Git 교육자료 하편

2021.01 제작

목차

- Git 브랜치
  - git branch
  - git checkout
  - git merge
- GitHub
  - Fork
  - Pull request
  - Collaborate
- Git 고급
  - git stash
  - git reset
  - git flow

Git 브랜치 – git branch & git checkout

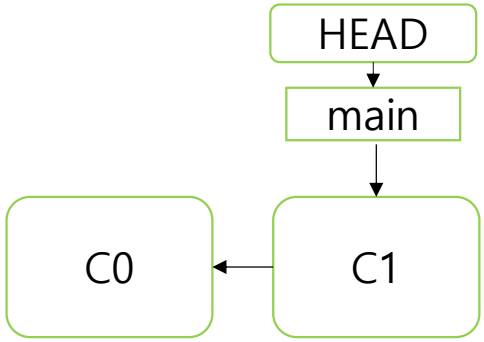
- 주로 다른 사람과의 협업할 때 사용.
  각자 다른 부분을 작업하고 있을 때, 브랜치를 생성한다.

- **git branch <branch name>** : 현재 위치에서 브랜치를 하나 생성함.

- **git checkout <branch name>** : 현재 작업 위치(HEAD)를 변경함.

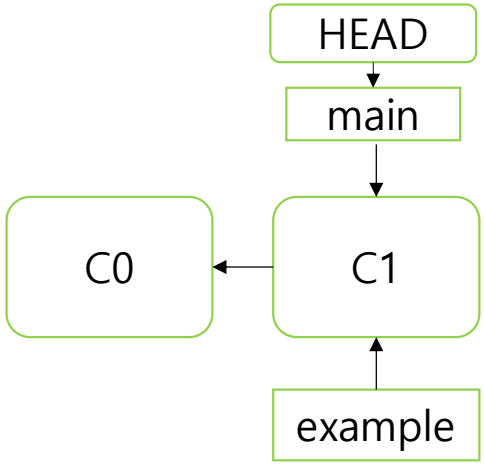
Git 브랜치 – git branch & git checkout

- 초기 상태 (C0~C1은 커밋 된 순서, 예시에서 git add와 commit message는 생략)



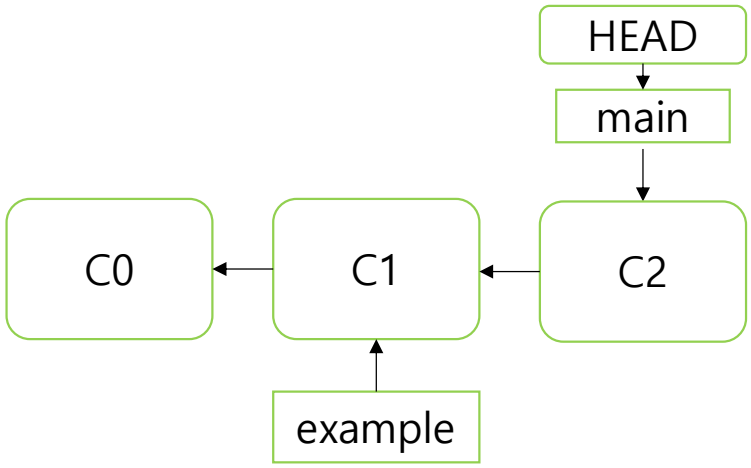
Git 브랜치 – git branch & git checkout

- **git branch example** : example이라는 브랜치 생성



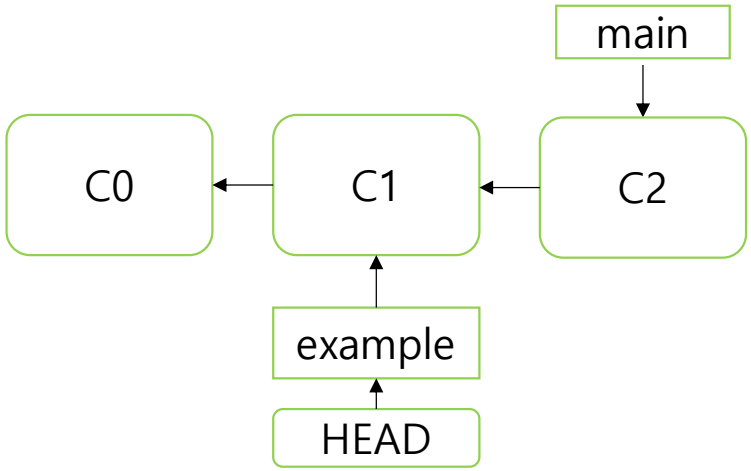
Git 브랜치 – git branch & git checkout

- **git commit**



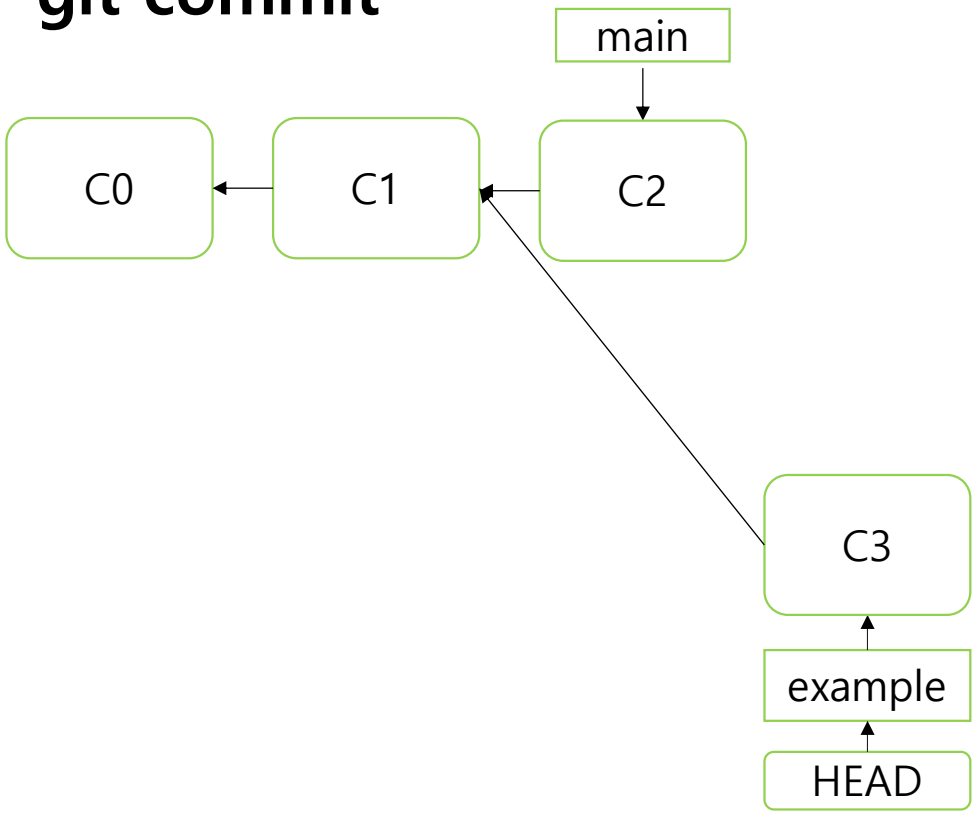
Git 브랜치 – git branch & git checkout

- **git checkout example** : HEAD를 example 위치로 이동



Git 브랜치 – git branch & git checkout

- **git commit**



Git 브랜치 – git merge

- 각각의 브랜치에서 작업한 내용을 한데 모아야함.

- **git merge <branch name>** : 해당 브랜치를 HEAD에 병합한다.

Git 브랜치 – git merge

- 만약 merge도중, 두 브랜치 간에 코드가 겹치는 부분이 있어 충돌이 발생 할 수 있다. 이때는

1. 충돌한 파일 수정 (vim ,notepad 등등 이용…)
2. **git add <충돌했던 파일명>**
3. **git commit –m <commit message>**

를 해주면 된다.

- 만약 merge를 취소하고 싶다면 다음 명령어를 입력한다.

git merge --abort

Git 브랜치 – git merge

- merge 하는 중간에 conflict가 발생하면 다음과 같이 (HEAD 브랜치|MERGING)이 표시된다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git branch
(main)
$ git merge test
Auto-merging a
CONFLICT (content): Merge conflict in a
Automatic merge failed; fix conflicts and then commit the result.

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git branch
(main|MERGING)
$
```

Git 브랜치 – git merge

- 충돌이 발생한 파일 a를 확인해보면 다음과 같다.
- **vim a**

```
1 <<<<<<< HEAD
2 5
3 =======
4 3
5 >>>>>>> test
```

- 구분선을 기준으로 위쪽은 HEAD의 내용(5),
  아래쪽은 HEAD에 병합할 test 브랜치의 내용(3)이다.

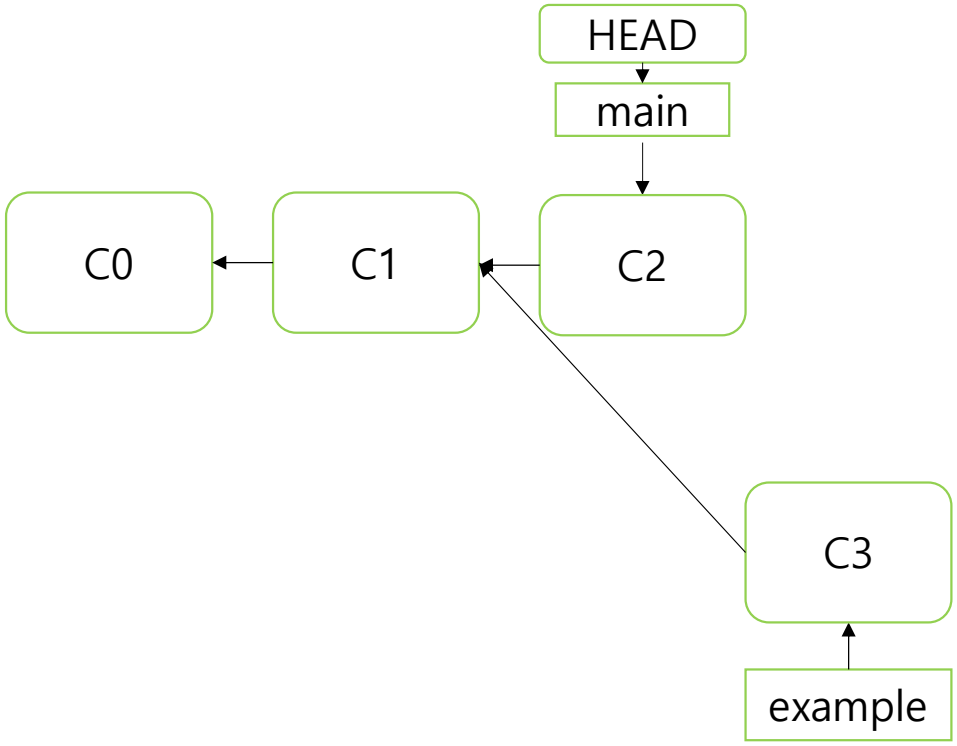
Git 브랜치 – git merge

- 보통은 팀원과 어떤 브랜치의 내용을 택할 지 상의해야하지만,
  이번 예시에서는 test 브랜치의 내용을 택하기로 했다.
- 충돌이 발생한 파일 a를 위와 같이 수정하면 다음과 같다.

- 이제 파일을 저장하고 나와서 다음 명령어를 입력하면 merge가 완료된다.
- **git add a**
- **git commit –m “Merge branch test into main”**

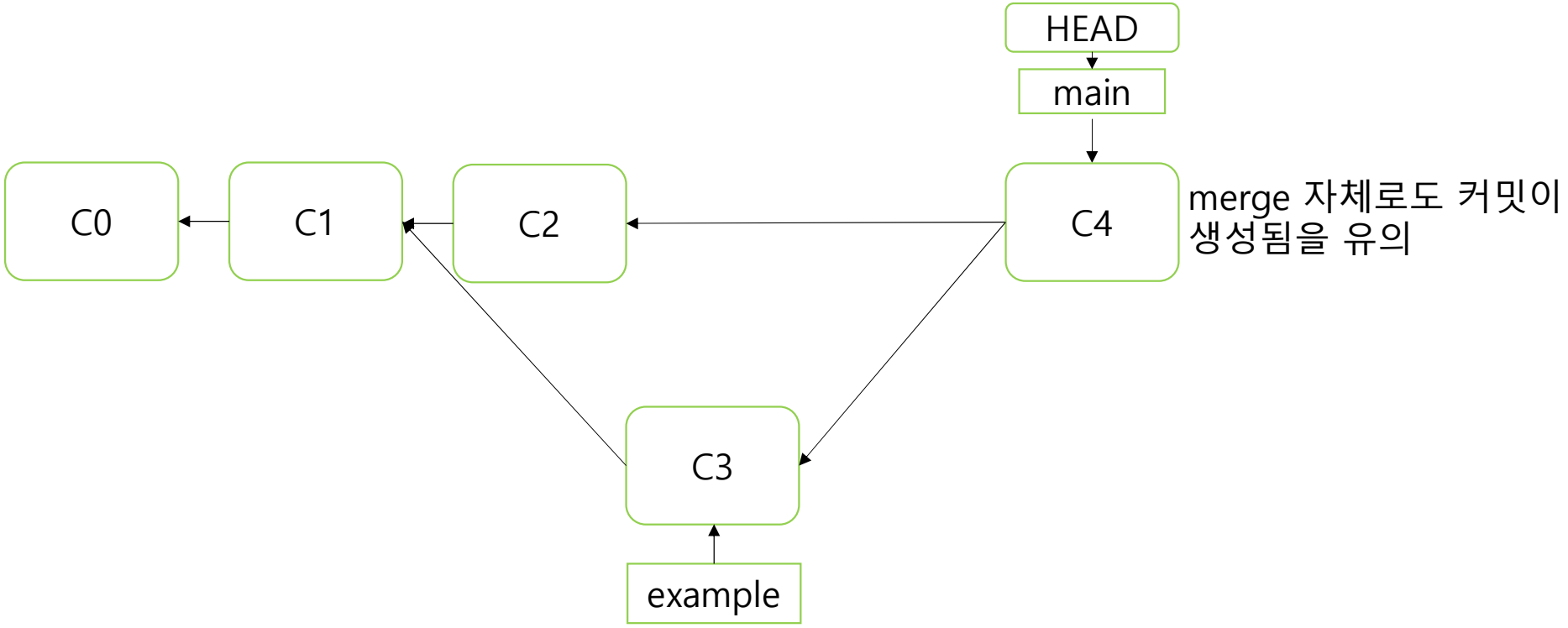
Git 브랜치 – git merge

- **git checkout main** : main 브랜치로 HEAD를 이동



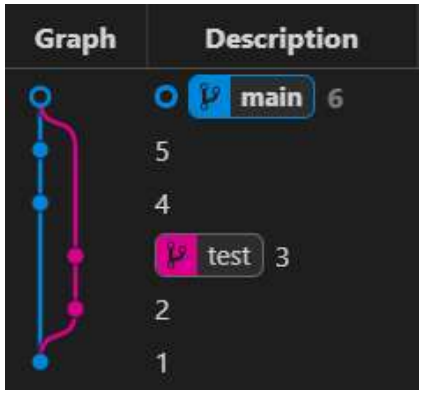
Git 브랜치 – git merge

- **git merge example** : example 브랜치를 HEAD에 병합



Git 실습 7 – branching

- git 명령어들을 이용하여 해당 그림처럼 구성.
  Commit 메시지는 중요하지 않으나, commit간의 순서는 중요.



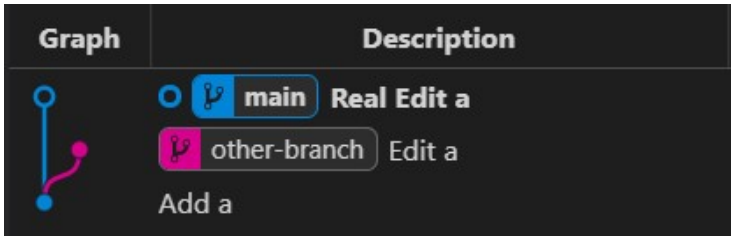
- 아직 reset을 배우지 않은 시점이니, 잘못하면 지우고 다시시작

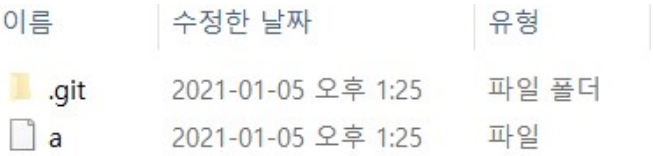
Git 고급

- Git 고급에서는 세가지를 다뤄보도록 하겠습니다.
- **git stash** : 작업하던 도중에 다른 브랜치로 이동할 수 있게 한다.
- **git reset** : 옵션에 따라 git commit, git add 또는 파일 수정 자체를 되돌린다.
- **git flow** : 브랜치를 효율적으로 사용하고 관리하기 위한 전략

Git 고급 – git stash

- 현재 상황은 다음과 같다. (HEAD는 main)





- 이 상태에서 파일 a를 수정하고 other-branch로 이동해보자

git checkout other-branch

Git 고급 – git stash

- 다음과 같은 오류가 뜨게 된다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a

no changes added to commit (use "git add" and/or "git commit -a")

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git checkout other-branch
error: Your local changes to the following files would be overwritten by checkout:
        a
Please commit your changes or stash them before you switch branches.
Aborting
```

Git 고급 – git stash

- 이때, 다음 명령어를 입력하여 현재 파일의 변경사항을 저장

- **git stash** : 현재 상황을 저장한다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git stash
Saved working directory and index state WIP on main: b75efd4 Real Edit a

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git status
On branch main
nothing to commit, working tree clean

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git checkout other-branch
Switched to branch 'other-branch'

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (other-branch)
$
```

Git 고급 – git stash

- **git stash list** : stash의 list를 출력한다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (other-branch)
$ git stash list
stash@{0}: WIP on main: b75efd4 Real Edit a
```

Git 고급 – git stash

- 다시 main 브랜치로 돌아와서 stash를 적용

- **git stash apply** : 가장 최근 stash를 적용한다.
- **git stash apply stash@{n}** : 특정 stash를 지정하여 적용한다

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (other-branch)
$ git checkout main
Switched to branch 'main'

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git stash (main)
$ git stash apply
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a

no changes added to commit (use "git add" and/or "git commit -a")
```

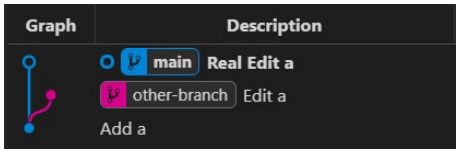
Git 고급 – git stash

- **git stash --keep-index** : staging area의 상태까지 stash에 저장한다.
- **git stash apply --index** : staging area의 상태까지 적용한다.

- **git stash drop** : 가장 최근 stash를 drop한다.
- **git stash drop stash@{n}** : 특정 stash를 drop한다.

Git 실습 8 – git stash

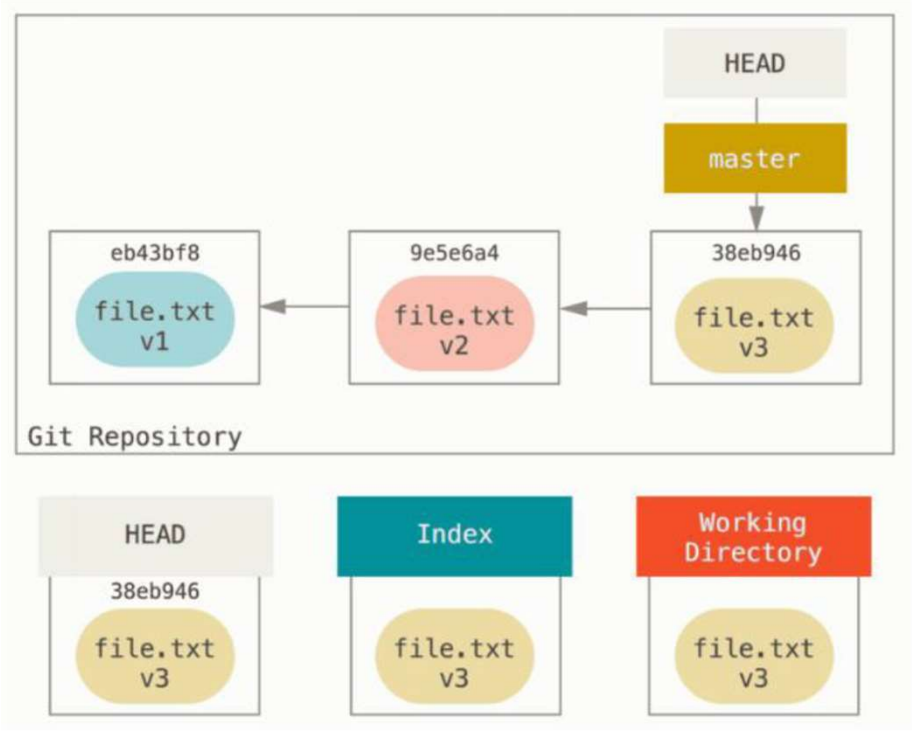
- 방금 예시를 실습해보자.



1. 초기 상태는 우측과 같다.
2. 파일 a를 수정해보고 other-branch로 checkout을 시도해본다.
3. **git stash**를 사용하고, other-branch로 이동한다.
4. other-branch에서 다시 main으로 이동한다.
5. **git stash apply**로 변경사항을 되돌린다.
6. **git stash drop**으로 stash를 삭제한다.

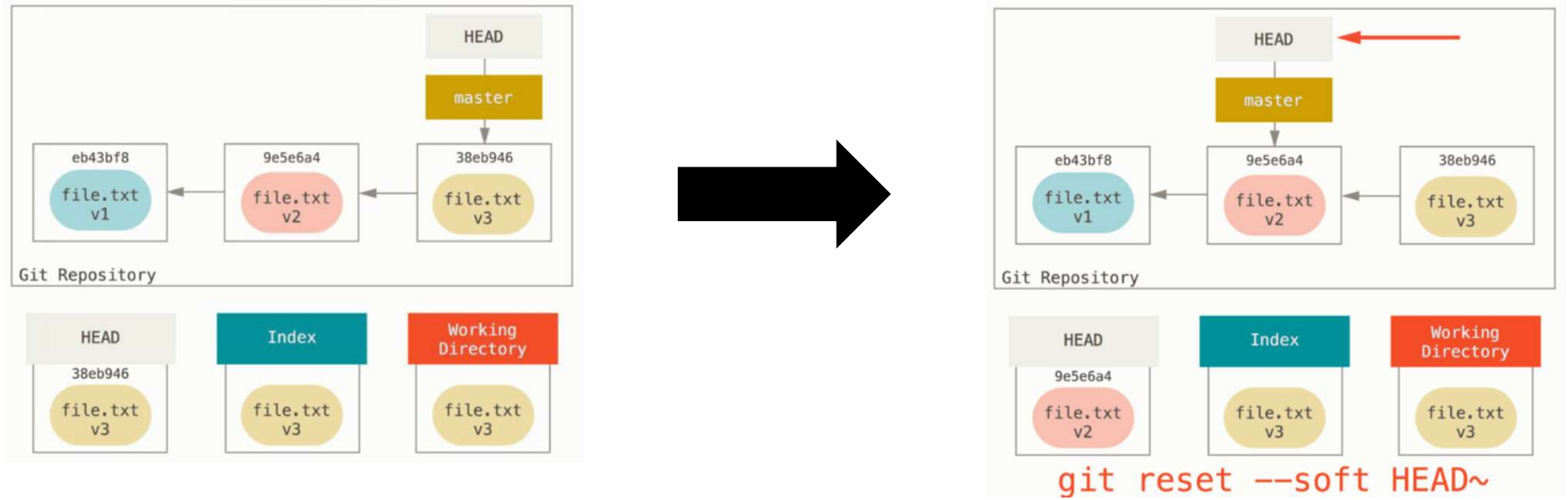
Git 고급 – git reset

- 초기 상태(index=staging area, master=main)



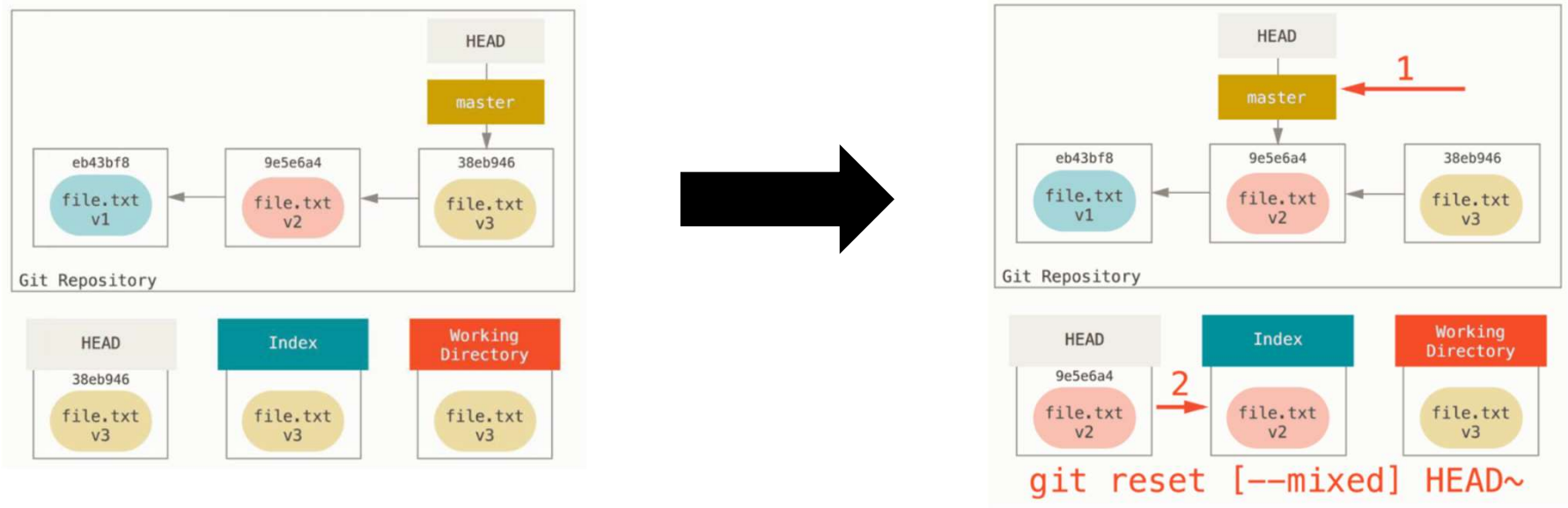
Git 고급 – git reset

- **git reset --soft HEAD~n** : HEAD포함 n개 소프트 리셋.
  git commit과 반대 역할. n이 없다면 1로 간주



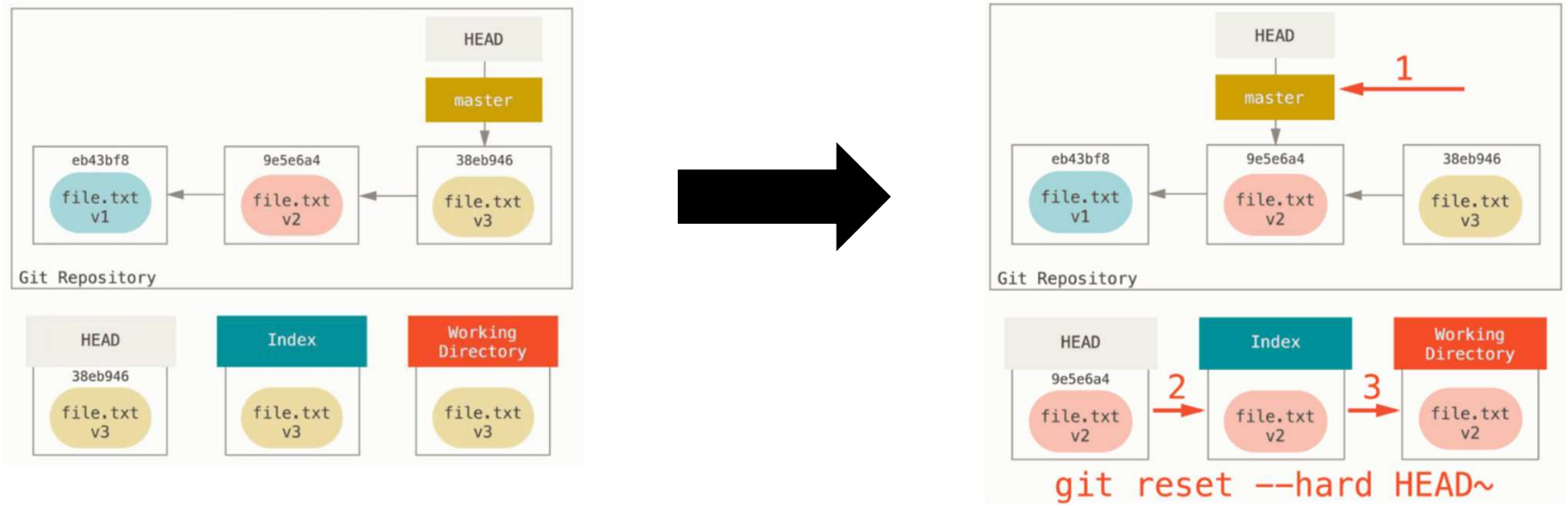
Git 고급 – git reset

- **git reset (--mixed) HEAD~n** : HEAD포함 n개 리셋.
  git add과 반대 역할



Git 고급 – git reset

- **git reset --hard HEAD~n** : HEAD포함 n개 리셋.
  파일을 수정하기 전으로 되돌림



Git 실습 8 – git reset

1. Local repository를 하나 만든다.

2. 3개의 커밋(C0, C1, C2)을 생성한 다음

3. C1의 상태로 돌아간다. (파일도 C1의 상태로)

Git 고급 – git flow

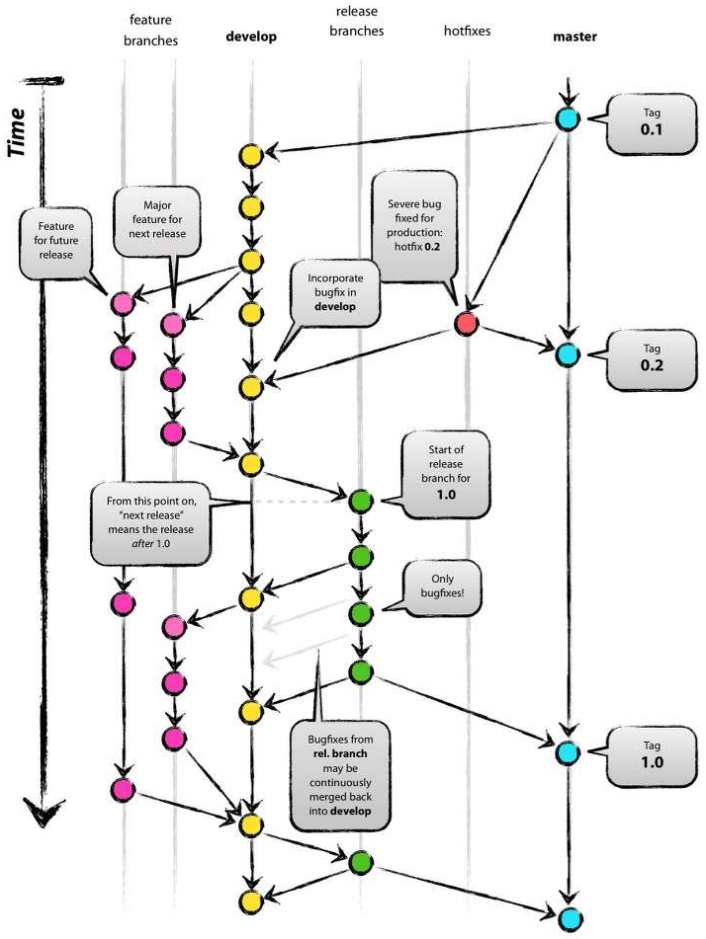
- **git flow** : 브랜치를 효율적으로 관리하기 위한 전략

크게 다음의 브랜치로 구성

- **main(=master)** : 제품으로 출시될 수 있거나 현재 배포중인 브랜치
- **release** : 배포를 준비하는 브랜치
- **develop** : 다음 출시버전을 개발하는 브랜치
- **feature** : 기능을 개발하는 브랜치

Git 고급 – git flow

1. develop에서 특정 기능을 위해 feature를 분기한다.
2. feature에서 개발이 끝나면 develop으로 병합한다.
3. develop에서 배포를 위해 release를 분기한다.
4. 테스트가 완료되면 release를 develop과 master에 병합한다.



Git 고급 – git flow

- 직접 브랜치를 만들고 병합하여 git flow를 따라 갈 수 있지만, git flow 명령어 집합은 이를 편하게 해준다.

- **git flow init** : 현재 디렉토리에 git init을 포함하여 기본 세팅을 한다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow
$ git flow init
Initialized empty Git repository in E:/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master] main
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [E:/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow/.git/hooks]
```

이때, production release branch를 main으로 설정한다.

Git 고급 – git flow

- **git flow feature start <name>** : 새 feature을 개발할 때 사용, 명령어를 실행하면 feature/<name>의 브랜치가 생성된다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow (develop)
$ git flow feature start setting
Switched to a new branch 'feature/setting'

Summary of actions:
- A new branch 'feature/setting' was created, based on 'develop'
- You are now on branch 'feature/setting'

Now, start committing on your feature. When done, use:

     git flow feature finish setting


jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow (feature/setting)
$
```

Git 고급 – git flow

- **git flow feature finish <name>** : 해당 feature 브랜치를 develop 브랜치에 병합한다. 병합이 완료된 feature 브랜치는 삭제된다.

- **git flow feature finish <name> -k** : feature 브랜치를 삭제하지 않으면서 develop 브랜치에 병합한다.

- **git flow feature finish <name> --no-ff** : fast-forward 방식을 적용하지 않은 채 develop 브랜치에 병합한다.

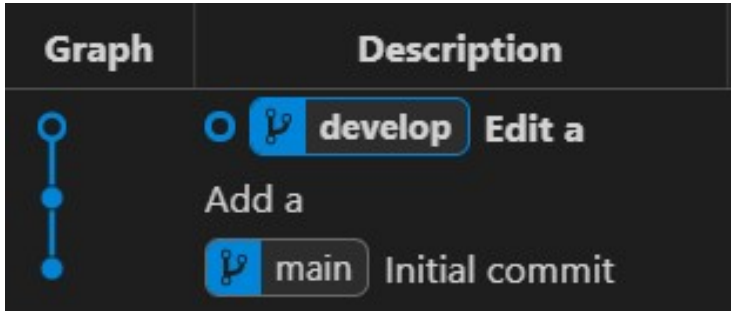
Git 고급 – git flow

- **git flow feature publish <name>** : 해당 feature 브랜치를 remote에 게시한다.

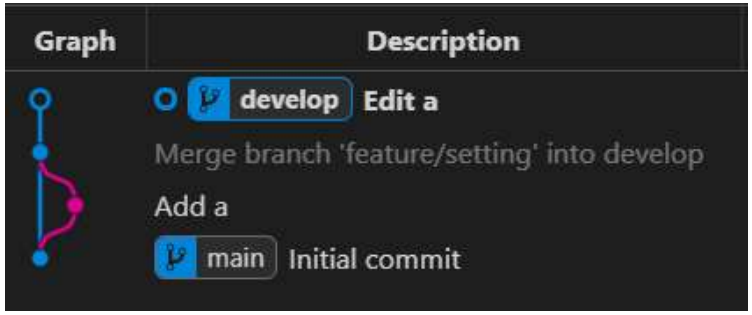
- **git flow feature pull origin <name>** : 해당 feature 브랜치를 remote(origin)에서 가져온다.

Git 고급 – git flow

- fast-forward vs non-fast-forward



git flow finish setting



git flow finish setting --no-ff

Git 고급 – git flow

- **git flow release start <release version>**
  : develop 으로부터 새 release를 분기한다. 명령이 실행되면 release/<release version>이라는 브랜치가 생성된다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow (develop)
$ git flow release start 0.1
Switched to a new branch 'release/0.1'

Summary of actions:
- A new branch 'release/0.1' was created, based on 'develop'
- You are now on branch 'release/0.1'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

     git flow release finish '0.1'


jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/git-flow (release/0.1)
$
```

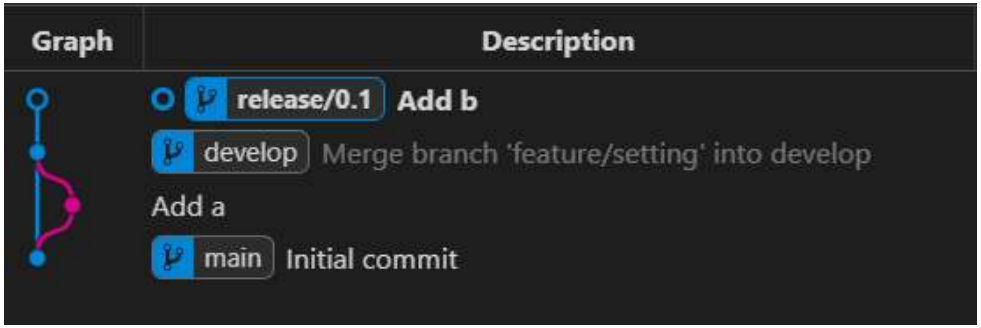
Git 고급 – git flow

- **git flow release finish <release version>** : release를 main에 병합시킨다. 이때 병합과정에서 생기는 커밋에 내용이 release version인 태그가 붙게 된다.

- **git flow release publish <release version>** : 해당 release 브랜치를 remote에 게시한다.

Git 고급 – git flow

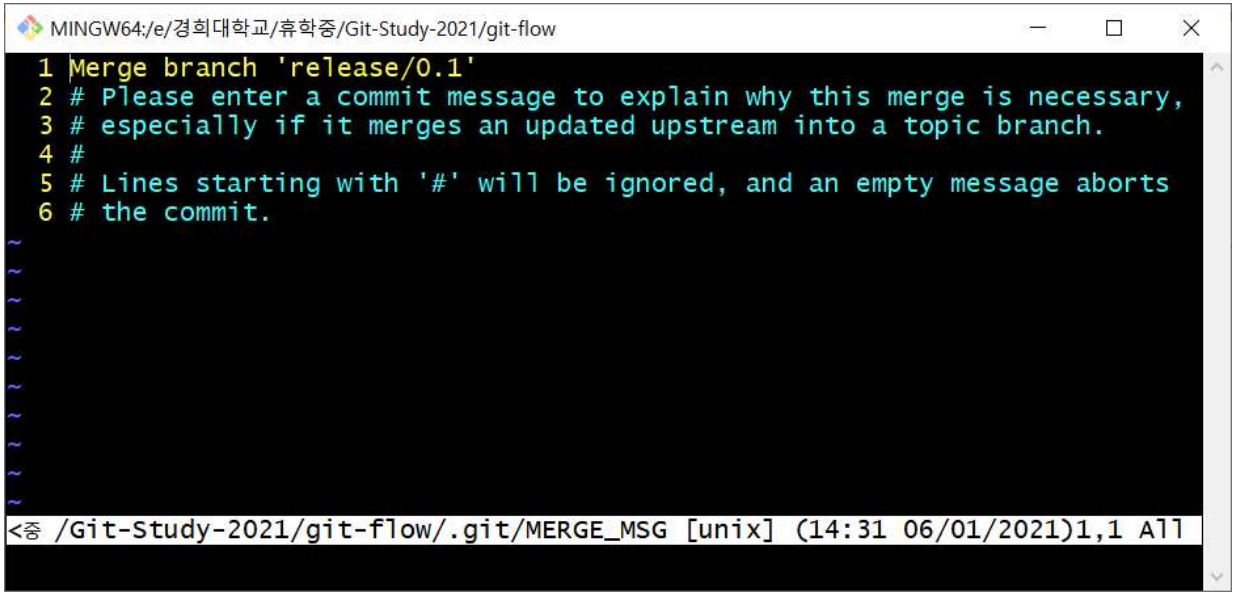
- 사용 예시



release/0.1에서 하나의 커밋을 한 상황

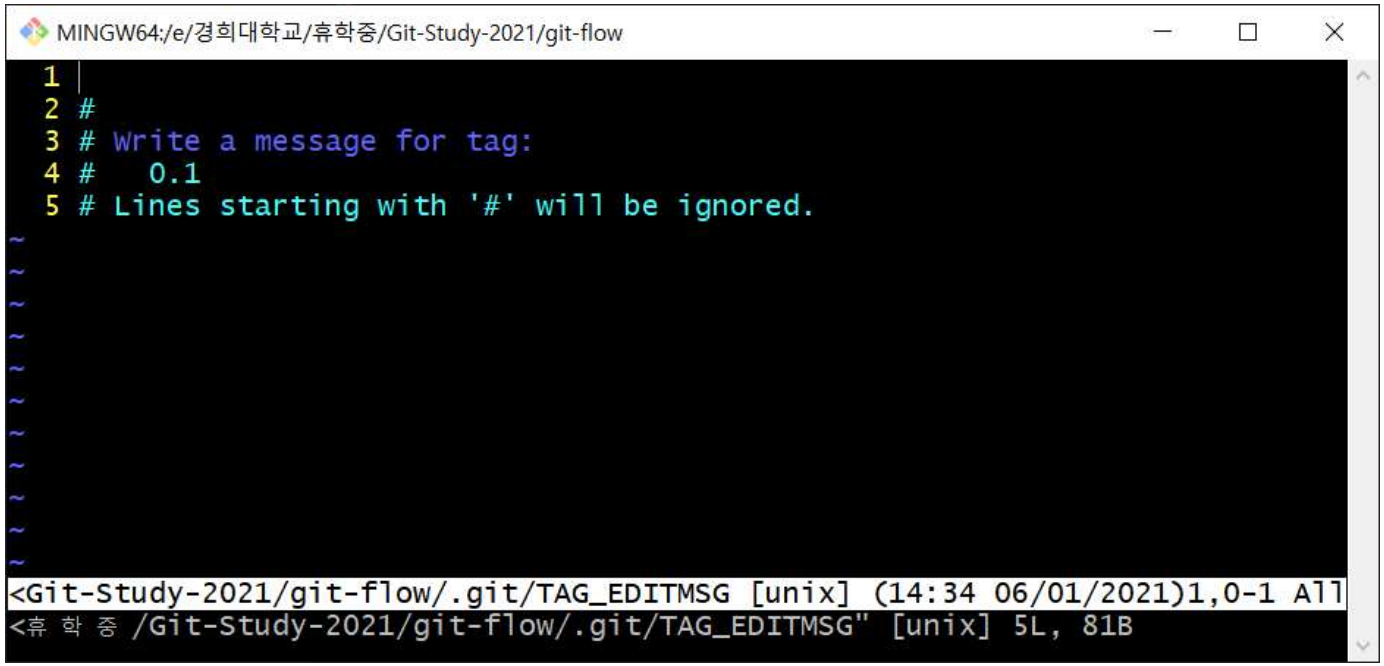
Git 고급 – git flow

- **git flow release finish 0.1**를 하게 되면 총 3개의 vim 화면이 열린다.



1. release를 main에 병합하는 과정에서 커밋 메시지, 그냥 저장하고 나가도록 한다. (:wq)

Git 고급 – git flow



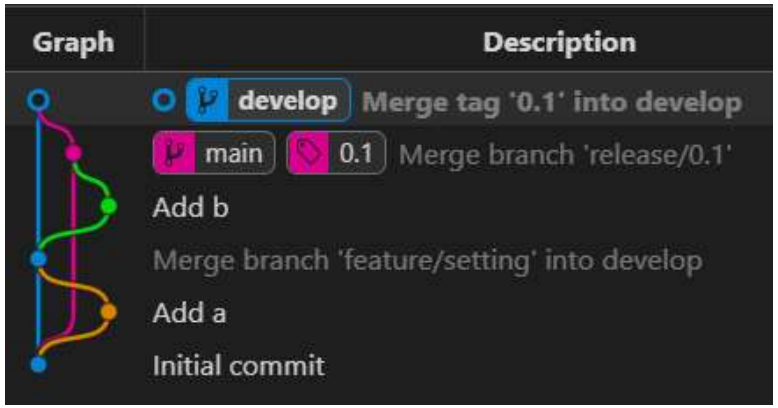
2. 태그를 작성하는 화면,
맨 윗줄에 release version(0.1)을 입력하고
저장하고 나간다. (:wq)

Git 고급 – git flow

3. 다시 main 을 develop에 합병할때의 커밋 메시지,
그냥 저장하고 나가도록 한다. (:wq)

Git 고급 – git flow

- 명령이 정상적으로 수행되면 다음과 같다.



- release 브랜치는 release version을 태그로 하여 main에 병합되었고,
- main 브랜치에서 develop 브랜치로 병합이 이루어졌다.

Git 실습 9 – git flow

- 위 과정을 전반적으로 한번 실습해 봅시다.

1. **git flow init**을 한다.
2. 새로운 feature인 setting을 생성하여 하나의 커밋을 한다.
3. feature/setting을 develop에 병합한다.
4. 버전이 0.1인 release를 생성하여 하나의 커밋을 한다.
5. release/0.1을 finish하여 main과 develop에 반영한다.
   태그는 0.1로 한다.

GitHub - fork

- Repository의 owner나 collaborator가 아니면, 해당 repository를 수정할 수 없음. => fork 사용

- Fork를 통해 다른 사람의 repository를 복사해 올 수 있음

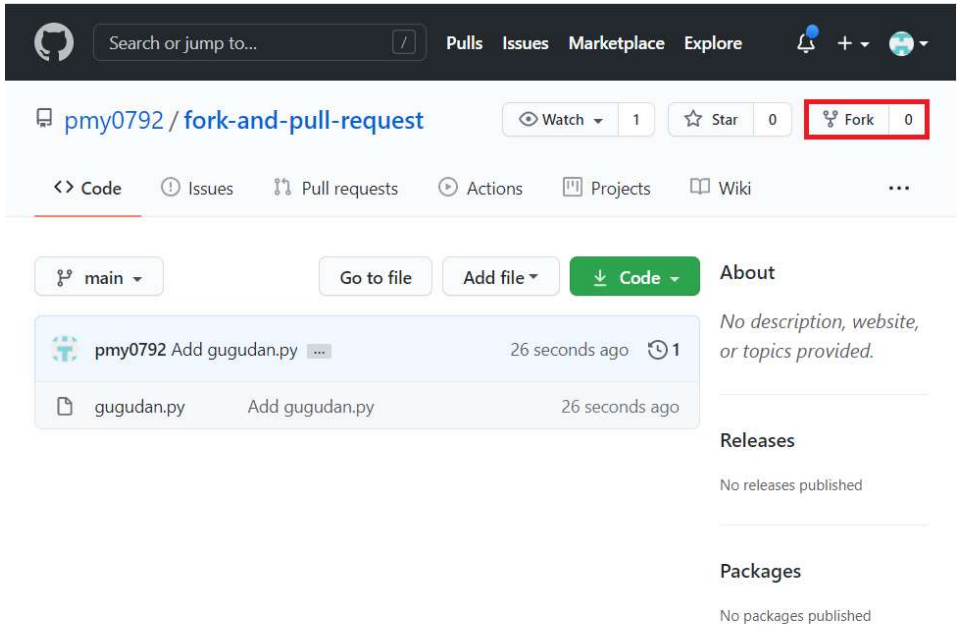
- Fork된 repository는 소유주가 본인으로 변경=>push 가능

ex) pmy0792/repo1을 fork하면 JJuOn/repo1이 됨

- Open Source Contribution은 주로 fork를 통해 이뤄짐

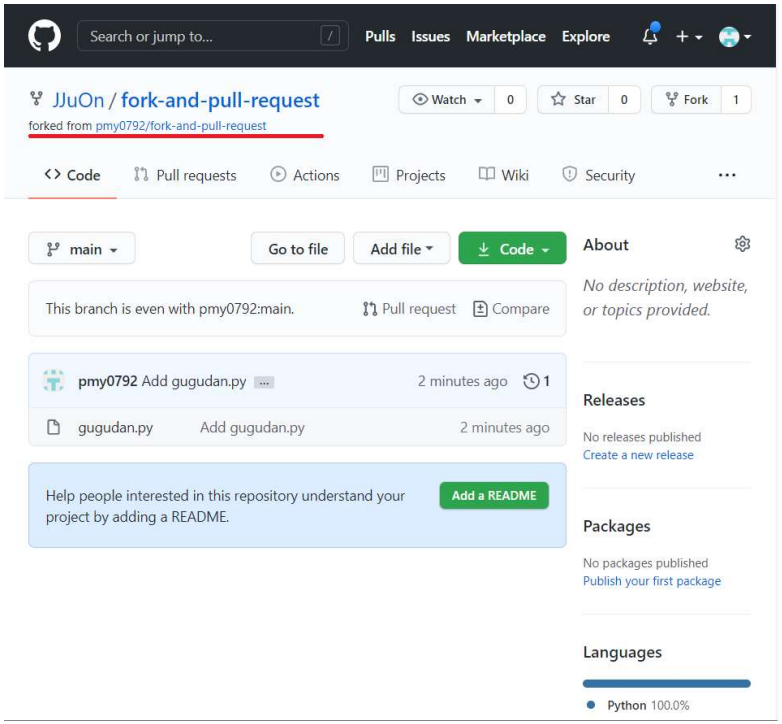
GitHub – fork

- 다른 사람의 repository에서 fork를 클릭, 본인을 선택



GitHub – fork

- 이제 수정 권한이 있는 fork된 repository 생성



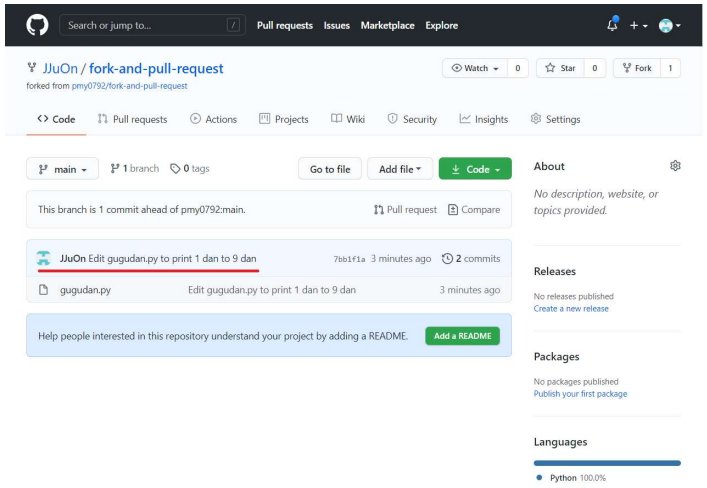
GitHub – fork

- **git clone <fork된 repo의 ssh 주소>**을 통해 작업할 수 있음.

- 해당 repo는 내 소유의 repo이기 때문에 모든 권한이 있다.

```
jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021
$ cd fork-and-pull-request/

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/fork-and-pull-
request (main)
$ vim gugudan.py

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/fork-and-pull-
request (main)
$ git add .

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/fork-and-pull-
request (main)
$ git commit -m "Edit gugudan.py to print 1 dan to 9 dan"
[main 7bb1f1a] Edit gugudan.py to print 1 dan to 9 dan
 1 file changed, 2 insertions(+), 3 deletions(-)

jwseo@DESKTOP-O5QF9NF MINGW64 /e/경 희 대 학 교 /휴 학 중 /Git-Study-2021/fork-and-pull-
request (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 321 bytes | 107.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:JJuOn/fork-and-pull-request.git
   c8a28aa..7bb1f1a  main -> main
```

원본 repo의 소스코드를 일부 변경 후
add ,commit, push



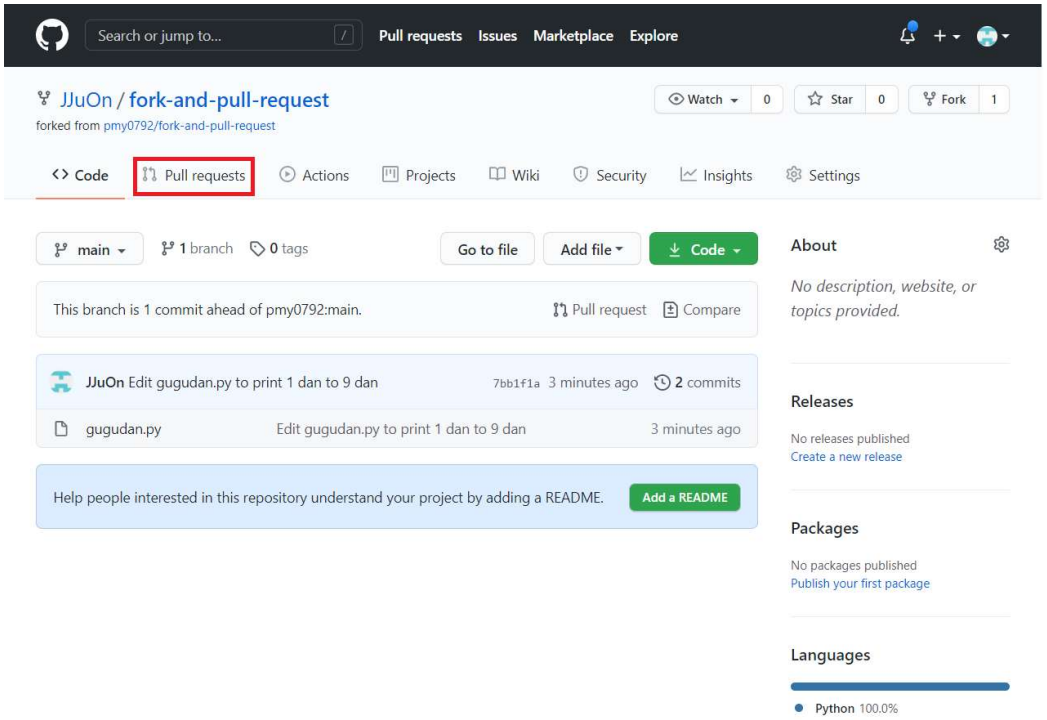
성공적으로 push 된 모습

GitHub – pull request

- Fork된 repository를 수정한 후, 원본 repository에 반영하기 위해 pull request 사용.
- Pull request 하는 도중, 소스코드에 대한 검토가 이루어짐.
- 또한 pull request는 협업할 때,
  remote에서 브랜치간 병합에도 자주 사용된다.

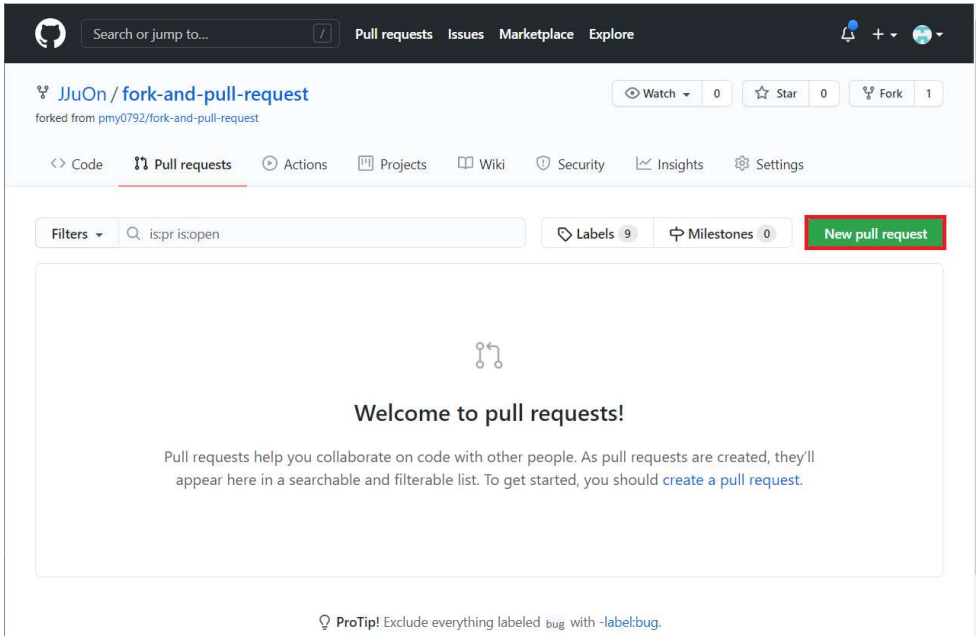
GitHub – pull request

- 자신의 fork된 repo에서 pull request를 클릭



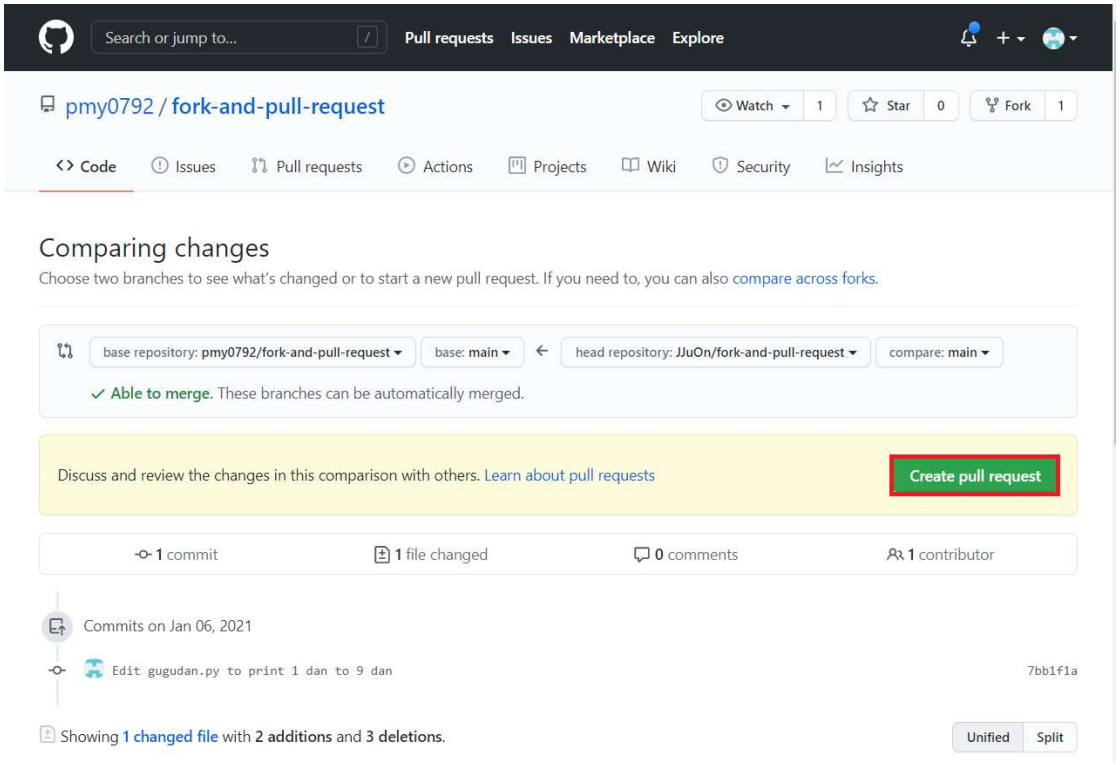
GitHub – pull request

- New pull request 클릭



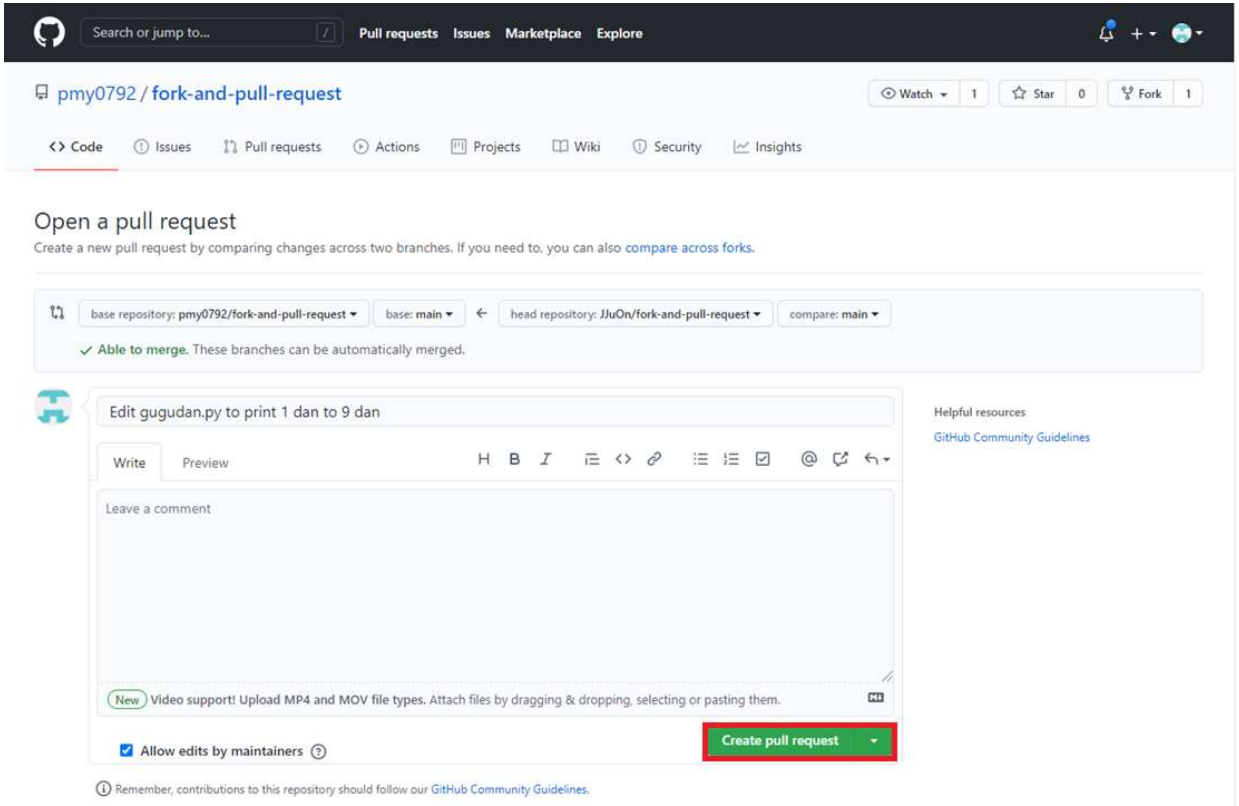
GitHub – pull request

- 자동으로 원본 repo로 이동하게 된다. Create pull request 클릭



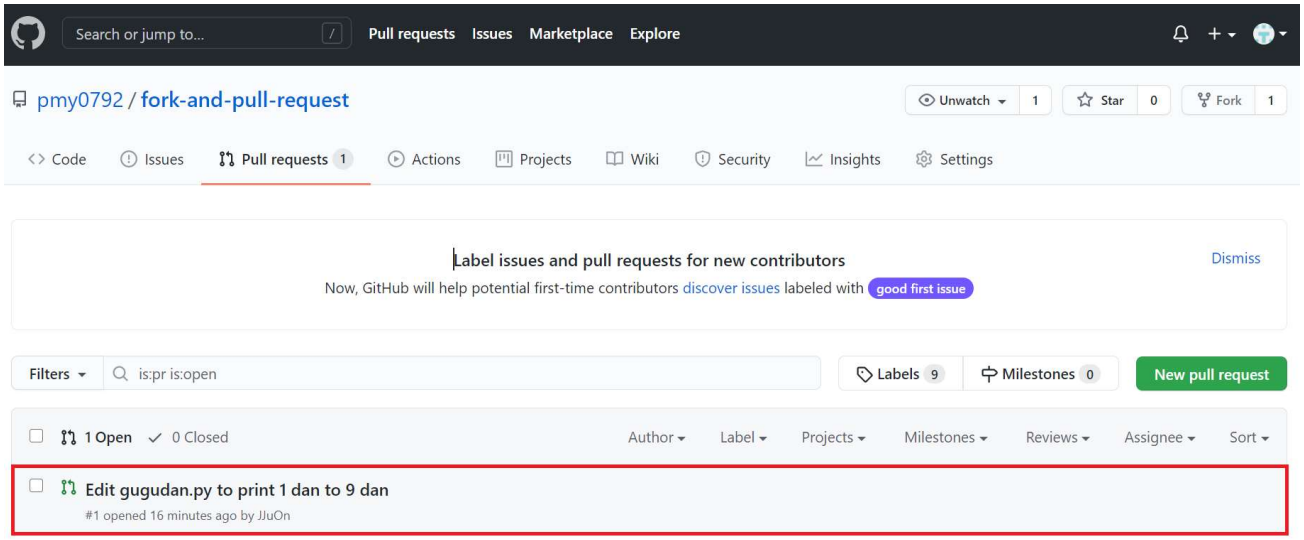
GitHub – pull request

- Create pull request 클릭



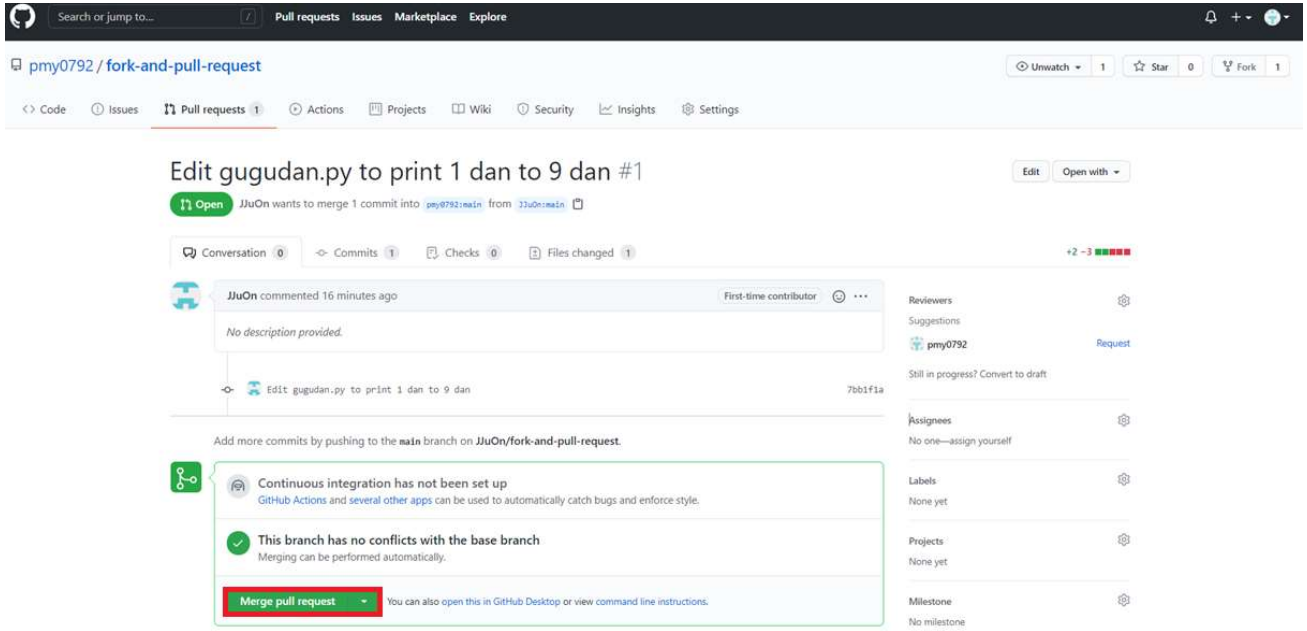
GitHub – pull request

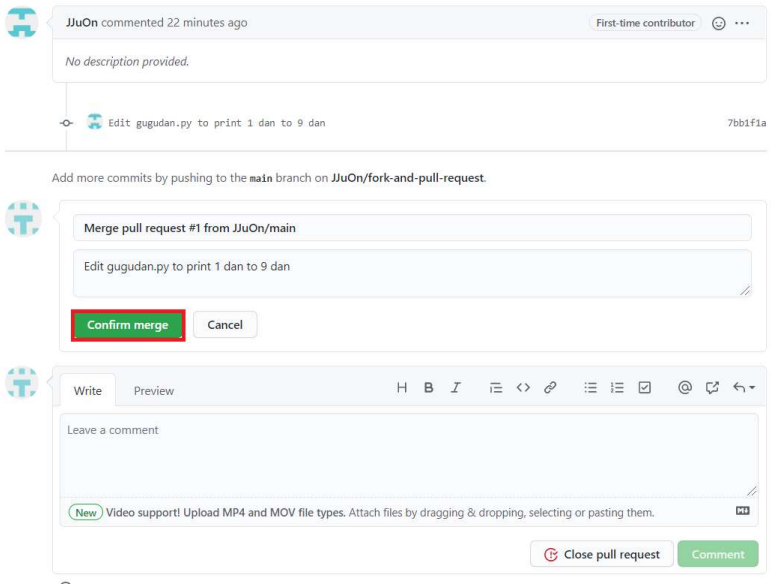
- 원본 repo의 owner는 pull request 탭에서 해당 pull request를 클릭



GitHub – pull request

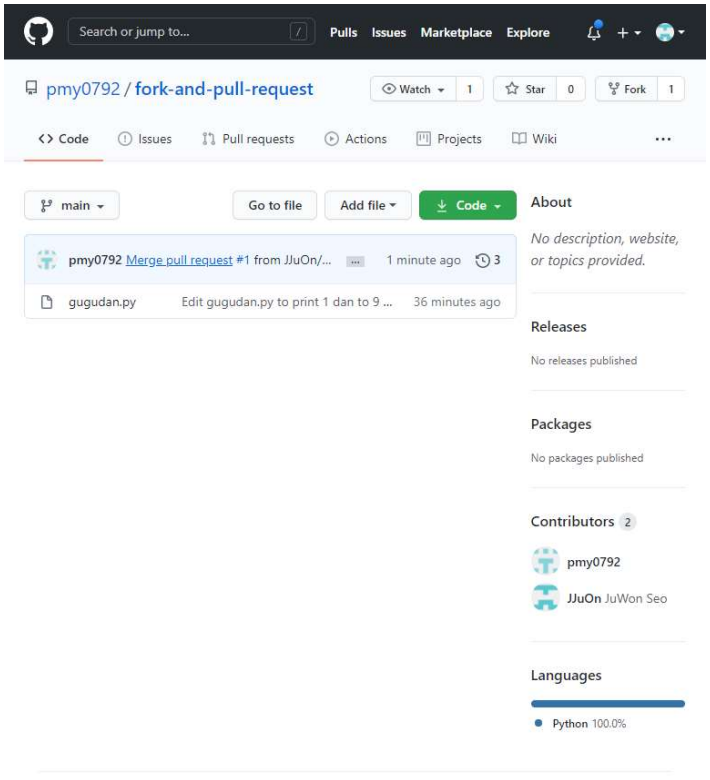
- Merge pull request, confirm merge를 클릭하여 원본 repo에 반영





GitHub – pull request

- 수정 사항이 반영된 모습, contributor에 추가 되었다.



Git 실습 10 – fork & pull request

1. 2인이 짝을 이룬다.
2. 각자 GitHub에서 새 repository를 생성하고, 텍스트 파일 하나를 commit한 다음 push도 한다.
3. 서로의 repository를 **fork**한 후, 텍스트 파일을 수정한 다음 commit, push한다.
4. 서로의 원본 repository로 **pull request**를 하고, 확인 후 **confirm merge**한다.

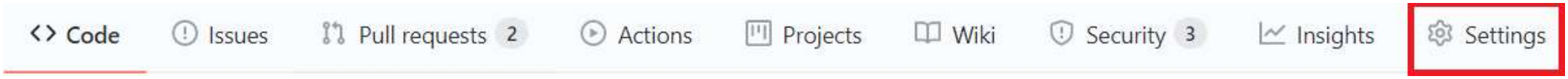
GitHub – collaborate

- GitHub에서는 collaborator를 등록하여
  remote repo에 대한 편집 권한을 부여할 수 있음.

- fork & pull request 또는 collaborate를 통해
  repository에 자신의 커밋이 반영되면
  해당 repository의 contributor가 된다.

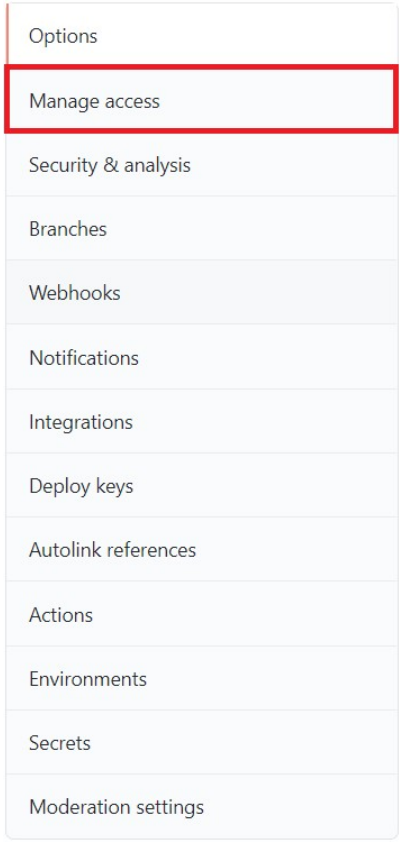
GitHub – collaborate

- GitHub repository 페이지에서 settings 클릭



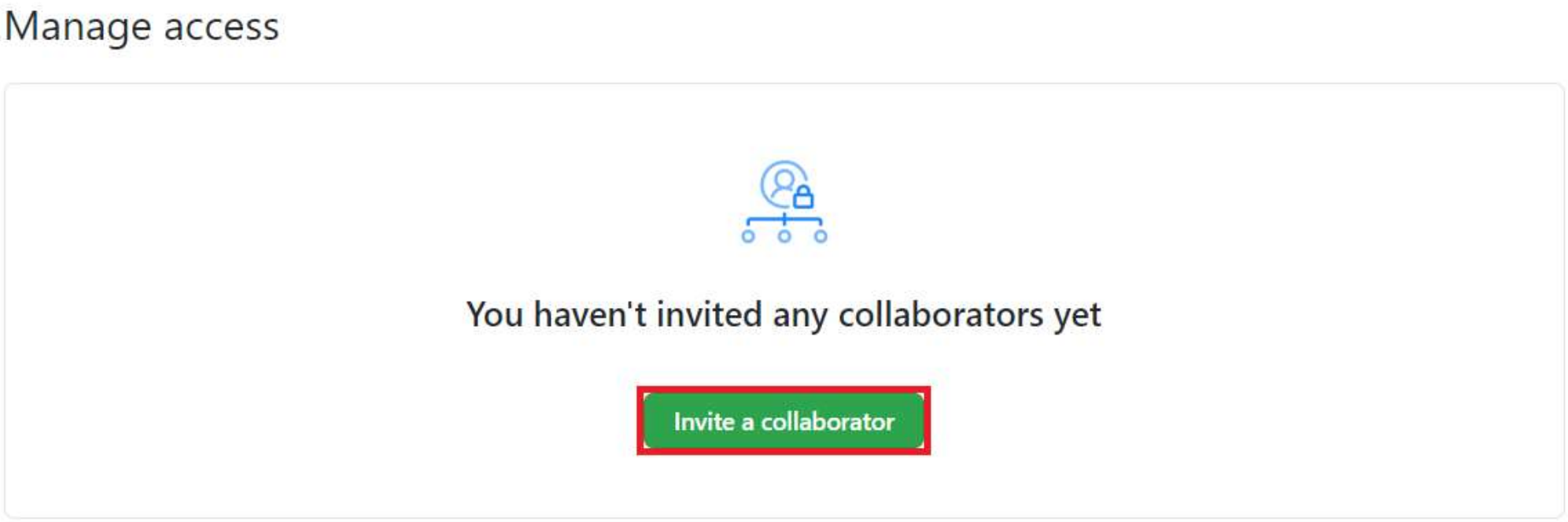
GitHub – collaborate

- Manage access



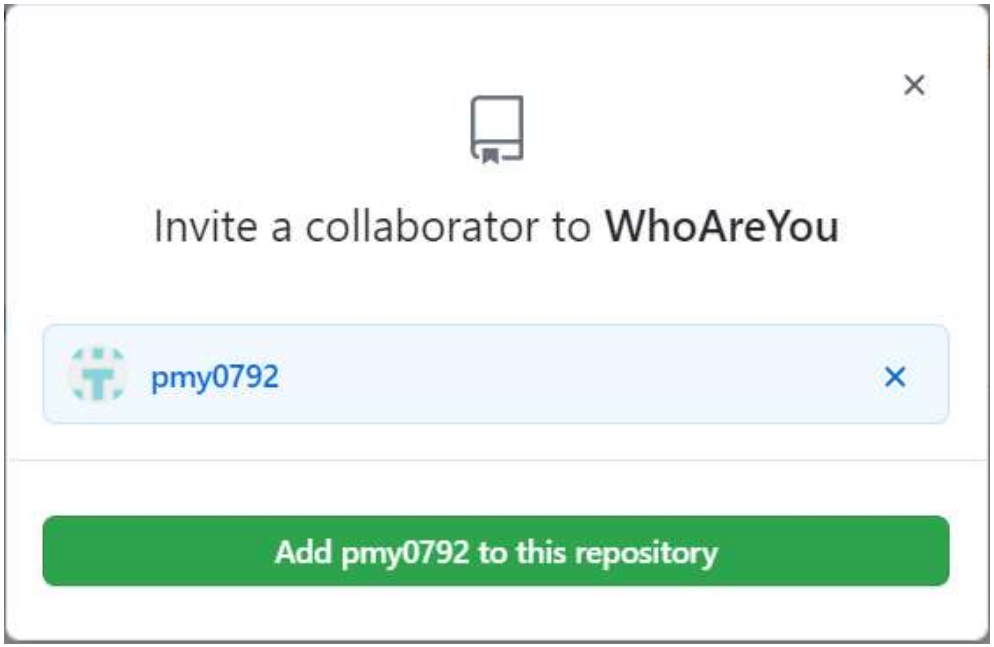
GitHub – collaborate

- Invite a collaborator



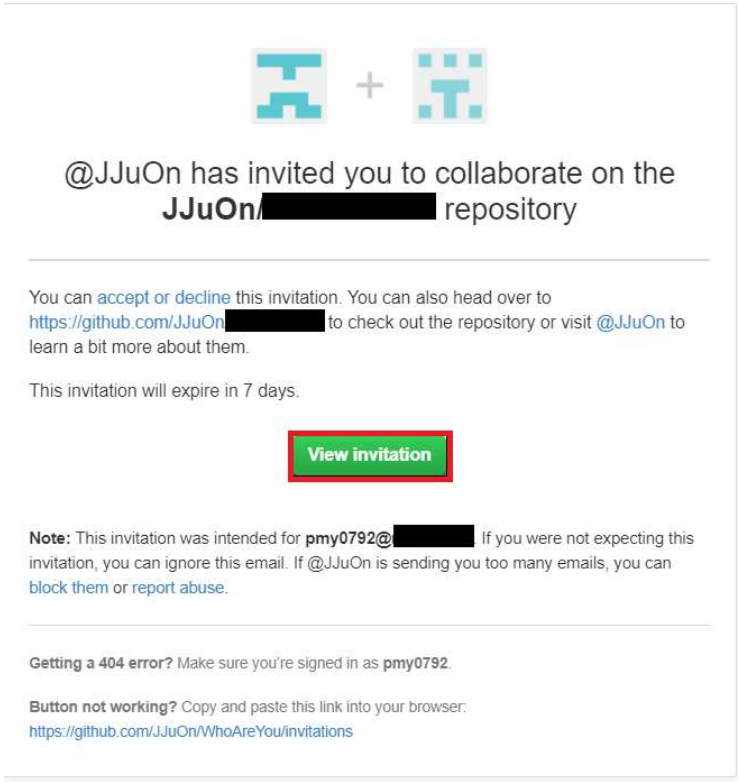
GitHub – collaborate

- 초대할 사람 검색 (username, email 등)



GitHub – collaborate

- 초대받은 사람은 이메일 들어가서 확인한다.



@JJuOn has invited you to collaborate on the **JJuOn/[redacted]** repository

You can accept or decline this invitation. You can also head over to https://github.com/JJuOn[redacted] to check out the repository or visit @JJuOn to learn a bit more about them.

This invitation will expire in 7 days.

View invitation

Note: This invitation was intended for **pmy0792@[redacted]**. If you were not expecting this invitation, you can ignore this email. If @JJuOn is sending you too many emails, you can block them or report abuse.

Getting a 404 error? Make sure you're signed in as **pmy0792**.

Button not working? Copy and paste this link into your browser:
https://github.com/JJuOn/WhoAreYou/invitations

Git 실습 11 – collaborate

1. 2인이 짝을 이룬다.
2. 각자 GitHub에 새로운 repo를 생성한 다음, collaborator로 서로를 등록한다.
3. 서로의 repo를 clone한 후, 텍스트 파일 하나를 생성하여 커밋한다.

참고자료 – git 연습하기 좋은 사이트

- https://learngitbranching.js.org/?locale=ko

- 좌측 터미널에 **sandbox**를 입력하면 자유롭게 연습할 수 있다!

수고하셨습니다!