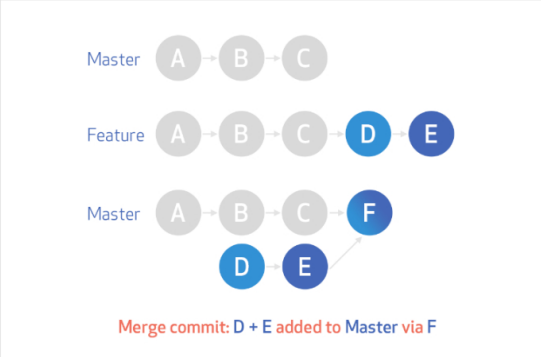
About pull request merges

You can merge pull requests by retaining all the commits in a feature branch, squashing all commits into a single commit, or by rebasing individual commits from the `head` branch onto the `base` branch.

In this article

When you click the default **Merge pull request** option on a pull request on GitHub, all commits from the feature branch are added to the base branch in a merge commit. The pull request is merged using the `--no-ff` option.

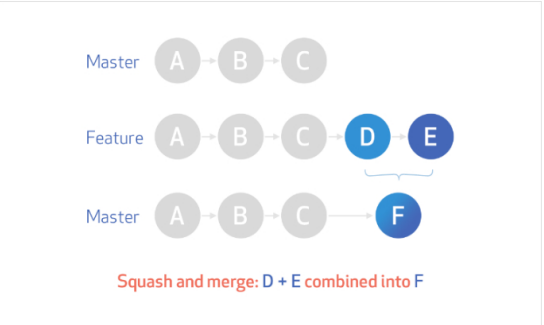
To merge pull requests, you must have write permissions in the repository.



Squash and merge your pull request commits

When you select the **Squash and merge** option on a pull request on GitHub, the pull request's commits are squashed into a single commit. Instead of seeing all of a contributor's individual commits from a topic branch, the commits are combined into one commit and merged into the default branch. Pull requests with squashed commits are merged using the fast-forward option.

To squash and merge pull requests, you must have write permissions in the repository, and the repository must allow squash merging.



You can use squash and merge to create a more streamlined Git history in your repository. Work-in-progress commits are helpful when working on a feature branch, but they aren't necessarily important to retain in the Git history. If you squash these commits into one commit while merging to the default branch, you can retain the original changes with a clear Git history.

Rebase and merge your pull request commits

When you select the **Rebase and merge** option on a pull request on GitHub, all commits from the topic branch (or head branch) are added onto the base branch individually without a merge commit. Pull requests with rebased commits are merged using the fast-forward option.

To rebase and merge pull requests, you must have write permissions in the repository, and the repository must allow rebase merging.

The rebase and merge behavior on GitHub deviates slightly from `git rebase`. Rebase and merge on GitHub will always update the committer information and create new commit SHAs, whereas `git rebase` outside of GitHub does not change the committer information when the rebase happens on top of an ancestor commit. For more information about `git rebase`, see the "Git rebase" chapter from the *Pro Git* book.

For a visual representation of `git rebase`, see The "Git Branching - Rebasing" chapter from the *Pro Git* book.

You aren't able to automatically rebase and merge on GitHub when:

- The pull request has merge conflicts.
- Rebasing the commits from the base branch into the head branch runs into conflicts.
- Rebasing the commits is considered "unsafe," such as when a rebase is possible without merge conflicts but would produce a different result than a merge would.

If you still want to rebase the commits but can't rebase and merge automatically on GitHub you must:

- Rebase the topic branch (or head branch) onto the base branch locally on the command line
- Resolve any merge conflicts on the command line.
- Force-push the rebased commits to the pull request's topic branch (or remote head branch).

Anyone with write permissions in the repository, can then merge the changes using the rebase and merge button on GitHub.

Further reading

- "About pull requests"
- "Addressing merge conflicts"