



Top 20 Linux Commands in 2023

Linux is an Operating System that can do many things. You can use the command line to type commands and get the task done from Linux. Linux has many commands, but you don't need to know them all. In this article, I will show you the **top 20 Linux commands** that are very useful. In simple words, these commands will help you to work with files, directories, user permissions, and installing packages.

Table of Contents

List of Top 20 Linux Commands	2
1. pwd command	2
2. echo command	2
3. su command	3
4. sudo command	3
5. clear command	4
6. exit command	5
7. cp command	6
8. mv command	7
9. rm command	8
10. grep command	9
11. cat command	10
12. head command	11
13. ls command	11
14. cd command	12
15. sort command	12
16. mkdir command	14
17. touch command	14
18. chmod command	15
19. install command	16
20. tar command	17
Conclusion	18

List of Top 20 Linux Commands

Here, I have selected the **top 20 Linux commands** based on working suitability with files, directories, user permissions, and installing packages.

1. pwd command

The **pwd command** in Linux is a shell building that prints the current working directory path, starting from the **root(/)**. It is one of the most basic and frequently used commands in **Linux**.

Syntax

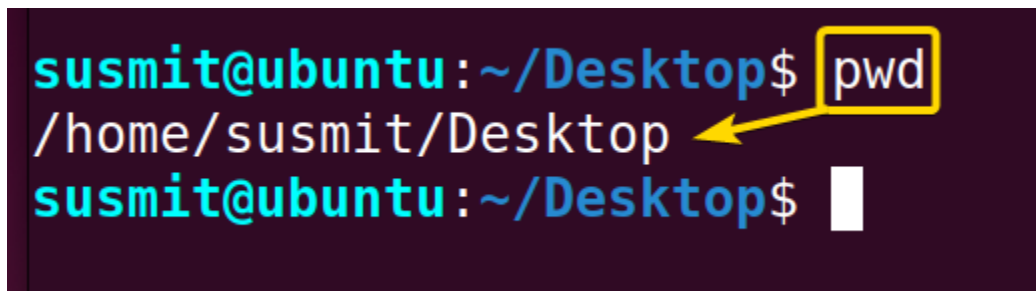
pwd [OPTION]...

Useful Options

- **-L/ --logical**: Uses PWD from the environment, even if it contains symlinks.
- **-P/--physical**: Avoids all symlinks.

Example

To print the current working directory, run “pwd” command in the [terminal](#).



```
susmit@ubuntu:~/Desktop$ pwd
/home/susmit/Desktop
susmit@ubuntu:~/Desktop$
```

A terminal window screenshot with a dark purple background. The prompt is 'susmit@ubuntu:~/Desktop\$'. The command 'pwd' is entered and highlighted with a yellow box. The output is '/home/susmit/Desktop', with a yellow arrow pointing from the box to the output. The prompt is then shown again as 'susmit@ubuntu:~/Desktop\$' with a white cursor.

To learn more read: [The “pwd” Command in Linux \[4 Practical Examples\]](#)

2. echo command

The **echo command** in **Linux** is a built-in feature that prints out arguments as the standard output. This command is commonly used to display text strings or command results as messages.

Syntax

echo [SHORT-OPTION]... [STRING]...

Useful Options

- **-n**: Does not output the trailing newline.
- **-e**: Enables interpretation of backslash escapes.
- **-E**: Disables the interpretation of backslash escapes (default).

Example

To echo a text from standard input, run **echo ‘Hello World!’** in the command prompt. It will echo the **‘Hello World!’** text on the terminal.

```
susmit@ubuntu:~/Desktop$ echo 'Hello World!'
Hello World!
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “echo” Command in Linux \[7 Practical Examples\]](#)

3. su command

The **su command** in **Linux** is used to switch to another user, by default the root user. It stands for **switch** user or **substitute** user. However, this command requires the password of the user to switch to unless it is run with **sudo** privileges.

Syntax

```
su [options] [-] [user [argument...]]
```

Useful Options

- **-c/--command=command**: Passes the command to the shell with the -c option.
- **-f/--fast**: Passes -f to the shell, which may or may not be useful, depending on the shell.
- **-g/--group=group**: Specifies the primary group. This option is available to the root user only.
- **-h/--help**: Shows the help file for the su command.
- **-l/--login**: Starts as a login shell with an environment similar to a real login.
- **-p/--preserve-environment**: Preserves the shell environment.
- **-s/--shell**: Allows you to specify a different shell environment to run in.

Example

To switch the user from current user to other_user, run the ‘**su other_user**’ in the command prompt.

```
susmit@ubuntu:~/Desktop$ su other_user
Password:
other_user@ubuntu:~/Desktop$
```

To learn more read: [The “su” Command in Linux \[6 Practical Examples\]](#)

4. sudo command

The **sudo command** in **Linux** allows you to run programs as another user, by default the **root user**. It stands for superuser do or substitutes user do. The sudo command is more secure than the **su** command because it does not require sharing the **root password** and it can grant limited administrative privileges to individual users.

Syntax

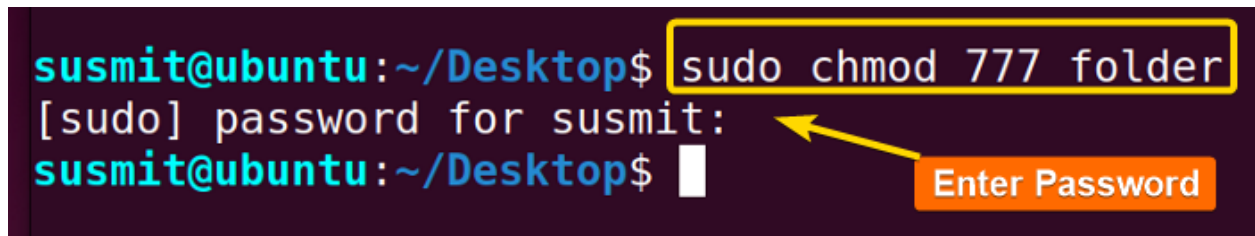
sudo [OPTION]... [COMMAND] [ARG]...

Useful Options

- **-D directory/–chdir=directory:** Executes the command in the specific directory.
- **-e:** Edits one or multiple files instead of executing commands.
- **-k/–reset-timestamp:** Invalidates the user’s cached credentials.
- **-l/–list:** Lists the allowed (and forbidden) commands for the invoking user on the current host.
- **-u user/–user=user:** Executes the command as a user other than the specific default user.
- **-v/–validate:** Updates the user’s cached credentials, authenticating the user if necessary.

Example

To change permission settings **availling sudo privilege** execute the ‘**sudo chmod 777 folder**’ command into the terminal.



```
susmit@ubuntu:~/Desktop$ sudo chmod 777 folder
[sudo] password for susmit:
susmit@ubuntu:~/Desktop$
```

An orange button labeled "Enter Password" has a yellow arrow pointing to the password prompt in the terminal screenshot.

To learn more read: [The “sudo” Command in Linux \[8 Practical Examples\]](#)

5. clear command

The **clear command** in **Linux** is a standard Unix command that is used to clear the terminal screen. It looks for a terminal type in the environment and figures out how to clear the screen from the terminal database.

Syntax

clear [-Ttype] [-V] [-x]

Useful Options

- **-Ttype:** Indicates the type of terminal. Normally this option is unnecessary because the default is taken from the environment variable TERM. If -T is specified, then the shell variables LINES and COLUMNS will also be ignored.
- **-V:** Reports the version of ncurses which was used in this program, and exits.
- **-x:** Does not attempt to clear the terminal's scrollbar buffer using the extended “E3” capability.

Example

To clear the previous command and its output from the terminal, run the ‘**clear**’ command into the terminal.

Before executing the clear command.

```
susmit@ubuntu:~/Desktop$ whoami
susmit
susmit@ubuntu:~/Desktop$ pwd
/home/susmit/Desktop
susmit@ubuntu:~/Desktop$
```

After executing the clear command:

```
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “clear” Command in Linux \[3 Practical Examples\]](#)

6. exit command

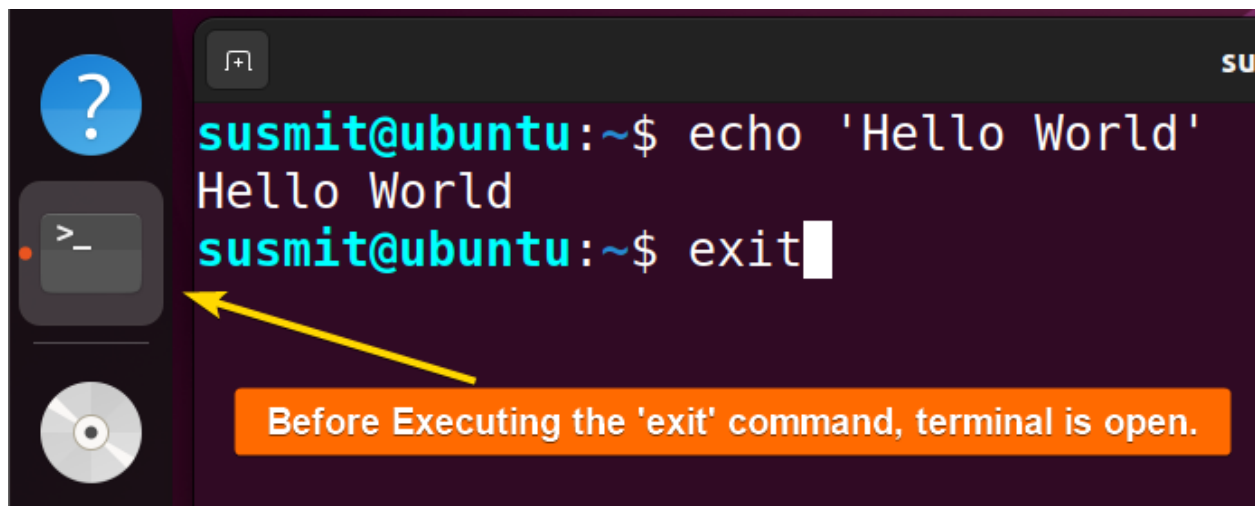
The **exit command** in Linux is a built-in command that is used to exit the shell where it is currently running. It can take one optional parameter as [N] and exits the shell with a return status of N. If N is not provided, then it returns the status of the last command that was executed.

Syntax

exit

Example

To close the current terminal, run the 'exit' command into the terminal. The following image shows that the terminal is open.



The image shows a terminal window with a dark background. The prompt is `susmit@ubuntu:~$`. The user enters `echo 'Hello World'` and the output is `Hello World`. Then the user enters `exit` and the prompt changes to `su`. A yellow arrow points to the `exit` command. An orange box at the bottom contains the text: "Before Executing the 'exit' command, terminal is open."

```
susmit@ubuntu:~$ echo 'Hello World'
Hello World
susmit@ubuntu:~$ exit
su
```

Before Executing the 'exit' command, terminal is open.

The following image shows that the terminal is closed after executing the exit command.



To learn more read: [The “exit” Command in Linux \[5 Practical Examples\]](#)

7. cp command

The **cp command** in **Linux** is a command line utility for **copying** files and directories on **Unix** and **Linux** systems. The cp command can accept one or more source files or directories and a single destination file or directory as arguments.

Syntax

```
cp [OPTION]... SOURCE... DIRECTORY
```

Useful Options

- **-i/--interactive:** Prompts for confirmation before overwriting an existing file.
- **-l/--link:** Hard links files instead of copying.
- **-L:** Follows symbolic links in SOURCE.
- **-n/--no-clobber:** Does not overwrite an existing file.
- **-p/--preserve:** Preserves the specified attributes such as mode, ownership, timestamps, context, links, xattr, and all.
- **-R/-r/ -recursive:** Copies directory recursively.
- **-s/ -symbolic:** Create a symbolic link instead of copying.
- **-t:** Copies all source arguments into the directory.
- **-T:** No target directory. Treats DEST as a normal File.
- **-u/ -update:** Copies only when the SOURCE file is newer than the DEST file.

Example

To copy **file1.txt** from **folder1** to **folder2** execute the '**cp folder1/file1.txt folder2**' command into the terminal.

```
susmit@ubuntu:~/Desktop$ ls folder1
file1.txt
susmit@ubuntu:~/Desktop$ ls folder2
susmit@ubuntu:~/Desktop$ cat folder1/file1.txt
hi
susmit@ubuntu:~/Desktop$ cp folder1/file1.txt folder2
susmit@ubuntu:~/Desktop$ ls folder2
file1.txt
susmit@ubuntu:~/Desktop$ cat folder2/file1.txt
hi
susmit@ubuntu:~/Desktop$
```

file1.txt has been copied to folder2.

To learn more read: [The “cp” Command in Linux \[6 Practical Examples\]](#)

8. mv command

The **mv** command in **Linux** is a command line utility for moving and renaming files and directories on **Unix** and **Linux** systems. The mv command can accept one or more source files or directories and a single destination file or directory as arguments.

Syntax

mv [OPTION]... SOURCE... DIRECTORY

Useful Options

- **-f/--force**: Overwrites an existing file without prompting.
- **-i/--interactive**: Prompts before overwriting.
- **-n/--no-clobber**: Does not overwrite an existing file.
- **-u/--update**: Moves only when the source file is newer than the destination file or when the destination file is missing.
- **-v/--verbose**: Explains what is being done.

Example

To move the **file2.txt** file from the current directory to the **folder** directory inside the current directory, execute the '**mv file2.txt folder**' command into the terminal.

```

susmit@ubuntu:~/Desktop$ ls -l
total 12
-rw-rw-r-- 1 susmit susmit 293 May 24 12:12 file1.txt
-rw-rw-r-- 1 susmit susmit 3 May 24 12:43 file2.txt
drwxrwxr-x 2 susmit susmit 4096 May 24 12:47 folder
susmit@ubuntu:~/Desktop$ ls -l folder
total 4
-rw-rw-r-- 1 susmit susmit 3 May 24 12:47 file3.txt
susmit@ubuntu:~/Desktop$ mv file2.txt folder
susmit@ubuntu:~/Desktop$ ls -l
total 8
-rw-rw-r-- 1 susmit susmit 293 May 24 12:12 file1.txt
drwxrwxr-x 2 susmit susmit 4096 May 24 12:47 folder
susmit@ubuntu:~/Desktop$ ls -l folder
total 8
-rw-rw-r-- 1 susmit susmit 3 May 24 12:43 file2.txt
-rw-rw-r-- 1 susmit susmit 3 May 24 12:47 file3.txt
susmit@ubuntu:~/Desktop$

```

file2.txt has been moved into folder.

To learn more read: [The “mv” Command in Linux \[8 Practical Examples\]](#)

9. rm command

The **rm command** in **Linux** is a command line utility for removing files and directories on **Unix** and **Linux** systems. The **rm** command can accept one or more files or directories as arguments.

Syntax

rm [OPTION]... [FILE]...

Useful Options

- **-d/--dir:** Removes an empty directory.
- **-f/--force:** Ignores nonexistent files and arguments and never prompts the user.
- **-i/--interactive:** Prompts for confirmation before removing each file or directory.
- **-r/-R/--recursive:** Removes directories and their contents recursively.
- **-I:** Similar to the **-i** option but less prominent as only shows a prompt for every three files or recursive removal.

Example

To remove the **file2.txt** file run the '**rm file2.txt**' command into the terminal.


```
susmit@ubuntu:~/Desktop$ ls -l
total 8
-rw-rw-r-- 1 susmit susmit 293 May 24 12:12 file1.txt
-rw-rw-r-- 1 susmit susmit   3 May 24 12:35 file2.txt
susmit@ubuntu:~/Desktop$ rm file2.txt
susmit@ubuntu:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 susmit susmit 293 May 24 12:12 file1.txt
susmit@ubuntu:~/Desktop$
```

file2.txt has been removed successfully.

To learn more read: [The “rm” Command in Linux \[7 Practical Examples\]](#)

10. grep command

The **grep command** in **Linux** is a command line utility for **searching** files and directories for a string or a pattern of characters. The **grep** command can accept one or more files or directories as arguments or read from the standard input.

Syntax

```
grep [OPTION...] PATTERNS [FILE...]
```

Useful Options

- **-c/--count**: Does not print any matches but rather the total occurring number.
- **-i/--ignore-case**: Ignores case distinctions in both the pattern and the input files.
- **-r/--recursive**: Reads all files under each directory, recursively.
- **-v/--invert-match**: Inverts the sense of matching, to select non-matching lines.
- **-V/--version**: Outputs the version number of grep and exit.
- **-w/--word-regexp**: Searches for a whole word.

Example

To print only 'travelling' containing sentences from the **file1.txt** run the '**grep travelling file1.txt**' into the terminal.

```
susmit@ubuntu:~/Desktop$ cat file1.txt
I enjoy travelling.
Travelling broadens horizons.
Discovering new places is exciting.
Travelling rejuvenates the soul.
Adventure awaits while travelling.
Travelling creates unforgettable memories.
Exploring different cultures through travelling is enriching.
Travelling is a wonderful escape.
susmit@ubuntu:~/Desktop$ grep travelling file1.txt
I enjoy travelling
Adventure awaits while travelling
Exploring different cultures through travelling is enriching.
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “grep” Command in Linux \[10+ Practical Examples\]](#)

11. cat command

The **cat** command in **Linux** is a command line utility for **concatenating** and **displaying** files and standard input. The **cat** command can accept zero or more files or directories as **arguments** or **read** from the standard input.

Syntax

```
cat [OPTION]... [FILE]...
```

Useful Options

- **-E/--show-ends**: Displays \$ at the end of each line.
- **-n/--number**: Displays line numbers when utilized.
- **-s/--squeeze-blank**: Suppresses repeated empty output lines.
- **-T/--show-tabs**: Displays TAB characters as ^I.

Example

To print the contents of a file named **file1.txt**, run the '**cat file1.txt**' command into the terminal.

```
susmit@ubuntu:~/Desktop$ cat file1.txt
Hello!
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “cat” Command in Linux \[10 Practical Examples\]](#)

12. head command

The **head command** in **Linux** is used to print the first lines of one or more files or piped data to standard output in Linux and other Unix-like operating systems. It is the opposite of the [tail command](#), which prints the last lines of a file or piped data. By default, the head command prints the first 10 lines of the specified files or data.

Syntax

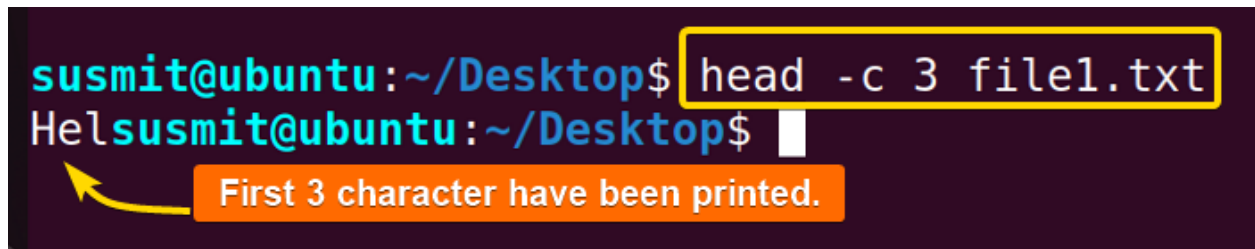
```
head [OPTION]... [FILE]...
```

Useful Options

- **-c/--bytes=[-]NUM:** Prints the first NUM bytes of each file; with the leading '-', print all but the last NUM bytes of each file.
- **-n/--lines=[-]NUM:** Prints the first NUM lines instead of the first 10; with the leading '-', print all but the last NUM lines of each file.

Example

To print only the first three characters of the file1.txt file, execute the '**head -c 3 file1.txt**' command into the terminal.



```
susmit@ubuntu:~/Desktop$ head -c 3 file1.txt
Hel
```

First 3 character have been printed.

To learn more read: [The “head” Command in Linux \[7 Practical Examples\]](#)

13. ls command

The **ls command** in **Linux** is a command line utility for **listing** files and directories on **Unix** and **Linux** systems. The ls command can accept zero or more files or directories as arguments or list the current working directory by default.

Syntax

```
ls [OPTION]... [FILE]...
```

Useful Options

- **-a:** Lists all files and directories, including hidden ones that start with a dot (.).
- **-d:** Only lists the current directory, not its contents.
- **-h:** With option -l it shows file size in human-readable format(i.e. 1K, 100M, 10G).
- **-l:** Lists files and directories in a long listing format, showing file type, permissions, owner, group, size, date, and name.
- **-n:** Similar to option -l but it lists numerically.
- **-R:** Lists all subdirectories recursively.
- **-S:** Displays in a sorted manner by size(largest first).
- **-t:** Lists contents in a sorted manner by time(newest first).

Example

Type the 'ls -l' command into the terminal to print all the contents of the current directory.

```
susmit@ubuntu:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 susmit susmit 7 May 23 13:45 file1.txt
susmit@ubuntu:~/Desktop$
```

Details of file1.txt has been printed

To learn more read: [The "ls" Command in Linux \[7+ Practical Examples\]](#)

14. cd command

The **cd command** in **Linux** is used to change the **current working directory** in Linux and other Unix-like operating systems. It is one of the most basic and frequently used commands when working on the Linux terminal.

Syntax

cd [OPTIONS]

Useful Options

- **-L** : Follows symbolic links. This is the default behavior of cd.
- **-P** : Don't follow symbolic links. This means that if you try to navigate to a symlink that points to a directory, cd will change into the directory instead of the symlink.

Example

To go one directory backward, run the 'cd ..' command into the terminal.

```
susmit@ubuntu:~/Desktop$ pwd
/home/susmit/Desktop
susmit@ubuntu:~/Desktop$ cd ..
susmit@ubuntu:~$ pwd
/home/susmit
susmit@ubuntu:~$
```

Previous working directory.

Current working directory.

To learn more read: [The "cd" Command in Linux \[6 Practical Examples\]](#)

15. sort command

The **sort command** in **Linux** is used to sort the contents of a text file, line by line. It supports sorting alphabetically, in reverse order, by number, by month, and can also remove duplicates.

Syntax

sort [OPTION]... [FILE]...

Useful Options

- **-c/--check**: Checks whether a file is sorted or not.
- **-f/--ignore-case**: Makes sort case insensitive.
- **-k/--key**: Sorts by key.
- **-M/--month-sort**: Sorts by month.
- **-n**: Sorts numerically, comparing according to string numerical value.
- **-o**: Writes the result to an output file instead of standard output.
- **-R/--random-sort**: Shuffles the contents of the file.
- **-r**: Reverses the result of comparisons, i.e., sort in descending order.
- **-u/--unique**: Removes duplicates.

Example

To sort the contents of the **letters** file, run the '**sort letters**' command into the terminal.

```
susmit@ubuntu:~/Desktop$ cat letters
d
a
c
f
e
b
susmit@ubuntu:~/Desktop$ sort letters
a
b
c
d
e
f
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “sort” Command in Linux \[16 Practical Examples\]](#)

16. mkdir command

The **mkdir command** in **Linux** is used to create directories. This command can create multiple directories at once as well as set the permissions for the directories. It is important to note that the user executing this command must have enough permission to create a directory in the parent directory, or he/she may receive a 'permission denied' error.

Syntax

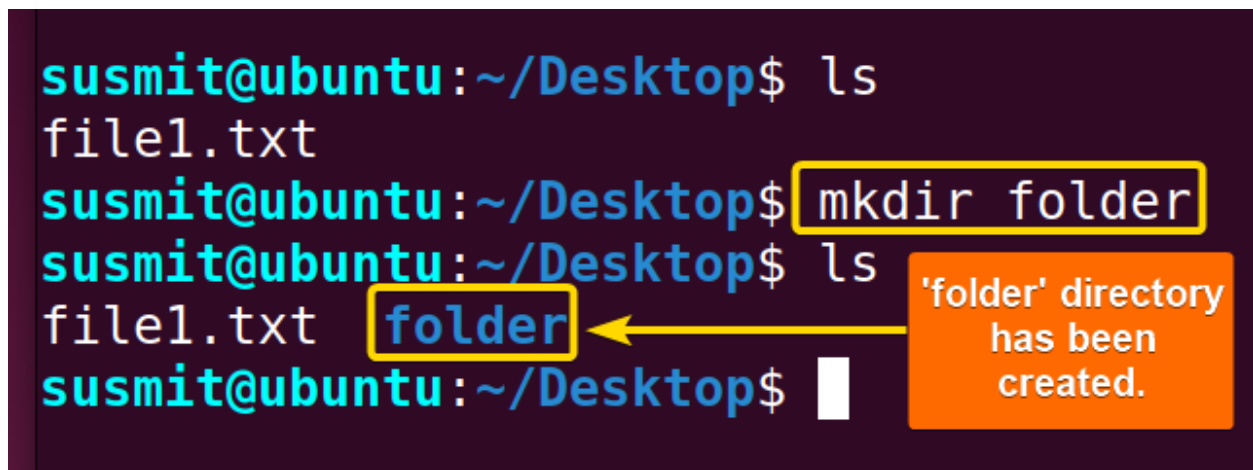
```
mkdir [OPTION]... DIRECTORY...
```

Useful Options

- **-m/--mode=MODE:** Sets the file mode (permissions) for the created directories. The syntax of the mode is the same as the `chmod` command.
- **-p/--parents:** Creates parent directories as necessary. If the directories exist, no error is specified.
- **-v/--verbose:** Displays a message for every directory created.

Example

To create a directory named **folder** in the current directory, execute the '**mkdir folder**' into the command line interface.



```
susmit@ubuntu:~/Desktop$ ls
file1.txt
susmit@ubuntu:~/Desktop$ mkdir folder
susmit@ubuntu:~/Desktop$ ls
file1.txt folder
susmit@ubuntu:~/Desktop$
```

The screenshot shows a terminal window with the following sequence of commands and output:

- `ls` command shows `file1.txt`.
- `mkdir folder` command is executed.
- `ls` command is executed again, showing `file1.txt` and `folder`.

An orange callout box with a yellow arrow points to the `folder` entry in the second `ls` output, containing the text: `'folder' directory has been created.`

To learn more read: [The "mkdir" Command in Linux \[6+ Practical Examples\]](#)

17. touch command

The **touch command** in **Linux** is used to create, change and modify the timestamps of a file or a set of files. It can also be used to create an empty file without any content. The touch command can change the access time, and modification time, and change the time of files and folders.

Syntax

```
touch [OPTION]... FILE...
```

Useful Options

- **-a:** Changes only the access time of the file or folder.
- **-c/--no-create:** Does not create any file if it does not exist. Avoid creating files.

- **-d/--date=STRING:** Changes both the access and modification times using a date string.
- **-m:** Changes only the modification time of the file or folder.

Example

To update the **modification time** of a file named **file1.txt**, run the **'touch file1.txt'** command into the terminal.

```

susmit@ubuntu: ~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 susmit susmit 293 May 23 16:21 file1.txt
susmit@ubuntu:~/Desktop$ touch file1.txt
susmit@ubuntu:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 susmit susmit 293 May 24 12:12 file1.txt
susmit@ubuntu:~/Desktop$

```

Annotations in the image:

- An orange box with an arrow pointing to the initial timestamp "May 23 16:21" contains the text "Initial modification time."
- A yellow box highlights the command "touch file1.txt".
- A yellow box highlights the updated timestamp "May 24 12:12".
- An orange box with an arrow pointing to the updated timestamp contains the text "Modification time has been changed."

To learn more read: [The “touch” Command in Linux \[8 Practical Examples\]](#)

18. chmod command

The **chmod command** in **Linux** is used to change the access **permissions** and modes of files and directories. It can control who can **read**, **write**, and **execute** the file or directory. The chmod command can use either **symbolic** or **octal** notation to specify the permissions.

Syntax

chmod [OPTION]... MODE[,MODE]... FILE...

Useful Options

- **-c/--changes:** Displays a message only if a change is made.
- **-f/--silent/--quiet:** Suppresses most error messages.
- **-R/--recursive:** Changes the permissions of the directory and its contents recursively.
- **-v/--verbose:** Outputs a Diagnostic when a file is processed.

Example

To give the execution permission of the **file1.txt** file to **other users**, run the **'chmod o+x file1.txt'** command into the terminal.

```
susmit@ubuntu:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 susmit susmit 7 May 23 14:22 file1.txt
susmit@ubuntu:~/Desktop$ chmod o+x file1.txt
susmit@ubuntu:~/Desktop$ ls -l
total 4
-rw-rw-r-x 1 susmit susmit 7 May 23 14:22 file1.txt
susmit@ubuntu:~/Desktop$
```

Executing permission is given for other users of file1.txt file

To learn more read: [The “chmod” Command in Linux \[6 Practical Examples\]](#)

19. install command

The **install command** in **Linux** is used to copy files and set attributes in **Linux** and other **Unix-like** operating systems. It is similar to the **cp** command, but it can also create directories, change owner and group, and set permissions of the copied files. It is often used to install programs and scripts on the system.

Syntax

install [OPTION]... SOURCE... DIRECTORY

Useful Options

- **-C**: Compares a pair of source and destination files.
- **-d/--directory**: Treats all arguments as directory names; creates all components of the specified directories.
- **-D**: Creates leading directories of the destination except for the target directory.
- **-g**: Sets group ownership.
- **-m/--mode=MODE**: Sets the file mode (permissions) of the copied files or directories. The syntax is the same as the **chmod** command.
- **-o/--owner=OWNER**: Sets the owner of the copied files or directories. This option requires superuser privileges.
- **-t**: Copies source files into a directory.
- **-T**: Treats destination as a file.

Example

To copy the **file1.txt** file which is existing in the current directory to the **~/Downloads** directory, execute the '**install file1.txt ~/Downloads**' command into the terminal.


```
susmit@ubuntu:~/Desktop$ ls ~/Downloads/
susmit@ubuntu:~/Desktop$ install file1.txt ~/Downloads
susmit@ubuntu:~/Desktop$ ls ~/Downloads/
file1.txt
susmit@ubuntu:~/Desktop$
```

file1.txt has been copied.

To learn more read: [The “install” Command in Linux \[6+ Practical Examples\]](#)

20. tar command

The **tar command** in **Linux** is used to create and extract compressed archive files in **Linux** and other **Unix-like** operating systems. It can also list, compare, update, and delete files from an archive. The tar command can use various compression methods, such as **gzip**, **bzip2**, and **xz**.

Syntax

```
tar {A|c|d|r|t|u|x}[GnSkUWOmpsMBiajJzZhPIRvwo] [ARG...]
```

Useful Options

- **-c/--create**: Creates a new archive. Arguments supply the names of the files to be archived. Directories are archived recursively, unless the **--no-recursion** option is given.
- **-delete**: Deletes file/directory from the tar archive.
- **-f**: Sets the name of a tar archive.
- **-j**: Creates tar archive with bzip2.
- **-r**: Updates files/directories of the existing tar archive.
- **-t**: Displays the list of the tar archive.
- **-v**: Displays archived files of a tar archive.
- **--wildcards**: Specifies patterns of archived files of a tar archive.
- **-x/--extract/--get**: Extracts files from an archive. Arguments are optional. When given, they specify the names of the archive members to be extracted.
- **-z**: Creates tar archive with gzip.

Example

To make a **tar** folder named **tarFile.tar** including all of the files from the **folder** directory execute the **'tar -cvf tarFile.tar folder'** in the terminal.

```
susmit@ubuntu:~/Desktop$ ls folder
file1.txt  file2.txt
susmit@ubuntu:~/Desktop$ tar -cvf tarFile.tar folder
folder/
folder/file2.txt
folder/file1.txt
susmit@ubuntu:~/Desktop$ ls
folder  tarFile.tar
susmit@ubuntu:~/Desktop$
```

To learn more read: [The “tar” Command in Linux \[12 Practical Examples\]](#)

Conclusion

In conclusion, mastering the **top 20 Linux commands** is important for efficient system control. These commands empower users to manage files, troubleshoot networks, monitor processes, and automate tasks. Embrace the power of the command line to unlock the full potential of Linux.