Collaborating with Git and GitHub

 (https://github.com/learn-co-curriculum/git-github-collaborating)  (https://github.com/learn-co-curriculum/git-github-collaborating/issues/new)

Learning Goals

- Explain how to use Git and GitHub collaboratively.
- Push/pull code on a specific branch to/from GitHub.
- Use pull requests to merge work.
- Resolve merge conflicts when working collaboratively.
- Explain ways to avoid conflicts when working collaboratively.
- Use the fork and clone workflow.

Introduction

We've seen how helpful Git can be when we're working on a personal project, but it is even more so when we're working on a project as part of a team. Using branches allows multiple people to work on the same project at the same time without interfering with each other's work. It also makes it easy to share in-progress code with teammates to get help or feedback.

Problems with Single-Branch Workflow

Imagine this scenario: you are on a team with several other people, and all of you are doing work on the `main` branch. You each clone down the repo and do your work on your own machine. One of your teammates pushes up their work and merges it in, so there is now work on the remote repo that you don't have in your local copy. When you later try to push up your code you'll get an error saying there's a conflict. In order to resolve it, you would need to pull down the updated code from the remote repo, merge it into your code (and resolve any conflicts), then push your code up. This would happen each time a team member pushes up their work to `main`, which creates a lot of extra work, as well as a lot of merge commits cluttering up your commit history.

Or let's consider another situation: say one member of the team is having some trouble getting their code working and wants to share it with team members to get help. If everyone is working on `main`, the only option would be for that team member to push up the broken code to GitHub and the others to clone it down. But they couldn't do that without affecting their own work and, if they did, their code would be broken as well.

A much better approach is for each member of the team to create a separate branch for their work. The advantages of this workflow include:

- The code on `main` will be kept working at all times. This is a "golden rule" adopted by many coding teams.
- If a team member needs help with their code, they can push up their branch to GitHub, and another team member can pull it down without affecting their own work on their own branch.
- A feature branch will only need to be merged into `main` once, after the work has been completed and approved.

The good news is that you've already learned a lot of what you need to know in order to work collaboratively: you know how to create and use branches, and how to push your work up to GitHub. The only real difference with working collaboratively is the use of *pull requests* (PRs) to merge code into the main codebase.

Example: Collaborating on a Book

Imagine you are working with some colleagues to write a book. Each member of the team is responsible for specific chapters, and will also provide feedback on their teammates' work. There will be a central repository for our book on GitHub, and each author will clone that project down and create local branches for the chapters they work on.

Let's do some setup to recreate that workflow. In an earlier lesson, we created a GitHub repo by first creating the project locally, then creating a remote repo on GitHub, and finally hooking the two together. This time, to mimic the typical collaborative workflow, we'll create the repo on GitHub first.

To create the repo on GitHub, go to your GitHub account, click "Repositories", then click the green "New" button in the upper right corner of the screen. In the form that appears, give your project a name (we'll call ours `our-great-book`), select the "Add a README file" option, and click the "Create repository" button. We're adding a README because GitHub won't create the repo unless it contains at least one file — it needs something to track.

This repo will serve as the central codebase for our example project.

Collaborative Workflow

For teams collaborating using GitHub, each team member will clone down the repo, create a new branch for the feature they're working on, write the code for that feature, then push the branch up to the GitHub repo to be reviewed or approved.

Let's walk through it using our example.

Example: Working on a feature branch and pushing it to GitHub

Clone the GitHub repo for the project down to your local machine. Create a branch, `chapter-1`, and switch to it.

Next we'll add some content. Create a file called "chapter-1.md" and open it in VS Code. Verify that the `chapter-1` branch is shown in the bottom left corner of the window, then add some text. You can add anything you like. One option is to use a **lorum ipsum (https://en.wikipedia.org/wiki/Lorem_ipsum)** generator such as **fillerama.io (http://fillerama.io/)**, which has markdown as one of its format options.

Tip: Remember to keep your content appropriate! Your GitHub account is publicly visible to anyone, including potential employers!

Once you've added some content, commit your changes, making sure you're on the `chapter-1` feature branch.

The next step is to push our branch up to GitHub. Let's run `git remote` to view our remote:

```
$ git remote
origin
```

Recall that, when we clone a repo down from GitHub, the nickname `origin` is automatically created to refer to the remote repo we cloned. To see the url `origin` is pointing to, we use the `v` (for verbose) flag:

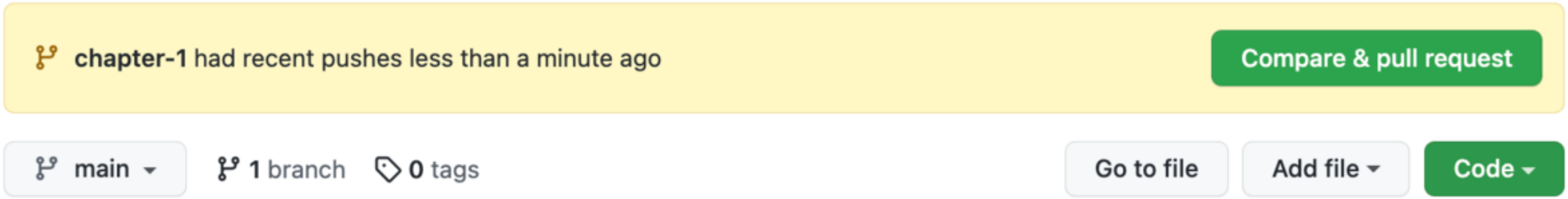
```
$ git remote -v
origin  git@github.com:your-github-account/our-great-book.git (fetch)
origin  git@github.com:your-github-account/our-great-book.git (push)
```

So far, when we've pushed code to GitHub, we've pushed to the `main` branch. In this case, however, we want to push up our feature branch — we're not ready to incorporate our changes into the main branch yet.

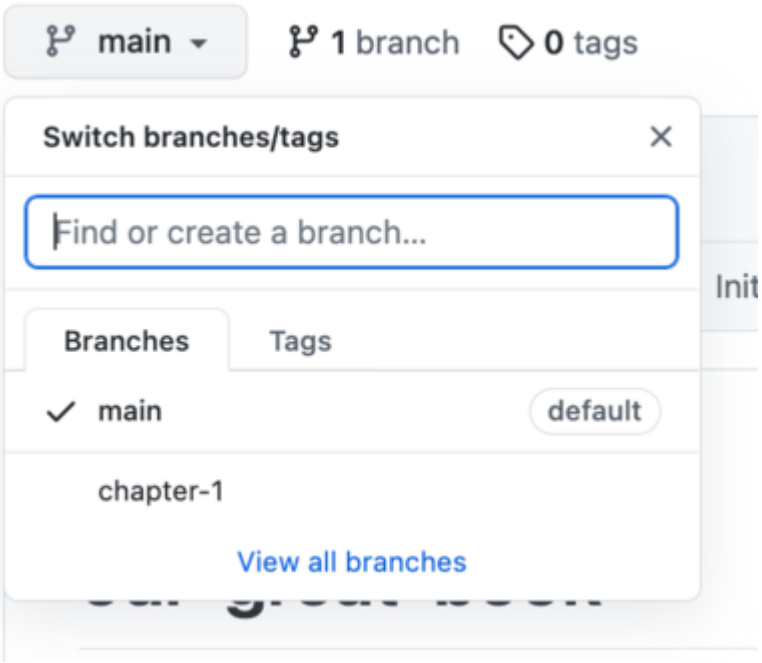
Recall that the command to push to the main branch on `origin` is `git push origin main`. To push a different branch, all we need to do is replace `main` with the branch we want to push:

```
$ git push origin chapter-1
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.94 KiB | 1.94 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'chapter-1' on GitHub by visiting:
remote:      https://github.com/your-github-account/our-great-book/pull/new/chapter-1
remote:
To github.com:your-github-account/our-great-book.git
 * [new branch]      chapter-1 -> chapter-1
```

This command has created the new branch on GitHub and populated it with our content for Chapter 1. Head back to the repo on GitHub. You should see a message at the top of the page indicating that `chapter-1` had recent pushes along with a "Compare & pull request" button:



If you don't see this message, click the branch dropdown and you should see the new branch listed there:



We have successfully pushed up our work on Chapter 1!

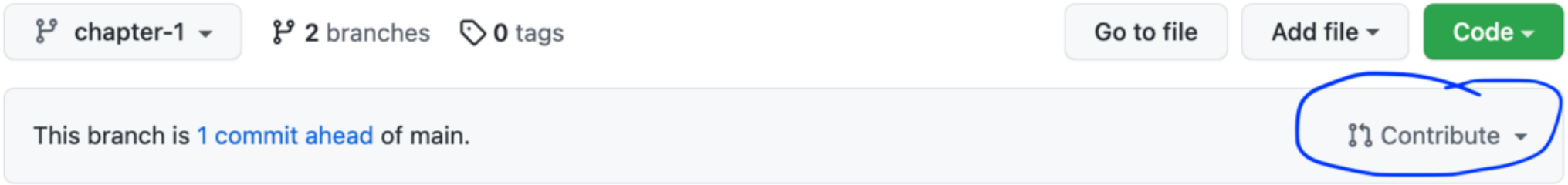
Pull Requests

When we're working on our local machine, we use the `git merge` command to merge in a branch. When we're working collaboratively, however, we may need to get our code reviewed by teammates or approval from our boss before merging it in. GitHub enables us to do this using a ***pull request*** (https://docs.github.com/en/enterprise-server@3.5/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests) (PR), which is simply a request to merge one branch (e.g., a feature branch) into another (e.g., the main branch). Using pull requests has some really nice features for working collaboratively.

Once a pull request has been created, the author can request reviews from others. Reviewers can view the specific changes made (the *diff*) in GitHub, and can comment or request changes. The author can then make any needed changes in the feature branch on their local machine. When they push the branch up again, those changes will be added to the existing pull request. Finally, once the work is approved and ready to go, the pull request is merged in.

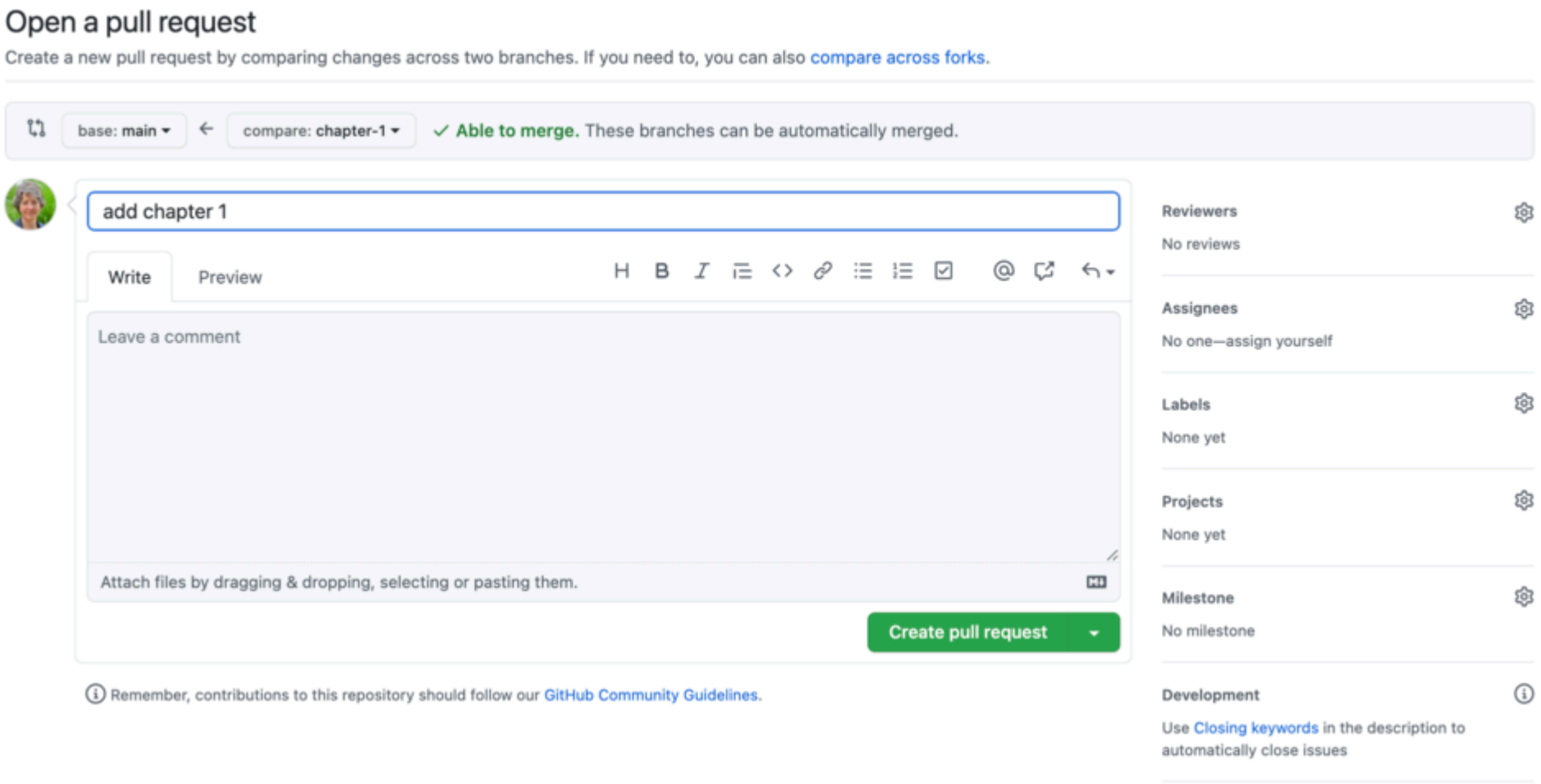
Example: Creating and Merging a Pull Request

Let's go ahead and create a pull request for our `chapter-1` branch. Return to the GitHub repo. If you see the "Compare & pull request" button at the top of the screen, go ahead and click it. Otherwise, click the branch drop-down and select the `chapter-1` branch. You should see a message indicating that the branch is one commit ahead of `main`, and a Contribute drop-down on the right:

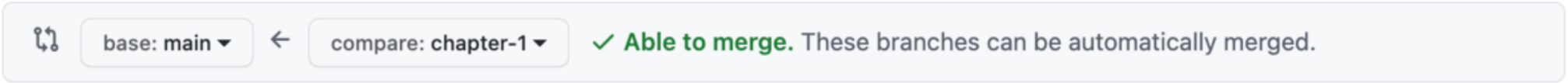


Click the "Contribute" drop-down, then the "Open pull request" button.

You now should see form that allows you to open your pull request:



Here you can change the message associated with the pull request or add a comment if you like. Just above those fields, you'll see the information about the pull request:

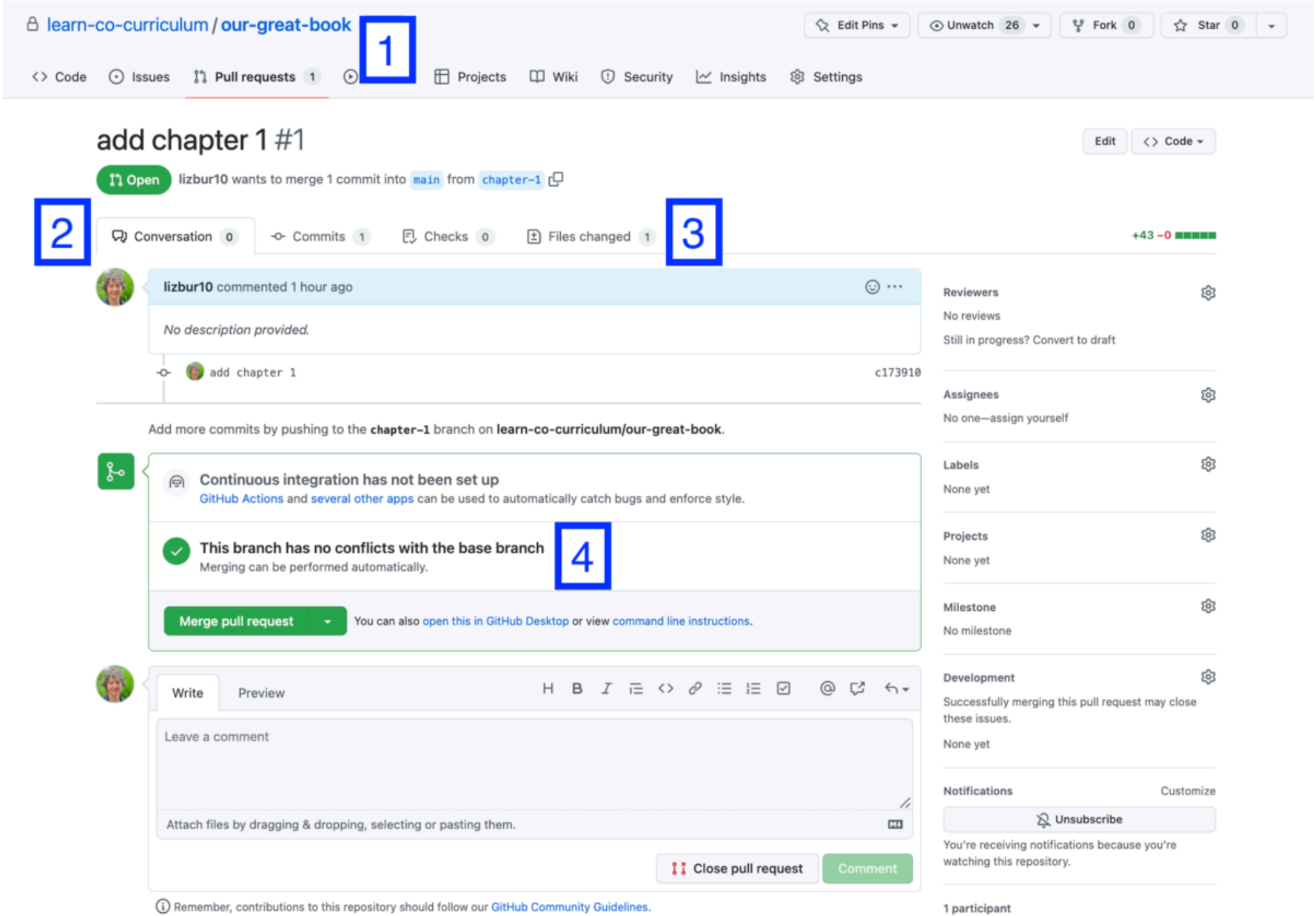


In this case, it's telling us that we are requesting that our feature branch (`chapter-1`) be merged into the main branch; it also indicates that the branches can be merged automatically, i.e., there are no merge conflicts. Note that the branch names here are drop-down menus, so we could

reverse them if we wanted, and merge `main` into `chapter-1`. Or, if we had other branches, we could merge chapter-1 into one of them instead. Neither of those options really makes sense for this example, but you might encounter situations where they do.

On the right side of the screen, you'll see some other options listed, including "Reviewers". If you were working for an organization and the PR had been submitted in the organization's GitHub account, clicking that option would reveal a list of GitHub usernames of other people in the organization. You could select one or more of them to request that they review your pull request.

Let's go ahead and click the "Create pull request" button. This will take you to a page with the information for the pull request:

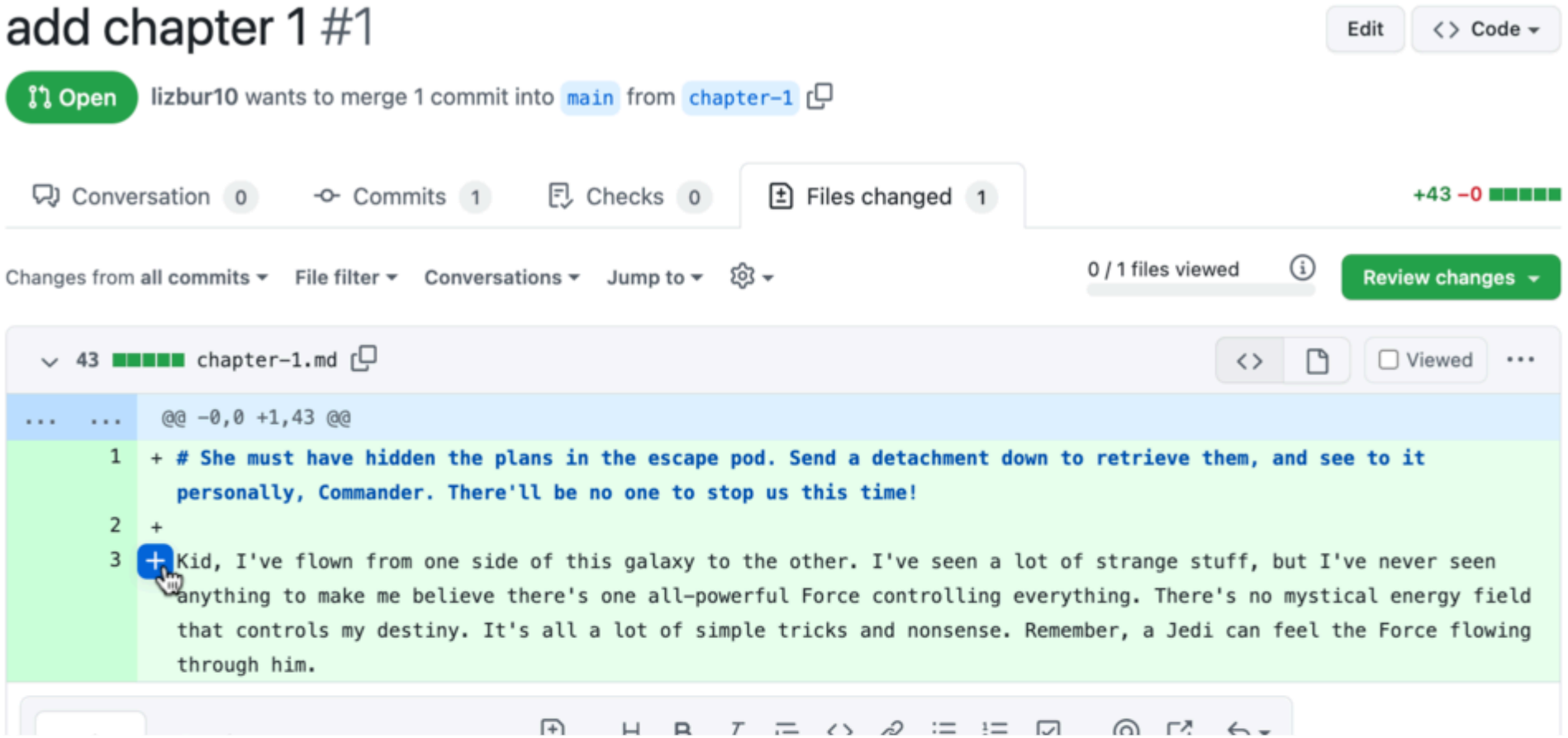

learn-co-curriculum / our-great-book
1
Edit Pins
Unwatch 26
Fork 0
Star 0
Code
Issues
Pull requests 1
Projects
Wiki
Security
Insights
Settings
add chapter 1 #1
Edit
Code
Open
lizbur10 wants to merge 1 commit into main from chapter-1
2
Conversation 0
Commits 1
Checks 0
Files changed 1
3
+43 −0
lizbur10 commented 1 hour ago
No description provided.
add chapter 1
c173910
Add more commits by pushing to the chapter-1 branch on learn-co-curriculum/our-great-book.
Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.
This branch has no conflicts with the base branch
Merging can be performed automatically.
4
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Close pull request
Comment
Remember, contributions to this repository should follow our GitHub Community Guidelines.
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant

There's quite a bit of information on this page; we'll cover a few of the most helpful things:

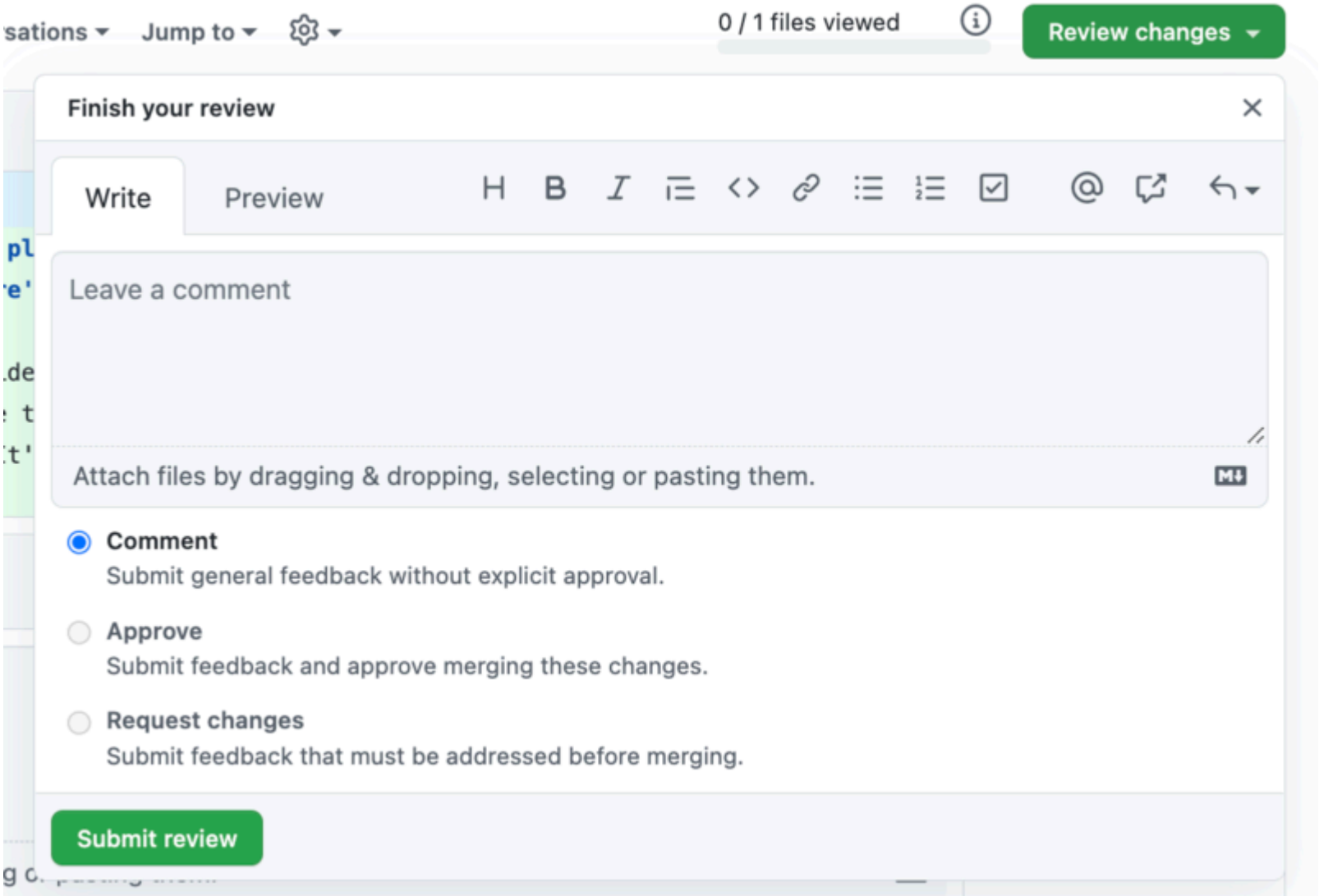
1. At the top of the page, note that the "Pull requests" tab is active. This is where you can go to see all the open (i.e., not yet merged) pull requests.
2. Below that, there is a second set of tabs in which the "Conversation" tab is currently active. This is where you can see the overview of the pull request, as well as any comments that have been added.
3. Another helpful source of information in this area is the "Files changed" tab. You can go here to see exactly which files have been changed as well as the diff for each file. In our case, there's only one file and all we did was add a bunch of content to it, so the information isn't very helpful. We'll see an example of a more informative diff a bit later.
4. Here GitHub tells you whether or not the PR can be merged. If there are merge conflicts, there will be information about how to resolve those conflicts. We'll see an example of that later as well.

Reviewing a Pull Request

Let's take a look at some of the reviewing tools that are available. Click on the "Files changed" tab (#3 in the screenshot above). Here you will see the list of files that have been changed (in our case, just `chapter-1.md`) and the diff for each. Note that, as you move your cursor over the diff, a small plus-sign icon appears at the left edge of whatever line your cursor is over. If you click on that icon, a text box will open where you can make a comment specific to that line in the file:

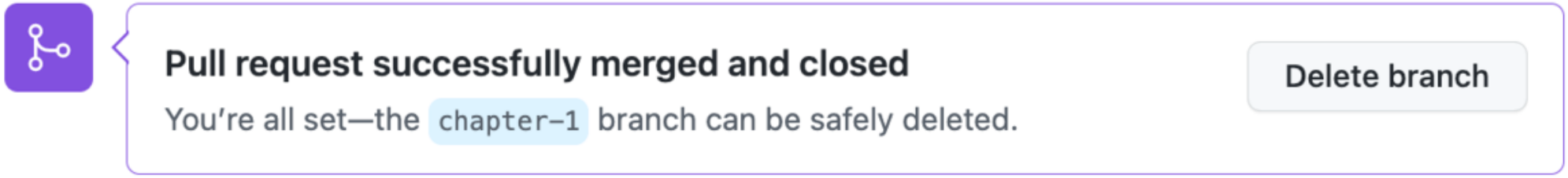

add chapter 1 #1
Edit
Code
Open
lizbur10 wants to merge 1 commit into main from chapter-1
Conversation 0
Commits 1
Checks 0
Files changed 1
+43 −0
Changes from all commits
File filter
Conversations
Jump to
0 / 1 files viewed
Review changes
43 chapter-1.md
Viewed
@@ -0,0 +1,43 @@
1 + # She must have hidden the plans in the escape pod. Send a detachment down to retrieve them, and see to it personally, Commander. There'll be no one to stop us this time!
2 +
3 + Kid, I've flown from one side of this galaxy to the other. I've seen a lot of strange stuff, but I've never seen anything to make me believe there's one all-powerful Force controlling everything. There's no mystical energy field that controls my destiny. It's all a lot of simple tricks and nonsense. Remember, a Jedi can feel the Force flowing through him.

Once you've typed in the comment, you can either add just that single comment, or, if you're going to make additional comments, start a review. You can also click the "Review changes" button in the upper right of the window to leave a general comment, approve the pull request, or request changes:



Merging a Pull Request

Once any comments have been resolved, requested changes made, etc., and the pull request has been approved, you can merge in the branch by going back to the "Conversation" tab and clicking "Merge pull request", then "Confirm merge". Go ahead and do that; you should see a message telling you that the pull request has been successfully merged. There will also be a button you can use to delete the branch you just merged:



Generally, it's a good idea to delete branches that you no longer need. Let's click "Delete branch", then navigate to the "Code" tab at the top of the page. There you'll see that you're on the main branch, and that `chapter-1.md` has been added to the files. If you click on the branch dropdown, you'll see that our `chapter-1` branch is gone.

Pulling Changes from GitHub to Local

We're going to continue working on the `our-great-book` project locally, but there's one more thing we need to do before we're ready to do that. Recall that our local copy of the project currently doesn't match what's stored on GitHub. We merged Chapter 1 into the main codebase on GitHub, but it's still only on the `chapter-1` branch locally. To fix this, do the following:

1. In the terminal, switch to the `main` branch.
2. Run `git pull origin main` (https://www.git-scm.com/docs/git-pull) to pull down the current version. This command fetches down the changes from the remote repo and merges them into the current branch. If you now run `git status`, you will see the following:

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

You should also see in VS Code that we're on the main branch, and that `chapter-1.md` is now present in the file list.

We're finished with the `chapter-1` branch so, to keep things tidy, go ahead and delete it in the terminal:

```
$ git branch -d chapter-1
Deleted branch chapter-1 (was 276fc72).
```

Merging Pull Requests that have Conflicts

As we've seen, the process of creating a pull request and merging it in is fairly straightforward when there are no merge conflicts. Next, we'll take a look at how to handle the situation when we do have conflicts.

Example: Creating a Merge Conflict

For purposes of illustration, as we did in an earlier lesson, we're going to intentionally create a merge conflict in order to simulate the situation in which two members of a team have pushed up conflicting changes to a file. Note that, in doing this, we will depart from best practices in a couple of instances.

Say one of our coauthors is working on Chapter 2. They create a new `chapter-2` branch and add their draft. At the same time (ignoring best practice), Author 2 also make a few edits to Chapter 1. Let's go ahead and do that:

1. Create a new branch, `chapter-2`, and switch to it.
2. Create a `chapter-2.md` file and add some content.
3. Still on the `chapter-2` branch, make a few small edits to the first couple of sentences or paragraphs in `chapter-1.md`. These can be anything you like.
4. When you're done, commit the changes.

Next, we'll make a few edits to Chapter 1 as Author 1. Normally, we would create a new branch to do this but, to keep things simple, we'll make the edits directly on the `main` branch:

1. Switch to the `main` branch; you should see in VS Code that `chapter-2.md` is no longer in the file list.
2. In `chapter-1.md`, make **different** changes to one or two of the pieces of text you edited above.
3. Commit the changes, and push them to GitHub.

Note that the order here is important! To create a merge conflict, we need to make sure that we create the `chapter-2` branch **before** making the edits to Chapter 1 in the main branch. Otherwise the changes we made on `main` will be in `chapter-2` as well, and there will be no conflict.

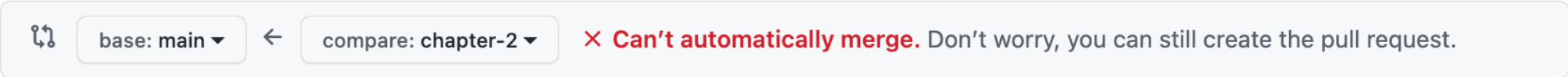
Finally, switch back to the `chapter-2` branch and push it up to GitHub.

We have now created the situation where Author 1 pushed changes up to GitHub **after** Author 2 created the `chapter-2` branch off of `main`. In other words, there are changes in `chapter-1.md` on GitHub that aren't reflected in Author 2's `chapter-2` branch.

Create a Pull Request

Next, we'll create a pull request. The instructions are the same as those given earlier, except now we're making the request for the `chapter-2` branch instead of `chapter-1`.

When you get to the form that enables you to create the pull request, you should see a message indicating that the branch can't be automatically merged:

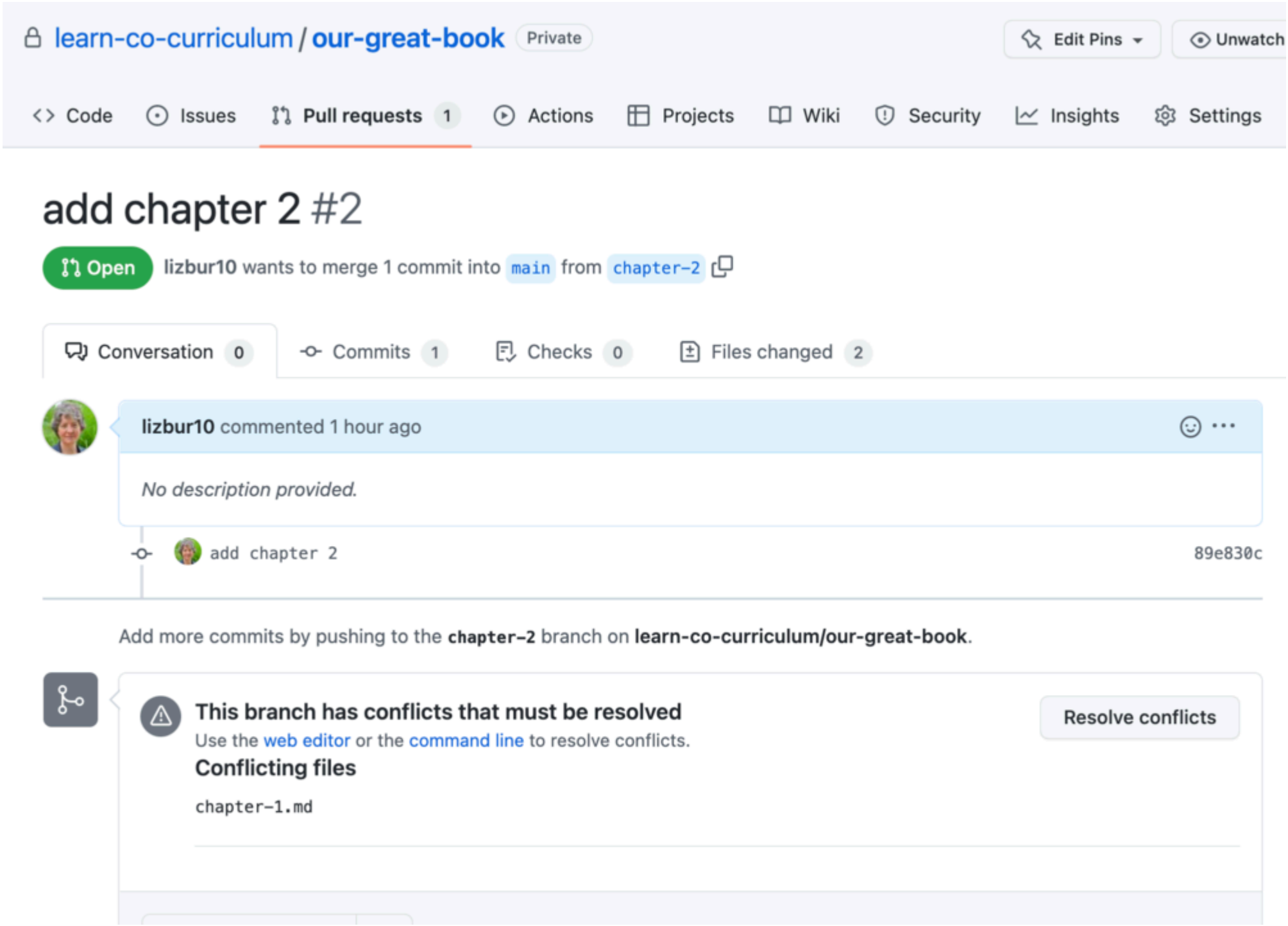


Note: if you do not get this message, it means you do not have conflicting edits. Make sure you've made different edits to the **same piece of text** in the two branches.

Go ahead and create the pull request.

Resolving Merge Conflicts

Once you've created the pull request, the page will look similar to what we saw earlier except it will indicate that the branch has conflicts that must be resolved:


learn-co-curriculum / our-great-book
Private
Edit Pins
Unwatch
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
add chapter 2 #2
Open
lizbur10 wants to merge 1 commit into main from chapter-2
Conversation 0
Commits 1
Checks 0
Files changed 2
lizbur10 commented 1 hour ago
No description provided.
add chapter 2
89e830c
Add more commits by pushing to the chapter-2 branch on learn-co-curriculum/our-great-book.
This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts.
Resolve conflicts
Conflicting files
chapter-1.md

Before we get to resolving the conflicts, let's take a look at the "Changed files" tab. When we created our first pull request, the diff just showed the addition of lines to our file. In this case, however, the output is a bit more informative. For our version of the file (yours will be different, of course), it looks like this:

The diff is comparing the original version of `chapter-1.md` to the edited version in the `chapter-2` branch. The lines in red were deleted (or modified) from the original version, and the lines in green were added. If we did not have any merge conflicts and could go ahead and merge in

the changes, the lines in green show what the file would look like after we did.

Click back to the "Conversation" tab. For this example, GitHub is providing two options for resolving the merge conflicts: using the web editor or using the command line. The web editor option is not always available, depending on the specifics of the conflicts, so we will go through the process using the command line option. When it's available, the web editor option is a bit less work, because you don't need to pull code down and push it back up later. We encourage you to explore that option on your own.

Click on the embedded "command line" web link. You will see a panel open just below with a set of steps to follow. The process is:

1. Make sure you have the most current version of the code on the `main` branch in your local repo.
2. Merge `main` into `chapter-2` so `chapter-2` has the current version as well. You'll need to resolve the conflicts at this time.
3. Push the `chapter-2` branch back up to GitHub.

When we push `chapter-2` up, it will no longer have any conflicts with `main`, so we should then be able to merge in the pull request.

Let's go through the steps together.

In your terminal, switch to the main branch, then run the first command provided by GitHub:

```
$ git pull origin main
From github.com:learn-co-curriculum/our-great-book
 * branch            main       -> FETCH_HEAD
Already up to date.
```

We see here that our local version is already up to date with the `main` branch on GitHub. This is because we've simulated the situation in which two different people have made conflicting changes to `chapter-1.md` so the changes are already in our local copy of `main`. When you go through this process in real life, `main` will be updated to match what's on GitHub.

Next, we'll switch to the `chapter-2` branch and run the command to merge it into `main`:

```
$ git checkout chapter-2
Switched to branch 'chapter-2'
$ git merge main
Auto-merging chapter-1.md
```

```
CONFLICT (content): Merge conflict in chapter-1.md
Automatic merge failed; fix conflicts and then commit the result.
```

Here you see Git telling us that we need to resolve the conflicts before we can merge the current version of `chapter-1.md` on `main` into the version on `chapter-2`. Using the process you learned in an earlier lesson, resolve each of the conflicts in whatever way you like: you can keep the current changes, the incoming changes, or some combination of the two. Once you've resolved all the conflicts, commit the changes. Doing that will complete the merge.

Finally, we'll re-push `chapter-2` up to GitHub, which will update the pull request:

```
$ git push -u origin chapter-2
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 402 bytes | 402.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:learn-co-curriculum/our-great-book.git
   89e830c..5388208  chapter-2 -> chapter-2
branch 'chapter-2' set up to track 'origin/chapter-2'.
```

Go back to the GitHub repo and refresh the page. You should now be able to merge in `chapter-2`. Don't forget to delete the branch when you're done.

Whew!

Avoiding Merge Conflicts when Working Collaboratively

When you are working as part of a team, it's pretty much inevitable that you will have merge conflicts from time to time. This is why it's important to understand how to resolve conflicts when they arise. However, there are some strategies you (and your teammates) can use that will help keep conflicts to a minimum:

1. Always create feature branches and do your coding there.
2. When working on a feature branch, as much as possible, avoid doing work that isn't directly related to the feature you're working on.
3. Update your feature branch frequently to make sure it's kept up to date with the `main` branch on GitHub.
4. Develop good communication strategies as a team, so everyone knows what their teammates are working on and can avoid doing work that might conflict.

As you can imagine, if you're working on a large project with many team members, resolving merge conflicts can become quite messy! Making an effort to adopt the habits listed above will help you avoid those situations.

Making Open Source Contributions: The Fork and Clone Workflow

So far we've only talked about cloning directly from the source repo, which works fine if you have permissions on the repo. However, what about the case when you're working on a repo you don't have permissions for? You can clone it down and make changes to it, but unless you're a member of the organization or explicitly identified as a collaborator on the repo, you can't push your work up. If you want to contribute to an open source project, for example, you won't have permissions to push up any changes you make to the central repo.

The workflow in this case is to fork the repo to your own GitHub account, then clone down your fork to your local machine to work on it. Once you've completed your work and push it back up to your fork on GitHub, you will see the familiar message indicating your branch had recent pushes and a button to submit a pull request to the source repo.

Making open source contributions is a great way to practice the skills you've learned in this module and build your GitHub profile at the same time. Here are some suggestions to help you get experience contributing to open source projects:

- We strongly recommend completing **this interactive GitHub tutorial (https://github.com/firstcontributions/first-contributions/blob/master/README.md)**. It will walk you through the process step by step.
- Check out this **curated list (https://firstcontributions.github.io/#project-list)** of projects that have issues suitable for beginners. You can filter the list by language and other attributes.
- Practice on the Flatiron School curriculum! The next time you see a typo or other simple error (we admit it — we're not perfect), submit a pull request to fix it!

To review, the process is:

1. Fork the repo to your own GitHub account.

2. Clone your fork down to your local machine.
3. Create a branch and switch to it.
4. Fix the error.
5. Push the branch up to your fork of the repo.
6. Go to your fork of the repo on GitHub, and submit a pull request.

Two Person Exercise

In this example we used for this lesson, we simulated the case where two different collaborators submit conflicting changes. To practice the skills you learned under somewhat more realistic circumstances, we strongly recommend that you team up with a classmate and practice:

- creating local branches and pushing them up,
- creating pull requests,
- requesting reviews,
- reviewing each other's work, and
- resolving merge conflicts.

You or your partner can create a new repo as we did in this lesson if you like, but you can also find an existing repo you would like to work on. In this case, one team member will fork the repo to use as the team's "central" codebase. Once you've forked it, for purposes of this exercise it's yours — you can treat it just the same as if you'd created it yourself.

The second team member can then contribute to this repo in one of two ways:

1. The second team member can fork the new "central" repo (i.e., fork the fork), and proceed using the fork and clone workflow, or
2. The first team member can **make them a collaborator (https://www.tutorialspoint.com/how-to-add-collaborators-to-a-repository-in-github)** on the new "central" repo so they can submit PRs directly to it.

Conclusion

In this lesson, we've learned the basics of how to use GitHub when working collaboratively with others. These are crucial skills that many potential employers look for. Practicing these skills will put you in a great position when you start your job search.

One of the ways you can get practice is by making contributions to open source projects. This will aid you in your job search even more, because open source contributions look great on your GitHub profile!

Resources

- **How to Contribute to Open Source Projects — A Beginner's Guide (https://www.freecodecamp.org/news/how-to-contribute-to-open-source-projects-beginners-guide/)**