Operating Systems & Linux Basics

Key Takeaways

TECHWORLD WITH NANA

Introduction to Operating Systems

TECHWORLD WITH NANA

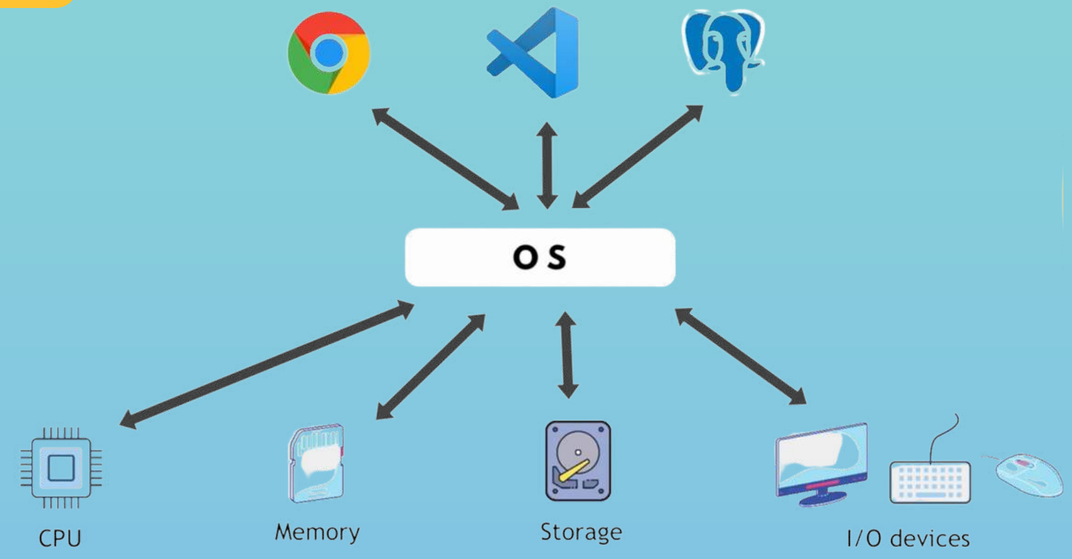
What is an Operating System?

OS is a software managing...

- computer hardware,
- software resources
- and provides common services for computer programs

OS as abstraction layer between applications and hardware

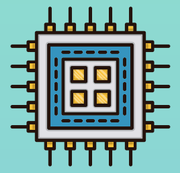
- Instead of applications (like browser) interacting with the computer hardware directly, they **can use the OS as abstraction layer between the two**
- Messy, if every app had to talk directly to the hardware parts
- Apps, like browser can't be installed directly on the hardware



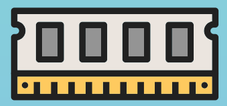
Tasks of an Operating System - 1

1) Resource Allocation and Management

OS manages resources among applications:



- **Process Management (CPU):**
  - 1 CPU can only process 1 task at a time
  - OS **decides which process gets the processor** (CPU), when and for how much time and allocates the processor to the process
  - CPU is switching so fast that you don't notice it



- **Memory Management (RAM)**:
  - Allocating working memory (RAM = Rapid Access Memory) to applications
  - **RAM is limited** on the computer

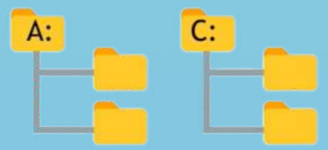
- **Storage Management (Hard Drive):**
  - Also called "secondary memory" - the hard drive
  - **Persisting data long-term**, like files, browser configurations, games, pictures, videos etc.

Tasks of an Operating System - 2

2) File Management

- Files are stored in **structured** way
- A file system is organized into directories

- On Unix system: **tree file system**
- On Windows OS: **multiple root folders**



3) Device Management

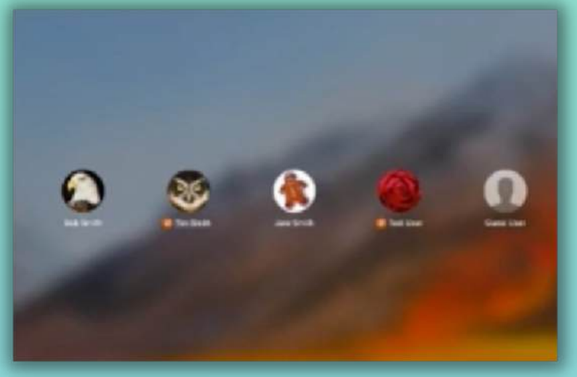
- **Manages device communication** via their respective drivers
- Activities like:
  - Keeping track of all devices
  - Deciding which process gets the device, when and how long
  - ...

OS

Tasks of an Operating System - 3

Other important tasks

- **Security:**
  - Managing Users and Permissions
  - Each user has its own space and permissions

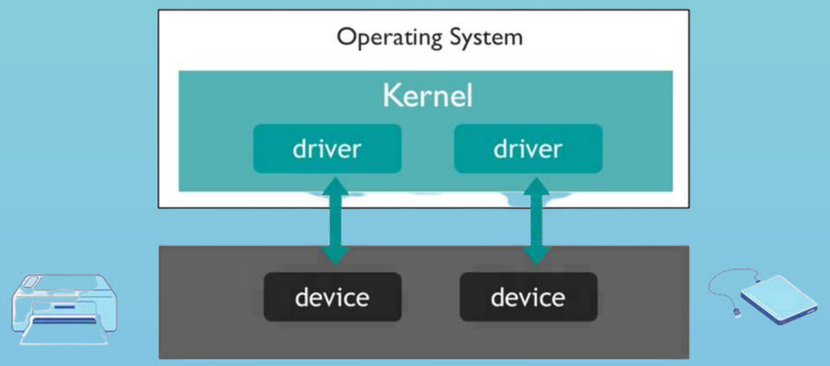


- **Networking**
  - Ports and IP addresses
  - Transmitting outgoing data from all application ports onto the network, and forwarding arriving network packets to processes

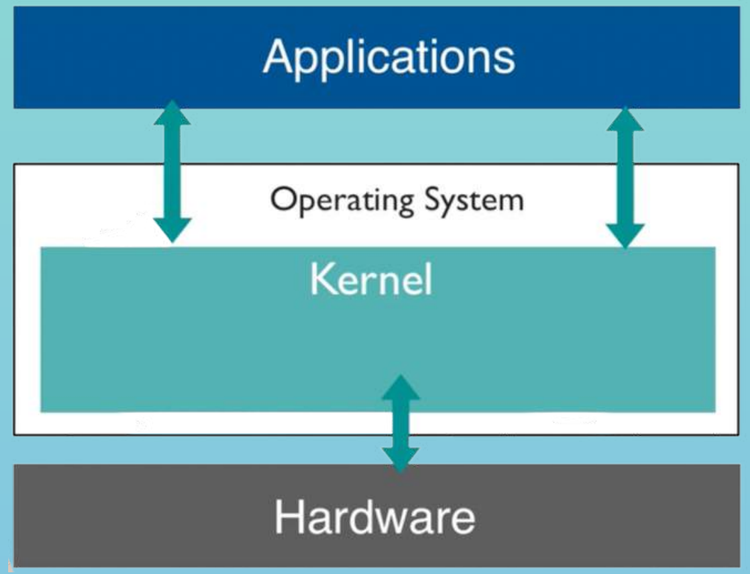
Operating System Components - 1

Kernel

- **Heart** of every OS
- The core program that provides basic services for all other parts of the OS
- Consists of device drivers, dispatcher, scheduler, file system etc.
- One of the first programs loaded on startup
- **Controls all hardware resources via device drivers**



- Kernel starts the process for app
- Allocates resources to app
- Cleans up the resources when app shuts down

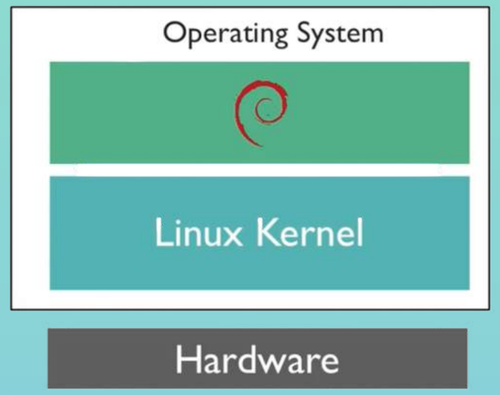


Operating System Components - 2

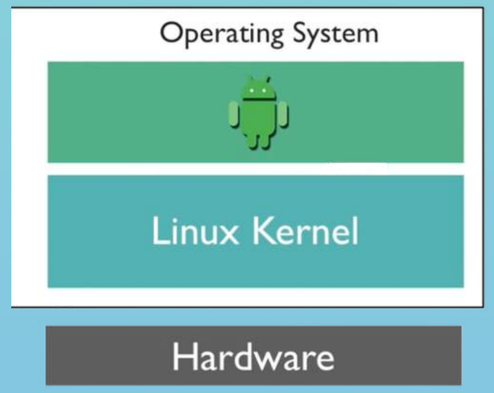
Application Layer

- **On top of Kernel is the application layer**

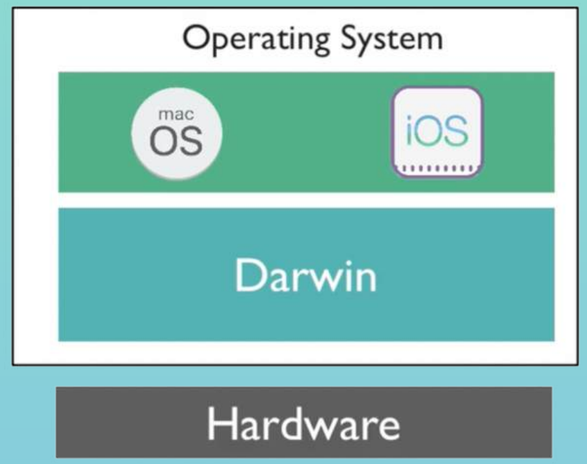
- For example different Linux distributions:
  - Ubuntu, Mint, CentOS, Debian

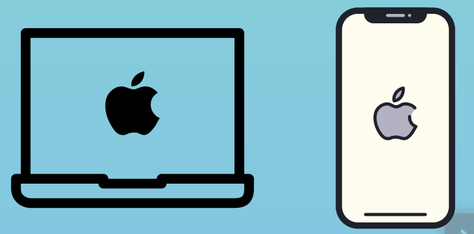


- Different application layers, but based on same Linux kernel
- Android is also based on Linux kernel
- Linux Kernel most widely used



- MacOS and iOS is based on a different Kernel

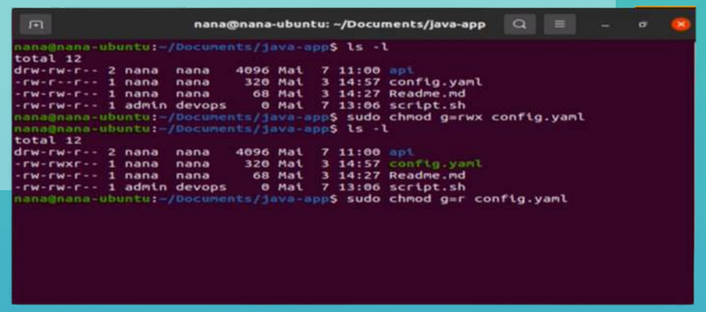




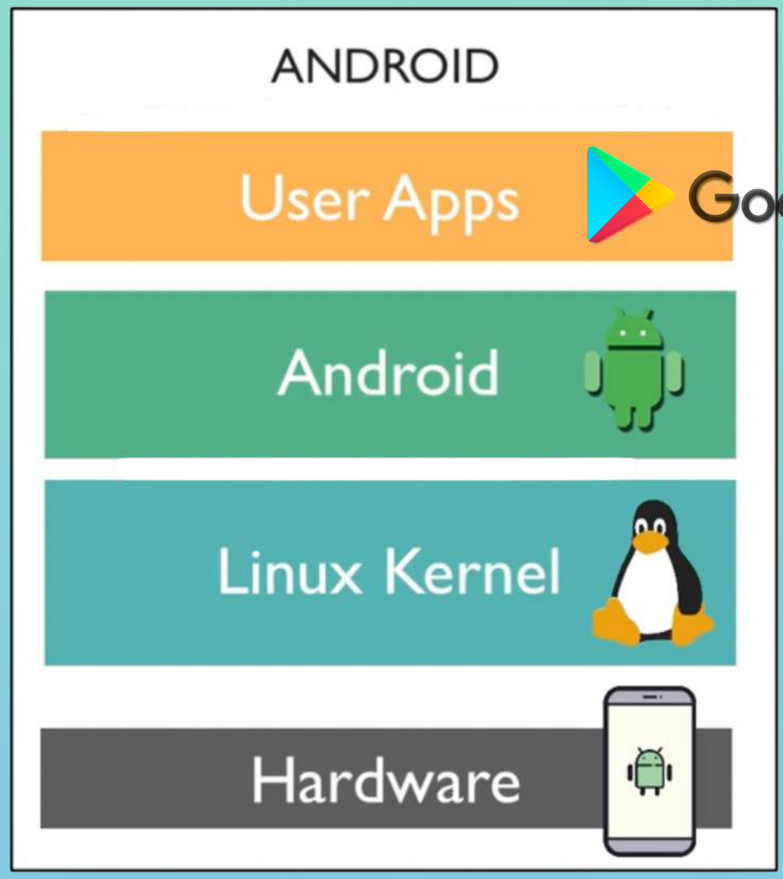
Operating System Components - 3

How to **interact** with Kernel

- Graphical User Interface
- Command Line Interface



Example: Layers of Android phone



3 Main Operating Systems

- Each OS has many versions
- But kernel stays the same!

Client OS vs Server OS

Client OS

- For personal computers with GUI and I/OS devices

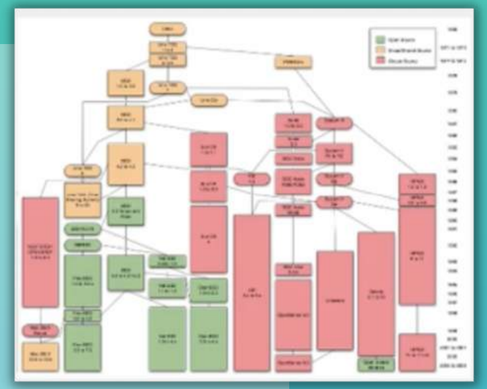
Server OS

- Linux and Windows have server distributions
- But Linux most widely used
- More **light-weight** and performant
- No GUI or other user applications

MacOS vs Linux

- MacOS and Linux: Command Line, File structure etc. **similar**
- Whereas Windows is completely different

UNIX 1970



- Codebase for many different OS
- Developed independent of Linux
- Many OS built on top of Unix
- MacOS Kernel is based on Unix
- To keep them compatible, standards were introduced
- POSIX (Portable Operating System Interface) is the most popular standard

Linux 1991

- Linux was **developed in parallel** to Unix based operating systems
- No source code of UNIX
- Created by Linus Torvalds
- Clone of UNIX or also called "**unix like**"
- Linux and MacOS **both POSIX compliant**

Why learn Linux as a DevOps Engineer

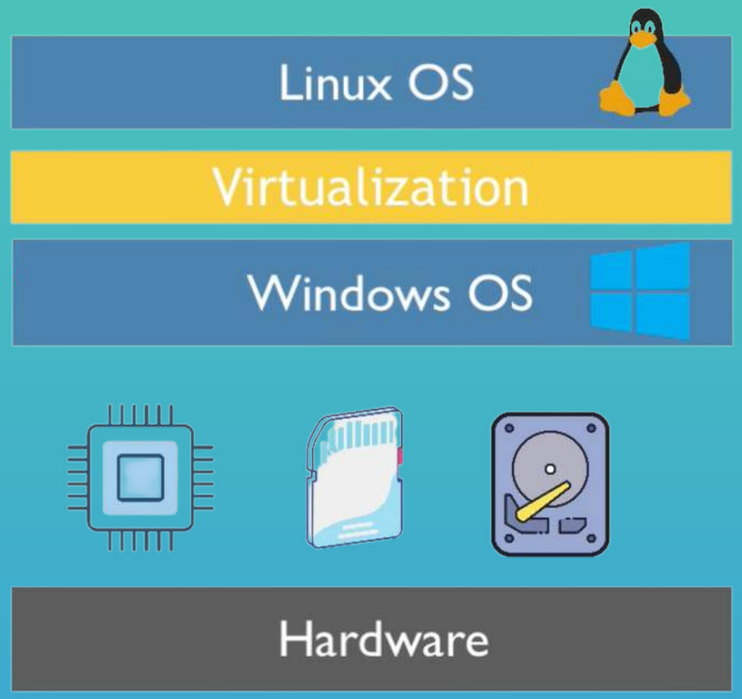
- Linux is mostly used OS for servers!
- Knowing Linux is a must for DevOps Engineers
  - you need to work with servers
  - installing and configuring servers
  - Linux native technologies

Introduction to Virtualization & Virtual Machines

TECHWORLD WITH NANA

What is Virtualization and a Virtual Machine - 1

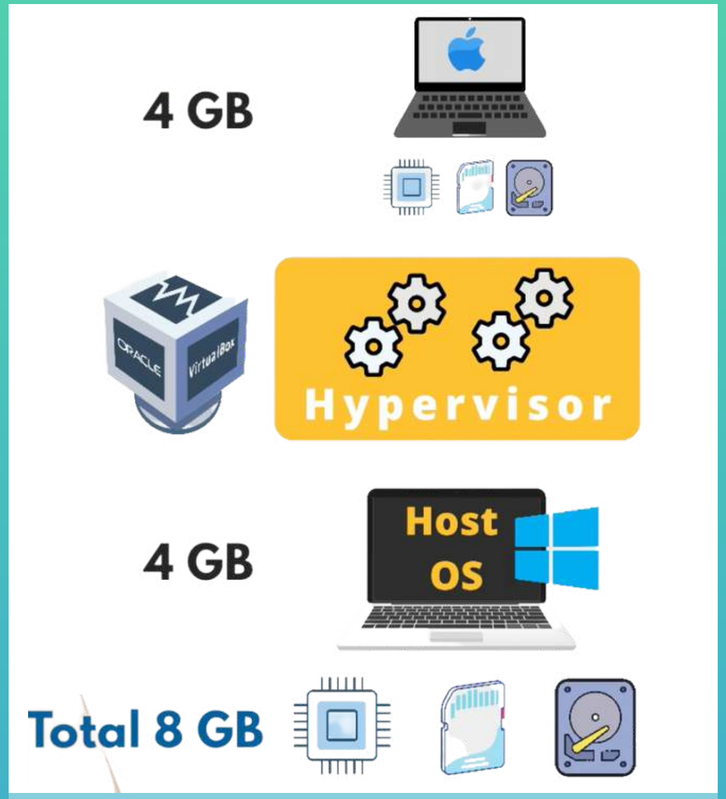
- Virtual Machine is commonly shortened to just "VM"
- VM is a virtual computer running on top of another host computer



How it works - Virtualization

- Virtualization is the process of creating a software-based, or **"virtual" version of a computer**, with dedicated amounts of CPU, memory, and storage that are "borrowed" from a physical host computer
- Makes it possible that **any OS can run on top of any other physical host machine**
- The VM is partitioned from the rest of the system, meaning it's **completely isolated** and can't interfere with the host computer's primary OS

What is Virtualization and a Virtual Machine - 2



- The **hardware resources are shared**
- So you can only give resources you actually have

Hypervisor

- The essential component in the virtualization stack is a piece of software called a hypervisor
- One of the most popular hypervisor is open-source **Oracle VM VirtualBox**

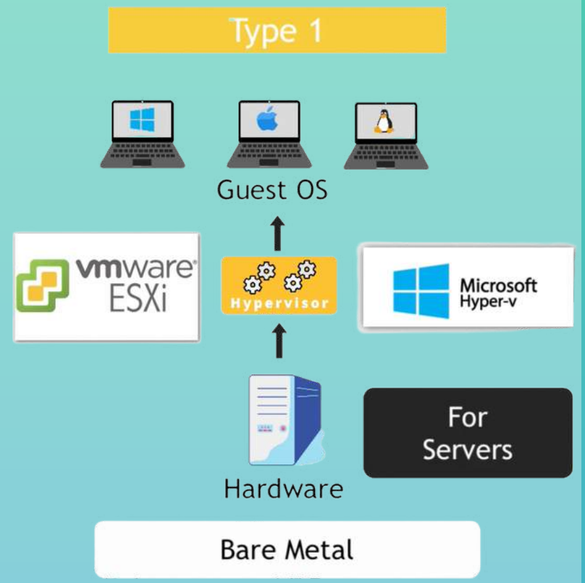
Host OS vs Guest OS

- **Host OS** = runs directly on the hardware
- **Guest OS** = runs on the virtual machine

Hypervisor Types - 1

Type 1 - Native or Bare Metal

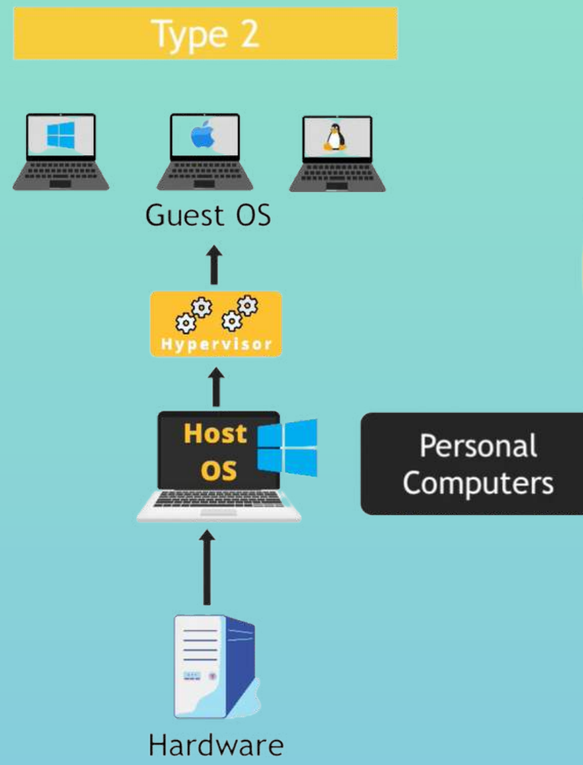
- That **run directly on the host's hardware** to control the hardware, and to monitor the guest OS. The guest OS runs on a separate level above the hypervisor



- Examples of this classic implementation of VM architecture are Oracle VM, Microsoft Hyper-V, VMware ESXi.

Type 2 - Hosted

- Designed to **run within a traditional OS**
- A hosted hypervisor adds a distinct software layer on top of the host OS, and the guest OS becomes a third software level above the hardware



- Examples: Oracle VM VirtualBox, VMware Server, Microsoft Virtual PC, KVM, QEMU and Parallels.

Hypervisor Types - 2

Type 1 - Use Cases

- **Efficient usage of hardware resources (e.g. cloud provider):**
  - Use all the resources of a performant big server
  - Users can choose any resource combinations

Type 2 - Use Cases

- **Learn and experiment**:
  - You don't need to buy a new computer for that
  - You don't endanger your main OS
- **Test your app on different OS**
- **Backing up your existing OS**

Why companies adopt Virtualization

Main benefit: Instead of OS being tightly coupled to the hardware, Virtualization gives an **abstraction** layer, with the following benefits:

- **Security**: Secure very easily
- **Agility and speed:** Spinning up a VM is easy and quick, compared to setting up an entire new server
- **Cost savings**: Efficient usage of hardware resources
- **Portable**: OS as a portable file (VMI - Virtual Machine Image)

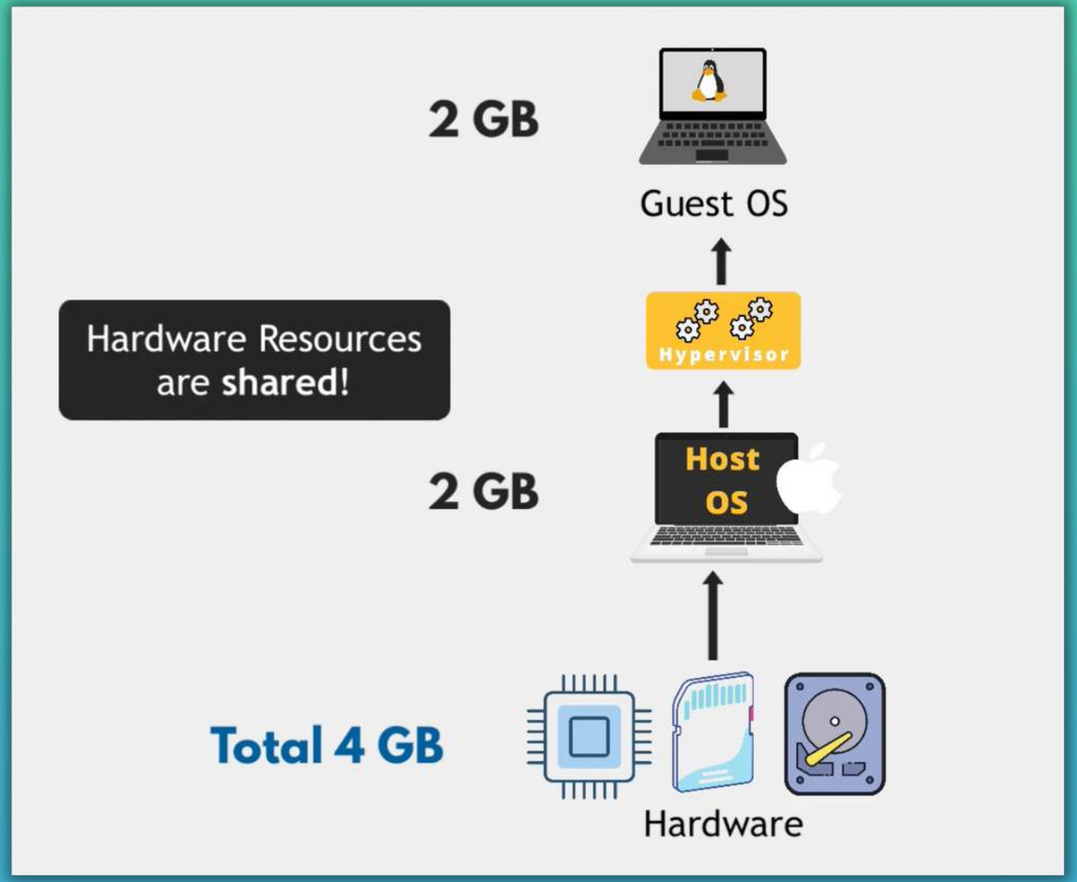
VMI:

- Includes OS and all applications on it
- You can have backups of your entire OS

Setup Linux VM

1) Install VirtualBox Hypervisor

2) Setup Linux Ubuntu Virtual Machine



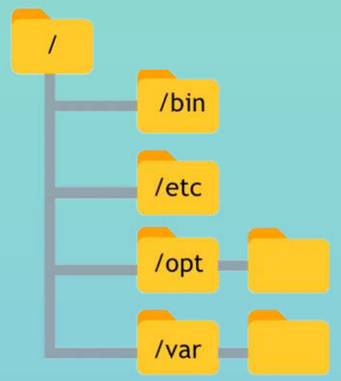
Keep in mind: shard resources

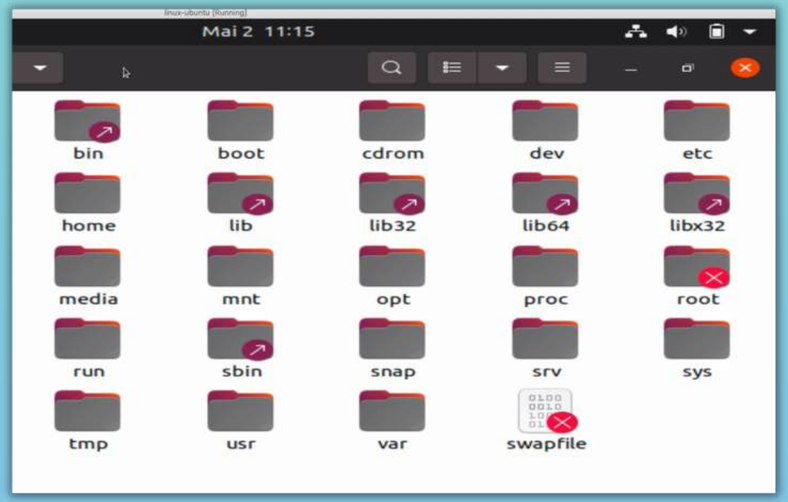
Linux File System

TECHWORLD WITH NANA

Linux File System - 1

- File system is used to handle the data management of the storage
- The Linux file system has a **hierarchical tree structure**
- With **1 root folder**





Windows in comparison

- Has multiple root folders
- C = internal hard drive

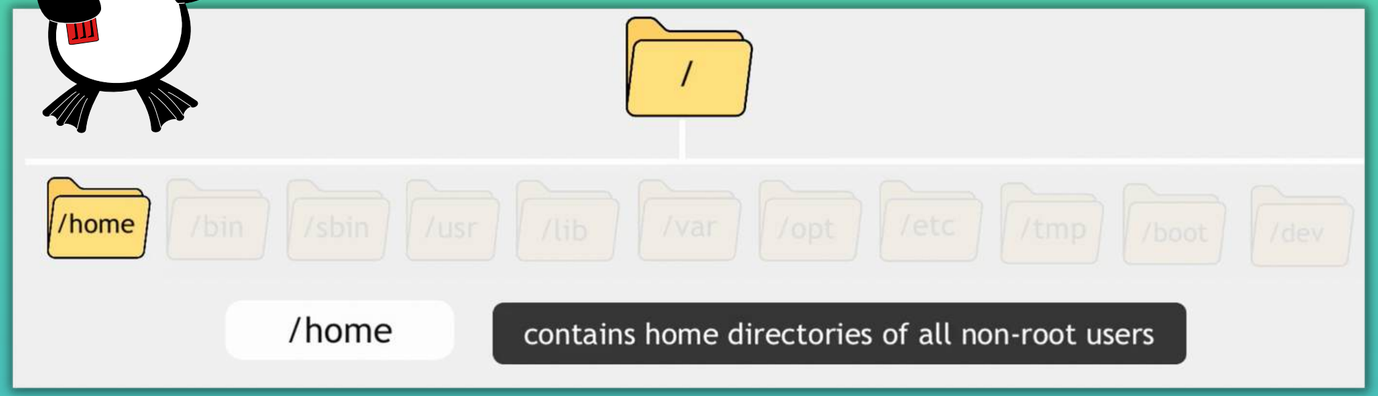
TECHWORLD WITH NANA

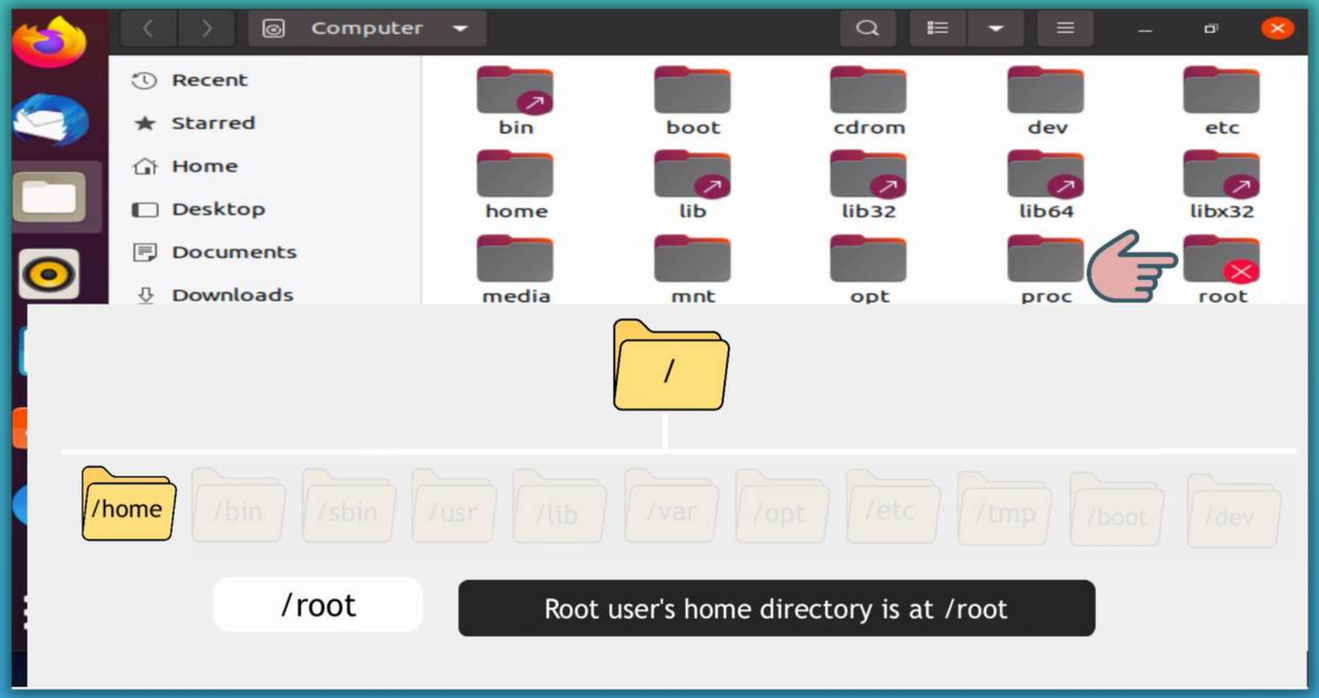
Linux File System - 2

EVERYTHING in Linux is a FILE

- **Everything in the system is represented by a file descriptor**
- Text documents, pictures etc
- Commands, like pwd, ls etc
- Devices like printer, keyboard, usb
- Even directories. Linux makes no difference between a file and a directory, since a directory is just a file containing names of other files

Linux File Structure - 1



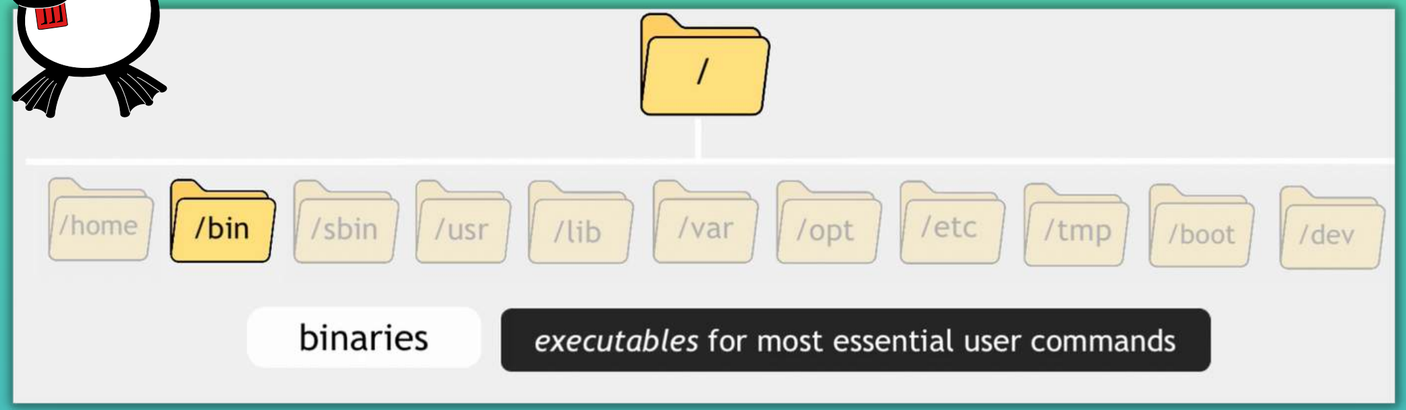


Users on OS

- Possible to have multiple user accounts on 1 computer
- **Each user has its own space**
- Each user can have own configurations

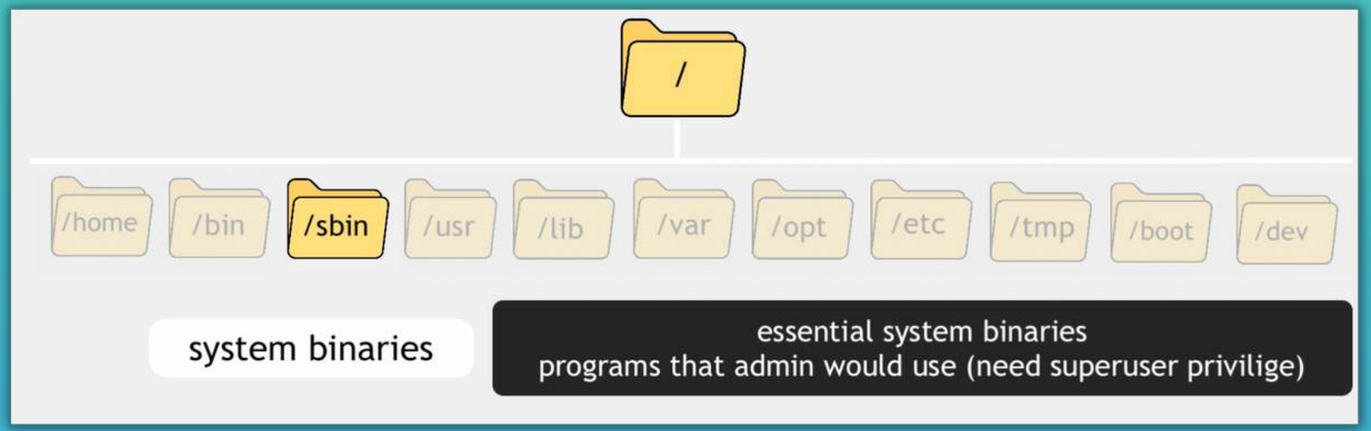
TECHWORLD WITH NANA

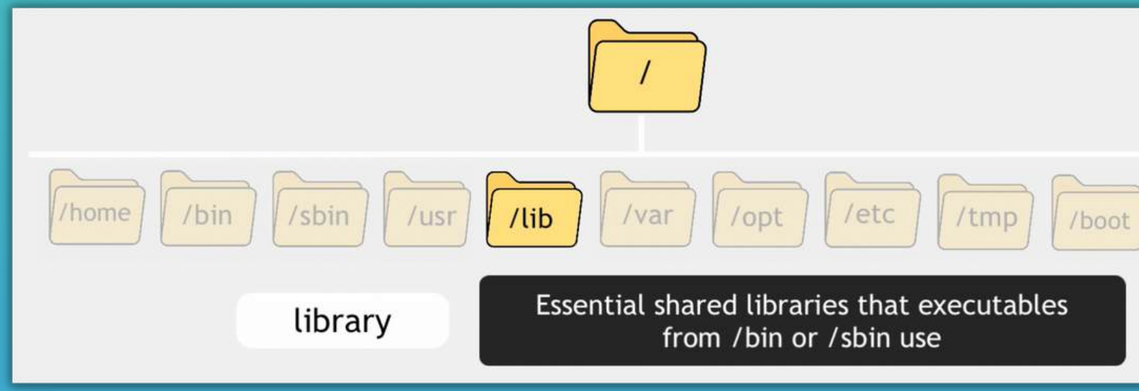
Linux File Structure - 2



Binary

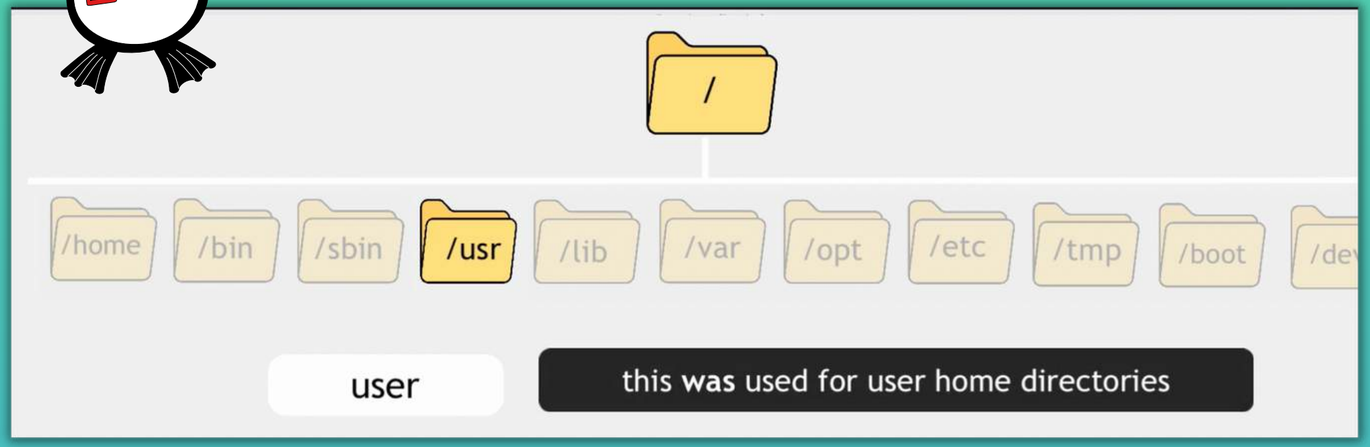
- A binary is a computer-readable format





TECHWORLD WITH NANA

Linux File Structure - 3



Why?

- Historic reasons
- Because of storage limitations it was split to root binary folders and user binary folders

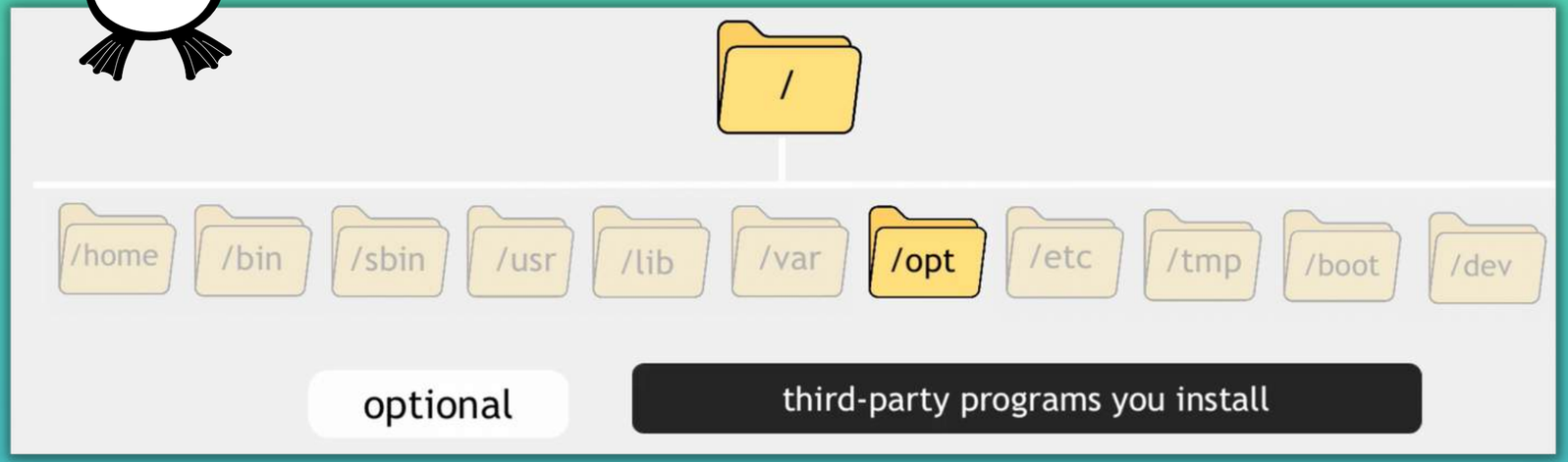
/usr/local

- **Programs that YOU install** on the computer
- Third-party applications like docker, minikube, java, ...
- Programs installed here, will be **available for all users on the computer**

- Inside /usr/local: App installation will be split again into different folders..

TECHWORLD WITH NANA

Linux File Structure - 4



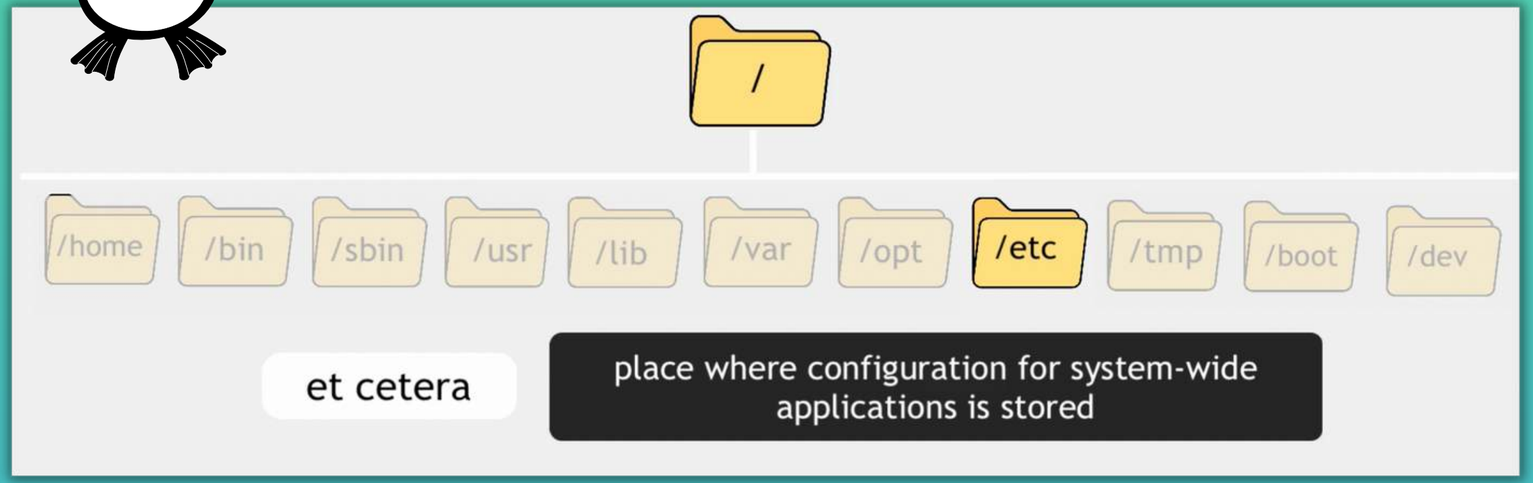
/usr/local

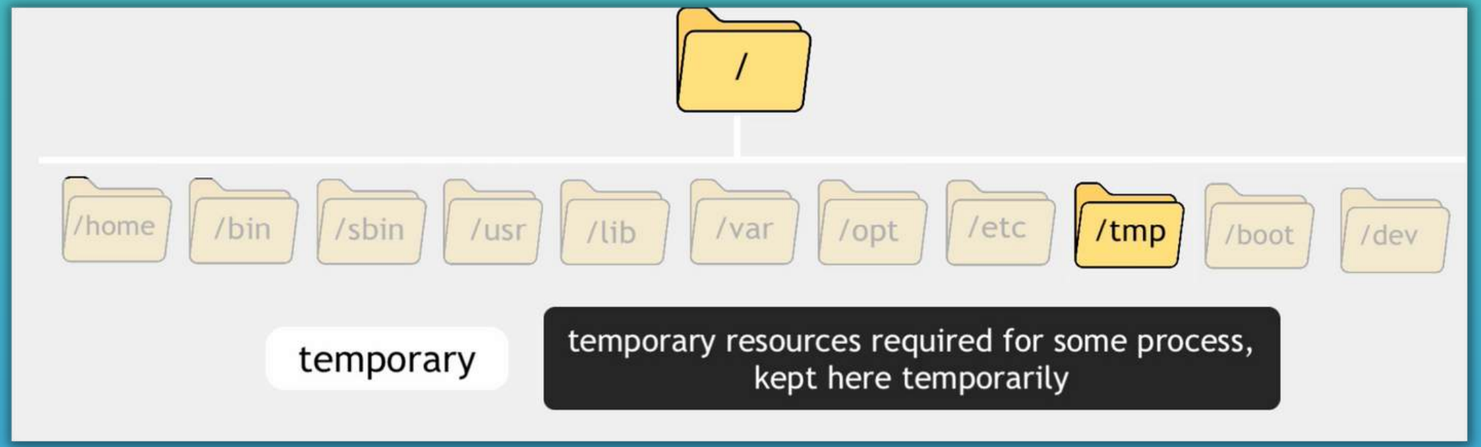
- Programs, which split its components

/opt

- Programs, which **NOT** split its components

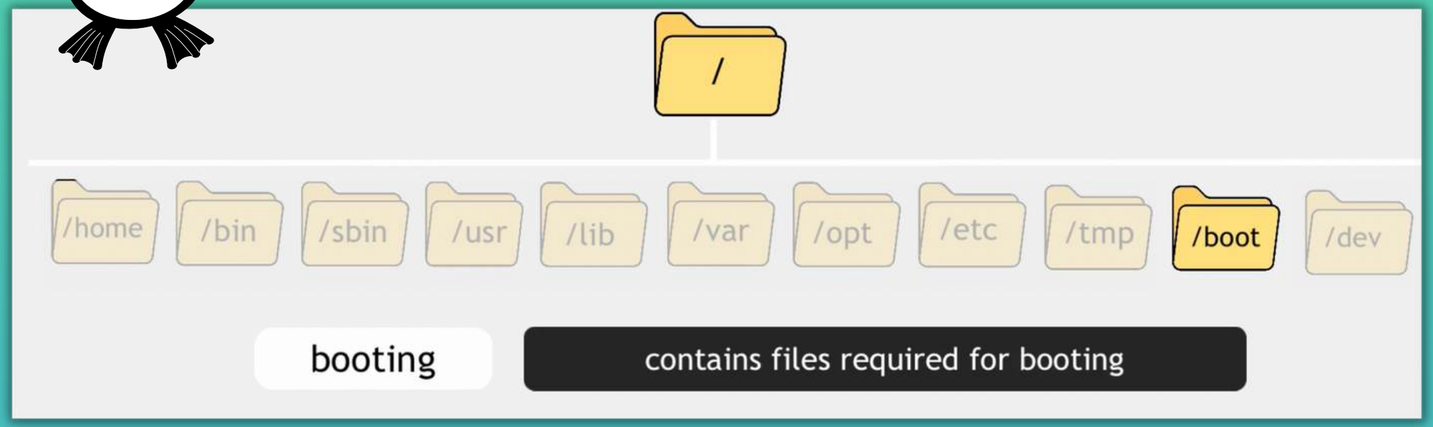
Linux File Structure - 5



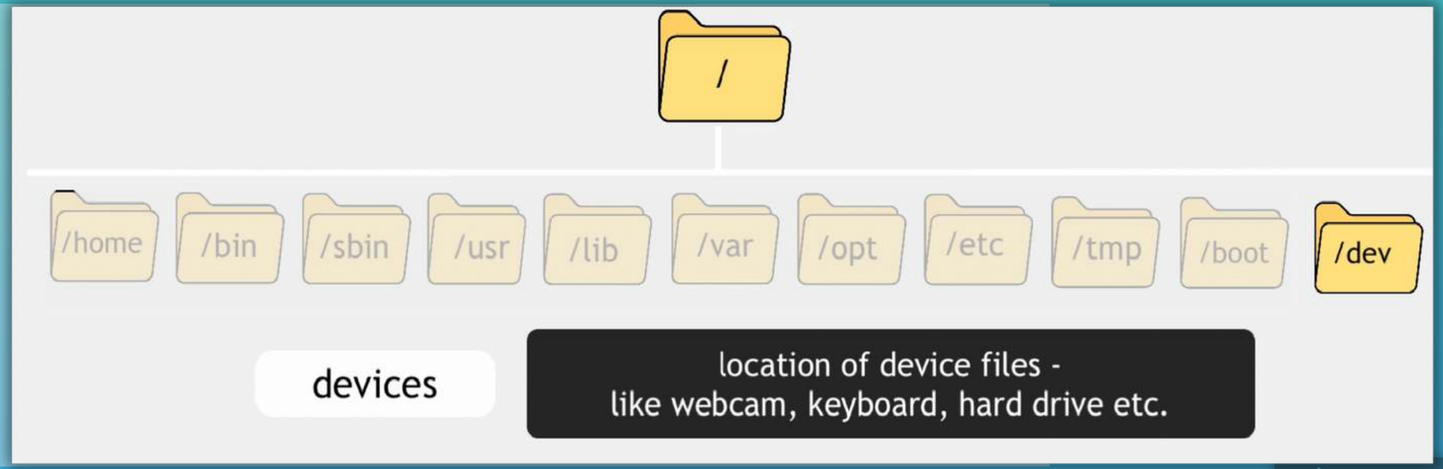


TECHWORLD WITH NANA

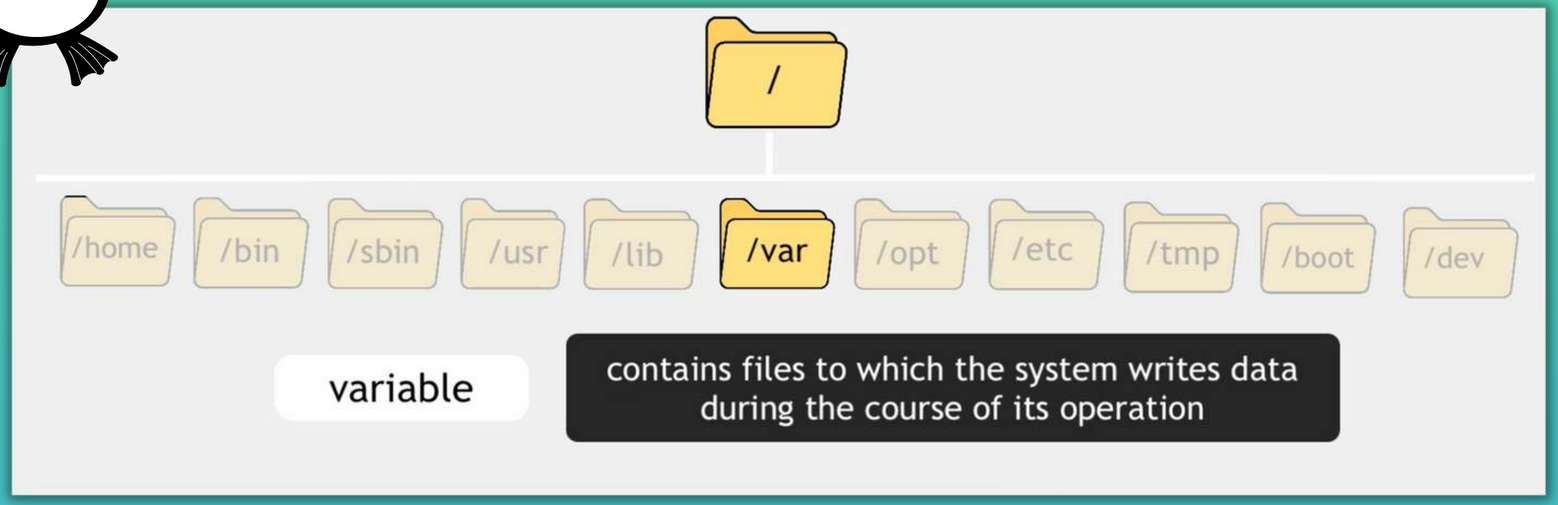
Linux File Structure - 6



- Apps and Drivers will access this, NOT the user
- Files that the system needs to interact with the device



Linux File Structure - 7



/var/log

- Contains log files

/var/cache

- Contains cached data from application programs

Linux File Structure - 8

/media

- Contains subdirectories, where **removable media devices** inserted into the computer are mounted
- E.g. when you insert a CD. A directory will automatically be created and you can access the contents of the CD inside the directory

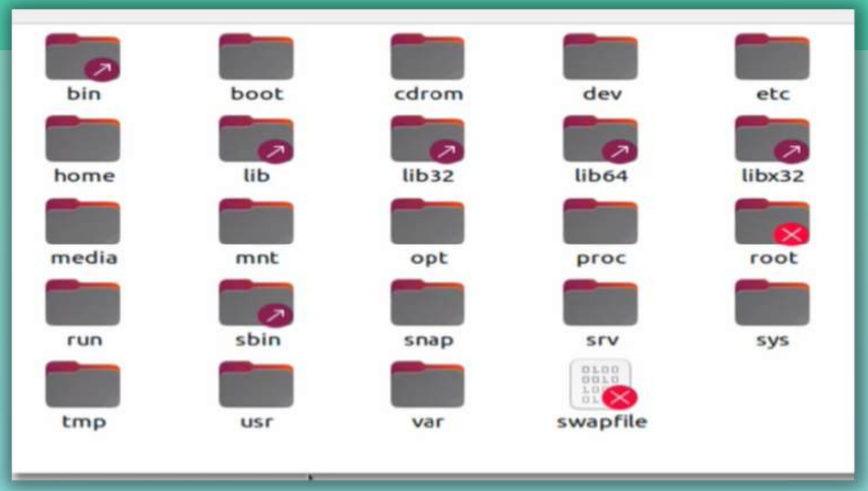
/mnt

- Temporary mount points
- Historically, system administrators mounted temporary file systems here

Linux File Structure - 9

Interactions with these root folders

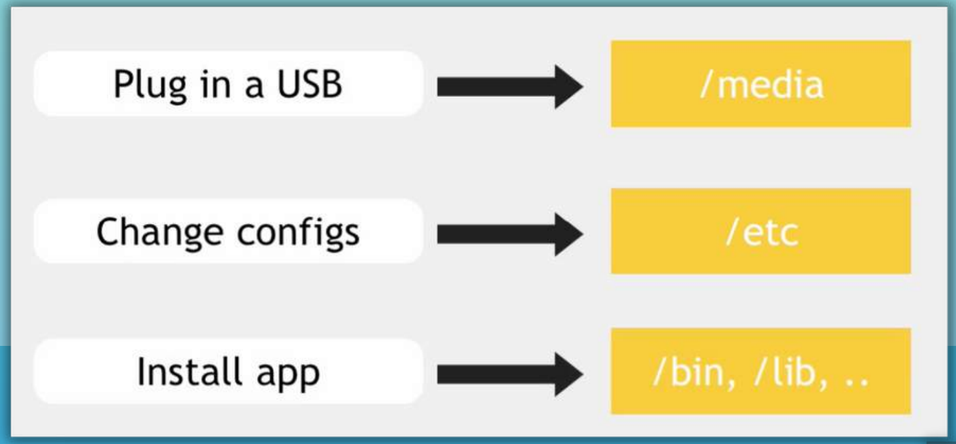
- Usually **you are not interacting with these root folders**



You install apps with package manager

Interacting with Operating System

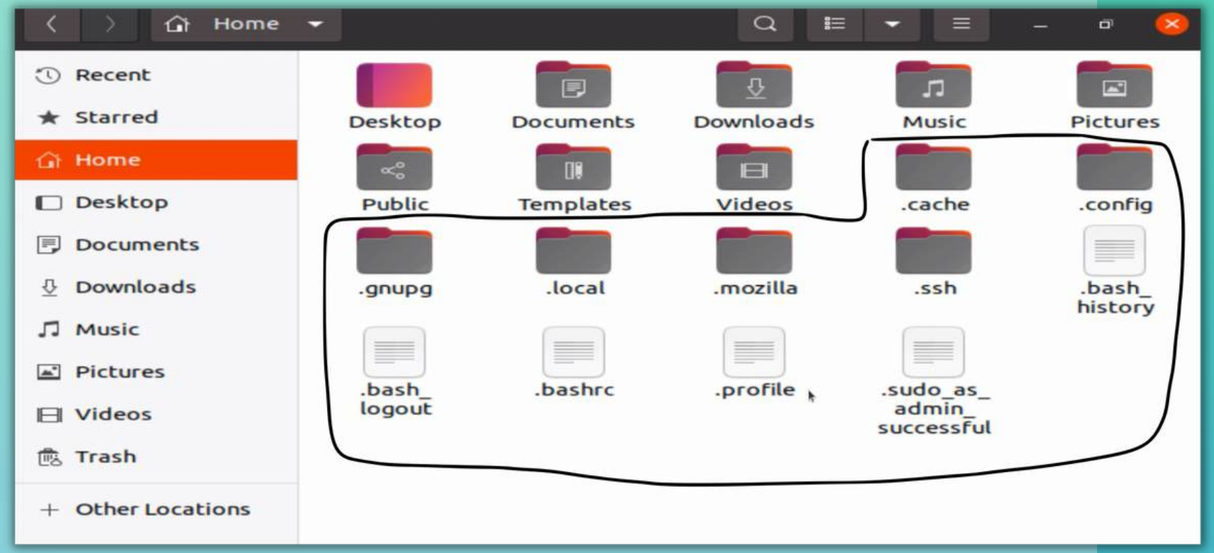
- Installer/Package Manager is handling this
- OS is handling this

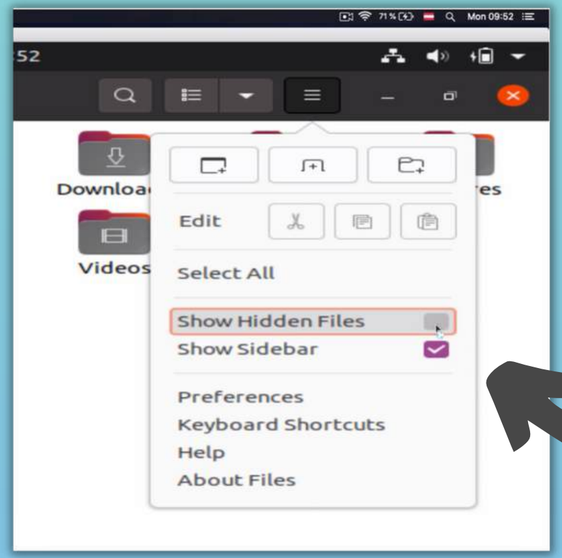


Linux File Structure - 10

Hidden Files

- Hidden files are primarily used to help prevent important data from being accidentally deleted
- Automatically generated by programs or OS
- File name **starts with a dot**
- In UNIX also called "**dotfiles**"



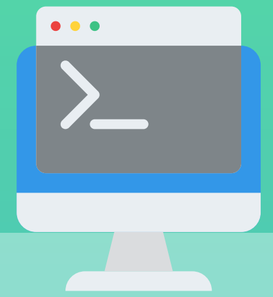


- By default: you can't see hidden files
- Select "Show Hidden Files" in settings

TECHWORLD WITH NANA

Introduction to Command Line Interface (CLI)

GUI vs CLI

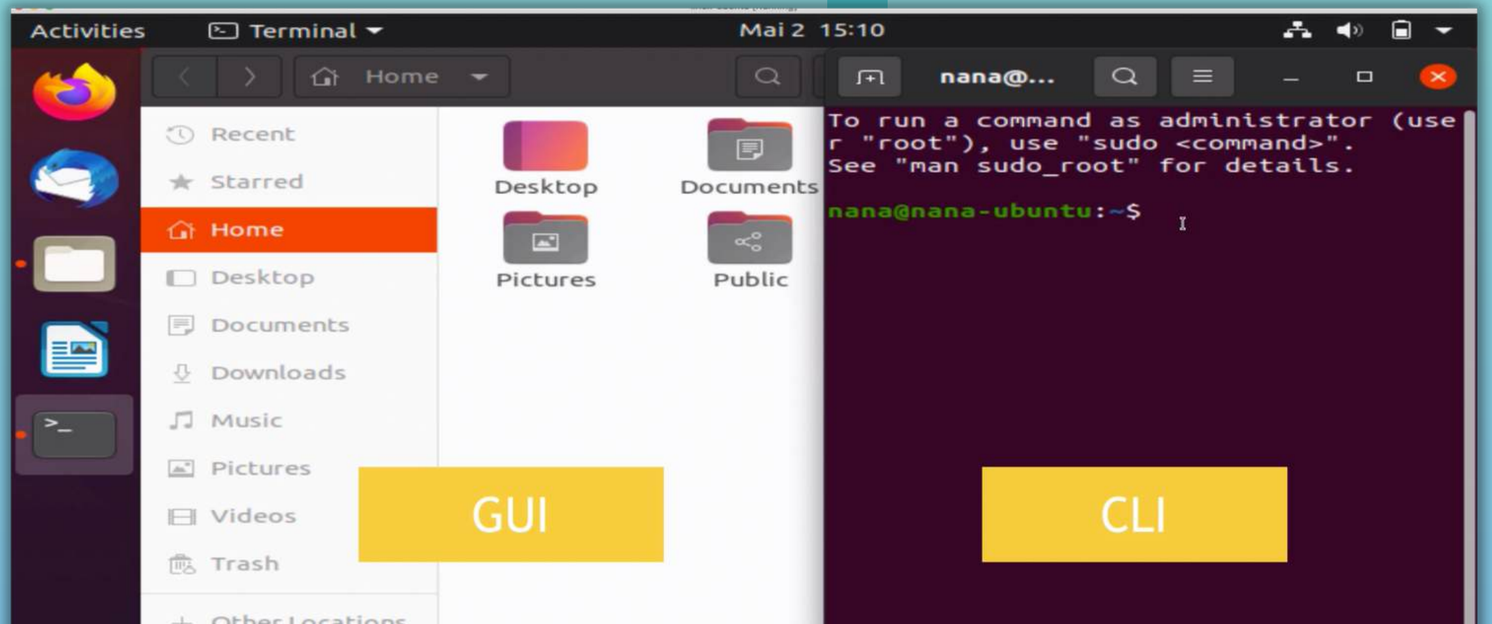


GUI

- GUI = A graphical user interface, where we have **graphical elements that you can interact with**, like buttons

CLI & Terminal

- **CLI** = Command Line Interface, where **users type in commands** and see the results printed on the screen
- **Terminal** = the **GUI window** that you see on the screen. It takes commands and shows output
- On servers you only have the CLI, no GUI

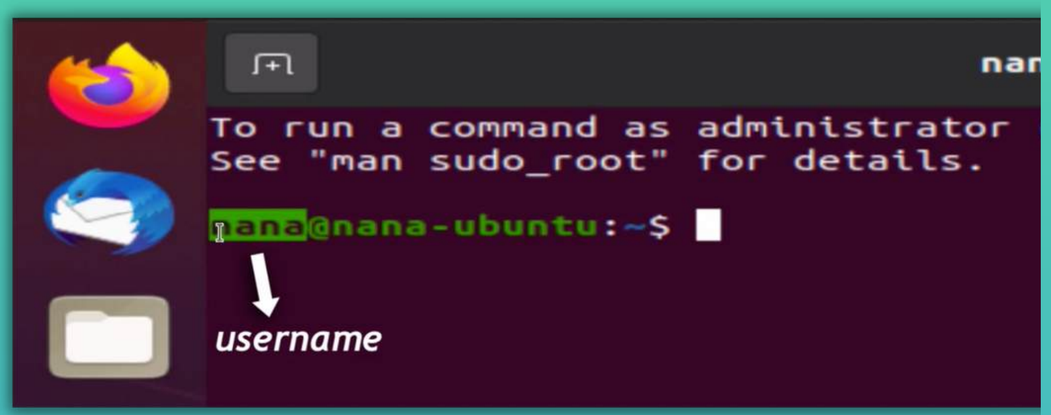


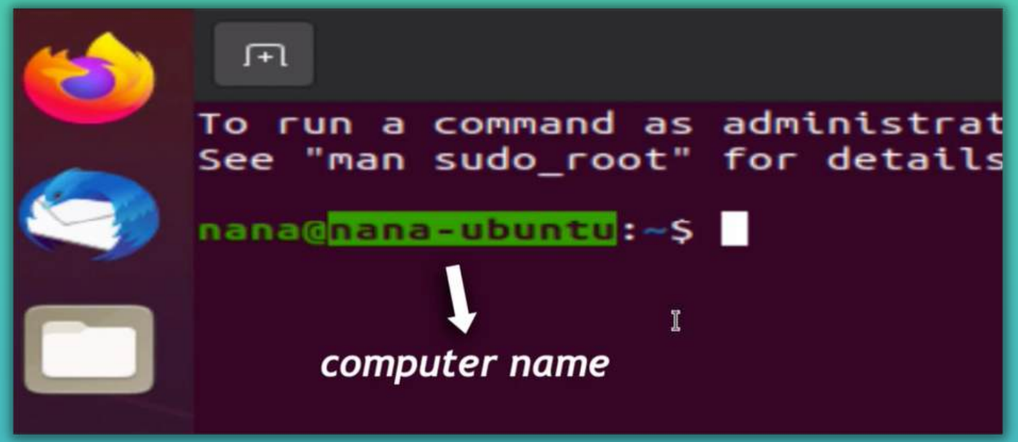
Why CLI over GUI?

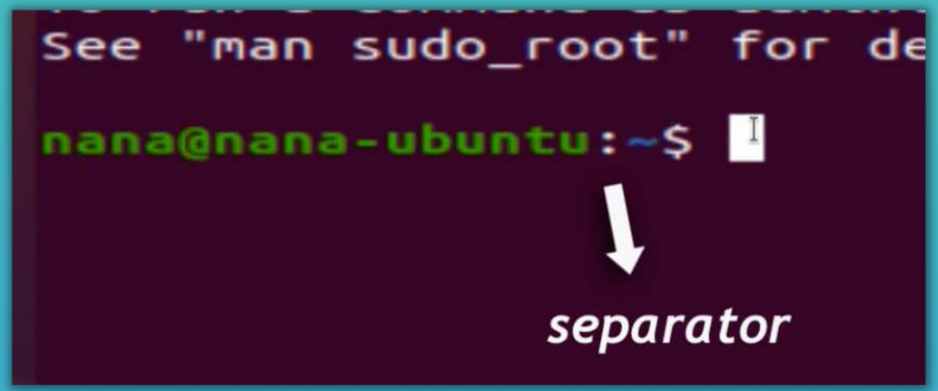
- Work more efficient
- Easier for bulk operations
- CLI is more powerful

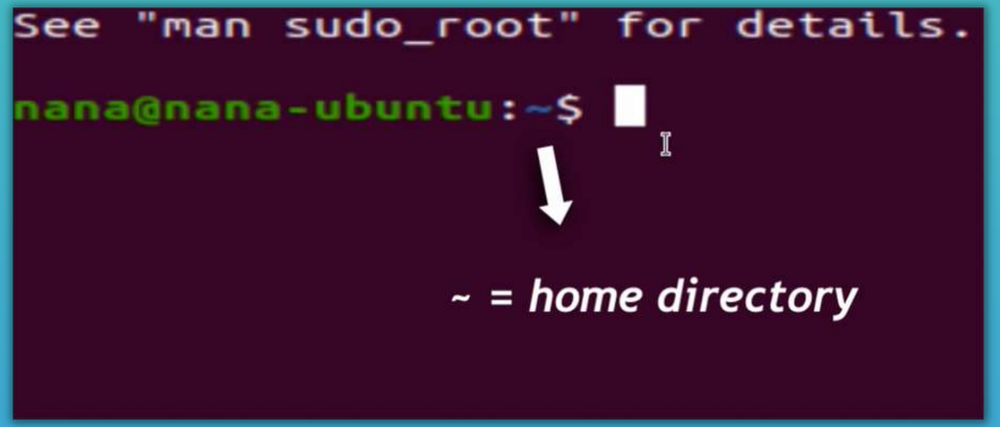
TECHWORLD WITH NANA

Introduction to CLI - 1



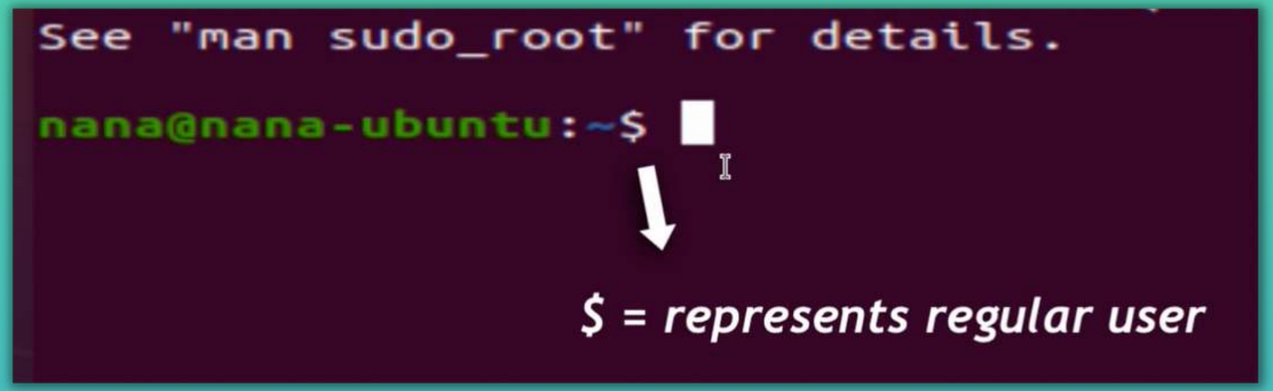






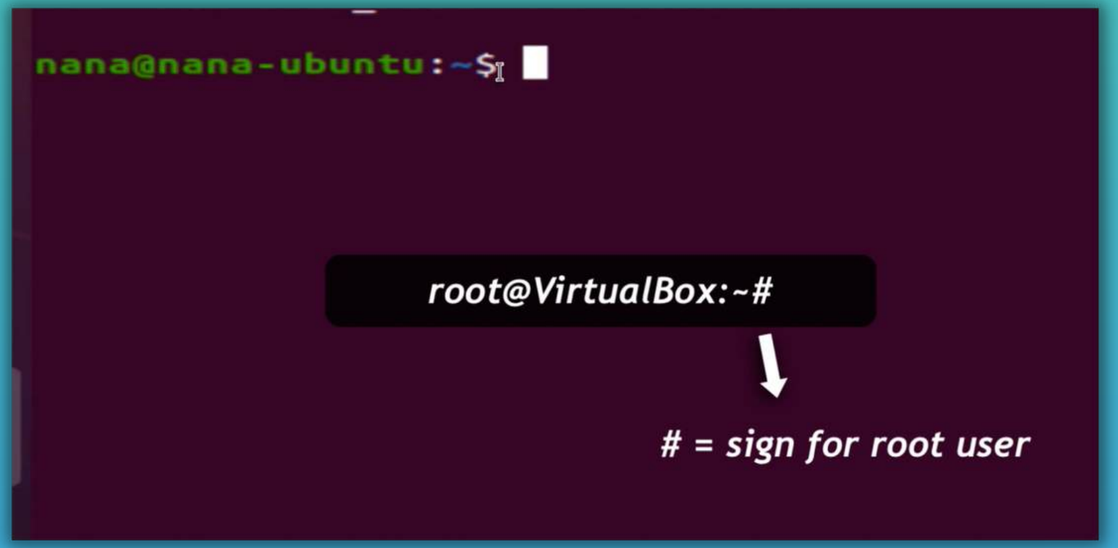
TECHWORLD WITH NANA

Introduction to CLI - 2



Regular user vs root

- You can work as regular user or as a root



TECHWORLD WITH NANA

Basic Linux Commands

```
pwd
ls
ls {dir-name}
cd {dir-name}
cd ..
cd ~
mkdir {new-dir-name}
touch {new-file-name}
rm {file-name}
rm -r {dir-name}
mv {file-name} {new-file-name}
cp {file-name} {new-file-name}
cp -r {dir-name} {new-dir-name}
```

```
ls -R {dir-name} - shows recursively, current dir and all its sub dir contents
ls -a {dir-name} - show all including hidden files
ls -l {dir-name} - detailed ouput, metadata of dir contents
```

```
history
history {number-of-lines}
clear

cat {file-name}
uname -a -> shows system kernel
```

sudo

- Allows regular users to run programs with the security privileges of the superuser or root

```
# commands with sudo
sudo adduser {new-user-name}
sudo addgroup {new-group-name}
```

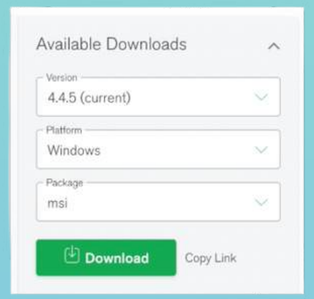
Introduction to Package Manager

TECHWORLD WITH NANA

How to install software

On Windows

- On Windows you have **download installer**
- Wizards that guide you through the installation



On Linux

- On Linux you will install applications mostly with **package manager tools** on CLI

TECHWORLD WITH NANA

Introduction to Package Manager - 1

What is a software package?

- A **compressed archive**, containing all required files for the software to run
- Apps usually have dependencies, which needs to be installed with it
- Files are split across different folders

Tasks of a Package Manager

- **Downloads, installs or updates existing software from a repository**
- Ensures the integrity and authenticity of the package
- Manages and resolves all required dependencies
- Knows where to put all the files in the Linux file system
- Easy upgrading of the software

Introduction to Package Manager - 2

Package Manager - pre-installed

- Package Manager is already **included in every Linux distribution**
- On Ubuntu, you have APT package manager available

APT = Advanced Package Tool

ubuntu

- Package Manager like apt has **commands you can use to install, uninstall or upgrade software**

Package Repository

- A repository is a **storage location**, where all the software packages are hosted
- Contains thousands of programs
- Package Manager fetches the packages from these repositories

Always update the package index before upgrading or installing new packages

"sudo apt update":

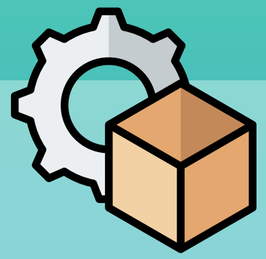
- Pulls the latest changes from the APT repositories
- The APT package index is basically a database
- Holding records of available packages from the repositories

Alternative Package Manager

APT and APT-GET

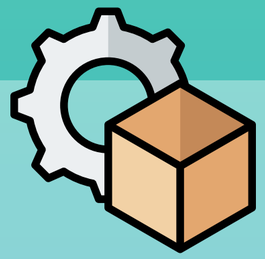
- There are different package manager available
- A very similar package manager is APT-GET, which you will often come across

APT



- Is **more user friendly**, like progress bar
- Fewer, but sufficient command options in a more organized way
- **Recommended by Linux distributions**!

APT-GET



- On Ubuntu, APT-GET also out of the box available
- Different set of commands
- You can achieve the same user friendly output, if you use additional command options
- E.g. "apt search" not available

Alternative ways to install software - 1

- On Linux you will mostly use apt package manager
- But generally, **you need to know different ways to install a software**

Why?

- When there are packages, that are **not available** in these official repositories
- Or package is available, but **not the latest version** (software packages are verified, before adding to repository and verification process takes time)

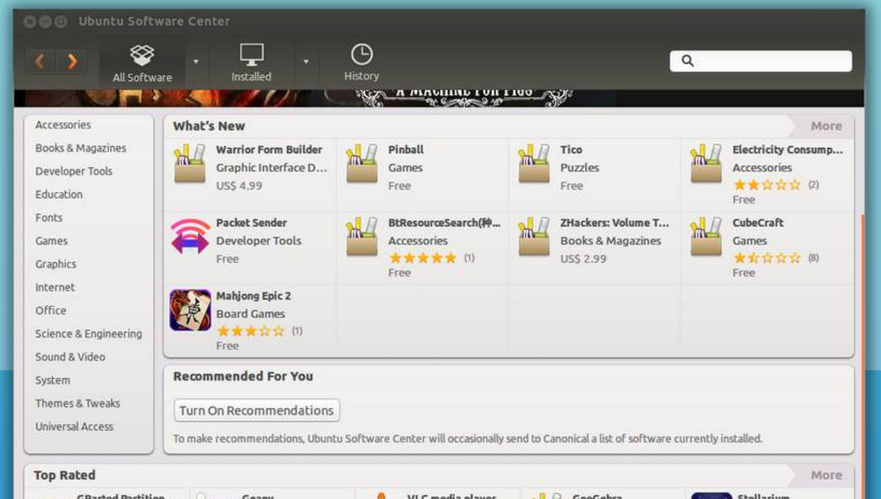
- Programs, which are not available: Browsers, Code editors etc.

Alternative ways to install software - 2

1) Alternative

Ubuntu Software Center

- Like an app store
- A utility for installing, purchasing, removing software in Ubuntu
- Has a graphical UI, so no need for using the CLI:



2) Alternative

Snap Package Manager

- Snap is a software packaging and deployment system
- A newer package manager, initial release in 2014
- Many still use the term "snappy", which it was called before
- A package manager for all OS, which use the Linux kernel

- **Snap** = A snap is a bundle of an app and its dependencies
- **Snap Store** = Provides a place to upload snaps, and for users to browse and install the software
- **Snapcraft** = Is the command and framework used to build and publish snaps

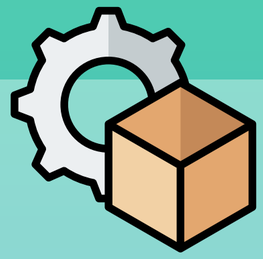
Alternative ways to install software - 3

Snap vs APT

Snap

- Self-contained - dependencies contained in the package
- Supports universal Linux packages (package type .snap)
- Automatic Updates
- Larger Installation size

APT



- Dependencies are shared
- Only for specific Linux distributions (package type .deb)
- Manual Updates
- Smaller Installation size

Alternative ways to install software - 4

3) Alternative

Add repository to official list of repos

Use Case:

- When installing relatively new applications, which are not yet in the official repositories

How to:

- Add the repository, which contains the package to official repositories list
- Often the command is listed in the installation guide of the tool:

- Command adds repository to APT sources (into **/etc/apt/sources.list**)
- Afterwards, you can install the package as usual - the next *apt install* will look into this new repository as well

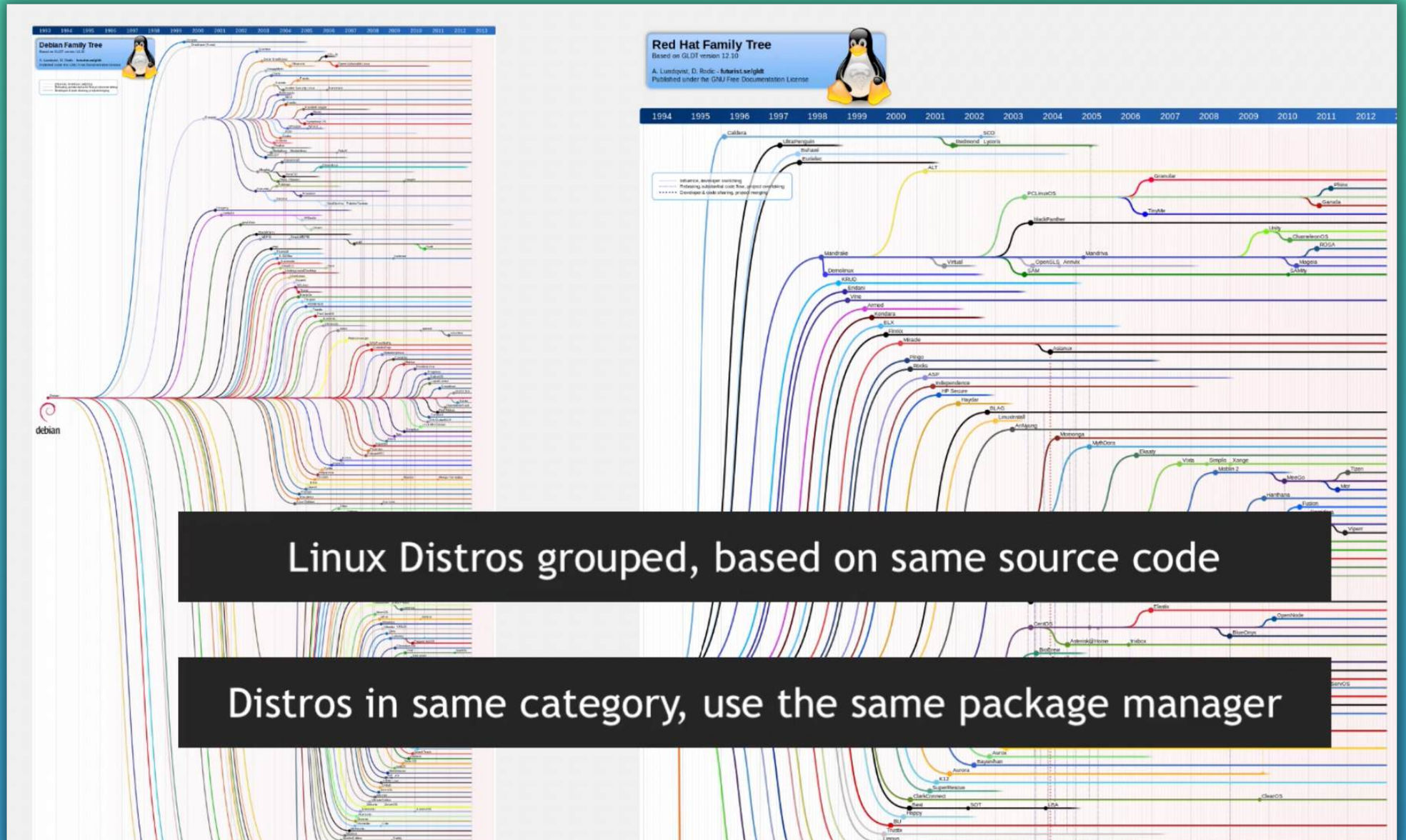
PPA = Personal Package Archive

- Some of these repositories are PPA
- PPAs are provided by the community
- **Private repository to distribute the software** - anybody can create it
- Usually used by developers to provide updates more quickly than in the official Ubuntu repositories
- **Be aware of possible risks** before adding a PPA
  - No guarantee of quality or security
  - Can cause difficulties e.g. when upgrading to a new Ubuntu release

Linux Distributions

TECHWORLD
WITH NANA

Linux Distributions - 1



TECHWORLD WITH NANA

Linux Distributions - 2

Debian based

- Ubuntu
- Debian
- Mint

APT

APT-GET

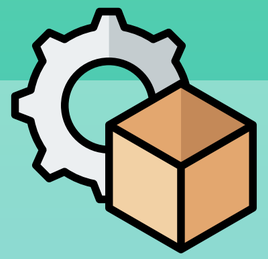
Red Hat based

- RHEL
- CentOS
- Fedora

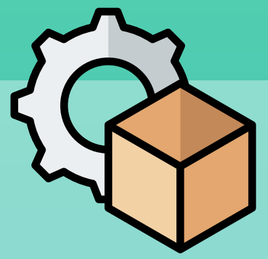
YUM

Linux Distributions - 3

APT



YUM



Similar concept:

- Package Manager uses official repositories
- Downloads packages, resolves dependencies etc

Difference of Repositories:

- More or newer versions of packages

Decision which Linux Distro to use

- Sys admins decide on Linux distributions they install based on package manager
- Within same category the difference is very small

TECHWORLD WITH NANA

Working with Vim

TECHWORLD
WITH NANA

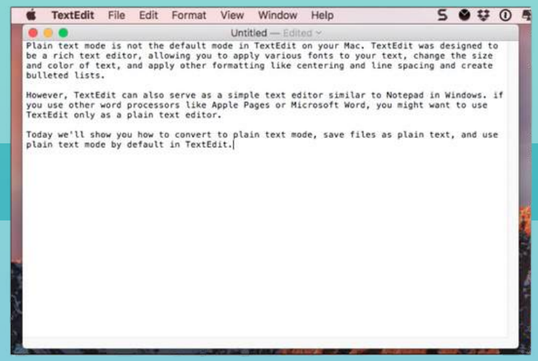
Introduction to Vi and Vim

What is it

- Vi and Vim are **built-in text editors** in Linux
- Vi is by far the most distributed and used text editor in Linux
- Depending on your Linux distro, Vim may or may not be installed

- Vim (Vi IMproved) is **improved version** of Vi
- Built to make creating and changing any kind of text very efficient

Why not just use the GUI Editors?



Use cases for using text editor in CLI

- Small modifications can be faster, especially when you are currently working in the CLI
- Faster to create and edit at the same time
- Supports multiple formats
- When working on a server

- Git CLI - writing the Git commit message
- Display Kubernetes configuration files
- Quickly editing one line or character in a file

Working with Vim - 1

Vim has 2 modes:

Insert Mode

Command Mode

```
vim {file-name} # if file doesn't exist, it will be create
```

Command Mode:

- This is the **default** mode
- You **can't edit the text**
- Whatever you type is interpreted as a command
- Navigate, Search, Delete, Undo etc.

Editing

```
i     #switch to insert mode
dd    #delete a line
d10   #delete 10 lines
u     #undo
0     #jump to start of line
$     #jump to end of line
A     #jump to end of line & switch to insert mode
12G   #go to line 12
```

Saving and Quitting

```
:wq #write file to disk and quit vim
:q! #quit vim without saving changes
```

Searching and Replacing

```
/pattern                    #will search the pattern
n                           #jump to next match
N                           #jump to previous match
:%s/my-string/my-new-string #will replace all occurences of "my-string" with "my-new-string"
```

Working with Vim - 2

Insert Mode:

- Allows you to **enter text**
- Pressing Escape, switches you back to command mode

Linux User Accounts & Groups

Linux User Accounts - 1

User account

- Each user account contains **2 unique identifiers**; username and UID
- When a user account is created, its username is mapped to a unique UID

Username

- Is used to access the user account
- Also known as **login name**
- Username is flexible
- Can be changed, but must be unique in system. Two users can't use the same username

UID = User Identifier

- A number assigned by Linux to each user on the system
- UID is fixed. It cannot be changed
- Once assigned, it always remains the same for that user account
- UID is **used to authenticate** and monitor the activity of user account
- So while username is used by the user, the **UID is used by the system**

Linux User Accounts - 2

There are **3 types** of user accounts:

Superuser account

- Built-in root user - **unrestricted permissions**
- For administrative tasks: You need to login as Root user or execute commands as root (sudo command)
- Has always a UID of 0

User account

- A **regular user** we create to login
- E.g. tom => /home/tom

Service account

- Also called **system accounts**
- Relevant for Linux Server Distros
- Each service will get its own user
- E.g. mysql user will start mysql application

Best Practice in Security:

- Don't run services with root user
- Instead create own user for it

→ Always **1 root user per computer**

→ But **multiple regular and service users** possible

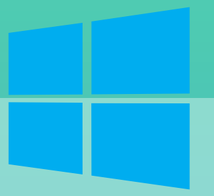
Linux User Accounts - 3

There are **2 ways to manage the user accounts**

Linux - **Standalone management**

- Unlike Windows, users and groups are **unique to the computer** but not unique across all computers
- User accounts are **managed on that specific hardware**
- This means that the UID on Computer 1 might be the exact same UID on Computer 2, even if it is not the same user.

Windows - **Centrally Managed User Accounts**



- Users and groups are **unique across all computers** in the system
- Login to any hardware that is connected to this system
- That's one of the reasons, why Windows is often preferred in companies or universities
- Employees can login to their account on every computer

TECHWORLD WITH NANA

Linux User Accounts - 4

- For Linux having multiple users is **important for administering servers**
- Instead of having a shared user, each user of the team should have a separate user account:

→ To **give permissions per team member** (junior - less permissions, senior - more permissions)

→ **Traceability** - who did what on the system?

Junior

Senior

How to manage permissions

- You can give permissions on user or group level

User Level

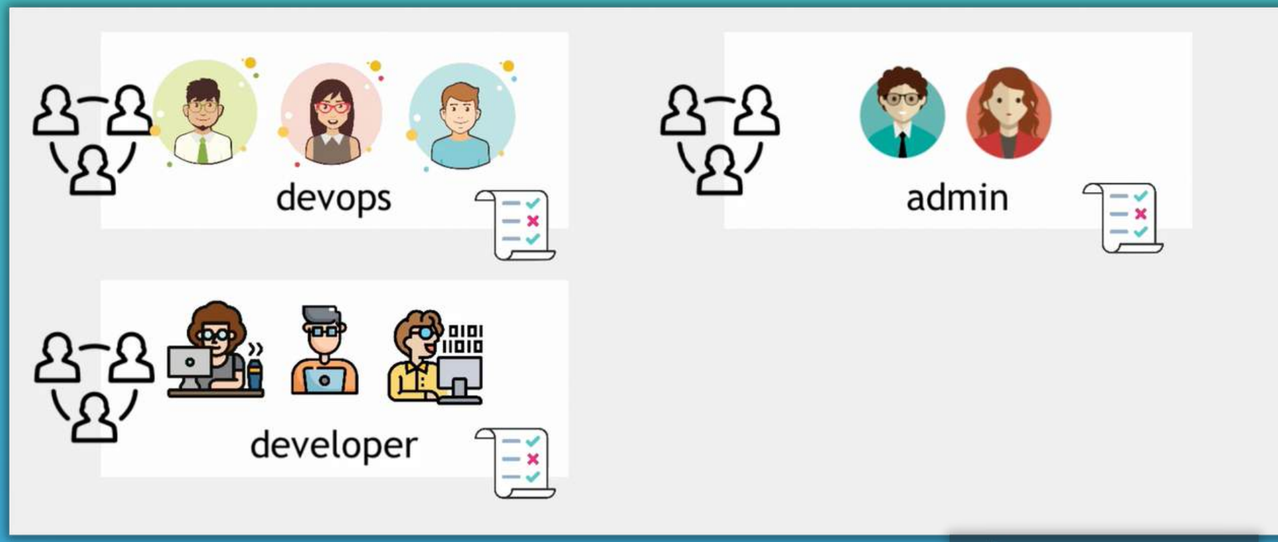
- Give permissions to Users directly

Group Level

- **Group Users into Linux Groups**
- Give permissions to the Group
- The way to go, if you manage multiple Users
- Built-in root user always has a GID (group ID) of 0

/etc/passwd file

- Contains a list of all user accounts
- Each user account is stored in an individual line
- Everyone can read it, but only root user can change the file



Basic User & Group Commands - 1

- There are different Linux commands to manage user accounts and groups

Create a new User

```
ana-ubuntu:~$ sudo adduser tom
```

adduser <username> = Create a new user

- Automatically chooses policy-conformant UID and GID values
- Automatically creates a home directory with skeletal configuration

Create a new Group

```
nana@nana-ubuntu:~$ sudo groupadd devops
```

groupadd <groupname> = Create new group

- By default, system assigns the next available GID from the range of group IDs specified in the *login.defs* file

Modify a User account

usermod [options] <username>

```
sudo usermod -g devops tom
```

```
sudo usermod -G admin tom
sudo usermod -aG newgroup tom
```

TECHWORLD WITH NANA

Basic User & Group Commands - 2

Note there are 2 available user & group commands:

adduser

addgroup

deluser

delgroup

- Interactive
- More user friendly, so **easier to use**
- It chooses conformant UID and GID values for you
- Creates a home directory with skeletal config automatically
- Or asks for input in interactive mode

user**add**

group**add**

user**del**

group**del**

- **Low-level utilities**
- You need to provide the infos yourself

File Ownership & Permissions

TECHWORLD WITH NANA

File Ownership & Permissions - 1

- User permissions are related to reading, writing and executing files
- Each file and folder has 2 different owners (user + group)

3 types of rights

- Read
- Write
- Execute

3 categories of access

- **Owner** = user who owns the file/folder
- **Group** = group that owns the file/folder
- **Others** = People who aren't part of the owning group or is not the owner

Change ownership

- chown <username>:<groupname> <filename>

→ By default, owner is the user, who created the file

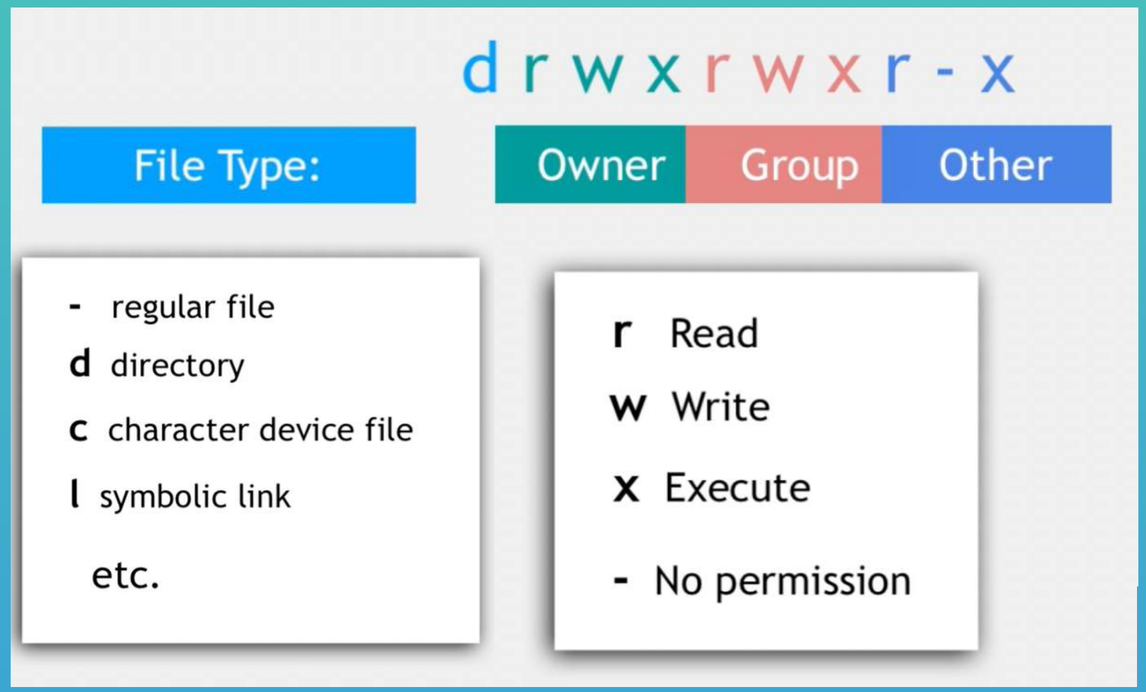
→ Owning group is the primary group of that user

TECHWORLD WITH NANA

File Ownership & Permissions - 2

- **List file permissions**, by printing files in a long listing format (ls -l):

```
drwxrwxr-x 2 nana   nana    4096 Mai   7 11:00 api
-rw-rw-r-- 1 nana   nana     320 Mai   3 14:57 config.yaml
-rw-rw-r-- 1 nana   nana      68 Mai   3 14:27 Readme.md
-rw-rw-r-- 1 admin devops      0 Mai   7 13:06 test.txt
```



File Ownership & Permissions - 3

- **3 ways** to set or change permissions:

1) Symbolic Mode

+ add

- remove

r Read
w Write
x Execute
- No Permission

2) Set permission

= set the permission and override the permissions set earlier

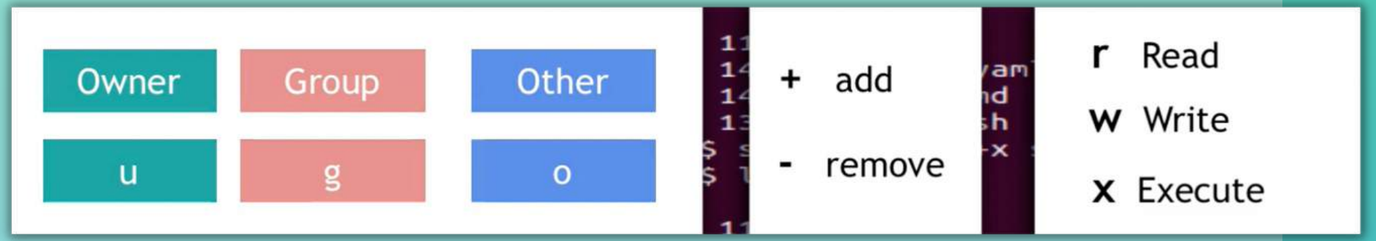
3) Numeric Mode

4 Read
2 Write
1 Execute
0 No Permission

File Ownership & Permissions - 4

1st way: Adding or removing permissions

- Example: *sudo chmod g-w config.yaml*
- Means: Remove write permission of group owner from config.yaml file



2nd way: Set the whole permission

- Example: *sudo chmod g=rwx config.yaml* *sudo chmod o=r-x config.yaml*
- Means: Set read, write and execute permission for group for config.yaml file

  Set read and execute permission for others for config.yaml file

File Ownership & Permissions - 5

3rd way: Set permission with numeric values

- Example: *sudo chmod 740 script.sh*
- Means: Sets full permissions for the owner, read-only for the group, and nothing for the other users.

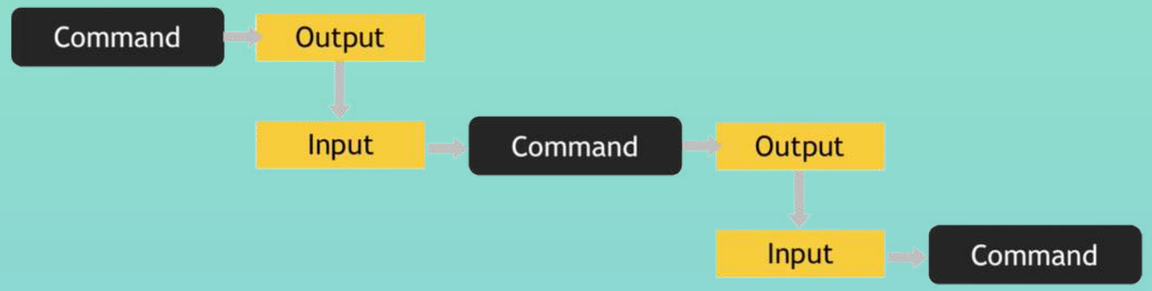
Absolute(Numeric) Mode

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write + Execute	rwx

Basic Linux Commands - Pipes & Redirects

Basic Linux Commands - 1

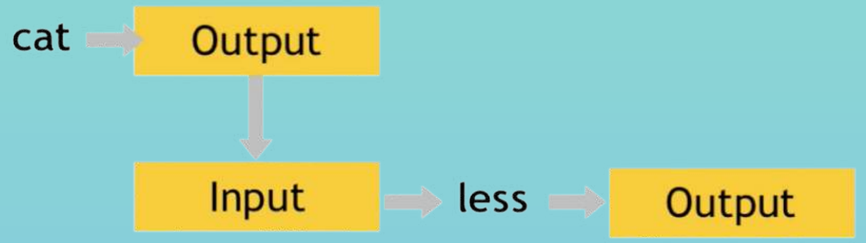
- Every program has: **Input** and **Output**
- The output from one program can become the input of another command



"pipe" command: |

- Pipes the output of the previous command as an input to the next command

cat /var/log/syslog | less



"less"

- Displays the contents of a file or a command output, one page at a time
- Allows you to navigate forward and backward through the file
- Mostly **used for opening large files**, as *less* doesn't read the entire file, which results in faster load times

Basic Linux Commands - 2

"grep"

- **Searches for a particular pattern** of characters and displays all lines that contain that pattern
- Example usage is to use the *grep* command in combination with pipe

history | grep sudo

```
# less
cat /var/log/syslog | less
ls /usr/bin | less
history | less

# grep
history | grep sudo
history | grep sudo chmod
ls /usr/bin | grep java
cat {file-name} | grep {string}

# chained
history | grep sudo | less
```

"redirect" operator: >

- Takes the output from the previous command and sends it to a file that you give

```
# set file contents
history | grep sudo > sudo-commands.txt
cat {existing-file} > {new-file}
# append to file contents
history | grep rm >> sudo-commands.tx
```

Standard Input and Output

- Every program has **3 built-in streams**:

STDIN (0) = Standard Input

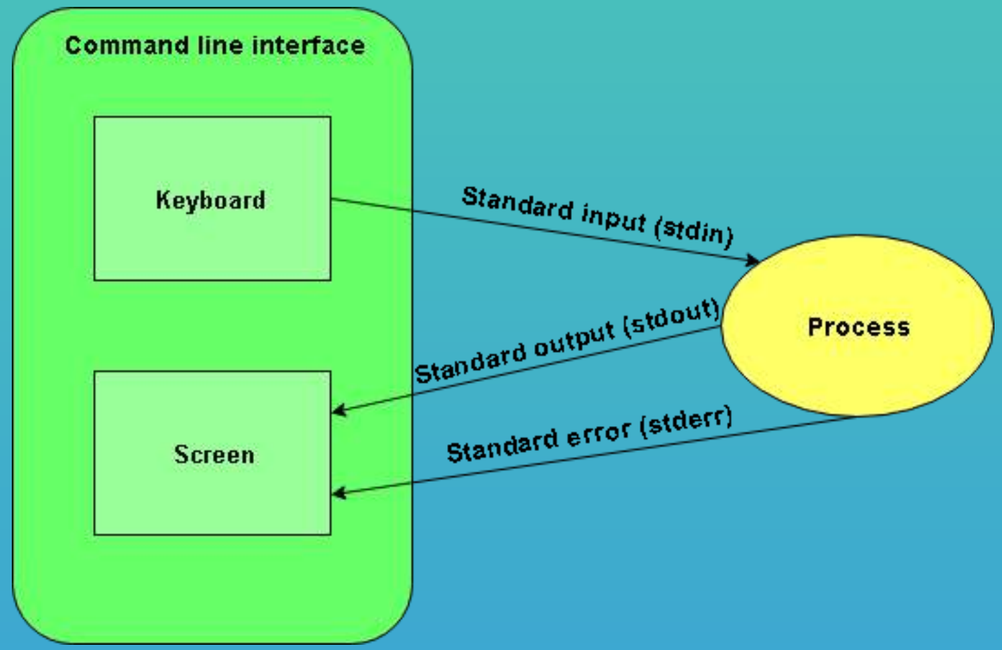
STDOUT (1) = Standard Output

STDERR (2) = Standard Error

```
nana@nana-ubuntu:~$ ls -l doesntexist
ls: cannot access 'doesntexist': No such file or directory
nana@nana-ubuntu:~$
```

STDERR (2) = Standard Error

- Standard streams are streams of data that travel from where a program was executed, to the places where the program is processed and then back again



Introduction to Shell Scripting

TECHWORLD WITH NANA

Why Shell Scripting?

- When administering servers, you need to **execute tons of commands**

Configured a server

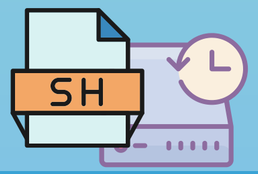
- **useradd** tim
- **groupadd** devops
- **mkdir** project
- **touch** file.txt
- **chmod** 750 /path
- **sudo apt** docker
- docker run ...

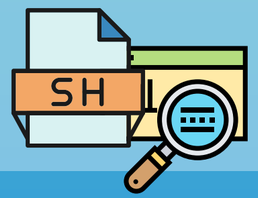
- Instead of executing the command one by one, you can **save the commands in a file and execute that file**

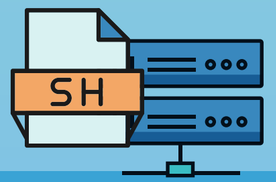
- Avoid repetitive work
- Keep history of configuration
- Share the instructions
- Logic & Bulk Operations

- Such file is called a **shell script**
- **File extension is .sh**
- Automate all your work:

→ Backup Scripts → Monitoring Scripts → Server Configuration Scripts







How to execute Shell Scripts - 1

Shell

- The **program that interprets and executes the various commands** that we type in the terminal
- **Translates** our command that the OS Kernel can understand

Different Shell implementations

sh

- Bourne **Sh**ell - /bin/sh

Bash

- **B**ourne **a**gain **sh**ell - /bin/bash
- Improved version of sh
- **Default shell program** for most UNIX like systems
- Bash is a shell program and a programming language

- Shell and Bash terms are often used interchangeable
- All shell script files have the same .sh file extension

How to execute Shell Scripts - 2

How OS knows which shell to use

SH	BASH	ZSH

We need to tell the OS!

#!/bin/sh	*#!/bin/bash*	*#!/bin/zsh*

- This is called "**shebang**"

Why?

- Because of the first 2 characters "#!"
- **#** = in musical notation, also called "sharp"
- **!** = also called "bang"
- Shebang became a shortening of sharp-bang

Make shell script executable

- To execute a shell script, you need to make it executable first, by **adding executing permission**

```
nana@nana-VirtualBox:~$ vim setup.sh
nana@nana-VirtualBox:~$ ./setup.sh
bash: ./setup.sh: Permission denied
nana@nana-VirtualBox:~$ ls -l setup.sh
-rw-rw-r-- 1 nana nana 47 Mai 12 13:56 setup.sh
nana@nana-VirtualBox:~$ sudo u+x setup.sh
sudo: u+x: command not found
nana@nana-VirtualBox:~$ sudo chmod u+x setup.sh
nana@nana-VirtualBox:~$ ls -l setup.sh
-rwxrw-r-- 1 nana nana 47 Mai 12 13:56 setup.sh
nana@nana-VirtualBox:~$
```

Writing Shell Scripts - 1

Variables

- Used to **store data** and can be referenced later
- Similar to variables in general programming languages
- Store output of a command in a variable:

```
variable_name=$(command)
```

Conditionals

- Allow you to alter the control flow of the program
- E.g. execute a command only when a certain condition is true

```
if [  condition ]
then
  statement
else
  statement
fi
```

[...] builtin command

- Square brackets enclose expressions
- It's a shorthand notation for the "test" command
  - if test -d "config" is the same

Boolean

- A datatype that can only have 2 values:

True False

- Bash does have Boolean expressions in terms of comparison and conditions

Writing Shell Scripts - 2

File Test Operators

- Test various properties associated with a file

	becomes true.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.
-r file	Checks if file is readable; if yes, then the condition becomes true.
-w file	Checks if file is writable; if yes, then the condition becomes true.
-x file	Checks if file is executable; if yes, then the condition becomes true.
-s file	Checks if file has size greater than 0; if yes, then

Relational Operators

- Works only for numeric values

-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.

String Operators

- Test strings for equality and more

=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.
str	Checks if **str** is not the empty string; if it is empty, then it returns false.

TECHWORLD WITH NANA

Writing Shell Scripts - 3

Passing arguments to a script

- You can pass arguments when executing a script
- We can read these passed parameters using special variables

Positional Parameters

- Arguments passed to script are processed in the same order in which they're sent
- The indexing of arguments starts at 1: `$1`

`$*` = represent all the arguments as a single string

`$#` = Total number of arguments provided

Writing Shell Scripts - 4

Loops

- Enables you to **execute a set of commands repeatedly**
- There are different types of loops:
  - while loop
  - for loop
  - until loop
  - select loop

For Loop

- Operates on lists of items
- Repeats a set of commands for every item in the list

```
for var in word1 word2 ...
do
  statement(s) to be executed for every word
done
```

While Loop

- Executes a set of commands repeatedly until some condition is matched


```
while condition
do
  statement(s) to be executed if command is true
done
```

TECHWORLD WITH NANA

Writing Shell Scripts - 5

Functions

- Enables you to break down the overall functionality of a script into **smaller, logical code blocks**

```
function_name () {
   list of commands
}
```

- This code block can then be referenced ("called") anywhere in the script multiple times
- You can **accept parameters in a function**
- The parameters passed in are represented by $1, $2, etc.

Best Practices

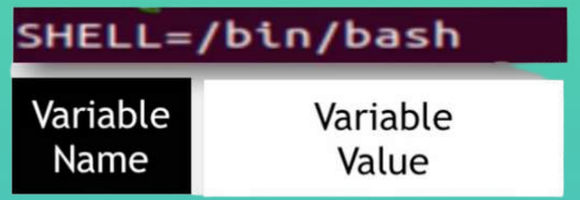
- **Don't use too many parameters**
- Apply the Single Responsibility Principle: A **function should only do one thing**
- Declare variables with a **meaningful name** for positional parameters within a function

Environment Variables

TECHWORLD WITH NANA

Environment Variables - 1

- Environment variables are **KEY=value pairs**
- These variables are **available system-wide**
- Variables store information, which can be changed
- By convention, names are defined in UPPER CASE
- Allow you to customize how the system works and the behavior of the applications on the system

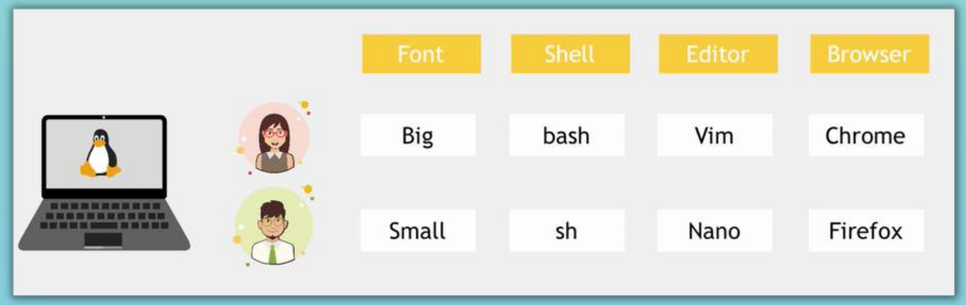


Environment Variables - 2

There are **2 use cases** for environment variables

1) OS stores information about the environment

- When you have multiple users on a computer, each user can configure its own environment/account by setting preferences



- OS stores all these configurations in environment variables

- Some examples:

```
SHELL=/bin/bash
HOME=/home/nana
USER=nana
```

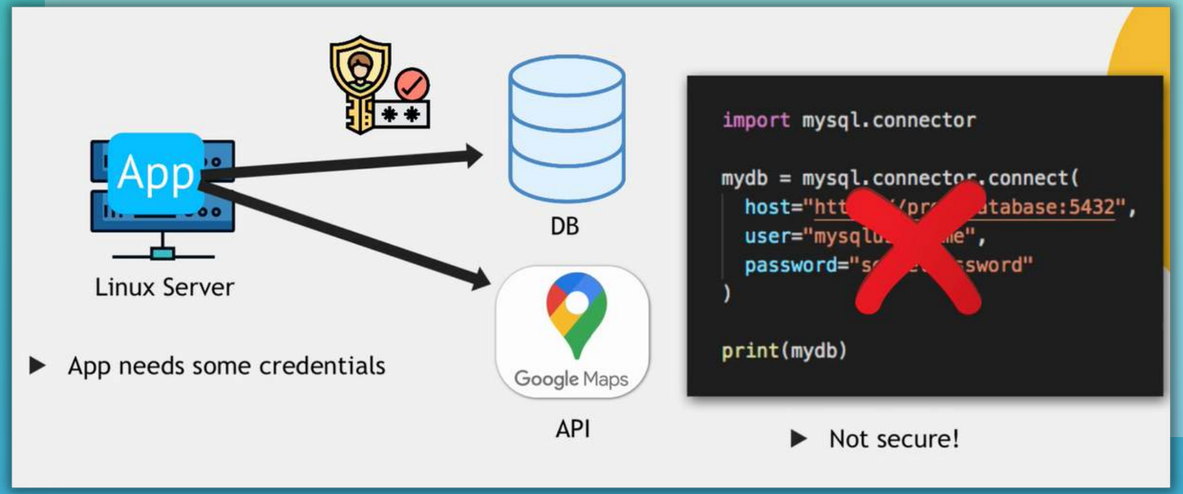
1. Default shell program of the user
2. Current user's home directory
3. Currently logged in user

Environment Variables - 3

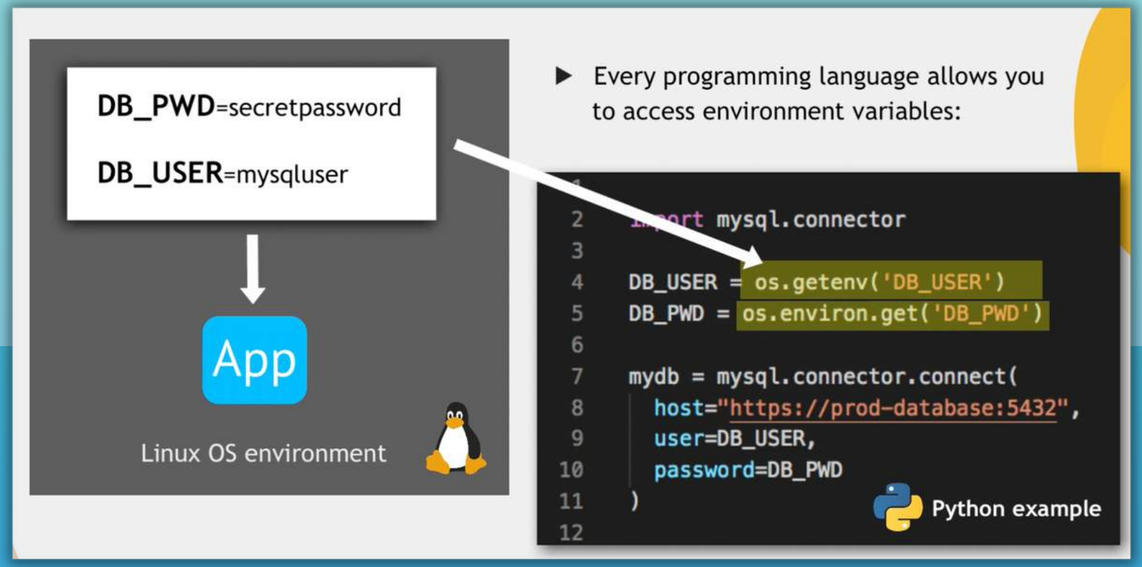
2) Create our own environment variables

Use Case: Sensitive data for application

- If application needs some credentials, **it's not secure to hardcode it directly in the source code**



- Set these data as env vars on server
- By creating these env vars, we make them available in the environment
- All apps and processes can now access these env vars

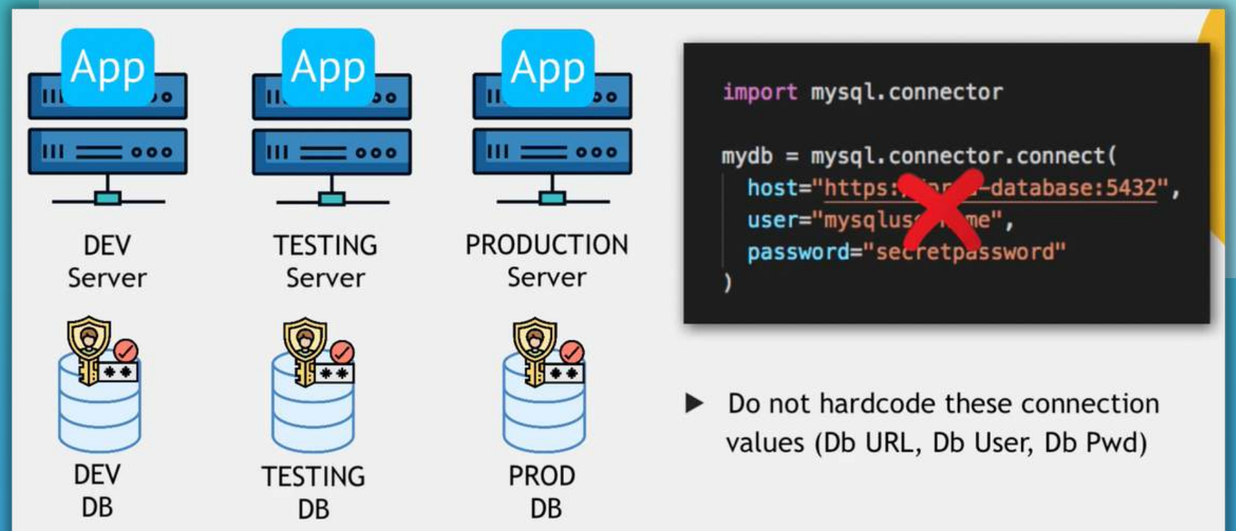


Environment Variables - 4

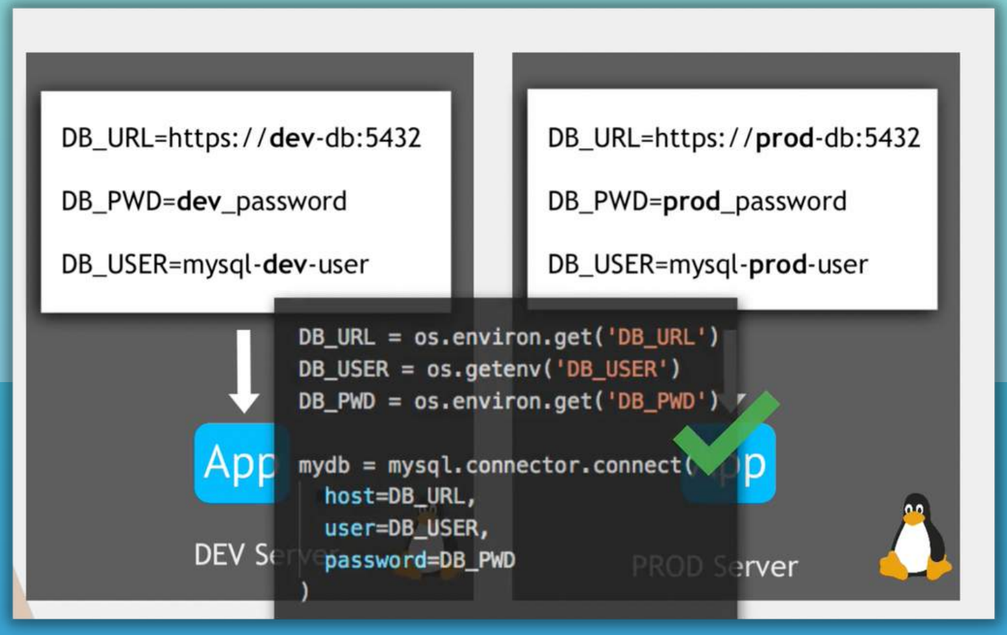
2) Create our own environment variables

Use Case: Make application more flexible

- If your application connects to a database, you need **different credentials for different dev, test and prod environments**
- So you should not hardcode these values in the source code



- Instead one option is to set the different values as env vars on each server
- Now no need to change the application code, as the values are dynamically set



Environment Variables - 5

Set environment variables

- The *export* command is used to set environment variables

```
export MY_VAR="myvalue"
```

- Env vars created in this way are **available only in the current session**. If you open a new shell or if you log out all variables will be lost!

Display environment variables

- Print environment variable in terminal

```
printenv MY_VAR
```

Persisting environment variables

- To make env vars persistent you need to define those variables in a shell configuration files
- These are shell specific configuration files
- E.g. if you are using bash, you can declare the variables in the ~/.bashrc file
- Variables set in this file are loaded whenever a bash login shell is entered

```
source ~/.bashrc
```

- To load the new env vars into the current shell session

- **Shell config files are user specific**
- Set **system-wide env vars in */etc/environment* file**

Environment Variables - 6

PATH

- *PATH* is a environment variable
- Its value includes a list of directories to executable files, separated by :
- Tells the shell **which directories to search for the executable** in response to our executed command
- When you run a command, the system will search those directories in this order and use the first found executable
- Example value:

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Networking

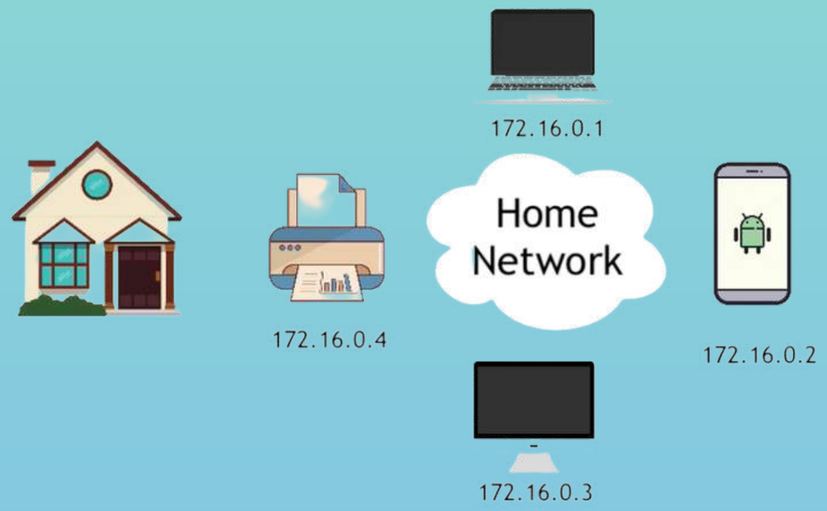
TECHWORLD
WITH NANA

Introduction to Networking - 1

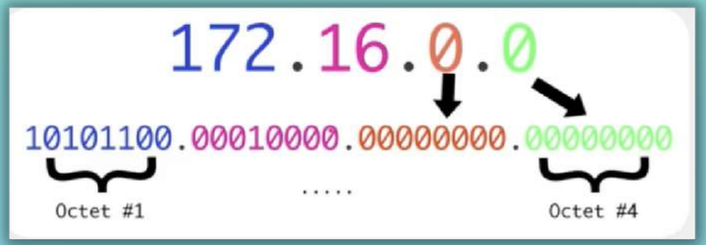
- Networks can be as small as 2 computers connected at your home and as large as in a large company or connected systems worldwide known as Internet
- Linux has a very strong set of networking instruments to provide and manage routing, bridging, virtual networks etc

LAN - Local Area Network

- LAN is a **collection of devices** connected together in one physical location
- **Each device has a unique IP address**, with which the devices communicate with each other



IP - Internet Protocol

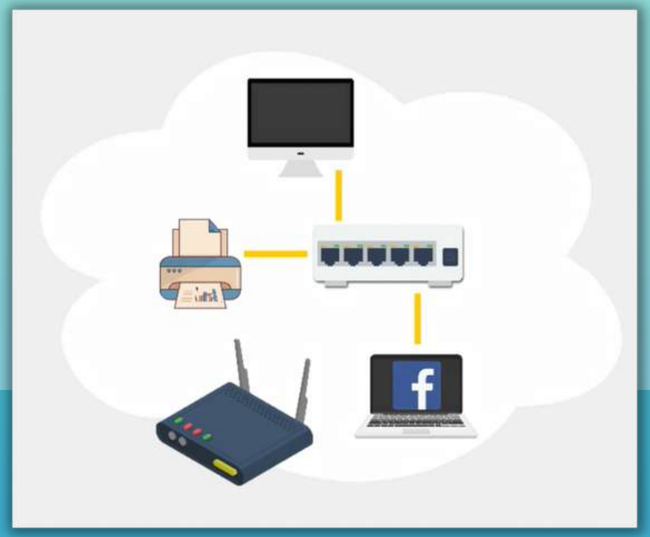


- 32 bit value
- 1 bit = 1 or 0
- 00000000 = **0**
- 11111111 = **255**
- IP addresses can range from **0.0.0.0** to **255.255.255.255**

Introduction to Networking - 2

Switch

- Sits within the LAN
- Faciliates the connection between all the devices within the LAN



Router

- Sits between LAN and outside networks (WAN)
- WAN = Wide Area network
- Connects devices on LAN and WAN
- **Allows networked devices to access the internet**

Gateway

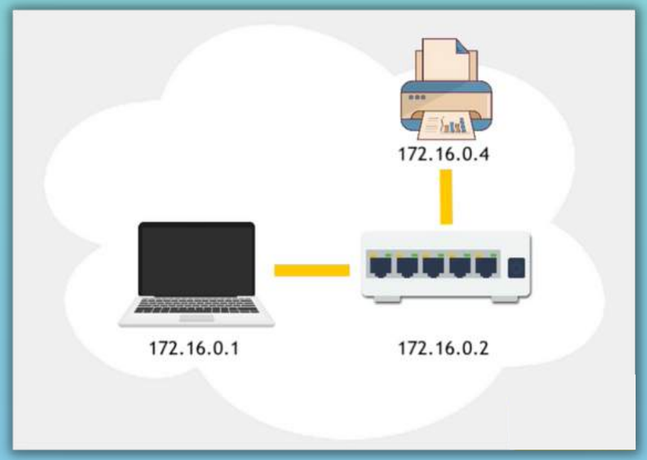
- IP address of router

Introduction to Networking - 3

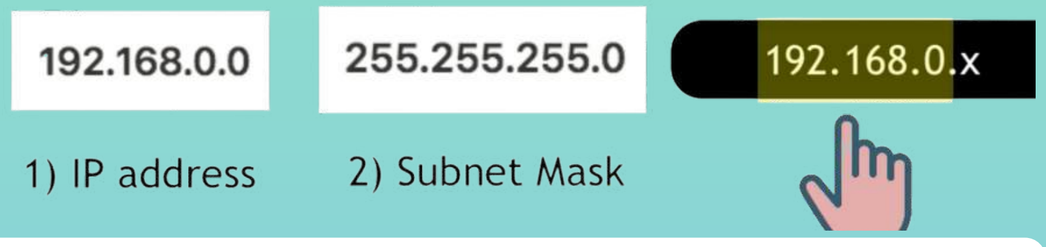
Subnet

- Subnet is a **logical subdivision of an IP network**
- **Subnetting** is the process of dividing a network into 2 or more networks

- Devices in the LAN belong to same IP address range

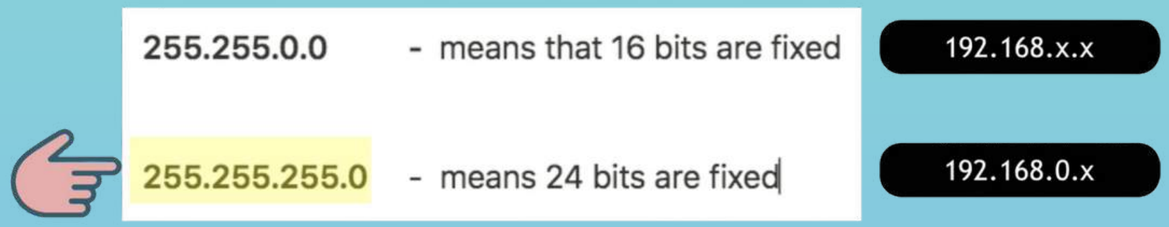


- Example of IP address range:



1) Starting point of IP range, the first IP in the range

2) Sets the IP range



- Value 255 fixates the Octet
- Value 0 means free range

TECHWORLD WITH NANA

Introduction to Networking - 3

CIDR Block = **C**lassless **I**nter-**D**omain **R**outing

- CIDR blocks are groups of addresses that share the same prefix and contain the same number of bits
- Again subnet mask dictates how many bits are fixed

255.255.0.0	- means that 16 bits are fixed
255.255.255.0	- means 24 bits are fixed

- The CIDR notation is a shorthand way of writing this
- CIDR notation looks like this:

192.168.0.0/16 or 192.168.0.0/24

- **/16** bits are fixed
- **/24** bits are fixed

TECHWORLD WITH NANA

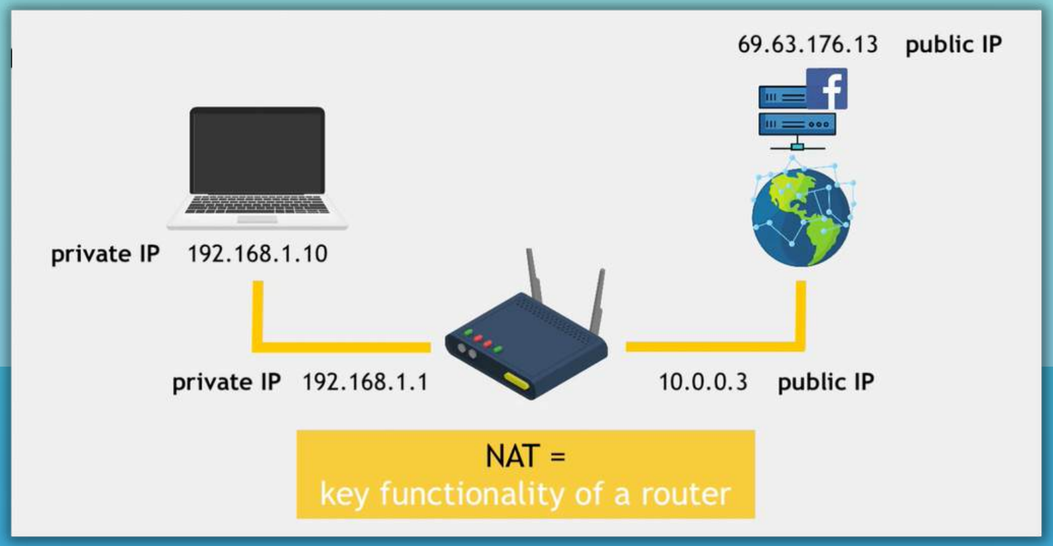
Introduction to Networking - 4

Any device needs 3 pieces of data for communication

- IP Address
- Subnet
- Gateway

NAT = Network Address Translation

- NAT is a way to **map multiple local private addresses to a public one** before transferring the information
- So if you make a request to the internet, the router replaces your private IP address with the router's IP

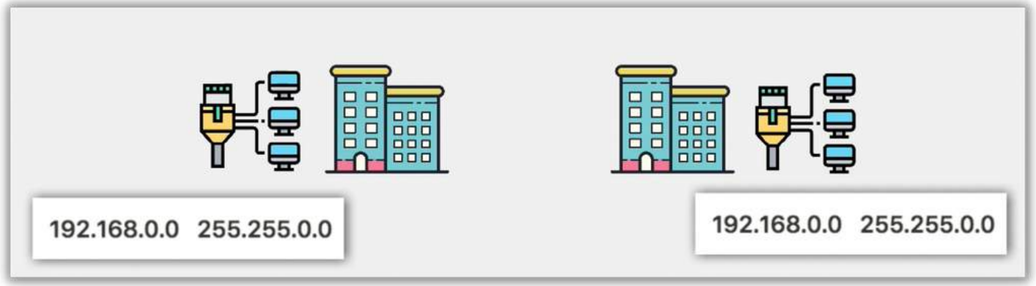


Why?

- IP address within LAN are not visible to the outside network or internet, meaning they are private IPs
- This was necessary because there is only a **limited number of IPv4 addresses** available

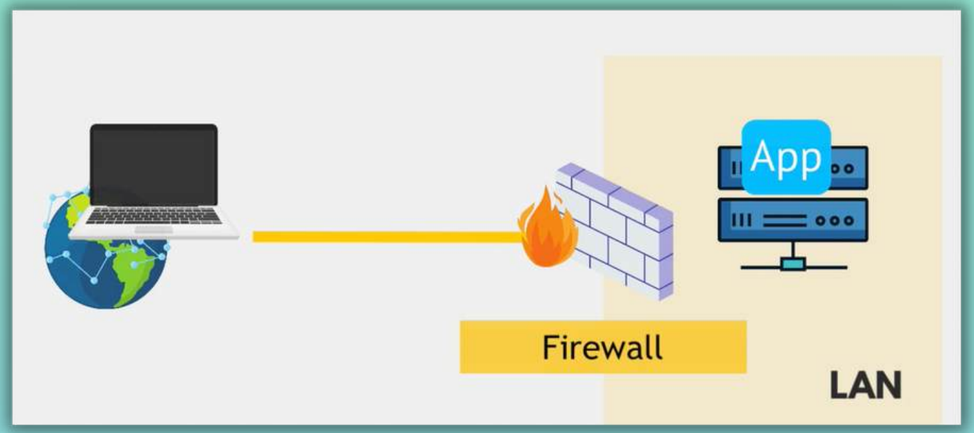
Benefits of NAT:

- **Security** and protection of devices within LAN
- **Reuse** IP addresses

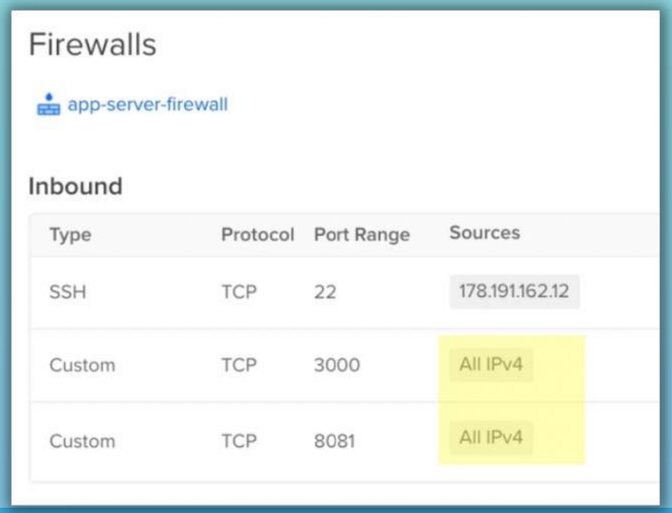


Firewall

- A Firewall **prevents unauthorized access** from entering a private network
- So by default, the server is not accessible from outside the LAN



- Using **Firewall Rules** you can define, which requests are allowed:



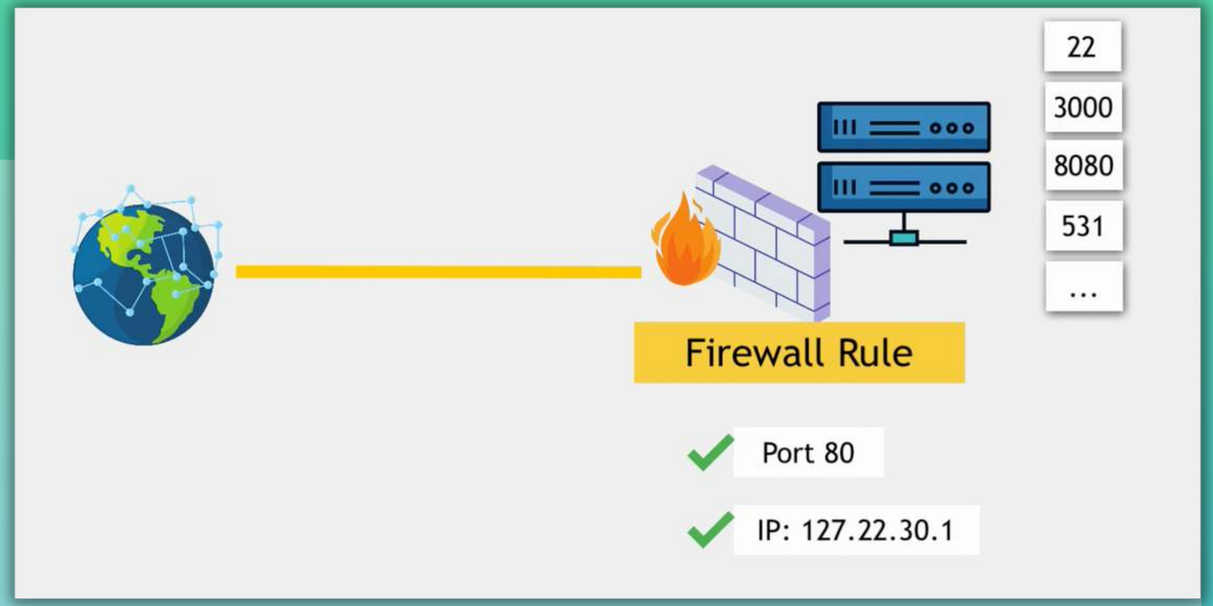
→ Which IP address in your network is accessible

→ Which IP address can access your server

→ For example: You can allow any device access your server

Port

- Every device has a set of ports
- **Each port is unique** on a device
- You can **allow specific ports** (doors)
- You can allow specific ports (doors) AND specific IP addresses (guests)

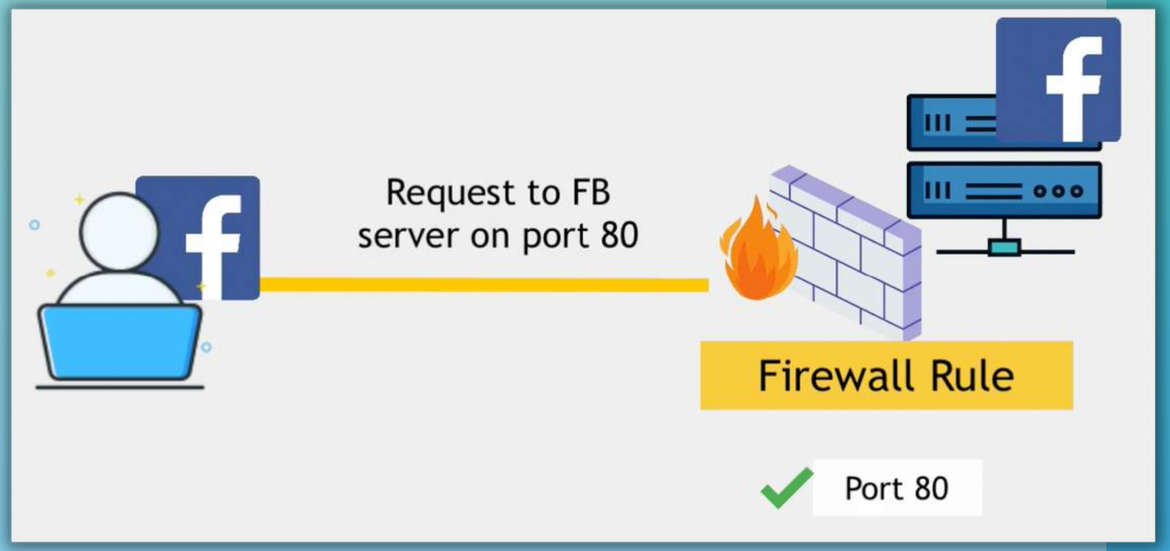


- For every application you need a port
- Different applications **listen** on specific ports
- There are many **standard ports** for many applications

→ Web servers: Port 80

→ Mysql DB: Port 3306

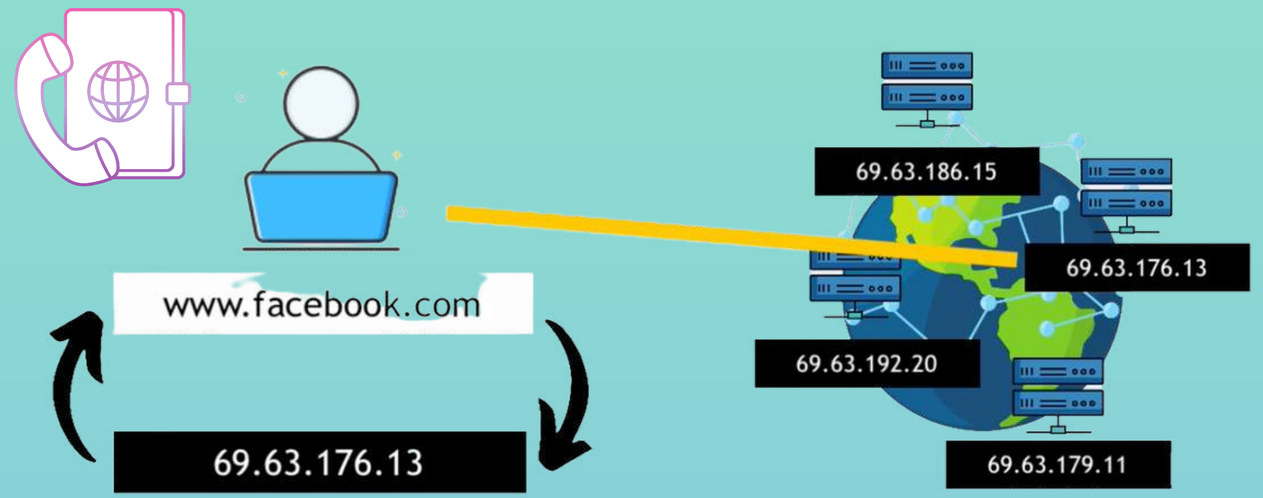
→ PostgreSQL DB: Port 5432



DNS - Domain Name System - 1

Translates domain names to IP addresses

- DNS is the **phonebook of the internet**
- It translates human readable domain names (for example: www.amazon.com) to machine readable IP addresses



www.google.com

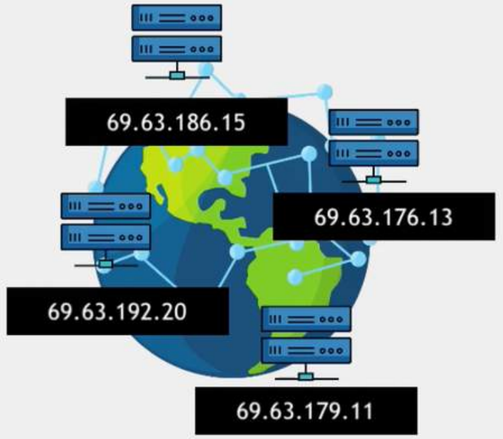
www.mit.edu

www.cncf.io

www.fairtrade.net

www.marines.mil

www.nyc.gov

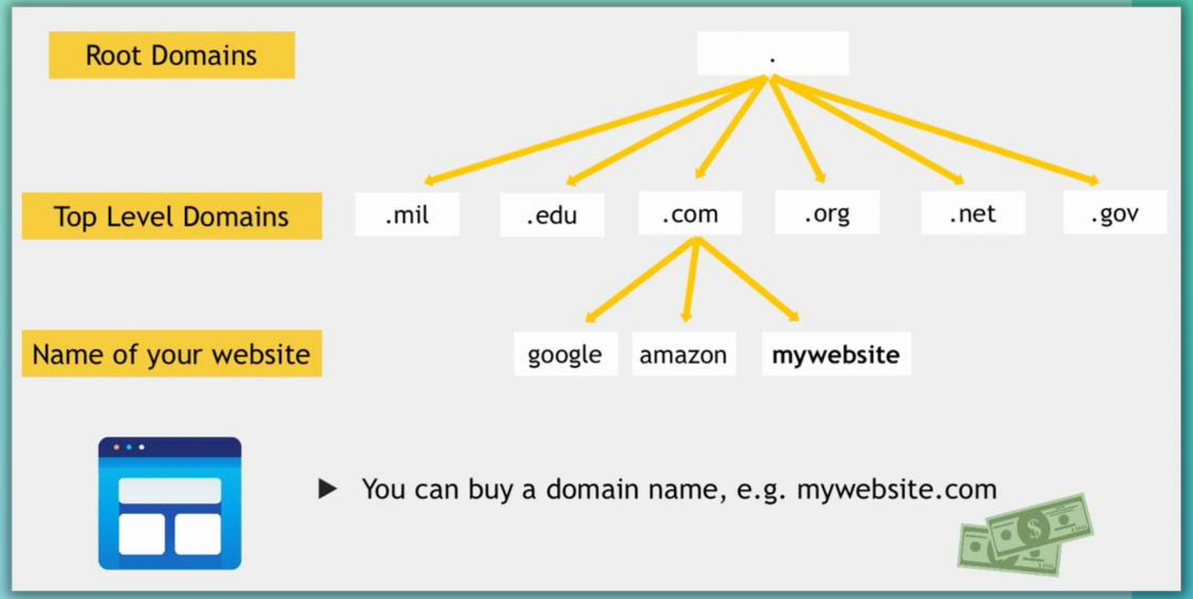


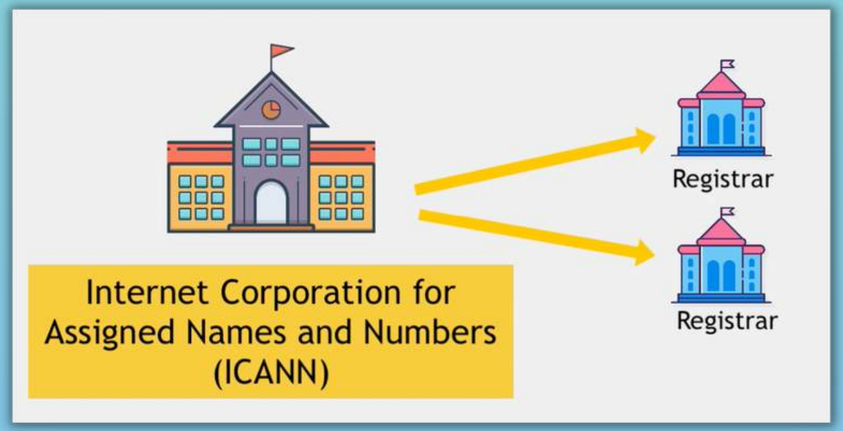
TECHWORLD WITH NANA

DNS - Domain Name System - 2

Domain Names

- Formed by the rules and procedures of the DNS
- Domain names are organized in subordinate levels (subdomains) of the DNS root domain, which is nameless
- The first-level set of domain names are the **top-level domains (TLDs)**
- Below are the second-level and third-level domain names that are open for end-users





ICANN

- Manages the TLD development and architecture of the internet domain space
- **Authorizes Domain Name Registrars**, which register and assign domain names

DNS - Domain Name System - 3

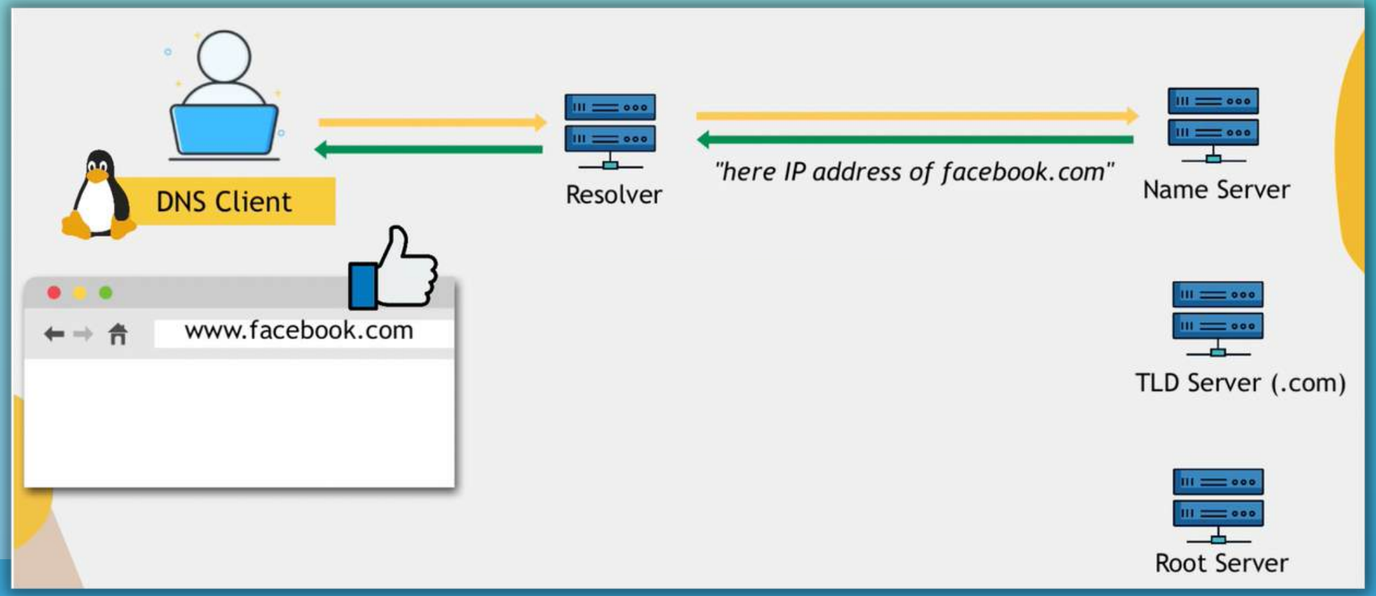
How DNS resolution works

- When you enter a website in a browser, a DNS client on your computer needs to look up the corresponding IP address
- It queries DNS servers to resolve the name
- DNS queries can resolve in different ways

- **DNS Cache**: A client can sometimes answer a query locally using cached (stored) information obtained from a previous query. Or the DNS server can use its own cache of resource record information to answer a query.

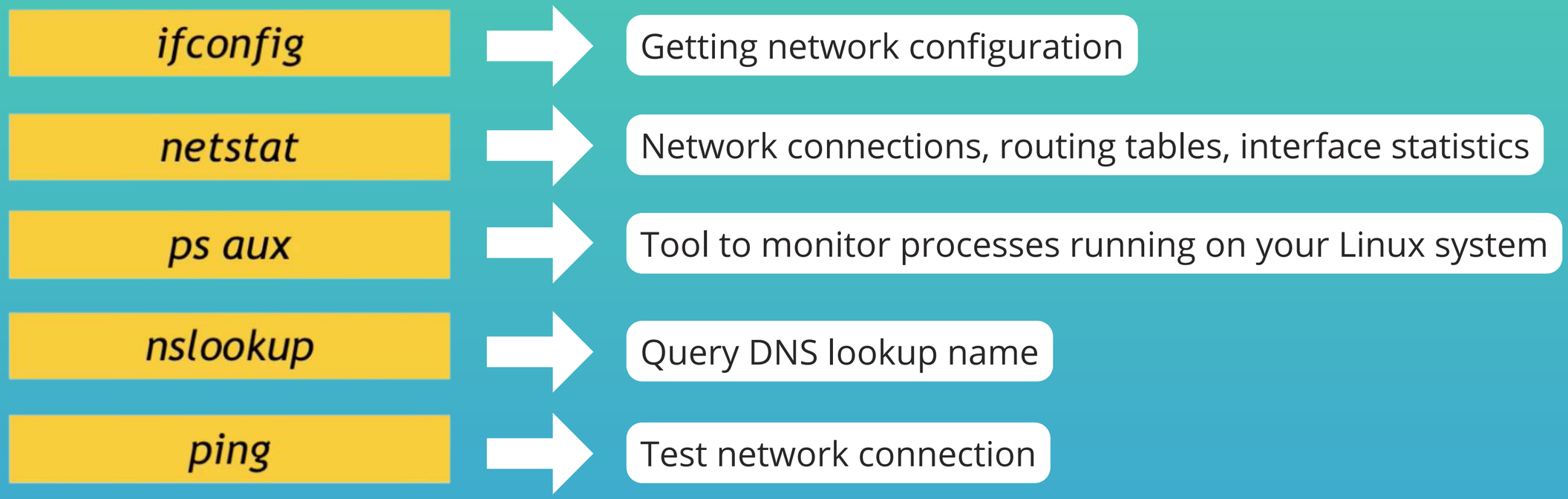
Request Flow

1. **Name Server**: Usually your Internet Service Provider
2. **Root Server**: Requests for top level domains
3. **TLD Server**: Stores the address information for top level domains
4. **Authoritative Name Server:** Responsible for knowing everything about the domain, including IP address



Networking Commands

Some useful networking commands to troubleshoot your network:



SSH - Secure Shell

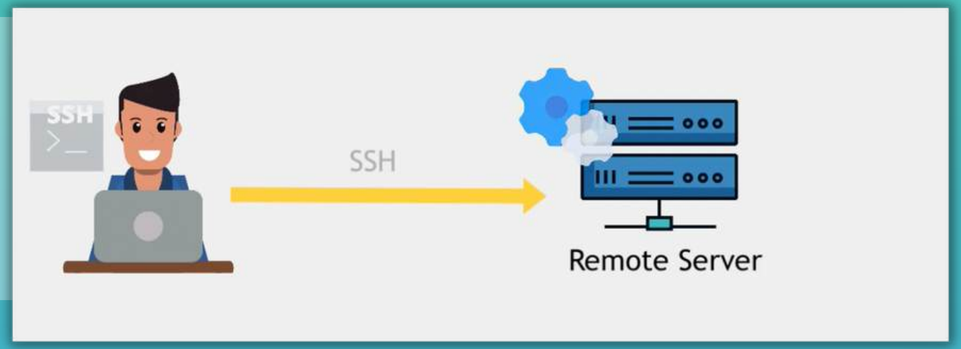
TECHWORLD
WITH NANA

Introduction to SSH - 1

- SSH or Secure Shell is a network protocol, that enables **two computers to communicate securely**
- It protects the communications security and integrity with strong encryption

Use Case

- Often used to "login" and perform tasks on remote computers, like installing software on a new server or to copy a file to the server



How it works

- The protocol works in the client-server model
- Meaning you use a program on your computer (**ssh client**) to connect to the remote server (**SSH server**)
- For that you can use a graphical user interface, but mostly the CLI

Introduction to SSH - 2

2 ways to authenticate

1) Username & Password

- Admin creates a user on the remote server
- User can then connect with the username and password

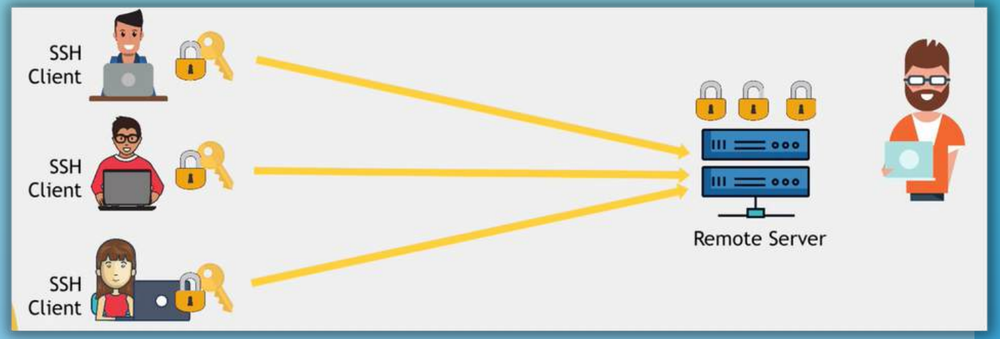
2) SSH Key Pair

- Client creates a SSH Key Pair

Key Pair = Private Key + Public Key

- **Private Key** = Secret key. Is stored securely on the client machine
- **Public Key** = Public. Can be shared, e.g. with the remote server

- If public key of a person is not registered on the remote server, they cannot connect to it



Introduction to SSH - 3

SSH for Services

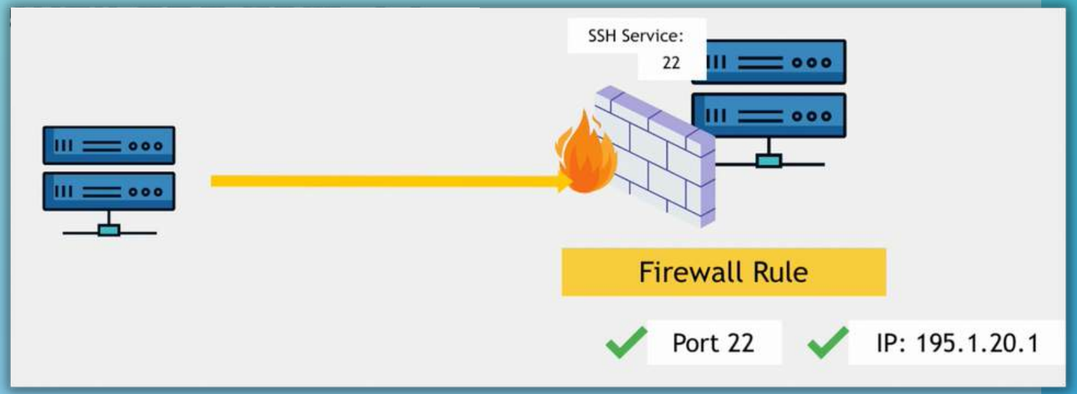
- Services, like Jenkins, often need to connect to another server via SSH

How to:

1. Create Jenkins User on application server
2. Create SSH key pair on Jenkins server
3. Add public key to *authorized_keys* on application server

Firewall and Port 22

- SSH service (on SSH server) runs on **port 22**
- That's why in firewall rule, we need to allow access on that port
- But also configure the source (who can access the server on that port). Because SSH is powerful and needs to be **restricted to specific IP addresses**



Introduction to SSH - 4

`ssh UserName@SSHserver` → Connect to a remote host

`~/.ssh` → .ssh under home directory is the default location for your ssh key pair

known_hosts file → Lets the client authenticate the server to check that it isn't connecting to an impersonator

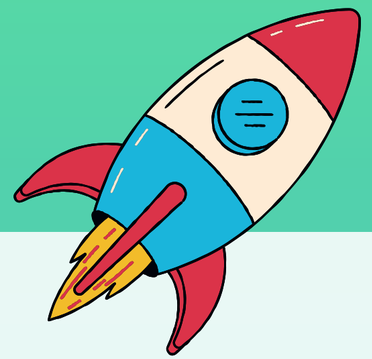
authorized_keys file → Let's the server authenticate the user

Best Practices - 1

Security Best Practices for Linux Servers:

- Use SSH instead of password for logging into servers
- If somehow that's not possible, use strong and unique passwords:
  - Same password should never be used for multiple users or software systems
  - Configure expiration, to update the passwords regularly
  - Use password manager for two-factor authentication, password generation, cloud password storage etc.
- Update your software regularly or enable automatic updates
- Avoid unnecessary software, as each new software can expose the server to potential problems
- Regularly backup your data. The application "rsync" is a popular option in Linux

Best Practices - 2



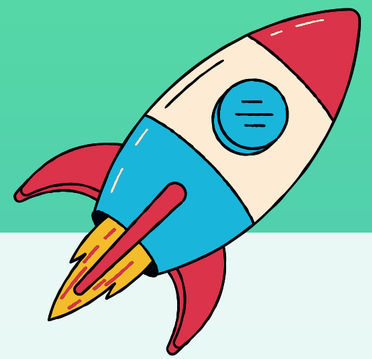
Users & Permissions

- Only use the root account for systems administration. Login with normal user and su to root
- Remove or lock user accounts that are no longer needed
- Make sure that passwords on active accounts are changed regularly
- Lock User Accounts after a number of unsuccessful login attempts

Bash Scripting

- Best Practices: https://bertvv.github.io/cheat-sheets/Bash.html

Best Practices - 3



Networking

- Always have a Firewall
- Restrict network access as much as possible
    - Ports, which you don’t need should be closed

SSH

- SSH, like all network services, should be disabled if not needed
- Disable root logins via SSH
    - Can be configured here: /etc/ssh/sshd_config
    - Login with a normal user and su to root