

File and Directory Management

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. ls 1

2. pwd 7

3. cd 11

4. mkdir 17

5. rmdir 22

6. rm 27

7. cp 33

8. mv 39

9. touch 44

10. find 49

11. locate 56

12. tree 62

1. ls

The `ls` command is one of the most commonly used Linux commands. It is used to **list files and directories** in the current directory or a specified directory.

Syntax

```
ls [OPTIONS] [DIRECTORY]
```

- **OPTIONS** – Modify how the output is displayed.
- **DIRECTORY** – The directory whose contents you want to view. If omitted, `ls` lists the current directory.

Basic Examples

1. List files in the current directory

```
ls
```

Example Output

```
Documents  Downloads  file.txt  Pictures
```

---

2. List contents of another directory

```
ls /home/user
```

Lists all files and directories inside `/home/user`.

Common Options

`ls -l` (Long Listing Format)

Displays detailed information.

```
ls -l
```

Example Output

```
-rw-r--r-- 1 user user 1024 Jul 23 10:20 file.txt
drwxr-xr-x 2 user user 4096 Jul 23 09:00 Documents
```

Columns Explained

Column	Meaning
`-rw-r--r--`	File type and permissions
`1`	Number of hard links
`user`	Owner
`user`	Group
`1024`	File size (bytes)
`Jul 23 10:20`	Last modification date/time
`file.txt`	File name

`ls -a`

Shows **all files**, including hidden files (those beginning with `.`).

```
ls -a
```

Example:

```
.  ..  .bashrc  .profile  file.txt
```

`ls -la`

Shows **all files** in **long format**.

```
ls -la
```

`ls -lh`

Shows file sizes in a human-readable format.

```
ls -lh
```

Example:

```
-rw-r--r-- 1 user user 1.5K Jul 23 file.txt
```

Instead of:

```
1536
```

`ls -R`

Lists files **recursively**.

```
ls -R
```

Shows the contents of all subdirectories.

`ls -t`

Sorts files by modification time (newest first).

```
ls -t
```

`ls -r`

Reverses the sorting order.

```
ls -r
```

`ls -S`

Sorts files by size (largest first).

```
ls -S
```

`ls -i`

Displays the inode number of each file.

```
ls -i
```

Example:

```
12345 file.txt
12346 Documents
```

`ls -F`

Appends symbols to indicate file types.

```
ls -F
```

Example:

```
Documents/
script*
link@
```

- `/` → Directory
- `*` → Executable
- `@` → Symbolic link

`ls -d`

Lists directory names themselves instead of their contents.

```
ls -d /home/*
```

Useful Combinations

List all files with details

```
ls -al
```

Human-readable sizes

```
ls -lh
```

Show hidden files with human-readable sizes

```
ls -lah
```

Sort by size

```
ls -lhS
```

Sort by newest files

```
ls -lt
```

Reverse order

```
ls -ltr
```

Oldest files appear first.

Practical Examples

View files in `/etc`

```
ls /etc
```

View details of your home directory

```
ls -l ~
```

List only hidden files

```
ls -d .*
```

Count files in the current directory

```
ls | wc -l
```

Exit Status

- **0** – Command executed successfully.
- **Non-zero** – An error occurred (such as a directory not existing or insufficient permissions).

Example:

```
$ ls /doesnotexist

ls: cannot access '/doesnotexist': No such file or directory
```

Summary of Common Options

Command	Description
`ls`	List files and directories
`ls -l`	Long listing format
`ls -a`	Include hidden files
`ls -la`	Long listing including hidden files
`ls -lh`	Human-readable file sizes
`ls -R`	Recursive listing
`ls -t`	Sort by modification time
`ls -r`	Reverse sort order
`ls -S`	Sort by file size
`ls -i`	Show inode numbers
`ls -F`	Append file type indicators
`ls -d`	Show directory names instead of contents

Key Points

- `ls` is the standard command for viewing directory contents.
- Without options, it lists only visible files in the current directory.
- `ls -la` is one of the most frequently used combinations because it shows detailed information about both visible and hidden files.
- Combine options (for example, `ls -lah`, `ls -ltr`, or `ls -lhS`) to customize the output for different tasks.

2. pwd

The `pwd` command displays the **absolute path of the current working directory** (the directory you are currently in).

Syntax

```
pwd
```

Example

```
$ pwd
/home/bharath/Documents
```

This output means you are currently inside the `Documents` directory located in `/home/bharath`.

Purpose

- Find your current location in the filesystem.
- Verify which directory you are working in before creating, deleting, or modifying files.
- Useful in shell scripts to determine the current working directory.

Common Examples

1. Display the current directory

```
pwd
```

Output:

```
/home/user
```

2. After changing directories

```
cd /etc
pwd
```

Output:

```
/etc
```

---

3. Inside a nested directory

```
cd /var/log/apache2
pwd
```

Output:

```
/var/log/apache2
```

Options

`pwd -L` (Logical Path)

Displays the logical directory path, preserving symbolic links.

Example:

```
pwd -L
```

`pwd -P` (Physical Path)

Displays the actual physical directory by resolving symbolic links.

Example:

```
pwd -P
```

Difference Between `-L` and `-P`

Suppose:

```
/home/user/project -> /mnt/storage/project
```

where `project` is a symbolic link.

Commands:

```
cd /home/user/project
pwd -L
```

Output:

```
/home/user/project
```

Now:

```
pwd -P
```

Output:

```
/mnt/storage/project
```

- `-L` shows the symbolic link path.
- `-P` shows the real filesystem path.

Exit Status

- `0` – Command executed successfully.
- Non-zero – An error occurred (rare, such as if the current directory is no longer accessible).

Practical Uses

- Confirm your current location before running commands like `rm`, `cp`, or `mv`.
- Display the current directory in shell scripts.
- Debug scripts that depend on the working directory.
- Verify that `cd` changed to the intended directory.

Related Commands

Command	Purpose
`cd`	Change the current directory
`ls`	List files and directories
`dirname`	Extract the directory portion of a pathname
`basename`	Extract the filename from a pathname
`realpath`	Show the canonical absolute path

Example Session

```
$ pwd
/home/bharath

$ cd Documents

$ pwd
/home/bharath/Documents

$ cd ..

$ pwd
/home/bharath
```

Summary

- **Full Form:** Print Working Directory
- **Purpose:** Displays the absolute path of the current directory.
- **Common Usage:** `pwd`
- **Logical Path:** `pwd -L`
- **Physical Path:** `pwd -P`

`pwd` is one of the most frequently used Linux commands because it helps you always know where you are in the filesystem.

3. cd

The `cd` command is a **shell built-in command** used to **change the current working directory**. It allows you to move between directories (folders) in the file system.

Syntax

```
cd [directory]
```

- `directory` – The path to the directory you want to move into.

Common Examples

1. Change to a specific directory

```
cd /home/bharath/Documents
```

Moves to the `Documents` directory.

---

2. Change to your home directory

```
cd
```

or

```
cd ~
```

Both commands take you to your home directory.

Example:

```
pwd
```

Output:

```
/home/bharath
```

3. Move to the parent directory

```
cd ..
```

Example:

Current directory:

```
/home/bharath/Documents
```

After:

```
cd ..
```

New directory:

```
/home/bharath
```

---

4. Move up two levels

```
cd ../..
```

Example:

```
/home/bharath/Documents/Projects
```

becomes

```
/home/bharath
```

---

5. Return to the previous directory

```
cd -
```

Example:

```
cd /etc
cd /var
cd -
```

Output:

```
/etc
```

This toggles between the current and previous directories.

6. Change using a relative path

Current directory:

```
/home/bharath
```

Command:

```
cd Downloads
```

Moves to:

```
/home/bharath/Downloads
```

---

7. Change using an absolute path

```
cd /usr/local/bin
```

Moves directly to:

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

Useful Options

Unlike many Linux commands, `cd` has very few options.

`cd -`

Return to the previous working directory.

`cd ~username`

Go to another user's home directory (if permissions allow).

Example:

```
cd ~john
```

Special Directory Symbols

Symbol	Meaning
.	Current directory
..	Parent directory
~	Current user's home directory
-	Previous directory

Examples:

```
cd .
```

Stay in the current directory.

```
cd ..
```

Move up one directory.

```
cd ~
```

Go to your home directory.

```
cd -
```

Go back to the previous directory.

Check Your Current Directory

Use the pwd command:

```
pwd
```

Example:

```
cd /etc
pwd
```

Output:

```
/etc
```

Common Errors

Directory does not exist

```
cd abc
```

Output:

```
bash: cd: abc: No such file or directory
```

Permission denied

```
cd /root
```

Output (for a non-root user):

```
bash: cd: /root: Permission denied
```

Practical Example

```
pwd
```

Output:

```
/home/bharath
```

```
cd Documents
pwd
```

Output:

```
/home/bharath/Documents
```

```
cd ..
pwd
```

Output:

```
/home/bharath
```

```
cd /
pwd
```

Output:

```
/
```

```
cd -
pwd
```

Output:

```
/home/bharath
```

Key Points

- `cd` stands for **Change Directory**.
- It is a **shell built-in command**, not an external executable.
- It changes the shell's current working directory.
- Supports **absolute paths** (e.g., `/etc`) and **relative paths** (e.g., `Documents`).
- `cd` with no arguments or `cd ~` goes to your **home directory**.
- `cd ..` moves to the **parent directory**.
- `cd -` switches to the **previous working directory**.
- Use `pwd` to verify your current location in the file system.

4. `mkdir`

The `mkdir` command in Linux is used to **create one or more new directories (folders)**.

Syntax

```
mkdir [OPTIONS] DIRECTORY_NAME
```

Basic Example

Create a directory named `projects`:

```
mkdir projects
```

Result:

```
projects/
```

Common Options

1. Create Multiple Directories

```
mkdir dir1 dir2 dir3
```

Creates:

```
dir1/
dir2/
dir3/
```

---

2. Create Nested Directories (`-p`)

The `-p` (parents) option creates parent directories if they don't already exist.

```
mkdir -p college/semester6/linux
```

Creates:

```
college/
└── semester6/
    └── linux/
```

Without `-p`, this command would fail if `college` or `semester6` do not exist.

---

3. Set Permissions While Creating (`-m`)

```
mkdir -m 755 myfolder
```

Permissions:

```
rwxr-xr-x
```

Meaning:

- Owner: Read, Write, Execute
- Group: Read, Execute
- Others: Read, Execute

---

4. Show Each Directory Created (`-v`)

```
mkdir -v test
```

Output:

```
mkdir: created directory 'test'
```

Practical Examples

Example 1

```
mkdir Documents
```

Creates:

```
Documents/
```

Example 2

```
mkdir Project1 Project2 Project3
```

Creates three directories.

Example 3

```
mkdir -p /home/student/Linux/Labs
```

```
Creates:

/home/student/
└── Linux/
    └── Labs/
```

Example 4

```
mkdir -pv backup/2026/July
```

Output:

```
mkdir: created directory 'backup'
mkdir: created directory 'backup/2026'
mkdir: created directory 'backup/2026/July'
```

Verify the Directory

List directories:

```
ls
```

Or:

```
ls -l
```

Common Errors

Error 1: Directory Already Exists

```
mkdir test
mkdir test
```

Output:

```
mkdir: cannot create directory 'test': File exists
```

Solution:

```
mkdir -p test
```

```
No error is shown if the directory already exists.
```

Error 2: Parent Directory Doesn't Exist

```
mkdir home/user/docs
```

Output:

```
mkdir: cannot create directory 'home/user/docs': No such file or directory
```

Solution:

```
mkdir -p home/user/docs
```

Error 3: Permission Denied

```
mkdir /root/test
```

Output:

```
mkdir: cannot create directory '/root/test': Permission denied
```

Solution (if you have administrative privileges):

```
sudo mkdir /root/test
```

Frequently Used `mkdir` Commands

Command	Description
`mkdir folder`	Create a single directory
`mkdir dir1 dir2`	Create multiple directories
`mkdir -p parent/child`	Create nested directories
`mkdir -m 755 folder`	Create a directory with specific permissions
`mkdir -v folder`	Display a message when creating a directory
`mkdir -pv dir/subdir`	Create parent directories and show progress

Summary

- **Command:** `mkdir`
- **Purpose:** Creates new directories (folders).
- **Most commonly used options:**
  - `-p` → Create parent directories as needed.
  - `-m` → Set directory permissions during creation.
  - `-v` → Show a message for each created directory.

Example

```
mkdir -pv /home/bharath/LinuxPractice/Commands
```

This command:

- Creates `LinuxPractice` and `Commands` if they don't already exist.
- Prints a message for each directory it creates.

5. `rmdir`

The `rmdir` command is used to **remove (delete) empty directories** from the filesystem.

Syntax

```
rmdir [OPTIONS] DIRECTORY...
```

- **DIRECTORY** – The name of the empty directory you want to remove.

Basic Examples

1. Remove an empty directory

```
rmdir testdir
```

Before:

```
testdir/
```

After:

```
Directory removed.
```

---

2. Remove multiple empty directories

```
rmdir dir1 dir2 dir3
```

This deletes all the specified directories if they are empty.

---

3. Remove nested empty directories

```
rmdir -p parent/child/grandchild
```

If all directories are empty, it removes:

```
grandchild
child
parent
```

Common Options

Option	Description
`-p`	Remove parent directories as they become empty.
`-v`	Display a message for each directory removed.
`--ignore-fail-on-non-empty`	Ignore errors if a directory is not empty.
`--help`	Display help information.
`--version`	Show version information.

Examples

Remove with verbose output

```
rmdir -v myfolder
```

Output

```
rmdir: removing directory, 'myfolder'
```

Remove parent directories

```
mkdir -p demo/a/b
rmdir -p demo/a/b
```

Result:

```
demo/
├── a/
│   └── b/
```

becomes

```
All directories removed.
```

Ignore non-empty errors

```
rmdir --ignore-fail-on-non-empty folder
```

If `folder` contains files:

- No error is displayed.
- The directory is not removed.

If the Directory is Not Empty

Suppose:

```
project/
├── file.txt
```

Command:

```
rmdir project
```

Output:

```
rmdir: failed to remove 'project': Directory not empty
```

`rmdir` **cannot delete directories that contain files or other directories**.

Difference Between `rmdir` and `rm -r`

Feature	`rmdir`	`rm -r`
Deletes empty directories	Yes	Yes
Deletes non-empty directories	No	Yes
Deletes files	No	Yes
Safer	Yes	Less safe (recursive deletion)

Example:

```
rm -r project
```

This removes:

- the directory
- all subdirectories
- all files inside it

Exit Status

- `0` – Directory removed successfully.
- Non-zero – An error occurred (directory not empty, does not exist, or insufficient permissions).

Real-World Usage

Remove an empty project folder

```
rmdir old_project
```

Remove nested empty directories

```
rmdir -p backups/2026/july
```

Remove several empty directories

```
rmdir temp1 temp2 temp3
```

Important Notes

- `rmdir` **only removes empty directories**.
- It **cannot delete files**.
- If a directory contains even one file or subdirectory, `rmdir` will fail.
- Use `rm -r` only when you intentionally want to remove a directory and **all of its contents**.

Summary

Item	Description
Command	`rmdir`
Purpose	Remove empty directories
Syntax	`rmdir [OPTIONS] DIRECTORY`
Recursive removal of empty parent directories	`rmdir -p`
Verbose mode	`rmdir -v`
Deletes non-empty directories?	No
Deletes files?	No
Safer than `rm -r`?	Yes

6. `rm`

The `rm` (remove) command is used to **delete files and directories** from the filesystem. Unlike moving files to a recycle bin, `rm` **permanently removes** them (unless your desktop environment intercepts it).

Syntax

```
rm [OPTIONS] FILE...
```

or

```
rm [OPTIONS] DIRECTORY...
```

Common Options

Option	Description
`-f`	Force removal without prompting, even if files are write-protected.
`-i`	Ask for confirmation before deleting each file.
`-I`	Ask once before deleting many files.
`-r` or `-R`	Remove directories and their contents recursively.
`-d`	Remove an empty directory.
`-v`	Display each file or directory as it is removed.

Examples

1. Delete a single file

```
rm file.txt
```

Deletes `file.txt`.

---

2. Delete multiple files

```
rm file1.txt file2.txt file3.txt
```

Deletes all three files.

---

3. Delete a file with confirmation

```
rm -i report.pdf
```

Output:

```
rm: remove regular file 'report.pdf'?
```

Type:

```
y
```

to delete.

---

4. Force delete a file

```
rm -f file.txt
```

Deletes the file without asking for confirmation.

---

5. Delete an empty directory

```
rm -d myfolder
```

Works only if `myfolder` is empty.

6. Delete a directory and everything inside it

```
rm -r myfolder
```

Removes:

```
myfolder/
├── file1
├── file2
└── subfolder/
```

Everything is deleted.

---

7. Force delete a directory recursively

```
rm -rf myfolder
```

This is one of the most powerful and dangerous Linux commands.

- `-r` → Recursive deletion
- `-f` → Force deletion without prompts

---

8. Show deleted files

```
rm -rv project/
```

Example output:

```
removed 'project/file1'
removed 'project/file2'
removed directory 'project'
```

---

9. Delete all `.log` files

```
rm *.log
```

Deletes every file ending with `.log` in the current directory.

10. Delete all files in a directory

```
rm /tmp/*
```

Deletes all non-hidden files in `/tmp`.

---

11. Delete hidden files

```
rm .*
```

Do not use this command blindly. It can match special entries and hidden configuration files. A safer pattern is:

```
rm .[!.]*
```

or

```
rm .??*
```

depending on what you intend to remove.

Exit Status

Exit Code	Meaning
`0`	Files deleted successfully.
Non-zero	An error occurred (e.g., file not found, permission denied).

Common Errors

File does not exist

```
$ rm test.txt
```

Output:

```
rm: cannot remove 'test.txt': No such file or directory
```

Permission denied

```
rm protected.txt
```

Output:

```
rm: cannot remove 'protected.txt': Permission denied
```

Use appropriate permissions or `sudo` if required:

```
sudo rm protected.txt
```

Directory without `-r`

```
rm myfolder
```

Output:

```
rm: cannot remove 'myfolder': Is a directory
```

Use:

```
rm -r myfolder
```

Safety Tips

- Use `rm -i` if you're unsure; it asks before deleting.
- Be very careful with wildcards like `*`.
- Double-check the path before using `rm -rf`.
- Avoid running `sudo rm -rf` unless you are absolutely certain of the target.

Summary

Command	Description
`rm file.txt`	Delete a file
`rm file1 file2`	Delete multiple files
`rm -i file.txt`	Delete with confirmation
`rm -f file.txt`	Force delete a file
`rm -d dir`	Delete an empty directory
`rm -r dir`	Delete a directory recursively
`rm -rf dir`	Force delete a directory and its contents
`rm -v file.txt`	Show what is being deleted
`rm *.txt`	Delete all `.txt` files in the current directory

Warning

The `rm` command **permanently deletes** files and directories. There is **no built-in undo**. Always verify the path and options—especially when using `rm -rf`—before pressing Enter.

7. cp

The `cp` (copy) command is used to **copy files and directories** from one location to another. It creates a duplicate of the original without removing or modifying the source.

Syntax

```
cp [OPTIONS] SOURCE DESTINATION
```

or

```
cp [OPTIONS] SOURCE1 SOURCE2 ... DIRECTORY
```

- **SOURCE** – File or directory to copy.
- **DESTINATION** – Target file or directory.

Basic Examples

1. Copy a file

```
cp file1.txt file2.txt
```

Copies `file1.txt` to `file2.txt`.

If `file2.txt` doesn't exist, it is created.

If it exists, it is overwritten (unless prevented).

---

2. Copy a file to another directory

```
cp file1.txt /home/user/Documents/
```

Copies `file1.txt` into the `Documents` directory.

3. Copy multiple files

```
cp file1.txt file2.txt file3.txt /backup/
```

Copies all three files into `/backup/`.

---

4. Copy a directory

```
cp -r Projects Backup/
```

Copies the entire `Projects` directory into `Backup/`.

`-r` means **recursive**.

---

5. Copy while preserving file attributes

```
cp -p file.txt backup.txt
```

Preserves:

- Ownership
- Permissions
- Timestamp

---

6. Interactive copy

```
cp -i file.txt backup.txt
```

Asks before overwriting.

Example:

```
cp: overwrite 'backup.txt'? y
```

---

7. Force overwrite

```
cp -f file.txt backup.txt
```

Forces overwrite without prompting.

8. Verbose mode

```
cp -v file.txt backup.txt
```

Shows what is being copied.

Output:

```
'file.txt' -> 'backup.txt'
```

---

9. Copy only if source is newer

```
cp -u file.txt backup.txt
```

Copies only when:

- Source is newer than destination
- Destination does not exist

Useful for backups.

---

10. Preserve everything

```
cp -a source_dir backup_dir
```

Archive mode preserves:

- Permissions
- Ownership
- Symbolic links
- Timestamps
- Recursive copying

Equivalent to:

```
cp -dR --preserve=all
```

Common Options

Option	Description
`-r`, `-R`	Copy directories recursively
`-a`	Archive mode (preserves almost everything)
`-i`	Prompt before overwrite
`-f`	Force overwrite
`-n`	Never overwrite existing files
`-u`	Copy only newer files
`-v`	Verbose output
`-p`	Preserve file attributes
`-l`	Create hard links instead of copying
`-s`	Create symbolic links instead of copying
`--backup`	Create a backup before overwriting

Practical Examples

Backup a file

```
cp report.doc report.doc.bak
```

Copy an entire website

```
cp -r website/ backup/
```

Copy all `.txt` files

```
cp *.txt /backup/
```

Copy hidden files

```
cp -r source/. destination/
```

The `.` after `source/` ensures hidden files (like `.bashrc`) are included.

Create a backup before overwriting

```
cp --backup file.txt backup/
```

Exit Status

- `0` – Copy completed successfully.
- Non-zero – An error occurred (e.g., source file not found, permission denied, or invalid option).

Common Errors

Source file does not exist

```
cp abc.txt backup/
```

Output:

```
cp: cannot stat 'abc.txt': No such file or directory
```

Copying a directory without `-r`

```
cp mydir backup/
```

Output:

```
cp: -r not specified; omitting directory 'mydir'
```

Use:

```
cp -r mydir backup/
```

Permission denied

```
cp file.txt /root/
```

Output:

```
cp: cannot create regular file '/root/file.txt': Permission denied
```

Use `sudo` if appropriate:

```
sudo cp file.txt /root/
```

Difference Between `cp`, `mv`, and `rsync`

Command	Purpose	Original File
`cp`	Copies files/directories	Remains unchanged
`mv`	Moves or renames files/directories	Removed from original location after move
`rsync`	Synchronizes files/directories efficiently	Remains unless synchronization options remove it

Summary

- `cp` copies files and directories.
- Use `-r` or `-R` to copy directories.
- Use `-a` for backups while preserving attributes.
- Use `-i` to avoid accidental overwrites.
- Use `-v` to display copied files.
- Use `-u` to copy only newer files.
- Use `-p` to preserve permissions, ownership, and timestamps.

8. mv

- The `mv` (move) command in Linux is used to **move files or directories** from one location to another. It is also used to **rename files and directories**.

Syntax

```
mv [OPTIONS] SOURCE DESTINATION
```

or

```
mv [OPTIONS] SOURCE1 SOURCE2 ... DIRECTORY
```

- **SOURCE** – File or directory to move.
- **DESTINATION** – New location or new name.

Common Uses

1. Rename a File

```
mv file1.txt file2.txt
```

Before:

```
file1.txt
```

After:

```
file2.txt
```

---

2. Move a File to Another Directory

```
mv report.txt /home/user/Documents/
```

Moves `report.txt` into the `Documents` directory.

3. Move Multiple Files

```
mv file1.txt file2.txt file3.txt /home/user/Documents/
```

Moves all three files into the `Documents` directory.

---

4. Rename a Directory

```
mv old_project new_project
```

Changes the directory name from `old_project` to `new_project`.

---

5. Move a Directory

```
mv project/ /home/user/Backup/
```

Moves the entire `project` directory into `Backup`.

Common Options

Option	Description
`-i`	Ask before overwriting an existing file.
`-f`	Force overwrite without prompting.
`-n`	Never overwrite an existing file.
`-v`	Show what is being moved (verbose).
`-u`	Move only if the source is newer than the destination or the destination doesn't exist.
`-t DIRECTORY`	Specify the target directory first.

Examples

Interactive Mode (`-i`)

```
mv -i notes.txt /home/user/Documents/
```

If the destination already contains `notes.txt`, you'll see:

```
mv: overwrite '/home/user/Documents/notes.txt'?
```

Verbose Mode (`-v`)

```
mv -v file.txt /tmp/
```

Output:

```
renamed 'file.txt' -> '/tmp/file.txt'
```

Force Overwrite (`-f`)

```
mv -f file.txt /tmp/
```

Overwrites the destination file without asking.

Do Not Overwrite (`-n`)

```
mv -n report.txt /backup/
```

If `report.txt` already exists in `/backup`, the move is skipped.

Update Only (`-u`)

```
mv -u report.txt /backup/
```

Moves the file only if:

- the destination file does not exist, or
- the source file is newer than the destination.

Specify Target Directory (`-t`)

```
mv -t /home/user/Documents file1.txt file2.txt file3.txt
```

Equivalent to:

```
mv file1.txt file2.txt file3.txt /home/user/Documents
```

Important Notes

- `mv` **does not create a copy**. The original file is removed from its old location after being moved.
- If the source and destination are on the **same filesystem**, the move is usually very fast because only directory entries are updated.
- If moving across **different filesystems or disks**, `mv` copies the data to the new location and then deletes the original.

Exit Status

Exit Code	Meaning
`0`	Command completed successfully.
Non-zero	An error occurred (e.g., file not found, permission denied).

Practical Examples

Rename a file:

```
mv photo.jpg vacation.jpg
```

Move a file:

```
mv vacation.jpg ~/Pictures/
```

Move all text files:

```
mv *.txt ~/Documents/
```

Rename a directory:

```
mv ProjectA ProjectB
```

Move a directory:

```
mv ProjectB ~/Backup/
```

Show each move:

```
mv -v *.log /var/log/archive/
```

Summary

Task	Command
Rename a file	`mv old.txt new.txt`
Rename a directory	`mv olddir newdir`
Move a file	`mv file.txt /path/`
Move a directory	`mv dir/ /path/`
Move multiple files	`mv file1 file2 /path/`
Prompt before overwrite	`mv -i file.txt /path/`
Force overwrite	`mv -f file.txt /path/`
Never overwrite	`mv -n file.txt /path/`
Show operations	`mv -v file.txt /path/`
Move only if newer	`mv -u file.txt /path/`

The `mv` command is one of the core Linux file management commands, commonly used alongside:

- `cp` – Copy files and directories.
- `rm` – Remove files and directories.
- `ls` – List files and directories.
- `mkdir` – Create directories.
- `find` – Search for files and directories.

9. `touch`

The `touch` command is used to **create empty files** or **update the timestamps** (access and modification times) of existing files.

Syntax

```
touch [OPTIONS] FILE...
```

- `OPTIONS` – Optional flags to modify the command's behavior.
- `FILE...` – One or more file names.

What `touch` Does

If the file **does not exist**:

- Creates a new empty file.

If the file **already exists**:

- Updates its **access time (atime)** and **modification time (mtime)** to the current date and time.

Common Examples

1. Create an empty file

```
touch file.txt
```

Creates `file.txt` if it doesn't already exist.

---

2. Create multiple files

```
touch file1.txt file2.txt file3.txt
```

Creates all three files.

3. Update the timestamp of an existing file

```
touch file.txt
```

The file contents remain unchanged; only its timestamps are updated.

---

4. Create files with different extensions

```
touch notes.txt script.sh data.csv
```

Creates:

```
notes.txt
script.sh
data.csv
```

---

5. Create files inside another directory

```
touch /home/user/report.txt
```

Creates `report.txt` in `/home/user`.

Common Options

`-a` — Change only the access time

```
touch -a file.txt
```

Updates only the **access time (atime)**.

`-m` — Change only the modification time

```
touch -m file.txt
```

Updates only the **modification time (mtime)**.

`-c` or `--no-create` — Do not create the file

```
touch -c file.txt
```

- If the file exists, its timestamp is updated.
- If it does not exist, nothing is created.

`-t` — Specify a timestamp

Format:

```
[[CC]YY]MMDDhhmm[.ss]
```

Example:

```
touch -t 202607231530 file.txt
```

Sets the timestamp to:

```
23 July 2026 15:30
```

`-d` — Specify a date in human-readable format

```
touch -d "2026-07-23 15:30:00" file.txt
```

or

```
touch -d "yesterday" file.txt
```

or

```
touch -d "2 days ago" file.txt
```

`-r` — Copy timestamps from another file

```
touch -r source.txt destination.txt
```

Copies the timestamps from `source.txt` to `destination.txt`.

Viewing Timestamps

Use `ls -l` to see the modification time:

```
ls -l file.txt
```

For detailed timestamps:

```
stat file.txt
```

Example output:

```
Access: 2026-07-23 15:40:12
Modify: 2026-07-23 15:40:12
Change: 2026-07-23 15:40:12
```

Practical Uses

Create placeholder files

```
touch index.html style.css script.js
```

Create a log file

```
touch app.log
```

Update a file's modification time

```
touch Makefile
```

This can trigger build tools (such as `make`) to rebuild targets that depend on the file.

Create files in a loop

```
for i in {1..5}
do
    touch file$i.txt
done
```

Creates:

```
file1.txt
file2.txt
file3.txt
file4.txt
file5.txt
```

Exit Status

- `0` – Success.
- Non-zero – An error occurred (for example, permission denied or invalid file path).

Summary

Command	Description
`touch file.txt`	Create an empty file or update timestamps
`touch file1 file2`	Create multiple files
`touch -a file.txt`	Update only the access time
`touch -m file.txt`	Update only the modification time
`touch -c file.txt`	Update timestamps without creating the file
`touch -t 202607231530 file.txt`	Set a specific timestamp
`touch -d "yesterday" file.txt`	Set the timestamp using a readable date
`touch -r source.txt dest.txt`	Copy timestamps from another file
`stat file.txt`	Display detailed file timestamps

In short: `touch` is a simple but essential command for creating empty files and managing file timestamps, making it useful for scripting, development, testing, and system administration.

10. find

The `find` command in Linux is used to **search for files and directories** within a directory hierarchy based on different criteria such as name, type, size, permissions, owner, and modification time.

Syntax

```
find [starting_directory] [options] [expression]
```

- **starting_directory** – Directory where the search begins.
- **options** – Control how `find` behaves (optional).
- **expression** – Conditions used to match files and actions to perform.

Basic Examples

1. Find a file by name

```
find /home -name "notes.txt"
```

Searches for `notes.txt` under `/home`.

---

2. Find a directory

```
find /home -type d -name "Documents"
```

- `-type d` → Search only directories.

---

3. Find only files

```
find /home -type f
```

- `-type f` → Search only regular files.

4. Find files regardless of case

```
find /home -iname "report.txt"
```

Matches:

- Report.txt
- REPORT.TXT
- report.txt

Common Search Options

Option	Description
`-name`	Search by filename (case-sensitive)
`-iname`	Search by filename (case-insensitive)
`-type f`	Regular files
`-type d`	Directories
`-type l`	Symbolic links
`-empty`	Empty files/directories
`-size`	Search by file size
`-user`	Search by owner
`-group`	Search by group
`-perm`	Search by permissions
`-mtime`	Modified days ago
`-atime`	Accessed days ago
`-ctime`	Status changed days ago

Search by Extension

```
find . -name "*.txt"
```

Finds all `.txt` files in the current directory and its subdirectories.

Search by Size

Larger than 100 MB

```
find /home -size +100M
```

Smaller than 10 MB

```
find /home -size -10M
```

Exactly 50 MB

```
find /home -size 50M
```

Units:

- `c` → bytes
- `k` → KB
- `M` → MB
- `G` → GB

Search by Owner

```
find /home -user bharath
```

Finds files owned by user `bharath`.

Search by Permissions

Find files with permission `644`:

```
find /home -perm 644
```

Find files with permission `755`:

```
find /home -perm 755
```

Search by Modification Time

Modified within the last day:

```
find /home -mtime -1
```

Modified exactly 7 days ago:

```
find /home -mtime 7
```

Modified more than 30 days ago:

```
find /home -mtime +30
```

Find Empty Files

```
find /home -type f -empty
```

Find Empty Directories

```
find /home -type d -empty
```

Delete Files Found

Delete all `.tmp` files:

```
find . -name "*.tmp" -delete
```

Be careful: `-delete` permanently removes files.

Execute a Command on Each Result

Delete `.log` files using `rm`:

```
find . -name "*.log" -exec rm {} \;
```

Explanation:

- `{}` → Replaced with each matching file.
- `\;` → Ends the `-exec` command.

Find and Change Permissions

```
find /project -type f -exec chmod 644 {} \;
```

Changes all files under `/project` to permission `644`.

Find and Change Owner

```
find /project -type f -exec chown user:user {} \;
```

Find Broken Symbolic Links

```
find /home -xtype l
```

Exclude a Directory

```
find /home -path "/home/cache" -prune -o -name "*.txt" -print
```

Searches for `.txt` files while skipping `/home/cache`.

Search Multiple Conditions

Find `.txt` files larger than 10 MB:

```
find . -name "*.txt" -size +10M
```

Find files owned by `root` and larger than 100 MB:

```
find / -user root -size +100M
```

Common Actions

Action	Description
`-print`	Print matching files (default)
`-delete`	Delete matching files
`-exec command {} \;`	Run a command for each match
`-ls`	Show detailed listing
`-quit`	Stop after the first match

Example:

```
find . -name "*.sh" -ls
```

Difference Between `find` and `locate`

`find`	`locate`
Searches the filesystem in real time	Searches a prebuilt database
Slower	Much faster
Can search by size, owner, permissions, dates, etc.	Mainly searches by filename
Always up to date	Database may need updating (`updatedb`)

Practical Examples

```
# Find all PDF files
find ~/Documents -name "*.pdf"

# Find files larger than 1 GB
find / -size +1G

# Find directories named backup
find / -type d -name "backup"

# Find files modified in the last 24 hours
find /var/log -mtime -1

# Delete all .bak files
find . -name "*.bak" -delete

# Find executable shell scripts
find . -name "*.sh" -perm 755

# Find all symbolic links
find / -type l

# Find empty directories
find /tmp -type d -empty
```

Summary

The `find` command is one of the most powerful Linux file management tools. It can:

- Search by **name**, **type**, **size**, **owner**, **permissions**, and **timestamps**.
- Combine multiple search conditions.
- Execute commands (`-exec`) on matching files.
- Delete or modify matching files.
- Search recursively through directory trees.

It is widely used by Linux administrators and developers for system administration, troubleshooting, automation, and file management.

11. `locate`

The **`locate`** command is used to **quickly find files and directories by name**. Unlike the `find` command, `locate` searches a **pre-built database** of file paths instead of scanning the filesystem each time, making it much faster.

Syntax

```
locate [OPTIONS] PATTERN
```

- **PATTERN** – The filename or part of the filename you want to search for.

How `locate` Works

1. Linux maintains a database of file names and paths.
2. The `updatedb` command periodically updates this database.
3. When you run `locate`, it searches the database instead of the disk.

Because it uses a database:

- Very fast
- May not show recently created files until the database is updated

Examples

1. Find a file by name

```
locate notes.txt
```

Example output:

```
/home/bharath/Documents/notes.txt
/backup/notes.txt
```

---

2. Find all PDF files containing "linux"

```
locate linux | grep "\.pdf$"
```

Output:

```
/home/user/Documents/linux_guide.pdf
/home/user/Downloads/linux_notes.pdf
```

---

3. Case-insensitive search

```
locate -i README
```

Matches:

```
README
Readme
readme
README.md
```

---

4. Show only the first 5 results

```
locate bash | head -5
```

5. Count matching files

```
locate -c python
```

Output:

```
245
```

---

6. Show only existing files

Sometimes the database contains files that have been deleted.

```
locate -e notes.txt
```

The `-e` option displays only files that currently exist.

---

7. Limit the number of results

```
locate -l 10 log
```

Shows only the first 10 matches.

Common Options

Option	Description
`-i`	Ignore case while searching
`-c`	Count matching entries
`-e`	Show only existing files
`-l N`	Limit output to N results
`-r REGEX`	Search using a regular expression
`-b`	Match only the basename (filename, not full path)
`-0`	Separate output with a null character (useful in scripts)

Updating the Database

If a file was created recently and `locate` cannot find it, update the database:

```
sudo updatedb
```

After updating:

```
locate myfile.txt
```

Difference Between `locate` and `find`

Feature	`locate`	`find`
Search method	Uses a database	Scans the filesystem
Speed	Very fast	Slower
Finds newly created files	Only after `updatedb`	Yes
Search by permissions	No	Yes
Search by size	No	Yes
Search by owner	No	Yes
Search by modification time	No	Yes

Exit Status

- **0** – Command executed successfully (whether or not matches were found).
- **1** – An error occurred (such as invalid options or database issues).

Practical Examples

Find all JPEG images:

```
locate .jpg
```

Find all `.conf` files:

```
locate "*.conf"
```

Note: Quotes prevent the shell from expanding the wildcard. `locate` treats the pattern itself as the search term.

Find all files related to Apache:

```
locate apache
```

Find all files named exactly `passwd`:

```
locate -b '\passwd'
```

Advantages

- Extremely fast searches.
- Simple to use.
- Ideal for finding files by name.
- Uses very little CPU during searches.

Limitations

- Results may be outdated until the database is refreshed.
- Cannot search by attributes such as size, permissions, owner, or modification time.
- Depends on the `updatedb` database being available.

Summary

- **Purpose:** Quickly locate files and directories by name.
- **Database-based:** Uses a pre-built index, making searches much faster than `find`.
- **Best for:** Fast filename searches.
- **Limitation:** May not include recently created or deleted files until the database is updated with `updatedb`.

12. `tree`

The `tree` command displays the contents of a directory in a **tree-like hierarchical format**. It makes it easy to visualize the directory structure, including subdirectories and files.

Syntax

```
tree [OPTIONS] [DIRECTORY]
```

- **OPTIONS** – Modify the output.
- **DIRECTORY** – The directory to display. If omitted, the current directory is used.

Basic Example

Display the current directory

```
tree
```

Example Output

```
.
├── Documents
│   ├── report.pdf
│   └── notes.txt
├── Downloads
│   └── ubuntu.iso
├── Pictures
│   ├── image1.jpg
│   └── image2.png
└── script.sh

3 directories, 5 files
```

Common Options

1. Display a specific directory

```
tree /home/bharath
```

Example:

```
/home/bharath
├── Desktop
├── Documents
├── Downloads
└── Pictures
```

---

2. Show only directories (`-d`)

```
tree -d
```

Output:

```
.
├── Documents
├── Downloads
└── Pictures
```

---

3. Limit the depth (`-L`)

Show only the first two levels.

```
tree -L 2
```

Example:

```
.
├── Documents
│   ├── report.pdf
│   └── notes.txt
├── Downloads
└── Pictures
```

4. Show hidden files (`-a`)

```
tree -a
```

Output includes files beginning with `.`

```
.
├── .bashrc
├── .profile
├── Documents
└── script.sh
```

---

5. Show file permissions (`-p`)

```
tree -p
```

Example:

```
.

├── [-rw-r--r--] notes.txt
├── [-rwxr-xr-x] script.sh
└── [drwxr-xr-x] Documents
```

---

6. Show file sizes (`-s`)

```
tree -s
```

Example:

```
.
├── [4.0K] notes.txt
├── [2.3M] movie.mp4
└── [8.0K] script.sh
```

---

7. Show file owner (`-u`)

```
tree -u
```

8. Show group owner (`-g`)

```
tree -g
```

---

9. Display full path (`-f`)

```
tree -f
```

Example:

```
.
├── ./Documents
├── ./Documents/report.pdf
└── ./script.sh
```

---

10. Save output to a file

```
tree > directory_structure.txt
```

---

11. Ignore a directory (`-I`)

Ignore one or more directories/files.

```
tree -I "node_modules"
```

Ignore multiple names:

```
tree -I "node_modules|.git"
```

---

12. Show only files matching a pattern (`-P`)

```
tree -P "*.txt"
```

Shows only `.txt` files.

13. Show inode numbers (`-i`)

```
tree -i
```

Removes indentation lines, making the output easier to parse by scripts.

Installing `tree`

Ubuntu/Debian

```
sudo apt install tree
```

RHEL/CentOS/Rocky Linux

```
sudo dnf install tree
```

or

```
sudo yum install tree
```

Real-World Examples

View your home directory

```
tree ~
```

Display only directories up to three levels

```
tree -d -L 3
```

Show all files including hidden ones

```
tree -a
```

Display only Python files

```
tree -P "*.py"
```

Export the directory structure

```
tree /var/www > website_structure.txt
```

Advantages

- Easy to visualize directory hierarchies.
- Useful for documentation.
- Helps inspect project layouts.
- Can display permissions, ownership, and sizes.
- Can export directory structures to a text file.

Summary Table

Command	Description
`tree`	Display current directory as a tree
`tree /path`	Display a specific directory
`tree -d`	Show only directories
`tree -L 2`	Limit display depth to 2 levels
`tree -a`	Include hidden files
`tree -p`	Show file permissions
`tree -s`	Show file sizes
`tree -u`	Show file owners
`tree -g`	Show group owners
`tree -f`	Show full paths
`tree -I "pattern"`	Exclude files/directories matching a pattern
`tree -P "*.txt"`	Show only files matching a pattern
`tree > output.txt`	Save the tree output to a file

Exit Status

- **0** – Command executed successfully.
- **1** – An error occurred (such as an invalid option or inaccessible directory).

The `tree` command is particularly useful for exploring and documenting directory structures, especially in software projects, system administration, and file organization tasks.