Git Pull Requests & Team Flow

What is a Git pull request? Step by step: fork, clone, branch, commit, push, open the PR, review, merge.

Introduction

On a real team, nobody pushes straight to `main`. Instead, every change travels as a **pull request (PR)**: a formal proposal that says "here is my branch — please review these exact changes and merge them if they are good." Teammates read the **diff** line by line, leave comments, request fixes, and only then does someone approve and merge. This lesson walks one change through the whole journey — branch, push, PR, review, fix, re-push, merge — the exact flow you will meet in your first job. *(Roman Urdu: Pull request aik darkhwast hai — "meri branch ke changes dekh lo, theek hon to main mein mila do." Team ka kaam isi se chalta hai.)*

How to create a pull request — the full steps

1. **Fork the repo** — only if it is not yours and you cannot push to it. The Fork button on GitHub copies it to your account.
2. **Clone your copy** — `git clone <your-fork-url>` downloads it to your computer.
3. **Make a branch** — `git switch -c fix-header-mobile`. Never work directly on `main`.
4. **Commit your change** — `git add .` then `git commit -m "clear message"`.
5. **Push the branch** — `git push -u origin fix-header-mobile`. GitHub prints a "Create a pull request" link in the output.
6. **Open the pull request** — click that link (or the yellow "Compare & pull request" banner). Check base = `main` and compare = your branch, write a title and a two-line description, then **Create pull request**.
7. **Handle the review** — reviewers comment on lines; fix on the SAME branch and push again. The PR updates itself.
8. **Merge and clean up** — after approval and green checks, merge, delete the branch, and `git pull` on main.

Key points

- **A PR compares two branches:** base (`main` — the branch that receives) and compare (your feature branch). It shows commits + the total diff, and it updates live as you push more commits to the same branch.
- **The review is the point, not the paperwork:** reviewers catch bugs, style drift and missing tests BEFORE the code joins main. A comment on a line is a conversation anchored to that exact line.

  (Roman Urdu: PR ka asal faida review hai — ghalti main mein jane se pehle pakri jati hai.)

- **Fixes are just more pushes:** when a reviewer requests changes, you do NOT open a new PR. Commit the fix on the same branch and push — the PR gains the new commit and re-runs its checks automatically.
- **Checks guard the door:** many repos run tests on every PR (green ticks / red crosses on the PR page). A red check means "not ready", even if a human already approved.
- **Merge completes the story:** the green Merge button folds the branch into main; then the branch is deleted (locally and on GitHub) and everyone pulls the updated main. Small PRs — one purpose, a few files — get reviewed fastest.

Code examples

Example 1 — Push the branch; GitHub invites you to open the PR

```
$ git switch -c fix-header-mobile
Switched to a new branch 'fix-header-mobile'
$ git commit -am "Mobile par header ka size theek kiya"
[fix-header-mobile c3d4e5f] Mobile par header ka size theek kiya
 1 file changed, 2 insertions(+), 2 deletions(-)
$ git push -u origin fix-header-mobile
Enumerating objects: 5, done.
Writing objects: 100% (3/3), 342 bytes | 342.00 KiB/s, done.
To https://github.com/ayesha-khan/notes-app.git
 * [new branch]      fix-header-mobile -> fix-header-mobile
remote: Create a pull request for 'fix-header-mobile' on GitHub by visiting:
remote:      https://github.com/ayesha-khan/notes-app/pull/new/fix-header-mobile
```

What it does: the fix is committed on its own branch (never directly on main) and pushed. GitHub notices the new branch and prints the PR link right in the push output — and shows a yellow "Compare & pull request" banner on the repo page.

What you will see: opening that link gives the PR form: base = `main`, compare = `fix-header-mobile`, a title (prefilled from your commit), a description box, and the full diff below — every changed line, red for removed, green for added. You write two honest lines ("What: header shrink on small screens. Tested: 360px and 390px widths.") and click **Create pull request**.

Example 2 — The reviewer's comment becomes a fix on the same branch

```
$ git switch fix-header-mobile
Switched to branch 'fix-header-mobile'
Your branch is up to date with 'origin/fix-header-mobile'.
$ git commit -am "Review fix: logo 24px rakha, margin bhi theek"
[fix-header-mobile d4e5f6g] Review fix: logo 24px rakha, margin bhi theek
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git push
To https://github.com/ayesha-khan/notes-app.git
   c3d4e5f..d4e5f6g  fix-header-mobile -> fix-header-mobile
```

What it does: the reviewer commented on one line: "Logo is still 32px — the issue said 24px." You switch to the same branch, fix it, commit with a message that names the review, and push. No new PR, no new branch.

What you will see: the PR page instantly gains the new commit, the diff updates to the fixed code, the conversation shows your reply next to the original comment, and any automated checks re-run. The reviewer gets notified, re-reads just the new change, and approves.

Example 3 — Merge, clean up, and bring main home

```
$ git switch main
Switched to branch 'main'
$ git pull
Updating 7c8d9e0..d4e5f6g
Fast-forward
 style.css | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
$ git branch -d fix-header-mobile
Deleted branch fix-header-mobile (was d4e5f6g).
$ git push origin --delete fix-header-mobile
To https://github.com/ayesha-khan/notes-app.git
 - [deleted]         fix-header-mobile
```

What it does: after approval, the Merge button on the PR folds the branch into `main` on GitHub. Back locally: switch to main, `git pull` the merged result, delete the finished branch locally, and delete its remote copy — a tidy repo has no graveyard of dead branches.

What you will see: the PR page shows the purple **Merged** label with the full conversation preserved forever, and your local `main` now contains the fix — the complete professional loop, start to finish.

Quick recap

- One change = one branch = one PR. The PR is a living proposal: push more commits and it updates.
- Review happens on the diff, line by line; fixes are follow-up pushes to the same branch, never a new PR.

- Green checks + an approval → merge → delete the branch everywhere → `git pull` on main. Small, single-purpose PRs move fastest.

Common mistakes

1. **The giant PR** — three features and a refactor in one pull request; nobody can review it, so it sits for days. The fix: one purpose per PR; if the diff passes a few files, split it — two small PRs merge faster than one big one.
2. **Opening a second PR for the review fixes** — now two half-stories confuse everyone. The fix: fixes go on the SAME branch with a normal push; the original PR is the one story of that change.
3. **Merging with red checks or unresolved comments** — "it probably works" is how bugs reach main. The fix: red check = not ready, every conversation resolved or consciously deferred in a comment — then merge.

Frequently asked questions

What is a pull request? A pull request is a proposal to merge one branch into another, shown as a reviewable diff. Teammates read the changes, comment, and approve — only then does the code join `main`.

Do I need to fork to open a pull request? Only for repos you cannot push to, like public open-source projects. On your own repo or your team's repo, just make a branch and open the PR from it — no fork needed.

What is the difference between a pull request and a merge request? They are the same idea with different names. GitHub calls it a pull request; GitLab calls it a merge request.

What happens when a reviewer asks for changes? You fix the code on the same branch, commit, and push again. The same pull request gains the new commit and updates itself — never open a second PR for review fixes.

What if the checks are red? Red means not ready, even if a human already approved. Fix the failing test or problem, push again, and wait for green before merging.

Practice

1. On your own GitHub repo (solo is fine — you can review your own PR), make a branch `add-about-page`, commit a small page, push it, and open a PR with a two-line description: what changed, how you tested it.
2. Play reviewer: open the PR's **Files changed** tab, leave one honest line-comment on any line (even "rename this to be clearer"), then push a fix commit from the terminal and watch the PR update itself. Merge it, delete the branch, and `git pull` on main.

Practice quiz

1. In a pull request, what are the base and compare branches?

- Base is your branch, compare is main

- Base is main, the branch that receives; compare is your feature branch; the PR shows commits plus the total diff
- Both are main
- Base is GitHub, compare is your computer

Answer: Base is main, the branch that receives; compare is your feature branch; the PR shows commits plus the total diff

2. A reviewer requests changes on your PR. What do you do?

- Open a new PR for the fixes
- Commit the fix on the same branch and push; the PR gains the new commit and re-runs its checks automatically
- Close the PR and start over
- Merge it anyway, fixes can wait

Answer: Commit the fix on the same branch and push; the PR gains the new commit and re-runs its checks automatically

3. When is a PR ready to merge?

- As soon as it is opened
- When the checks are green and it has an approval, then merge, delete the branch everywhere, and pull the updated main
- When it has red checks but a human approved
- Only after a week

Answer: When the checks are green and it has an approval, then merge, delete the branch everywhere, and pull the updated main

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/git-pull-requests · More free tools and notes at https://scopetreasure.com