# Pull Requests

What a Pull Request is, how to create/review/approve one through a realistic feature branch workflow, the difference between merge and squash-and-merge, deleting branches after merge, and branch protection rules.

**Intermediate** 30 min

Last updated: August 27, 2026
Read online: https://www.learnforgex.com/en/topics/pull-requests

---

Pushing a branch to GitHub gets your code onto the shared server — it doesn't get it into `main`. That gap is intentional: professional teams almost never merge straight into `main` without another person reviewing the change first. The Pull Request is GitHub's mechanism for that review-then-merge process, and it's the single most common way real Java/Spring Boot teams collaborate day to day.

## What Is a Pull Request?

A **Pull Request** (PR) is a GitHub feature that proposes merging one branch into another (typically your feature branch into `main`) and opens a dedicated page for discussing, reviewing, and eventually approving that specific set of changes before it actually gets merged. Despite the name, it doesn't "pull" anything by itself — it's a request for someone to review and merge your branch.

## Feature Branch Workflow

Pull Requests are the natural continuation of the feature-branch workflow from the "Branches & Merging" lesson — the only thing that changes is *how* the branch gets merged into `main`:

```
Without a PR:  branch → git merge (locally, no review)
With a PR:     branch → push to GitHub → open PR → review → merge (on GitHub)
```

The branch, commit, and push steps are identical to what you already know — a PR simply replaces the local `git merge` step with a reviewed, GitHub-hosted merge.

## Creating a Pull Request

Once your feature branch is pushed to GitHub:

```
git push -u origin add-password-reset
```

GitHub will typically show a banner offering to open a PR for the branch you just pushed. Creating one requires: the source branch (yours), the target branch (usually `main`), a title, and a description explaining *what* the change does and *why* — the PR description is where reviewers form their first impression, so treat it with the same care as a good commit message.

## Reviewing a Pull Request

Reviewing means reading the proposed diff (GitHub shows it file-by-file, exactly like `git diff`) and leaving feedback. A reviewer can leave inline comments on specific lines, general comments on the PR as a whole, and ultimately choose one of three review outcomes: comment only, request changes, or approve.

## Requesting Changes

If the reviewer finds a real problem — a bug, a missing test, an approach they disagree with — they submit their review as **"Request changes."** This is a formal signal (distinct from a casual comment) that, on many projects, actually blocks the PR from being merged until it's resolved. The author then pushes new commits to the same branch; GitHub automatically updates the PR with them, no new PR needed.

## Approving a Pull Request

Once the reviewer is satisfied, they submit their review as **"Approve."** On most professional teams, at least one approval is required before the merge button becomes available (enforced by branch protection rules, covered below) — this is the actual moment a second pair of eyes has signed off on the change reaching `main`.

## Merging a Pull Request

Once approved, GitHub's merge button performs the merge on the server, exactly as `git merge` would locally — the difference is *when* it happens (after review, not before) and *where* (GitHub, visible to the whole team, with a permanent record of who reviewed and approved it).

## Merge vs Squash and Merge

GitHub actually offers a choice of merge strategies on that button:

- **Create a merge commit** — behaves exactly like the `git merge` you already know: keeps every individual commit from the branch, plus a merge commit tying them together.
- **Squash and merge** — combines *every* commit on the branch into a single new commit on `main`. This is especially useful for feature branches with a messy history ("fix typo", "fix typo again", "actually fix it") — the branch's `main`-facing history becomes one clean commit, even though the individual work-in-progress commits still existed along the way. We cover squashing (including the manual `git rebase -i` version of this) in its own dedicated lesson later in this course.

## Deleting Branches After Merge

Once a PR is merged, its branch has served its purpose — GitHub offers a one-click "Delete branch" button right on the merged PR page. This is the remote equivalent of the `git branch -d` cleanup you already know from local branches, and just as safe: the commits live on permanently in `main`'s history.

## Branch Protection Rules

**Branch protection rules** are settings a repository admin configures on GitHub to enforce these practices automatically rather than relying on everyone remembering to follow them — for example: require at least one approval before merging, require the branch to be up to date with `main` first, or block direct pushes to `main` entirely (forcing *everyone*, even admins, to go through a PR). On professional teams, `main` is almost always protected this way.

## Common Mistakes

- **Opening a PR with a vague title and no description**, leaving the reviewer to reverse-engineer *why* the change exists from the diff alone.
- **Merging your own PR without waiting for review** on a team that expects one — branch protection rules exist specifically to prevent this from being possible by accident.
- **Force-pushing over a branch mid-review** without warning the reviewer — this rewrites the exact commits they were looking at (we cover why this is risky in detail in the Force Push lesson).

## Best Practices

- Keep PRs small and focused on one thing — smaller PRs get reviewed faster and more thoroughly.
- Write a description that explains *why*, the same discipline as a good commit message.
- Respond to review comments by pushing new commits, not by silently editing and force-pushing over what the reviewer already saw.

# Summary, Cheat Sheet, and Glossary

### Summary

- A Pull Request proposes merging one branch into another and provides a dedicated space for review before the merge happens.
- Reviews resolve to comment / request changes / approve — approval is usually required before merging.
- "Create a merge commit" preserves every commit; "Squash and merge" combines them into one clean commit on `main`.
- Branch protection rules enforce review requirements automatically, rather than relying on discipline alone.

### Cheat Sheet

```
git push -u origin <branch>      # push your branch, then open a PR on GitHub
# ... review happens on GitHub: comment / request changes / approve ...
# ... merge on GitHub: "Create a merge commit" or "Squash and merge" ...
git branch -d <branch>           # clean up your local copy after merge
```

### Glossary

- **Pull Request (PR)** — a GitHub proposal to merge one branch into another, with a dedicated review space.
- **Request changes** — a formal review outcome that (on protected branches) blocks merging until resolved.
- **Branch protection rule** — a repository setting that enforces requirements like required approvals before allowing a merge.