

1. **ls** - The most frequently used command in Linux to list directories
2. **pwd** - Print working directory command in Linux
3. **cd** - Linux command to navigate through directories
4. **mkdir** - Command used to create directories in Linux
5. **mv** - Move or rename files in Linux
6. **cp** - Similar usage as mv but for copying files in Linux
7. **rm** - Delete files or directories
8. **touch** - Create blank/empty files
9. **ln** - Create symbolic links (shortcuts) to other files
10. **clear** - Clear the terminal display
11. **cat** - Display file contents on the terminal
12. **echo** - Print any text that follows the command
13. **less** - Linux command to display paged outputs in the terminal
14. **man** - Access manual pages for all Linux commands
15. **uname** - Linux command to get basic information about the OS
16. **whoami** - Get the active username
17. **tar** - Command to extract and compress files in linux
18. **grep** - Search for a string within an output
19. **head** - Return the specified number of lines from the top
20. **tail** - Return the specified number of lines from the bottom
21. **diff** - Find the difference between two files
22. **cmp** - Allows you to check if two files are identical
23. **comm** - Combines the functionality of diff and cmp
24. **sort** - Linux command to sort the content of a file while outputting
25. **export** - Export environment variables in Linux
26. **zip** - Zip files in Linux
27. **unzip** - Unzip files in Linux
28. **ssh** - Secure Shell command in Linux
29. **service** - Linux command to start and stop services
30. **ps** - Display active processes
31. **kill and killall** - Kill active processes by process ID or name
32. **df** - Display disk filesystem information
33. **mount** - Mount file systems in Linux
34. **chmod** - Command to change file permissions
35. **chown** - Command for granting ownership of files or folders
36. **ifconfig** - Display network interfaces and IP addresses
37. **traceroute** - Trace all the network hops to reach the destination
38. **wget** - Direct download files from the internet
39. **ufw** - Firewall command
40. **iptables** - Base firewall for all other firewall utilities to interface with
41. **apt, pacman, yum, rpm** - Package managers depending on the distribution
42. **sudo** - Command to escalate privileges in Linux
43. **cal** - View a command-line calendar
44. **alias** - Create custom shortcuts for your regularly used commands
45. **dd** - Majorly used for creating bootable USB sticks
46. **whereis** - Locate the binary, source, and manual pages for a command
47. **whatis** - Find what a command is used for
48. **top** - View active processes live with their system usage
49. **useradd and usermod** - Add a new user or change existing user data
50. **passwd** - Create or update passwords for existing users

Now let's dive a little deeper into each of these commands and understand them in more detail. We already have a lot of existing articles for each of those individual commands. For your convenience, we'll add links to all the existing articles, and continue to update the article as new topics are covered.

The `ls` command in Linux

The `ls` command is used to list files and directories in the current working directory. This is going to be one of the most frequently used Linux commands you must know of.

```
root@ubuntu:/# ls
bin  dev  go1.13.5.linux-amd64.tar.gz  initrd.img  lib  lost+found  mnt  proc  run  snap  sys  usr  vmlinuz
boot  etc  home  initrd.img.old  lib64  media  opt  root  sbin  srv  tmp  var  vmlinuz.old
root@ubuntu:/#
```

As you can see in the above image, using the command by itself without any arguments will give us an output with all the files and directories in the directory. The command offers a lot of flexibility in terms of displaying the data in the output.

Learn more about using the [ls command](#)

[Jump back to commands list ↑](#)

The `pwd` command in Linux

The `pwd` command allows you to print the current working directory on your terminal. It's a very basic command and solves its purpose very well.

```
root@ubuntu:/etc/network/if-pre-up.d# pwd
/etc/network/if-pre-up.d
```

Now, your terminal prompt should usually have the complete directory anyway. But in case it doesn't, this can be a quick command to see the directory that you're in. Another application of this command is when creating scripts where this command can allow us to find the directory where the script has been saved.

[Jump back to commands list ↑](#)

The `cd` command in Linux

While working within the terminal, moving around within directories is pretty much a necessity. The `cd` command is one of the important Linux commands you must know, and it will help you navigate through directories. Just type `cd` followed by directory, as shown below.

Copy

```
root@ubuntu:~# cd <directory path>
```

```
root@ubuntu:~# pwd
/root
root@ubuntu:~# cd /etc/
root@ubuntu:/etc# pwd
/etc
root@ubuntu:/etc#
```

As you can see in the above command, I simply typed `cd /etc/` to get into the `/etc` directory. We used the `pwd` command to print the current working directory.

[Jump back to commands list ↑](#)

The `mkdir` command in Linux

The `mkdir` command allows you to create directories from within the terminal.

Copy

```
root@ubuntu:~# mkdir <folder name>
```

```
root@ubuntu:~# ls
root@ubuntu:~# mkdir JournalDev
root@ubuntu:~# ls
JournalDev
root@ubuntu:~#
```

As you can see in the above screenshot, we created the `JournalDev` directory with just this simple command.

[Jump back to commands list ↑](#)

The `cp` and `mv` commands

The `cp` and `mv` commands are equivalent to the copy-paste and cut-paste commands in Windows. But since Linux doesn't really have a command for renaming files, we also use the `mv` command to rename files and folders.

Copy

```
root@ubuntu:~# cp <source> <destination>
```

```
root@ubuntu:~# ls
Sample
root@ubuntu:~# cp Sample Sample-Copy
root@ubuntu:~# ls
Sample Sample-Copy
root@ubuntu:~# █
```

In the above command, we created a copy of the file named Sample. Let's see how what happens if we use the `mv` command in the same manner.

[Copy](#)

```
root@ubuntu:~# mv <source> <destination>
```

```
root@ubuntu:~# ls
Sample
root@ubuntu:~# mv Sample Sample-Copy
root@ubuntu:~# ls
Sample-Copy
root@ubuntu:~# █
```

In the above case, since we were moving the file within the same directory, it acted as a rename. The file name is now changed.

[Jump back to commands list ↑](#)

The `rm` command in Linux

In the previous section, we deleted the Sample-Copy file. The `rm` command is used to delete files and folders and is one of the important Linux commands you must know.

[Copy](#)

```
root@ubuntu:~# rm <file name>
```

```
root@ubuntu:~# ls
Sample-Copy
root@ubuntu:~# rm Sample-Copy
root@ubuntu:~# ls
root@ubuntu:~#
```

To delete a directory, you must add the `-r` argument to it. Without the `-r` argument, the `rm` command won't delete directories.

[Copy](#)

```
root@ubuntu:~# rm -r <folder/directory name>
```

The `-r` flag in the `rm` command in Linux stands for “**recursive**”. When used with the `rm` command, it will remove not only the specified file but also all of its subdirectories and the files within those subdirectories recursively.

Note: It's important to be careful when using the `rm` command with the `-r` flag, as it can quickly and permanently delete a large number of files and directories. It's a good idea to use the `-i` flag in conjunction with the `-r` flag, which will prompt you for confirmation before deleting each file and directory.

For example, to remove the `mydir` directory and its contents with confirmation, you can use this command:

[Copy](#)

```
root@ubuntu:~# rm -ri mydir
```

This will prompt you for confirmation before deleting each file and directory within the `mydir` directory.

[Jump back to commands list ↑](#)

The touch command in Linux

The `touch` command in Linux creates an empty file or updates the timestamp of an existing file.

[Copy](#)

```
root@ubuntu:~# touch <file name>
```

```
root@ubuntu:~# ls
root@ubuntu:~# touch New-File
root@ubuntu:~# ls
New-File
root@ubuntu:~#
```

[Jump back to commands list ↑](#)

The `ln` command in Linux

To create a link to another file, we use the `ln` command. This is one of the most important Linux commands that you should know if you're planning to work as a Linux administrator.

Copy

```
root@ubuntu:~# ln -s <source path> <link name>
```

```
root@ubuntu:~# ls
New-File
root@ubuntu:~# ln -s New-File New-File-Link
root@ubuntu:~# ls -l
total 0
-rw-r--r-- 1 root root 0 Jan 26 15:47 New-File
lrwxrwxrwx 1 root root 8 Jan 26 16:03 New-File-Link -> New-File
root@ubuntu:~#
```

The `-s` flag creates a **symbolic link** (also known as a symlink or soft link) to a file or directory. A symbolic link is a special type of file that acts as a shortcut or pointer to another file or directory.

By default, the `ln` command will make *hard links* instead of symbolic or soft links.

Note: Say you have a text file. If you make a **symbolic link** to that file, the link is only a pointer to the original file. If you delete the original file, the link will be broken, as it no longer has anything to point to.

A **hard link** is a mirror copy of an original file with the exact same contents. Like symbolic links, if you edit the contents of the original file, those changes will be reflected in the hard link. If you delete the original file, though, the hard link will still work, and you can view and edit it as you would a normal copy of the original file.

Learn more about [Soft and Hard Links](#).

[Jump back to commands list](#) ↑

The `clear` command in Linux

The `clear` command in Linux clears the terminal screen. It removes all the text and output currently displayed on the terminal and gives you a clean slate to work with.

Here is an example of how to use the `clear` command:

Copy

```
root@ubuntu:~# clear
```

This will clear the terminal screen and move the cursor to the top-left corner of the screen.

You can also use the `clear` command in combination with other commands, like this:

Copy

```
root@ubuntu:~# ls -l; clear
```

This will list the files and directories in the current directory, and then clear the terminal screen.

Note: The `clear` command does not delete any files or data from your system. It only affects the display of the terminal.

The `cat`, `echo`, and `less` commands

When you want to output the contents of a file or print anything to the terminal output, we use the `cat` or `echo` commands. Let's see their basic usage.

Copy

```
root@ubuntu:~# cat <file name>
root@ubuntu:~# echo <Text to print on terminal>
```

```
root@ubuntu:~# cat New-File
Hello, welcome to JournalDev. The one spot to learn everything related to programming.
root@ubuntu:~# echo New-File
New-File
root@ubuntu:~# █
```

As you can see in the above example, the `cat` command, when used on our `New-File`, prints the contents of the file. At the same time, when we use `echo` command, it simply prints whatever follows after the command.

The `less` command is used when the output printed by any command is larger than the screen space and needs scrolling. The `less` command allows the user to break down the output and scroll through it with the use of the enter or space keys.

The simple way to do this is with the use of the pipe operator (`|`).

Copy

```
root@ubuntu:~# cat /boot/grub/grub.cfg | less
```

Note: Use the `-S` flag with `less` to enable line wrapping. This will allow you to view long lines of text without scrolling horizontally.

Use the `-N` flag with `less` to display line numbers. This can be useful when you need to know the line number of a specific piece of text.

You can use these useful flags in the following way:

Copy

```
root@ubuntu:~# cat /boot/grub/grub.cfg | less -SN
```

Using `less` with the pipe operator can be useful in many different situations. Here are a few examples:

- Viewing the output of a long-running command, such as `top` or `htop`.
- Searching for specific text in the output of a command, such as `grep` or `cat`.

[Jump back to commands list ↑](#)

The `man` command in Linux

The `man` command is a very useful Linux command one must know. When working with Linux, the packages that we download can have a lot of functionality. Knowing it all is impossible.

The `man` command in Linux is used to display the manual page for a specific command. It provides detailed information about the command, including its syntax, options, and examples.

Here's an example of how to use the `man` command:

1. Open a terminal and type `man ls` to display the manual page for the `ls` command.

Copy

```
root@ubuntu:~# man ls
```

This will display a page that looks something like this:

Copy

Output

```
LS(1)                                User Commands                                LS(1)

NAME
  ls - list directory contents

SYNOPSIS
  ls [OPTION]... [FILE]...

DESCRIPTION
  List information about the FILES (the current directory by default).

  Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

  Mandatory arguments to long options are mandatory for short options too.

  -a, --all
      do not ignore entries starting with .

  -A, --almost-all
      do not list implied . and ..

  -c      with -lt: sort by, and show, ctime (time of last modification
          of file status information) with -l: show ctime and sort
          by name;

  ...
```

[Jump back to commands list ↑](#)

The `uname` and `whoami` commands

The `uname` and `whoami` commands allow you to access some basic information that comes in handy when you work on multiple systems.

The `uname` command in Linux displays information about the system's kernel, including the kernel name, hostname, kernel release, kernel version, and machine hardware name.

The `whoami` command in Linux returns the current user's username. It stands for "who am I?" and it's often used to determine the current user's identity in shell scripts or the terminal.

Let's see the output of both the commands and the way we can use these.

[Copy](#)

```
root@ubuntu:~# uname -a
```

```
root@ubuntu:~# uname -a
Linux ubuntu 4.15.0-74-generic #84-Ubuntu SMP Thu Dec 19 08:06:28 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
root@ubuntu:~# whoami
root
root@ubuntu:~#
```

The parameter `-a` with `uname` command stands for "all". This prints out the complete information. If the parameter is not added, all you will get as the output is "Linux".

Note: Some important flags you can use with the `uname` command.

1. Use `uname -s` to display the kernel name.
2. Use `uname -n` to display the hostname.
3. Use `uname -r` to display the kernel release.
4. Use `uname -v` to display the kernel version.
5. Use `uname -m` to display the machine hardware name.

[Jump back to commands list ↑](#)

The tar, zip, and unzip commands

The `tar` command in Linux is used to create and extract archived files. We can extract multiple different archive files using the `tar` command.

To create an archive, we use the `-c` parameter, and to extract an archive, we use the `-x` parameter. Let's see how it works.

[Copy](#)

```
#Compress
root@ubuntu:~# tar -cvf <archive name> <files separated by space>
#Extract
root@ubuntu:~# tar -xvf <archive name>
```

```
root@ubuntu:~# tar -cvf Compress.tar New-File New-File-Link
New-File
New-File-Link
root@ubuntu:~# tar -xvf Compress.tar
New-File
New-File-Link
root@ubuntu:~# ls
```

In the first line, we created an archive named **Compress.tar** with the New-File and New-File-Link. In the next command, we have extracted those files from the archive.

Let's discuss the `zip` and `unzip` commands. Both are very straightforward. You can use them without any parameters, and they'll work as intended. Let's see an example below.

Copy

```
root@ubuntu:~# zip <archive name> <file names separated by space>
root@ubuntu:~# unzip <archive name>
```

```
root@ubuntu:~# zip Sample.zip New-File New-File-Edited
updating: New-File (deflated 16%)
updating: New-File-Edited (deflated 19%)
root@ubuntu:~# unzip Sample.zip
Archive: Sample.zip
replace New-File? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: New-File
  inflating: New-File-Edited
root@ubuntu:~#
```

Since we already have those files in the same directory, the `unzip` command prompts us before overwriting those files.

[Jump back to commands list ↑](#)

The `grep` command in Linux

The `grep` command is a powerful and versatile text search tool in Linux and Unix-based operating systems. It can search for specific patterns or strings in one or more files and filter the output of

other commands.

The `grep` command stands for “global regular expression print,” which reflects its ability to search for regular expressions across multiple lines and files.

Copy

```
root@ubuntu:~# <Any command with output> | grep "<string to find>"
```

```
root@ubuntu:~# cat New-File
Hello, welcome to JournalDev.
The one spot to learn everything related to programming.
Adding a few more lines
root@ubuntu:~# cat New-File | grep "learn"
The one spot to learn everything related to programming.
root@ubuntu:~#
```

This was a simple demonstration of the command. Learn more about the [grep command](#).

[Jump back to commands list](#) ↑

The head and tail commands

When outputting large files, the `head` and `tail` commands come in handy. These commands display the beginning or end of a file, respectively. They are commonly used to quickly view the contents of a file without having to open it in a text editor.

The `head` and `tail` commands display the first 10 lines of a file by default. To display a different number of lines, you can use the `-n` option, followed by the number of lines you want to display.

Here’s an example of using the `head` and `tail` commands:

Copy

```
root@ubuntu:~# head <file name>
root@ubuntu:~# tail <file name>
```

```
root@ubuntu:~# head Words
Carrot

Cave

Chair

Chess Board

Chief

root@ubuntu:~#
```

As you can see, the head command showed 10 lines from the top of the file.

```
root@ubuntu:~# tail Words

Horse

Hose

Ice

Ice-cream

Insect
root@ubuntu:~#
```

The tail command outputted the bottom 10 lines from the file.

These commands can be used to quickly view a file's contents, monitor real-time updates for troubleshooting issues, filter output from other commands, and perform log analysis.

[Jump back to commands list ↑](#)

The diff, comm, and cmp commands

The diff, comm, and cmp commands are all used to compare files in Linux and Unix-based operating systems. These commands can be used to identify differences between two files, merge changes, and perform other file comparison tasks.

Copy

```
root@ubuntu:~# diff <file 1> <file 2>
```

```
root@ubuntu:~# diff New-File New-File-Edited
3c3
< Adding a few more lines
---
> Adding a few more lines - This line is edited
root@ubuntu:~#
```

As you can see above, we have added a small piece of text saying, “This line is edited” to the New-File-Edited file.

The `cmp` command is used to compare two files and display the first byte that is different between them. It can be used to identify differences between binary files or to check for corruption in files.

Copy

```
root@ubuntu:~# cmp <file 1> <file 2>
```

```
root@ubuntu:~# cmp New-File New-File-Edited
New-File New-File-Edited differ: byte 112, line 3
root@ubuntu:~#
```

The `cmp` command only tells us the line number, which is different. Not the actual text.

The `comm` command is used to compare two sorted files and display the lines that are unique to each file, as well as the lines that are common to both files.

Copy

```
root@ubuntu:~# comm <file 1> <file2>
```

```
root@ubuntu:~# comm New-File New-File-Edited
      Hello, welcome to JournalDev.
      The one spot to learn everything related to programming.
Adding a few more lines
comm: file 1 is not in sorted order
      Adding a few more lines - This line is edited
comm: file 2 is not in sorted order
root@ubuntu:~#
```

The text that's aligned to the left is only present in `file 1`. The centre-aligned text is present only in `file 2`. And the right-aligned text is present in both files.

By the looks of it, `comm` command makes the most sense when we're trying to compare larger files and would like to see everything arranged together.

All three of these commands are essential tools for working with files in Linux and Unix-based operating systems. By understanding how to use the `diff`, `comm`, and `cmp` commands effectively, you can identify differences between files, merge changes, and perform other file comparison tasks.

These commands can help you to identify and resolve issues with files, as well as to track changes and maintain version control. Whether you're a developer or a system administrator, these commands are an essential part of your toolkit.

[Jump back to commands list ↑](#)

The `sort` command in Linux

The `sort` command is used to sort lines in a text file or standard input in Linux and Unix-based operating systems. It can be used to sort lines in ascending or descending order and to perform other sorting operations, such as sorting by fields or using a custom sorting order.

The basic syntax of the `sort` command is:

Copy

```
root@ubuntu:~# sort <filename>
```

```
root@ubuntu:~# cat New-File
Hello, welcome to JournalDev.
The one spot to learn everything related to programming.
Adding a few more lines
root@ubuntu:~# sort New-File
Adding a few more lines
Hello, welcome to JournalDev.
The one spot to learn everything related to programming.
root@ubuntu:~# █
```

By default, the `sort` command sorts lines in **ASCII collating sequence**, which can lead to unexpected results when sorting numbers or special characters. To sort numbers in numerical order, you can use the `-n` option.

Here's an example of using the `-n` option:

Copy

```
root@ubuntu:~# sort -n file.txt
```

The above command will sort the lines in `file.txt` in numerical order.

The `sort` command can also be used to sort lines based on specific fields using the `-k` option.

Here's an example of using the `-k` option:

Copy

```
root@ubuntu:~# sort -k 2 file.txt
```

This command will sort the lines in `file.txt` based on the second field.

The `sort` command is a powerful and flexible tool for working with text files in Linux and Unix-based operating systems. By understanding how to use the `sort` command effectively, you can sort lines in text files, sort lines based on specific fields, and perform other sorting operations.

These commands can help you organize and analyze data and perform other file manipulation tasks. Whether you're a developer or a system administrator, the `sort` command is an essential part of your toolkit.

[Jump back to commands list ↑](#)

The `export` command in Linux

The `export` command in Linux and Unix-based operating systems is used to set environment variables. Environment variables are used to store information that can be used by processes or commands.

Once an environment variable is set, it can be accessed by any process or command that is running in the same shell.

Environment variables can be used to store a wide range of information, such as configuration settings, user preferences, or system information.

Here's an example of using the `export` command:

Copy


```
root@ubuntu:~# export <variable name>=<value>
```

```
root@ubuntu:~# export PS1="\u@\h:\w -->> "  
root@ubuntu:~ -->> █
```

Learn more about the `export` [command](#)

[Jump back to commands list](#) ↑

The `ssh` command in Linux

The `ssh` command in Linux and Unix-based operating systems establishes a secure shell connection to a remote server. The command provides a secure encrypted connection between the local and remote servers, allowing users to run commands and transfer files securely.

The basic syntax of the `ssh` command is:

Copy

```
root@ubuntu:~ ssh username@remote-server
```

This command establishes an `ssh` connection to the `remote-server` using the `username` account.

The `ssh` command supports a wide range of options and configurations, including:

- Configuring authentication methods (password, public key, etc.)
- Configuring encryption algorithms
- Configuring compression
- Configuring port forwarding
- Configuring X11 forwarding
- Configuring SSH keys

The `service` command in Linux

The `service` command in Linux is used to manage system services, which are long-running processes that are started at boot time and run in the background. These services are responsible for providing various system functionalities, such as networking, database management, and user authentication.

The `service` command is used to start, stop, restart, and check the status of these services. It is a front-end to the `systemctl` command, which is used to manage the `systemd` service manager.

The basic syntax of the command is as below.

Copy

```
root@ubuntu:~ service ssh status
root@ubuntu:~ service ssh stop
root@ubuntu:~ service ssh start
```

```
root@ubuntu:~ -->> service ssh status
• ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2020-01-20 02:58:32 UTC; 6 days ago
  Process: 744 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Main PID: 770 (sshd)
  Tasks: 5 (limit: 503)
  CGroup: /system.slice/ssh.service
          └─ 770 /usr/sbin/sshd -D
             └─14158 sshd: [accepted]
                └─14159 sshd: [net]
                   └─14176 sshd: unknown [priv]
                      └─14177 sshd: unknown [net]
```

As you can see in the image, the `ssh` server is running on our system.

[Jump back to commands list ↑](#)

The `ps`, `kill`, and `killall` commands

The `ps`, `kill`, and `killall` commands are all used to manage processes in Linux.

The `ps` command is used to display information about the current running processes on the system. Here are some examples of using the `ps` command:

Display a list of all running processes:

Copy

```
root@ubuntu:~ ps -ef
```

Display a list of all processes for a specific process ID (PID):

Copy

```
root@ubuntu:~ ps -p PID
```

Let's see all of this in action:

Copy

```
root@ubuntu:~ ps
root@ubuntu:~ kill <process ID>
root@ubuntu:~ killall <process name>
```

For demonstration purposes, we will create a shell script with an infinite loop and will run it in the background.

With the use of the & symbol, we can pass a process into the background. As you can see, a new bash process with PID 14490 is created.

```
root@ubuntu:~ -->> ps
  PID TTY          TIME CMD
  9740 pts/0        00:00:01 bash
 14487 pts/0        00:00:00 ps
root@ubuntu:~ -->> bash loop.sh &
[1] 14490
root@ubuntu:~ -->> ps
  PID TTY          TIME CMD
  9740 pts/0        00:00:01 bash
 14490 pts/0        00:00:00 bash
 14491 pts/0        00:00:00 sleep
 14492 pts/0        00:00:00 ps
root@ubuntu:~ -->> █
```

Now, to kill a process with the `kill` command, you can type `kill` followed by the PID(Process Id) of the process.

```
root@ubuntu:~ -->> ps
  PID TTY          TIME CMD
  9740 pts/0        00:00:01 bash
 14490 pts/0        00:00:00 bash
 14491 pts/0        00:00:00 sleep
 14499 pts/0        00:00:00 ps
root@ubuntu:~ -->> kill 14491
root@ubuntu:~ -->> loop.sh: line 4: 14491 Terminated          sleep infinity
```

But if you do not know the process ID and just want to kill the process with the name, you can make use of the `killall` command.

```
root@ubuntu:~ -->> ps
  PID TTY          TIME CMD
  9740 pts/0    00:00:01 bash
 14490 pts/0    00:00:00 bash
 14502 pts/0    00:00:00 sleep
 14513 pts/0    00:00:00 ps
root@ubuntu:~ -->> killall sleep
loop.sh: line 4: 14502 Terminated          sleep infinity
root@ubuntu:~ -->> █
```

You will notice that PID 14490 stayed active. That is because, both times, we killed the sleep process.

Learn more about the [ps command](#) and the [kill command](#).

[Jump back to commands list ↑](#)

The df and mount commands

When working with Linux, the `df` and `mount` commands are very efficient utilities to mount filesystems and get details of the file system.

The `df` command is used to display the amount of disk space used and available on the file systems, and the `mount` command is used to mount a file system or device to a specific directory.

When we say `mount`, it means that we'll connect the device to a folder so we can access the files from our filesystem. The default syntax to mount a filesystem is below:

Copy

```
root@ubuntu:~ mount /dev/cdrom /mnt
root@ubuntu:~ df -h
```

In the above case, `/dev/cdrom` is the device that needs to be mounted. Usually, a mountable device is found inside the `/dev` folder. `mnt` is the destination folder to which to mount the device. You can change it to any folder you want, but we have used `/mnt` as it's the system's default folder for mounting devices.

To see the mounted devices and get more information about them, we use the `df` command. Just typing `df` will give us the data in bytes, which is not readable. So, we'll use the `-h` parameter to make the data human-readable.

```
root@ubuntu:~ -->> df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            210M   0  210M   0% /dev
tmpfs           49M   892K   48M   2% /run
/dev/vda1       9.8G  7.0G  2.4G  75% /
tmpfs           241M   8.0K  241M   1% /dev/shm
tmpfs           5.0M   0   5.0M   0% /run/lock
tmpfs           241M   0   241M   0% /sys/fs/cgroup
tmpfs           49M   16K   49M   1% /run/user/120
tmpfs           49M   0   49M   0% /run/user/0
root@ubuntu:~ -->> █
```

[Jump back to commands list ↑](#)

The chmod and chown commands

The `chmod` and `chown` commands are used to modify file permissions and ownership in Linux.

The `chmod` command is used to change the permissions of a file or directory, and the `chown` command is used to change the ownership of a file or directory

The default syntax for both the commands is `chmod <parameter> filename` and `chown <user:group> filename`

Copy

```
root@ubuntu:~ chmod +x loop.sh
root@ubuntu:~ chown root:root loop.sh
```

```
root@ubuntu:~ -->> ls -l loop.sh
-rw-r--r-- 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> chmod +x loop.sh
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> █
```

In the above example, we're adding executable permissions to the `loop.sh` file with the `chmod` command. In addition, with the `chown` command, we've made it accessible only to the root user and users within the root group.

```
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 root root 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> chown www-data:www-data loop.sh
root@ubuntu:~ -->> ls -l loop.sh
-rwxr-xr-x 1 www-data www-data 32 Jan 26 18:24 loop.sh
root@ubuntu:~ -->> █
```

As you will notice, the `root root` part is now changed to `www-data` which is the new user who has full file ownership.

Learn more about the [Linux file Permissions](#) and using the `chmod` [command](#).

[Jump back to commands list](#) ↑

The `ifconfig` and `traceroute` commands

The `ifconfig` and `traceroute` commands manage network interfaces and trace the route of network packets in Linux.

The `ifconfig` command will give you the list of all the network interfaces along with the IP addresses, MAC addresses and other information about the interface.

Copy

```
root@ubuntu:~ ifconfig
```

There are multiple parameters that can be used, but we'll work with the basic command here.

```
root@ubuntu:~ -->> ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:3b:09:02:00 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

The `traceroute` command is used to trace the route of network packets and determine the path they take to reach a specific destination.

When working with `traceroute`, you can simply specify the IP address, hostname, or domain name of the endpoint.

Copy

```
root@ubuntu:~# traceroute <destination address>
```

```
root@ubuntu:~# traceroute localhost
traceroute to localhost (127.0.0.1), 30 hops max, 60 byte packets
 1 localhost (127.0.0.1) 0.029 ms 0.007 ms 0.006 ms
root@ubuntu:~#
```

Now, obviously, `localhost` is just one hop (the network interface itself). You can try this same command with any other domain name or IP address to see all the routers your data packets pass through to reach the destination.

Learn more about the handy [Linux Networking tools](#).

[Jump back to commands list ↑](#)

The `wget` command in Linux

If you want to download a file from within the terminal, the `wget` command is one of the handiest command-line utilities available. It is one of the important Linux commands you should know when working with source files.

When you specify the link for download, it has to directly be a link to the file. If the file cannot be accessed by the `wget` command, it will simply download the webpage in HTML format instead of the actual file that you wanted.

Let's try an example. The basic syntax of the `wget` command is :

Copy

```
root@ubuntu:~# wget <link to file>
```

Or,

Copy

```
root@ubuntu:~# wget -c <link to file>
```

The `-c` argument allows us to resume an interrupted download.

[Jump back to commands list ↑](#)

The ufw and iptables commands

The `ufw` and `iptables` commands are used to manage firewalls in Linux.

UFW and IPTables are firewall interfaces for the Linux Kernel's netfilter firewall. IPTables directly passes firewall rules to Netfilter while UFW configures the rules in IPTables, which then sends those rules to Netfilter.

Why do we need UFW when we have IPTables? Because IPTables is pretty difficult for a newbie. UFW makes things extremely easy. See the below example where we are trying to allow port 80 for our webserver.

Copy

```
root@ubuntu:~# iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
root@ubuntu:~# ufw allow 80
```

I'm sure you now know why UFW was created! Look at how easy the syntax becomes. Both these firewalls are very comprehensive and can allow you to create any kind of configuration required for your network. Learn at least the basics of UFW or IPTables firewall, as these are the Linux commands you must know.

Learn more about [managing firewall with ufw](#) and [managing firewall with iptables](#).

[Jump back to commands list ↑](#)

Package Managers in Linux

Different Linux distributions use different package managers. Since we're working on a Ubuntu server, we have the `apt` package manager. But for someone working on a Fedora, Red Hat, Arch, or Centos machine, the package manager will be different.

Below are the commands on how to use these package managers on different Linux distributions.

- **Debian and Debian-based distros** - `apt install <package name>`
- **Arch and Arch-based distros** - `pacman -S <package name>`
- **Red Hat and Red Hat-based distros** - `yum install <package name>`
- **Fedora and CentOS** - `yum install <package>`

Getting yourself well versed with the package manager of your distribution will make things much easier for you in the long run. So even if you have a GUI based package management tool installed, try and make use of the CLI based tool before you move on to the GUI utility. Add these to your list of Linux commands you must know.

[Jump back to commands list ↑](#)

The `sudo` command in Linux

“With great power, comes great responsibility”

This is the quote displayed when a `sudo`-enabled user (sudoer) first uses the `sudo` command to escalate privileges. This command is equivalent to logging in as `root` (based on what permissions you have as a sudoer).

Copy

```
non-root-user@ubuntu:~# sudo <command you want to run>
Password:
```

Just add the command `sudo` before any command that you need to run with escalated privileges, and that's it. It's very simple to use, but can also be an added security risk if a malicious user gains access to a sudoer.

Learn more about how to [create a new sudo user](#) and [how to edit sudoers file](#).

[Jump back to commands list](#) ↑

The `cal` command in Linux

Have you ever wanted to view the calendar in the terminal? There apparently are people who want it to happen, and well, here it is.

The `cal` command displays a well-presented calendar on the terminal. Just enter the command `cal` on your terminal prompt.

Copy

```
root@ubuntu:~# cal
root@ubuntu:~# cal March 2024
```

```
root@ubuntu:~# cal
      January 2020
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

root@ubuntu:~# cal May 2019
      May 2019
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

root@ubuntu:~# █
```

Although we don't need it often, it's a great addition! It's an amazing option for terminal fans.

[Jump back to commands list ↑](#)

The `alias` command

Do you have some commands that you run very frequently while using the terminal? It could be `rm -r` or `ls -l`, or it could be something longer like `tar -xvzf`.

This is one of the productivity-boosting Linux commands you must know.

If you know a command that you run very often, it's time to create an `alias`.

What's an alias? In simple terms, it's another name for a command that you've defined.

Copy

```
root@ubuntu:~# alias ls1="ls -l"
OR
root@ubuntu:~# alias rmd="rm -r"
```

Now, every time you enter `ls1` or `rmd` in the terminal, you'll receive the output that you'd have received if you had used the full commands.

The examples here are for really small commands that you can still type by hand every time. But in some situations where a command has too many arguments that you need to type, it's best to create a shorthand version of the same.

[Jump back to commands list ↑](#)

The dd command in Linux

This command was created to convert and copy files from multiple file system formats. In the current day, the command is simply used to create bootable USB for Linux but there still are some things important you can do with the command.

The `dd` command in Linux is a versatile command used for low-level copying and conversion of data. It stands for “data-description” or “data definition,” and it can be used to copy and convert data between different file formats and storage devices.

For example, if we wanted to back up the entire hard drive as is to another drive, we would use the `dd` command.

Copy

```
root@ubuntu:~# dd if=/dev/sdb of=/dev/sda
```

The **if** and **of** arguments stand for **input file** and **output file**.

It's a powerful and flexible tool, but it can also be dangerous if not used carefully. Always double-check your syntax and make sure you know what the command will do before executing it.

[Jump back to commands list ↑](#)

The whereis and whatis commands

The `whereis` and `whatis` commands are used in Linux to search for information about programs and files.

The `whereis` command locates the binary, source, and manual pages for a specific command or program, and the `whatis` command displays a short description of a command or program.

Copy

```
root@ubuntu:~# whereis sudo
sudo: /usr/bin/sudo /usr/lib/sudo /usr/share/man/man8/sudo.8.gz
```

The `whatis` command gives us an explanation of what a command actually is.

[Copy](#)

```
root@ubuntu:~# whatis sudo
sudo (8) - execute a command as another user
```

[Jump back to commands list ↑](#)

The `top` command in Linux

A few sections earlier, we talked about the `ps` command. You observed that the `ps` command will output the active processes and end itself.

The `top` command is like a CLI version of the task manager in Windows.

The `top` command in Linux is a system monitoring tool that displays real-time information about system processes and resource usage. It provides a dynamic, real-time view of system activity, including CPU usage, memory usage, and process information.

```
top - 20:41:20 up 6 days, 17:42, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 124 total, 1 running, 86 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7 us, 1.0 sy, 0.0 ni, 96.9 id, 1.3 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 492644 total, 11616 free, 387228 used, 93800 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 77676 avail Mem

  PID USER      PR  NI   VIRT    RES    SHR  S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0 225344   4428   2056  S   0.0   0.9   1:27.94 systemd
    2 root        20   0     0     0     0  S   0.0   0.0   0:00.11 kthreadd
    4 root         0 -20     0     0     0  I   0.0   0.0   0:00.00 kworker/0:0H
    6 root         0 -20     0     0     0  I   0.0   0.0   0:00.00 mm_percpu_wq
    7 root        20   0     0     0     0  S   0.0   0.0   0:12.93 ksoftirqd/0
    8 root        20   0     0     0     0  I   0.0   0.0   0:50.10 rcu_sched
    9 root        20   0     0     0     0  I   0.0   0.0   0:00.00 rcu_bh
   10 root        rt    0     0     0     0  S   0.0   0.0   0:00.00 migration/0
   11 root        rt    0     0     0     0  S   0.0   0.0   0:01.81 watchdog/0
```

Note: Here are some examples of using the `top` command:

1. Sort processes by memory usage:

[Copy](#)

```
root@ubuntu:~# top -o MEM
```

This will sort the process list by memory usage, with the most memory-intensive processes at the top.

2. Display detailed information about a specific process:

Copy

```
root@ubuntu:~# top -p PID
```

Replace `PID` with the ID of the process you want to inspect.

3. Display a summary of system resource usage:

Copy

```
root@ubuntu:~# top -n 1
```

This will display a single summary screen of system resource usage.

It's a powerful and flexible tool for monitoring system activity and troubleshooting performance issues.

[Jump back to commands list ↑](#)

The `useradd` and `usermod` commands

The `useradd` and `usermod` commands are used in Linux to manage user accounts.

The `useradd` or `adduser` commands are the exact same commands where `adduser` is just a symbolic link to the `useradd` command. This command allows us to create a new user in Linux.

Copy

```
root@ubuntu:~# useradd JournalDev -d /home/JD
```

The above command will create a new user named `JournalDev` with the home directory as `/home/JD`.

The `usermod` command, on the other hand, is used to modify existing users. You can modify any value of the user including the groups, the permissions, etc.

For example, if you want to add more groups to the user, you can type in:

[Copy](#)

```
root@ubuntu:~# usermod JournalDev -a -G sudo, audio, mysql
```

[Jump back to commands list ↑](#)

The `passwd` command in Linux

Now that you know how to create new users, let's also set the password for them.

The `passwd` command lets you set the password for your own account, or if you have the permissions, set the password for other accounts.

Here are some examples of using the `passwd` command:

1. Change the password for the current user:

[Copy](#)

```
root@ubuntu:~# passwd
```

This will prompt you to enter a new password for the current user.

2. Change the password for a specific user:

[Copy](#)

```
root@ubuntu:~# passwd username
```

Replace `username` with the name of the user whose password you want to change.

3. Force a user to change their password at the next login:

[Copy](#)

```
root@ubuntu:~# passwd -f username
```

4. Set an expiration date for a user's password:

```
root@ubuntu:~# passwd -e -n days -w warndays username
```

Replace `days` with the number of days before the password expires and `warn days` with the number of days before the password expires that the user will be warned.

These are just a few examples of using the `passwd` command in Linux. By understanding how to use this command effectively, you can manage user accounts and ensure that your system is secure.

```
root@ubuntu:~# passwd
New password:
Retype new password:
passwd: password updated successfully
root@ubuntu:~# █
```