Open Source Version 관리 Git

구명완교수

Office: K관 239호

Email: mwkoo9@gmail.com

Reference

- 참고 자료

Git 메뉴얼 (한글)

http://git-scm.com/book/ko

Learn Git Branching

http://pcottle.github.io/learnGitBranching/

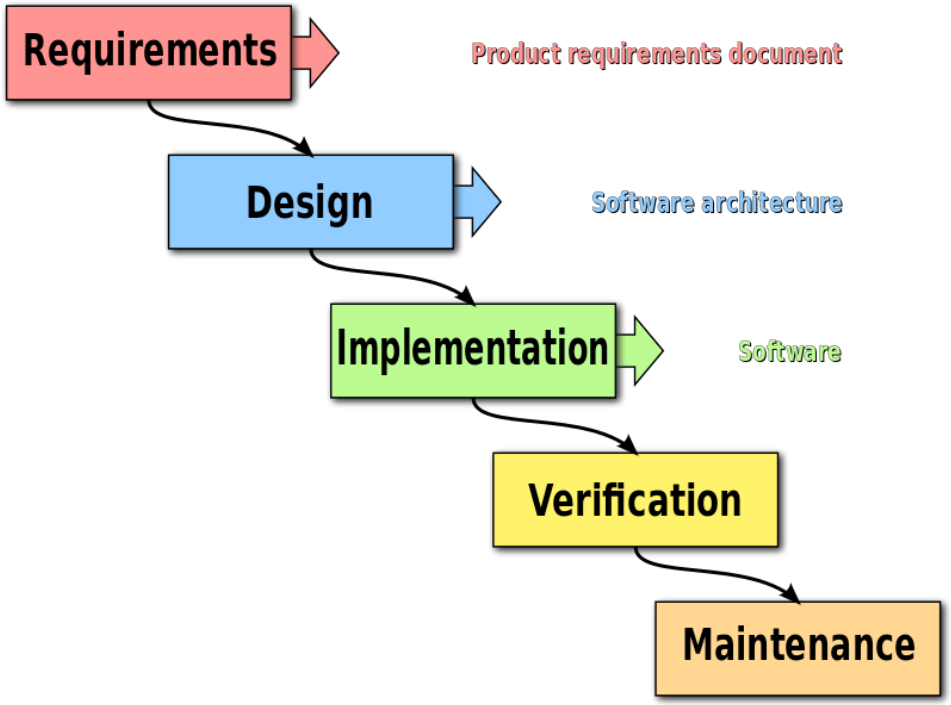
Code School - Try git

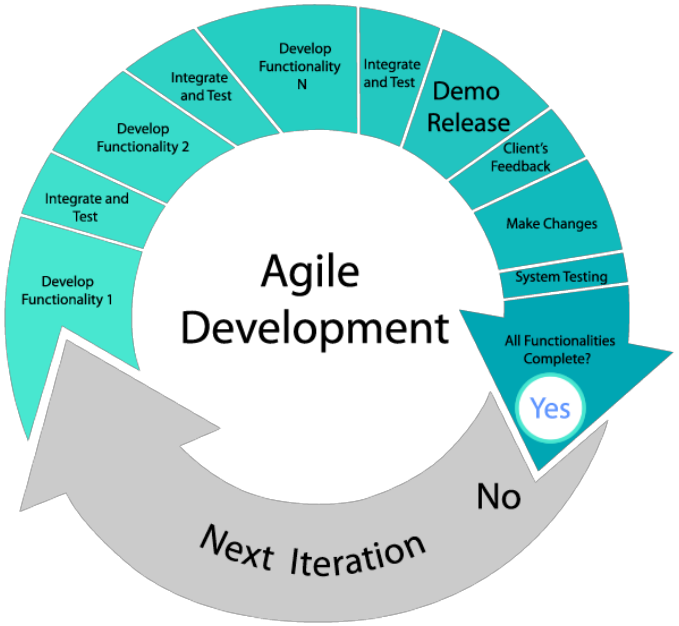
https://try.github.io/levels/1/challenges/1

Atlassian git tutorial

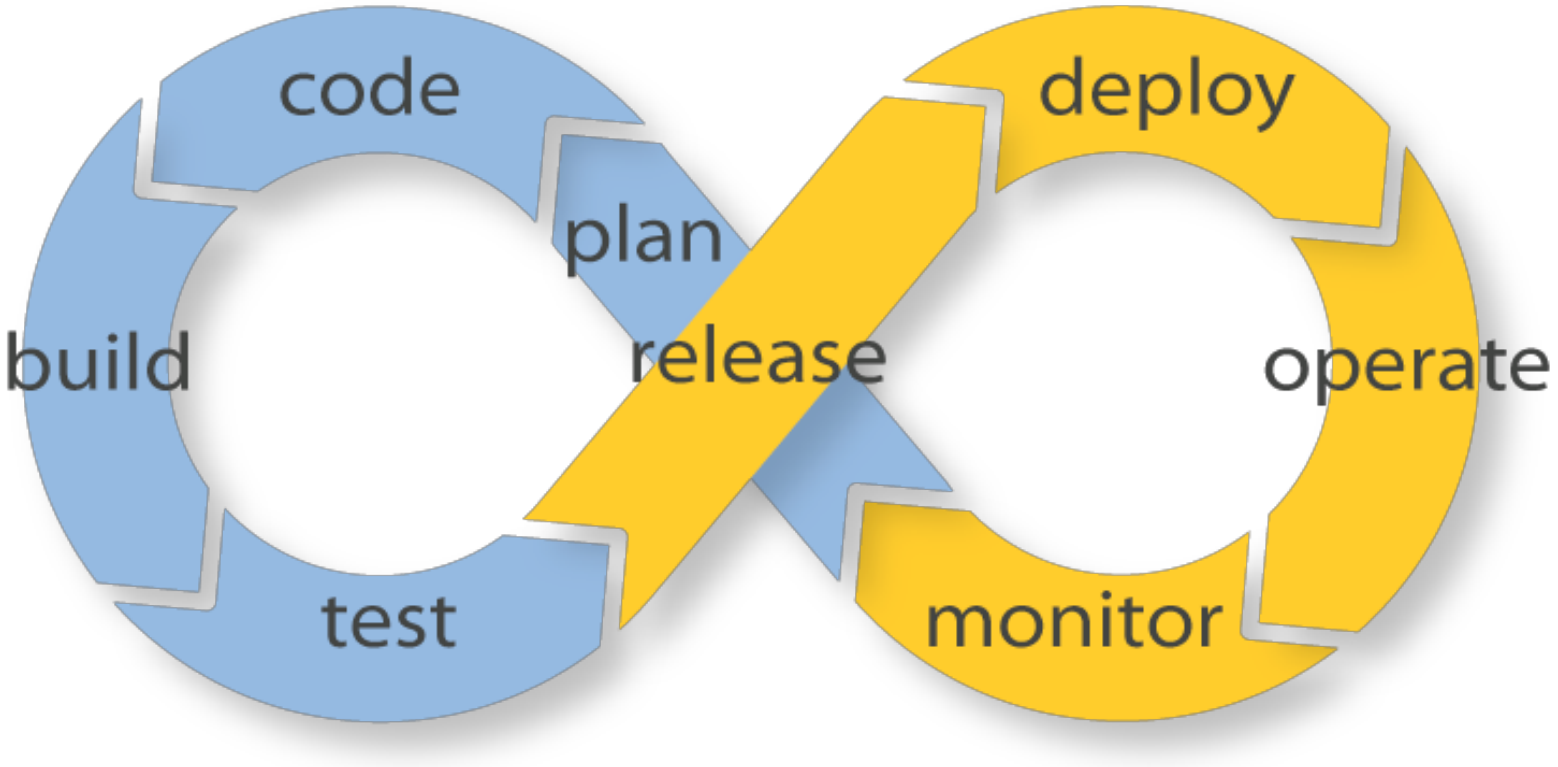
https://www.atlassian.com/git/tutorial

Waterfall vs. Agile

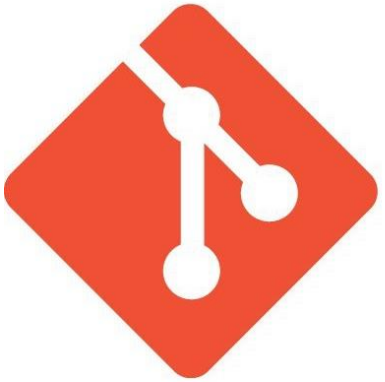




DevOps



Git & GitHub



git

GitHub

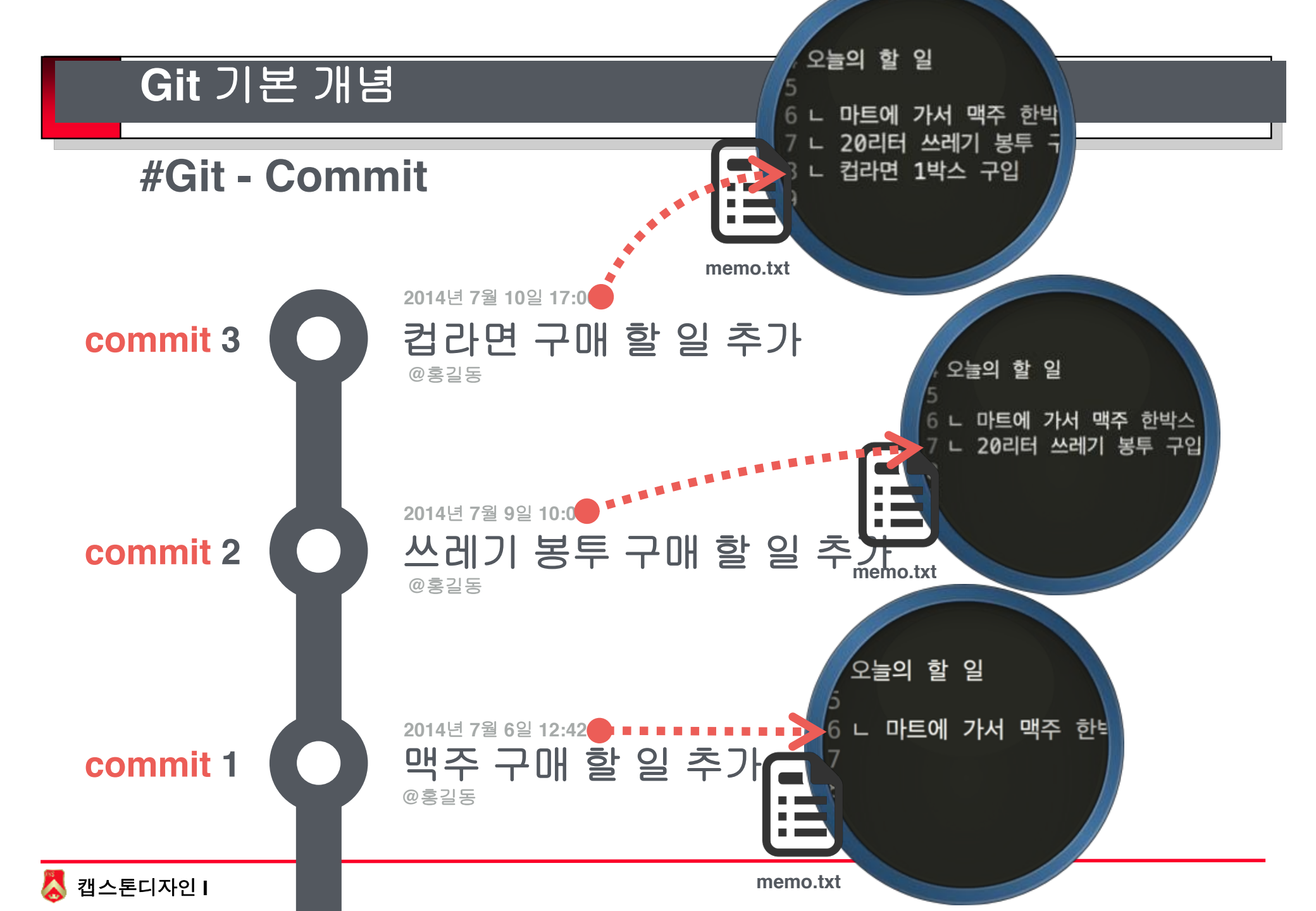
Git 기본 개념

- **Memo.txt 변경**

"컵라면 1박스 구입" 이라는 수정이 발생

Git 기본 개념

#Git - Commit



Git 기본 개념

#Git - Commit [File Tree]

commit 3

2014년 7월 10일 17:00

수,목,금 교육 시간 수정

@홍길동

수요일.txt 목요일.txt 금요일.txt

commit 2

2014년 7월 9일 10:03

수요일 학습 내용 추가

@홍길동

수요일.txt

commit 1

2014년 7월 6일 12:42

월,화 자율 학습 일정 추가

@홍길동

월요일.txt 화요일.txt

Git 기본 개념

#Git - Branch

commit 2

2014년 7월 9일 10:03

수요일 학습 내용 추가

@홍길동

수요일.txt

commit 1

2014년 7월 6일 12:42

월,화 자율 학습 일정 추가

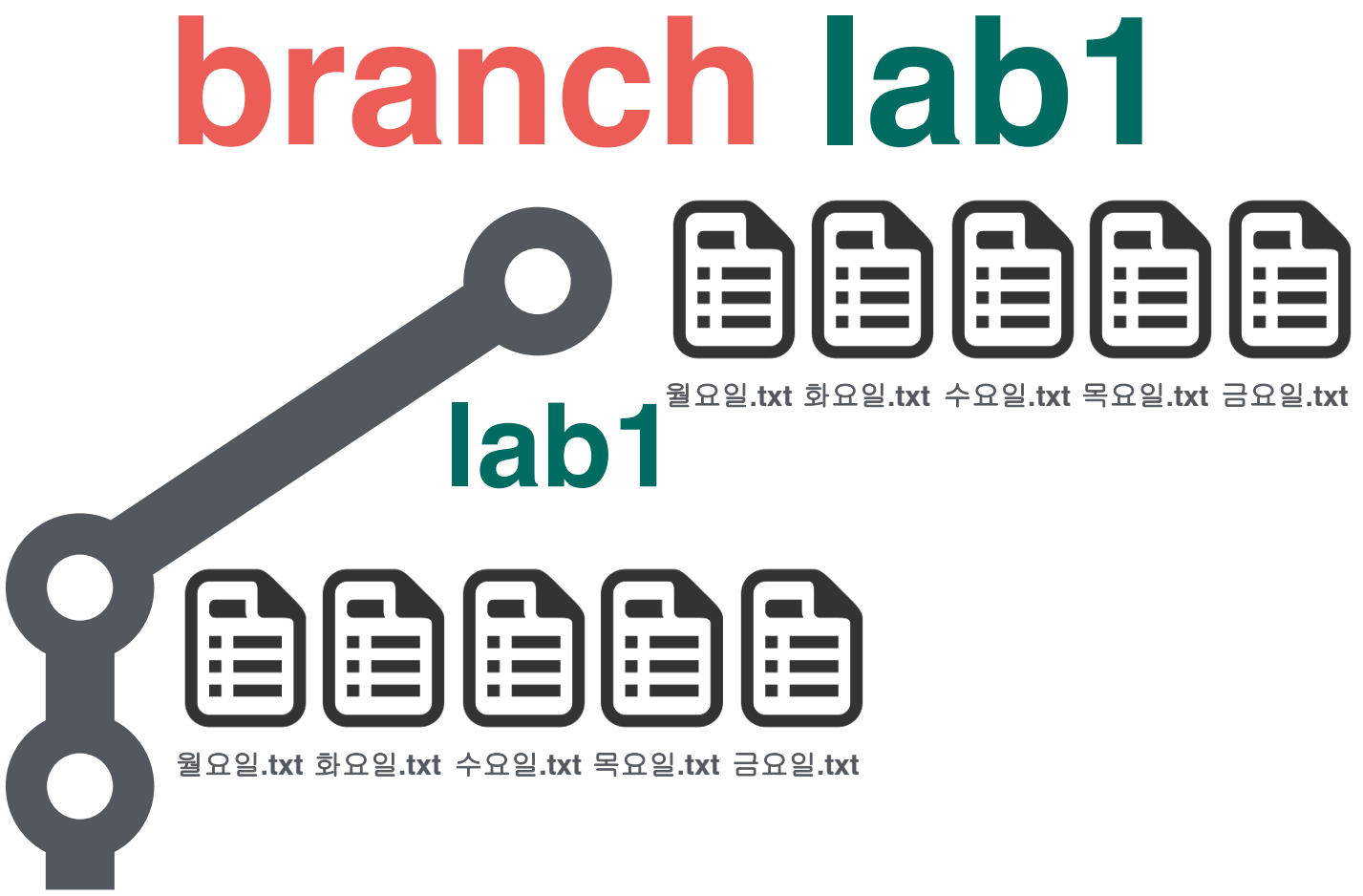
@홍길동

월요일.txt 화요일.txt

master

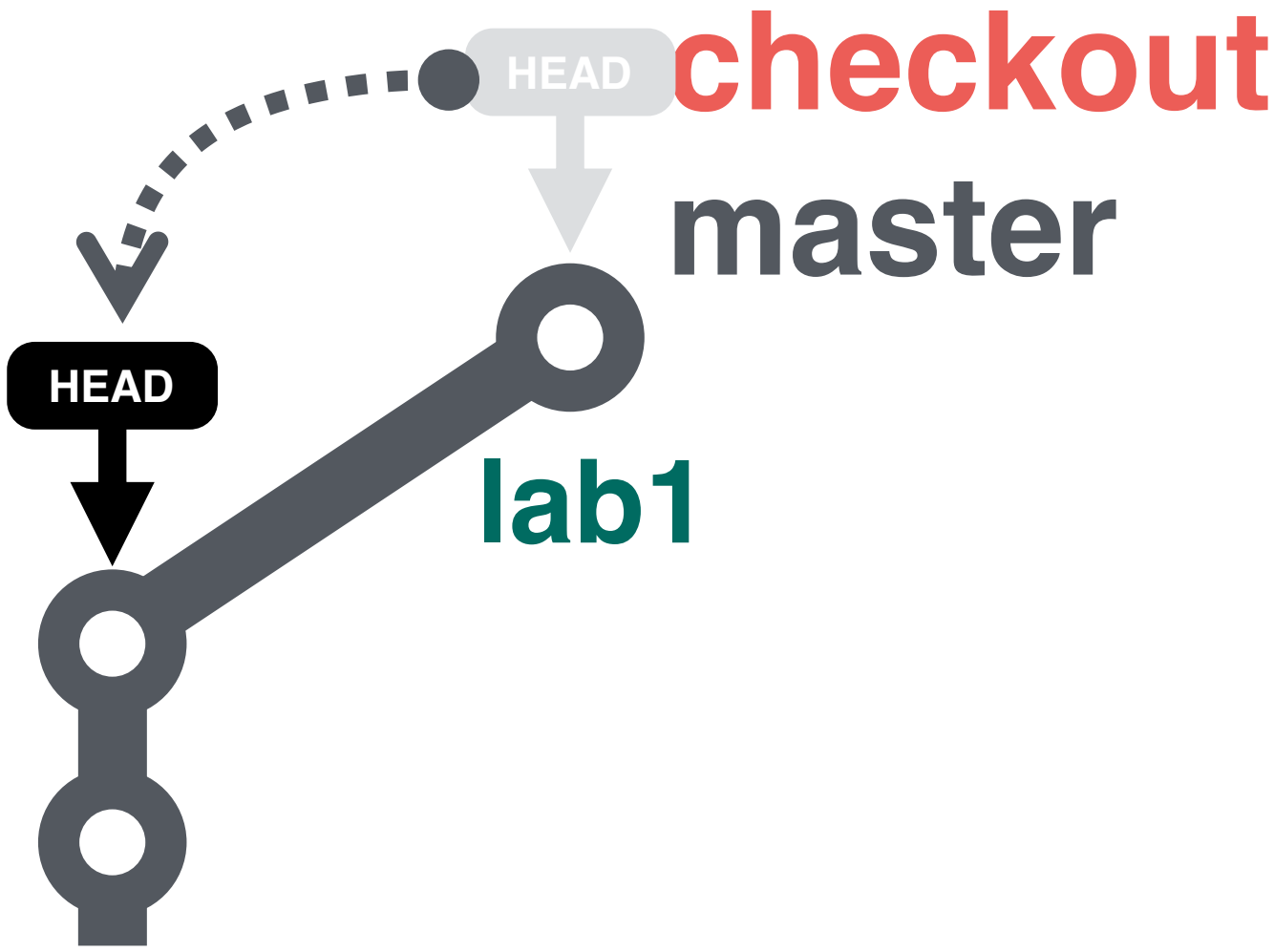
Git 기본개념

#Git - Branch



Git 기본 개념

#Git - Checkout



#Git - Delete Branch

실험 실패

lab1에서 진행했던
실험이 예상과 달리
필요없는 작업이 되었다.
어떻게 하면 될까?

master로 이동 후 **lab1**
브랜치를 삭제한다. **lab1**의
모든 기록이 제거된다. 기록
보관 차원에서 삭제하지 않아도
아무 문제 없다

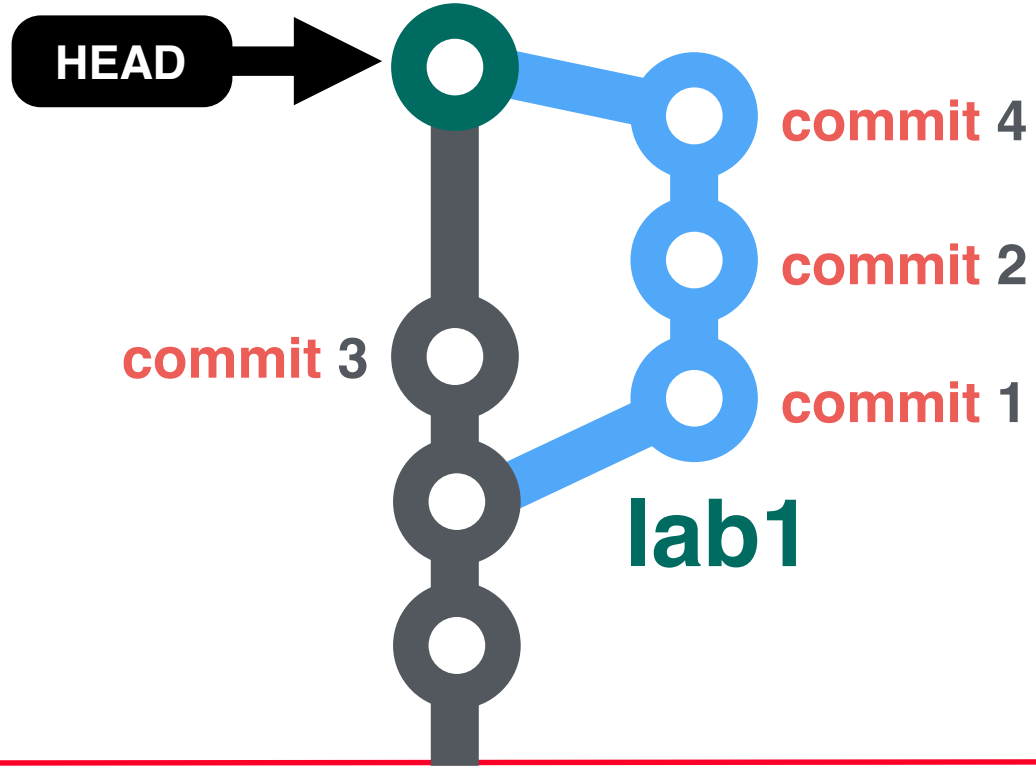
#Git - Merge

실험 성공

lab1에서 진행했던 실험이 성공적으로 끝났다. 실험의 결과를 **master** 브랜치에 옮기려한다. 어떻게 하면 될까?

lab1 브랜치의 내용을 마스터 브랜치와 병합(merge) 한다.

#Git - Merge



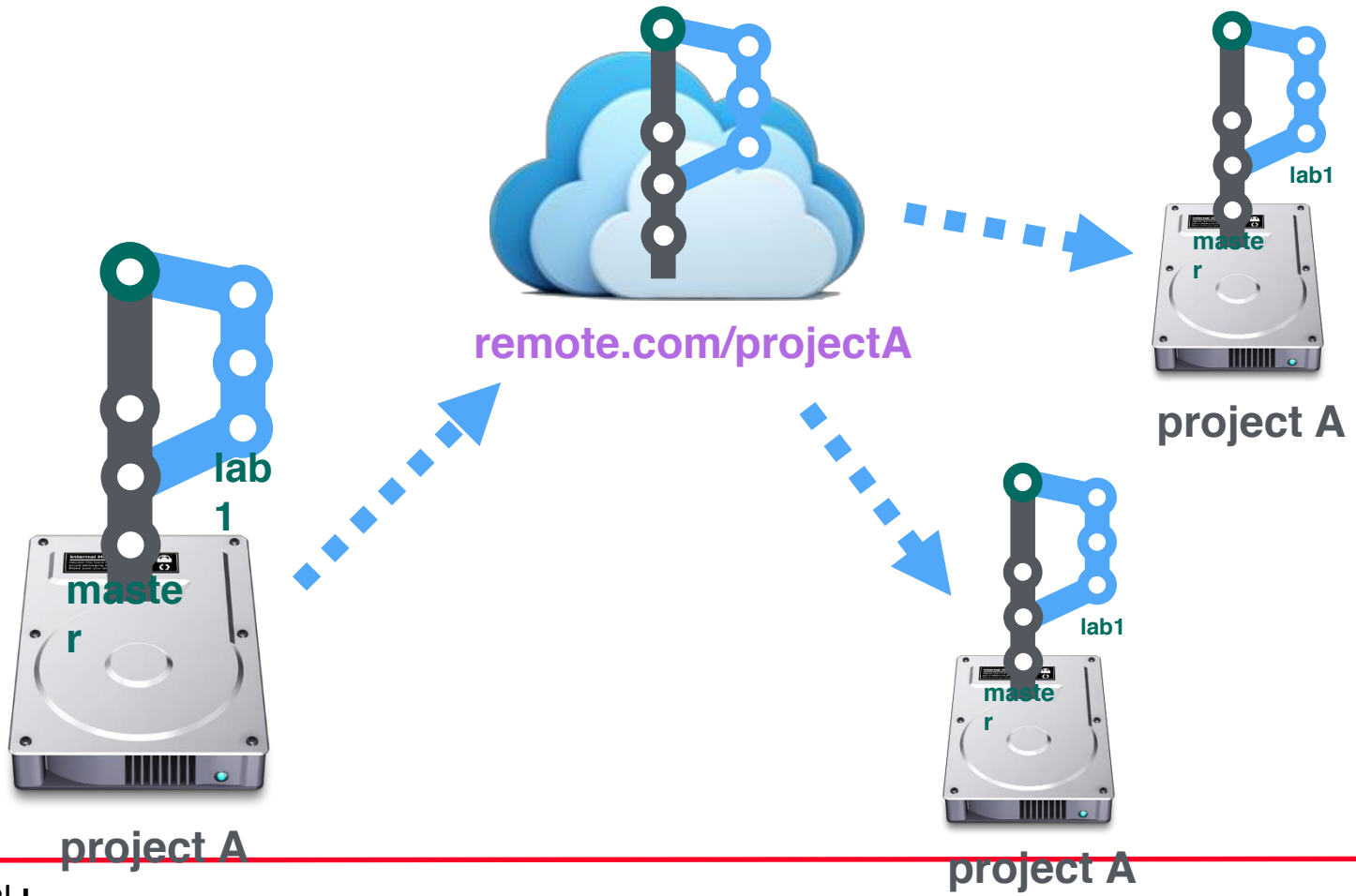
병합 결과

master 브랜치에 **lab1** 브랜치를 병합하면 **git** 은 **lab1** 브랜치의 내용과 **master** 브랜치의 **commit 3** 의 내용을 포함하여 두 브랜치를 병합한다

변경 내용에 따라 파일 내용이 변경되고 때론 파일이 삭제될 수 도 있으며 추가될 수 도 있다.

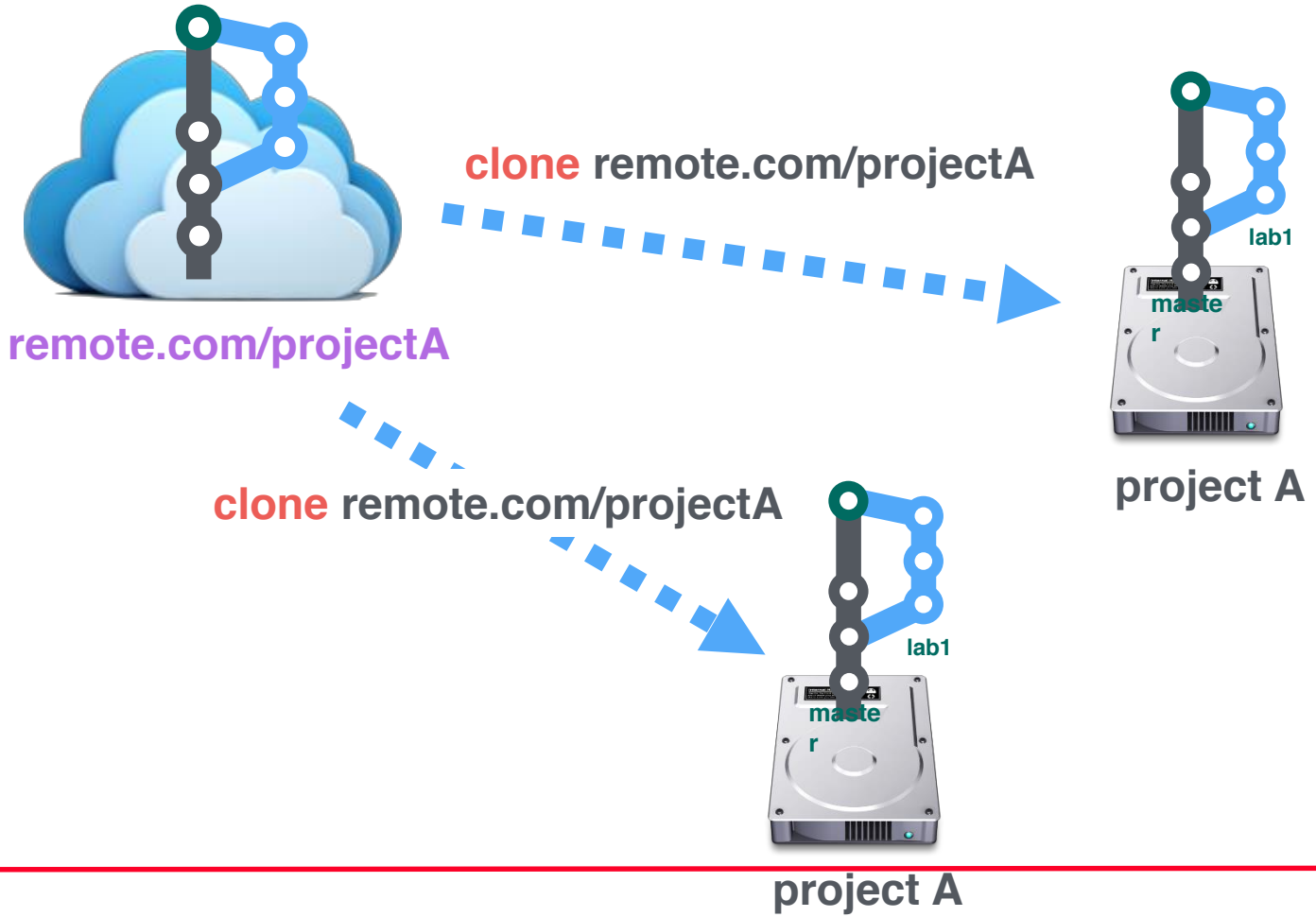
GitHub 기본 개념

#Git - Remote Repo



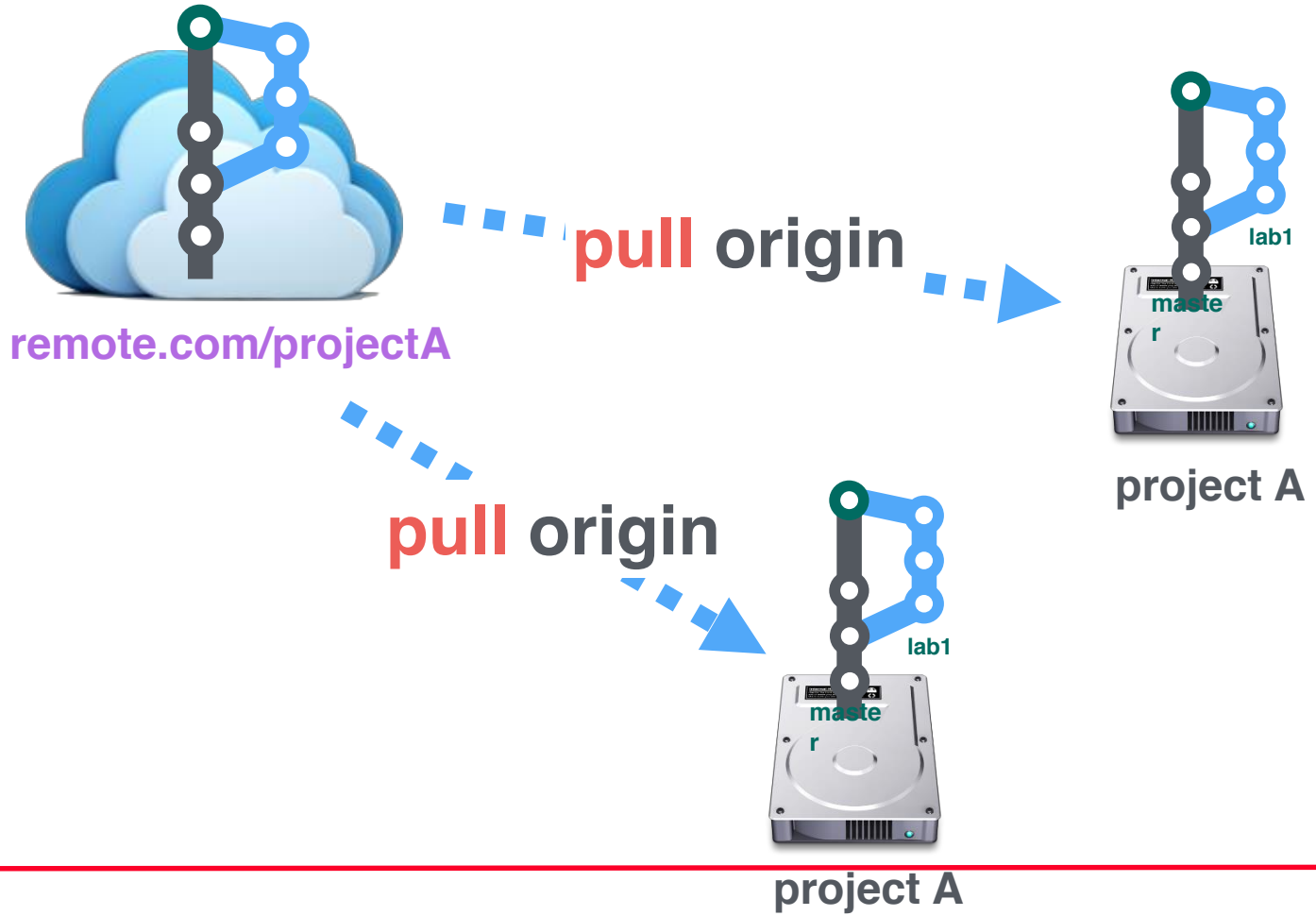
GitHub 기본 개념

#Git - Clone



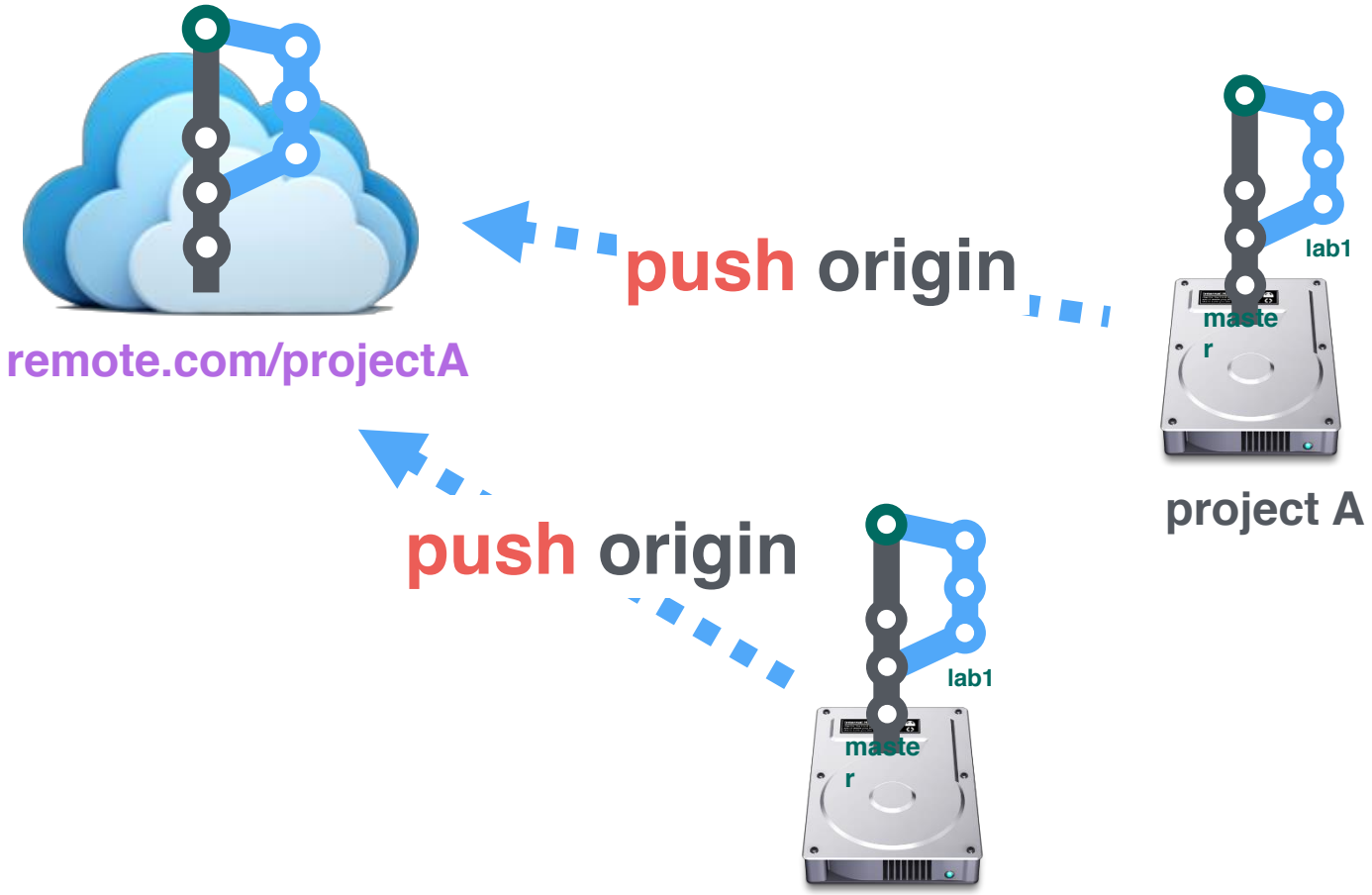
GitHub 기본 개념

#Git - Pull



GitHub 기본개념

#Git - Push

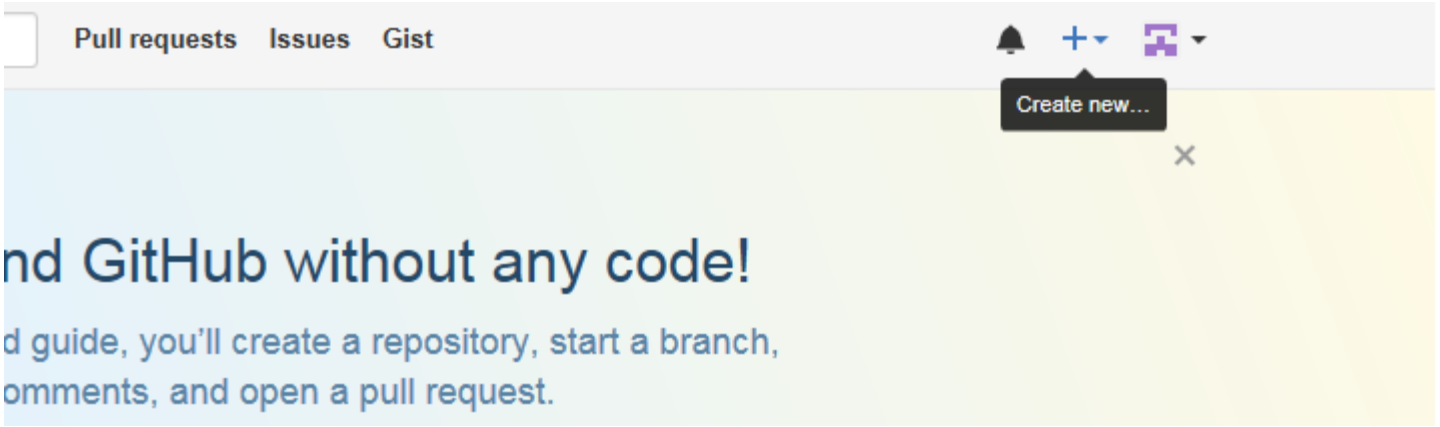


GitHub

- GitHub is a platform for hosting and collaborating on projects
  - a collaborative and asynchronous workflow for building software better, together
- Hello World project
  - a time-honored tradition in computer programming
- Repositories, Branches, Commits, Issues and Pull Requests.
- Install & Code Free Zone
  - need a GitHub account

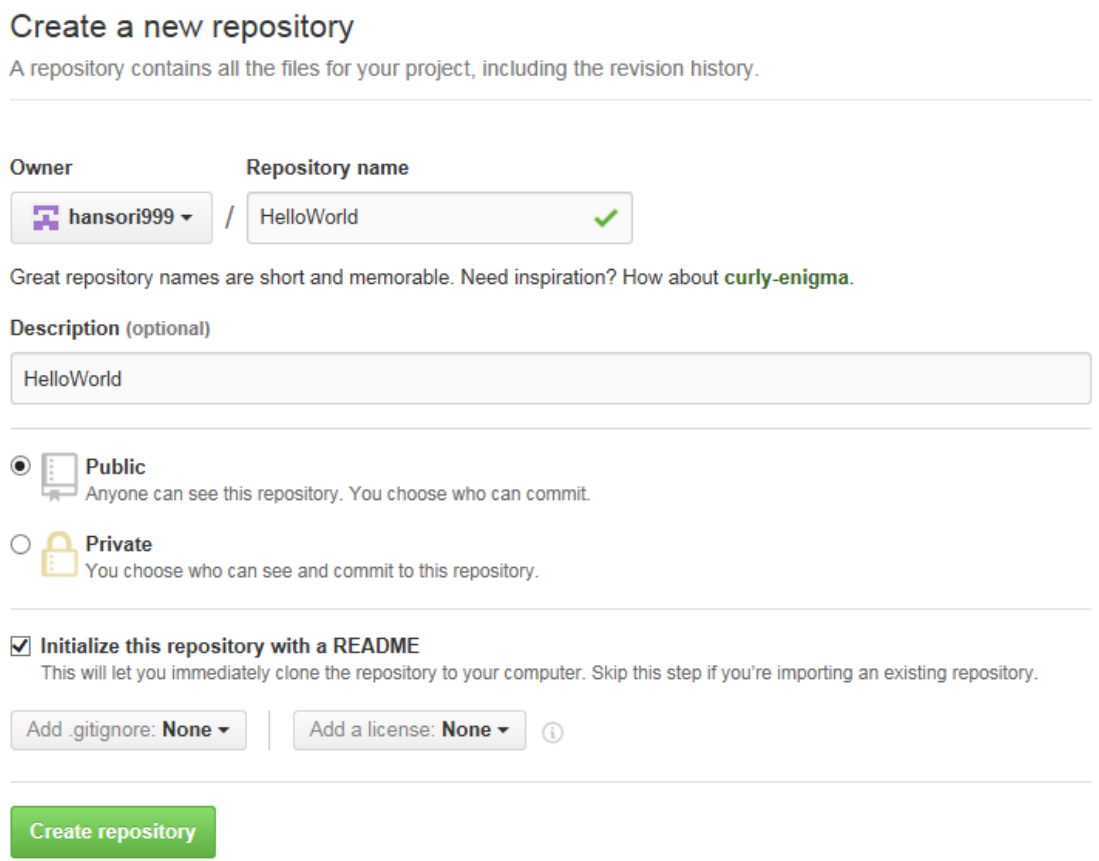
GitHub

- Create a Repository
  - repository is the basic unit of GitHub
  - contain folders and files, including images
  - recommend including a README
  - also offers other common options such as a license file
- To create a new repository
  - 1.Click the + icon next to your username, top-right



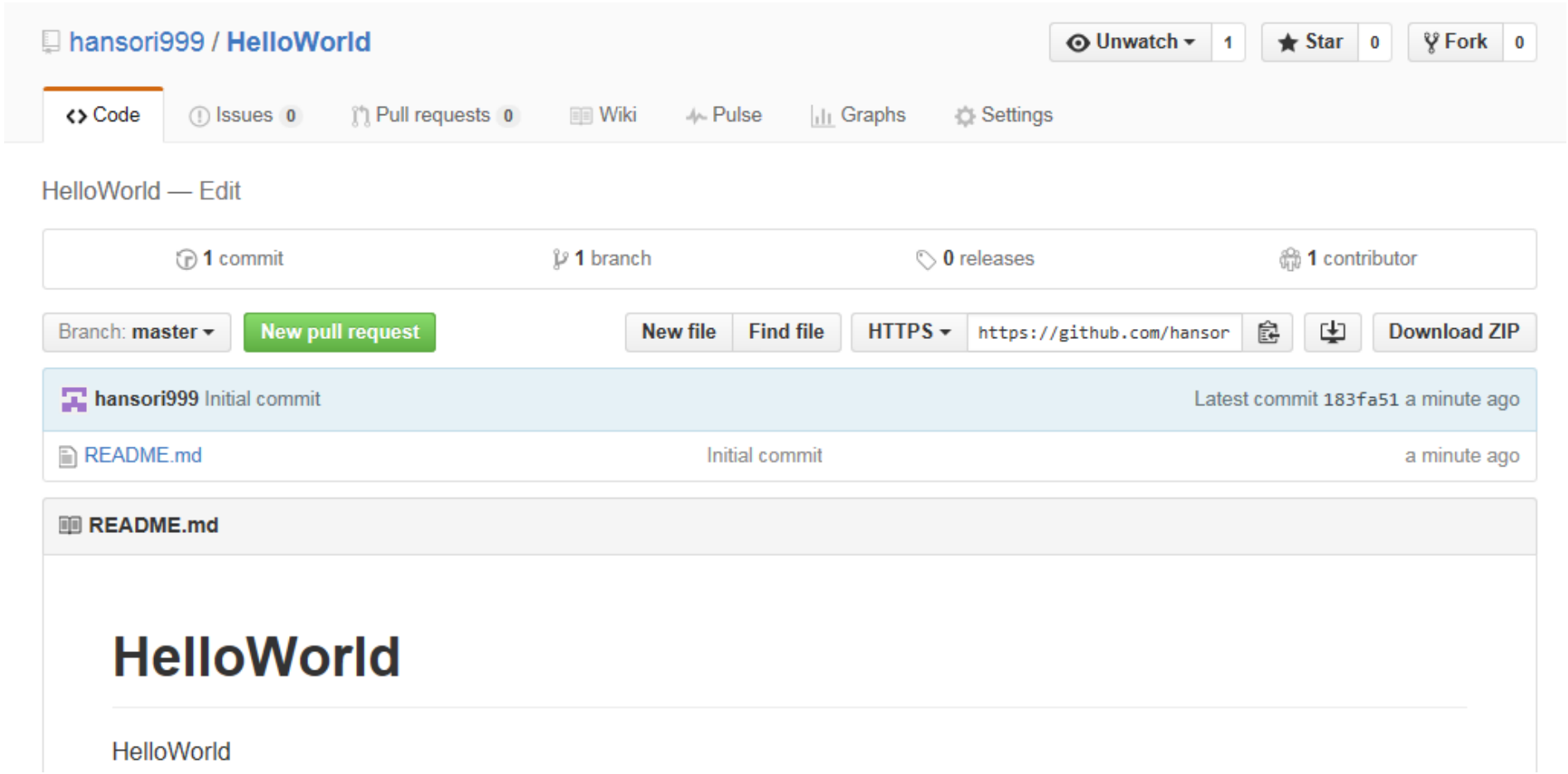
GitHub

- Name your repository HelloWorld
- Write a short description.
- Select Initialize this repository with a README



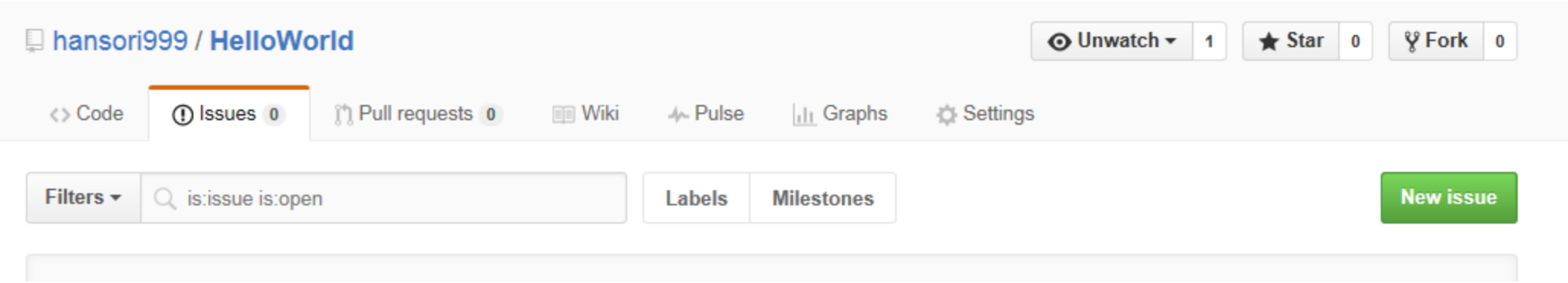
GitHub

- **repository is created**



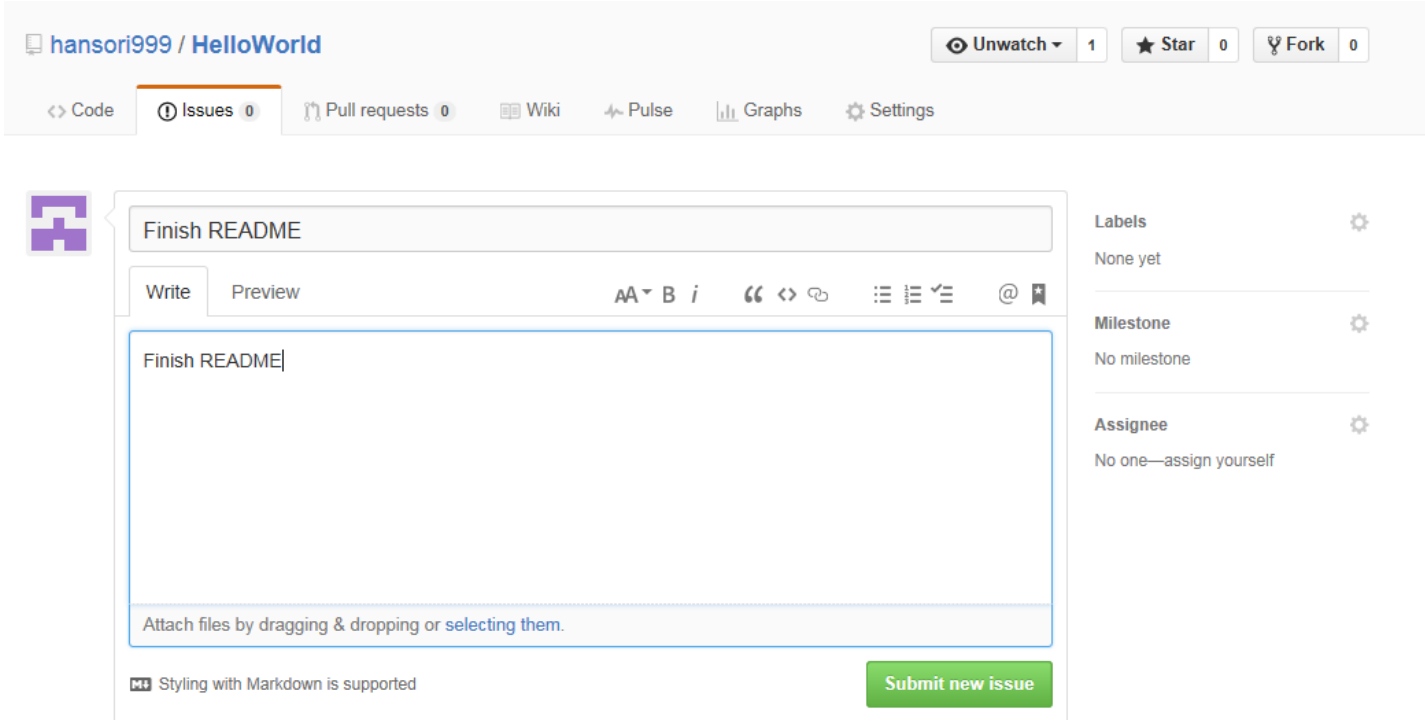
GitHub

- **Open an Issue**
  - **Issue is a note on a repository about something that needs attention**
    - **a bug, a feature request, a question or lots of other things**
    - **label, search and assign issues,**
      - making managing an active project easier
- **Open an Issue for README edits**
  - **1.Click the ! Issues tab from the sidebar**



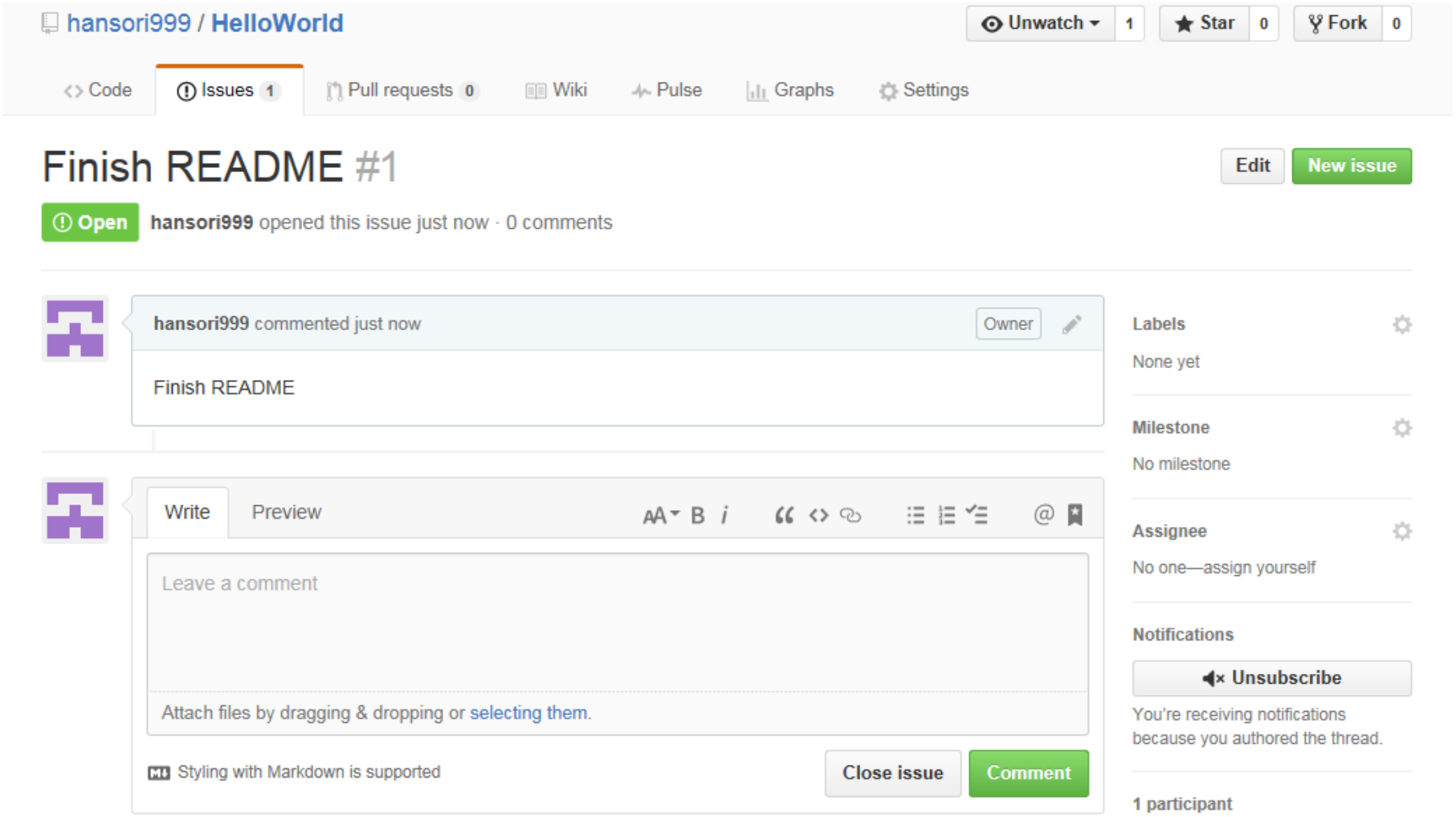
GitHub

- Click New Issue
- Give your Issue a title and description



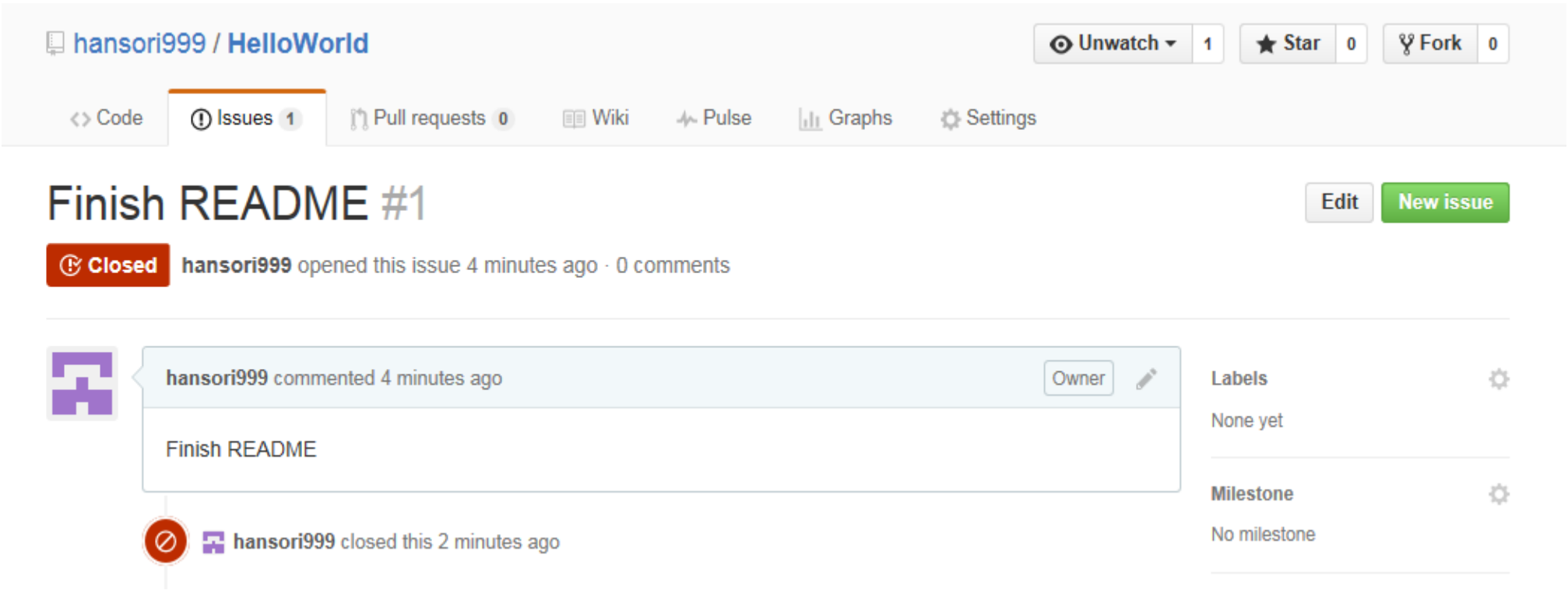
- Click Submit new Issue when you're done
  - a permanent home (URL) that you can reference

GitHub


hansori999 / HelloWorld
Unwatch 1
Star 0
Fork 0
Code
Issues 1
Pull requests 0
Wiki
Pulse
Graphs
Settings
Finish README #1
Edit
New issue
Open
hansori999 opened this issue just now · 0 comments
hansori999 commented just now
Owner
Finish README
Write
Preview
Leave a comment
Attach files by dragging & dropping or selecting them.
Styling with Markdown is supported
Close issue
Comment
Labels
None yet
Milestone
No milestone
Assignee
No one—assign yourself
Notifications
Unsubscribe
You're receiving notifications because you authored the thread.
1 participant

GitHub

- Next, work towards editing your README
- closing this issue

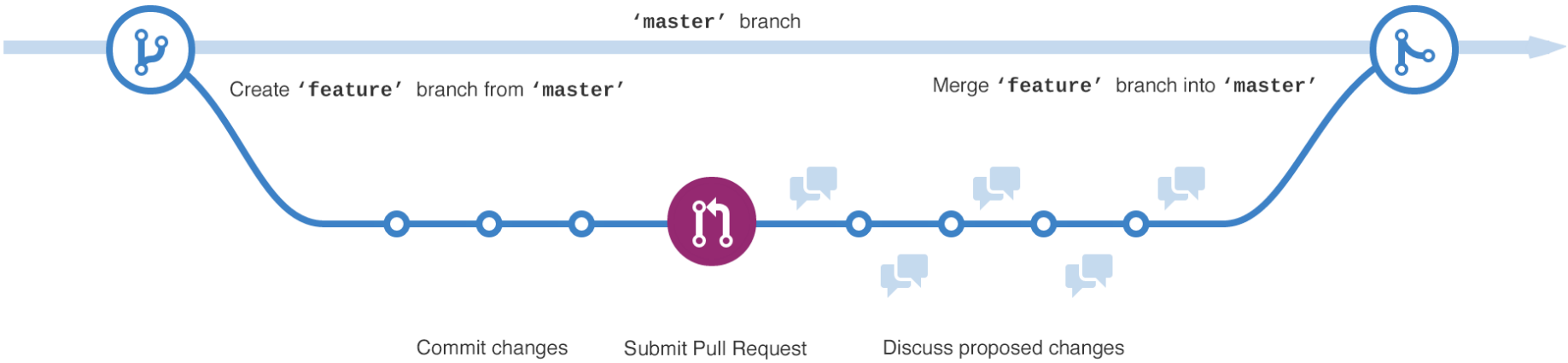


GitHub

- **Create a Branch**
  - **the way to work on different parts of a repository at one time**
  - **by default it has one branch "master" after creating repository**
    - **keep working on this branch and have only one**
  - **if you have another feature or idea you want to work on**
    - **can create another branch, starting from master, so that you can leave master in its working state**
    - **making a copy of the original branch like a photo snapshot**
  - **use branches for keeping bug fixes and feature work separate from our master (production) branch**
    - **When a feature or fix is ready,**
      - the branch is merged into master.

GitHub

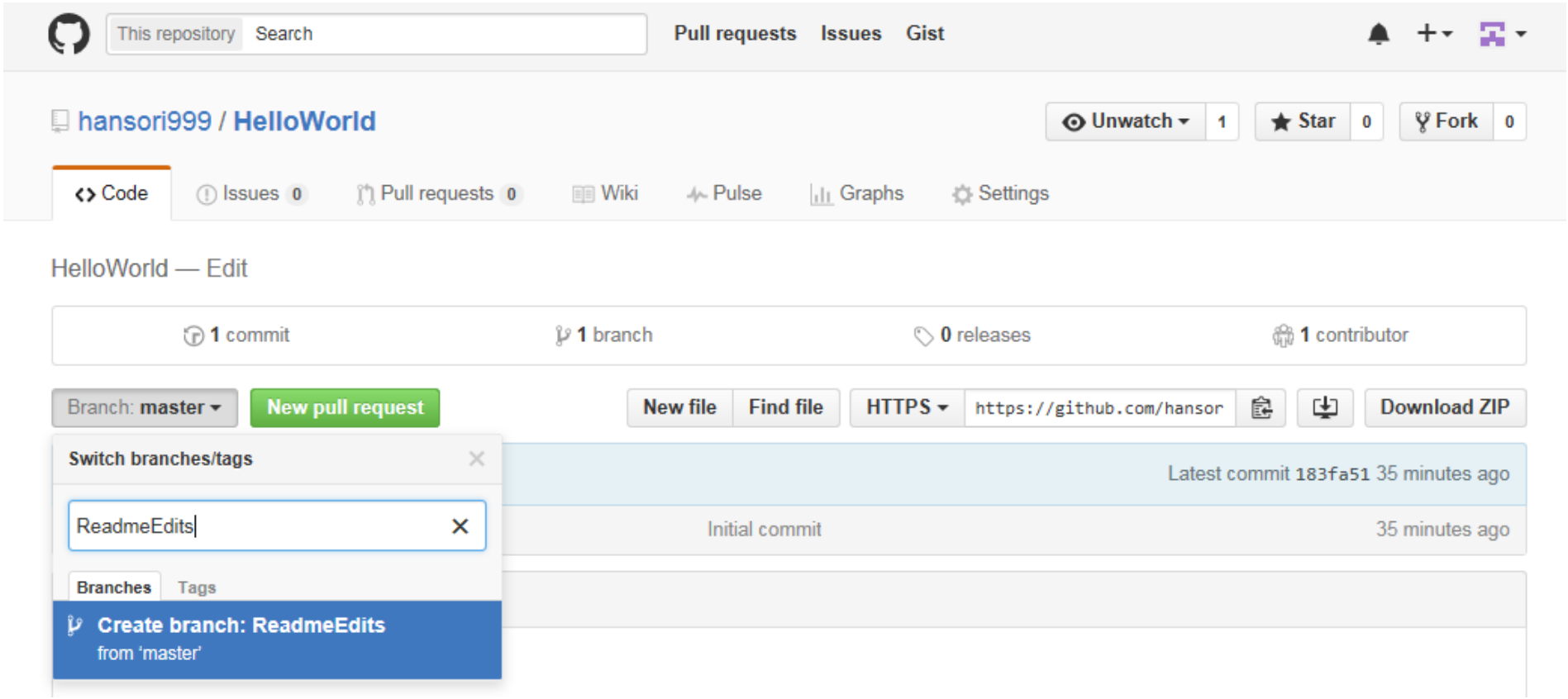
- **To create a new branch**



- **Go to your new repository HelloWorld**
- **Click the drop down at the top of the file list that says branch: master**

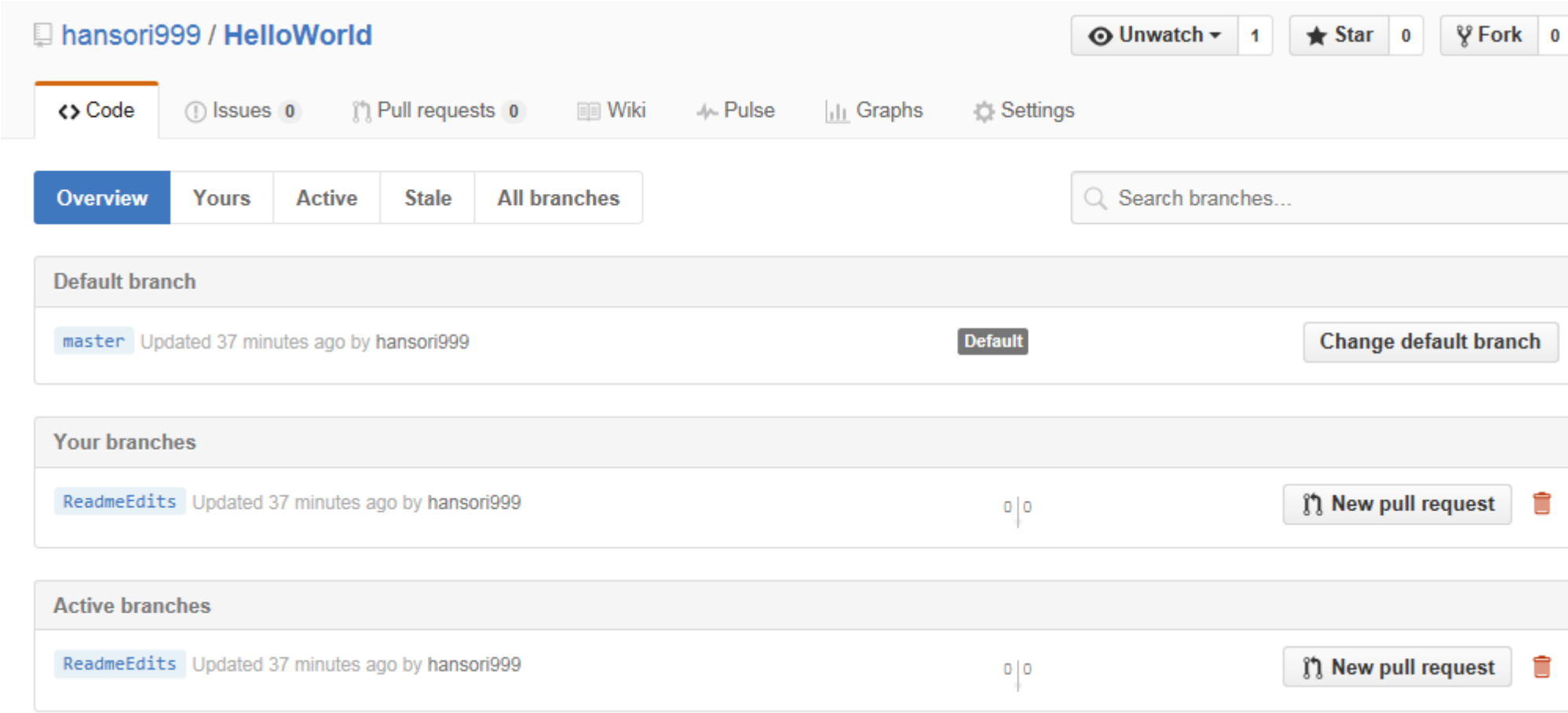
GitHub

- Type a branch name, ReadmeEdits, into the new branch text box
- Select the blue Create branch box or hit “Enter” on your keyboard



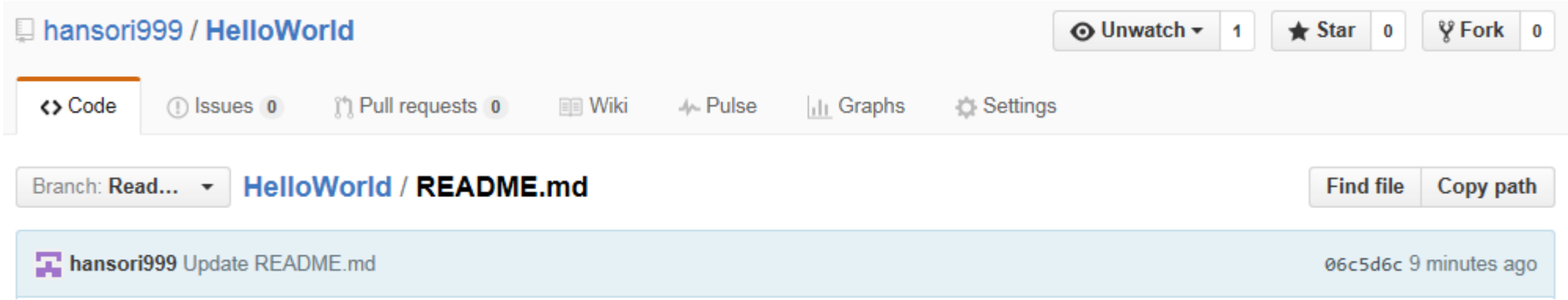
GitHub

- **Now you have two branches, master and ReadmeEdits**



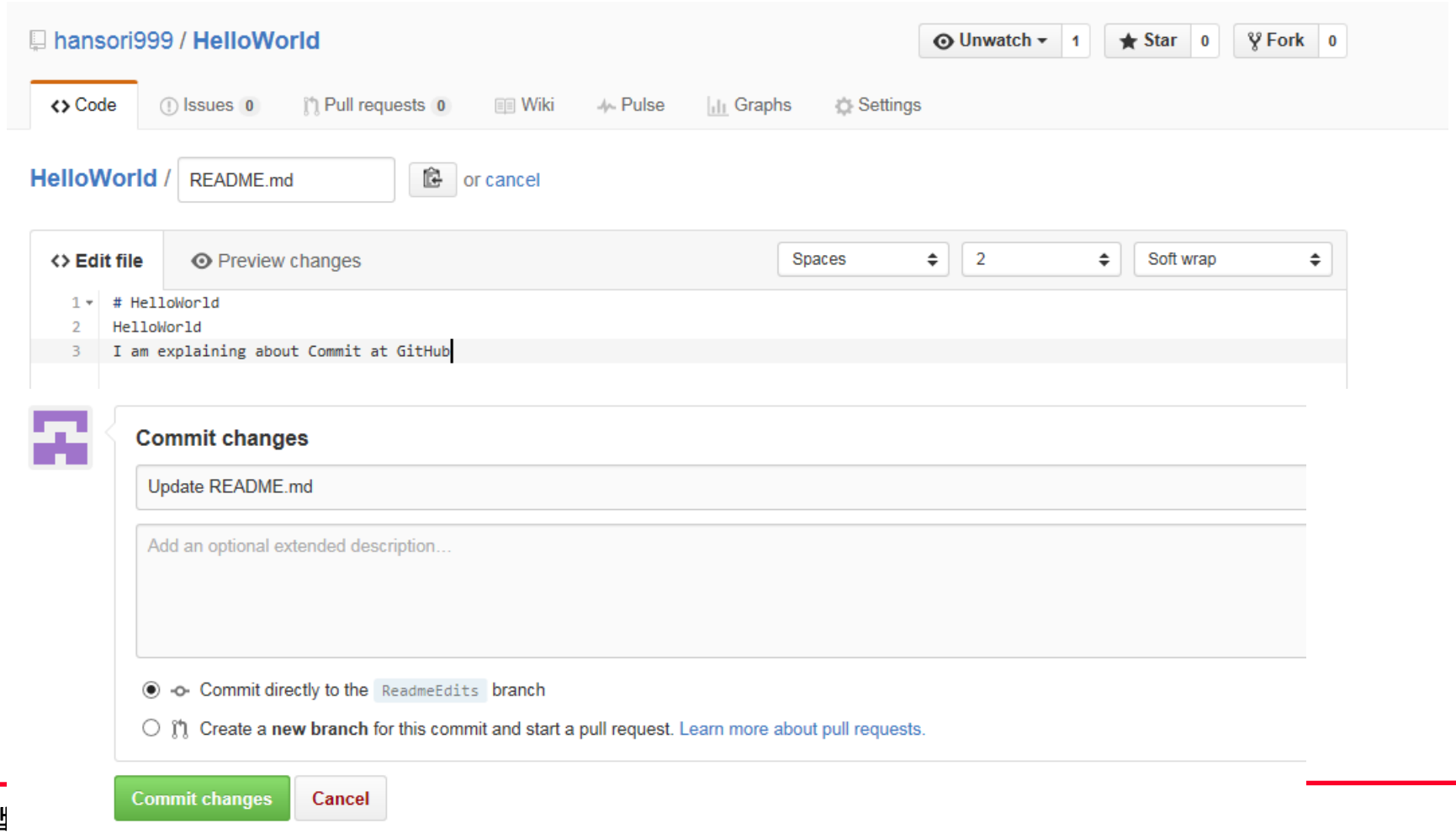
GitHub

- Make a commit
  - Commits are saved changes On GitHub
  - Each commit has an associated commit message
    - a description explaining why a particular change was made
- To commit changes
  - Click the README file on Branch ReadMeEdits
  - Click the pencil icon in the upper right corner of the file view to edit.



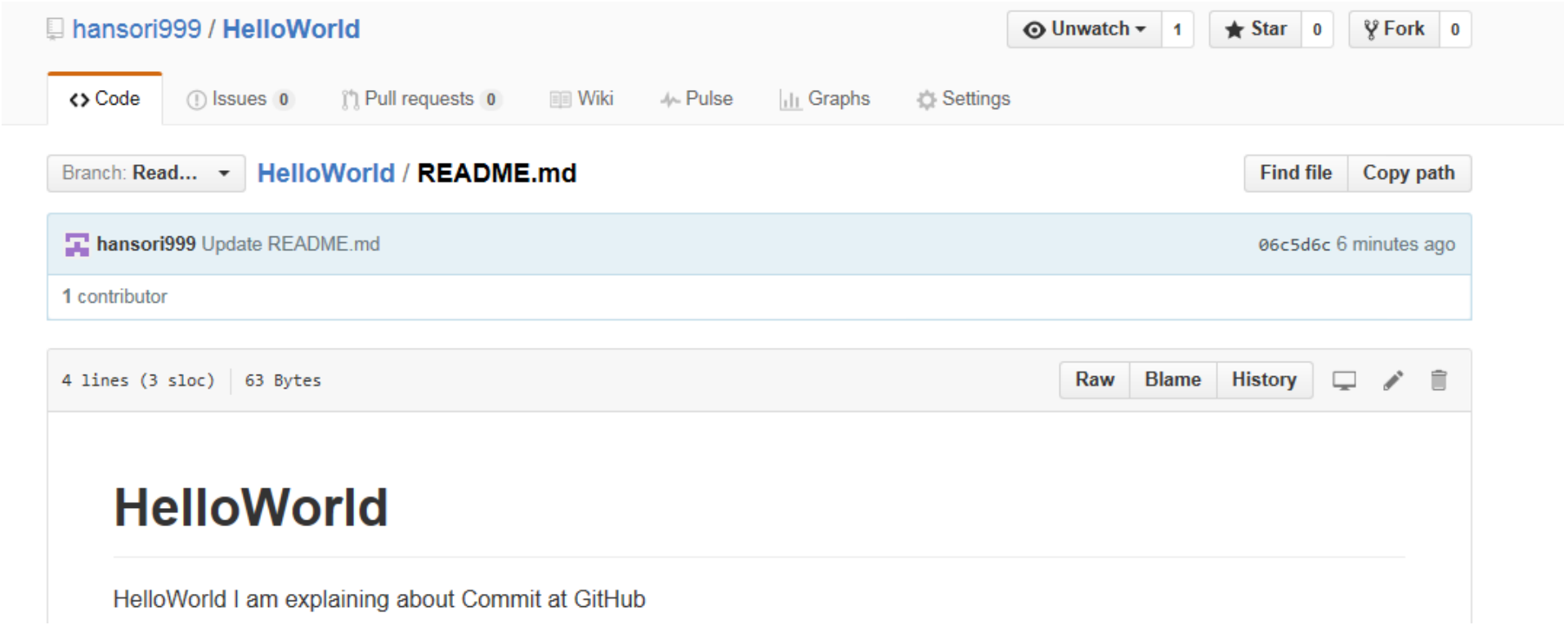
GitHub

- In the editor, write some text, tell a bit about yourself
- Write a commit message that describes your changes



GitHub

- Click Commit changes
- Changes in the README file on your readme-edits branch and now this branch contains different content and commits than master

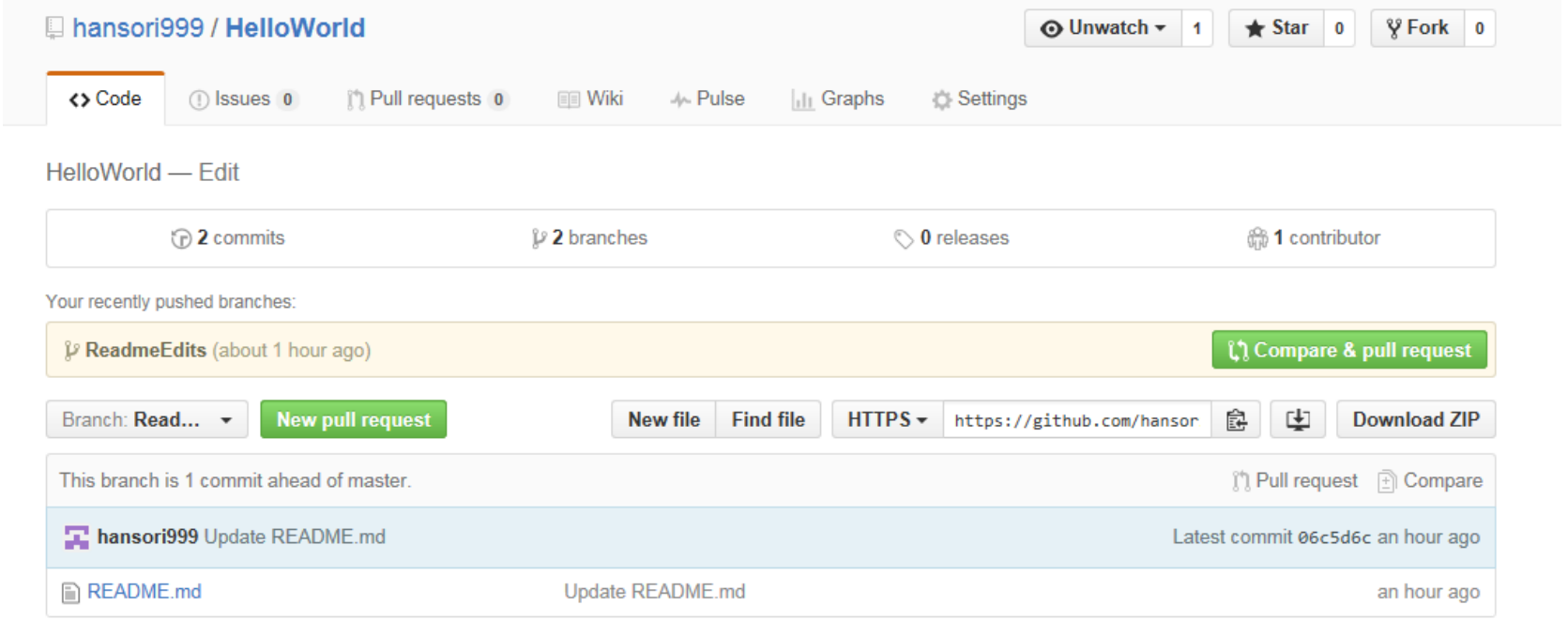


GitHub

- Open a Pull Request
  - allows you to compare the content on two branches
    - The changes, additions and subtractions, are shown in green and red and called diffs
  - Use Pull Requests to start a discussion about commits (code review) even before the code is finished.
  - a Pull Request in your own repository and merge it yourself

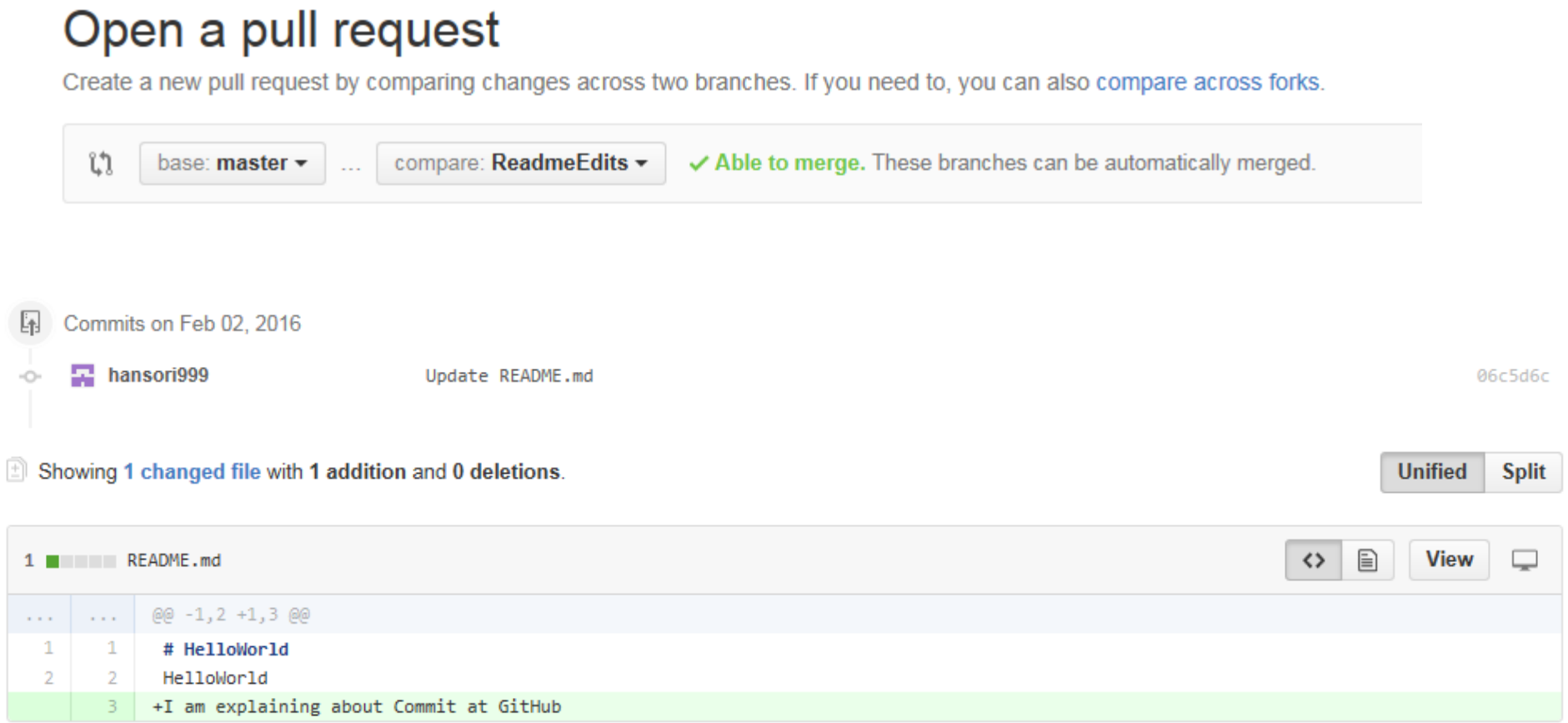
GitHub

- Create a Pull Request for changes to the README
  - Click the Pull Request icon
  - click the green New pull request button.



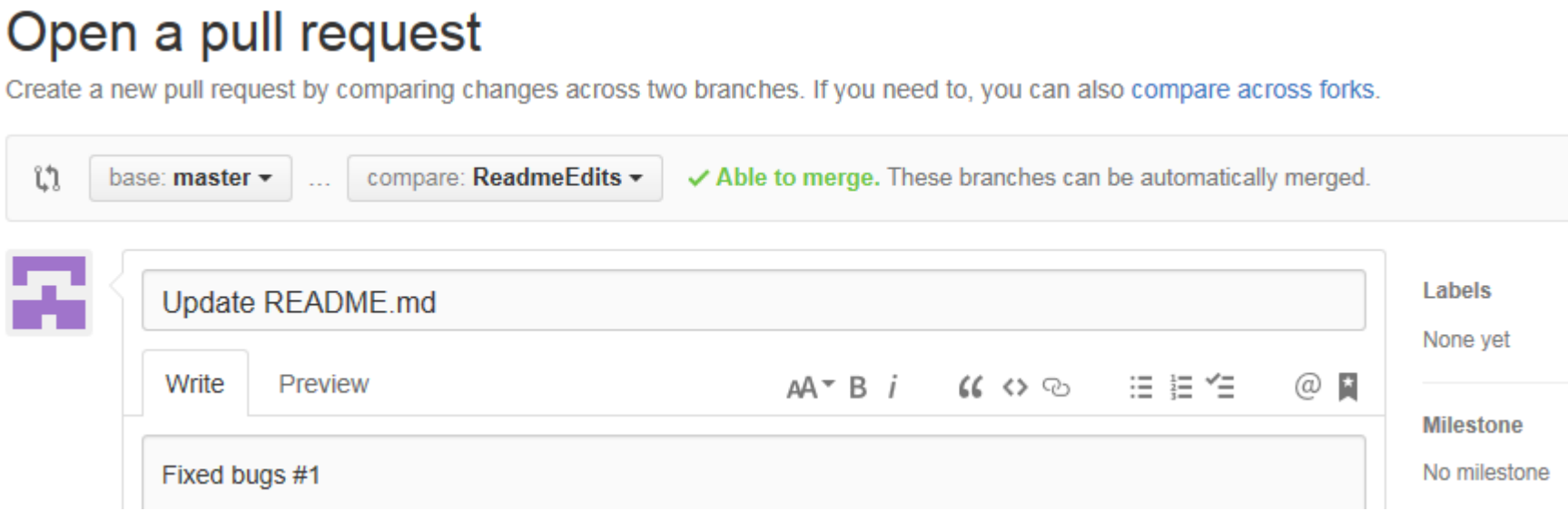
GitHub

- **Click the green New pull request button.**



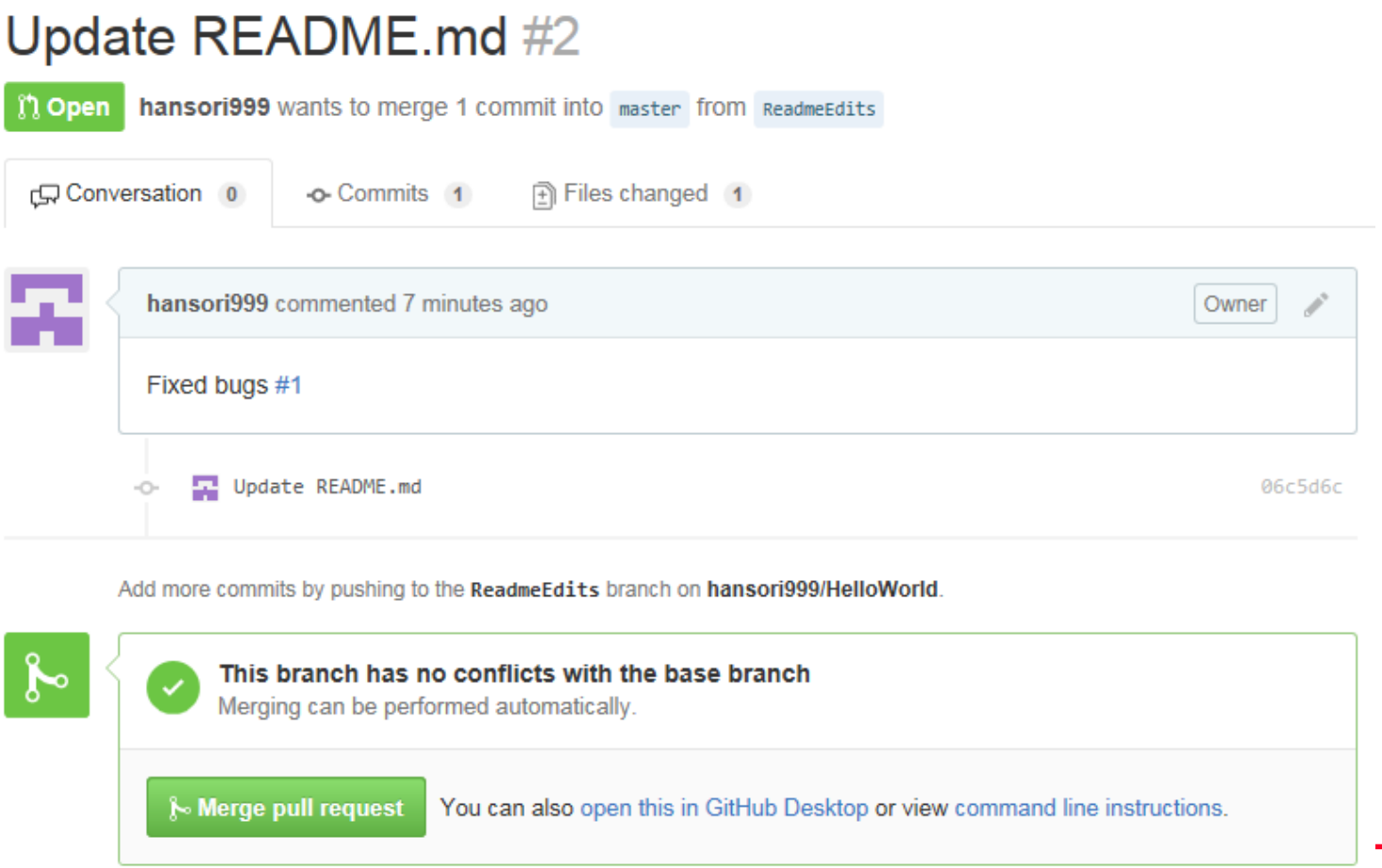
GitHub

- Click **create pull request**



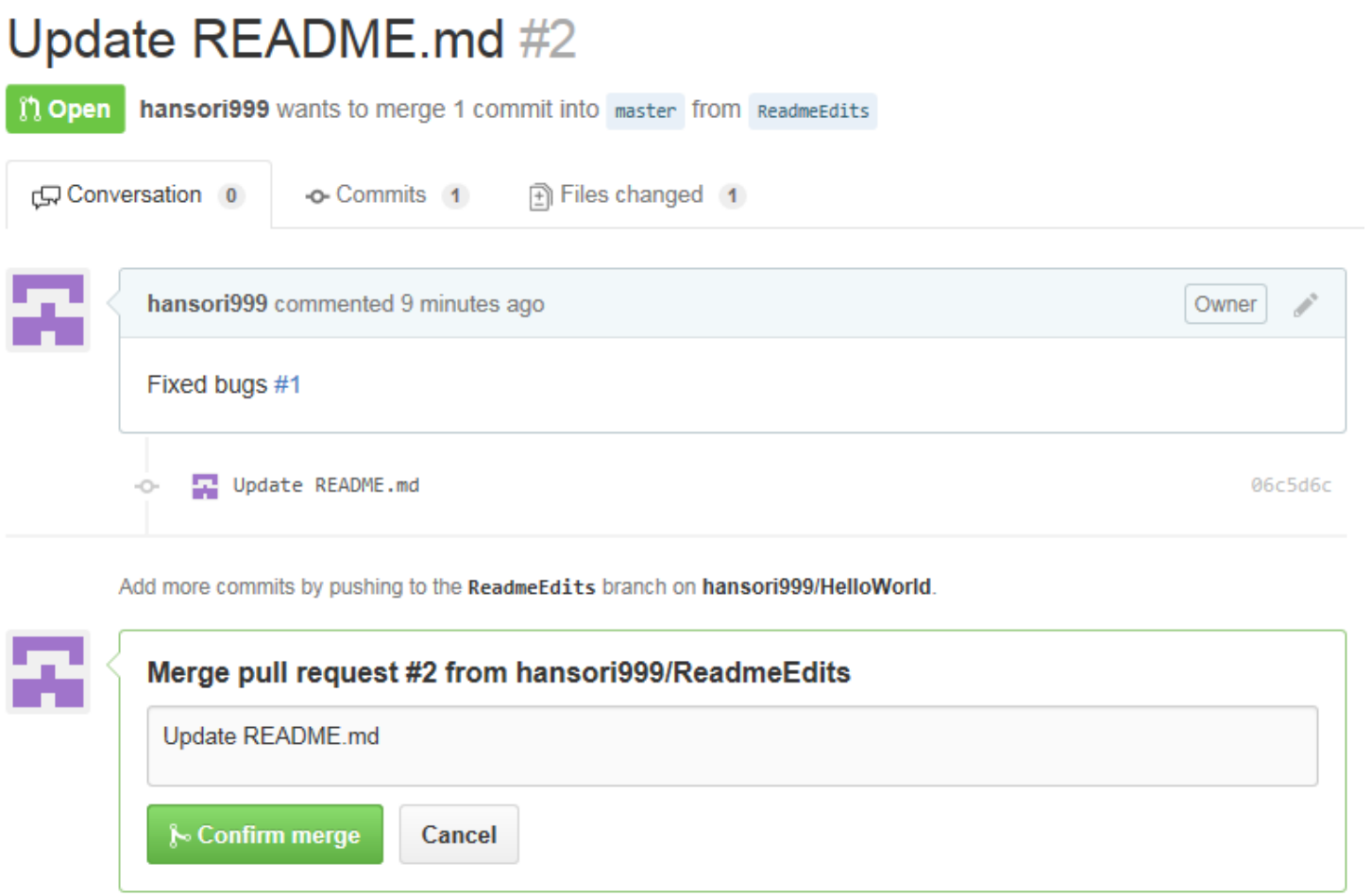
GitHub

- **Merge your Pull Request**
  - **Click Merge pull request**



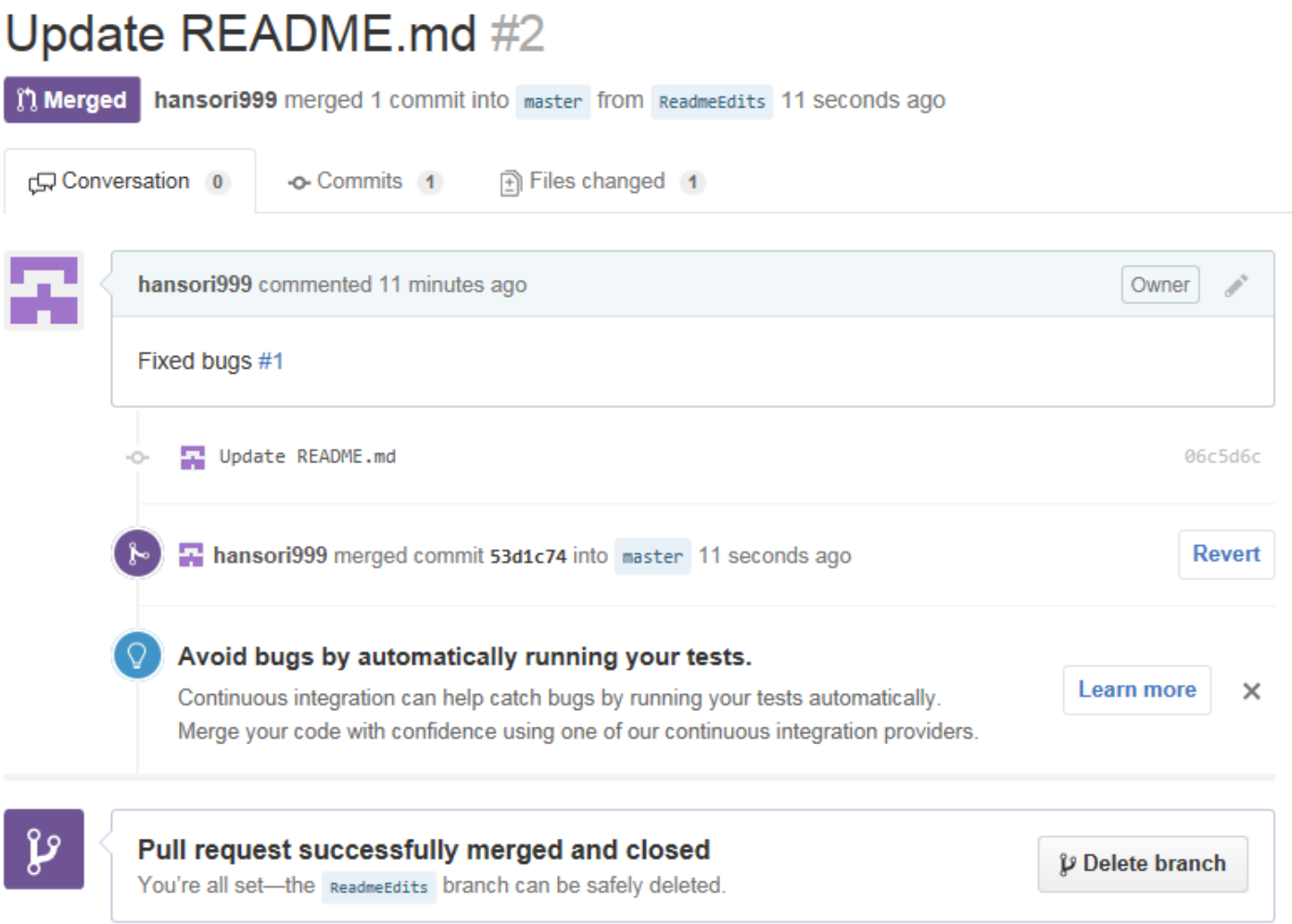
GitHub

- Click **Confirm merge**



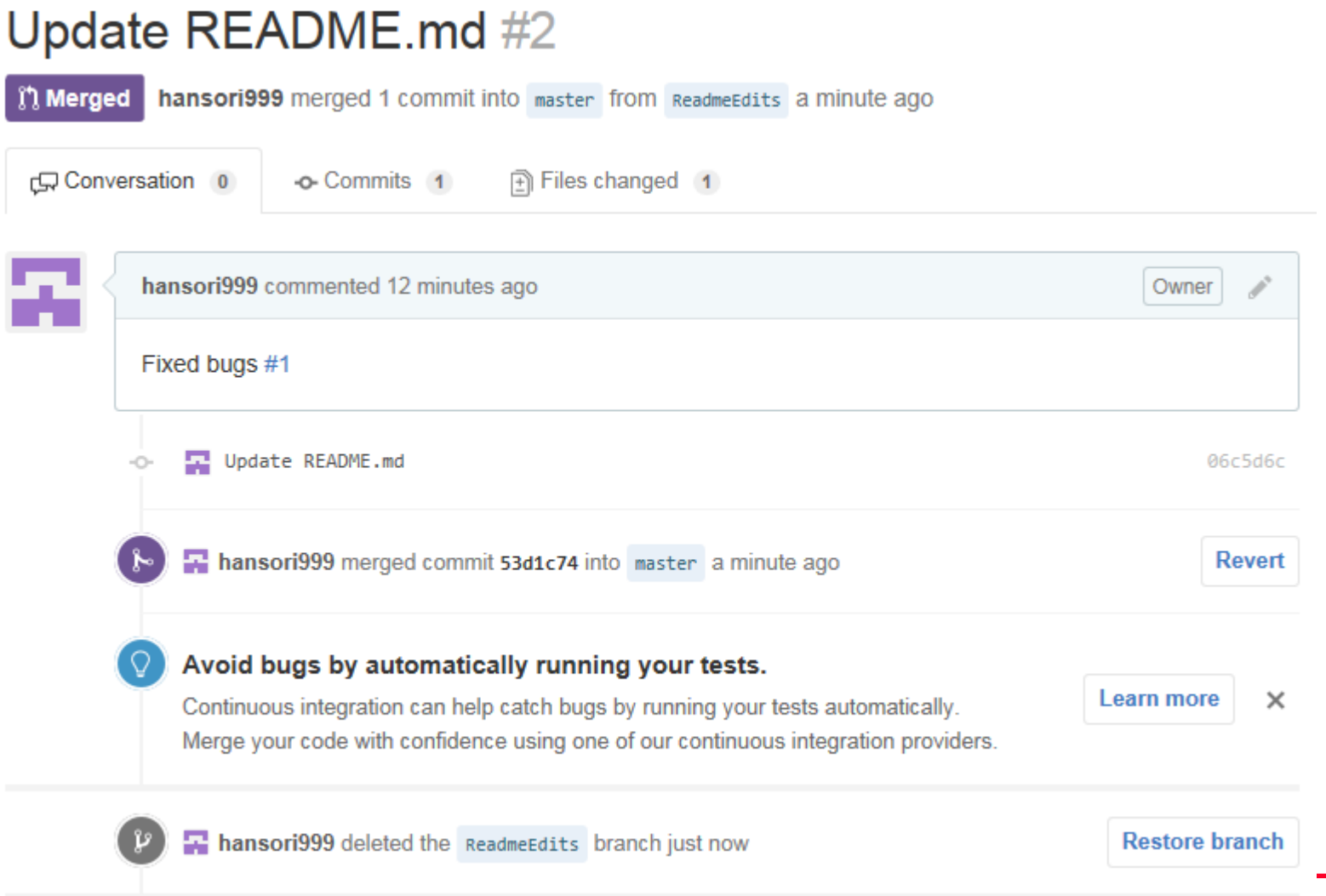
GitHub

- Go ahead and delete the branch Delete branch



GitHub

- **Final Result**



Git 기본

- **git help**
  - 기본적인 명령어 표시
- **git help –a**
  - 모든 명령어 표시
- **git help config 혹은 man git-config**
  - "config"라는 명령어 도움말 표시 사용자 이름과 이메일 설정
  - **git config --global user.name “Myoung-Wan Koo”**
  - **git config --global user.email mwkoo9@gmail.com**
- 설정확인
  - **git config –list**
  - **git config user.name**

Git 기본

- Color 설정
  - git config --global color.status auto
  - git config --global color.branch auto
- Ignore 설정
  - Git에게 특정 파일을 무시하도록 하는 기능
  - 프로젝트의 root directory에 .gitignore파일을 만들고, commit하고 싶지 않은 폴더나 파일명을 작성
  - 소스코드로 컴파일한 binary파일이나, 대용량의 리소스 파일, 그리고 임시파일 등을 제외시키는 용도로 사용
  - 표준 Glob 패턴(정규표현식을 단순하게 만든 것)을 사용함

Git

- **How To : 프로젝트 다운 받아 개발시작하기**

```
$ git clone http://github.com:hansori999/HelloWorld4.git    (1)
$ cd HelloWorld4
$ echo “first file” > first_file.txt
$ git add first_file.txt
$ git commit  –m “add new file”                              (2)
$ git push                                                   (3)
```

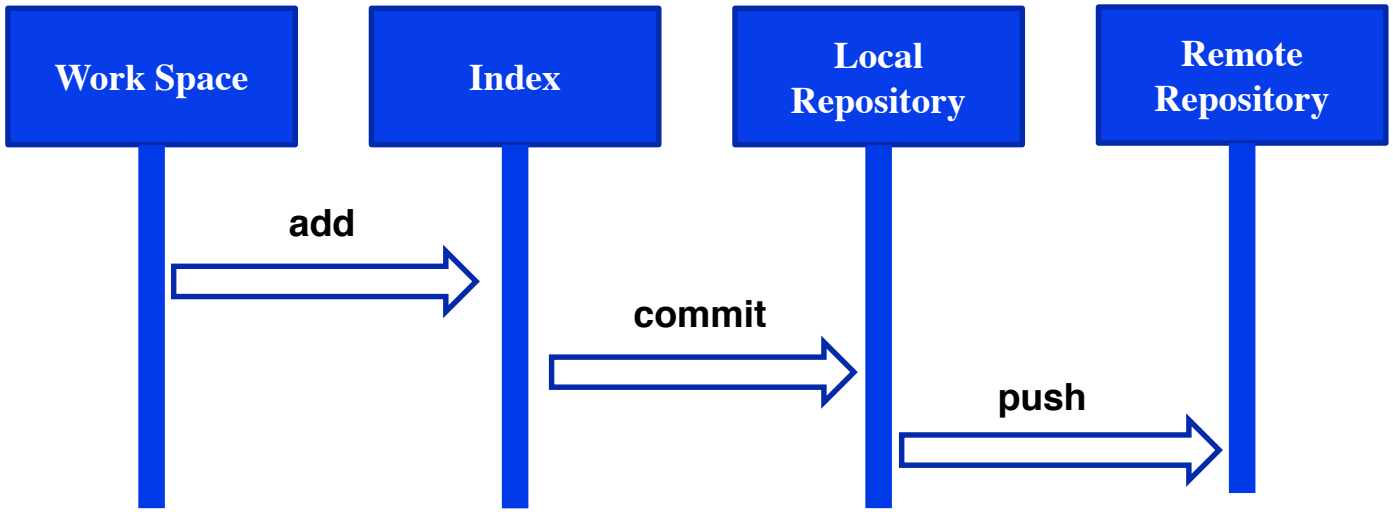
(1) 프로젝트 hansori999/HelloWord4.git를 HelloWorld 라는 폴더로 다운받는다.

(2) 변경된 것을 local repository에 commit하겠다.

(3) local repository에 commit 된 것을 remote repository에 반영하겠다.

- Work Space는 단지 개발하고 있는 파일시스템의 폴더일 뿐이다. (‘.git’ 폴더가 존재하는)
- 변경사항을 Local Repository에 적용.
- 그리고 이를 공유하기 위해 push로 Remote Repository(Server)에 올린다.

Git의 구성



- 앞의 예에서 'commit –a'라 한 것은 변경된 사항 전부를 commit의 대상으로 한다.
- 정확히는 commit의 대상은 오직 Index에 포함된 것만이다.
- 'commit –a'는 변경된 사항 전부를 add하고 나서 commit을 실행한 것과 같다.
- 앞의 예에서는 전부 commit –a를 사용하였다. 설명의 간편함을 위한 것이다. 선별적으로 commit할 대상을 add하는 것이 바람직하다.

How To : 내가 수정한 것과 타인이 수정한 것을 merging하여 commit하기

```
$ echo “some modification” >> first_file.txt          (1)
$ git commit –a –m “add first_file.txt”              (2)
$ git fetch                                          (3)
$ git diff origin
$ git merge origin                                   (4)
$ git push                                           (5)
```

commit의 –a 옵션은 git add 없이 모든 변경된 것을 대상으로 하라는 의미

(1) 파일을 수정했다.

(2) 변경된 사항 전부를 local repository에 commit한다.

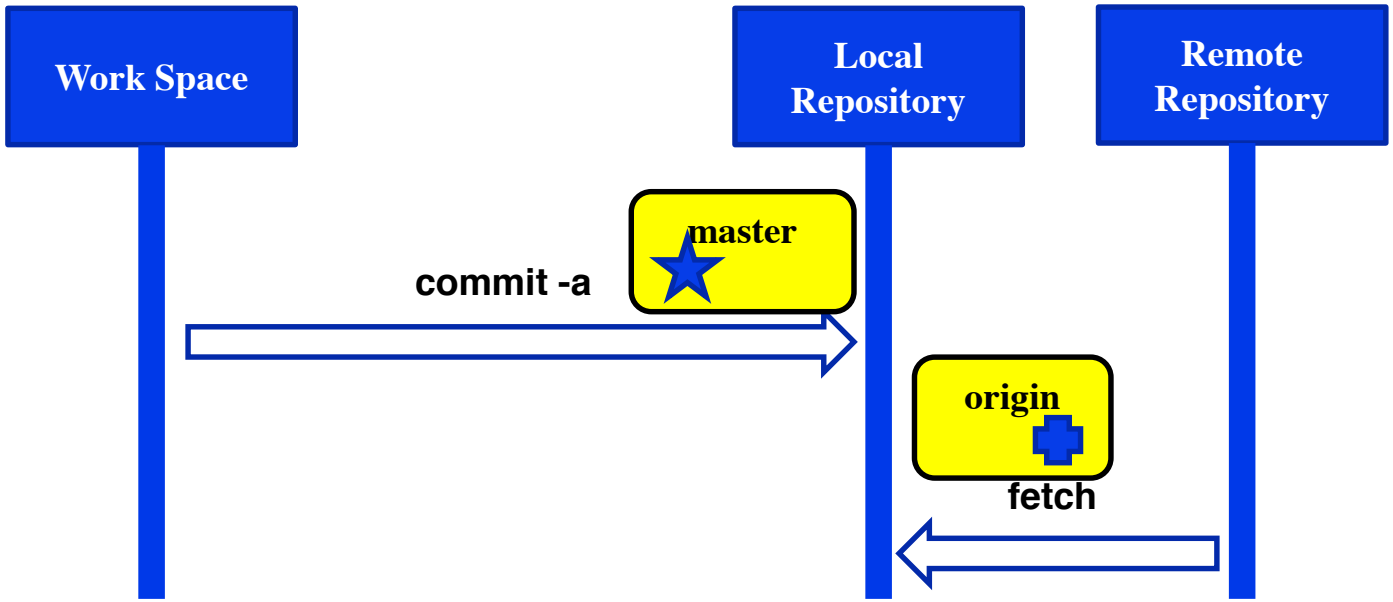
(3) remote repository에서 최신의 것을 local repository로 가져온다.

(4) local repository에 가져온 최신의 것과 내가 commit한 내용을 merging한다.

(3)/(4) git pull

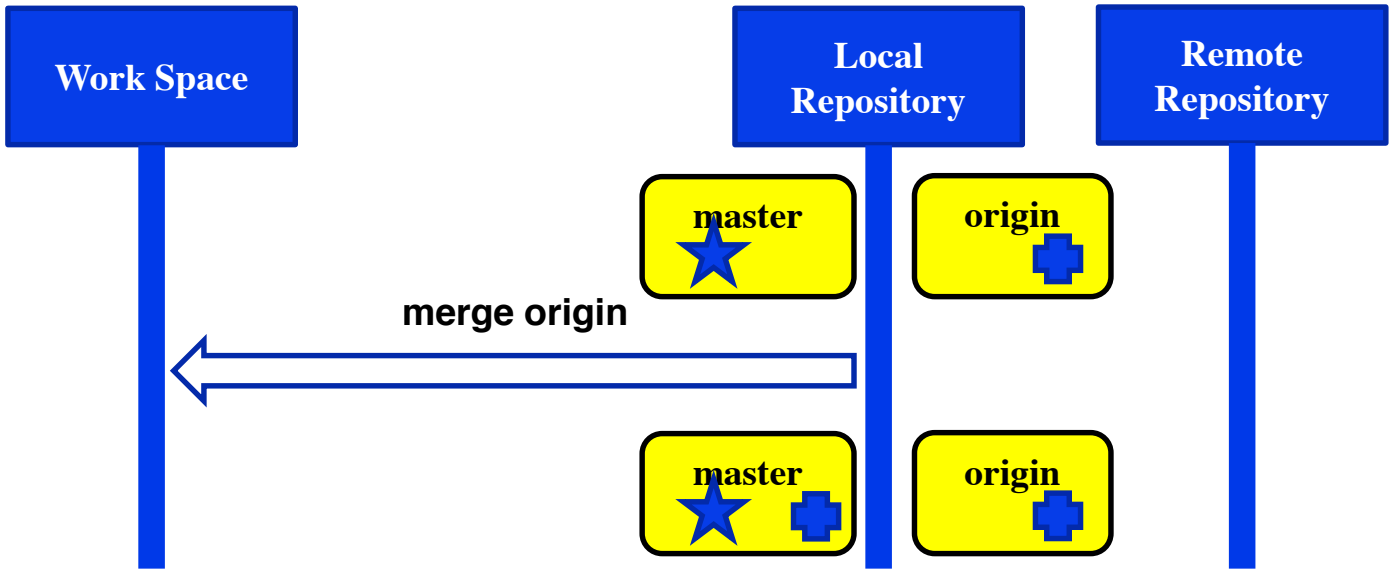
(5) merging된 사항을 remote repository에 올린다.

브랜치



- Git는 브랜치라는 개념으로 작업본을 관리한다.
- 내가 commit한 본은 'master'라는 이름의 브랜치로, Remote Repository 에서 가져온 것은 'origin'이라는 이름의 브랜치로 존재

브랜치 간 merge



- merge origin은 현재 작업중인 브랜치 master에 origin이라는 브랜치의 수정된 것을 병합하라는 것이다.
- merge origin을 실행한 후에는 origin의 수정 사항(십자가)가 HEAD에 반영되었다.
- 또한 merge된 결과는 다른 명령 없이 Work Space에 그대로 반영된다.

사용자가 정의하는 브랜치

How To : Local Repository에 새로운 branch 생성

```
$ ls first_file.txt
$ git checkout –b test              (1)
$ git branch                        (2)
  master
* test
$ echo “my test” > test.txt         (3)
$ git status                        (4)
$ git add test.txt
$ git commit –m “ Adding test”

$ git checkout master               (5)
$ git status                        (6)
```

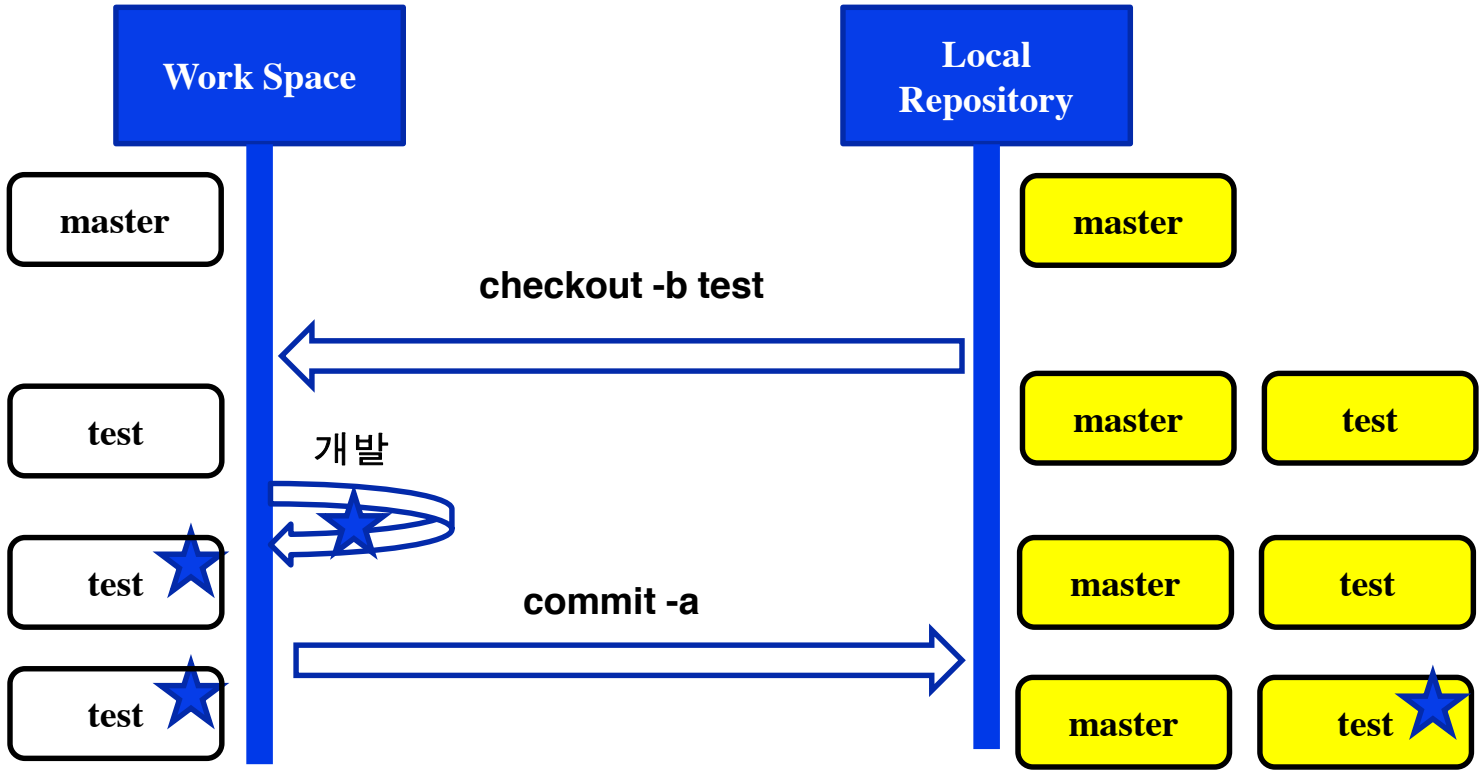
(1) 새로운 브랜치 ‘test’를 생성하고, 그 브랜치로 변경한다.

(2) 브랜치 리스트를 조회한다. 별표(*)가 붙은 test 브랜치가 현재의 브랜치이다.

(3), (4) 새로운 파일 test.txt를 만들었다.

(5), (6) 기존의 브랜치 master로 변경했다..

branch 생성, 수정



- 브랜치가 생성될 때 Work Space(그냥 폴더)의 내용은 master와 같다.
- 임의의 수정(별표)하고 commit 하면 Local repository의 test 브랜치에 그 내용이 적용된다. master하고는 관계없다.

branch merge

```
$ git branch                    (1)
*  master
   test
$ git merge test                (2)
$ git push                      (3)
$ git branch –D test            (4)
```

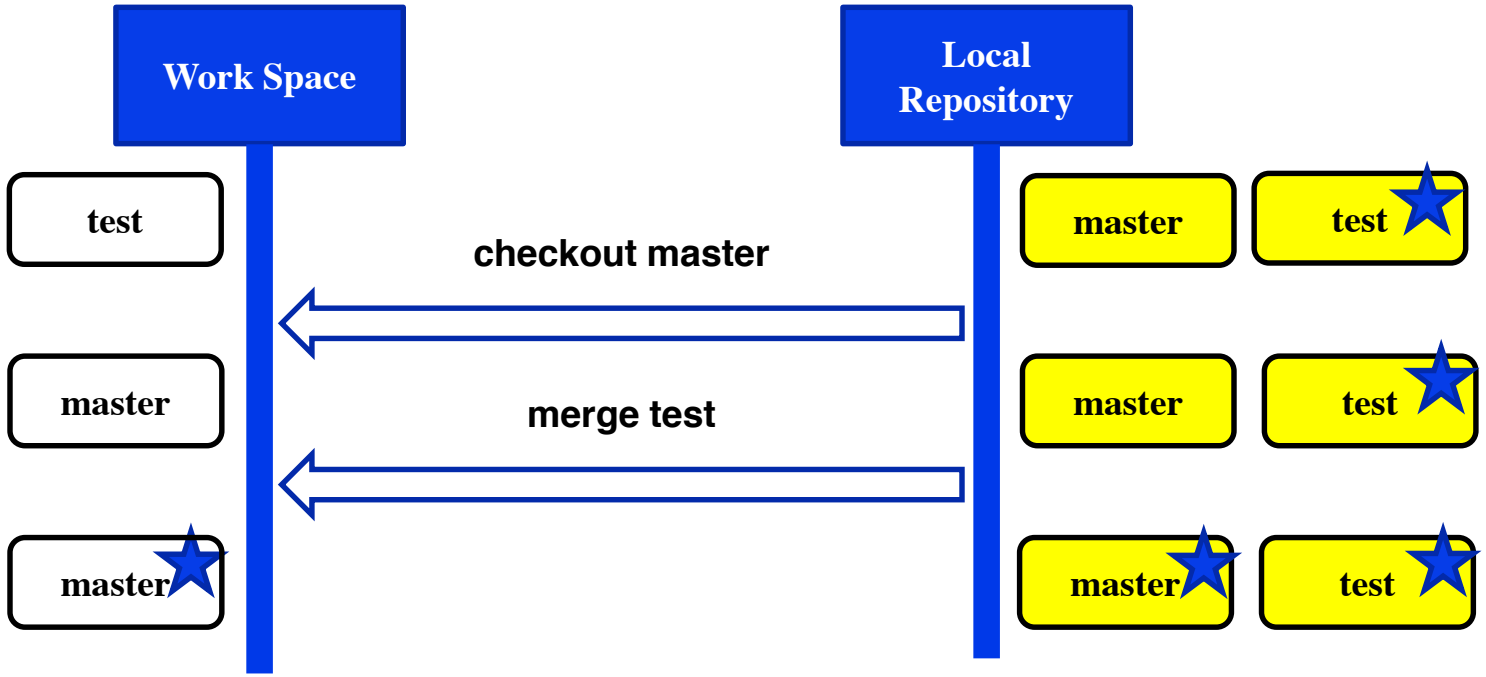
(1) 브랜치 리스트를 조회한다. '*'표 붙은 master가 현재의 브랜치이다.

(2) 브랜티 test 를 master 로 merge 한다.

(3) 서버로 push 한다.

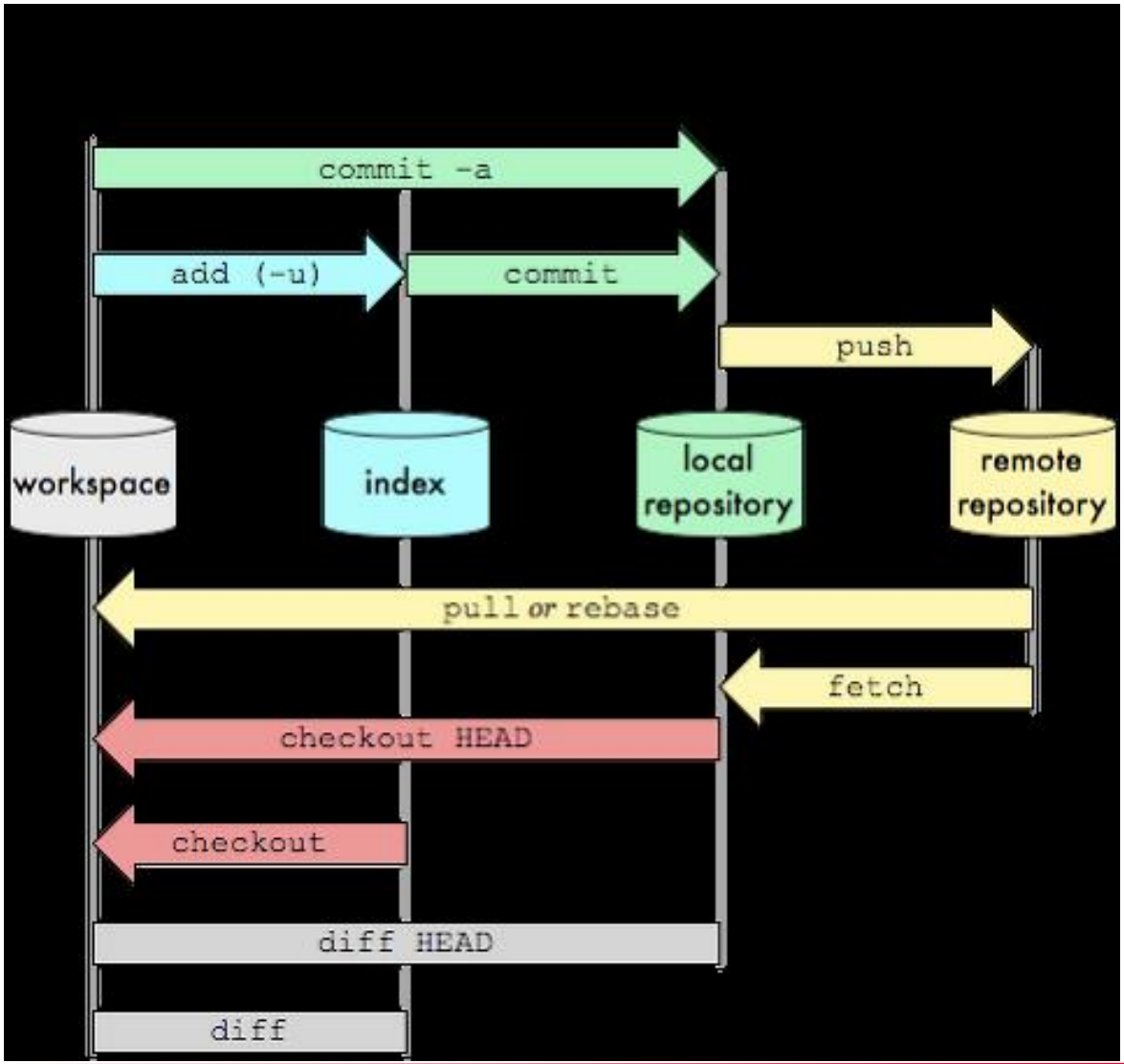
(4) 브랜치 test를 삭제한다.

branch merge



- 새 브랜치 test에 적용한 사항을 master에 적용하려면 master 브랜치로 변환한 후 ‘merge test’를 한다.

Pull 과 Fetch


commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
diff HEAD
diff

diff 하기

How To : 현재 작업된 것과 Local Repository과의 차이를 비교하기

```
$ git commit –m “old change”
$ echo “hi” >> test.txt
$ git diff                                                    (1)
…
+hi
```

(1) Local repository에 commit된 것과 현재 Work Space의 차이를 보여준다.

How To : 좀더 다양한 diff

```
$ git diff                  : Local Repository의 최근 commit된 상태와
$ git diff --cached         : Index와 Local Repository간 차이
$ git diff HEAD                    : git diff와 동일
$ git diff HEAD^            : Local Repository의 바로 전 revision과
$ git diff HEAD^^           : Local Repository의 바로 전전 revision과
$ git diff master test      : 브랜치 master와 test 간 차이
$ git diff a1b2c3           : revision “a1b2c3”간 차이
$ git diff HEAD^ a1b2c3 : Local Repository의 바로전 revision과
revision ‘a1b2c3’간 차이
$ git diff origin                  : fetch해서 가져온 origin과 비교
```

- **git diff**에 사용되는 옵션은 브랜치 이름이거나 **revision**이다. 똘똘하게 알아서 찾아 준다.
- **Remote Repository**와 현재의 것을 비교하려면 우선 **fetch** 명령으로 **Local Repository**로 가져온 다음 **diff origin**으로 비교할 수 있다.

branch, HEAD, revision, master, origin

- Git는 revision에 순차적인 번호을 사용하지 않는다.
- 특정 revision에 숫자와 문자가 섞인 7 character를 사용한다.
- 그렇기 때문에 어떤 브랜치의 어떤 리비전이라고 하지 않고, diff 시에 그냥 리비전만 주면 된다.
- 만약 브랜치의 이름이 주어지면, 가장 최근의 리비전을 사용한다.

- HEAD는 현재 작업중인 브랜치의 가장 최근 리비전을 가리키는 예약어이이다.
  - 만약 브랜치를 변경하지 않았으면 master가 현재의 브랜치가 되고 HEAD 대신 master로 하여도 결과는 같다.

- master는 기본 브랜치의 이름이다.
- origin은 Remote Repository에서 가져온 브랜치의 이름이다.

기타 팁들

How To : 파일 삭제, 파일 명 변경

```
$ git mv old.txt new.txt          (1)
$ git rm useless.txt              (2)
```

(1) 파일명을 변경.

(2) 파일을 삭제

- Git는 파일명의 변경을 일일이 알려 주어야 한다. 지우고 새로 만든 것인지, 파일명만 변경되었는지 파악하지 못한다.
- 그래서 파일시스템의 rm, mv를 사용하면 곤란하다.

How To : 로그 보기

```
$ git log -5 --decorate --graph –-oneline                    (1)
$ git log -5 --author=hansori999 –-oneline                   (2)
$ git log --before={3.weeks.ago} --after={2015-04-18}        (3)
$ git log --grep=“some message”                              (4)
```

(1) 최근 5개의 리비전을 그래프로 보여준다.

(2) 저자가 hansori999인 리비전을 5개 보여준다.

(3) 주어진 기간내의 리비전만 보여준다.

(4) commit 메시지에 “some message”가 포함된 것만 보여준다.

How To : commit 취소하기

```
$ git reset --soft HEAD^                                    (1)
```

(1) **Local Repository의 최신 commit 리비전을 바로전 리비전(HEAD^)로 설정한다. Index와 Work Space는 건드리지 않는다.**

How To : 특정 리비전 다운 받기

```
$ git checkout –b new a1b2c3                    (1)
```

(1) 새로운 브랜치 ‘new’를 생성하고 그 곳에 리비전 ‘a1b2c3’을 내려받는다.