What I have learned from Born2beroot Project

What is Linux/Unix?

Linux inherits its heritage from the competition between BSD and UNIX (AT&T). Also, Linux adapted the same philosophy as UNIX and GNU tools, then developed the Linux kernel, building an entire OS called Linux.

Linux isn't a version of UNIX; it is a clone of UNIX. Today, it is referred to as UNIX-like because UNIX was a product sold by AT&T.

In summary, UNIX was a commercial product sold by AT&T. Its source code was licensed early on, leading to the creation of operating systems like FreeBSD and, to some extent, macOS.

Linux was written using MINIX to create a UNIX clone or UNIX-like OS. Today, it doesn't share any files with the original version of UNIX.

Technically, Linux is just a kernel, the core component of the operating system that interacts with your hardware and various running processes. What you see when using the system is called GNU/Linux (GNU is not UNIX).

GNU is a free and open-source collection of software.

The Relationship Between Unix and Linux:

Linux was inspired by Unix, and it shares many concepts and design principles with its predecessor. The command-line interface, file system structure, and the idea of small, modular utilities working together are all inherited from Unix. However, Linux is not a direct descendant of any specific version of Unix. Instead, it is a Unix-like operating system that was developed independently.

While Linux and Unix share similarities, they also have differences. Unix systems often have proprietary code and may not be freely available, whereas Linux is open source.

In summary, Unix and Linux are both Unix-like operating systems with shared roots but distinct development paths. Linux emerged as a free and open-source alternative, carrying forward the Unix philosophy of simplicity, modularity, and portability. The success of Linux has made it a dominant force in the world of operating systems, particularly in server environments.

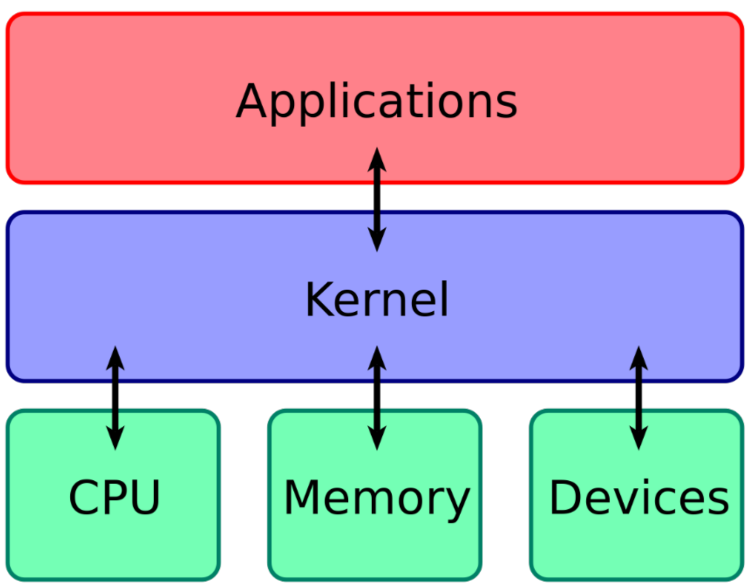
How and why was the term 'Unix-like' developed?

The original Unix operating system was developed at Bell Labs in the late 1960s and early 1970s. Over time, various institutions and vendors created their own versions of Unix, leading to different Unix variants. These variants were often adapted to specific hardware architectures and had unique features and implementations.

As the Unix family tree diversified, it became necessary to categorize operating systems that followed the Unix design philosophy but were not direct descendants of the original codebase. The term "Unix-like" emerged to encompass this broader category.

What is a Kernel?

The kernel, in simple terms, is the core of an operating system.



Applications <=> Kernel <=> (CPU | Memory | Devices)

The kernel is the essential foundation of a computer's operating system (OS). It is the core that provides basic services for all other parts of the OS. It is the main layer between the OS and underlying computer hardware, and it helps with tasks such as process and memory management, file systems, device control and networking.

During normal system startup, a computer's basic input/output system (BIOS), completes a hardware bootstrap or initialization. It then runs a bootloader which loads the kernel from a storage device -- such as a hard drive -- into a protected memory space. Once the kernel is loaded into computer memory, the BIOS transfers control to the kernel. It then loads other OS components to complete the system startup and make control available to users through a desktop or other user interface.

What is the Purpose of a Kernel?

The kernel is the core part of an operating system that manages communication between software (applications) and hardware (computer's resources like CPU, memory, and peripherals). It acts as a bridge, ensuring that different programs can use the computer's resources efficiently and without interfering with each other. The kernel also handles tasks like process management, memory allocation, and input/output operations.

Before the kernel, developers coded actions directly to the processor, instead of relying on an OS to complete interactions between hardware and software.

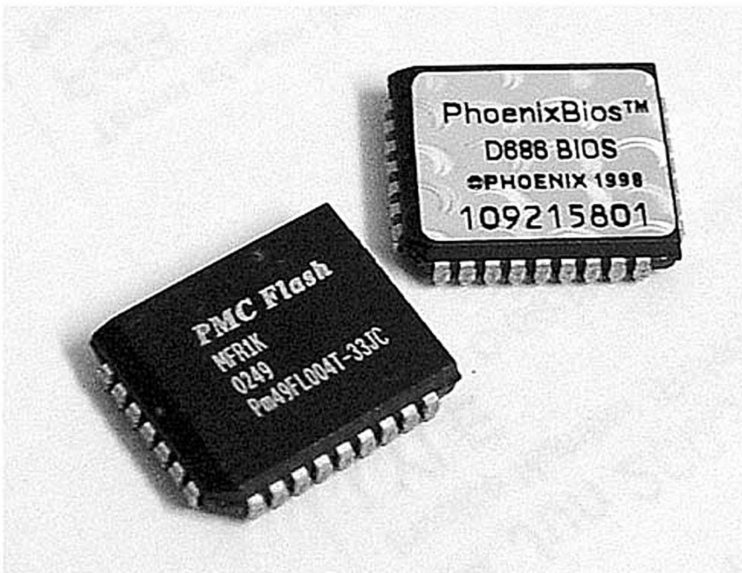
Kernel and Drivers

It is impractical to write an OS capable of interacting with every possible device in existence. Instead, kernels rely on the ability of device drivers to add kernel support for specialized devices, such as printers and graphics adapters.

What is a BIOS in simple terms?

BIOS stands for Basic Input/Output System. It is a type of firmware embedded in a computer's motherboard. <u>BIOS serves as a bridge between the computer's hardware and its operating system.</u> When you turn on your computer, the BIOS kicks in first, initializing and checking the basic hardware components like the processor, memory, and storage. It also provides a set of low-level instructions that enable the operating system to start loading.

A kernel is not to be confused with a BIOS, which is an independent program stored on a chip within a computer's circuit board.



What is an Operating System?

An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. All computer programs, excluding firmware, require an operating system to function.

An operating system (OS) is the most crucial piece of software that runs on a computer, as it manages and controls the entire system's resources and activities. In terms of managing memory, peripherals, and hardware interfaces, the OS has several key responsibilities:

1. Memory management:

- **a.** Allocation.
- **b.** Protection.
- **c.** Virtual memory.
- **d.** Paging and swapping.

2. Peripheral management.

3. Hardware interface management:

The OS is responsible for interfacing with the underlying hardware components (e.g., CPU, GPU, RAM, and storage devices) and ensuring that they work together smoothly.

In summary, the operating system plays a crucial role in managing memory, peripherals, and hardware interfaces. By efficiently allocating resources, ensuring data protection, and providing a stable platform for applications to run, the OS enables a seamless user experience and optimal system performance.

Denis Richie and others They wanted to separate the OS into two parts:

First was the core functionality of the OS, things like memory management, multitasking, and dealing with I/O, **which is called the kernel.**

The second part was a wide array of useful tools that came bundled with, but not part of the kernel, things like programs and libraries.

Building a compact, lean kernel meant intentionally leaving some functionality out.

In summary, the kernel is a core component of an operating system responsible for managing low-level tasks and interacting directly with hardware. The operating system is a broader concept that includes the kernel and additional components, providing a complete environment for users and applications, managing higher-level functions, and offering a user-friendly interface.

What is the difference between Rocky and Debian OS?

Debian:

- Established in the early 1990s.
- Community-driven with a democratic development process.
- Uses APT (Advanced Package Tool) for package management.
- Strong commitment to free software principles.
- Debian has more software/packages available.

Rocky Linux:

- Emerged in response to changes in CentOS in 2020.
- Created as a downstream, binary-compatible alternative to RHEL. (Rocky Linux is downstream from Red Hat Enterprise Linux (RHEL), meaning it is based on RHEL or follows its developments).
- Aims to provide a free alternative with binary compatibility to RHEL.
- Uses YUM (Yellowdog Updater, Modified) for package management.
- Rocky OS has a faster release cycle than Debian, ensuring more timely updates and access to newer software.
- Rocky focuses on enterprise more than normal users, unlike Debian.

Security Difference:

Rocky OS prioritizes SELinux with advanced security policies and exclusive tools like sealert.

In Debian, AppArmor is simpler but lacks these SELinux features. The choice depends on preferences, with Rocky OS offering enhanced security but Debian being easier to manage.

What is a Linux distro?

The versions of Linux, or "flavors," are called distributions or distros for short.

What is a Linux distro? A distro is a complete operating system based on the Linux kernel that contains a bunch of packages and libraries, typically with a package manager to install additional apps.

There are many distros today; some only run on servers, while others are designed for desktop, mobile, and embedded devices.

Red Hat is known for its focus on security and reliability for enterprise users. Red Hat has adopted a business model where they maintain an open-source operating system, making money by selling consulting and integration services to big enterprise clients.

What is Debian?

Debian releases a new major version about every two years. They number each major release and then assign it a name based on Toy Story characters.

What is Stable, unstable, and testing version of an OS?

Stable Version: Well-established, reliable, and thoroughly tested OS release.

Purpose: Suitable for everyday use, especially in production environments due to lower likelihood of bugs or issues.

Unstable Version: Actively developed release with the latest features and updates.

Purpose: Not recommended for production use; ideal for developers or enthusiasts testing and contributing to development.

Testing Version: An intermediate stage between unstable and stable releases.

Purpose: Used for gathering user feedback, identifying, and fixing issues before a stable release; provides a balance between new features and stability.

What is a package manager?

A package manager or package-management system is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer in a consistent manner.

Package managers are designed to eliminate the need for manual installs and updates. This can be particularly useful for large enterprises whose operating systems typically consist of hundreds or even tens of thousands of distinct software packages.

Package managers are charged with the task of finding, installing, maintaining or uninstalling software packages upon the user's command. Typical functions of a package management system **include:**

- Working with file archivers to extract package archives.
- Ensuring the integrity and authenticity of the package by verifying their checksums and digital certificates, respectively.
- Looking up, downloading, installing, or updating existing software from a software repository or app store.
- Grouping packages by function to reduce user confusion.

- Managing dependencies to ensure a package is installed with all packages it requires, thus avoiding "dependency hell".

To install new software on this machine we can use a package manager like advanced package tool (apt) or aptitude, which will retrieve the code from the cloud and place it in the proper location on your system.

Unix-like File System and Mounting Points

<u>What is a File system?</u>

In computing, a file system or filesystem (often abbreviated to fs) is a method and data structure that the operating system uses to control how data is stored and retrieved. Without a file system, data placed in a storage medium would be one large body of data with no way to tell where one piece of data stopped and the next began, or where any piece of data was located when it was time to retrieve it. By separating the data into pieces and giving each piece a name, the data are easily isolated and identified.

In Linux, the most used file system for the system disk (where the operating system is installed) is the ext4 file system (Fourth Extended File System).

<u>File system in Simple terms:</u>

A file system is a method and data structure used by an operating system to control the organization, storage, and retrieval of data, providing a logical way to organize and access files on storage medium.

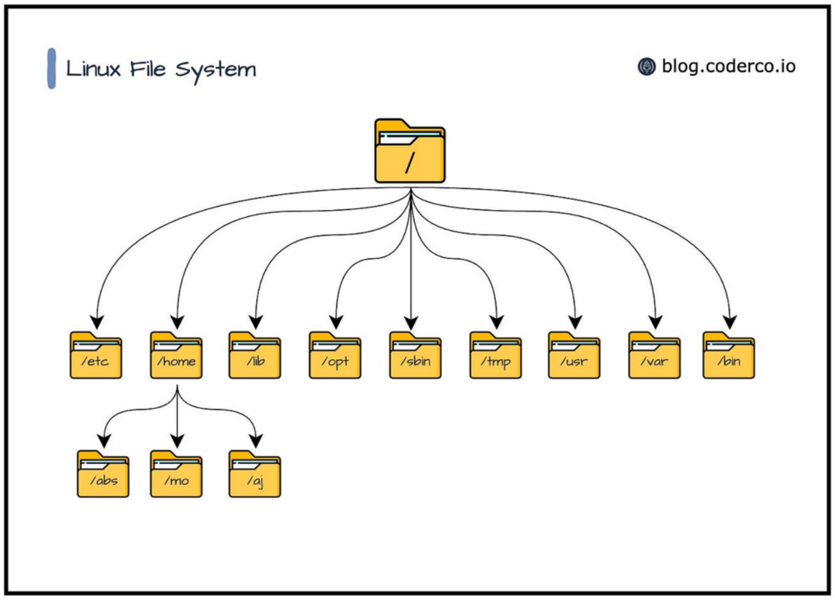
The Linux file system contains the following sections:

- The root directory (/)
- A specific data storage format (EXT3, EXT4, BTRFS, XFS, SWAP and so on)
- Each partition or logical volume has a particular file system.

Linux File System Structure

Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

Directory Structure



- / (root filesystem): It is the top-level filesystem directory. It must include every file needed to boot the Linux system before another filesystem is mounted. Every other filesystem is mounted on a well-defined and standard mount point because of the root filesystem directories after the system is started.

- /boot: It includes the static kernel and bootloader configuration and executable files needed to start a Linux computer.

- /root: It's the home directory for a root user. Keep in mind that it's not the '/' (root) file system.

What is root?

In Unix-like operating systems, including Linux, the 'root' user is the superuser or administrator. This user has the highest level of privileges and can perform tasks that regular users cannot.

General facts about root:

- root is the super user and can do anything on a system. Therefore, to have an additional layer of security, a sudo user is generally used in place of root.
- You can put directories in the root folder, but you will need root permission to access it.
- Only the administrator (root user) typically has the authority to authorize the mounting of file systems.
- The Linux filesystem unifies all physical hard drives and partitions into a single directory structure. It all starts at the top–the root (/) directory.

Types of Linux File System

1. Ext, Ext2, Ext3 and Ext4 file system:
Ext4 file system is the faster file system among all the Ext file systems. It is a very compatible option for the SSD (solid-state drive) disks, and it is the default file system in Linux distribution.

6. Swap File System:
The swap file system is used for memory paging in Linux operating system during the system hibernation. A system that never goes in hibernate state is required to have swap space equal to its RAM size.

What is Mount Point in Linux?

A mount point is a directory where an external partition, storage device, or file system can become accessible to the user or application requiring it (e.g., when a Linux installation process requires access to the USB device containing the ISO image or installation files).

Mounting means attaching a partition, hard disk, or file system to the system in use. Similarly, when we insert a CD in Windows operating systems, the default mount point would be D:/ or Cdrom, the “directory” where files become accessible to the user. Of course, this is different than in Linux because in Linux mount points are regular directories, like any other directory.

Usually, the mount point is a dedicated directory for mounting purposes, but users can use any directory as a mount point, including directories containing files and subdirectories. But it is important to remember that only mounted files will be accessible. Original files will only become accessible again after the user unmounts the mounted filesystem.

Normally, the default mount points in Linux are /media, /mnt, /usb and /media/mnt, but users can mount devices in any directory.

In other words, Mount points can be considered as doors to inaccessible devices. Many Linux distributions by default mount filesystems when they are attached. This is not a good practice and can be disabled because it is always recommended the intervention of a sysadmin to make sure the attached device will not harm the system,

- The root filesystem of Linux is mounted on the / directory (root directory) very early inside the boot sequence.
- Several filesystems are later mounted by the start-up programs of Linux.
- Filesystem mounting during startup is handled by the configuration file, i.e., /etc/fstab.
- fstab is short for "file system table", and it's a filesystem list that is to be mounted, their options, and designated mount points that may be required for particular filesystems.

What is the Important of Mounting?

Organization: Mount points help organize and structure storage, ensuring a systematic layout for efficient data management.

Example of mounting:

Suppose you have a USB drive with DATA on it, and you want to access these files on your Linux system.

The USB has its own file system, typically formatted as FAT32 or exFAT.

By default, when you plug in the USB, the system may automatically detect it but might not mount it immediately.

Let's assume the USB drive is identified as /dev/sdb1, but it's not yet accessible from the file system.

You need to choose a directory (mount point) in the existing file system where you want the USB drive's contents to be accessible. Let's use /media/usb as the mount point.

Mounting Process (using the command line):

- sudo mkdir /media/usb
- sudo mount /dev/sdb1 /media/usb

Once the USB drive is mounted, you can navigate to the mount point (/media/usb) and access the files as if they are part of your system's file hierarchy.

Practical Applications of Key Mount Points

boot: The /boot directory contains the files needed for the system's bootloader.

root: the root directory contains the core system files and directories. you might have the root directory mounted on the primary storage device to host the operating system and essential system files.

[SWAP]: the [SWAP] partition is used for swap space, providing additional virtual memory when the physical RAM is fully utilized.

home: the /home directory typically contains users home directories. you might have this directory mounted on a separate partition to isolate user data from the operating system files.

var: the /var directory contains variable data, including log files, spool directories, and temporary files. having /var on a separate partition can prevent the system from running out of space due to excessive logging or temporary files.

Srv: the /srv directory is often used to store content served by web servers. For example, the web root directory for a web server like Apache or Nginx might be set to /srv/http or /srv/www. This makes it easy to organize and manage the website files separately from system files.

Tmp: system application uses /tmp as a location to store temporary files during their operation. For example, when you open a document in a text editor, the editor might create a temporary copy of the file in /tmp for editing purposes. After saving changes, the temporary file is typically removed.

/var/log: The /var/log directory stores log files generated by various system processes.

What i came up with about mounting points?

When we are portioning our logical volumes, we must choose the mount points where we wanted them.

From google Bard

Examples:

- Mounting a separate partition for documents: You could create a mount point at /home/user/documents to store all your work documents.
- Mounting a network share: You might have a mount point at /mnt/network_share to access files stored on a remote server.

What is a hypervisor? = مراقب الأجهزة الافتراضية

Hypervisor:

A computer on which a hypervisor runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine.

Hypervisor is a technology that allows hosting multiple virtual computers on a physical computer on top of the operating system.

The Hypervisor requests the host machine to create virtual CPU, virtual RAM, virtual storage for each guest machine.

The guest machine is isolated from the host machine, so if something gets wrong in the guest machine won't affect the host machine.

Benefits of virtual machine:

- Learn and experiment.
- Testing app on different OS.

Hypervisor types:

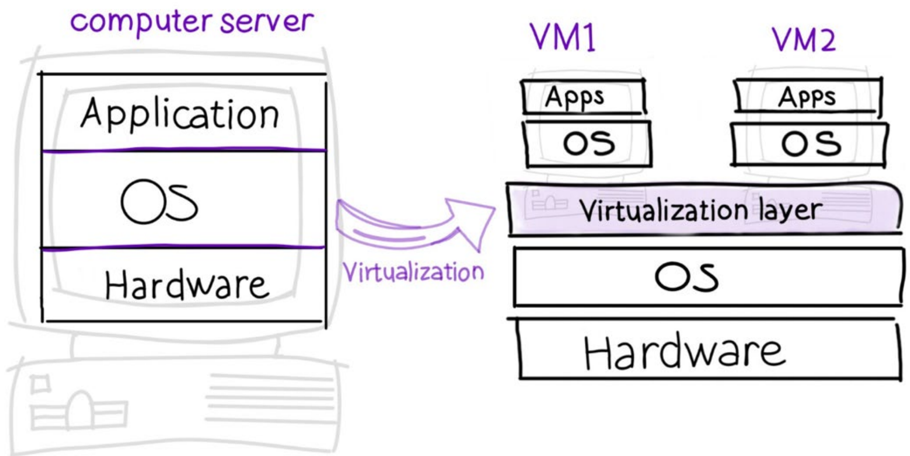
The main difference between a bare-metal (type 1) hypervisor and a hosted hypervisor (type 2) is that, instead of installing a hypervisor on a host operating system, you install it directly on the hardware.

Type 1 is for servers (bare metal).

Type 2 for personal use (hosted hypervisor).

One big benefit of Virtualization is Portability. You have your operating system as a portable file that you can move, and this file is a virtual machine image (VMI). This image includes the operating system, the applications, and all the configurations.

You can make a backup of these images, which are called snapshots.



Demonstrate Virtualization

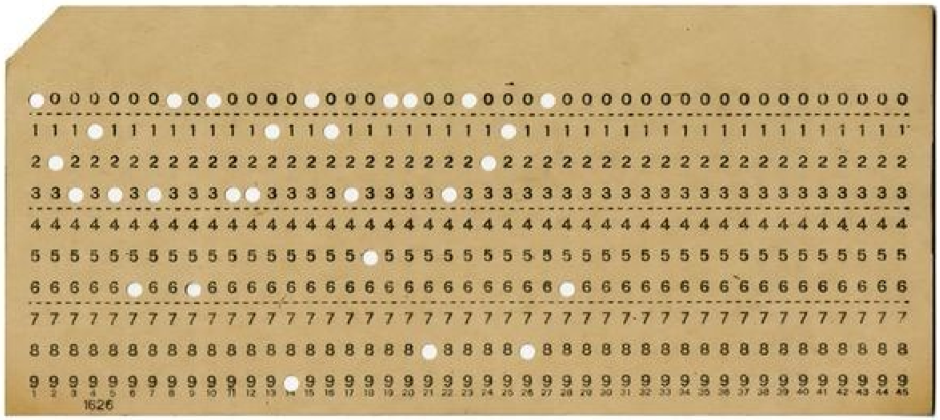
What is the Difference between a program and a software?

In summary, Adobe Photoshop and Microsoft Word are both programs that serve specific purposes (image editing and word processing, respectively). However, they are also part of larger software suites (Adobe Creative Cloud and Microsoft Office) that encompass multiple programs designed for various tasks. The software represents the comprehensive package, while the programs are individual components within that package.

What is a TTY?

The History behind Teletypewriter

Before keyboard Punch cards were used as the input method.



A punch card computer program which was inserted into
computers instead of typing it via keyboard (TTY)

While computers were evolving, batch input cards were eventually replaced by Teletypewriters as a convenient input/output device.

LGP-30 computer in 1956 with a TTY attached to it

With technological advancements, the Teletypewriters were 'virtualized' using electronics. So, you wouldn't need a physical, mechanical TTY, but a virtual, electronic TTY.

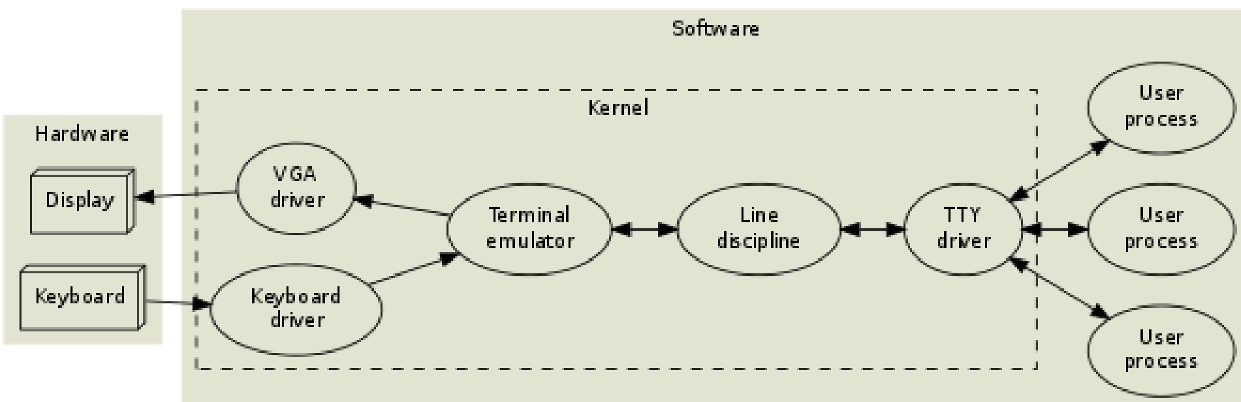
Earlier computers didn't even have video screens. Things were printed on paper instead of displaying on a screen

What is a TTY?

A TTY is essentially a pseudo device, you can call it a kernel resource.

TTY is a subsystem in Linux and Unix that makes process management, line editing, and session management possible at the kernel level through TTY drivers.

Whenever you launch a terminal emulator or use any kind of shell in your system, it interacts with virtual TTYs that are known as pseudo-TTYs or PTY.



When Would You Use TTY in Linux?

1. In case the graphical desktop environment freezes. In some cases, reinstalling the desktop environment from the TTY helps resolve the program.

2. If your Linux system encounters issues during the boot process, you may need to access one of the TTY consoles to diagnose and fix the problem.

3. Running Commands Remotely: TTY is often used in remote shell sessions, especially when connecting to a Linux server via ssh. It allows you to interact with the command-line interface of the remote system.

GENOM terminal or iterm are just emulators to the tty.

When you are working on Unix-like OS you can navigate TYYs by pressing windows key + left/right arrow.

Each virtual console or terminal emulator is considered a TTY.

These TTYs facilitate textual communication between the user and the kernel.

Within the Linux kernel, TTYs are managed by TTY drivers. These drivers handle low-level interactions with hardware devices, translating input and output operations between user-space applications and the underlying terminal hardware.

TTY's play a crucial role in managing input and output streams. They handle the interpretation of keystrokes from users, manage display output, and facilitate communication between processes and the user interface. TTY's also support job control, allowing users to manage multiple processes within a single terminal session.

A good and full article about tty. Another great article explains tty.

ME: TTY is the core part of inputting on Unix-based systems. Yes, you can control your operating system from the terminal, but that is only because each terminal is internally linked to a TTY device.

Google Bard: That's correct. TTYs play a crucial role in Unix-based systems, serving as the foundation for text-based input and output. Here's a breakdown of their key aspects:

1. What is a TTY?

- TTY stands for "TeleTYpewriter," a throwback to the era of electromechanical typewriters that were used to interact with computers.
- In modern Unix-based systems, it refers to a virtual terminal device that provides a text-based interface for user interaction.

2. How TTY's Work:

- Each terminal window or session on a Unix system is associated with a specific TTY device.
- When you type commands or text in a terminal, the TTY device handles the input and transmits it to the operating system.
- The operating system then processes the input and sends any output back to the TTY device, which displays it in the terminal window.

3. Key Functions of TTYs:

- **Input Handling:** TTYs capture keystrokes from the keyboard and make them available to the operating system.

- **Output Display:** TTYs receive text output from the operating system and display it on the screen.
- **Command Execution:** TTYs provide the interface for executing commands and interacting with the shell.
- **Session Management:** TTYs enable multiple users to log in and work on the same system simultaneously, each with their own terminal sessions.

4. TTYs and Terminals:

While often used interchangeably, TTYs and terminals are technically distinct:

- TTYs are the underlying devices that manage text-based input/output.
- Terminals are software programs that provide the visual interface for interacting with TTYs.

5. Essential for Unix-based Systems:

TTYs are integral to Unix-based systems, forming the basis for:

- Command-line interaction
- System administration tasks
- Text-based applications
- Remote access via SSH
- Troubleshooting and recovery

In essence, TTYs provide the essential infrastructure for text-based communication between users and the operating system in Unix-based environments.

What is Logical Volume Manger (LVM)?

What is LVM?

LVM offers a more flexible approach to storage management compared to traditional static partitions. Instead of fixed-size physical partitions, LVM allows you to create, resize, and delete logical volumes (LVs) on the fly, without rebooting the system. This dynamic nature comes in handy for various tasks like:

Growing partitions: Need more space for your /home directory? With LVM, you can expand the corresponding LV even while it's mounted and in use (assuming the filesystem supports it).

Shrinking partitions: If a partition seems oversized, you can reclaim unused space by shrinking its corresponding LV.

Adding/removing storage: Adding a new disk? Simply include it in the LVM group and create additional LVs. Conversely, removing a disk from the group safely removes its associated LVs.

Why use logical volume management?

The goal of LVM is to facilitate managing the conflicting storage needs of multiple end users. The most common use case for LVM is to mark physical devices as PVs and then use these devices to create a VG pool from which LVs can be allocated to increase disk space.

What is a logical volume?

A Logical Volume is like a partition on a physical disk. Multiple LVs can be carved out of a Volume Group, which itself is a storage pool consisting of multiple PVs. The combined storage capacity of the PVs is abstracted by the VG in order to create logical units of storage called LVs.

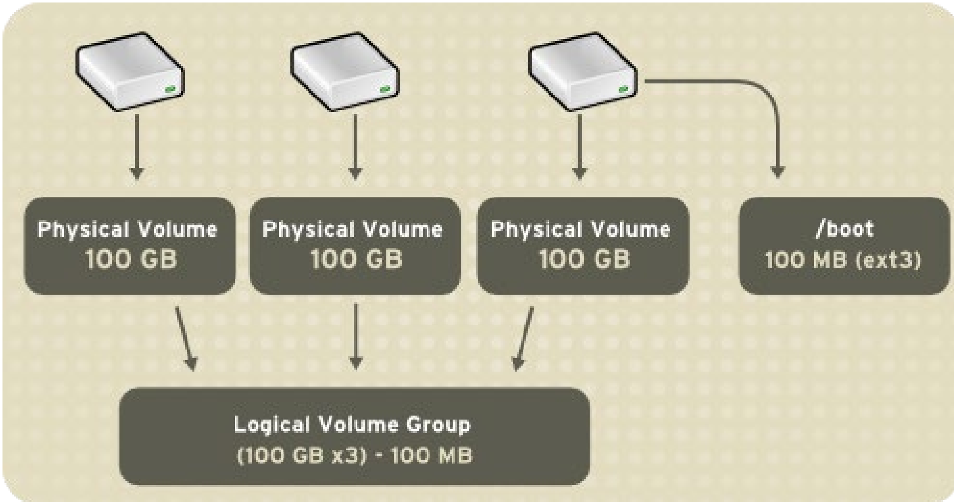


Figure 11.1. Logical Volumes

The volume groups can be divided into logical volumes, which are assigned mount points, such as /home and / and file system types, such as ext2 or ext3. When "partitions" reach their full capacity, free space from the volume group can be added to the logical volume to increase the size of the partition. When a new hard drive is added to the system, it can be added to the volume group, and partitions that are logical volumes can be increased in size.

/media for temporary mounted file system and /mnt is for permanent file system.

Great article about Linux directories.

/root

I have said that /home is the place where all users and their files reside. Now what about root? Root is a special user. It has total control over everything in the system. It would not be right to treat it as a normal user, am I right?

Since root is all mighty and powerful, it deserves its own home folder. The /root folder is the home folder of the root user.

What is the difference between update and upgrade an application?

Update:

- An update typically refers to a release that includes fixes for bugs, security patches, or minor improvements to an existing version of the software.
- Updates are usually smaller in scope and are focused on addressing specific issues or adding minor features.
- Updates are often released more frequently and are intended to keep the software running smoothly and securely.

Upgrade:

- An upgrade involves a more signiaficant change to the software, often with the introduction of new features, major improvements, or a shift to a new version number.
- Upgrades might also include changes to the user interface, performance enhancements, or other substantial modifications.
- Unlike updates, upgrades are less frequent and may represent a new iteration or version of the software.

In summary, an update is a minor change addressing specific issues, while an upgrade is a major revision introducing significant improvements to the software.

What AppArmor is?

AppArmor
It's a security system that sets rules for different programs on your operating system. It controls what each program can access and do, like reading files, connecting to the internet, or changing settings.

My understanding of AppArmor:

AppArmor is a mandatory access control (MAC) system, not media access control. It assigns strict access rules to individual programs through defined profiles. These profiles determine which resources, such as files and network connections, each program can access.

AppArmor offers two main modes:

Enforce mode: When a program attempts to access resources beyond its designated permissions, AppArmor blocks the action. This helps prevent unauthorized programs from compromising the system.

Complain mode: This mode allows you to create profiles and monitor program behavior before enforcing them. AppArmor records attempted actions that would be blocked in enforce mode, providing valuable insights into the program's resource needs. This information helps to refine and finalize the security profile without compromising system stability.

~ouboukou

For example, one could restrict the web browser to only letting users access files in their home directories. This would prevent a scenario where Alice would try to upload or share files owned by Bob without his knowledge.

While AppArmor has different ways to interact with profiles and control behavior, technically, it only has two primary modes:

1. Enforce Mode: This is the default mode where AppArmor actively enforces the access control defined in a profile. Any attempt by a program to violate the policy will be blocked, and the violation logged. This mode provides the most security, but it requires profiles to be well-defined and tested to avoid hindering legitimate program functionality.

2. Complain Mode: This mode serves for learning and development purposes. While it defines access control like enforce mode, it doesn't actively block violations. Instead, it logs them, allowing you to observe and understand how a program interacts with the system. This is helpful for creating new, accurate profiles without risking disruption.

Great article about AppARmor

Command to check APParmor status:

- aa-status
- sudo systemctl status apparmor

What is a default gateway?

A default gateway forwards data from one network to another.

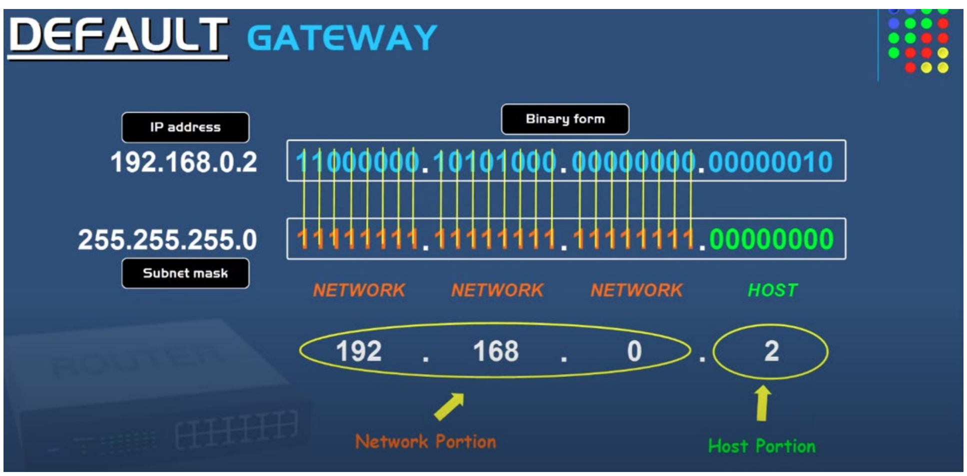
Command to check Public IP address:

- curl ifconfig.me

An IP address consist of two parts:

- Network address
- Host address

A subnet mask reveals how many bits in the IP address are used for the network by masking the network portion on the IP address.



Public vs Private IP addresses

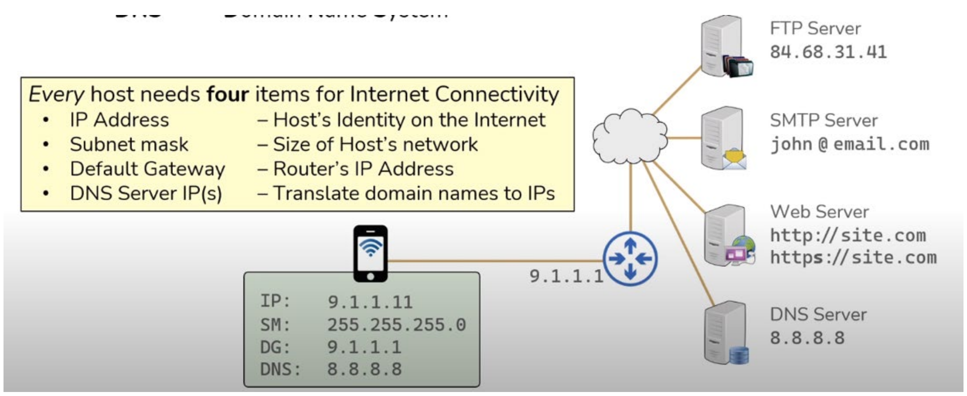
- Public IP addresses are registered on the Internet.
- Public IP addresses are unique.
- Engineers developed private IP addresses because all IP addresses (IPV4) will be consumed in a matter of time.
- Private IP addresses are not publicly registered on the internet.
- If you want to access the Internet, you will need a public IP Address.
- Private IP addresses are only used internally.
- DHCP is a built-in service in routers which assign each device a Private IP.

- When internal devices want to access the Internet their private IP addresses will be translated to the one public IP Address, the service that translates IP addresses is called NAT.
- NAT (Network Address Translation) is what translates a set of IP addresses to another set of IP addresses.
- NAT is a built-in service for routers.
- NAT translates from private to public and vice versa.

Servers are computers with software installed that respond to requests, software such (ftp, SMTP, web...)

Every host needs 4 items to achieve internet connectivity:

- IP Address - Host's Identity on the internet
- Subnet mask - Size of Host's network
- Default Gateway - Router's IP address
- DNS Server IP - Translate domain names to IPs



The most Common Intrnet Protocols:

DNS: Domain Name System

DHCP: Dynamic Host Configuration Protocol
DHCP Server provides IP address, subnet mask, Default gateway, and DNS for client.

HTTP: Hyper Text Transfer Protocol

SSL: Secure Sockets layer

TLS: Transport layer security

HTTPS: HTTP secured with SSL/TLS

FTP: File Transfer Protocol

SMTP: Simple Mail Transfer Protocol

What is SSH?

OpenSSH is a remote management tool, that gives you access to run commands on another machine.

SSH is a network communication protocol that uses client server model to let two computers communicate with a secure and remote connection.

The Secure Shell (SSH) protocol is a method for securely sending commands to a computer over an unsecured network. SSH uses cryptography to authenticate and encrypt connections between devices. SSH also allows for tunneling, or port forwarding, which is when data packets are able to cross networks that they would not otherwise be able to cross. SSH is often used for controlling servers remotely, for managing infrastructure, and for transferring files.

What does SSH do?

- Remote encrypted connections.
- The ability to tunnel.

What is the difference between the two configurable files sshd_config and ssh_config?

If you don't change the port in SSHD file, while connecting if you don't insert –p followed by the port number will not connect, meanwhile if you didn't change the port number the ssh command will try to connect to the server on the default port 22.
In other hand, changing the port number secure your ssh connection more.

In this file there is a line PermitRootLogin, changing this line to no will be more secure because you will have to connect with a username, and all the hackers will try to log in as a root user.

WHAT IS SSH PORT FORWARDING (SSH TUNNELING)?

SSH port forwarding, also known as SSH tunneling, is the process of transmitting data over an encrypted secure shell connection between a local and distant server. It allows users to access services that firewalls would otherwise restrict or prevent.

Port forwarding is like forwarding a message between two people. Bob may send a message to Alice, who in turn passes it to Dave. Similarly, port forwarding sends data packets directed at an IP address and port on one machine to an IP address and port on a different machine.

For example, imagine an administrator wants to make a change on a server inside a private network they manage, and they want to do so from a remote location. However, for security reasons, that server only receives data packets from other computers within the private network. The administrator could instead connect to a second server within the network — one that is open to receiving Internet traffic — and then use SSH port forwarding to connect to the first server. From the first server's perspective, the administrator's data packets are coming from inside the private network.

What is SSH used for?

- Remotely managing servers, infrastructure, and employee computers
- Securely transferring files (SSH is more secure than unencrypted protocols like FTP)
- Accessing services in the cloud without exposing a local machine's ports to the Internet
- Connecting remotely to services in a private network
- Bypassing firewall restrictions

Bet article about SSH from Cloudflare

Private key and public key

A public key is designed to encrypt data, while the private key is used to both encrypt and decrypt data.

How The Private key and public works

In SSH connections using public and private keys, the client never shares its private key with the server. Instead, the server sends a challenge to the client. The client uses its private key to sign the challenge and sends the signed challenge back to the server. The server then uses the client's public key to verify the signature. If the signature is valid, the connection is established, proving the client possesses the corresponding private key.

What is a Port?

A port is a logical connection that is used by programs and services to exchange information.

Ports are always associated with an IP address, example:
66.94.34.13 : 3389
21.49.120.53 : 80

Ports numbers range from 0 – 65535

Examples of ports used on daily basis:

- 80 – HTTP – used for web pages
- 20/21 – FTP – File Transfer Protocol
- 443 – HTTPS – Secure web pages
- 22 – SSH – Secure shell Connection

When a server on the internet wants to communicate with your gadgets on your home network (external connection from outside your home network) most router by default block that request for security, it only allow outside traffic IN if you initiated the connection.

An IP address determines the location of that server.
A Port number determines which service or program on that server it wants to use.

Port numbers are broken down into 3 categories:

1. Port Numbers from 0 – 1023 are called System or Well-known ports.
2. Port Numbers 1024 – 49151 are called user or registered ports.
   These are ports that can be registered by companies and developers for a particular service.
   example: 1102 – Adobe Server | 1433 – Microsoft SQL Server | 1527 – Oracle
3. Port Numbers 49152 – 65535 are called Dynamic or Private ports. These are client-side ports that are free to use.

Private ports are used by regular users to establish a connection with another server.

Shell Command to Transfer files using SSH between two servers:

scp = OpenSSH secure file copy

- scp -P 4242 ~/Desktop/text oubokou@10.13.248.231:~/text

To generate public and private key:

- Ssh-keygen

To get hostname IP address:

- hostname –I :

3 Methods to connect to tour server via SSH

Method 00: Shell command to connect via SSH.

- ssh usrname@123.25.00.21 -p 4242
- ssh -p 2222 john@132.256.25.1
- To disconnect from the server in command line press: ctrl + d

Method 01: Connect to a server via SSH using config File.

```
c3r7p7% cd ~/.ssh
c3r7p7% pwd
/nfs/homes/ouboukou/.ssh
c3r7p7% ls
config  id_rsa  id_rsa.pub  known_hosts  known_hosts.old
c3r7p7% cat config
Host b2br
  Hostname 10.14.55.18
  Port 4242
  User ouboukou
c3r7p7%
```

Method 02: Authorized Keys.

You can connect to the server using the pub key. To do so just past your machine pub key in a file inside .SSH directory on your server and call it "authorized_keys".

Or you can use this command:

- ssh-copy-id -i ~/.ssh/id_rsa.pub username@172.105.7.26 (replace it with machine Ip address)

To allow specific user to connect via SSH, you can edit sshd_config and add:

- AllowUsers user_name_1 user_name_2 user_name_3

The above command to allow users is valid but it is not practical, a practical way is to create a group called for ex: SSH_USERS and then add user to that groups:

- AllowUsers SSH_USERS

While generating the public key and the private key, it's a good practice to set passphrase, in case someone gets hold of that key they can't do anything with it, because it's protected with a password.

Best YouTube video that shows best practices of SSH

What are Internet Protocols?

There are protocols for sending packets between devices on the same network (Ethernet), for sending packets from network to network (IP), for ensuring those packets successfully arrive in order (TCP), and for formatting data for websites and applications (HTTP). In addition to these foundational protocols, there are also protocols for routing, testing, and encryption. And there are alternatives to the protocols listed above for different types of content — for instance, streaming video often uses UDP instead of TCP.

What is IP&TCP?

TCP maintains the connection with the sender from before the first puzzle piece is sent to after the final piece is sent.

IP is a connectionless protocol, which means that each unit of data is individually addressed and routed from the source device to the target device, and the target does not send an acknowledgement back to the source. That's where protocols such as TCP come in.

TCP = Transmission Control Protocol.

IP = internet Protocol.

UDP = The User Datagram Protocol.

What is TCP/IP?

The Transmission Control Protocol (TCP) is a transport protocol, meaning it dictates the way data is sent and received. A TCP header is included in the data portion of each packet that uses TCP/IP. Before transmitting data, TCP opens a connection with the recipient. TCP ensures that all packets arrive in order once transmission begins. Via TCP, the recipient will acknowledge receiving each packet that arrives. Missing packets will be sent again if receipt is not acknowledged.

TCP is designed for reliability, not speed. Because TCP has to make sure all packets arrive in order, loading data via TCP/IP can take longer if some packets are missing.

TCP and IP were originally designed to be used together, and these are often referred to as the TCP/IP suite. However, other transport protocols can be used with IP.

What is host-only, NAT and bridged mode in virtual box?

- **Host-only:** In this mode, the virtual machine can only communicate with the host machine and other VMs in the same network mode. This offers isolation and is useful for testing purposes.
- **NAT:** The VM utilizes the host machine's network connection to access the internet, but its network activity appears to originate from the host. This provides basic external connectivity while maintaining some privacy.
- **Bridged:** The VM acts like a separate device on the physical network. It gets its own IP address and can communicate with any other device on the network, including the host machine. This offers full network access but comes with security considerations.

What is a Firewall?

Definition:

A firewall is a network security device that monitors incoming and outgoing network traffic and permits or blocks data packets based on a set of security rules. Its purpose is to establish a barrier between your internal network and incoming traffic from external sources (such as the internet) in order to block malicious traffic like viruses and hackers.

How does a firewall work?

Firewalls carefully analyze incoming traffic based on pre-established rules and filter traffic coming from unsecured or suspicious sources to prevent attacks. Firewalls guard traffic at a computer's entry point, called ports, which is where information is exchanged with external devices. For example, "Source address 172.18.1.1 is allowed to reach destination 172.18.2.1 over port 22."

A firewall works by filtering the incoming network data and determines by its rules if it is allowed to enter a network, these rules are also known as an access control list.

Firewall Rules can be based on:

- IP addresses - Domain names
- Protocols - Programs
- Ports - Keywords

What is Linux firewall (= Uncomplicated Firewall)?

For the longest time, the security of Linux was in the hands of **iptables** (which works with the underlying netfilter system). Although incredibly powerful, iptables is complicated—especially for newer users. To truly make the most out of that system, it may take weeks or months to get up to speed. Thankfully, a much simpler front end for iptables is ready to help get your system as secure as you need.

That front-end is the Uncomplicated Firewall (UFW). UFW provides a much more user-friendly framework for managing netfilter and a command-line interface for working with the firewall. On top of that, if you'd rather not deal with the command line, UFW has a few GUI tools that make working with the system incredibly simple.

The default firewall configuration tool for Ubuntu is ufw. Developed to ease iptables firewall configuration, ufw provides a user-friendly way to create an IPv4 or IPv6 host-based firewall.

By default, UFW is disabled.

UFW Command line:

Note: UFW command requires root permissions.

UFW can be used in two ways:

1. **Simple syntax:** Specifies a port and (optionally) the protocol.
2. **Full syntax:** Specifies source, destination, port, and (optionally) the protocol.

To turn UFW on with the default set of rules:

- Sudo ufw enable

To check the status of UFW:

- Sudo status ufw
- Sudo status ufw verbose

The verbose option, when added to the ufw status command, provides more detailed information about the firewall configuration.

If you want to deny or allow traffic on port 22 (ssh):

- Sudo ufw [allow / deny] 22

In case you want to add protocol to this port:

- sudo ufw deny 22/tcp

What happens if you don't happen to know the port number for a service?

You can use the name of the service, for ex:

- sudo ufw allow ssh

Of the available arguments, the ones you'll use the most with the ufw command are:

- **allow**
- **deny**
- **reject**
- **limit**
- **status:** displays if the firewall is active or inactive
- **show:** displays the current running rules on your firewall
- **reset:** disables and resets the firewall to default
- **reload:** reloads the current running firewall
- d**isable:** disables the firewall

Delete Existing Rule

To delete a rule, simply prefix the original rule with delete. For example, if the original rule was:

- ufw deny 80/tcp

Use this to delete it:

- sudo ufw delete deny 80/tcp

Working with numbered rules

Listing rules with a reference number

You may use status numbered to show the order and id number of rules:

- sudo ufw status numbered

Delete numbered rule

You may then delete rules using the number. This will delete the first rule and rules will shift up to fill in the list.

- sudo ufw delete 1

Allow / Deny Access

Allow by Specific IP

- sudo ufw allow/deny from <ip address>

example: To allow packets from 207.46.232.182:

- sudo ufw allow/deny from 207.46.232.182

Great article about ufw command and best practices.

MAJOR And MINOR

The major Click the link to understand MAJ:MIN

identifies the driver associated with the device. For example, /dev/null and /dev/zero are both managed by driver 1, whereas virtual consoles and serial terminals are managed by driver 4; similarly, both vcs1 and vcsa1 devices are managed by driver 7. The kernel uses the major number at open time to dispatch execution to the appropriate driver.

The minor number is used only by the driver specified by the major number; other parts of the kernel don't use it, and merely pass it along to the driver. It is common for a driver to control several devices (as shown in the listing); the minor number provides a way for the driver to differentiate among them.

Additional Details about MAJOR and MINOR:

- **Number ranges:** Major numbers typically range from 0 to 255, while minor numbers can vary depending on the device driver.
- **Device nodes:** Major and minor numbers are combined to form device nodes (e.g., /dev/sda1) which applications use to access devices.

Click the link to understand MAJ:MIN

Shell Commands for Managing Users and Groups

Command to change group name:

- sudo groupmod -n NEW_GROUP_NAME OLD_GROUP_NAME

Adding a user into a group without group creation:

- sudo adduser --ingroup users john

How to delete a user

- Sudo userdel user_name

You can add a user to sudo group by two ways:

1. Editing sudoers file in /etc/sudoers
2. Running the bellow command
    - sudo usermod -aG sudo <username>

      -G: This option specifies the groups to which the user should be added.

To see what groups a user is assigned to:

- Groups user_name

To delete a group:

- Delgroup groupe_name

To remove a user from a specific group MTHODE 01:

- gpasswd –d Tim python

To remove a user from a specific group MTHODE 02:

- sudo gpasswd –d user_name group_name

If you want to know the location of specific command:

- Which command_name

Note: chage command only makes changes for specific users and not all users or added users on the futures, so it will be necessary to edit the password configurations files.

How to check user expiration and password:

- sudo chage -l user_name

Command to set password expiration date:

- sudo chage -M 30 user_name

To set the minimum before changing the password:

- Sudo chage –m 2 user_name

To change a user password:

- Passwd user_name

To check groups on a system:

- cat /etc/groups
- cut -d : -f1 /etc/passwd

To create a new group:

- Sudo groupadd goupe_name

What is a shell?

From Wikipedia: a shell is a computer program that exposes an operating system's services to a human user or other programs. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation. It is named a shell because it is the outermost layer around the operating system.

A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Types

In Unix, there are two major types of shells:

- Bourne shell − If you are using a Bourne-type shell, the $ character is the default prompt.
- C shell − If you are using a C-type shell, the % character is the default prompt.

Bash Scripting

Shebang is telling the system which interpreter is intended to run the script.

And by the shebang we can now type of the script we are running.

#!/bin/bash

Whenever we reference a variable in bash, we must include a dollar sign ($) in front of its name.

Example:

Declaring variable:	My_name="OUSSAMA"
Referencing variable:	Echo $My_name

When you declare a variable in bash, it is tied to that session, once you close the window the session is gone, so any variable that you might have declared are whipped out.

By adding the variable to the bash script, we don't have to declare them each time we want to run the script!

We can store the output of a command in a variable (variable substitution) by following this syntax:

- Files=$(ls)   And then echo the file: echo $files
- Curnetdirctory=$(pwd)   And then echo the file: echo $Curnetdirctory

It's better when you declare variables to use small letters so if someone tries to understand your script will be able to distinguish built-in variables by shell environment and the variable you declared.

To check the environment variable withing the session use this command:

- env

To do math in bash, you must use the command expr (short for: evaluate expressions), and here is an example:

- expr 100 / 10
- expr 102+41
- expr 10 /* 4 (in bash * is considered a wildcard so you must escape it using the \)

If Statements

The correct syntax for using if statement is:

```
My_var=200

If [ $my_var –eq 200 ]

then

Echo “the condition is true.”

else

Echo “the variable does not equal to 200.”

Fi
```

NOTE: The spaces before & after the square bracket are requested.

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

-eq stands for equals to.

-ne stands for not equals to.

-gt stands for greater than.

An example of if condition that check if a file exists or not:

If [-f ~/my_file]

To check test* manual page for test; run **man test**

* (if condition statement for bash).

The command below is preferably used in scripts, so you don't have to be present at the moment the script is running to choose yes or no to proceed.

- Sudo apt update && sudo apt install –y package

Exit Codes

Exit codes provide feedback on a command's execution status, indicating success or failure.

In Bash scripting, a built-in variable ($?) stores the exit code of the most recently executed command.

Example:

- cd /root => Permission denied
- echo $? => 0

Note: zero means success, other than zero means non-success.

Bash script example of using exit code:

```
-----------------------------------------------------------------------------------------
#!/bin/bash
        package = htop
        sudo apt install $package
        if [ $? -eq 0 ]
        then
                echo “The installation of $package was successful.”
                echo “The new command is available here:”
                whiche $package
        else
                echo “$package failed to install.”
        fi
-----------------------------------------------------------------------------------------
```

NOTE: When you want to differentiate what happens in a script based on the exit code you need to make sure that you are checking the exit code at the appropriate time.

How to manipulate the exit code of a task?

```
--------------------------------------------------------------------------

#!bin/bash
sudo apt install nonexist
exit 0
echo $?

--------------------------------------------------------------------------
```

When you run this code, you will always exit the code at line 3, and set the exit code to 0. the next lines will not be executed. (exit in bash script means exit!)

While Loops

There is a new concept in while loops while using them in bash scripts, that concept is sleep as shown in the bash script example bellow:

```
--------------------------------------------------------------------------------

#!/bin/bash

while [ -f ~/testfile ]
do
        echo “As of $(date), the test file exists.”
        sleep 5
done

echo “As of $(date), the test file has gone missing”

--------------------------------------------------------------------------------
```

What this code does is keep running till the testfile does not exist, but the script only executes every 5 second, without the sleep 5 the script will be executed every second dozen times and that will harm the CPU.

For Loops

Bash script example:

```
#!/bin/bash
for file in logfiles/*.log
do
	tar -czvf $file.tar.gz $file
done
```

---

the script takes each .log file in the 'logfiles' directory, creates a compressed archive (.tar.gz) with the same name as the original log file, and repeats this process for all log files in the directory.

Where to store bash scripts

It's better to store your script in /usr/local/bin so that any user can access it with the appropriate permissions.

When running the command echo $PATH you see the path that you execute command from.

NOTE about the $PATH: When you type a command in the terminal, the system searches for the executable file associated with that command in each directory listed in the PATH environment variable, from left to right.

The PATH variable contains all the directories that shell will look inside to find a script or binary that can run.

If you want to add a different directory to your path variable, you can use the bellow command:

Export PATH=/another/dir/bin:$PATH

But that is not recommended!

NOTE: when you put your script on one of the $PATH directories, you easily run the script from terminal by only typing the name of the command, without the execution part (./command)

DATA streams (standard input and standard output)

Standard output is when executing a command and seeing the result, that standard output.

In short standard output is output that printed to the screen that doesn't encounter an error.

To check which exit status you received, you can run echo $? to display the content of the $? variable.

When scripting, it's essential to avoid any prompts. The primary purpose of scripting is automation, and it becomes challenging to automate if someone needs to babysit a script!

The command bellow will send standard output to find_rsulat.txt and standard error to the file find_errors.txt:

- Find /etc -type f >find_resualt 2>find_error

Now after executing the command all the outputs will be sent to the 2 both files, based on their standards. Numbers 1 and 2 allow us to do that, but we can also use & operator to send both std_error and std_output.

Note: if you don't specify the 1, because standard output is applied if you don't specify anything.

Real-word bash script example:

```
------------------------------------------------------------------------------------------

        #!/bin/bash
        logfile=/var/log/updater.log
        errorlog=/var/log/updater_error.log

                sudo apt update 1>>$logfile 2>>$errorlog
                sudo apt dist-upgrade 1>> $logfile 2>>$errorlog

------------------------------------------------------------------------------------------
```

> to override

>> to append == تضيف | تلحق

Functions

An example of a function in bash scripting:

```
Check_exit_status()
{
	if [ $? -ne 0 ]
	then
		echo "An error occurred, please check the $errorlog file"
	fi
}
```

Case Statement

```
  GNU nano 5.6.1                    myscript.sh
echo "1 - Arch"
echo "2 - CentOS"
echo "3 - Debian"
echo "4 - Mint"
echo "5 - Ubuntu"
echo "6 - Something else..."

read distro;

case $distro in
    1) echo "Arch is a rolling release.";;
    2) echo "CentOS is popular on servers.";;
    3) echo "Debian is a community distribution.";;
    4) echo "Mint is popular on desktops and laptops.";;
    5) echo "Ubuntu is popular on both servers and computers.";;
    6) echo "There are many distributions out there.";;
    *) echo "you didn't enter an appropriate choice."
esac
```

When the script is running and the case statement is compared from value 1 – 6 or however how many you create, then asterisk case will be executed. It's good for checking corner cases.

The asterisk statement doesn't require a semicolon;

On each case statement you can have as many commands as you wish but must end with two semicolons.

Scheduling Jobs (Part 1)

There is a command called at, that allows users and sysadmin to run jobs and scripts at a specific time. The at command doesn't come built-in on the system, but rather you must install it.

Scheduling Jobs (Part 2)

Scheduling jobs and script using crontab.

If you are using crontab for the first time, you must run the command crontab –r and a popup will be displayed to choose the editor you want to proceed with.

When using the crontab using the command crontab on terminal a text editor will open the crontab, but the path to that crontab will be in temp dir.

That is because if you misused that file, will not affect the original file, and if you used crontab correctly as far as crontab is concerned, then changes will be saved on the original file on etc directory.

When opening the crontab file the last line will be a format on how to use the crontab, and i will explain it in detail:

Crontab format:

 # m h dom mon dow command

Real-world example:

30 1 * * 5 /usr/local/bin/script_name

The corn job i used in b2br:

***/10 * * * * ./Monitoring.sh | wall –n**
(-n flag with will broadcast the output without a header.)

m: means minutes, and the command will be executed every time based on the minute you set here. But as we move to the right, we will narrow this down.

h: This is the hour section.

dom: short for “Day of the month”, and day of the month will from 1 – 31, if you set an asterisk (*) that mean it doesn't matter when it comes to that filed, that filed isn't a deciding factor. If we set dom to 10, then the script will be run March 10, April 10 and so on...

mon: the field for the month, and by coming further to the right we are narrowing down the jobs on what month we want the jobs to run.

dow: short for “day of the week”, 0 and 7 are for Sunday, 1 for Monday, 5 for Friday.

And the last part is the path to the script.

To display the corn jobs by the use:

- crontab –l

To edit / add a job to the crontab:

- crontab –e

You can set cron jobs based on these scheduling options (@yearly, @annually, @monthly, @weekly, @daily, @hourly, @reboot):

- @weekly echo “Now we are using corn jobs” >> /home/ouboukou/output.txt

Website generator for cron jobs

Arguments

The variable $1 is the first argument when you run a bash script

Real world example of bash script:

```
#!/bin/bash
lines=$(ls –lh $1 | wc -l)

if [ $# -ne 1 ]
    then
        echo “This script requires exactly one directory path passed to it.”
        echo “Please try again!”
    exit 1

Fi
echo “You have $(($lines-1)) objects in the $1 directory.”
```

#: aka($#) is a shell built-in variable that stores the number of arguments passed in command line.

In bash script we can use aliases so we only call the name we give to a set of command, and it will run as you type the whole command, and here is an example:

Alias Ip-address=”ifconfig | grep broadcast | awk ‘{print $2}’”

After creating this alias wen how

Scheduling Tasks with Cron

Every user on the system has his own set of cronjobs.

Great tutorials about bash scripting:

- [Bash scripting on Linux – YouTube playlist]()
- [Bash Scripting Tutorial - articles]()

~ End Bash scripting

Understanding Physical CPU, Virtual CPU, Cores and Threads

Difference between CPU Cores and Threads

While CPU cores and threads are closely related, they serve different functions in a computer's architecture. A core is a physical component of the CPU that can execute instructions, while a thread is a virtual sequence of instructions that can be executed by a core.

Cores can be visualized as the workers, while threads are the tasks they perform.

Demystifying CPUs, vCPUs, and Cores

Computers rely on central processing units (CPUs) to execute instructions and perform tasks. These CPUs are the hardware brains of our machines, and their performance significantly impacts our computing experience. In the age of virtualization, virtual CPUs (vCPUs) have emerged as software counterparts to physical CPUs, enabling us to run multiple operating systems on a single machine. But how do CPUs and vCPUs work, and how are they related? This article delves into the world of CPUs, vCPUs, and cores, explaining their concepts, functionalities, and the calculations behind vCPU allocation.

Understanding CPUs and Cores

- **CPUs:** CPUs, or central processing units, are the physical chips in our computers responsible for processing instructions and data. They are the workhorses of the system, handling calculations, running applications, and managing overall system performance.

- **Cores:** Modern CPUs often have multiple cores, which are individual processing units within a single CPU. Each core can process instructions independently, allowing for parallel execution of tasks and improved performance. The number of cores in a CPU is a crucial factor in its multitasking capabilities.

vCPUs: The Virtual Counterparts

- **vCPUs:** Virtual CPUs, or vCPUs, are software-created processors that function within virtual machines (VMs). VMs are virtual environments that emulate physical computers, allowing us to run multiple operating systems on a single machine. Each VM can have its own vCPUs allocated from the physical CPU resources of the host machine.

- **vCPU Allocation:** The number of vCPUs that can be allocated to a VM depends on the available physical CPU cores and the capabilities of the hypervisor, which is the software that manages VMs. A common formula for calculating the potential number of vCPUs is (Threads per Core) x (Cores per CPU) x (Number of CPUs). However, this is a theoretical maximum, and the actual number of usable vCPUs may be lower due to resource constraints and hypervisor overhead.

Hyperthreading and Threading

Hyperthreading: Hyperthreading is a technology that allows a single physical CPU core to simulate two logical cores. This means that the operating system can see two virtual cores instead of one, potentially improving multitasking performance. However, it's important to note that hyperthreading does not double the processing power; it only allows for more efficient scheduling of tasks on a single core.

Threading: Threading is a software technique that allows a single process to be divided into multiple threads. Each thread can execute independently, but they share the resources of the process. This enables processes to handle multiple tasks concurrently, improving responsiveness and performance.

Key Takeaways

- CPUs are the physical processing units in our computers, while vCPUs are their virtual counterparts within VMs.
- Cores are individual processing units within a CPU, and hyperthreading allows a single core to simulate multiple cores.
- The number of vCPUs that can be allocated to a VM depends on the physical CPU resources and the hypervisor's capabilities.

What is a virtual CPU (vCPU)?

A virtual CPU (vCPU) is the processor of a virtual machine (VM). Each vCPU represents the ability to run one processing thread at a time. The VM hypervisor manages the scheduling of the vCPU time to the physical CPU.

To check if your CPU supports hyperthreading technology, use the lscpu command and look for "Threads(s) per core", if the value is greater than 1, hyperthreading is supported; if its 1, hyperthreading is not supported.

If the CPU supports hyper-threading, the operating system will recognize each physical core as 2 virtual or logical cores.

So hyperthreading. is virtually double the number of cores that's on the CPU.

What i come up with on the matter of vCPUs:

When a virtual machine is assigned 2 cores from our physical CPU that supports hyperthreading technology and has 10 cores (each physical core has two logical cores). However, assigning a virtual machine with 2 cores "vCPUs" will not give us 4 logical cores but rather 2 logical cores only.

On the host machine, when i looked for the line "Threads(s) per core" on the output of lscpu command i get 2 threads per core, and when i look for the same line on the Virtual Machine i get only 1 thread per core.

The Math Behind Calculating vCPU

vCPU count is largely determined by the manufacturer. It’s calculated by taking the number of processing threads that a chipset offers per core and multiplying the number of occupied sockets. Here’s how it looks:

(Threads per Core x Cores per CPU) x Physical CPU = Number vCPU

- To check the number of threads you have to look for "Thread(s) per core".
- To check the number if core you have look for "Core(s) per socket".
- To check the number of physical CPU, look for "Socket(s)".

Why do we need requiretty?

We can understand the conceptual value of requiretty by considering two scenarios:

Scenario 1: Script automation

Create a simple script on the server that contains a sudo command to perform some privileged action (e.g., sudo echo "Sensitive information" > secret.txt).

Try to run the script from your local machine using ssh [user]@[server] ./script.sh.

With requiretty enabled, the script should fail because it's running in a non-interactive context (no pty allocated), even though SSH provides remote access. This demonstrates how requiretty can prevent accidental or malicious privilege escalation through automated scripts.

Scenario 2: Compromised user account

Imagine a user account on the server gets compromised, and the attacker gains access via SSH.

With requiretty enabled, the attacker would still need physical access to the server to run sudo commands and escalate their privileges for further harm, even if they have SSH access to the account.

Since the attacker only has access through SSH, their pty isn't linked to a physical terminal. This means:

The sudo command will detect the lack of a physical terminal and refuse to execute, even though the attacker has valid SSH credentials.

CPU load & System Load average

Think of it this way:

CPU load: How much work is currently being done by the CPU.

Load average: How many people are waiting in line for their turn to do work.

CPU idle: Percentage of time the CPU is not actively processing tasks. It essentially represents the remaining capacity available for handling new work immediately. Think of it as the "empty space" in the waiting line for processes.

Wall Command

wall is short for write to all. The purpose of the command is to send a quick message to the terminals of all currently logged in users.

Wall command syntax

The syntax of wall is straightforward. The command is followed by options and either the text of the message or a filename:

- wall [options] [message-text/filename]

wall Command Examples:

To Broadcast a message:

- wall [message-text]

To Broadcast a message without a header:

- wall –n [message-text]

Broadcast a Message from a File:

- wall [filename]

Broadcast a Message to a Group:

- wall -g [group-name] [message-text]

Use the wall Command with echo:

- echo "[message-text]" | wall

Use the wall command with a script:

- ./my_script.sh | wall -n

What is the main difference between Primary and logical Volume?

Primary Volumes.

Creation: Directly carved onto the physical disk.

Limitation: Up to four primary partitions per physical disk with MBR partitioning scheme (now largely outdated).

Use Cases:

- Booting an operating system.
- Storing critical system files.
- Storing data, although logical volumes are often preferred for this.

Logical Volumes.

Creation: Located within an extended partition, which itself is one of the four primary partitions.

Limitation: Theoretically unlimited number within an extended partition (limited by disk size).

Use Cases:

- Mainly for storing data due to flexibility and expandability.
- Cannot be used for booting directly (with older MBR scheme).

Additional Differences:

Boot capability: Only primary partitions can be directly booted with MBR (not relevant with GPT).

Flexibility: Logical volumes offer easier resizing and expansion as they are not limited by the four-partition restriction.

- Use primary partitions for essential needs like booting and system files.
- Use logical partitions for flexible data storage.

What is MBR & GPT Partitioning scheme

MBR (Master Boot Record):

- An older partitioning scheme for hard drives.
- Limited to 4 primary partitions per disk (plus an extended partition for more logical partitions).
- Each partition has a maximum size of 2 TB.
- Used on older systems with BIOS firmware.

GPT (GUID Partition Table):

- A newer and more advanced partitioning scheme.
- Allows for up to 128 partitions per disk, all treated equally.
- Supports drives larger than 2 TB.
- Used on modern systems with UEFI firmware (replacing BIOS).

What is Extended Partition:

In the older MBR partitioning scheme, due to the limitation of only having 4 primary partitions, an extended partition was created as a workaround to overcome this restriction. It's essentially a container within one of the four primary partitions that can hold multiple logical partitions. You can think of it as a large box placed inside one of your drawers, and within that box, you can create smaller compartments (logical partitions) to organize your belongings (data).

Here are some key points about extended partitions:

- **Not directly usable:** The extended partition itself can't store data directly. Its sole purpose is to house logical partitions.
- **Logical partition container:** Each logical partition within the extended partition acts like a regular primary partition and can be used for data storage.

What i come up with about sda2 that has only 1k, and extended partition, and primary and logical volume:

Sda2 is the extended partition that is the primary volume that acts as a container for all the logical volume.

```
oubboukou@ouboukou42:~$ sudo fdisk -l /dev/sda
Disk /dev/sda: 30.8 GiB, 33071247872 bytes, 64592281 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xafbf9cb3

Device     Boot   Start      End  Sectors  Size Id Type
/dev/sda1  *       2048  1026047  1024000  500M 83 Linux
/dev/sda2       1028094 64591871 63563778 30.3G  5 Extended
/dev/sda5       1028096 64591871 63563776 30.3G 83 Linux
ouboukou@ouboukou42:~$
```

This output indicates that sd2 occupies most of disk space and serves as a container

What is the main use of shasum command in Linux?

The main use of the shasum command in Linux is to calculate and verify the message digest (also known as a checksum) of a file. This message digest is a unique string of characters that serves as a "fingerprint" of the file's content. Any change to the file, even a single byte, will result in a completely different message digest.

<u>Here are some key applications of the shasum command:</u>

1. Verifying file integrity:

Downloading files from the internet can introduce errors or corruption. You can use shasum to generate the checksum of the downloaded file and compare it to the published checksum provided by the source. If they match, you can be confident that the file hasn't been tampered with.

Checking the integrity of system files is crucial for maintaining system stability. Some packages provide checksums alongside their downloads, allowing you to verify their integrity before installation.

2. Identifying duplicate files

3. Monitoring file changes