Ubuntu Linux Essentials Course

Chapter 1: Introduction to Ubuntu

What is FOSS?

- Free Open Source Software (**FOSS**) provides many freedoms, including the ability to:

- - View source code used to compile programs.
- - Make modifications.
- - Distribute these modifications.

** it does not necessarily mean it is free of charge.*

Where is the benefit?

- Customers are usually willing to pay for training, support, and consultation.

- Most **FOSS** is covered under a public license. The most common public license is the GNU General Public License (**GPL**).

An open-source license

is a type of license for computer software and other products that allows the source code, blueprint or design to be used, modified and/or shared under defined terms and conditions.

Capabilities (Without Application Licensing Restriction)	GPL (Linux)	Dual-GPL (MySQL)	LGPL/ MPL (OpenOffice, Firefox)	Apache/ BSD (Apache, FreeBST)
1) Download	✓	✓	✓	✓
2) Evaluate	✓	✓	✓	✓
3) Deploy	✓	✓	✓	✓
4) Redistribute	⊘ 1	✓ 3	✓	✓
5) Modify	⊘ 2	⊘ 2	⊘ 2	✓ 4

1) Application needs to be licensed under **GPL** if redistributed with the **GPL** asset.

2) Library code modifications need to be licensed under the same license as the originating asset.

3) Usually requires a commercial license from the copyright holder.

4) Although much more permissive than an OSI license, some BSD based licenses, such as Apache V2, still have some copyleft materials.

- UNIX was developed by (the Massachusetts Institute of Technology, AT&T Bell Labs, and General Electric) in the late 1960s as a result of efforts to create a time-sharing operating system called Multics.
- Unix first version created in Bell Labs in 1969.
- "**epoch date**" is the date when the time started for Unix computers.

Why Linux?

- Linux is growing in the home users sector and the dominant of the professional and servers sector.

- Internet service providers (**ISPs**), e-commerce sites, and other commercial applications all use Linux today and continue to increase their commitment to Linux.

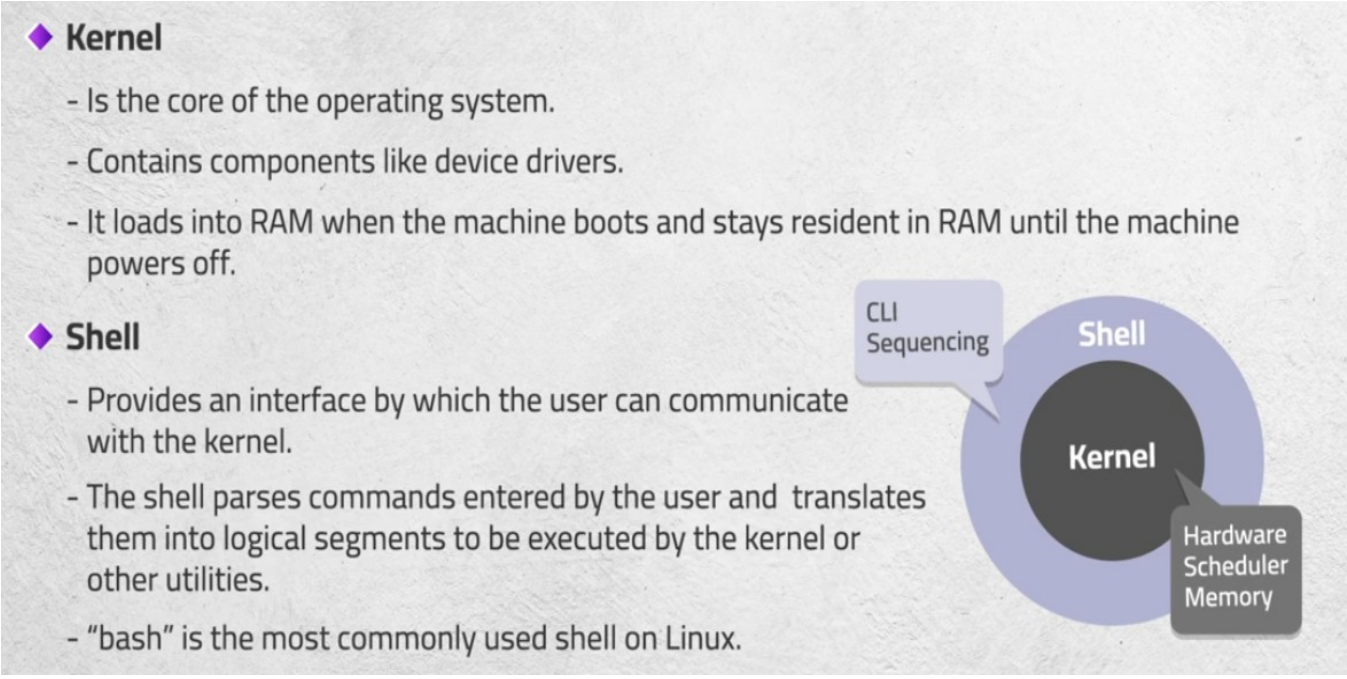
- Fast

-User-Friendly

-Secured

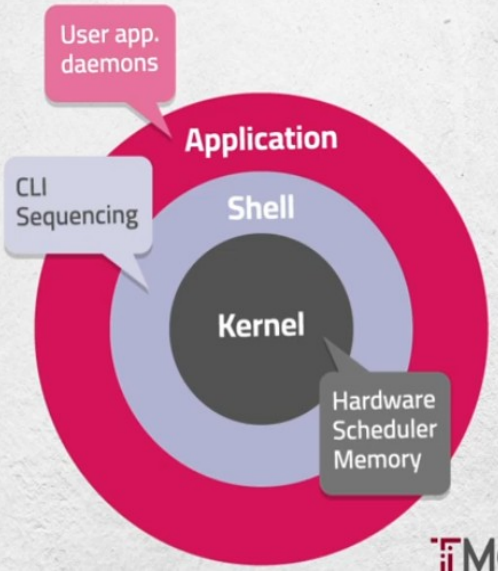
-Provide a high-performance environment with over 600 active distros.

Linux Components



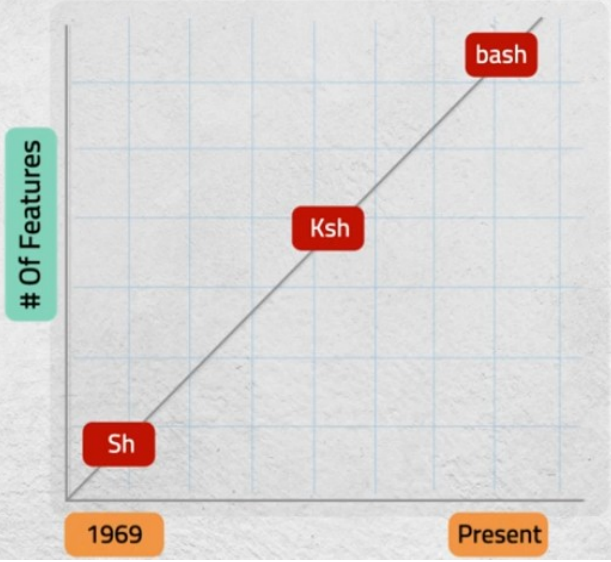
Terminal

- Gives the shell a place to accept typed commands and to display their results



Command-Line Shells

There are lot of shells as: Bourn Shell (sh), Korn Shell (ksh), C Shell (csh) and Bourn Again Shell (bash). They have different features that will be discussed later.



Chapter 2: Getting Started With Ubuntu Desktop

* shortcut to open terminal: ***ctrl+alt+t***

Basic Commands

uname *--> can provide various details about the operating system and hardware.*

uname *--> returns kernel name. (ex: Linux)*

uname -n *--> returns host name. (ex: MohamedPC)*

uname -o *--> returns operating system name. (ex: GNI/Linux)*

---

cal *--> returns calendar of current month.*

date *--> returns current date and time.*

man --> added before any command to return 'manual' of this command. ex: man uname

---

cal 12 2022; date; cal 10 2010 --> executing multiple commands in order

---

ctrl + c --> interrupts a command

ctrl + d --> terminate login session

---

man -k [keyword]

The man -k command in Unix-like systems searches the manual page database for entries related to a given keyword. The keyword can be part of the title or description of the manual pages.

Example: **man -k passwd** --> return a list of manual pages related to password management. The output could look something like this:

```
passwd (1)              - change user password
chpasswd (8)            - update passwords in batch
passwd (5)              - password file format
passwd (8)              - passwd database
```

The numbers in parentheses represent the section of the manual where the entry is found:

- **1**: User commands
- **2**: System calls
- **3**: Library calls
- **4**: Special files (e.g., devices)
- **5**: File formats and conventions
- **6**: Games
- **7**: Miscellaneous
- **8**: System administration commands

man 5 passwd will show the manual page from section 5.

man 1 passwd will show the manual page from section 8.

whatis [command]

The **whatis** command in Unix-like operating systems is used to display a brief description of a command or program. It provides a one-line summary from the manual pages (man pages) of the specified command. This can be useful for quickly understanding what a command does without having to read the entire manual page.

bash Copy code

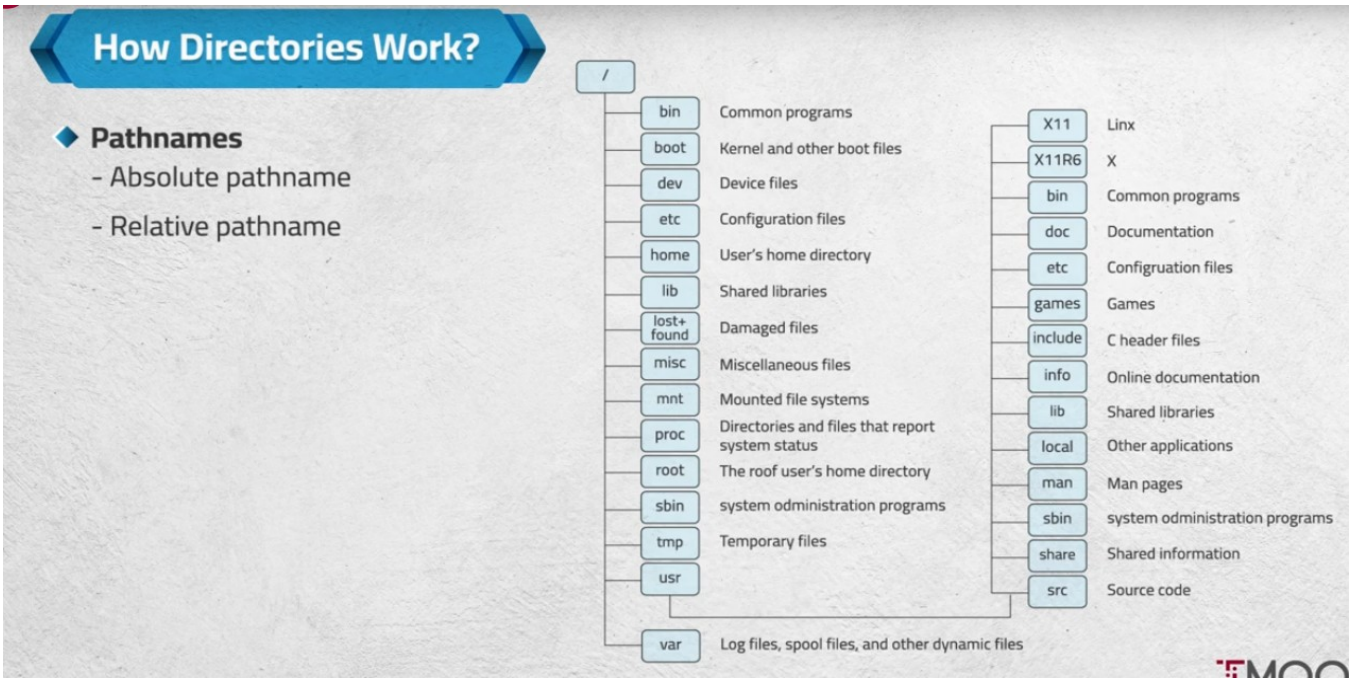
```
whatis ssh
```

Output:

scss Copy code

```
ssh (1)                 - OpenSSH SSH client (remote login program)
ssh-add (1)             - adds private key identities to the authentication ag
ssh-agent (1)           - opens a new SSH agent
ssh-keygen (1)          - generate, manage and convert SSH keys
```

Chapter 3: File and Directory Basics



PWD --> Print working directory: returns the current directory

---

mkdir [abs path or rel path] --> creates folder

ex:

mkdir /home/drogo/Desktop/newfolder (abs)

mkdir newfolder (rel) creates folder here

TOUCH: Creates new file

touch [filename]

touch file1

touch dir1/file1

cd - --> returns to previous place**

cd .. --> one level up

ls -a --> Shows hidden files and unhidden files

ls -F --> Appends '/' after directories and '*' after executables, '@' after shortcuts (soft links) ...etc.

u can use ***ls -aF*** at once

---

ln -s [/path/to/original] [/home/username/Desktop/linkname]

creates a shortcut

---

ls -l [directory]

`*[directory]`: Optional. If specified, lists files in the given directory. If omitted, it lists files in the current directory.

ls -l → return list with content, types, and permissions

Example Output

bash Copy code

```
-rw-r--r-- 1 username groupname 4096 Aug  4 11:55 file1.txt
drwxr-xr-x 2 username groupname 4096 Aug  4 11:55 directory1
lrwxrwxrwx 1 username groupname   11 Aug  4 11:55 symlink -> targetfile.txt
```

Output Explanation

The output of ls -l typically includes the following columns:

1- File Permissions

Example: -rw-r--r--

Shows the permissions for the file or directory. The first character indicates the type (- for a regular file, d for a directory, l for a symbolic link, etc.). The following characters indicate permissions for the owner, group, and others.

2- Number of Hard Links

Example: 1

Indicates the number of hard links to the file or directory. For directories, this includes the number of subdirectories (including . and ..).

3- Owner

Example: username

The name of the user who owns the file or directory.

4- Group

Example: groupname

The name of the group that owns the file or directory.

5- File Size

Example: 4096

The size of the file or directory in bytes.

6- Modification Date and Time

Example: Aug 4 11:55

The date and time when the file or directory was last modified.

7- File/Directory Name

Example: file1.txt

The name of the file or directory.

---

ls -l Desktop/dir1 : returns long format of what is inside dir1

if i want to see long format of dir1 itself:

ls -ld Desktop/dir1

---

df [options]

-h: This option stands for "human-readable". It formats the sizes in a more readable form using units like KB (kilobytes), MB (megabytes), GB (gigabytes), etc.

The df -h command is a convenient way to check disk space usage in a format that's easy to read and understand. It helps users and administrators quickly assess how much disk space is used and available across various file systems and mount points on the system.

Example Output

```bash
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       100G   45G   50G  48% /
tmpfs            16G  1.1G   15G   7% /dev/shm
/dev/sdb1       500G  300G  175G  64% /mnt/data
```

du [options] [path]

The du command in Unix-like operating systems stands for "disk usage" and is used to estimate and report the amount of disk space used by files and directories. Unlike df, which provides information about disk space usage for entire file systems, du focuses on the space consumed by individual files and directories.

Example Output

```bash
4.0K    ./dir1
12M     ./dir2
2.1G    ./file1
8.0K    ./file2
```

du -hs /boot

show me disk usage in Human-readable way [sum only]

---

to show file content :

cat [path] --> will show all content

if file has a lot of things use ***more [path]*** and scroll by using enter for line by line or space for rest of list at once.

---

head [path] : lists first 10 lines of file content

head -n [path]: lists first n lines of file content

tail [path]: lists last 10 lines of file content

tail -n [path]: lists last n lines of file content

File globing

* means 0 or more char except leading .

ls f* : list any files that starts with f

ls *3 : list any files that ends with 3

? means any single char except the leading .

so:

ls file? -->list any thing that has the word file followed by any char ex: file4 file1 file2

---

square brackets means range:

ls [a-f]* lists any file starts with a,b,c,d,e,f followed by any number of chars

ls [pf]* : lists any file starts with p or f followed by any number of chars

---

copying files and directories :

cp [options] source(s) target

-i prevents you from accidentally overwriting existing files or directories

-r copy a directory including the contents of all sub-directories

moving and renaming files and directories:

mv [options] source(s) target

-i :prevents you from accidentally overwriting existing files or directories

---

to remove files (deleted forever):

rm [-i] file(s)_name

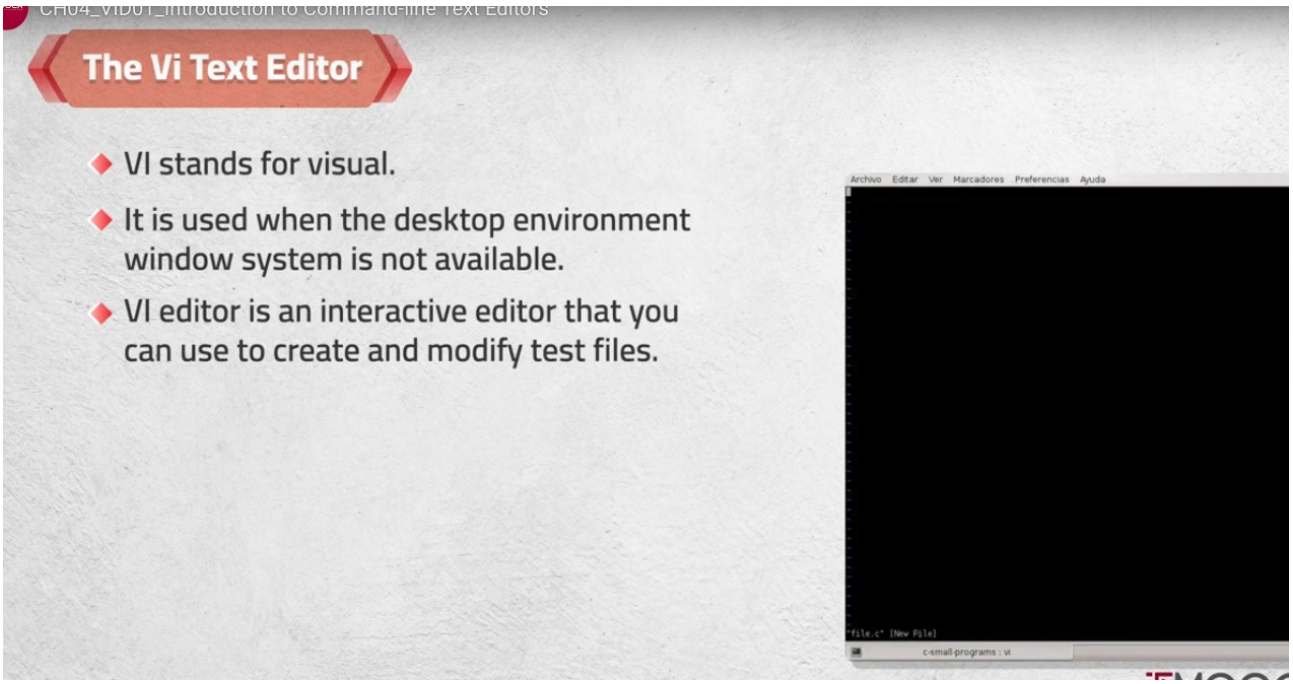
to remove directories

rmdir dir(s)_name --> remove an empty directory

rm [-r] dir(s)_name -->remove whole directory with its content

Chapter 4: Manipulating Files with Vi

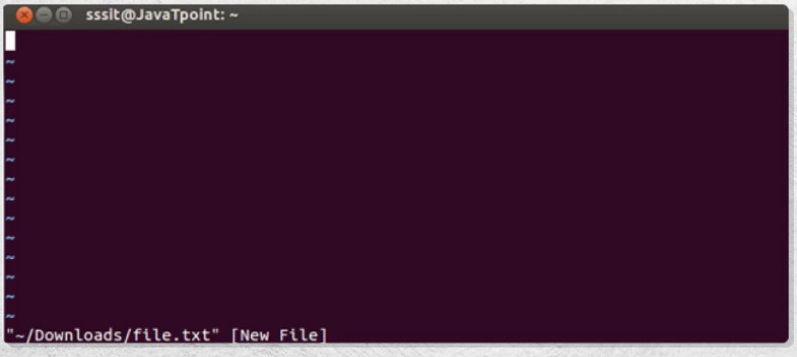
The vi (visual editor) editor is a widely used, powerful text editor available on Unix-like operating systems, including Linux and macOS. It stands out for its efficiency, flexibility, and minimalistic design. Despite its learning curve, many users prefer vi (or its enhance version, vim) for its speed and the control it offers over text manipulation.



There is another editor called ***gedit***. It is very similar to normal notepad:

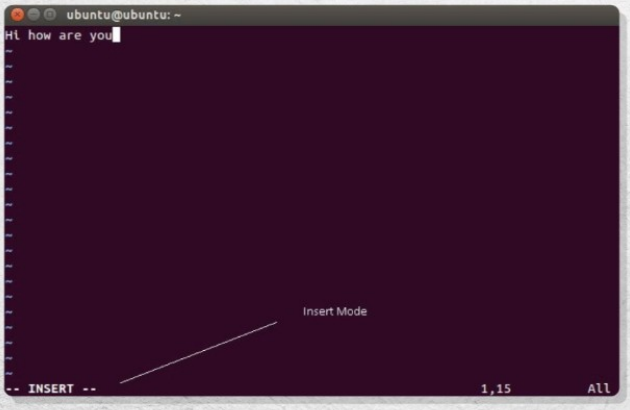
VI three basic modes

- Command mode
  - Default mode
  - Perform commands to delete, copy, ...



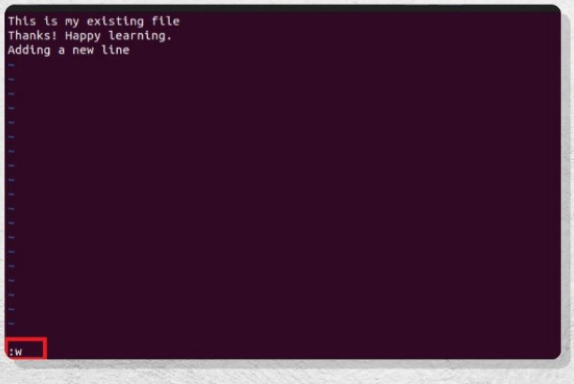
VI three basic modes

- Edit mode
  - Enter text into the file



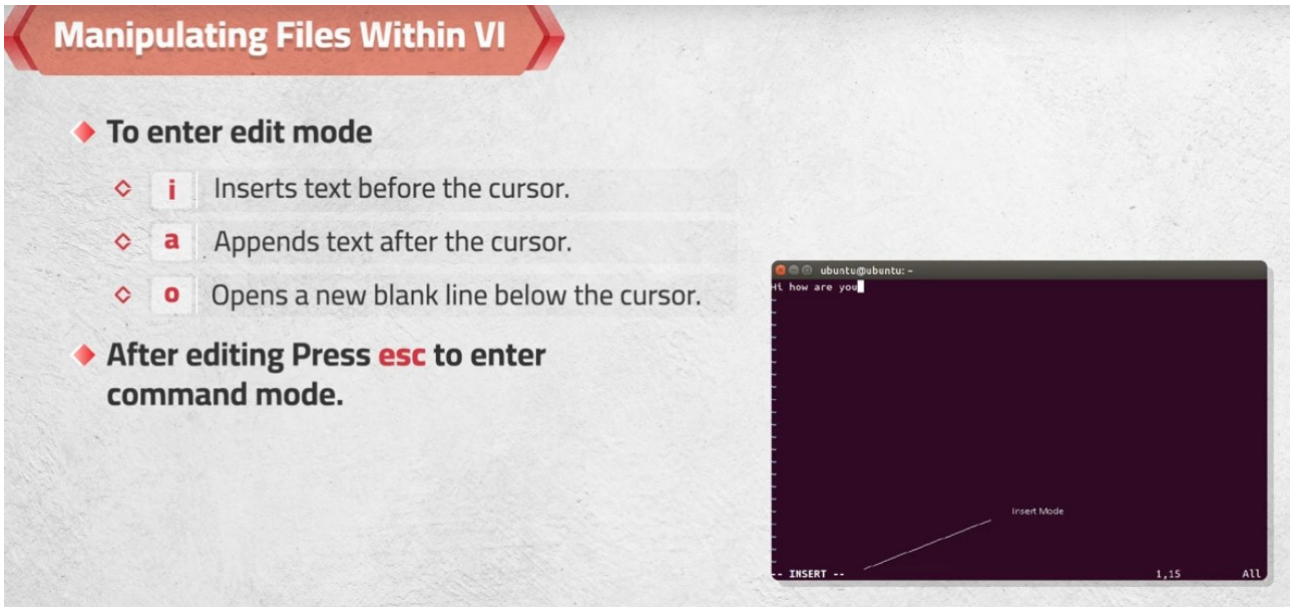
VI three basic modes

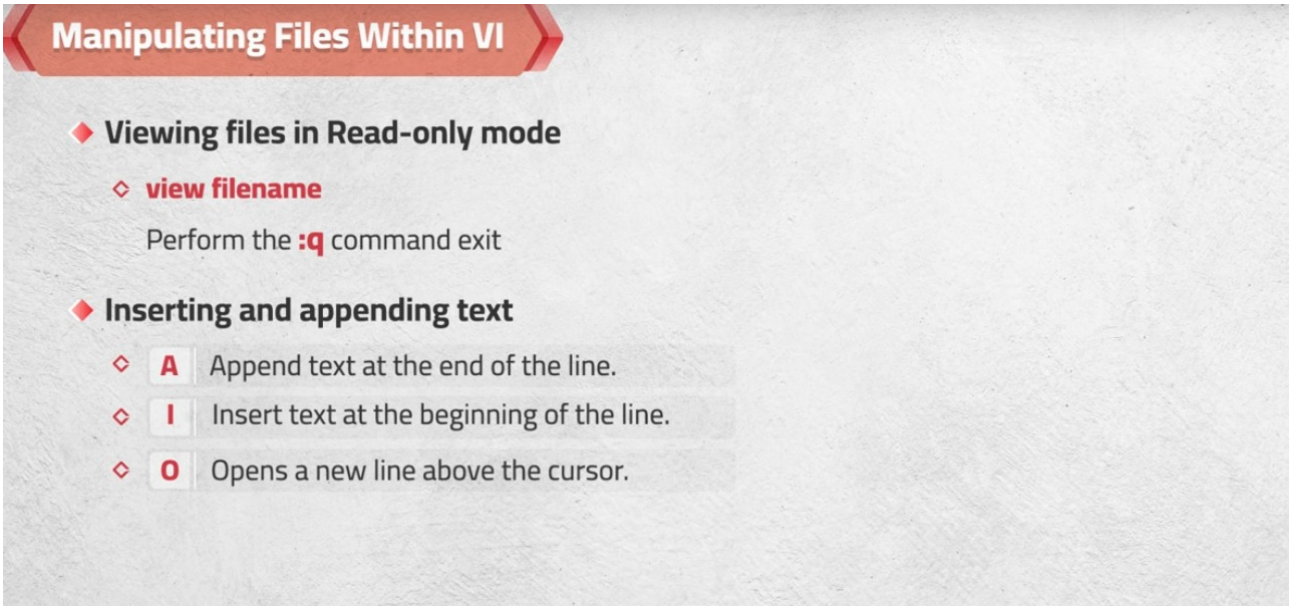
- Last line mode
  - Advanced editing commands
  - To access it, enter a colon (:) while in the command mode



vi test → will open new file in vi editor

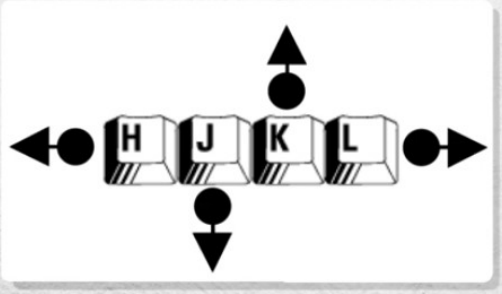
if I click any letter nothing happens, I have to switch to insert mode by hitting “I” and start writing.





Moving the cursor within the vi

- **h** Left arrow, or backspace: left one character.
- **j** Down arrow: down one line.
- **k** Up arrow: up one line.
- **l** Right arrow or space: right one character.



Moving the cursor within the vi

- **w** Forward one word.
- **b** Back one word.
- **e** to the end of the current word.
- **0** To the beginning of the line.
- **Enter** Down to the beginning of the next line.

Moving the cursor within the vi

- **nG** Goes to Line n.
- **G** Goes to the last line of the file.
- **:n** Goes to Line n.
- **Control-B** Pages back one screen.
- **Control-F** Pages forward one screen.
- **Control-L** Refresh the screen.

Substitute and delete text

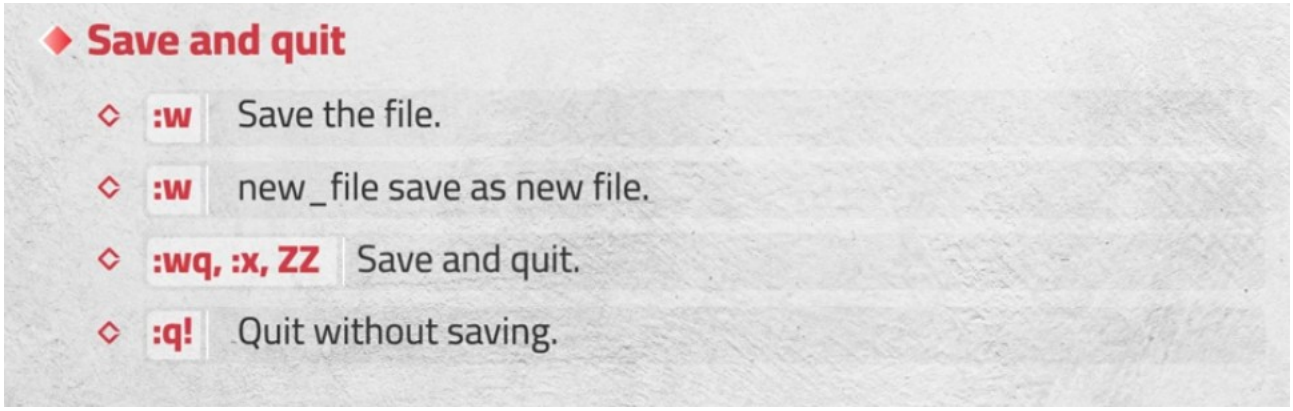
- **s** Substitutes a string for a character at the cursor.
- **x** Deletes a character at the cursor.
- **dw** Deletes a word or part of the word to the right of the cursor.
- **dd** Deletes the line containing the cursor.
- **D** Deletes the line from the cursor to the right end of the line.
- **n,nd** Deletes Lines n through n.

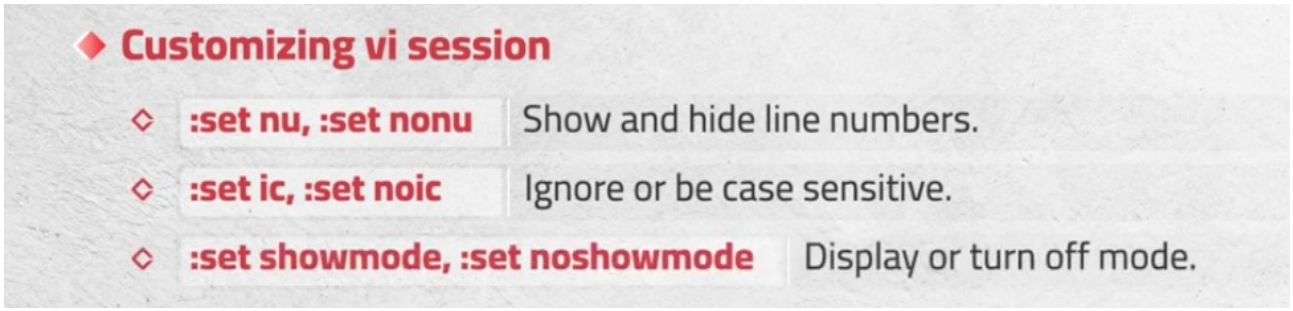
Search and replace

- **/string** Searches forward for the string.
- **?string** Searches backward for the string.
- **n** Searches for the next occurrence of the string.
- **N** Searches for the previous occurrence of the string.
- **%s/old/new/g** Searches for the old string and replaces it with the new string globally.

Copy and Paste

- **yy** Yank a copy of a line.
- **p** Put yanked text under the line containing the cursor.
- **P** Put yanked text before the line containing the cursor.
- **n,n co n** Copy Lines n though n and puts them after Line n.
- **n,n m n** Move Lines n through n to Line n.

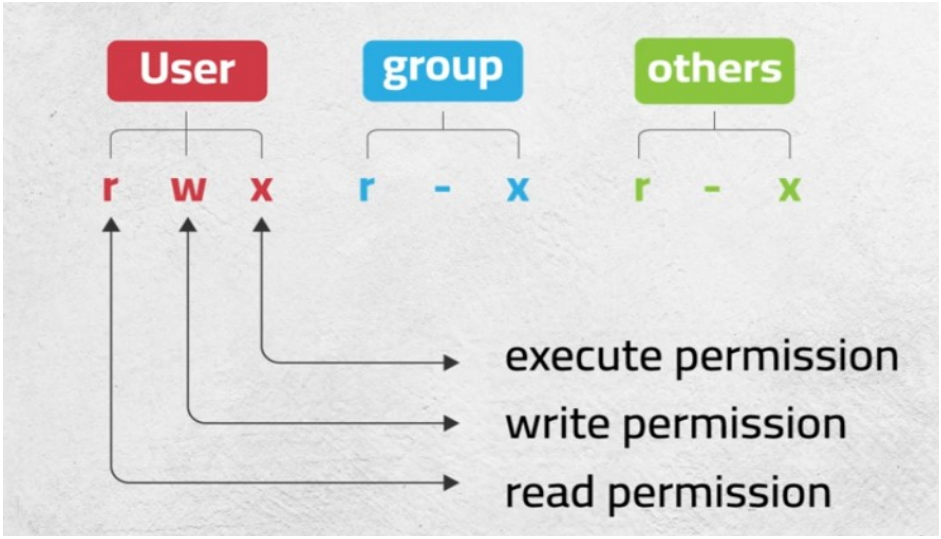




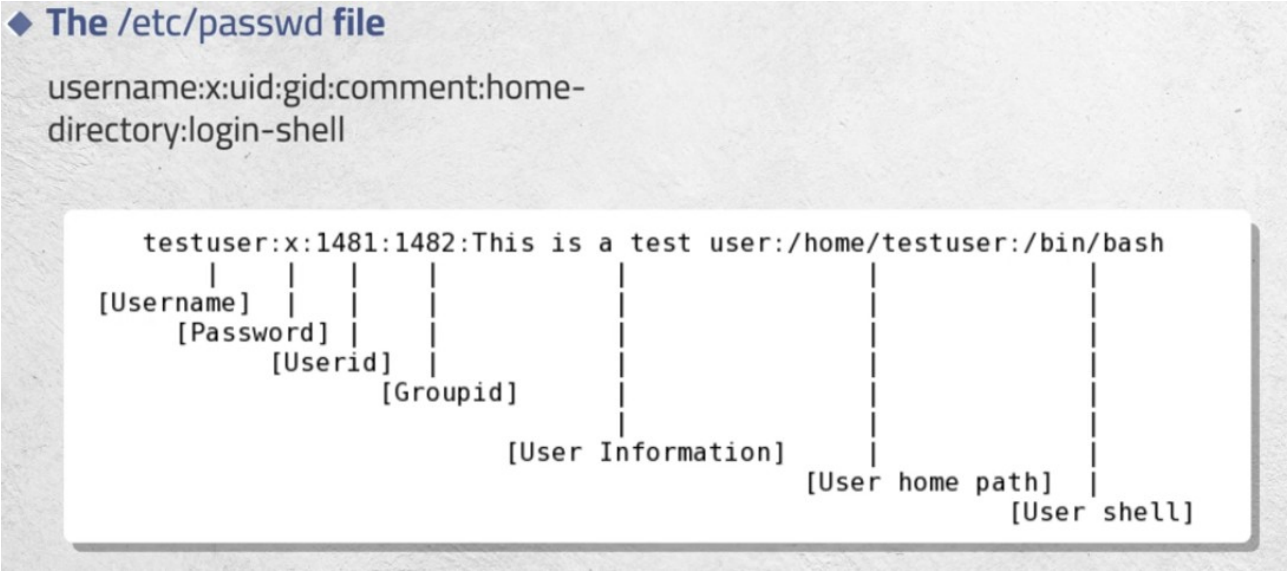
Chapter 5: User and group administration

- Any user created will be inside a group by default, usually if no group specified, Linux will create a private group for this user and put it into.

- Groups are useful for specifying permissions and controlling users.



User and groups databases , such as this file: **/etc/passwd**



- The /etc/shadow file

*passwords were firstly stored in */etc/passwd* but then Linux replaced them with x placeholder as they can be exposed by any user through this */etc/passwd* file and put them in the */etc/shadow* file which only the root user can access it.

The `/etc/shadow` file on Unix-like operating systems is a system file that contains encrypted user password information and other related account details. It's used in conjunction with the `/etc/passwd` file, which holds basic user account information but does not include encrypted passwords.

Here’s a brief overview of its structure and content:

- **Username**: The login name of the user.
- **Encrypted Password**: The hashed password, which is usually stored in an encrypted form. If the field contains a special character like `*` or `!`, it typically means the account is locked or disabled.
- **Last Password Change**: The date of the last password change, usually expressed as the number of days since January 1, 1970.
- **Minimum Password Age**: The minimum number of days required between password changes.
- **Maximum Password Age**: The maximum number of days the password is valid before requiring a change.
- **Warning Period**: The number of days before password expiration that the user will be warned.
- **Inactive Period**: The number of days after password expiration before the account is disabled.
- **Account Expiration Date**: The date on which the user account will expire, given as the number of days since January 1, 1970.

Because `/etc/shadow` contains sensitive information, it is typically readable only by the root user and certain privileged system processes. The file is often used by the system’s authentication mechanisms to verify user passwords during login.

The **/etc/group** file

groupname:×:gid:comma-separated list of group members

The **/etc/gshadow file**

groupname:×:gid:comma-separated list of group members

- same syntax as group file but the password is either shown as encrypted password or empty **.**

- Adding new user account

sudo useradd user1

- will create user with defaults for uid, group,...etc

if you want to specify these fields:

sudo useradd -u <userid> -g <user primary group> -c <comment> -md <home directory> -s <shell> username

sudo useradd -u 1005 -g user1 -c "user2 HR dept office 1010" -md /home/user2 -s /bin/bash user2

sudo: This command is used to execute the following command (useradd) with superuser (root) privileges. It's necessary because creating new users typically requires administrative permissions.

1. `useradd`: This is the command used to add a new user to the system.
2. `-u 1005`: Specifies the UID (User ID) for the new user `user2`. In this case, `1005` is the UID assigned to `user2`. UID `1005` should be unique and not already in use by another user on the system.
3. `-g user1`: Specifies the initial login group (`user1`) for the new user `user2`. This means `user2` will be a member of the group `user1` upon creation.
4. `-c "user2 HR dept office 1010"`: This option provides a comment or description about the user. The comment `"user2 HR dept office 1010"` typically includes additional information about the user, such as their name, department, and office location.
5. `-md /home/user2`: This specifies the home directory (`/home/user2`) for the new user `user2` and also ensures that the directory is created if it doesn't already exist (`-m` option). Here, `-m` is implicit when specifying `-d`.
6. `-s /bin/bash`: Specifies the login shell (`/bin/bash`) for the new user `user2`. The login shell determines the command interpreter that the user will use when they log into the system.
7. `user2`: Finally, `user2` is the username of the new user being created.

Summary

- **Purpose**: The command `sudo useradd` is used to add a new user (`user2`) to the system with specific attributes such as UID, initial group, comment, home directory, and login shell.
- **User Configuration**:
  - **UID (`-u`)**: Unique identifier for the user.
  - **Initial Group (`-g`)**: The group `user2` will belong to initially (`user1` in this case).
  - **Comment (`-c`)**: Additional information about the user (`"user2 HR dept office 1010"`).
  - **Home Directory (`-md`)**: Specifies the home directory path (`/home/user2`) and ensures it is created (`-m`).
  - **Login Shell (`-s`)**: Specifies the shell (`/bin/bash`) used for interactive sessions.

This command ensures that `user2` is created with the specified settings and can start using their account on the system.

***To check if user is added

tail -1 /etc/passwd → shows the last line of passwd file

tail -1 /etc/group → shows the last line of group file

---

su – user2 → switch to user2 the dash mean start at his home dir

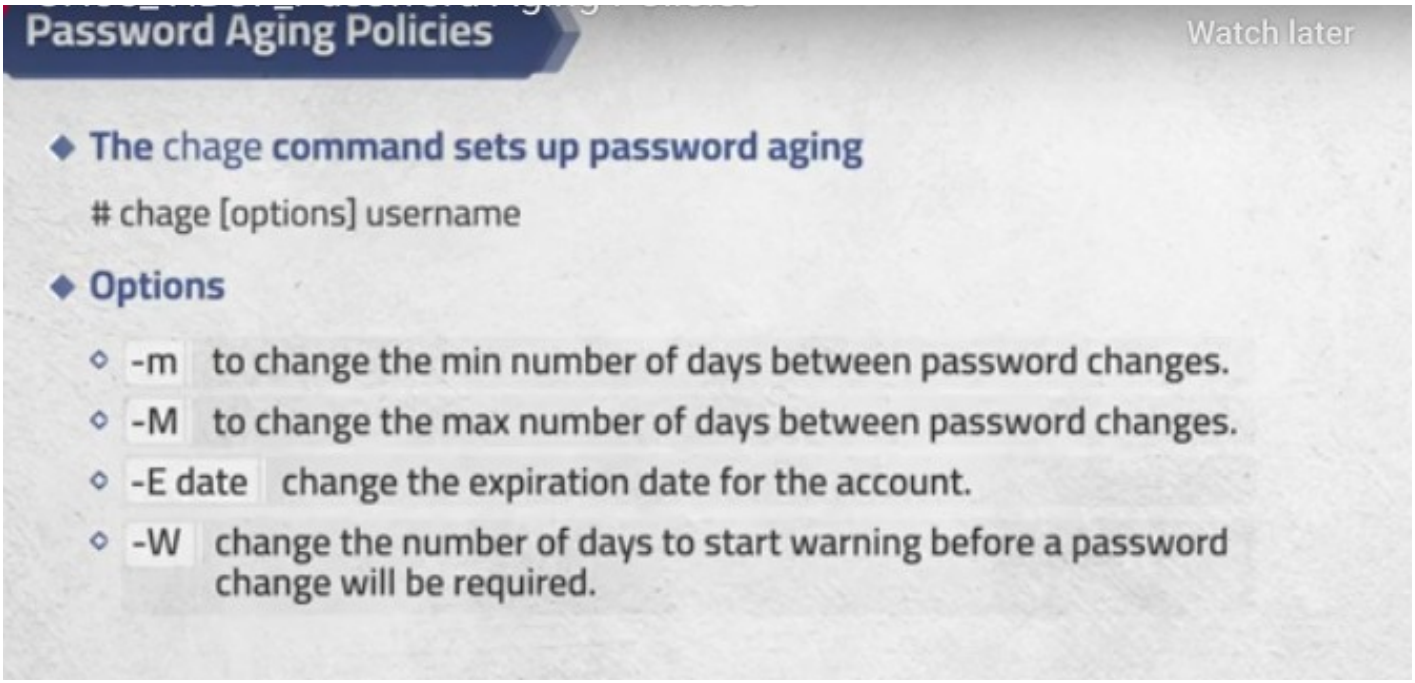
sudo password user3 → sets the password for user3

id username→ returns the uid and gid and group name

sudo usermod -aG avahi user3 → add new secondary group (avahi) to (user3)

userdel user3 → *deletes user3 without deleting its home dir*

userdel -r user3 → *deletes user3 with its home dir*



example:

sudo chage -M 90 username → This command sets the maximum password age (-M) to 90 days for the user username.

sudo chage -l username → To view the current password aging information for a user, you can use the chage command without any options

```
drogo@drogoPC:~$ sudo chage -l user3
Last password change                                    : Aug 07, 2024
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

Managing Groups

sudo groupadd open-source → creates new group named open-soruce

sudo groupadd -r iti3 → the -r give the group id in different range of usersids (converting group to system group instead of regular group).

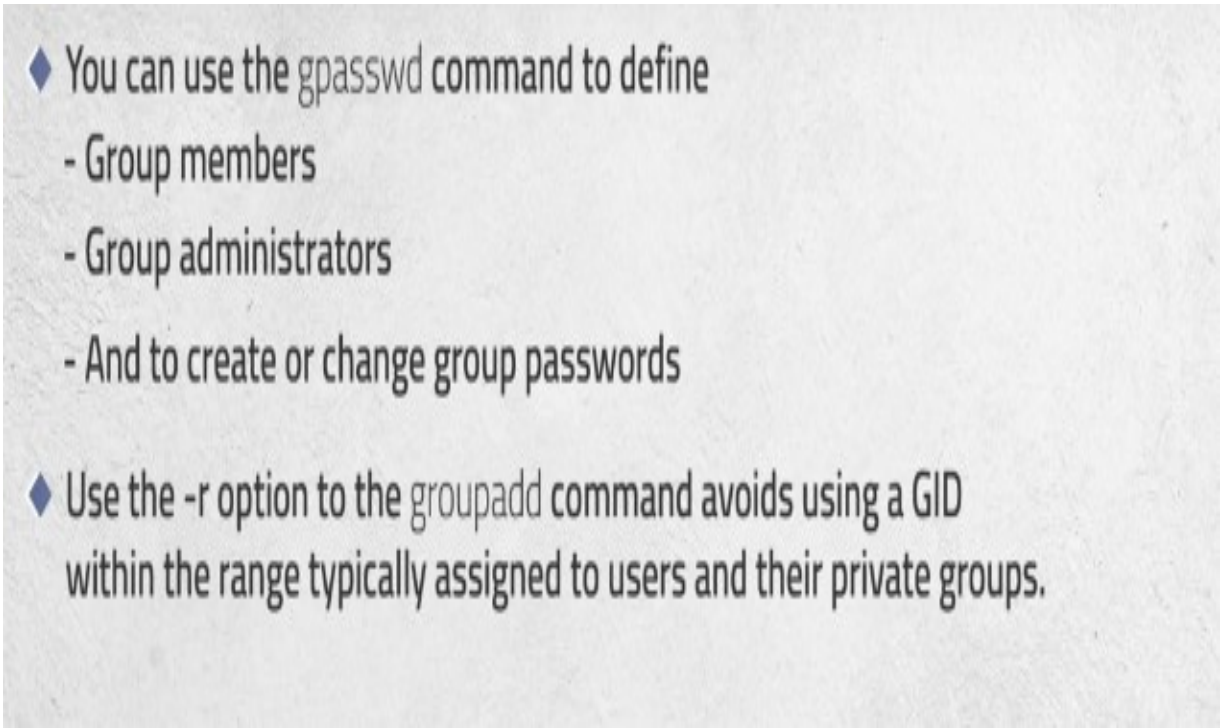
find / ~nogroup → This command searches the entire filesystem (/) for files and directories that have no associated group.

sudo groupmod -n os open-source → renames the group to be os

sudo groupdel user1 → deletes group named user1

** cannot remove group if it is primary for a user

```
drogo@drogoPC:~$ sudo groupdel user1
groupdel: cannot remove the primary group of user 'user1'
```



chgrp <new_group_name> <file_name> → To change the group of a file.

newgrp dip → open a session with setting the active group 'dip'; so that any file that will be created during this session will be in group 'dip', you can exit session using (exit).

Groups → *returns all the groups of the current user.*

Using Sudo Command

- Sudo access is controlled by /etc/sudoers – this file is edited by ***visudo, an editor with syntax checker.***

** To give a specific group of users limited root privileges (which commands he can use or execute)

sudo visudo → to open the visudo and edit the *sudoers* file

**inside the visudo:

Cmnd_Alias USER = /sbin/useradd, /sbin/usermod, /bin/passwd →

defines a command alias named USER. Let's break down what this means:

Explanation:

1. **Cmnd_Alias**:
    - `Cmnd_Alias` is used in the sudoers file to create an alias for one or more commands. It simplifies the management of command permissions by grouping related commands under a single alias.

2. **Alias Name (USER)**:
    - USER is the alias name defined in this line. You can choose any descriptive name for the alias that makes sense within your organization or system administration context.

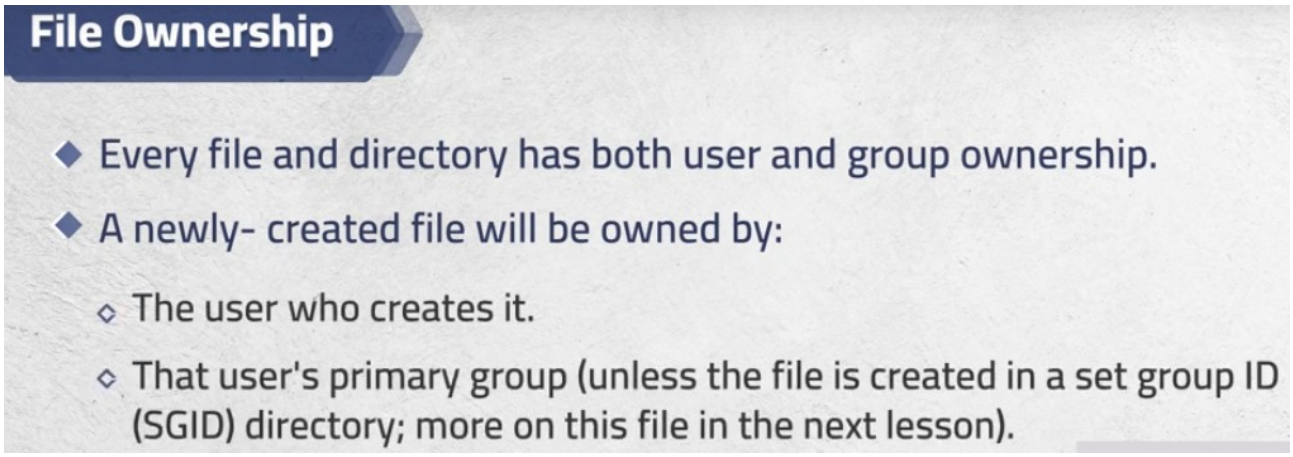
3. **Command List**:
    - Following the alias name (USER), there is a list of commands that belong to this alias, separated by commas.
    - In this case, the commands included in the USER alias are:
        - `/sbin/useradd`: This command is used to create new user accounts.
        - `/sbin/usermod`: This command is used to modify existing user accounts.
        - `/bin/passwd`: This command is used to change a user's password.

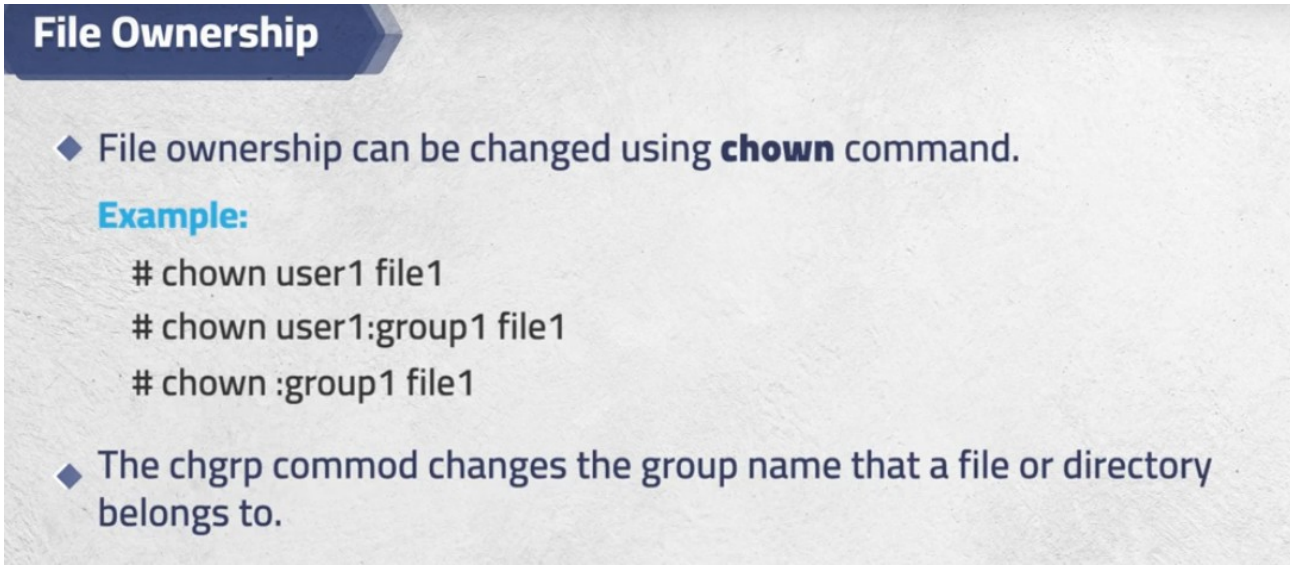
Usage Example:

- Once USER alias is defined in the sudoers file, you can assign permissions to users or groups to execute these commands with elevated privileges using `sudo`.
- For example, if you want to allow a user or group to manage user accounts (create, modify, and change passwords), you can grant them access to the USER alias in the sudoers file:

username ALL=(ALL) USER → This line allows the user username to run any command listed in the USER alias (/sbin/useradd, /sbin/usermod, /bin/passwd) with elevated privileges (sudo).

Chapter 6: File Ownership and Permissions





chown user1 file1 → changes the owner of file1 to be user1

chown user1:group1 file1 → changes the owner and group of file1

chown :group1 file1 → changes the owner group of file1 to group1

the ***setuid*** permission:

type of permission in Unix-like operating systems that allows a program to run with the privileges of the file's owner rather than the privileges of the user who executes the program. This can be crucial for certain operations that need elevated privileges to perform tasks that a regular user cannot.

How Setuid Works

When the setuid bit is set on an executable file, it changes the behavior of that executable:

1- Execution with Elevated Privileges: When a user executes the file, the process runs with the permissions of the file's owner (usually root), not the permissions of the user who initiated the process. This is useful for programs that need to perform actions that require higher privileges than those available to the regular user.

2- Permissions String: In the output of the ls -l command, the setuid bit is represented by an s in the user's execute position. For example:

-rwsr-xr-x 1 root root 12345 Aug 9 12:34 file

Here, rws indicates that the setuid bit is set (the s replaces the x in the owner's permissions).

Example Use Case

/usr/bin/passwd: *The passwd command is a common example of a setuid program. It needs to modify system files (like /etc/shadow) that are owned by root. By running with root privileges, it can update these files to change a user's password.*

Setting and Removing Setuid:

To set the setuid bit:

chmod u+s file

To remove the setuid bit:

chmod u-s file

Setting and removing the setuid bit require superuser (root) permissions for files owned by other users.

The chmod command:

The **chmod** command is used to change the permissions of files or directories. Permissions determine who can read, write, or execute a file.

Understanding Permissions

Permissions are usually represented in three parts:

1- User (Owner)

2- Group

3- Others

Each part can have the following permissions:

- Read (r): Permission to read the file or list the directory.
- Write (w): Permission to modify the file or add/remove files in the directory.
- Execute (x): Permission to execute the file (for scripts or programs) or access the directory.

Using chmod

There are two main ways to use chmod: **symbolic** mode and **numeric** (octal) mode.

1- Symbolic Mode

In symbolic mode, you use letters to set or modify permissions:

- u stands for user (owner)
- g stands for group
- o stands for others
- a stands for all (user, group, others)

Operators:

- + adds a permission
- - removes a permission
- = sets the permission exactly

Examples:

chmod g+r file →Add read permission for the group

chmod o-x file →Remove execute permission for others

chmod u=rw,o= file → Set read and write permissions for the user and remove all permissions for others

2- Numeric (Octal) Mode:

In numeric mode, you use numbers to set permissions. Each permission is represented by a digit:

- Read: 4
- Write: 2
- Execute: 1

You add these numbers to set the permissions. Each part (user, group, others) is represented by a digit.

Examples:

chmod 644 file ***→Set read and write permissions for the user, and read permissions for the group and others***

**Here, 6 (4+2) gives read and write permissions to the user, and 4 gives read-only permissions to the group and others.

chmod 755 file **→ Set read, write, and execute permissions for the user, and read and execute permissions for the group and others**

** Here, 7 (4+2+1) gives full permissions to the user, and 5 (4+1) gives read and execute permissions to the group and others.

Recursive Changes:

To change permissions recursively for directories and their contents, use the -R option:

chmod -R 755 directory → This sets read, write, and execute permissions for the owner, and read and execute permissions for the group and others for the directory and all its files and subdirectories.

Note:

if you apply chmod u+s file2, and this file is not executable it will capital s (S) in permission which indicates something is wrong. The file should be executable to apply chmode u+s file2.

Chmod a+x file2 → grants execute permissions to everyone for the file file2

chmod 4775 file3 →

The command chmod 4775 file3 sets the permissions on file3 using numeric (octal) mode. Let's break down what this command does:

Numeric Mode Breakdown

In numeric mode, file permissions are represented by a three-digit number, where each digit specifies permissions for the user (owner), group, and others, respectively. Each digit is a sum of the following values:

- Read: 4
- Write: 2
- Execute: 1

4775 Explanation

- 4: This is the first digit and it represents the setuid bit (4) combined with the user’s permissions.
- 4 sets the setuid bit, which makes the file run with the privileges of the file's owner when executed.
- 7: This is the second digit and represents the permissions for the group. 7 corresponds to read (4), write (2), and execute (1) permissions, so the group has full permissions.
- 5: This is the third digit and represents the permissions for others. 5 corresponds to read (4) and execute (1) permissions, so others have read and execute permissions.

Resulting Permissions

When you apply chmod 4775 file3, the resulting permissions are:

- Setuid Bit: The file will have the setuid bit set, so when the file is executed, it will run with the privileges of the file's owner.
- Owner: The owner will have read, write, and execute permissions (rwx).
- Group: The group will have read, write, and execute permissions (rwx).
- Others: Others will have read and execute permissions (r-x).

Example ls -l Output

After running chmod 4775 file3, the permissions might look like this:

-rwsrwxr-x 1 owner group 12345 Aug 9 12:34 file3

Here's what this indicates:

rws: The owner has read, write, and execute permissions, with the s indicating the setuid bit is set.

rwx: The group has read, write, and execute permissions.

r-x: Others have read and execute permissions.

Summary

The command chmod 4775 file3 sets the following permissions for file3:

Setuid bit: Enabled

User (owner): Read, write, and execute

Group: Read, write, and execute

Others: Read and execute

Be cautious when using the setuid bit due to potential security implications, as it allows a file to be executed with the privileges of its owner.

Sticky Bit

In Linux, the "sticky bit" is a permission setting that can be applied to directories. Its primary function is to control file deletion within that directory. When the sticky bit is set on a directory, only the file's owner, the directory's owner, or the root user can delete or rename the file within that directory. Other users are restricted from removing or renaming files they do not own, even if they have write permissions on the directory.

How the Sticky Bit Works:

Without Sticky Bit: Users with write permissions on a directory can delete or rename any file in that directory, regardless of ownership.

With Sticky Bit: Only the file owner, directory owner, or root user can delete or rename the file.

Setting the Sticky Bit:

To set the sticky bit on a directory, you use the chmod command with the +t option. For example:

chmod +t /path/to/directory

Checking the Sticky Bit:

You can check if a directory has the sticky bit set by using the ls -l command. Directories with the sticky bit will have a t at the end of their permissions. For example:

drwxrwxrwt 2 user group 4096 Aug 14 10:00 /path/to/directory

Here, the t at the end of the permissions (rwxrwxrwt) indicates that the sticky bit is set.

Common Use Cases:

Temporary Directories: The sticky bit is often used on directories like /tmp, which are used by many users for temporary file storage. Setting the sticky bit ensures that users cannot delete or modify each other's files.

Shared Directories: In shared directories where multiple users need to create files but should not delete or rename others' files, the sticky bit helps manage permissions effectively.

By setting the sticky bit, you add an extra layer of protection against unwanted file deletion or modification in shared environments.

Default Permissions:

In Linux, default permissions and the umask command control the permissions set on new files and directories.

Files: Typically, new files are created with permissions rw-rw-r-- (664), meaning the owner and group have read and write permissions, and others have read-only permissions.

Directories: New directories usually have permissions rwxrwxr-x (775), meaning the owner and group have read, write, and execute permissions, while others have read and execute permissions.

Umask Command:

Purpose: The umask command sets default permissions for newly created files and directories by specifying which permissions to remove.

Syntax: umask [value]

The value is a three-digit octal number where each digit represents permissions to mask for user, group, and others, respectively.

How Umask Works:

Default Value: The default umask value is often 022, which means new files will have permissions 644 (666 - 022) and new directories will have 755 (777 - 022).

022 masks out the write permissions for group and others.

Umask → *Check Umask Value*

umask 022 → *To set a umask that gives new files rw-r--r-- (644) and directories rwxr-xr-x (755)*

Set Umask Temporarily → *This would result in new files having rw-r----- (640) and directories having rwxr-x--- (750).*

Chapter 7: Shutting Down / Rebooting The System:

Virtual Consoles: usually used when pc freezes to open terminal

Virtual consoles in Linux are a way to provide multiple, independent terminal sessions within a single physical or virtual machine. They allow you to switch between different command-line environments without needing to open multiple terminal windows or sessions.

What Are Virtual Consoles?

Definition: Virtual consoles (or virtual terminals) are text-based interfaces that you can access from a single physical or virtual machine. They provide different terminal sessions that you can switch between.

Purpose: **They are useful for managing multiple tasks, troubleshooting issues, or working in different environments without requiring a graphical user interface.**

Accessing Virtual Consoles

Switching Between Consoles:

On most Linux systems, you can switch between virtual consoles using the Ctrl + Alt + F1 through F6 keys. Each function key corresponds to a different virtual console. For example:

Ctrl + Alt + F1: Virtual Console 1

Ctrl + Alt + F2: Virtual Console 2

and so on up to F6.

If you're in a graphical environment (like GNOME or KDE), you might find that F7 or F8 is used for your graphical user interface.

Login to Virtual Consoles:

When you switch to a virtual console, you will be presented with a login prompt. You can log in using your user credentials. Each virtual console operates independently, so you can have different users logged in to different consoles.

Returning to the Graphical Environment:

If you want to return to your graphical environment, you can use the Ctrl + Alt + F7 (or sometimes F8) keys, depending on your distribution and configuration.

System Shutdown

- **It only requires reboot or shutdown when you need to:**
  - Add or remove hardware
  - Upgrade to a new version of Ubuntu
  - Or upgrade your kernel
- **System shutdown commands**
  - init 0
  - poweroff
  - shutdown -h time **# Halt after shutdown**
  - shutdown -K now

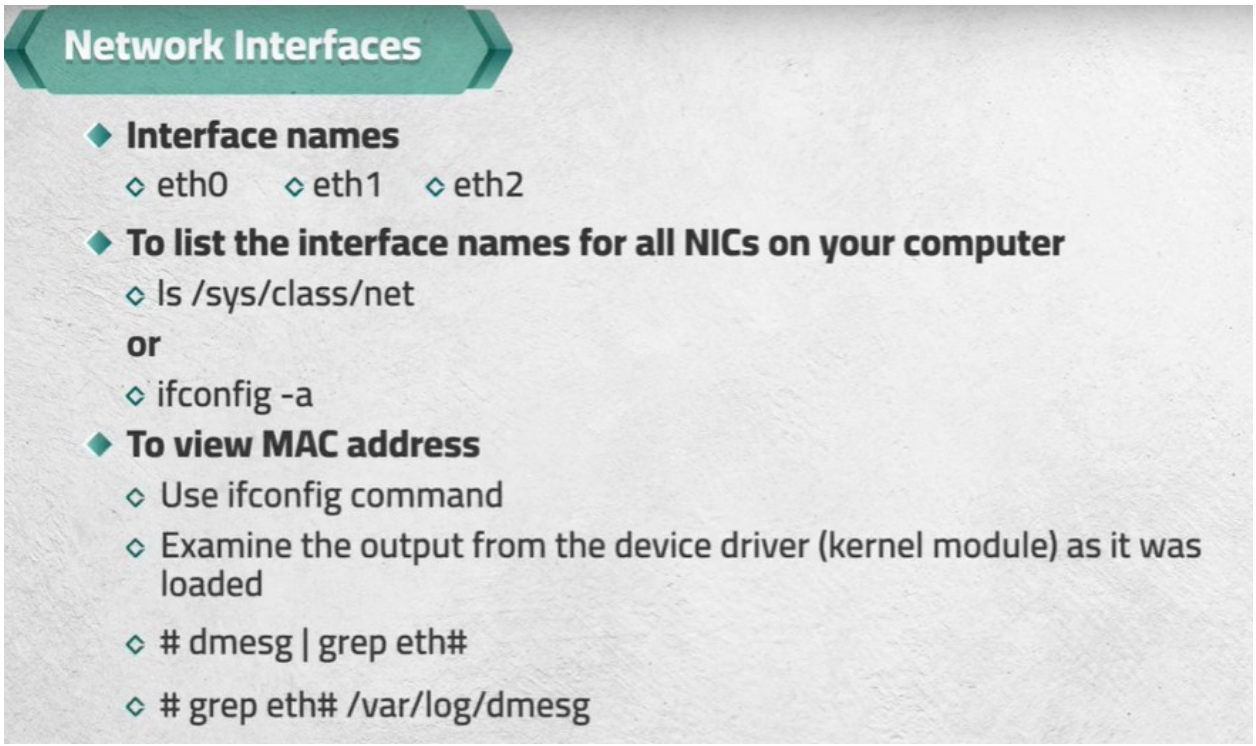
    # doesn't really shutdown only send the warning messages and disable logins.

System reboot

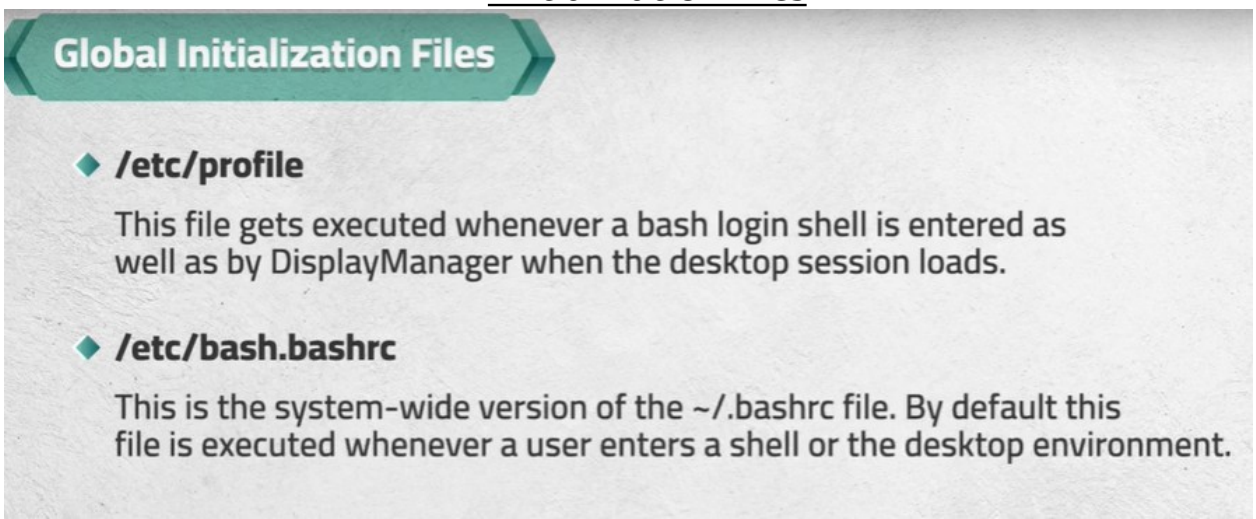
- shutdown -r
- reboot
- init 6
- **Press** CTRL+ALT+DEL

Chapter 8: Network Configuration & Initialization

Hostname → show the name of this pc
Hostname <**newname**> → Changes the hostname temporarily, it will be lost after restart.
 this change is temporary and will reset after a reboot unless you make the change permanent in the */etc/hosts***.

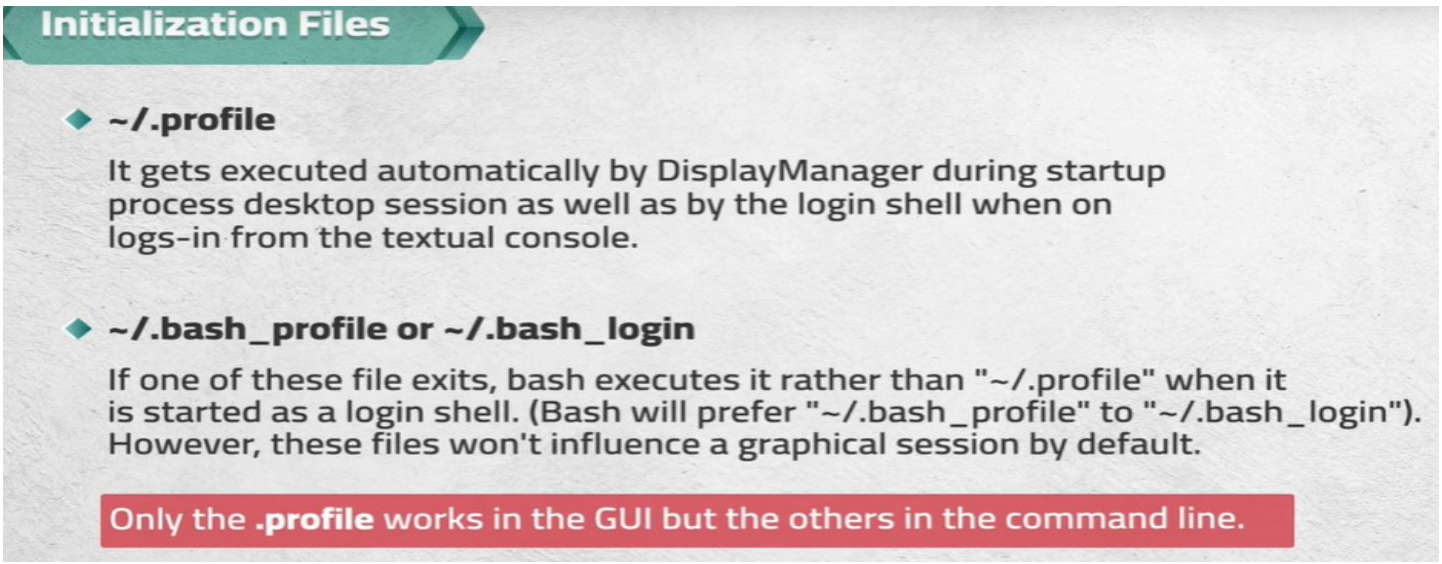


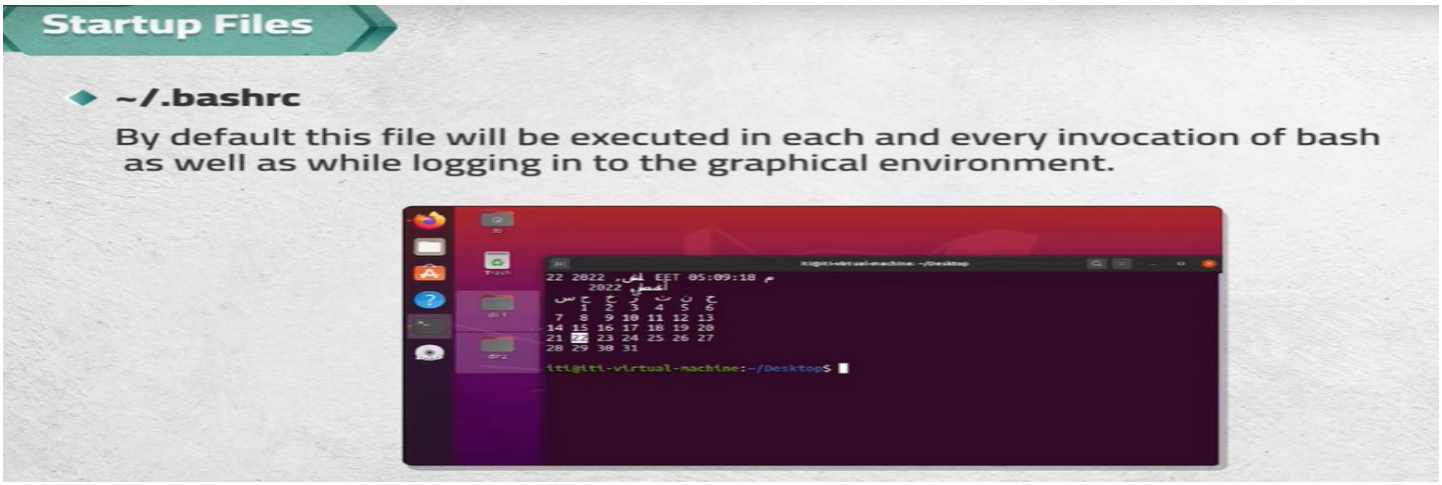
Initialization files



The **/etc/profile** file in Linux is a system-wide configuration file that is executed by all users' login shells. It sets up environment variables and startup commands for bash and other sh-like shells when a user logs in interactively.
The **/etc/bash.bashrc** file in Linux is a system-wide configuration file for the bash shell. Unlike **/etc/profile**, which is executed only for login shells, the **/etc/bash.bashrc** file is executed for every interactive non-login shell started by any user.
Purpose of /etc/bash.bashrc:
It sets up environment variables, aliases, and bash-specific options that apply to all users of the system.
It configures the behavior of interactive shells (such as terminals) that aren't login shells.
Difference Between **/etc/profile** and **/etc/bash.bashrc**:
/etc/profile: Executes when a login shell starts (e.g., when a user logs in via SSH or a terminal session).
/etc/bash.bashrc: Executes for interactive non-login shells (e.g., when a user opens a new terminal window or tab in a desktop environment).

You can edit the .bashrc file using the vi editor. For ex: by writing date cal at end of file it will show as in the image above





Environment Variables

- **$HOME**

  Complete path of the user home directory

  Example:

  - mkdir $HOME/file1

- **$PATH**

  A colon-separated list of directories used by the shell to look for executable program names

  Example:

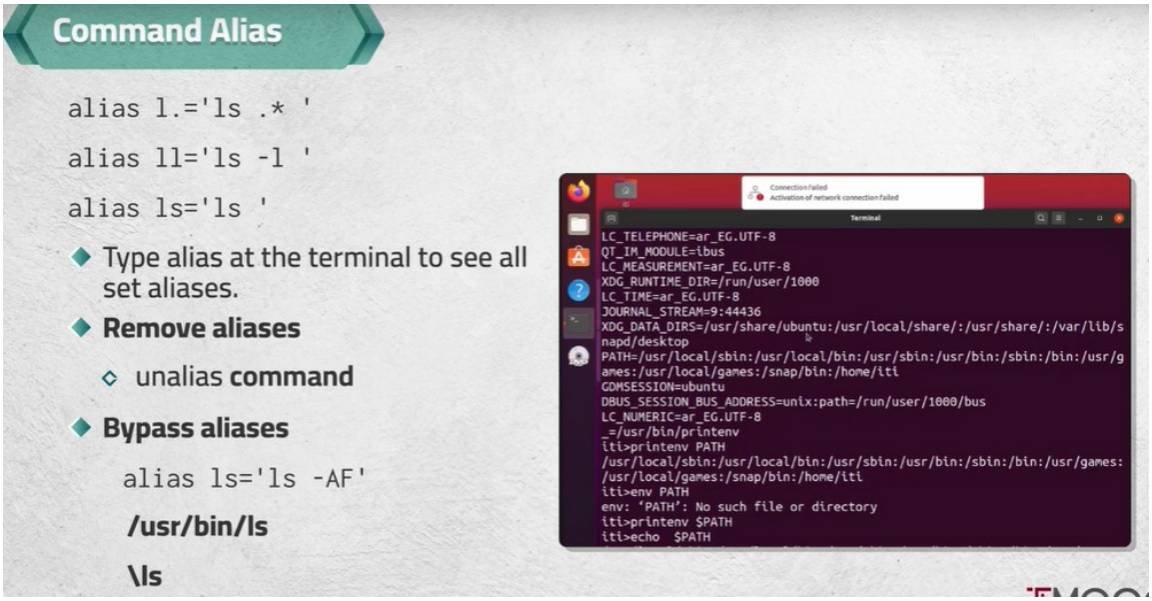
  - echo $PATH

  /usr/bin:/bin:/usr/local/java/bin

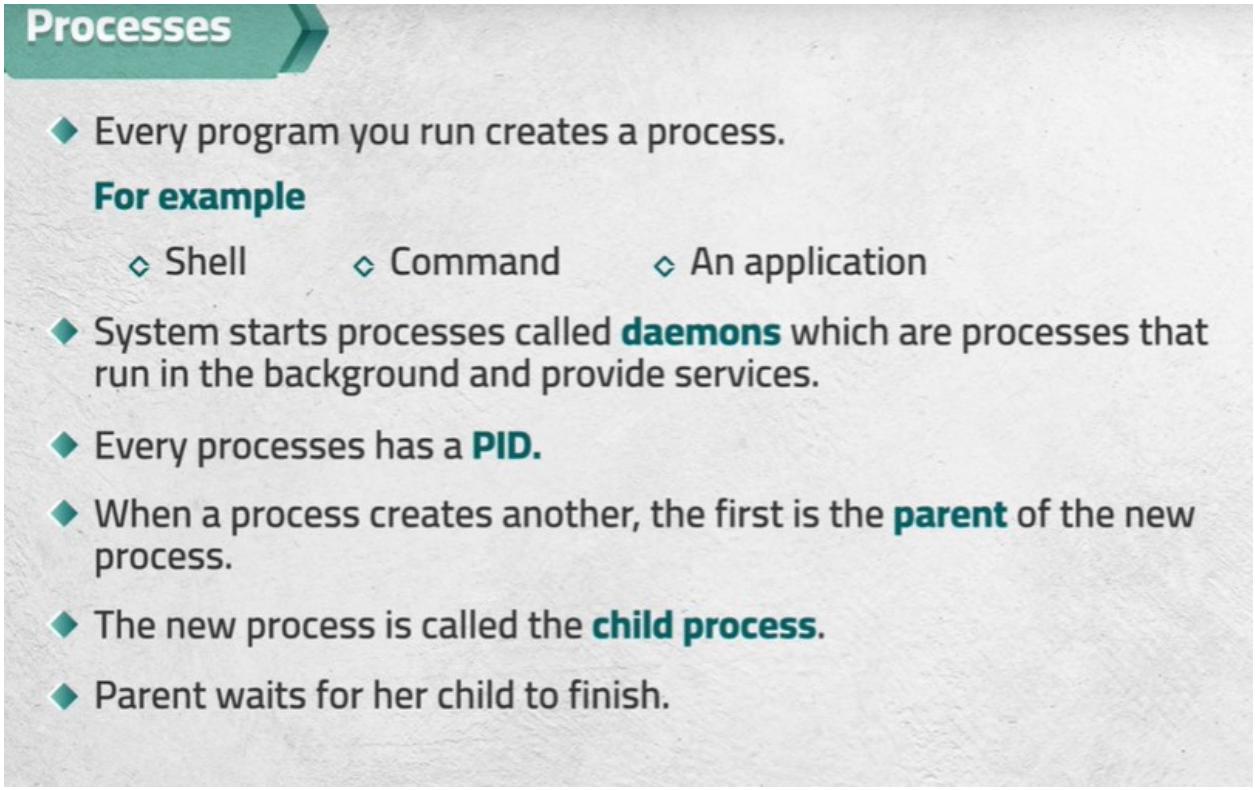
Environment Variables

- **$PWD**
  The user current working directory
- **$SHELL**
  Path name of the login shell
- **$USER**
  Currently logged in user
- **$HOSTNAME**
  Name of the computer

env → returns all environment variables
printenv → returns all environment variables
printenv <variable name> → returns values of this variable



Chapter 9: Overview Processes, Priorities and Signals



Process Priority

- Only process at a time may be executed on the CPU.
- Every process which is ready to run has a scheduling priority.
- The Linux process divides CPU time into time slices, in which each process will get a turn to run, higher priority processes first.
- User can affect the priority by setting the niceness value for a process.

Process Priority

- Niceness values range from -20 to +19, which indicates how much of a bonus or penalty to assign to the priority of the process.
- Most processes run with a niceness value of 0 (no change).

```
linuxuser@ubuntu: $ ps -eo pid,ppid,ni,comm
    PID    PPID  NI COMMAND
      1       0   0 systemd
      2       0   0 kthreadd
      3       2 -20 rcu_gp
      4       2 -20 rcu_par_gp
      5       2 -20 netns
      7       2 -20 kworker/0:0H-events_highpri
      9       2 -20 kworker/0:1H-events_highpri
     10       2 -20 mm_percpu_wq
     11       2   0 rcu_tasks_kthread
     12       2   0 rcu_tasks_rude_kthread
     13       2   0 rcu_tasks_trace_kthread
     14       2   0 ksoftirqd/0
     15       2   0 rcu_preempt
     16       2   - migration/0
     17       2   - idle_inject/0
     19       2   0 cpuhp/0
     20       2   0 cpuhp/1
     21       2   - idle_inject/1
     22       2   - migration/1
     23       2   0 ksoftirqd/1
     25       2 -20 kworker/1:0H-events_highpri
     26       2   0 cpuhp/2
```

***Ps* → The ps command in Linux displays information about active processes. For a snapshot of all processes, use ps aux. To delve deeper into specific process details, you might use options like -ef for a full-format listing.**

- ps -f provides a more human-readable output with full information such as user, PID, PPID, CPU time, etc.

Example of `ps -f`:

```bash
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:33 ?        00:00:01 /sbin/init
root      1023     1  0 08:33 ?        00:00:00 /usr/lib/systemd/systemd-journald
user1     1234  1000  0 08:35 pts/0    00:00:00 bash
user1     1256  1234  0 08:35 pts/0    00:00:00 ps -f
```

Explanation of columns:

- **UID**: The user ID of the process owner.
- **PID**: Process ID.
- **PPID**: Parent process ID (ID of the process that started this process).
- **C**: CPU usage percentage.
- **STIME**: Start time of the process.
- **TTY**: Terminal associated with the process.
- **TIME**: Cumulative CPU time used by the process.
- **CMD**: Command that started the process, along with its arguments.
    - ps -l shows more detailed technical information like priority, nice value, flags, and the process state.

nice -n 10 bash → The command nice -n 10 bash is used to start a new bash process with a niceness value of 10. In Linux, niceness determines the scheduling priority of a process. A higher niceness value means the process will be more "polite" in terms of CPU usage, getting less priority compared to other processes.

Renice 20 -p 2636 → The command renice 20 -p 2636 is used to change the niceness (priority) of an already running process with the process ID (PID) 2636 to 20.

- Use nice to set the niceness of a process before it starts.
- Use renice to change the niceness of a process after it is already running.

Sigterm vs sigkill

SIGTERM (Signal 15)

Description: SIGTERM is a signal that requests a process to terminate gracefully.

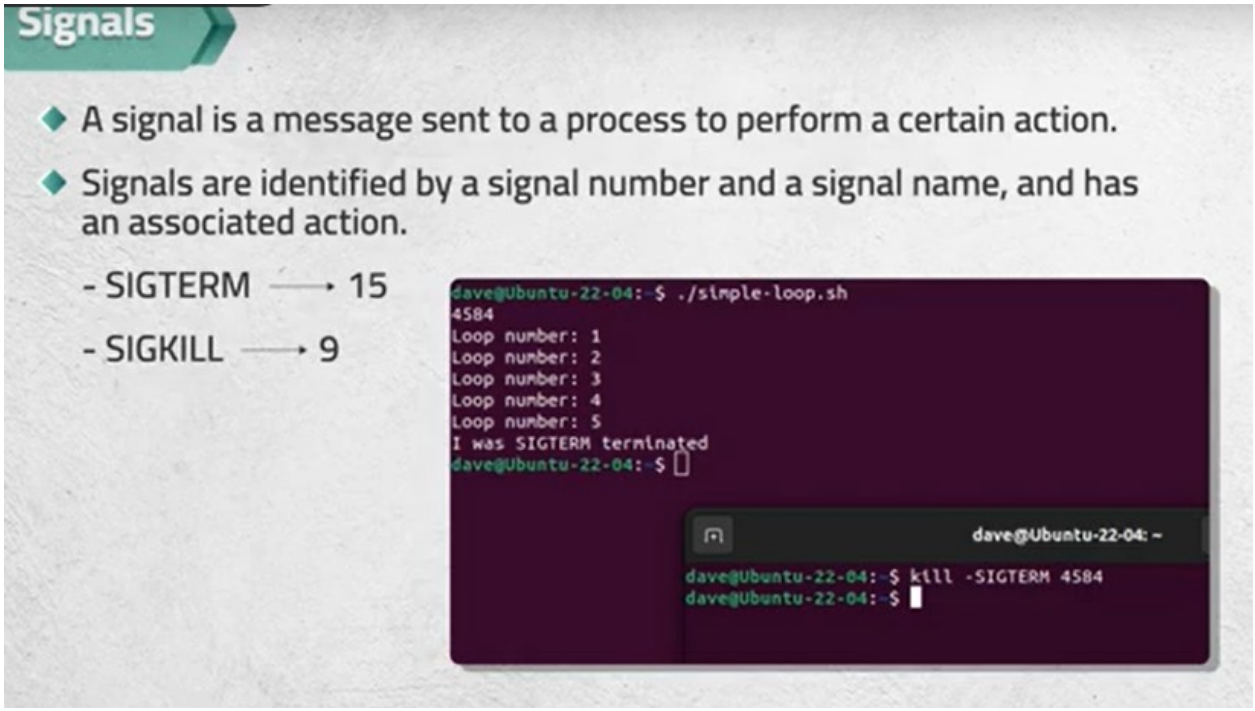
Behavior: When a process receives SIGTERM, it gets a chance to clean up resources (like closing files, saving state, releasing memory, etc.) before shutting down. The process can handle or ignore this signal if it has a handler for it.

Default Action: If no specific handler is defined, the process terminates.

Use Case: It's the preferred way to terminate a process because it allows the process to close itself properly.

Kill -SIGTERM 4584

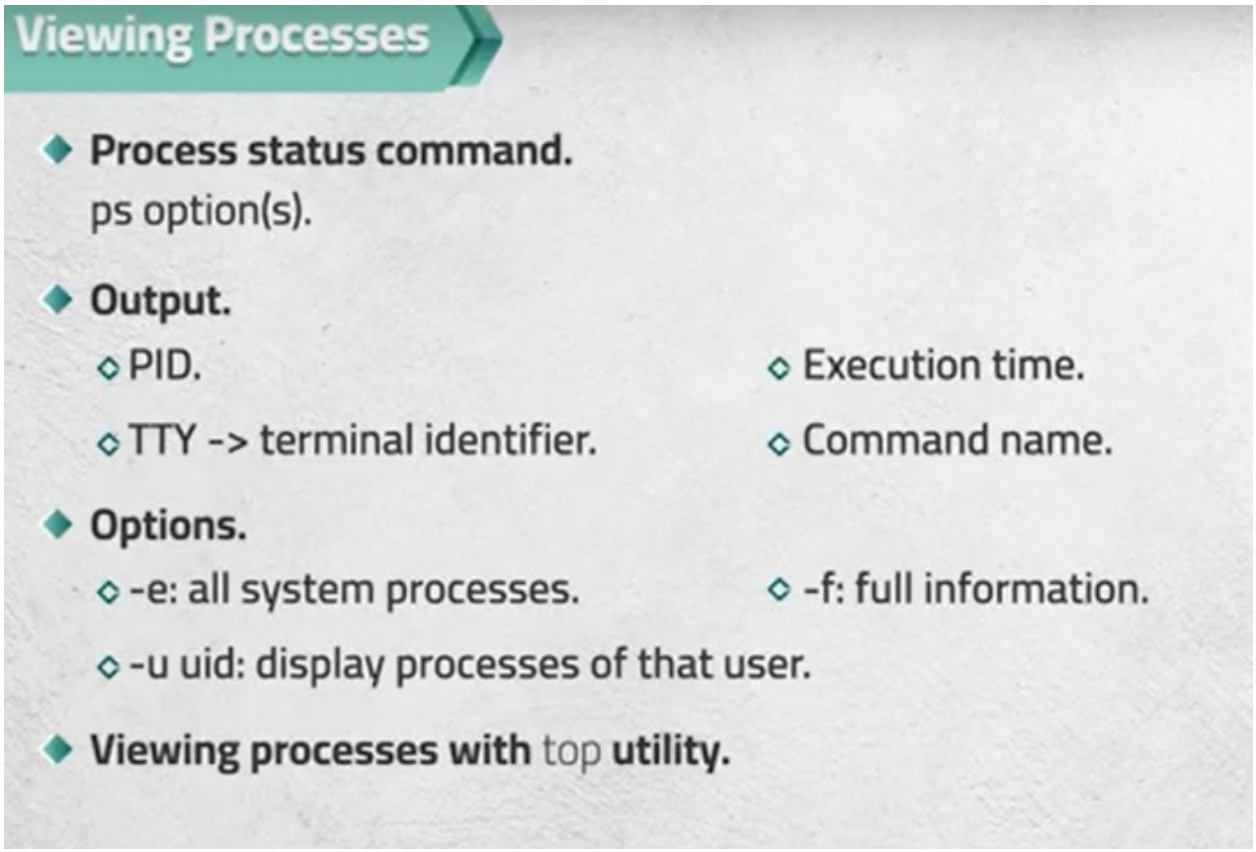
Kill -SIGKILL 4584

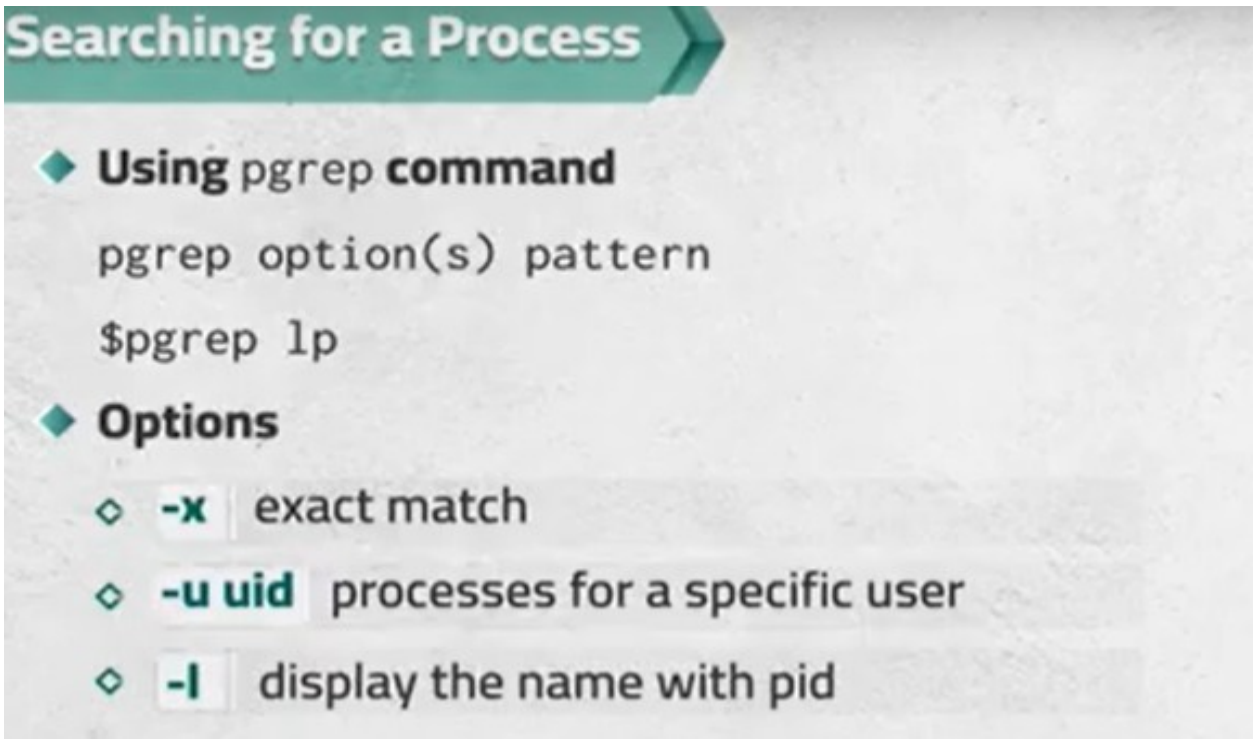


Description: SIGKILL is a forceful kill signal that cannot be caught, blocked, or ignored by the process.

Behavior: When a process receives SIGKILL, the kernel immediately terminates the process. The process has no chance to clean up resources or execute any shutdown code.

Default Action: The process is immediately killed.
Use Case: Use SIGKILL when a process becomes unresponsive or does not respond to SIGTERM. However, it is generally a last resort because it can lead to data corruption or other issues if the process was in the middle of a critical operation.

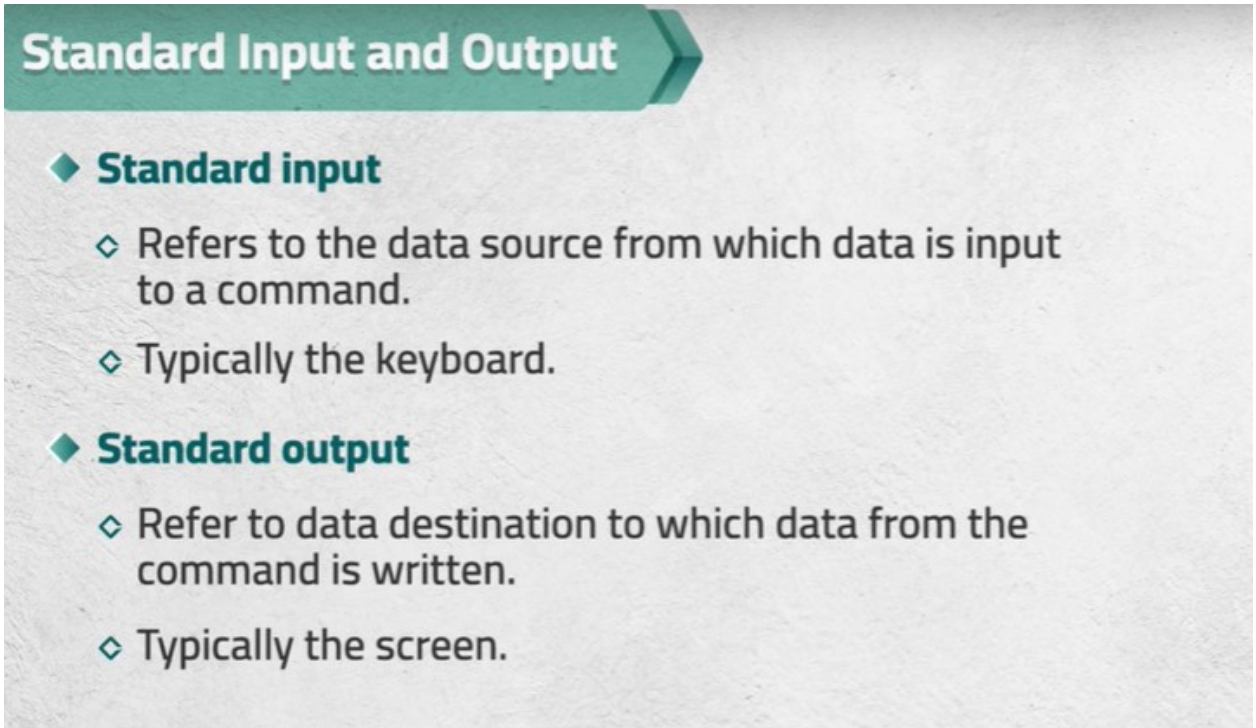




pkill sendmail → Terminate All Processes Named sendmail

8. kill sends signals to processes specified by their PIDs.
9. pkill sends signals to processes based on their names or other attributes.

Chapter 10: Input/output redirection and piping

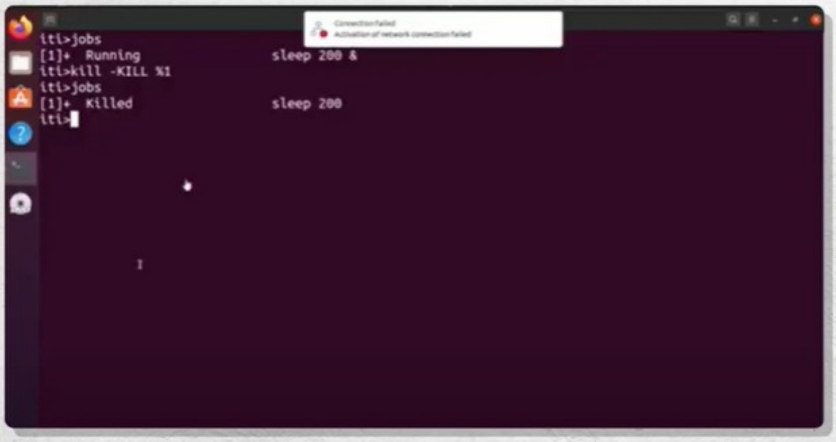


- **Standard error**
  - Refer to the output destination for the errors and messages generated by the command.
  - Typically the screen also.

Redirecting input and output means that terminal will write the output of command in a file for example instead of the default output destination which is the screen. Same idea for input.

Redirecting Input and Output

- Command > fname
- Command >> fname
- Command < fname



Ex:

ls > output_file → ***will execute ls and write its output in the output_file***

** use >> instead of > to append to the file and not overwrite it.

Example:

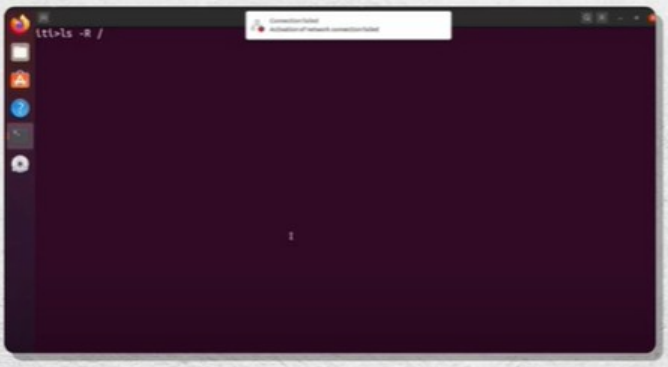
- $ find /etc –name passwd > findresult
- $ ls –l /etc >> findresult
- mail < file1

Redirecting Standard Error

- Standard error is redirected to a file using the regular output redirection operator, but you must place a 2 in front of the operator (2>).

Example:

- $ find / -name passwd 2> errs
- $ find / -name passwd 2> errs > results

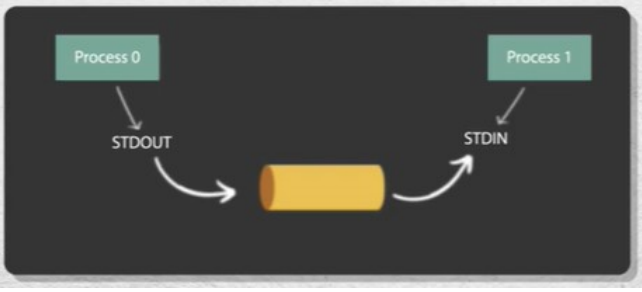


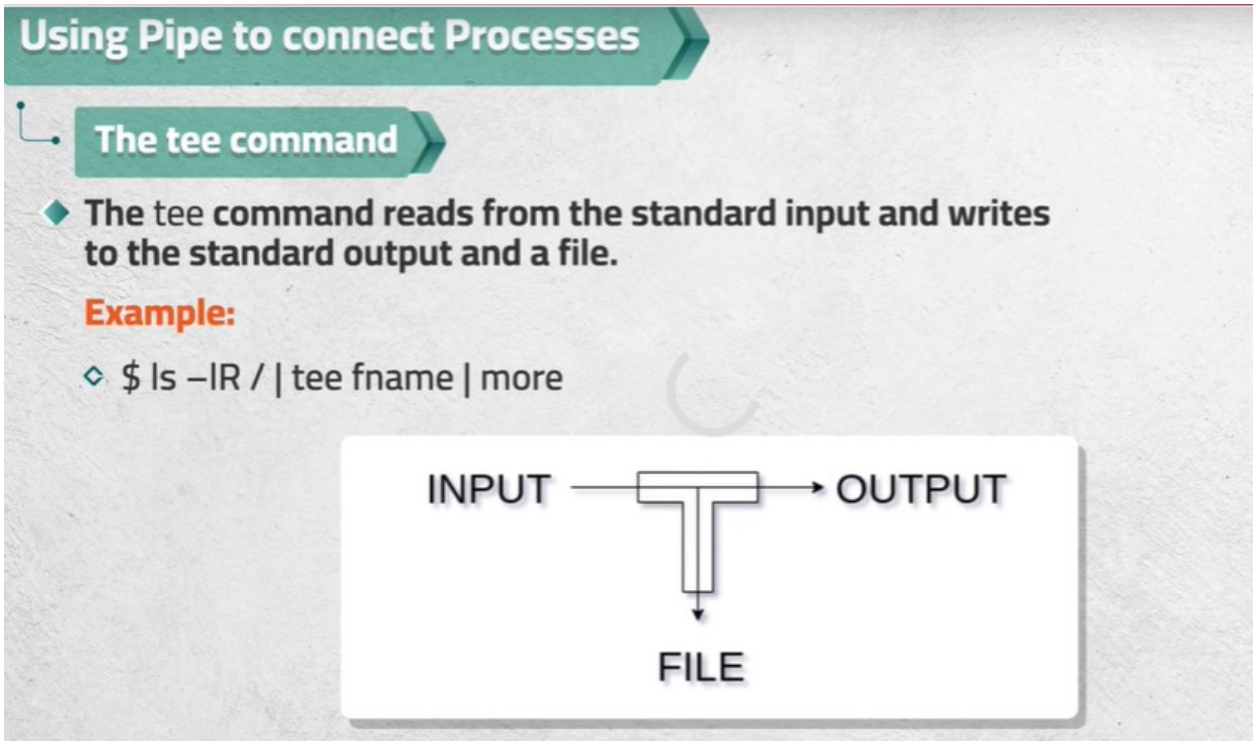
Using Pipe to connect Processes

- The pipe
  - A pipe (|) is used to send the output of one command as the input to another.
  - The most common use of a pipe is to take a command that's output might go on for pages (such as cat or ls -l) and feed it through more.

Example:

- $ ls –lR / | more





Chapter 11: Overview Sort and Compare Files

wc /etc/passwd → returns number of **lines** followed by number of **words** followed by number of **characters** in this file (word count).
wc -l /etc/passwd → returns number of lines
wc -w /etc/passwd → returns number of words
wc -c /etc/passwd →returns number of characters
diff file1 file2 → returns the difference between two files, and tries to make file1 like file2, it will tell you to add word, remove line, …etc. like in GitHub

**3c3 means: line 3 in file1 has something changed not in line 3 in file2. Which is word “hello”

The `cut` command in Linux is used to extract specific sections from each line of a file or input stream. It is useful for manipulating and processing text files, such as extracting columns from a file or splitting strings.

Syntax

```bash
cut OPTION [FILE...]
```

Common Options

- `-f` : Specifies the fields to extract (based on the delimiter).
- `-d` : Specifies the delimiter that separates fields (default is TAB).
- `-c` : Cuts specific character positions from each line.
- `--complement` : Complement the selection, i.e., print the fields or characters that are **not** selected.
- `--output-delimiter` : Change the output delimiter to something else (default is the same as the input delimiter).

Examples

1. **Extracting Specific Fields (Columns)**

   ```bash
   cut -d',' -f2 file.txt
   ```

   This extracts the second field from each line of `file.txt`, assuming the fields are separated by commas.

2. **Extracting Multiple Fields**

   ```bash
   cut -d':' -f1,3 /etc/passwd
   ```

   This extracts the first and third fields from each line of the `/etc/passwd` file, using `:` as the delimiter.

The grep command:

The `grep` command in Linux is used to search for patterns within files or input streams. It stands for "Global Regular Expression Print," and it is one of the most widely used commands for text processing. `grep` allows you to search for specific strings or patterns and display the lines that contain them.

Syntax

bash Copy code

```
grep [OPTIONS] PATTERN [FILE...]
```

Examples

1. Basic Search

```bash
grep "hello" file.txt
```

This searches for the string "hello" in `file.txt` and prints all lines that contain it.

2. Case-Insensitive Search

```bash
grep -i "hello" file.txt
```

This searches for "hello" in a case-insensitive manner, matching "hello", "Hello", "HELLO", etc.

3. Search Recursively in Directories

```bash
grep -r "TODO" /path/to/dir
```

- **Searching Log Files for Errors:**

```bash
grep "error" /var/log/syslog
```

This searches for the word "error" in the system log file.

- **Finding Text in Multiple Files:**

```bash
grep "main" *.c
```

This searches for the word "main" in all `.c` files in the current directory.

- **Identifying Files with Specific Content:**

```bash
grep -rl "database" /etc/
```

This finds and lists all files under `/etc/` that contain the word "database".

sort /etc/passwd | more → will sort the passwd file alphabetically and pipline the output as input to command more so it be human readable.

sort -k 4 -t : /etc/passwd → will sort using “key” k , the fourth field (4), which is userid in this case, with delimiter “t” (terminator) “:”.

grep bash$ /etc/passwd | cut -f1 -d: →

`grep bash$ /etc/passwd`: This part searches through the `/etc/passwd` file for any lines that end with `bash`. These lines typically represent user accounts where `/bin/bash` is the login shell.

`cut -f1 -d:`: After `grep` finds the matching lines, `cut` processes these lines and extracts the first field. In `/etc/passwd`, the first field represents the username.

What the Command Does

The full command filters the `/etc/passwd` file to find all users whose login shell is `/bin/bash` and then extracts and displays only the usernames of those users.

Example Output

If you run this command on a typical Linux system, you might see output like:

bash Copy code

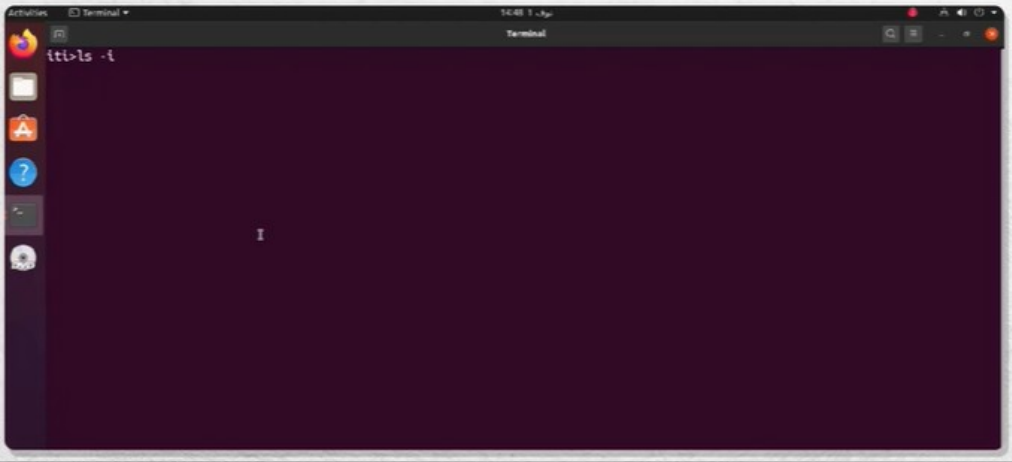
```
root
user1
user2
```

This output lists all usernames for users who have `/bin/bash` as their default shell.

Chapter 12: Inodes, Links, Search for files

Inode

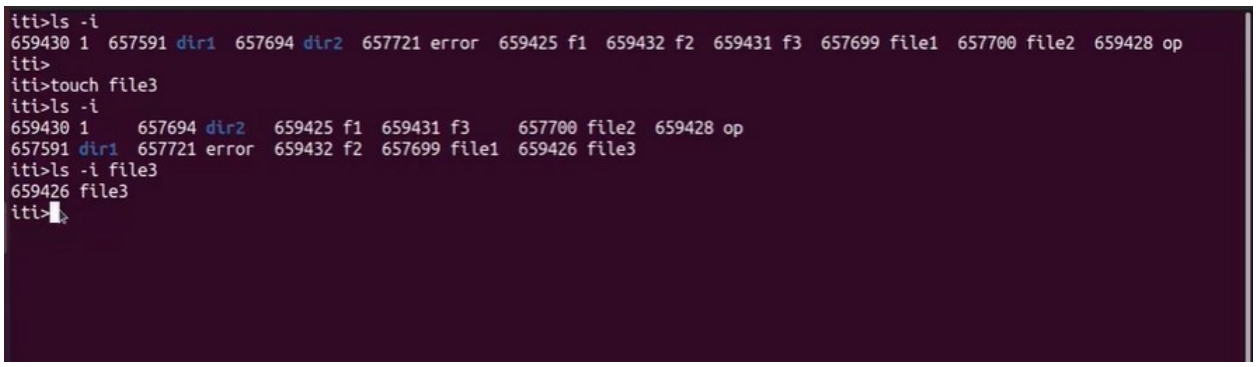
- Linux see all files as numbers called “inodes“, or index nodes.
- Within each filesystem is an inode table, in which all of the used inodes are mapped to particular files.



Inode

- The information stored in this table for each entry includes the following:

1. The type of file.
2. The file's permissions.
3. The group owner's GID.
4. The number of links.
5. The file owner's user ID.
6. When the file was last changed.
7. When the file was last accessed.
8. Where the file is on the media.



ls -i → returns the filename and inode of each file inside the dir.

An **inode** (index node) in Linux is a fundamental data structure used by the file system to store information about a file or directory. Each file and directory in a Linux file system is represented by an inode, which contains metadata about the file, but not the file's name or its actual data content.

Key Information Stored in an Inode:

- **File Type**: Indicates whether the inode represents a regular file, directory, symbolic link, etc.
- **Permissions**: The access rights associated with the file (e.g., read, write, execute).
- **Owner**: The user ID (UID) of the file's owner.
- **Group**: The group ID (GID) of the file's group.
- **File Size**: The size of the file in bytes.
- **Timestamps**: Including the time of last access (`atime`), last modification (`mtime`), and last status change (`ctime`).
- **Link Count**: The number of hard links pointing to the inode.
- **Pointers to Data Blocks**: Pointers that point to the disk blocks where the actual data of the file is stored.

How Inodes Work:

- When you create a file, an inode is allocated that stores all of the metadata except the file name. The file name is stored in a directory entry, which maps the name to the corresponding inode number.
- The inode number is unique within a file system and serves as an identifier for the file.

Directory Structure and Inodes:

- A directory is essentially a list of mappings between file names and inode numbers. When you access a file by name, the file system looks up the inode number associated with that name in the directory, retrieves the inode, and uses the metadata and pointers within the inode to locate the file's data on the disk.

Example Commands:

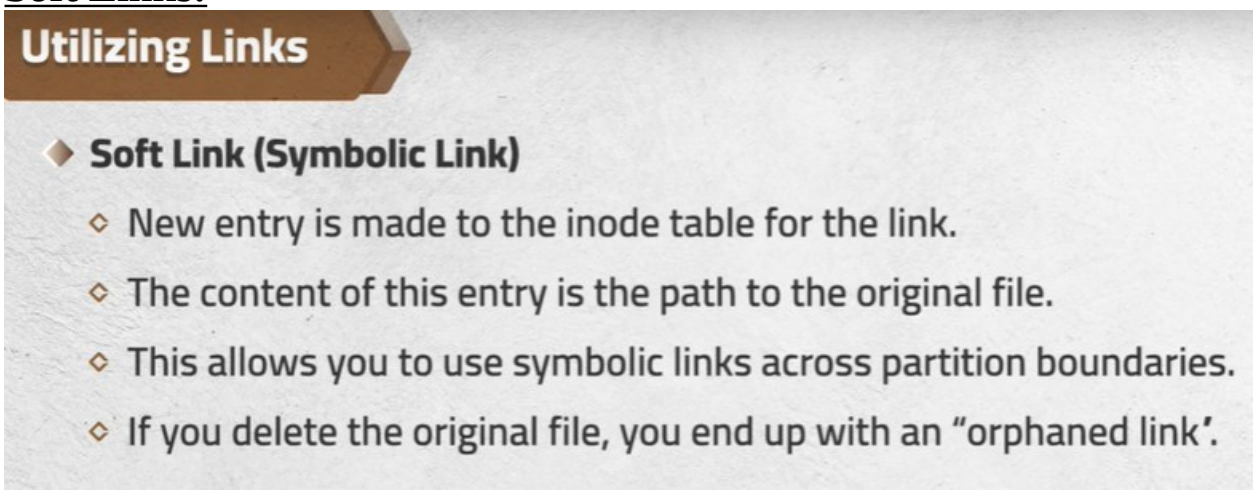
- **`ls -i`**: Displays the inode number of each file in a directory.
- **`stat filename`**: Displays detailed information about the file, including the inode number.
- **`df -i`**: Shows inode usage of file systems, helping to identify when a file system is running out of inodes.

Inode Limitation:

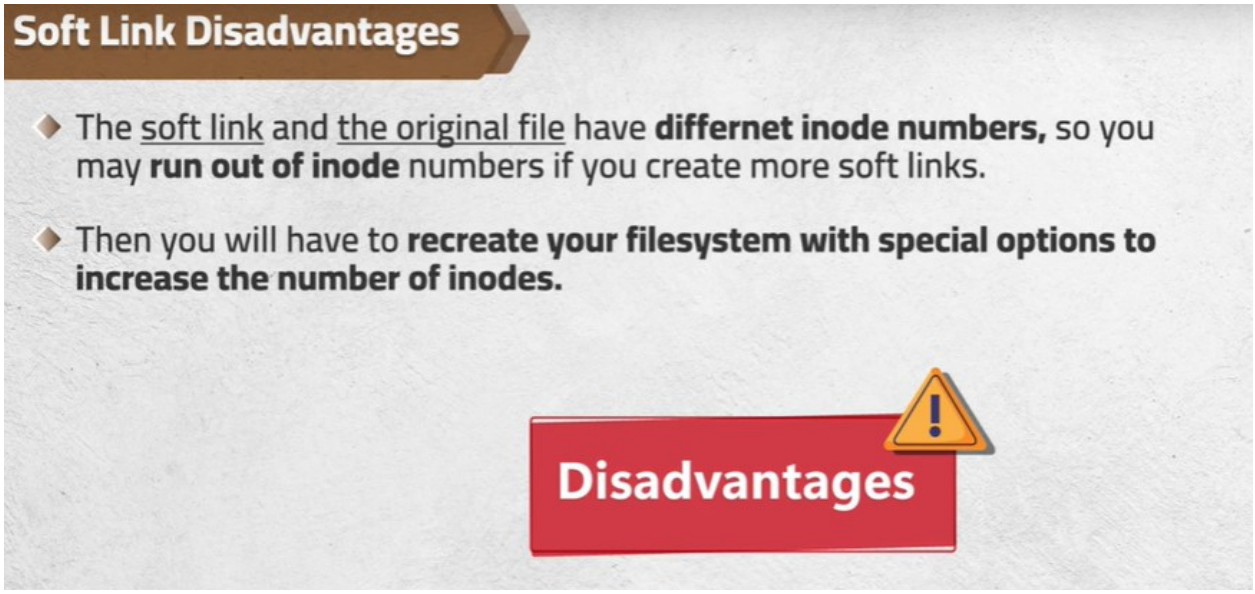
- A file system can run out of inodes even if there is available disk space. This limits the number of files and directories that can be created, as each new file or directory requires a new inode.

In essence, inodes are the backbone of file management in Linux, providing the essential metadata and structure needed to organize and access files efficiently.

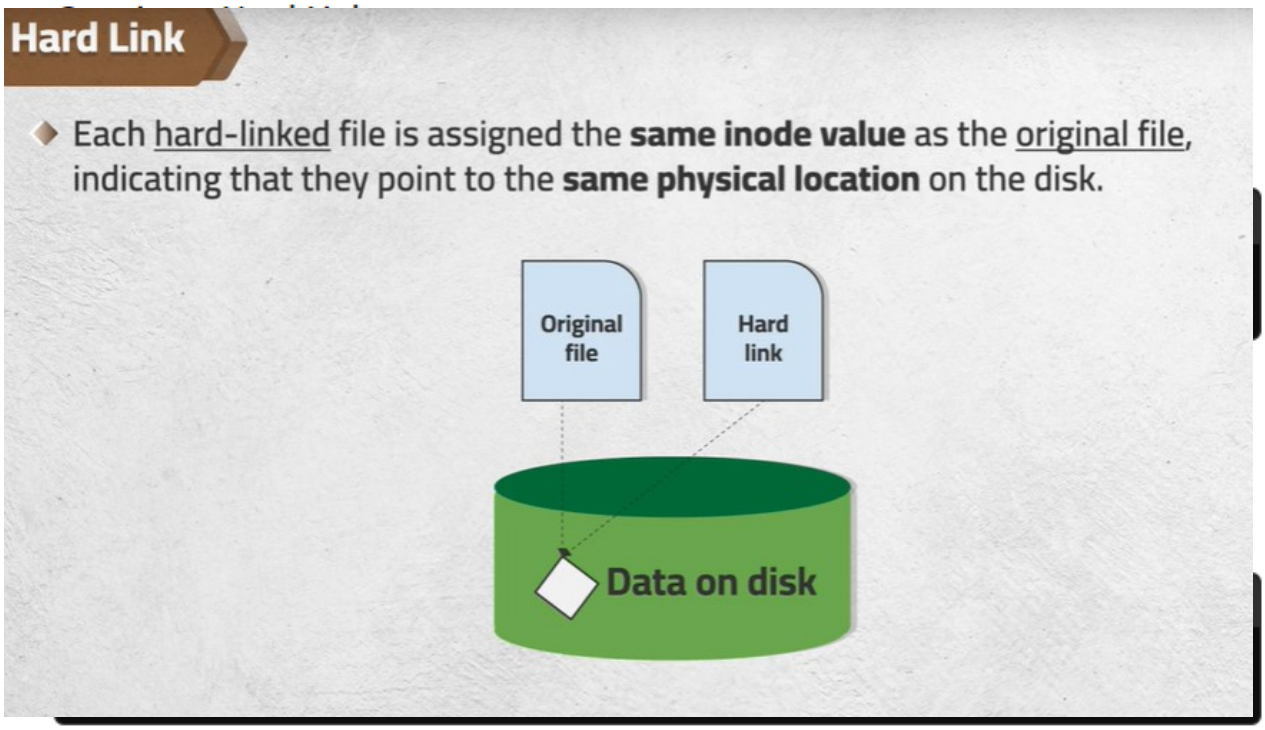
Soft Links:



ln -s [target] [link_name] → creates soft link target: The file or directory you want to link to, link_name: The name of the symbolic link to be created.



Hard Links



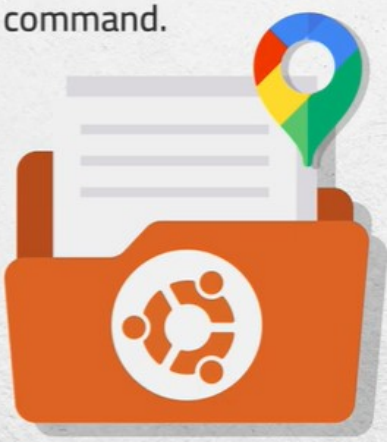
- This command creates a hard link named `file_link.txt` that points to `file.txt`.
- Both `file.txt` and `file_link.txt` now refer to the same inode, meaning they are effectively the same file.

Finding Files with locate

- The locate command searches through a prebuilt database containing the contents of your filesystem at the time the database was last updated.
- The locate database is built by using the **updatedb** command.

Example:

- locate passwd



Locating Files with find

- The find command searches the live filesystem.
- find is slower than locate, causes more of a load on the system, but more powerful than locate.
- You are also limited by your own permissions.

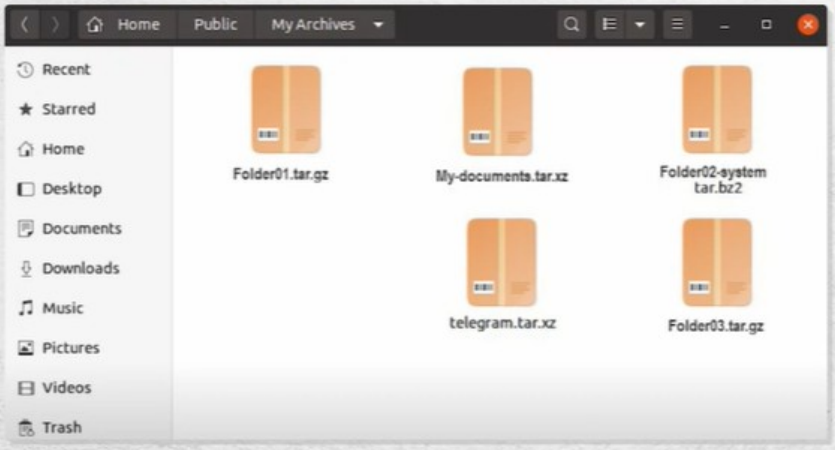
Expression	Definition
- name filename	Finds files matching the specified filename. Metacharacters are acceptable if placed inside " ".
- size [+|-]n	Finds files that are larger than +n, smaller than -n, or exactly n. The n represents 512-byte blocks.
- atime [+|-]n	Finds files that have been accessed more than +n days, less than -n days, or exactly n days.
- mtime [+|-]n	Finds files that have been modified more than +n days ago, less than -n days ago, or exactly n days ago.
- user loginID	Finds all files that are owned by the loginID name.
- type	Finds a file type, for example, f (file) or d (directory).
- perm	Finds files that have certain access permission bits

find / -name passwd → searches for a file in root dir with the name passwd

Chapter 13: Overview Compress and Archive Files

Introduction To Archiving

- To safeguard your files and directories, you can create a copy, or archive, of the files and directories on a removable medium, such as a cartridge tape. You can use the archived copies to retrieve lost, deleted, or damaged files.



Archiving Files

- tar **command archives files to and extracts files from a single file called a tar file.**
- The default device for a tar file is a magnetic tape device.
- tar functions archivefile filenames.

- Function

- c: create a new tar file.
- t: list table of content.
- x: extracts files from the tar command.
- f: specify the archive file.
- v: verbose mode.
- file3.

***tar -cvf myarch.tar ** → The command tar -cvf myarch.tar is used in Linux to create a tar archive file named myarch.tar.

Breakdown of the Command:

- `tar`: The `tar` command is short for "tape archive" and is used to create, extract, and manipulate tar archive files.
- `-c`: The `-c` option stands for "create." It tells `tar` to create a new archive.
- `-v`: The `-v` option stands for "verbose." It tells `tar` to list all the files it is processing (i.e., being added to the archive) on the terminal as it works. This helps you see what's being archived.
- `-f`: The `-f` option allows you to specify the name of the archive file. In this case, `myarch.tar` is the name of the archive that will be created.
- `myarch.tar`: This is the name of the archive file that will be created. It will contain all the files and directories specified by the wildcard `*`.
- `*`: The asterisk (`*`) is a wildcard that matches all files and directories in the current directory. It tells `tar` to include everything in the current directory in the archive.

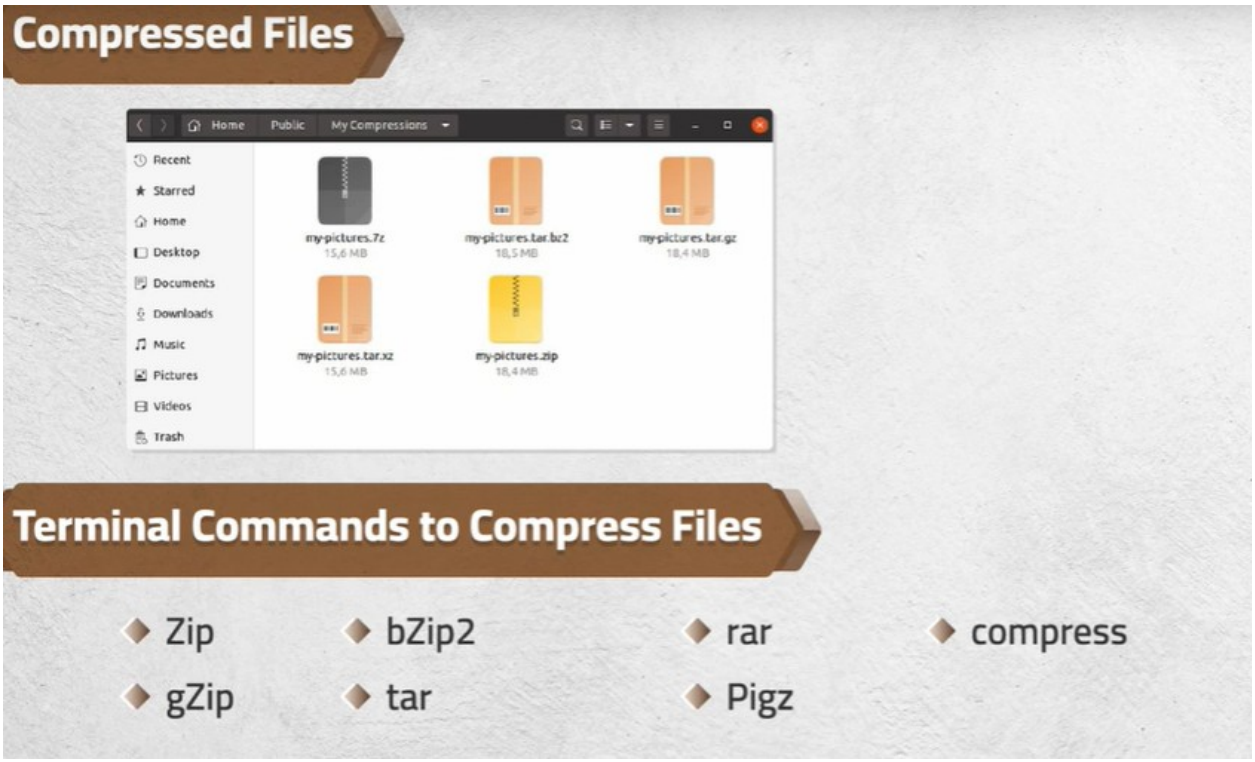
What Happens When You Run This Command:

1. **Creation of the Archive:** `tar` creates a new archive file named `myarch.tar`.
2. **Archiving Files and Directories:** All files and directories in the current directory are added to `myarch.tar`. This includes hidden files if they exist and match the wildcard.
3. **Verbose Output:** As `tar` processes each file and directory, it prints the name to the terminal due to the `-v` option, so you can see what's being included.

***tar tf compressed.tar* → returns the content of the archived file.
***tar xvf compressed.tar file1* → The command tar xvf compressed.tar file1 is used to extract a specific file from a tar archive.

Breakdown of the Command:

- `tar`: The `tar` command is used for creating and manipulating tar archive files.
- `x`: The `-x` option stands for "extract." It tells `tar` to extract files from the archive.
- `v`: The `-v` option stands for "verbose." It instructs `tar` to display the names of the files being extracted to the terminal, which helps you see the progress of the extraction.
- `f`: The `-f` option indicates that you are specifying the name of the archive file. It should be followed by the name of the archive file you want to work with.
- `compressed.tar`: This is the name of the tar archive from which files will be extracted. The `tar` command will look inside this file for the specified files.
- `file1`: This is the name of the specific file you want to extract from the archive. If `file1` exists in the `compressed.tar` archive, it will be extracted to the current directory.



Manipulating Files Compression

1-Compress Command

Compressing:

- Compression reduces a text file by 50 percent to 60 percent.
- compress [-v] filename
- Compress command replaces the original file with a new file that has a .Z extension.

Example:

- compress -v files.tar
  - files.tar: Compression: 70.20% --
  - replaced with files.tar.Z

Manipulating Files Compression

1-Compress Command

Listing:

To display file contents while leaving the file intact, you can use the **'zcat'** command.

- zcat filename

Example:

- zcat file1

Decompressing:

To decompress file, you can use **'uncompress'** command

- uncompress options filename

Example:

- uncompress -v files.tar.Z
  - files.tar.Z:-- replaced with files.tar

Manipulating Files Compression

2-gzip Command

Compressing:

- The gzip command reduces the size of files.
- gzip [-v] filenames
- The original file is replaced by a file with the same name and a .gz extension.

Examples:

- gzip file1 file2 file3 file4
- ls *.gz
  - file1.gz file2.gz file3.gz file4.gz

Manipulating Files Compression

2-gzip Command

Decompressing

- Restoring gzip file using the **gunzip** command

Examples:

- gunzip file1.gz

Listing:

- **gzcat** command display the content of files compressed by gzip- gzcat filename

Manipulating Files Compression

3-bzip2 Command

Compressing:

- The bzip2 command reduces the size of files.
- bzip2 [-v] filenames
- The original file is replaced by a file with the same name and a .bz2 extension.

Example:

- bzip2 file1 file2 file3 file4
- ls *.bz2
  - file1.bz2 file2.bz2 file3.bz2 file4.bz2

Manipulating Files Compression

3-bzip2 Command

Decompressing:

- Restoring bzip2 file using the bunzip2 command

Example:

- bunzip2 file1.bz2

Listing:

- bzcat command display the content of files compressed by bzip2
  - bzcat filename

Manipulating Files Compression

4-zip Command

Compressing:

- zip command compresses multiple files into a single archive file.
- zip command adds the .zip extension to the file name of the compressed archive file if you do not assign a new file name with an extension.
- zip target_filename source_filenames

Example:

- zip file.zip file2 file3
  - adding: file2 (deflated 16%)
  - adding: file3 (deflated 26%)
- ls
  - file.zip file2 file3

Manipulating Files Compression

4-zip Command

Listing:

- To list the files in a zip archive
  - unzip –l file.zip

Decompressing:

- To restore a zip file
  - unzip file.zip