# GIT

## Complete Developer Reference Guide

*From Basics to Advanced Workflows*

---

**@coding_dev_**

Version 1.0 | 2026

# 1. Introduction to Git

Git is a distributed version control system created by Linus Torvalds in 2005. It tracks changes in source code during software development, enabling multiple developers to work together on non-linear development.

## 1.1 Key Concepts

### Repository (Repo)

A repository is a storage location for your project. It contains all project files and the entire revision history.

### Working Directory

The local copy of your project files where you make changes before staging and committing.

### Staging Area (Index)

An intermediate area where changes are listed before being committed. Think of it as a draft for your next commit.

### Commit

A snapshot of your staged changes saved permanently to the repository history. Each commit has a unique SHA hash.

### Branch

A lightweight, movable pointer to a specific commit. Branches allow isolated development without affecting the main codebase.

### Remote

A version of your repository hosted on the internet or network (e.g., GitHub, GitLab, Bitbucket).

## 1.2 Git vs Other VCS

Unlike centralized systems (SVN, CVS), Git is distributed — every developer has a full copy of the repository, including history. This makes Git faster, more flexible, and more resilient.

# 2. Setup & Configuration

Before using Git, configure your identity. This information is attached to every commit you make.

## 2.1 Initial Setup

Set your name:

```
git config --global user.name "Your Name"
```

Set your email:

```
git config --global user.email "you@example.com"
```

Set default branch name to main:

```
git config --global init.defaultBranch main
```

Set default editor (e.g., VS Code):

```
git config --global core.editor "code --wait"
```

## 2.2 View Configuration

```
git config --list
git config --global --list
git config user.name
```

## 2.3 .gitignore

A .gitignore file tells Git which files or directories to ignore. Create it in your project root:

```
# Ignore node_modules
node_modules/

# Ignore build output
dist/
build/

# Ignore environment files
.env
.env.local
```

**Tip:** Use gitignore.io to generate .gitignore templates for your stack.

# 3. Core Commands

## 3.1 Repository Initialization

### git init

Creates a new Git repository in the current directory. Initializes a hidden .git folder.

```
git init
git init my-project    # init inside a new folder
```

### git clone

Downloads an existing repository from a remote URL, including all history and branches.

```
git clone https://github.com/user/repo.git
git clone https://github.com/user/repo.git my-folder
git clone --depth 1 https://github.com/user/repo.git   # shallow clone
```

**Tip:** Use --depth 1 for large repos when you only need the latest snapshot.

## 3.2 Staging & Committing

### git status

Shows the state of your working directory and staging area — modified, staged, and untracked files.

```
git status
git status -s    # short format
```

### git add

Stages changes from your working directory into the index.

```
git add .                  # stage all changes
git add filename.js        # stage a specific file
git add src/               # stage an entire folder
git add -p                 # interactively stage chunks
```

**Tip:** git add -p is powerful — it lets you review and stage changes piece by piece.

### git commit

Records the staged snapshot permanently into the repository history.

```
git commit -m "feat: add login functionality"
git commit -am "fix: update styles"   # stage tracked files + commit
git commit --amend            # modify the most recent commit
```

```
git commit --amend --no-edit  # amend without changing message
```

**Tip:** Follow Conventional Commits: feat:, fix:, chore:, docs:, refactor:, test:

### git diff

Shows differences between your working directory, staging area, and commits.

```
git diff                    # unstaged changes
git diff --staged           # staged changes
git diff HEAD               # all changes since last commit
git diff branch1..branch2   # diff between branches
```

# 4. Branching & Merging

## 4.1 Branch Commands

### git branch

Lists, creates, renames, or deletes branches.

```
git branch              # list local branches
git branch -a           # list all branches (local + remote)
git branch feature-x    # create a new branch
git branch -d feature-x # delete branch (safe)
git branch -D feature-x # force delete branch
git branch -m old-name new-name  # rename branch
```

### git checkout / git switch

Switches between branches or restores files. git switch is the modern alternative.

```
git checkout main           # switch to main
git checkout -b feature-x   # create + switch
git switch main             # modern syntax
git switch -c feature-x     # create + switch (modern)
```

## 4.2 Merging

### git merge

Combines the history of one branch into another. Run from the target branch.

```
git checkout main
git merge feature-x
git merge --no-ff feature-x    # always create merge commit
git merge --squash feature-x   # squash all commits into one
git merge --abort              # abort conflicted merge
```

**Tip:** Use --no-ff to preserve branch history and make merges visible in the log.

## 4.3 Rebasing

### git rebase

Replays commits from one branch on top of another. Creates a cleaner linear history.

```
git rebase main            # rebase current branch onto main
git rebase -i HEAD~3       # interactive rebase last 3 commits
git rebase --continue      # continue after resolving conflicts
```

```
git rebase --abort          # cancel the rebase
```

**Tip:** Never rebase commits that have been pushed to a shared remote branch.

## 4.4 Resolving Merge Conflicts

When Git cannot automatically merge, it marks the conflict in the file:

```
<<<<<<< HEAD
Your changes here
=======
Incoming changes here
>>>>>>> feature-x
```

Edit the file to keep the desired changes, then:

```
git add conflicted-file.js
git commit -m "merge: resolve conflict in login"
```

# 5. Remote Repositories

## 5.1 Managing Remotes

```
git remote -v                          # list remotes
git remote add origin <url>            # add remote
git remote remove origin               # remove remote
git remote rename origin upstream      # rename remote
git remote set-url origin <new-url>    # change URL
```

## 5.2 Push & Pull

### git push

Uploads local commits to the remote repository.

```
git push origin main
git push -u origin main      # set upstream tracking
git push --all               # push all branches
git push --tags              # push all tags
git push --force-with-lease  # safer force push
```

**Tip:** Always prefer --force-with-lease over --force to avoid overwriting others' work.

### git pull

Fetches changes from remote and merges into current branch.

```
git pull
git pull origin main
git pull --rebase            # fetch + rebase instead of merge
```

### git fetch

Downloads remote changes without merging them into your local branch.

```
git fetch origin
git fetch --all              # fetch from all remotes
git fetch --prune            # remove stale remote-tracking branches
```

# 6. History & Inspection

## 6.1 Viewing History

```
git log
git log --oneline
git log --oneline --graph --all     # visual branch map
git log --author='Name'             # filter by author
git log --since='2 weeks ago'       # filter by date
git log -- filename.js             # history of a file
git log -p                          # show patches (diffs)
```

## 6.2 Inspecting Changes

```
git show <commit-hash>     # show a specific commit
git show HEAD              # show last commit
git blame filename.js      # who changed each line
git diff HEAD~1 HEAD       # diff last two commits
```

## 6.3 Searching

```
git grep "search term"           # search working directory
git log -S "function name"       # find when string was added/removed
git log --all --grep="bug fix"   # search commit messages
```

# 7. Undoing Changes

Git provides multiple ways to undo work. Choose the right tool based on whether changes are committed and shared.

## 7.1 Before Staging

```
git restore filename.js     # discard changes in working dir
git restore .               # discard ALL unstaged changes
git clean -fd               # remove untracked files + dirs
```

## 7.2 After Staging

```
git restore --staged filename.js   # unstage a file
git reset HEAD filename.js         # older syntax (same effect)
```

## 7.3 After Committing

### git revert

Creates a new commit that undoes the changes of a previous commit. Safe for shared history.

```
git revert <commit-hash>
git revert HEAD         # undo last commit
git revert HEAD~3..HEAD # revert last 3 commits
```

### git reset

Moves HEAD to a previous commit. Use with caution — rewrites history.

```
git reset --soft HEAD~1   # keep changes staged
git reset --mixed HEAD~1  # keep changes unstaged (default)
git reset --hard HEAD~1   # discard all changes (DANGEROUS)
```

**Tip:** Never use git reset --hard on commits that have been pushed to a shared branch.

## 7.4 Stashing

### git stash

Temporarily shelves changes so you can work on something else.

```
git stash                   # stash current changes
git stash push -m "message" # stash with description
git stash list              # list all stashes
git stash pop               # apply + remove latest stash
git stash apply stash@{2}   # apply specific stash
```

```
git stash drop stash@{0}    # delete a stash
git stash clear             # delete all stashes
```

# 8. Tags & Releases

Tags mark specific points in your repository history, typically used for releases.

## 8.1 Creating Tags

```
git tag v1.0.0                          # lightweight tag
git tag -a v1.0.0 -m "Release v1.0.0"   # annotated tag
git tag -a v1.0.0 <commit-hash>          # tag a past commit
```

## 8.2 Managing Tags

```
git tag                    # list all tags
git show v1.0.0            # show tag details
git tag -d v1.0.0          # delete a local tag
git push origin v1.0.0     # push a tag to remote
git push origin --tags     # push all tags
git push origin --delete v1.0.0   # delete remote tag
```

# 9. Advanced Commands

## 9.1 Cherry-pick

Apply a specific commit from one branch onto another without merging the whole branch.

```
git cherry-pick <commit-hash>
git cherry-pick A..B            # pick a range of commits
git cherry-pick --no-commit <hash>  # apply without committing
```

## 9.2 Bisect

Binary search through commit history to find which commit introduced a bug.

```
git bisect start
git bisect bad              # current commit is bad
git bisect good v1.2.0      # last known good commit
# Git checks out a middle commit; test it, then:
git bisect good             # or: git bisect bad
git bisect reset            # end bisect session
```

## 9.3 Worktrees

Check out multiple branches simultaneously in separate directories.

```
git worktree add ../hotfix hotfix-branch
git worktree list
git worktree remove ../hotfix
```

## 9.4 Submodules

Include another Git repository inside your project.

```
git submodule add https://github.com/user/lib.git
git submodule update --init --recursive
git submodule update --remote
```

## 9.5 Reflog

A safety net — records every movement of HEAD, even resets and amends.

```
git reflog
git reflog show branch-name
git checkout HEAD@{3}   # restore to a previous HEAD state
```

**Tip:** Reflog is your undo history for local operations. Entries expire after 90 days by default.

# 10. Quick Reference Cheat Sheet

## Setup

| Command | Description |
|---|---|
| `git init` | Initialize a new repository |
| `git clone <url>` | Clone a remote repository |
| `git config --global user.name` | Set your Git username |
| `git config --global user.email` | Set your Git email |

## Daily Workflow

| Command | Description |
|---|---|
| `git status` | Show working tree status |
| `git add .` | Stage all changes |
| `git commit -m 'msg'` | Commit staged changes |
| `git pull` | Fetch and merge from remote |
| `git push` | Push commits to remote |
| `git diff` | Show unstaged changes |
| `git log --oneline` | Compact commit history |

## Branching

| Command | Description |
|---|---|
| `git branch` | List branches |
| `git branch <name>` | Create a new branch |
| `git switch -c <name>` | Create and switch to branch |
| `git merge <branch>` | Merge branch into current |
| `git rebase main` | Rebase current onto main |
| `git branch -d <name>` | Delete a branch |

## Undoing

| Command | Description |
|---|---|
| `git restore <file>` | Discard working dir changes |
| `git restore --staged <file>` | Unstage a file |
| `git revert HEAD` | Undo last commit (safe) |
| `git reset --soft HEAD~1` | Undo commit, keep staged |
| `git stash` | Shelf current changes |
| `git stash pop` | Apply and remove latest stash |

## Remote

| Command | Description |
|---|---|
| `git remote -v` | List remotes |
| `git remote add origin <url>` | Add a remote |
| `git fetch --all` | Fetch all remotes |
| `git push -u origin main` | Push and set upstream |
| `git push --force-with-lease` | Safe force push |
| `git pull --rebase` | Pull with rebase |