Git in 4 Weeks - Part 4 Labs

Revision 2.5 - 12/12/23

Brent Laster for Tech Skill Transformations LLC

Important: Prior to lab 12, you will need to have your GitHub Personal Access Token (PAT) available. If you need to generate one again, follow the instructions at the link below to generate the token. Keep a copy of it somewhere so you can copy and paste it to use in the labs.

Instructions

Direct link is below if you need it (same as link for "Instructions" above)

https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic

Lab 11 - Working with Worktrees

Purpose: In this lab, we'll get some experience with how to work on multiple branches at the same time.

1. In my GitHub space, I have a project called *calc2* which is a simple javascript calculator program. It has multiple branches for *features*, *documentation*, etc. For this lab, I have split it up into three separate projects:

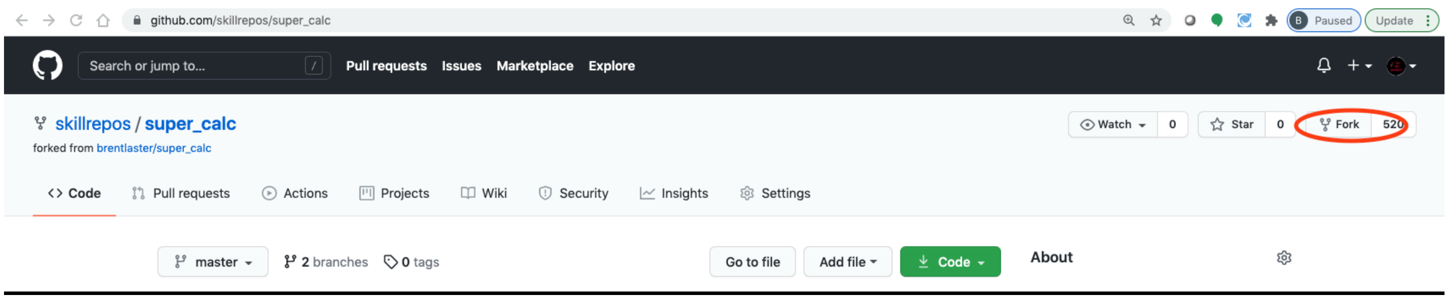
    - *super_calc*, a version of the calc2 project with only the main and feature branches
    - *sub_ui*, a separate repository consisting of only the content of the ui branch split out from the calc2 project
    - *sub_docs*, a separate repository consisting of only the content of the docs branch split out from the calc2 project

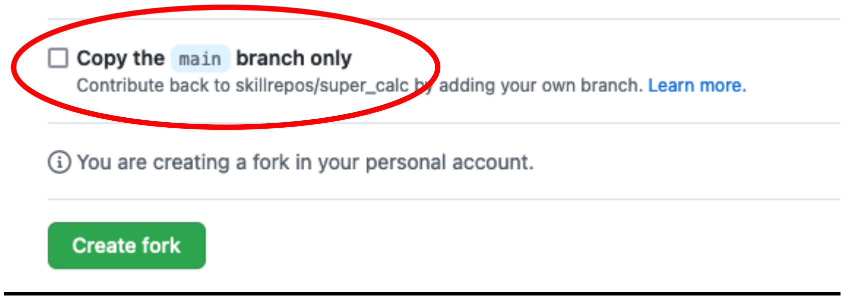
   Log in to your GitHub account and fork the three projects from the following listed locations. (As a reminder, the **Fork** button is in the upper-right corner of the pages.) This will prepare your area on GitHub for doing this lab, as well as the labs on subtrees. Also make sure to **uncheck** the "Copy the main branch only" option as shown below.

    https://github.com/skillrepos/super_calc.git

    https://github.com/skillrepos/sub_ui.git

    https://github.com/skillrepos/sub_docs.git




Copy the `main` branch only

Contribute back to skillrepos/super_calc by adding your own branch. Learn more.

You are creating a fork in your personal account.

2. In a new directory, clone down the super_calc project that you forked in step 1, using the following command:

```
$ git clone https://github.com/<your github userid>/super_calc.git
```

3. Now, change into the cloned directory - super_calc.

```
$ cd super_calc
```

4. In this case, you want to work on both the main branch and the features branch at the same time. You can work on the main branch in this directory, but you need to create a separate working tree (worktree) for working on the features branch. You can do that with the ***worktree add*** command, passing the ***-b*** to create a new local branch from the remote tracking branch.

```
$ git worktree add -b features ../super_calc_features origin/features
```

5. Change into the new subdirectory with the new worktree. Note that you are on the features branch. Edit the calc.html file and update the line in the file surrounded by <title> and </title>. The process is described below.

```
$ cd ../super_calc_features

$ git branch (note which one is selected automatically)
```

```
Edit calc.html and change

     <title>Calc</title>i
```

```
                    to

        <title> github_user_id's Calc</title>

        substituting in your GitHub user ID for "github_user_id".
```

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

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

7. Switch over to your original worktree.

```
        $ cd ../super_calc
```

8. Look at what branches you have there.

```
        $ git branch
```

9. Note that you have the features branch you created for the other worktree. Do a log on that branch; you can see your new commit just as if you had done it in this worktree.

```
        $ git log --oneline features
```

10. You no longer need your separate worktree. However, before you remove it, take a look at what worktrees are currently there.

```
        $ git worktree list
```

11. You can now remove the worktree. First, remove the actual directory; then use the prune option to get rid of the worktree reference.

```
        $ rm -rf ../super_calc_features
        $ git worktree prune
```

==

END OF LAB

==

Lab 12 - Working with Pull Requests

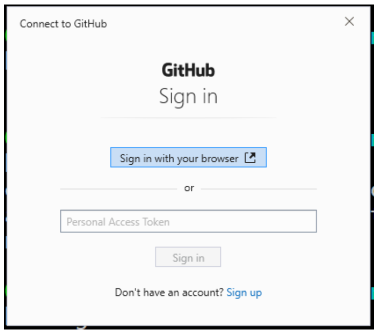
Purpose: In this lab, we'll see how to create and merge Pull Requests within a GitHub repository

In the last lab, you created a new branch named "features" and committed a change into it. We can leverage this commit to create a Pull Request to work with in GitHub.

1. Let's go ahead and push the change over to our remote on the GitHub side to a new branch. (You can use a different name in place of "dev" if you want. Just use that name consistently wherever "dev" is used here.)

```
$ git push origin features:dev
```

2. At this point, you'll be prompted for a Private Access Token or password (unless you've setup ssh access). 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.



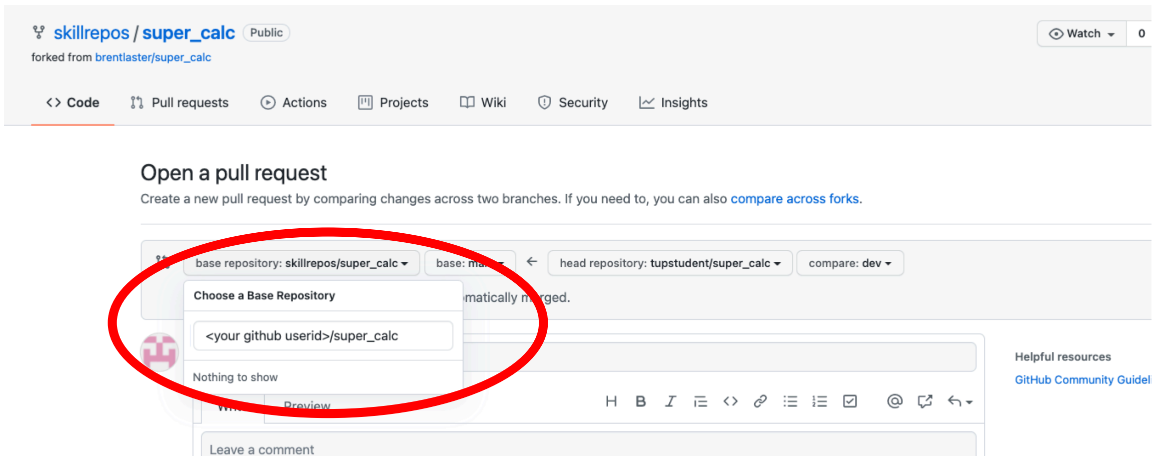
3. In addition to pushing your changes to the new branch, GitHub will also provide a link in the output that you can go to to create a pull request for this change.

```
:super_calc developer$ git push origin features:dev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/<github-userid>/super_calc/pull/new/dev
remote:
To https://github.com/tupstudent/super_calc
 * [new branch]      features -> dev
```

4. Highlight the link in the output (the one that ends with …/super_calc/pull/new/dev) and then open that link. (You may be able to right-click and have an option to open it, or you can simply copy and paste it into the browser.

5. You'll now be on the screen to open a pull request. Notice that it defaulted to the original project. But we want to do a pull request between two branches in the same project NOT from your project to the parent one. So first, make sure to change the "base repository: skillrepos/super_calc" to "<your github userid>/super_calc".

   Change the base repository now. Click on the "base repository" box and then type in the dialog.



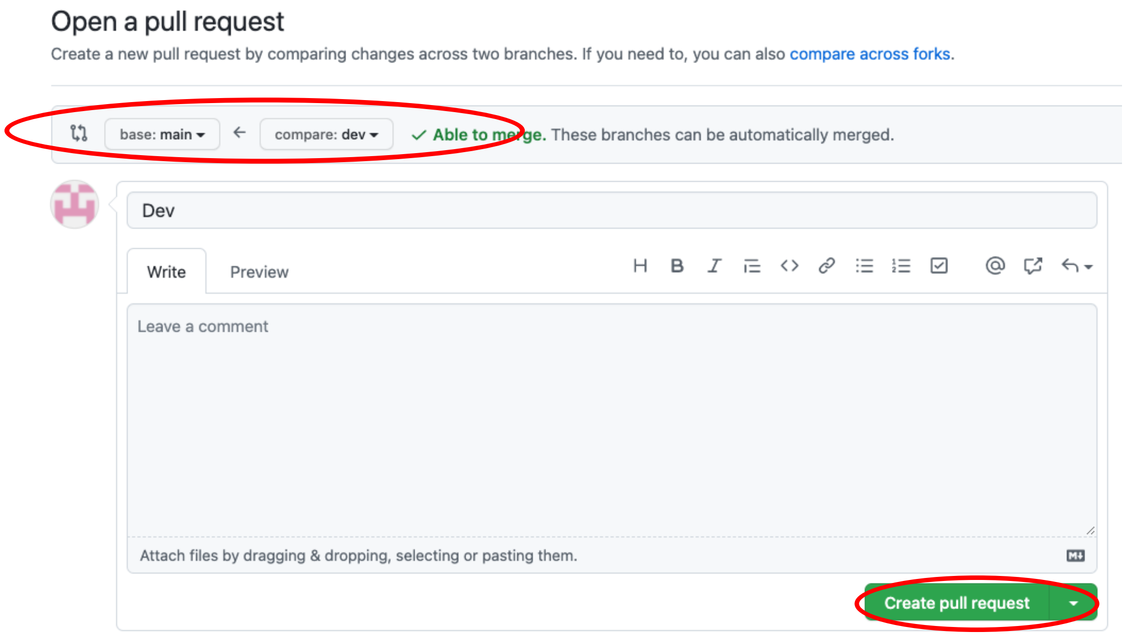
skillrepos / super_calc Public
forked from brentlaster/super_calc

Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base repository: skillrepos/super_calc ▾ | base: main ▾ | ← | head repository: tupstudent/super_calc ▾ | compare: dev ▾

Choose a Base Repository
<your github userid>/super_calc
Nothing to show

6. At this point, your current project should be back to <your-github-userid>/super_calc at the top. Next, ensure that the "main" branch is selected in the left dropdown and the "dev" branch in the right dropdown - as shown in the figure. You can also change the title or comment if you want. Afterwards, you can click on the "Create pull request" button.



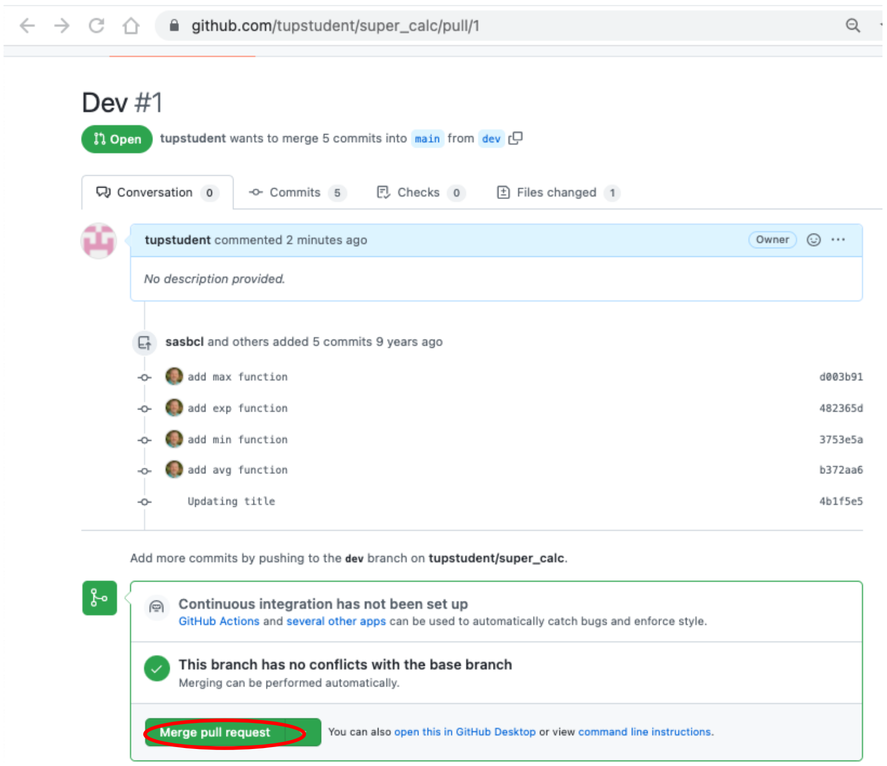
Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: main ▾ ← compare: dev ▾ ✓ Able to merge. These branches can be automatically merged.

Dev

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Create pull request

7. The pull request will open at "github.com/<your github userid>/super_calc/pull/1". You will now be able to see the changes that will be merged in if we proceed. When ready, go ahead and select the "Merge pull request" button.



tupstudent wants to merge 5 commits into main from dev

tupstudent commented 2 minutes ago

No description provided.

sasbcl and others added 5 commits 9 years ago

```
add max function      d003b91
add exp function      482365d
add min function      3753e5a
add avg function      b372aa6
Updating title        4b1f5e5
```

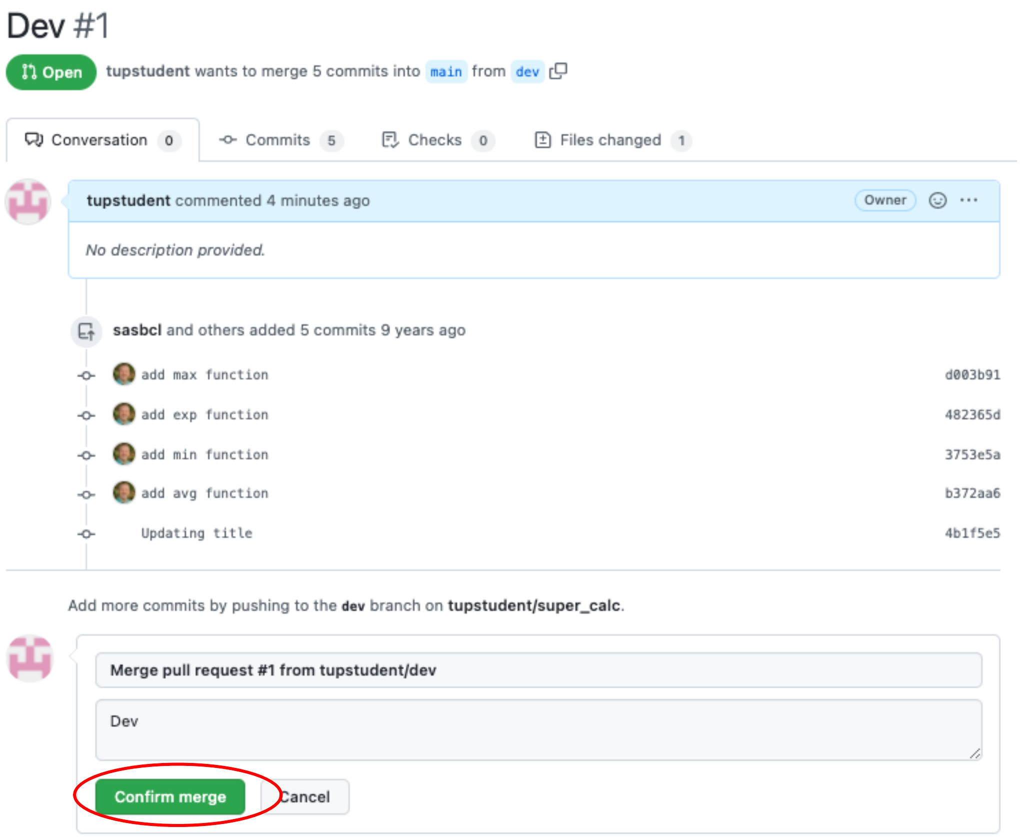
Add more commits by pushing to the dev branch on tupstudent/super_calc.

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.

You can also open this in GitHub Desktop or view command line instructions.

8. Click on the "Confirm merge" button.



tupstudent wants to merge 5 commits into main from dev

tupstudent commented 4 minutes ago

No description provided.

sasbcl and others added 5 commits 9 years ago

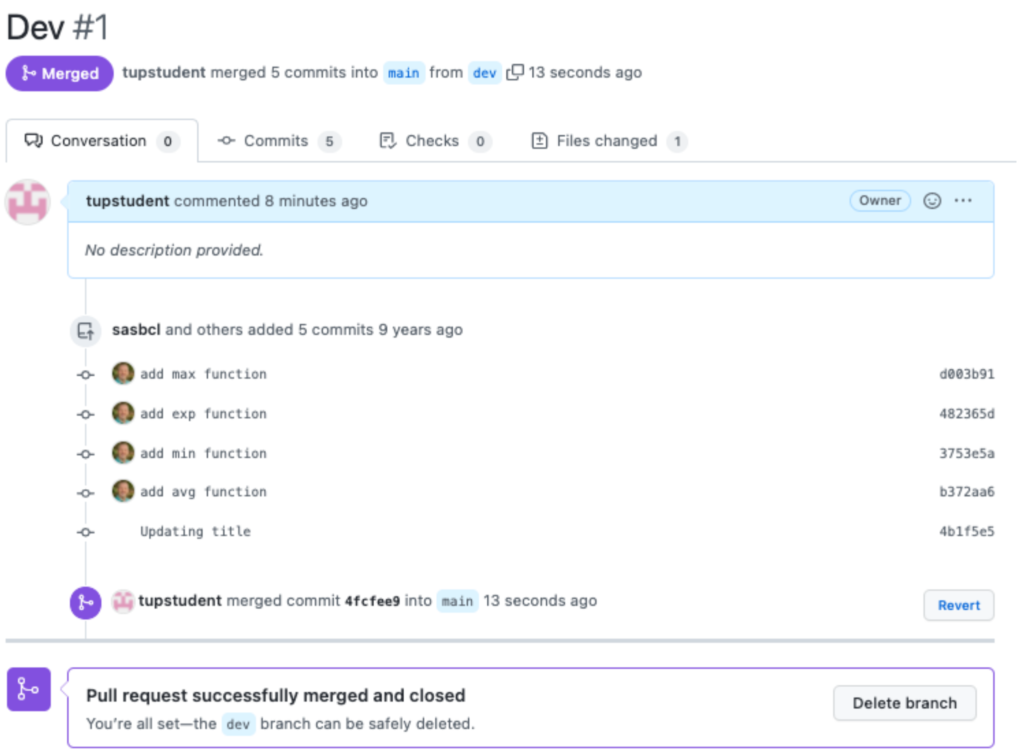
```
add max function      d003b91
add exp function      482365d
add min function      3753e5a
add avg function      b372aa6
Updating title        4b1f5e5
```

Add more commits by pushing to the dev branch on tupstudent/super_calc.

Merge pull request #1 from tupstudent/dev

Dev

9. After this, you should see the screen indicating your Merge Request has been successfully merged and closed.


Dev #1

tupstudent merged 5 commits into main from dev 13 seconds ago

tupstudent commented 8 minutes ago

No description provided.

sasbcl and others added 5 commits 9 years ago

```
add max function      d003b91
add exp function      482365d
add min function      3753e5a
add avg function      b372aa6
Updating title        4b1f5e5
```

tupstudent merged commit 4fcfee9 into main 13 seconds ago

Pull request successfully merged and closed

You're all set—the dev branch can be safely deleted.

10. Let's do one more. Back in your terminal, make sure you are on the main branch. Then create a new branch called "test".

```
$ git checkout main
$ git checkout -b test
```

11. Then, as you did before, edit the calc.html file and add in "Test" instead of your GitHub userid.

```
Edit calc.html and change

     <title>Calc</title>
                      to

     <title>Test Calc</title>
```

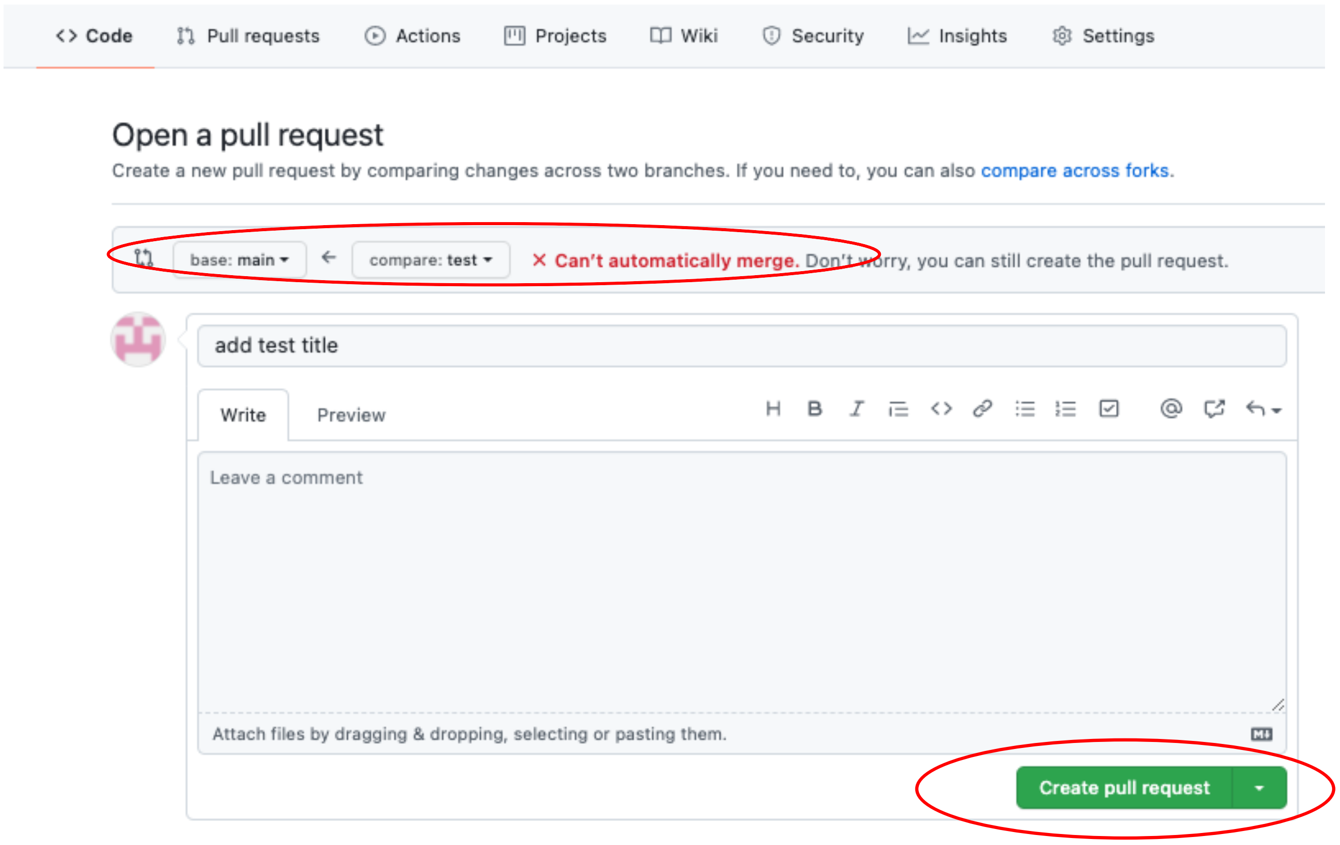
10. Save your changes and commit them back into the repository. Then push the new branch back to the remote. (Enter your PAT when prompted as before.)

```
$ git commit -am "add test title"
$ git push origin test:test
```

12. You'll see the same automatic message giving you a URL to go to create a new pull request. Something like:

```
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:      https://github.com/<your github userid>/super_calc/pull/new/test
remote:
To https://github.com/<your github userid>/super_calc.git
          * [new branch]      test -> test
```

13. Open/go to the link in the middle of the message for "Create a pull request…". You'll be at a similar screen as before. Change the project in the left top dropdown to be the same project as the one on the right. (As you did back in step 5.) At that point, the targets will switch to "base:main" and "compare:test". But there will be a warning message "Can't automatically merge." Go ahead and click the button to "Create pull request".



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

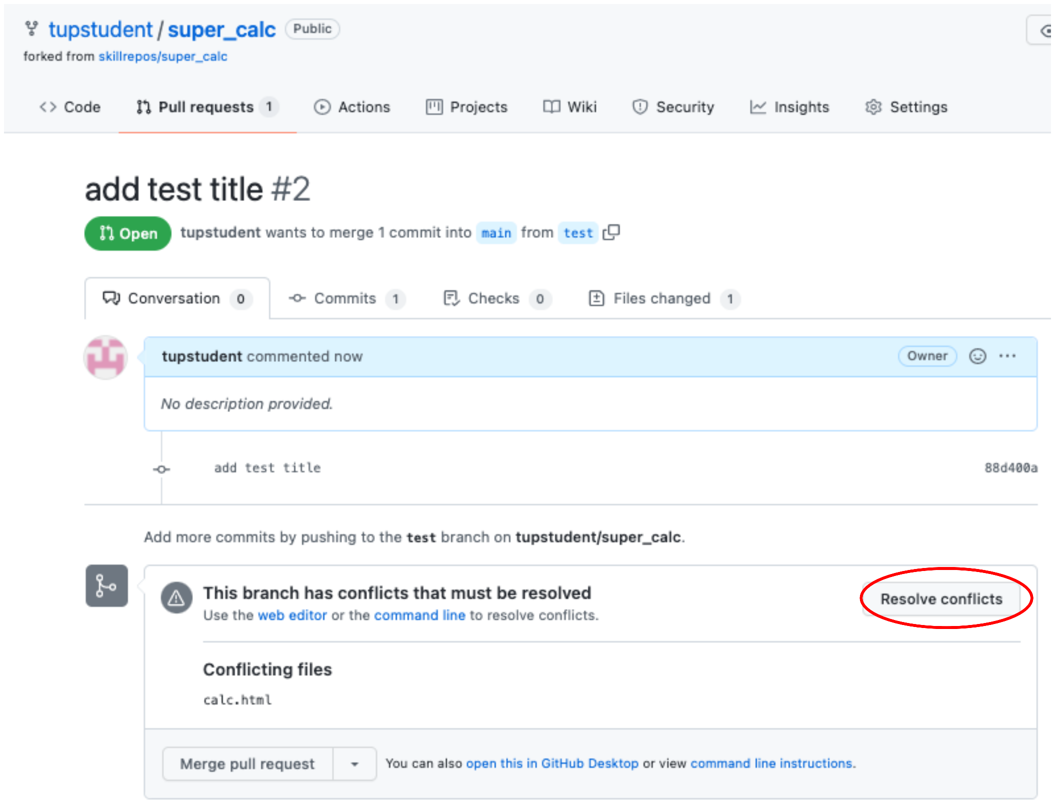
base: main ← compare: test ✕ Can't automatically merge. Don't worry, you can still create the pull request.

add test title

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

14. Notice the error message near the bottom "This branch has conflicts that must be resolved". Also there is a "Resolve conflicts" button. Click on that.


tupstudent / super_calc Public
forked from skillrepos/super_calc
Code
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings

add test title #2

Open tupstudent wants to merge 1 commit into `main` from `test`

tupstudent commented now

No description provided.

```
add test title                                   88d400a
```

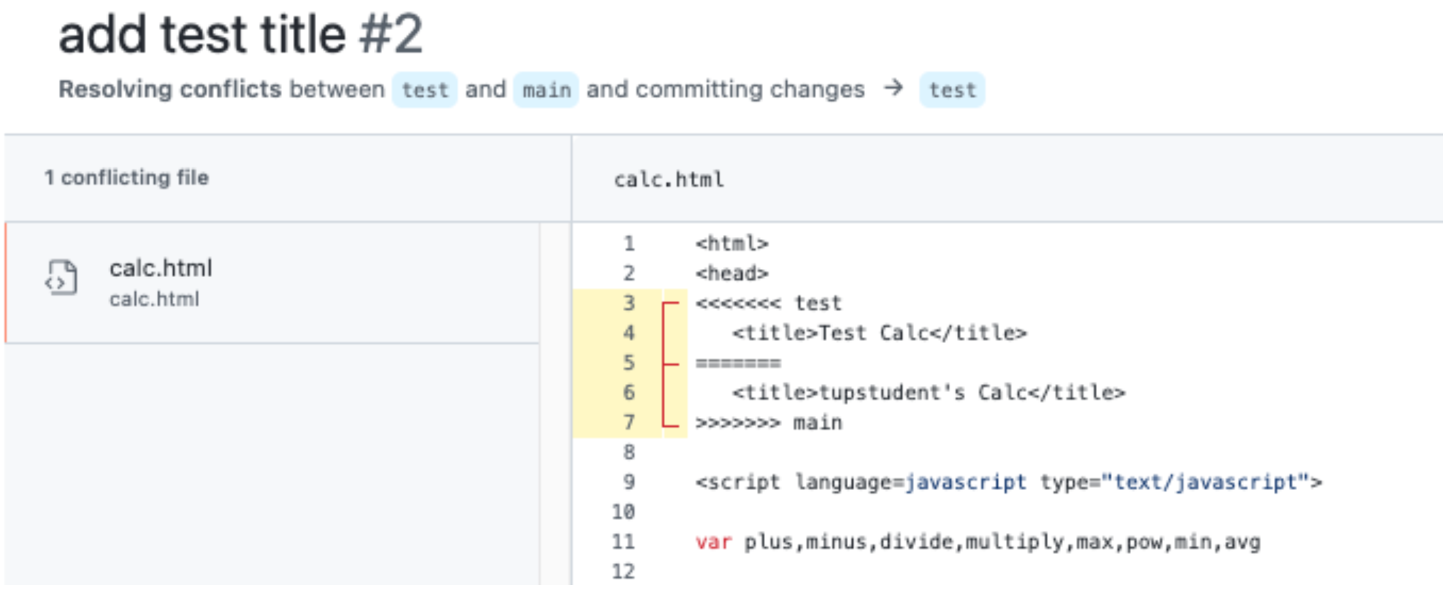
Add more commits by pushing to the **test** branch on **tupstudent/super_calc**.

This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts.

Conflicting files
`calc.html`

You can also open this in GitHub Desktop or view command line instructions.

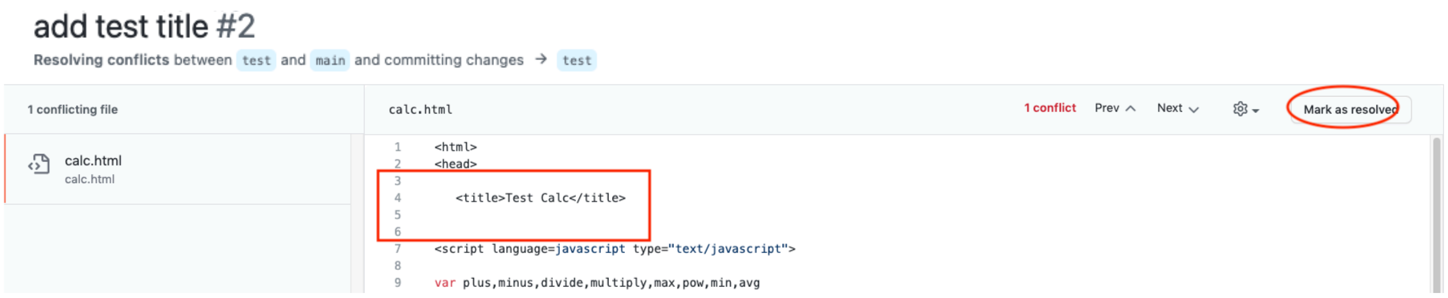
15. This will now put you on a screen showing the conflicts.


add test title #2

Resolving conflicts between `test` and `main` and committing changes → `test`

```
1   <html>
2   <head>
3   <<<<<<< test
4      <title>Test Calc</title>
5   =======
6      <title>tupstudent's Calc</title>
7   >>>>>>> main
8
9   <script language=javascript type="text/javascript">
10
11  var plus,minus,divide,multiply,max,pow,min,avg
12
```

16. You can highlight and hit the "Del" key to get rid of lines 3, 5, 6, and 7 to remove the markers and resolve the conflict. Then you can click on the "Mark as resolved" button.


add test title #2

Resolving conflicts between test and main and committing changes → test

1 conflicting file

calc.html
calc.html

calc.html — 1 conflict — Prev — Next — Mark as resolved

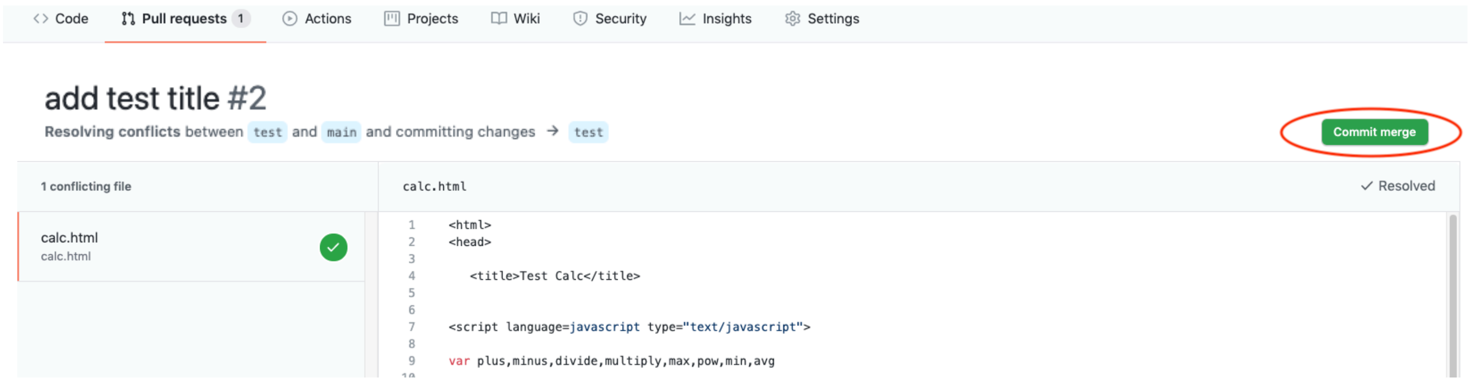
```
<html>
<head>

    <title>Test Calc</title>


<script language=javascript type="text/javascript">

var plus,minus,divide,multiply,max,pow,min,avg
```

17. At this point, the check should turn green, the status should indicate "Resolved" and you can click the new green "Commit merge" button.



add test title #2

Resolving conflicts between test and main and committing changes → test

Commit merge

1 conflicting file

calc.html
calc.html

calc.html — ✓ Resolved

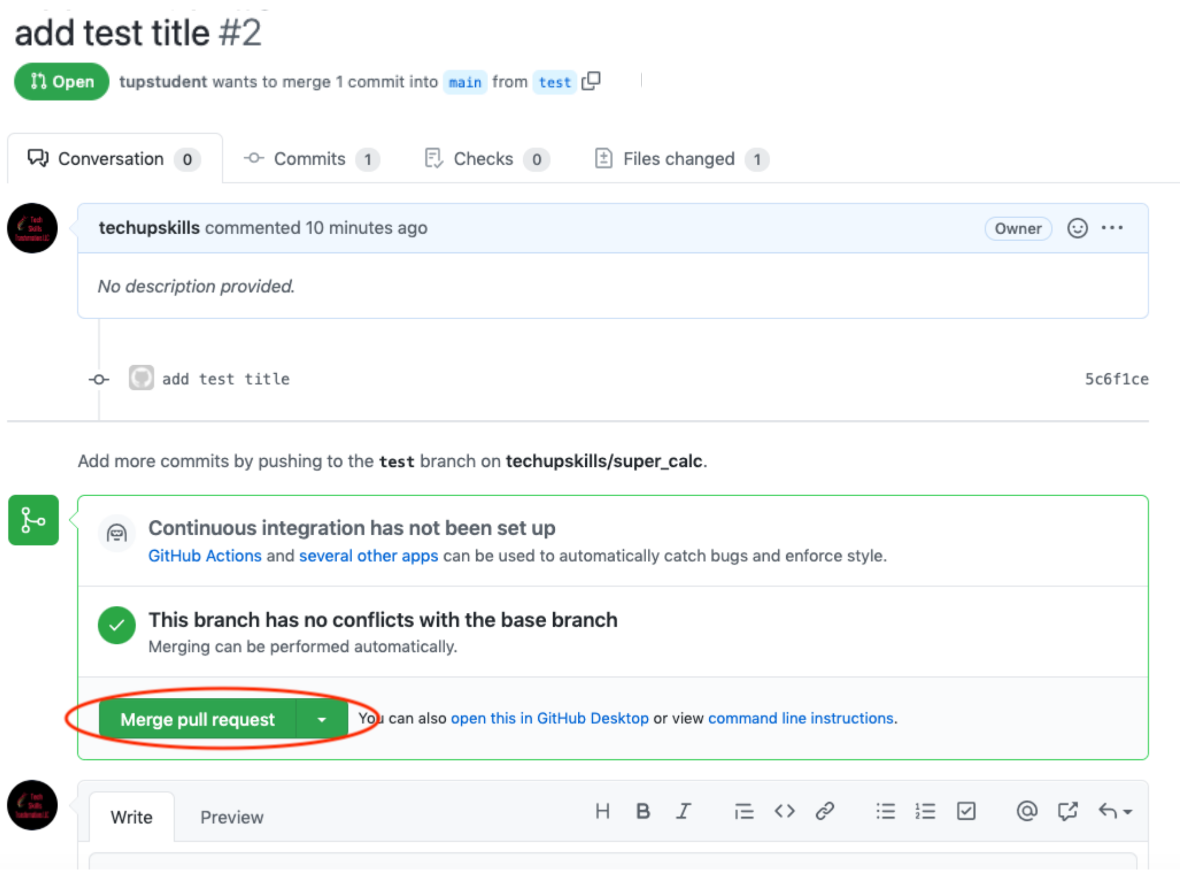
```
<html>
<head>

    <title>Test Calc</title>


<script language=javascript type="text/javascript">

var plus,minus,divide,multiply,max,pow,min,avg
```

18. Now you can merge the pull request.


add test title #2

Open tupstudent wants to merge 1 commit into main from test

Conversation 0 | Commits 1 | Checks 0 | Files changed 1

techupskills commented 10 minutes ago — Owner

No description provided.

add test title 5c6f1ce

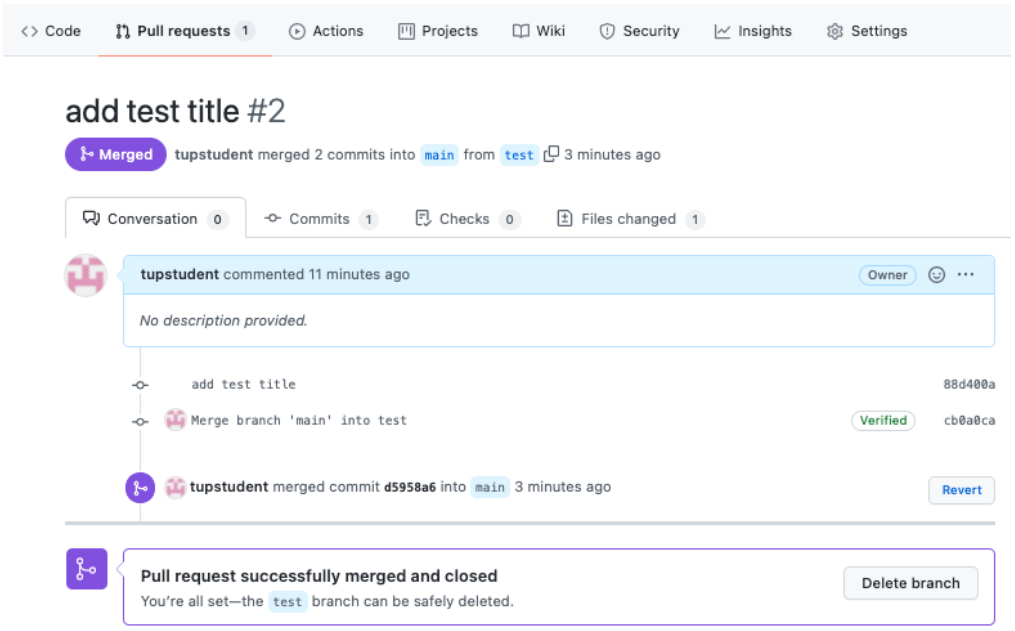
Add more commits by pushing to the **test** branch on **techupskills/super_calc**.

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.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

19. Click the button to Confirm. And then you'll have another completed pull request.



add test title #2

Merged tupstudent merged 2 commits into `main` from `test` 3 minutes ago

tupstudent commented 11 minutes ago Owner

No description provided.

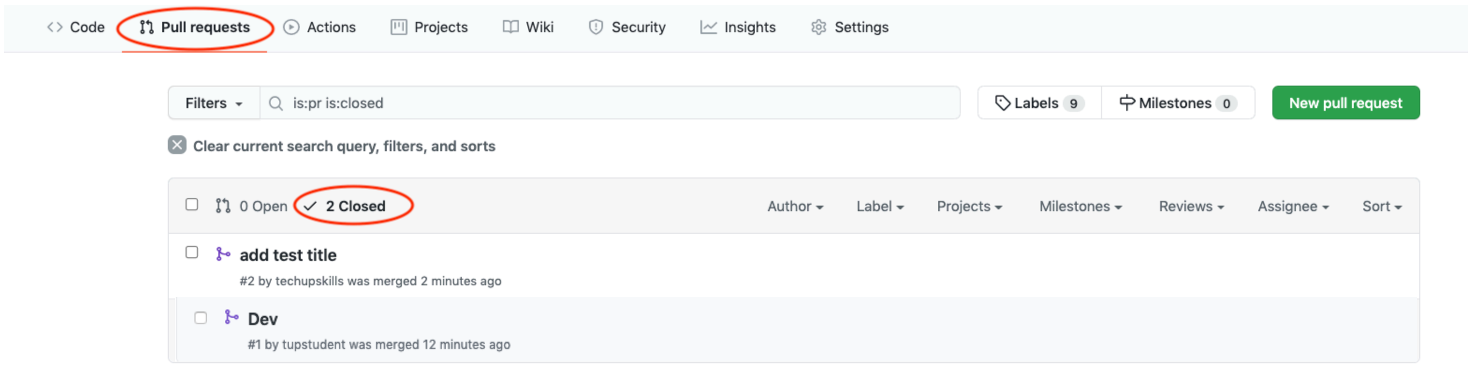
```
add test title                                   88d400a
Merge branch 'main' into test          Verified  cb0a0ca
```

tupstudent merged commit **d5958a6** into `main` 3 minutes ago

Pull request successfully merged and closed

You're all set—the `test` branch can be safely deleted.

20. If you click on the "Pull requests" link in the line that begins with "<> Code" you can see the pull request summary. Then if you click on the "2 Closed" link, you can see the ones that you successfully completed.



add test title
#2 by techupskills was merged 2 minutes ago

Dev
#1 by tupstudent was merged 12 minutes ago

==

END OF LAB

==

Lab 13 - Subtrees

Purpose: In this lab, we'll see one way to manage "groups" of repositories in Git

1. Start out in the *super_calc* directory for the *super_calc* repository that you used in the last two labs. You'll want to be in the main branch.

```
$ git checkout main
```

2. You're going to add another repository (the sub_docs one) as a subtree to super_calc.

```
$ git subtree add -P sub_docs --squash https://github.com/<your github user id>/sub_docs main
```

Even though you don't have much history in this repository, you used the --squash command to compress it. Note that the -P stands for prefix, which is the name your subdirectory gets.

3. Look at the directory structure; note that the sub_docs subdirectory is there under your super_calc project. Also, if you do a git log, you can see where the subproject was added, and the history squashed.

```
$ ls sub_docs
$ git log --oneline
```

Note that there is only one set of history here because there is only one project effectively - even though we have added a repository as a subproject.

4. Now, you will see how to update a subproject that is included as a subtree when the remote repository is updated. First, clone the sub_docs project down into a different area.

```
$ cd ..
$ git clone https://github.com/<your github user id>/sub_docs sub_docs_temp
```

5. Change into the sub_docs_temp project, and create a simple file. Then stage it, commit it, and push it. (Enter your PAT when prompted as before.)

```
$ cd sub_docs_temp
$ echo "readme content" > readme.txt
$ git add .
$ git commit -m "Adding readme file"
$ git push
```

6. Go back to the super_calc project where you have sub_docs as a subtree.

```
$ cd ../super_calc
```

7. To simplify future updating of your subproject, add a remote reference for the subtree's remote repository.

```
$ git remote add sub_docs_remote https://github.com/<your github user id>/sub_docs
```

8. You want to update your subtree project from the remote. To do this, you can use the following subtree pull command. Note that it's similar to your add command, but with a few differences:

- You use the long version of the prefix option.
- You are using the remote reference you set up in the previous step.
- You don't have to use the squash option, but you add it as a good general practice.

```
$ git subtree pull --prefix sub_docs sub_docs_remote main --squash
```

Because this creates a merge commit, you will get prompted to enter a commit message in an editor session. You can add your own message or just exit the editor.

9. After the command from step 8 completes, you can see the new README file that you created in your subproject *sub_docs*. If you look at the log, you can also see another record for the squash and merge commit that occurred.

```
$ ls sub_docs
$ git log --oneline
```

10. Changes you make locally in the subproject can be promoted the same way using the subtree push command. Change into the subproject, make a simple change to your new README file, and then stage and commit it.

```
$ cd sub_docs
$ echo "update" >> readme.txt
$ git commit -am "update readme"
```

11. Change back to the directory of the *super_project*. Then use the subtree push command below to push back to the project's remote repository. (Enter your PAT when prompted as before.)

```
$ cd ..
$ git subtree push --prefix sub_docs sub_docs_remote main
```

Note the similarity between the form of the subtree push command and the other subtree commands you've used.

==

END OF LAB

==

Lab 14: Creating a post-commit hook

Purpose: In this lab, we'll see how to put a post-commit hook in place to be active in our local Git environment. The hook could be created in many different programming languages, but we'll use shell scripting here because it's portable among most unix implementations (including the Git bash shell for Windows).

Setup:

Suppose that you want to mirror out copies of files when you commit them into your local repository - but only for branches that start with "web". Further you must have the config value of hooks.webdir set to a valid directory on your system.

1. The code for the hook is already done for you. Clone down a copy of the repo containing it.

```
$ cd ..

$ git clone https://github.com/skillrepos/git4-hooks
```

2. In the cloned directory are two files - *post-commit-v1.txt* and *post-commit-v2.txt*. You can start out doing this lab with the v1 version, but on some systems, you may need to switch out the v2 version if v1 doesn't work as expected. You can also see the current set of sample hooks in the .git/hooks directory.

```
$ cd git4-hooks

$ ls (and cat files if desired)

$ ls .git/hooks
```

3. Move (or copy) the post-commit-v1.txt file to the .git/hooks directory as the name *post-commit* WITHOUT the extension, and make it executable.

```
$ mv post-commit-v1.txt .git/hooks/post-commit

$ chmod +x .git/hooks/post-commit
```

4. Create a directory (somewhere outside of your local Git working directory) for the hook to eventually mirror your code into.

```
$ mkdir (mirror-directory-name)
```
(such as mkdir ../mirror)

(in your working directory with the git project)

5. In your working directory configure the hooks.webdir value to point to the location you set above.

```
$ git config hooks.webdir  (mirror-directory-name)
```

(make sure to use the correct absolute or relative path to get to the directory you created above, such as git config hooks.webdir ../mirror)

6. Next, we'll test out our hook. In your working directory, create a new file, then stage and commit it.

```
$ echo stuff > some-file-name

$ git add .

$ git commit -m "new file"
```

7. Notice that after you do the commit, you should see the "*Running post-commit hook*" message. (This comes from the 4th line of our script that is the hook.) However, the hook won't do anything else because the branch is main and not web*. To verify that's the case, you can inspect the directory you created for the mirror.

```
$ ls (mirror-directory-name)
```

There should be nothing there.

8. Now, let's set things up so our hook will mirror out the contents of our repository. Create a new branch and switch to it. You can name it anything you want as long as it starts with "web". An example is shown below.

```
$ git checkout -b webtest
```

9. Create another file and stage and commit it into your repository.

```
$ echo stuff > another-file-name

$ git add .

$ git commit -m "new file"
```

10. This time, the hook should fire since we have the config value defined and are in a branch that starts with "web". You should see the "Running" message from the hook and then be able to see the contents of your repository in the directory that you configured.

```
$ ls (mirror-directory-name)
```

11. If the hook doesn't seem to work, you may need to repeat step 3 with the post-commit-v2.txt version of the hook and then steps 8 and 9.

==

END OF LAB

==

(Optional) Lab 15: Using a Git Attributes File to Create a Custom Filter

Purpose: In this lab, we'll see how to work with two kinds of custom filters to modify file contents automatically on checkout and commit.

Setup:

Suppose that you have a couple of HTML files that you use as a header and footer across multiple divisions in your company. They contain a placeholder in the form of the text string %div to indicate where the proper division name should be inserted.

You want to use the smudge and clean filters to automatically replace the placeholder with your division name (ABC) when you check the file out of Git and set it back to the generic version if you make any other changes and commit them.

1. Pick one of your directories that already has a git project in it. (You can use the roarv2 directory from the recent labs for simplicity if you want.) Create two sample files to work with. Commands to create them are below (you may use an editor if you prefer).

```
$ echo "<H1>Running tests for division:%div</H1>" > div_test_header.html

 $ echo "<H1>Division:%div testing summary</H1>" > div_test_footer.html
```

2. Go ahead and stage and commit these new files into your local Git repository.

```
$ git add div*.html

$ git commit -m "adding div html files"
```

3. Now, we'll create a custom filter named "insertDivisionABC" that will have the associated "clean" and "smudge" actions. This is done by using git config to define this in our Git configuration. Execute the following two config commands:

```
$ git config filter.insertDivisionABC.smudge "sed 's/%div/ABC/'"

    $ git config filter.insertDivisionABC.clean "sed 's/ision:ABC/ision:%div/'"
```

4. To see how this is setup in the config file, take a look at your .git/config file and find the section for the insertDivisionABC filter:

```
$ cat .git/config
```

Look for the section starting with "[filter "insertDivisionABC"]

5. Now, we just need to create a Git attributes file with a line that tells Git to run your filter for files matching the desired pattern. We will create a very simple one-line file for this.

```
$  echo "div*.html filter=insertDivisionABC" > .gitattributes
```

6. Now, we'll remove our local copies of the files and check them out again so they will get the filter applied.

```
$ rm div*.html
$ git checkout div*.html
```

7. Take a look at the contents of the two files after the checkout has applied the smudge filter.

```
$ cat div*.html
```

Notice that we have the division name (ABC) inserted into the files now.

8. Edit the div*.html files and make a simple modification. For example, you might change "testing summary" to "full testing summary" and/or "tests for division" to "all tests for division". Save the changes.

9. Now do a git diff on the files. Notice that with the clean filter in place, the diff takes into account the filter and just shows you the differences without the substitution being applied.

```
$ git diff
```

10. Add the files back into Git and run a status command to see the state.

```
$ git add div*.html

$ git status
```

11. Run a git diff command and note that it shows no differences. Then cat the local versions of the files. Notice that they still show the substitution from the smudge.

```
$ git diff

$ cat div*.html
```

12. We know that the clean filter should have removed the substitution when the files were staged. To verify that, we'll use a special form of the show command to see the versions of the files in the staging area.

```
$ git show :0:div_test_header.html

$ git show :0:div_test_footer.html
```

Notice that the change to insert the division has been "cleaned" per the clean filter.

==

END OF LAB

==