

# LINUX COMMAND TIPS



---

100+ Practical Solutions for Daily  
Operations

---

# Linux Command Tips

## 100+ Practical Solutions for Daily Operations

Linux Handbook

## Contents

<b>Acknowledgment</b>	<b>1</b>
<b>Introduction</b>	<b>1</b>
<b>How to use this book</b>	<b>2</b>
Reading the Commands . . . . .	2
Understanding Command Output and Screenshots . . . . .	2
<b>File Management Commands</b>	<b>2</b>
<b>Disk Management Commands</b>	<b>11</b>
<b>Process Management Commands</b>	<b>12</b>
<b>Networking Commands</b>	<b>14</b>
<b>User and Group Management</b>	<b>15</b>
<b>Editor Specific Commands</b>	<b>16</b>
<b>Miscellaneous</b>	<b>17</b>
<b>Keyboard shortcuts</b>	<b>18</b>

## Acknowledgment

Our deepest thanks go to the official **man pages** of various commands, which remain the single most reliable and authoritative source of information. We also want to acknowledge our dedicated community and our own work.

The tips and insights initially shared on the **Linux Handbook Twitter account**, detailed articles on the **Linux Handbook blog**, and the man pages for commands formed the bedrock of this collection.

from Linux Handbook

## Introduction

You're staring at a terminal. You need to get something done. Fast. You don't want a 20-page explanation about the history of Unix or a deep dive into system architecture—you want **the exact command that solves your problem right now**.

That's exactly what this book delivers.

**101 razor-sharp tips.** Each one gets you from problem to solution in seconds, not hours. No fluff. No lengthy backstories. Just the essential command, the key flags you actually need, and maybe a lightning-quick “why this works” when it matters.

Think of it as your **Linux command cheat sheet on steroids**. The kind of resource you wish you had during those late-night troubleshooting sessions when Google gives you 47 different answers and you just need the one that works.

## How to use this book

This book is designed to be a quick and practical reference for essential Linux commands. It is organized into the following sections to help you easily find what you need:

- **File Management Commands**
- **Disk Management Commands**
- **Process Management Commands**
- **Networking Commands**
- **User & Group Management Commands**
- **Text-Editor Specific Commands**
- **Miscellaneous**
- **Keyboard Shortcuts**

## Reading the Commands

Throughout the book, you will see commands presented in code blocks. The command that you need to run in your terminal will always be preceded by a dollar sign (\$).

For example: `$ ls -l`

## Understanding Command Output and Screenshots

To help you understand the results of a command, we have included examples of the output for many entries. This output is displayed in a code block without a preceding dollar sign.

For example:

```
-rw-r--r-- 1 user group 1234 Jan 1 12:00 file.txt
```

In cases where the output is more visual or involves a graphical interface, screenshots are provided.

Please note that not every command includes an output example or a screenshot. This is especially true for dynamic commands like `top` or commands that involve live editing within a file, as their output changes in real-time.

**User Input** Some commands require you to provide specific information, such as a filename or a username. When this is the case, the required input will be indicated with angle brackets (<>). You should replace the text inside the brackets with your own information.

For example: `$ mkdir <new_directory_name>`

In this example, you would replace `<new_directory_name>` with the name you want to give to your new directory.

## File Management Commands

With the `ls` command, you can hide patterns while listing contents of a directory.

```
$ ls
```

```
agatha_complete.txt agatha.txt doc.a doc.b doc.c hello.sh sample.sh
```

```
$ ls --hide="*.txt" --hide="[a-m]*"
```

```
sample.sh
```

Also, give multiple filters by and match patterns as well, like `[a-m]`.

---

List all files that have execution permission using:

```
$ ls -l | grep '^-..x'
```

```
-rwxrwxr-x 1 linuxhandbook linuxhandbook  0 Aug 13 16:25 sample.sh
```

---

The `--hyperlink` option creates clickable file and directory names in your terminal output. The files and directories become actual hyperlinks that you can click to open or navigate to them directly from the terminal.

```
$ ls --hyperlink=auto    # Only when output is to terminal
$ ls --hyperlink=always  # Always show hyperlinks
$ ls --hyperlink=never   # Never show hyperlinks
```

---

The **Archive Mode** in `cp command` maintains all file attributes, permissions, timestamps, and symbolic links exactly as the original.

```
$ cp -a /source/directory/ /destination/
```

---

The `-u` option in `cp` command updates files only when the source is more recent than the destination, perfect for incremental backups.

```
$ cp -u *.txt /backup/directory/
```

---

Interactively copy data using the `-i` option in `cp` command:

```
$ cp -i --verbose file.txt destination/
```

It provides detailed information before overwriting, showing file sizes and timestamps.

---

There is an option for making automatic backups. If you use `-b` option with the `cp` command, it will overwrite the existing files but before that, it will create a backup of the overwritten files.

```
$ cp -b file.txt target_dir/file.txt
```

---

You can display the file copying progress while copying a super large file:

```
$ pv testfile.img > Documents/testfile.img
```

```
1.10GiB 0:00:01 [1.10GiB/s] [=====>] 21% ETA 0:00:03
```

---

With the `mv command`, you can swap the contents of two files atomically without using a temporary file, perfect for configuration file backups.

```
$ mv --exchange file1.txt file2.txt
```

Instead of moving one file to the location of another, it exchanges their places in a single operation.

---

To prevent overwriting existing files, you can use the `-n` option. This way, `mv` won't overwrite existing files.

```
$ mv -n source_file target_directory
```

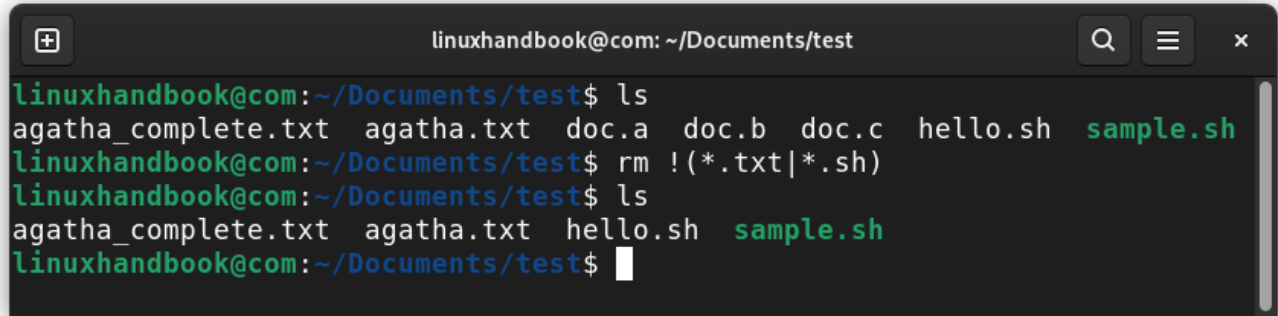
Or, do it interactively!

```
$ mv -i source_file target_directory
```

---

In [rm command](#), you can use extended globbing to remove all files except specified patterns.

```
$ rm !(*.txt|*.csv)
```

A terminal window titled 'linuxhandbook@com: ~/Documents/test' showing a sequence of commands and their output. The user runs 'ls' and lists files: 'agatha\_complete.txt', 'agatha.txt', 'doc.a', 'doc.b', 'doc.c', 'hello.sh', and 'sample.sh'. Then they run 'rm !(\*.txt|\*.sh)', which removes 'doc.a', 'doc.b', and 'doc.c'. A second 'ls' command shows the remaining files: 'agatha\_complete.txt', 'agatha.txt', 'hello.sh', and 'sample.sh'.

```
linuxhandbook@com:~/Documents/test$ ls
agatha_complete.txt agatha.txt doc.a doc.b doc.c hello.sh sample.sh
linuxhandbook@com:~/Documents/test$ rm !(*.txt|*.sh)
linuxhandbook@com:~/Documents/test$ ls
agatha_complete.txt agatha.txt hello.sh sample.sh
linuxhandbook@com:~/Documents/test$
```

Figure 1: Remove except mentioned files.

This will remove all the files except those with the mentioned extensions.

---

To remove a file whose name starts with a '-', for example '-foo', use one of these commands:

```
$ rm -- -foo
```

OR

```
$ rm ./-foo
```

---

To remove a directory and its ancestors, use:

```
$ rmdir -p a/b/c
```

This is similar to `rmdir a/b/c a/b a`. The directories should be empty for [rmdir command](#) to work.

---

You can add the following line to your `~/.bashrc` for case-insensitive tab completion for filenames and directories.

```
$ bind 'set completion-ignore-case on'
```

That is, you don't need to use a **D**oc and tab to move into *Documents*, instead, just use **d**oc and tab!

---

Navigate to another user's home directory using [cd command](#):

```
$ cd ~user
```

```
$ cd ~itsfoss
```

Also, move into directories inside that user's home directory as well:

```
$ cd ~itsfoss/Documents
```

---

Set [CDPATH](#) to your most-used parent directories and jump to any subdirectory from anywhere in your filesystem. Add it to your `~/.bashrc` like:

```
$ export CDPATH=".:~/var/log:/etc:/opt"
```

Always include `"."` first to preserve normal relative navigation, then add your frequent project roots for lightning-fast directory hopping.

---

Reading the [PATH variable](#) output can be made easy by using the `tr` command. Use a newline character as the delimiter instead of the `:`:

```
$ echo $PATH | tr : "\n"
```

```
/usr/local/bin
/usr/bin
/bin
/usr/local/games
/usr/games
```

---

You can create sophisticated directory hierarchies with multiple levels and branching patterns in single [mkdir command](#).

```
$ mkdir -p company/{departments/{hr,it,finance},projects/{web,mobile,desktop}}/archives
```

This will create a directory structure:

```
└─ company
   └─ departments
      │   └─ finance
      │       └─ archives
      │   └─ hr
      │       └─ archives
      │   └─ it
      │       └─ archives
      └─ projects
         └─ desktop
             └─ archives
         └─ mobile
             └─ archives
         └─ web
             └─ archives
```

---

Need a safe, unique temporary file or directory in your scripts? Use:

```
$ mktemp
/tmp/tmp.tygPAqouFZ
```

```
$ mktemp -d
/tmp/tmp.P59I0iBd5k
```

It will create a temporary file with a unique name in the `/tmp` directory.

Use `-d` option for temporary directory.

---

With the [cat command](#), number all the non-empty line using:

```
$ cat -b stories.txt
 1 The Mysterious Affair at Styles
```

```
2 The Murder on the Links
3 The Man in the Brown Suit

4 The Murder of Roger Ackroyd
5 The Big Four
6 The Mystery of the Blue Train

7 The Murder at the Vicarage
8 Giant's Bread
9 The Floating Admiral
```

---

Does multiple consecutive empty lines bother you? Print by avoiding those using:

```
$ cat -s file.txt
```

---

You can use shell globbing and brace expansion to show multiple file results in [head command](#):

```
$ head -n 5 *.{txt,sh}
```

```
==> agatha.txt <==
The Mysterious Affair at Styles
The Murder on the Links
The Man in the Brown Suit
The Murder of Roger Ackroyd
The Big Four
```

```
==> sample.sh <==
echo "The Big Four"
echo "The Mystery of the Blue Train"
echo "The Murder at the Vicarage"
echo "Giant's Bread"
echo "The Floating Admiral"
```

This will show first 5 lines of files with extension `.txt` and `.sh`.

---

You can exclude a specific number of lines at the end of the file and print the remaining content of the file by providing a negative number to `-n` option.

```
$ head -n -15 file.txt
```

---

Monitor multiple log files in real-time, displaying new content as it appears across different files with clear headers.

```
$ tail -f /var/log/syslog /var/log/auth.log /var/log/apache2/access.log
```

Globbing works [tail command](#) as well:

```
$ tail -f /var/log/{syslog,auth.log,kern.log}
```

---

To automatically stop monitoring a log file when the associated process dies:

```
$ tail -f --pid=$(pgrep nginx) /var/log/nginx/access.log
```

In the above command, the tail command stops, when the nginx process is terminated.

---

If you have a log file that changes less often, you can adjust the polling frequency in tail command:

```
$ tail -f -s 5 logfile.log
```

Here, the command checks every 5 seconds for changes.

---

To display lines from number x to y, combine head and tail:

```
$ head -y lines.txt | tail +x
```

You can also use [sed command](#) for this purpose:

```
$ sed -n 'x,yp' lines.txt
```

---

Not liking the search function in [less command](#)? You can enable real-time search as you type characters, showing matches immediately without waiting for Enter.

```
$ less --incsearch file.txt
```

Now, press the / key and the search results will be highlighted as you type.

---

While viewing a file inside less, you can set a mark to the current line (the one at the bottom) using:

```
m<a character>
ma
```

The above command will set **a** as mark for the current line. Use:

```
'a
```

To go to the marked line later.

---

While reading a file inside less, if you want to copy that file to some other location, use:

```
! cp % <destination>
```

Here the % refers to the current file. The ! invokes a shell to run the shell-command given.

---

While viewing a file in less, you can get a small status by pressing the = key. This will display a small bar on the bottom that shows current line numbers in the view, total line number, percent of the file already scrolled past, etc.

---

Finds files within specific size ranges using multiple size criteria:

```
$ find . -size +100M -size -2G
./testfile2.img
./testfile1.img
```

The above command will find files between 100MB and 2GB.

---

You can find files and prompts for confirmation before deletion, showing file details for informed decision-making.

```
$ find . -name "*.tmp" -exec rm -i {} \;
rm: remove regular file './testfile1.tmp'? y
```

---

The [find command](#) keeps on searching for all the matches. Make it stop after the first match with -quit:

```
$ find . -name <search> -print -quit
```



---

You can find all shell script files in a given directory and give execution permission at once.

```
$ find /path/to/directory -name "*.sh" -exec chmod +x {} \;
```

---

To find all the files that were modified in the last 5 minutes:

```
$ find . -type f -mmin -5
```

Useful when you troubleshoot something and want to know which files were modified recently.

---

Get some context to your [grep](#) search by using `-C`

```
$ grep -C3 "search-string" filename
```

```
$ grep -C3 "Mystery" agatha.txt
```

```
The Murder at the Vicarage
Giant's Bread
The Floating Admiral
The Sittaford Mystery
Peril at End House
Lord Edgware Dies
Murder on the Orient Express
```

This will show 3 lines before and after the match.

---

You can use `grep` to display all the lines that DO NOT match the given pattern.

```
$ grep -v search_pattern filename
```

---

If you are searching for a specific set of patterns using `grep`, better use a pattern file, than providing it as inline options.

```
$ grep -e "Mystery" -e "Lord" -e "House" log.txt
```

Can be replaced with:

```
$ grep -f patterncheck.txt agatha.txt
```

```
The Sittaford Mystery
Peril at End House
Lord Edgware Dies
```

Where, `patterncheck.txt` will be:

```
Mystery
Lord
House
```

---

You can use the [grep](#), [xargs](#), and [sed](#) command to search and replace items in multiple files at once.

For example, the command below will replace all occurrences of word “foo” with “bar” in all shell script files (.sh) in the current directory.

```
$ grep -rl 'foo' --include="*.sh" . | xargs sed -i 's/foo/bar/g'
```

---

If you are using [sort](#) and [uniq](#) command often, why not use the `-u` option in `sort` command instead?

```
$ sort -u file.txt
```

Both produce identical output.

---

It is possible to show disk usage for all files including hidden ones while excluding specific file patterns in [du command](#):

```
$ du -ah --exclude="*.txt" ~/Documents
```

```
4.0K   /home/linuxhandbook/Documents/sample.tar.gz
4.0K   /home/linuxhandbook/Documents/testfile
1.1G   /home/linuxhandbook/Documents/testfile2.img
2.1G   /home/linuxhandbook/Documents/testfile3.img
4.0K   /home/linuxhandbook/Documents/new.sh
4.0K   /home/linuxhandbook/Documents/sample.sh
3.1G   /home/linuxhandbook/Documents
```

If you are more into disk usage, try `ncdu` as well.

---

Got gzipped log files to investigate? Instead of extracting the `.gz` files and then looking into them, you can use:

- `zcat`
- `zless`
- `zgrep`
- `zdiff`

And a bunch of other ‘z commands’ on the zipped files directly.

---

Quickly create a file of predefined big size with [fallocate command](#):

```
fallocate -l 1G testfile1.img
```

---

Find it difficult to understand [file ownership and permission](#) with `ls -l` command?

Use `getfacl` command instead:

```
$ getfacl sample.sh
```

```
# file: sample.sh
# owner: linuxhandbook
# group: linuxhandbook
user::rwx
group::rwx
other::r-x
```

---

You can interactively password protect a zip file while creating it. For this, use the `-e` option of `zip` command.

```
$ zip -e filename.zip <your-files-to-zip>
```

Enter password:

Verify password:

```
adding: agatha.txt (deflated 24%)
adding: new.sh (stored 0%)
adding: patterncheck.txt (stored 0%)
adding: sample.sh (deflated 47%)
adding: sample.tar.gz (stored 0%)
adding: testfile/ (stored 0%)
```

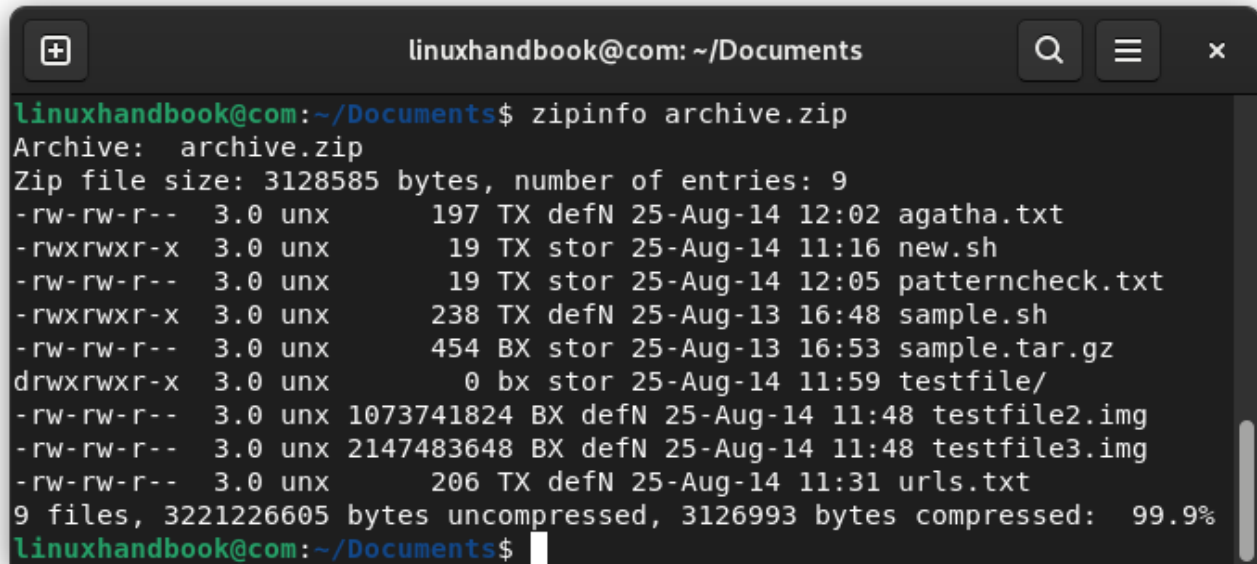
```
adding: testfile2.img (deflated 100%)
adding: testfile3.img (deflated 100%)
adding: urls.txt (deflated 57%)
```

This will prompt for password. Enter and verify it. While unzipping using the `unzip` command, you need to use the password. If the uncompressed file exists already in the directory, it will ask several options like replace, rename, etc.

---

Use the `zipinfo` command to get the details of a zip archive.

```
$ zipinfo <filename>
```

A terminal window titled 'linuxhandbook@com: ~/Documents' showing the output of the 'zipinfo archive.zip' command. The output lists 9 files with their permissions, sizes, compression methods, and timestamps. At the bottom, it summarizes the total uncompressed and compressed sizes, showing a 99.9% compression ratio.

```
linuxhandbook@com:~/Documents$ zipinfo archive.zip
Archive:  archive.zip
Zip file size: 3128585 bytes, number of entries: 9
-rw-rw-r--  3.0 unx      197 TX defN 25-Aug-14 12:02 agatha.txt
-rwxrwxr-x  3.0 unx       19 TX stor 25-Aug-14 11:16 new.sh
-rw-rw-r--  3.0 unx       19 TX stor 25-Aug-14 12:05 patterncheck.txt
-rwxrwxr-x  3.0 unx      238 TX defN 25-Aug-13 16:48 sample.sh
-rw-rw-r--  3.0 unx      454 BX stor 25-Aug-13 16:53 sample.tar.gz
drwxrwxr-x  3.0 unx        0 bx stor 25-Aug-14 11:59 testfile/
-rw-rw-r--  3.0 unx 1073741824 BX defN 25-Aug-14 11:48 testfile2.img
-rw-rw-r--  3.0 unx 2147483648 BX defN 25-Aug-14 11:48 testfile3.img
-rw-rw-r--  3.0 unx      206 TX defN 25-Aug-14 11:31 urls.txt
9 files, 3221226605 bytes uncompressed, 3126993 bytes compressed:  99.9%
linuxhandbook@com:~/Documents$
```

Figure 2: Zipinfo command

You can use the `-l` option to get the detailed view of the contents of the archive.

```
$ zipinfo -l <filename>
```

---

You can extract selected files from an archive without extracting everything using [tar command](#).

First, check the contents using:

```
$ tar -tf sample.tar --wildcards "*.txt"
./agatha.txt
./patterncheck.txt
./testfile/urls.txt
./testfile/agatha.txt
./urls.txt
```

Now, extract the selected files.

```
$ tar -xf sample.tar --wildcards "*.txt"
```

OR

```
$ tar -xf sample.tar --wildcards ./agatha.txt ./urls.txt
```

---

```
linuxhandbook@com: ~/Documents
linuxhandbook@com:~/Documents$ zipinfo -l archive.zip
Archive:  archive.zip
Zip file size: 3128585 bytes, number of entries: 9
-rw-rw-r--  3.0 unx      197 TX      162 defN 25-Aug-14 12:02 agatha.txt
-rwxrwxr-x  3.0 unx       19 TX        31 stor 25-Aug-14 11:16 new.sh
-rw-rw-r--  3.0 unx       19 TX        31 stor 25-Aug-14 12:05 patterncheck.txt
-rwxrwxr-x  3.0 unx      238 TX      137 defN 25-Aug-13 16:48 sample.sh
-rw-rw-r--  3.0 unx      454 BX      466 stor 25-Aug-13 16:53 sample.tar.gz
drwxrwxr-x  3.0 unx        0 bx        0 stor 25-Aug-14 11:59 testfile/
-rw-rw-r--  3.0 unx 1073741824 BX 1042063 defN 25-Aug-14 11:48 testfile2.img
-rw-rw-r--  3.0 unx 2147483648 BX 2084099 defN 25-Aug-14 11:48 testfile3.img
-rw-rw-r--  3.0 unx       206 TX       100 defN 25-Aug-14 11:31 urls.txt
9 files, 3221226605 bytes uncompressed, 3126993 bytes compressed:  99.9%
linuxhandbook@com:~/Documents$
```

Figure 3: Zipinfo with details

## Disk Management Commands

Quickly identifying external USB storage devices using [lsblk command](#):

```
$ lsblk -o NAME,SIZE,TYPE,TRAN,MODEL | grep usb
```

```
sdb      28.7G disk usb      Cruzar Blade
```

Use [watch](#) and [free](#) command together for real-time memory tracking at specified intervals.

```
$ watch -n 1 'free -h'
```

```
Every 5.0s: free -h
```

```
com: Wed Aug 13 16:14:55 2025
```

	total	used	free	shared	buff/cache	available
Mem:	3.8Gi	2.2Gi	416Mi	79Mi	1.5Gi	1.6Gi
Swap:	1.3Gi	524Ki	1.3Gi			

The above command refreshes every second.

Find Gigabytes sized directories in Linux using the [du command](#):

```
$ du -h / 2>/dev/null | grep '^[0-9.]*G' | sort -hr | head -10
```

```
11G /
5.7G /usr
3.9G /var
3.5G /var/lib
3.3G /var/lib/flatpak
3.2G /usr/lib
2.7G /var/lib/flatpak/runtime
1.7G /usr/share
1.6G /usr/lib/x86_64-linux-gnu
1.3G /var/lib/flatpak/runtime/org.gnome.Sdk/x86_64/48/
```

---

You can remove empty files using the [find command](#):

```
$ find . -type f -size 0 -delete
```

Or, remove empty directory recursively using:

```
$ find . -type d -empty -delete
```

---

You can see how much disk space is used by stored system logs with the [journalctl command](#):

```
$ journalctl --disk-usage
```

Archived and active journals take up 139.9M in the file system.

---

You can edit the *journalld.conf* file:

```
$ sudo nano /etc/systemd/journalld.conf
```

And add the line:

```
SystemMaxUse=1G
```

To limit the total journal size to 1GB.

---

## Process Management Commands

The [top command](#) doesn't have to run continuously, taking your terminal prompt all the time. You can run it non-interactively like this:

```
$ top -b -n 1
```

It displays a one-time snapshot of running processes and system information.

---

To kill a process accessing a specific port, save time by using **fuser** command.

For example, terminate a process on TCP port 3306:

```
$ fuser -k 3306/tcp
```

---

The [ps command](#) can display all processes in a hierarchical tree format with ASCII art branching.

```
$ ps -AH --forest -o pid,ppid,user,%cpu,%mem,cmd | less
```

It shows parent-child relationships along with essential system information, that we asked in the **-o** option.

---

With the **ps** command, you can list the top N process by memory usage.

```
$ ps -eo pid,ppid,cmd,%mem --sort=-%mem | head -n 5
```

PID	PPID	CMD	%MEM
5908	5737	/usr/bin/gnome-shell	11.2
6357	5874	/usr/bin/gnome-software --g	8.3
7783	5737	/usr/bin/nautilus --gapplic	6.6
6300	5908	/usr/libexec/mutter-x11-fra	3.1

Or, by CPU usage:

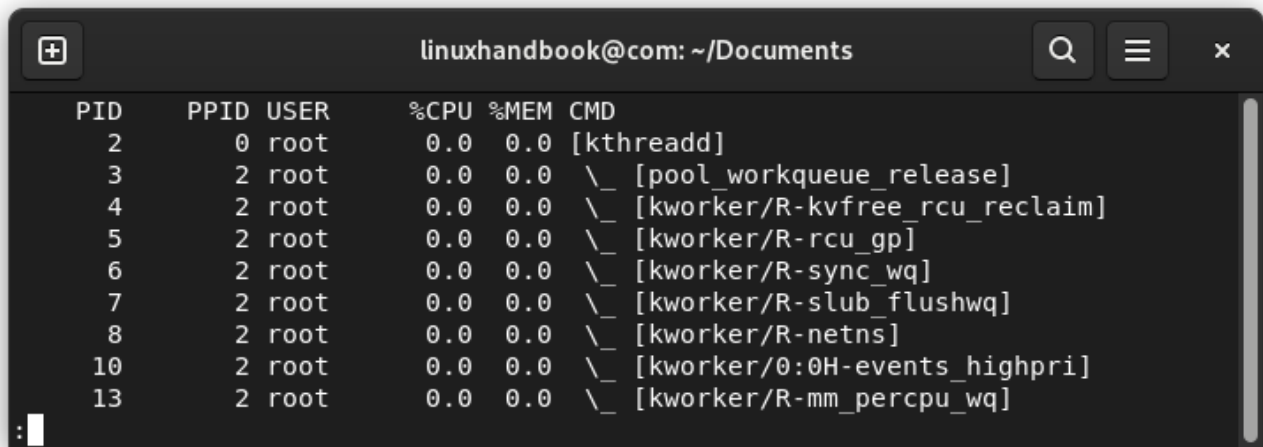


Figure 4: Process Heirarchical Tree

```
$ ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -n 5
```

PID	PPID	CMD	%CPU
7931	7101	ps -eo pid,ppid,cmd,%cpu --	100
5908	5737	/usr/bin/gnome-shell	3.9
7783	5737	/usr/bin/nautilus --gapplic	1.8
6357	5874	/usr/bin/gnome-software --g	1.2

You can use the [watch command](#) in Linux to spot the changes happening to a directory.

Let's say we need to track the changes in the directory ~/Documents/test. We will use the command:

```
$ watch -d "ls -l ~/Documents/test"
```

This will update every two seconds (default) the changes like new directories, files, change area like modification time, etc.

You can use the [pgrep command](#) with the `-t` or `--terminal` option to list processes associated with a specific terminal.

To get the list of open terminals, you can list the `/dev/pts/` directory.

```
$ ls /dev/pts/
```

```
0 1 2 3 ptmx
```

To get the name of your current terminal session, use:

```
$ tty
/dev/pts/0
```

Once you get the name, use the command (Should not use `/dev/`):

```
$ pgrep --terminal pts/0
7101
$ pgrep --terminal pts/1
7934
7938
```

If you have to [find the process ID](#), you will use the pidof command.

```
$ pidof <process_name>
```

Similarly, if you have a PID number of a process, you can find the name of that PID by using the command:

```
$ ps -q <PID Number> -o comm=
```

```
$ ps -q 7938 -o comm=
top
```

Also, you can find the PID of only processes with a particular name (like gedit), using:

```
$ ps -C <name-of-process> -o pid=
```

```
$ ps -C top -o pid=
7938
```

---

In Linux, child process goes into zombie state when the parent process is not aware of its death. These zombie processes (also known as defunct processes) will stay in the system, unless treated properly.

Get the PID of the zombie process and the PID of its parent process using:

```
$ ps -A -ostat,pid,ppid | grep -e '[zZ]'
```

Now, you can notify the parent process to clear the zombies using. This will most likely be ignored.

```
$ kill -s SIGCHLD <parent_PID>
```

At this point, try gracefully kill using `kill <parent_PID>` and then forcefully [kill the parent process](#) using the `kill -9 <parent_PID>` command.

---

You can use the pstree command to show process hierarchy visually.

```
$ pstree
```

---

## Networking Commands

Easily find your gateway IP using the [ping command](#)!

```
$ ping _gateway
```

---

Use ping command with the -a option to get an audible ping!

```
$ ping -a google.com
```

---

You can send exactly specified number of pings and then stop using:

```
$ ping -c 4 google.com
```

```
PING google.com (142.251.221.174) 56(84) bytes of data.
```

```
64 bytes from pnmaaa-au-in-f14.1e100.net (142.251.221.174): icmp_seq=1 ttl=255 time=21.4 ms
```

```
64 bytes from pnmaaa-au-in-f14.1e100.net (142.251.221.174): icmp_seq=2 ttl=255 time=21.5 ms
```

```
64 bytes from pnmaaa-au-in-f14.1e100.net (142.251.221.174): icmp_seq=3 ttl=255 time=21.6 ms
```

```
64 bytes from pnmaaa-au-in-f14.1e100.net (142.251.221.174): icmp_seq=4 ttl=255 time=21.3 ms
```

```
--- google.com ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
```

---

If downloading with curl delay is higher than what you would like to wait, you can specify a 'timeout' duration.

```
$ curl --connect-timeout <duration> <URL>
```

---

With the [curl command](#), you can download a file with the condition, the file's modification time is latest than the given time.

```
$ curl url -z "DD MMM YY MM:HH:SS"
```

---

You can download multiple items from links stored in a text file using [wget command](#):

```
$ wget -i urls.txt -P ~/Downloads/batch/
```

This will download the items to the mentioned directory. Directory will be created if not exists.

---

If you have [lsof command](#) installed, you can see which processes are using network connection using:

```
$ lsof -P -i -n
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
gnome-sof	6357	linuxhandbook	42u	IPv4	39817			

---

## User and Group Management

You can lock a specific user temporarily using [usermod command](#):

```
$ sudo usermod -L username
```

This will disable their password authentication. To unlock:

```
$ sudo usermod -U username
```

---

You can force a user to change password at custom time periods, you can use the [passwd](#) and [chage](#) command.

```
$ sudo passwd -n 7 -x 90 -w 14 -i 7 username && sudo chage -d 0 username
```

The above command will force the user to change the password every 90 days. Also, it expires the current password immediately, forcing the user to set a new password. There is a 14 day warning (-W) and a 7 day inactive period (-I). Still not logged in after this period, the account is locked.

---

With the [chown command](#), you can change the user and group ownership of a file:

```
$ sudo chown user_name:group_name file_name
```

Or, if you want to change the ownership of files inside a directory recursively,

```
sudo chown -R user_name:group_name directory_name
```

---

You can use the **last** command to show login history for specific users:

```
$ last username
```

linuxhan	tty2	Thu Aug 14 11:11 - still logged in
linuxhan	tty3	Wed Aug 13 16:13 - still logged in



It will display when and from where user logged in.

---

You can use the [chmod command](#) to copy the permission of a file to another file.

```
$ chmod --reference hello world
```

This makes file2 have the exact same permission settings as file1.

---

With the [who command](#), you can check when was the last system boot.

```
$ who -b
```

```
system boot 2025-08-14 16:34
```

---

Set specific file permissions directly during directory creation, avoiding the need for separate chmod commands.

```
$ mkdir -m 755 public_dir
```

```
$ mkdir -m 700 private_dir
```

```
$ mkdir -m a=rwx shared_dir
```

```
$ ls -lA
```

```
total 12
```

```
drwx----- 2 linuxhandbook linuxhandbook 4096 Aug 13 16:39 private_dir
```

```
drwxr-xr-x 2 linuxhandbook linuxhandbook 4096 Aug 13 16:39 public_dir
```

```
drwxrwxrwx 2 linuxhandbook linuxhandbook 4096 Aug 13 16:39 shared_dir
```

---

## Editor Specific Commands

While opening a file in [nano editor](#), you can specify where to place the cursor inside that file.

```
$ nano +c/Foo <filename>
```

Where,

- **+**: Search for the specified word from the beginning of the text. Use **-** instead, to start searching for the word from the end of the file.
  - **c**: Small letter c indicates a case-sensitive search.
- 

In nano editor, use the **-v** option to start in a read-only mode.

```
$ nano -v <filename>
```

---

While using nano editor, press

```
Alt+#
```

```
# This is `ALT + SHIFT + 3` in major keyboards.
```

To display line numbers. Press again to hide line numbers.

---

Launch [Vim](#) with multiple files opened in individual tabs:

```
$ vim -p file1.py file2.py file3.py
```

---

In Vim, while your cursor is under a word in normal mode, press **\*** (SHIFT + 8). This will search forward with that word.

Instead, press **#** (SHIFT + 3) to search backward.

---

In Vim, you can add semi-colon to the end of a line based on pattern. For this, use the command:

```
:g/background/normal A;
```

To find lines containing the word *background* and add a semi-colon at the end of the line.

---

In Vim, to jump to nth character on current line, use:

```
n|
```

For example, to go to the 15th character on the current line, use:

```
15|
```

---

In Vim, use the **:earlier** command to undo changes in specified time. Say:

```
:earlier 2m
```

To undo all changes made in the last 2 minutes.

---

Reverted changes too far? You can use the **:later** command to go forward in time.

```
:later 30s
```

Will jump forward to the state your file was in 30 seconds later than the current undo position.

Use **:undolist** to view the undo tree structure.

---

## Miscellaneous

Seeing duplicate command entries in the shell history? Not very useful.

Avoid saving duplicate command entries.

Set variable **HISTCONTROL** to **ignoredups** or **ignoreboth** (ignores case as well).

You'll have a lean and clean history now.

---

Display command history with time and date they were run on with:

```
$ export HISTTIMEFORMAT="%d/%m/%y %T"
```

```
41 14/08/25 11:17:11who -b
42 14/08/25 11:18:04export HISTTIMEFORMAT="%d/%m/%y %T"
43 14/08/25 11:18:06history
```

---

You can use the **fc** command to edit a range of commands from history. For example:

```
$ fc 129 130
```

It will open in text editor. Make the modification as desired. Saving and exiting the editor will run the command.

---

Replace part of a command with `fc`.

Say, `apt` is mistyped as `atp`, you can replace it in the command using:

```
$ fc -s atp=apt
```

---

You can use:

```
$ ^old^new
```

To replace the first occurrence of “old” with “new” in your previous command.

Let’s say you ran:

```
$ sudo atp update
```

Use:

```
$ ^atp^apt
sudo apt update
```

To run the proper `sudo apt update` command.

---

Put a space before a command and it won’t be registered in the history.

---

You can spell-check text files in the Linux terminal itself with:

```
$ spell filename
```

It will check for spelling mistakes in the given file.

---

You can truncate the number of directories shown on the prompt, when inside a very long path. To do this, add the following line to the end of your `~/.bashrc` file.

```
PROMPT_DIRTRIM=3
```

This will make only the current directory and two parent directory on the prompt and the rest are truncated.

---

To create environment variable in bash, you use [export command](#):

```
$ export my_var
```

You can use the same export command to remove an exported variable:

```
$ export -n my_var
```

## Keyboard shortcuts

You can navigate the current line in a terminal using:

- CTRL + a: Move cursor to beginning of the line
  - CTRL + e: Move cursor to the end of the line
- 

While entering a command in the terminal, use:

- CTRL + u: Delete everything before cursor position

- **CTRL + k**: Delete everything after cursor position
- 

While entering a long command in the terminal, press **CTRL + x** and then **CTRL + e** to open current command in default editor.

Edit the command in the editor. Save and exit will run the edited command.

---

You can manipulate text while typing commands in the terminal.

- **Alt+u**: Uppercase current or next word from cursor
  - **Alt+l**: Lowercase current or next word from cursor
  - **Alt+c**: Capitalize first letter of current/next word
- 

If you are using glob patterns in your commands, you can expand them while editing.

After entering the glob pattern:

```
ls *.txt
```

Press **CTRL + x** and then **\*** (**SHIFT + 8**).

The preceding glob pattern will be expanded in the terminal.

---

You can insert last argument from previous command using:

**ALT + .**

---

These are the ways you can reuse the arguments from the last command:

- **!^**: Grab just the first parameter from your last command.
  - **!\***: Reuse all parameters while changing the command.
- 

Enter key is not working? Use **CTRL + m** (Carriage Return) instead.

In modern terminals, **Ctrl+M** is almost always mapped to the same function as the Enter key!

---

# Linux Command Tips

---

Master the Linux command line with practical, real-world solutions that system administrators use every day. This comprehensive collection brings together over 100 essential tips and techniques that go beyond basic command syntax to solve actual problems you'll encounter in production environments.

---