NOTEBOOK EDITION

Git & GitHub 50 Commands

The 50 that cover almost all of your actual work — including the undo buttons.

A free sample from the Junior Dev Cheat Sheet Vault

by Nat · First Commit Lab



Listen — you do not need to memorise this vault. Nobody does. Working developers look things up constantly. The real skill is simple: know what exists, then find it fast. That's what this trains.

WORDS IN THIS VAULT — IN PLAIN ENGLISH:

Repo = your project folder, tracked by Git.

Staging area = the waiting room for changes: you pick what goes into the next snapshot, then you commit it.

HEAD = where you are right now in the history.

Commit = a saved snapshot of your work.

Branch = a parallel copy of the project where you can experiment safely.

Remote / upstream = the online copy (e.g. on GitHub) that your branch follows.

Print it. Pin it. Keep it next to your keyboard. Future you says thanks.

SHEET 1 · GIT & GITHUB — 50 COMMANDS

The 50 that cover almost all of your actual work — including the undo buttons

Plain English first: Git saves snapshots of your project. Each snapshot is a commit. You choose what goes into the next snapshot — that choosing step is called staging. Simple idea. The 50 commands below are just ways to save, share, branch off and undo those snapshots.

Setup & getting started (1–8)

#	Command	What it does
1	`git config --global user.name "Name"`	Set the name attached to your commits
2	`git config --global user.email "you@mail.com"`	Set the email attached to your commits
3	`git init`	Turn the current folder into a Git repo
4	`git clone <url>`	Copy a remote repo onto your machine
5	`git clone <url> <folder>`	Clone into a named folder
6	`git status`	Show changed, staged and untracked files
7	`git status -s`	Short status (M = modified, A = added, ?? = untracked)
8	`git help <command> / git <command> --help`	Open the manual for a command

Stage, commit & inspect (9–20)

#	Command	What it does
9	`git add <file>`	Stage one file
10	`git add .`	Stage all changes in the current folder
11	`git add -p`	Stage changes interactively, piece by piece (each piece is called a hunk)
12	`git commit -m "message"`	Commit staged changes with a message
13	`git commit -am "message"`	Stage tracked files and commit in one step
14	`git commit --amend`	Rewrite the last commit (message or content)
15	`git diff`	Show unstaged changes
16	`git diff --staged`	Show staged changes (what will be committed)
17	`git log --oneline`	Compact history, one line per commit
18	`git log --oneline --graph --all`	Visual branch history
19	`git show <commit>`	Show what one commit changed
20	`git blame <file>`	Who changed each line, and in which commit

SHEET 1 · GIT & GITHUB — 50 COMMANDS

Branching & merging (21–31)

#	Command	What it does
21	`git branch`	List local branches (* = current)
22	`git branch <name>`	Create a branch (does not switch to it)
23	`git switch <name>`	Switch to a branch (modern, preferred)
24	`git switch -c <name>`	Create and switch to a new branch
25	`git checkout -b <name>`	Older equivalent of switch -c — still very common
26	`git merge <branch>`	Merge a branch into the current branch
27	`git merge --abort`	Cancel a merge with conflicts
28	`git rebase main`	Replay your commits on top of main (linear history)
29	`git branch -d <name>`	Delete a merged branch (-D forces deletion)
30	`git cherry-pick <commit>`	Apply one commit onto the current branch
31	`git tag v1.0.0`	Mark a commit as a release / version

Remotes — GitHub & collaboration (32–40)

#	Command	What it does
32	`git remote -v`	List remotes and their URLs
33	`git remote add origin <url>`	Connect this repo to a remote
34	`git fetch`	Download remote changes without merging them
35	`git pull`	Fetch + merge the upstream branch (= fetch + merge)
36	`git pull --rebase`	Fetch + rebase (avoids a merge commit)
37	`git push`	Push commits to the tracked upstream branch
38	`git push -u origin <branch>`	Push and set upstream (first push of a branch)
39	`git push origin --delete <branch>`	Delete a remote branch
40	`git clone --depth 1 <url>`	Shallow clone — faster for large repos

SHEET 1 · GIT & GITHUB — 50 COMMANDS

Undo, stash & rescue (41–50) ☆

#	Command	What it does
41	`git restore <file>`	Discard unstaged changes in a file
42	`git restore --staged <file>`	Unstage a file (keeps the changes)
43	`git reset --soft HEAD~1`	Undo last commit, keep changes staged
44	`git reset --mixed HEAD~1`	Undo last commit, keep changes unstaged (default)
45	`git reset --hard HEAD~1`	Undo last commit AND delete its changes — destructive
46	`git revert <commit>`	Undo a commit with a new commit — safe for shared branches
47	`git stash / git stash pop`	Shelve changes / bring the last stash back
48	`git stash list / git stash apply stash@{0}`	List stashes / apply a specific one
49	`git reflog`	History of everywhere HEAD has been — your rescue log
50	`git clean -fd`	Delete untracked files/folders — destructive, preview with -n first

NAT'S RULE — DON'T LEARN THIS ONE BY ACCIDENT: ⚠

reset --hard and clean -fd delete work permanently. Not 'sort of' permanently. If a commit was ever made, reflog can usually find it. If it was never committed or stashed, it is gone. Check the command before you run it. Twice.

INTERVIEW BOX — GIT: 🎤

They will ask this.

- **Merge vs rebase?** Merge keeps the true history and adds a merge commit. Rebase rewrites your commits on top of the target, so history stays in one straight line. Never rebase commits that other people have already downloaded.
- **reset vs revert?** reset rewrites history — use it on your own, private branches. revert adds a new 'undo' commit — safe on shared branches.
- **Good commit message:** a short subject that gives an order ("Fix login redirect"). The body explains WHY, not what.



Nat's closing note:

If you master one section here, make it Undo, stash & rescue. Fixing a mistake calmly — instead of panicking — is what makes you look experienced.

That was sheet 1 of 6.



You just got the 50 Git commands — the undo buttons included. Here's the rest of the vault they're from:

- **Python, SQL, JavaScript** — the core syntax and patterns junior roles test first
- **Docker & Linux** — containers and the terminal, without the panic
- **Interview Rapid-Fire** — 30 questions with one-line answers, plus a 7-day study plan

I'm Nat. I dig through the boring documentation, verify every command, and turn it into clean notes I wish someone had handed me earlier. No copied-and-broken code. No filler.

$9, one payment. Print it, pin it, keep it next to your keyboard.

Get the full Vault here:

`first-commit-lab.whop.site`

— Nat · First Commit Lab