| Command | |
|---|---|
| **ls** | Lists a directory’s content |
| **pwd** | Shows the current working directory’s path |
| **cd** | Changes the working directory |
| **mkdir** | Creates a new directory |
| **rm** | Deletes a file |
| **cp** | Copies files and directories, including their content |
| **mv** | Moves or renames files and directories |
| **touch** | Creates a new empty file |
| **file** | Checks a file’s type |
| **zip and unzip** | Creates and extracts a ZIP archive |
| **tar** | Archives files without compression in a TAR format |
| **nano, vi, and jed** | Edits a file with a text editor |
| **cat** | Lists, combines, and writes a file’s content as a standard output |
| **grep** | Searches a string within a file |
| **sed** | Finds, replaces, or deletes patterns in a file |
| **head** | Displays a file’s first ten lines |
| **tail** | Prints a file’s last ten lines |
| **awk** | Finds and manipulates patterns in a file |
| **sort** | Reorders a file’s content |
| **cut** | Sections and prints lines from a file |
| **diff** | Compares two files’ content and their differences |
| **tee** | Prints command outputs in Terminal and a file |

| | |
|---|---|
| **locate** | Finds files in a system's database |
| **find** | Outputs a file or folder's location |
| **sudo** | Runs a command as a superuser |
| **su** | Runs programs in the current shell as another user |
| **chmod** | Modifies a file's read, write, and execute permissions |
| **chown** | Changes a file, directory, or symbolic link's ownership |
| **useradd and userdel** | Creates and removes a user account |
| **df** | Displays the system's overall disk space usage |
| **du** | Checks a file or directory's storage consumption |
| **top** | Displays running processes and the system's resource usage |
| **htop** | Works like **top** but with an interactive user interface |
| **ps** | Creates a snapshot of all running processes |
| **uname** | Prints information about your machine's kernel, name, and hardware |
| **hostname** | Shows your system's hostname |
| **time** | Calculates commands' execution time |
| **systemctl** | Manages system services |
| **watch** | Runs another command continuously |
| **jobs** | Displays a shell's running processes with their statuses |
| **kill** | Terminates a running process |
| **shutdown** | Turns off or restarts the system |
| **ping** | Checks the system's network connectivity |
| **wget** | Downloads files from a URL |

| | |
|---|---|
| **curl** | Transmits data between servers using URLs |
| **scp** | Securely copies files or directories to another system |
| **rsync** | Synchronizes content between directories or machines |
| **lfconfig** | Displays the system's network interfaces and their configurations |
| **netstat** | Shows the system's network information, like routing and sockets |
| **traceroute** | Tracks a packet's hops to its destination |
| **nslookup** | Queries a domain's IP address and vice versa |
| **dig** | Displays DNS information, including record types |
| **history** | Lists previously run commands |
| **man** | Shows a command's manual |
| **echo** | Prints a message as a standard output |
| **ln** | Links files or directories |
| **alias and unalias** | Sets and removes an alias for a file or command |
| **cal** | Displays a calendar in Terminal |
| **apt-get** | Manages Debian-based distros package libraries |

### Linux Commands for File and Directory Management

This section will explore basic Linux commands for file and directory management.

## 1. ls command

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

```
ls [/directory/folder/path]
```

```
root@srv:/# ls /directory/folder/path
file1.txt
```

If you remove the path, the **ls** command will show the current working directory's content. You can modify the command using these options:

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

## 2. pwd command

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

```
pwd [option]
```

It supports two options. The **-L** or –**-logical** option prints environment variable content, including symbolic links. Meanwhile, **-P** or **–physical** outputs the current directory's actual path.

```
root@srv:/directory/folder/path# pwd
/directory/folder/path
```

## 3. cd command

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

```
cd /directory/folder/path
```

```
root@srv:/# cd /directory/folder/path
root@srv:/directory/folder/path#
```

Depending on your current location, it requires either the full path or the directory name. For example, omit **/username** from **/username/directory/folder** if you are already within it.

Omitting the arguments will take you to the home folder. Here are some navigation shortcuts:

- **cd ~[username]** – goes to another user's home directory.
- **cd ..** – moves one directory up.
- **cd-** – switches to the previous directory.

## 4. mkdir command

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

```
mkdir [option] [directory_name]
```

To create a folder within a directory, use the path as the command parameter. For example, **mkdir music/songs** will create a **songs** folder inside **music**. Here are several common **mkdir** command options:

- **-p** – creates a directory between two existing folders. For example, **mkdir -p Music/2024/Songs** creates a new **202**4 directory.
- **-m** – sets the folder permissions. For instance, enter **mkdir -m777 directory** to create a directory with read, write, and execute permissions for all users.
- **-v** – prints a message for each created directory.

```
root@srv:/# mkdir -v new-folder
mkdir: created directory 'new-folder'
```

## 5. rmdir command

Use the **rmdir** command to delete an empty directory in Linux. The user must have **sudo** privileges in the parent directory. Here's the syntax:

```
rmdir [option] directory_name
```

If the folder contains a subdirectory, the command will return an error. To force delete a non-empty directory, use the **-p** option.

## 6. rm command

Use the rm command to permanently delete files within a directory. Here's the general syntax:

rm [filename1] [filename2] [filename3]

Adjust the number of files in the command according to your needs. If you encounter an error, ensure you have the **write** permission in the directory.

To modify the command, add the following options:

- **-i** – prompts a confirmation before deletion.
- **-f** – allows file removal without a confirmation.

- **-r** – deletes files and directories recursively.

**Warning!** Use the **rm** command with caution since deletion is irreversible. Avoid using the **-r** and **-f** options since they may wipe all your files. Always add the **-i** option to avoid accidental deletion.

## 7. cp command

Use the **cp** command to copy files or directories, including their content, from your current location to another. It has various use cases, such as:

> Copying one file from the current directory to another folder. Specify the file name and target path:

```
cp filename.txt /home/username/Documents
```

> Duplicating multiple files to a directory. Enter the file names and the destination path:

```
cp filename1.txt filename2.txt filename3.txt /home/username/Documents
```

> Copying a file's content to another within the same directory. Enter the source and the destination file:

```
cp filename1.txt filename2.txt
```

> Duplicating an entire directory. Pass the **-R** flag followed by the source and destination directory:

```
cp -R /home/username/Documents /home/username/Documents_backup
```

## 8. mv command

Use the **mv** command to move or rename files and directories. To move items, enter the file name followed by the destination directory:

```
mv filename.txt /home/username/Documents
```

Meanwhile, use the following syntax to rename a file in Linux with the **mv** command:

```
mv old_filename.txt new_filename.txt
```

## 9. touch command

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

```
touch [option] /home/directory/path/file.txt
```

If you omit the path, the command will create the item in the current folder. You can also use **touch** to generate and modify a timestamp in the Linux command line.

## 10. file command

The file command lets you check a file type – whether it is a text, image, or binary. Here's the syntax:

```
file filename.txt
```

To bulk-check multiple files, list them individually or use their path if they are in the same directory. Add the **-k** option to display more detailed information and **-i** to show the file's MIME type.

## 11. zip, unzip commands

The **zip** command lets you compress items into a **ZIP** filewith the optimal compression ratio. Here's the syntax:

```
zip [options] zipfile file1 file2….
```

For example, this command compresses **note.txt** into **archive.zip** in the current working directory:

```
zip archive.zip note.txt
```

Use the **unzip** command to extract the compressed file. Here's the syntax:

unzip [option] file_name.zip

## 12. tar command

The tar command archives multiple items into a **TAR** file– a format similar to **ZIP** with optional compression. Here's the syntax:

```
tar [options] [archive_file] [target file or directory]
```

For instance, enter the following to create a new **newarchive.tar** archive in the **/home/user/Documents** directory:

```
tar -cvzf newarchive.tar /home/user/Documents
```

### Linux Commands for Text Processing and Searching

The following section explores several Linux commands for processing and searching text.

## 13. nano, vi, jed commands

Linux lets users edit files using a text editor like nano, **vi**, or **jed**. While most distributions include **nano** and **vi**, users must install **jed** manually. All these tools have the same command syntax:

```
nano filename
```

```
vi filename
```

```
jed filename
```

If the target file doesn't exist, these editors will create one. We recommend **nano** if you want to quickly edit text files. Meanwhile, use **vi** or **jed** for scripting and programming.

## 14. cat command

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

```
cat filename.txt
```

There are various ways to use the **cat** command:

- **cat > filen.txt** –creates a new file.
- **cat file1.txt file2.txt > file3.txt** –merges **file1.txt** with **file2.txt** and stores the output in **filename3.txt**.
- **tac file.txt** – displays content in reverse order.

## 15. grep command

The **global regular expression** or **grep** command lets you find a word by searching the content of a file. This Linux command prints all lines containing the matching strings, which is useful for filtering large log files.

For example, to display lines containing **blue** in the **notepad.txt** file, enter:

```
grep blue notepad.txt
```

## 16. sed command

The **sed** command lets you find, replace, and delete patterns in a file without using a text editor. Here's the general syntax:

```
sed [option] 'script' input_file
```

The script contains the searched regular expression pattern, the replacement string, and subcommands. Use the **s** subcommand to replace matching patterns or **d** to delete them.

At the end, specify the file containing the pattern to modify. Here's an example of a command that replaces **red** in **colors.txt** and **hue.txt** with **blue**:

```
sed 's/red/blue' colors.txt hue.txt
```

## 17. head command

The **head** command prints the first ten lines of a text file or piped data in your command-line interface. Here's the general syntax:

```
head [option] [file]
```

For instance, to view the first ten lines of **note.txt** in the current directory, enter:

```
head note.txt
```

The **head** command accepts several options, such as:

- **-n** – changes the number of lines printed. For example, **head -n 5** showsthe first five lines.
- **-c** – prints the file's first customized number of bytes.
- **-q** – disables headers specifying the file name.

```
root@srv:/directory/folder/path# head -n 5 file1.txt
This is the first line
This is the second line
This is the third line
This is the fourth line
This is the fifith line
```

## 18. tail command

The **tail** command displays the last ten lines of a file, which is useful for checking new data and errors. Here's the syntax:

```
tail [option] [file]
```

For example, enter the following to show the last ten lines of the **colors.txt** file:

```
tail -n colors.txt
```

## 19. awk command

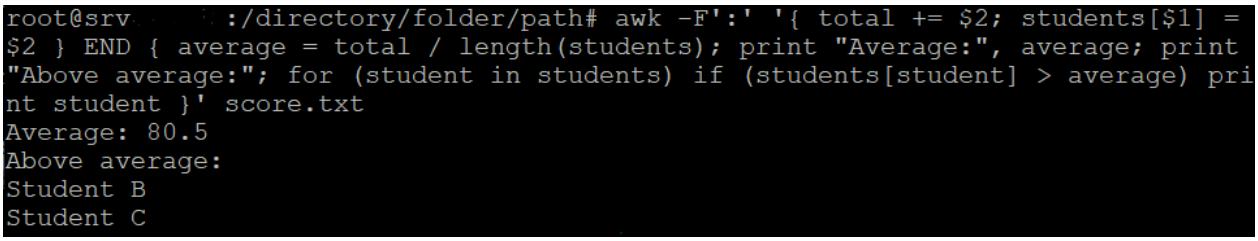
The **awk** command scans regular expression patterns in a file to retrieve or manipulate matching data. Here's the basic syntax:

```
awk '/regex pattern/{action}' input_file.txt
```

The action can be mathematical operations, conditional statements like **if**, output expressions such as **print**, and a **delete** command. It also contains the **$n** notation, which refers to a field in the current line.

To add multiple actions, list them based on the execution order, separated using semicolons. For example, this command contains mathematical, conditional, and output statements:

```
awk -F':' '{ total += $2; students[$1] = $2 } END { average = total /
length(students); print "Average:", average; print "Above average:"; for (student
in students) if (students[student] > average) print student }' score.txt
```




## 20. sort command

The **sort** command rearranges lines in a file in a specific order. It doesn't modify the actual file and only prints the result as Terminal outputs. Here's the syntax:

```
sort [option] [file]
```

By default, this command will sort the lines in alphabetical order, from A to Z. To modify the sorting, use these options:

- **-o** – redirects the command outputs to another file.
- **-r** – reverses the sorting order to descending.
- **-n** – sorts the file numerically.
- **-k** – reorders data in a specific field.

```
root@srv:/directory/folder/path# sort file1.txt
Line A
Line B
Line C
Line D
```

## 21. cut command

The **cut** command retrieves sections from a file and prints the result as Terminal outputs. Here's the syntax:

```
cut [option] [file]
```

Instead of a file, you can use data from standard input. To determine how the command sections the line, use the following options:

- **-f** – selects a specific field.
- **-b** – cuts the line by a specified byte size.
- **-c** – sections the line using a specified character.
- **-d** – separates lines based on delimiters.

You can combine these options, use a range, and specify multiple values. For example, this command extracts the third to fifth field from a comma-separated list:

```
cut -d',' -f3-5 list.txt
```

```
root@srv:/directory/folder/path# cut -d',' -f3-5 list.txt
 item3, item4, item5
 item3, item4, item5
 item3, item4, item5
 item3, item4, item5
```

## 22. diff command

The **diff** command compares two files' content and outputs the differences. It is used to alter a program without modifying the code. Here's the general format:

```
diff [option] file1 file2
```

Below are some acceptable options:

- **-c** – displays the difference between two files in a context form.
- **-u** – shows the output without redundant information.
- **-i** – makes the **diff** command case insensitive.

```
root@srv:/directory/folder/path# diff list.txt list2.txt
1,4c1,4
< Item1, item2, item3, item4, item5
< Item1, item2, item3, item4, item5
< Item1, item2, item3, item4, item5
< Item1, item2, item3, item4, item5
---
> Item2, item2, item3, item4, item5
> Item1, item2, item4, item4, item5
> Item1, item3, item3, item4, item5
> Item5, item3, item3, item4, item5
```

## 23. tee command

The **tee** command writes the user's input to Terminal's output and files. Here's the basic syntax:

```
command | tee [option] file1
```

For example, the following pings Google and prints the output in Terminal, **ping_result.txt**,and the **19092024.txt** file:

```
ping google.com | tee ping_result.txt 19092024.txt
```

## 24. locate command

The **locate** command lets you find a file in the database system. Add the **-i** option to turn off case sensitivity and an asterisk (*) to find content with multiple keywords. For example:

```
locate -i school*note
```

The command searches for files containing **school** and **note**, regardless of their letter case.

## 25. find command

Use the **find** command to search for files within a specific directory. Here's the syntax:

```
find [option] [path] [expression]
```

For example, to find a file called **file1.txt** within the **directory** folder and its subdirectories, use this command:

```
find /home -name file1.txt
```

```
root@srv:/# find /directory -name file1.txt
/directory/folder/path/file1.txt
```

If you omit the path, the command will search the current working directory. You can also find directories using the following:

```
find ./ -type d -name directoryname
```

**Linux Commands for User and Permission Management**

Below are several Linux commands for managing the system's users and permissions.

## 26. sudo command

**Superuser do** or **sudo** is one of the most basic commands in Linux. It runs your command with administrative or root permissions. Here's the general syntax:

```
sudo (command)
```

When you run a sudo command, Terminal will request the root password. For example, this snippet runs **useradd** with the superuser privilege:

```
sudo useradd username
```

You can also add an option, such as:

- **-k** – invalidates the timestamp file.
- **-g** – executes commands as a specified group name or ID.
- **-h** – runs commands on the host.

**Warning!** Running a command with sudo privileges can modify all aspects of your system. Since misusing it may break your system, run the command with caution and only if you understand its possible repercussions.

## 27. su command

The **su** command lets you run a program in the Linux shell as a different user. It is useful to connect via SSH while the root user is disabled. Here's the syntax:

```
su [options] [username [argument]]
```

Without any option or argument, thiscommand runs through **root** and prompts you to use the **sudo** privileges temporarily. Some options are:

- **-p** –keeps the same shell environment, consisting of **HOME**, **SHELL**, **USER**, and **LOGNAME**.

- **-s** – lets you specify another shell environment to run.
- **-l** –runs a login script to switch users. It requires you to enter the user's password.

To check the current shell's user account, run the **whoami** command:

```
root@srv:/# su user1
$ whoami
user1
```

## 28. chmod command

The **chmod** command modifies directory or file permissions in Linux. Here's the basic syntax:

```
chmod [option] [permission] [file_name]
```

In Linux, each file is associated with three user classes – **owner**, **group member**, and **others**. It also has three permissions – **read**, **write**, and **execute**. If an owner wants to grant all permissions to every user, the command looks like this:

```
chmod -rwxrwxrwx note.txt
```

## 29. chown command

The **chown** command lets you change a file, directory, or symbolic link's ownership to the specified username. Here's the syntax:

```
chown [option] owner[:group] file(s)
```

For example, to make **linuxuser2** the owner of **filename.txt**, use:

```
chown linuxuser2 filename.txt
```

## 30. useradd, userdel commands

Use **useradd** to create a new Linux user account and change its password with the **passwd** command. Here are the syntaxes:

```
useradd [option] username
```

```
passwd username
```

Both the **useradd** and **passwd** commands require sudo privileges. To delete a user, use the **userdel** command:

```
userdel username
```