Lab 2 - Linux Fundamentals

What is Linux?

Linux is an open-source operating system kernel that serves as the foundation for various operating systems, commonly referred to as "Linux distributions" or "Linux distros." These distributions combine the Linux kernel with a collection of software packages, libraries, utilities, and a user interface to create a complete and functional operating system.

Here are the top 5 most commonly used Linux distributions for general use:

- **Ubuntu**: Known for its user-friendly interface and strong community support, Ubuntu is one of the most popular Linux distributions. It offers regular releases and long-term support versions suitable for both desktop and server environments.
- **Linux Mint**: Built on top of Ubuntu, Linux Mint focuses on providing a user-friendly and familiar experience, often incorporating user interface elements similar to those found in traditional operating systems like Windows.
- **Debian**: Debian is known for its stability and focus on free and open-source software. It serves as the basis for many other distributions and offers multiple releases, including the stable, testing, and unstable branches.
- **Fedora**: Sponsored by Red Hat, Fedora aims to showcase the latest open-source technologies while maintaining a balance between cutting-edge features and stability. It serves as a testing ground for innovations that might later appear in Red Hat Enterprise Linux.
- **CentOS (now CentOS Stream)**: Historically, CentOS provided a free, community-supported version of Red Hat Enterprise Linux (RHEL). However, as of late 2020, CentOS Stream has become the upstream development branch for RHEL, and its role has shifted.

Now, for the hacking-focused distributions, these are not inherently used for malicious purposes, but rather for ethical hacking, penetration testing, and security research:

1. **Kali Linux**: Perhaps the most well-known ethical hacking distribution, Kali Linux is tailored for penetration testing and security assessments. It includes a vast array of tools for various security testing purposes.
2. **Parrot Security OS**: Similar to Kali Linux, Parrot Security OS is designed for security researchers and ethical hackers. It provides a comprehensive suite of tools for various security tasks.
3. **BlackArch Linux**: BlackArch is a penetration testing distribution that specializes in providing a large repository of tools for security professionals. It's built on top of Arch Linux.
4. **BackBox**: BackBox is a relatively lightweight distribution that focuses on penetration testing, vulnerability assessment, and forensic analysis. It's designed to be fast and easy to use.
5. **Pentoo**: Pentoo is a Gentoo-based Linux distribution with a strong emphasis on penetration testing and security assessment. It's optimized for system and network auditing.

What is a Shell?

A **shell** is a command-line interface (CLI) program that provides a way for users to interact with an operating system (OS). It acts as an intermediary between the user and the OS, allowing users to execute various commands and programs by typing text-based instructions. Shells interpret the commands entered by the user, interact with the OS to execute those commands, and display the results.

What is a Terminal?

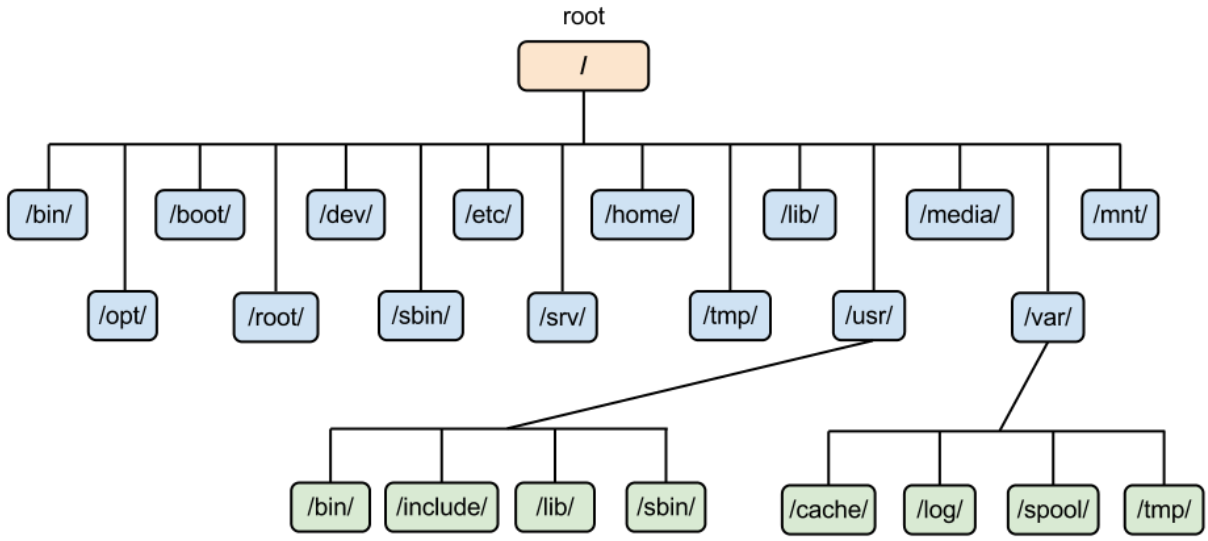
A **terminal** is a software application that provides an interface for users to interact with a shell. It's essentially a window into which you can type commands and receive their output. Terminals allow you to enter text-based commands, and the terminal then communicates with the shell to execute those commands. The shell processes the commands and returns the output to the terminal, which displays it for the user.

Difference between a shell and a Terminal

In many cases, people use the terms "shell" and "terminal" interchangeably because they often work together to provide a command-line environment. Common shells include Bash, Zsh, and PowerShell, while terminals can be various software applications like GNOME Terminal, macOS Terminal, or Windows Terminal.

Linux Filesystem Hierarchy

The Linux filesystem hierarchy is the structure of directories that is used in Linux and other Unix-like operating systems. It resembles a tree-like structure, with directories branching off from the root directory, which is represented by a forward slash (`/`). Each directory can contain files and other directories, which can contain additional files and directories, and so on.




```
/ - The root directory, which contains all other directories and files on the system and serves as the base of the filesystem hierarchy.
├── bin - Contains essential command-line utilities and programs that are required for the system to function properly.
|   ├── bash - The Bash shell.
├── boot - Contains the files required for booting the system, such as the Linux kernel and the bootloader configuration.
├── dev - Contains device files that represent hardware devices connected to the system.
├── etc - Contains system-wide configuration files.
├── home - Contains the home directories for regular users.
|   ├── /<username> - The home directory for the <username> user.
├── lib - Contains shared libraries that are required by the essential binaries in the /bin and /sbin directories.
├── media - Contains mount points for removable media, such as USB flash drives and CD-ROMs.
├── mnt - Contains mount points for temporarily mounted filesystems.
├── opt - Contains optional software packages that are not installed by default.
├── proc - Contains virtual files that represent system and process information.
├── root - The home directory for the root user.
├── run - Contains runtime data for system services.
├── sbin - Contains essential system binaries that are required for the system to function properly.
├── srv - Contains data for services provided by the system.
├── sys - Contains virtual files that represent system information.
├── tmp - Contains temporary files.
├── usr - Contains user utilities and applications.
|   ├── /bin - Contains non-essential command-line utilities and programs.
|   ├── /lib - Contains shared libraries that are required by the binaries in the /bin directory.
|   |-- /include - Contains header files that are required for compiling C programs.
|   ├── /local - Contains software packages that are installed locally.
|   ├── /sbin - Contains non-essential system binaries.
|   ├── /share - Contains shared data used by applications.
|   ├── /src - Contains source code.
|── var - Contains variable data, such as logs, databases, and websites.
|   |-- /log - Contains log files.
|   |-- /www - Contains websites.
```

▼ `~` in Linux refers to the home page of the user.

Relative vs Absolute Paths:

In Linux, as well as in other operating systems, you can specify file and directory paths using two main types: absolute paths and relative paths. These paths determine how you reference the location of a file or directory within the file system.

1. **Absolute Paths:**
    - An absolute path specifies the exact location of a file or directory from the root directory of the file system. The root directory is denoted by a single forward slash (`/`).
    - Absolute paths always start from the root directory, providing a full and unambiguous path to the file or directory.
    - Examples of absolute paths:
        - `/home/user/documents/file.txt`: This is an absolute path to a file called "file.txt" located in the "documents" directory within the "user" directory of the root directory.
        - `/var/log/syslog`: This is an absolute path to a system log file.
2. **Relative Paths:**
    - A relative path specifies the location of a file or directory relative to your current working directory.
    - Relative paths do not start with a forward slash (`/`) and assume that you are referencing the file or directory in relation to your current position in the file system.
    - Examples of relative paths:
        - `documents/file.txt`: If your current working directory is `/home/user`, this is a relative path to the "file.txt" located in the "documents" directory.
        - `../parent_directory/another_file.txt`: Using `..` allows you to reference the parent directory of your current working directory.

Here's an example to illustrate the difference between absolute and relative paths:

Suppose your current working directory is `/home/user`:

- An absolute path to the file "file.txt" in the "documents" directory would be `/home/user/documents/file.txt`.
- A relative path to the same file would be `documents/file.txt`.

You would use absolute paths when you need to specify the exact location of a file or directory regardless of your current working directory. Relative paths are handy when you want to reference files or directories in relation to your current working location, making your commands more flexible and concise.

To change your current working directory in Linux, you can use the `cd` command. For example, to change to the "documents" directory in the above example, you would use:

```
cd /home/user/documents
```

In order to further explain this concept, we'll utilize an Analogy.

Analogy: The Treasure Hunt

Imagine you're on a treasure hunt in a huge, maze-like garden. You have a map that shows you the path to the treasure. The garden has a single, massive tree at its center, and that tree represents the "root" of the garden.

1. **Absolute Path:**
    - An absolute path is like giving someone the precise GPS coordinates to the treasure on the map.
    - For example, you say, "The treasure is located at latitude 12.3456 and longitude 78.9012."

- In this analogy, the latitude and longitude are like the absolute path in Linux, starting from the "root" of the garden (the massive tree). It's a direct and exact way to find the treasure no matter where you are in the garden.

2. **Relative Path:**
    - A relative path, on the other hand, is like giving someone directions to the treasure based on where they are in the garden.
    - For example, you say, "From where you are right now, take three steps to the right, walk straight for five paces, and then turn left. You'll find the treasure."
    - In this analogy, the directions are like a relative path in Linux. They depend on your current position in the garden and guide you to the treasure based on your starting point.

Now, let's translate this analogy to a real-world computer scenario:

- The "garden" is your computer's file system.
- The "root" of the garden is represented by the forward slash (/), which is the starting point of your file system.
- The "treasure" is a file or directory you want to access.

Absolute Path Example (Linux):
Suppose the treasure (file) you want to find is located in your home directory:

- In Linux, the absolute path to your home directory might be: `/home/your_username/`
- It's like giving someone the exact coordinates to the treasure, starting from the root of the garden.

Relative Path Example (Linux):
Now, let's say you're currently in your home directory:

- A relative path to a file in your "Documents" directory might be: `Documents/file.txt`
- It's like giving someone directions to the treasure starting from where they are in the garden (your current working directory).

So, in summary, absolute paths provide the full path from the root, while relative paths describe how to get to a location based on your current position in the file system, just like giving directions in a treasure hunt.

. vs .. in Linux:

In Linux, the symbols "." and ".." are used to represent specific directories within a file system. They have special meanings and are often used in relative paths to navigate and reference directories. Here's what they mean:

1. `.` **(Dot):**
    - The single dot (`.`) represents the current directory.
    - When you use `.` in a file path, you are referring to the directory you are currently in.
    - It's often used to specify the current directory when working with relative paths.
    - For example, if you are in the directory `/home/user/documents`, using `.` in a relative path like `./file.txt` refers to a file named "file.txt" in the current directory (`/home/user/documents`).
2. `..` **(Double Dot):**
    - The double dot (`..`) represents the parent directory.
    - When you use `..` in a file path, you are referring to the directory one level above your current directory.
    - It's commonly used to move up one level in the directory hierarchy.
    - For example, if you are in the directory `/home/user/documents`, using `..` in a relative path like `../file.txt` refers to a file named "file.txt" in the parent directory (`/home/user`).

Linux Basic Commands

`whoami`

`whoami` prints the current user.

```
whoami
```

`pwd`

`pwd` stands for `Print Working Directory`. It prints the current working directory.

```
pwd
```

`man`

`man` stands for manual. It can be used to display the manual pages about a specific command.

```
man <command-name>
```

`ls`

`ls` stands for `List`. It lists the contents of the current directory.

```
ls
# NOTE: ls command has several different flags that can be used to modify its output
```

`cd`

`cd` stands for `Change Directory`. It changes the current working directory.

```
cd <directory>
```

`mkdir`

`mkdir` stands for `Make Directory`. It creates a new directory.

```
mkdir <directory>
```

`touch`

`touch` creates a new file.

```
touch <file>
```

`rm`

`rm` stands for `Remove`. It removes a file or directory.

```
rm <file>
## To remove a folder:
rm -rf <folder>
```

`mv`

`mv` stands for `Move`. It moves a file or directory, can also be used to rename a file or directory.

```
mv <source> <destination>
```

`cp`

`cp` stands for `Copy`. It copies a file or directory.

```
cp <source> <destination>
```

`cat`

`cat` stands for `Concatenate`. It prints the contents of a file to the terminal.

```
cat <file>
```

`echo`

`echo` prints text to the terminal.

```
echo <text>
```
