

Sidebar

Programming · Git Command Line Cheat Sheet

The Git commands you use every day, grouped by task: setup, the daily loop, history, branches, merging and rebasing, undoing, remotes, stash and tags, searching, ignoring files and naming commits. Written for Git 2.49 on Windows, the version installed here; the commands and options were run in a scratch repository to check they work. This is the command-line companion to the VS Code Git and Source Control sheet, which covers the editor's settings.

SETUP & CONFIGURATION

- `git --version` Check the install (here: 2.49.0.windows.1)
- `git config --global user.name "Your Name"` Who your commits are from; also `user.email`
- `git config --list --show-origin` Every setting and the file it came from
- `--system --global --local` Where a setting is stored: all users / your account (`~/.gitconfig`) / this repository (`.git/config`); local beats global beats system
- `git config --global init.defaultBranch main` New repositories start on this branch. **Yours is currently `master`**, set in Git's system config
- `git config --global alias.st "status -sb"` Create a shortcut: then `git st` works
- `git config --global core.editor "code --wait"` Use VS Code for commit messages
- `core.autocrlf` Line-ending conversion; `true` here (Windows), which causes the "LF will be replaced by CRLF" warnings
- `credential.helper` `manager` here: Git Credential Manager remembers your logins
- `git help <command>` or `git <command> -h` Full manual page / one-screen summary

STARTING A REPOSITORY

- `git init` Turn the current folder into a repository; `-b main` (or `--initial-branch`) names the first branch
- `git clone <url> [folder]` Copy a remote repository
- `git clone --depth 1 <url>` Only the latest commit: much faster for big repositories
- `git clone --branch <name> --single-branch <url>` Only one branch
- `git clone --filter=blob:none <url>` A partial clone: file contents are fetched when needed
- `git clone --recurse-submodules <url>` Also fetch submodules
- `git clone --bare / --mirror` Server-style copies with no working folder

THE DAILY LOOP

`git status -sb` Short status with the branch and its upstream (`-s` short, `-b` branch)

`git add <file>` `git add .` `git add -u` Stage a file / everything here / only files Git already tracks

`git add -p` Choose which pieces of a file to stage, one hunk at a time

`git diff` Changes not yet staged

`git diff --staged (or --cached)` Changes that are staged for the next commit

`git diff --stat --name-only --word-diff` Summary / just file names / word-level changes; `git diff --check` finds whitespace mistakes

`git commit -m "message"` Commit what is staged; `-a` also stages tracked files; `-v` shows the diff in the editor

`git commit --amend` Change the last commit; `--no-edit` keeps its message; only before you push it

`git commit --allow-empty --signoff --fixup <sha>` Empty commit / add a Signed-off-by line / mark a fix for a later autosquash

`git rm <file>` `git rm --cached <file>` Delete and stop tracking / stop tracking but keep the file

`git mv old new` Move or rename a tracked file

LOOKING AT HISTORY

`git log --oneline --graph --decorate --all` The classic compact picture of every branch

`git log -p` `git log --stat` With the changes / with a change summary; add `-3` to limit

`git log --author="Name" --since="2 weeks ago"` Filter by author and date

`git log --grep="fix"` Commits whose message matches

`git log -S"text"` `git log -G"regex"` Commits that added or removed that text / changed lines matching the pattern

`git log --follow -- <file>` History of one file, through renames

`git log -L 10,20:<file>` History of a range of lines

`git log main..feature` Commits on `feature` that are not on `main`

`git log main...feature --left-right` Commits on either side but not both

`git log --first-parent --no-merges` Follow only the main line / hide merge commits

`git log --pretty=format:"%h %an %s"` Your own layout: hash, author, subject

`git show <rev>` `git show <rev>:<path>` One commit / a file as it was then

`git blame -L 1,20 <file>` Who last changed each line

`git shortlog -sn` Commit counts per author

`git describe --tags --always` A readable name for the current commit

BRANCHES

`git branch` List local branches; `-vv` adds the last commit and upstream, `-a` includes remote ones, `-r` only remote ones

`git branch --show-current` Print just the current branch name

`git switch <branch>` Move to a branch

`git switch -c <new>` Create a branch and move to it (`git checkout -b` in older guides)

`git switch -` Back to the previous branch

`git switch --detach <rev>` Look at an old commit without being on a branch

`git branch <new>` Create a branch without moving to it

`git branch -m <old> <new>` Rename a branch

`git branch -d <b>` `git branch -D <b>` Delete if fully merged / delete regardless

`git branch --merged --no-merged` Which branches are already in the current one, and which are not

MERGING, REBASING & CHERRY-PICKING

`git merge <branch>` Bring that branch into the current one

`git merge --no-ff / --ff-only / --squash` Always make a merge commit / refuse unless it fast-forwards / combine everything into one change you commit yourself

`git merge --abort` Give up a merge that has conflicts

`<<<<<< ===== >>>>>>` Conflict markers in a file: edit the file, `git add` it, then `git merge --continue` (or commit)

`git mergetool` Open your configured merge tool

`git rebase main` Replay your commits on top of `main`

`git rebase -i HEAD~3` Edit the last three commits: `pick`, `reword`, `edit`, `squash`, `fixup`, `drop`

`git rebase --autosquash --autostash` Fold `--fixup` commits automatically / stash and restore your changes around it

`git rebase --onto <new> <old> <branch>` Move a branch to a different base

`git rebase --continue --skip --abort` After fixing a conflict / skip this commit / go back to before the rebase

`git cherry-pick <sha>` Copy one commit here; `-x` notes where it came from, `--no-commit` leaves it staged

`git revert <sha>` Make a new commit that undoes one (`-m 1` for a merge commit)

`Do not rebase shared history` Rebasing rewrites commits: only rebase branches that nobody else has pulled

UNDOING THINGS (FROM SAFEST TO RISKIEST)

- `git restore --staged <file>` Unstage a file; your edits stay
- `git restore <file>` **Discard** your uncommitted edits to a file: they cannot be recovered
- `git restore --source=HEAD~1 <file>` Bring back a file as it was one commit ago
- `git commit --amend` Fix the last commit (not yet pushed)
- `git reset --soft HEAD~1` Undo the last commit; changes stay staged
- `git reset --mixed HEAD~1` Undo the last commit; changes stay in your files, unstaged (the default)
- `git revert <sha>` Undo a commit you already pushed, safely, with a new commit
- `git reset --hard <rev>` **Throw away** commits and uncommitted changes: use with great care
- `git reflog` Where HEAD has been: your safety net for "lost" commits
- `git switch -c rescue <sha>` Give a lost commit from the reflog a branch again
- `git clean -n -d` Dry run: which untracked files and folders would be deleted
- `git clean -fd` Delete them (`-x` also removes ignored files)

REMOTES: FETCH, PULL & PUSH

- `git remote -v` List remotes and their addresses
- `git remote add origin <url>` Add one; `set-url` changes it, `rename` and `remove` also exist
- `git remote show origin` Details of a remote and how your branches track it
- `git fetch --prune --all` Download new commits and forget deleted remote branches; nothing in your files changes
- `git pull` Fetch and then merge; `--rebase` rebases instead, `--ff-only` refuses a merge, `--autostash` saves and restores your changes
- `git push` Upload the current branch
- `git push -u origin <branch>` First push of a branch; `-u` (`--set-upstream`) remembers it
- `git push --force-with-lease` A forced push that fails if someone else pushed in the meantime: use this instead of `--force`
- `git push --tags --follow-tags` Upload all tags / annotated tags that point at pushed commits
- `git push origin --delete <branch>` Delete a remote branch
- `git push --dry-run` Show what would be pushed
- `git branch --set-upstream-to=origin/<b>` Tell Git which remote branch this one follows

STASH & TAGS

<code>git stash push -u -m "wip"</code>	Put away your uncommitted work (<code>-u</code> includes untracked files)
<code>git stash list</code>	See the stack of stashes
<code>git stash show -p stash@{0}</code>	See what is in a stash
<code>git stash pop</code> <code>git stash apply</code>	Restore the latest stash and remove it / restore it and keep it
<code>git stash drop</code> <code>git stash clear</code>	Delete one stash / all of them
<code>git stash branch <name></code>	Make a branch from a stash
<code>git tag v1.0</code>	A lightweight tag on the current commit
<code>git tag -a v1.0 -m "Release 1.0"</code>	An annotated tag with a message (better for releases)
<code>git tag -l</code> <code>git tag -d v1.0</code>	List tags / delete one
<code>git push origin v1.0</code>	Upload one tag

SEARCHING & INSPECTING

<code>git grep -n -i "text"</code>	Search tracked files; <code>-c</code> counts per file
<code>git ls-files -m --others --exclude-standard</code>	Modified files / untracked files that are not ignored
<code>git check-ignore -v <file></code>	Which ignore rule applies to a file
<code>git rev-parse --short HEAD --show-toplevel</code>	Short hash of the current commit / the repository's top folder
<code>git ls-tree -r HEAD --name-only</code>	Every file in a commit
<code>git cat-file -p <sha></code>	Show the raw contents of a Git object
<code>git bisect start / bad / good <sha></code>	Binary-search for the commit that introduced a bug; finish with <code>git bisect reset</code>
<code>git bisect run <command></code>	Let a script decide good or bad automatically
<code>git archive --format=zip -o out.zip HEAD</code>	Make a zip of a commit without the <code>.git</code> folder
<code>git worktree add ../dir -b <branch></code>	A second working folder for another branch; <code>list</code> , <code>remove</code> manage them
<code>git sparse-checkout set <folders></code>	Check out only some folders of a big repository
<code>git fsck</code> <code>git gc</code> <code>git maintenance run</code>	Check, tidy and optimise the repository

IGNORING FILES & LINE ENDINGS

<code>.gitignore</code>	A file of patterns for files Git should not track
<code>*.log</code>	Ignore by pattern
<code>build/</code>	A trailing slash means a folder
<code>/config.local</code>	A leading slash anchors it to the top folder
<code>**/temp</code>	Match at any depth
<code>!keep.log</code>	A leading <code>!</code> re-includes something an earlier pattern ignored
<code># comment</code>	Comment line
<code>git rm --cached <file></code>	A file already tracked stays tracked until you do this; adding it to <code>.gitignore</code> is not enough
<code>core.excludesFile</code>	A personal ignore file that applies to every repository
<code>.gitattributes</code>	Per-path rules, such as <code>* text=auto</code> or <code>*.sh text eol=lf</code> to fix line endings across machines
<code>Git LFS</code>	Installed here (<code>filter.lfs.*</code> in the system config); stores large files outside the normal history

NAMING COMMITS & RANGES (REVISION SYNTAX)

<code>HEAD @</code>	The current commit
<code>HEAD~1</code> <code>HEAD~3</code>	The parent / the great-grandparent along the first-parent line
<code>HEAD^</code> <code>HEAD^2</code>	First parent / second parent (the merged-in side of a merge)
<code>main@{upstream}</code> <code>@{u}</code>	The remote branch a local branch follows
<code>main@{yesterday}</code> <code>main@{3}</code>	Where a branch was yesterday / three moves ago (from the reflog)
<code>A..B</code>	Commits in B but not in A
<code>A...B</code>	Commits in either but not both
<code>:/text</code>	The most recent commit whose message contains the text
<code><rev>:<path></code>	A file as it was in that commit
<code>short hashes</code>	The first 7 or more characters of a hash are enough if they are unique

HANDY ALIASES TO COPY (MY SUGGESTIONS)

<code>git config --global alias.st "status -sb"</code>	Short status
<code>git config --global alias.lg "log --oneline --graph --decorate --all"</code>	The graph in two letters
<code>git config --global alias.co "switch"</code>	A short name for switching branches
<code>git config --global alias.unstage "restore --staged"</code>	Say what you mean
<code>git config --global alias.last "log -1 HEAD --stat"</code>	Show the last commit
<code>git config --global alias.amend "commit --amend --no-edit"</code>	Add staged changes to the last commit
<code>git config --global pull.ff only</code>	Make <code>git pull</code> refuse anything but a fast-forward (a safe habit)
<code>git config --global push.autoSetupRemote true</code>	First <code>git push</code> sets the upstream for you

I MADE A MISTAKE! WORK DOWN THIS LADDER

1) Is it only an uncommitted edit? `git restore <file>` discards it for good, so be sure. 2) Is it staged by mistake? `git restore --staged <file>`. 3) Is it the last commit and not yet pushed? `git commit --amend` or `git reset --soft HEAD~1`. 4) Already pushed? `git revert <sha>` adds a new commit that undoes it, without rewriting shared history. 5) Think you lost commits? `git reflog` lists where HEAD has been for a long while: pick the hash and `git switch -c rescue <sha>`. Git rarely loses committed work, but it cannot bring back uncommitted changes you discarded.

MASTER OR MAIN? CHECK YOUR DEFAULT BRANCH

Many hosting sites now call the main branch main, but your Git installation's system settings still create master in new repositories (`init.defaultBranch=master`). Either name works, but if you push to a new repository on a site that expects main, rename with `git branch -m main` or set `git config --global init.defaultBranch main` before running `git init`.

NEVER FORCE-PUSH TO A SHARED BRANCH

`git push --force` overwrites the remote branch with yours and can destroy other people's commits. If you must update a branch you have rebased or amended, use `--force-with-lease`, which stops if someone else has pushed since you last fetched, and only do it on a branch that is yours alone.