$ cd reference / Git by task

Git commands by task

Commands are grouped by intent rather than by subcommand. Where two commands do the same job, the newer and narrower one is listed first.

[01] Set up once

TASK	COMMAND
Name for commits	`git config --global user.name "Your Name"`
Email for commits	`git config --global user.email "<email>"`
Default branch for new repositories	`git config --global init.defaultBranch main`
Rebase instead of merge on pull	`git config --global pull.rebase true`
Show the effective setting and where it came from	`git config --show-origin --get pull.rebase`

Settings live in three places: `--system`, `--global` (your user) and the repository's `.git/config`. The most specific one wins.

[02] Start or copy a repository

TASK	COMMAND
New repository in the current directory	`git init`
Copy an existing repository	`git clone <url> [directory]`
Copy only recent history	`git clone --depth 1 <url>`
Add a remote to an existing repository	`git remote add origin <url>`

[03] See what changed

TASK	COMMAND
Short status with branch info	`git status -sb`
Unstaged changes	`git diff`
Staged changes (what will be committed)	`git diff --staged`
Changes in one file against the last commit	`git diff HEAD -- path/to/file`
Word-level diff for prose	`git diff --word-diff`

[04] Stage and commit

TASK	COMMAND
Stage a file	`git add path/to/file`
Stage parts of files interactively	`git add -p`
Unstage, keep the edits	`git restore --staged path/to/file`
Commit staged changes	`git commit -m "Describe the change"`
Fix the last commit (message or content)	`git commit --amend`
Add forgotten changes to the last commit, same message	`git commit --amend --no-edit`

Amend only commits you have not pushed. Amending rewrites the commit id, and anyone who already pulled the old one will see diverged history.

[05] Branches

TASK	COMMAND
List local and remote branches	`git branch -a`
Create a branch and switch to it	`git switch -c feature/x`
Switch to an existing branch	`git switch main`
Rename the current branch	`git branch -m new-name`
Delete a merged branch	`git branch -d feature/x`
Merge a branch into the current one	`git merge feature/x`
Replay the current branch on top of main	`git rebase main`
Squash or reorder the last 4 commits	`git rebase -i HEAD~4`

`git switch` and `git restore` split the two jobs of the older `git checkout`: changing branches and restoring files. `checkout` still works, but a typo in it can overwrite a file when you meant to change branch.

[06] Work with a remote

TASK	COMMAND
Download without changing your branches	`git fetch --prune`
Fetch and integrate the upstream branch	`git pull`

TASK	COMMAND
Push and set the upstream the first time	`git push -u origin feature/x`
Push after a rebase, refusing to overwrite others' work	`git push --force-with-lease`
Show remotes and their URLs	`git remote -v`
Delete a remote branch	`git push origin --delete feature/x`

Prefer `--force-with-lease` over `--force`. It refuses the push if the remote branch moved since your last fetch, so a colleague's commit is not silently dropped.

[07] Read history

TASK	COMMAND
Compact graph of all branches	`git log --oneline --graph --all`
Commits touching one file, following renames	`git log --follow -p -- path/to/file`
Who last changed each line	`git blame -w path/to/file`
Commits whose diff adds or removes a string	`git log -S "functionName" --oneline`
Show one commit	`git show <commit>`
Commits on this branch that main does not have	`git log main..HEAD --oneline`

[08] Undo things

SITUATION	COMMAND	EFFECT
Discard unstaged edits to a file	`git restore path/to/file`	edits are gone for good
Undo the last commit, keep changes staged	`git reset --soft HEAD~1`	rewrites history
Undo the last commit, keep changes unstaged	`git reset HEAD~1`	rewrites history
Throw away the last commit and its changes	`git reset --hard HEAD~1`	rewrites history, deletes work
Undo a pushed commit	`git revert <commit>`	new commit that reverses it
Find a commit you lost with reset or rebase	`git reflog`	read-only

SITUATION	COMMAND	EFFECT
Abort a merge or rebase in progress	`git merge --abort` / `git rebase --abort`	back to the state before

`git reflog` records where `HEAD` has pointed. Entries are kept for 90 days by default (`gc.reflogExpire`), or 30 days once the commit is no longer reachable from any branch (`gc.reflogExpireUnreachable`), and are pruned when `git gc` or `git reflog expire` runs. After a bad reset, `git reset --hard HEAD@{1}` usually brings the previous state back. Uncommitted work that `--hard` discarded is not in the reflog.

[09] Put work aside

TASK	COMMAND
Stash tracked changes	`git stash push -m "wip: parser"`
Stash including untracked files	`git stash push -u`
List stashes	`git stash list`
Apply the latest stash and drop it	`git stash pop`
Apply a specific stash, keep it	`git stash apply stash@{2}`

[10] Find the commit that broke something

```
git bisect start
git bisect bad                # current commit is broken
git bisect good v1.4.0        # this tag was fine
# test, then mark each step:
git bisect good   # or: git bisect bad
git bisect reset              # when done
```

With a script that exits 0 for good and non-zero for bad, `git bisect run ./test.sh` does the stepping for you. Exit code 125 means "cannot test this commit, skip it".

[11] Clean up

TASK	COMMAND
Show untracked files that would be deleted	`git clean -n`
Delete untracked files and directories	`git clean -fd`
Also delete ignored files (build output)	`git clean -fdx`

TASK	COMMAND
Stop tracking a file but keep it on disk	`git rm --cached path/to/file`

Always run `git clean -n` first. Files deleted by `clean` were never in git, so there is nothing to recover them from.

[12] Sources

[1] git-reflog documentation

[2] git-config: gc.reflogExpire and gc.reflogExpireUnreachable

[3] git-push: --force-with-lease

[4] git-bisect: bisect run and exit code 125

[5] git-switch

[6] git-restore