GitHub introduction
for team project

Integrated Exercise for Software I & II

Introduction

Learning Material for GitHub

- Official resource page
  [https://help.github.com/articles/git-and-github-learning-resources/]

- Overview Guides [https://guides.github.com/]

- Hello World
  Hands-on tutorial to create repository, create branch, perform commits, and create pull-requests

- GitHub Flow
  Explanation of “branch-commit-pull requests-merge” workflow

- Git Handbook
  Concise overview of git commands

Learning Material and References for git

- Official resources to learn about git and version control

- ProGit ebook
  - https://git-scm.com/book/en/v2
    [Japanese translation is of low quality]

- Command list
  - https://git-scm.com/docs

Git - Config

After git command properly installed, link your GitHub account with your local git environment by using “git config” command.

$ git config --global user.email “you@example.com”

$ git config --global user.name “Your name”

Example:

In the case of an mail address registered to GitHub is: “m5211143@u-aizu.ac.jp” and

Github ID is: ”m5211143-Saito” , then you can run:

```
m5211143[71]$ git config --global user.email "m5211143@u-aizu.ac.jp"
m5211143[72]$ git config --global user.name "m5211143-Saito"
```

Also you can see registered information by using “--list” option after above process.

$ git config --list

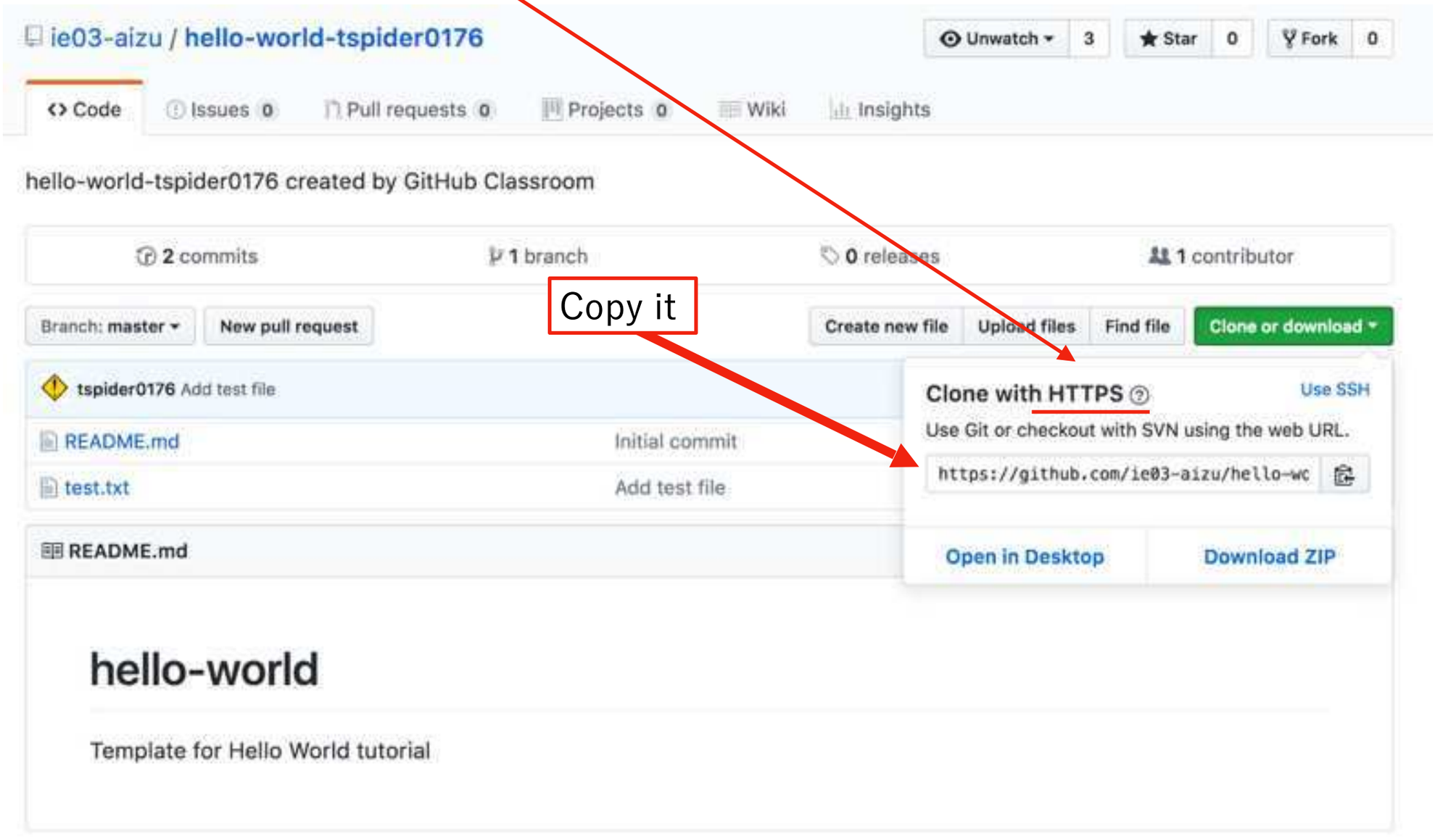
If you input wrong information for them, you can simply re-run above “git --config” command and fix them.

Basic commands for git

Git - Clone

When you contribute your team project, first you must clone team project to your local environment as a local repository from remote repository. In this time, hello-world project will be used for example.

You can see URL for cloning repository on the page shown below. Make sure to display “Clone with HTTPS” on appeared window.



Git - Clone

Move to appropriate directory and clone repository. To clone already existed remote repository to your local environment, use below command.

$ git clone <repository_url> <local_path>

Input URL for cloning for <repository_url>.

Input your local directory name for <local_path>.

You can omit <local_path>, and then “git” will make directory which is naming same as remote repository on current directory. Make sure there is no directory which has same name with remote repository in current directory. If so, it would fail to cloning repository.

When you clone remote repository, you must input your GitHub name and password every time even if you after executing “git config” command due to using HTTPS. By using SSH connection, you need not to input any information every time.

If you want to use SSH connection, you have to register SSH public key to GitHub account. Detail information for it is in last of this presentation.

Git - add, commit, push commands

To record (commit) your file changes to remote repository, you must assign these files as a tracked file of “git” command (in other words, record the changes to staging area), so that “git” recognizes what file should be uploaded next time.

Create some test file on cloned repository and push to local repository.

$ touch <user_name>-testfile

$ git add <user_name>-testfile

$ git commit -m “Add test file by <user_name>”

$ git push origin master

These [add -> commit -> push] flow is important process when you want to apply your changes to remote repository.

Each commands briefly described below:

- add -> Record your changes on local repository to git (add file changes to tracked file list).
- commit -> Record staging files with a short message which indicates what you have done.
- push -> Upload your changes to remote repository.

Git - add

By using “git add” command, we can add files or directories to next commit. This command tell Git that assigned files or directories by “git add” commands should be observed.

If you want to remove your files or directories from staging area, you can use “git rm” command opposite of “git add” command.

$ git add <user_name>-testfile

$ git add . # You can use regular expression after “git add”

“git add” and “git rm” command can handle files like:

- Newly added files at working directory (**untracked file**)
- Files with several changes from previous commit (**changed file**)
- Deleted (to be deleted) or untracked files from Git (**deleted file**)

Git - commit

“git commit” command record files in staging area which is created by “git add” command. One unique ID is assigned to one commit, and then we can manage these commits by using them.

$ git commit -m “commit message for changes”

$ git commit

If you didn’t any option like “-m” after “git commit”, default text editor will executed like Emacs or Vim. Same as using option “-m”, you can type commit message about your changes.

Generally, team member **must write** commit message which provides **detail explanation about your changes** to files. Then the other team member easily know what you have done from your commits.

These commit message should be formatted between team members like:

```
[WIP] implementing function …
[Add] new function which manages…
[Modify] function for some operation on file …
[Fix] bugs detecting on one files…
```

Git - push

After “git add” and “git commit” command, you must upload your changes to remote repository.

By using “git push” command, you can upload your changes which assigned by “git commit” command to remote repository.

$ git push origin <branch_name>

On development by some team member, it is rare to use master branch to avoid conflicts on files in remote repository. There are some possibility to occur conflicts and bugs on some files on remote repository. Imagine different team member changes same file and upload their changes to remote repository at same time. “git” command cannot recognize which commit is right.

In general, team member should be create their own branch at their local repository, and then they push their changes to same branch on remote repository separated from master branch. It is explained in detail on next chapter.

Reference: https://romtin.gitbooks.io/gittutorial-for-sccp2016/content/04/04_1.html (in Japanese)

Managing branch

Git - Managing Branches

The commits in one repository is represents tree-like structure start with one root commit (first commit) to newest commit. Commit log is separated by creating new branch, and it is unified by merging branch.

Raise one example for advantages to use branch on development with several members.

As shown commit log, implementation of the base system has done, and they consider about next implementation for adding some new function to their system.

```
* 9fe47fb (HEAD, master) feat: add extension
* 15643db chore: add README
* 108c3e4 feat: implement base program
```

“master” is a branch like trunk of a tree which is automatically created when you created a repository on GitHub, and “HEAD” indicates newest commit.

In this case, same as previous development, they start to implement new function only on master branch. Then, imagine some bugs are detected after kick-start and they implement multiple function in same time on master branch.

Git - Managing Branches

If they worked on same branch, master branch in this time, the commits from team member on master branch will be like:

```
* 7fbef9a (HEAD, master) bugfix: catch exception in main
* 2195b01 fix: syntax error for issue1
* f889005 feat: implement Model for issue2
* 5cb0ae2 feat: implement Controller for issue2
* 47421a7 feat: implement Controller for issue1
* 9fe47fb feat: add extension
* 15643db chore: add README
* 108c3e4 feat: implement base program
```

This is very complicated due to confusion of commits for implementing some function and fixing bugs. If they use branch flow, it can be know easily where bug fixed, new function progress.

```
*   d7842d1 (HEAD, master) Merge branch 'fix/exception_main'
|\
| * cd051e5 (fix/exception_main) bugfix: catch exception in main
* |   9a021a3 Merge branch 'feat/issue2'
|\ \
| * | 35a961d (feat/issue2) issue2: feat: implement Model
| * | 82be355 issue2: feat: implement Controller
| |/
* | ae31340 (feat/issue1) issue1: feat: implement Controller
|/
* 9fe47fb feat: add extension
* 15643db chore: add README
* 108c3e4 feat: implement base program
```

Git - Managing Branches

“git branch” command is to create new branch on your local repository.

$ git branch <new_branch>

$ git branch

When you use “git branch” command with no option, you can see already exists branches. If you firstly use this command after you created the repository, only “*master” will be outputted to terminal.

You can change branch by using command:

$ git checkout <branch_name>

Also, if you change files on new branch which you created, and you want to upload your changes to remote repository, then you below command:

$ git push origin <new_branch_name>

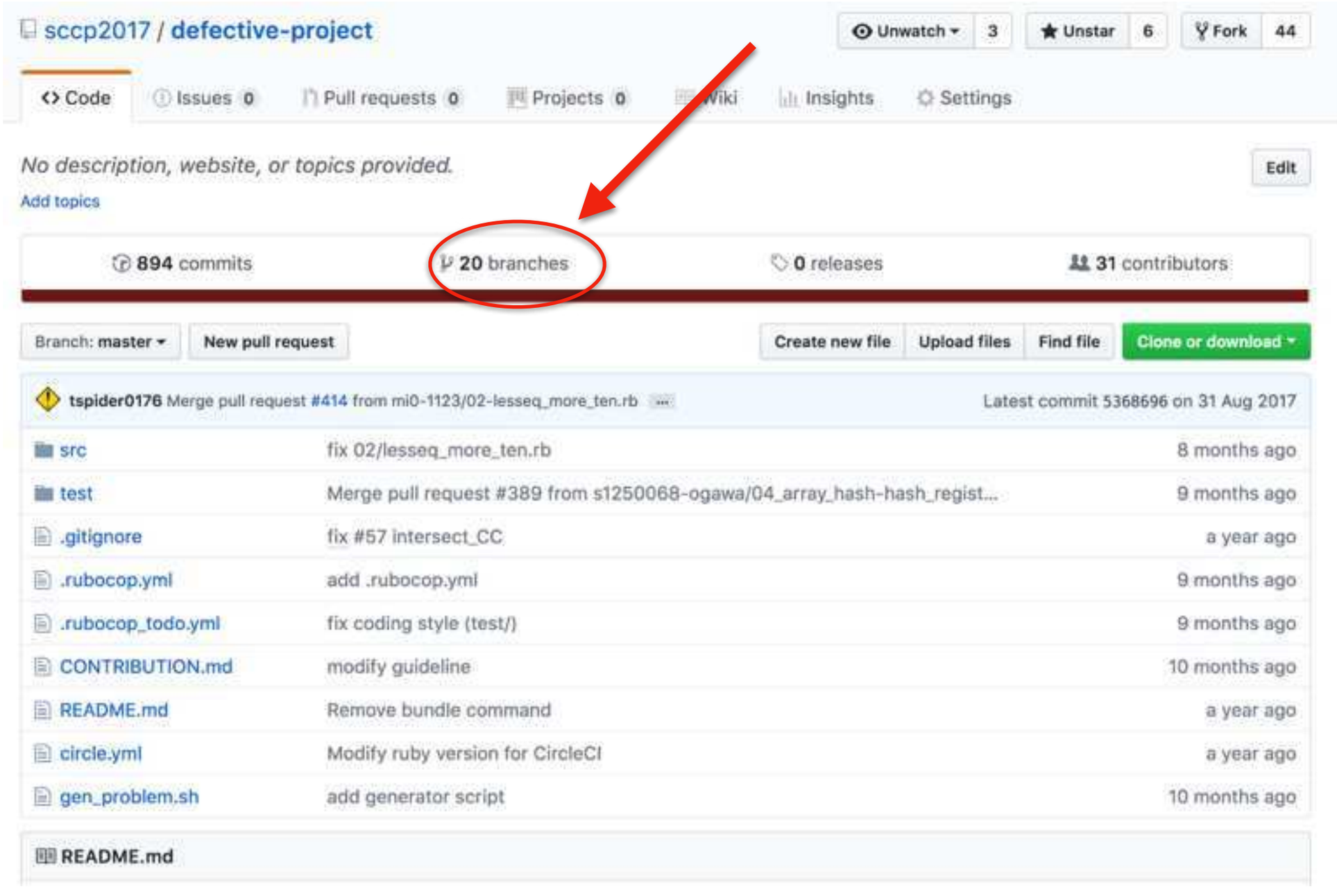
Make sure commit your changes before move branch.

Pull Request (PR)

Git - Pull Request (PR)

If you have finished implementing your duty, and also you have done pushed your changes to remote repository, you need to create **Pull Request** (**PR**) to apply your changes to master branch of remote repository.

You can find your branch on remote repository page if you appropriately pushed.



Git - Pull Request (PR)

Below command lines indicates the flow of creating new branch, apply changes to file, and push these changes to the new branch on remote repository.

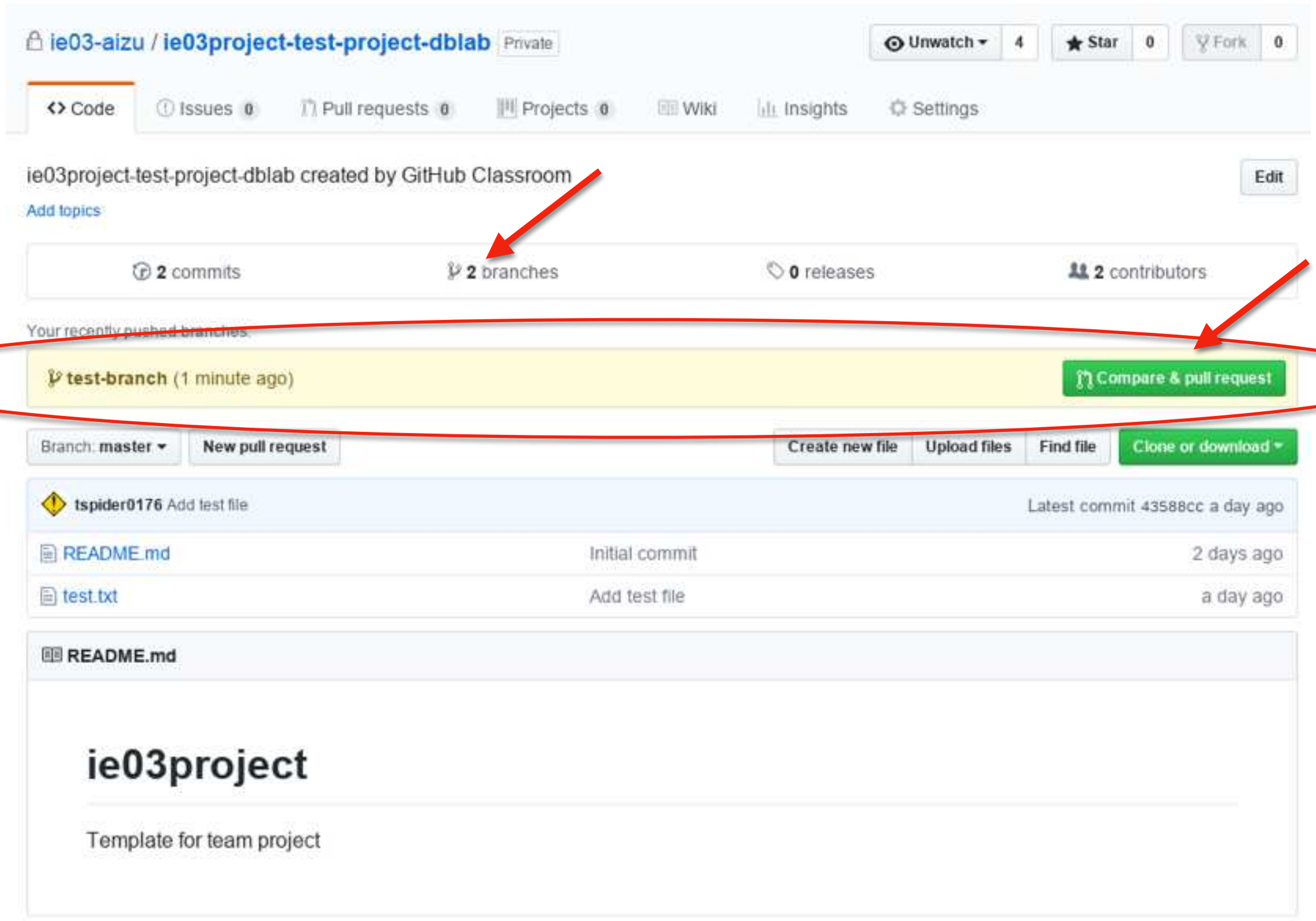
```
lsrdc9:ie03project-test-project-dblab m5211143[60]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
lsrdc9:ie03project-test-project-dblab m5211143[61]$ git branch test-branch
lsrdc9:ie03project-test-project-dblab m5211143[62]$ git checkout test-branch
Switched to branch 'test-branch'
lsrdc9:ie03project-test-project-dblab m5211143[63]$ git branch
  master
* test-branch
lsrdc9:ie03project-test-project-dblab m5211143[64]$ touch new-file
lsrdc9:ie03project-test-project-dblab m5211143[65]$ git add .
lsrdc9:ie03project-test-project-dblab m5211143[66]$ git commit -m "Add new file"
[test-branch 4432d1c] Add new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new-file
lsrdc9:ie03project-test-project-dblab m5211143[67]$ git push origin test-branch
Username for 'https://github.com': tspider0176
Password for 'https://tspider0176@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 283 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://github.com/ie03-aizu/ie03project-test-project-dblab.git
 * [new branch]      test-branch -> test-branch
lsrdc9:ie03project-test-project-dblab m5211143[68]$
```

Create branch

Apply change

Git - Pull Request (PR)

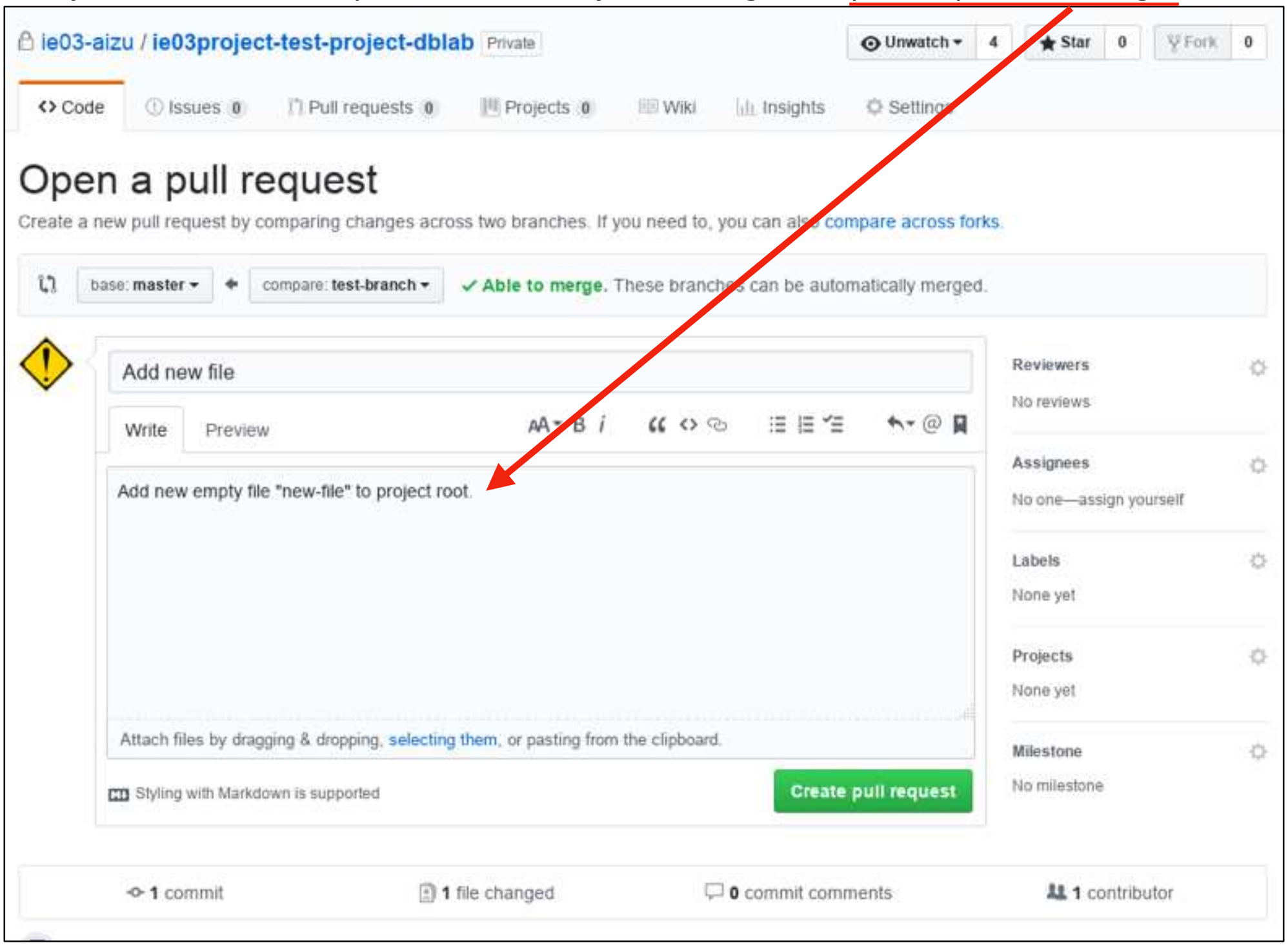
After push changes on new branch, you can check your branch on GitHub. The web-page may changes as shown below. If you want to create pull request, click Compare & pull request button.



Git - Pull Request (PR)

After click Compare & pull request button, you can open pull request to master branch.

Make sure you wrote detail explanation about your changes on pull request message.



Git - Pull Request (PR)

If you want to delete your local branch, use:

$ git branch -d <branch_name>

By using this command, local branch will be deleted.

If you have not commit and not apply your change master branch, git tell us there are some changes which is not applied to master branch.

Other related commands:

$ git checkout -b <branch_name> # git branch <branch_name> & git checkout <branch_name>

$ git branch -r # See branches on remote repository

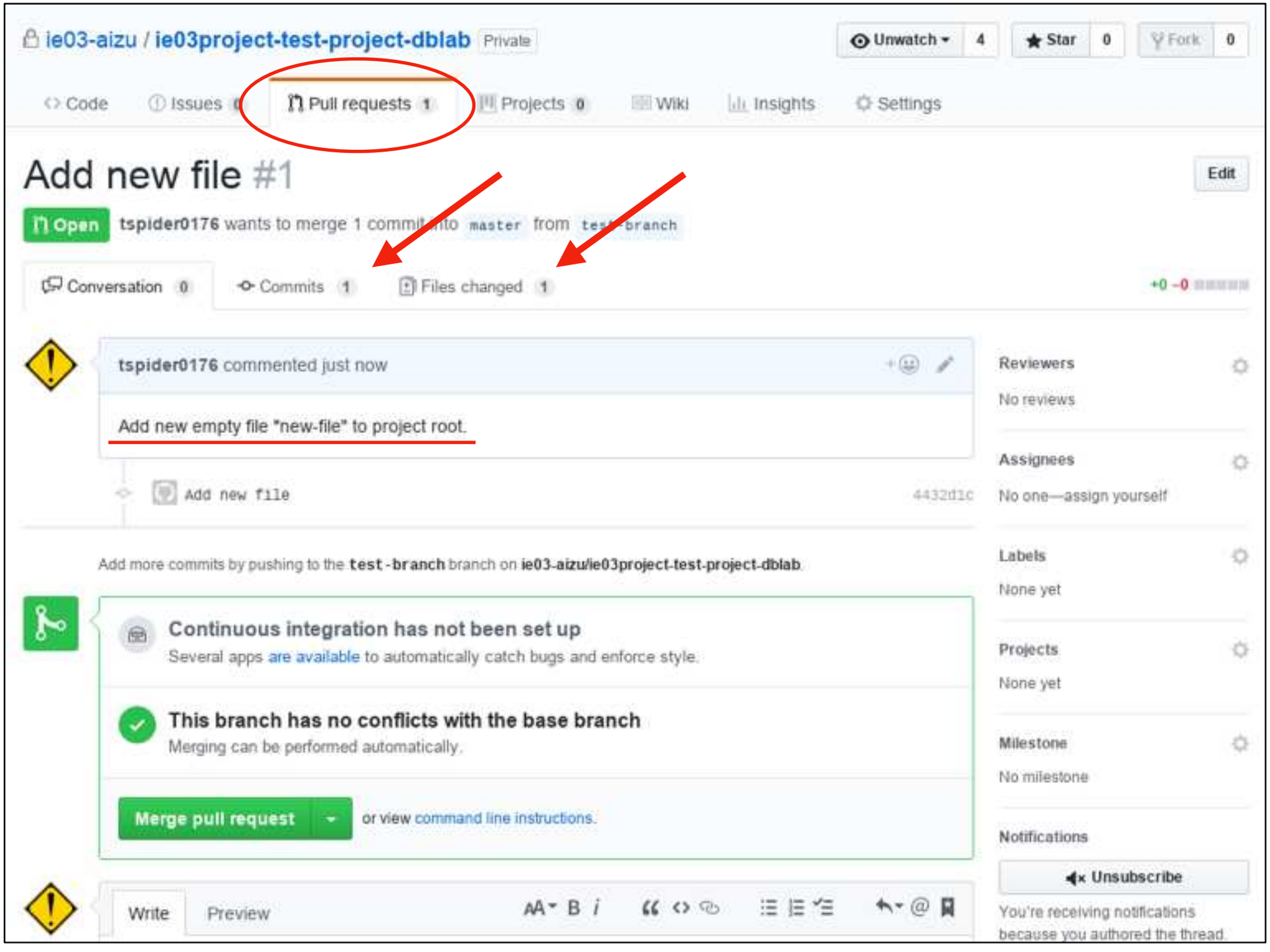
$ git branch -D # Delete local branch forcedly

$ git checkout <commit_name> # Move commit

Merge

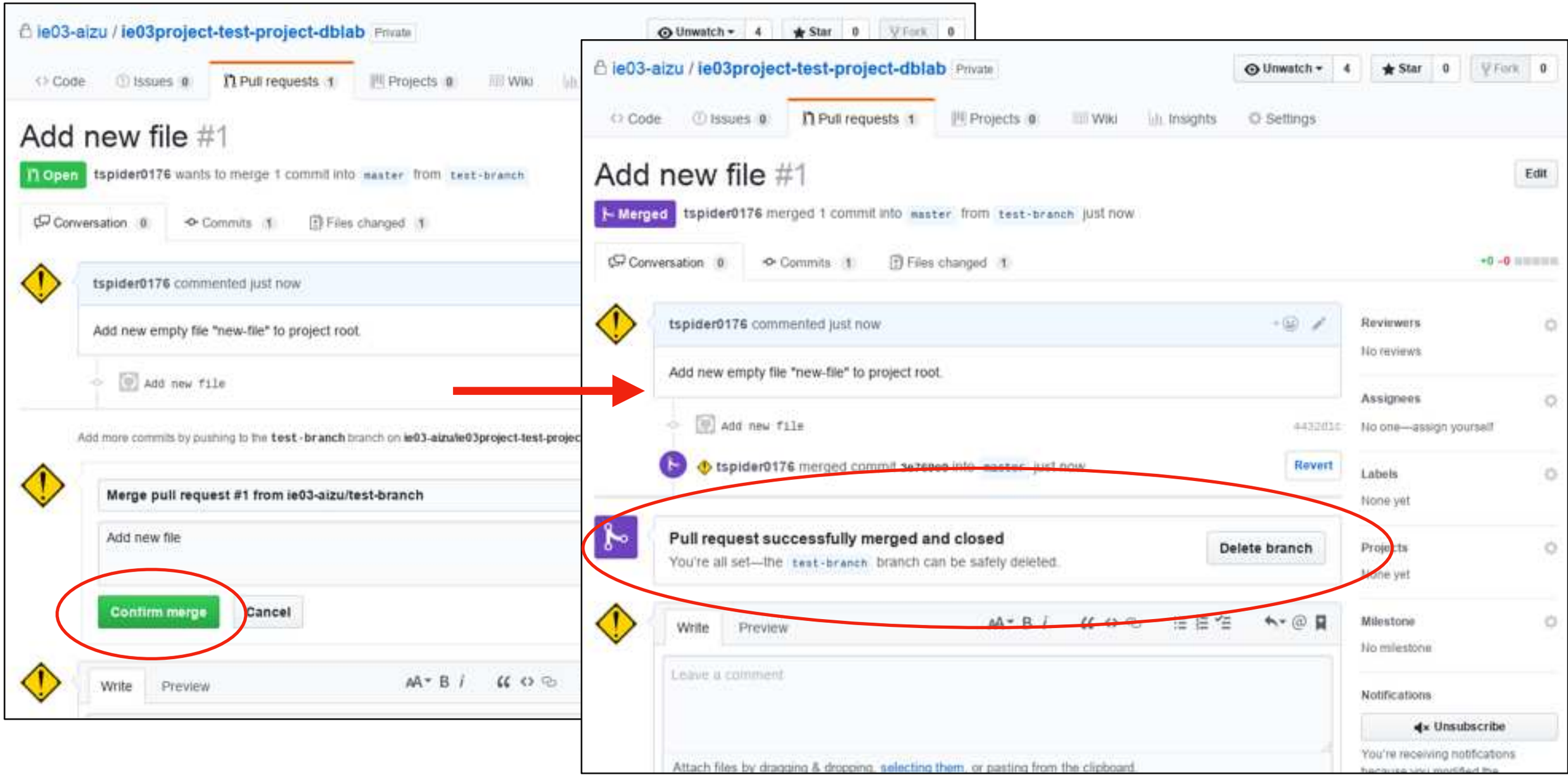
Git - Merge

After pull request created, team leader can check commits, file changed, and pull request messages in pull request tab of remote repository.



Git - Merge

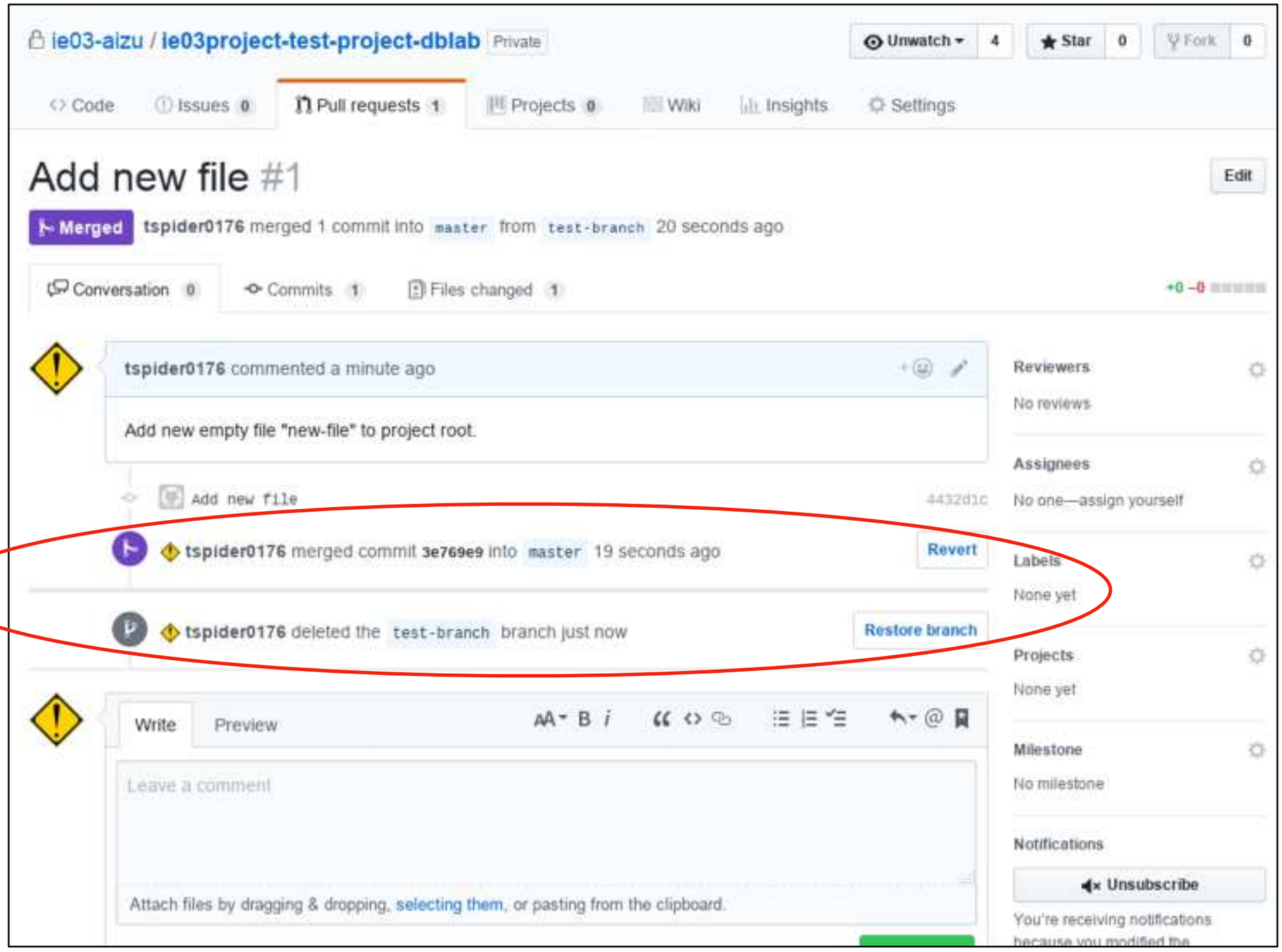
If it is determined that the pull request should be apply to master branch, then click “Merge pull request”, and then “Confirm merge”.



If merge process has done successfully, “Pull request successfully merged and closed” will be displayed.

Git - Merge

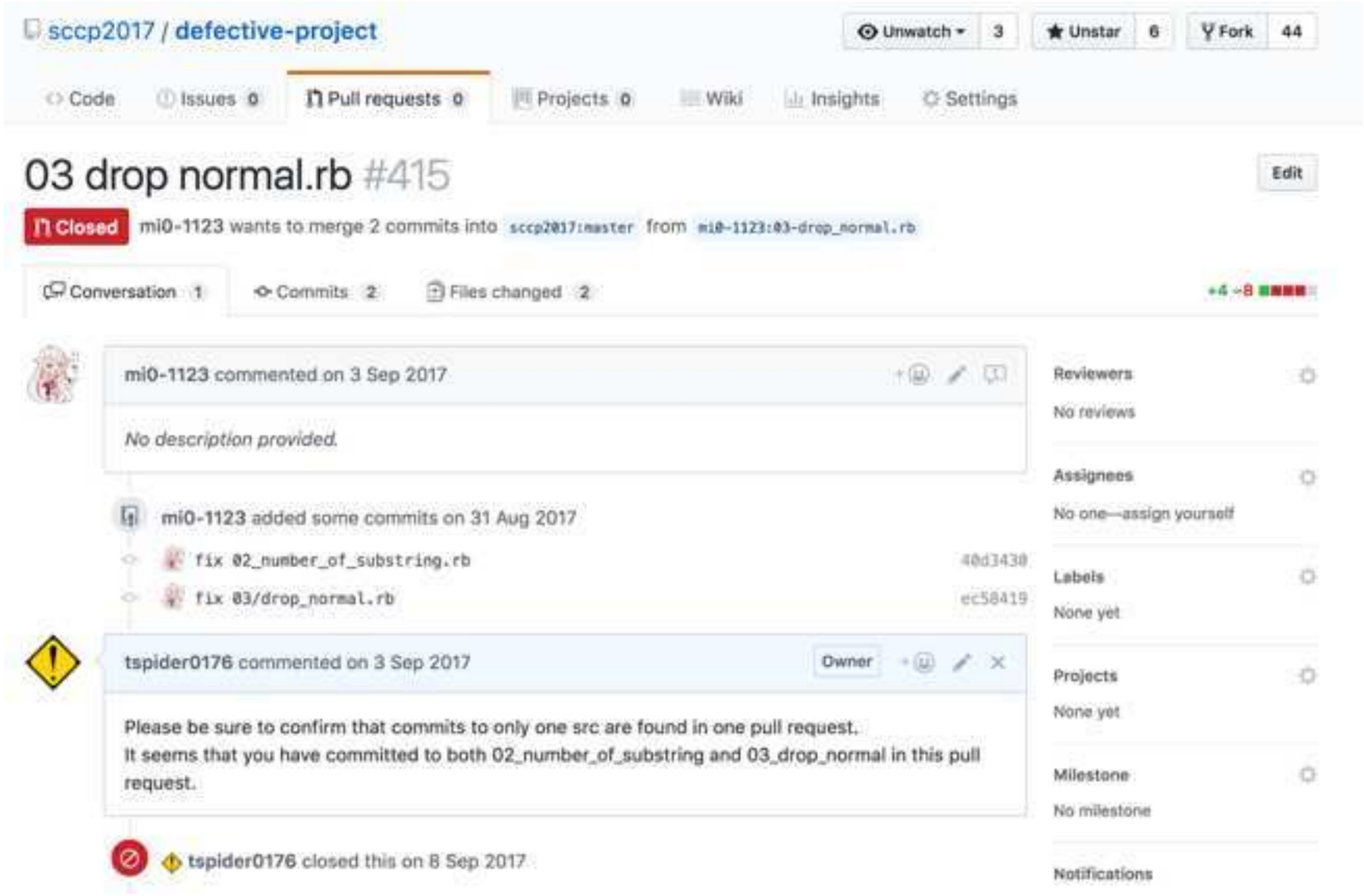
After merge pull requests and if never use merged branch, you can remove the branch from remote repository. Even if it is removed branch in remote repository, committer can see his/her created branch in local repository.



Git - Merge

In this time, “Opening pull requests” and “Merging pull requests” process has done by same person as you can see the user on previous page. This is only example of these process, and of course, pull requests must be checked the other team member (e.g. team leader).

Check pull request before merging to master branch of the remote repository and leave some comments about it. DO NOT merge pull requests without any considering. It may occur conflicts.



Git - Merge

After branch has merged master branch of remote repository, team member should update your local repository according to master branch.

Below command updates your local repository.

$ git pull origin <branch_name>

If you want to update your master branch of local repository, then:

$ git pull origin master

```
lsrdc9:ie03project-test-project-dblab m5211143[69]$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
lsrdc9:ie03project-test-project-dblab m5211143[70]$ git pull origin master
Username for 'https://github.com': tspider0176
Password for 'https://tspider0176@github.com':
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/ie03-aizu/ie03project-test-project-dblab
 * branch            master     -> FETCH_HEAD
   43588cc..3e769e9  master     -> origin/master
Updating 43588cc..3e769e9
Fast-forward
 new-file | Bin
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new-file
lsrdc9:ie03project-test-project-dblab m5211143[71]$
```

Move master branch

Pull remote repository updates

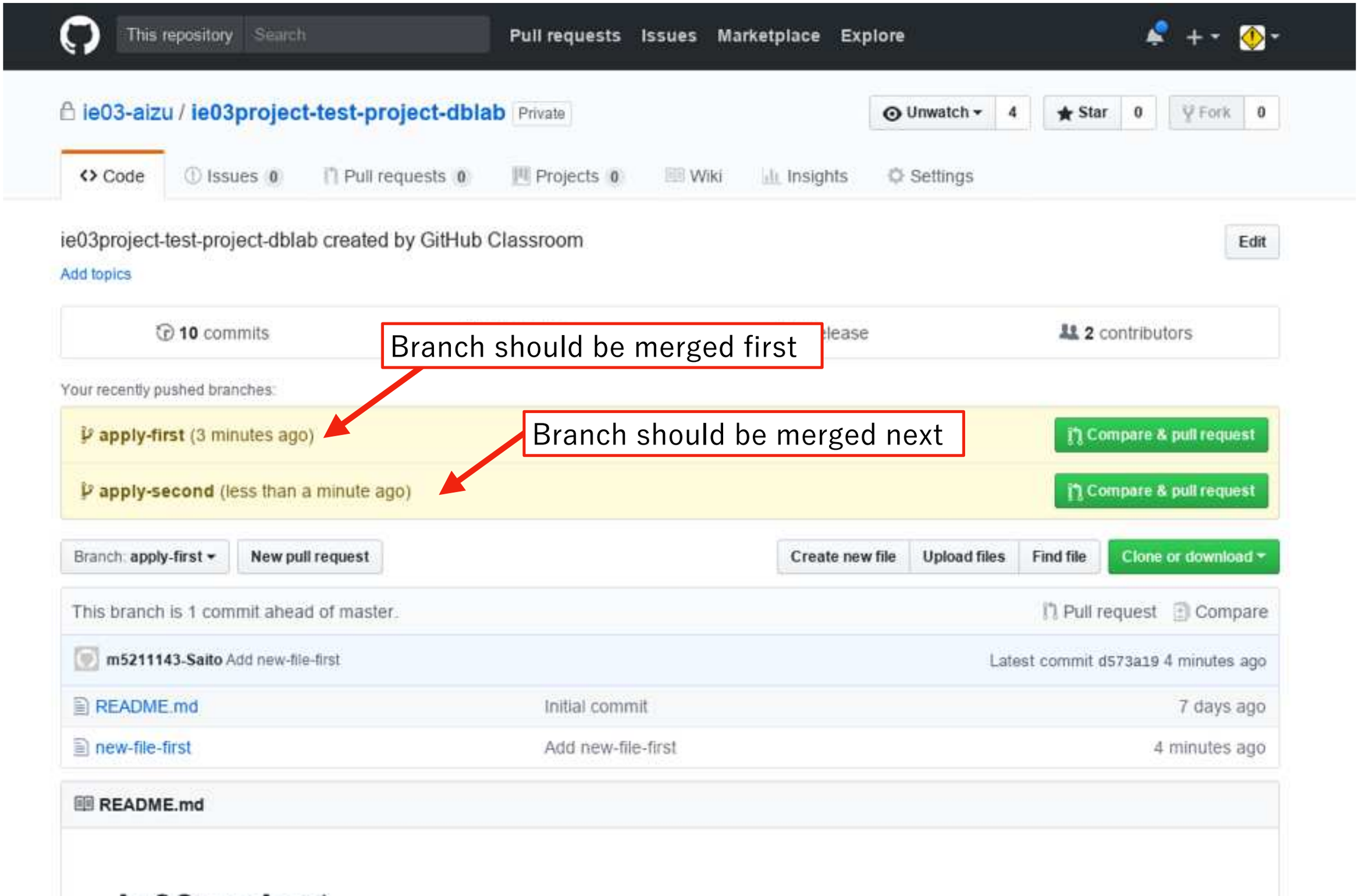
You can check updates here.

Whole process

Create branch -> PR -> Merge

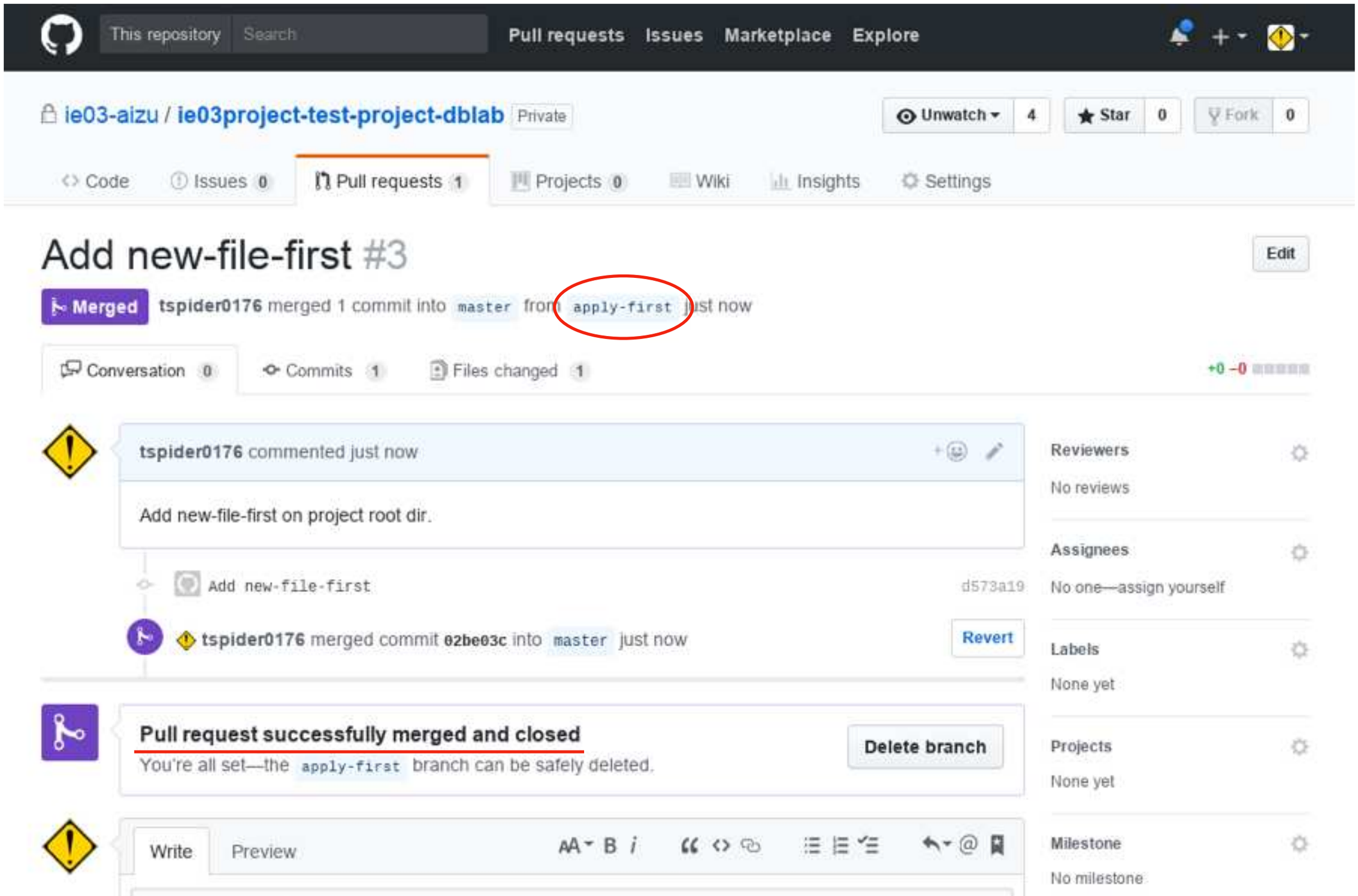
Git - Merge

Raise example project flow according to below sample project.



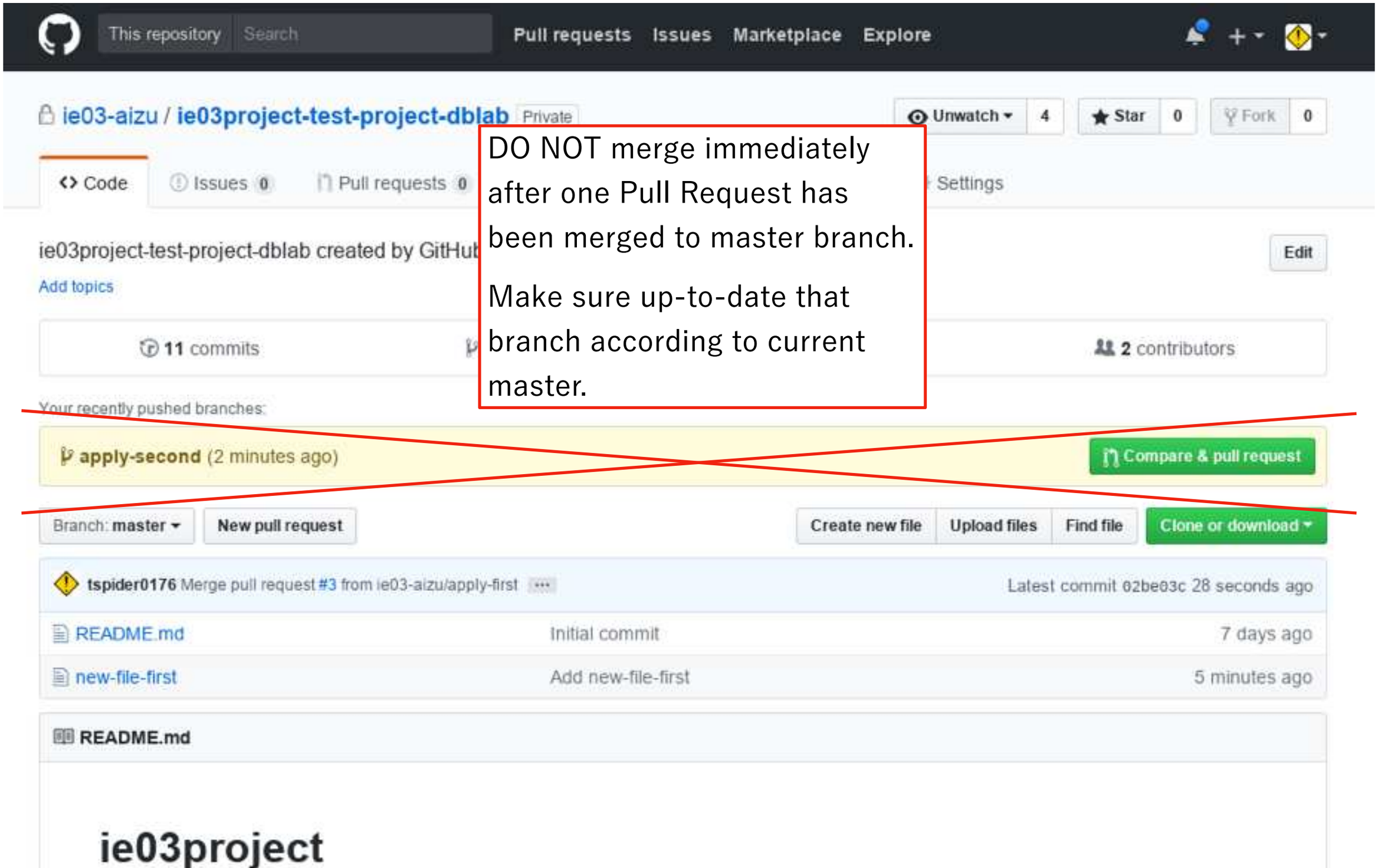
Git - Merge

Merge first pull request from branch “apply-first” same as process indicated in Pull Request section of this presentation.



Git - Merge

Before merge “apply-second” branch, the committer who make the branch should apply changes on master branch on remote repository to his/her local branches.



Git - Merge

After one pull request has been merged to master branch on remote repository, the other team member must update their local branch and also their development branch.

One Pull Request has been merged…

⋮

Team member must do:

```
lsrdc7:ie03project-test-project-dblab m5211143[68]$ git branch
  apply-first
  apply-second
* master
lsrdc7:ie03project-test-project-dblab m5211143[69]$ git pull origin master
Username for 'https://github.com': tspider0176
Password for 'https://tspider0176@github.com':
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/ie03-aizu/ie03project-test-project-dblab
 * branch            master     -> FETCH_HEAD
   f91f26e..02be03c  master     -> origin/master
Updating f91f26e..02be03c
Fast-forward
 new-file-first | Bin
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new-file-first
lsrdc7:ie03project-test-project-dblab m5211143[70]$
```

①Move master branch

②Update local master branch

③Check updates

Git - Merge

After one pull request has been merged to master branch on remote repository, the other team member must update their local branch and also their development branch.

```
lsrdc7:ie03project-test-project-dblab m5211143[70]$ git checkout apply-second
Switched to branch 'apply-second'
lsrdc7:ie03project-test-project-dblab m5211143[71]$ git merge master
Merge made by the 'recursive' strategy.
 new-file-first | Bin
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new-file-first
lsrdc7:ie03project-test-project-dblab m5211143[72]$ git push origin apply-second
Username for 'https://github.com': tspider0176
Password for 'https://tspider0176@github.com':
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 337 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://github.com/ie03-aizu/ie03project-test-project-dblab.git
   ec2f6f6..0fbd8fe  apply-second -> apply-second
lsrdc7:ie03project-test-project-dblab m5211143[73]$
```

④Move local branch

⑤Apply changes on master to local branch

⑥ Push remote branch

Note: default text editor (Emacs, vim, etc.) has launched and asked merge message when merge master to local branch.

Solving conflicts

Git - Solving conflict

As I mentioned previous slide, there are some possibility to occur conflicts on your project.

The solution of it is, all of conflicted file fix before any commits **by your hand manually, not automatically**.

Of course it is better not to occur conflicts on your project, but you should know how to fix conflict just in case you face to this annoying problem.

In this time, I intentionally make conflicts on a file “new-file” in a test repository for the class. According to this, you may notice how conflicts are occurred.

First of all, make new branch which will occur conflicts between master branch. And modify a file “new-file” to contain a text “test message”. Then push changes to remote repository.

```
lsrdc9:ie03project-test-project-dblab m5211143[72]$ git branch this-branch-may-conflict
lsrdc9:ie03project-test-project-dblab m5211143[73]$ git checkout this-branch-may-conflict
Switched to branch 'this-branch-may-conflict'
lsrdc9:ie03project-test-project-dblab m5211143[74]$ git branch
  master
  test-branch
* this-branch-may-conflict
lsrdc9:ie03project-test-project-dblab m5211143[75]$ echo "test message" >> new-file
lsrdc9:ie03project-test-project-dblab m5211143[76]$ more new-file
test message
lsrdc9:ie03project-test-project-dblab m5211143[77]$
```

Git - Solving conflict

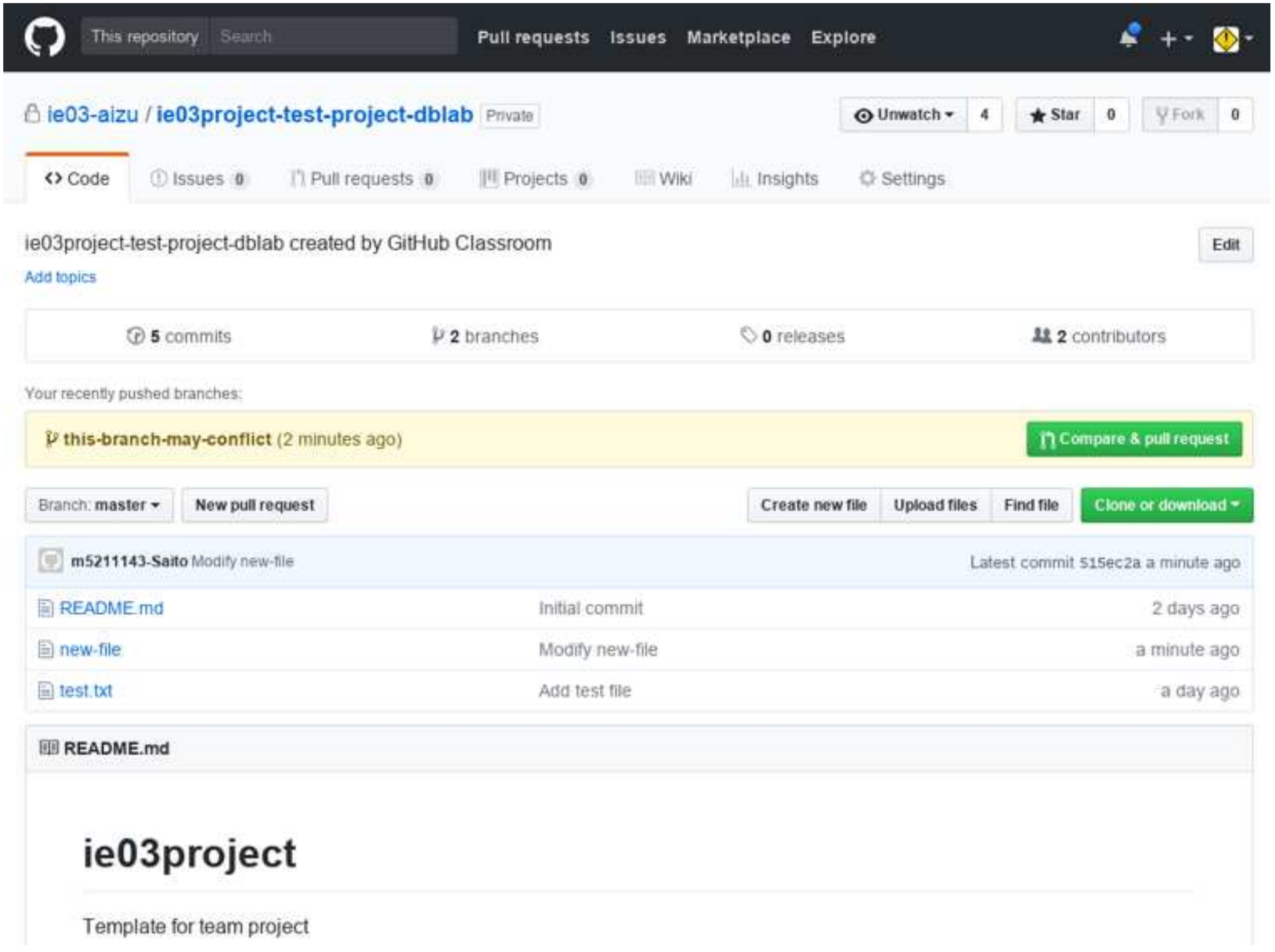
Then, checkout master branch of your local repository, and modify also “new-file” to contain a text “testtesttesttesttest”.

After modifying, push changes to master branch (this is only example processes for occurring conflict on project file, please make sure this act is prohibited on normal development team)

```
lsrdc9:ie03project-test-project-dblab m5211143[93]$ git branch
* master
  test-branch
  this-branch-may-conflict
lsrdc9:ie03project-test-project-dblab m5211143[94]$ echo "testtesttesttesttest" >> new-file
lsrdc9:ie03project-test-project-dblab m5211143[95]$ git add .
lsrdc9:ie03project-test-project-dblab m5211143[96]$ git commit -m "Modify new-file"
[master 515ec2a] Modify new-file
 1 file changed, 0 insertions(+), 0 deletions(-)
lsrdc9:ie03project-test-project-dblab m5211143[97]$ git push origin master
Username for 'https://github.com': tspider0176
Password for 'https://tspider0176@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 324 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ie03-aizu/ie03project-test-project-dblab.git
   3e769e9..515ec2a  master -> master
lsrdc9:ie03project-test-project-dblab m5211143[98]$
```

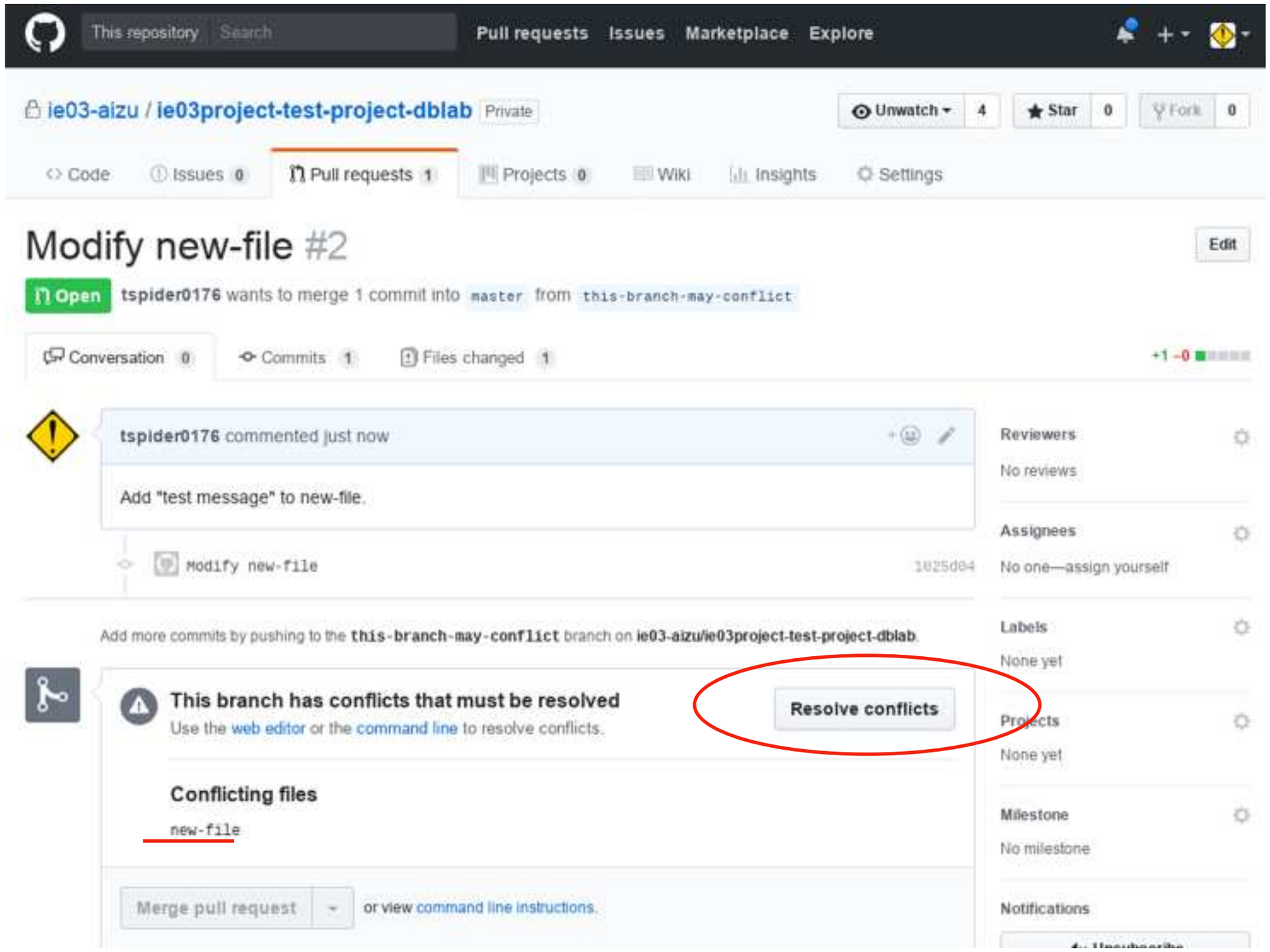
Git - Solving conflict

Then same as previous slides about pull request, merge opened pull request on the remote repository page.



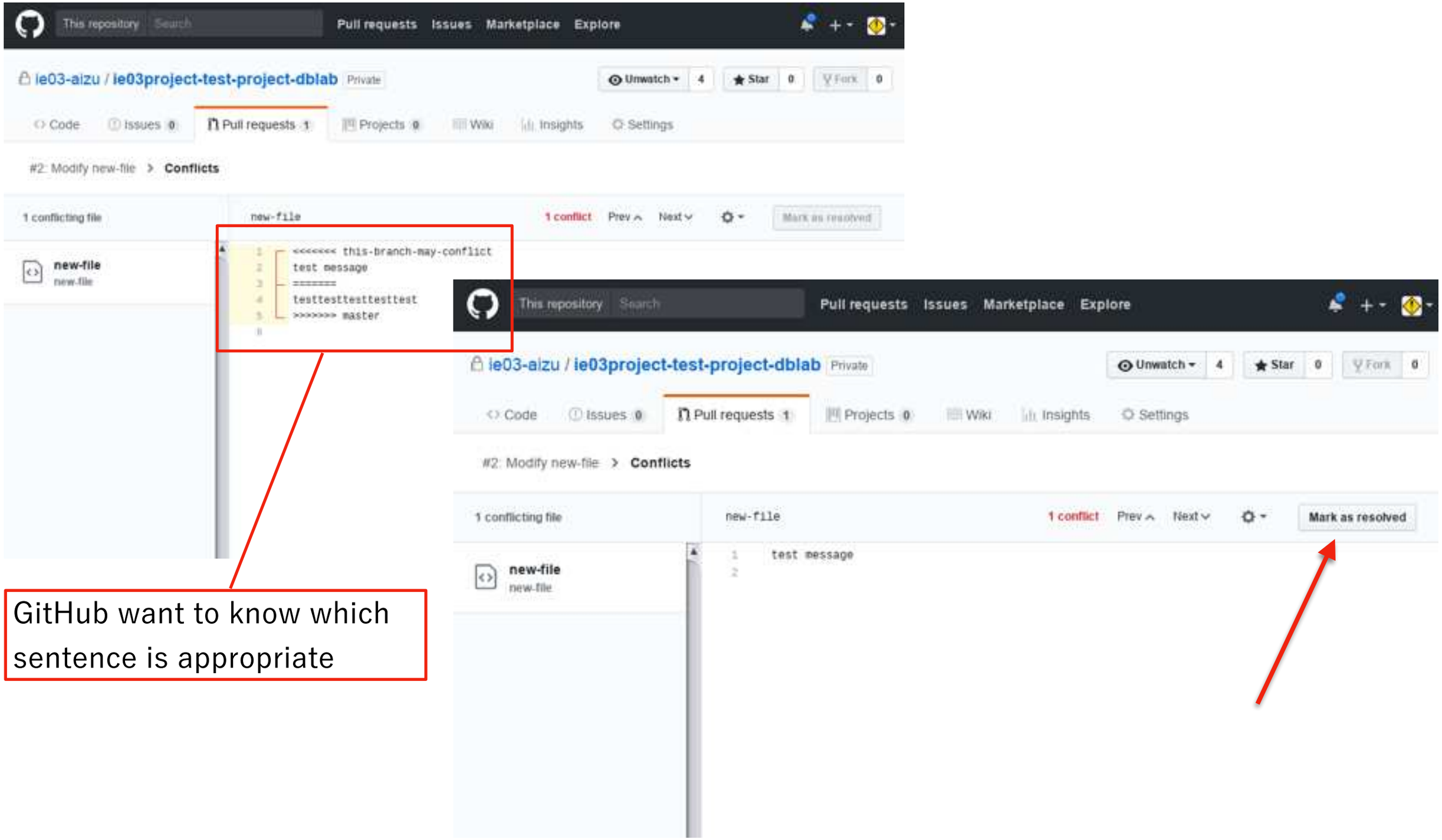
Git - Solving conflict

However, automatically merge function on GitHub doesn't work for this pull request to master branch due to conflict between files in two branch. To resolve conflicts, click button indicated red circle. (or you can resolve conflicts on your command line environment)



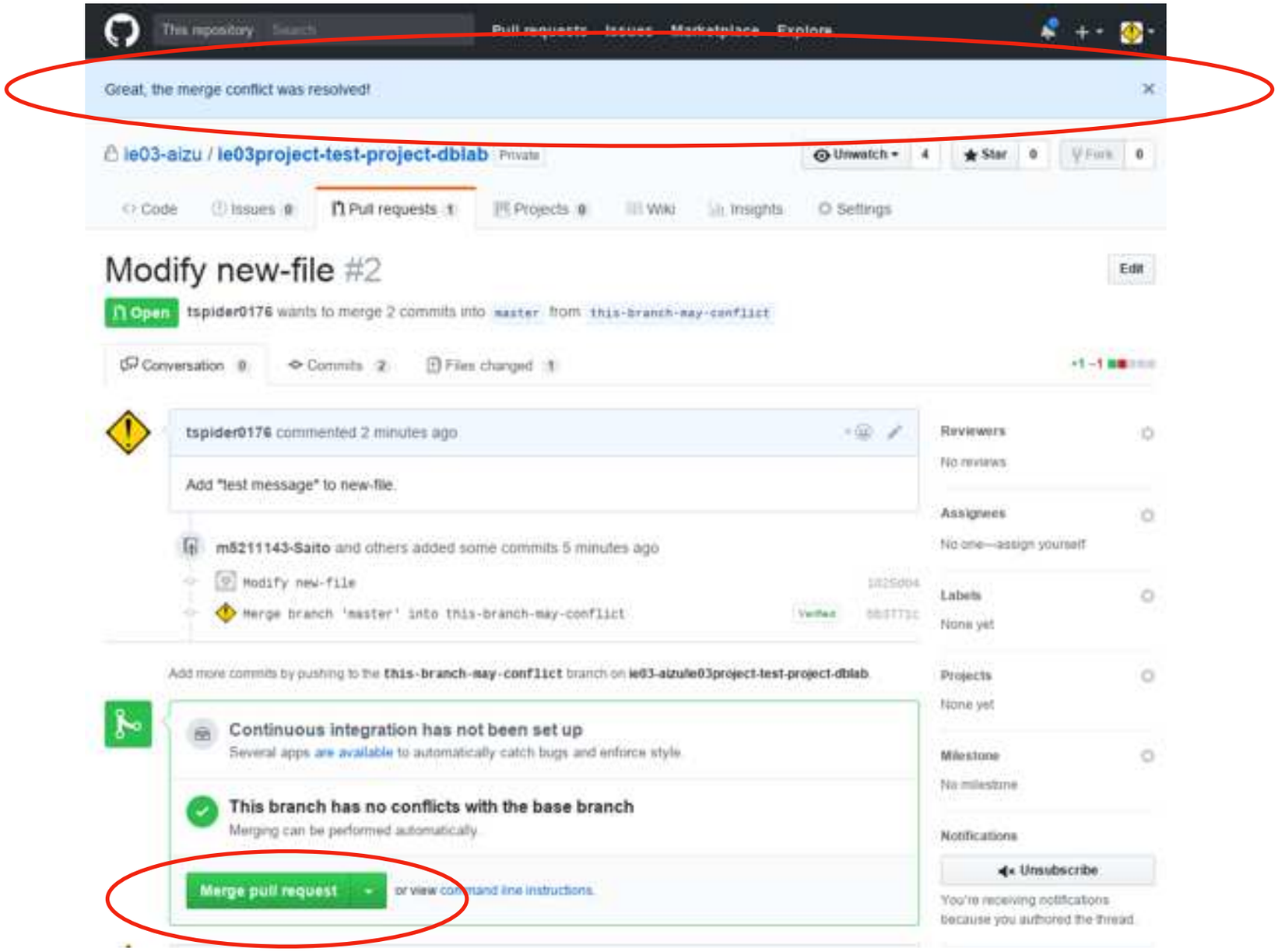
Git - Solving conflict

Then conflicts file and its contents will be displayed. The text indicates difference between conflicted file on two branches. After typing appropriate text in the editor, click “Mark as resolved”.



Git - Solving conflict

After resolving conflicts, you can merge the pull request same as ordinary process.



In this time, only simple sentence is conflicted on one file. If conflicts occurred some programs with too many lines between branches, it is difficult to fix them. Please make sure every time your branch is already up-to-date according to master branch of remote repository.

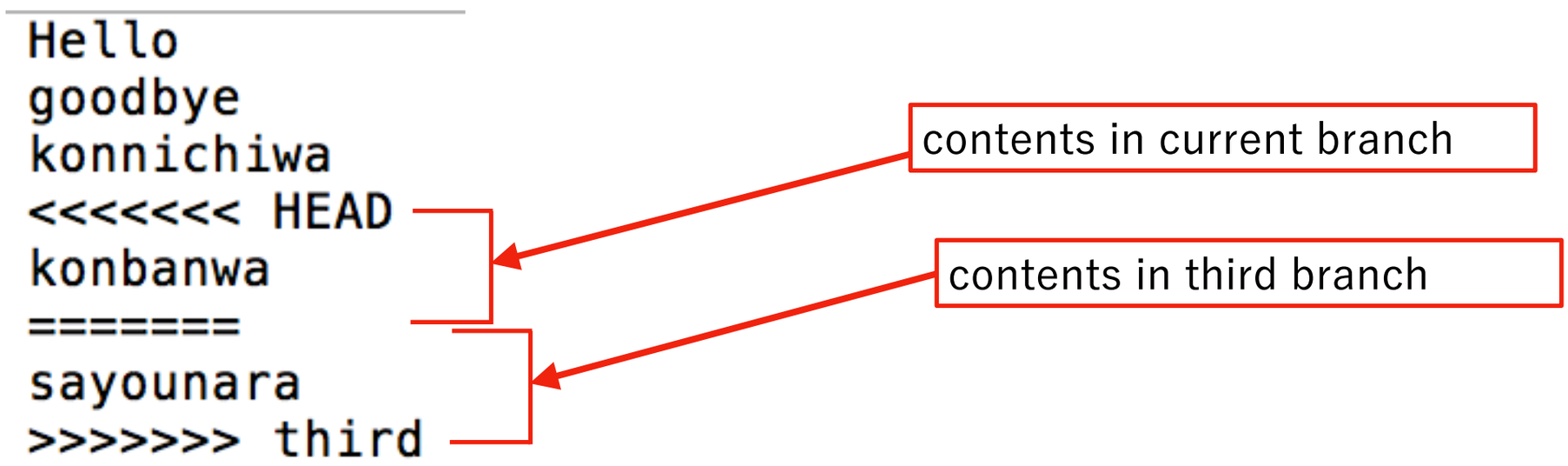
Git - Solving conflict on local command line

You can also solve conflicts on your local command line environment.

After applying changes on master branch to your local branch by using “git merge” command, conflict may occur if there are some conflicts to the files which you changed on local branch.

```
Auto-merging index.php
CONFLICT (content): Merge conflict in index.php
Automatic merge failed; fix conflicts and then commit the result.
```

After conflict occurred, file which contains conflict modified like below.

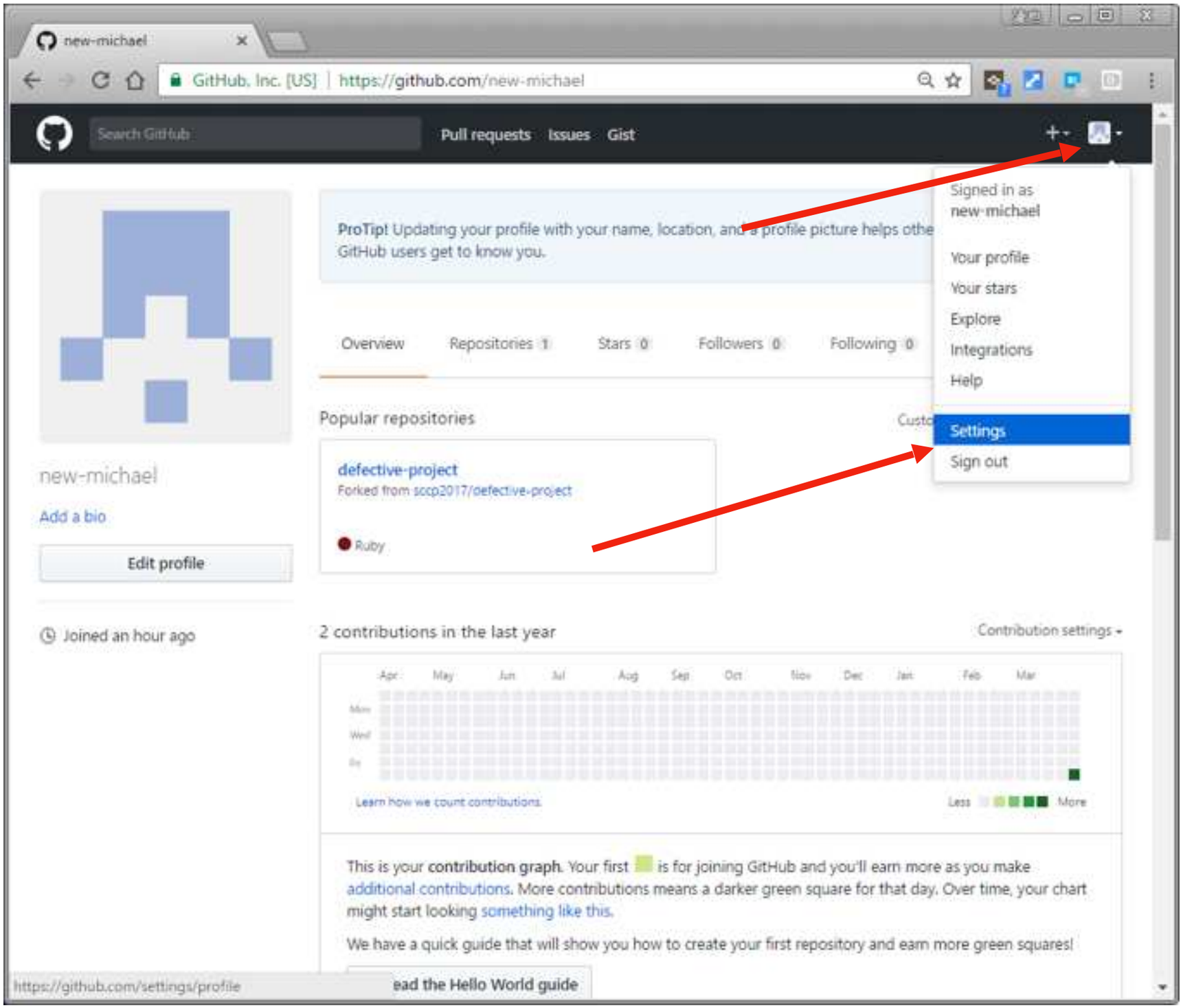


You should decide which one is correct and modify it for each file.

SSH registration

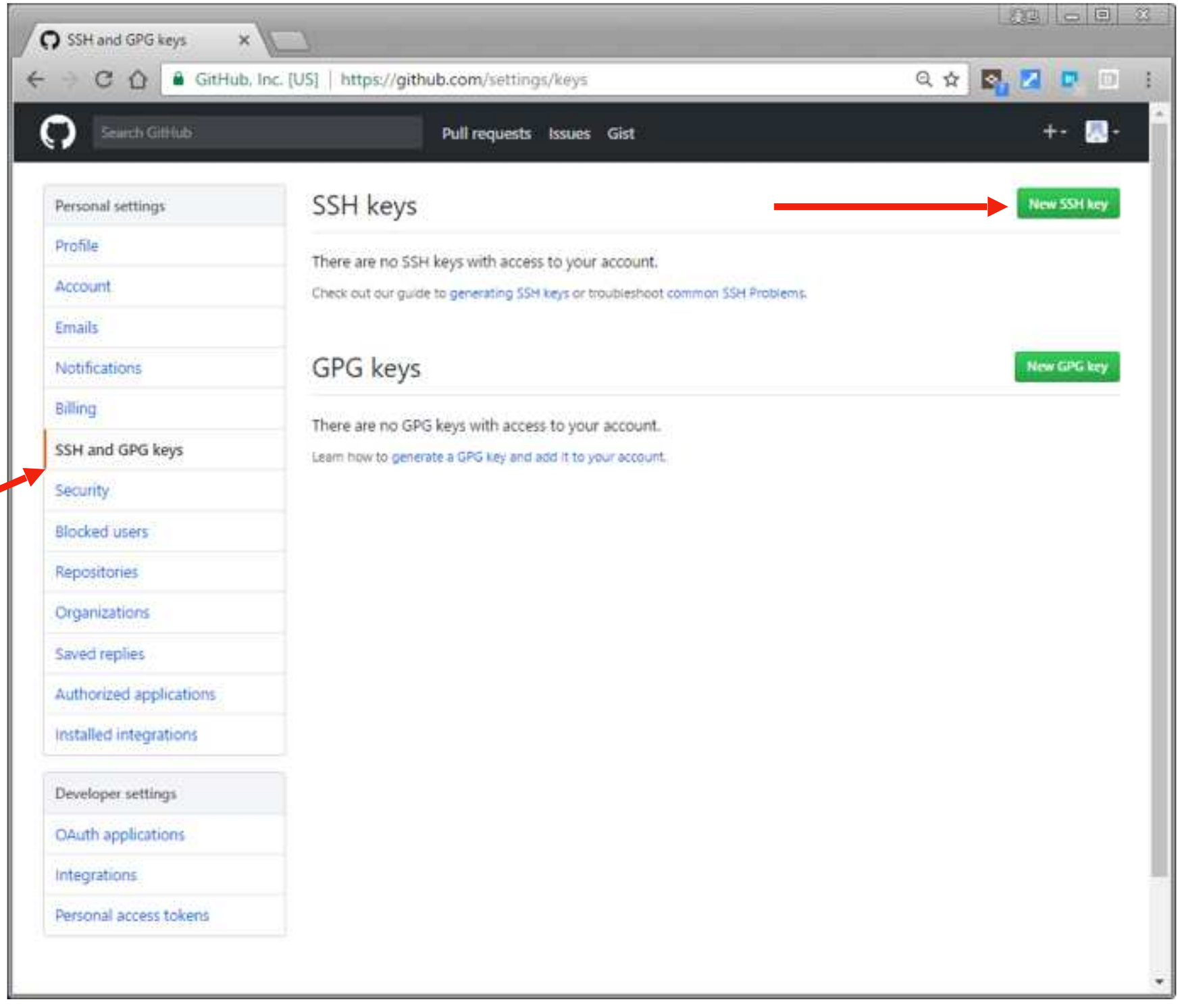
Git - SSH settings (Optional)

After logged-in, click “Settings” on pulldown menu at GitHub user page.



Git - SSH settings (Optional)

Click SSH and GPG keys and select New SSH Key.

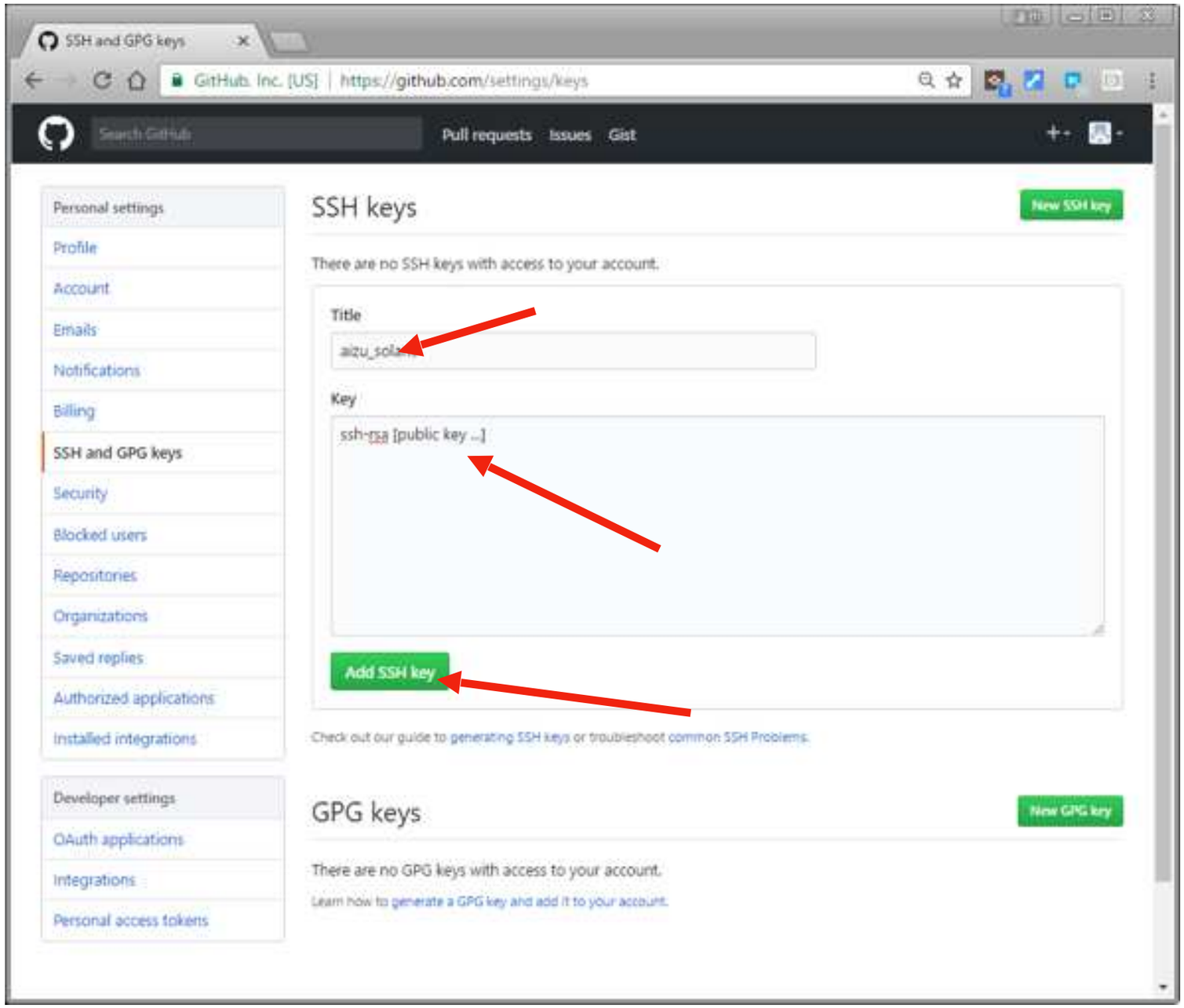


Git - SSH settings (Optional)

For an pair of private key(~/.ssh/id_rsa) and public key(~/.ssh/id_rsa.pub), copy public key and paste “Key” text area beginning with “ssh-rsa ”.

Make sure appropriate title was inputted at “Title” text box. It is useful when you register another SSH Keys from the other environment like your laptop.

After inputted, click “Add SSH Key”.



When you create a pair of public key and private key on Solaris environment, use below command.

```
$ ssh-keygen -t rsa
$ cd ~/.ssh
$ cat id_rsa.pub
```