SkillNewton

Git Commands Cheat Sheet

Everyday Git commands on one page: setup, branching with switch and restore, merging, rebasing, remotes, safe undo, stash, tags and .gitignore.

> This cheat sheet lists the Git commands you will use every day, grouped by task: setting up, committing, branching, merging, working with remotes, reading history and undoing mistakes. It uses the modern `git switch` and `git restore` commands and shows the older `git checkout` equivalents. Keep it open while you work, and read the warnings on the reset and force-push rows before you run them on shared code.

How to use it

1. **Set up Git once.** Run the configuration commands in the first section so every commit carries your name and email address.
2. **Find the task, not the command.** Sections are grouped by what you want to do, such as undoing a change or updating from a remote. Scan the section titles first.
3. **Check before you change.** Run `git status` and `git log --oneline` before any command that rewrites history, so you know exactly where you are.
4. **Practise on a test repository.** Create a throwaway repository with `git init` and try merging, rebasing and resetting there before using them on real work.

Setup and starting a repository

`git config --global user.name "Your Name"`	Set the name recorded on your commits.
`git config --global user.email "you@example.com"`	Set the email recorded on your commits. Use the one linked to your GitHub account.
`git config --global init.defaultBranch main`	Name the first branch `main` in new repositories.
`git config --global core.autocrlf true`	Windows only: convert line endings to CRLF on checkout and back to LF on commit. Use `input` on macOS and Linux.
`git config --list --show-origin`	Show every setting and the file it comes from.
`git init`	Turn the current folder into a new Git repository.
`git clone https://github.com/example/repo.git`	Copy a remote repository, with its full history, into a new folder.
`git clone --depth 1 https://github.com/example/repo.git`	Shallow clone: only the latest commit. Faster for large projects you only need to read or build.
`git status`	Show the current branch and which files are modified, staged or untracked.
`git status -s`	Short status, one line per file.
`git help commit`	Open the manual for any command (here `commit`).

Staging and committing

`git add index.php`	Stage one file for the next commit.
`git add .`	Stage all changes in the current folder and below, including new files.
`git add -p`	Review changes piece by piece and choose which parts to stage.
`git commit -m "Add contact form validation"`	Commit the staged changes with a message.
`git commit -am "Fix typo in footer"`	Stage all modified tracked files and commit in one step. New, untracked files are not included.
`git commit --amend`	Change the last commit (message or content). Rewrites the commit, so only use it before pushing.
`git rm old-page.php`	Delete a file and stage the deletion.
`git rm --cached config.local.php`	Stop tracking a file but keep it on disk.
`git mv about.html about-us.html`	Rename or move a file and stage the change.

Branching

`git branch`	List local branches. The current one is marked with *.
`git branch -a`	List local and remote-tracking branches.
`git branch -vv`	List branches with their last commit and the remote branch each one tracks.
`git switch -c feature/login`	Create a new branch and switch to it.
`git switch main`	Switch to an existing branch.
`git switch -`	Switch back to the previous branch.
`git checkout -b feature/login`	Older equivalent of `git switch -c`; still widely used.
`git checkout main`	Older equivalent of `git switch main`.
`git branch -m new-name`	Rename the current branch.
`git branch -d feature/login`	Delete a branch that has been merged.
`git branch -D feature/login`	Force-delete a branch even if it is not merged. Unmerged commits become hard to find.

Merging and rebasing

`git merge feature/login`	Merge the named branch into the current branch.
`git merge --no-ff feature/login`	Always create a merge commit, even when a fast-forward is possible, so the feature stays visible in history.
`git merge --abort`	Stop a merge that has conflicts and go back to the state before it started.
`git rebase main`	Replay the current branch's commits on top of `main`. Rewrites those commits.

`git rebase --continue`	Carry on after fixing a conflict during a rebase (stage the fixed files first).
`git rebase --abort`	Cancel the rebase and return to where you started.
`git rebase -i HEAD~3`	Interactive rebase of the last three commits: reword, squash, reorder or drop them. Only for unpushed commits.
`git cherry-pick a1b2c3d`	Copy a single commit from another branch onto the current branch.

```
git switch main
git merge feature/login
# CONFLICT (content): Merge conflict in index.php
# 1. Open index.php and edit the block between <<<<<<<, ======= and >>>>>>>
# 2. Stage the fixed file and finish the merge:
git add index.php
git commit
```

Remotes: fetch, pull and push

`git remote -v`	List remotes and their URLs.
`git remote add origin https://github.com/example/repo.git`	Connect a local repository to a remote called `origin`.
`git remote set-url origin git@github.com:example/repo.git`	Change a remote URL, for example from HTTPS to SSH.
`git fetch origin`	Download new commits and branches without changing your files.
`git fetch --prune`	Fetch and remove remote-tracking branches that were deleted on the remote.
`git pull`	Fetch and merge the tracked remote branch into the current branch.
`git pull --rebase`	Fetch and rebase your local commits on top, avoiding extra merge commits.
`git push -u origin feature/login`	Push a new branch and set it to track the remote, so plain `git push` works next time.
`git push`	Send your commits to the tracked remote branch.
`git push --force-with-lease`	Overwrite the remote branch after a rebase, but refuse if someone else pushed in the meantime. Never force-push `main`.
`git push origin --delete feature/login`	Delete a branch on the remote.

Inspecting history and changes

`git log --oneline --graph --all`	Compact history of every branch, drawn as a graph.
`git log -p index.php`	History of one file, with the changes in each commit.
`git log --author="Priya"`	Commits by one author.
`git log -S "sendInvoice"`	Find commits that added or removed this text.
`git diff`	Unstaged changes in your working folder.

`git diff --staged`	Changes that are staged and will go into the next commit.
`git diff main...feature/login`	What the feature branch changed since it split from `main`.
`git show a1b2c3d`	Show one commit: message, author and changes.
`git blame index.php`	Show who last changed each line, and in which commit.
`git blame -L 10,20 index.php`	Blame only lines 10 to 20.
`git reflog`	Every position HEAD has been at recently. Use it to find commits after a bad reset or rebase.

Undoing changes (read the warnings)

`git restore index.php`	Discard unstaged edits to a file. Warning: those edits are lost for good.
`git restore --staged index.php`	Unstage a file but keep your edits.
`git restore --source=HEAD~1 index.php`	Bring back a file as it was one commit ago.
`git revert a1b2c3d`	Create a new commit that reverses an earlier one. The safe way to undo pushed commits.
`git reset --soft HEAD~1`	Undo the last commit but keep its changes staged. Rewrites history: local commits only.
`git reset HEAD~1`	Same as `--mixed` (the default): undo the last commit and keep its changes unstaged.
`git reset --hard HEAD~1`	Undo the last commit and delete its changes and any uncommitted work. Warning: destructive; check `git status` first.
`git reset --hard origin/main`	Make the current branch match the remote exactly, discarding all local commits and edits. Warning: destructive.
`git clean -n`	Dry run: list untracked files that would be deleted.
`git clean -fd`	Delete untracked files and folders. Warning: they are not in Git, so they cannot be recovered.

```
# Which undo do I need?
git restore index.php            # throw away unstaged edits to a file
git restore --staged index.php   # unstage, keep the edits
git commit --amend               # fix the last commit (not yet pushed)
git revert a1b2c3d               # undo a pushed commit with a new commit
git reset --soft HEAD~1          # undo last commit, keep changes staged
git reset --hard HEAD~1          # undo last commit AND delete its changes
git reflog                       # find a "lost" commit after a bad reset
```

Stash: put work aside

`git stash push -m "half-done header"`	Save uncommitted changes to tracked files and clean the working folder.
`git stash -u`	Stash untracked files as well.
`git stash list`	List saved stashes, newest first as `stash@{0}`.

`git stash show -p stash@{0}`	Show the changes inside a stash.
`git stash apply stash@{0}`	Reapply a stash and keep it in the list.
`git stash pop`	Reapply the latest stash and remove it from the list (it is kept if there is a conflict).
`git stash drop stash@{0}`	Delete one stash.
`git stash branch fix/header stash@{0}`	Create a branch from the commit where the stash was made and apply the stash there.

Tags and releases

`git tag`	List tags.
`git tag -a v1.2.0 -m "Release 1.2.0"`	Create an annotated tag (stores author, date and message). Use these for releases.
`git tag v1.2.0`	Create a lightweight tag: just a name pointing at a commit.
`git tag -a v1.1.0 9fceb02 -m "Release 1.1.0"`	Tag an older commit.
`git show v1.2.0`	Show the tag details and the commit it points to.
`git push origin v1.2.0`	Push one tag. Tags are not pushed by plain `git push`.
`git push origin --tags`	Push all local tags.
`git tag -d v1.2.0`	Delete a local tag.
`git push origin --delete v1.2.0`	Delete a tag on the remote.
`git switch --detach v1.2.0`	Look at the code as it was at a tag (detached HEAD). Create a branch before making commits.

.gitignore

`*.log`	Ignore every file ending in .log, in any folder.
`/vendor/`	Ignore the `vendor` folder at the repository root only.
`node_modules/`	Ignore any folder called `node_modules`, at any depth.
`.env*`	Ignore `.env`, `.env.local` and similar files that hold secrets.
`!.env.example`	Exception: keep this file tracked. The ! rule must come after the rule it overrides.
`git check-ignore -v storage/app.log`	Show which .gitignore rule is ignoring a file.
`git rm --cached .env`	Stop tracking a file that was committed before it was ignored. It stays in old commits, so change any exposed passwords or keys.
`git config --global core.excludesFile ~/.gitignore_global`	Use a personal ignore file for every repository (good for editor and OS files).

```
# .gitignore for a typical PHP project
/vendor/
node_modules/
.env*
```

```
!.env.example
*.log
/writable/cache/
.DS_Store
Thumbs.db
.idea/
.vscode/
```

Online version, with the matching free guide: skillnewton.com/resources/cheat-sheets/git-commands/
Learn this hands-on: PHP Web Development: Foundation & Practical Implementation – skillnewton.com/courses/php-foundation-course/