Table of Contents

$ cd

Beginner

Change the current working directory

The cd (change directory) command is a shell built-in that changes the current working directory. It is the most fundamental navigation command in Linux, used constantly in daily operations.

cd supports absolute paths (starting from /), relative paths (from current location), and several shortcu...

Options & Flags

`-`	Switch to the previous directory ($OLDPWD)
`~`	Navigate to the home directory
`..`	Move up one directory level
`-P`	Use physical directory structure (resolve symlinks)
`-L`	Follow symbolic links (default behavior)
`~username`	Navigate to another user's home directory

Practical Examples

Example: Navigate to an absolute path

```
$ cd /var/log/nginx
```

Changes to the Nginx log directory using an absolute path.

Example: Move up two directory levels

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

Moves up two levels in the directory hierarchy.

Example: Return to home directory

```
$ cd ~
```

Shortcut to return to your home directory. Just typing cd without arguments does the same.

Example: Switch between two directories

```
$ cd -
/home/user/projects
```

Toggles between the current and previous directory. Shows the new path.

Example: Navigate using environment variable

```
$ cd $HOME/Documents
```

Uses the HOME environment variable to build the path.

Tips & Best Practices

Pro Tip: Use tab completion: Press Tab while typing a path to auto-complete directory names. Double-tap Tab to see all matching options.

Note: CDPATH variable: Set CDPATH to frequently used parent directories: export CDPATH=.:~:/var/www - then cd mysite works from anywhere if /var/www/mysite exists.

Pro Tip: pushd/popd for complex navigation: Use pushd/popd instead of cd to maintain a directory stack. pushd /var/log saves your current location, and popd returns to it.

$ cp

Beginner

Copy files and directories

The cp command copies files and directories from one location to another. It is one of the fundamental file management commands in Linux, used daily by system administrators and developers.

cp can copy single files, multiple files, or entire directory trees. It supports preserving file attribute...

Options & Flags

`-r, -R`	Copy directories recursively (required for directories)
`-i`	Interactive mode - prompt before overwriting
`-v`	Verbose - show files being copied
`-p`	Preserve permissions, ownership, and timestamps
`-a`	Archive mode - preserve all attributes and copy recursively
`-u`	Copy only when source is newer than destination
`-n`	Do not overwrite existing files
`-l`	Create hard links instead of copying
`-s`	Create symbolic links instead of copying

Practical Examples

Example: Copy a single file

```
$ cp report.pdf /home/user/Documents/
```

Copies report.pdf to the Documents directory.

Example: Copy and rename a file

```
$ cp config.yml config.yml.bak
```

Creates a backup copy with a different name in the same directory.

Example: Copy a directory recursively

```
$ cp -r /var/www/html /backup/www-backup
```

Copies the entire directory tree including all subdirectories and files.

Example: Copy preserving all attributes

```
$ cp -a /etc/nginx/ /backup/nginx-config/
```

Archive mode preserves permissions, ownership, timestamps, and symlinks.

Example: Copy multiple files to a directory

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

Copies multiple named files into the /backup/ directory.

Tips & Best Practices

Warning: Overwrite danger: cp overwrites destination files without warning by default. Always use -i for important files, or add "alias cp='cp -i'" to .bashrc.

Pro Tip: Backup before copy: Use --backup=numbered to automatically create numbered backups: cp --backup=numbered file.txt /dest/

Note: cp vs rsync: For large directory copies or network transfers, rsync is more efficient as it only transfers changed data.

$ ln

Intermediate

Create hard and symbolic links between files

The ln command creates links between files in Linux. There are two types: hard links and symbolic (soft) links. Hard links create an additional directory entry pointing to the same inode, while symbolic links are special files that point to the target path.

Hard links share the same data blocks ...

Options & Flags

`-s`	Create a symbolic (soft) link instead of a hard link
`-f`	Force - remove existing destination files
`-i`	Interactive - prompt before removing existing files
`-n`	Do not dereference symlink that is a directory
`-v`	Verbose - print name of each linked file
`-r`	Create relative symbolic links
`-b`	Make a backup of each existing destination file
`-t`	Specify target directory

Practical Examples

Example: Create a symbolic link

```
$ ln -s /var/www/html/site /home/user/site
```

Creates a symbolic link in your home directory pointing to the web root.

Example: Create a hard link

```
$ ln data.csv data_backup.csv
```

Creates a hard link - both names point to the same file data on disk.

Example: Update an existing symlink

```
$ ln -sfn /var/www/releases/v2.1 /var/www/current
```

Atomically updates a symlink to point to a new release directory. Common in deployment workflows.

Example: Link a config file

```
$ ln -s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/
```

Standard Nginx pattern for enabling sites using symlinks.

Example: Create relative symlink

```
$ ln -sr ../shared/.env .env
```

Creates a relative symlink, which works better when moving directory trees.

Tips & Best Practices

Warning: Broken symlinks: If the target of a symlink is moved or deleted, the link becomes broken. Use find -L /path -type l to find broken symlinks.

Pro Tip: Deployment with symlinks: Use ln -sfn for zero-downtime deployments: keep releases in versioned directories and symlink "current" to the active release.

Note: Hard link limitations: Hard links cannot cross filesystem boundaries, cannot link to directories, and all links share the same inode number (verify with ls -i).

$ ls

Beginner

List directory contents and file information

The ls command is one of the most frequently used commands in Linux. It lists the contents of a directory, showing files, subdirectories, and their attributes. By default, ls shows the names of files and directories in the current working directory.

When used with various options, ls can display...

Options & Flags

`-l`	Use long listing format showing permissions, owner, size,...
`-a`	Show all files including hidden files (starting with .)
`-h`	Human-readable file sizes (KB, MB, GB)
`-R`	List subdirectories recursively
`-t`	Sort by modification time, newest first
`-S`	Sort by file size, largest first
`-r`	Reverse the order of sort
`-d`	List directories themselves, not their contents
`-i`	Print the inode number of each file
`--color`	Colorize the output (auto, always, never)

Practical Examples

Example: List all files with details

```
$ ls -la /home/user
total 48
drwxr-xr-x 6 user user 4096 Mar 14 10:30 .
drwxr-xr-x 3 root root 4096 Jan 15 09:00 ..
-rw-r--r-- 1 user user  220 Jan 15 09:00 .bash_logout
```

Shows all files including hidden ones, with permissions, owner, group, size, and modification date.

Example: List files sorted by size

```
$ ls -lhS /var/log
-rw-r----- 1 syslog adm 15M Mar 14 12:00 syslog
-rw-r----- 1 syslog adm 5.2M Mar 14 12:00 auth.log
-rw-r----- 1 syslog adm 2.1M Mar 14 12:00 kern.log
```

Shows files in /var/log sorted by size (largest first) with human-readable sizes.

Example: List only directories

```
$ ls -d */
Documents/  Downloads/  Music/  Pictures/  Videos/
```

Shows only directories in the current location, not their contents.

Example: List files modified today

```
$ ls -lt --time=mtime | head -20
total 128
-rw-r--r-- 1 user user 4096 Mar 14 12:30 report.pdf
-rw-r--r-- 1 user user 2048 Mar 14 11:15 notes.txt
```

Shows the 20 most recently modified files, newest first.

Example: List files with inode numbers

```
$ ls -li
262144 -rw-r--r-- 2 user user 1024 Mar 14 10:00 file1.txt
262144 -rw-r--r-- 2 user user 1024 Mar 14 10:00 file1_link.txt
```

Shows inode numbers, useful for identifying hard links.

Tips & Best Practices

Pro Tip: Alias for convenience: Add "alias ll='ls -lah'" to your .bashrc for a quick detailed listing command.

Note: Color output: Most modern distros have ls aliased to "ls --color=auto". Blue = directory, green = executable, cyan = symlink, red = archive.

Warning: Large directories: For directories with thousands of files, ls can be slow. Use "ls -f" to skip sorting, or "find" for better performance.

Pro Tip: Hidden files only: Use "ls -d .*" to list ONLY hidden files without showing all files.

$ mkdir

Beginner

Create new directories

The mkdir command creates new directories (folders) in the Linux filesystem. It is one of the core file management commands used constantly in daily work.

mkdir can create single directories, multiple directories at once, and entire directory trees in one command using the -p flag. It supports s...

Options & Flags

`-p`	Create parent directories as needed (no error if existing)
`-m`	Set permission mode during creation
`-v`	Verbose - print each directory as it is created
`-Z`	Set SELinux security context
`--context`	Like -Z but for a specified SELinux context
`multiple`	Create multiple directories in one command

Practical Examples

Example: Create a simple directory

```
$ mkdir projects
```

Creates a single directory called projects in the current location.

Example: Create nested directory tree

```
$ mkdir -p /var/www/myapp/public/assets/{css,js,images}
```

Creates the entire directory structure including all parent directories. Brace expansion creates css, js, and images subdirectories.

Example: Create with specific permissions

```
$ mkdir -m 700 ~/.ssh
```

Creates the .ssh directory with permissions allowing only the owner to read, write, and execute.

Example: Create project scaffold

```
$ mkdir -pv myproject/{src/{controllers,models,views},tests,config,public/{css,js}}
mkdir: created directory 'myproject'
mkdir: created directory 'myproject/src'
mkdir: created directory 'myproject/src/controllers'
```

Creates a complete project directory structure in one command with verbose output.

Example: Create directory for each date

```
$ mkdir -p /backup/$(date +%Y/%m/%d)
```

Creates a date-organized backup directory like /backup/2025/03/14.

Tips & Best Practices

Pro Tip: Brace expansion for scaffolding: Use bash brace expansion with -p to create complex directory trees: mkdir -p app/{frontend/{components,pages,styles},backend/{routes,middleware,models}}

Warning: Check umask before creating shared directories: The default umask (usually 022) affects permissions. For shared directories, explicitly set permissions with -m to avoid access issues.

Note: install command alternative: The install -d command combines mkdir and chmod in one step: install -d -m 755 -o www-data -g www-data /var/www/site

$ mktemp

Beginner

Create temporary files and directories securely

The mktemp command creates temporary files or directories with unique, unpredictable names. It is the secure way to handle temporary files in shell scripts, eliminating race conditions and symlink attacks that plague naive approaches like using fixed filenames in /tmp.

mktemp generates a random ...

Options & Flags

`(no options)`	Create a temporary file in /tmp
`-d`	Create a temporary directory instead of a file
`-p DIR`	Create temp file in specified directory
`-t NAME`	Use NAME as template prefix
`--suffix=SUFF`	Append suffix to template
`-u`	Unsafe mode - print name without creating (not recommended)
`TEMPLATE`	Custom template with X placeholders

Practical Examples

Example: Create temporary file

```
$ tmpfile=$(mktemp) && echo "Data" > "$tmpfile" && cat "$tmpfile" && rm "$tmpfile"
Data
```

Create a temp file, write to it, read it, and clean up. The file has 0600 permissions.

Example: Create temporary directory

```
$ tmpdir=$(mktemp -d) && echo "Dir: $tmpdir" && ls -la "$tmpdir" && rm -rf "$tmpdir"
```

Create a temp directory with 0700 permissions. Use for multiple temp files or complex operations.

Example: Script with automatic cleanup

```
$ tmpfile=$(mktemp) && trap "rm -f $tmpfile" EXIT && curl -s https://example.com > "$tmpfile" && wc -l "$tmpfile"
```

Use trap EXIT to automatically clean up temp files when the script exits (normally or on error).

Example: Custom template with suffix

```
$ mktemp --suffix=.csv /tmp/report.XXXXXX
/tmp/report.a3bK9x.csv
```

Create a temp file with .csv extension and custom prefix. Useful when tools require specific file extensions.

Example: Temp file in specific directory

```
$ mktemp -p /var/tmp backup.XXXXXXXX
/var/tmp/backup.4f8Gk2x1
```

Create temp file in /var/tmp (survives reboots on some systems) instead of /tmp.

Tips & Best Practices

Pro Tip: Always use trap for cleanup: In scripts, use: trap "rm -f $tmpfile" EXIT - this ensures temp files are removed even if the script fails or is interrupted with Ctrl+C.

Warning: Never use predictable temp names: Never use /tmp/myapp.$$ or /tmp/data.tmp. These are predictable and vulnerable to symlink attacks. Always use mktemp.

Note: XXXXXX minimum: Templates must have at least 3 consecutive X characters. More Xs mean more randomness. The default template uses 10 random characters.

Pro Tip: Use $TMPDIR: mktemp respects the $TMPDIR environment variable. Set it to use a different base directory for all temp files.

$ mv

Beginner

Move or rename files and directories

The mv command moves files and directories from one location to another, or renames them. Unlike cp, mv does not create a copy - it relocates the original file.

When moving files within the same filesystem, mv simply updates the directory entry, making it nearly instantaneous regardless of file ...

Options & Flags

`-i`	Interactive - prompt before overwriting
`-f`	Force - do not prompt before overwriting
`-n`	Do not overwrite existing files
`-v`	Verbose - explain what is being done
`-u`	Move only when source is newer than destination
`--backup`	Make a backup of existing destination files

Practical Examples

Example: Rename a file

```
$ mv oldname.txt newname.txt
```

Renames the file in the current directory.

Example: Move file to another directory

```
$ mv report.pdf /home/user/Documents/
```

Moves the file to Documents, removing it from the current location.

Example: Move multiple files

```
$ mv *.jpg *.png /home/user/Pictures/
```

Moves all JPEG and PNG files to the Pictures directory.

Example: Rename a directory

```
$ mv old-project/ new-project/
```

Renames a directory. No -r flag needed unlike cp.

Example: Move with verbose output

```
$ mv -v /tmp/downloads/* /home/user/Downloads/
renamed '/tmp/downloads/file1.zip' -> '/home/user/Downloads/file1.zip'
renamed '/tmp/downloads/file2.tar.gz' -> '/home/user/Downloads/file2.tar.gz'
```

Shows each file being moved.

Tips & Best Practices

Warning: No undo: mv has no undo operation. If you overwrite a file, it is gone. Always use -i for important files.

Pro Tip: Batch rename: For batch renaming, use the "rename" command (perl-based): rename "s/old/new/" *.txt

Note: Cross-filesystem moves: Moving files across filesystems (e.g., from /home to /tmp on different partitions) is slower because mv must copy then delete.

$ rclone

Intermediate

Sync files to and from cloud storage

The rclone command is a powerful cloud storage synchronization tool - often called "the Swiss Army knife of cloud storage" or "rsync for cloud". It supports over 70 cloud storage providers including AWS S3, Google Drive, Dropbox, Azure Blob Storage, Backblaze B2, Wasabi, MinIO, SFTP servers, and ...

Options & Flags

`config`	Interactive configuration wizard for adding remotes
`copy`	Copy files from source to destination
`sync`	Sync source to destination (delete extras at dest)
`move`	Move files from source to destination
`mount`	Mount remote as local filesystem (FUSE)
`ls/lsd/lsf`	List files, directories, or formatted listing
`--dry-run`	Preview what would be transferred
`--progress`	Show real-time transfer progress
`--bwlimit`	Limit bandwidth usage
`serve`	Serve remote over HTTP/WebDAV/FTP/SFTP

Practical Examples

Example: Configure a remote

```
$ rclone config
```

Interactive wizard to set up a new cloud storage remote. Supports OAuth flow for Google Drive, Dropbox, etc.

Example: Backup to S3

```
$ rclone sync /var/backups s3remote:mybucket/backups --progress
```

Sync local backups to an S3 bucket. Files deleted locally are deleted from S3 too (mirror sync).

Example: Copy without deleting

```
$ rclone copy /data remote:bucket/data -P --transfers 8
```

Copy new/changed files to remote. Does NOT delete files at destination. Uses 8 parallel transfers.

Example: Encrypted backup

```
$ rclone copy /sensitive encrypted-remote:backup/
```

Copy to an encrypted remote (configured with rclone config using crypt type). Files encrypted at rest.

Example: Mount cloud storage

```
$ rclone mount gdrive: /mnt/gdrive --vfs-cache-mode full --daemon
```

Mount Google Drive as a local filesystem. vfs-cache-mode full enables read-write caching for best performance.

Tips & Best Practices

Warning: sync deletes files at destination: rclone sync makes the destination match source exactly - including deleting files. Use rclone copy if you only want to add new files without deleting.

Pro Tip: Use --dry-run first: Always preview with --dry-run before sync operations: rclone sync /data remote:backup --dry-run. Shows exactly what would be transferred or deleted.

Note: Filter files: Use --include/--exclude patterns: rclone copy /data remote:backup --include "*.sql" --exclude "*.tmp". Or use --filter-from file for complex rules.

Pro Tip: Bandwidth scheduling: Limit bandwidth during work hours: --bwlimit "08:00,1M 18:00,off" - 1 MB/s during 8-18h, unlimited at night.

$ realpath

Beginner

Resolve absolute file path and symlinks

The realpath command resolves the absolute path of a file or directory, resolving all symbolic links, . (dot) and .. (dot-dot) references along the way. It outputs the canonical, fully-resolved path - the true location on disk.

realpath is essential in shell scripts where you need absolute paths...

Options & Flags

`FILE`	Resolve the canonical absolute path
`-e`	All components must exist (error if not)
`-m`	No component needs to exist
`-s`	Do not resolve symlinks
`--relative-to=DIR`	Output path relative to DIR
`--relative-base=DIR`	Print relative if under DIR, absolute otherwise

Practical Examples

Example: Get absolute path

```
$ realpath ./script.sh
/home/user/projects/myapp/script.sh
```

Convert a relative path to an absolute path. Essential in scripts that may be called from any directory.

Example: Resolve symlinks

```
$ realpath /usr/bin/python3
/usr/bin/python3.11
```

Follow symlinks to find the actual binary location.

Example: Script: get own directory

```
$ SCRIPT_DIR=$(dirname "$(realpath "$0")") && echo "Script is at: $SCRIPT_DIR"
```

Find the real directory of the running script, even if invoked through a symlink.

Example: Compute relative path

```
$ realpath --relative-to=/home/user /home/user/projects/app/src
projects/app/src
```

Calculate the relative path from one directory to another.

Example: Verify path exists

```
$ realpath -e /etc/nginx/nginx.conf && echo "File exists" || echo "File missing"
```

Use -e to verify all path components exist. Returns error code 1 if file does not exist.

Tips & Best Practices

Pro Tip: Use in scripts for reliability: Always use realpath when your script needs to reference files relative to itself: BASEDIR=$(dirname "$(realpath "$0")"). This works regardless of how the script is invoked.

Note: realpath vs readlink -f: realpath FILE is equivalent to readlink -f FILE on most systems. realpath has more options (--relative-to, -m) and is the recommended modern tool.

Pro Tip: Compare file paths: Two different paths may point to the same file. Use realpath to canonicalize both and compare: ["$(realpath a)" = "$(realpath b)"] && echo "Same file"

$ rename

Intermediate

Bulk rename files using patterns

The rename command renames multiple files at once using pattern matching - either Perl regular expressions (Perl rename) or simple string substitution (util-linux rename). It is far more powerful than manual mv commands when you need to rename dozens or hundreds of files following a pattern.

The...

Options & Flags

`'s/old/new/'`	Perl rename: substitute old with new using regex
`-n`	Dry run - show what would be renamed without doing it
`-v`	Verbose - show each rename operation
`-f`	Force - overwrite existing files
`'s/old/new/g'`	Global - replace all occurrences (not just first)
`'s/PATTERN/\L$&/'`	Convert matching text to lowercase
`'s/PATTERN/\U$&/'`	Convert matching text to uppercase
`OLD NEW FILES`	Util-linux rename: simple string substitution

Practical Examples

Example: Change file extension

```
$ rename 's/\.txt$/.md/' *.txt
```

Rename all .txt files to .md. The $ anchor ensures only the extension is matched.

Example: Replace spaces with underscores

```
$ rename 's/ /_/g' *.pdf
```

Replace all spaces in filenames with underscores. The g flag replaces all occurrences.

Example: Add prefix to all files

```
$ rename 's/^/2026-/' *.log
```

Add "2026-" prefix to all .log files. ^ matches the beginning of the filename.

Example: Convert filenames to lowercase

```
$ rename 'y/A-Z/a-z/' *
```

Convert all filenames to lowercase using the transliteration operator.

Example: Remove part of filename

```
$ rename 's/_backup//' *.sql
```

Remove "_backup" from all .sql filenames.

Tips & Best Practices

Warning: Always use -n first: Test with rename -n (dry run) before executing. Complex regex patterns can produce unexpected results. Verify the preview before running without -n.

Note: Two different rename commands: Debian/Ubuntu has Perl rename (regex). RHEL/Fedora has util-linux rename (string substitution). Install Perl version with: apt install rename or dnf install prename.

Pro Tip: Use with find for recursive rename: Rename files in subdirectories: find . -name '*.txt' -exec rename 's/\.txt$/.md/' {} +

Pro Tip: Backup before bulk rename: For irreversible renames, create a backup first: cp -r dir/ dir.bak/ - or save the rename mapping: rename -v ... > rename.log