University May 8, 1945 GUELMA
faculty of Mathematics, Computer Sciences, Physics and Chemistry
Computer Science department
First year Computer Engineer
Semester 1: Introduction to Operating Systems 1

Chapter 2
LINUX OPERATING SYSTEM

Dr FERKOUS chokri

Linux /Unix users

- In Linux operating systems, there are different types of users, each with specific permissions and purposes.
- These user types help manage access to the system and ensure that users have appropriate levels of control and security.
- Here are some common types of users in Linux:
  1. Superuser (Root)
  2. Regular Users
  3. Group Users
  4. Guest Users
  5. System Users & Service Accounts

Linux /Unix users

- **Superuser (Root)**

The superuser, often referred to as "**root**," has the highest level of privileges and authority on a Linux system.

Root can perform any action, access any file, and modify any system configuration.

Root is responsible for system administration tasks, including software installation, configuration, and system maintenance.

It should be used sparingly, as mistakes or unauthorized changes made by the root user can have significant consequences.

Linux /Unix users

Regular Users

- Regular users are standard user accounts created for individuals who interact with the system.
- These accounts have fewer privileges than the superuser or system users.
- **Regular users** have home directories where they can store personal files and configurations.
- They can execute programs, access their own files, and perform various tasks based on their assigned permissions.

Linux /Unix users

Group Users

- In Linux, users can be organized into groups to simplify permission management.
- Groups allow multiple users to share access to files and directories with the same permissions.
- Each user can be a member of one or more groups, and group permissions can be set on files and directories to control access for group members.

Linux /Unix users

Guest Users

- Some Linux distributions offer guest user accounts with limited access and privileges.
- Guest users are typically used in scenarios where temporary or restricted access is required.

- Guest accounts often have restrictions on file access, system configuration changes, and the ability to install software.

Linux /Unix users

Remote Users (SSH Users)

- Remote users are individuals or systems that access the Linux server remotely, typically through SSH (Secure Shell) or other remote access methods.

- Remote users can be regular users or specific user accounts created for remote access, and their access can be restricted based on security policies.

Linux /Unix users

Sudo Users

- **Sudo** users are regular users who are granted limited superuser privileges through the sudo command.
- They can execute specific commands with elevated permissions after entering their own password or an administrator's password.

- **Sudo** users are often used to delegate administrative tasks while maintaining accountability and control.

Linux /Unix users

System Users & Service accounts

- System users are typically created during the installation of various system services and applications.
- They are used to run specific processes or services.
- System users often have limited or no login capabilities and exist solely for the purpose of running background tasks or services.
- They usually have their home directories set to locations like /var or /usr.
- Service accounts are user accounts created for specific services or applications. These accounts are used to run services securely with limited permissions.
- Service accounts are common in web server, database server, and email server configurations, where the service requires its own user account for security and resource isolation.

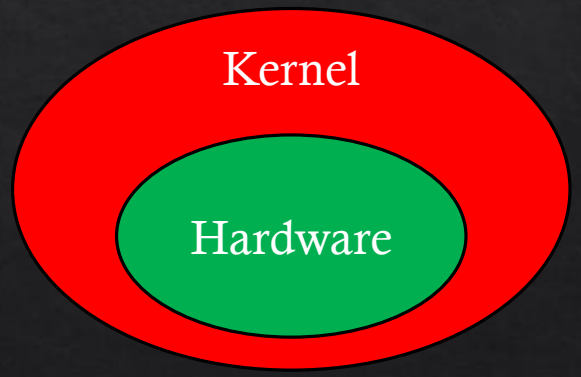
Linux Architecture

- The Linux operating system's architecture mainly contains some of the components:
  - **The Kernel**
  - **System Library**
  - **Hardware layer**
  - **System calls**
  - **Shell utility** & application

Linux Architecture

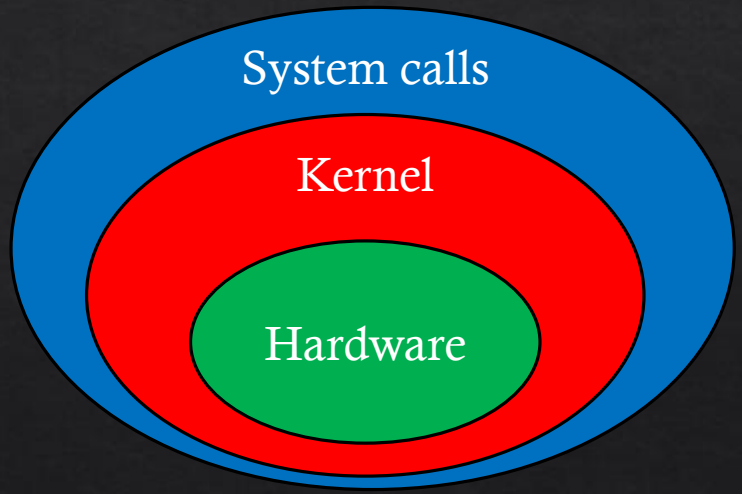
- Hardware layer:- Linux operating system contains a hardware layer that consists of several peripheral devices like CPU, HDD, and RAM.

- **Kernel:-** The kernel is one of the core section of an operating system.
- It is responsible for each of the major actions of the Linux OS. This operating system contains distinct types of modules and cooperates with underlying hardware directly.
- The kernel facilitates required abstraction for hiding details of low-level hardware or application programs to the system.



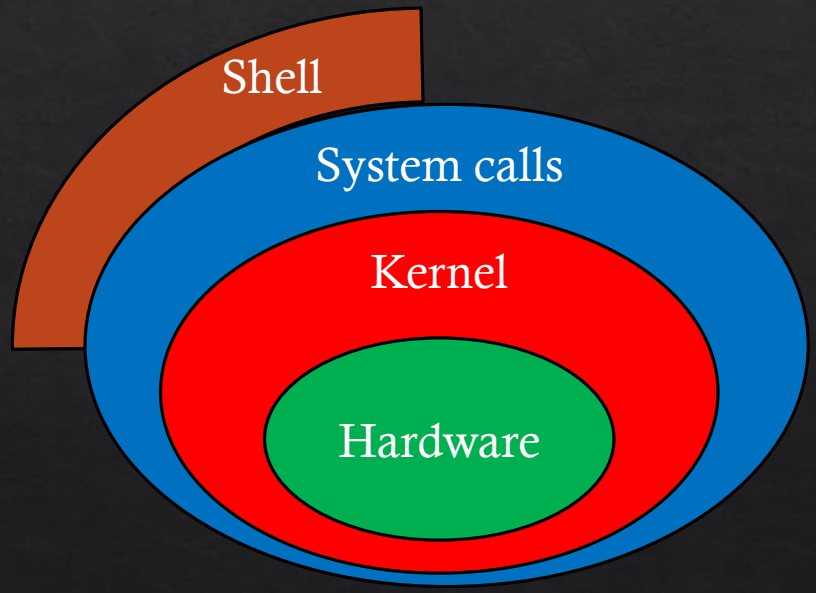
Linux Architecture

- **System call**: a method for application to interact with the **kernel**



Linux Architecture

- **Shell:-** It is an interface among the kernel and user. It can afford the services of kernel. It can take commands through the user and runs the functions of the kernel.
- The shell is available in distinct types of OSes.
- These operating systems are categorized into two different types, which are the graphical shells and command-line shells.

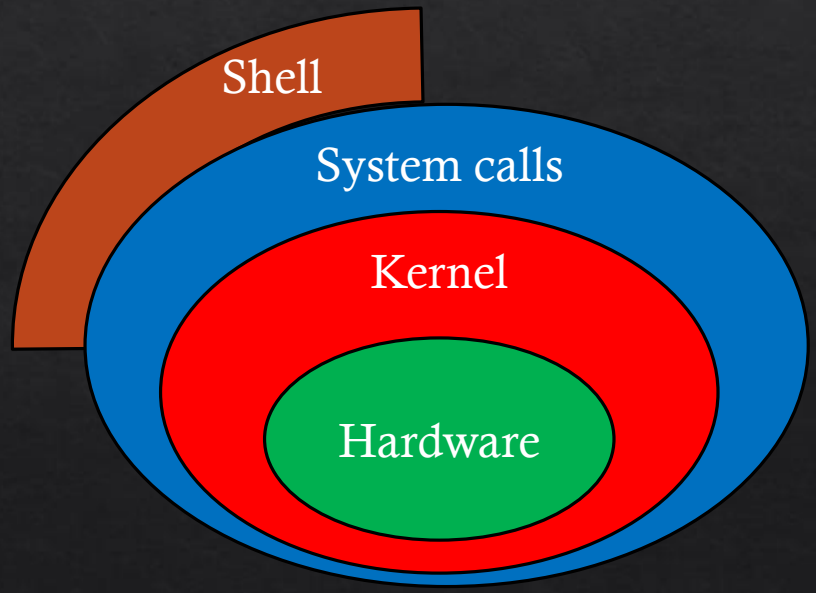


The graphical line shells facilitate the graphical user interface, while the command line shells facilitate the command line interface.

Thus, both of these shells implement operations. However, the graphical user interface shells work slower as compared to the command-line interface shells.

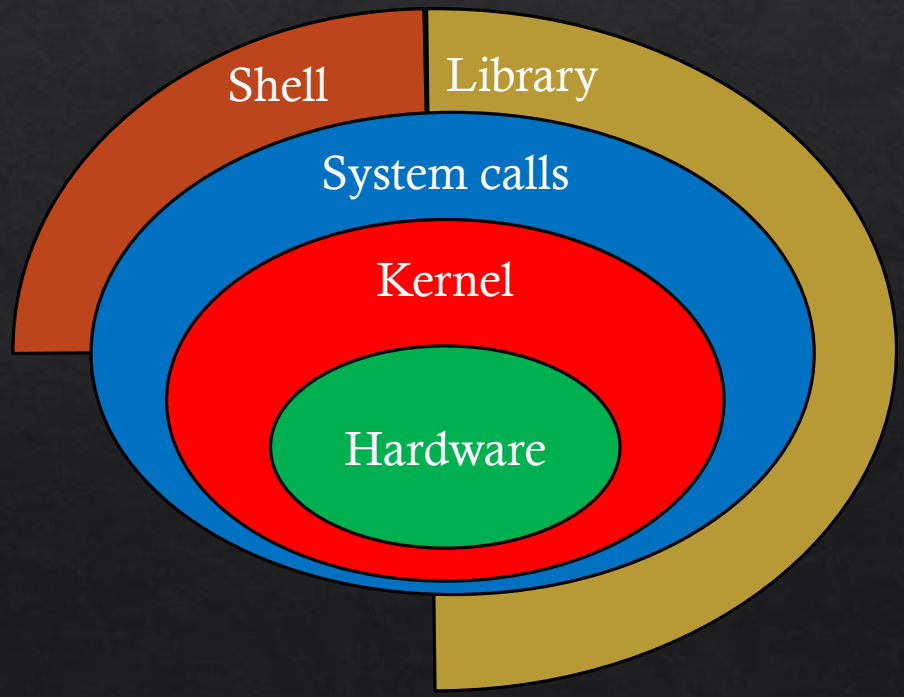
Linux Architecture

- There are a few types of these shells which are categorized as follows:
  - Korn shell
  - Bourne shell
  - C shell
  - POSIX shell
  - ….



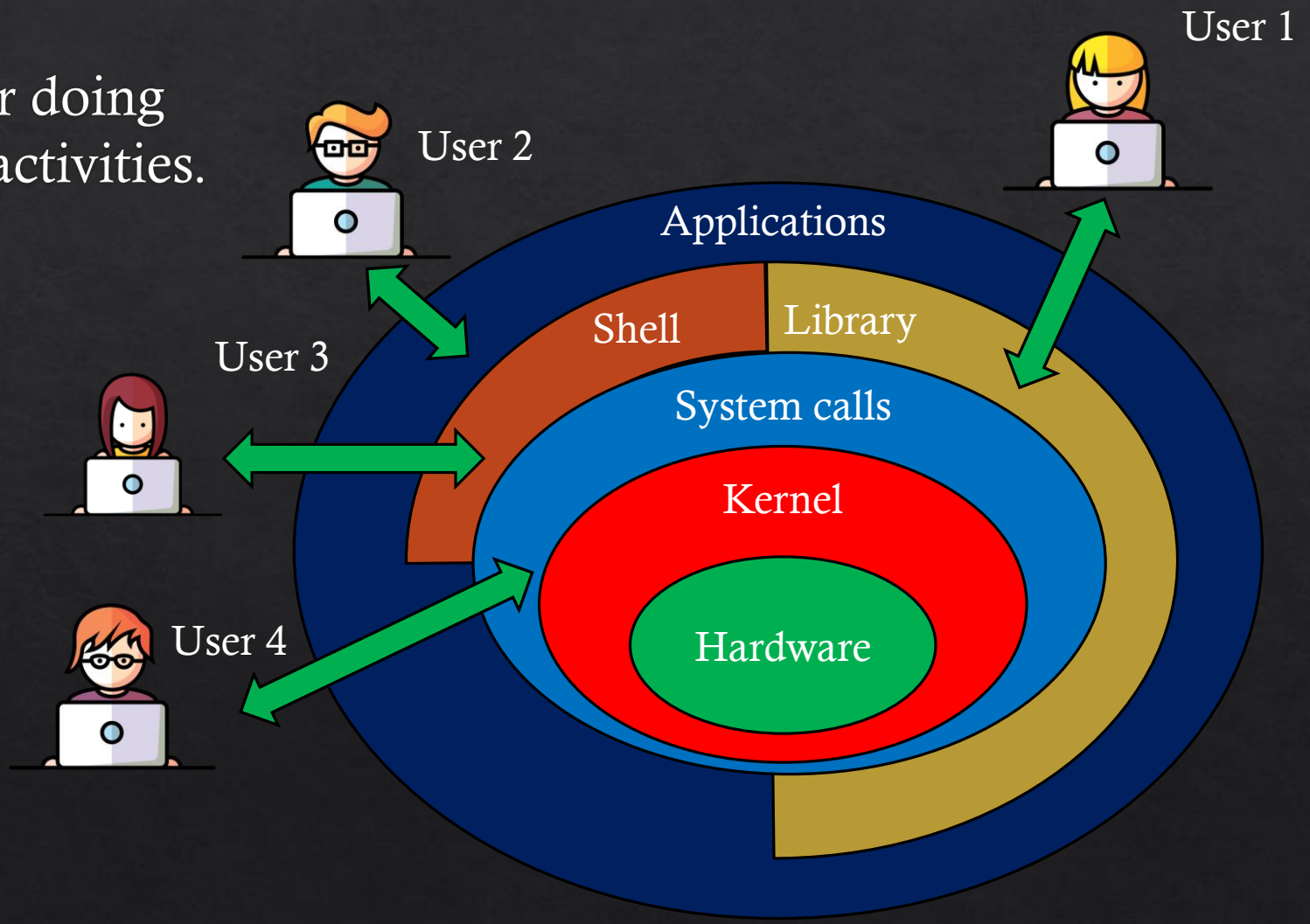
Linux Architecture

- **System Libraries:-** These libraries can be specified as some special functions.
- These are applied for implementing the operating system's functionality and don't need code access rights of the modules of kernel.



Linux Architecture

- Applications: It is responsible for doing specialized level and individual activities.



Command-Line Interface (CLI)

- A shell is a command-line interface (CLI) program that provides users with a way to interact with an operating system's kernel and execute commands. It acts as an intermediary between the user and the computer's internal processes, allowing users to issue commands, run programs, and manipulate files and directories by typing text-based instructions.

Functions of a shell

- **Command Interpretation**: The shell interprets the commands entered by the user and converts them into system calls and actions that the operating system kernel can understand and execute.
- **Text-Based Interface:** Shells provide a text-based interface where users type commands in plain text. Users can interact with the system by entering commands, arguments, and options.
- **Command Execution:** Shells execute system commands and external programs by invoking them. Users can run utilities, launch applications, and perform various tasks through the shell.
- **Scripting:** Shells support scripting, allowing users to create and run scripts—sequences of commands that can be saved in a file and executed as a program. Shell scripts are used for automation and task automation.

Shell

- Common Unix-like operating systems, such as Linux and macOS, come with a default shell. Some of the most well-known shells include:

  Bash (Bourne Again Shell): Bash is one of the most widely used shells in Unix-like systems. It is the default shell on many Linux distributions and macOS.

  Zsh (Z Shell): Zsh is an extended shell with features like improved autocompletion, custom prompt theming, and advanced scripting capabilities. It's often chosen for interactive use.

  Fish (Friendly Interactive Shell): Fish is designed for ease of use and offers features like syntax highlighting, autosuggestions, and an interactive help system.

  tcsh: tcsh is an enhanced version of the C shell (csh) with added features like command history and better interactive usage.

  Korn Shell (ksh): Korn Shell, often referred to as ksh, is known for its scripting capabilities and is used for both interactive and scripting purposes.

Users can typically switch between different shells based on their preferences and needs. The choice of shell can impact the user experience and productivity, as each shell may offer different features and capabilities.

The syntax of a shell command

- The syntax of a shell command in Linux typically consists of the following elements:

1. **Command Name**: This is the actual name of the command you want to execute.

   For example: "ls," "mkdir," "cd," "cp," "mv," …etc.

2. **Options (Flags):** Options modify the behavior of a command. They are typically preceded by a hyphen (-) or double hyphen (--).

   For example:

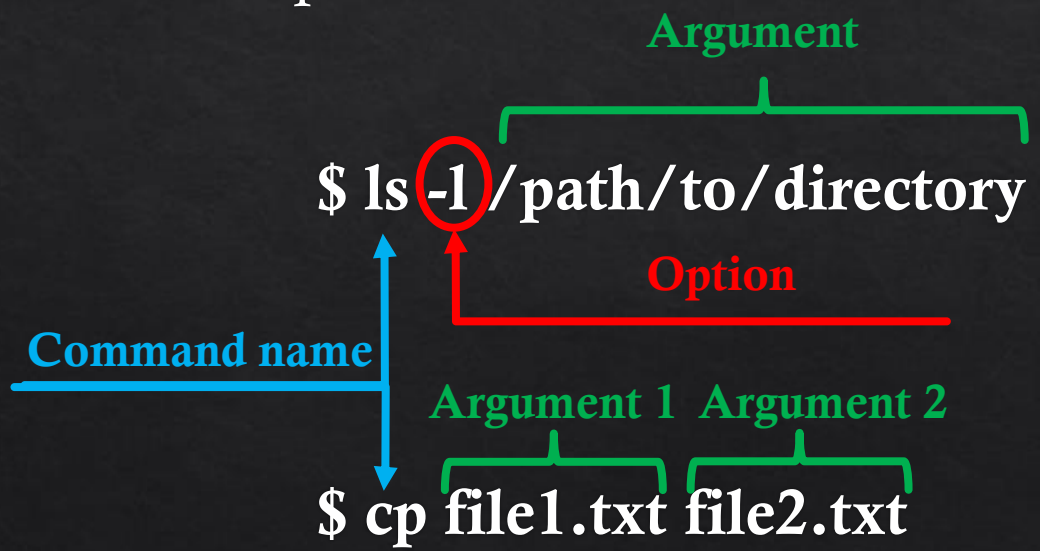
   Single-letter options: -l, -a, -r, …etc.

   Long options: --list, --all, --recursive, etc.

The syntax of a shell command

3. Arguments: These are the inputs or parameters that the command requires to perform its operation. Arguments are often files, directories, or values that the command operates on. Some commands might not require any arguments.

For example:



The syntax of a shell command

4. Command Separators:
Example:

Semicolon (;): Allows you to run multiple commands on a single line sequentially.

Pipe (|): Redirects the output of one command as input to another.

Logical AND (&&): Executes the second command only if the first one succeeds.

Logical OR (||): Executes the second command only if the first one fails.

Linux Directory Structure

- In a Linux operating system, various principal files and directories play important roles in system configuration, control, and organization. Here are some of the principal files and directories commonly found in Linux:

- **1. / (Root Directory):** The root directory is the top-level directory in the Linux file system hierarchy. It contains all other directories and files, and it is represented by a single forward slash (/).

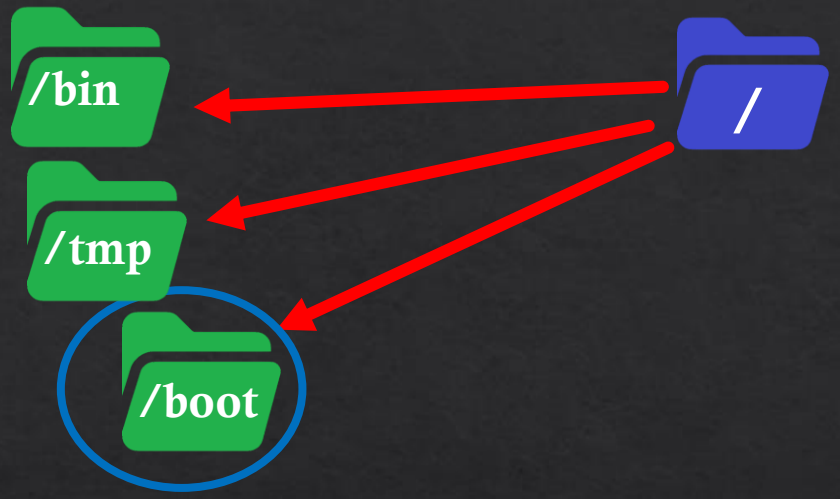
Linux Directory Structure

2. /bin (Binary Binaries): This directory contains essential system binary executable files. for all users, e.g., cat, ls, cp.

Linux Directory Structure

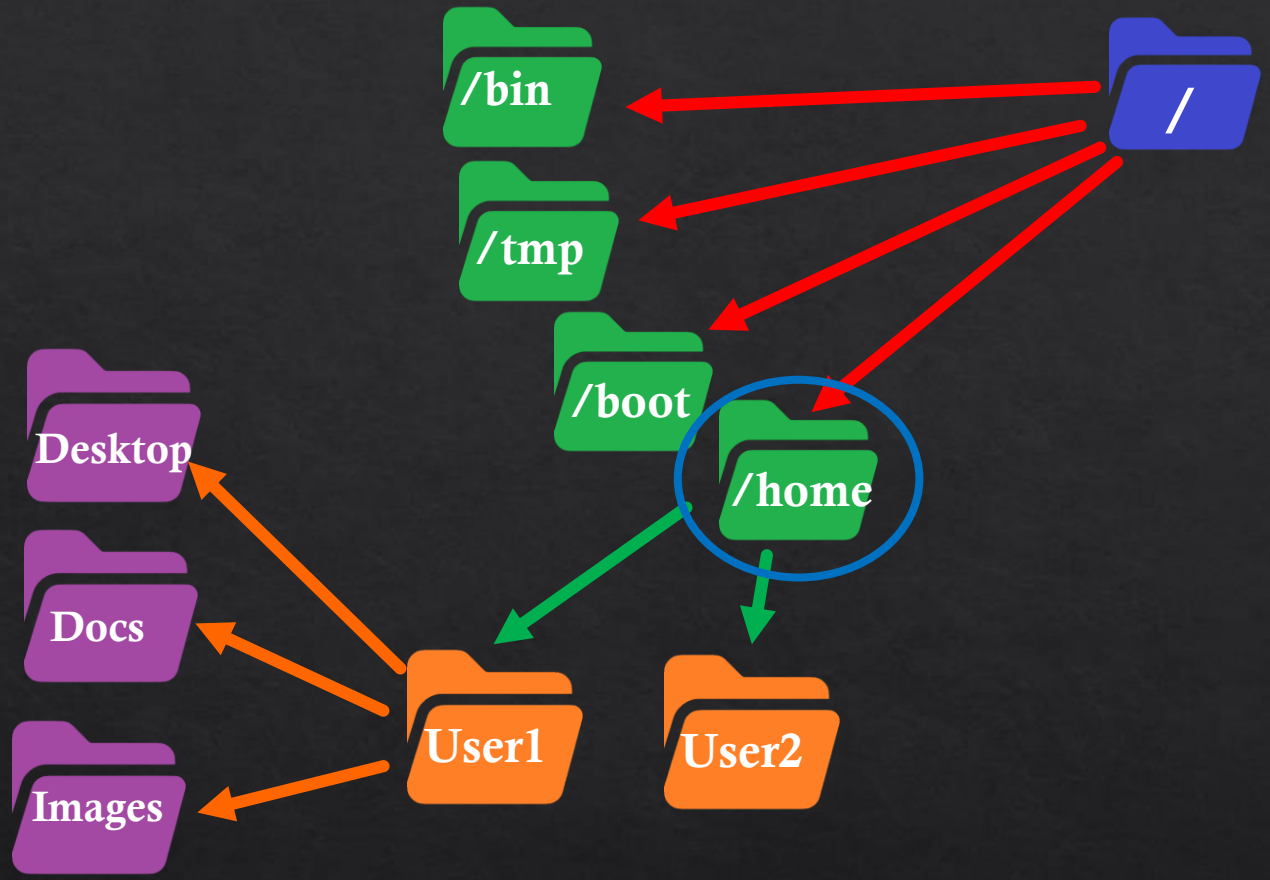
3. /tmp (Temporary): The /tmp directory is used for temporary files. It is typically emptied at boot or periodically to free up space.

Linux Directory Structure



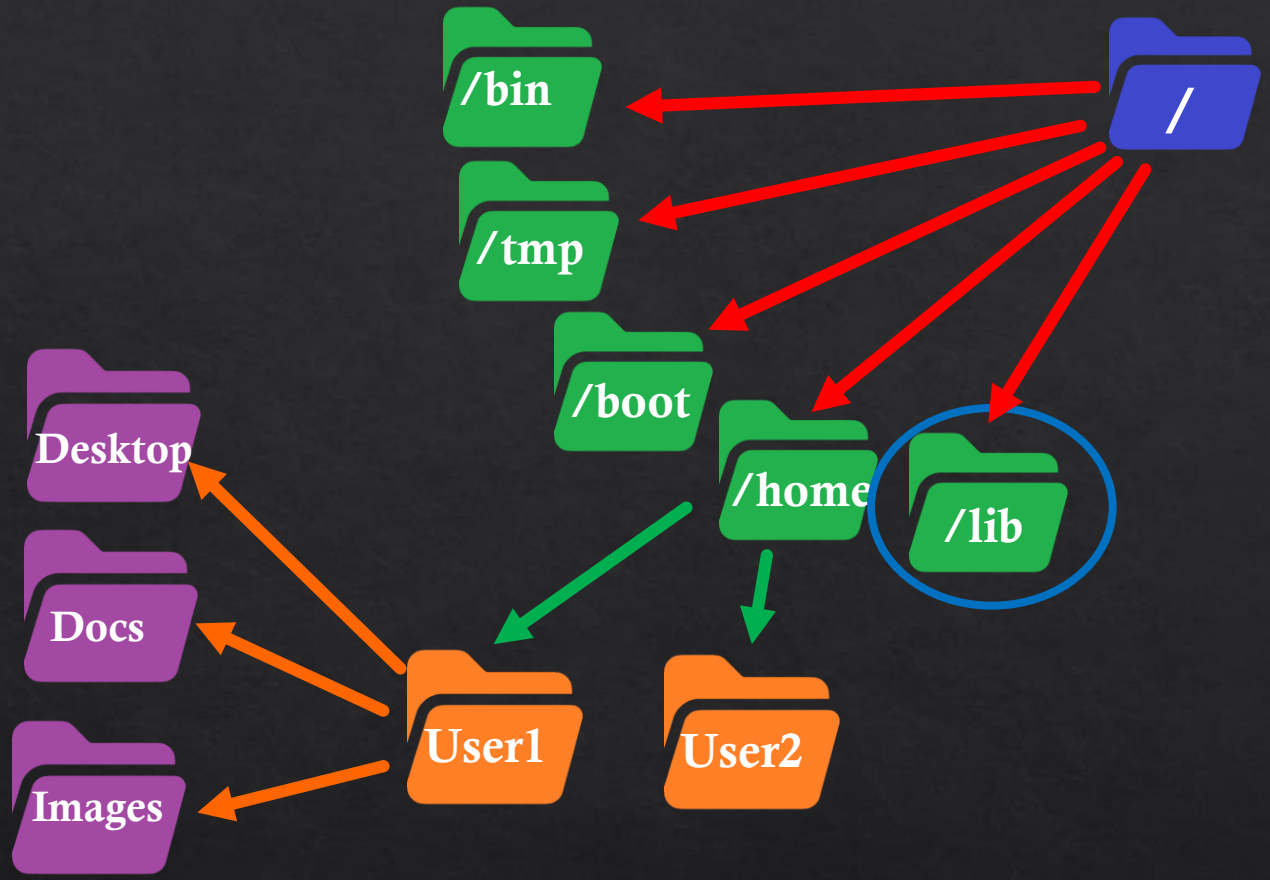
4. /boot: This directory holds the kernel and other files required for booting the system.

Linux Directory Structure



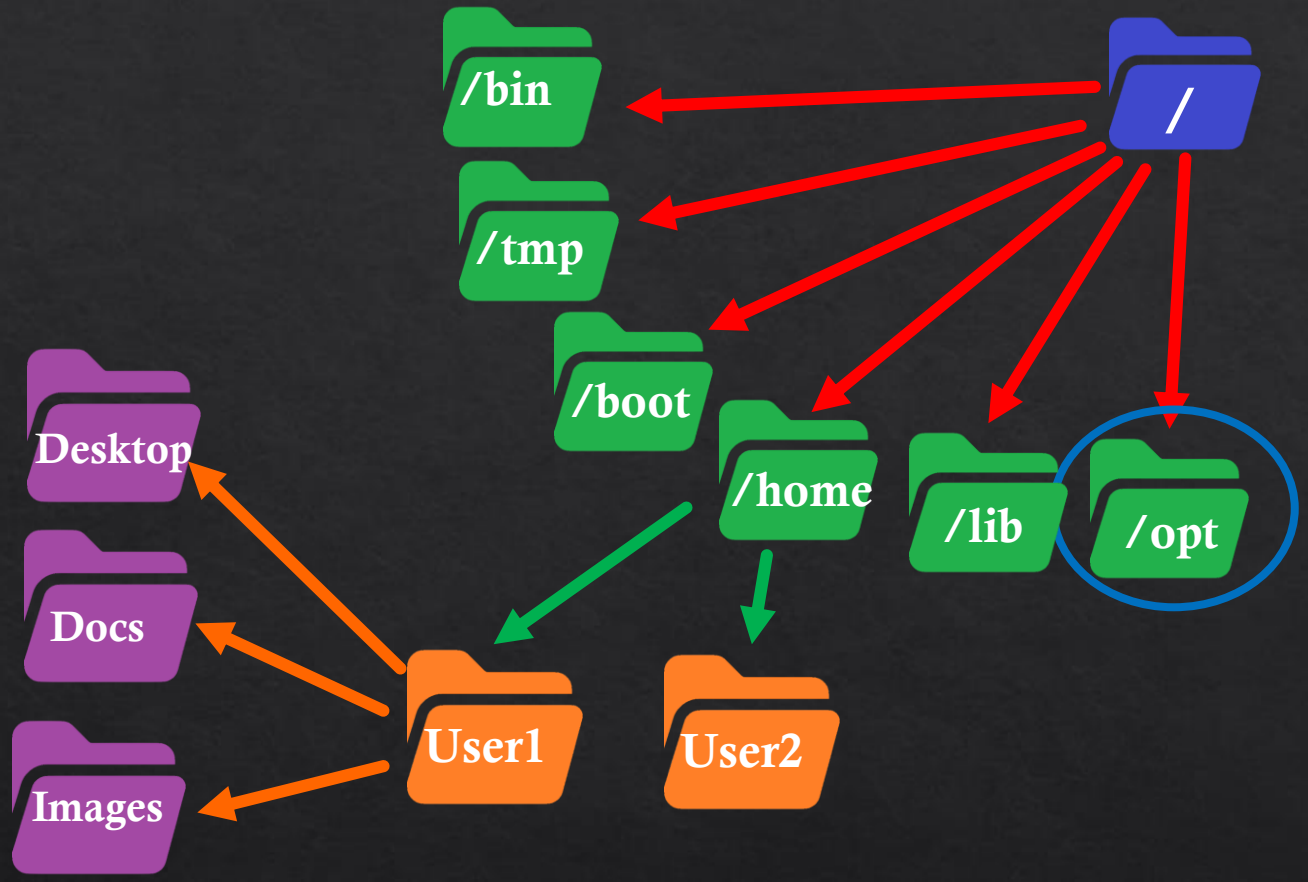
5. /home: The /home directory is the location for user home directories. Each user typically has a subdirectory here for their personal files and settings.

Linux Directory Structure



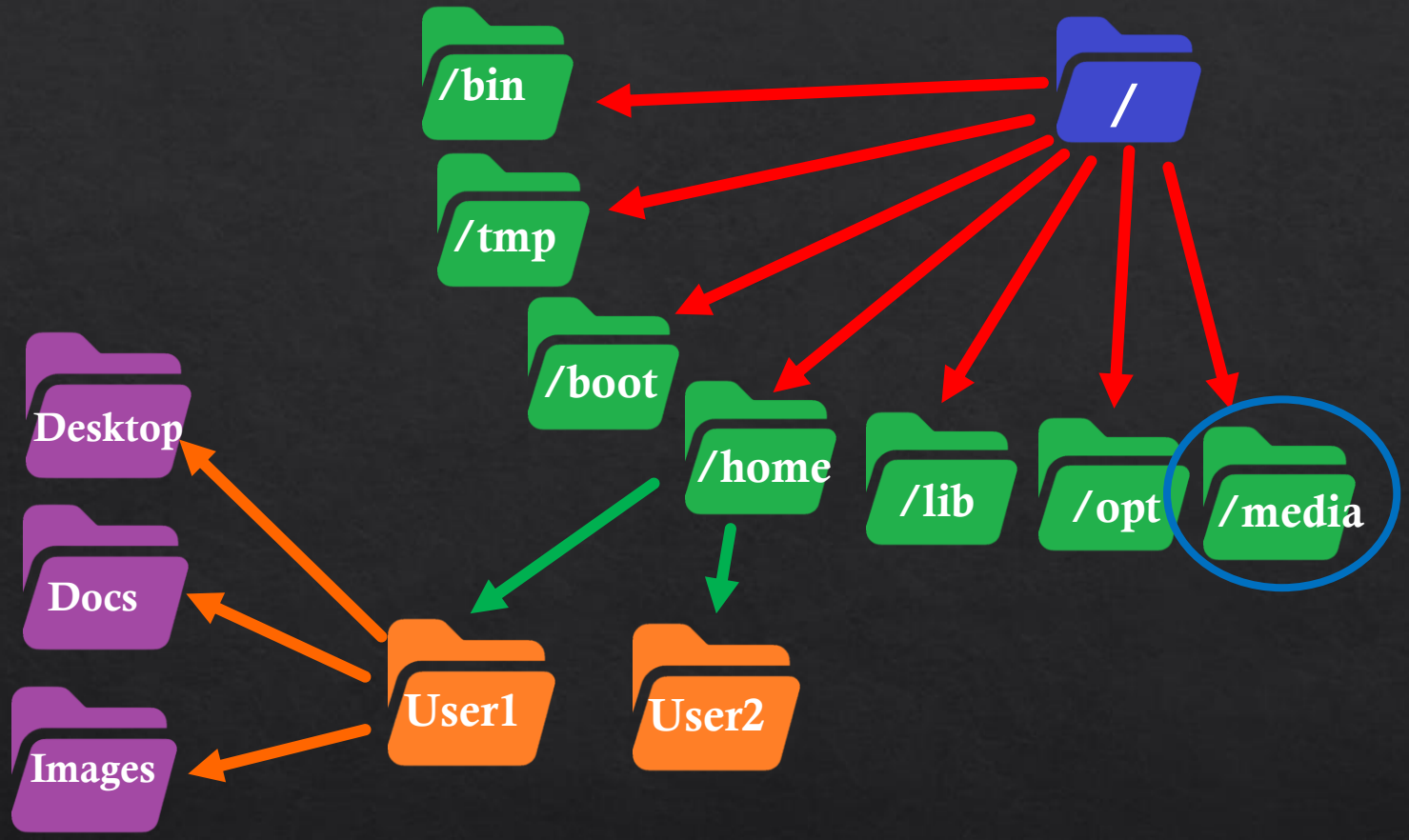
6. /lib (Library): The /lib directory contains shared libraries and kernel modules needed by programs and the system. It includes both dynamically linked libraries (shared libraries) and kernel modules.

Linux Directory Structure



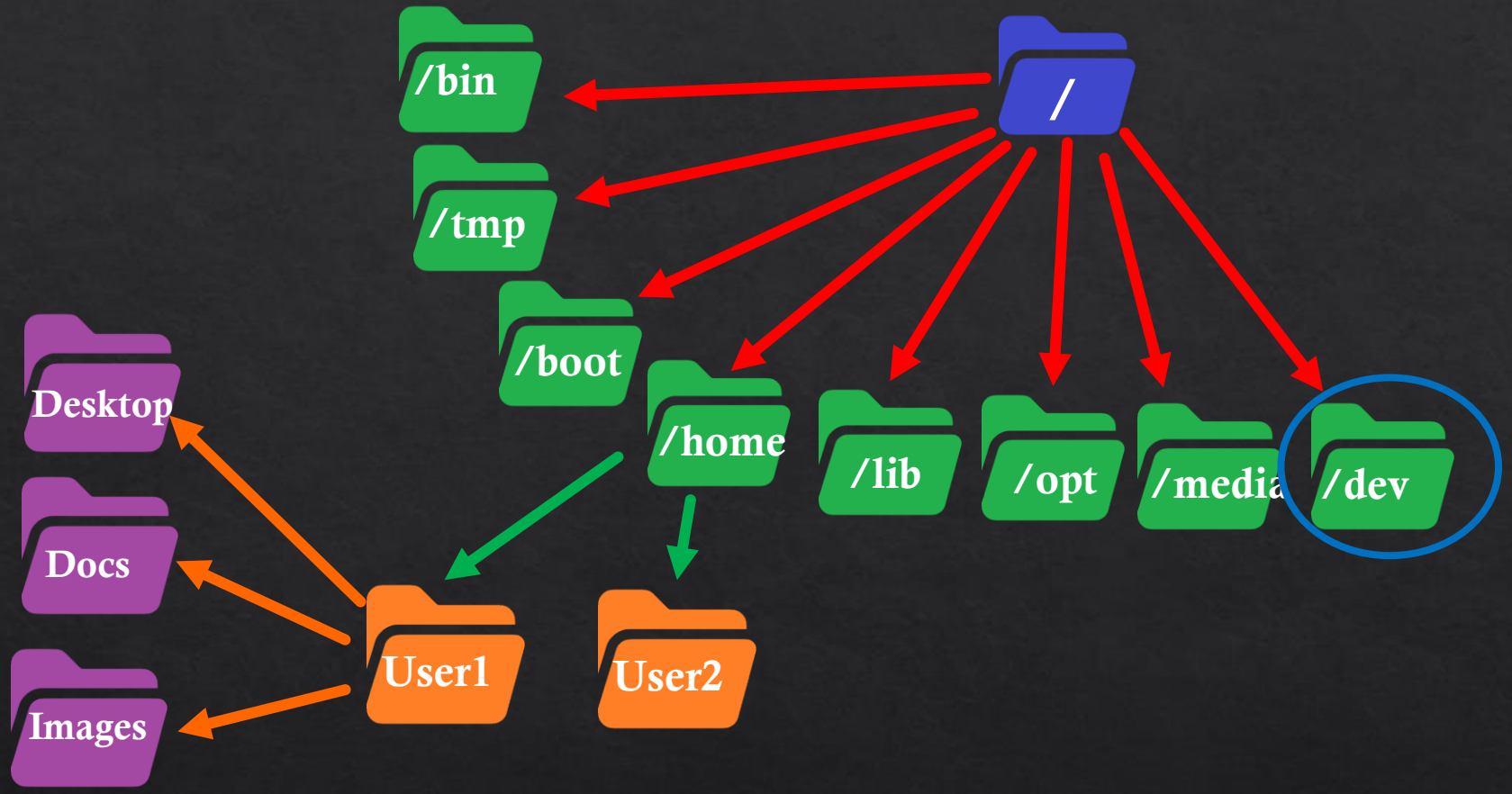
7. /opt (Optional): The /opt directory is used for installing optional software and applications.

Linux Directory Structure



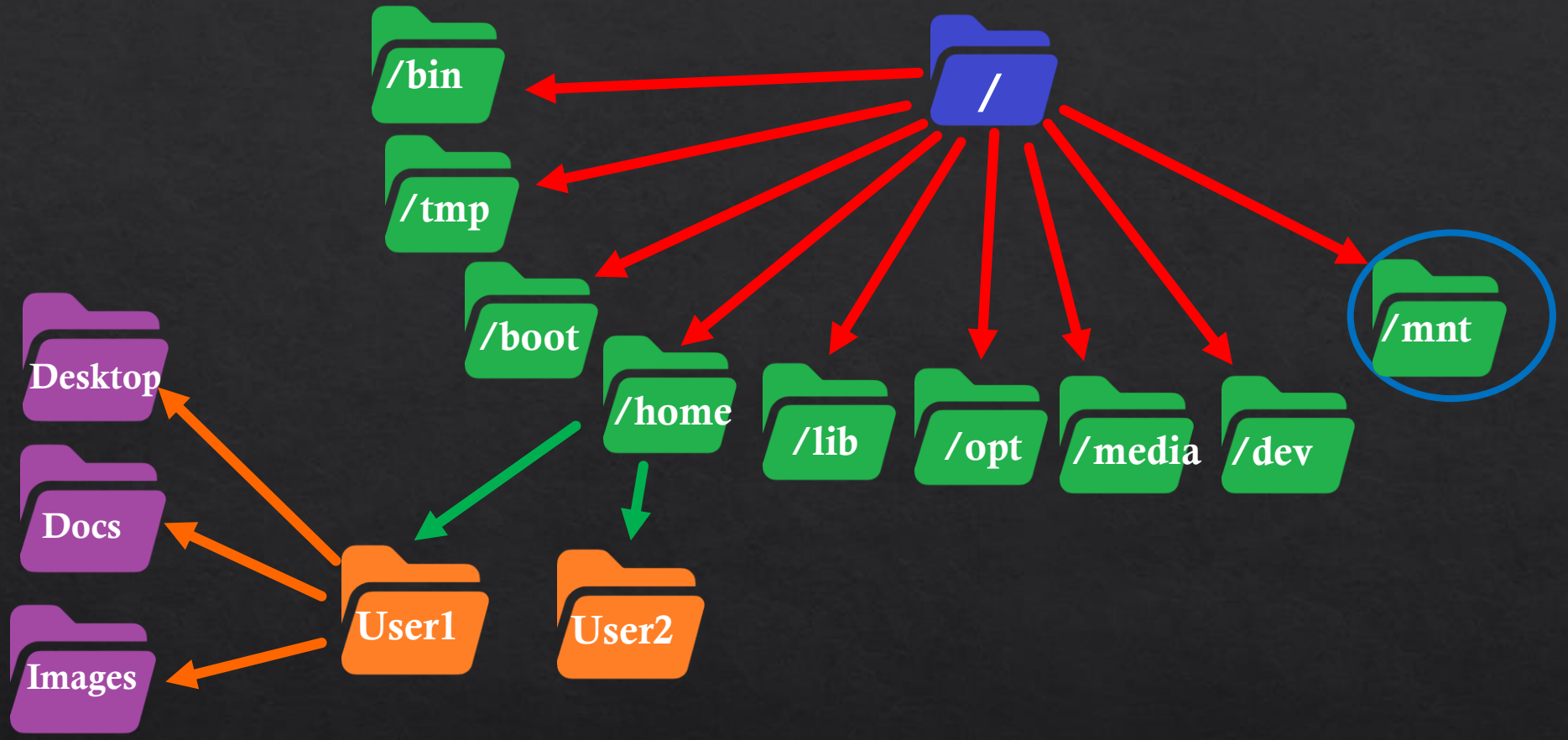
8. /media: Removable media devices (e.g., USB drives, optical discs) are often mounted under the /media directory.

Linux Directory Structure



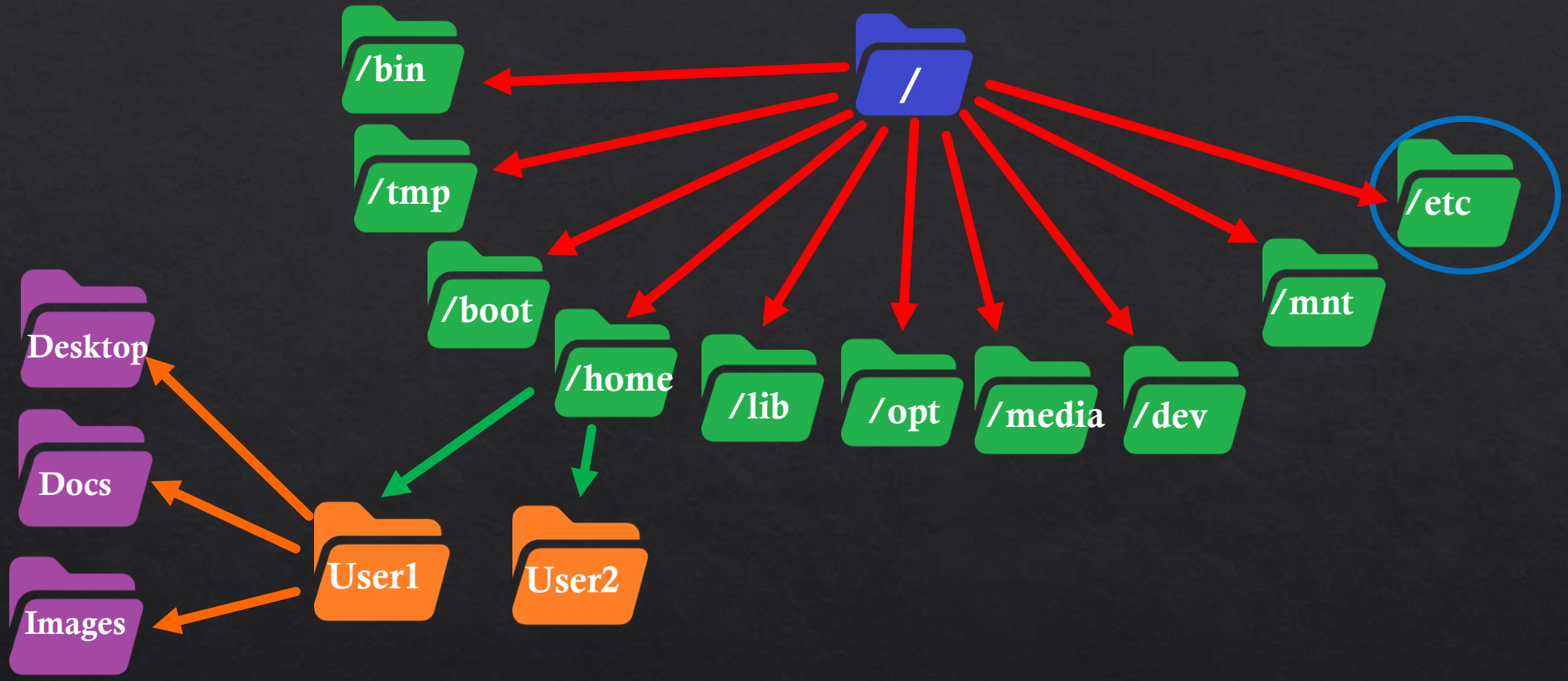
9. /dev (Device): The /dev directory contains device files that represent hardware devices and interfaces, allowing programs to communicate with hardware.

Linux Directory Structure



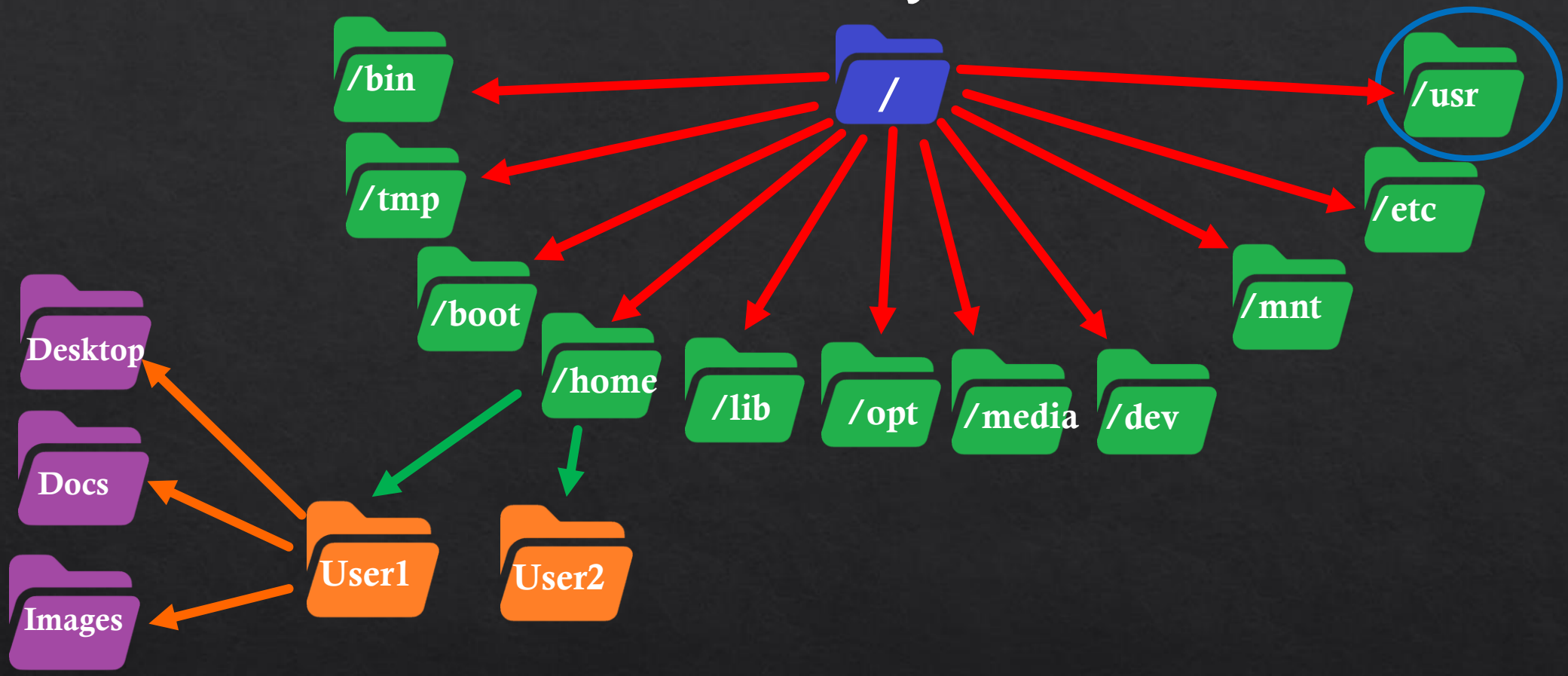
10. /mnt (Mount): The /mnt directory is used for temporarily mounting file systems or devices, especially those not mounted at boot.

Linux Directory Structure



11. /etc (Etcetera): The /etc directory contains system-wide configuration files and scripts. It's where you'll find configuration files for system services, user accounts, network settings, and more.

Linux Directory Structure



12. /usr (Unix System Resources): The /usr directory contains user data, software applications, and libraries, with a structure similar to the root directory.

Print working directory

- **pwd command**

The **pwd** command prints your current working directory's path, like **/home/path**. Here's the command syntax:

```
$ pwd
```

Listing files

- **ls command**

The **ls** command lists files and directories in your system. Here's the syntax:

```
$ ls [options] [path]
```

If you remove the path, the ls command will show the current working directory's content.

You can modify the command using these options:

-R – lists all the files in the subdirectories.
-a – shows all files, including hidden ones.
-lh – converts sizes to readable formats, such as **MB**, **GB**, and **TB**.

Essential Linux Commands

- **man command**

man command in Linux is used to display the user manual of any command that we can run. Here's the command syntax:

```
$ man [command Name]
```

- **Example:**

```
$ man ls
```

File access

- **cd command**

Use the **cd** command to navigate the Linux files and directories. To use it, run this syntax :

```
$ cd [path]
```

Depending on your current location, it requires either the **absolute path** or **relative path**.

Absolute Vs Relative Path

- **Absolute and Relative Pathnames in UNIX**

A path is a unique location to a file or a folder in a file system of an OS.

A path to a file is a combination of / and **alpha-numeric characters**.

`Example: /home/mohamed/Desktop`

1. An absolute path: is defined as the specifying the location of a file or directory from the **root directory(/)**.

To write an absolute path-name:

- Start at the root directory (/) and work down.
- Write a slash (/) after every directory name (last one is optional)

Absolute Vs Relative Path

2. Relative path: Relative path is defined as the path related to the present working directly (pwd). It starts at your current directory and **never starts with a /** .

UNIX offers a shortcut in the relative **pathname**; that uses either the **current** or **parent directory** as reference and specifies the path relative to it.
A relative path-name uses one of these cryptic symbols:

- **.(a single dot)** - this represents the current directory.
- **..(two dots)** - this represents the parent directory.

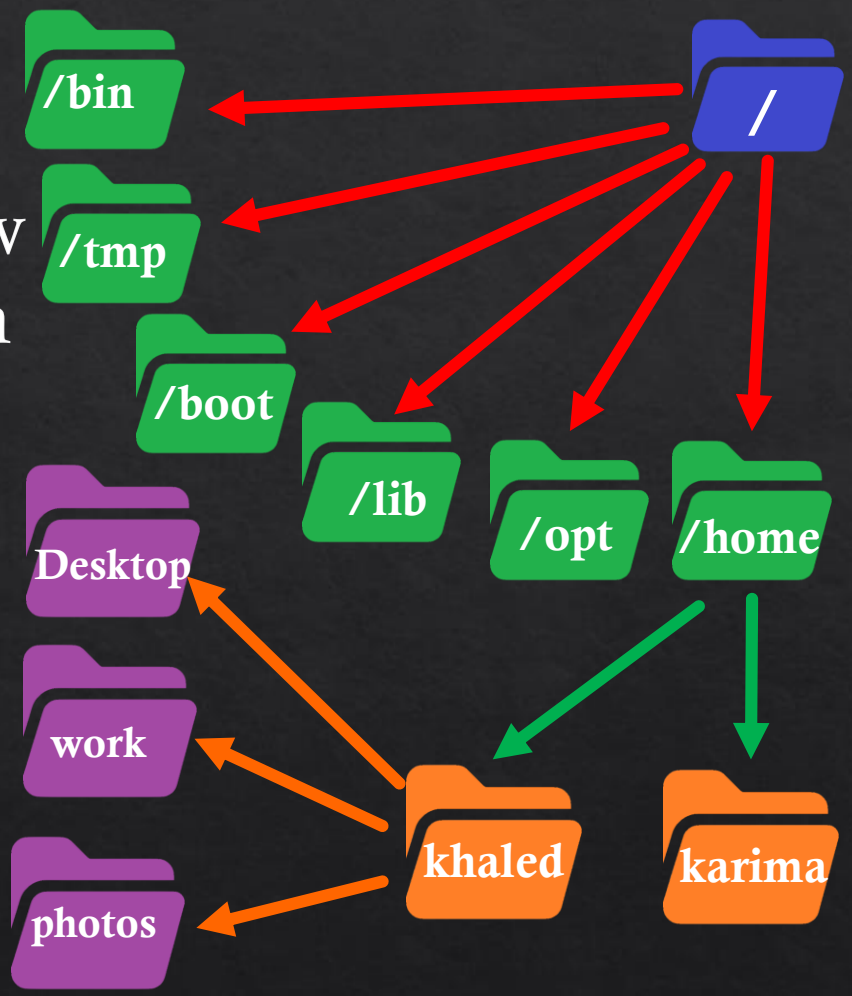
Absolute Vs Relative Path

- **Example of Absolute and Relative Path**
- To be more specific let's take a look on the below figure in which if we are looking for photos then **absolute path** for it will be provided as

`/home/khaled/photos`

- but assuming that we are already present in khaled directory then the **relative path** for the same can be written as simple photos.

`./photos` or `photos`



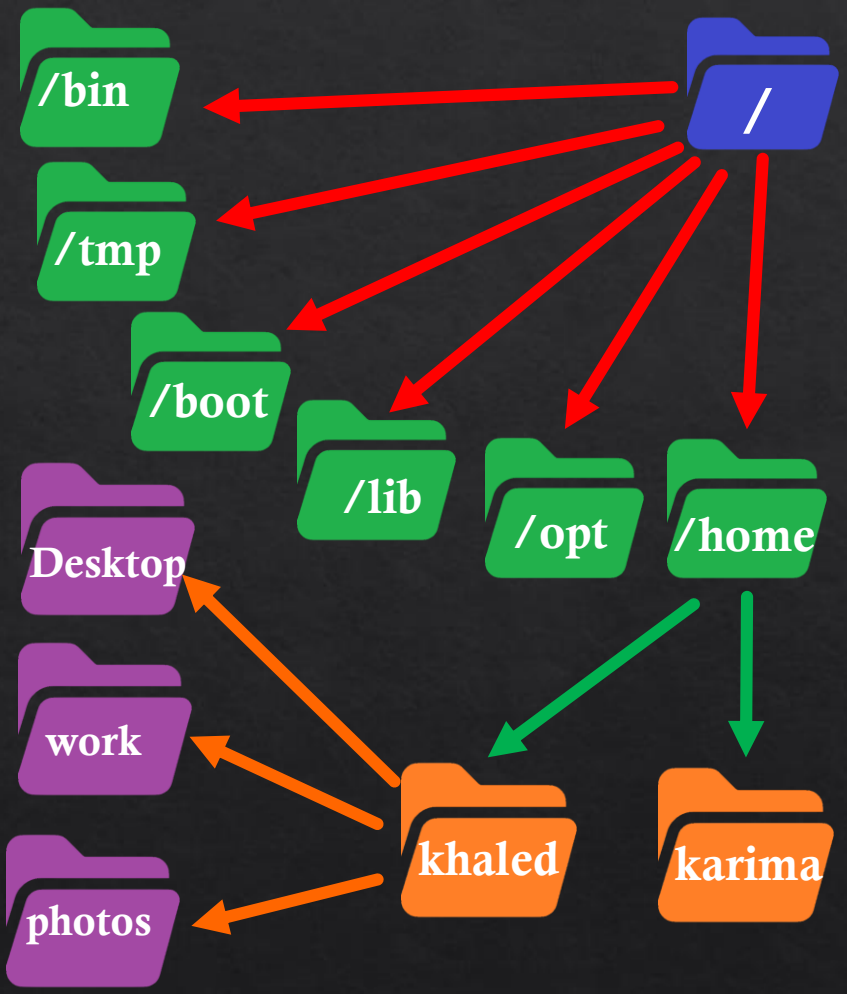
Changing Directories

- **File access**

Now, what this actually means is that if we are currently in directory **/home/khaled/photos**

and now you can use **..** as an argument to **cd** to move to the parent directory **/home/khaled**:

```
$cd ..  ***moves one level up***
```



Example2:

```
$pwd
/home/Khaled/photos
$cd ../..       ***moves two level up***
$pwd
/home/
```

Creating Directories

- **mkdir command:** we use the **mkdir** command to create one or multiple directories and set their permissions. Ensure you are authorized to make a new folder in the parent directory. Here's the basic syntax:

```
$ mkdir [option..] [directories ..]
```

- To create a folder within a directory, use the path as the command parameter.

For example, **mkdir courses/chapter1** will create a **chaptre1** folder inside **courses**.

Here are several common **mkdir** command options:

- **-p** – A flag which enables the command to create parent directories as necessary. If the directories exist, no error is specified.
  For example, **mkdir -p courses/chapter1/pw1**.
- **-v** – prints a message for each created directory.

Creating Directories

Example:

`$ mkdir mydir` Creates the directory **mydir** in the **current directory**.

`$ mkdir /tmp/mydir` This command creates the directory **mydir** in the **/tmp** directory.

- **Creating Directories**

Create multiple directories using **mkdir** command

Example: If we want to create a directory name “chapter1, chapter2 and chapter3”.

```
$ mkdir chapter1 chapter2 chapter3
```

Removing Directories

- Directories can be deleted using the **rmdir** command as follows :

```
$rmdir dirname
```

- **Note** – To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

- You can remove multiple directories at a time as follows :

```
$rmdir dir1 dir2 dir3
```

- The above command removes the directories dir1, dir2, and dir3, if they are empty.

Coping Files and Directories

- The **cp (copy)** command can be used to copy a file or a directory. The syntax is as follows

```
$ cp sourcePath destinationPath
```

- **Example:**

```
~:$ cp OS.txt /tmp/file.txt
```

Moving/Renaming Files and Directories

- The **mv (move)** command can be used to move a file or a directory. The syntax is as follows

```
$ mv sourcePath destinationPath
```

- **Example:**

```
$ mv SE1 /tmp/OS1
```

- The **mv** command can be used also to rename a file or a directory. The syntax is as follows

```
$ mv oldname newname
```

- **Example:**

```
$ mv SE1 OS1
```

Options for cp **and** mv

Options	**Option Meaning**
-i, --interactive	Prompt before overwrite (overrides a previous -n option).
-n	Do not overwrite an existing file (overrides a previous –i option).

Delating Files and Directories

The rm Command

- **rm** stands for remove, and it is used to remove files, directories, and links.

By default, it does not remove directories.

- This command normally works silently and it should be used carefully, because once you delete a file in Linux the content cannot be recovered.
- **Syntax:**

```
$ rm [OPTION]... FILE...
```

- There are a couple of common options, which you might notice are similar to the options for both **cp** and **mv**

Options	Option Meaning
-i	Prompt before every removal.
-r, -R, --recursive	Remove directories and their contents recursively.

Essential Linux Commands

- **touch command**

The **touch** command lets you create an empty file in a specific directory path. Here's the syntax:

```
$ touch [option…] [files …]
```

Touch command to create multiple files:

Touch command can be used to create multiple numbers of files at the same time. These files would be empty while creation.

Syntax:

```
$ touch File1.txt File2.txt File3.txt
```

Essential Linux Commands

- **whoami command**

The whoami command allows Linux users to see the currently logged-in user. The output displays the username of the effective user in the current shell. Additionally, whoami is useful in bash scripting to show who is running the script.

```
$ whoami
```

Essential Linux Commands

- **cat command**

Concatenate or **cat** is one of the most used Linux commands. It lists, combines, and writes file content to the standard output. Here's the syntax:

```
$ cat [files_paths…]
```

`$ cat file.txt` – displays content of '**file.txt**'.
`$ cat file1.txt  file2.txt`– displays content of '**file1.txt**' and '**file2.txt**'.
`$ tac file.txt` – displays content in reverse order.

Metacharacter 1 (Asterisk *)

The asterisk * or wildcard is a favorite when looking for files with the same extension like .jpg or .png.

Example 1:

```
mohamed@univ-guelma-dz:~$ ls m*
```

Lists all files starts with the letter m

Example 2:

```
mohamed@univ-guelma-dz:~$ ls m*.txt
```

Lists all files starts with the letter m and finished by .txt

Metacharacter 2 (Tilde ~)

The **tilde ~** is a quick way to get back to your home directory on a Linux system by entering the following command:

```
$ cd ~
$ pwd
/home/mohamed
```

Metacharacter 3 (Question mark ?)

The **Question mark (?)**: Matches a single character or a pattern occurrence

Example 1:

```
mohamed@univ-guelma-dz:~$ ls m???
```

Lists all files starting with the letter m and consisting of 4 letters

Metacharacter 4 ([ ] Square Brackets)

The **Square Brackets []**: Matches any hyphen-separated number, symbol, or alphabets specified inside the squared brackets

Example 1:

```
mohamed@univ-guelma-dz:~$ ls *[1-9]*
```

Lists all files containing at least one number

Example 2:

```
mohamed@univ-guelma-dz:~$ ls [BI]*
```

using **[BI]*** with the ls command lists all files included in directories that start with **B** or **I**

Metacharacter 4 ({} Brace)

The **brace {}** expansion metacharacter allows you to expand the characters across directories, file names, or other command-line arguments.
For instance, you can make a new directory **brace** inside the **/tmp** folder and create a set of files using the touch command as follows:

Example 1:

```
:~$ mkdir /tmp/brace
:~$ touch test{1,2,3,4,5}
```

you can check if touch created the files or not using the ls command.

```
:~$ ls /tmp/brace
```

Output:
test1 test2 test3 test4 test5

Metacharacters

Character	Meaning	Instance
*	Match 0 or more characters	A*b A and B can have any character of any length, or there can be none, such as AABCB, AXYZB, a012b, AB.
?	Match any one character	A?b A and B must also have only one character, which can be any character, such as AAB, ABB, ACB, a0b.
[List]	Match any single character in the list	A[xyz]b A and B must also have only one character, but only x or Y or Z, such as: AXB, Ayb, Azb.
[!list]	Matches any single character except the list	A[!0-9]b A and B must also have only one character, but not Arabic numerals, such as AXB, AAB, a-a.
[C1-C2]	Match any single word in c1-c2 such as: [0-9] [A-z]	A[0-9]b 0 and 9 must also have only one word such as a0b, a1b ... a9b.
{Str1,Str2,..}	Match Str1 or Str2 (or more) one string	{a,b,c}.{1,2,3} specify multiple lists to generate file names based on the combinations of the elements in the two lists. a.1, a.2, a.3, b.1, b.2, b.3, c.1, c.2, c.3.

Login and logout

- To login to the system. Each user has a “connection account” which is associated with a “password”.
- **A known user of the system is therefore:**
  - an account = username + password,
  - a storage space in secondary memory (disk) ➔ ~ (home directory),
  - an interpretation environment ➔ “Shell” (bash under Linux),
- By using the keys **CTRL+ALT+F1**, **CTRL+ ALT+F2**, **CTRL+ALT+F3** … **CTRL+ALT+F8**
- The user has many screens from which he can open many sessions in the same time.

Login and logout

- On Unix-like operating systems, the **login** command begins a new login session on the system.

- The **login** program is used to establish a new session with the system.
- It is normally invoked automatically by responding to the "login:" prompt on the user's terminal.
- Syntax:

```
$ login [username]
```

Login and logout

- On Unix-like operating systems, the **login** command begins a new login session on the system.
- The **login** program is used to establish a new session with the system. It is normally invoked automatically by responding to the "login:" prompt on the user's terminal.

Login and logout

- The **logout** command is a **Linux/UNIX** shell command that performs the task of logging out the logged-in user from the system in that session
- **Syntax:**

```
$ logout
```

Login and logout

- To shutdown the Linux system;
  - type `$sudo shutdown -h now`
  - Then wait for some time and the Linux server will poweroff
- Or type `$ sudo shutdown -h now` to shutdown immediately
- To shutdown system in 10 minutes with the following warning message:

```
$ shutdown -h +10 "Development server is going down for
  maintenance. Please save your work ASAP."
```

- Here is what all users will see on their terminal:

Broadcast message from root@wks01 (pts/0) (Wed Oct 25 09:45:30 2023):

Development server is going down for maintenance. Please save your work ASAP.

The system is going DOWN for system halt in 10 minutes!

Login and logout

- If you want to cancel a pending shutdown command
- Pass the **-c** from another terminal session. For instance:

```
$ sudo shutdown -c
```

- The last reboot or last shutdown command will show a log of all reboots and shutdown since the log file was created under Linux:

```
$ last reboot
```

OR

```
$ last shutdown
$ last -x shutdown
```