward).
    - Usage: `less [file]`
    - Example: `less file.txt` (allows scrolling through content).
- **head**: Display the first few lines of a file.
    - Usage: `head [file]`
    - Example: `head file.txt` (displays the first 10 lines by default).
- **tail**: Display the last few lines of a file.
    - Usage: `tail [file]`
    - Example: `tail file.txt` (displays the last 10 lines by default).

Understanding File Permissions and Ownership

- **chmod**: Change file permissions.
    - Usage: `chmod [permissions] [file]`
    - Example: `chmod 755 file.sh` (set read, write, and execute permissions for the owner, and read & execute for others).
- **Permission Format**:
    - **r**: Read (4)
    - **w**: Write (2)
    - **x**: Execute (1)
    - Permissions are represented as a 3-digit number (Owner, Group, Others).
- Example: `chmod 644 file.txt` (Owner can read/write, Group/Other can only read).

- **chown**: Change file owner and group.
    - Usage: `chown [owner]:[group] [file]`
    - Example: `chown john:admin file.txt` (Change ownership to user 'john' and group 'admin').
- **chgrp**: Change the group ownership of a file.
    - Usage: `chgrp [group] [file]`
    - Example: `chgrp staff file.txt` (Change group to 'staff').

Key Concepts and Tips

- **Permissions Breakdown**:
    - **rwx**: Read, Write, Execute permissions for the user, group, and others.
    - Each file has an **owner** and a **group**. The **owner** can be different from the **group**.
    - Permissions are crucial for file security, especially in multi-user systems.
- **Wildcard Usage**:
    - `*` (matches all characters): `ls *.txt` (list all `.txt` files).
    - `?` (matches a single character): `ls file?.txt` (list files like `file1.txt`, `file2.txt`).
- **Redirection & Piping**:
    - `>`: Redirect output to a file (overwrites).

- `>>`: Append output to a file.
- `|`: Pipe output of one command to another.
- Example: `ls | more` (view the output of `ls` one page at a time).

Let's break down the concepts of **Redirection** and **Piping** in Linux. These are powerful tools for controlling the flow of input and output in the terminal.

1. `>`: Redirect Output to a File (Overwrites)

The `>` operator is used to redirect the output of a command to a file. If the file already exists, it will be **overwritten**.

Example:

```
echo "Hello, World!" > output.txt
```

Explanation:

- The `echo` command generates the string "Hello, World!".
- The `>` operator redirects this output to `output.txt`.
- If `output.txt` exists, its contents will be replaced with the new string.

Output in `output.txt`:

```
Hello, World!
```

2. `>>`: Append Output to a File

The `>>` operator appends the output of a command to a file. If the file doesn't exist, it will be created. **It does not overwrite the file** like `>` does.

Example:

```
echo "Appended text" >> output.txt
```

Explanation:

- The `echo` command generates the string "Appended text".
- The `>>` operator appends this output to the end of `output.txt`.
- If `output.txt` doesn't exist, it will be created with this content.

Updated content in `output.txt`:

> Hello, World!
>
> Appended text

3. `|`: Pipe Output of One Command to Another

The `|` (pipe) operator is used to send the output of one command directly as input to another command. It's useful for chaining commands together, allowing you to process the output further without saving it to a file.

Example:

```
ls | more
```

Explanation:

- The `ls` command lists the contents of the current directory.
- The `|` operator takes the output of `ls` and sends it to the `more` command.
- The `more` command displays the output one page at a time, making it easier to view long lists of files.

How it works:

- The output of `ls` (a list of directory contents) is sent to `more`.
- The `more` command lets you scroll through the list of files interactively.
- You can press the spacebar to go to the next page or `q` to quit.

Combining These Concepts

You can combine redirection and piping to perform more complex operations.

Example:

```
ls | grep "txt" > txt_files.txt
```

Explanation:

- `ls` lists the contents of the current directory.
- The `grep` command filters the output to only show lines containing the string "txt".
- The `>` operator redirects the filtered output to a file named `txt_files.txt`.

In this case, if any files or directories have "txt" in their name, they will be saved in `txt_files.txt`.

Summary:

- `>`: Redirect output to a file and overwrite its contents.
- `>>`: Append output to a file, creating it if it doesn't exist.
- `|`: Pipe output from one command to another, allowing you to chain commands.