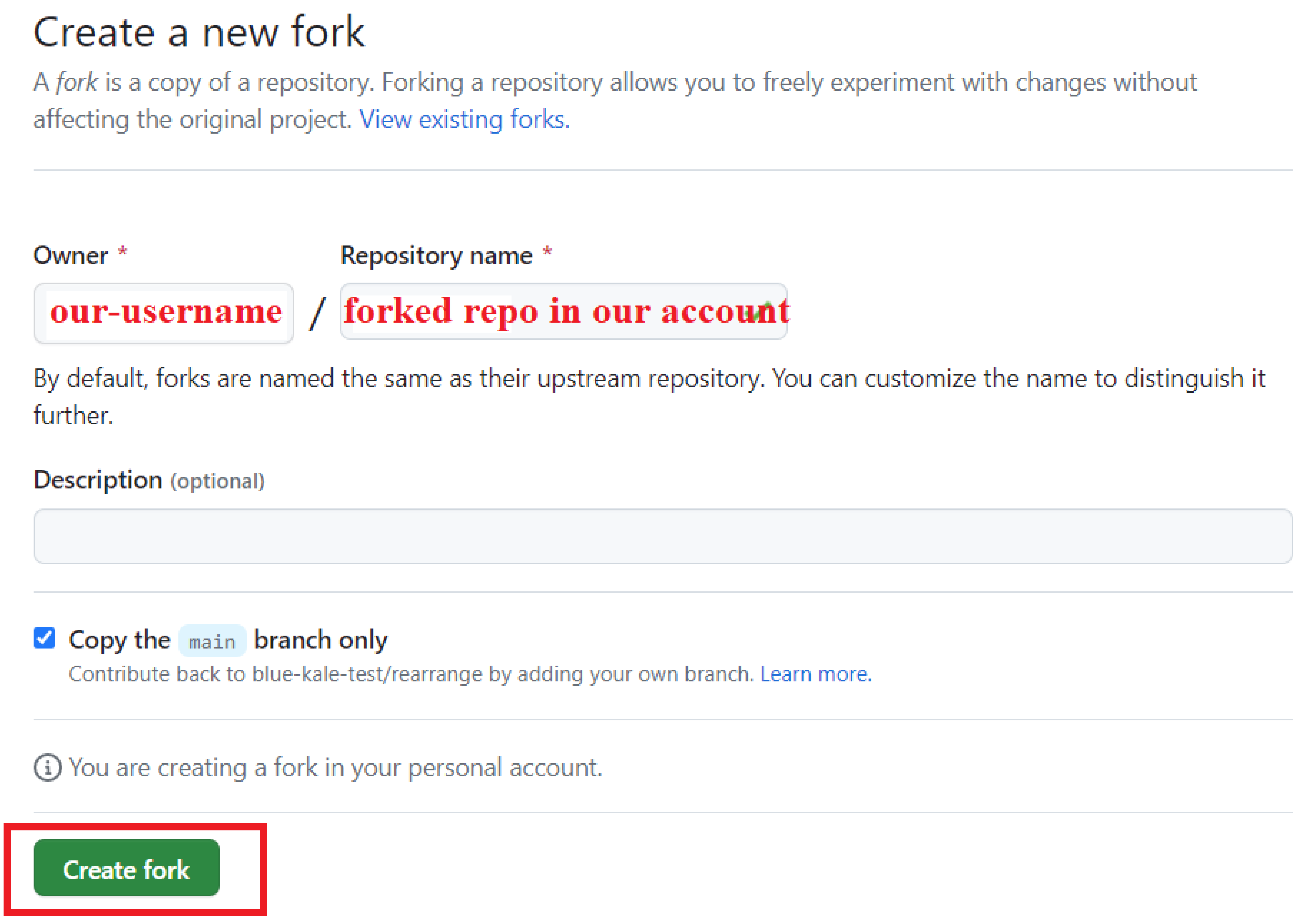
Module 4 Collaboration

1. Pull Requests

(1) Pull request

- **Forking**
  - A way of creating a copy of the given *repository* so that it belongs to our user
  - Our user can only push changes to the *forked* copy when we can't push it to the original *repository*



Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.

Owner * / Repository name *

our-username / forked repo in our account

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

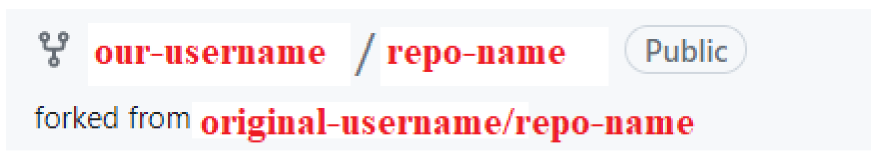
Description (optional)

☑ Copy the `main` branch only

Contribute back to blue-kale-test/rearrange by adding your own branch. Learn more.

You are creating a fork in your personal account.

forking (creating a fork)



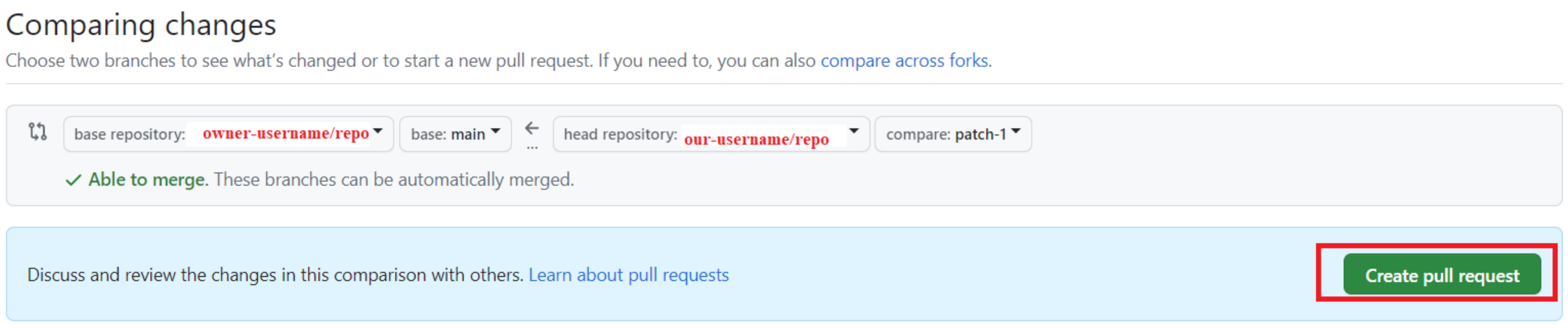
our-username / repo-name Public

forked from original-username/repo-name

forked repository

- **Pull request**
  - A commit/series of commits that you send to the owner of the *repository* so that they incorporate it into their tree
  - Used to <u>suggest</u> patches/bug fixes/new figures to the owner of the *repository*

- The owners will review it before merging it to their *repository*



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

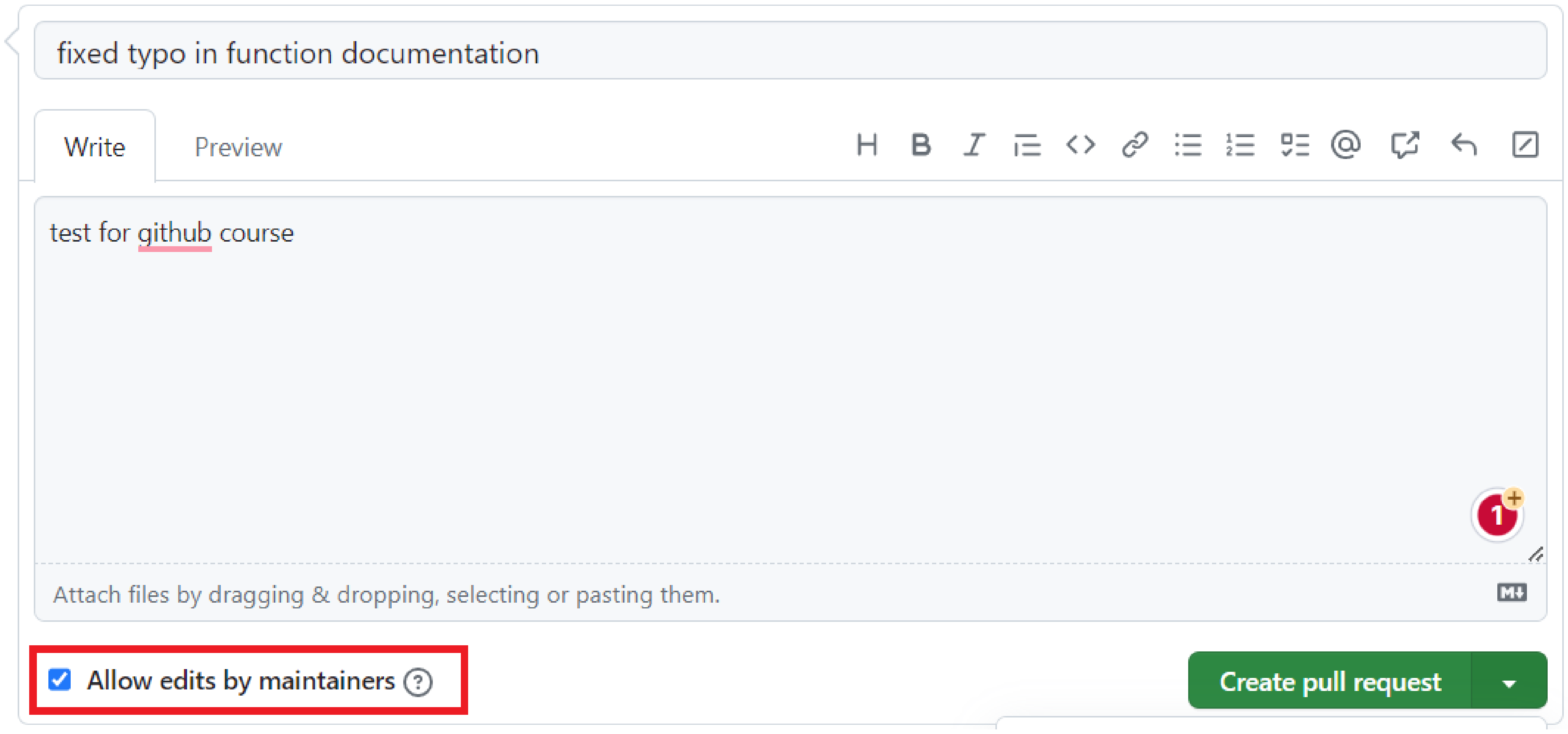
base repository: owner-username/repo base: main ← head repository: our-username/repo compare: patch-1

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

comparing changes and create pull request

- **Check-box**: *allow edits from maintainers*
    - the maintainer can edit or rebase the *commit* afterwards without asking us to do it



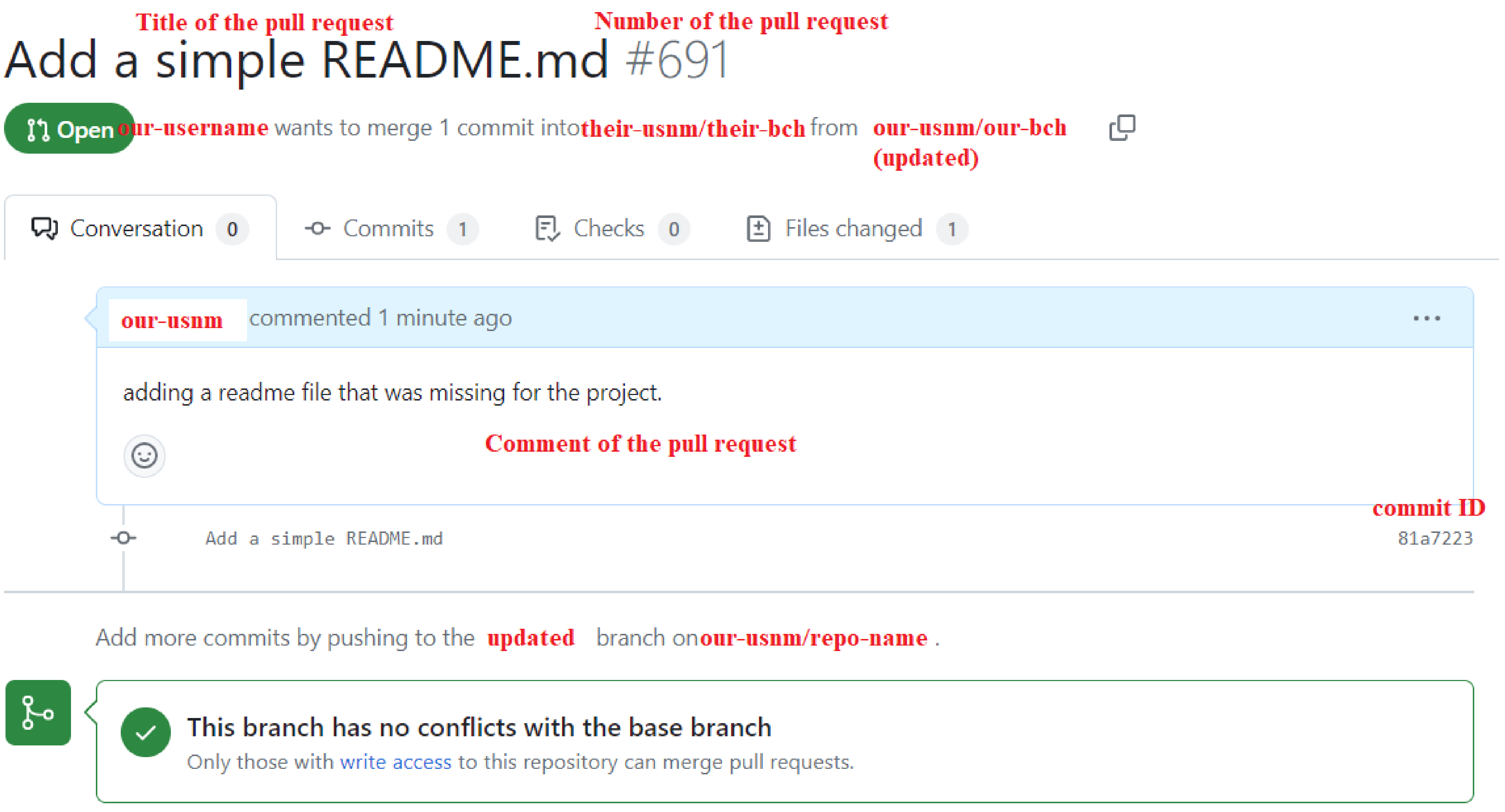
fixed typo in function documentation

test for github course

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

☑ Allow edits by maintainers

checkbox: allow edits by maintainers



Add a simple README.md #691

Open our-username wants to merge 1 commit into their-usnm/their-bch from our-usnm/our-bch (updated)

Conversation 0 · Commits 1 · Checks 0 · Files changed 1

our-usnm commented 1 minute ago

adding a readme file that was missing for the project.

Add a simple README.md 81a7223

Add more commits by pushing to the updated branch on our-usnm/repo-name .

This branch has no conflicts with the base branch

Only those with write access to this repository can merge pull requests.

Title of the pull request
Number of the pull request
Comment of the pull request
commit ID

pull request interface

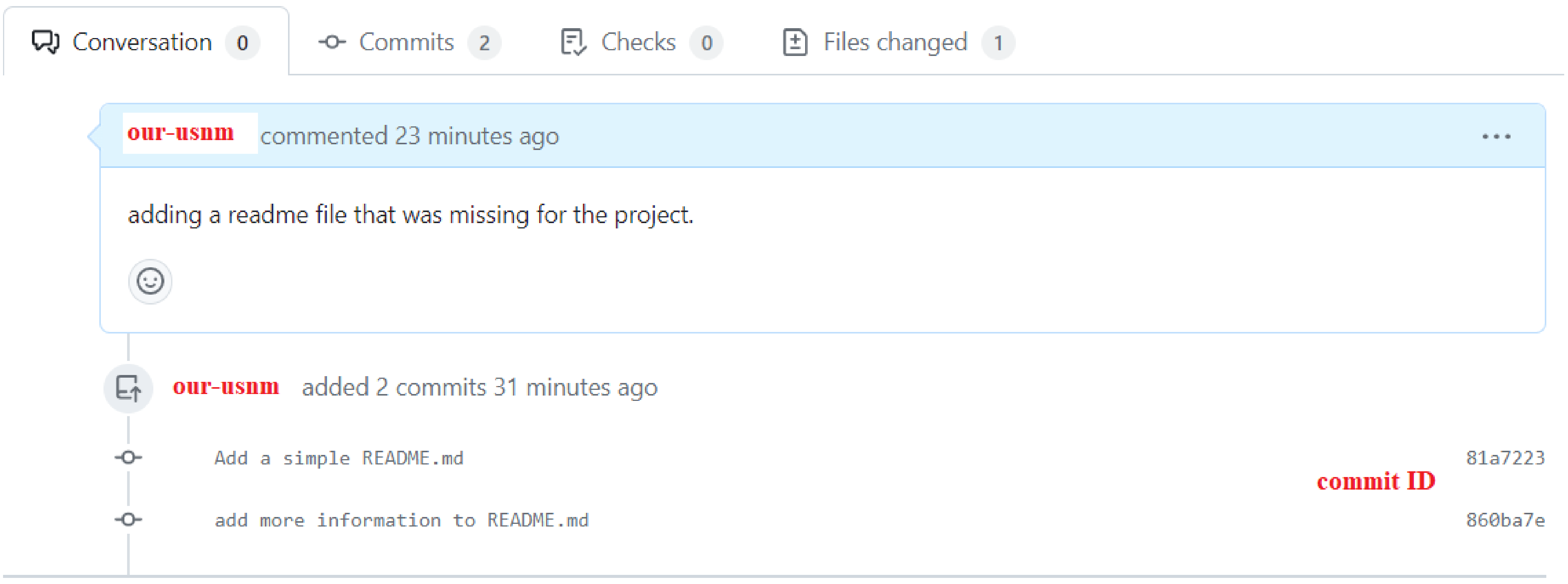
(2) Pull request workflow

```
# Fork the repository in GitHub
cd [storage-path] # Change the directory to the local path for storing the forked repo
git clone https://github.com/[our-username]/[repo-name].git # Clone the forked repo to our local
storage
cd [repo-name] # Direct the git to the path of the forked repo
git log # Check previous log files from the original repo owners
git checkout -b [new-branch] # Create a new branch for making changes
code README.md # Use Visual Studio Code (other code editors are also fine) to create a README.md file
git add README.md # Add the newly created file to the git repo
git commit -m "[commit message]" # Commit the change with a short [commit message]
git push -u origin [new-branch] # Create a new branch online and push the local new branch to that
branch
# Create a pull request in GitHub
```

a general pull request workflow

(3) Updating an existing pull request

- Pushing another *commit* to the same branch as before will result in the update of the same *pull request*



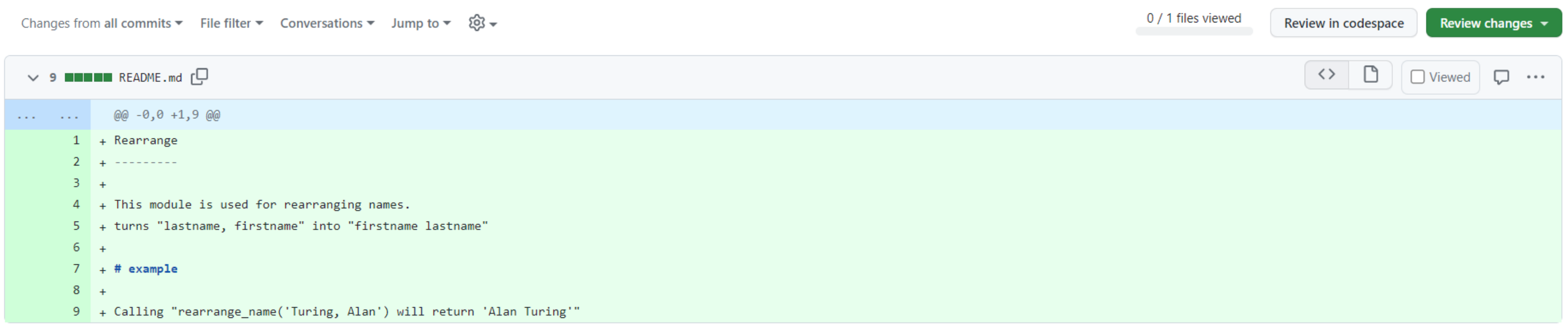
our-usnm commented 23 minutes ago

adding a readme file that was missing for the project.

our-usnm added 2 commits 31 minutes ago

```
Add a simple README.md                          81a7223
add more information to README.md               860ba7e
```

2 commits in 1 pull request



```
@@ -0,0 +1,9 @@
+ Rearrange
+ ---------
+
+ This module is used for rearranging names.
+ turns "lastname, firstname" into "firstname lastname"
+
+ # example
+
+ Calling "rearrange_name('Turing, Alan') will return 'Alan Turing'"
```

changes are squeezed in one diff

- To create a new *pull request*, we need to create another *branch*

(4) Squashing changes

```
git rebase -i <branch-2>
```

Function: interactively rebases the current branch onto , i.e., shows the *commits* of the current branch from the oldest to the most recent

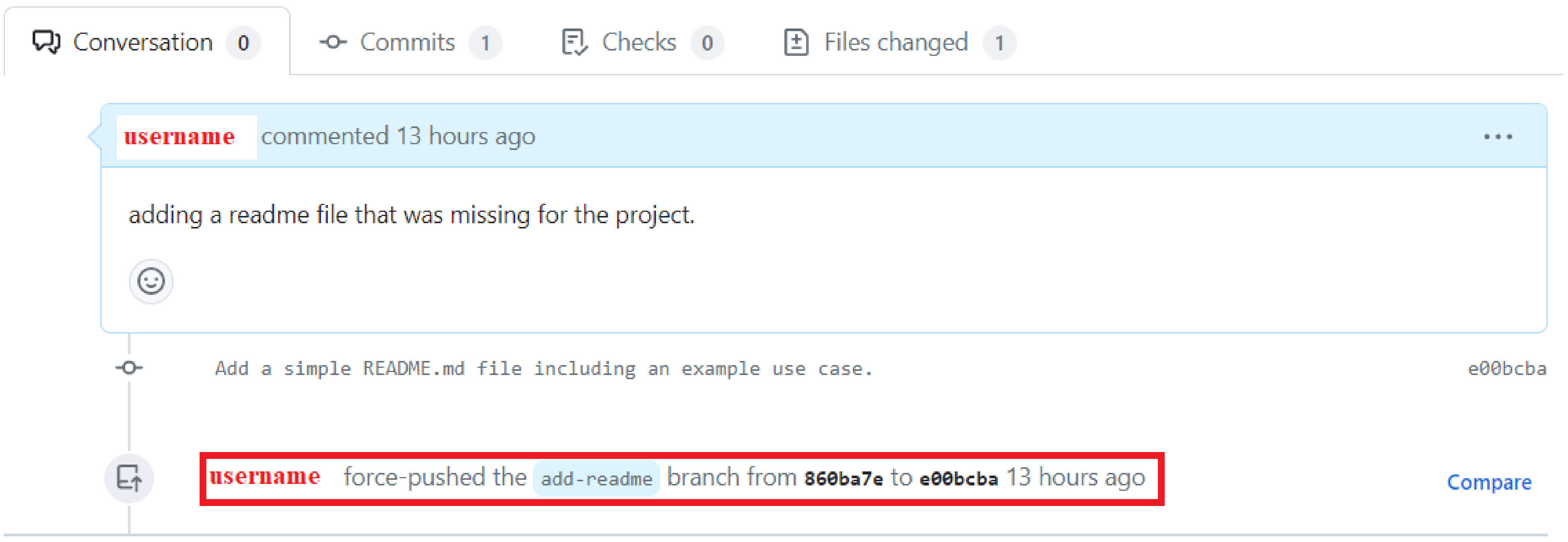
- **Interactive commands**
    - *Squash*: meld all *commit messages* together while allowing for changes
    - *Fixup*: take the *commit* while discarding its *commit message*

```
git show
```

Function: checks the latest *commit* and the changes in it

```
git push -f
```

Function: forces git to push the current snapshot into the *online repository* as is (generally used for squashing several *commits* into one *commit*; **May result in permanent data loss**)


Conversation 0 · Commits 1 · Checks 0 · Files changed 1

username commented 13 hours ago

adding a readme file that was missing for the project.

```
Add a simple README.md file including an example use case.    e00bcba
```

username force-pushed the `add-readme` branch from **860ba7e** to **e00bcba** 13 hours ago

Compare

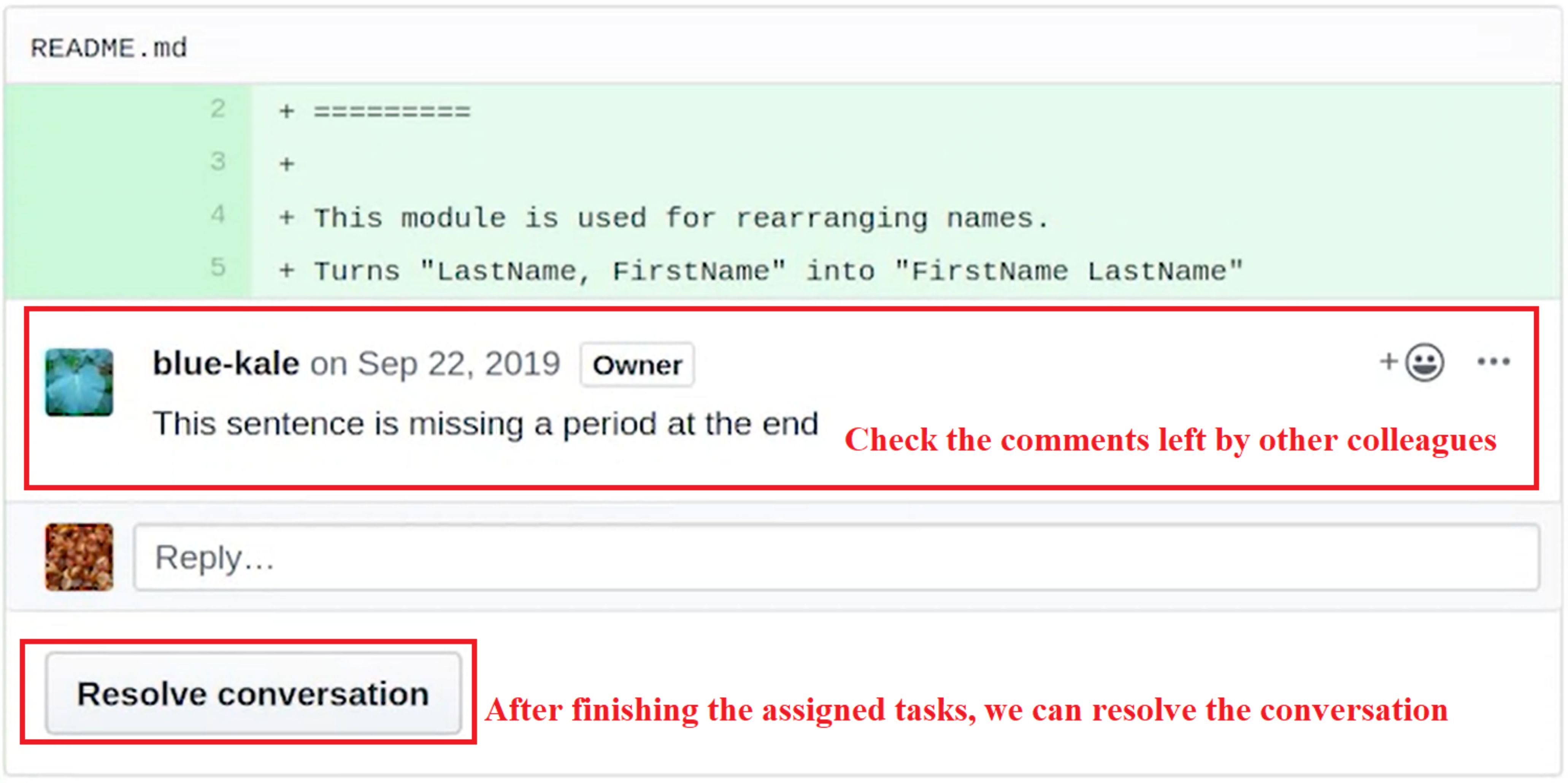
2 commits are squashed into a single commit

2. Code Reviews

(1) Code review

- **Code review**
  - Going through someone else's code/documentation/configuration and checking that it all makes sense and follows the expected patterns
- **Function of code review**
  - Improve the project by making sure:
    - the changes are in high quality
    - the contents are easy to understand
    - the style is consistent with the overall project
    - important cases are not forgotten
    - allows as many people as it can to review the code
- **General issues for code review to address**
  - Using unclear variable names
  - Forgetting to handle a specific condition
  - Forgetting to add tests
  - Making typos or syntax error in the code

(2) Code review in GitHub: working on others' review


README.md

```
2  + =========
3  +
4  + This module is used for rearranging names.
5  + Turns "LastName, FirstName" into "FirstName LastName"
```

blue-kale on Sep 22, 2019 Owner

This sentence is missing a period at the end

Check the comments left by other colleagues
Reply…
Resolve conversation
After finishing the assigned tasks, we can resolve the conversation

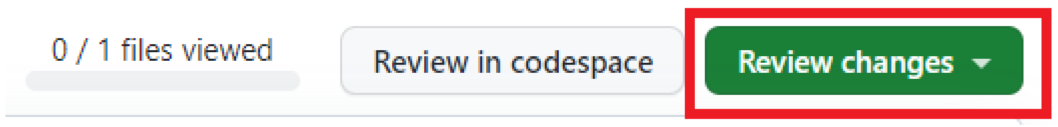
A code review left by other users

```
git commit -a --amend
```

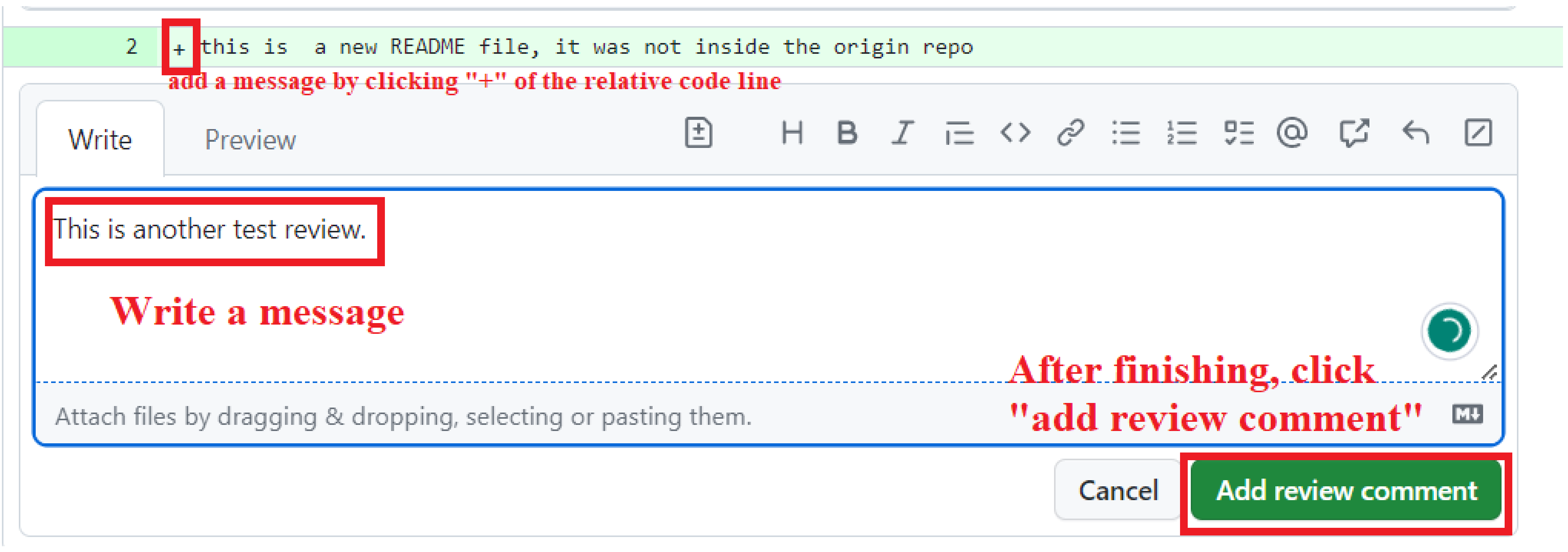
Function: amends the previous *commit* instead of creating a new one

- Caution should be taken on the *commits* that has been pushed to the *remote repository*

(3) Code review in GitHub: initiate a new review



Create a code review (p1)

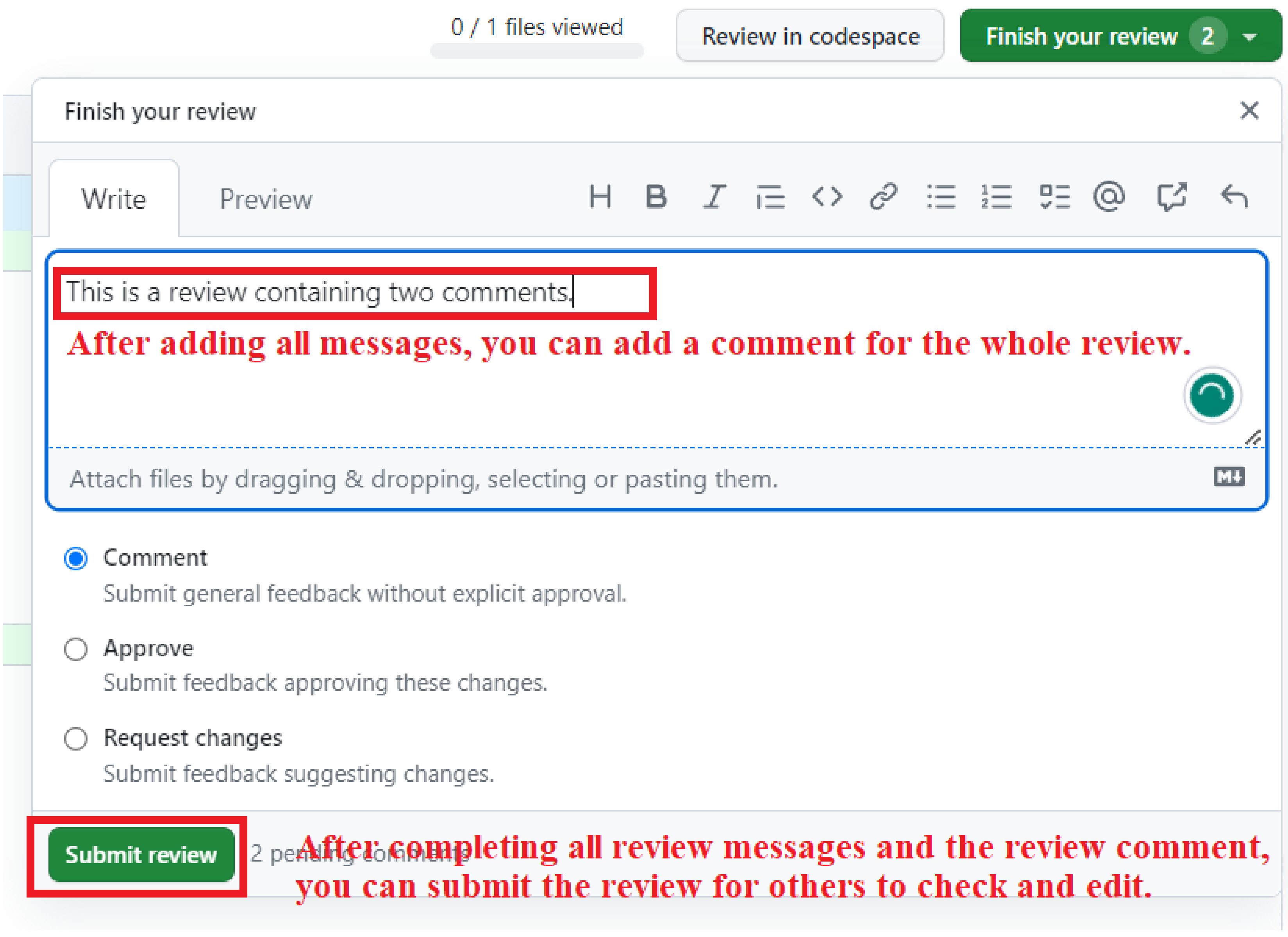


```
2  + this is  a new README file, it was not inside the origin repo
```

This is another test review.

add a message by clicking "+" of the relative code line
Write
Preview
Write a message
Attach files by dragging & dropping, selecting or pasting them.
After finishing, click "add review comment"
Cancel
Add review comment

Create a code review (p2)


0 / 1 files viewed

Review in codespace

Finish your review 2

Finish your review

Write Preview

This is a review containing two comments.

After adding all messages, you can add a comment for the whole review.

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

- [✓ Comment] Submit general feedback without explicit approval.
- Approve: Submit feedback approving these changes.
- Request changes: Submit feedback suggesting changes.

Submit review 2 pending comments

After completing all review messages and the review comment, you can submit the review for others to check and edit.

Create a code review (p3)

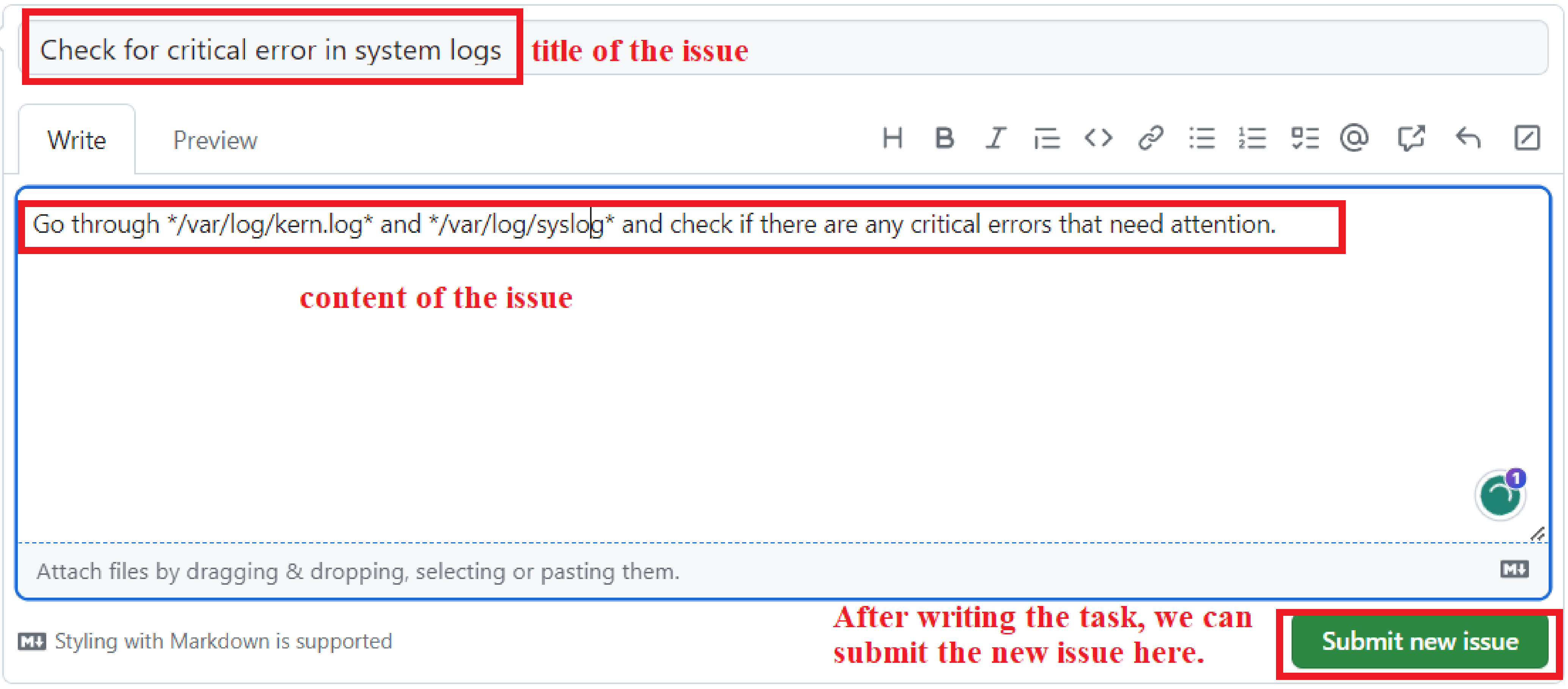
3. Manage Projects

(1) Managing collaboration

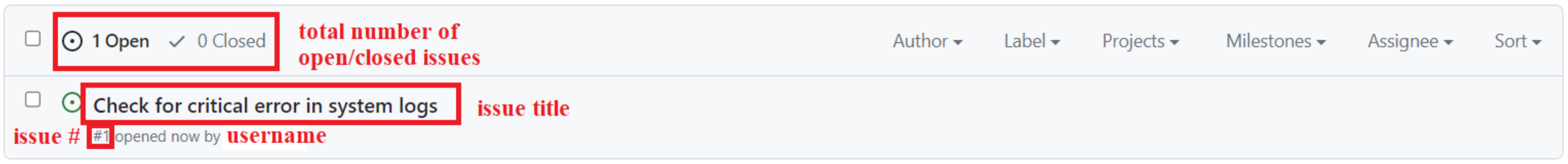
- **Notes for collaboration management**
  - Replying the *pull requests* promptly can reduce the chance of future conflict when a new *commit* raises.
  - It is important to understand every change you accept.
  - Make sure the maintenance follows a fixed style guideline.
  - When it comes to coordinating who does what and when, a common strategy for active software projects is to use an **issue tracker**.
  - It is important to have an efficient communication tool for coordinators when the project is very large.

(2) Tracking issues

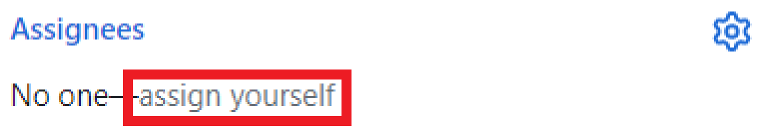
- **Issue tracker**
  - Tells us the tasks that need to be done, the state they are in, and who is working on them
  - Allows us to add comments to the issues


Check for critical error in system logs
title of the issue
Write
Preview
Go through */var/log/kern.log* and */var/log/syslog* and check if there are any critical errors that need attention.
content of the issue
Attach files by dragging & dropping, selecting or pasting them.
Styling with Markdown is supported
After writing the task, we can submit the new issue here.
Submit new issue

create a new issue

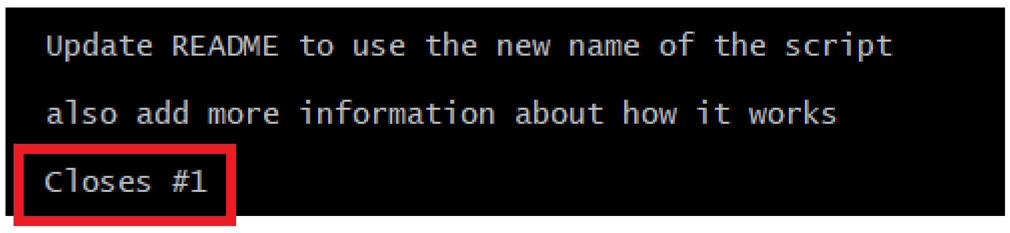

1 Open
0 Closed
total number of open/closed issues
Author
Label
Projects
Milestones
Assignee
Sort
Check for critical error in system logs
issue title
issue # #1 opened now by username

issues shown in the repository

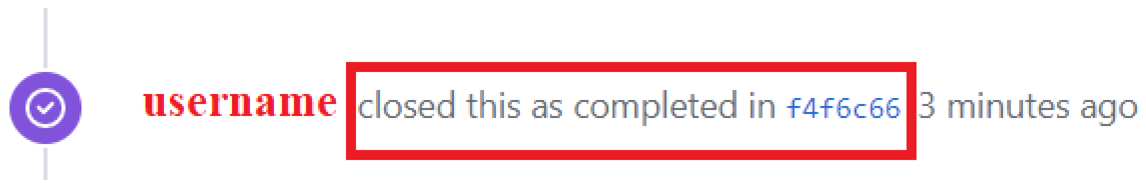


assign someone to work on the issue

- **Note**: Adding a "Closes #[number of issue]" will close the *issue* we are working on after pushing the *commit* to the *remote repository*



```
Update README to use the new name of the script
also add more information about how it works
Closes #1
```



Issue #1 is closed after typing "Closes #1" in the commit message

(3) Continuous integration

- **Continuous integration (CI)**
  - A system that builds and tests our code every time there is a change
- **Continuous deployment/delivery (CD)**
  - Deploying the new code often, i.e., with frequent incremental updates with a few changes each time

    - Allows errors to be caught and fixed early
- **CI/CD Platform Example**: Jenkins, GitHub Actions, Travis
- **Concepts to create CI/CD**
    - **Pipelines**: specify the steps that need to run to get the result you want
    - **Artifacts**: the name used to describe any files that are generated as part of the *pipeline*
- **Secret management**
    - Make sure the *authorized entities* for the *test servers* are not the same *entities authorized* to deploy on the *product servers*
    - Always have a plan to recover your access in case your *pipeline* gets compromised