Object-Oriented Programming

Lecturer:

Teaching Assistant:

**Lab 03: Basic Object-Oriented Techniques**

# Assignment Submission

# Branch your repository

Merge từ nhánh refactor/apply-release-flow

A screenshot of a computer

Description automatically generated

# Working with method overloading

Nạp chồng thêm 2 phương thức trong class Cart

A computer screen shot of text

Description automatically generated

Tạo pullrequest và merge từ nhánh topic/method-overloading

A screenshot of a computer

Description automatically generated

# Passing parameter

Add a setTitle

A black screen with white text

Description automatically generated

TestPassingParamete

A screen shot of a computer program

Description automatically generated

**Questions:**

* After the call of **swap(jungleDVD, cinderellaDVD)** why does the title of these two objects still remain?
* After the call of **changeTitle(jungleDVD, cinderellaDVD.getTitle())** why is the title of the JungleDVD changed?

Answer:  
- Sau khi gọi hàm swap, vì ta truyền tham chiếu vào nên khi swap 2 object, con trỏ sẽ tham chiếu tới địa chỉ mới mà không ảnh hưởng tới đối tượng cũ, do vậy khi kết thúc hàm, đối tượng cũ vẫn không bị thay đổi.

-Sau khi gọi hàm changeTitle, ta truyền cả tham chiếu và giá trị cinderella.title(), do vậy khi dvd.setTitle, giá trị title sẽ được cập nhật. Tiếp sau đó dvd sẽ được khởi tạo giá trị tham chiếu mới tuy nhiên điều này không ảnh hưởng tới đối tượng cũ.

-> Java thay đổi khi được truyền value

Viết lại hàm swap để swap two object

A screen shot of a computer code

Description automatically generated

Merger

A screenshot of a black screen

Description automatically generated

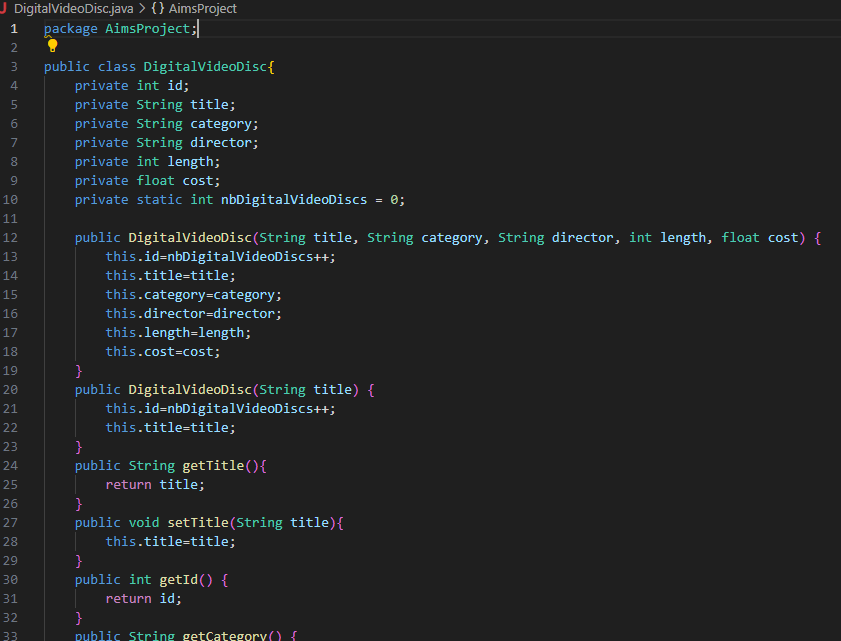
# Use debug run:

A screen shot of a computer

Description automatically generated

# Classifier Member and Instance Member

Add id and nbDigitalVideoDiscs



Pullrequest and merge :’)

A screenshot of a computer

Description automatically generated

# Open the **Cart** class

Thêm các hàm in và tìm kiếm

A screen shot of a computer program

Description automatically generated

Thêm toString và isMatch

A black background with white text

Description automatically generated

Chạy chương trình

A screenshot of a computer program

Description automatically generated

Pullrequest and merge

A screenshot of a computer

Description automatically generated

# Implement the **Store** class

# Re-organize your projects

# **String**, **StringBuilder** and **StringBuffer**

# Release flow demonstration

## 10.1. Hypothesis

We hypothesis that the Figure 25 shows the branches of our current remote repository.

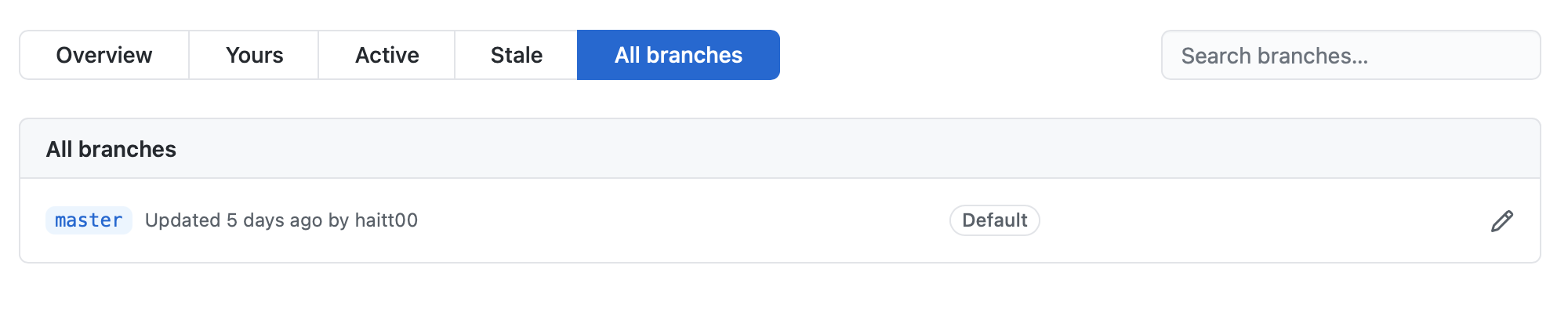


Figure . Branches of Remote Repository

Now we add a new topic or a new feature to our application. The next section shows us how to apply Release Flow in this hypothesis.

## 10.2. Demonstration

### Step 1. Update local repository.

Issue the following command and resolve conflicts if any.

(master) $ git pull

### Step 2. Create and switch to a new branch in the local repository.

(master) $ git checkout -b feature/demonstrate-release-flow

### Step 3. Make modification in the local repository.

### Step 4. Commit the change in the local repository.

(feature/demonstrate-release-flow) $ git commit –m “Add a feature for demonstration”

### Step 5. Create a new branch in the remote repository (GitHub through GUI).

* Firstly, under the “Code” tab of the top navigation bar, choose the drop-down button with the branch name (in this case “master”) on the top left.

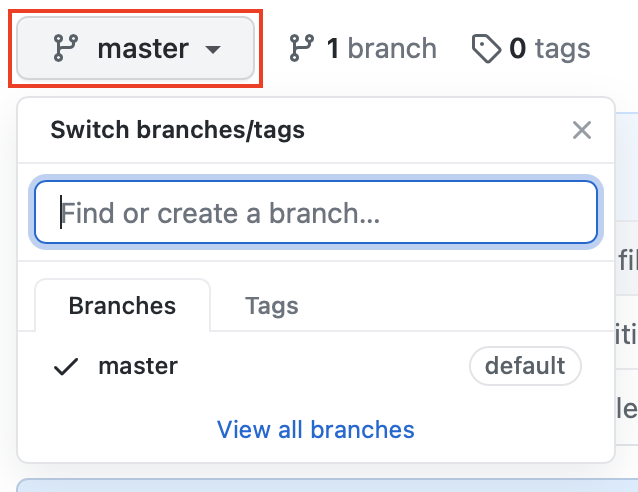


Figure . Branch Creation in GitHub GUI (1/3)

* Secondly, enter the new branch name “feature/demonstrate-release-flow” into the text field and click “Create branch: feature/demonstrate-release-flow from ‘master’”.

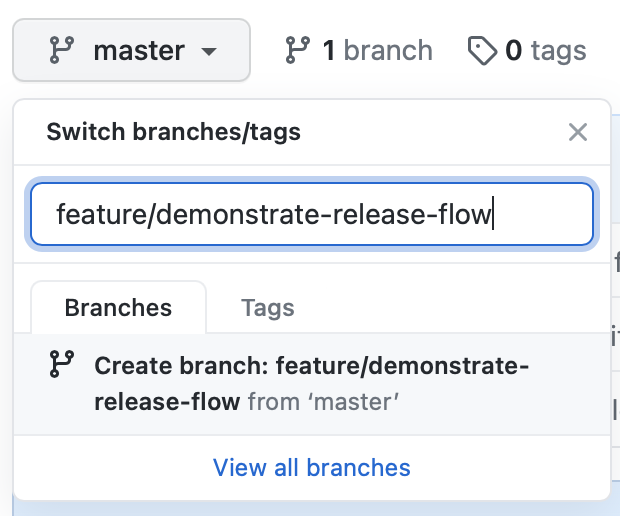


Figure . Branch Creation in GitHub GUI (2/3)

* The following figure shows the result of our efforts in this step.

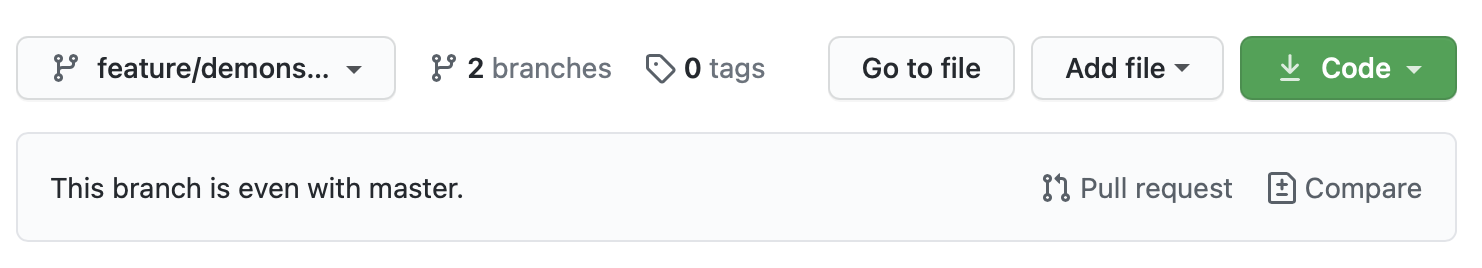


Figure . Branch Creation in GitHub GUI (3/3)

### Step 6. Push the local branch to the remote branch

(feature/demonstrate-release-flow) $ git push origin feature/demonstrate-release-flow

### Step 7. Create a pull request in GitHub GUI (for working in a team only)

* Firstly, choose “Pull requests” tab from the top navigation bar.

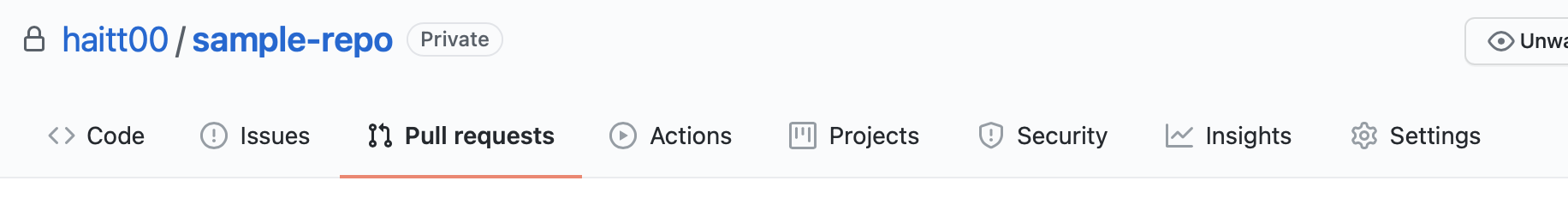


Figure . Creation of a Pull Request in GitHub GUI (1/4)

* Secondly, click the button “New pull request” in the top right corner of the interface.

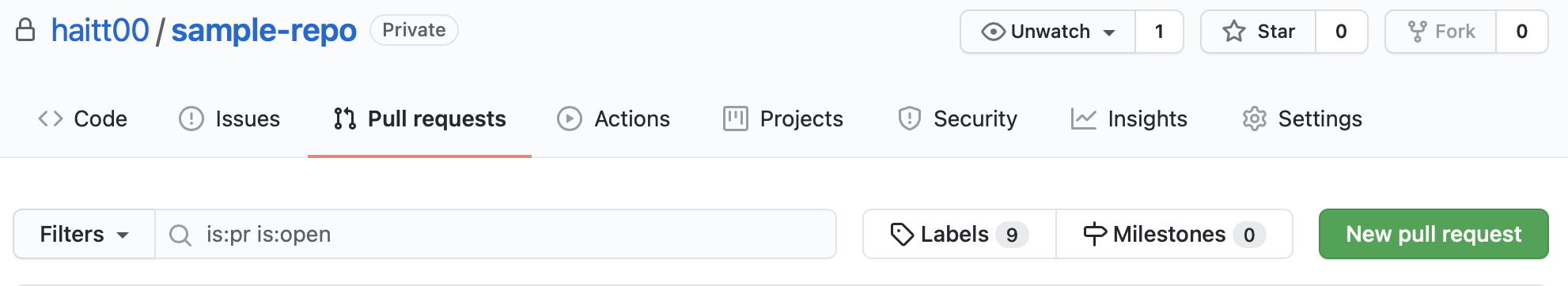
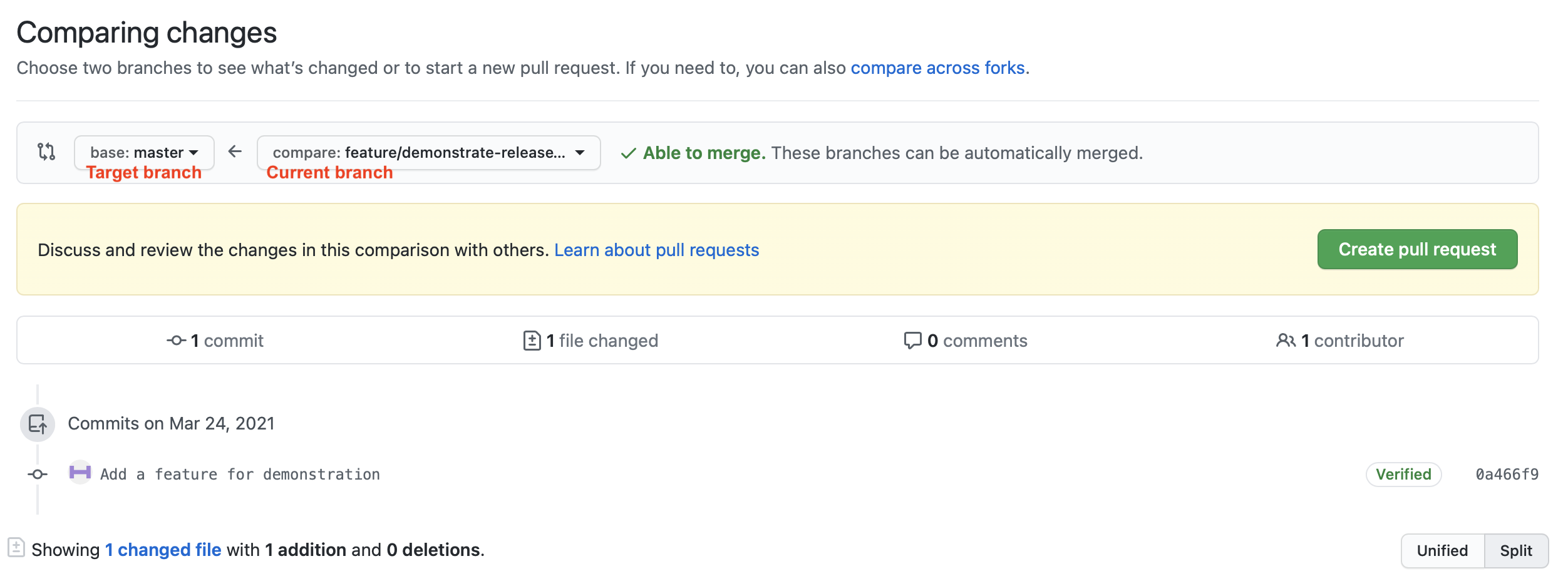


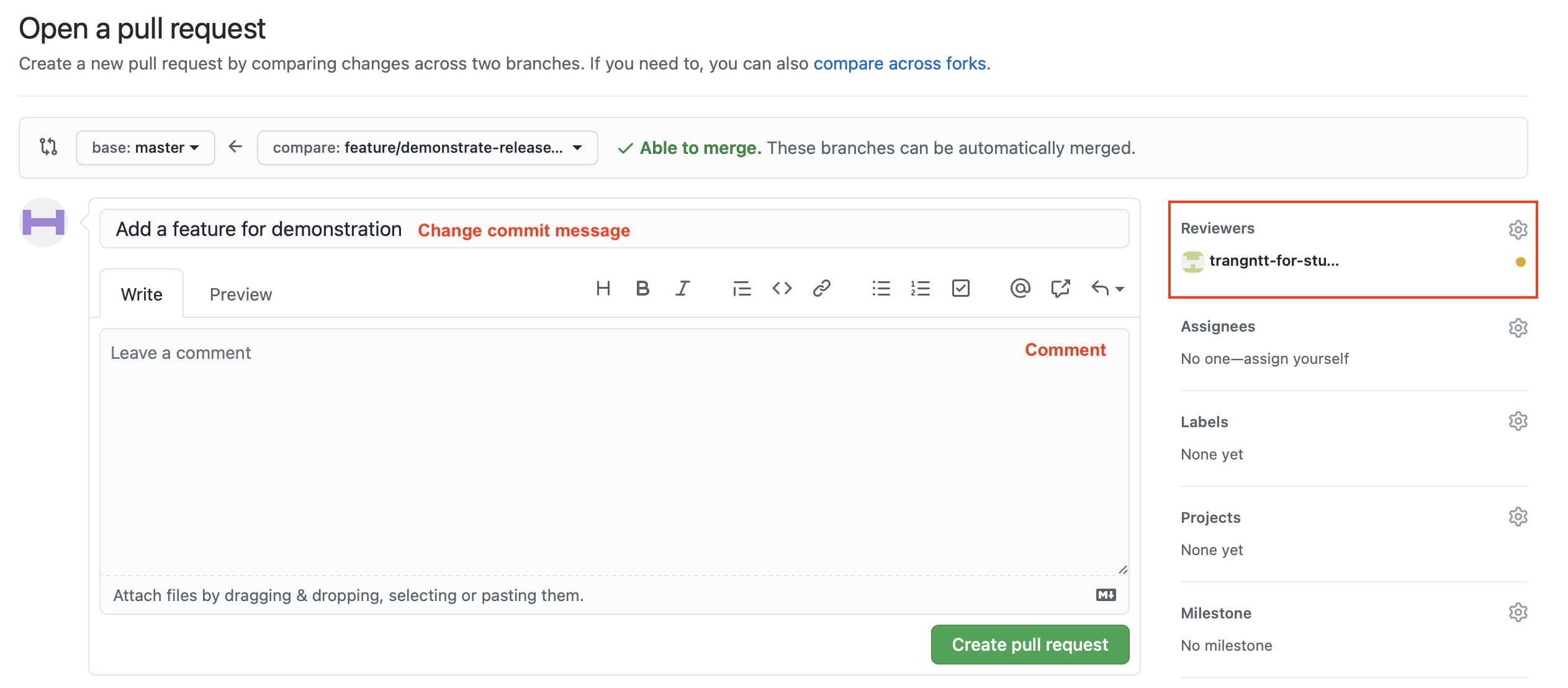
Figure . Creation of a Pull Request in GitHub GUI (2/4)

* Then, pick the target branch and current branch. Besides, at the bottom of the interface, we can see the changes between current branch and the target branch. Choose “Create pull request” to the top right.

**Note:** the target branch will affect the destination branch which we want our branch merge to in the next step.



* Figure . Creation of a Pull Request in GitHub GUI (3/4)
* Lastly, choose reviewers for the pull request. We can also change the commit message, and add comment as we desire. Choose “Create pull request”



* The following figure shows the result of our efforts in the dashboard of GitHub. The added reviewers also can see the pull requests in their dashboard. When the changes are viewed, we can merge the branches.

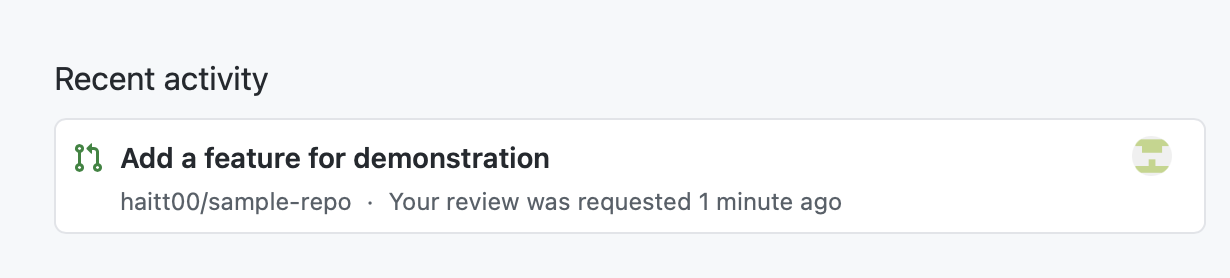


Figure . Creation of a Pull Request in GitHub GUI (4/4)

### Step 8. Merge the new remote branch to the master branch.

* Open the pull request.
* Choose “Merge pull request”. You can choose one of several merge options from the drop-down menu

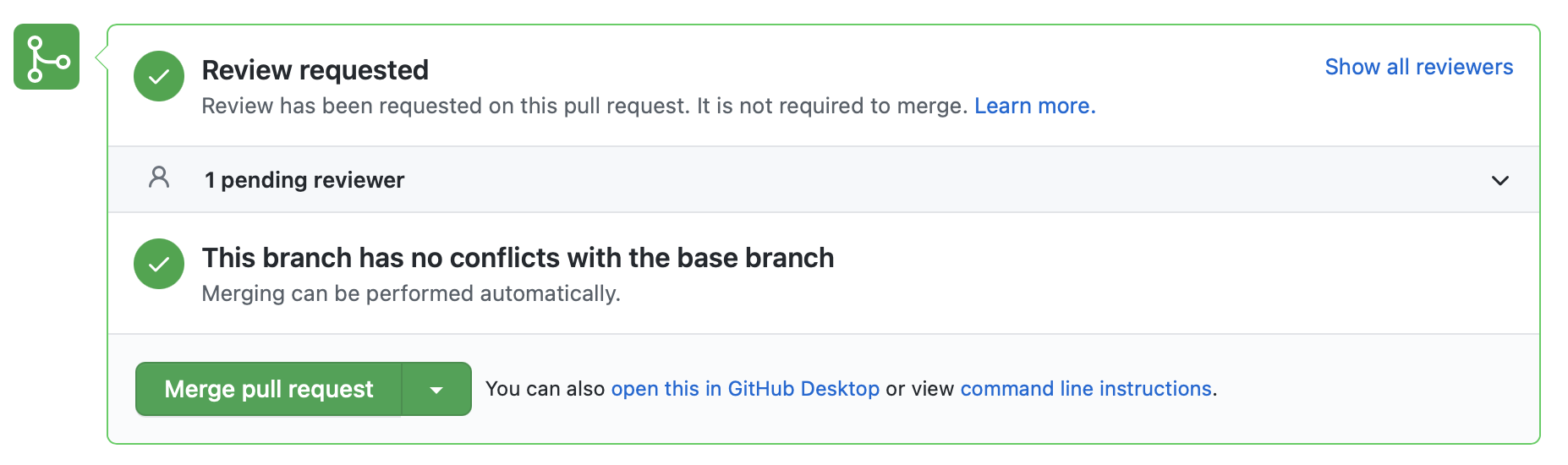


Figure . Branch merging (1/3)

* Lastly, change the commit message if need be. We cannot change the destination branch. Choose “Confirm merge” (as shown in Figure 33)

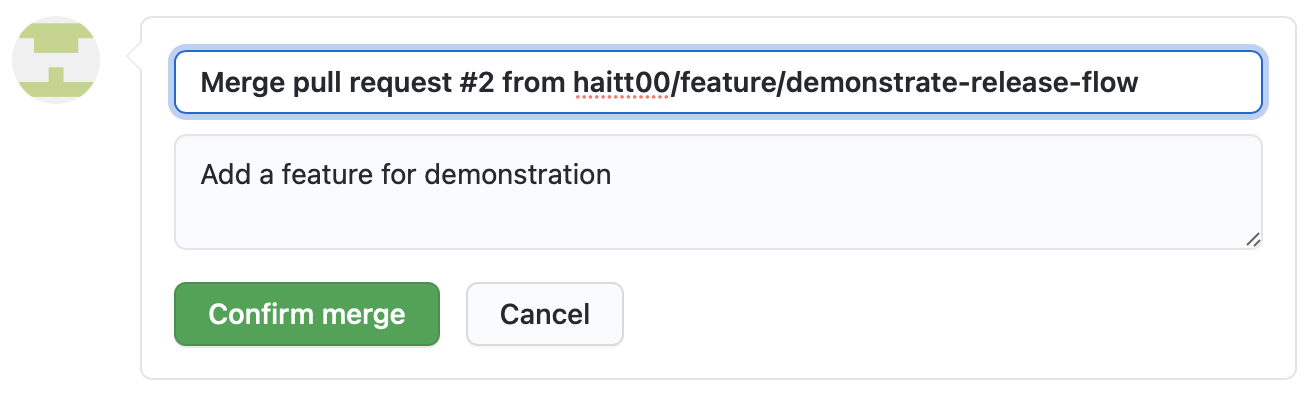


Figure . Branch merging (2/3)

* The following figure shows the result of our efforts. The changes from the target branch have been merged to the target branch “master”.

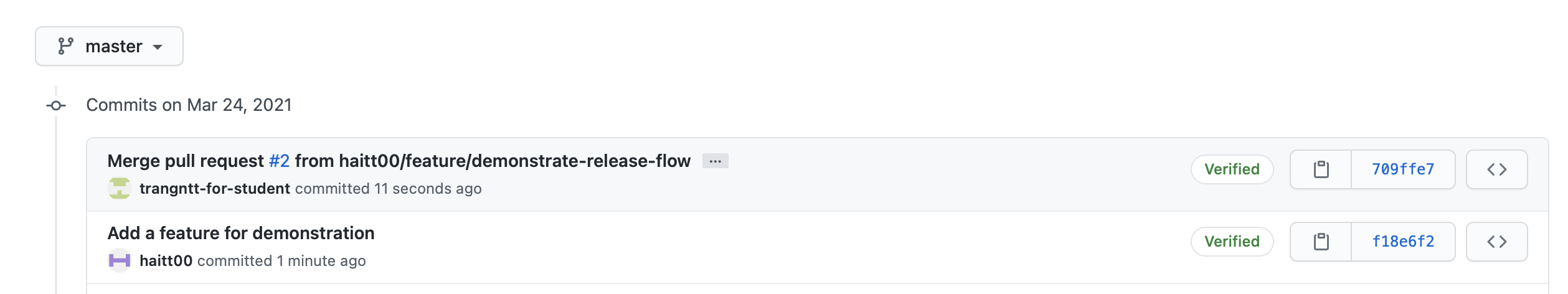


Figure . Branch merging (3/3)