

Git Interview Questions and Answers

Intern to senior level

Git interview questions and answers: Git vs GitHub, fetch vs pull, merge vs rebase, resolving merge conflicts and reset vs revert, with commands and examples.

5 questions Read online: <https://www.fredevtemplates.com/interview-questions/git>

1. What is Git and how is it different from GitHub?

Git Easy Intern, Junior

Git is a distributed version control system that tracks changes to files on your machine; GitHub is a hosting service for Git repositories that adds collaboration features such as pull requests, code review and CI.

Git records snapshots of a project as commits. Every clone holds the full history, so you can commit, branch, view logs and compare versions without a network connection. Branches are cheap pointers to commits, which is what makes feature branches and experiments practical.

GitHub, GitLab and Bitbucket host a copy of the repository on a server so a team can share it. On top of Git they add pull requests, code review, issue tracking, permissions and automation such as GitHub Actions. You can use Git without GitHub, but not the other way round.

In day-to-day work you commit locally with Git, push your branch to the hosted remote, open a pull request for review, and merge it once checks pass.

In the interview

Git is the version control tool that runs locally and stores the full history in every clone. GitHub is a service that hosts Git repositories and adds pull requests, reviews, issues and CI on top. I use Git to commit and branch, and GitHub to share work and get it reviewed.

Key points

- Git: the version control tool, works offline
- GitHub: a hosting and collaboration platform for Git repositories
- Every Git clone contains the full history

```
git init # start tracking a project
git add src/app.js # stage a change
git commit -m "Add login form"
git remote add origin https://github.com/you/project.git
git push -u origin main # share it on GitHub
```

2. What is the difference between git fetch and git pull?

Git Easy Intern, Junior

git fetch downloads new commits from the remote without changing your working branch; git pull runs a fetch and then merges (or rebases) those commits into your current branch.

git fetch updates your remote-tracking branches such as origin/main. Your own branch and working files are untouched, so it is always safe. You can then inspect what changed with `git log main..origin/main` before deciding how to integrate it.

git pull is git fetch followed by git merge by default. If your branch and the remote have both moved, it creates a merge commit or stops with conflicts. Many teams set `git config pull.rebase true` so pull replays your local commits on top of the remote instead, which keeps history linear.

A good habit before starting work is to fetch, review, then merge or rebase deliberately, rather than pulling blindly into a branch with uncommitted changes.

Key points

- fetch: download only, never touches your branch
- pull: fetch + merge (or rebase with --rebase)
- Fetch first when you want to review incoming changes

```
git fetch origin
git log --oneline main..origin/main # what is new on the remote?
git merge origin/main # integrate when ready

# or in one step
git pull --rebase origin main
```

3. What is the difference between git merge and git rebase?

Git Medium Junior, Mid-level

Both integrate changes from one branch into another: merge keeps both histories and joins them with a merge commit, while rebase rewrites your commits so they appear on top of the target branch, giving a linear history.

Merging main into a feature branch creates a merge commit with two parents. Nothing is rewritten, so it is safe on shared branches, and the history shows exactly when branches were combined. The cost is a busier history with many merge commits.

Rebasing a feature branch onto main takes each of your commits and replays it on the latest main, creating new commits with new hashes. The result reads as if you started your work from the current main, which makes git log and git bisect easier to follow.

Because rebase rewrites history, the golden rule is: never rebase commits that other people have already based work on. Rebase your private feature branch to tidy it up, then merge it (often with a squash or fast-forward) into the shared branch.

In the interview

Merge combines two branches with a merge commit and never rewrites history, so it is safe on shared branches. Rebase replays my commits on top of the target branch, which gives a clean linear history but changes commit hashes. I rebase my own feature branch to keep it current and tidy, and I never rebase a branch someone else is working on.

Key points

- Merge preserves history and is safe on shared branches
- Rebase rewrites commits to produce a linear history
- Never rebase commits that others already use
- Use --force-with-lease, not --force, after rebasing

```
# merge: keeps history, adds a merge commit
git switch feature
git merge main

# rebase: replays feature commits on top of main
git switch feature
git rebase main
git push --force-with-lease # needed because hashes changed
```

4. How do you resolve a merge conflict in Git?

Git Medium Junior, Mid-level

Open the conflicted files, decide what the final code should be between the conflict markers, remove the markers, run the tests, then stage the files and continue the merge or rebase.

A conflict happens when two branches change the same lines, or one deletes a file the other edited. Git pauses the merge and marks each conflicted region with <<<<<<, ===== and >>>>>>. git status lists the files that need attention.

Resolving is not choosing a side. Read both changes, understand why each was made, and write the code that keeps the intent of both. Editors and git mergetool show the two versions side by side, but the decision is yours.

After editing, build and run the tests, because a conflict-free file can still be logically broken. Then git add the resolved files and run git merge --continue or git rebase --continue. If it goes wrong, git merge --abort returns you to the state before the merge.

Key points

- git status shows conflicted files
- Combine the intent of both changes, then remove the markers
- Run tests before continuing
- git merge --abort or git rebase --abort to start over

After editing: git add config.js && git merge --continue

```
<<<<<< HEAD
const timeout = 5000;
=====
const timeout = Number(process.env.TIMEOUT_MS);
>>>>>> feature/config

# resolved: keep the env variable, fall back to the old default
const timeout = Number(process.env.TIMEOUT_MS ?? 5000);
```

5. What is the difference between git reset and git revert?

Git Medium Mid-level, Senior

git revert creates a new commit that undoes an earlier one and is safe on shared branches; git reset moves the branch pointer back to an earlier commit, effectively removing later commits from that branch's history.

git revert <commit> calculates the inverse of a commit and records it as a new commit. History only grows, so teammates who already pulled are not affected. This is how you undo a bad change that has reached main or production.

git reset moves the current branch to another commit. With --soft the undone changes stay staged, with the default --mixed they stay in your working tree unstaged, and with --hard they are discarded. It is ideal for cleaning up local commits that have not been pushed.

If you reset by mistake, the commits are usually still recoverable for a while through git reflog, which records where your branch pointed before each move.

Key points

- revert: new commit, history preserved, safe when shared
- reset: moves the branch pointer, rewrites history
- --soft keeps changes staged, --mixed unstaged, --hard discards them
- git reflog can recover "lost" commits

```
# undo a pushed commit safely
git revert a1b2c3d

# rewrite local history (not yet pushed)
git reset --soft HEAD~1 # undo commit, keep changes staged
git reset --hard HEAD~1 # undo commit and discard changes

# recover from a bad reset
git reflog
git reset --hard HEAD@{2}
```

These answers are written for real interview conversations. Prefer explaining a trade-off out loud over reciting a definition. Full explanations and related questions: <https://www.fredevtemplates.com/interview-questions/git>