

Git & GitHub

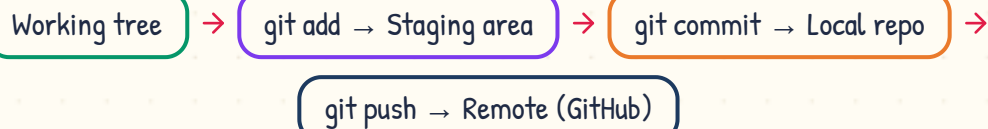
Commits, branches, merge vs rebase, undoing, team workflow

1

How Git thinks

Definition

Git is a distributed version control system. A commit is a **snapshot of the project + a pointer to its parent(s)**; a branch is just a movable pointer to a commit.



Everyday commands

```
git status                # what changed?
git add -p                # stage chunks interactively
git commit -m "Add login form validation"
git switch -c feature/login # new branch
git pull --rebase          # update without merge bubbles
git push -u origin feature/login
git log --oneline --graph --all
git diff / git diff --staged
```

2

Merge vs rebase

	<i>merge</i>	<i>rebase</i>
History	keeps the true history + a merge commit	rewrites commits on top of the target → linear
Safe on shared branches?	yes	no — never rebase commits others have pulled
Conflicts	resolved once	possibly per commit

- ✎ Squash merge: the whole PR becomes one commit on main (clean history).
- ✎ Resolve conflicts: edit the marked files, `git add`, then `git merge --continue` / `git rebase --continue`.
- ✎ Golden rule: rebase your own local work; merge shared branches.

3

Undoing things

Situation	Command
Discard changes to a file	<code>git restore file</code>
Unstage a file	<code>git restore --staged file</code>
Fix the last commit message / add a file	<code>git commit --amend</code> (only if not pushed)
Undo a pushed commit safely	<code>git revert <sha></code> (new commit that reverses it)
Move the branch back, keep changes	<code>git reset --soft HEAD~1</code>
Throw local commits away	<code>git reset --hard <sha></code> (destructive)
Find "lost" commits	<code>git reflog</code>
Park work in progress	<code>git stash</code> / <code>git stash pop</code>
Copy one commit to another branch	<code>git cherry-pick <sha></code>
Find the commit that broke something	<code>git bisect</code>

Common mistake

`git push --force` on a shared branch overwrites teammates' work. Use `--force-with-lease`, and only on your own branches.

4

Team workflow on GitHub

1. Create a short-lived feature branch from `main`.
 2. Commit small, focused changes with clear messages (Conventional Commits: `feat:`, `fix:`, `chore:`).
 3. Open a pull request; CI runs lint, tests and build.
 4. Review, address comments, keep the branch updated.
 5. Squash-merge, delete the branch, deploy.
- ✎ Protect `main`: required reviews and passing checks.
 - ✎ `.gitignore` build output, `node_modules`, `.env`.
 - ✎ Tags (`v1.4.0`) mark releases; semantic versioning.
 - ✎ Never commit secrets – if you do, rotate them (history keeps them).

Revision

Commit = snapshot, branch = pointer. Merge shared branches, rebase local ones. revert for pushed commits, reset for local ones, reflog to recover. Small PRs, protected main, CI on every PR.

Q. git merge vs git rebase?

A. Merge joins histories with a merge commit and keeps them as they were; rebase replays your commits on top of another branch for a linear history but rewrites commit ids.

Q. git reset vs git revert?

A. reset moves the branch pointer (rewrites history, for local work); revert adds a new commit that undoes an earlier one (safe for pushed commits).

Q. git fetch vs git pull?

A. fetch downloads remote changes without touching your branch; pull = fetch + merge (or rebase).

Q. What is the staging area?

A. The index where you prepare exactly what goes into the next commit.

Q. How do you recover a commit after a bad reset?

A. Find its hash in git reflog and reset or cherry-pick it back.

Q. What does git stash do?

A. Saves uncommitted changes and cleans the working tree so you can switch tasks, then restores them with stash pop.

Q. What is cherry-pick?

A. Applying the changes of one specific commit onto the current branch.

Q. How do you resolve a merge conflict?

A. Edit the conflicted files to the right content, remove the markers, git add them, then continue the merge or rebase.

Q. Why is force-pushing dangerous?

A. It overwrites the remote history, deleting commits others may have pulled; use --force-with-lease and only on your own branches.