GitHub Fundamentals BootCamp Labs

Learn the complete GitHub – from code management to Copilot

Revision 2.0 – 01/09/25

Tech Skills Transformations LLC / Brent Laster

Setup and prerequisites

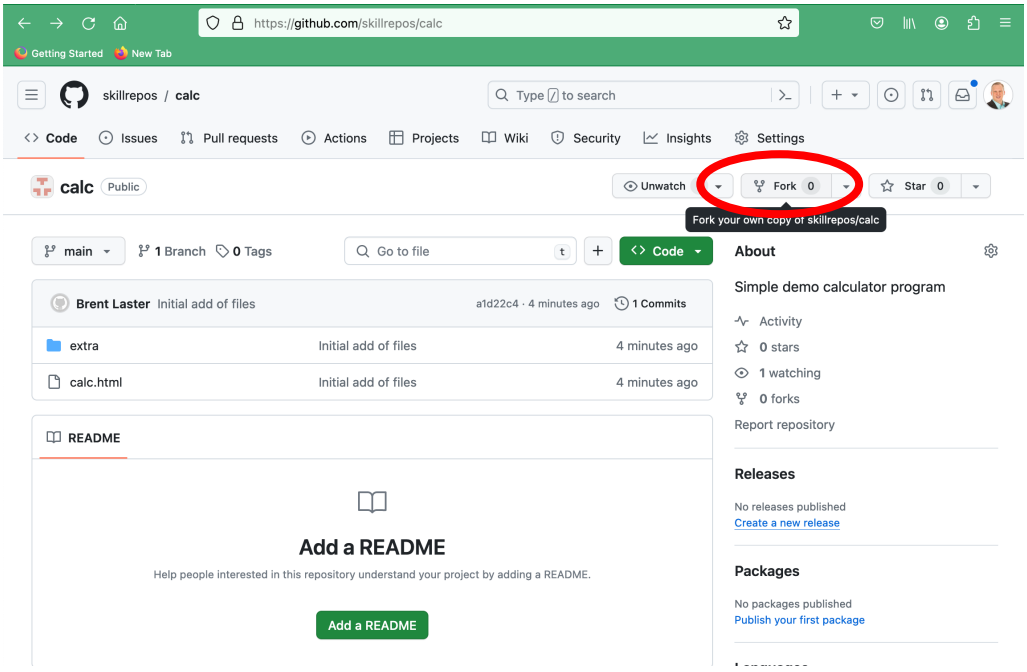
1. In order to do some of the labs in this class, you will need to have a personal access token (PAT) setup and also two separate GitHub userids, as well as a version of Git installed.
2. Git can be installed by going to https://git-scm.org and following the instructions there for your OS.
3. To create the second GitHub userid, just select another email address and sign up for the free tier at GitHub.com.
4. You can set up the PAT in advance by following the instructions here or do it as part of the first lab.
5. If you are doing the labs on Windows, it is recommended to use the Git Bash shell that can be installed with Git for Windows.

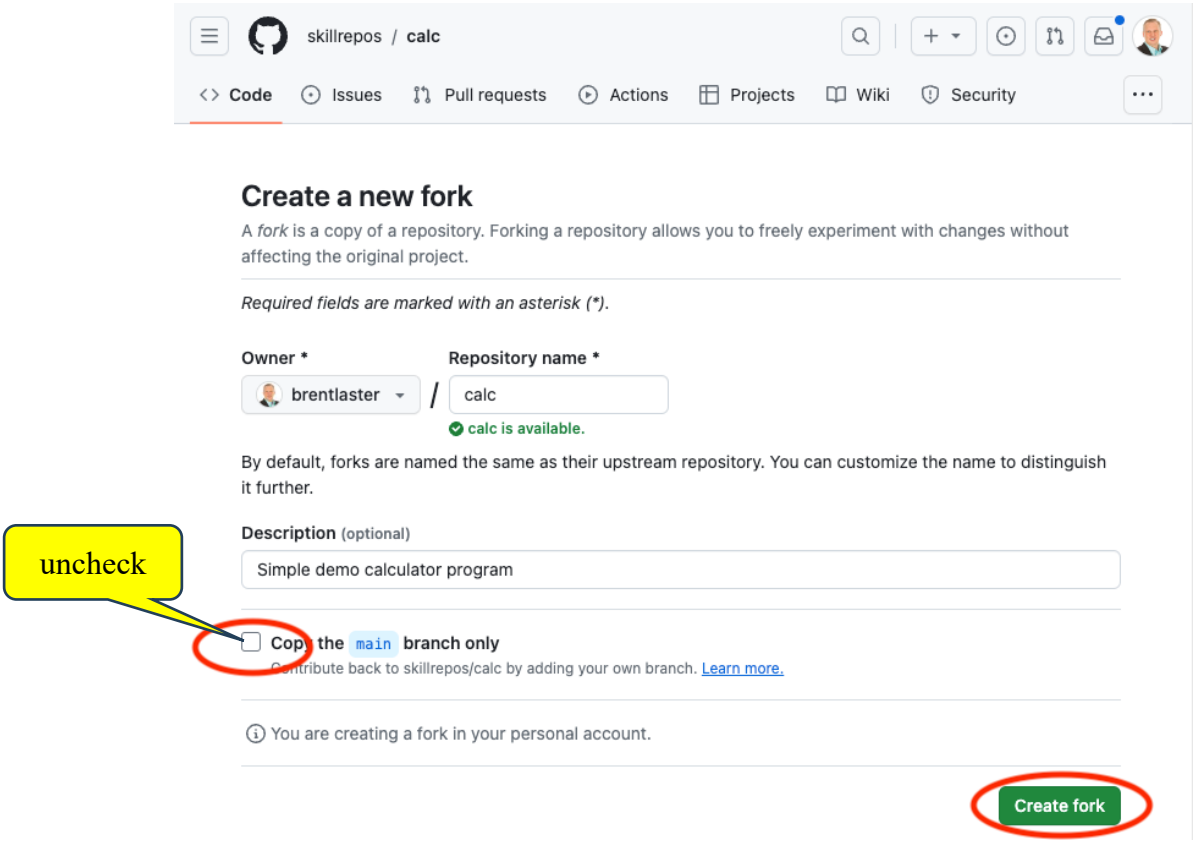
Lab 1 – Getting Started

Purpose: *In this lab, we'll get a quick start learning about GitHub through forking a project, creating a new file and committing it.*

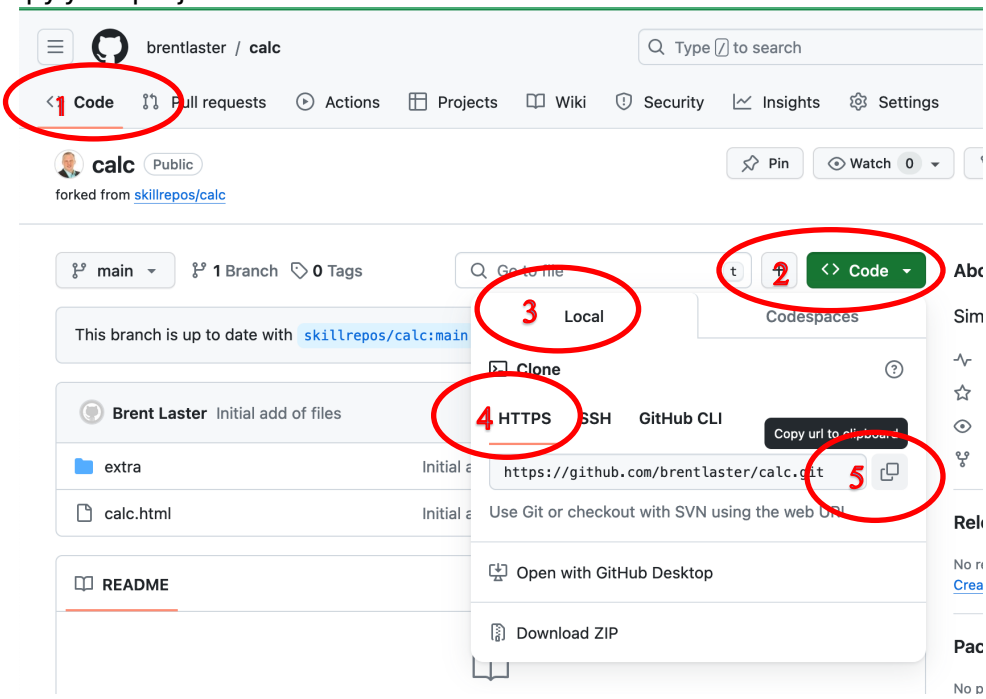
1. Log in to GitHub with your primary GitHub account.

2. Go to https://github.com/skillrepos/calc and fork that project into your own GitHub space. Do this by clicking on the ***Fork*** button. On the next screen, **make sure to uncheck** the box next to ***Copy the main branch only*** . Then click the ***Create Fork*** button. (Note: If you cannot use the *Fork* functionality, see alternatives in **Appendix 2** at the end of this document.)





3. Now you'll be on your fork of the repo. Next, let's clone your repo down to your local system so we can make changes there. In your project, ensure you are on the ***Code*** tab, then click on the large green **<> *Code*** button. In the ***Local*** tab, select ***HTTPS*** under Clone and then click on the ***copy icon*** to copy your project's URL.



4. Open a terminal on your system and clone down the repository from GitHub. You can use the following command – just paste (or type) the URL you copied from the step above and hit Enter/Return. Then change into the subdirectory that was created from the clone.

```
$ git clone <url from repo>
$ cd calc
```

5. If not already set globally, configure your name and email. Best practice would be for your email to be the same as the one you're using for your userid on GitHub.

```
$ git config user.name "your name"
$ git config user.email  <same email as you're using on GitHub>
```

6. After this you can run the command below and see that GitHub is setup as your remote repository.

```
$ git remote -v
```

7. Let's make a simple edit to a file so we can have a change to push back to GitHub. Edit the calc.html file and update the line in the file surrounded by ***<title>*** and ***</title>*** to customize it with your name. The process is described below.

Edit calc.html and change

<title>Calc</title>i

to

<title> ***name's*** **Calc</title>**

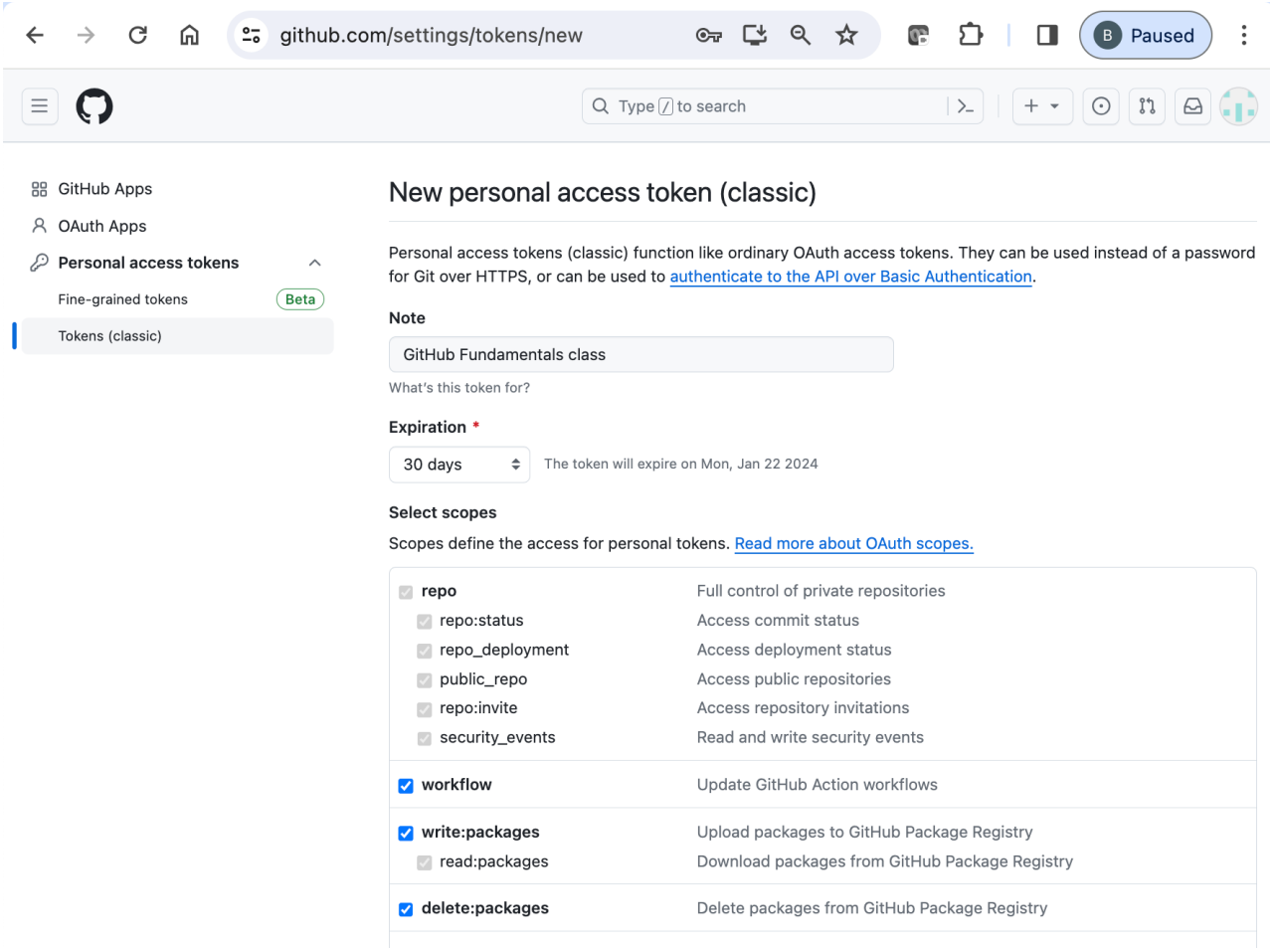
substituting in your name (or some other text) for "name's".

8. Save your changes and commit them back into the repository.

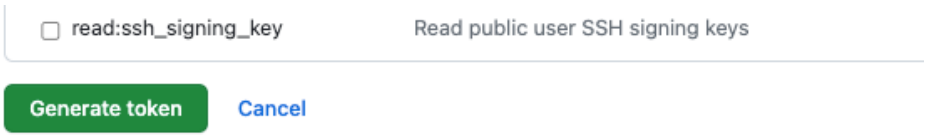
```
$ git commit -am "Updating title"
```

9. Several aspects of using GitHub rely on options you can set in the user ***Settings*** menu. To demonstrate this and in preparation for the next lab, we'll go to settings to create your Personal Access Token (PAT) that you'll need for securely pushing changes over to GitHub in place of a password.
   To create your PAT, follow the instructions for creating a classic token at https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic

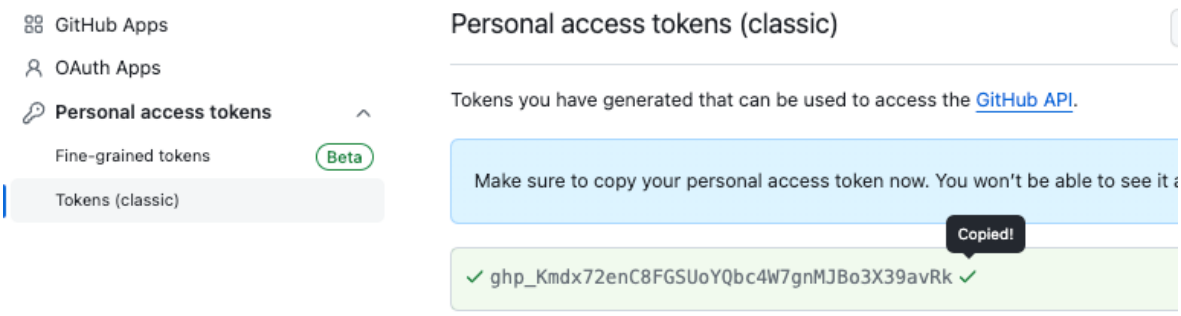
(Shortcut to token page is https://github.com/settings/tokens/new)
When setting up your token, ensure that you have the boxes checked for the first four scopes (*repo – delete:packages*) as shown below. **Also make sure to copy and save the token for future use.**



When done, click on the green ***Generate Token*** button.



Make sure to save a copy of the token string from this screen - you won't be able to see it again.

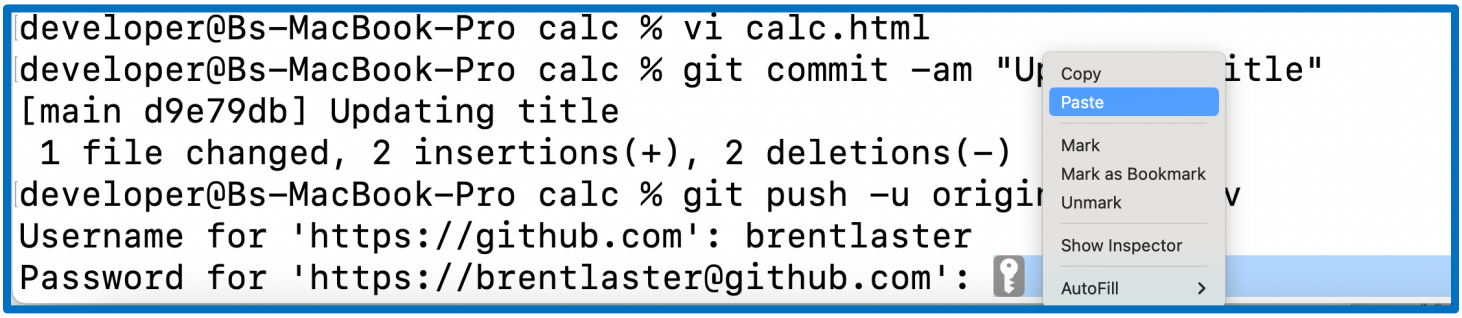


9. Now, let's go ahead and push your change back into GitHub. We'll push to a new branch in preparation for the next lab.

```
$ git push -u origin main:dev
```

10. After this, you'll be prompted for username (your GitHub username) and then a sign-in/Private Access Token or password. Wherever it asks for a token or a password, you can just copy and paste in the token you generated in GitHub prior to this lab. An example dialog that may come up is shown below.

If instead, you are on the command line and prompted for a password, just paste the token in at the prompt. Note that it will not show up on the line, but you can just hit enter afterwards.



NOTE: If you hit run into problems trying to push with the token, such as it saying invalid password, you may be getting caught by previously saved credentials. See the very end of this doc for some other options.

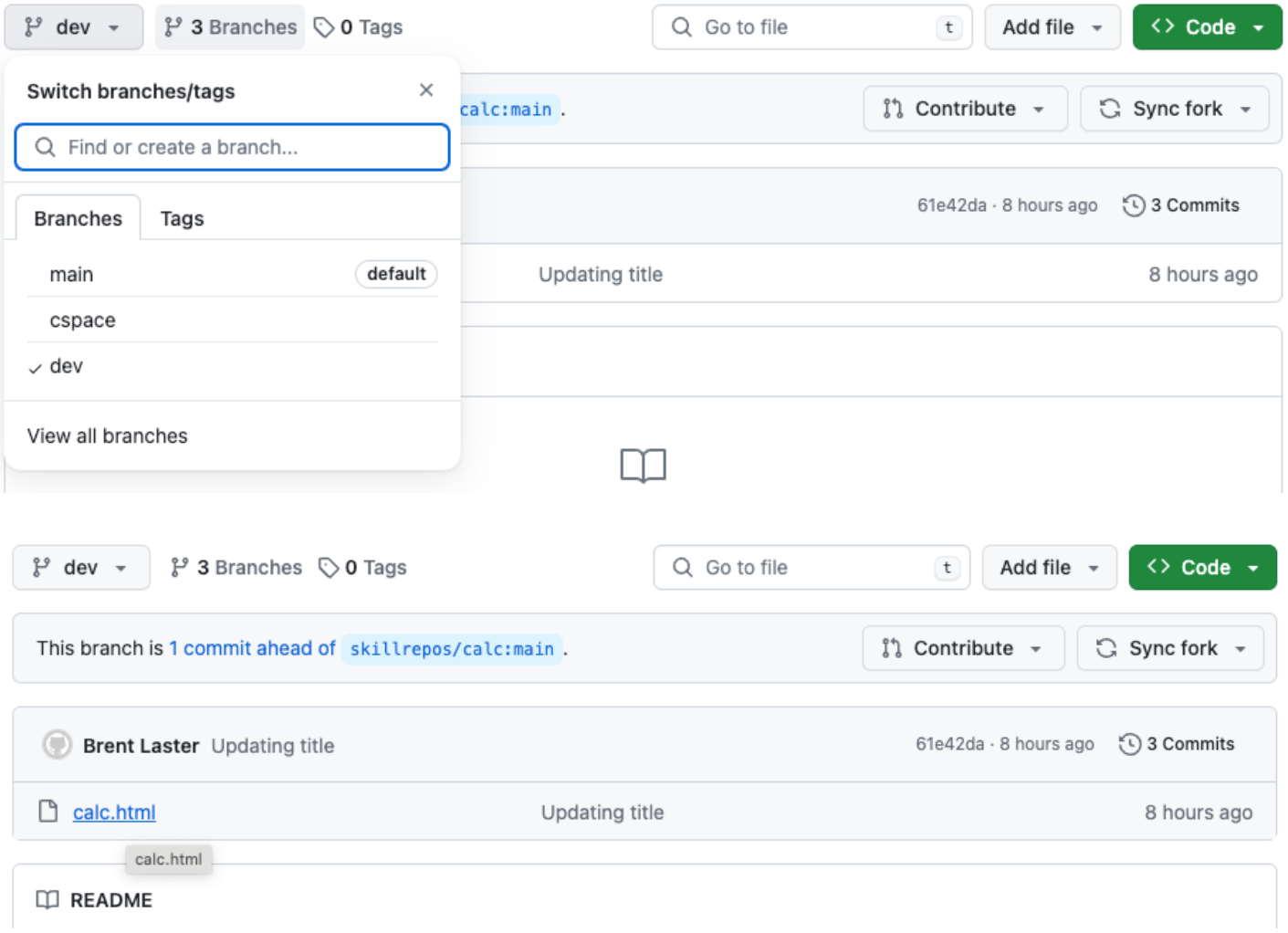
END OF LAB

Lab 2 – Pull requests

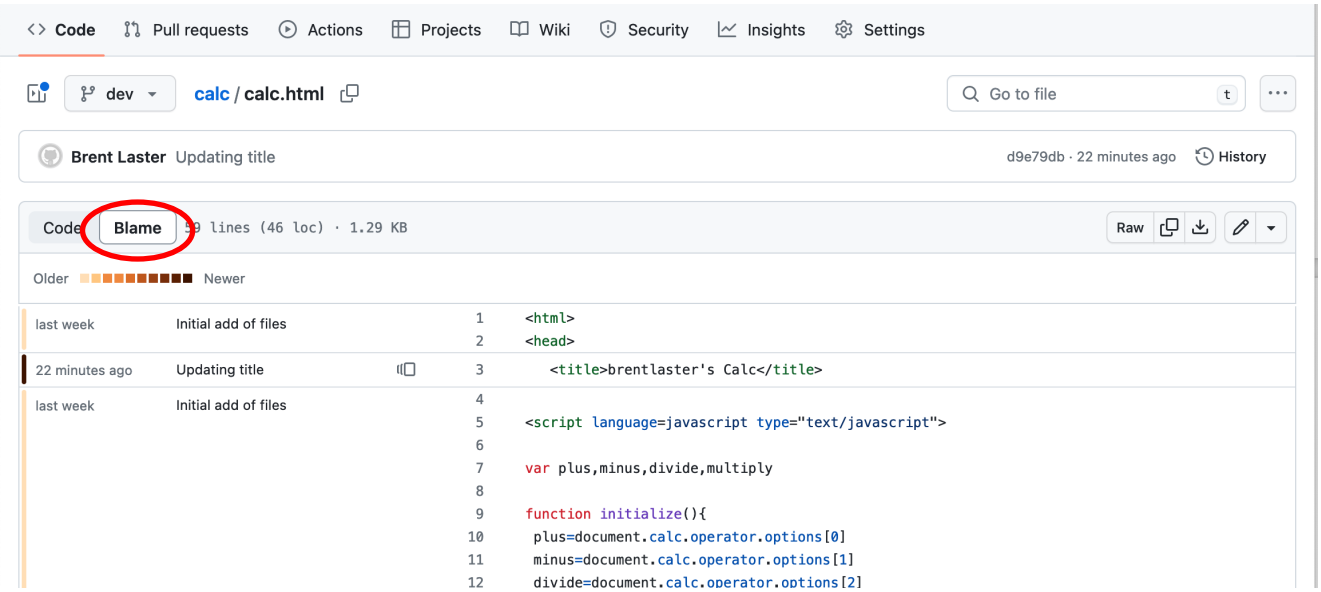
Purpose: *In this lab, we'll see how to merge a change using a pull request.*

1. After the push is complete, you can switch back to the GitHub repo in the browser, change the branch to *dev* and click on the calc.html file to see the change.

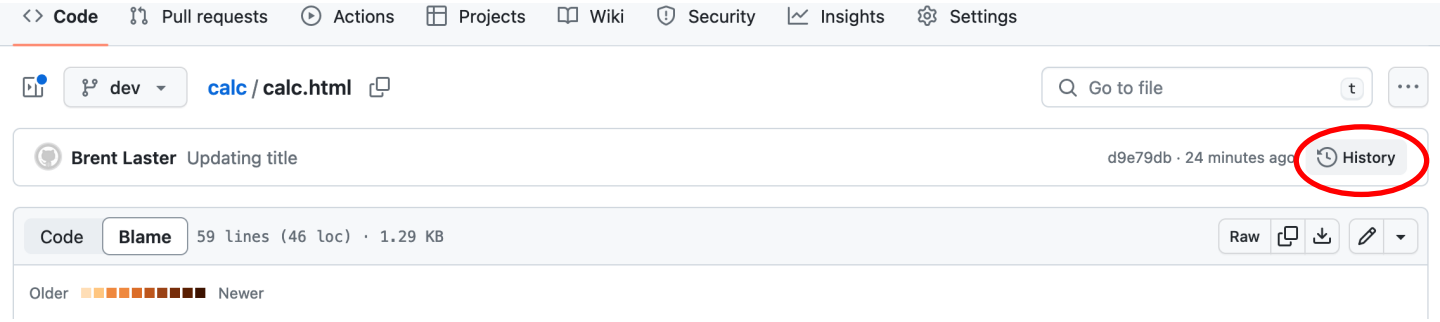
(If you don't see ***dev*** listed in the branch dropdown list, click on the ***3 Branches*** button next to the dropdown and you should be able to see it there. Alternatively, you can go to ***github.com/<github userid>/calc/tree/dev*** in the browser.)



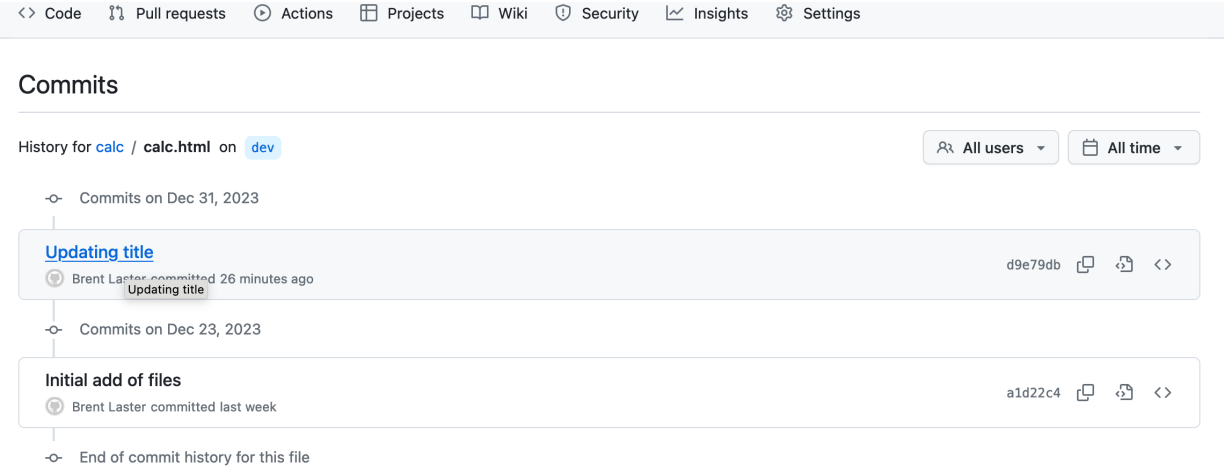
2. Click on the file name to open the file in the browser. While you have the file open there, click on the *Blame* button in the gray bar at the top to see additional information about who made changes to the content.

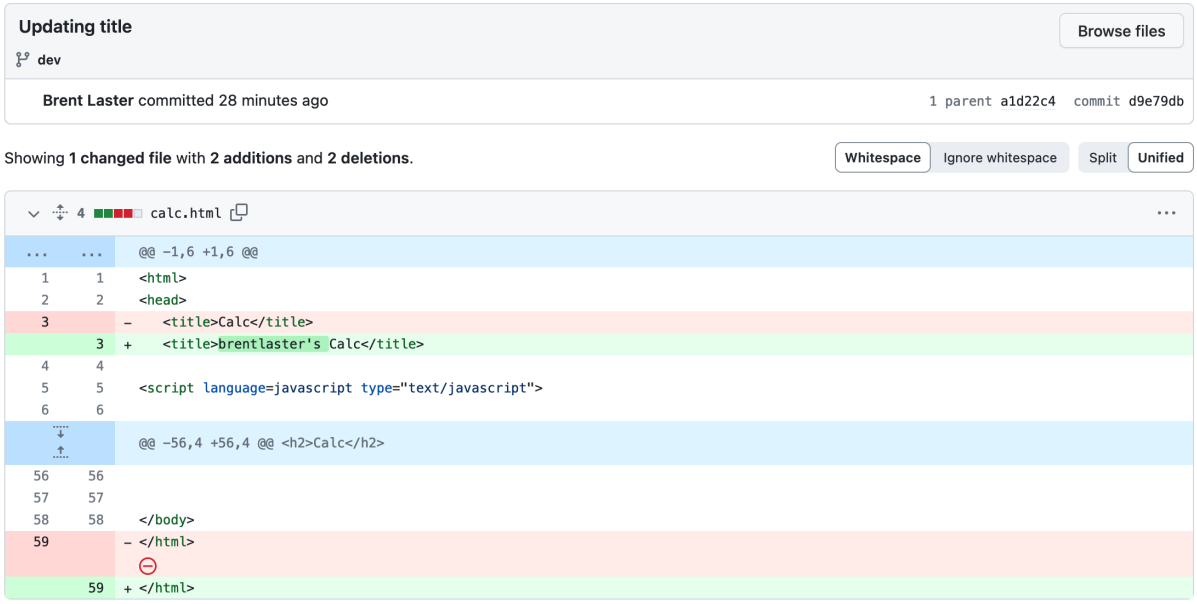


3. Also, click on the *History* button (upper right) to see the change history for the file.



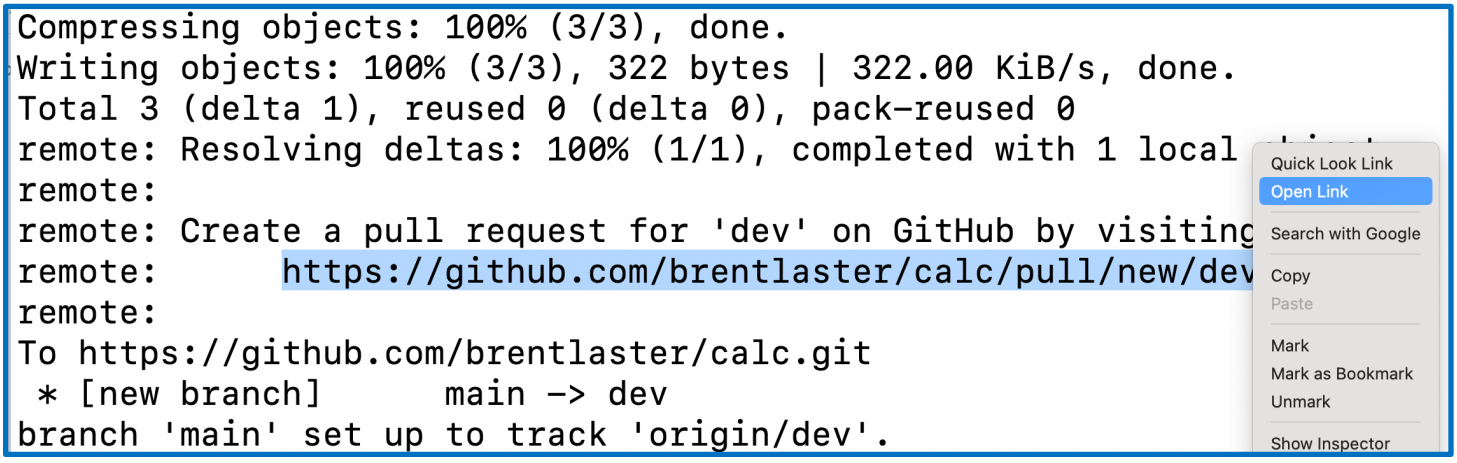
4. In the history screen, click on the commit message for your change. You'll then be able to see the differences introduced by your commit.



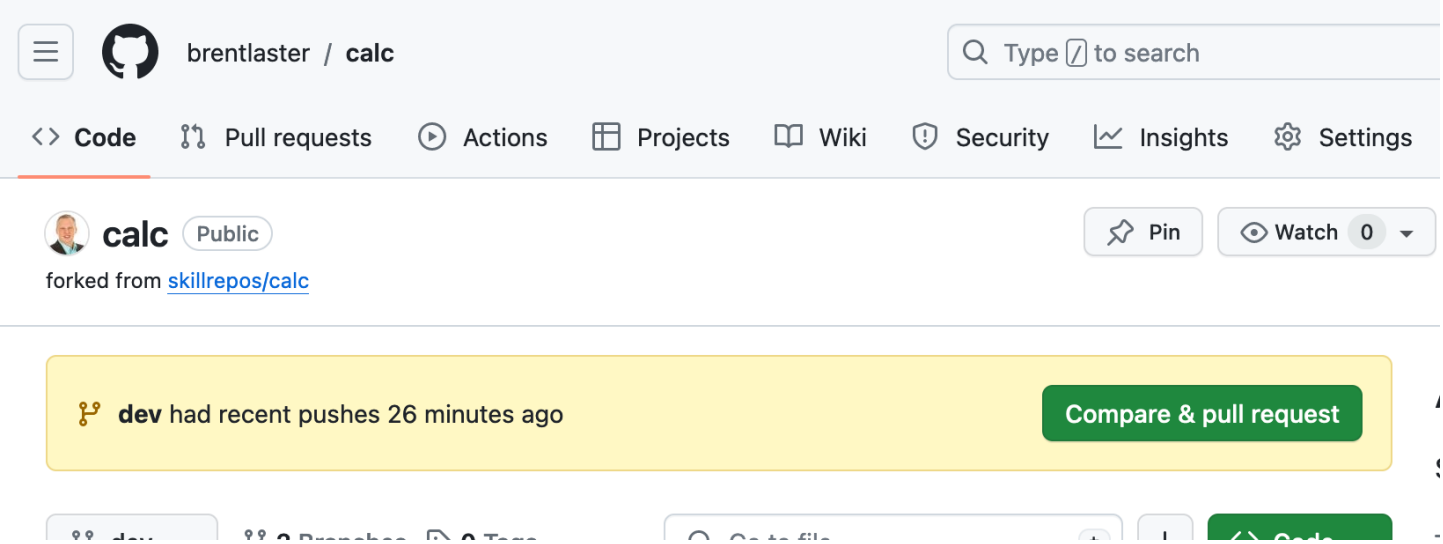


5. Let's now merge our change from the dev branch to main via a pull request. **Switch back to the terminal where you did the commit and the subsequent push.**

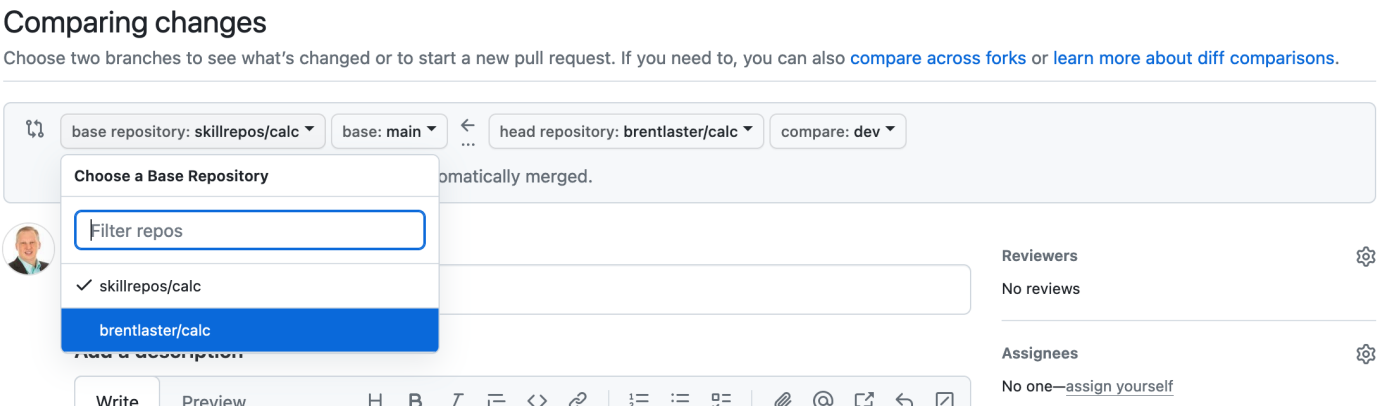
In the output from the push, you should see a link (*highlighted in the screenshot below*). Highlight/select the link and then right-click and open the link. (Alternatively, you can go back to the main page of your repo and if you see a message there that looks like the second picture below, you can just click on the *Compare & pull request* button.)



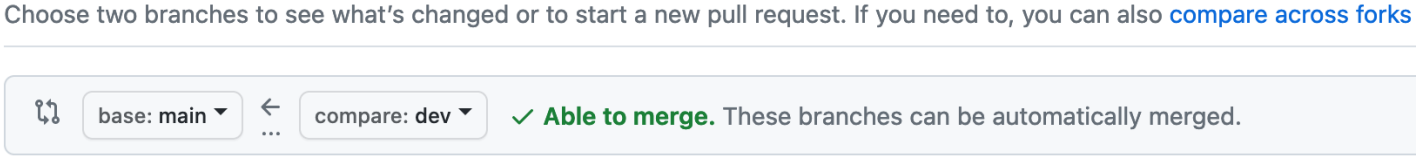
-- OR –



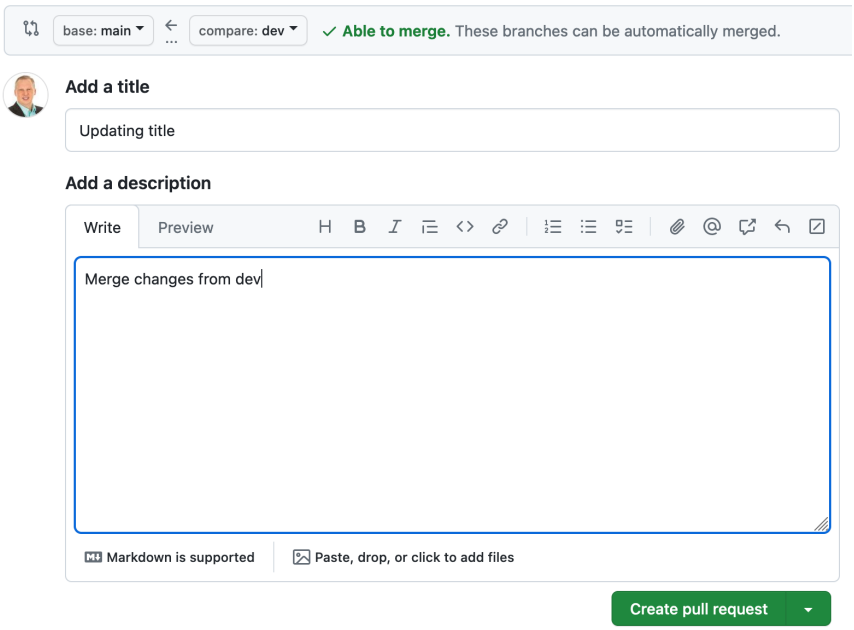
6. Depending on which option you chose in the step above, you may either be on a *Comparing Changes* screen or *Open a pull request* screen. In either case, we need to update the base repository in the gray bar at the top to make the merge go **to your repo and NOT to *skillrepos/calc*.** Click on the dropdown (small downward pointing arrow) in the "base repository" box, and select your repo from the list.



7. After making that change, the gray bar showing the base and compare should look like the screenshot below.

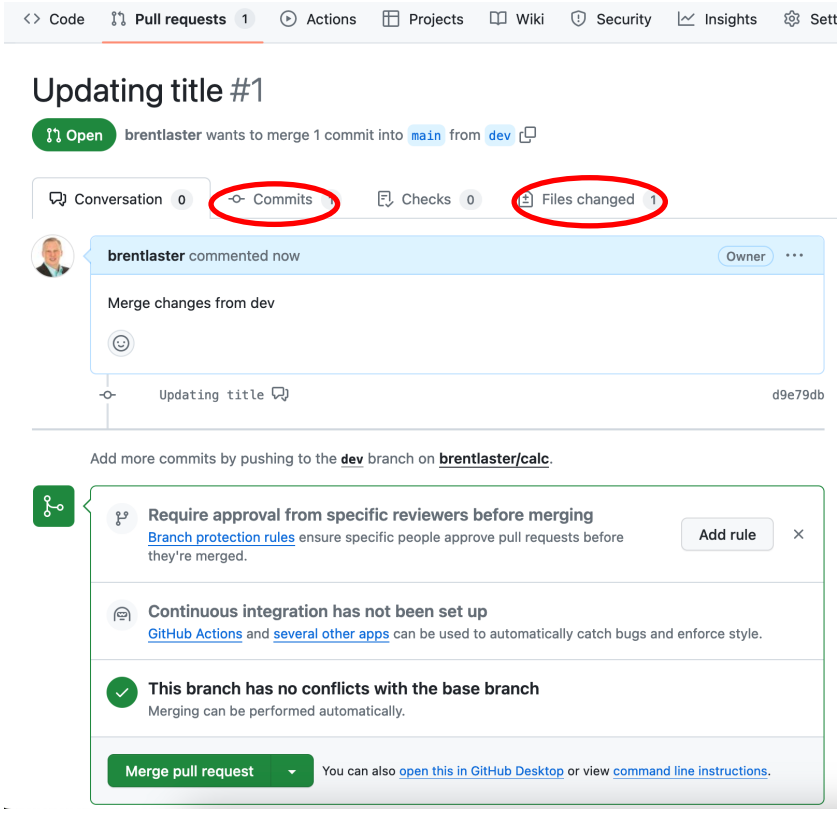


8. Now, with your repo selected for the base, add an optional description if you want and then click on the ***Create pull request*** button.

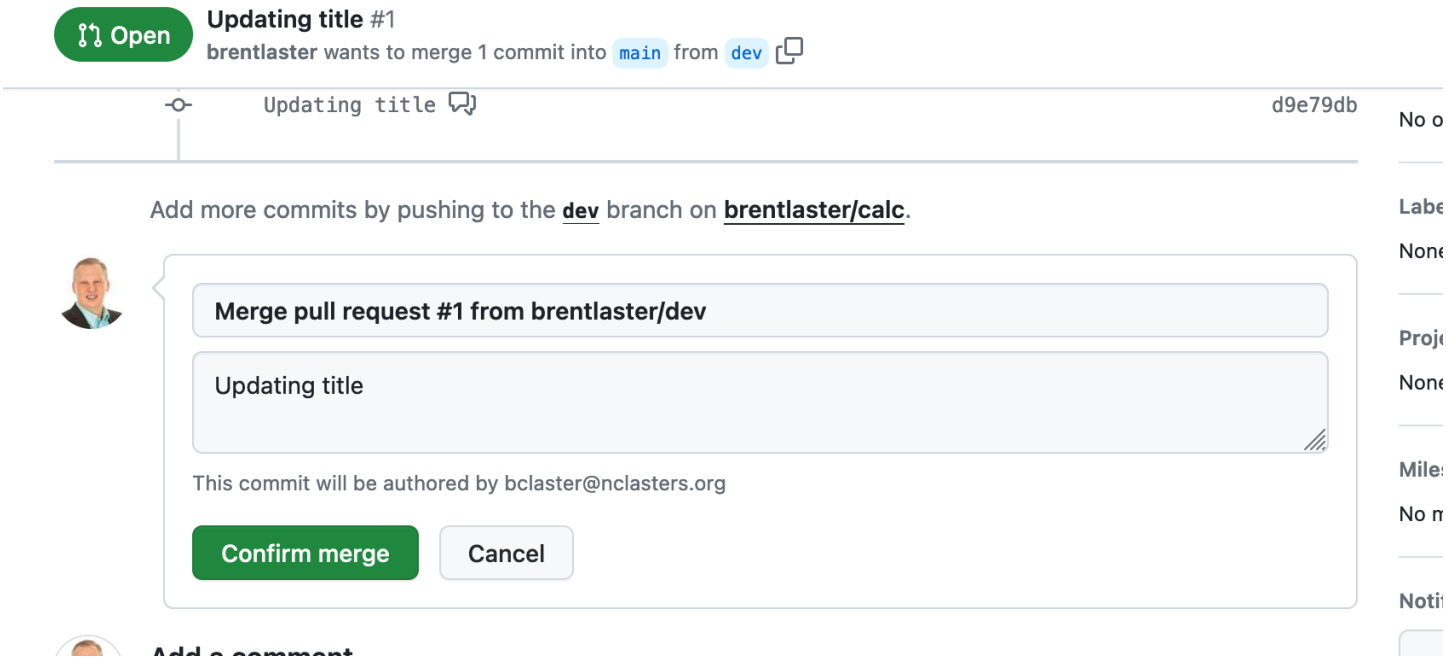


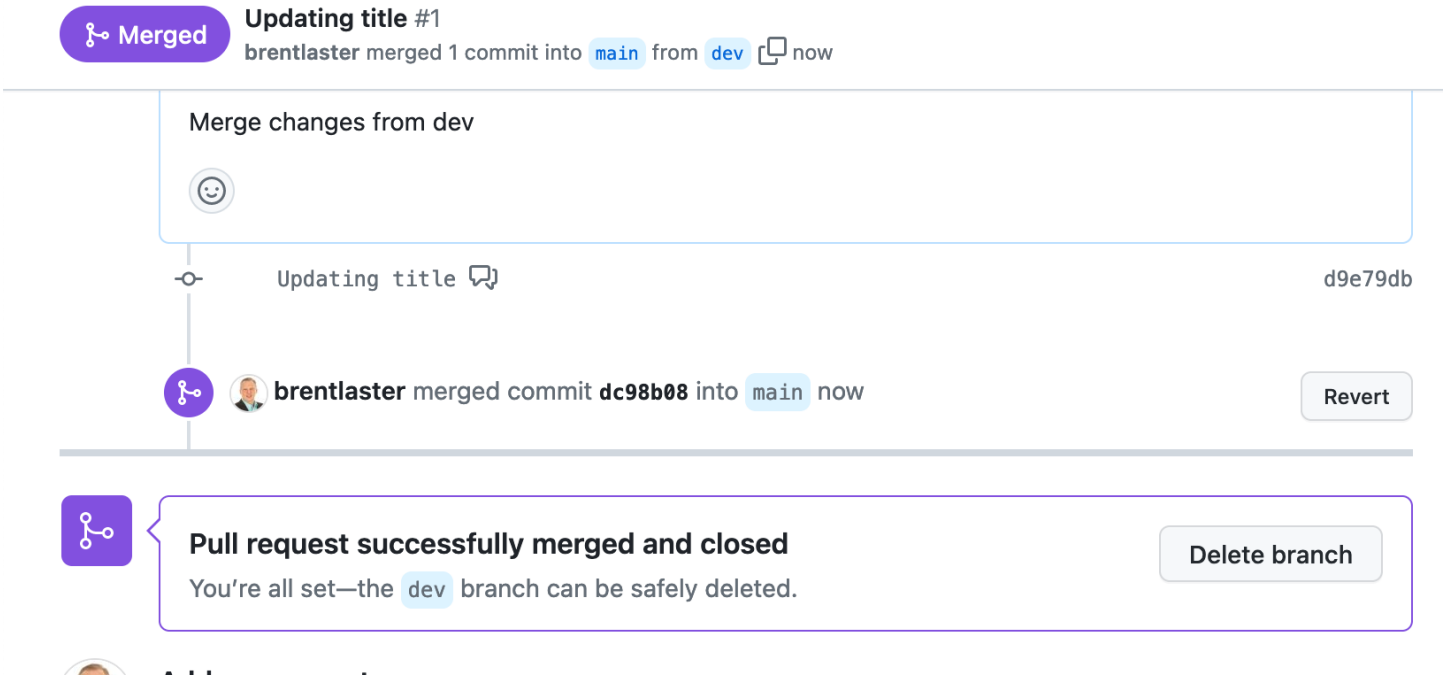
9. At this point, you have created a new pull request. (Note that the *Pull Requests* tab at the top shows 1 pull request in the repo.) It will check for any conflicts for merging.

   We haven't set up any CI processes or reviewers so there is nothing for those sections. Note the check in the middle section that says *This branch has no conflicts with the base branch*. You can look at the *Commits* or *File Changed* tabs if you want to see more details on the changes.



10. When you're ready, switch back to the ***Conversation*** tab. Then click on the ***Merge pull request*** button and then the ***Confirm merge*** *button* to complete the pull request. After that, the pull request will be completed and closed (shown in second screenshot). Afterwards, you can click on the button to delete the *dev* branch if you want.



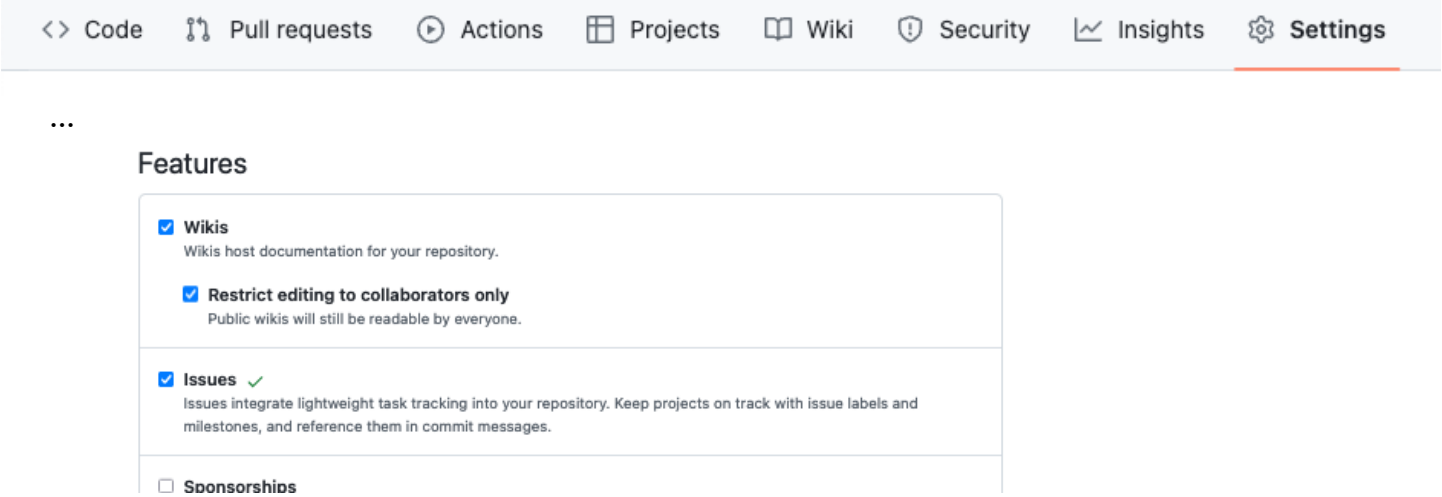


END OF LAB

Lab 3: Creating GitHub issues

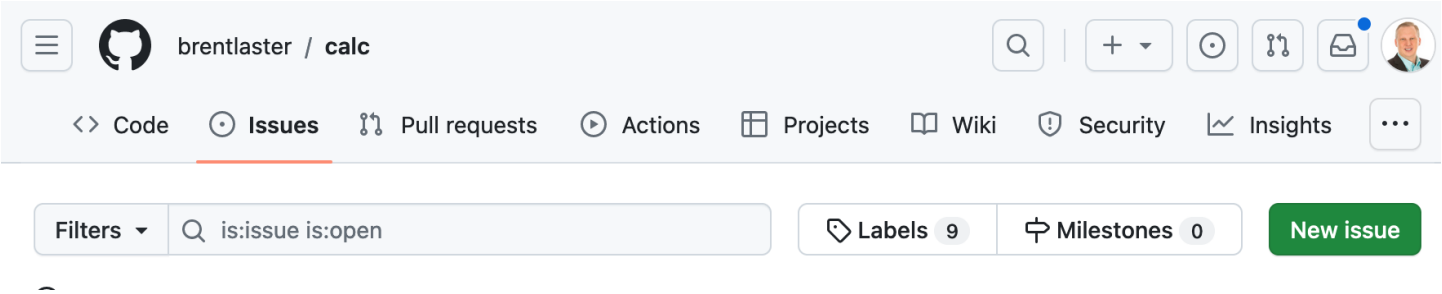
Purpose: *In this lab, you'll create an issue, assign it to a user, and add labels for it.*

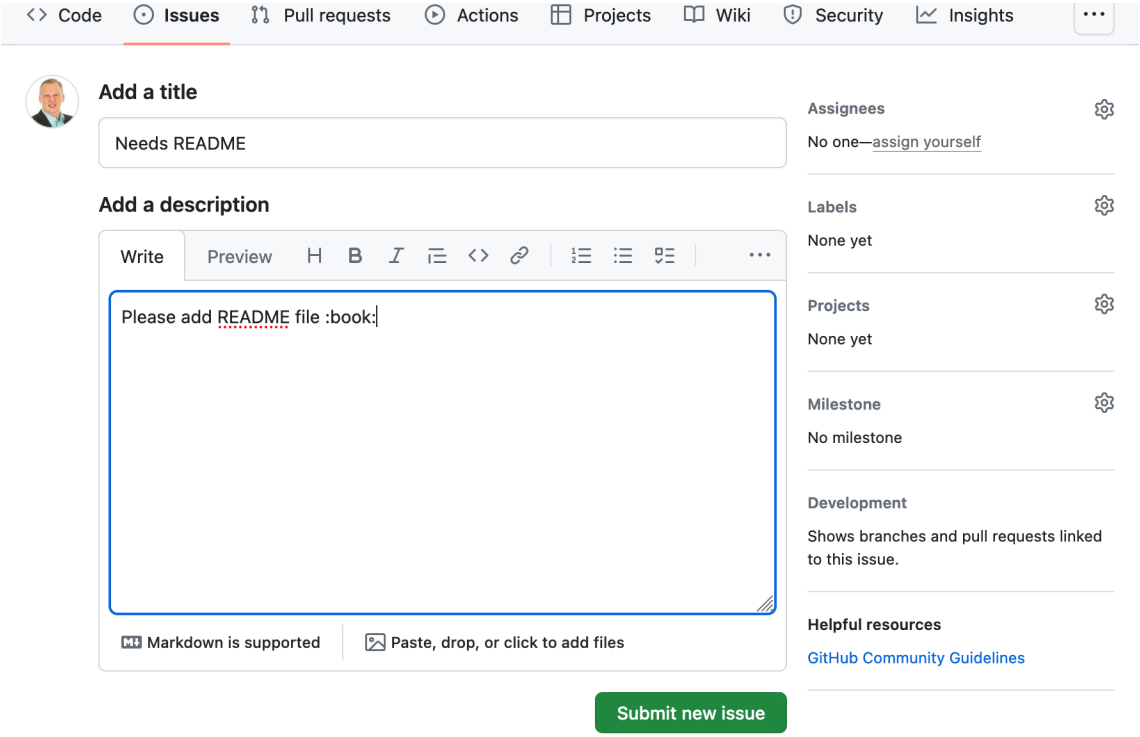
1. We'd like to have a *README* file in our project to make it more standard. So, let's create an issue to document that. First, ensure that the repository has the *Issues* feature turned on. On the main repo page, go to the repository's ***Settings*** tab, and then scroll down until you see the ***Features*** section. Then, check the box for ***Issues***.



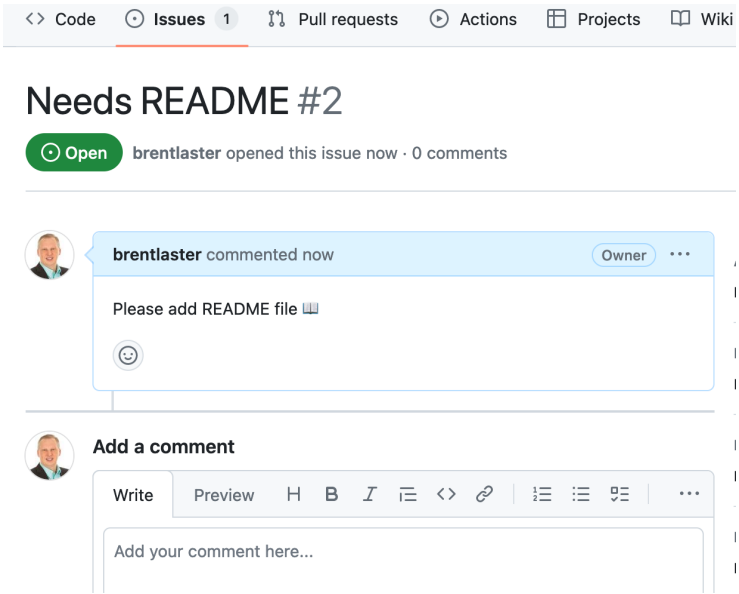
2. Now, click on the ***Issues*** tab at the top of the repository page, then the ***New issue*** button on the right. Then fill in the title with something like "*Needs README*".

   For the description, you can enter something like "Please add a README file :book:". (:book: will be changed to an emoji.) Then click the ***Submit new issue*** button.

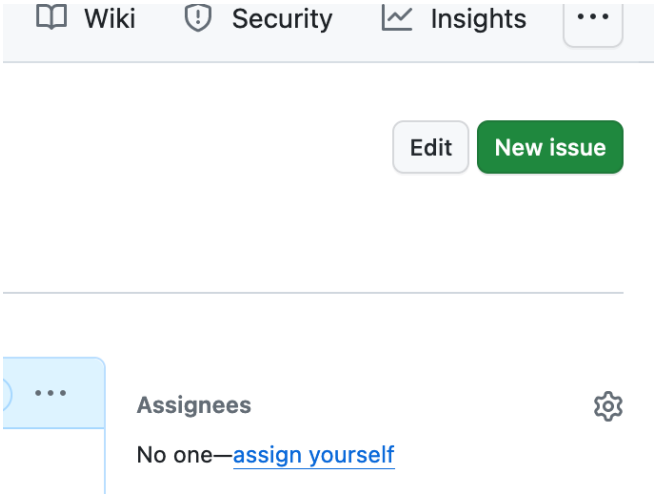




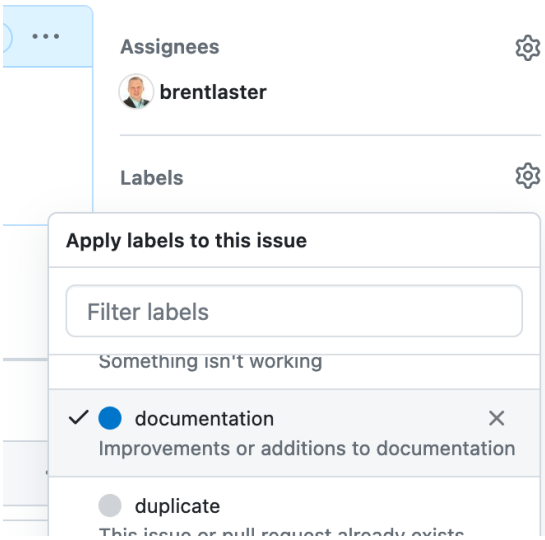
3. Take note of what number is assigned to the issue – you will need it later. (It will probably be #2 for you)



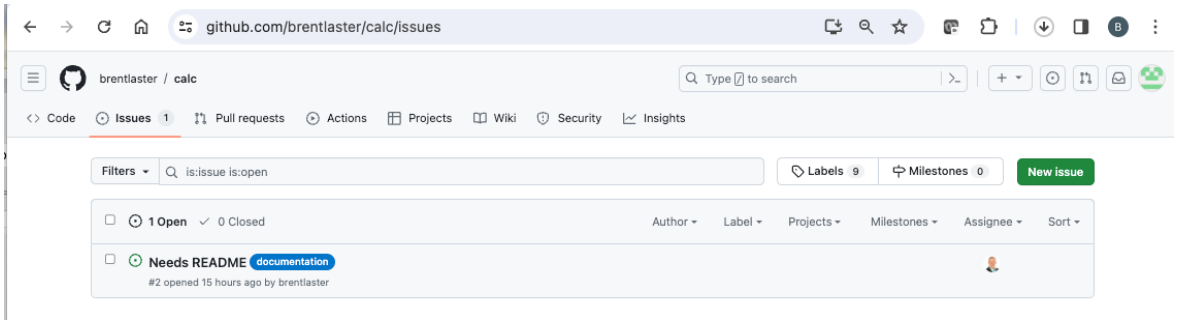
4. Assign the issue to yourself by clicking on the ***Assign yourself*** link under the *Assignees* section on the right.



5. Add the documentation label to the issue by clicking on *Labels* and selecting the *documentation* one. (After selecting it, just click anywhere else outside of the box to apply it.)



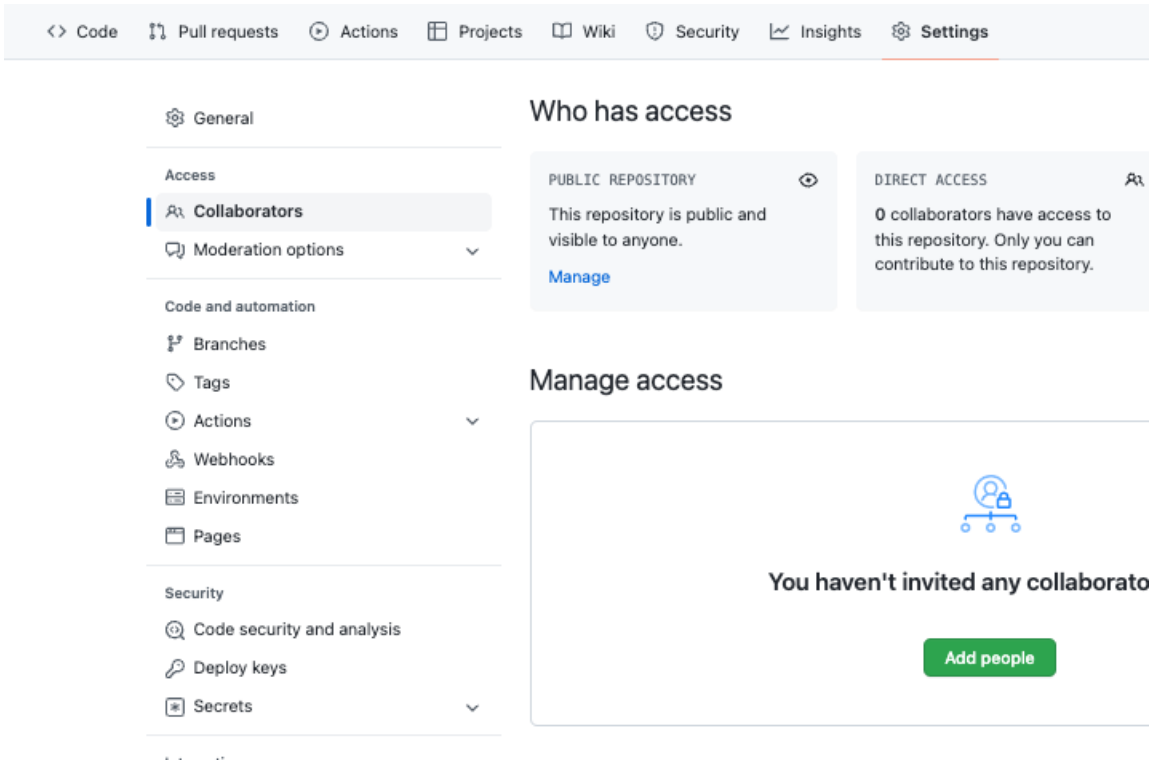
6. After this, if you click on the ***Issues*** tab at the top, and look at your issue, it should look like the following.



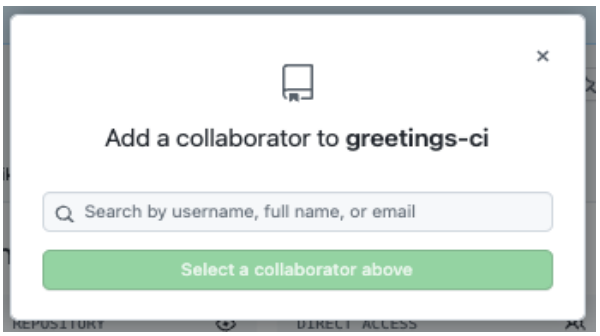
7. In preparation for the next lab, we need to add your second GitHub userid as a *collaborator* to this repository.

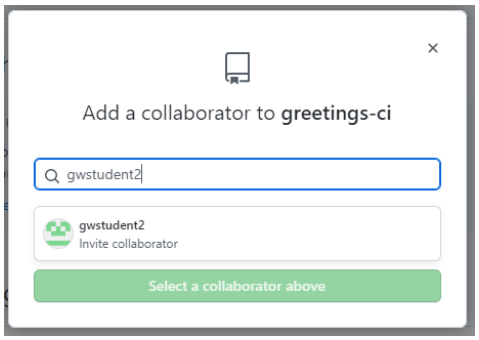
   Go to the repository's ***Settings*** tab and then select ***Collaborators*** on the left under ***Access***.

   Then click the ***Add people*** button.



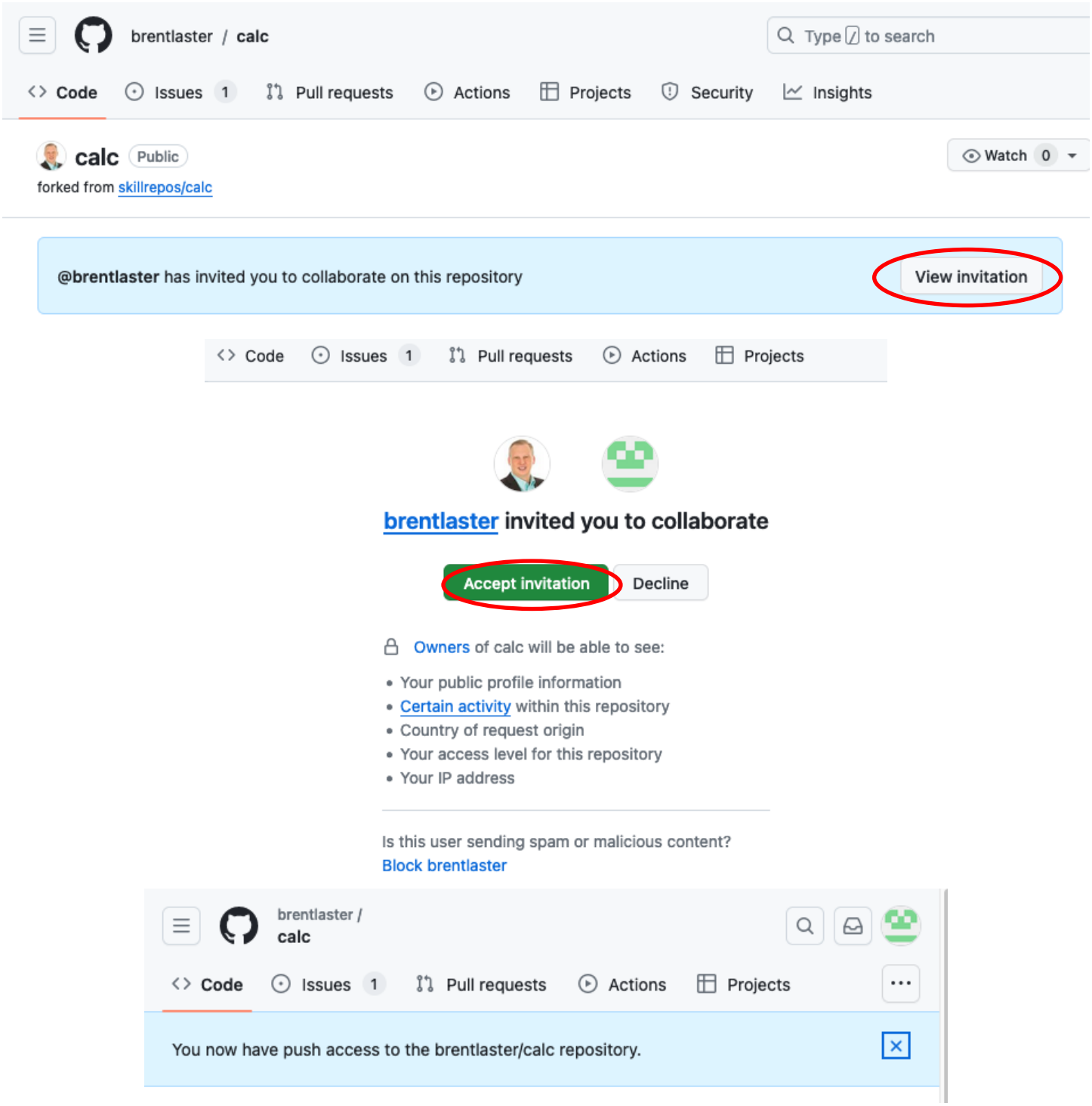
8. In the dialog box that pops up, type in the other GitHub userid you have and then click on the specific id or click on ***Select a collaborator above***. Then, click on ***Add <userid> to this repository***. That userid should then receive an email with the invite which you can accept**.**





9. **Make sure to respond to the email and accept the invitation!** (You will need to sign in as the invited id in a different browser or a private tab or sign out/sign in, and then view and accept the invitation.)

You can get to the invitation through the email link -or- if you sign in as the secondary id and go to *https://github.com/<primary github userid>/calc* you can also view the invitation via clicking on the button.

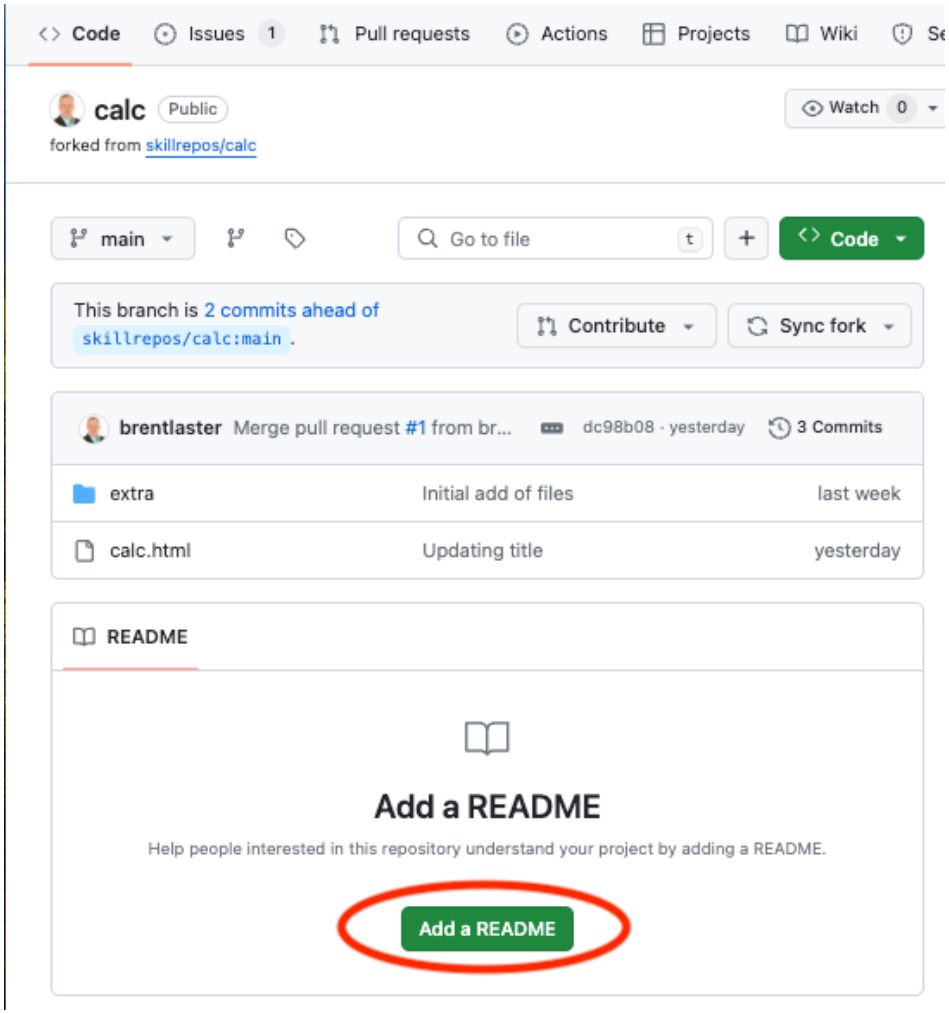


END OF LAB

Lab 4: Setting up a pull request with reviewers

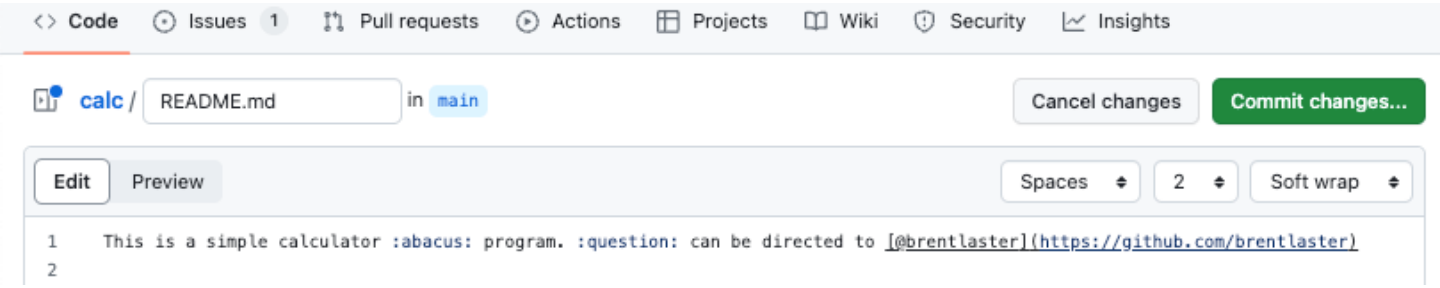
Purpose: *In this lab, you'll use a pull request with a reviewer and an associated issue to make a change.*

1. Now, we'll address adding the README itself per the issue we previously created. If you're not signed in as your original/primary GitHub userid, sign in as that id now. In the ***Code*** tab of the *calc* repository, click on the green button to add a README.md file.

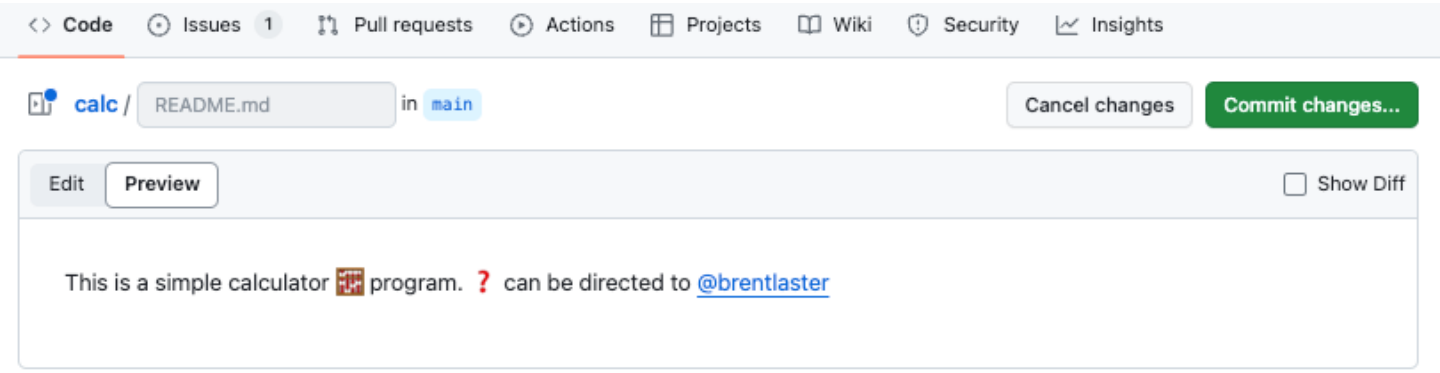


2. This will bring up the editor in GitHub. Enter the text below in the new file text input area for README.md. Fill in your github userid in both places instead of github-userid. (Notes: Do this on a single line. Also, there is no space between the "]" and "(". And since we don't have a calculator emoji, we're using an abacus emoji. Finally, if you cut and paste from this doc, that may add an image link at the end of the line that has to be removed.)

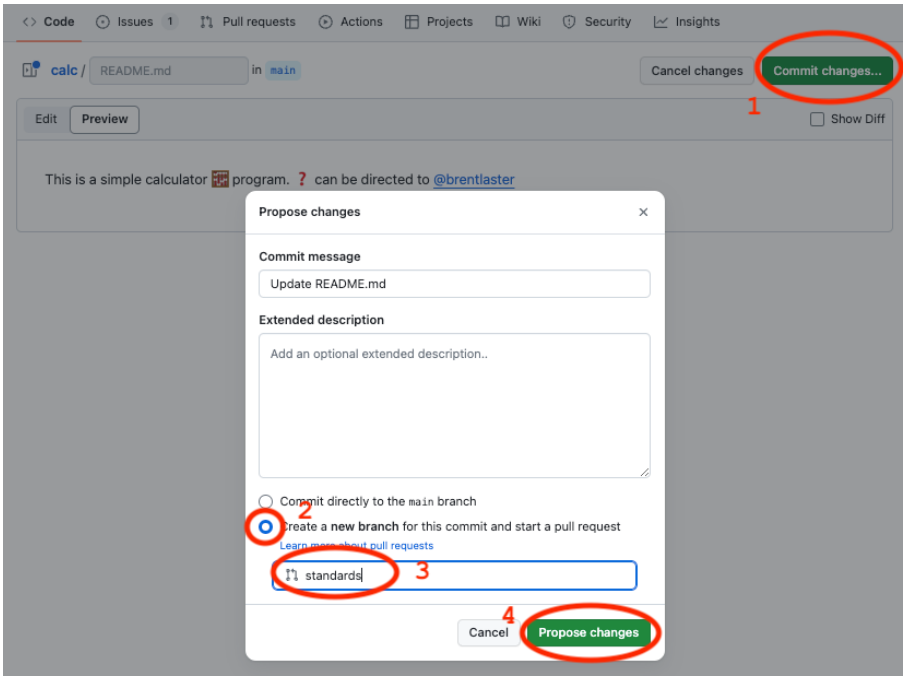
```
This is a simple calculator :abacus: program. :question: can be directed to
[@github-userid](https://github.com/github-userid)
```



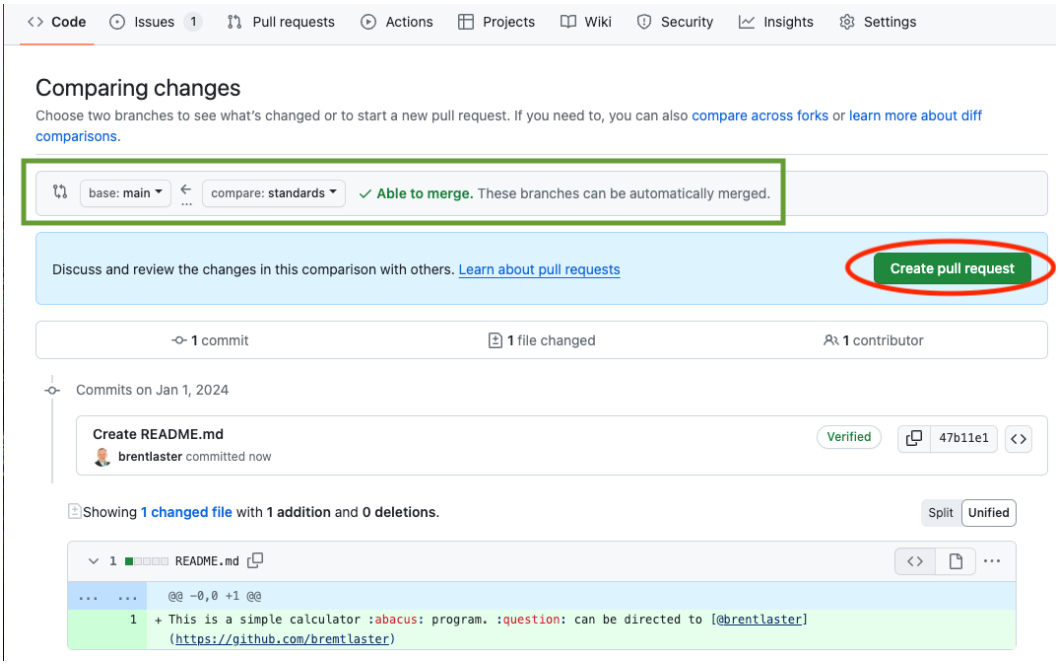
3. Click on the Preview tab (next to Edit) to see how this will render once committed.



4. Now let's commit these changes to a new branch and open a pull request to merge them. click on the green ***Commit changes...*** button in the upper right corner. In the dialog, enter a comment if you want and select the option to ***Create a new branch...*** You can change the generated branch name if you want. In this case, I've changed it to "*standards*". Then click ***Propose changes***.

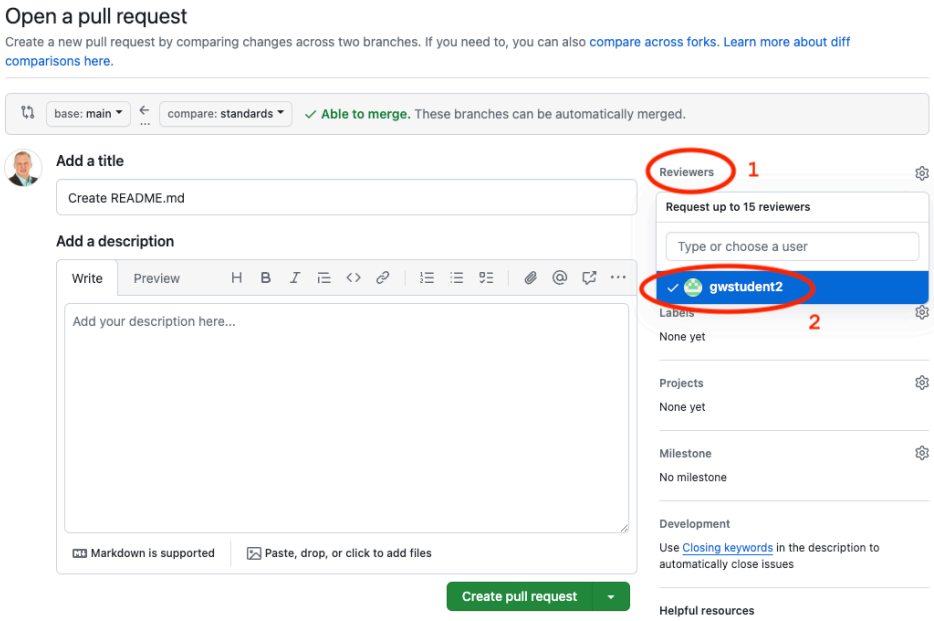


5. At this point, you'll see a screen showing you the changes and what's being compared at the top. This should only be branches in the same repo, not different repos. It should also show a green checkmark with "*Able to merge*." next to it. We're going to create a pull request to be reviewed. Click on the ***Create pull request*** button.



6. You'll now be on the screen to create the pull request. Let's add your secondary GitHub id as a reviewer.

   In the upper right, click on the ***Reviewers*** link, then select your other id from the list. (You can just make sure it's checked and hit ESC or click outside of the field to apply it.) Make sure your other userid shows up in the Reviewers section now.

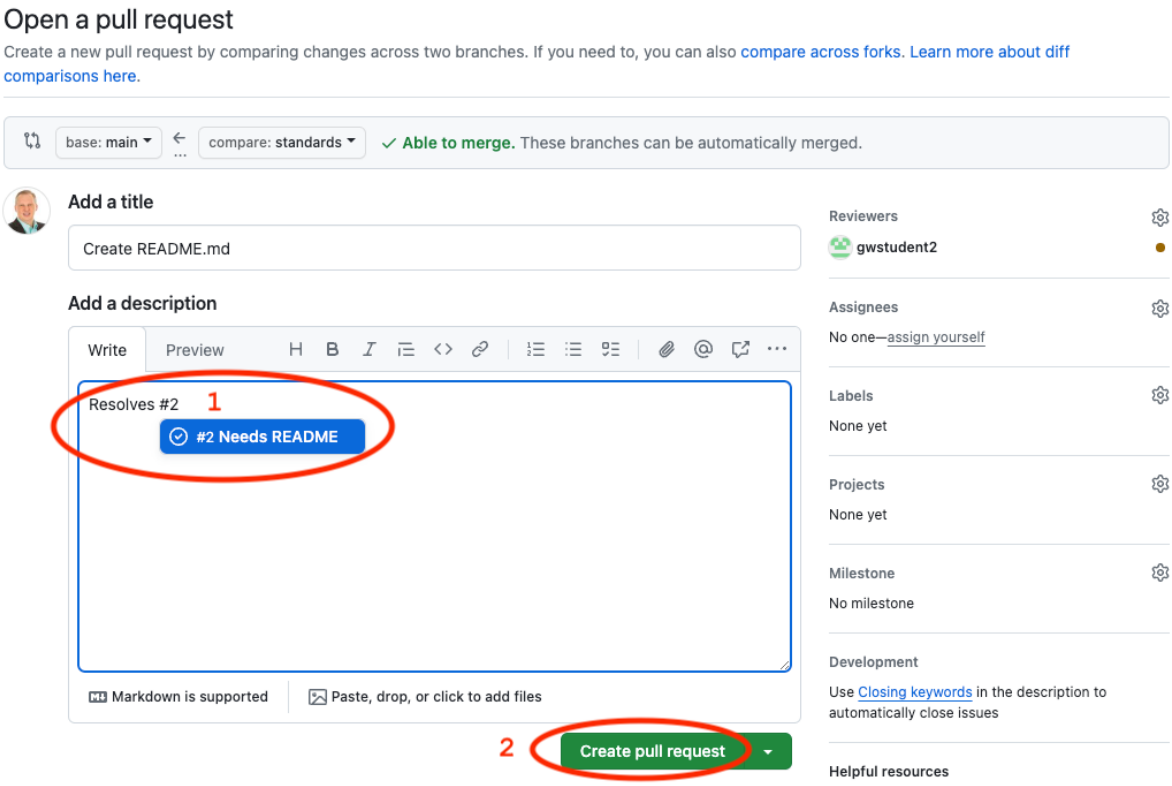


7. Also, we can add in a description that will automatically close the associated issue when we resolve this pull request. Click in the "Add your description here..." field and enter

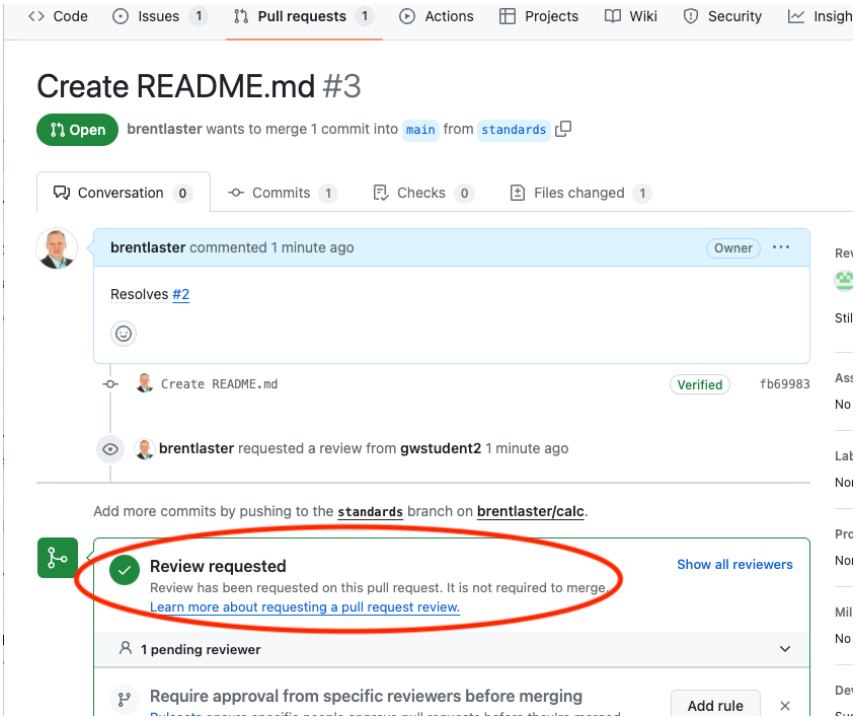
Resolves #2

If you have a different issue number, change the 2 to your issue number.

Then click on the "*Create pull request*" button.



8. Afterwards, you'll be on the screen for the open pull request. Around the middle of the screen, you can see the conditions that need to be satisfied before the pull request can be merged. This includes the pending review you have from your secondary GitHub userid.

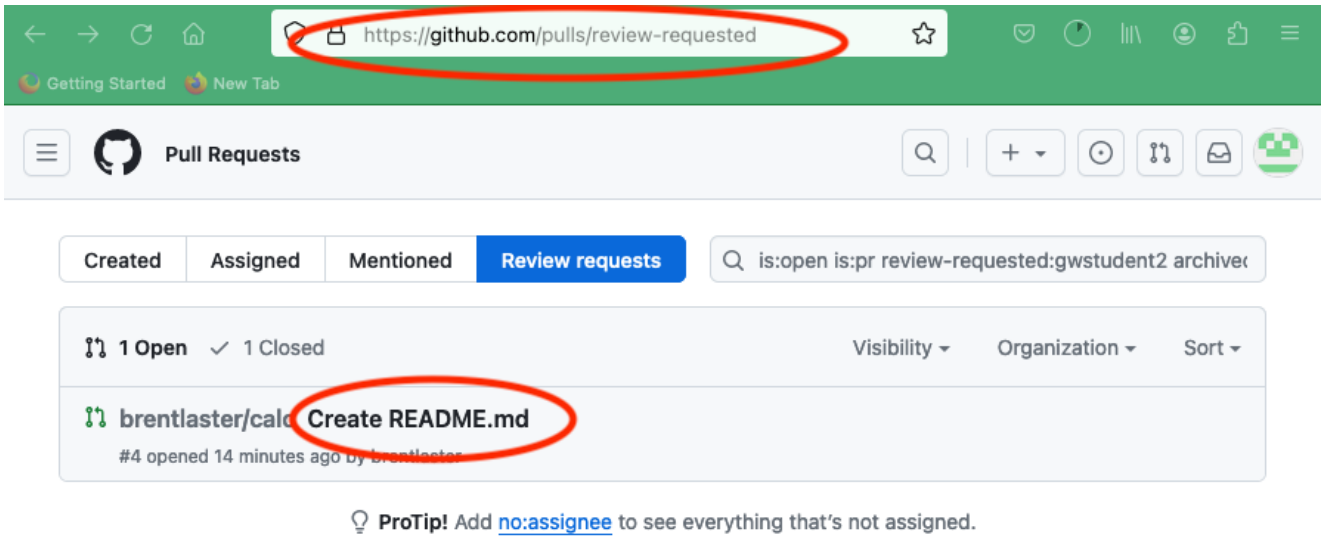


END OF LAB

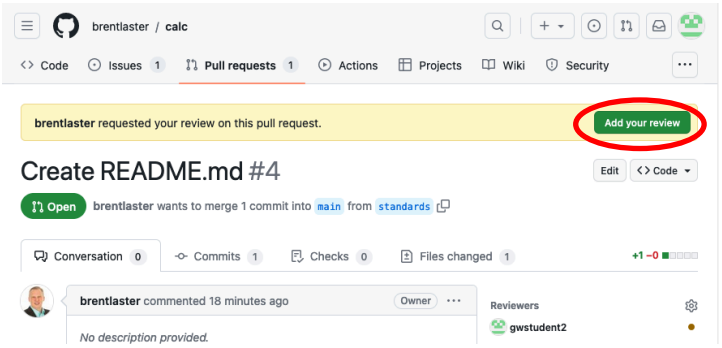
Lab 5: Completing a pull request with reviewers

Purpose: *In this lab, we'll complete the pull request we started in the last lab.*

1. In a separate browser or a private tab, log in to your secondary GitHub userid (the one you added as a collaborator and a reviewer). After you log in, you go to https://github.com/pulls/review-requested . Then click on the commit message for the pull request.



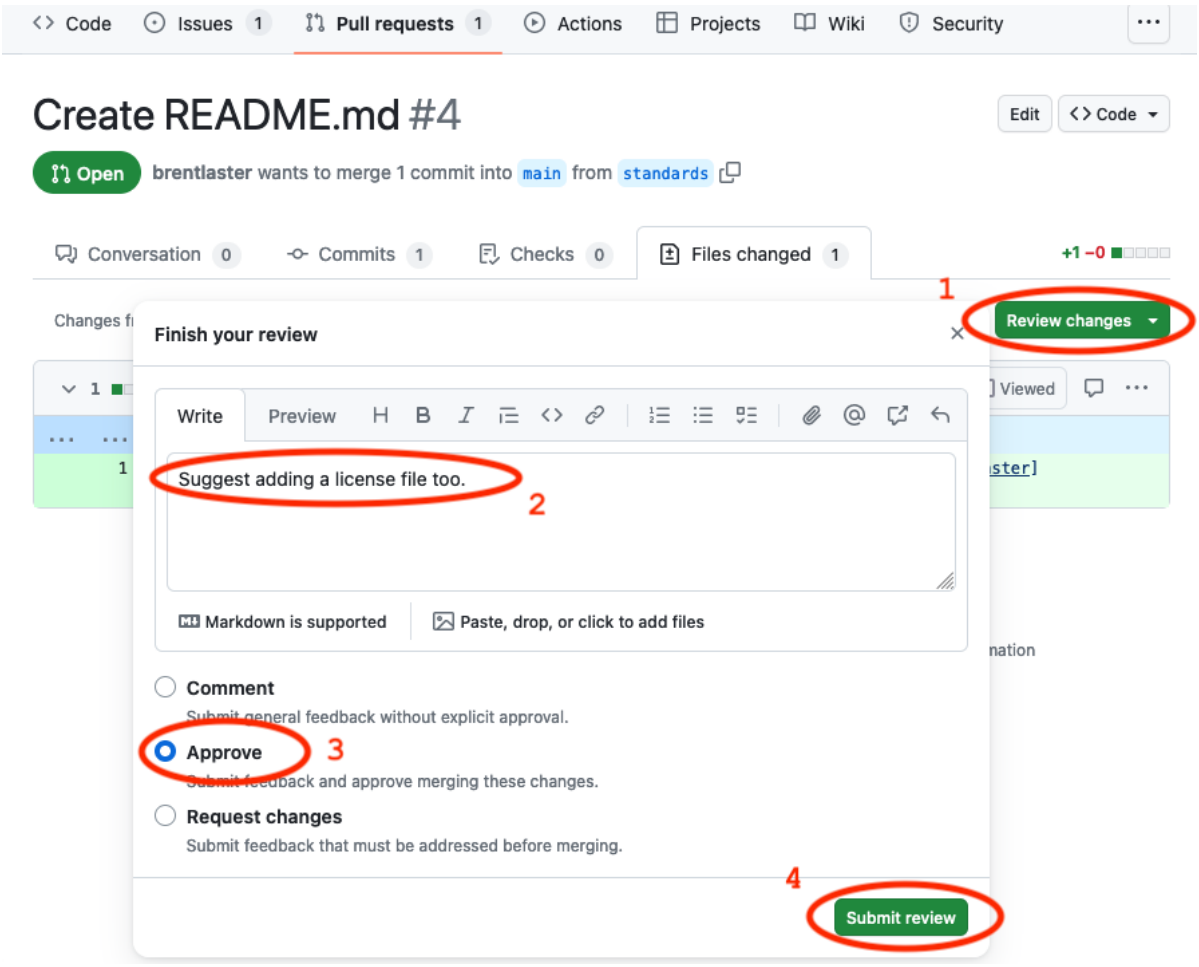
2. This will open up the pull request. There is a button at the top to “Add your review”. Click on that.



3. We could click on any of the lines and add a comment if we wanted, but since this is simply adding a README file, it looks ok. However, since this is about standards, let’s make a suggestion to also add a license for the repo.

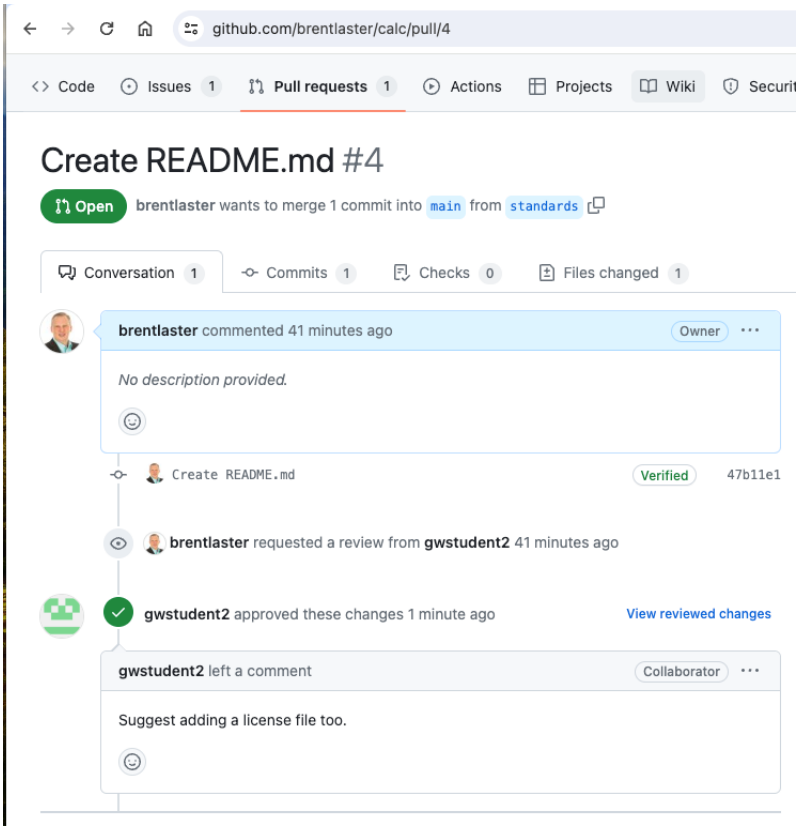
   Select the “*Review changes*” button and add a comment to that effect.

   Then select the “Approve” option, and then “*Submit review*”.



4. Go to the session with your original GitHub userid or log out of the other one and log back in if you need to.

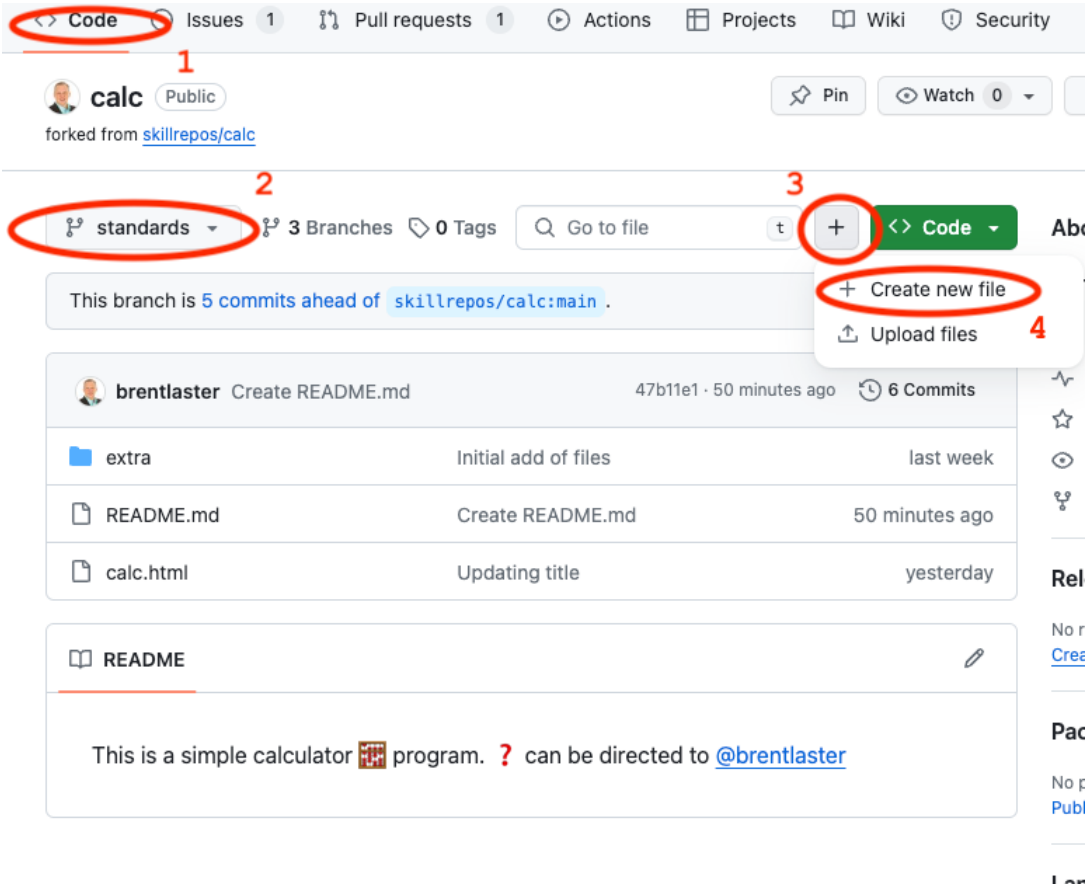
   Go to the ***Pull requests*** menu at the top, find the pull request and click on the commit message. Then you should see a screen like below. (Some icons and wording may look different.)



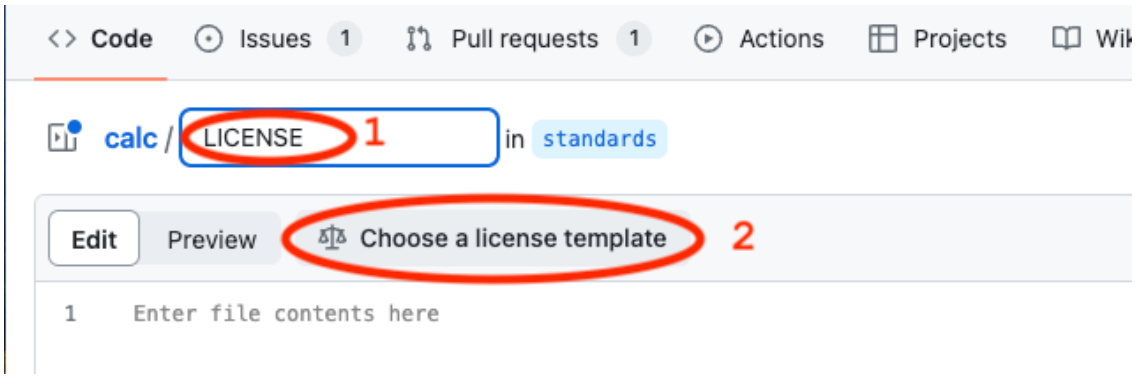
5. Since there was a suggestion to add a license file, that sounds like a good idea, so let's do that.

   Click on the ***Code*** tab at the top, then select the ***standards*** branch (or whatever name you gave the new branch) from the branch dropdown.

   Then select the "**+**" sign (or "*Add file*" option) and the option to ***+ Create new file***.

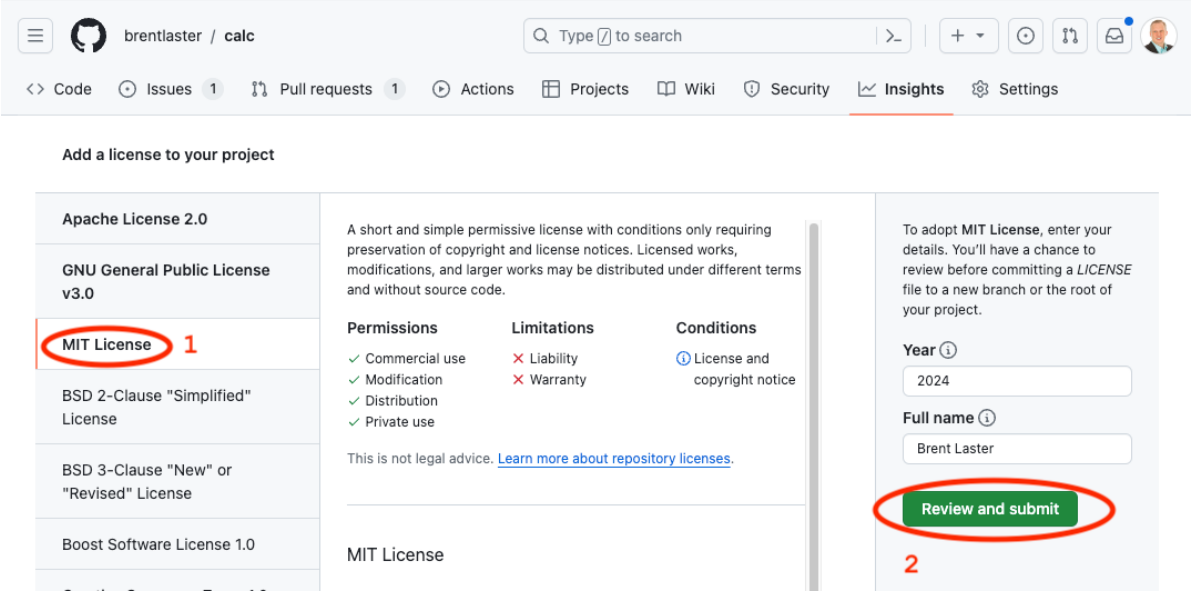


6. In the next screen, there will be a text entry area for the name of the file. Type in "LICENSE" for the name. Then, an option will display that says ***Choose a license template***. Click on that option. You will be asked about discarding changes. It's ok in this case, so click on "OK".



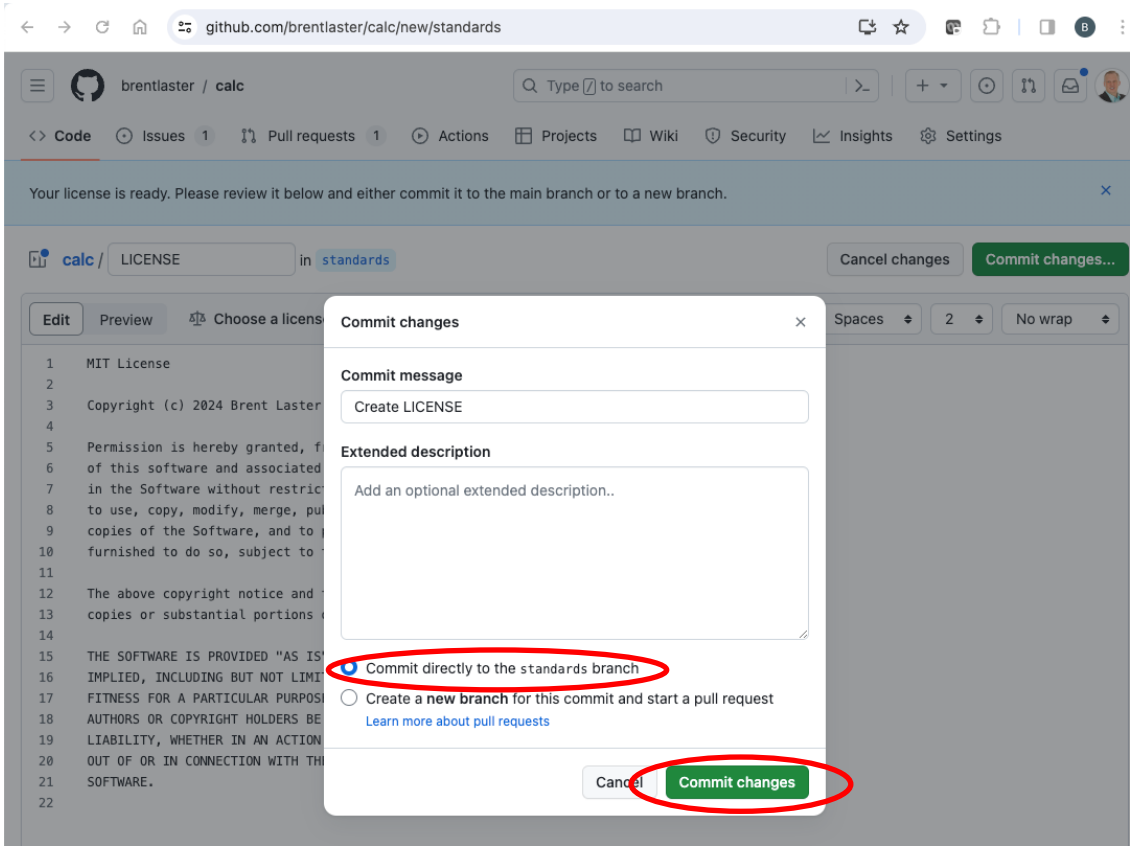
7. On the next screen, you'll be able to pick the license you want. You can select the "MIT License" or another one if you prefer.

   Once done, click the "*Review and submit*" button on the right.



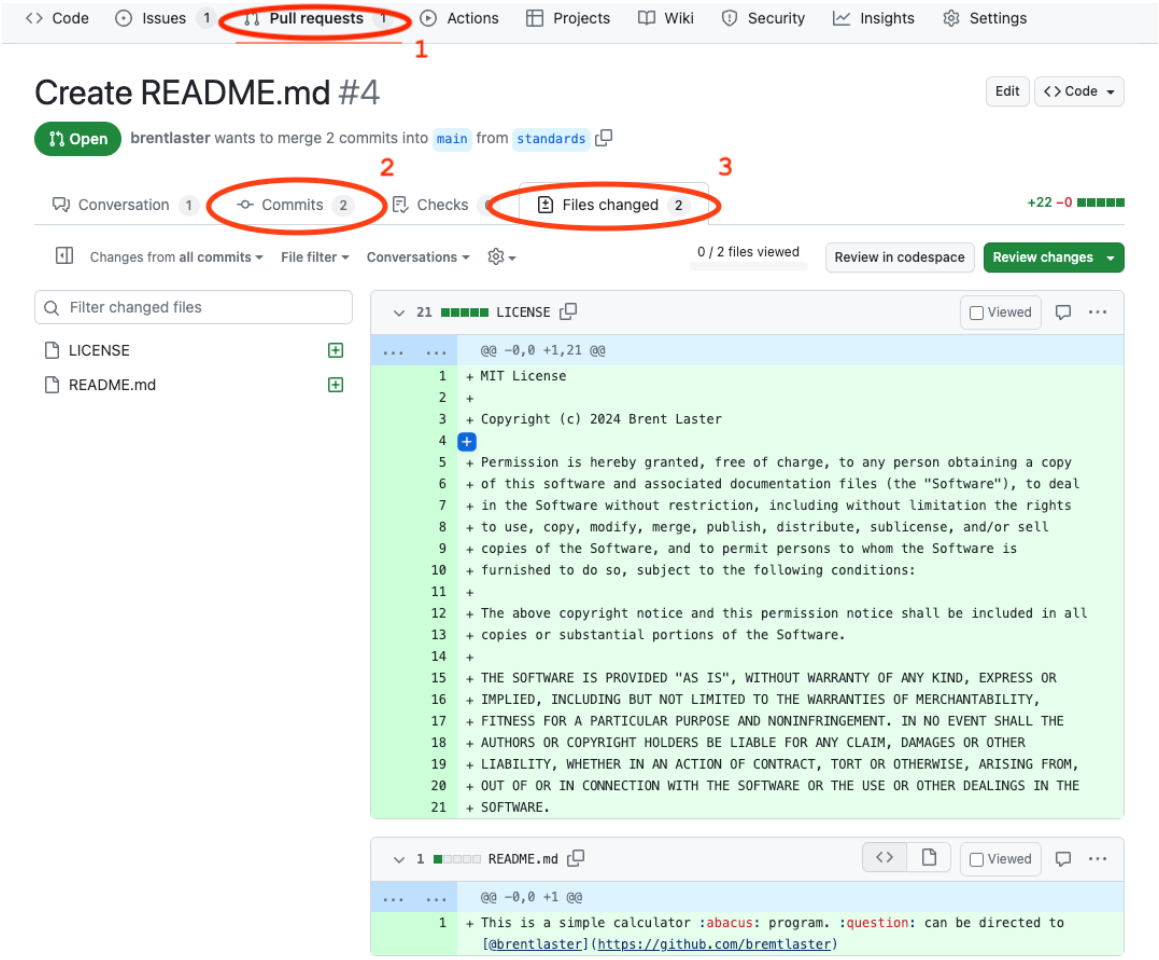
You'll have an opportunity to review the license. When ready, just click on the ***Commit Changes*** buttons to commit the file to the *standards* branch.

<u>Be sure to leave it on the *standards* branch so it will be added to the existing pull request.</u>



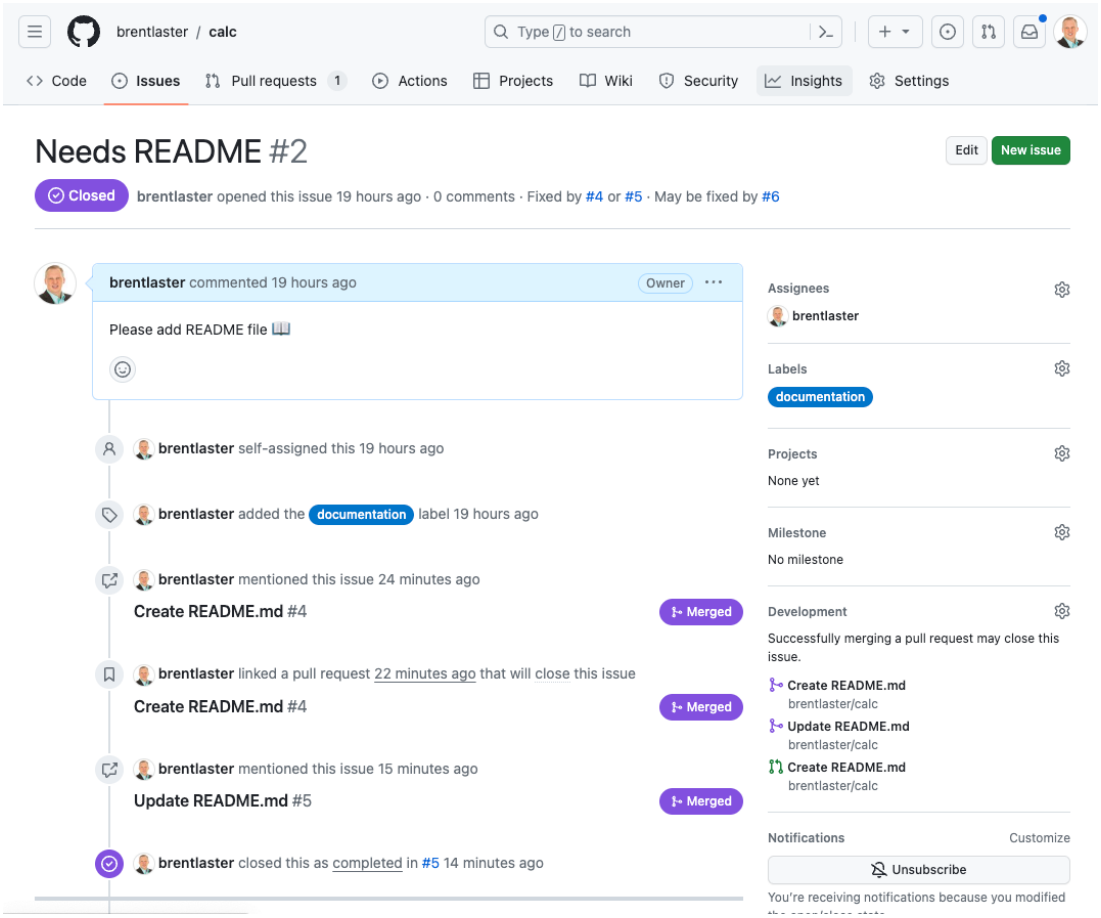
8. Go back to the pull request by selecting ***Pull requests*** at the top and selecting the one open pull request.

   You can look at the changes currently in the pull request by clicking on the ***Commits*** tab and also the ***Files changed*** tab.



9. Click back on the ***Conversation*** tab in the pull request and go ahead and merge (*Merge pull request)* and close (*Confirm merge*) the pull request.

   After completing the merge, you should be able to click on the ***Issues*** tab and see that your issue has been automatically closed. You can click on the ***Closed*** list and then open the issue to see the automatically generated log of comments and actions if you want.



END OF LAB

Lab 6: Adding a GitHub Pages website for your repository

Purpose: *In this lab, we'll setup a GitHub Pages repo for your repository.*

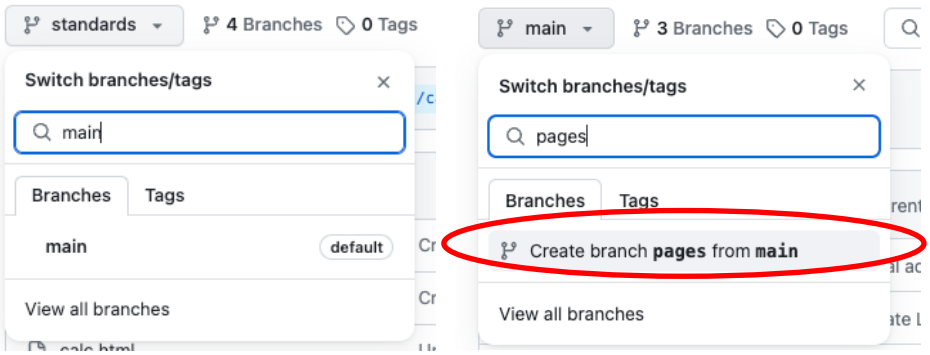
1. This lab is done with your primary GitHub id.

   In order to prepare for publishing a page, let's create a new branch in our repo.

   In the ***Code*** tab, if not on the ***main*** branch, click on the branch dropdown, type in ***main*** and select it.

   Click in the branch dropdown again, and, in the text area that says, ***Find or create a branch...***, enter the text ***pages***.

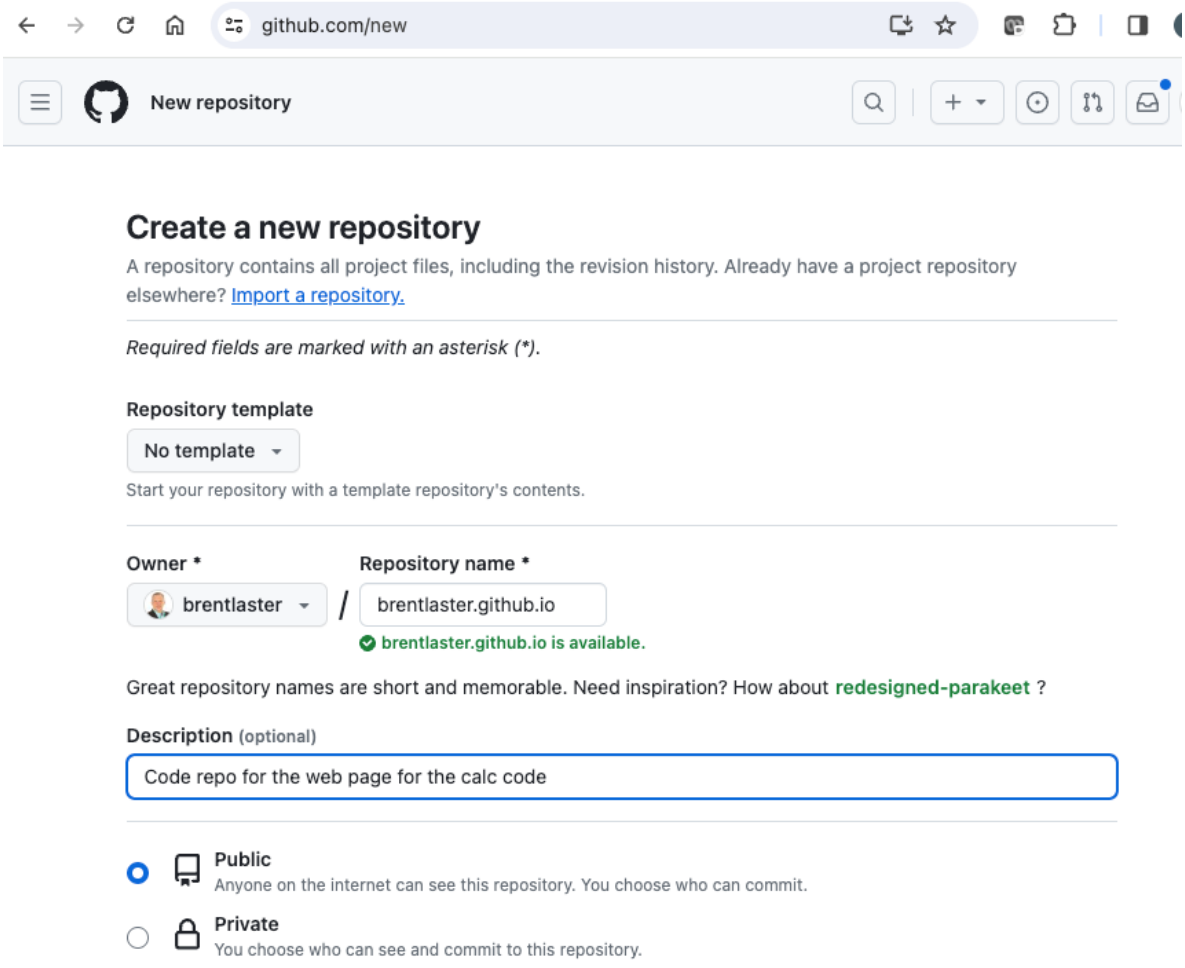
   Then click on the "*Create branch:* ***pages*** *from* ***main***" link.



2. Now create a new repository to use for the "*pages*" repo.

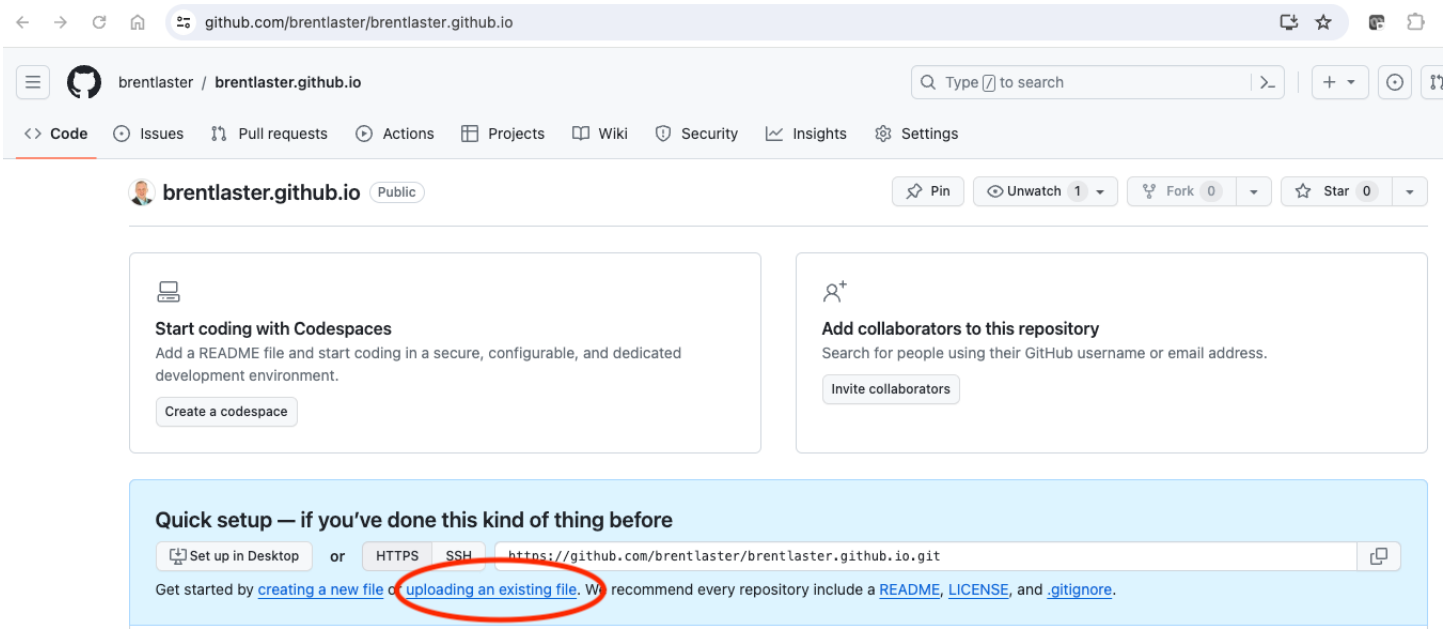
Go to https://github.com/new to create a new repository. (Alternatively, you could go to your home page, then to ***Repositories***, then to ***New***.)

Refer to the picture below. Name the new repository precisely ***<github-userid>.github.io*** replacing your actual GitHub userid for the <*github-userid*> item. (This will look like a repeat of your userid since the project has your userid in the name in your github userid space.) You can optionally add a description if you want.

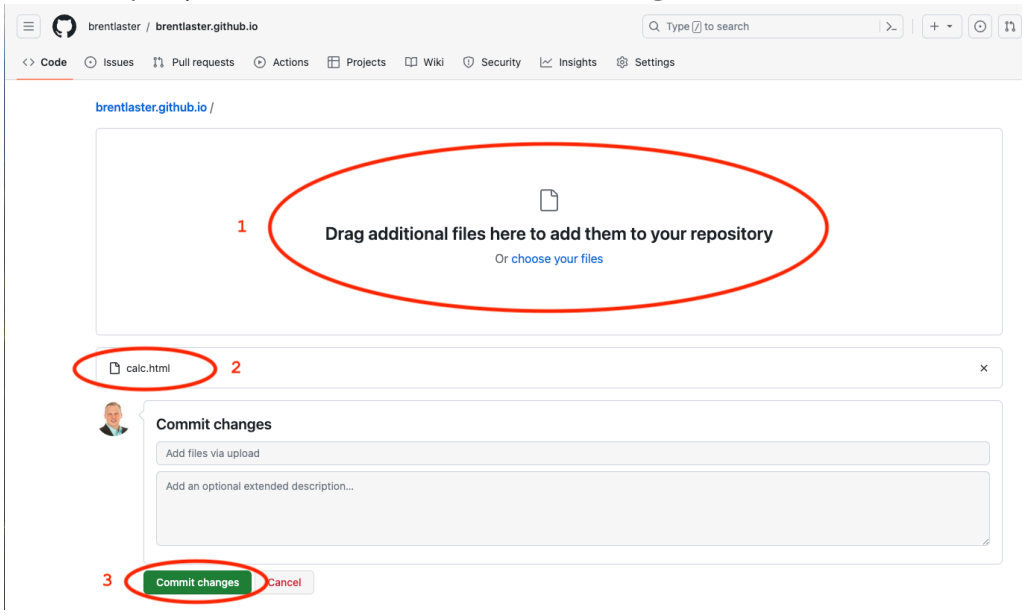


When done, click on the ***Create repository*** button at the bottom of the page.

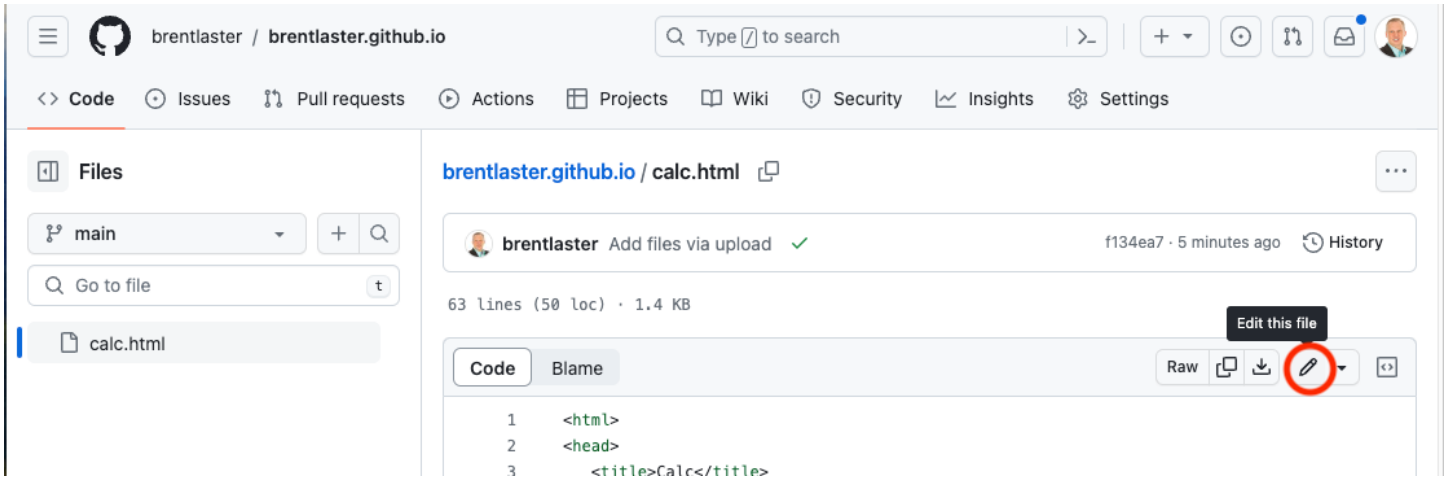
3. So that we have content to publish, we'll grab the code from the calc.html file in your local repository from Lab 1 and add it here. On the screen with the ***Quick setup – if you've done this kind of thing before*** instructions, click on the link in the big blue bar for ***uploading an existing file***.



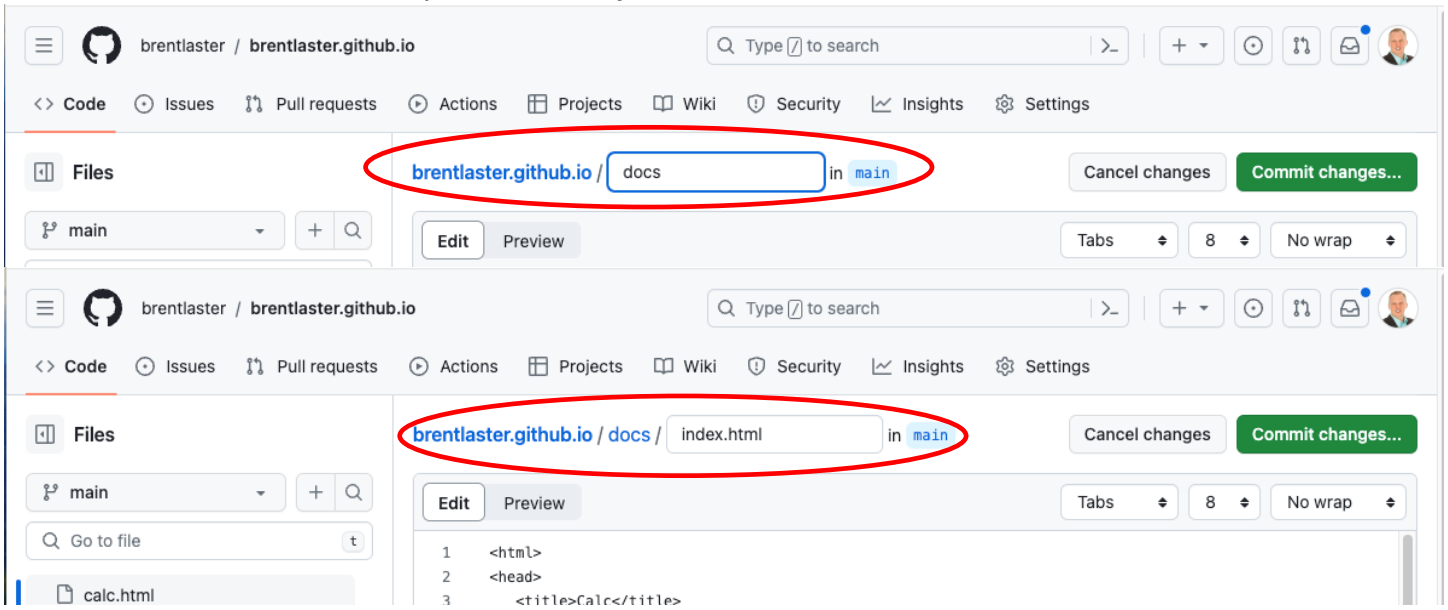
4. On the "upload" screen, drag the file *calc.html* from your local directory (where you cloned it in Lab 1) to the indicated area -or- click on the ***choose your files*** link and browse out and select the file (and click *Open*). Then click on the ***Commit changes*** button to add the file to the repo.



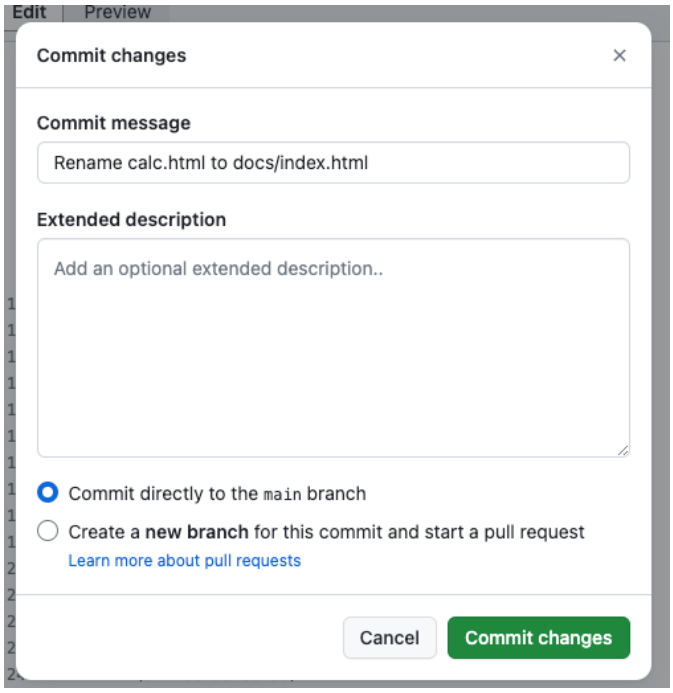
5. You should now be back in the new repo's ***Code*** tab. Let's change the name and location of this file to make it more consistent for GitHub Pages. Click on the *calc.html* file and then click on the "pencil" icon to edit it.



5. In the edit dialog, in the text entry box for the name, type over the "calc.html" text with the replacement text of "docs/index.html". Note you are adding the directory *docs* to the path. The screenshots show this in 2 parts for clarity.



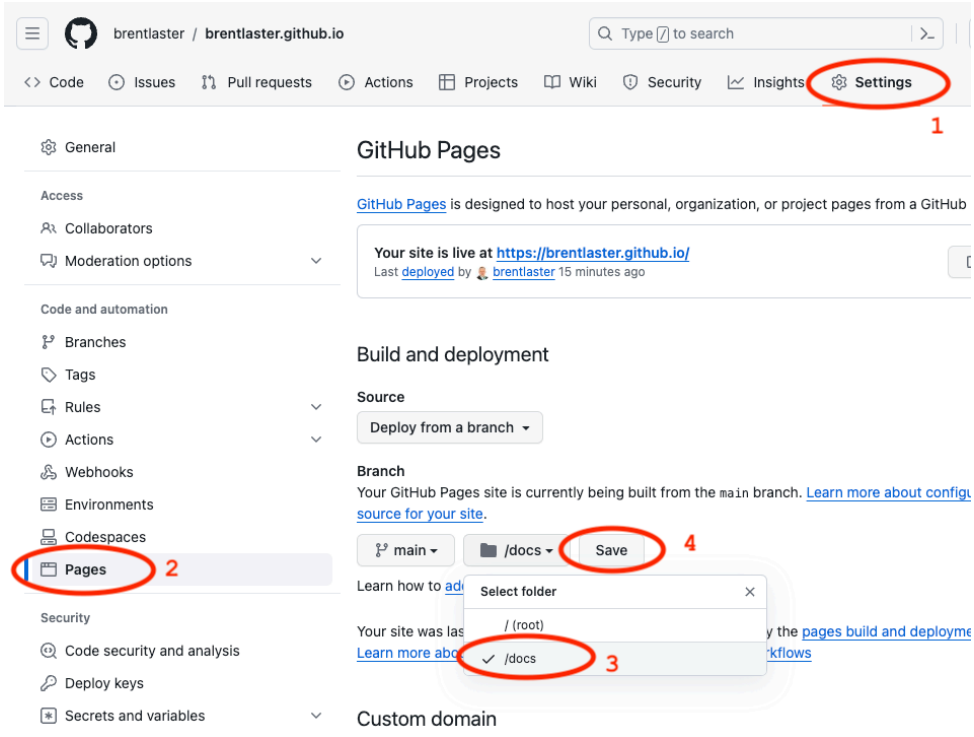
6. Commit your changes for the rename directly to the *main* branch by clicking on the ***Commit changes...*** button.



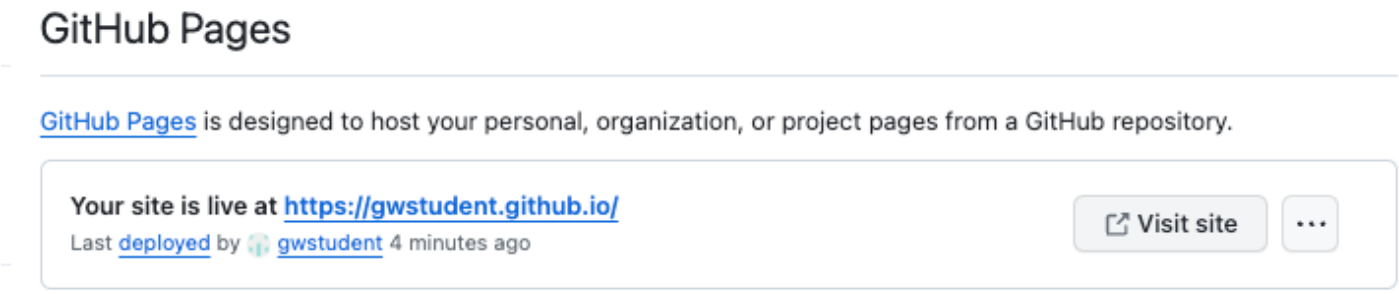
7. Now we need to set the source from the repo for the web page. Go to the repo's ***Settings*** tab.

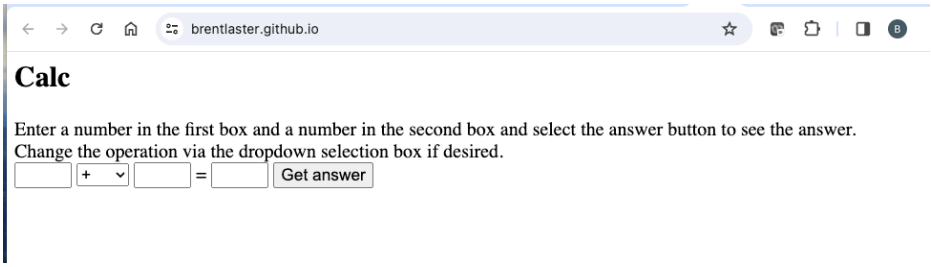
   On the left side, select the ***Pages*** entry.

   Under the ***Build and deployment/Branch*** section, under the folder dropdown, select ***main*** on the left and then ***/docs*** entry on the right and then click the ***Save*** button.



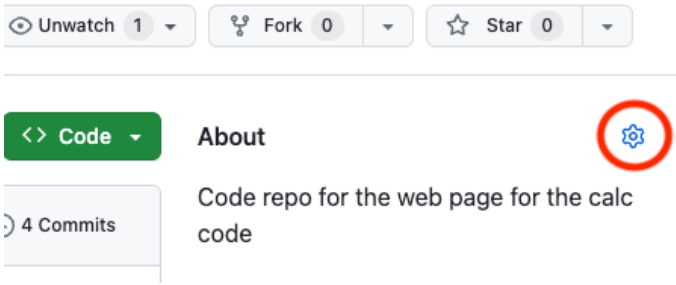
8. After these changes, you can visit the site at https://<github-userid>.github.io and using the button in the page or just go to the URL to see the automatic web page.



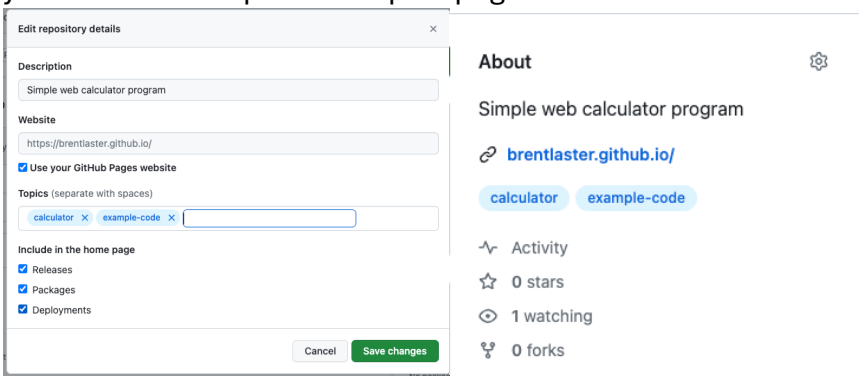


OPTIONAL:

9. Change the displayed metadata about the github.io repo to show more details about the project. On the repo's ***Code*** page, on the right side, click on the gear icon next to ***About***.



10. Add repository details such as the ones below. For the Website, you can just click the checkbox. For Topics, just start typing in the field. Once you are done, click the ***Save changes*** button and you should see your edits show up on the repo's page.



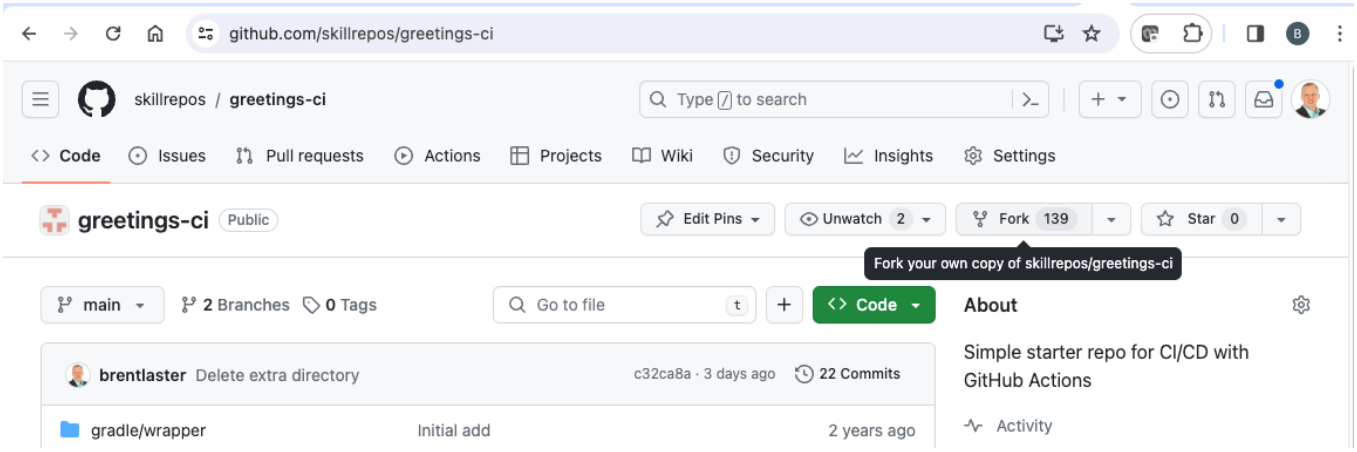
END OF LAB

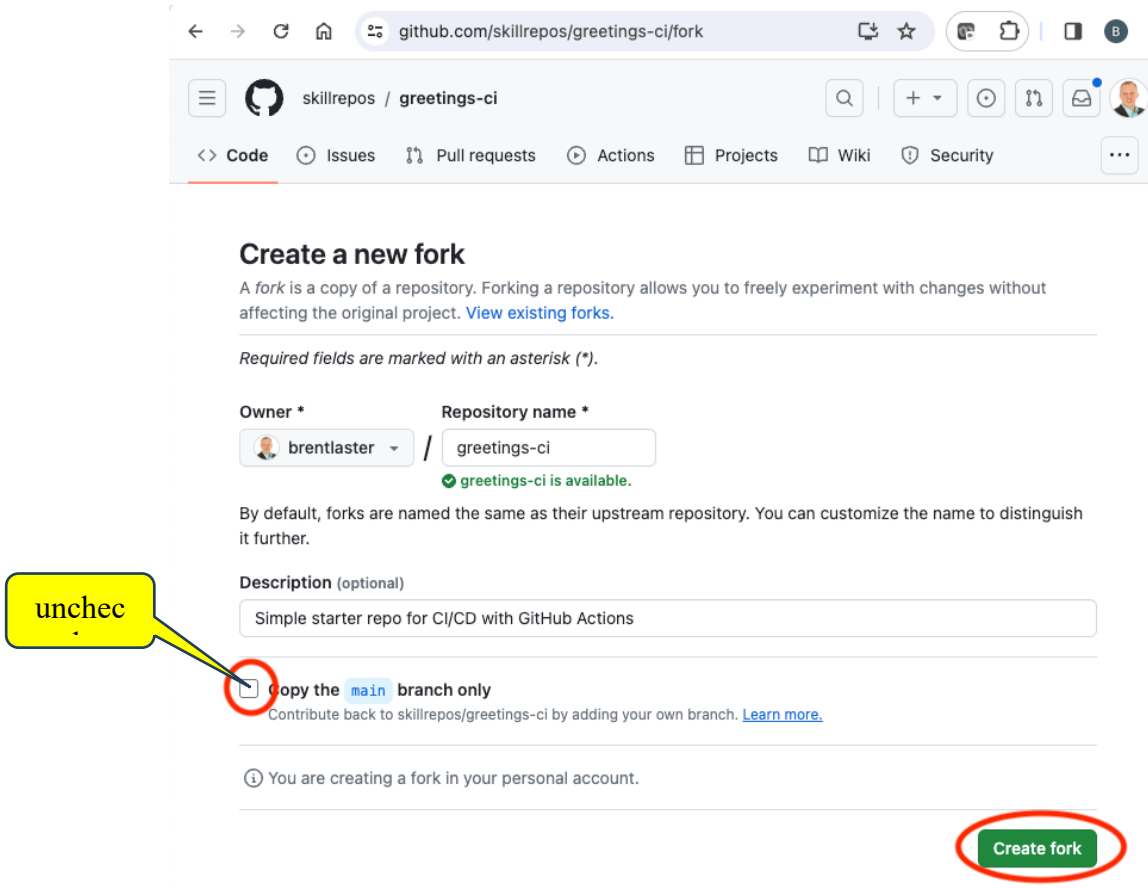
Lab 7: Learning about GitHub Actions

Purpose: *In this lab, we'll learn about how GitHub Actions can be used to automate workflows for repositories.*

1. Start out in GitHub with your primary GitHub account.

2. Go to https://github.com/skillrepos/greetings-ci and fork that project into your own GitHub space. After this, you'll be on the project in your user space. **Make sure again to uncheck the box next to *Copy the main branch only*,** so that both branches will be included in the fork.



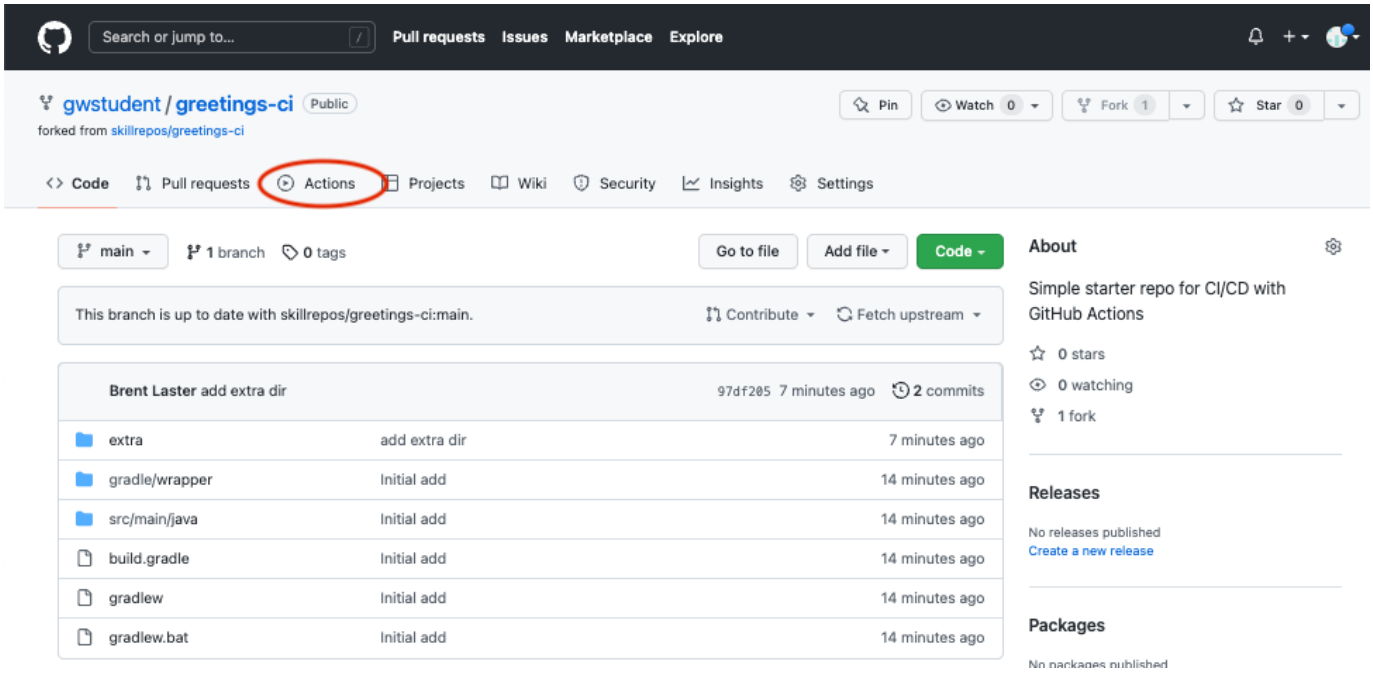


3. We have a simple java source file named *echoMsg.java* in the subdirectory *src/main/java*, a Gradle build file in the root directory named *build.gradle,* and some other supporting files.

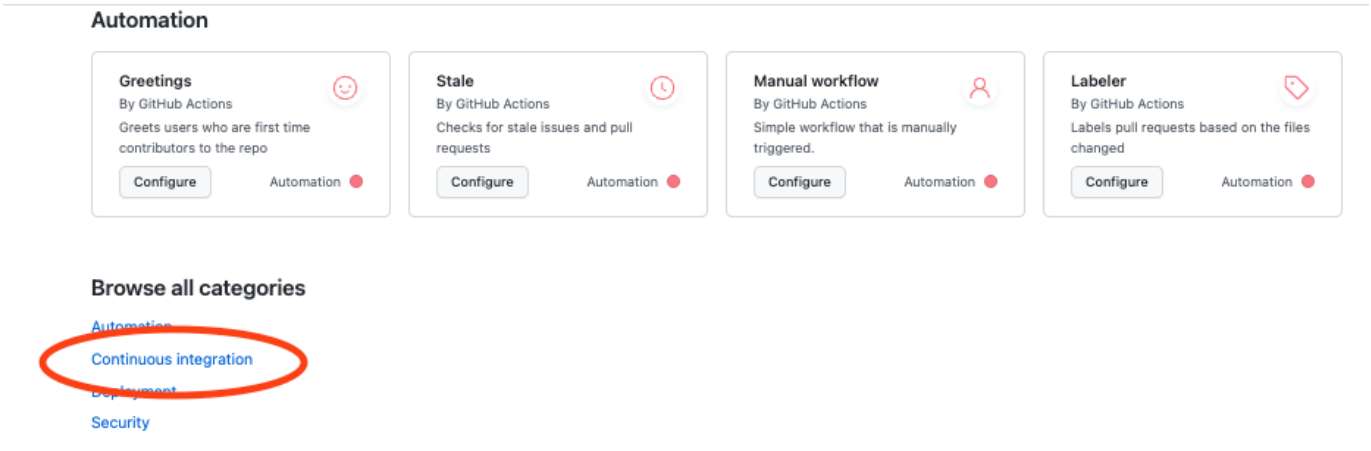
   We could clone this repository and build it manually via running Gradle locally.

   But let's set this to build with an automatic CI process specified via a text file.

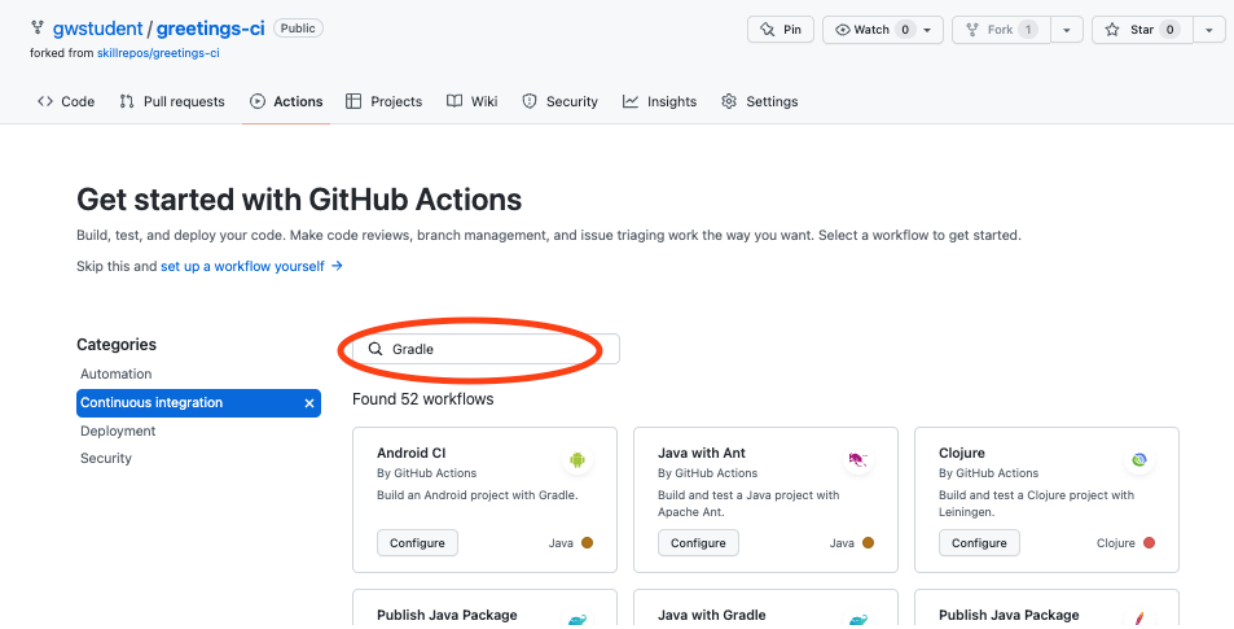
   On the ***Code*** tab, click on the ***Actions*** button in the top menu under the repository name.



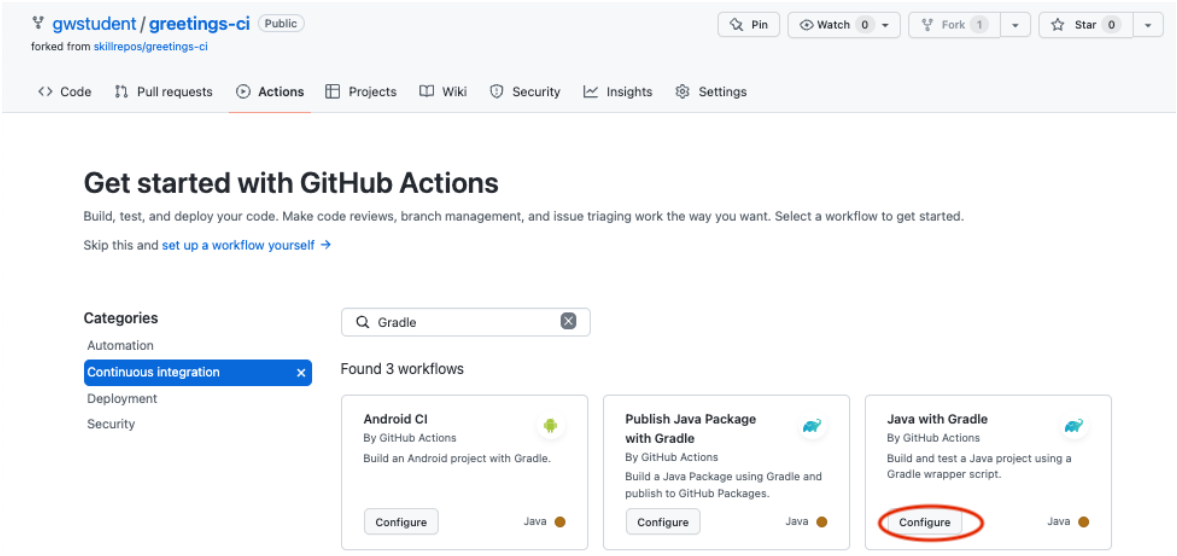
4. This will bring up a page with categories of starter actions that GitHub thinks might work based on the contents of the repository. We'll select a specific CI one. Scroll down to near the bottom of the page under ***Browse all categories*** and select ***Continuous integration***.



5. In the CI category page, let's search for one that will work with Gradle. Type *Gradle* in the search box and press Enter.



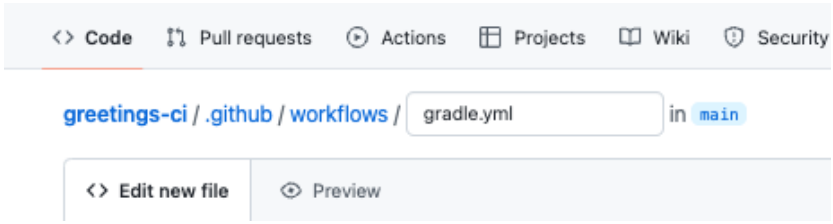
6. From the results, select the ***Java with Gradle*** one and click the ***Configure*** button to open a predefined workflow for this.



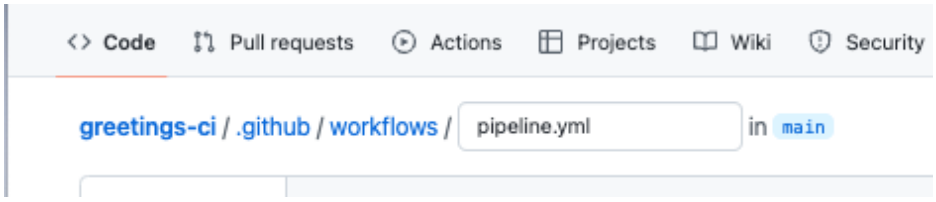
7. This will bring up a page with a starter workflow for CI that we can edit as needed. We need to make two edits here. The first edit is to change the name.

   In the top section where the path is, notice that there is a text entry box around *gradle.yml*. This is the current name of the workflow. Click in that box and edit the name to be *pipeline.yml*. (You

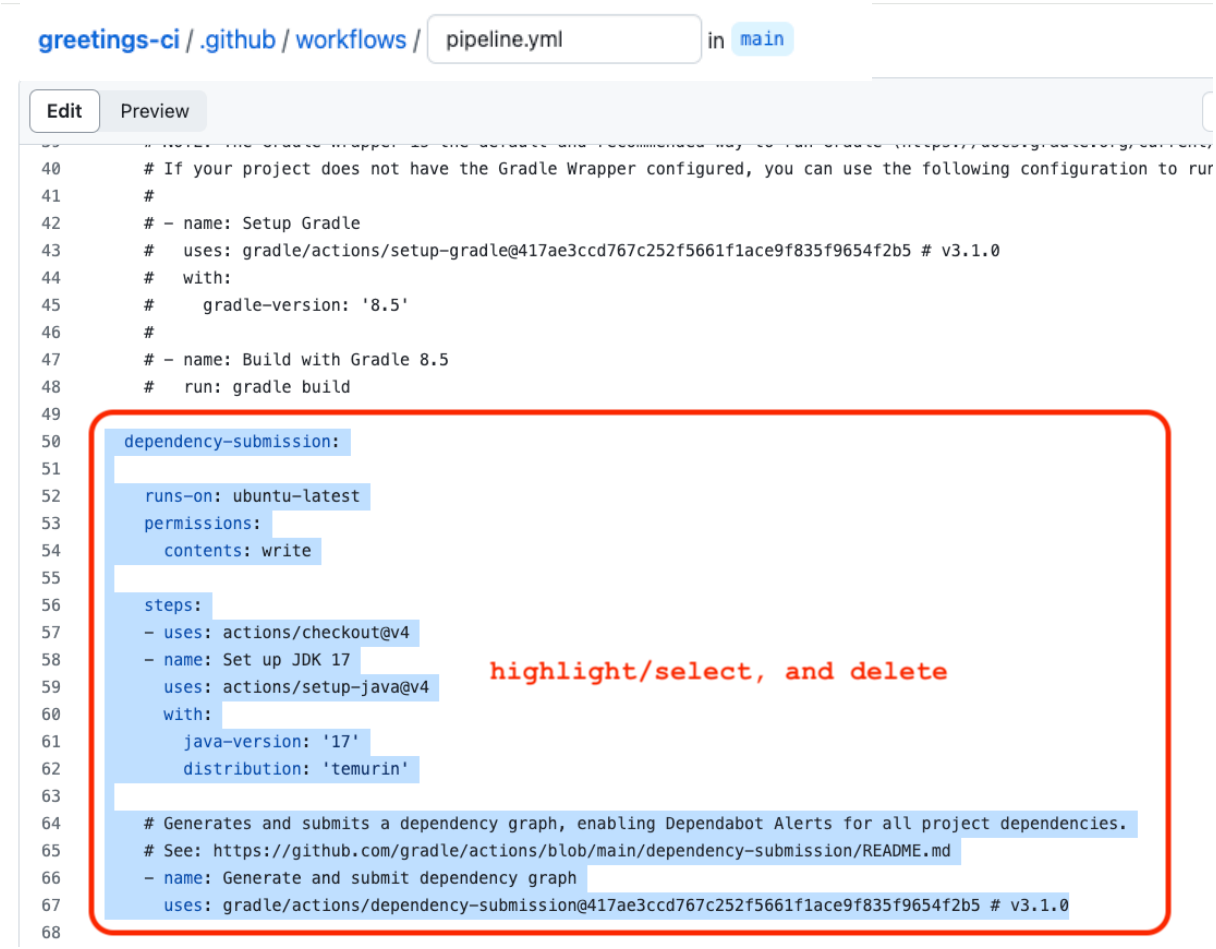
can just backspace over or delete the name and type the new name.)



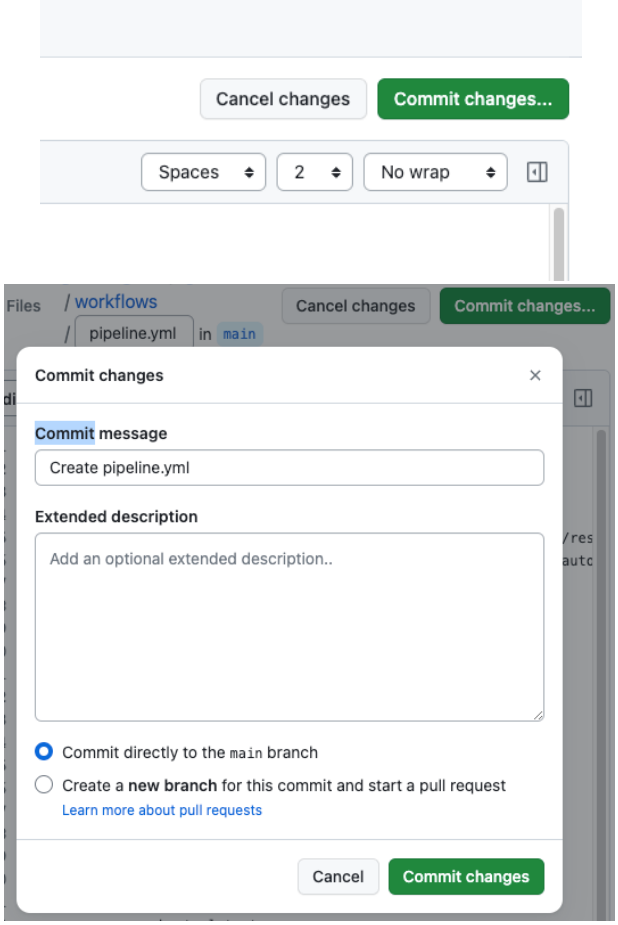
TO



8. The second edit is to remove the second job in this workflow since it would require some additional setup. To do this we will just highlight/select the code from line 50 on and hit delete. (*If you have trouble just selecting that code, try starting at the bottom and selecting/highlighting from the bottom up.)* The code to be deleted is highlighted in the next screenshot.

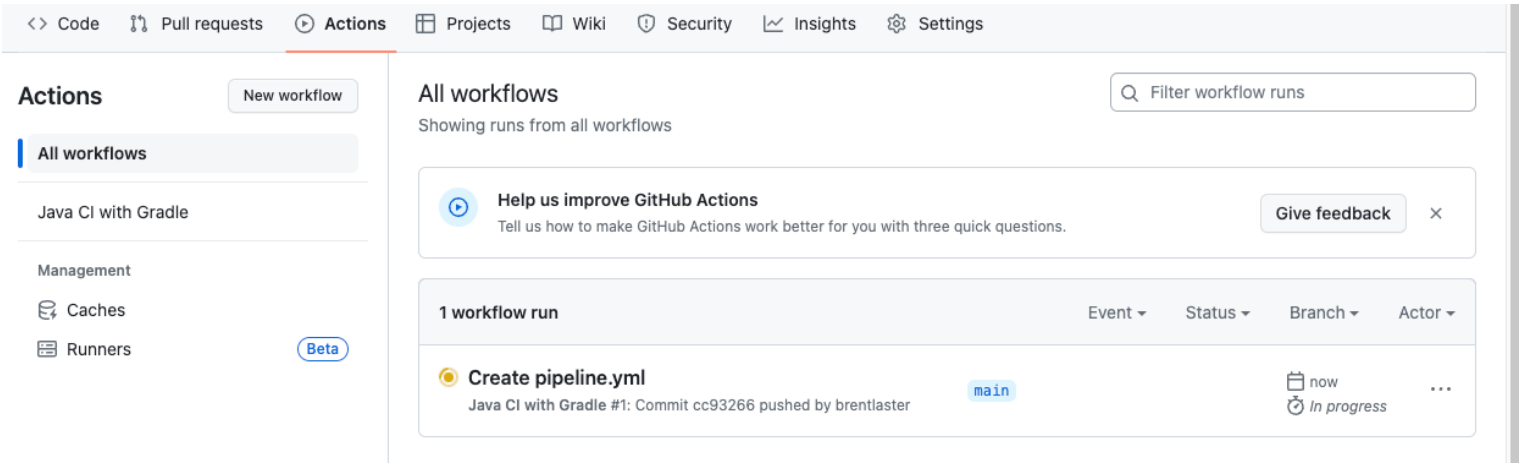


9. Now, we can go ahead and commit the new workflow via the ***Commit changes...*** button in the upper right. In the dialog that comes up, you can enter an optional comment if you want. Leave the **Commit directly...** selection checked and then click on the ***Commit changes*** button.

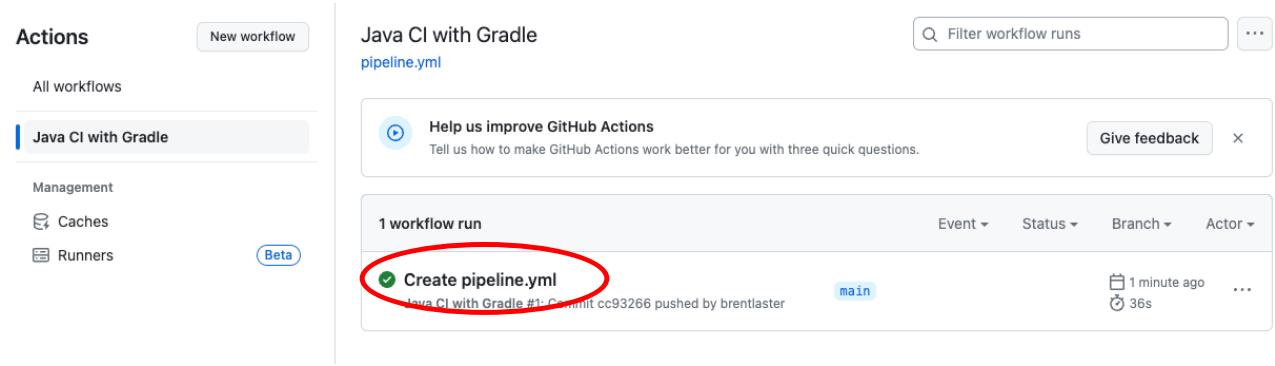


10. Since we've committed a new file and this workflow is now in place, the *on: push:* event is triggered and the CI automation kicks in.

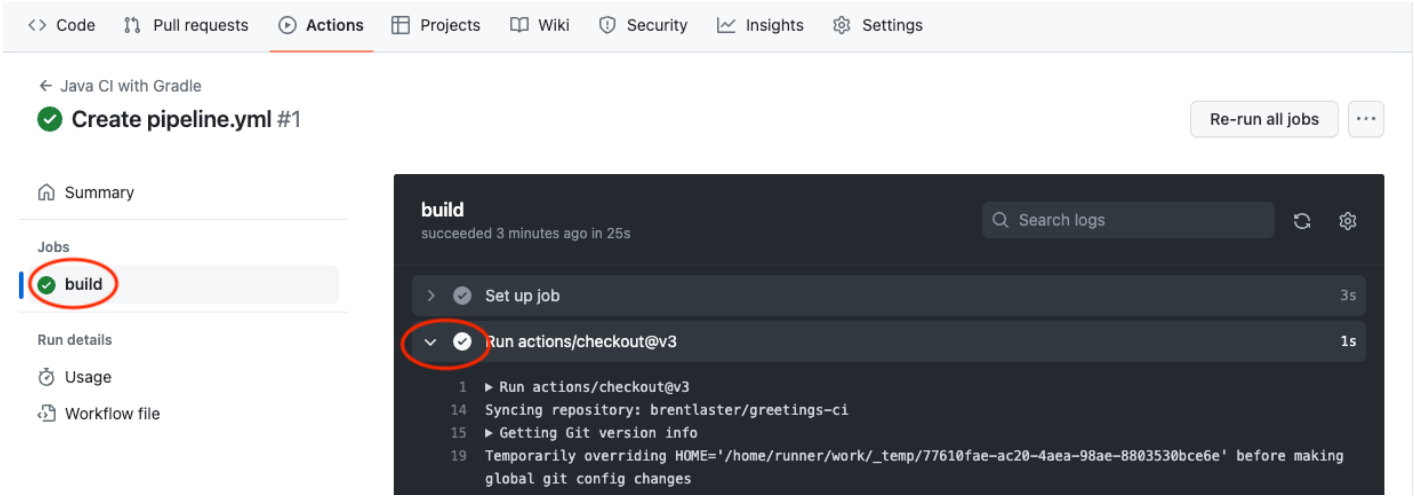
    Click on the ***Actions*** menu again to see the automated processing happening. (You may have to wait a moment for it to start.)



11. After a few moments, the workflow should succeed. (You may need to refresh your browser.) After it is done, you can click on the commit message for the run to get to the details for that particular run.



12. From here, you can click on the build job in the graph or the *build* item in the list of jobs to get more details on what occurred on the runner system. You can expand any of the steps in the list to see more details.



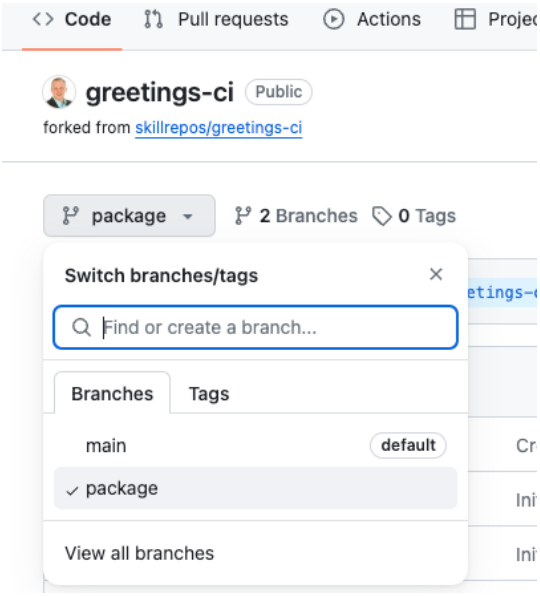
END OF LAB

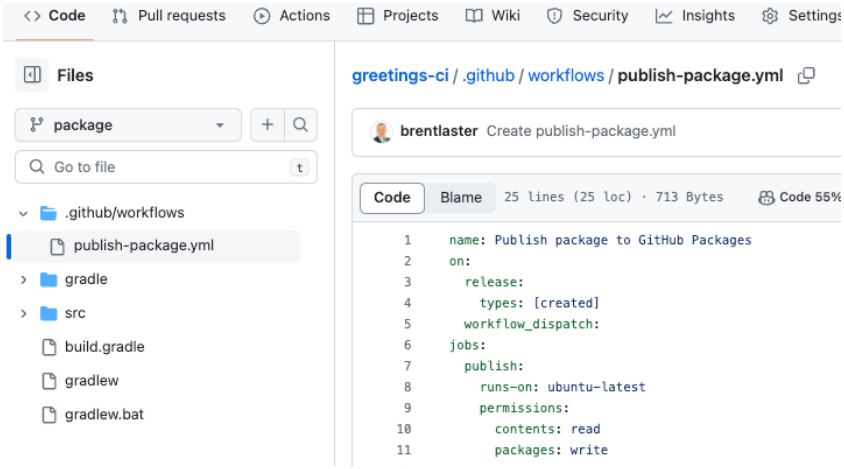
Lab 8: Creating packages

Purpose: *In this lab, we'll see how to create GitHub packages.*

1. We'll continue working in your fork of the ***greetings-ci*** repo under your primary userid.

   In a separate branch named ***package***, we have an updated *build.gradle* file and a new Actions workflow file - *.github/workflows/publish-package.yml*. You can switch to the *package* branch and look at those if you want. (You don't need to make any changes.)

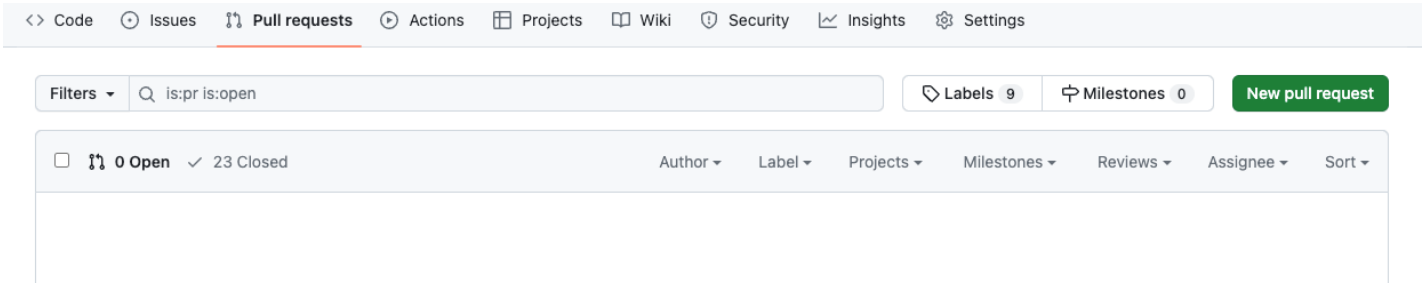


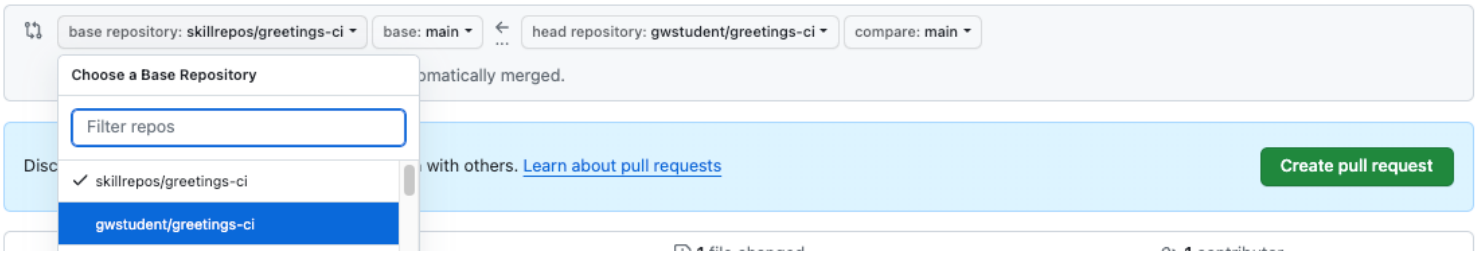


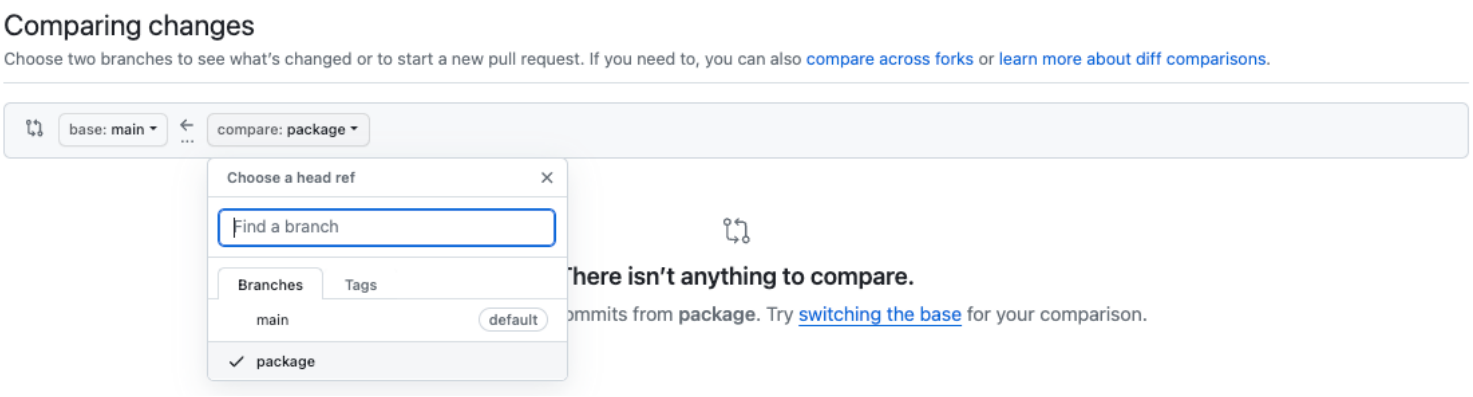
2. Let's create a pull request to merge those into *main*.

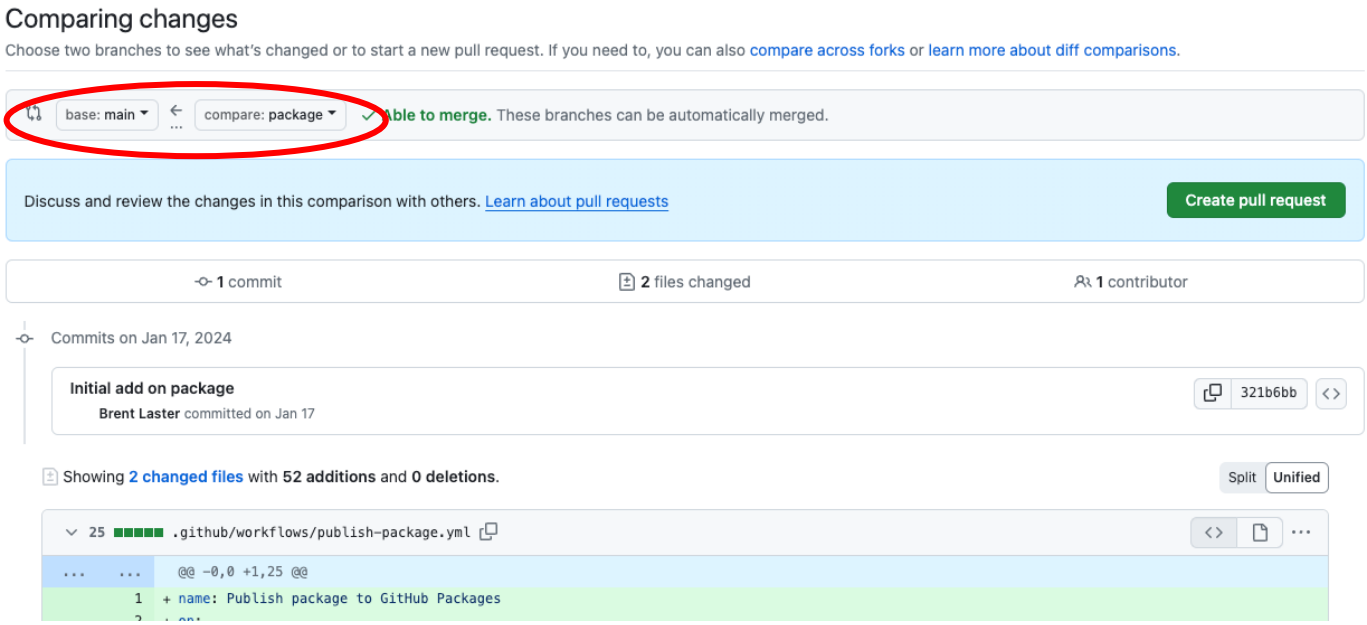
   Go the Pull Requests menu and open a new pull request (via the button) to merge the *package* branch into the *main* branch - **on your fork** NOT skillrepos/greetings-ci. Make sure to set the ***base repository = <your repo> main*** and ***compare = package*** in the gray bar so you are merging in the same repo and **NOT** into skillrepos/greeting-ci.

   After you make those changes, go ahead and create the pull request (by clicking through the ***Create pull request*** buttons on the screens).

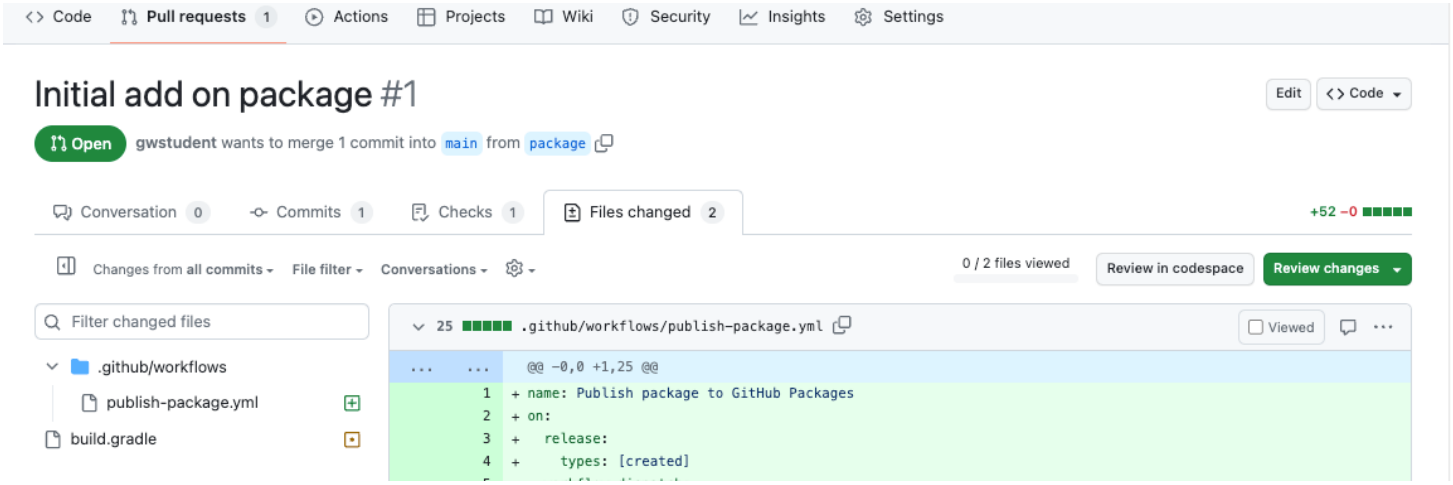






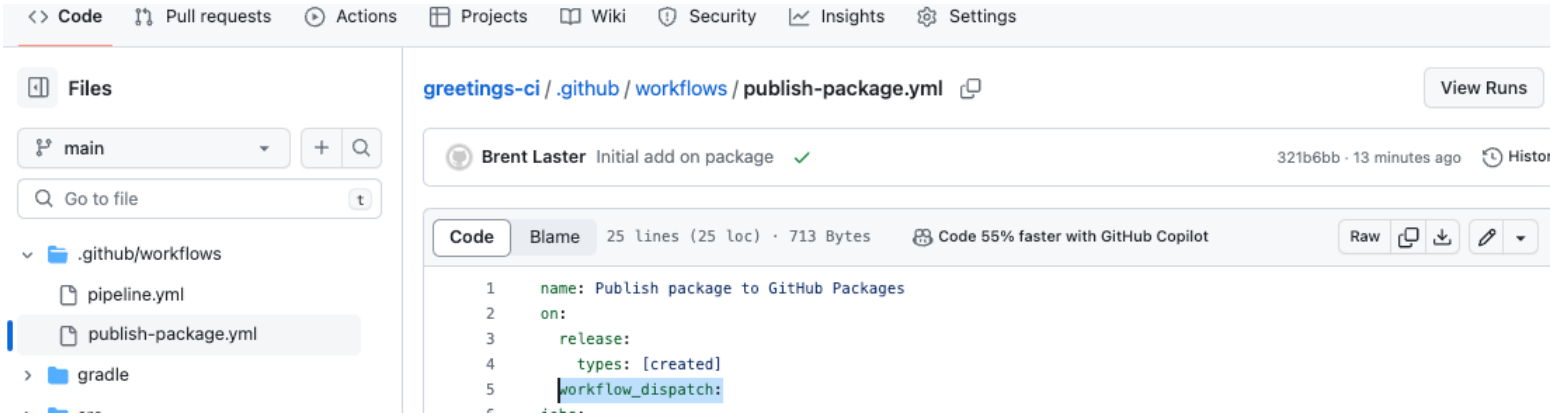


3. Look at the pull request and review the changes we've made to publish the package via the *Commits* and *Files changed* tabs.

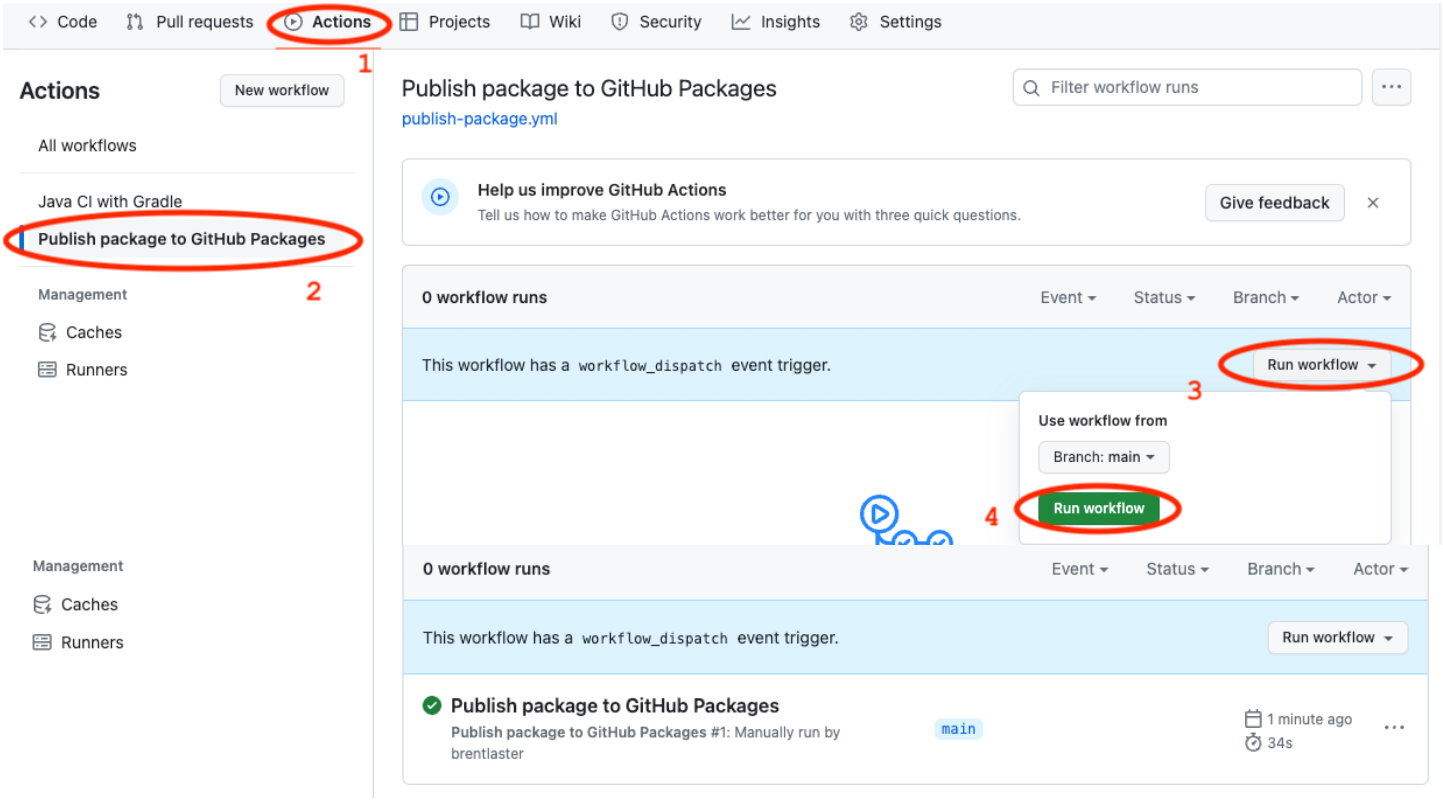


4. Back in the ***Conversation*** tab, merge the pull request. You can choose to delete the ***package*** branch or not.

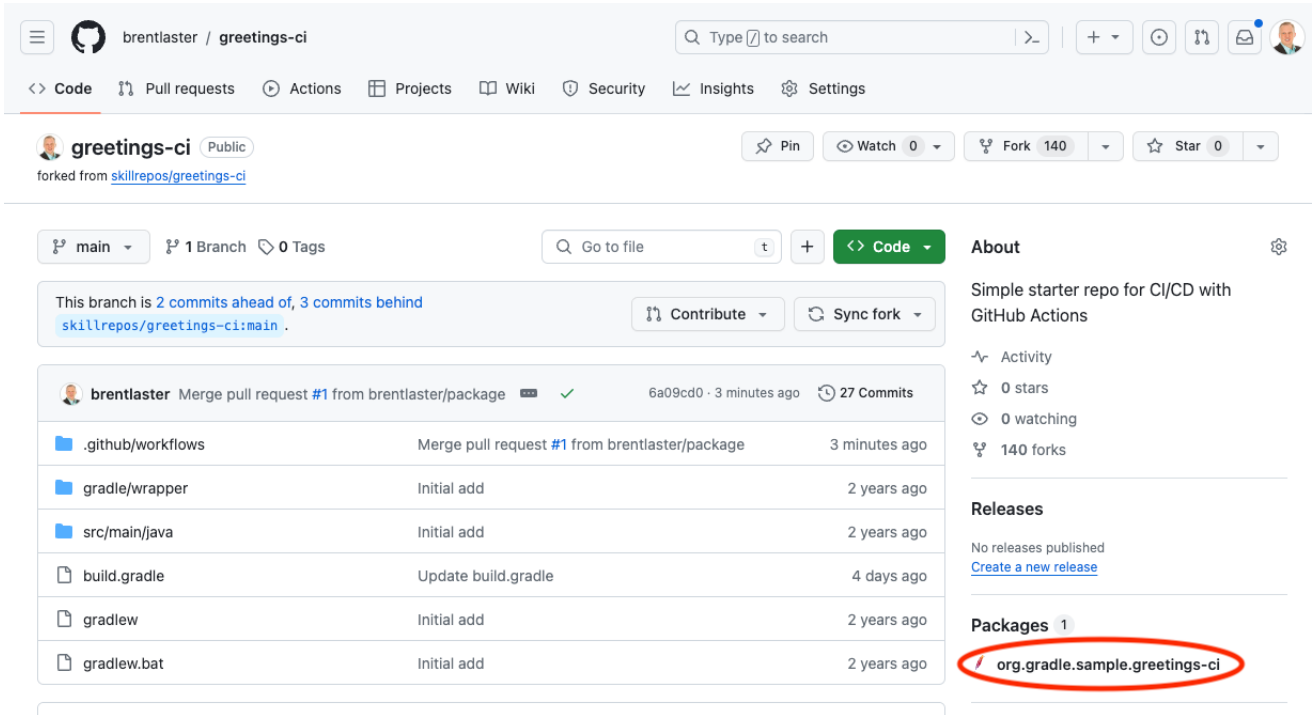
5. Open the new ***.github/workflows/publish-package.yml*** file on the *main* branch. Notice that it has a *workflow-dispatch* trigger. This allows the workflow to be invoked manually. (No changes need to be made.)



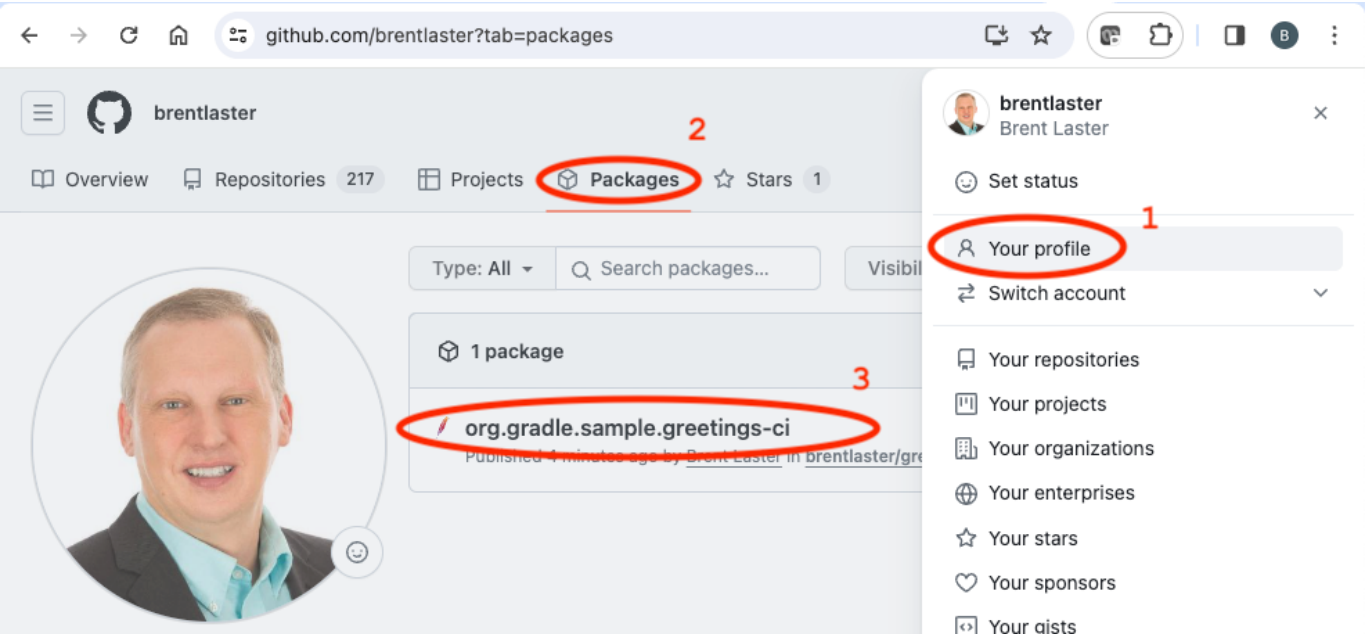
6. Switch to the ***Actions*** menu, then select the ***Publish package to GitHub Packages*** workflow on the left and select the ***Run workflow*** button that shows up in the blue bar.



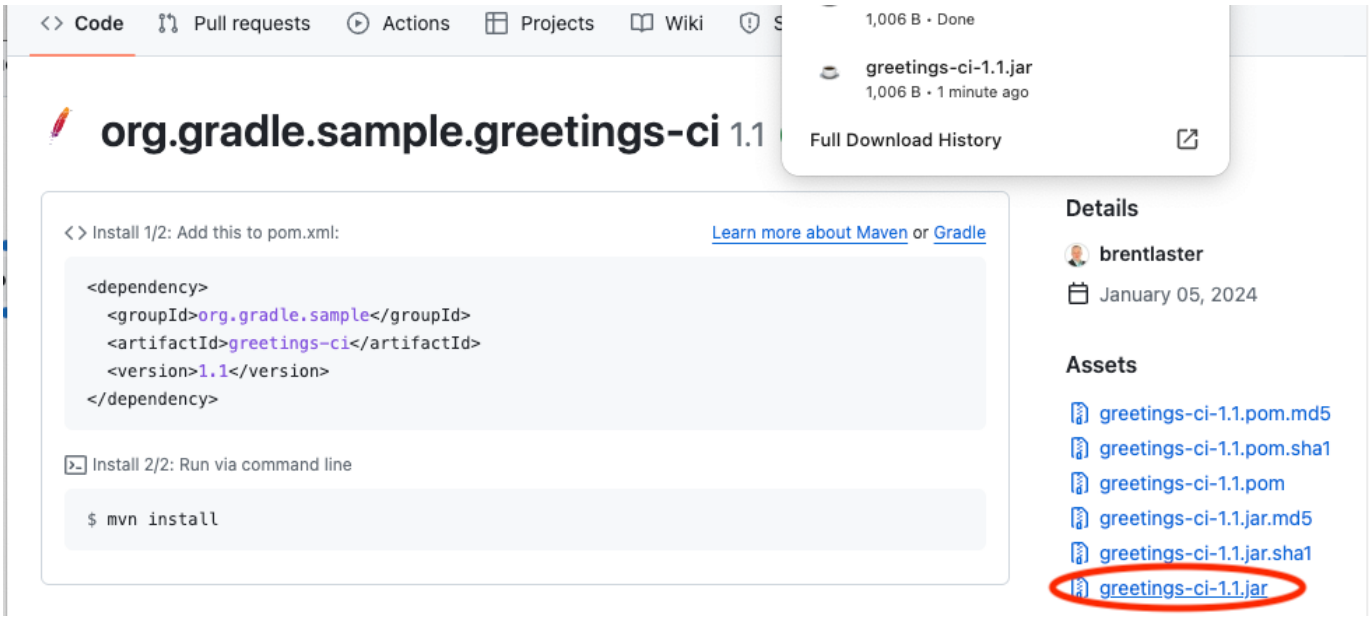
6. After this, you can switch to the ***Code*** tab and you should be able to see the new package listed in the ***Packages*** area in the lower right of the screen. Click on the link to find out more details about it.



7. You can also see the new package in your profile area. Click on your picture in the upper right, then select ***Your profile*** and then the ***Packages*** tab.



8. You can also download the individual artifacts by clicking on the link to the package and then in the list of assets, clicking on individual items. Do this for the ***greetings-ci-1.1 jar*** file.

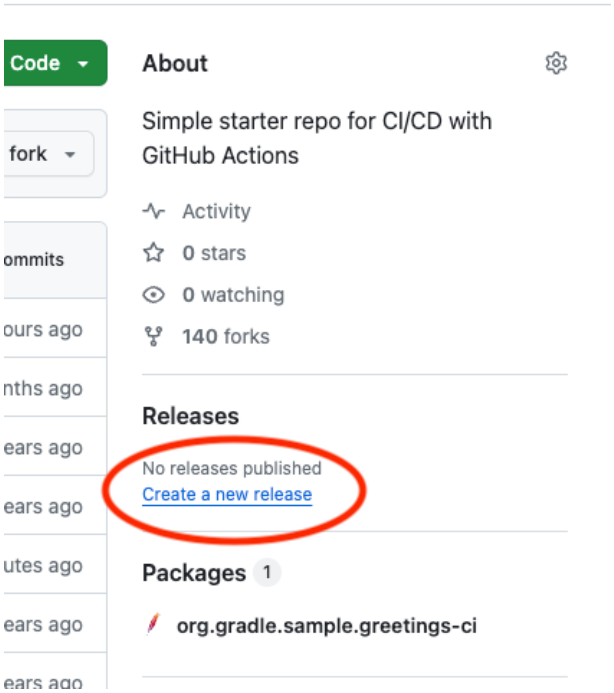


END OF LAB

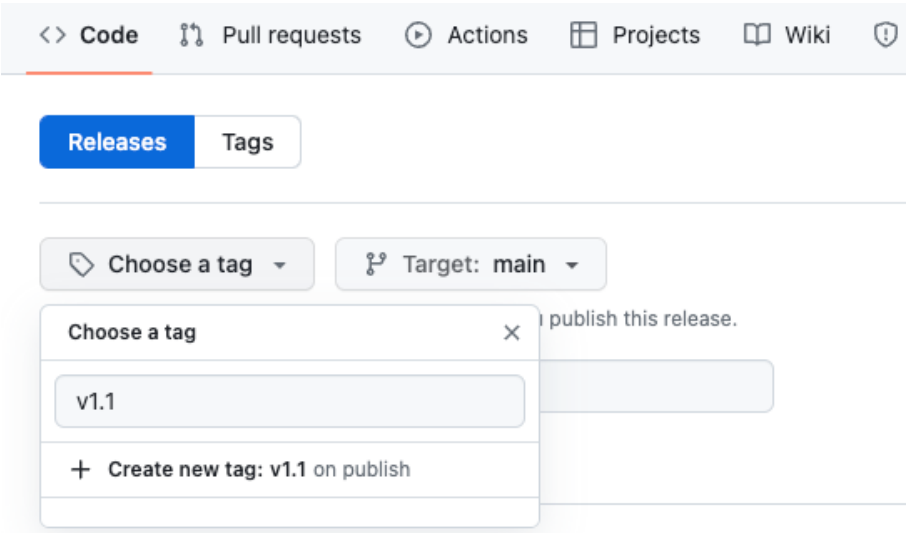
Lab 9: Creating a release

Purpose: *In this lab, we'll create a new release of our project's code.*

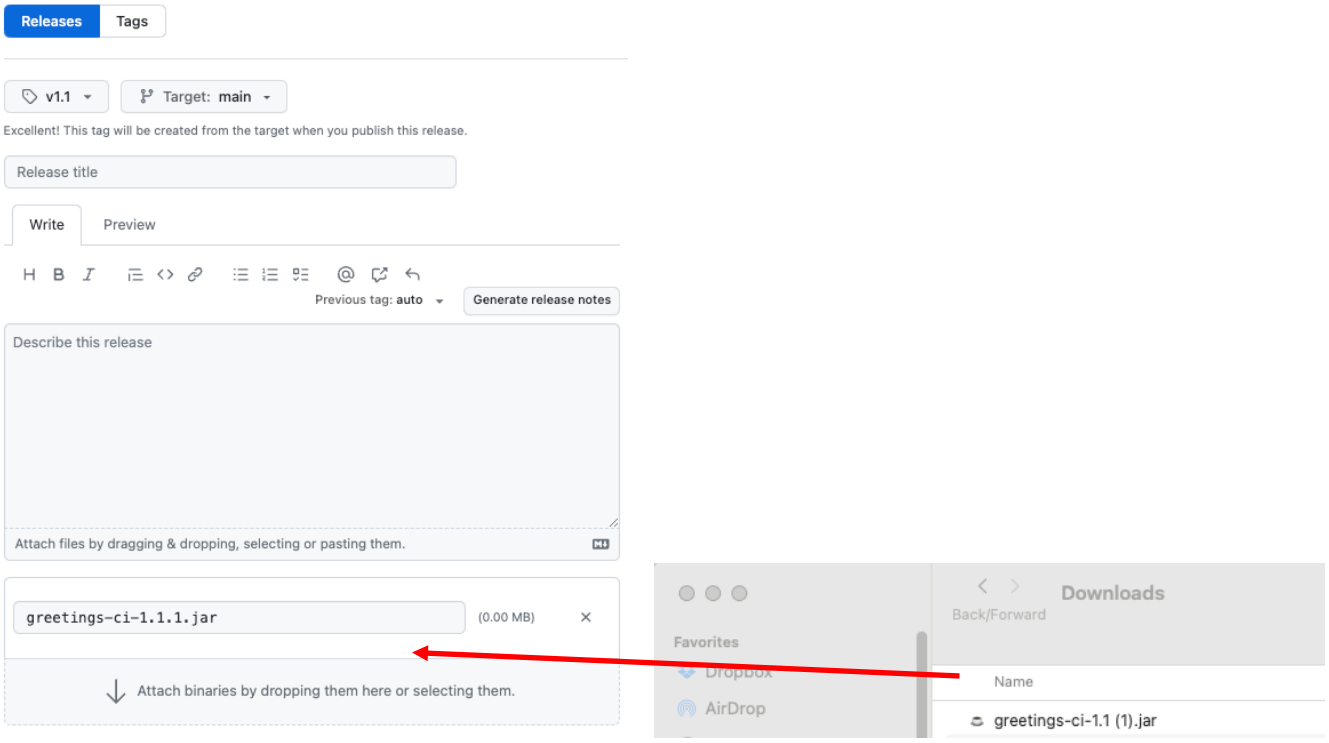
1. On the ***Code*** tab of the *greetings-ci* repo, on the right-hand side, find the ***Releases*** section and click on the ***Create a new release*** link. (You can also go directly to the page by going to https://github.com/<github-userid>/greetings-ci/releases/new .)



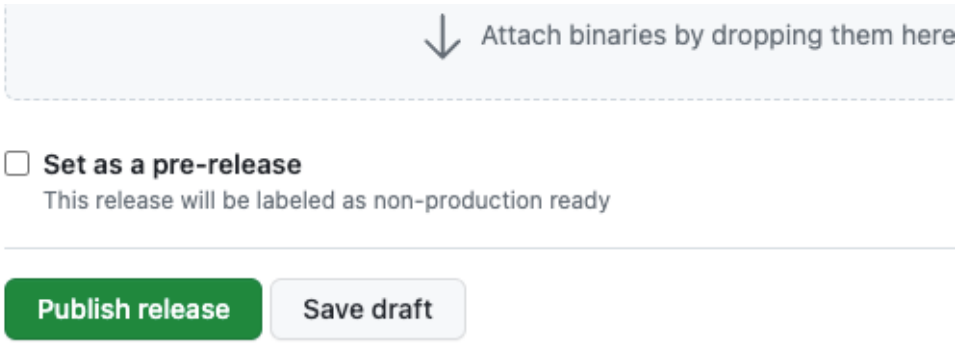
2. We need to create a tag on the repo before we create a release. Click on the ***Choose a tag*** dropdown and enter *v1.1* (or some other name if you prefer) for the tag name. Then click on the ***+ Create new tag: v1.1 on publish*** line.



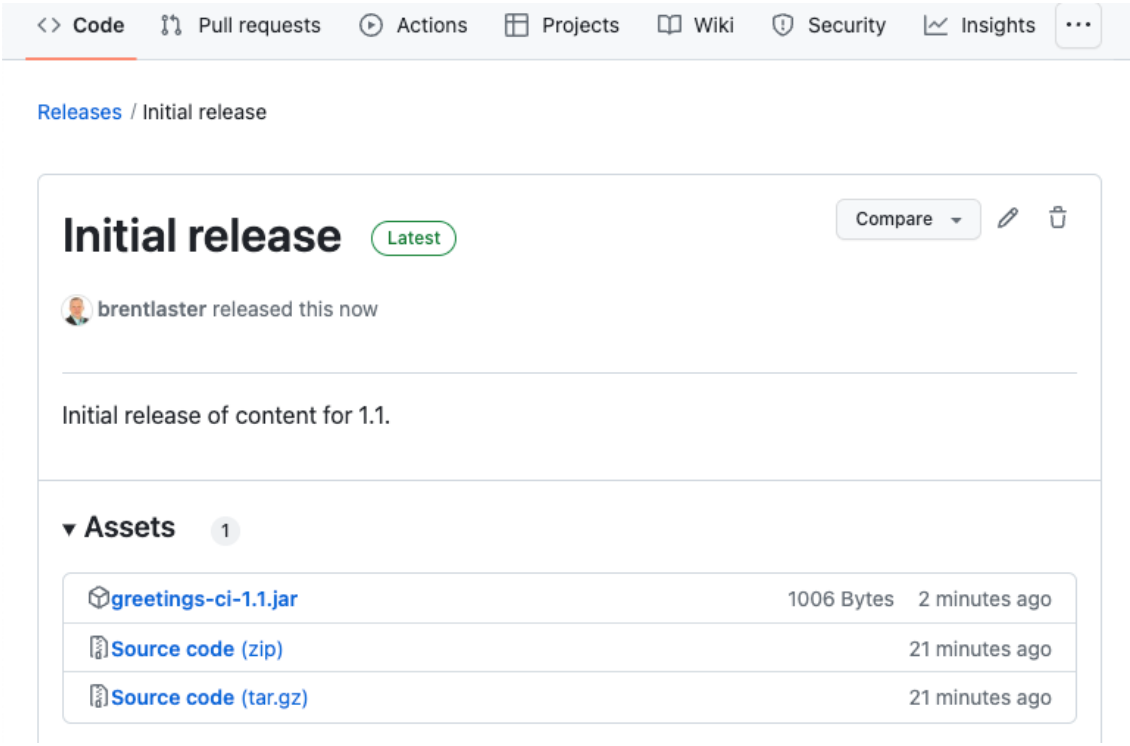
3. Now let's update a file for the release. Near the bottom of the screen, drag and drop or select a file. For simplicity, just drag and drop the file you downloaded locally at the end of the last lab. This will add the greetings-ci.jar file to the release.



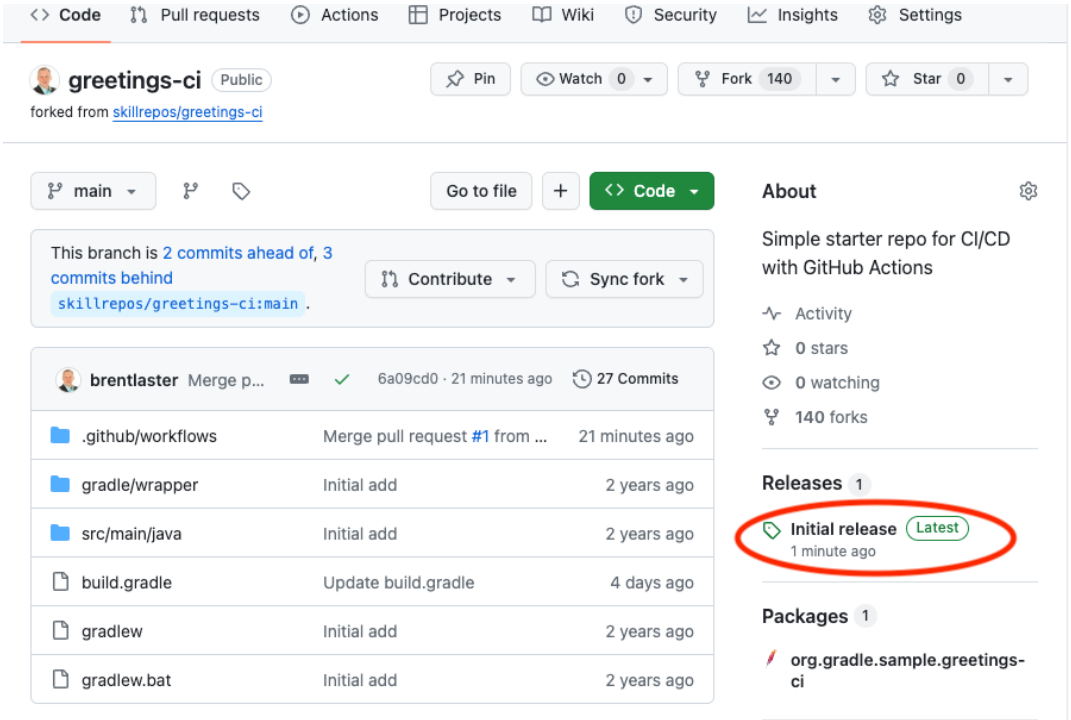
4. Click the button at the bottom of the page to publish the release.



5. After this, you'll see the published release page.



6. If you switch back to the main page of the repo, you'll see the new release under the *Releases* section on the right side of the page.



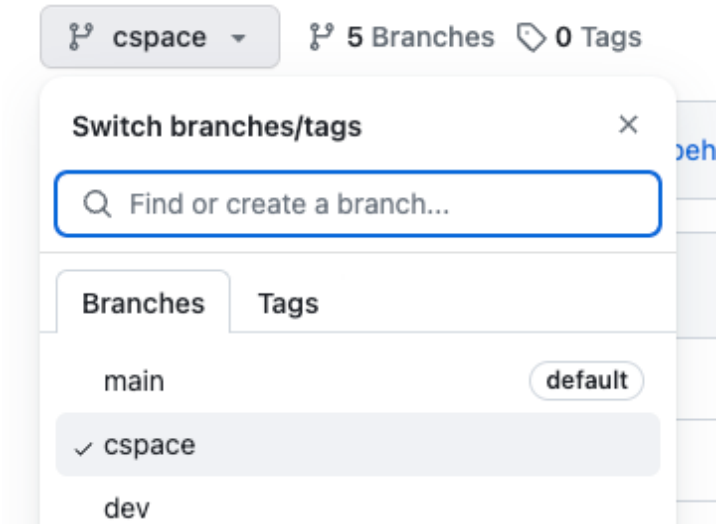
7. You can click on that if you want to see details about the release (same as output in step 5).

END OF LAB

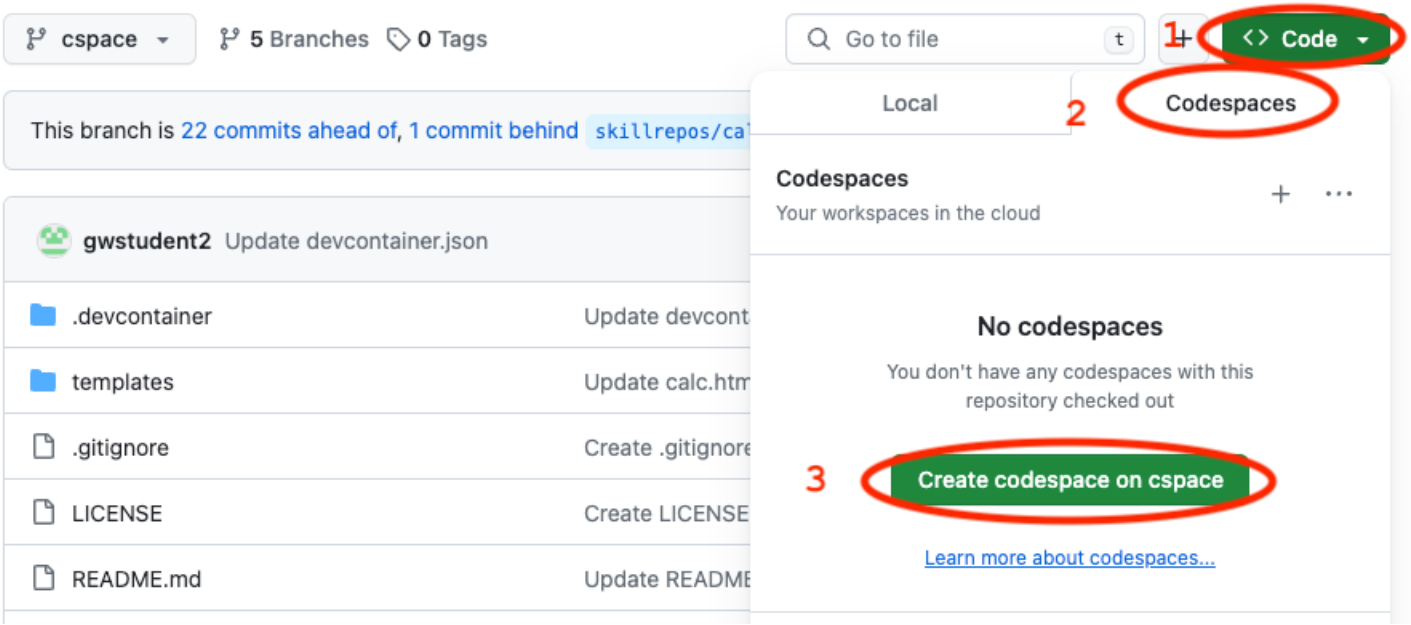
Lab 10: Working with Codespaces

Purpose: *In this lab, you'll see how to work with a GitHub Codespace*

1. Go back to the *github.com/<github-userid>/calc* project. In that project, select the ***cspace*** branch.



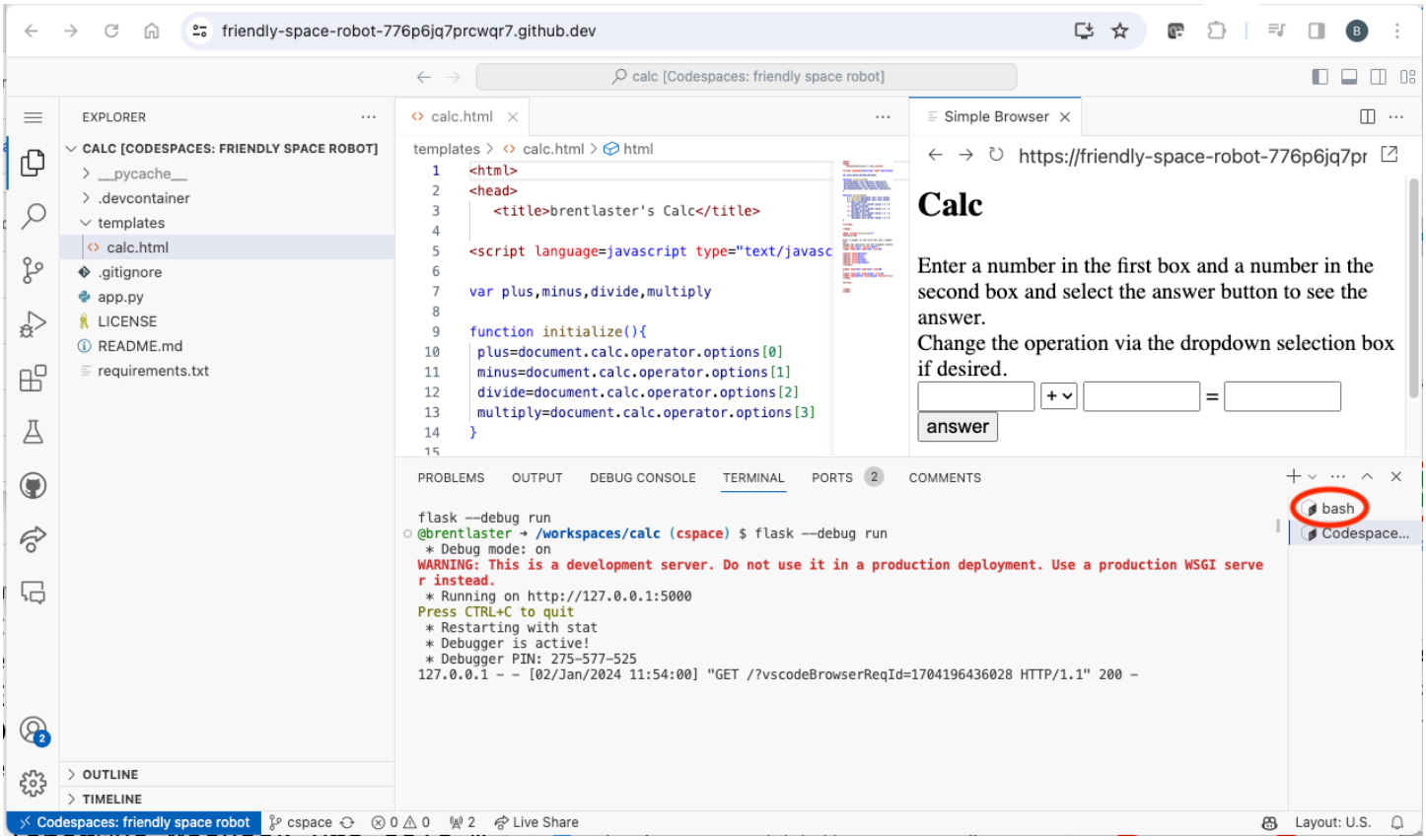
2. Click on the **<> *Code*** button, then select the ***Codespaces*** tab, and then select ***Create codespace on cspace***.



3. Creating the codespace will take a few minutes to complete.

When it's done, you'll now have a new codespace with this repo checked out and the calculator webpage open and running.

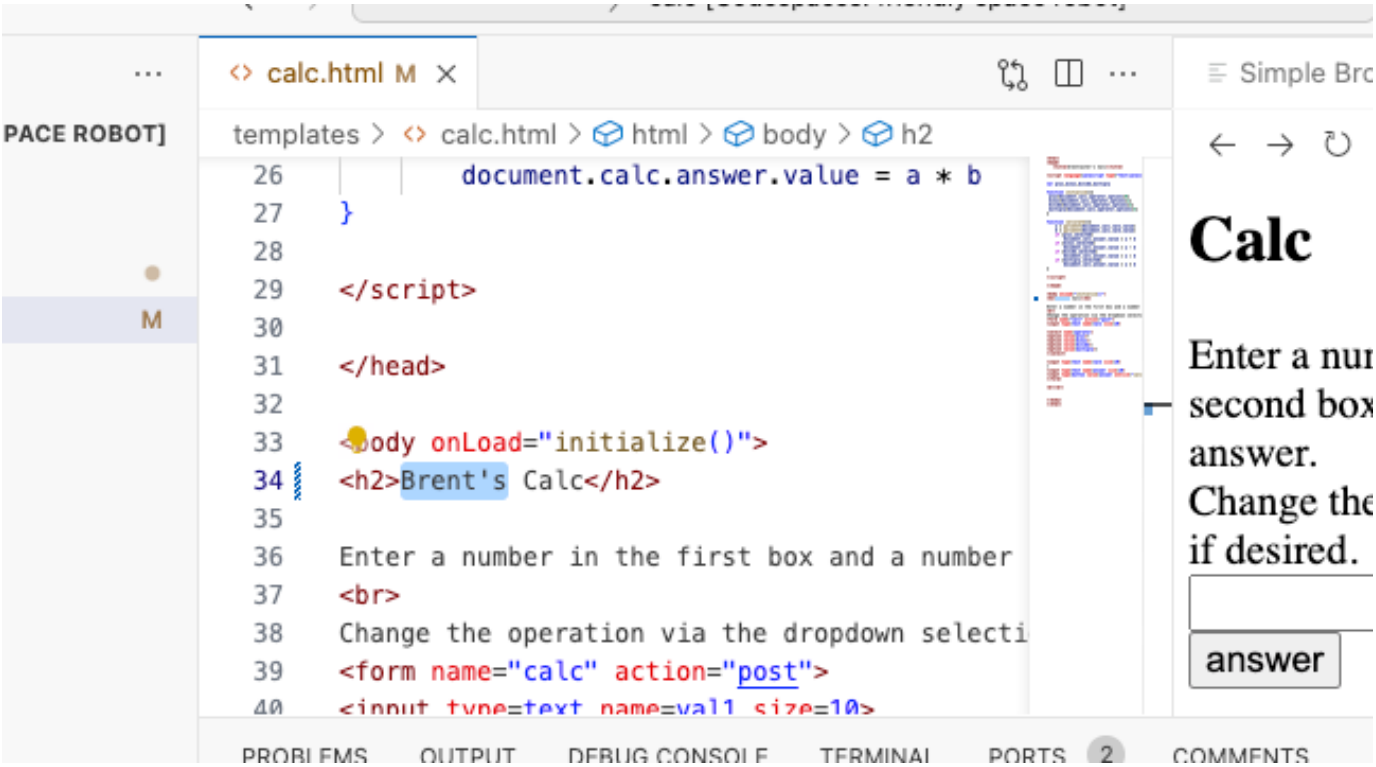
There is also a terminal at the bottom. This is a secondary terminal. To get back to the main terminal, click on the *bash* selection at the far right side. (You can also dismiss any dialogs about linting.)



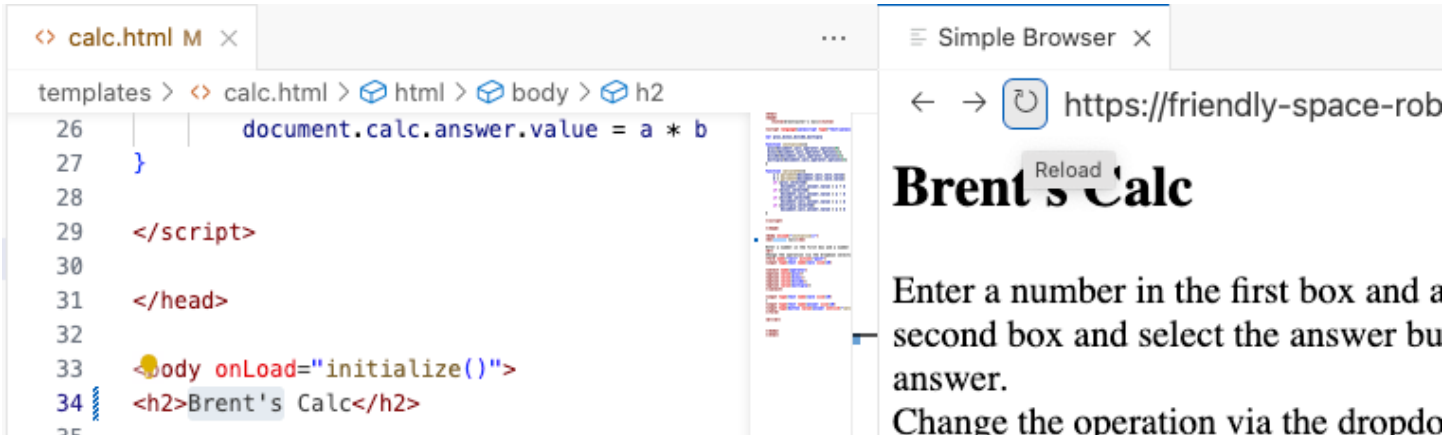
4. To see how to edit in a codespace, let's change the title displayed in the webapp.

The file *calc.html* was already opened automatically for you.

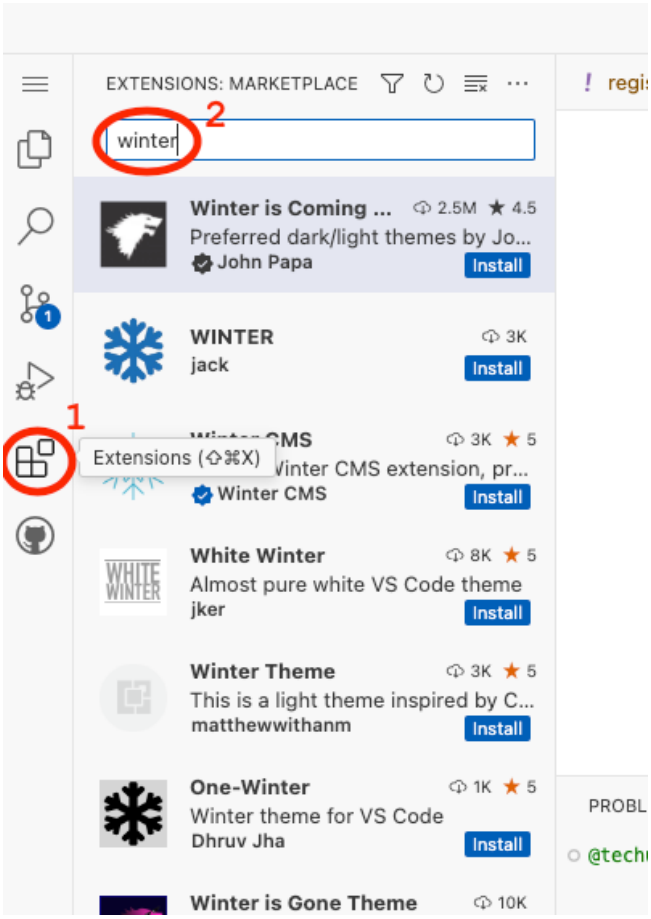
Click in the *calc.html* pane and scroll down to line 34 where the title is. Just type into that line and add your name in front of "Calc". The change is automatically saved.



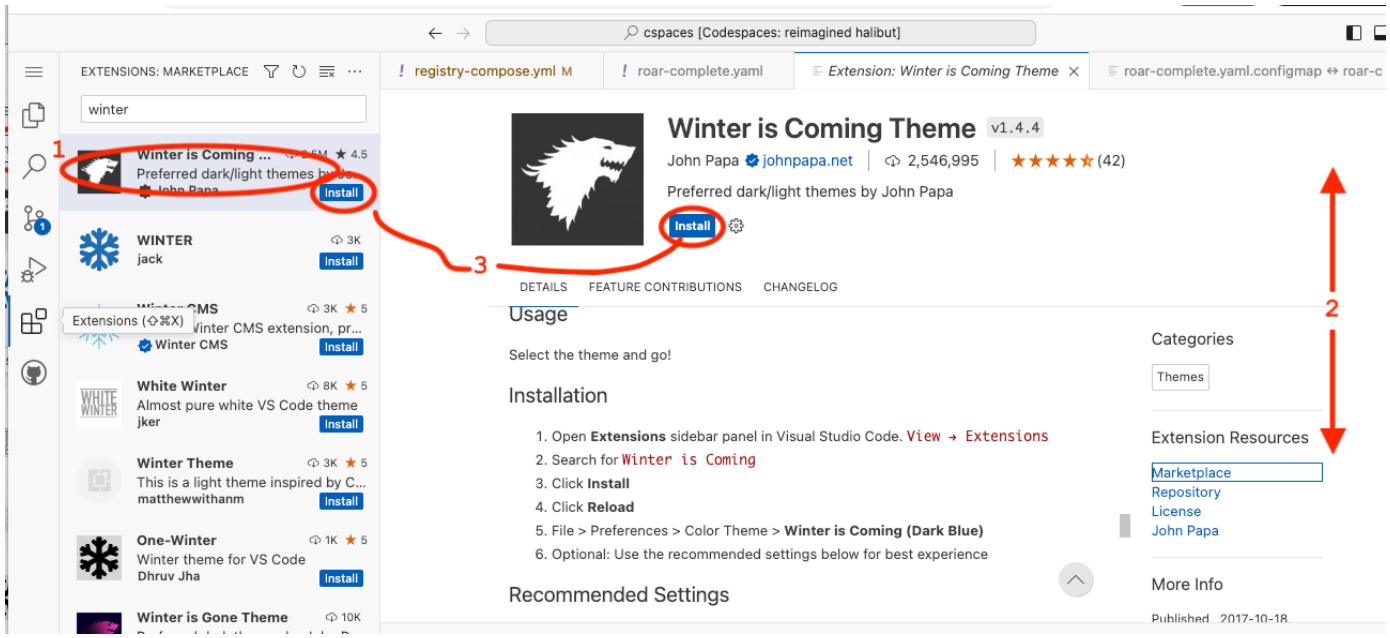
5. Now, in the Simple Browser pane, click the circular arrow icon to reload the webapp. You should see your change being displayed.



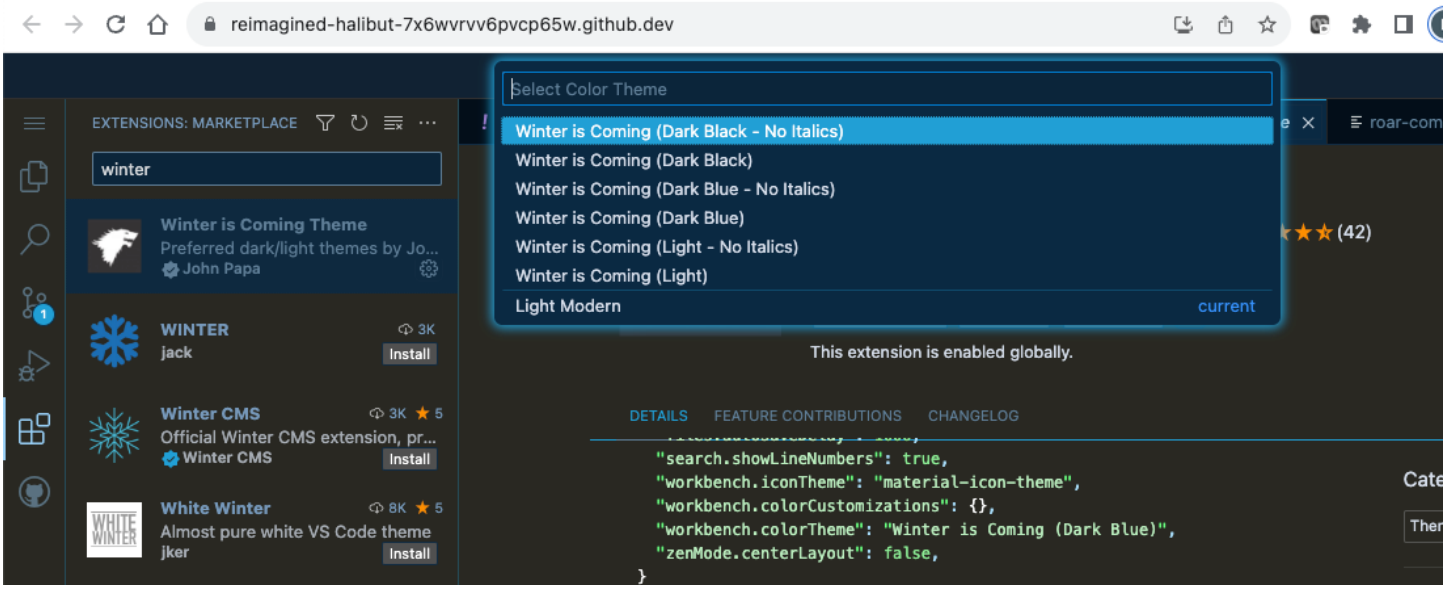
6. Next, let's see how to modify our codespace environment. We'll install an extension for the color theme. For practice, we'll use the "Winter is Coming" theme." Click on the *Extensions icon* (#1 in figure below), then in the *search bar* type in "winter" to quickly find the extension (#2).



7. Once found, you can directly install the extension (#3 in figure below) or click on it (#1) and bring up the info in an editor page and scroll around it (#2) to get more details. Go ahead and install it (via the *Install* button) when ready.



8. After installing, you'll see a list where you can select one of the new color themes. You can choose another one from the list if desired. (If you happen to get an error, try clicking on *Set Color Theme* and pick again.)



END OF LAB

Lab 11: GitHub Command Line

Purpose: In this lab, you'll get to work with the GitHub CLI.

1. In your codespace, the GitHub command line interface (CLI) is already installed. You can try it out from the terminal.

Click in the terminal and run the command ***gh*** by itself to see available options. You can page through the output.

```
$ gh | more
```

2. Take a look at the codespaces you have with the following command:

```
$ gh codespace list
```

3. For some commands, you need to set the default remote. Set it now to your current repo.

```
$ gh repo set-default <github-userid>/calc
```

4. Let's look at the issue you created in this repo for the earlier lab. (If your issue number was not 1, then use the appropriate issue number.) First, we'll look at it in the terminal and then in the browser.

```
$ gh issue view 1
```

```
$ gh issue view 1 --web
```

5. You can also do the same for one of your pull requests – just pick a number of one of them.

```
$ gh pr view 1

$ gh pr view 1 --web
```

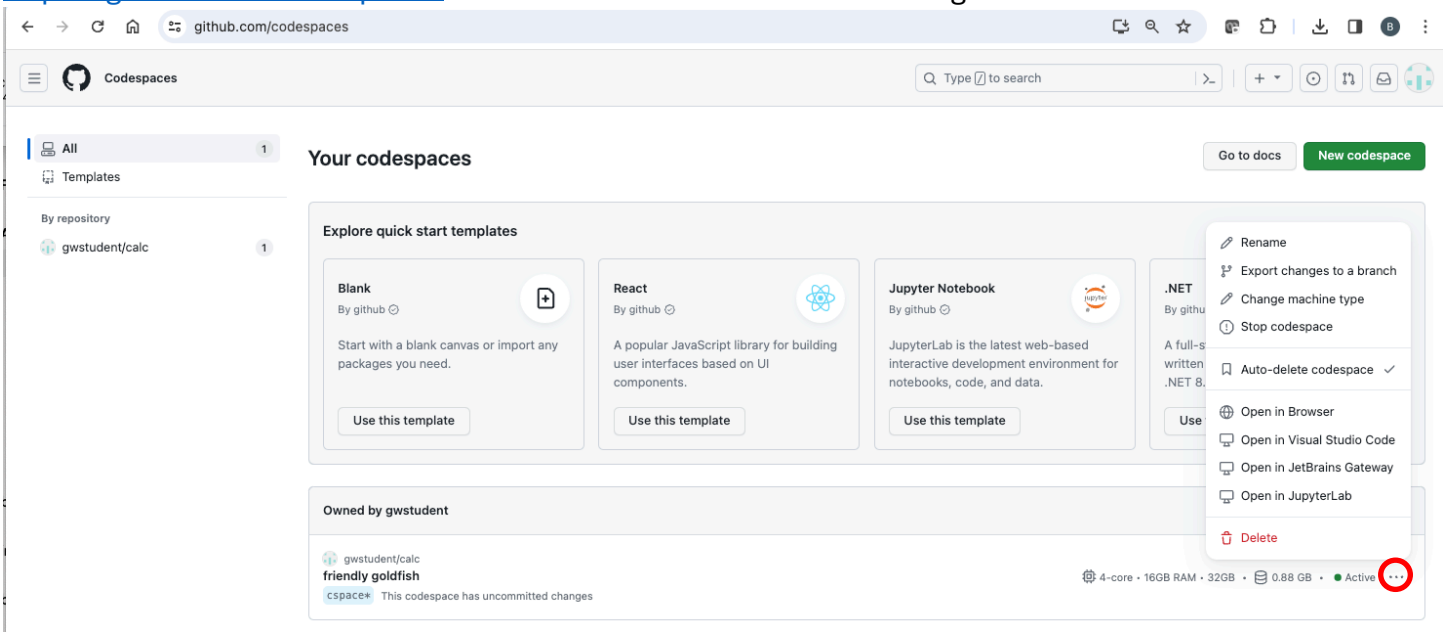
6. You can also clone repos easily with the command line. Change up one directory to not clone within the current directory. Then run the command to clone down your other repo.

```
$ cd ..

$ gh repo clone <github-userid>/greetings-ci
```

OPTIONAL:

7. Your codespace will time out eventually. But if you want stop it now or delete it, etc., you can go to https://github.com/codespaces and click on the **...** in its row to manage it.



END OF LAB

Demo: Copilot

That's all - THANKS!

APPENDIX 1

Other options for making changes in repo vs https (if the https approach doesn't work for you) – choose one of A,B, or C if and only if the https push did not seem to work…

A. Reseting credential helpers: Especially on Windows, if you are pasting in your token for the password, but still getting an error message referencing password authentication, you may be running into issues because you have previous credentials stored in the *credential helper*. One of the things you can try in this case is resetting the stored credentials via:

```
$ git config --global credential.helper store
```

Then you do your push as per the lab. It will probably pop up a text entry box for you to add your username in and another to paste in your password (PAT) and then will replace your credentials with those and complete the push.

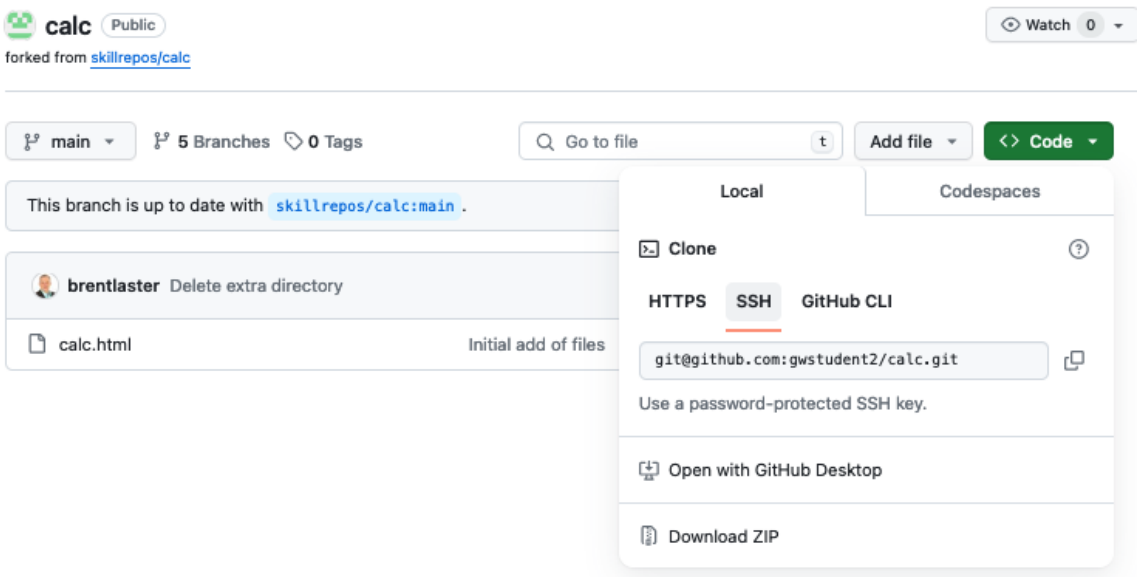
(Note: If you prefer to disable the global credentials helper entirely, you can try

```
$ git config --unset --system credentials.helper
```

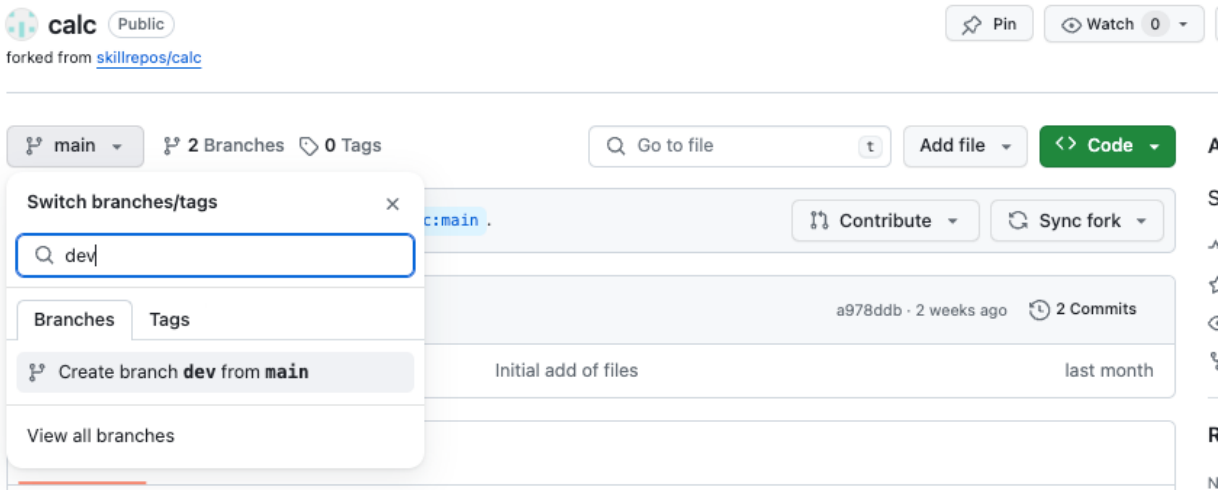
This may or may not work depending on if you have access to do this.)

B. SSH keys: If you are familiar with using ssh and have keys, you can add them into GitHub and use those. Ref https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account for more details.

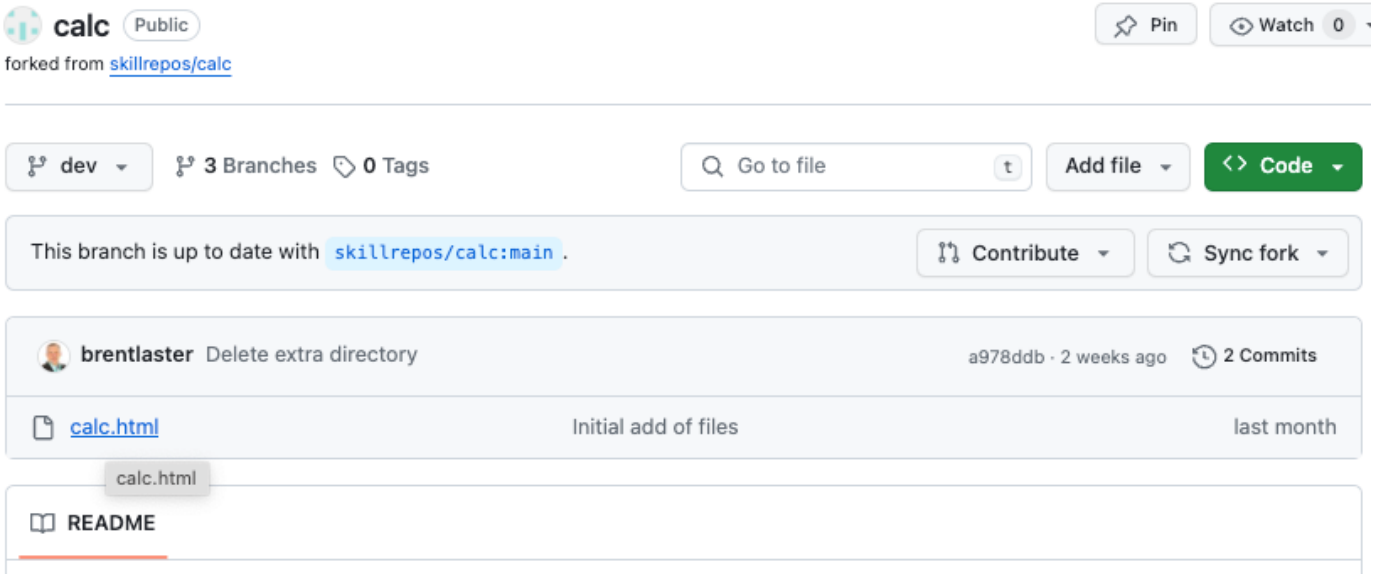
If you go this route, when you get the remote URL from the browser, select the SSH tab.



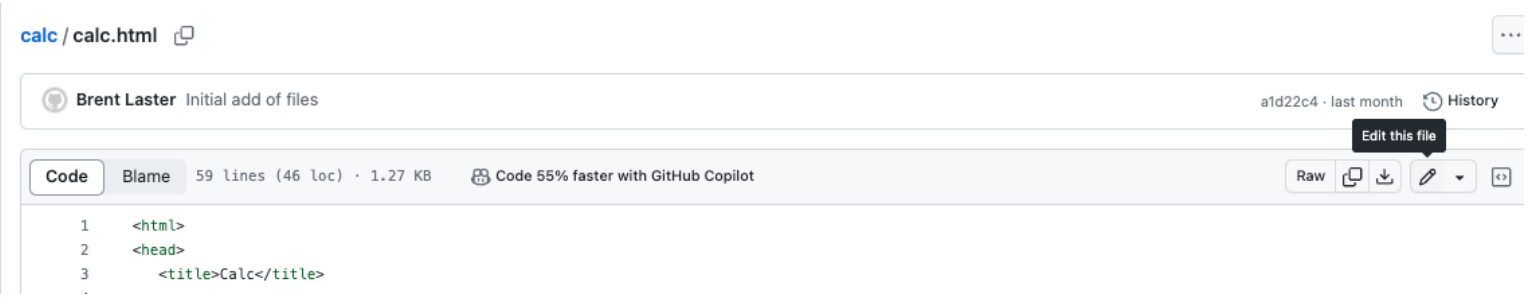
C. Commit directly in GitHub: Another option is to commit directly to GitHub in the browser. To do this, first create a *dev* branch in the repo. Clic on the branch dropdown under the title of the repo. In the *Find or create a branch* field, type **dev**. Then click on ***Create branch dev from main***.



In the *dev* branch, click on the *calc.html* file and open it up.

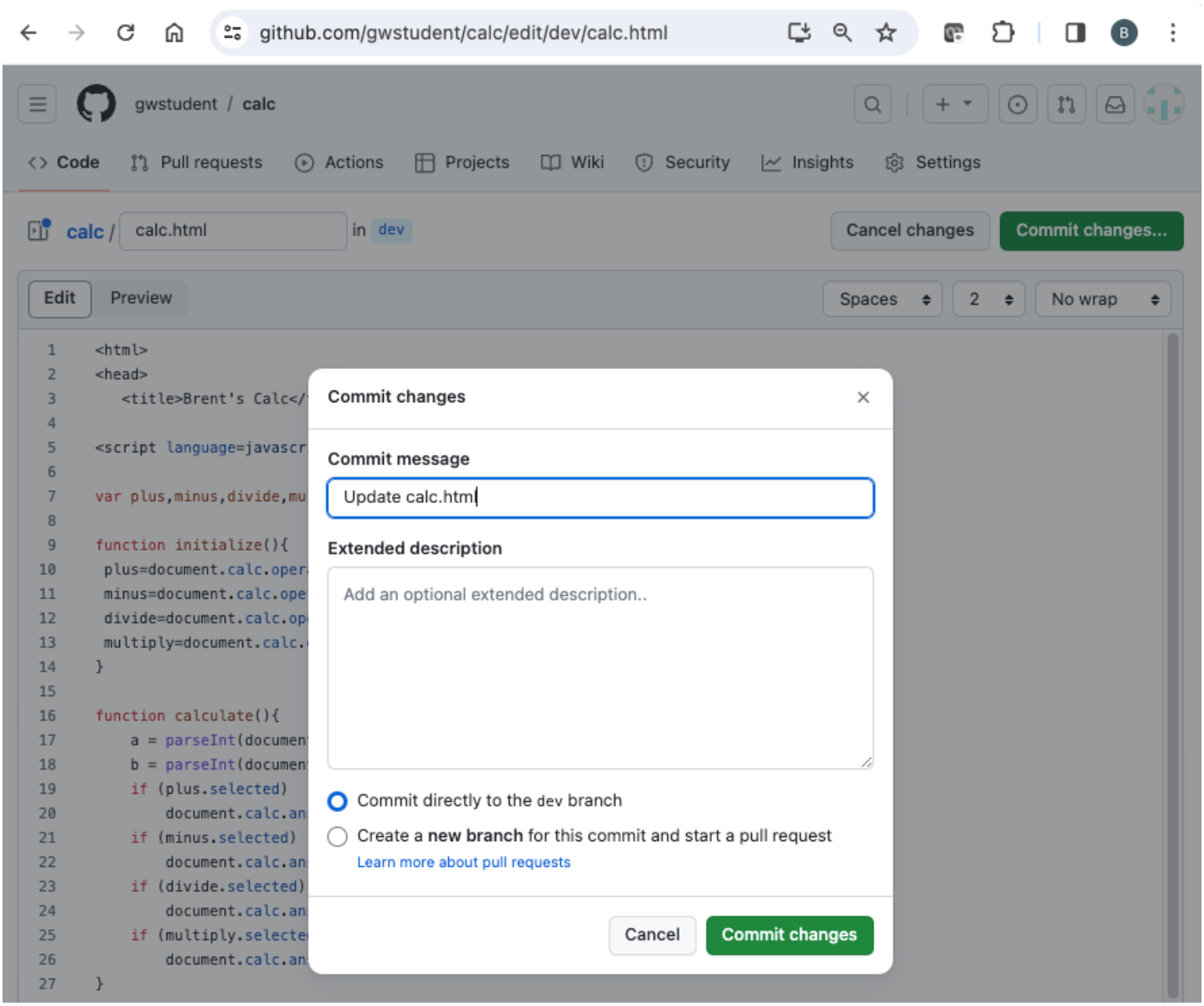


Click on the pencil icon to edit the file.



Make the changes noted in Lab 1 in the file.

When done editing, click on the ***Commit changes...*** button in the upper left, then in the dialog that comes up, you can leave all the options as they are, and then click on the ***Commit changes*** button to commit/push the file.



APPENDIX 2

Alternative approaches to forking a repo if you can't use the actual Fork button.

IMPORTANT NOTE: In these examples, the repository "sec-demo" is used. Change the name to whatever repository you are trying to fork.

Option 1 – Using import

1. Sign into GitHub if not already signed in.

2. Go to https://github.com/new/import

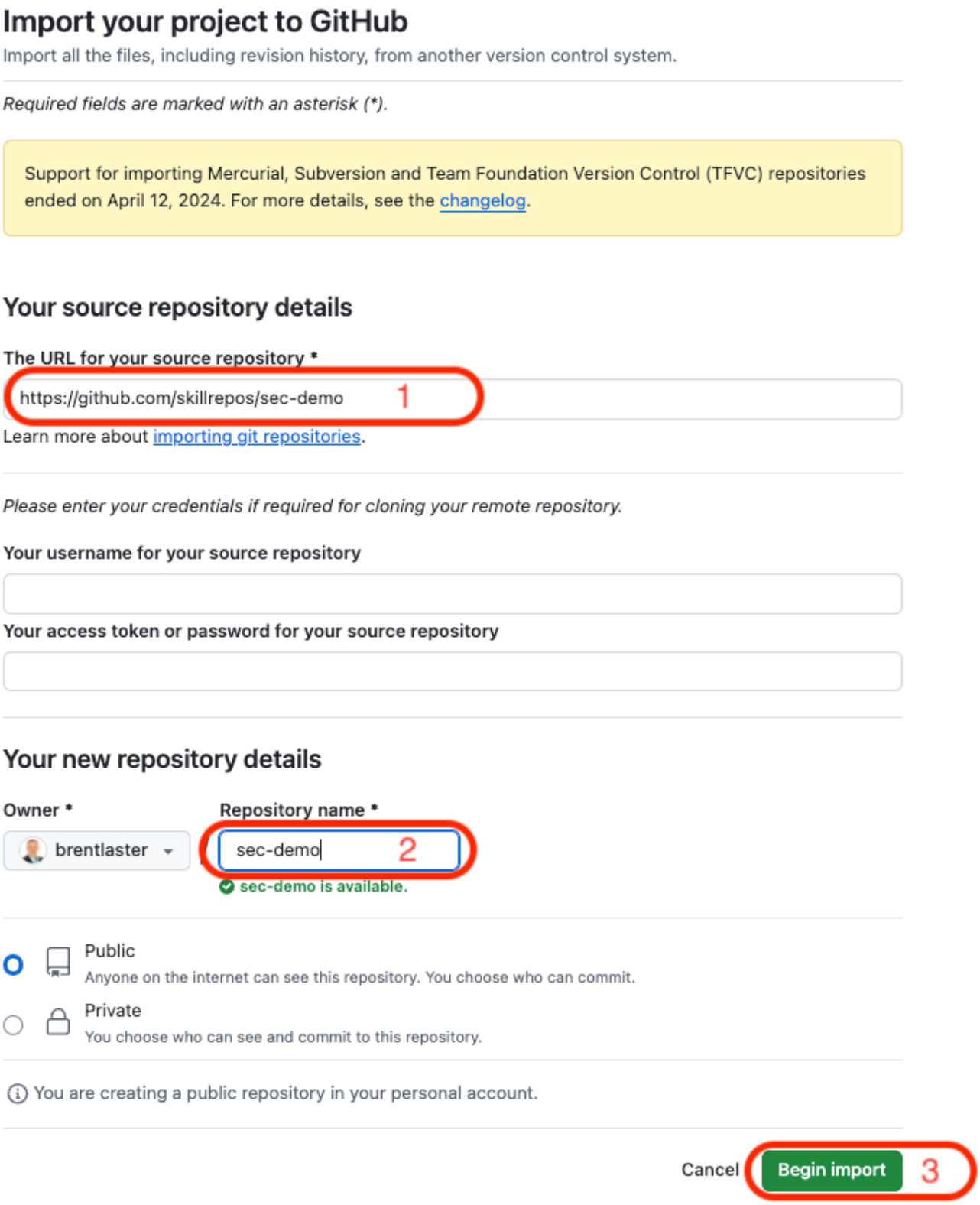
3. On that page, fill out the form as follows:

In "Your source repository details", in the "The URL for your source repository *" field, enter

https://github.com/skillrepos/<repo-name>

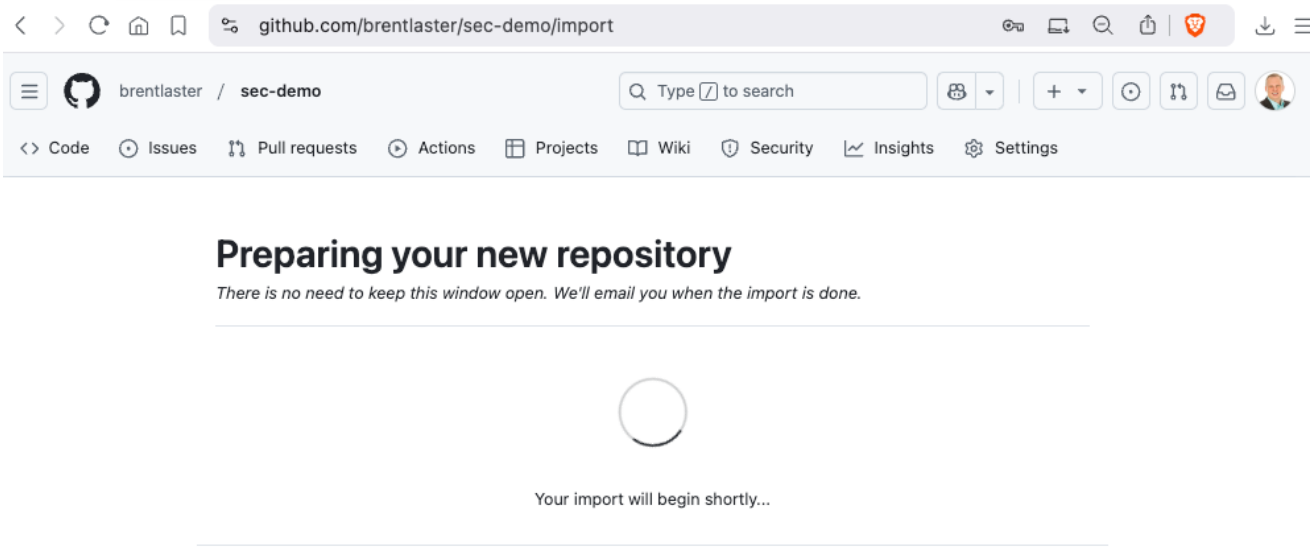
Under "Your new repository details", make sure your userid shows up in the "Owner *" field and enter
the desired <repo-name> in the "Repository name *" field.

The visibility field should be set to "Public".



4. If you haven't already, click on the green "Begin import" button.

5. After this, you should see the import processing...



Option 2 – Using clone and push

1. Sign into GitHub if not already signed in.

2. If you don't already have one, create a GitHub token or SSH key. If you are familiar with SSH keys, you can add your public key at **https://github.com/settings/keys**. Otherwise, you can just create a "classic" token by following the instructions at **https://docs.github.com/en/authentication/keeping-your-account-and-datasecure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic**. If you use a GitHub token, make sure to save a copy of it to use in the push step.

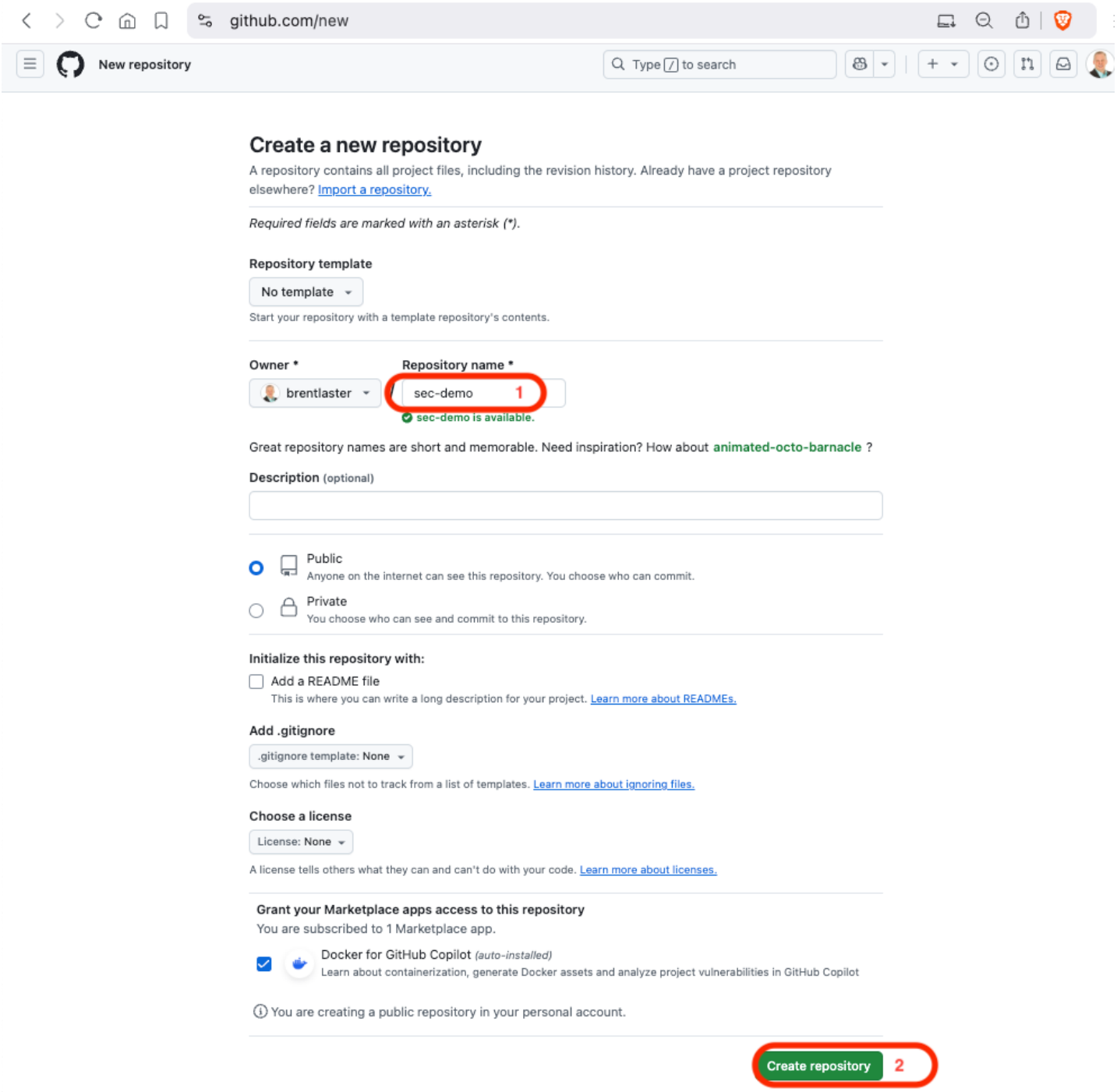
3. Clone down the **skillrepos/sec-demo** repository.

   git clone https://github.com/skillrepos/sec-demo (if using token)

or

   git clone git@github.com:skillrepos/sec-demo (if using ssh)

4. Create a new repository in your GitHub space named *sec-demo*. Go to https://github.com/new. Fill in the "repo name" field with "sec-demo" and then click on the "Create repository" button.



5. On the page that comes up after that, select the appropriate protocol (https or ssh) and then follow the instructions for "...or push an existing repository from the command line" to push your content back to the GitHub repository. If you're using https you will be prompted for a password at push time. Just paste in the classic token. (Note that for security reasons, you will not see the token displayed.)