

2.1	Directory Navigation Commands (pwd, cd, mkdir, rmdir, ls, tree)
2.2	File Management Commands (cat, rm, cp, mv, touch)
2.3	File Permission and ownership (chmod, chgrp, chown, umask)
2.4	Common System Commands (who, whoami, man, echo, date, clear)
2.5	Text Processing Commands (head, tail, cut, sort, cmp, tr, uniq, wc, tee)
2.6	Introduction to Process
2.7	Process Control Commands (ps, fg, bg, kill, sleep)
2.8	Job Scheduling Commands (at, batch, crontab)

2.1 Directory Navigation Commands (pwd, cd, mkdir, rmdir, ls, tree)

1. pwd

The pwd command in Linux, which stands for "**print working directory**", displays the full path of the current directory you are in. It is a fundamental command for navigating the file system.

How to use it:

- Open a terminal or command-line interface.
- Type pwd and press Enter.
- The output will be the full path of your current directory, starting from the root directory (/)

Example:

If you are in your home directory, which might be something like /home/vmp, then pwd will display /home/vmp

2. cd

The cd command in Linux stands for "**change directory**". It is used to navigate between different directories within the file system. By default, when you open a terminal, you are in your home directory. The cd command allows you to move to other directories, either by specifying an absolute path (starting from the root directory) or a relative path (relative to your current location).

How to use the cd command:

1. Changing to your home directory:

cd (or cd ~) will take you to your user's home directory.

2. Changing to a specific directory (absolute path):

cd /path/to/your/directory will move you to the specified directory, regardless of your current location.

3. Changing to a subdirectory (relative path):

cd subdirectory_name will move you into the subdirectory within your current directory.

`cd ..` will move you up one level (to the parent directory).

`cd ../../` will move you up two levels.

You can combine these to navigate complex directory structures.

4. Changing to the previous directory:

`cd -` will switch you back to the last directory you were in.

3. mkdir

The `mkdir` command in Linux is used to create new directories (folders). It is a fundamental command for organizing files and directories within a file system. The basic syntax is `mkdir [options] directory_name`. It can create single or multiple directories, and even nested directories if the `-p` option is used.

How to use the mkdir command:

`mkdir directory_name`: Creates a directory named "directory_name" in the current working directory.

`mkdir dir1 dir2 dir3`: Creates three directories named dir1, dir2, and dir3 in the current directory.

Options:

`-p` or `--parents`: This option is crucial for creating nested directories. It will create the parent directories if they don't already exist. For example, `mkdir -p path/to/vmpnew/directory` will create path, then path/to, then path/to/vmpnew, and finally path/to/new/directory, if they don't already exist.

4. rmdir

The `rmdir` command in Linux is used to remove empty directories. It is specifically designed to delete directories that do not contain any files or subdirectories. If you try to remove a directory that is not empty using `rmdir`, it will generate an error and the directory will not be deleted.

Example: `rmdir vmp_directory`

If you need to remove a directory and all its contents (including files and subdirectories), you should use the `rm -r` command instead.

5. ls

The `ls` command in Linux is used to list directory contents. It displays the names of files and directories within a specified directory or the current directory if none is specified. It can also display additional information about the listed items, such as file type, permissions, size, and modification time, depending on the options used.

Basic Usage:

ls: Lists the contents of the current directory.

ls <directory>: Lists the contents of the specified directory.

Common Options:

-l (long listing): Displays detailed information about each item, including permissions, owner, size, modification date, and more.

-a (all): Shows all files, including hidden files (those starting with a dot ".").

-t (time): Sorts the output by modification time, with the newest files first.

-r (reverse): Reverses the order of the output.

-R (recursive): Lists the contents of subdirectories as well.

-F (classify): Appends a character to each entry to indicate its type (e.g., "/" for directories, "*" for executables).

-h (human-readable): Displays file sizes in a human-readable format (e.g., KB, MB, GB).

-i (inode): Displays the inode number of each file.

-n (numeric UID/GID): Displays the user and group IDs numerically instead of by name.

6. tree

The tree command in Linux is a utility for displaying the contents of directories in a tree-like format. It recursively lists directories and files, making it easy to visualize the file system structure. It's a handy tool for navigating and understanding the organization of directories, especially when dealing with complex nested structures.

Basic Usage:

tree: Displays the directory structure starting from the current directory.

tree <directory>: Displays the directory structure of the specified directory. (\$ tree vmpdir)

Common Options:

-a or --all: Includes hidden files and directories in the output.

-d or --dirs-only: Lists only directories, not files.

-L <level> or --max-depth=<level>: Limits the depth of the tree to the specified level. For example, tree -L 2 will only show the current directory and its immediate subdirectories.

-f or --full-path: Displays the full path for each file and directory.

2.2 File Management Commands (cat, rm, cp, mv, touch)

1. cat

The cat command in Linux, which stands for "concatenate," is a versatile utility used for displaying, creating, and manipulating text files. It's primarily used for viewing file contents, combining files, and creating new ones.

Common uses:

1. Displaying File Contents:

The most basic use is to print the contents of one or more files to the standard output (your terminal).

`cat filename` will display the entire content of filename.

`cat file1 file2` will display the contents of file1 followed by file2.

2. Creating New Files:

You can create a new file using `cat > filename`. This will open a text input mode where you can type your content. Press Ctrl+D to save and exit.

Example: `cat > vmp_new_file.txt`

3. Combining Files:

`cat file1 file2 > combined_file` will create a new file named `combined_file` containing the contents of file1 followed by file2.

This can be used to append to an existing file as well: `cat new_content >> existing_file`.

4. Other Useful Options:

`-n` or `--number`: Displays line numbers along with the file content.

`-b` or `--number-nonblank`: Similar to `-n` but only numbers non-blank lines.

`-s` or `--squeeze-blank`: Suppresses repeated empty lines.

`-v` or `--show-nonprinting`: Displays non-printable characters.

2. rm

The `rm` command in Linux is used to remove or delete files and directories from the file system. It's a powerful command that permanently deletes the specified files and directories, meaning they won't be recoverable through the standard recycle bin or trash can.

Syntax: `rm [options] [file/directory]`

Basic Usage: `rm filename` (deletes the file named "filename")

Removing Directories: `rm -r directoryname` (deletes the directory named "directoryname" and all its contents recursively)

Force Removal: `rm -f filename` (forcefully removes the file without prompting for confirmation)

Recursive and Force Removal: `rm -rf directoryname` (forcefully removes the directory and all its contents)

Confirmation Prompts:

`rm -i filename` (prompts for confirmation before deleting each file)

3. cp

The cp command in Linux is used to copy files and directories. It takes a source file or directory and a destination as arguments, creating a duplicate of the source at the specified destination.

Basic Syntax: `cp source_file destination_path`

Examples:

1. `cp vmpfile1.txt vmpfile2.txt`

This command copies file1.txt to file2.txt in the same directory. If file2.txt already exists, it will be overwritten.

2. `cp file1.txt /path/to/destination/`

This copies file1.txt to the specified destination directory. If the destination directory doesn't exist, it will be created.

3. `cp vmpf1.txt vmpf2.txt /path/to/destination/`

This copies file1.txt and file2.txt to the specified destination directory.

4. `cp -r source_directory destination_directory`

The -r or -R option is used to copy directories recursively, including all subdirectories and files within them.

5. `cp -i source_file destination_file`

The -i option prompts the user before overwriting an existing file.

4. mv

The mv command in Linux is primarily used for moving files and directories or for renaming files and directories. It essentially changes the location or name of an item. If the destination is an existing directory, the item is moved into that directory; otherwise, if the destination is a new name, the item is renamed.

1. Moving Files and Directories:

- To move a file or directory to a new location, use the command `mv [source] [destination]`.
- For example, `mv file.txt /home/vmp/documents/` : moves file.txt to the documents directory.
- When moving a directory, all its contents are also moved.
- If the destination path already exists, the file/directory is moved into it. If the destination path doesn't exist, the file/directory will be renamed to the destination path.

2. Renaming Files and Directories:

- To rename a file or directory, use `mv [old_name] [new_name]`.

- For example, `mv document.txt vmpdoc.txt` renames `document.txt` to `vmpdoc.txt`.
- The `mv` command can rename directories in the same way: `mv old_directory new_directory`.

3. Overwriting Files:

- The `mv` command will overwrite existing files or directories without warning if they share the same name at the destination.
- To prevent accidental overwriting, use the `-i` option (interactive) to prompt for confirmation before overwriting.
- For example, `mv -i file.txt /path/vmp/existing_file.txt`. This will ask for confirmation before overwriting the file, if it exists.
- The `-f` option (force) can be used to overwrite without prompting.
- The `-n` option (no-clobber) prevents overwriting.

5. touch

The `touch` command in Linux is a versatile tool primarily used for creating new empty files and updating the timestamps of existing files.

1. Creating New Files:

When the specified file doesn't exist, `touch` creates an empty file with the given name.

For example, `touch vmp_new_file.txt` will create an empty file named "`my_new_file.txt`" in the current directory.

2. Updating Timestamps:

If the file already exists, `touch` updates its access and modification timestamps to the current time.

2.3 File Permissions and Ownership (chmod, chgrp, chown, umask)

File Permissions:

- In Unix, all files and directories have permission.
- There are three file level and directory level permission: read, write and execute.
- Symbolically, there are represented as follow:
 - Read permission: `r`
 - Write permission: `w`
 - Execute permission: `x`
- In UNIX, File permissions are assigned numerical octal values from 0 to 7.
- Individually, however they have the following octal values.
 - Read permission: 4
 - Write permission: 2
 - Execute permission: 1

- Thus, for a file that has a permission field like `-rwxrwxrwx` (symbolic notation), the permissions are said to be read, write and execute for the owner, group and others.
- In Octal notation, it would be written as `777`.
- To check file permission string of 10 characters, `ls -l` command is used.
- Sample output of `ls -l` :
`-rw-rw-r-- 1 vmp vmp 20 Jun 15 13:12 file1.txt`

Observe the first column representing permission for the file `demo.txt`.

Here, the first character says whether the file is ordinary file or directory.

So, leaving it apart, consider next 9 characters as a group of 3 characters each: `rw- rw-r--`

Each group represents a **category** viz. **u (user/owner)**, **g(group)**, **o(others)** and **a(all)** respectively.

For every group, permissions are represented by characters **r(read)**, **w(write)**, **x(execute)**.

File Ownership:

- The person who creates a file will be the owner of that file.
- The login name of that person will be showed as the owner when `ls -l` option is used.
- The group to which the person belongs to, will be the group owner of the file.
- If you copy someone's file, you will be the owner only for that copy.
- One cannot create files in other's home directory, because one may not have permission to do so.
- When the system administrator creates a user account, he has to assign the following parameters to the user:
 - The user-id (UID) – both its name and numeric representation
 - The group-id (GID) – both its name and numeric representation

1. **chmod**: CHANGING FILE PERMISSIONS

- The **chmod** (change mode) command is used for assigning/removing different permissions to/from **category** (user, group, others).
- This command can be run only by the user (owner) and the super-user(admin).
- The **chmod** command can be used in two ways –
 - In a relative manner by specifying the changes to the current permissions
 - In an absolute manner by specifying the final permissions

1) **Relative Permissions**

- When changing permissions in a relative manner, **chmod** changes only the permissions specified in the command line and leaves the other permissions unchanged.
- The syntax is–
`chmod [category operation permission] filenames`
`chmod u+x vmpf1.txt`

- The argument for **chmod** is an expression consisting of some letters and symbols describing user category and type of permission being assigned / removed.
- The expression contains three components:
 - User category (user: **u**, group: **g**, others: **o**, All: **a**)
 - The operation to be performed (assign: **+**, remove: **-**, assign absolute permission: **=**)
 - The type of permission (read: **r**, write: **w**, execute: **x**)

\$ chmod ugo+x vmp.txt #assign(+) x to u(user, group, others)

\$ chmod a+x vmp.txt #assign(+) x to a(all)

\$ chmod +x vmp.txt #assign(+) x to all

\$ chmod u+x demo test1 test2 #multiple files are affected

\$ chmod go-r vmp.txt #remove r permission from group & others

\$ chmod a-x, go+r vmp.txt # remove the execute permission from all and then to assign read permission to group and others

\$ chmod o+wx demo.txt #multiple permissions

2) Absolute Permissions

- Irrespective of existing permissions for a file, we may need to assign a new set of permissions.
- That is, we wish to set all nine permission bits explicitly.
- This is known as *absolute permissions*.
- For this purpose, *chmod* uses a **string of three octal numbers**.

Permission	symbolic code	Octal value
No permission	-(hyphen)	0 - 000
Execute	X	1 - 001
Write	W	2 - 010
Write and Execute(2+1)	Wx	3 - 011
read	R	4 - 100
Read and Execute(4+1)	Rx	5 - 101
Read and Write(4+2)	Rw	6 - 110
Read, write and execute (4+2+1)	Rwx	7 - 111

- Every possible combination of three different permissions is shown in above table.
- Now, let us see some examples of using the absolute permissions with the help of octal digits.

Ex 1. Assigning read and write(4+2=6) permissions to all –

\$ chmod 666 vmpf1.txt

\$ ls -l vmpf1.txt

-rw-rw-rw- 1 vmp faculty 853 Sep 5 23:38 vmpf1.txt

Ex 2. To remove the write permission from group and others:

```
$ chmod 644 vmpf1.txt
$ ls -l vmpf1.txt
-rw-r--r-- 1 vmp faculty 853 Sep 5 23:38 vmpf1.txt
```

Ex 3. To assign all permissions to owner, read and write permissions to group and only execute permission to others –

```
$ chmod 761 test
$ ls -l test
-rwxrw---x 1 vmp faculty 853 Sep 5 23:38 test
```

Using chmod Recursively (-R)

```
$ chmod -R a+x vmpProgs
```

This makes all files and subdirectories found in the tree-walk (starting from vmpProgs directory, includes all files in subdirectories) executable by all users.

2. chown: Changing File Owner

- The syntax of **chown** command is:
chown options owner [:group] files
- To use **chown** command, we need the super-user permission

```
# ls -l note
-rwxr---x 1 john metal 347 May 10 20:30 note
```

```
# chown ricky note
# ls -l note
-rwxr---x 1 ricky metal 347 May 10 20:30 note
```

- Here, the ownership of the file *note* has been changed from *john* to *ricky*.

3. chgrp: Changing Group Owner

- By default, the group owner of a file is the group to which the owner belongs to.
- But, the **chgrp** command changes a file's group owner.
- Assume that *john* is a member of two groups *metal* and *dba*.
- And he has created a file *dept.txt* in *metal* group.
- He can change the group owner as below –

```
$ ls -l dept.txt
-rw-r--r-- 1 john metal 129 Jun 8 16:42 dept.txt

$ chgrp dba dept.txt
-rw-r--r-- 1 john dba 129 Jun 8 16:42 dept.txt
```

- When the user is not a member of particular group, he cannot change the group owner of any file to that group. But super-user can do so.

4. umask

The umask command works by affecting the default Linux file and folder permissions.

There are three categories of permissions for every file and folder in Linux:

Syntax: umask [-p] [-S] [mask]

How to Calculate umask Values

Linux uses the following default mask and permission values: The system default permission values are 777 (rwxrwxrwx) for folders and 666 (rw-rw-rw-) for files.

The default mask for a non-root user is 002, changing the folder permissions to 775 (rwxrwxr-x), and file permissions to 664 (rw-rw-r--).

The default mask for a root user is 022, changing the folder permissions to 755 (rwxr-xr-x), and file permissions to 644 (rw-r--r--).

This shows us that the final permission value is the result of subtracting the umask value from the default permission value (777 or 666).

For example, if you want to change the folder permission value from 777 (read, write, and execute for all) to 444 (read for all), you need to apply a umask value of 333, since:

$$777 - 444 = 333$$

How to use umask:

Displaying the current umask: umask

Setting the umask: umask [octal_value]

For example, umask 022 will set the umask to 022.

Using symbolic representation: umask -S displays the symbolic representation of the current umask (e.g., u=rwx,g=rx,o=rx).

umask -S with a symbolic value sets the umask (e.g., umask -S u=rwx,g=rx,o=rx).

2.4 Common System Commands (who, whoami, man, echo, date, clear)

1. who

The who command is a simple and effective way to display information about currently logged-in users. By typing who in the terminal, you will receive a list of usernames, terminal IDs, login times, and originating IP addresses if applicable.

who command is used to find out the following information :

- Time of last system boot
- Current run level of the system
- List of logged-in users and more

2. whoami

The 'whoami' command in Linux is a built-in utility that displays the username of the current user. It is used with the syntax, whoami [option]. It's a quick and easy way to confirm your identity within the system.

3. man

The man command in Linux is used to display the user manual for commands, system calls, library functions, and other aspects of the operating system. It provides detailed information about a command's syntax, options, and usage examples, making it a crucial tool for understanding and using Linux effectively.

syntax: man command_name

e.g.: man ls

Key sections within a man page:

NAME: Provides a brief description of the command.

SYNOPSIS: Shows the command's syntax and available options.

DESCRIPTION: Gives a more detailed explanation of the command's functionality.

OPTIONS: Lists all available command-line options with descriptions.

EXAMPLES: Includes practical examples of how to use the command.

SEE ALSO: Points to related commands or manual pages.

4. echo

The echo command in Linux is a built-in command that allows users to display lines of text or strings that are passed as arguments. It is commonly used in shell scripts and batch files to output status text to the screen or a file.

Basic Syntax: echo [option] [string]

\$ echo "Welcome to Linux"

```
$ x=10
```

You can print the value of variable x by executing the command

```
$ echo The value of x is $x
```

5. date

The date command in Linux allows the user to display the current date and time in a variety of formats and set the system date and time. Being a part of the core utilities in Linux and Unix-like operating systems, it makes it an essential command for system configuration and scripting.

However, by default, the date displays the current time in the system's configured time zone. It supports formatting using the special date format specifier to display the time exactly the way you need it.

Key uses of the date Command

- Displaying the Current Date and Time
- Setting the System Date and Time
- Customizing the Output Format
- Performing Date Calculations
- Displaying Time in Different Time Zones

```
$ date
```

```
Sat Jul 5 11:04:45 IST 2025
```

The `-u` option with the date command allows you to display the current time in GMT (Greenwich Mean Time)

```
$ date -u
```

```
Sat Jul 5 05:35:42 GMT 2025
```

```
$ date -d "2 days ago"
```

Outputs the date two days prior.

```
$ sudo date -s "2024-10-03 14:00:00"
```

Sets the system date and time. (requires superuser privileges).

Format	Description	Example
<code>%Y</code>	Full year (4 digits).	<code>date +%Y</code> outputs 2024.
<code>%m</code>	Month (01-12).	<code>date +%m</code> outputs 10.

Format	Description	Example
%d	Day of the month (01-31).	<code>date +%d</code> outputs 08.
%H	Hour (00-23, 24-hour format)	<code>date +%H</code> outputs 14.
%M	Minute (00-59).	<code>date +%M</code> outputs 45.
%S	Second (00-59).	<code>date +%S</code> outputs 30.
%A	Full weekday name.	<code>date +%A</code> outputs Tuesday.
%a	Abbreviated weekday name.	<code>date +%a</code> outputs Tue.
%B	Full month name.	<code>date +%B</code> outputs October.
%b	Abbreviated month name.	<code>date +%b</code> outputs Oct.
%p	AM or PM indicator (12-hour format)	<code>date +%I %p</code> outputs 02 PM.
%I	Hour (01-12, 12-hour format)	<code>date +%I</code> outputs 02.
%T	Time in 24-hour format (<i>HH:MM:SS</i>)	<code>date +%T</code> outputs 14:45:30.
%D	Date in <i>MM/DD/YY</i> format.	<code>date +%D</code> outputs 10/08/24.

6. clear

The clear command in Linux is used to clear the terminal screen. It effectively removes all the text currently displayed, providing a clean slate for new commands and output. It's a simple but essential command for managing the terminal's display.

Simply type the word clear in your terminal and press Enter.

In many terminals, you can achieve the same result by pressing the Ctrl + L keys.

2.5 Text Processing Commands (head, tail, cut, sort, cmp, tr, uniq, wc, tee)

1. head

The head command in Linux is a command-line utility used to display the first few lines (or bytes) of one or more text files. By default, it outputs the first 10 lines of a file. It is a filter command (takes input -> filters it -> provides o/p).

Syntax: head [options] [file(s)]

\$ head state.txt

#above command will display top 10 lines of the file state.txt

\$ head -n 5 state.txt

#above command will display top 5 lines the file state.txt

\$ head -5 state.txt

#above command will display top 5 lines the file state.txt

\$ head -c 6 state.txt

#above command will display first 6 bytes of the file state.txt

\$ head state.txt capital.txt

#above command will display top 10 lines of each file along with file name as header.

\$ head -q state.txt capital.txt

#above command will display top 10 lines of each file without file name as header.

\$ head -v state.txt

-v option will always display file name, even if it is single file.

2. tail

The tail command in Linux is a command-line utility used to display the last few lines (or bytes) of one or more text files. By default, it outputs the last 10 lines of a file. It is a filter command (takes input -> filters it -> provides o/p).

Syntax: tail [options] [file(s)]

\$ tail state.txt

#above command will display last 10 lines of the file state.txt

\$ tail -n 5 state.txt

#above command will display last 5 lines the file state.txt

```
$ tail -5 state.txt
```

#above command will display last 5 lines the file state.txt

```
$ tail -c 6 state.txt
```

#above command will display last 6 bytes of the file state.txt

```
$ tail state.txt capital.txt
```

#above command will display last 10 lines of each file along with file name as header.

```
$ tail -q state.txt capital.txt
```

#above command will display last 10 lines of each file without file name as header.

```
$ tail -v state.txt
```

-v option will always display file name, even if it is single file.

```
$ tail +7 state.txt
```

+7 option will display lines starting from 7th to last.

3. cut

- Both head and tail commands are used to slice a file horizontally.
- A user can slice a file vertically with the cut command.
- It recognized both columns and fields of file(s).

Syntax: cut option [file(s)]

The syntax shows that option is compulsory. Without option, cut command will not work.

Options:

-c (character): It is used to extracts number of characters from the specified file. A user can also print range of characters. A range is one of the following:

n-m: It prints nth to mth characters of each line of input file.

-m: It prints 1th to mth characters of each line of input file.

n-: It prints nth to last characters of each line of input file.

```
$ cut -c2-5 f1.txt
```

above command will display characters 2 to 5 of each line for file f1.txt.

```
$ cut -c-5 f1.txt
```

characters 1st to 5th of each line.

```
$ cut -c7- f1.txt
```

characters 7th to last of each line.

```
$ cut -c-3,4-6,9- f1.txt
```

multiple ranges can be given. Range must be in ascending order.

-d (field-delimiter)

It specifies field delimiter. It uses the <tab> as the default field delimiter, but can also work with a different delimiter.

-f (field)

It specifies the fields list. Like -c option, a user can also use range with this option to extract fields from file. It requires two options -d for the delimiter and -f for specifying the field list.

```
$ cut -d " " -f 2,4 f1.txt
```

will display 2nd and 4th word of each line for f1.txt.

4. sort

Sort is a command that can sort text files alphabetically, numerically, in reverse, and removes duplicates. Sort command sorts the contents of a text file, line by line. Sort is a standard command-line program that prints the lines of its input or concatenation of all files listed in its argument list in sorted order. By default, blank space is the default field separator.

Options	Description
-o	Specifies an output file for the sorted data. Functionally equivalent to redirecting output to a file.
-r	Sorts data in reverse order (descending).
-n	Sorts a file numerically (interprets data as numbers).
-nr	Sorts a file with numeric data in reverse order. Combines -n and -r options.
-k	Sorts a table based on a specific column number.
-c	Checks if the file is already sorted and reports any disorder.

Options	Description
-u	Sorts and removes duplicate lines, providing a unique sorted list.

Examples:

\$ sort file.txt

Above command will sort file.txt in alphabetical ascending order. This command does not actually change the input file, but it displays the sorted content on the screen.

\$ sort f1 f2 f3

Above command will concatenate contents of three files and then will display sorted output.

\$ sort -o sort_file emp_data

Above command will sort emp_data file and store the output into sort_file.

\$ sort -u contacts.txt

Above command will remove duplicate lines and display only unique lines.

\$ sort -t "|" +1 emp_data

Will sort emp_data on 2nd column where columns are separated by "|".

5. cmp

The cmp command in Linux is used to compare two files byte by byte and report the first difference found. If no differences are found, cmp typically remains silent.

Basic Syntax: cmp [options] file1 file2

Compares file1 and file2 byte by byte. Reports the byte and line number where the first difference occurs. If the files are identical, it provides no output.

Key Options:

-c: Displays the differing bytes as characters.

-i N: Skips the first N bytes of the files during comparison.

-l: Lists all differing bytes, showing both the byte number (decimal) and the differing bytes (octal).

When to use cmp:

- To quickly determine if two files are identical.
- To find the exact location of the first difference between two files.
- When you need a byte-by-byte comparison, rather than a line-by-line comparison (like diff).

6. tr

The `tr` command in Linux is a powerful utility for translating or deleting characters from standard input and writing the result to standard output. It's a versatile tool for various text processing tasks, including case conversion, character replacement, and removing unwanted characters.

Core Functionality:

Character Translation: `tr` can replace characters from one set with characters from another set.

Character Deletion: It can remove specific characters from the input.

Squeezing Repeats: It can reduce multiple occurrences of the same character to a single instance.

Case Conversion: It can convert uppercase to lowercase or vice-versa

Common Use Cases:

1. Converting Case:

`tr '[:lower:]' '[:upper:]'` converts lowercase to uppercase.

`tr '[:upper:]' '[:lower:]'` converts uppercase to lowercase.

2. Deleting Characters:

`tr -d '[:punct:]'` deletes all punctuation characters.

`tr -d ' '` deletes spaces.

3. Replacing Characters:

`tr ' ' '\n'` replaces spaces with newline characters.

`tr '-' '_'` replaces hyphens with underscores.

4. Squeezing Repeated Characters:

`tr -s ' '` reduces multiple spaces to a single space.

The basic syntax is: `tr [options] set1 set2`

set1: The set of characters to be translated or deleted.

Set2: The set of characters to replace set1 with (if translation is desired).

Options: Flags that modify the command's behavior (e.g., `-d` for deletion, `-s` for squeezing).

Examples:

```
$ cat file1 | tr [a-z] [A-Z]
```

```
$ cat file1 | tr [:lower:] [:upper:]
tr [:lower:] [:upper:] < file1

$ echo "Welcome To Linux" | tr [:space:] "\t"
$ tr "{}" "()" < file1 > newfile.txt
$ echo "Welcome  To  Linux" | tr -s " "
$ echo "Welcome To Linux" | tr -d W
$ echo "my ID is 73535" | tr -d [:digit:]
```

To complement the char set we use -c option. For example, to remove all characters except digits, you can use the following.

```
$ echo "my ID is 73535" | tr -cd [:digit:]
```

Key Points:

tr works with standard input and standard output, making it useful in pipelines with other commands.

It's a character-based command, not a line-based one like sed.

It's often used for simple text transformations but can be very powerful when combined with other commands.

7. uniq

The uniq command in Linux is a command-line utility that reports or filters out the repeated lines in a file. In simple words, uniq is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines. uniq filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file.

Note: the file must be sorted in order to use uniq command.

Syntax: uniq [OPTIONS] [INPUT_FILE [OUTPUT_FILE]]

Options	Description
-c, --count	Prefix lines by the number of occurrences in the input, followed by a space.
-d, --repeated	Displays duplicate (repeated) lines.
-i, --ignore-case	Ignore differences in case when comparing lines.
-u, --unique	Displays only unique lines. (Removes Duplicate)

Examples:

`uniq f1.txt` (will delete duplicate lines from f1.txt)

`uniq -c f1.txt` (prefixes each line with the number of occurrences in the input)

`uniq -d kt.txt` (only prints duplicate lines)

`uniq -D kt.txt` (prints all duplicate lines, not just one per group)

`uniq -u kt.txt` (prints only unique lines – removes duplicate)

8. **wc**

The command `wc` is used for counting number of lines (in fact, total number of new-line characters), words and characters in a file.

→ Example:

```
$ wc file1
```

```
1 2 12 file1      (1 line, 2 words, 12 characters)
```

→ We can always provide multiple file names to `wc` command.

→ Options

(a) `-l`

This option prints the number of lines present in a file.

With this option `wc` command displays two-columnar output, 1st column shows number of lines present in a file and 2nd itself represent the file name.

(b) `-w`

This option prints the number of words present in a file.

With this option `wc` command displays two-columnar output, 1st column shows number of words present in a file and 2nd is the file name.

(c) `-c`

This option displays count of bytes present in a file.

With this option it displays two-columnar output, 1st column shows number of bytes present in a file and 2nd is the file name.

(d) `-L`

It can be used to print out the length of longest (number of characters) line in a file

9. tee

The tee command, used with a pipe, reads standard input, then writes the output of a program to standard output and simultaneously copies it into the specified file or files. Use the tee command to view your output immediately and at the same time, store it for future use.

Example:

```
$ ps -ef | tee processlist.txt
```

The -e option tells ps to display all processes running on the system, not just those associated with the current terminal. The -f option stands for full-format listing, which includes additional details about each process.

```
$ ls -l | tee -a filelist.txt      (-a; append)
```

Above command will display long list of all files from current directory and also store (-a; append) the same output to filelist.txt file.

2.6 Introduction to Process

1.1 PROCESS BASICS

- A process is simply an instance of a running program.
- A process is said to be born when the program starts execution and remains alive as long as the program is active. After execution is complete, the process is said to die.
- A process also has a name, usually the name of the program being executed.
- For example, when you execute the grep command, a process named grep is created.
- However, a process can't be considered synonymous with a program. When two users run the same program, there's one program on disk but two processes in memory.
- The kernel is responsible for the management of processes.
- It determines the times that are allocated to processes so that multiple processes are able to share CPU resources.
- Files have attributes and so do processes. Some attributes of every process are maintained by the kernel in memory in a separate structure called the **process table**.
- Two important attributes of a process are:
 - The Process-id (PID) Each process is uniquely identified by a unique integer called the Process-id (PID) that is allotted by the kernel when the process is born. We need this PID to control a process, for instance, to kill it.

The Parent PID (PPID) The PID of the parent is also available as a process attribute. When several processes have the same PPID, it often makes sense to kill the parent rather than all its children separately.

1.1.1 The Shell Process

- When you log on to a UNIX system, a process is immediately set up by the kernel.
- This process represents a UNIX command which may be sh (Bourne shell), ksh (Korn shell), csh (C shell) or bash (Bash).
- Any command that you key in is actually the standard input to the shell process.
- This process remains alive until you log out, when it is killed by the kernel.
- The shell maintains a set of environment variables, and you have already encountered some of them like PATH and SHELL.
- The shell's pathname is stored in SHELL, but its PID is stored in a special "variable", \$\$.
- To know the PID of your current shell, type "Echo \$\$" on the shell prompt.

2.7 Process Control Commands (ps, fg, bg, kill, sleep)

What is a Process?

A process is a running instance of a program. Every process in Linux has:

- A PID (Process ID).
- A PPID (Parent Process ID).
- A UID (User ID) of the user who started it.
- A status (Running, Sleeping, Stopped, etc.).

1. ps (Process Status)

Displays information about active processes. It's like Task Manager for the command line.

Syntax: ps [options]

Option	Meaning
a	All users' processes
u	Display user-oriented format
x	Include processes without TTY
e	Show all processes
f	Full-format listing

Examples:

```
ps                # Shows processes in current shell
ps aux            # All system processes in detailed view (BSD style)
ps -ef            # All system processes (UNIX style)
ps -u user1       # Processes of a specific user
```

Explanation of output

Column	Description
PID	Process ID
TTY	Terminal from which process started
TIME	Total CPU time used by process
CMD	Command that launched the process
USER	Owner of the process
%CPU	CPU usage
%MEM	Memory usage

RUNNING JOBS IN BACKGROUND

- A multitasking system lets a user do more than one job at a time. Since there can be only one job in the foreground, the rest of the jobs have to run in the background.
- There are two ways of doing this with the shell's & operator and the nohup command. The nohup permits you to log out while your jobs are running.

2. fg (fore ground) & bg (back ground)

The fg command in Linux is used to bring a background job into the foreground. It allows you to resume a suspended job or a background process directly in the terminal window, so you can interact with it.

The bg command is a useful tool that allows you to manage and move processes between the foreground and background. It's especially helpful when you want to multitask in the terminal by placing a process in the background, enabling you to continue using the terminal for other commands while the process runs quietly in the background.

Syntax: fg [job_spec] / bg [job_spec]

This is the primary use of the fg command, bringing a specified job running in the background back to the foreground. For example, if you create a dummy job using sleep 500, you can bring it back to the foreground by referencing its job number:

\$ jobs

\$ sleep 500

^Z

[1]+ Stopped sleep 500

\$ jobs

[1]+ Stopped sleep 500

\$ bg %1 (%n: Refers to job number n)

```
[1]+ sleep 500 &
```

```
$ jobs
```

```
[1]+  running      sleep 500 &
```

```
$ fg %1
```

```
Sleep 500
```

The fg and bg commands are powerful and essential tools for managing background processes in Linux. It allows users to easily bring background jobs to the foreground for interaction, making multitasking on Linux systems more efficient.

3. kill – Terminate a Process

This command Sends signals to a process to request termination or other actions.

Syntax: kill [-signal] PID

Common Signals:

Signal	Number	Description
SIGTERM	15	Graceful stop (default)
SIGKILL	9	Force stop
SIGSTOP	19	Pause a process
SIGCONT	18	Continue a paused process

Examples:

```
kill 1234           # Graceful stop
kill -9 1234        # Force stop (cannot be ignored)
kill -l             # Lists all signal names and numbers
kill -STOP 2345     # Pause process
kill -CONT 2345     # Resume paused process
```

4. sleep – Delay Execution

Pauses execution of commands for a given time.

Syntax: sleep NUMBER[SUFFIX]

Time Suffixes: If no suffix is given, seconds are assumed.

Suffix	Unit
s	Seconds
m	Minutes
h	Hours
d	Days

Examples:

```
sleep 5      # Pause for 5 seconds
sleep 2m     # Pause for 2 minutes
sleep 1h     # Pause for 1 hour
sleep 3      # 3-second pause before next command
```

Use in Scripts:

```
#!/bin/bash
echo "Starting"
sleep 10
echo "Finished after 10 seconds"
```

→ Job Control Commands summary

Command	Description
jobs	Lists current shell jobs
fg	Bring job to foreground
bg	Resume job in background
kill	Terminate job or process
Ctrl+Z	Suspend foreground job
&	Run command in background

→ Key Takeaways

- Use ps to monitor processes.
- Use &, bg, fg, jobs to manage foreground/background jobs.
- Use kill to stop unresponsive or unnecessary processes.
- Use sleep for timed pauses or delays in automation/scripts.

2.8 Job Scheduling Commands (at, batch, crontab)

Linux provides several commands to schedule tasks for future execution. These are especially useful for automating maintenance tasks, running scripts, or performing backups.

1. at (One time execution)

- As an argument to at command we provide the time the job is to be executed. Then it displays the at> prompt, where we input jobs.

Example:

```
$ at 14:08
at> empawk.sh
ctrl-d
```

o/p: command will be executed using /usr/bin/bash
job 1041188880.a at Sun Dec 29 14:08:00 2002

- The job goes to the queue, and at 2:08 p.m. today, the script file empawk2.sh will be executed.
- `at` shows the job number, the date and time of scheduled execution.
- A user may prefer to redirect the output of the command itself:
`$ at 15:08 empawk2.sh > rep.lst`
- You can also use the **-f option** to take commands from a file.
- However, any error messages that may be generated when executing a program will, in the absence of redirection, continue to be mailed to the user.
- To mail job completion to the user, use the **-m option**.
- The following forms show the use some the words and operators:

```
at 15                24-hour format assumed
at 5pm
at 3:08pm
at noon              at 12:00 hours today
at now+1 year         at current time after one year
at 3:08pm+1 day       at 3:08 p.m. tomorrow
at 15:08 December 18, 2001
at 9am tomorrow
```

- Jobs can listed with the **at -l** command and removed with **at -r**.

2. batch (Execute Batch Queue)

- The **batch** command also schedules job for later execution, but unlike `at`, jobs are executed as soon as the system load permits.
- The command doesn't take any arguments but uses an internal algorithm determine the execution time.
- Batch command adds the job in a special "at" queue.

Example:

```
$ batch < empawk.sh
$ echo "sh long_script.sh" | batch
```

- To list the jobs `at -l` command is used and to remove a job `at -r` command is used.

3. crontab – Repeating Jobs (Cron Jobs)

The `crontab` command in Linux is used to manage cron jobs, which are automated tasks scheduled to run at specific times or intervals. It allows users to create, edit, list, and remove cron jobs that are executed by the cron daemon.

Syntax:

```
$ crontab -e      # Edit crontab file / make new entries
$ crontab -l      # List crontab
$ crontab -r      # Remove crontab
```

Key aspects of crontab:

Cron Jobs: These are commands or scripts that are automatically executed by the cron daemon at predefined times or intervals.

crontab Command: The crontab command is the utility for managing these scheduled tasks.

Cron Daemon: The cron daemon (a background process) is responsible for running the tasks defined in the crontab files.

crontab File: Each user has a crontab file that contains their scheduled tasks.

Crontab entry: Crontab entries consist of five time fields (minute, hour, day of month, month, day of week) followed by the command to execute.

Special Characters: Special characters like * (wildcard), , (multiple values), - (range), and / (step values) can be used in the time fields for flexible scheduling.

Example:

\$ crontab -e (to make a new entry or edit old entries)

```
* * * * * command to run
- - - - -
| | | | |
| | | | | +----- day of week (0 - 6) (Sunday=0)
| | | | | +----- month (1 - 12)
| | | | | +----- day of month (1 - 31)
| | | | | +----- hour (0 - 23)
| | | | | +----- minute (0 - 59)
```

Examples:

Schedule	Cron Entry	Description
Every minute	* * * * * /path/script.sh	Runs every minute
Daily at 5 AM	0 5 * * * /path/backup.sh	Daily at 5:00 AM
Every Sunday	0 9 * * 0 /path/weekly.sh	Sundays at 9:00 AM
Every 15 mins	*/15 * * * * /path/job.sh	Every 15 minutes

→ The result of commands executed by crontab will not show on standard output. You need to redirect it to another file to see it.

```
* * * * * /path/script.sh >> /var/log/myscript.log
```