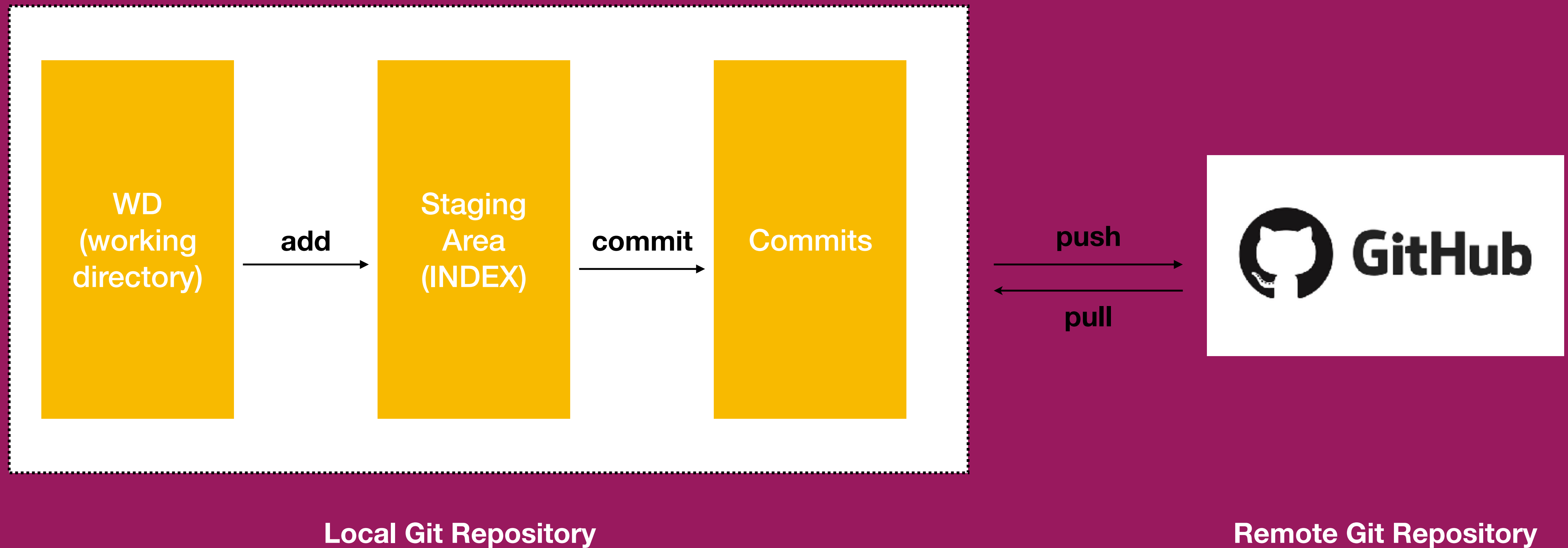


Github Flow

- Github으로 협업하는 흐름

Git (이미 알고 있어야하는 개념 정리)



Git 들어가기전에 ..

Git을 CLI(Command Line Interface)에서 활용하기 위해서는 아래의 명령어는 필수적이다!

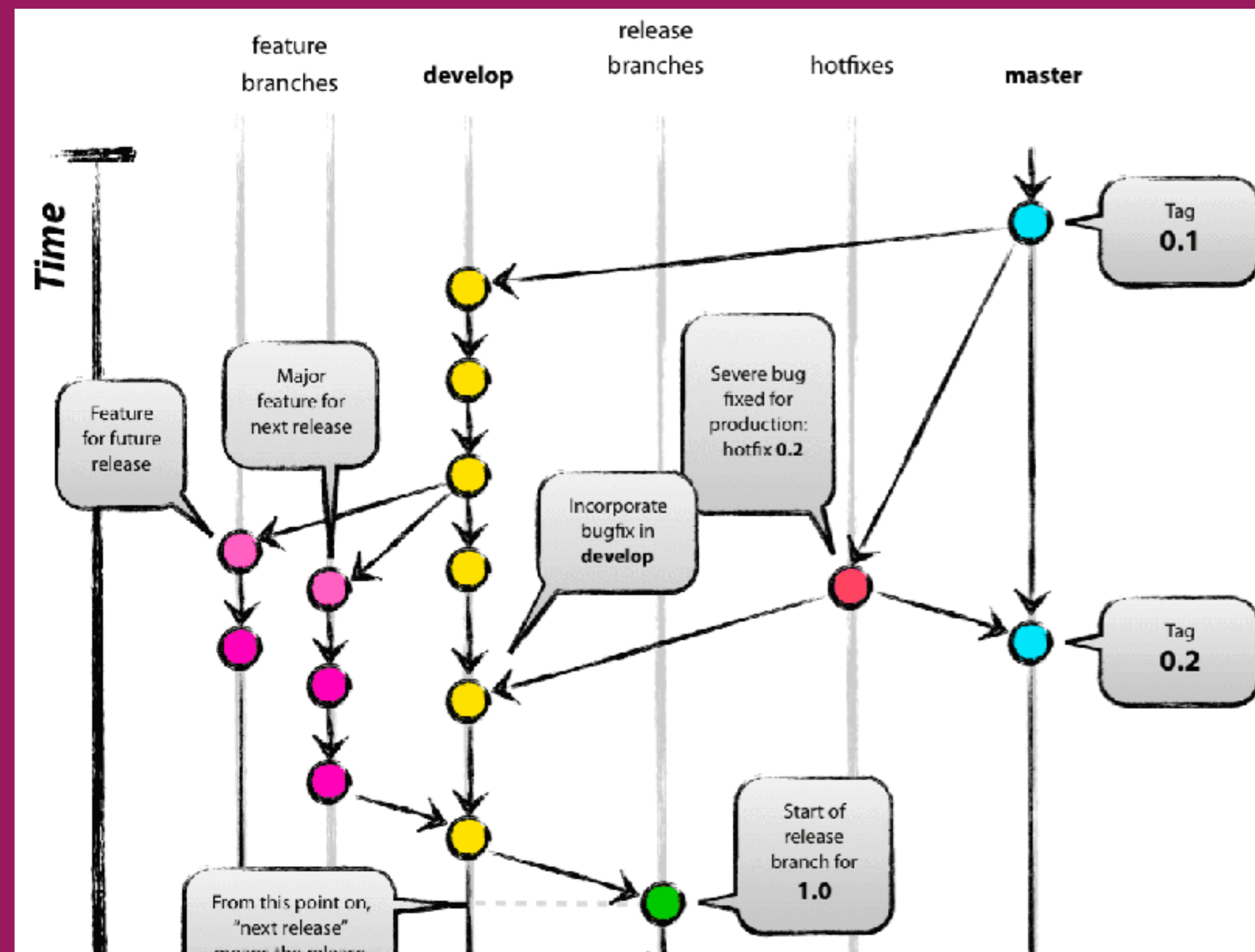
항상 모든 명령어 뒤에 상태를 확인하자.

```
$ git status
```

Git Flow

Git을 활용하여 협업하는 흐름으로 branch를 활용하는 전략을 의미한다.

가장 대표적으로 활용되는 전략은 다음과 같다.



Git Flow

branch	주요 특징	예시
master (main)	* 배포 가능한 상태의 코드	LOL 클라이언트 라이브 버전 (9.23.298.3143)
develop (main)	* feature branch로 나뉘어지거나, 발생된 버그 수정 등 개발 진행 * 개발 이후 release branch로 갈라짐.	다음 패치를 위한 개발 (9.24)
feature branches (supporting)	* 기능별 개발 브랜치(topic branch) * 기능이 반영되거나 드랍되는 경우 브랜치 삭제	개발시 기능별 예) 신규챔피언 세나, 드래곤 업데이트
release branches (supporting)	* 개발 완료 이후 QA/Test 등을 통해 얻어진 다음 배포 전 minor bug fix 등 반영	9.24a, 9.24b, ...
hotfixes (supporting)	* 긴급하게 반영 해야하는 bug fix * release branch는 다음 버전을 위한 것이라면, hotfix branch는 현재 버전을 위한 것	긴급 패치를 위한 작업 예) 버그로 인한 챔피언 선택 금지

Git Flow

Git Flow는 정해진 답이 있는 것은 아니다.

Github Flow, Gitlab Flow 등의 각 서비스별 제안되는 흐름이 있으며,
변형되어 각자의 프로젝트/회사에서 활용 되고 있다.

간단하게 브랜치를 활용하는 명령어를 알아보고,
프로젝트에 활용할 수 있는 간단한 버전의 브랜치 전략을 배워보자.

Branch basic commands

1. 브랜치 생성

```
(master) $ git branch {branch name}
```

2. 브랜치 이동

```
(master) $ git checkout {branch name}
```

3. 브랜치 생성 및 이동

```
(master) $ git checkout -b {branch name}
```

4. 브랜치 목록

```
(master) $ git branch
```

5. 브랜치 삭제

```
(master) $ git branch -d {branch name}
```

Branch merge

각 branch에서 작업을 한 이후 이력을 합치기 위해서는 일반적으로 merge 명령어를 사용한다.

병합을 진행할 때, 만약 서로 다른 이력(commit)에서 동일한 파일을 수정한 경우 충돌이 발생할 수 있다.

이 경우에는 반드시 직접 수정을 진행 해야 한다.

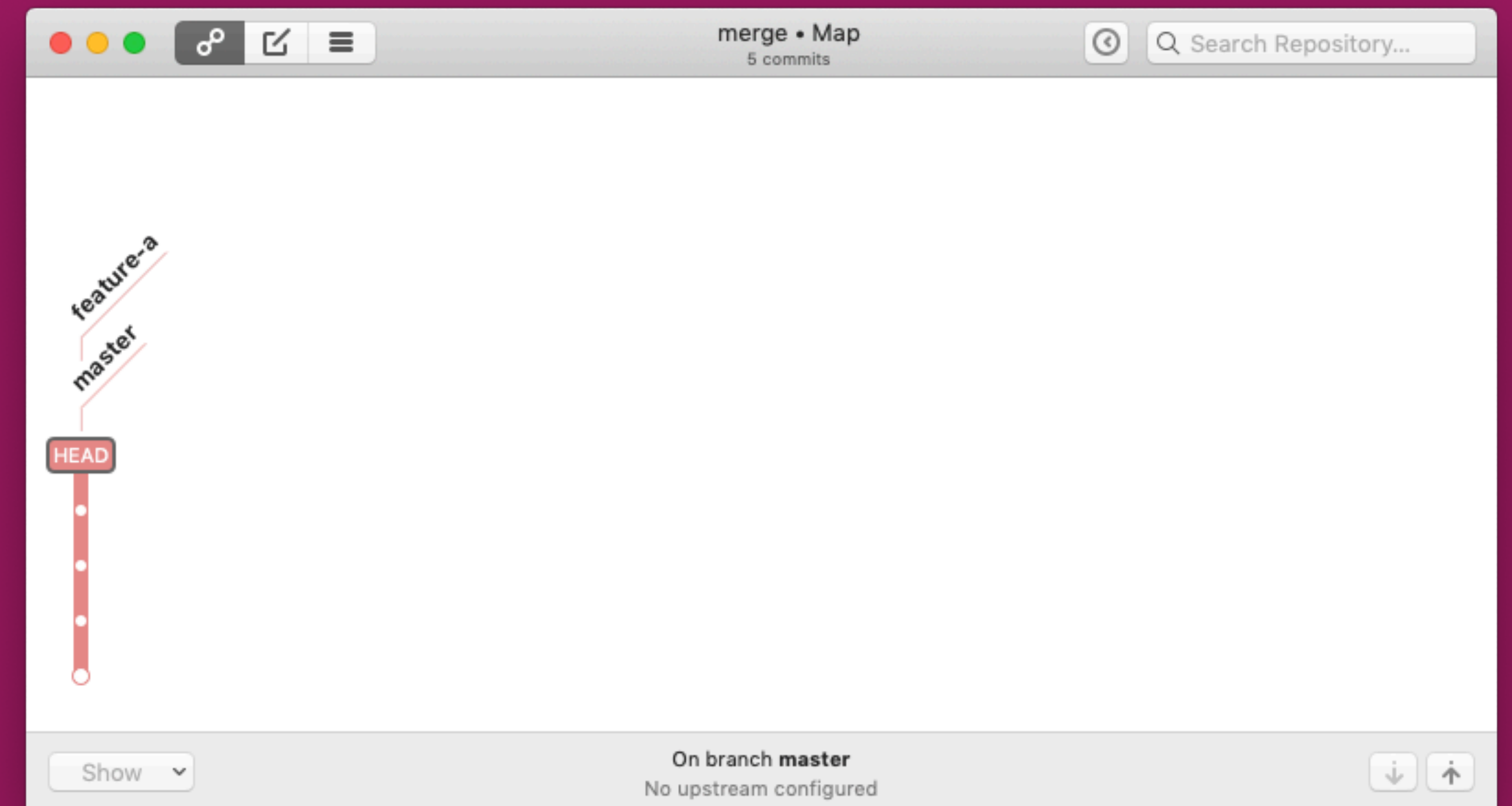
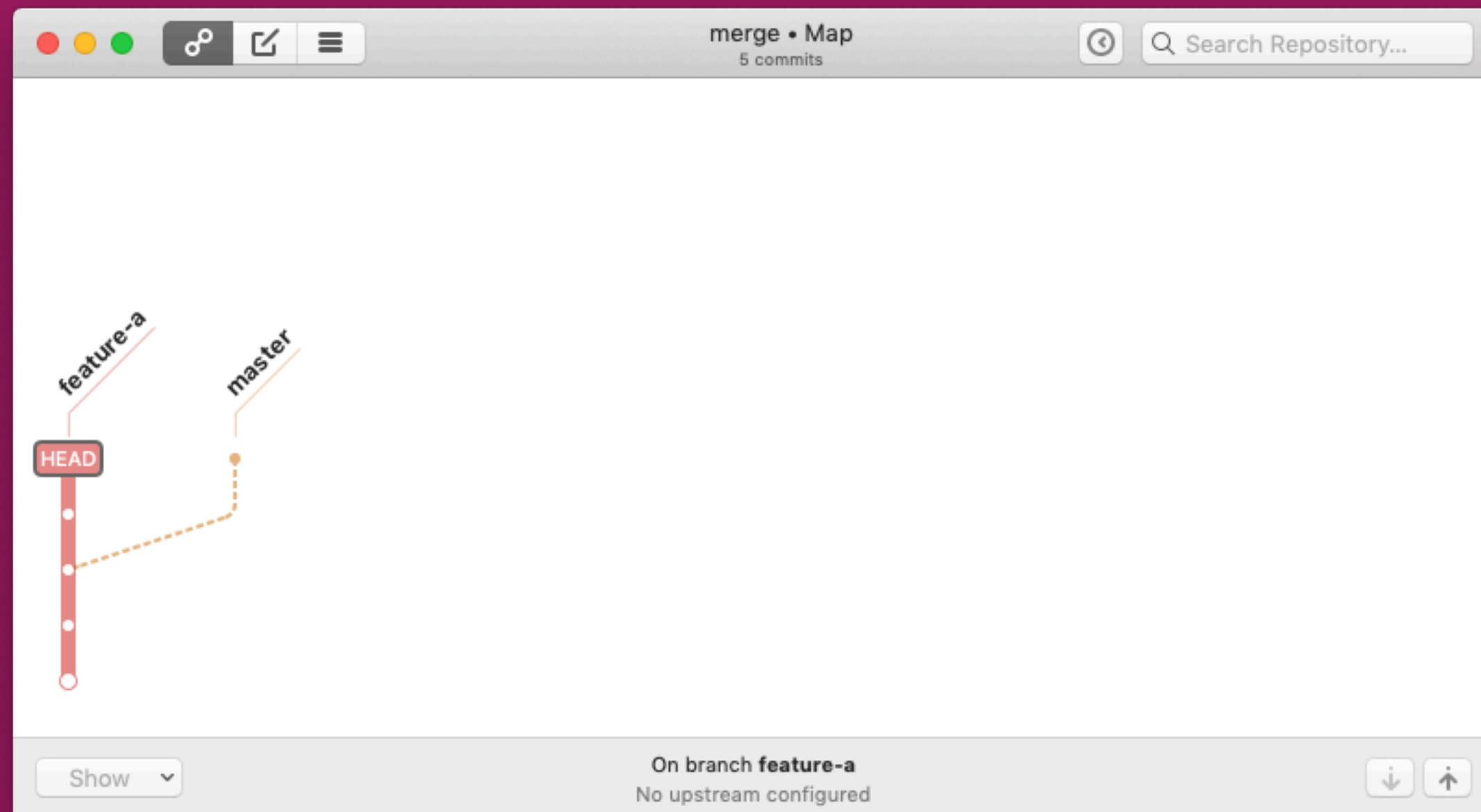
충돌이 발생한 것은 오류가 발생한 것이 아니라 이력이 변경되는 과정에서 반드시 발생할 수 있는 것이다.

Branch merge - fast-forward

기존 **master** 브랜치에 변경사항이 없어 단순히 앞으로 이동

```
(master) $ git merge feature-a
Updating 54b9314..5429f25
Fast-forward
```

1. feature-a branch로 이동 후 commit
2. master 별도 변경 없음
3. master branch로 병합

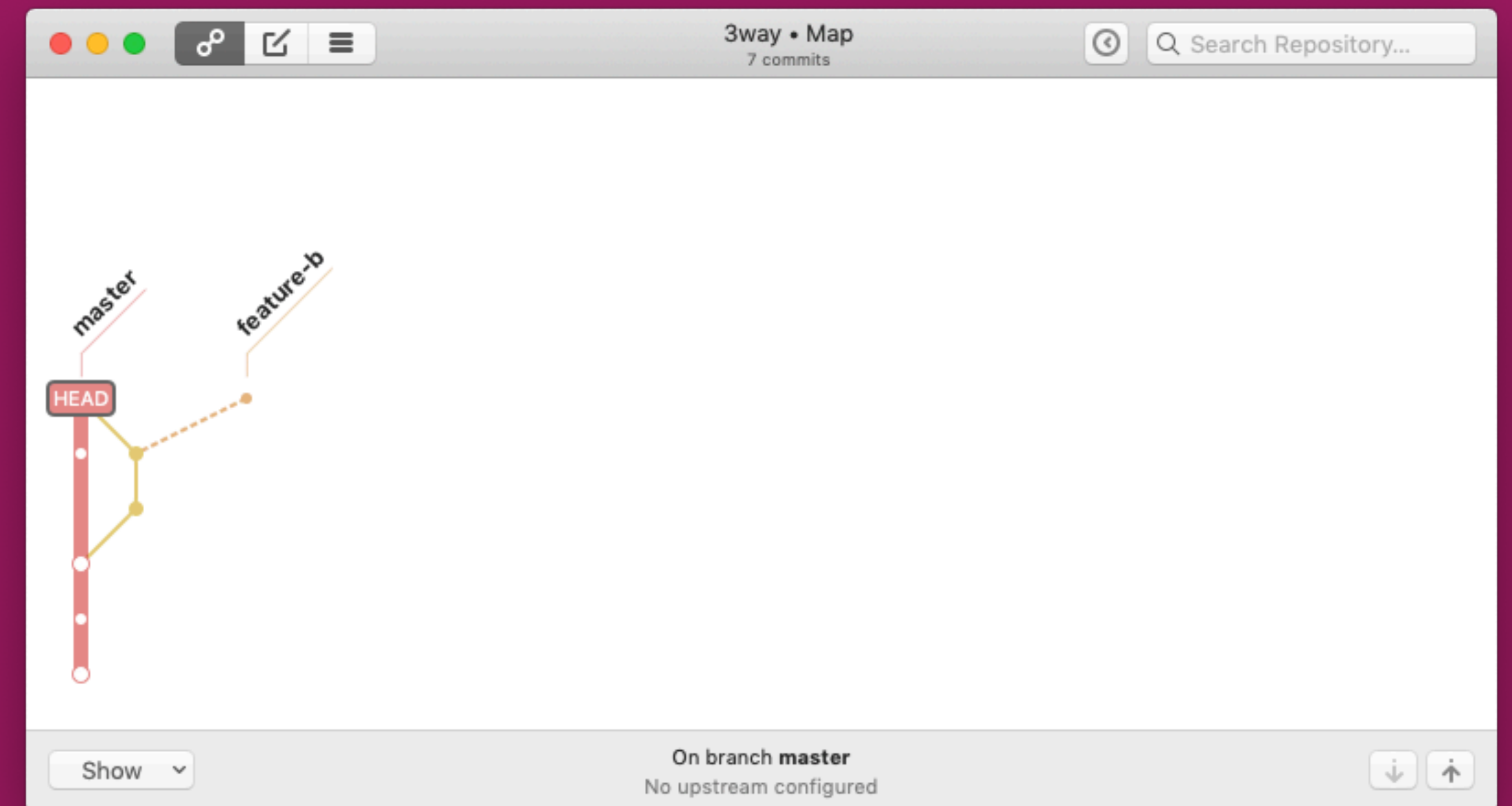
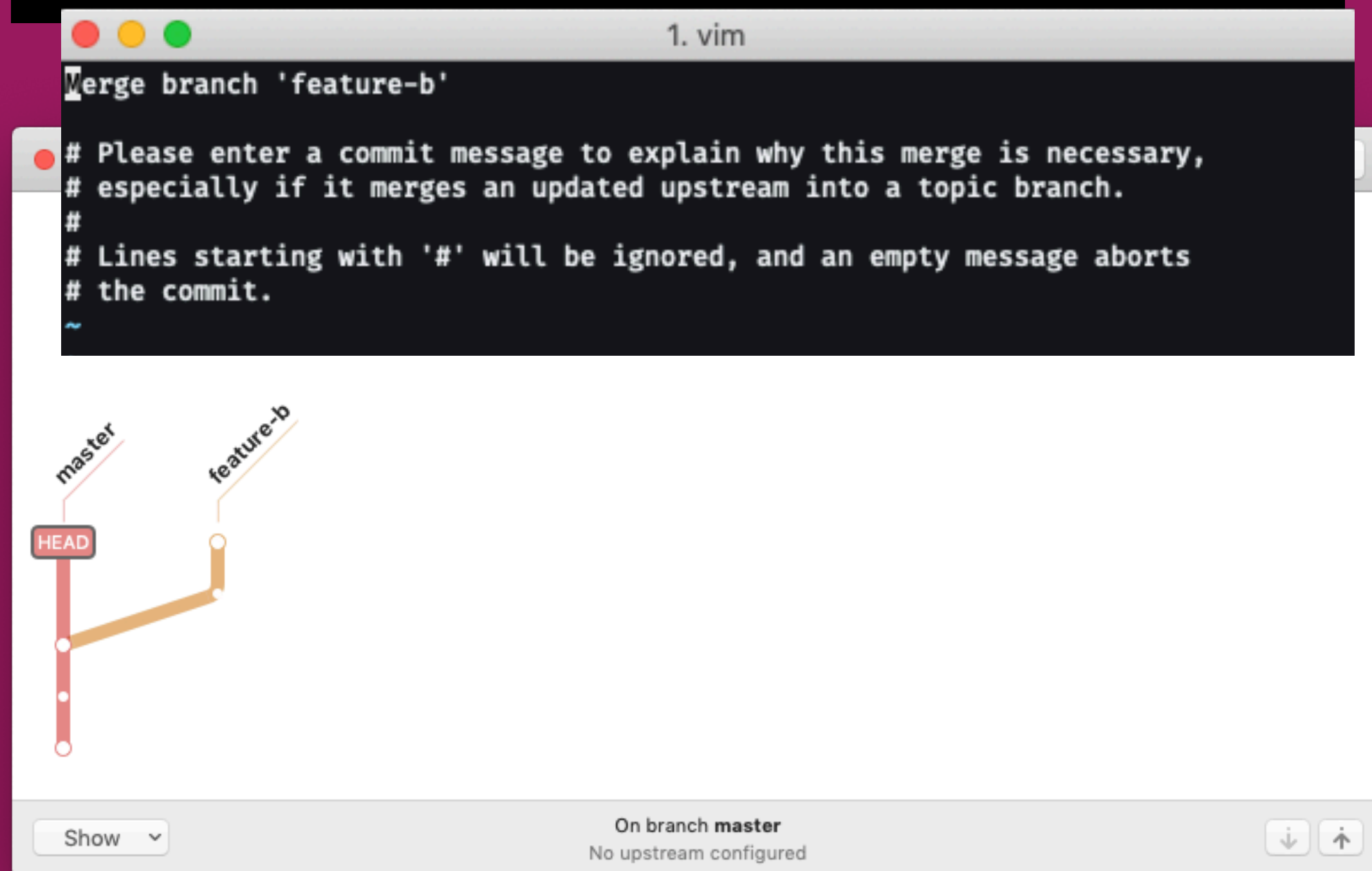


Branch merge - merge commit

기존 **master** 브랜치에 변경사항이 있어 병합 커밋 발생

```
(master) $ git merge feature-a
Already up to date!
Merge made by the 'recursive' strategy.
```

1. feature-a branch로 이동 후 commit
2. master branch commit
3. master branch로 병합

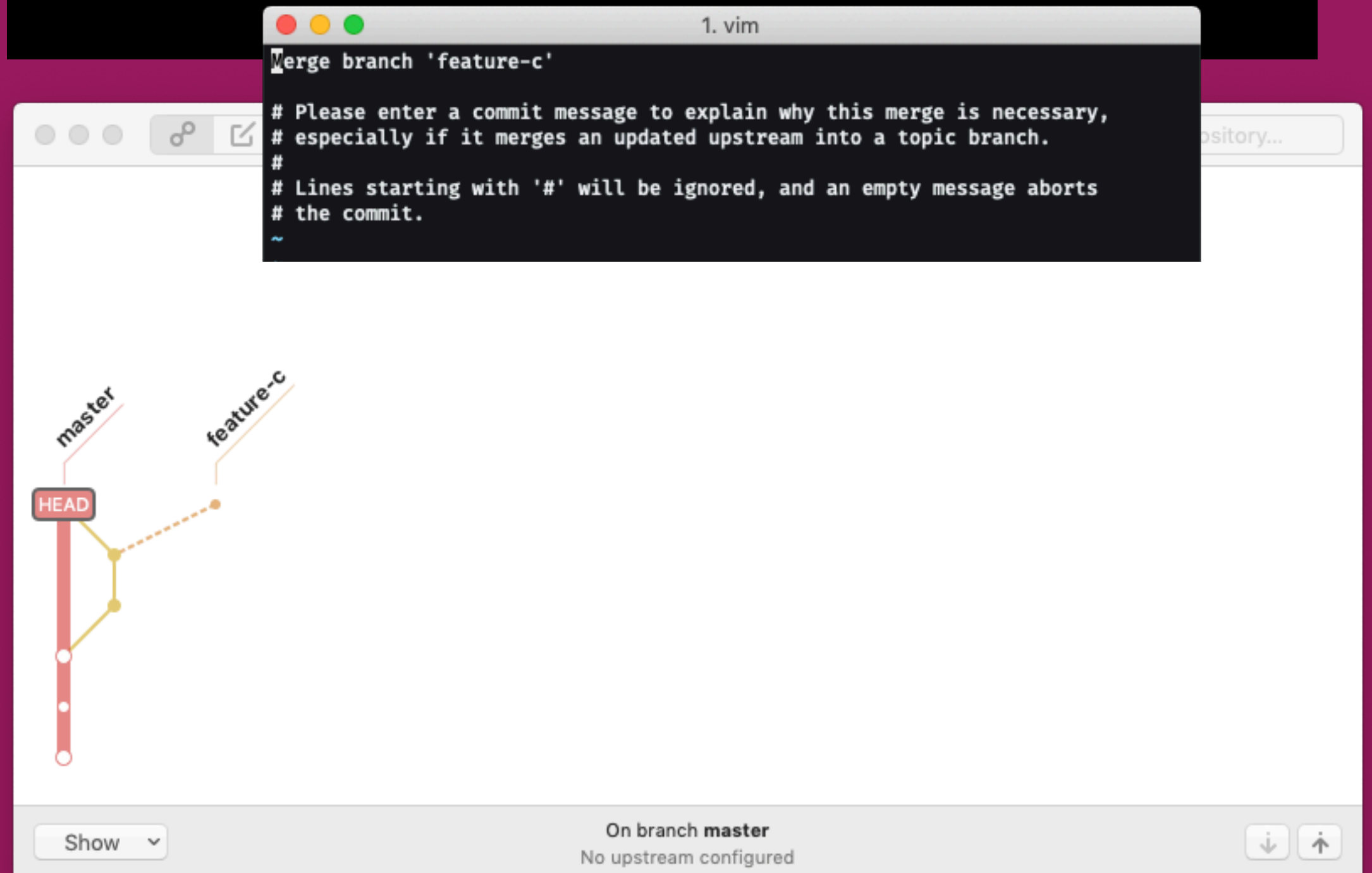


Branch merge - --no-ff (심화)

fast forwarding 상황에서도 commit을 발생시키는 옵션.
branch 이력을 유지한다는 장점이 있다.

```
(master) $ git merge feature-a
```

```
(master) $ git merge --no-ff feature-c
```



Branch rebase

* rebase는 merge와 최종 결과는 같으나, commit history만 다르게 형성된다.

따라서, remote repository에 push된 commit에 대해서는 절대 rebase를 진행하면 안된다.

따라서, 본 자료에서는 설명하지는 않을 예정이며 링크로 대체한다.

(<https://git-scm.com/book/ko/v2/Git-%EB%B8%8C%EB%9E%9C%EC%B9%98-Rebase-%ED%95%98%EA%B8%B0>)

Github Flow 기본 원칙

Github Flow는 Github에서 제안하는 브랜치 전략으로 다음과 같은 기본 원칙을 가지고 있다.

1. **master branch**는 반드시 배포 가능한 상태여야 한다.

There's only one rule: anything in the master branch is always deployable.

2. **feature branch**는 각 기능의 의도를 알 수 있도록 작성한다.

Your branch name should be descriptive,
so that others can see what is being worked on.

3. **Commit message**는 매우 중요하며, 명확하게 작성한다.

Commit messages are important. By writing clear commit messages,
you can make it easier for other people to follow along and provide feedback.

4. **Pull Request**를 통해 협업을 진행한다.

Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories.

5. 변경사항을 반영하고 싶다면, **master branch**에 병합한다.

Now that your changes have been verified in production, it is time to merge your code into the master branch.

Github Flow Models

앞서 설명된 기본 원칙 아래 Github에서 제시하는 방법이 2가지가 있다.

- * Shared Repository Model

- * Fork & Pull Model

이 두 모델의 가장 큰 차이점은 내(작업자)가 해당 프로젝트 저장소에 직접적인 **push** 권한이 있는지 여부 !!

* 추가적인 내용은 <https://guides.github.com/> 참고

Shared Repository Model

Shared Repository Model은 동일한 저장소를 공유하여 활용하는 방식.

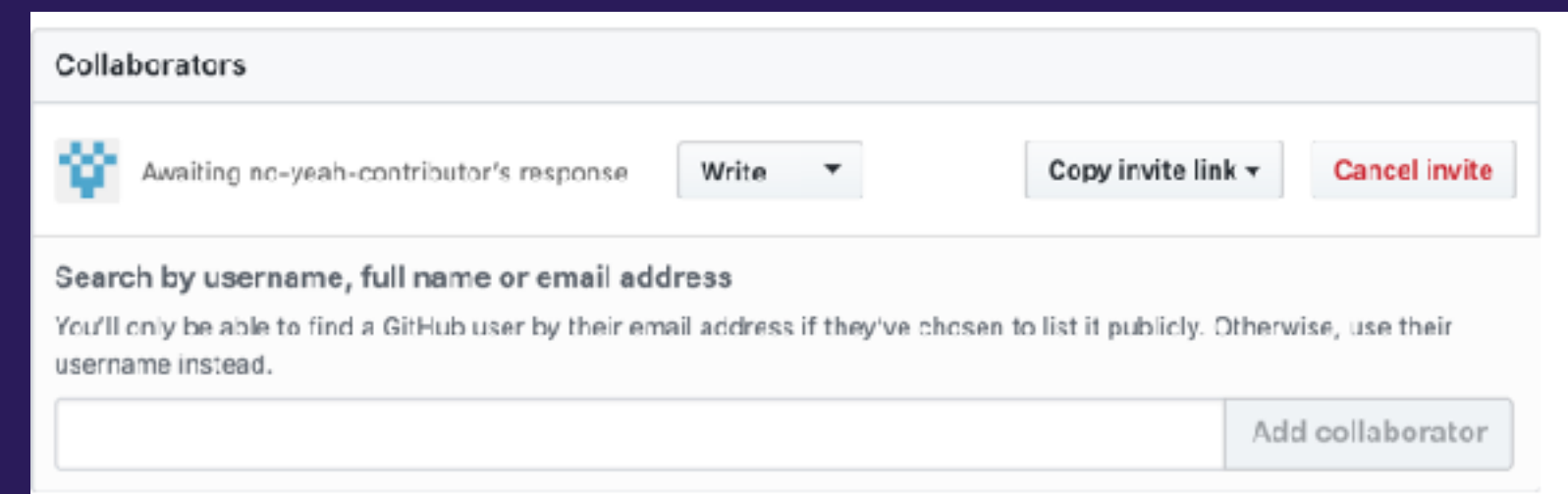
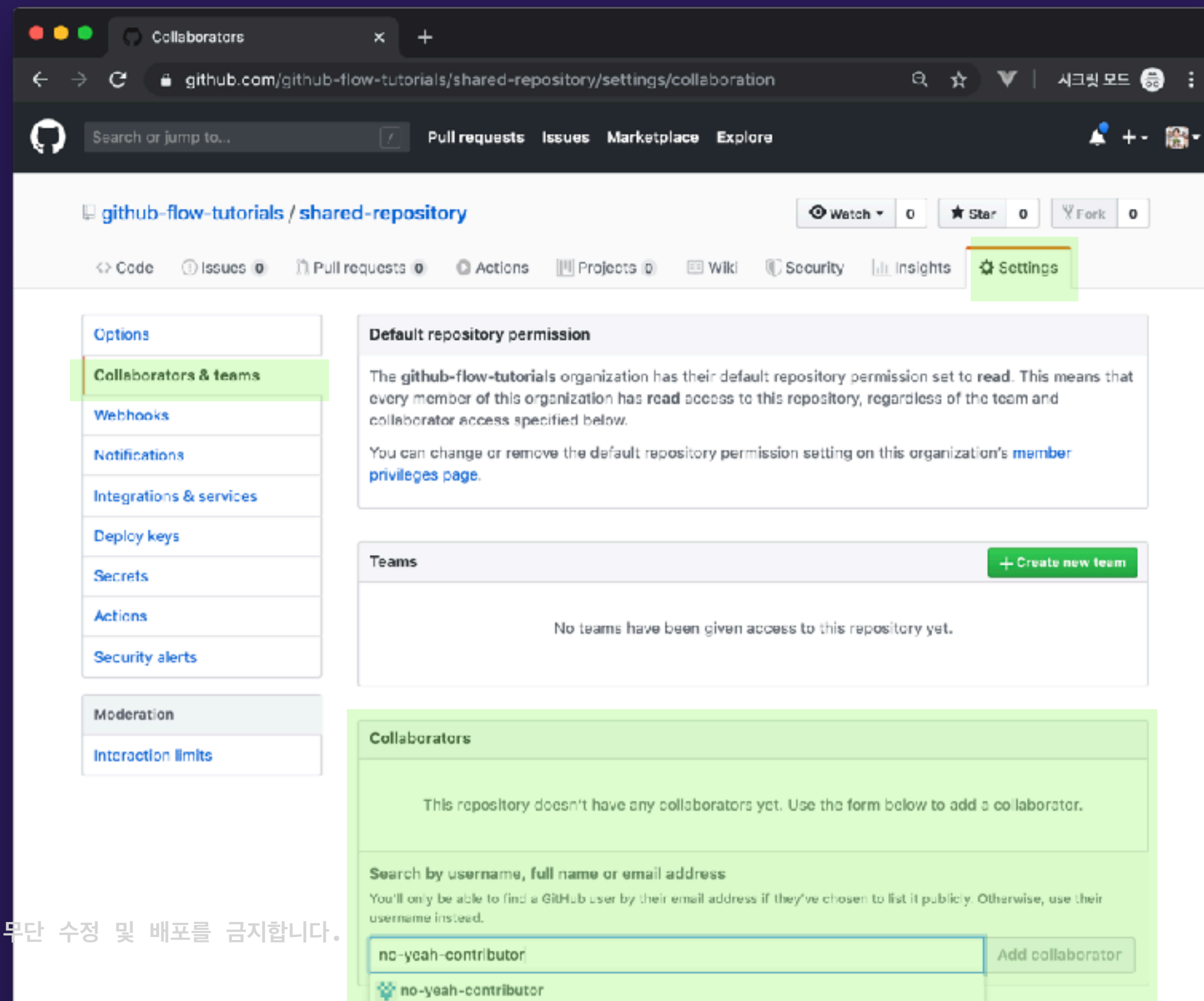
 : repository owner (project manager)
 : collaborator (a.k.a. no-yeah-contributor)

* 작업 흐름은 초보자를 위해 간단하게 master + feature 브랜치를 활용하는 방식으로 설명합니다.

step 0-1. Invite collaborator



* collaborator에 등록 되어야 해당 저장소에 대한 push 권한이 부여된다.

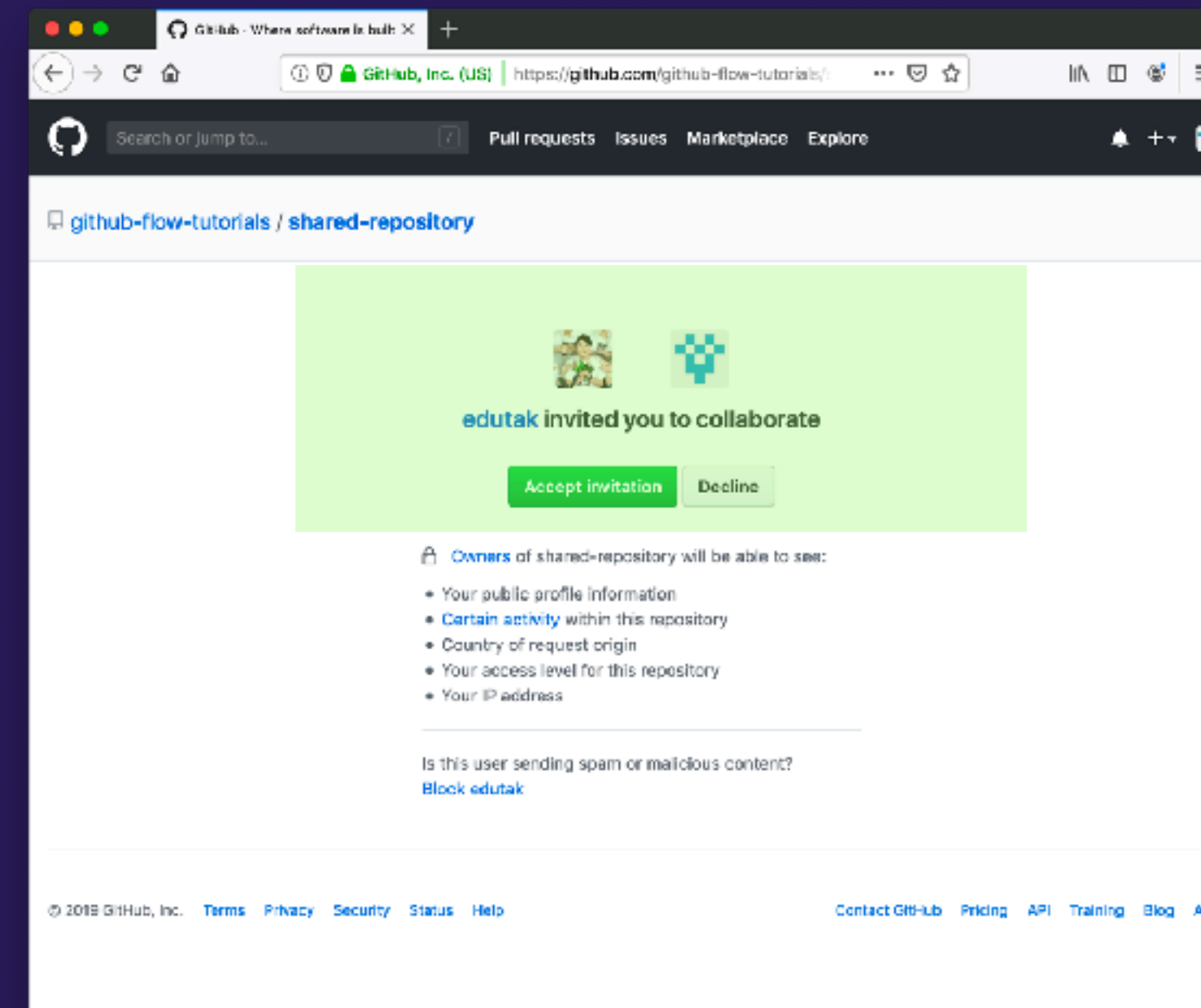
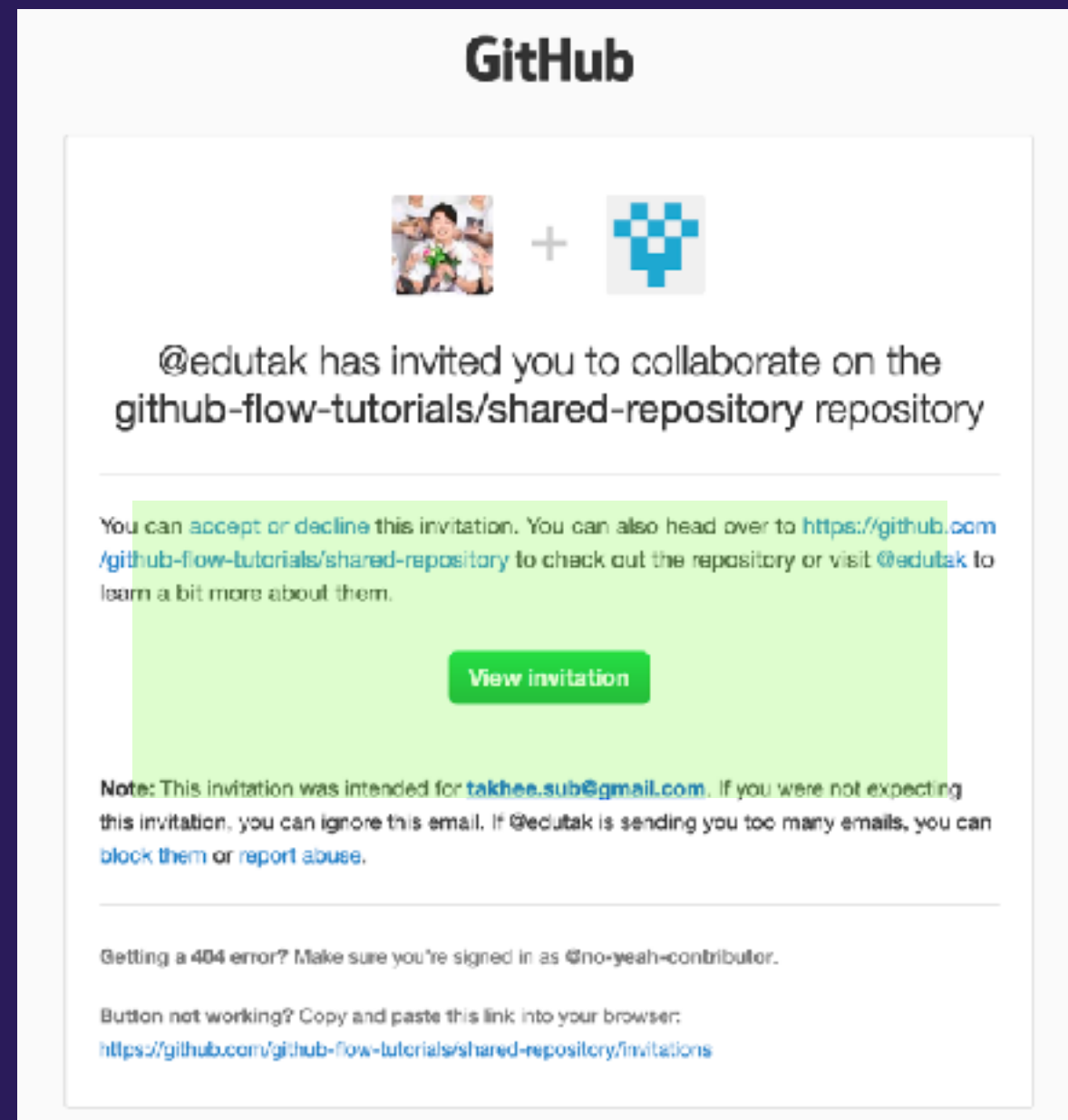


step 0-2. Accept Invitation



\$ 이메일을 통한 초대 수락

* Tip! 이메일이 아닌 해당 저장소 주소 뒤에 /invitation을 붙이면 오른쪽 화면을 볼 수 있다.



step 0-3. Clone project(remote) repository



\$ Clone을 하고 각 작업에 맞춘 작업 환경 설정을 마무리 한다.

* 예를 들면, node의 경우 일반적으로 .gitignore에
node_modules/가 등록되어 있으므로
npm install을 진행.

```
$ git clone {project repository url}
```

step 1. Create feature branch

 \$ 작업은 항상 독립적인 **feature branch**에서 한다.

* master branch는 항상 배포 가능한 상태를 유지하고,
영향이 가지 않도록 독립적인 branch에서 작업을 하는 것이다.

* feature branch는 이름을 생성할 때, 기능을 명시적으로 나타낸다.

```
(master) $ git checkout -b feature/accounts-login  
(feature/accounts-login) $ touch develop-login.txt
```

* 작업시 항상 어떠한 branch에 있는지 확인하는 것이 중요하다.

step 2-1. Commit



\$ Commit을 통해 작업의 이력(history)을 남긴다.

- * Commit은 다른 사람들이 내가 한 작업들을 확인할 수 있는 이력이며, 코드의 변화에 맞춰 실시한다.
- * Commit 메시지는 매우 중요하며, 일관된 형식으로 해당 이력을 쉽게 파악할 수 있도록 작성한다.
(commit 메시지를 활용하면 Github 작업을 이끌 수도 있다.)

```
(feature/accounts-login) $ git add develop-login.txt  
(feature/accounts-login) $ git commit -m 'Complete login feature'
```

* `git status`와 `git log` 명령어를 반드시 활용하여 상태를 파악하자.

step 2-2. Push to remote repository

 \$ 완성된 코드는 원격 저장소에 **push**를 한다.

* master branch에 Push 하지 않도록 유의한다.

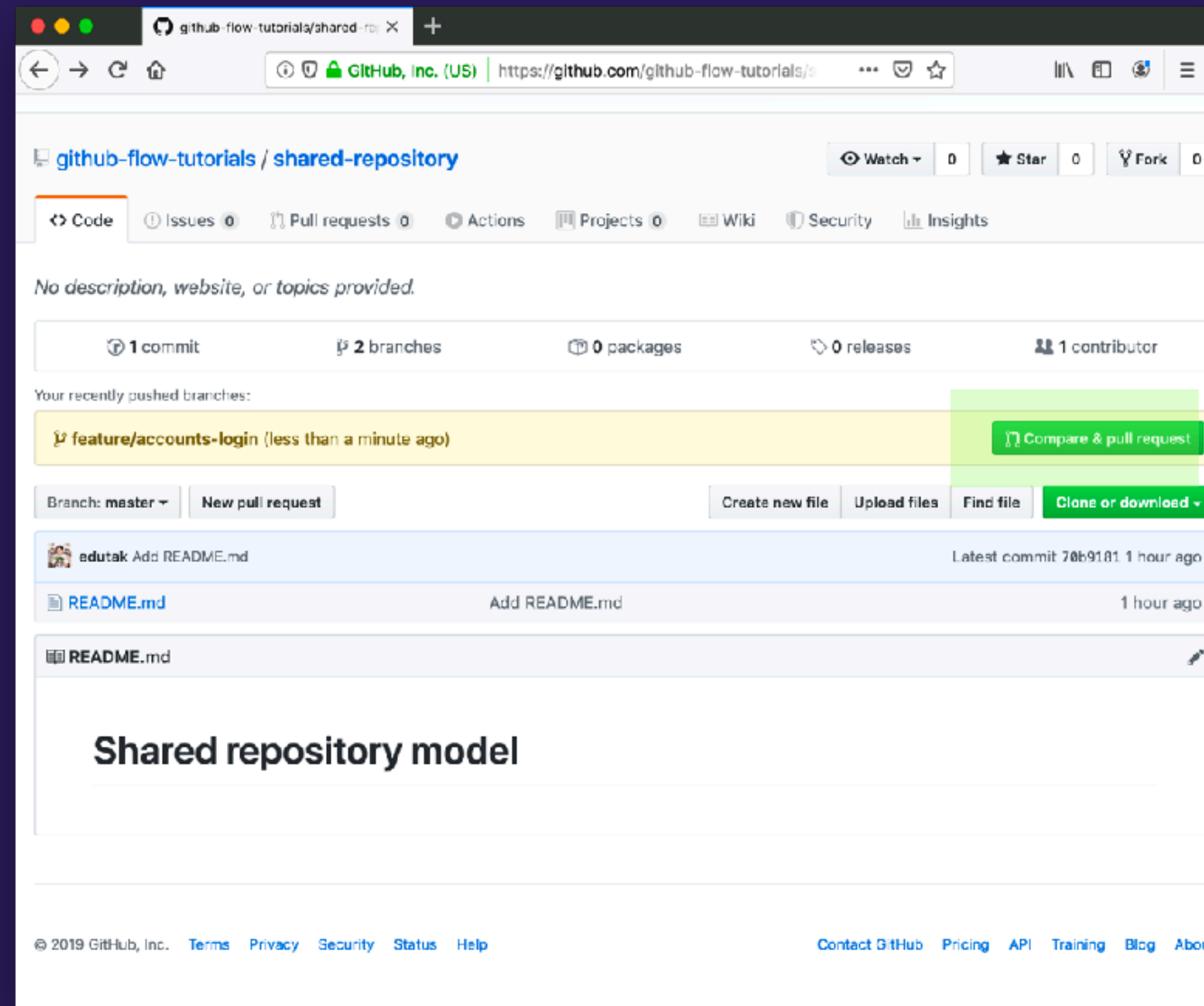
```
(feature/accounts-login) $ git push origin feature/accounts-login
```

* git push를 하기 이전에, 코드와 커밋 상태를 반드시 확인하자. (status, log)
원격 저장소에 공개된 이력은 절대 변경 하여서는 안된다.

step 3-1. Open a Pull Request

 \$ Github에 들어가서 Pull Request 버튼을 누른다.

* Github Flow에서 핵심은 Pull Request를 통한 협업이라고 할 수 있다.



step 3-2. Create Pull Request

👤 \$ PR과 관련된 설정을 진행한 후 요청을 생성한다.

Open a pull request * 브랜치 확인

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

base: master * compare: feature/accounts-login ✓ Able to merge. These branches can be automatically merged.

Complete login feature

Write Preview

- * Create accounts app
- * Add User model based AbstractUser
- * Add Login urls
- * Add Bootstrap for Login form

* 작업 내용 작성 (markdown)

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

Create pull request

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

1 commit 1 file changed 0 commit comments

Commits on Nov 20, 2019

* 리뷰어 지정 등 옵션

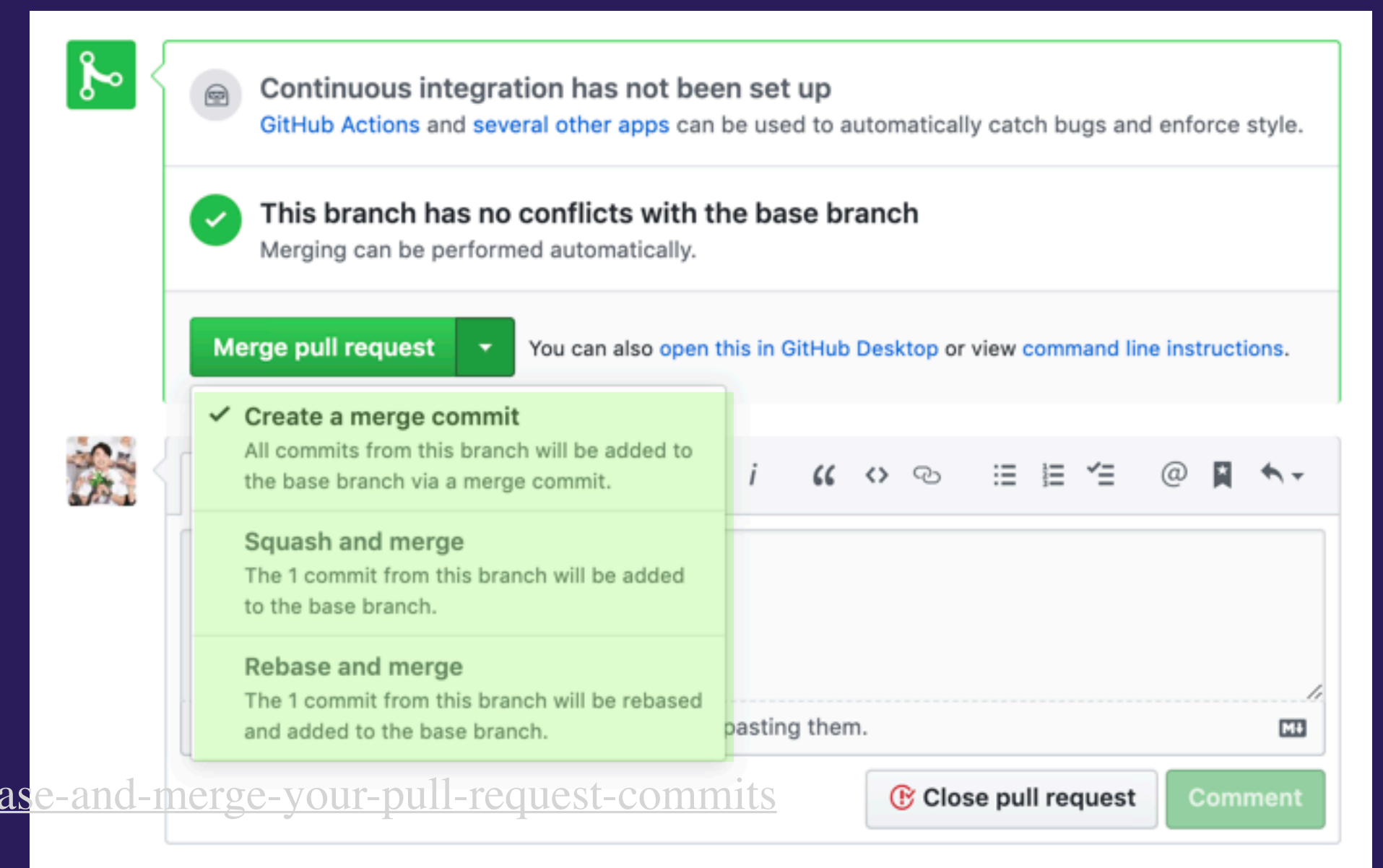
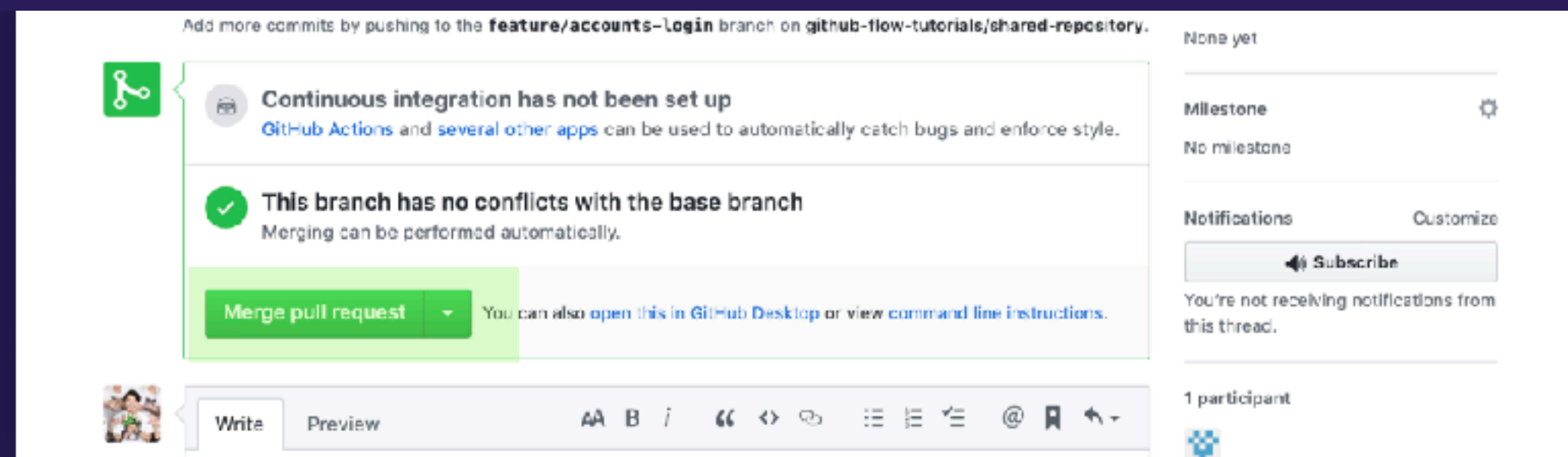
step 3-3. Merge pull request

👩💻 \$ 작성된 코드를 확인 후 병합

* 병합(merge) 과정에서 충돌이 발생할 경우 해결 후 병합을 진행한다.

* 병합시 커밋 이력을 정리하기 위한 추가 옵션을 선택할 수도 있다. (squash : 커밋 병합, rebase)*

* master branch로 병합의 경우 코드가 반드시 배포 가능한 상태여야 한다.

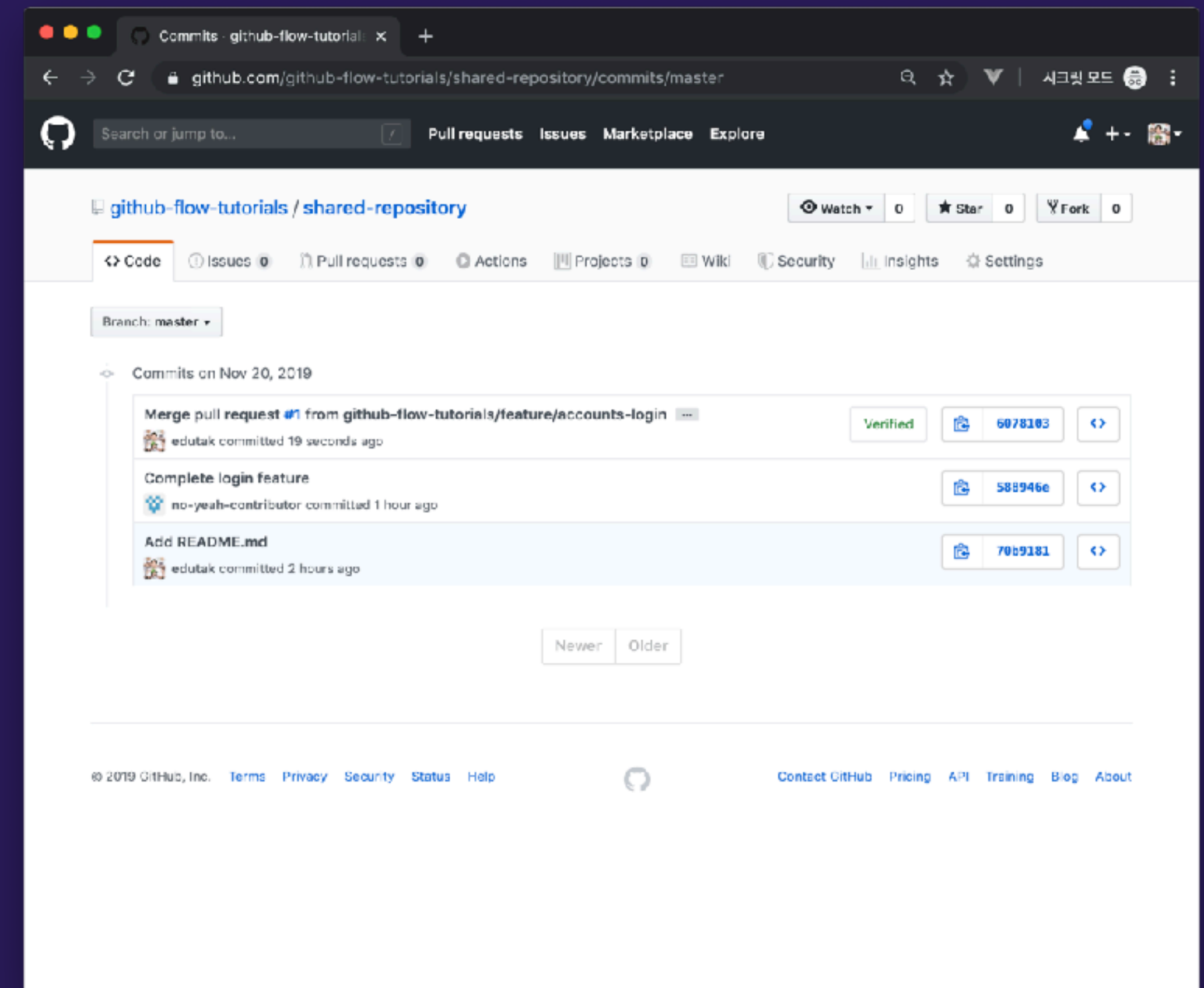
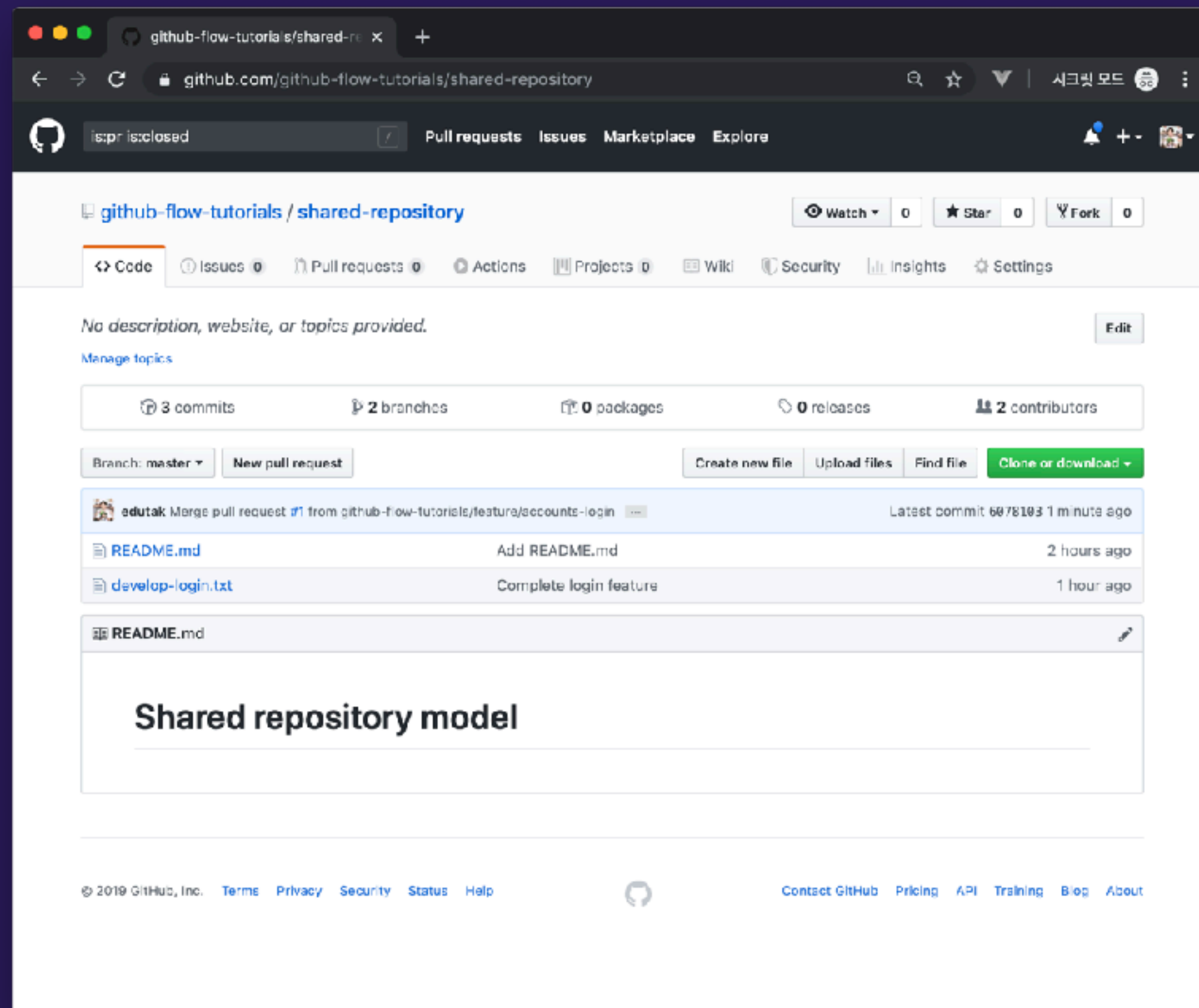


* <https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-request-merges#rebase-and-merge-your-pull-request-commits>

학생들을 위해 작성된 별도의 자료로써 무단 수정 및 배포를 금지합니다.

step 3-4. Merged !!

* 병합이 완료되면, master 브랜치에 반영된 것을 확인할 수 있다.



step 4. while True:

 \$ 다음 작업 준비!

* 로컬 저장소에서는
merge된 branch는 삭제하고 master branch를 업데이트 한다.

* 이후 1~3 과정을 반복한다.

```
(feature/accounts-login) $ git checkout master  
(master) $ git branch -d feature/accounts-login  
(master) $ git pull origin master  
(master) $ git checkout -b feature/new-feature  
(feature/new-feature) $
```

Fork & Pull Model

Fork & Pull Model은 Repository에 Collaborator에 등록되지 않고,
Pull request를 통한 협업이 가능. Github 기반의 오픈소스 참여 과정에서 쓰이는 방식.

 : repository owner (project manager)

 : forker(?) (a.k.a. no-yeah-contributor)

* 작업 흐름은 초보자를 위해 간단하게 master + feature 브랜치를 활용하는 방식으로 설명합니다.

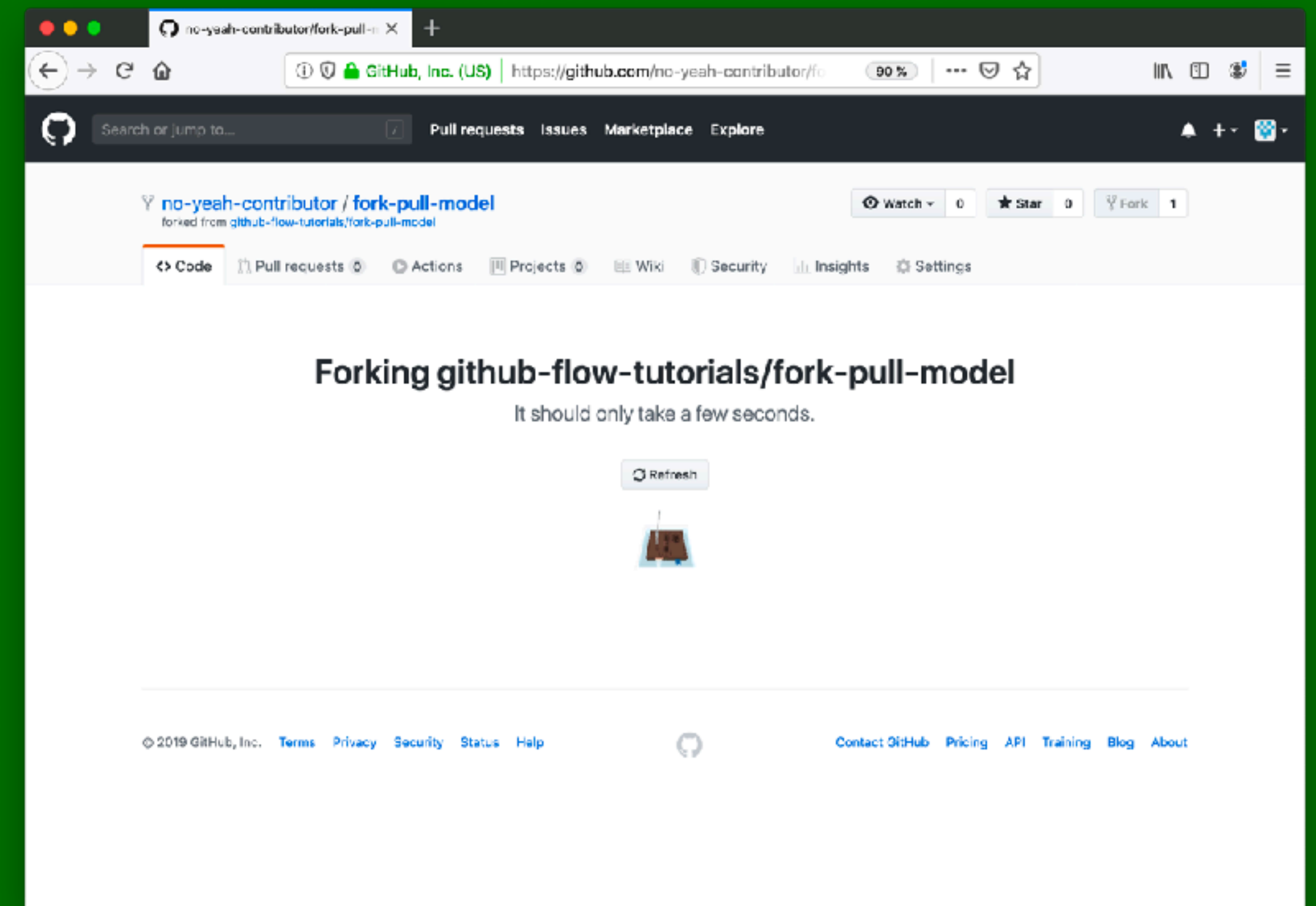
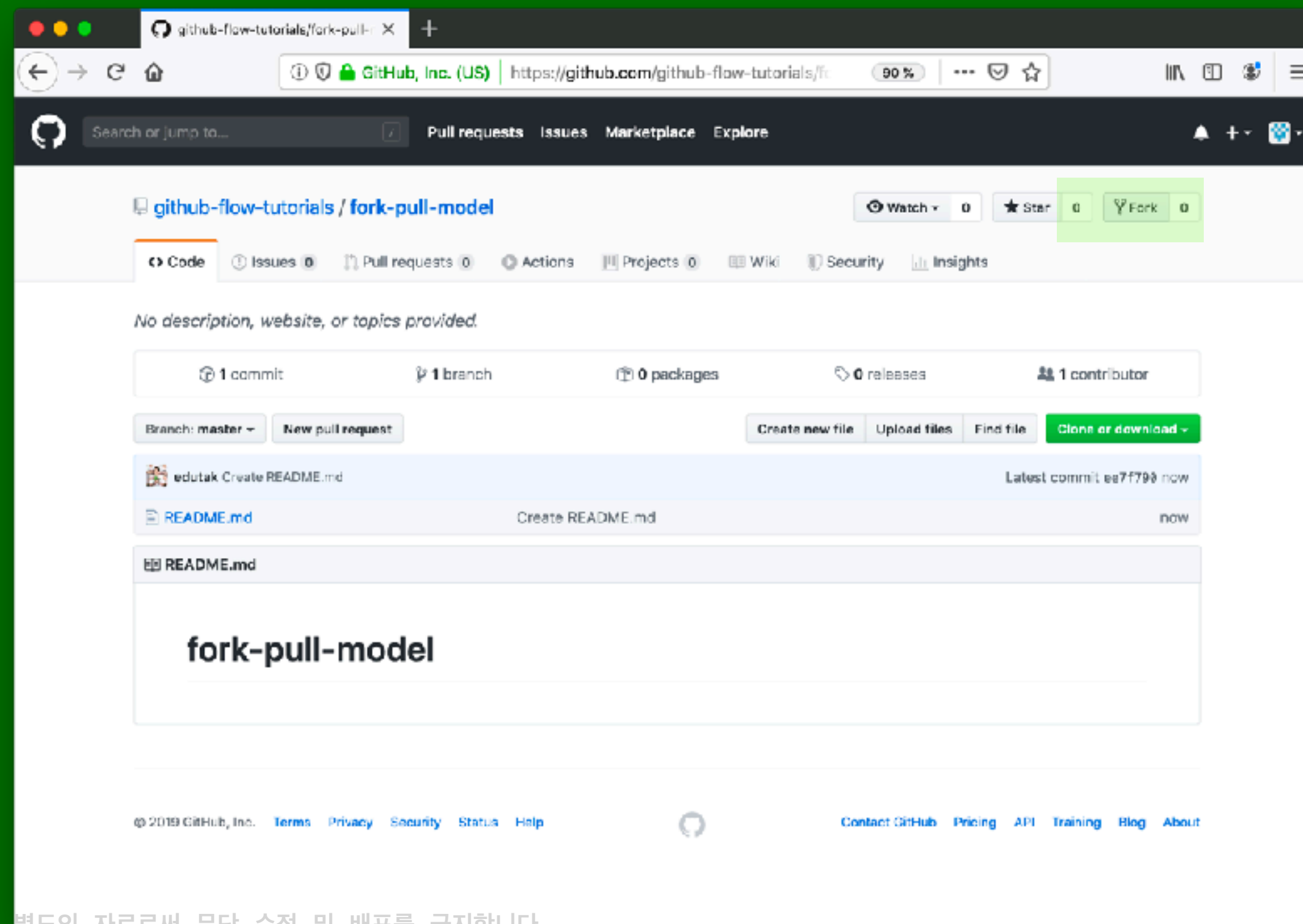
step 0-1. Fork repository



\$ Forking project repository

* 원격 저장소를 fork한다.

* 내 저장소로 복제본을 가져옴으로써 로컬에서 작업 후 원격 저장소로 push할 수 있게 되는 것.



step 0-2. Clone project(remote) repository



\$ Clone을 하고 각 작업에 맞춘 작업 환경 설정을 마무리 한다.

* 예를 들면, node의 경우 일반적으로 .gitignore에
node_modules/가 등록되어 있으므로
npm install을 진행.

```
$ git clone {project repository url}
```

step 1. Create feature branch

 \$ 작업은 항상 독립적인 **feature branch**에서 한다.

* master branch는 항상 배포 가능한 상태를 유지하고,
영향이 가지 않도록 독립적인 branch에서 작업을 하는 것이다.

* feature branch는 이름을 생성할 때, 기능을 명시적으로 나타낸다.

```
(master) $ git checkout -b feature/accounts-login
(feature/accounts-login) $ touch develop-login.txt
```

* 작업시 항상 어떠한 branch에 있는지 확인하는 것이 중요하다.

step 2-1. Commit

 \$ **Commit**을 통해 작업의 이력(**history**)을 남긴다.

- * Commit은 다른 사람들이 내가 한 작업들을 확인할 수 있는 이력이며, 코드의 변화에 맞춰 실시한다.
- * Commit 메시지는 매우 중요하며, 일관된 형식으로 해당 이력을 쉽게 파악할 수 있도록 작성한다.
(commit 메시지를 활용하면 Github 작업을 이끌 수도 있다.)

```
(feature/accounts-login) $ git add develop-login.txt
(feature/accounts-login) $ git commit -m 'Complete login feature'
```

* `git status`와 `git log` 명령어를 반드시 활용하여 상태를 파악하자.

step 2-2. Push to remote repository

 \$ 완성된 코드는 원격 저장소에 **push**를 한다.

* master branch에 Push 하지 않도록 유의한다.

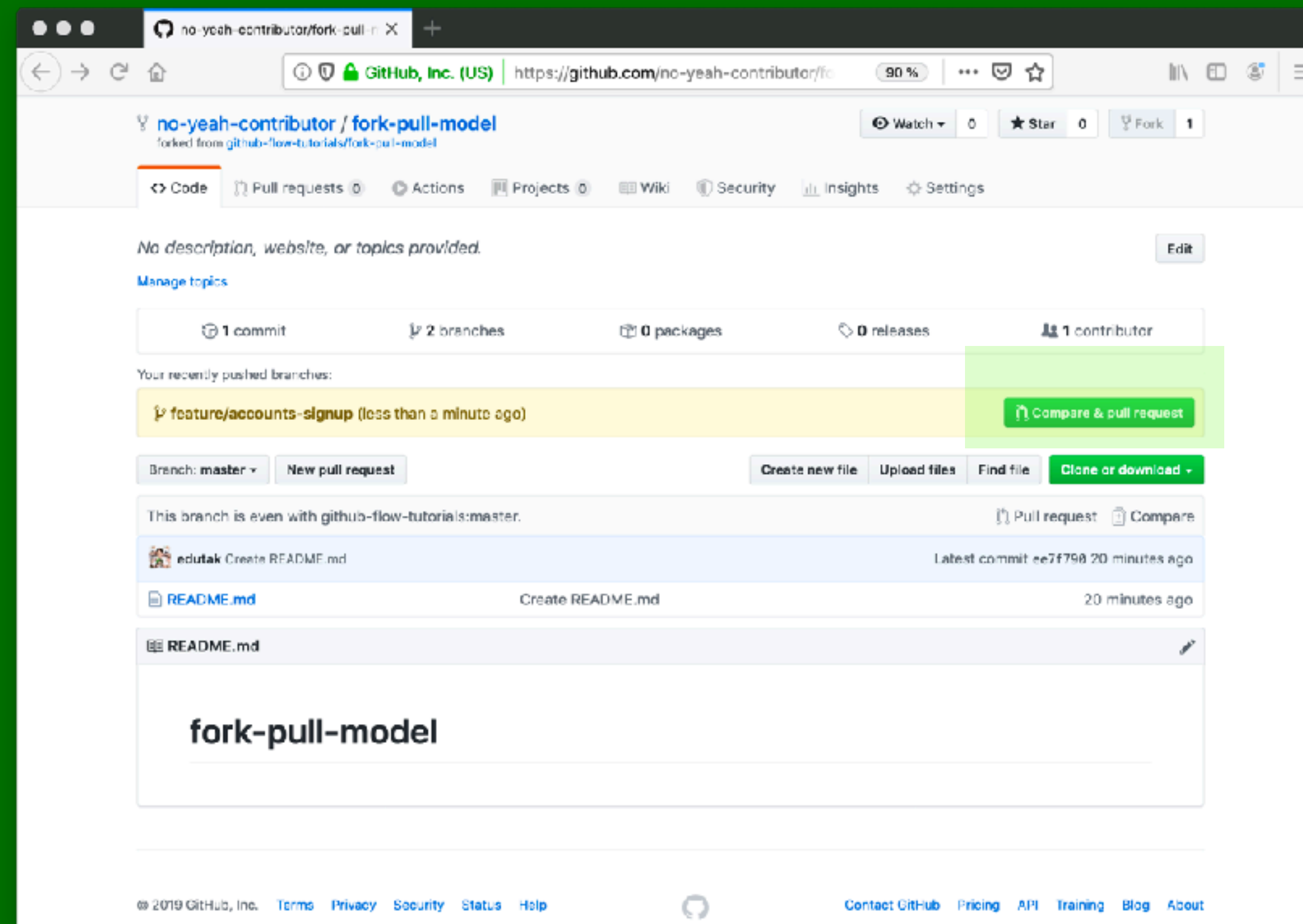
```
(feature/accounts-login) $ git push origin feature/accounts-login
```

* git push를 하기 이전에, 코드와 커밋 상태를 반드시 확인하자. (status, log)
원격 저장소에 공개된 이력은 절대 변경 하여서는 안된다.

step 3-1. Open a Pull Request

 \$ Github에 들어가서 Pull Request 버튼을 누른다.

* Github Flow에서 핵심은 Pull Request를 통한 협업이라고 할 수 있다.

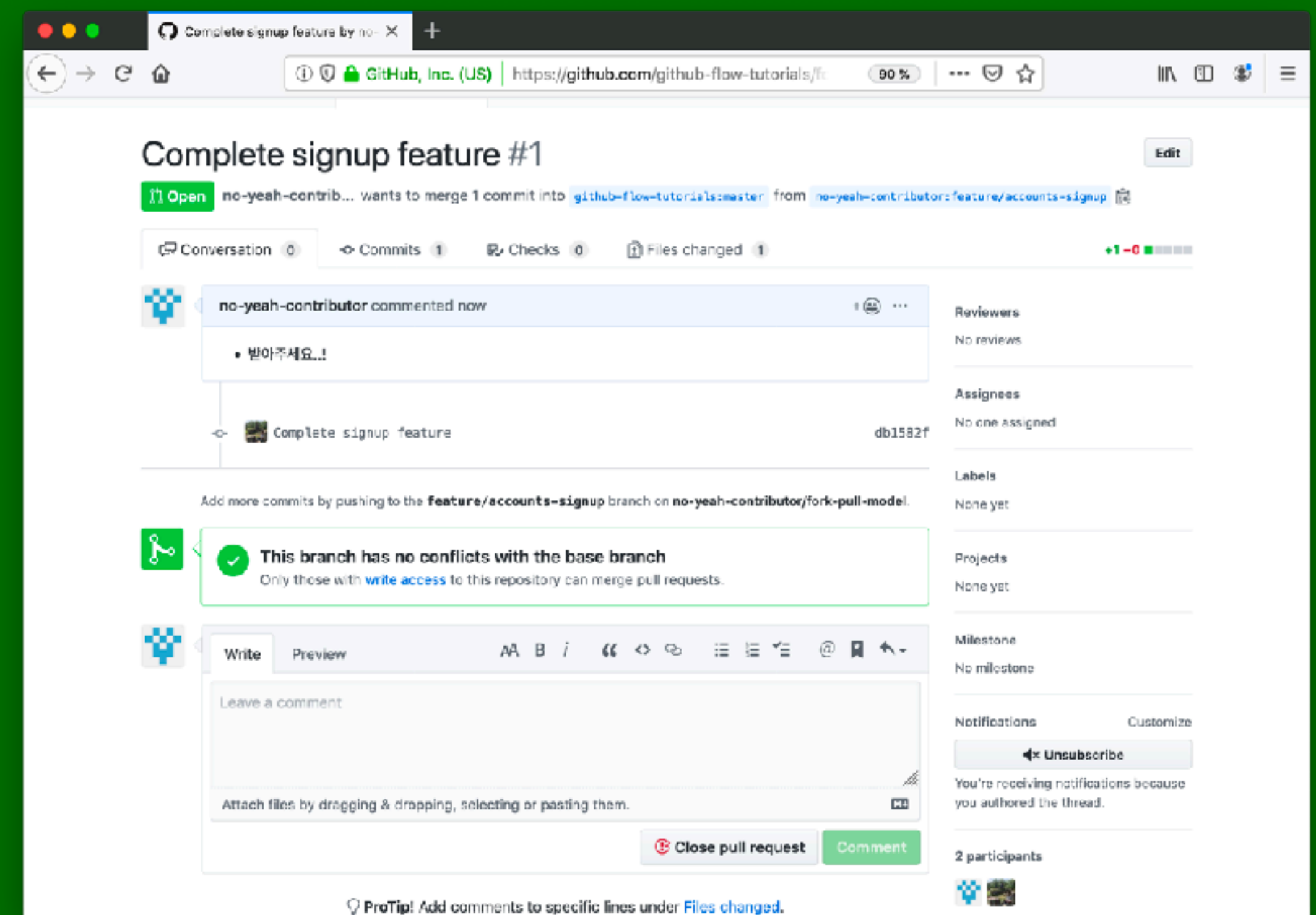
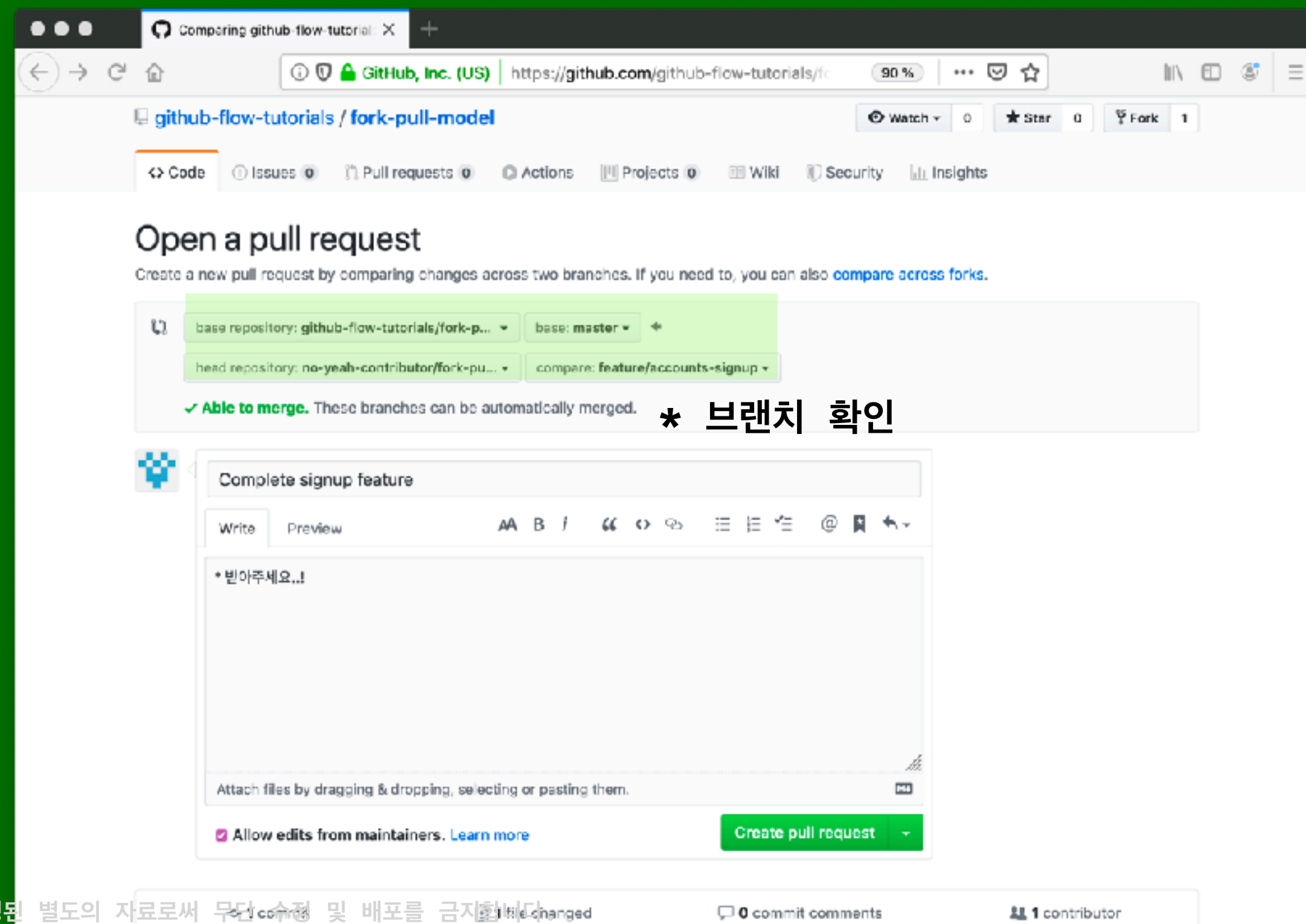


step 3-2. Create Pull Request



\$ PR과 관련된 설정을 진행한 후 요청을 생성한다.

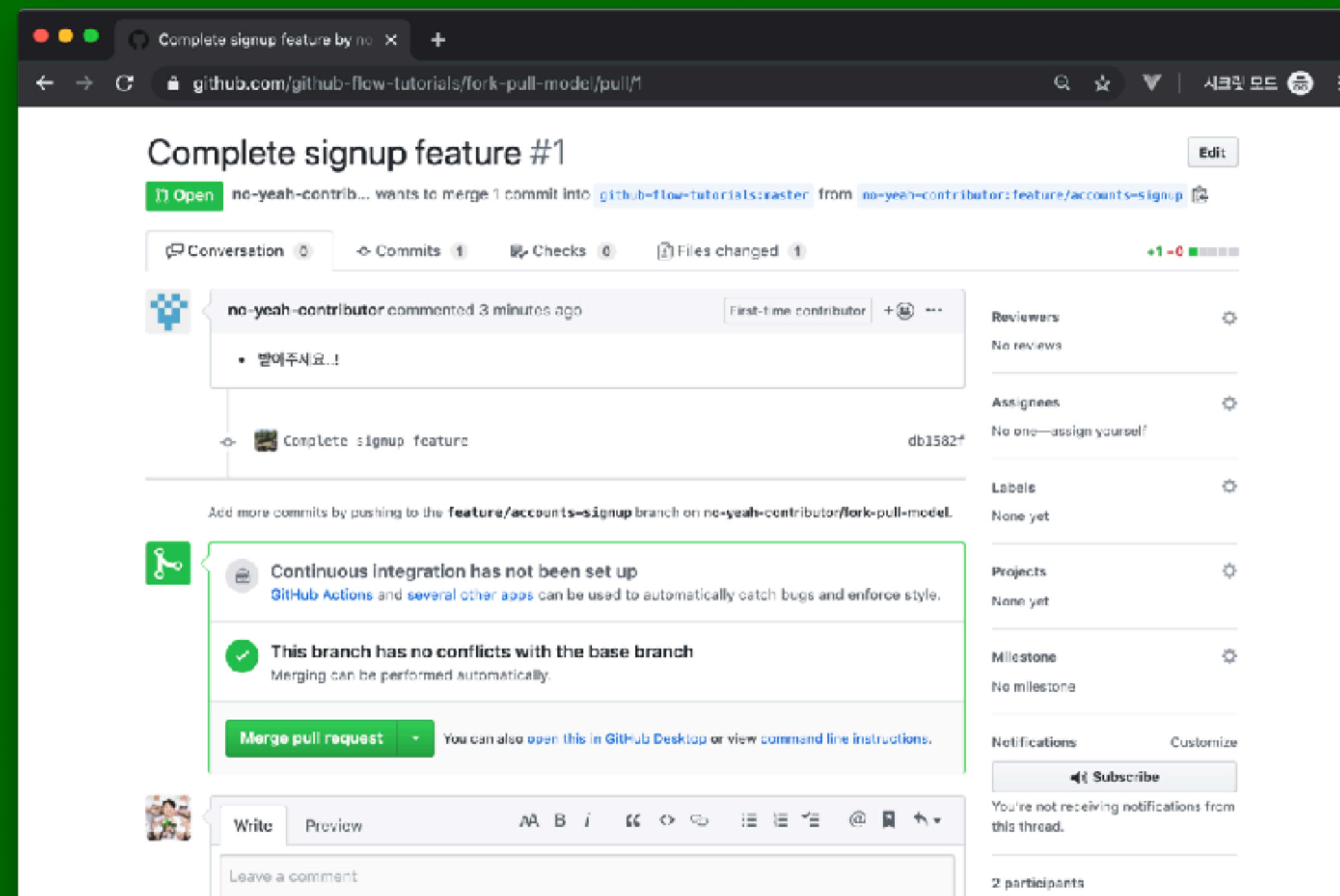
* 자동으로 fork 대상 저장소를 base repository로 설정하는 것을 확인할 수 있다.



step 3-3. Merge pull request

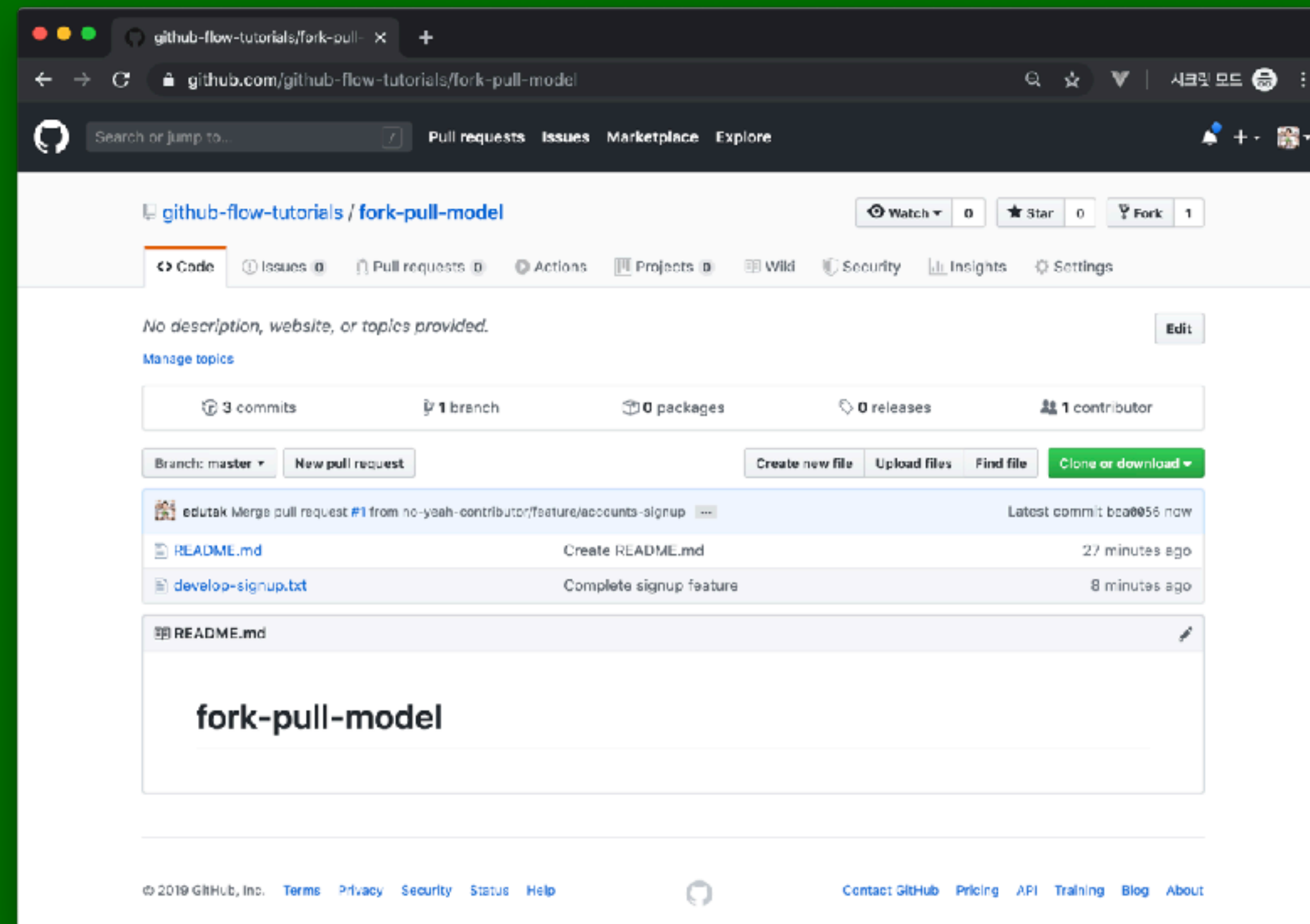
👩 \$ 작성된 코드를 확인 후 병합

- * 병합(merge) 과정에서 충돌이 발생할 경우 해결 후 병합을 진행한다.
- * 병합시 커밋 이력을 정리하기 위한 추가 옵션을 선택할 수도 있다. (squash : 커밋 병합, rebase)*
- * master branch로 병합의 경우 코드가 반드시 배포 가능한 상태여야 한다.



step 3-4. Merged !!

* 병합이 완료되면, master 브랜치에 반영된 것을 확인할 수 있다.



step 4. while True: but, upstream!!

 \$ 다음 작업 준비!

* 기존에는 단순히 새로 반영된 내용을 받아오기 위하여 origin으로 부터 pull을 받아왔지만, 지금 설정되어 있는 origin은 no-yeah-contributor의 저장소이다.

* 따라서, project 저장소를 새롭게 원격 저장소(**upstream**)로 등록하고 받아와야 한다.

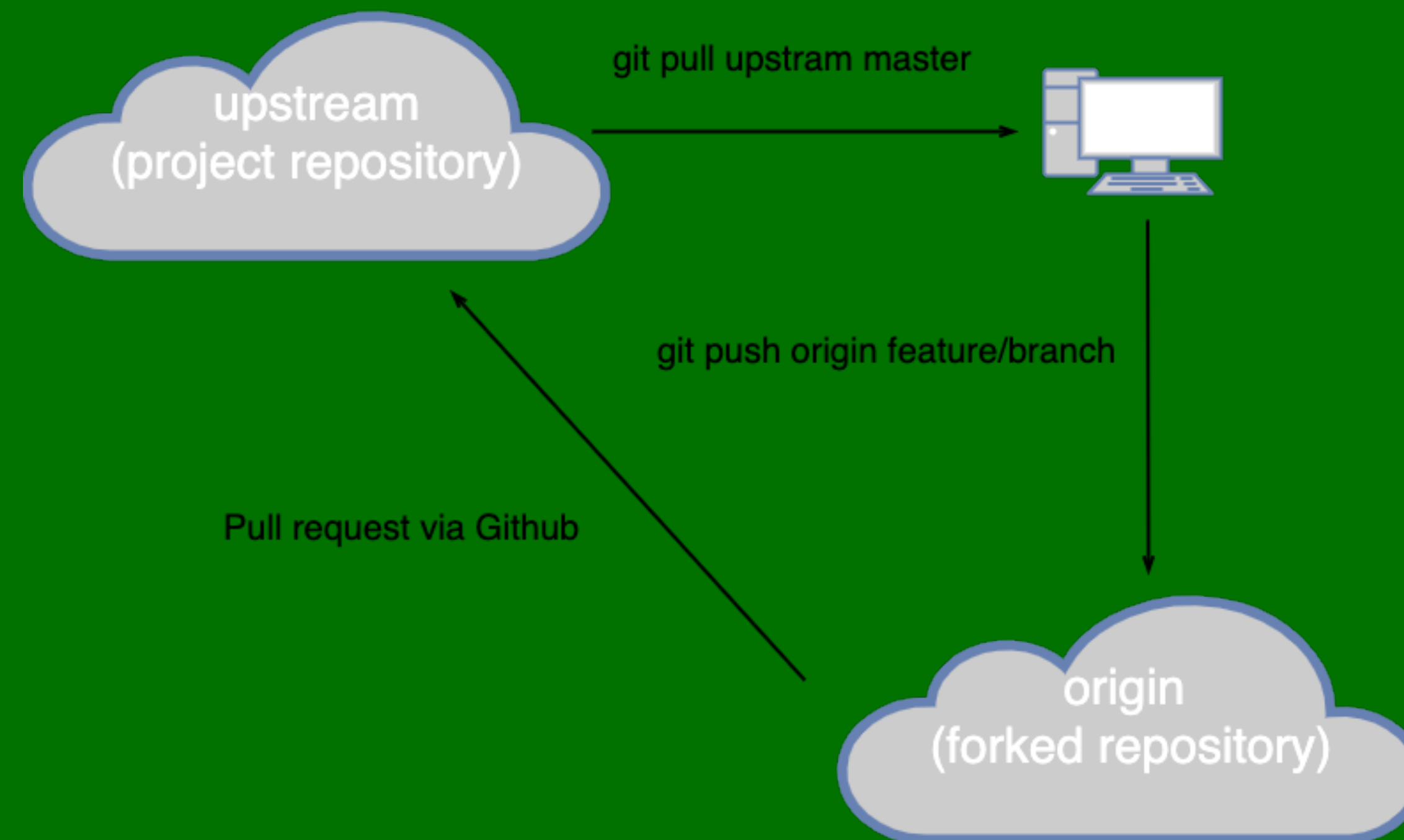
```
(feature/accounts-login) $ git checkout master
(master) $ git branch -d feature/accounts-login
(master) $ git remote -v
origin https://github.com/no-yeah-contributor/fork-pull-model.git(fetch)
origin https://github.com/no-yeah-contributor/fork-pull-model.git(push)
```

```
(master) $ git remote add upstream https://github.com/github-flow-
tutorials/fork-pull-model.git
(master) $ git pull upstream master
(master) $ git checkout -b feature/new-feature
(feature/new-feature) $
```

Fork & Pull Model

* 앞서 설명한 대로 Fork & Pull Model은 내가 직접적인 push 권한이 없다는 것이다!

* 즉, 오픈소스 프로젝트에 참여하기 위해서는 반드시 이 모델로 구성을 해야한다.



appendix. Review, Close..

* Pull request를 활용하면, 작업된 이력에 작성된 코드 라인별로 리뷰를 작성할 수도 있다.

* 물론, close가 될 수도 있다..!

The image displays two overlapping screenshots of a GitHub pull request interface. The left screenshot shows the pull request overview for 'Complete login feature #1', highlighting the changes made by 'no-yeah-contributor'. The right screenshot shows the pull request details, including a list of files changed and a comment from 'no-yeah-contributor'.

Complete login feature #1
no-yeah-contrib... wants to merge 1 commit into master from feature/accounts-login

Conversation 0 Commits 1 Checks 0 Files changed 1

Changes from all commits File filter... Jump to... ⚙

Complete login feature
feature/accounts-login (#1)

no-yeah-contributor committed 25 minutes ago commit 588946e5b701846...

2 develop Login.txt

... @@ -0,0 +1,2 @@

1 + Login!!

edutak Member Pending
음.. 변수명 검토 바랍니다.

Reply...

2 +

ProTip! Use **n** and **p** to navigate between commits in a pull request.

© 2019 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub

Complete login feature #1
no-yeah-contrib... wants to merge 1 commit into master from feature/accounts-login

Conversation 1 Commits 1 Checks 0 Files changed 1

no-yeah-contributor commented 25 minutes ago Collaborator

- Create accounts app
- Add User model based AbstractUser
- Add Login urls
- Add Bootstrap for Login form

Complete login feature 588946e

edutak reviewed 14 seconds ago Contributor

develop Login.txt

... @@ -0,0 +1,2 @@

1 + Login!!

edutak 14 seconds ago Contributor
음.. 변수명 검토 바랍니다.

no-yeah-contributor now Author Collaborator
죄송합니다...

Reply...

Resolve conversation

Lock conversation

Add more commits by pushing to the **feature/accounts-login** branch on **github-flow-tutorials/shared-repository**.

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.

Write Preview

Leave a comment

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

Close pull request Comment

ProTip! Add comments to specific lines under **Files changed**.

참고 문서

<https://git-scm.com/book/ko/v2>

<https://guides.github.com/>

<https://guides.github.com/introduction/flow/>

<https://guides.github.com/activities/hello-world/>

<https://guides.github.com/activities/forking/>

무한으로 즐길 수 있는 git tips

<https://github.com/mingrammer/git-tips> (한국어 번역)

프로젝트 시작전에 잊지 말아야 할 .gitignore

<https://github.com/github/gitignore/blob/master/Python.gitignore>

<https://github.com/github/gitignore/blob/master/Node.gitignore>