Git

Version 2.45 · verified 2026-07-25
A practical guide to Git for everyday development — branching, merging, and undoing mistakes.

Contents

Mental model

Git does not store diffs; it stores snapshots. Every commit is a full tree of your project at that moment. Branches are not distinct copies of files, they are just lightweight, movable pointers to a specific commit. When you merge, Git is simply combining the histories of two pointers. Understanding this makes “detached HEAD” and rebasing make sense.

Repository setup

Task	Command
Initialize	`git init`
Clone repo	`git clone <url>`
Add remote	`git remote add origin <url>`

Before you start tracking files, initialize a repository or clone an existing one. Cloning automatically sets up a remote called `origin`.

```
git init
git add .
git commit -m "Initial commit"
```

> **Gotcha:** Cloning into an existing non-empty directory is not allowed by default.

Staging & committing

Task	Command
Stage file	`git add <file>`
Stage all	`git add .`
Commit	`git commit -m "msg"`

Git has a two-step commit process. You first move changes from your working directory to the staging area (index), and then commit the staged changes to the repository history.

```
git add index.js
git commit -m "Update index"
```

> **Tip:** Use `git commit -a -m "msg"` to stage and commit all modified (but not new) files in one step.

Branching

Task	Command
List branches	`git branch`
Create & switch	`git checkout -b <name>`
Switch branch	`git switch <name>`

Branches allow you to work on features isolated from the main codebase. Creating a branch is nearly instantaneous because it just creates a new pointer.

```
git checkout -b feature/auth
git switch main
```

> **Note:** `git switch` is the newer, preferred command for changing branches over `git checkout`.

Merging

Task	Command
Merge branch	`git merge <branch>`
Abort merge	`git merge --abort`

To combine changes from one branch into another, switch to the destination branch and merge the source branch. Git will try to automatically resolve changes, but you may need to manually fix conflicts.

```
git switch main
git merge feature/auth
```

> **Warning:** Always ensure your working directory is clean before starting a merge to prevent losing uncommitted work.

Undoing changes

Task	Command
Unstage file	`git restore --staged <f>`
Discard changes	`git restore <file>`
Amend commit	`git commit --amend`

Git provides several ways to undo mistakes depending on where they occurred. You can easily discard uncommitted changes or add forgotten files to the very last commit.

```
git add forgotten_file.txt
git commit --amend --no-edit
```

> **Gotcha:** Never amend commits that have already been pushed to a shared remote branch.

Syncing with remotes

Task	Command
Fetch changes	`git fetch`
Pull & merge	`git pull`
Push changes	`git push origin <b-name>`

To share your work or get updates from others, you interact with remote repositories. `fetch` gets the changes without merging, while `pull` fetches and merges in one step.

```
git push -u origin feature/auth
```

> **Tip:** The `-u` flag sets the upstream tracking, so future pushes only require `git push`.

Viewing history

Task	Command
View log	`git log`
One-line log	`git log --oneline`
View changes	`git status`

You can inspect the history of commits and the current state of your working directory. The log shows commit hashes, authors, dates, and messages.

```
git log --oneline -n 5
```

Further reading

- Pro Git Book — The definitive, free guide to Git internals and advanced usage.