Test Automation Basics

테스트 자동화 기초

08 Git과 Jenkins를 활용한 CI/CD 파이프라인 구축

목차 08 Git과 Jenkins를 활용한 CI/CD 파이프라인 구축

주간 수업 목표

8주차 수업내용

- Git의 기본 개념과 기본 사용법을 이해합니다.
- Git을 통해 다른 사람들과 함께 테스트 스크립트를 작성하고, 공유합니다.
- Git branch 전략에 대해 알고, 효율적으로 협업하는 방법을 익힙니다.
- CI/CD의 기본 개념에 대해 이해합니다.
- Jenkins의 기본 개념과 기본 사용법을 알고, 기본적인 CI/CD 파이프라인을 구축해봅니다.
- Git과 Jenkins를 사용해 CI/CD 파이프라인을 구축하고 자동화 테스트 스크립트를 통합합니다.

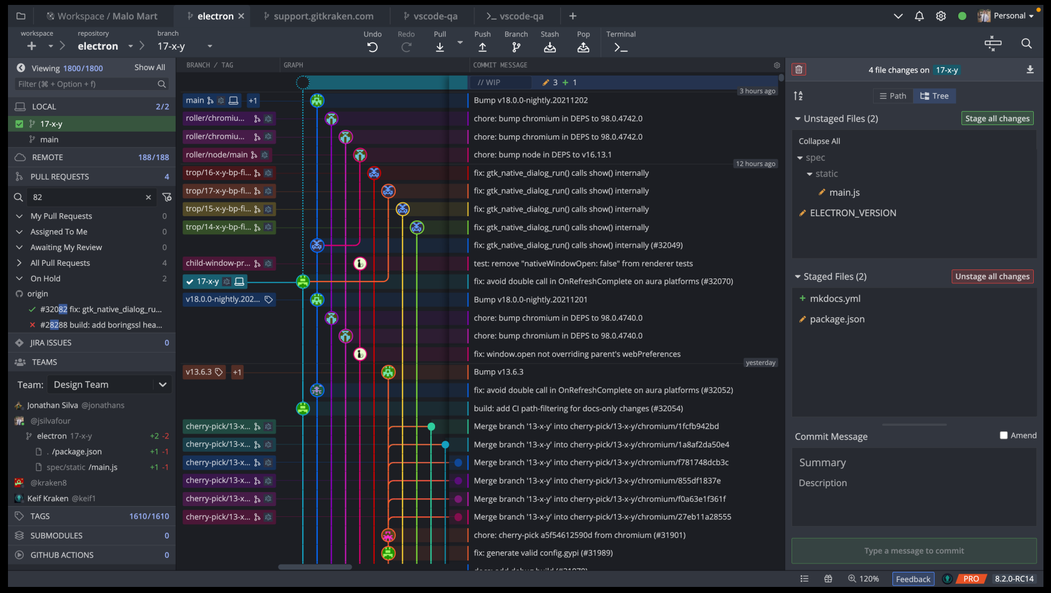
1. Git 개념 및 활용

오늘의 수업 목표

1일차 수업내용

- Git의 기본 개념과 기본 사용법을 이해합니다.
- 원격 저장소와 로컬 저장소의 차이점과 특징에 대해 이해합니다.
- Git을 설치하고 초기 설정을 완료할 수 있습니다.
- Git의 기본 명령어(add, commit, push, pull)를 익힙니다.
- Git을 통해 다른 사람들과 함께 테스트 스크립트를 작성하고, 공유합니다.
- 실습을 통해 Git의 기본 워크플로우를 반복 경험합니다.
- 협업 시 유용한 명령어에 대해 배웁니다.

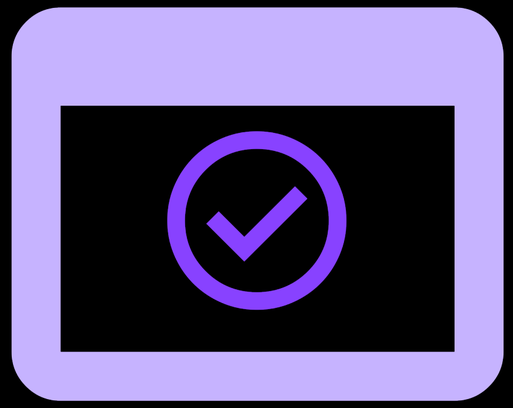
What is Git?



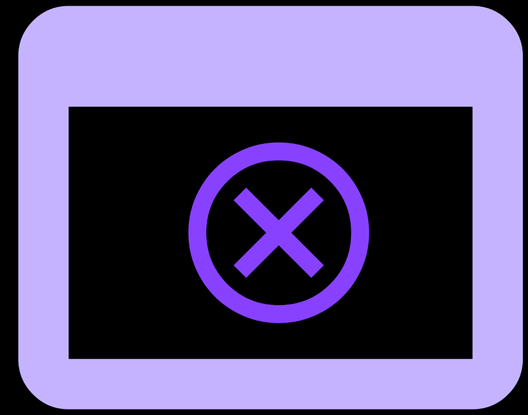
- 리눅스를 만든 '리누스 토르발스'가 개발한 시스템
- 분산 버전 관리 시스템
- 여러 명이 하나의 프로젝트를 개발할 때, 소스코드의 변경 내역을 추적하고 관리하여 버전을 관리할 수 있는 도구

Why Git?

1. 변경 이력 관리



Version 1.0 – 정상 실행

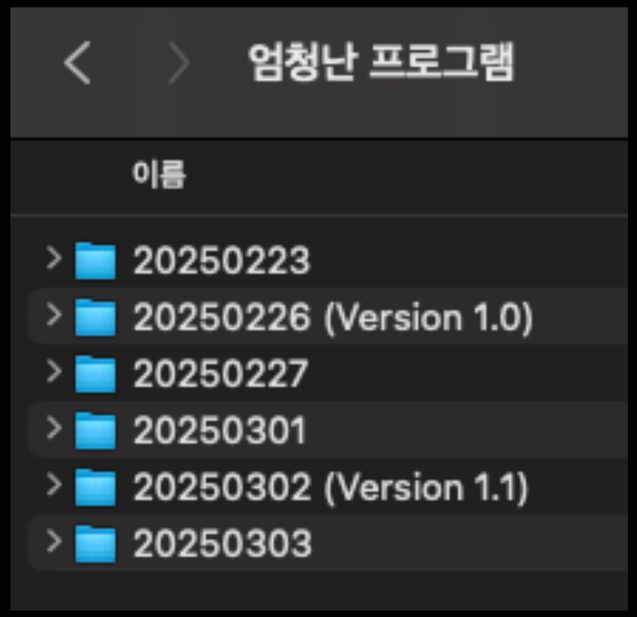


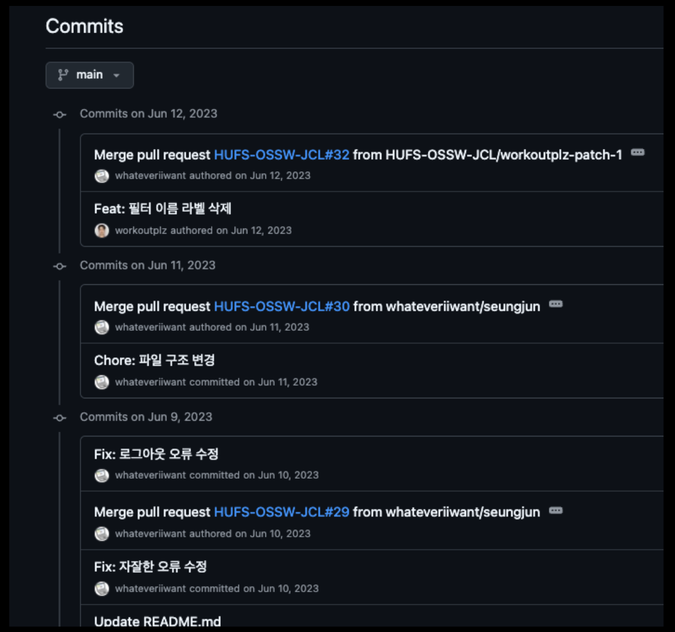
Version 1.1 – 오류

버전 업데이트 후 치명적 오류 발생 시,
(1) 이전 버전으로 변경 필요 (2) 문제 추적 필요

Why Git?

1. 변경 이력 관리



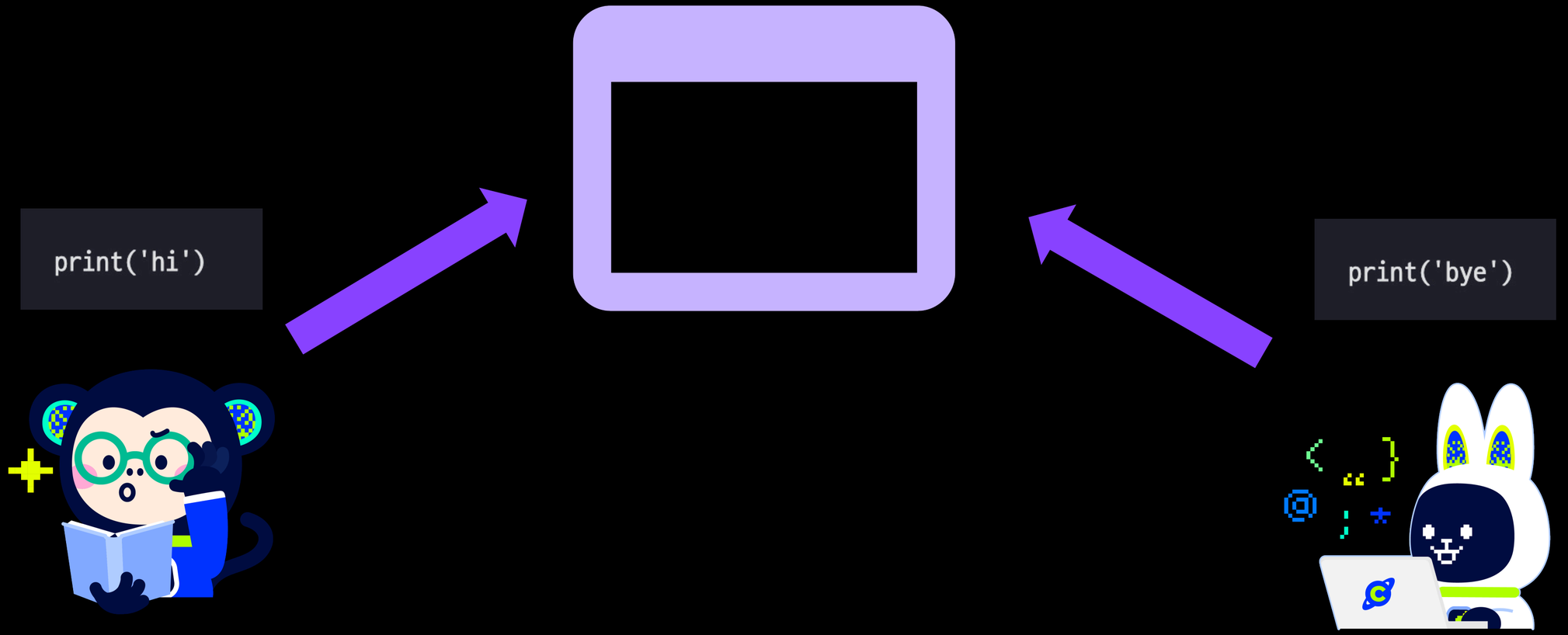


- 평소 작업하는 코드를 날짜 별로 폴더와 파일 생성 필요
- 이 자체만으로 굉장히 번거로운 일

- 일자 별로 변경 이력을 모두 확인 할 수 있음
- 이전 내역으로 손쉽게 복구 가능
- 문제가 발생한 버전과 직전 코드 비교 가능

Why Git?

2. 협업에 용이

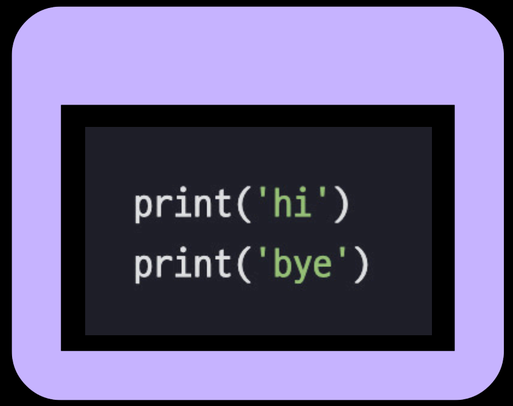


2025년 1월 1일 0시 0분 저장

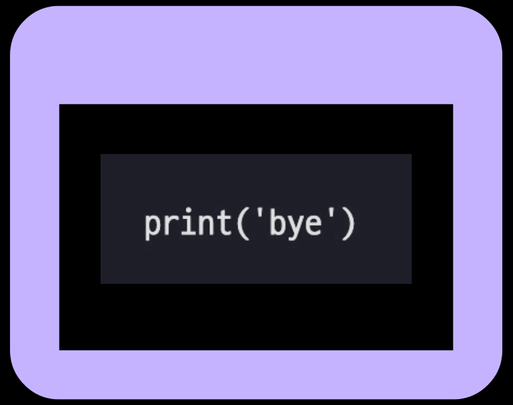
2025년 1월 1일 0시 1분 저장

Why Git?

2. 협업에 용이



우리가 원했던 결과물
(Git 사용 시)



실제 결과물
(Git 미사용 시)

여러 명이 동시에 작업해도 작업 내역이 독립적으로 관리, 이후 코드 통합 가능

What is GitHub?

!=

Git : 분산 관리 시스템 (오프라인 사용, 로컬에 직접 설치 필요)

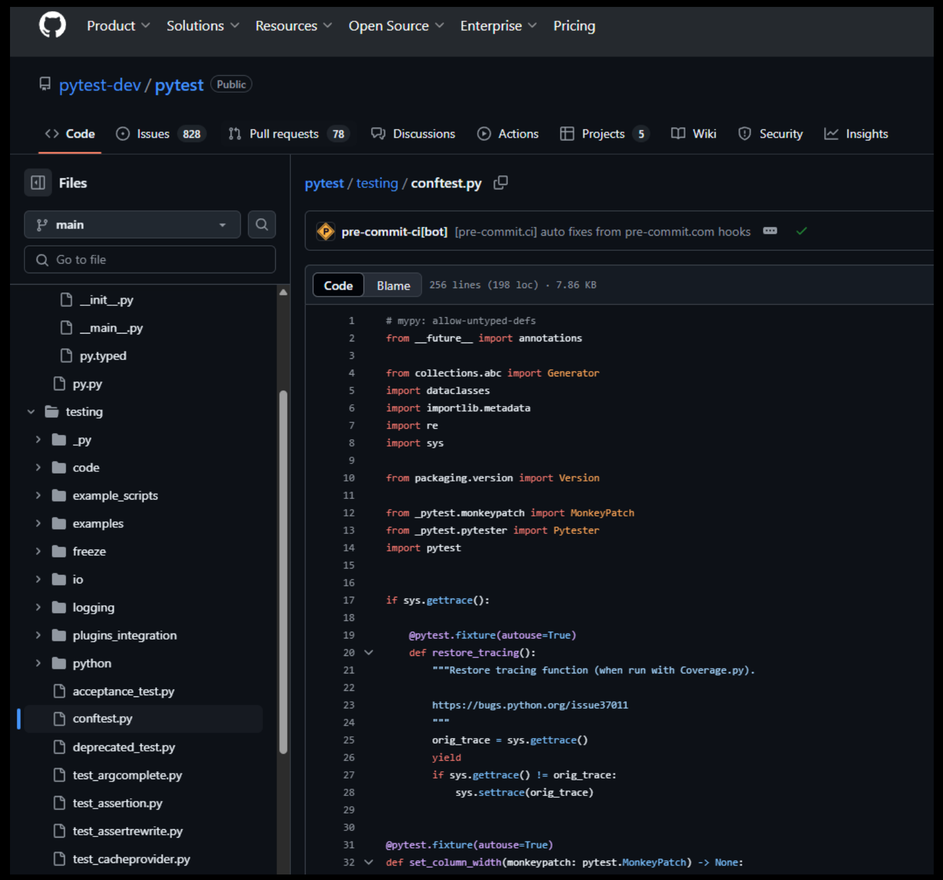
GitHub : Git을 기반으로 한 온라인 저장소, Git을 온라인으로 옮긴 것

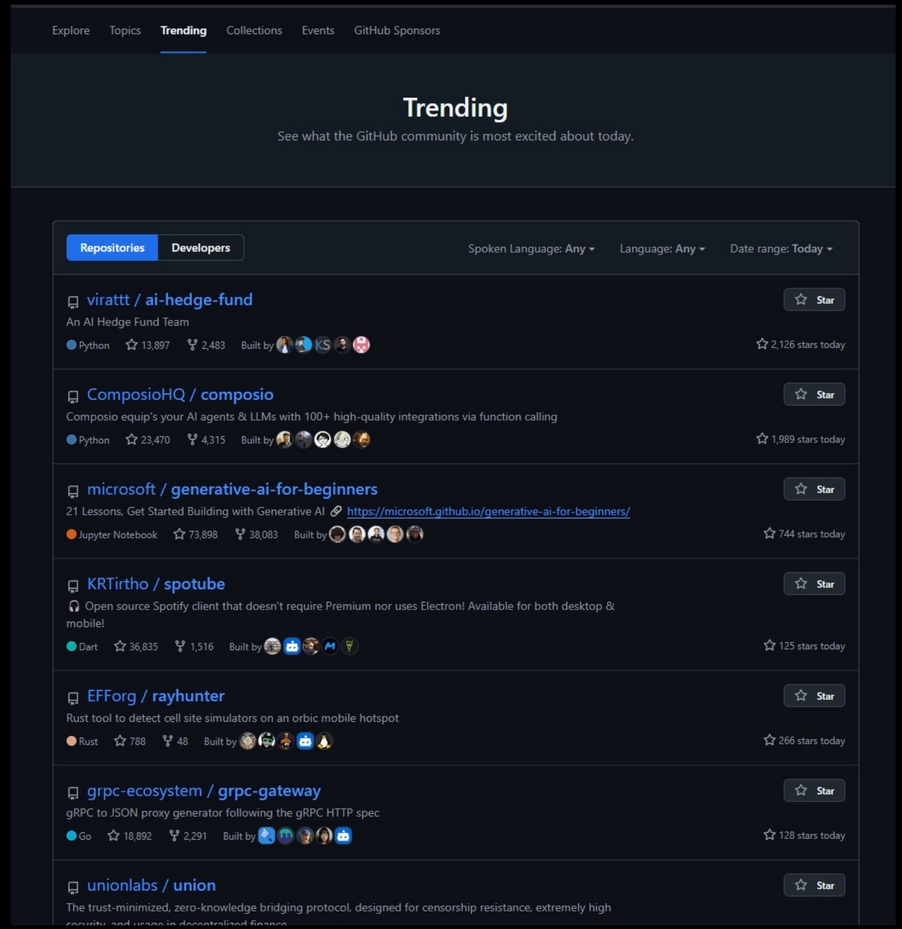
What is GitHub?

GitHub, GitLab, GitKraken, Bitbucket 등
Git 기반의 다양한 서비스들이 존재.

그 중 개인이 가장 접하기 쉽고 많이 쓰는 것이
GitHub

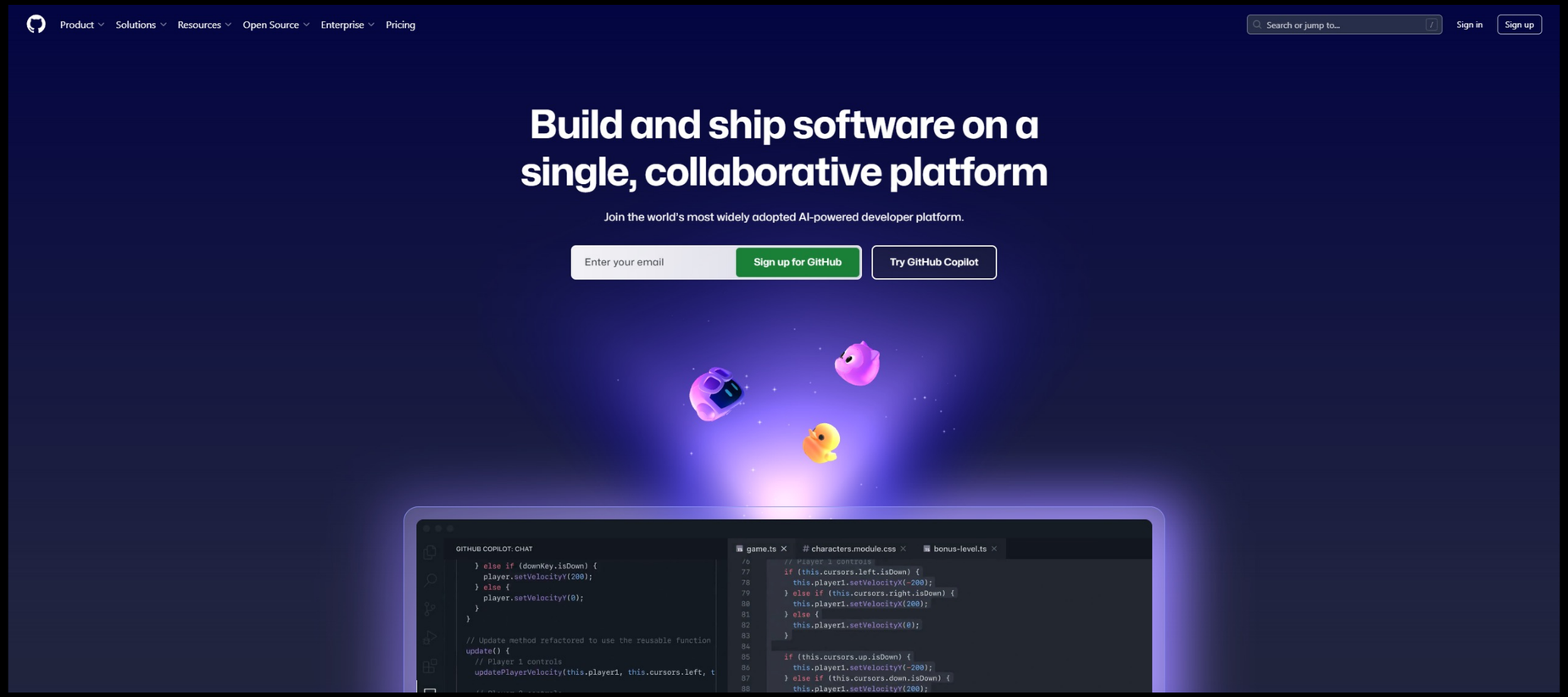
What is GitHub?





많은 사람들이 본인이 직접 개발한 결과물을 오픈소스로 공개

[실습1] GitHub 사용 준비



https://github.com/ 접속 후 가입

[실습1] GitHub 사용 준비

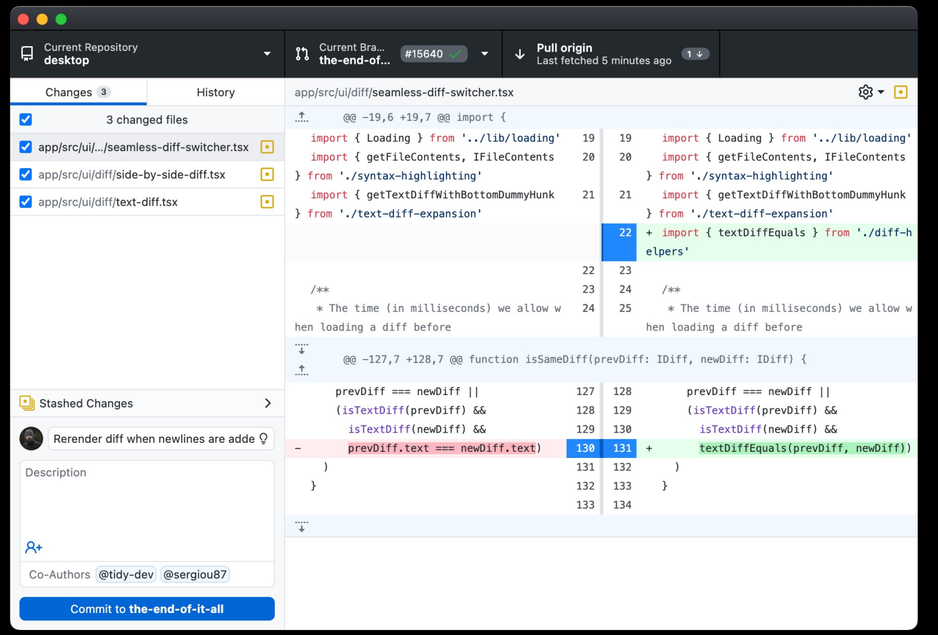

```
$ gh pr status

Current branch
    There is no pull request associated with [develop]

Created by you
    #1011  Update readme [readme-fix]
    - Checks pending - Review required

Requesting a code review from you
    #1015  Improve error handling [better-error-handling]
    ✓ Checks passing + Changes requested
```

명령어 기반의 CLI

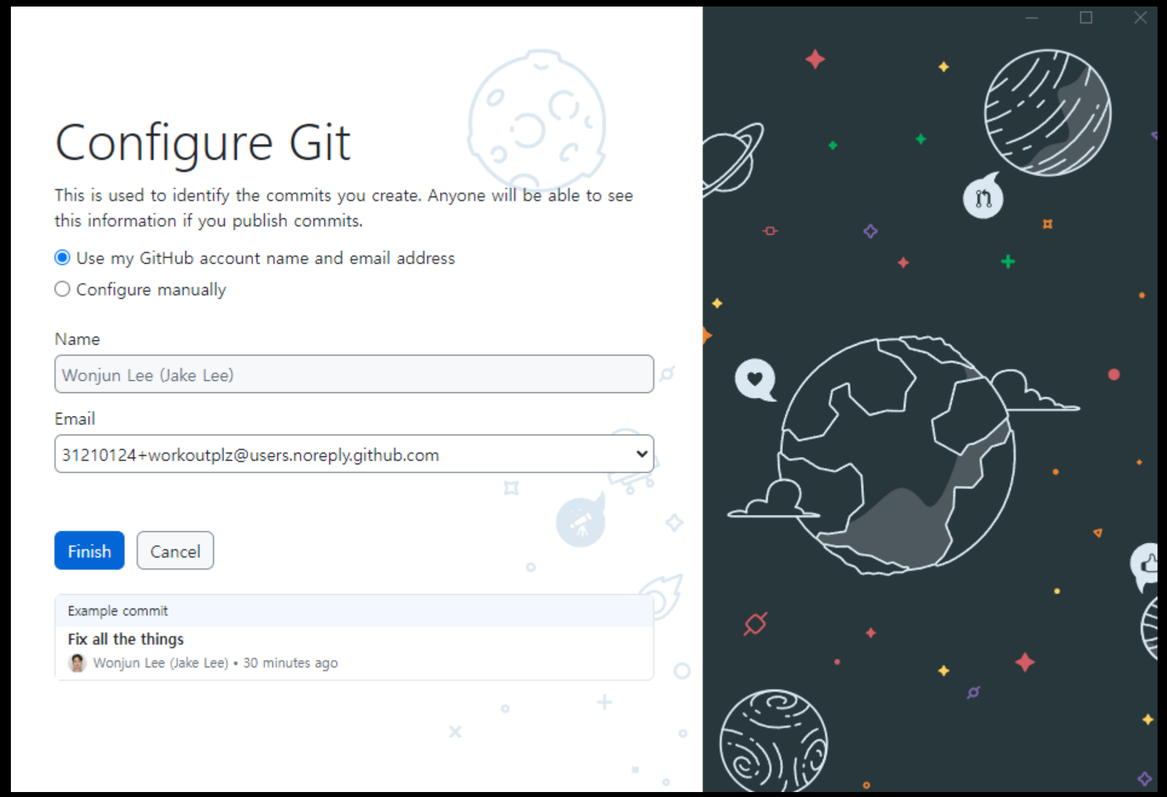


GUI 기반의 GitHub Desktop

⇒ 본 수업에서는 비교적 다루기 쉬운 GitHub Desktop 위주로 진행

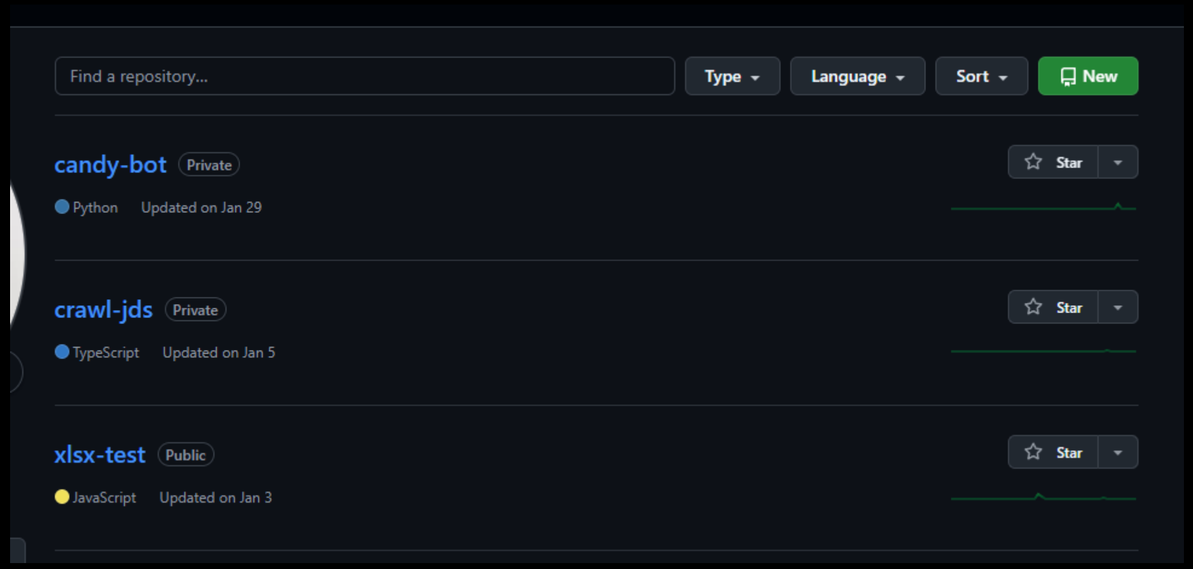
https://github.com

[실습1] GitHub 사용 준비



- https://desktop.github.com/download/ 에서 다운로드
- GitHub Desktop 설치 시, 왼쪽 화면과 같이 설정

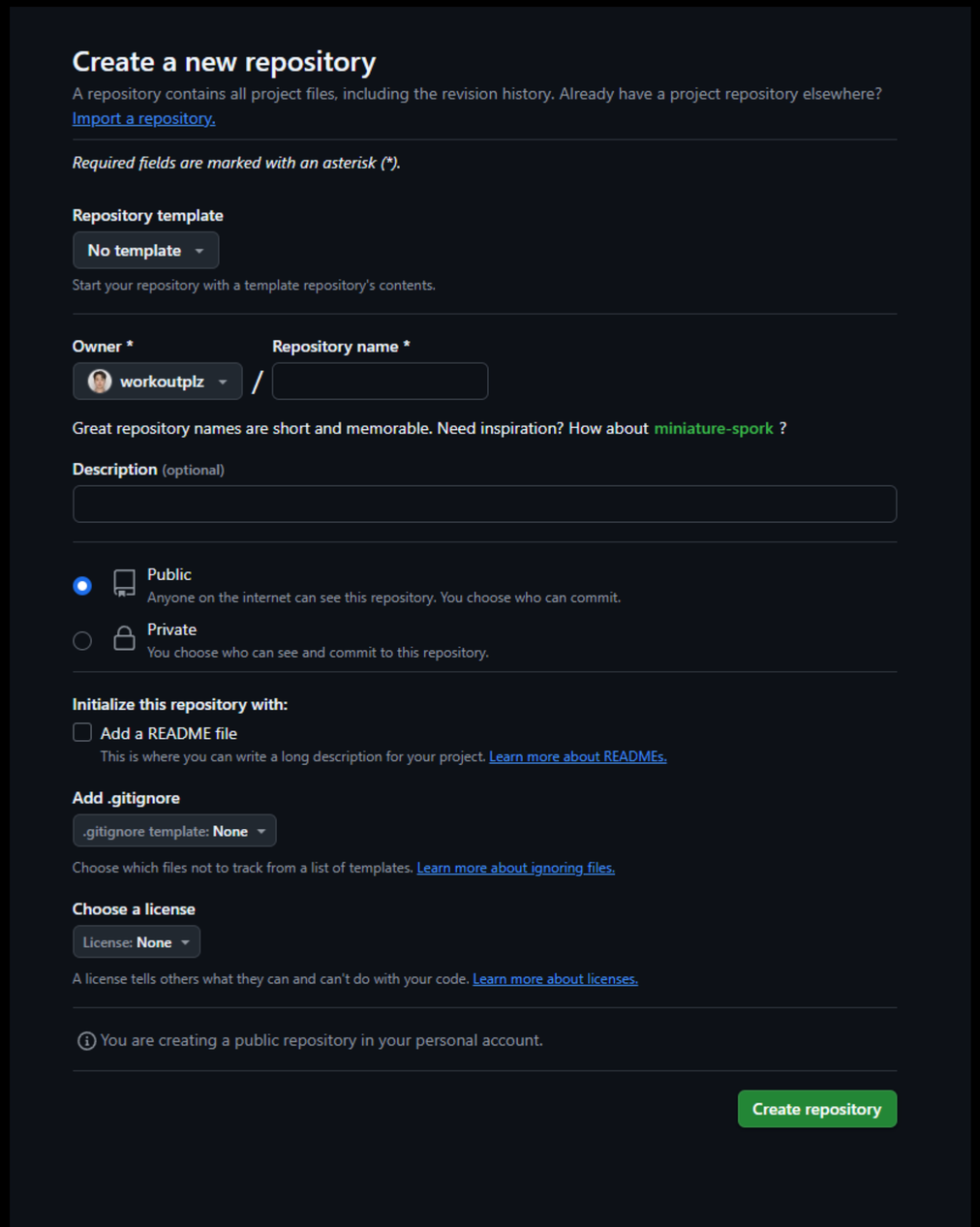
첫 Repository 만들기



레포지토리(Repository)란?

- 하나의 코드 저장소
- 로컬 파일 기준, 하나의 폴더
- 프로젝트 하나 = Repository 하나
- 기술 스택 등에 따라 같은 프로젝트 내 여러 개의 Repository 생성
- 줄여서 '레포(repo)' 라고도 표현

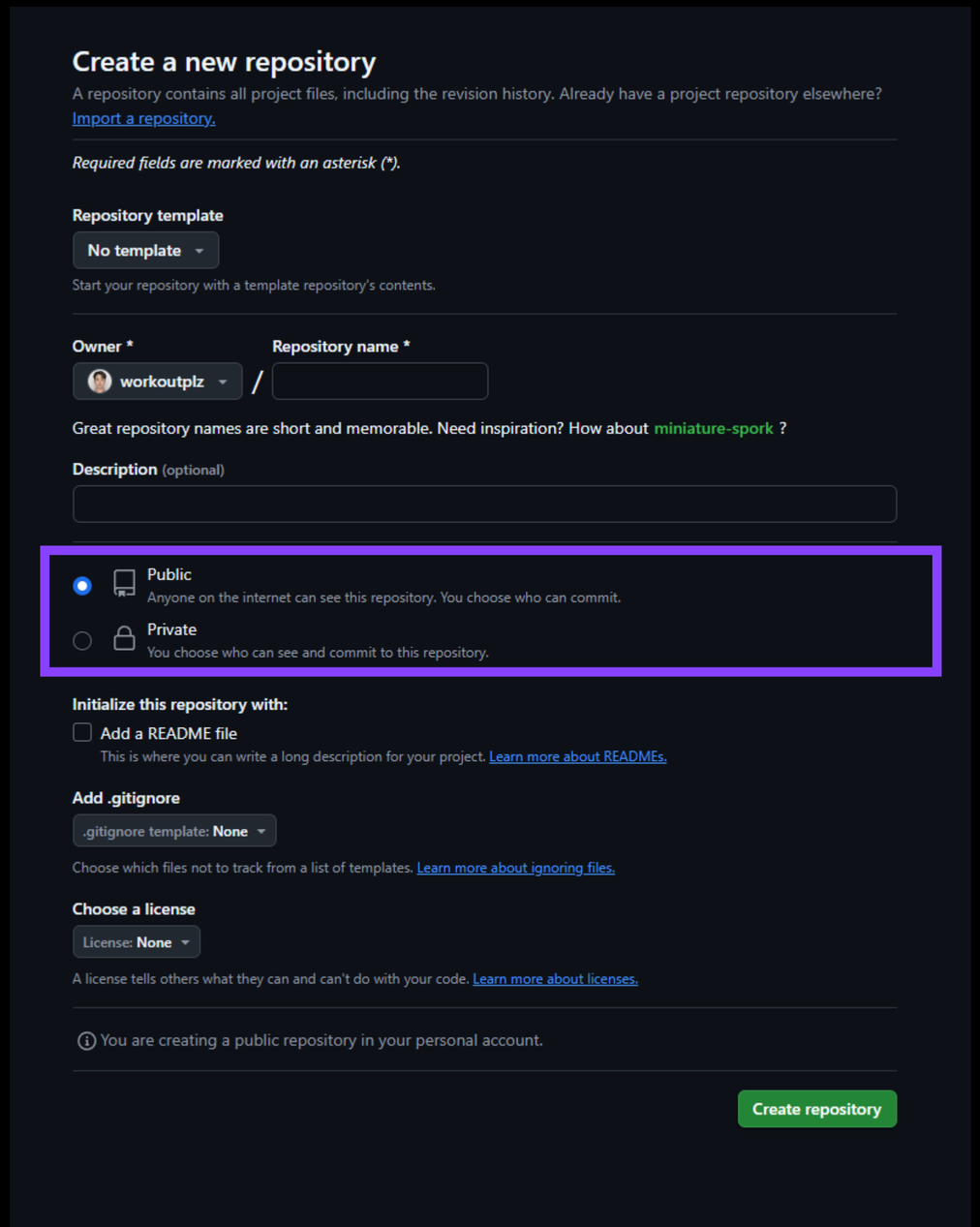
첫 Repository 만들기



Repository 생성 시 설정해줘야 하는 항목

- Template
- Owner
- Repository name
- Description
- 공개 범위 (Public / Private)
- README file initialize
- .gitignore template
- license

첫 Repository 만들기



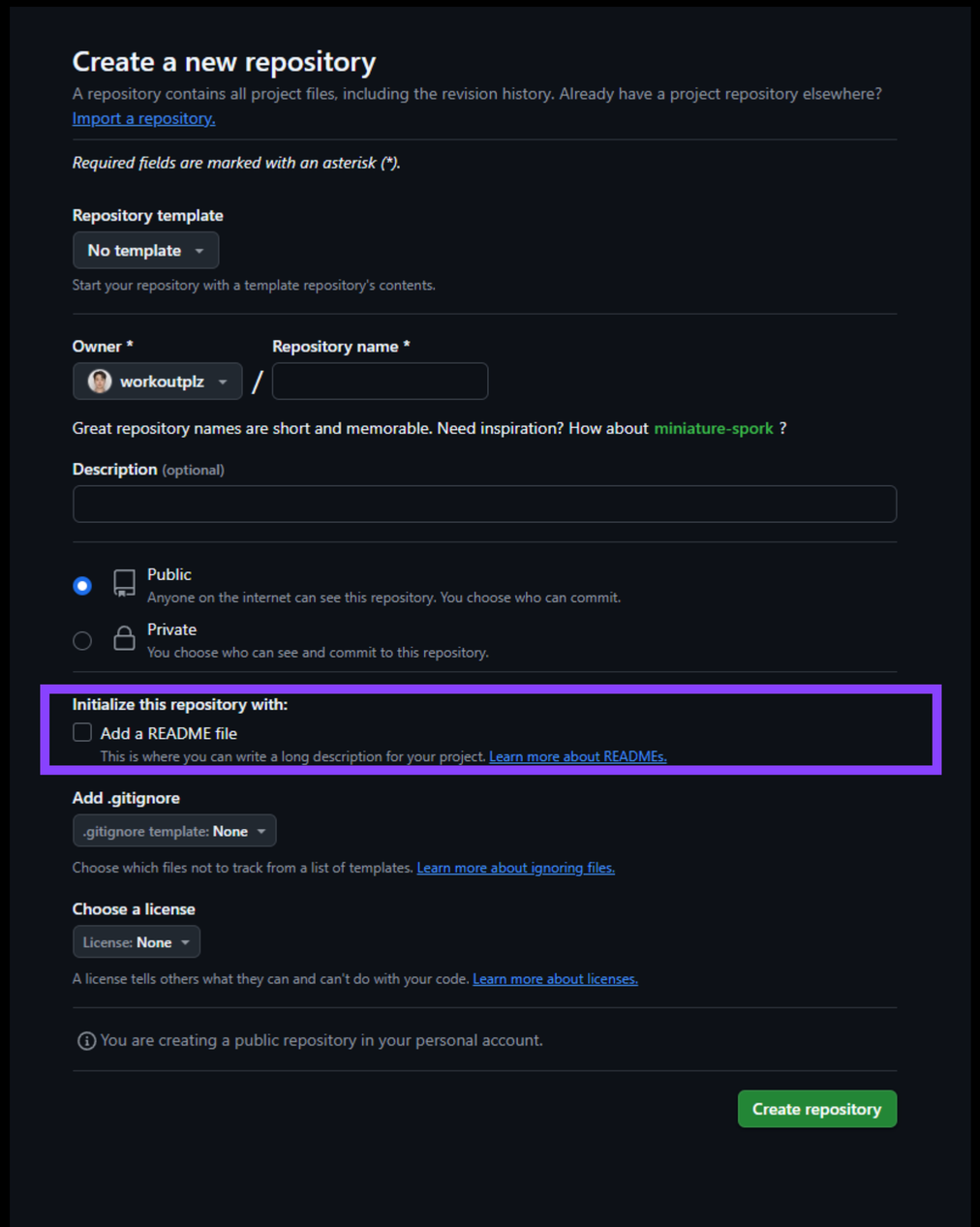
공개 범위가 Public 일 경우

- Repository를 누구나 조회 가능
- 하나의 포트폴리오가 되기도 함
- 외부 서비스 연동 시 비교적 간편
- 민감 정보는 절대 업로드 X (패스워드 등)

공개 범위가 Private 일 경우

- Repository를 특정 사용자만 조회 가능
- 외부 노출이 안되어 비공개 프로젝트 가능
- 무료 사용자에게는 기능이 일부 제한
- 주로 사내 업무는 Private

첫 Repository 만들기



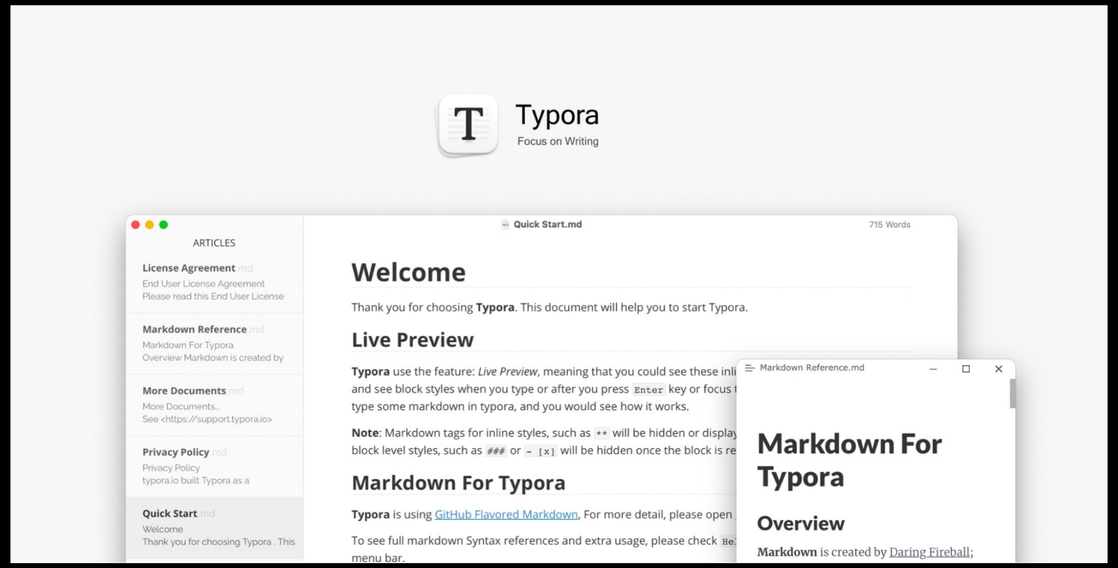
README?

- Repository에 대해 설명하는 글
- 설치 및 실행 방법, 사용법, 기여하는 방법, License 및 저작권 정보 제공
- 사용자, 개발자가 프로젝트에 대해 이해하기 쉬워짐
- 프로젝트 신뢰도와 완성도가 올라가는 효과
- 원활한 커뮤니케이션을 위한 기본기
- 주로 Markdown 문법으로 작성
- 좋은 README에 대한 참고 링크
  https://github.com/matiassingers/awesome-readme

README - Markdown

Markdown?

- 일반 텍스트 문서 내에서 간단한 표기법을 사용해 서식을 지정할 수 있도록 한 마크업 언어
- 문법이 간결해 텍스트를 손쉽게 서식화 할 수 있음
- GitHub 뿐만 아니라 블로그, 메모 앱 등에서 폭 넓게 사용
- README.md 와 같이 확장자가 .md



[실습2] 자주 쓰이는 Markdown 문법

마크다운 파일(.md)를 하나 생성 후 VS Code로 편집

1. 제목

```
# 제목1
## 제목2
### 제목3
```

[실습2] 자주 쓰이는 Markdown 문법

2. 텍스트 효과

```
~~취소선~~
**굵게**
*기울임*
__굵게__
_기울임_
```

3. 항목

```
- 순서 없는 항목 1
- 순서 없는 항목 2

1. 첫 번째 항목
2. 두 번째 항목
```

[실습2] 자주 쓰이는 Markdown 문법

4. 링크와 이미지

```
[Google](https://www.google.com)

![로고](https://cdn-cms.elice.io/elice-strapi/CI_8fd881eb6a.webp)
```

5. 인용문

```
> 이 문장은 인용문입니다.
```

[실습2] 자주 쓰이는 Markdown 문법

6. 취소선

```
---
***
___
```

7. 표

```
| 제목1 | 제목2 |
|-------|-------|
| 내용1 | 내용2 |
| 내용3 | 내용4 |
```

[실습2] 자주 쓰이는 Markdown 문법

8. 하이퍼 링크 적용

```
<https://www.example.com>
```

9. 코드

````
```python
print("Hello, Markdown!")
```
````

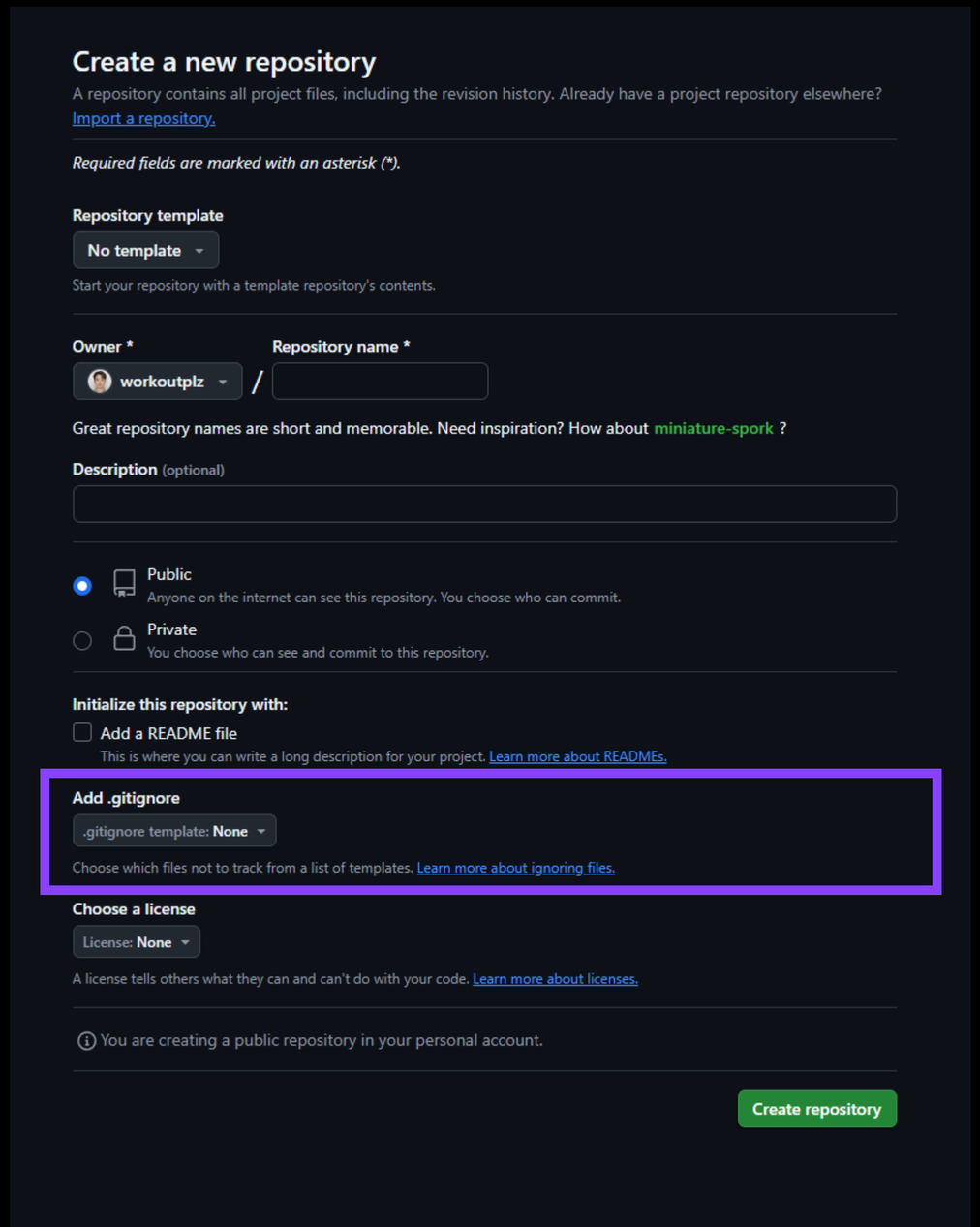
[실습2] 자주 쓰이는 Markdown 문법

10. 이스케이프 문자
    *, -, # 등 서식화하지 않고 문자 그대로를 나타내고 싶을 때 앞에 \ 를 붙임

```
\*별표\*
```

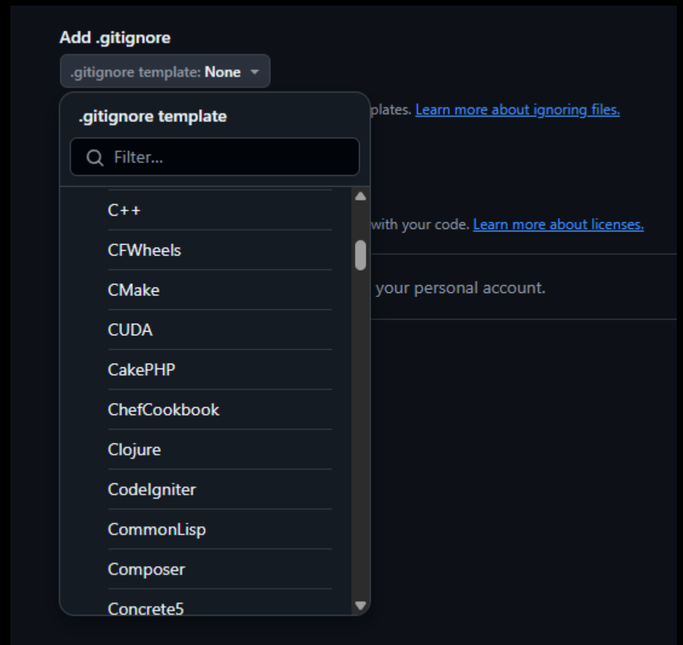
본인을 소개하는 README.md를 작성해봅시다.

첫 Repository 만들기

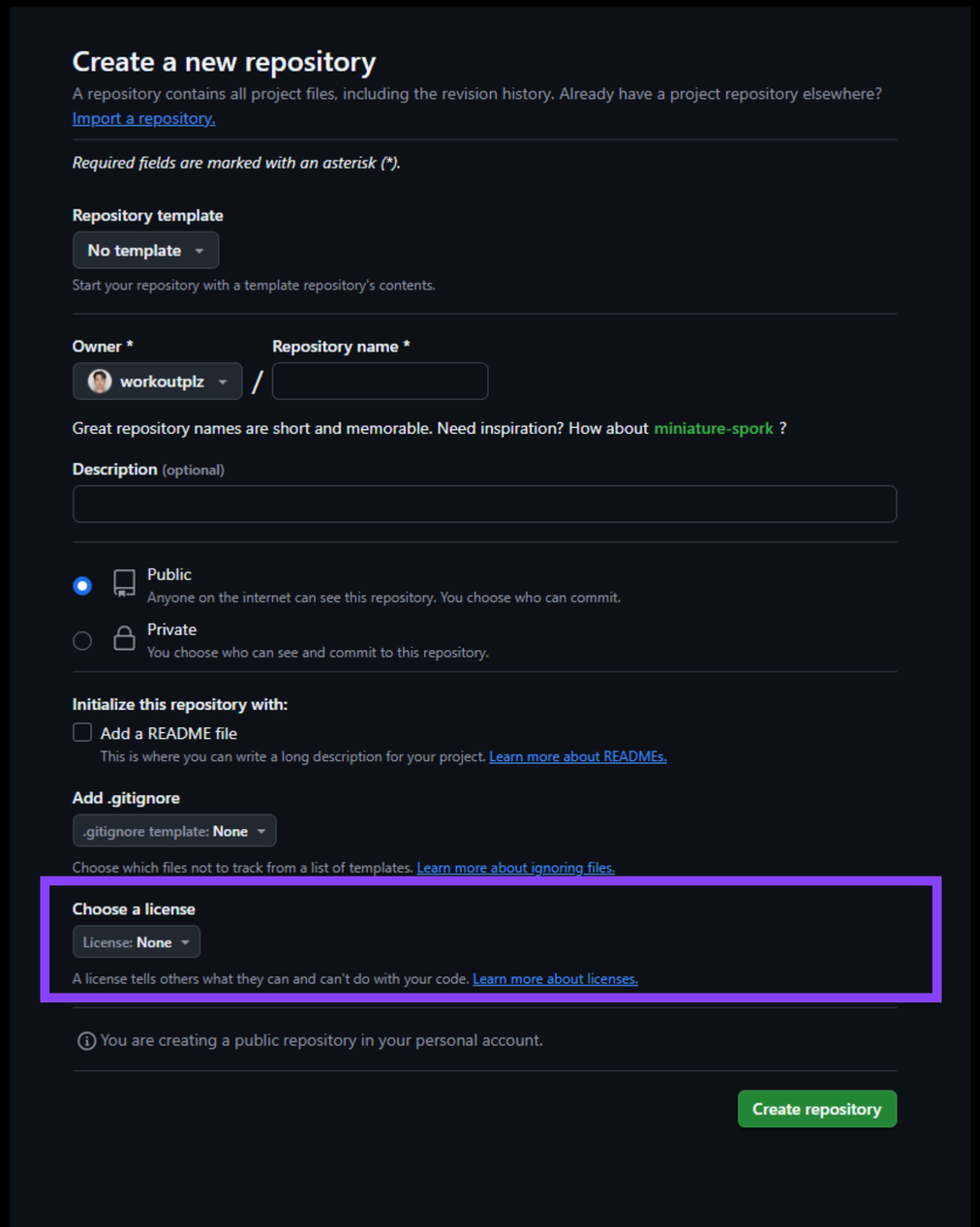


.gitignore?

- Git 이 무시해야하는 파일이나 폴더를 지정
- 지정되면 Repository에 업로드 되지 않음
- 불필요한 파일, 비밀번호 등 보안 파일을 지정하여 효율적이고 보안을 유지한 운용
- 다양한 언어와 프레임워크 별 템플릿 존재



첫 Repository 만들기



license?

- SW 사용, 배포, 수정 등의 권한을 정의
- 사람들이 어떤 조건으로 코드를 사용하고 변경할 수 있는지를 명확히 규정
- 각 license 마다 사용과 배포 조건이 다름
- 개인 공부나 간단한 프로젝트에는 지정하지 않아도 무방
- 타인의 코드 사용 시 반드시 확인해야 함
- 관련 사이트 https://choosealicense.com/

대표적인 LICENSE

MIT 라이선스

- 소스 코드의 수정, 복사, 배포, 상업적 이용 모두 허용
- 수정된 소스코드 공개 의무 없음
- 원본 저작권 및 라이선스 고지를 소스코드에 포함
- 매우 관대한 라이선스

대표적인 LICENSE

Apache License 2.0

- 기여자들의 특허권을 명시적으로 포함
- 특허 관련 소송 발생 시 사용자 보호
- 자유로운 소스코드 수정 및 배포
- 수정 내용에 대해 명시와 라이선스 고지 필수
- 수정한 코드를 공개 의무는 없으나, 라이선스 조건을 준수

대표적인 LICENSE

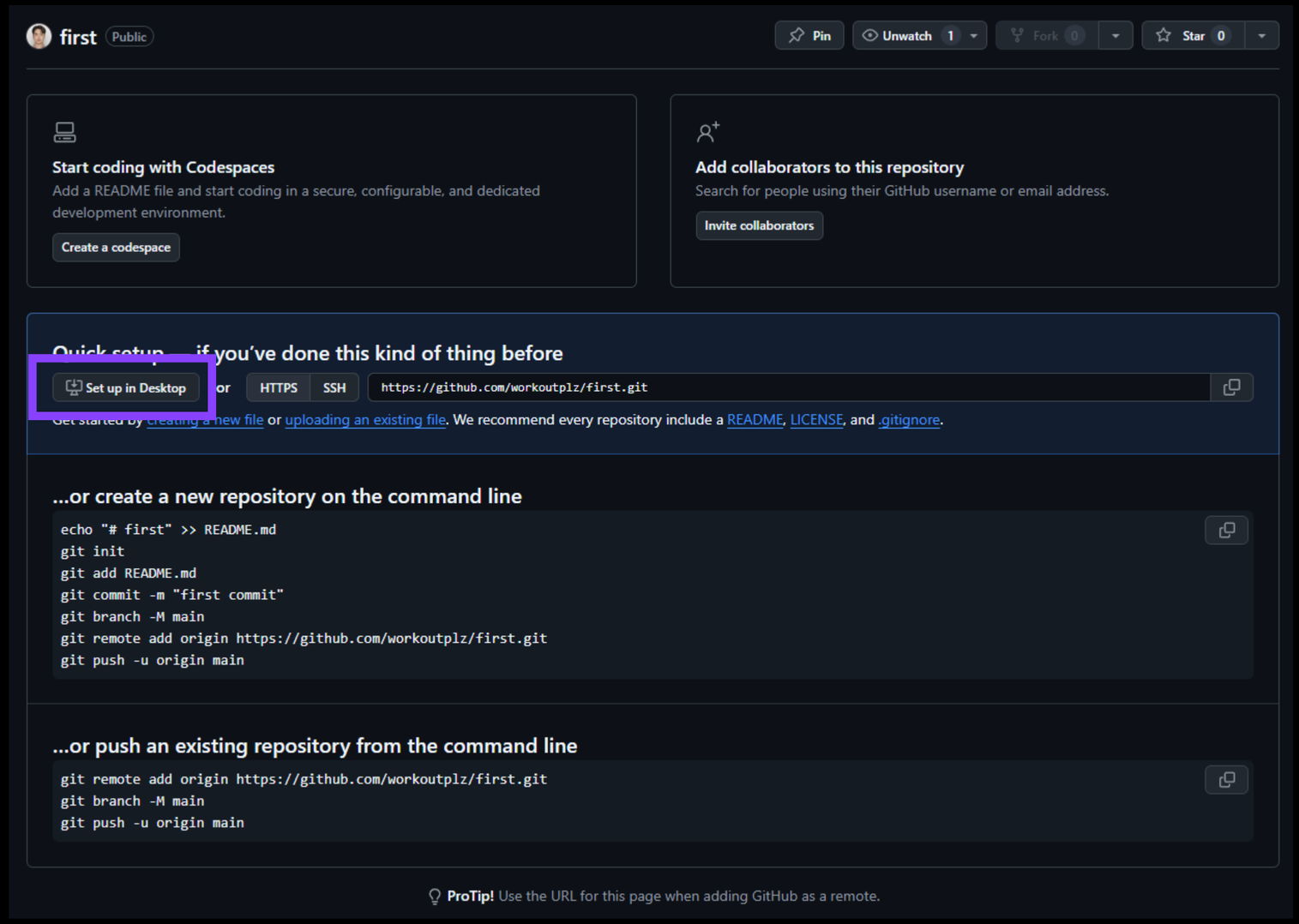
GNU GPL

- SW를 수정하거나 파생 작품을 만들 경우, 동일한 GPL 라이선스로 배포
- SW의 사용, 수정, 재배포에 관한 자유 보장
- 코드 공개 의무화
- 다른 라이선스와 호환성 문제가 발생할 수 있어 사용 전에 라이선스 조건 검토 필요

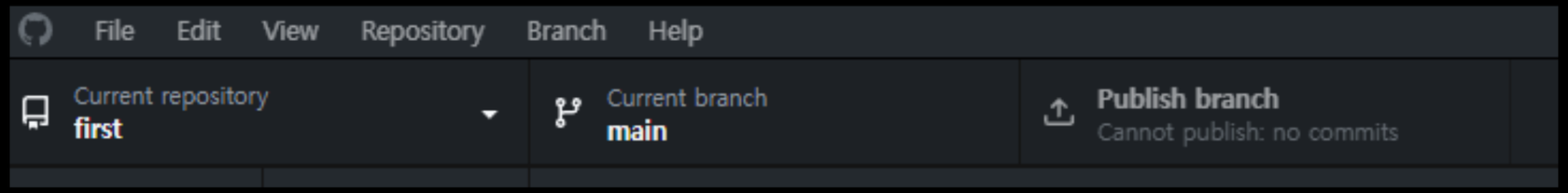
첫 Repository 만들기

생성 완료 후
‘Set up in Desktop’ 클릭

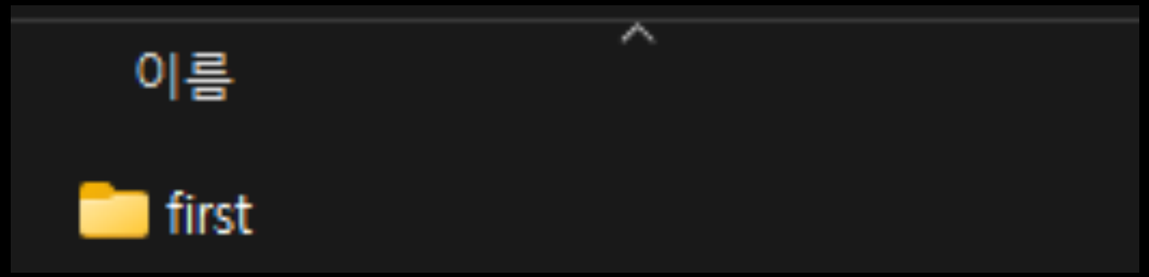
Repository를 연동할
로컬 경로를 지정하여 Clone



첫 Repository 만들기



Github Desktop에 Clone한 repository 이름이 뜨고



내가 지정한 로컬 경로에 Clone한 repository 이름으로 폴더가 생성

[실습3] Repository Clone

아래 조건에 맞는 Repository를 생성 후, Local 환경으로 Clone을 받아보세요.

1. Owner는 본인의 계정으로 설정해주세요.
2. 누구나 Repository 내 내용을 확인, 수정할 수 있도록 설정해주세요.
3. 현재는 README, LICENSE, .gitignore은 따로 설정하지 않습니다.
4. 그 외 Repository name 등은 자유롭게 설정해주세요.
5. 로컬 경로는 본인이 편한 곳으로 설정하되, 폴더 명은 Repository name과 일치해야합니다.

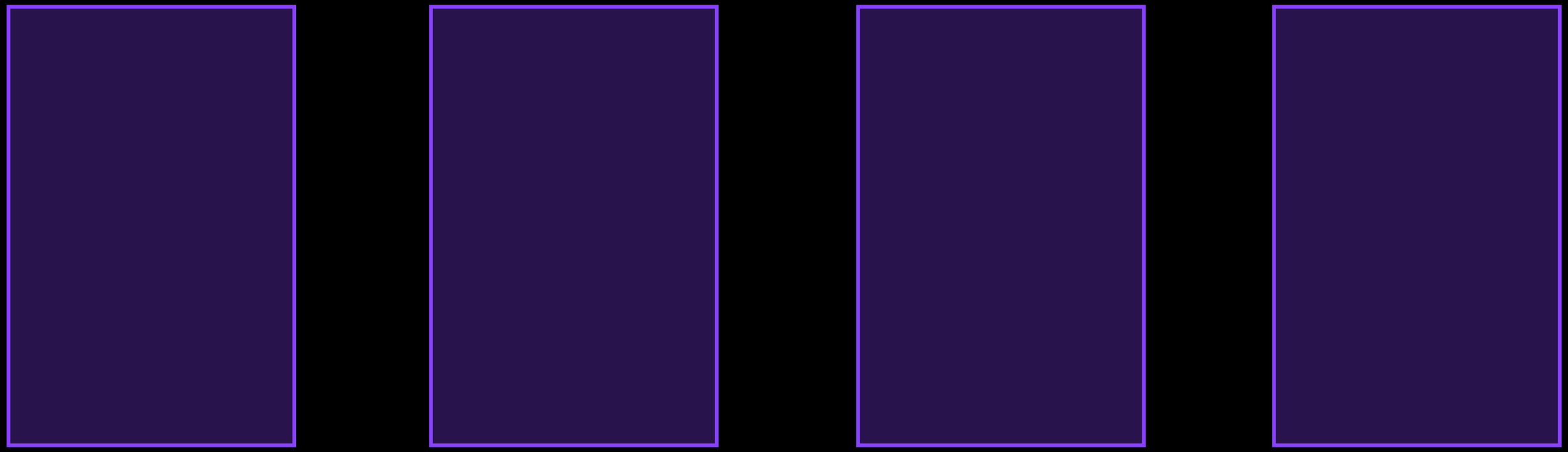
GitHub의 4개 영역

로컬 작업 폴더
(Working Directory)

Staging Area

Local Repository

Repository
(Remote Repository)



hello.txt를 Github repository에 업로드 하기 위해서 위 4개의 영역을 거쳐야 함

Working Directory

로컬 작업 폴더
(Working Directory)

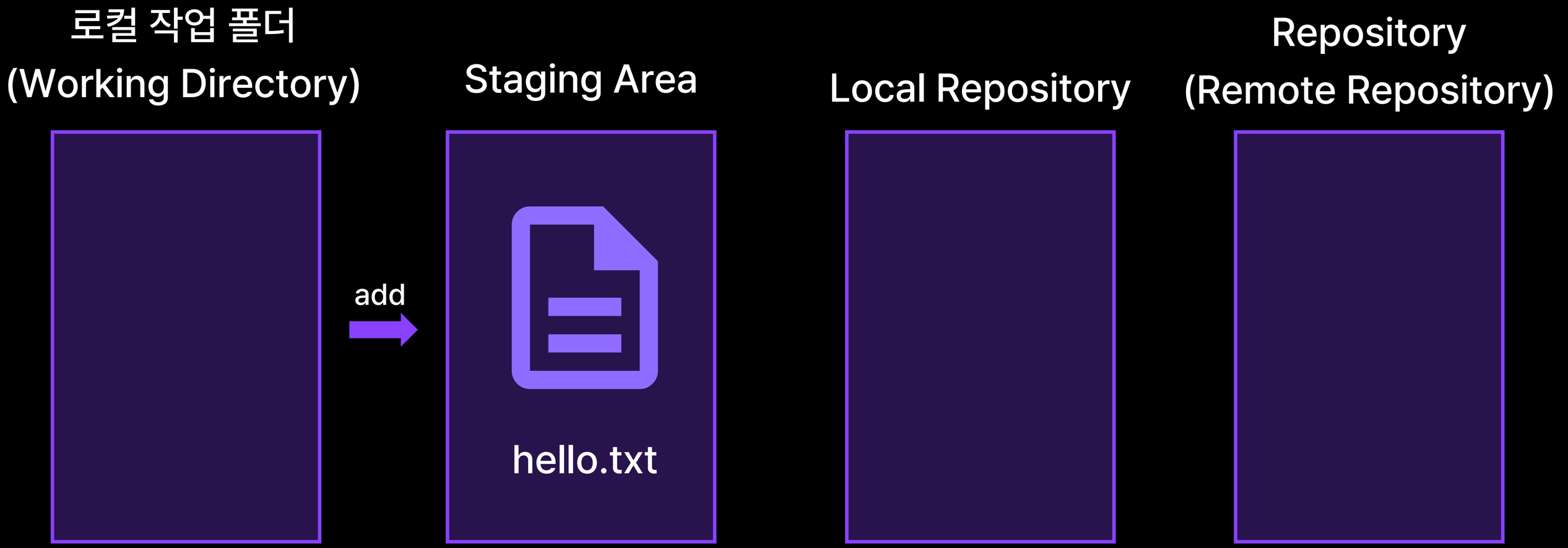
Staging Area

Local Repository

Repository
(Remote Repository)

최초 파일 생성 시에는 Working Directory에서만 변경 사항을 인지

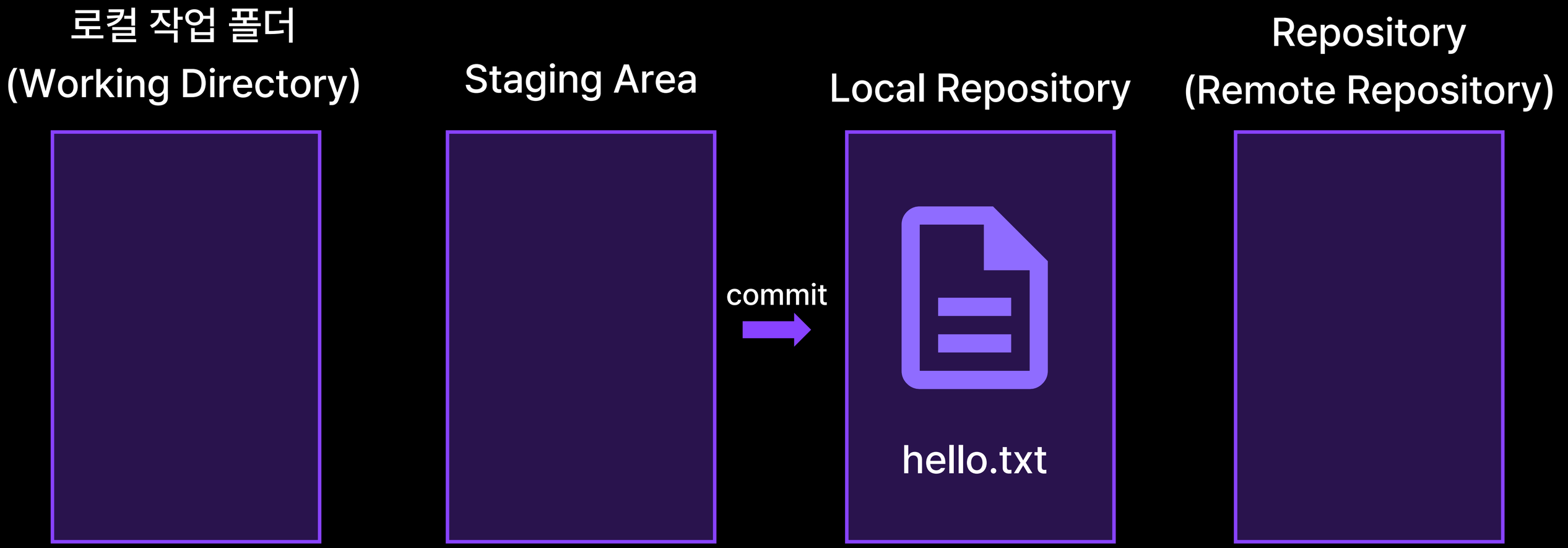
Staging Area



hello.txt 를 Git이 추적할 수 있도록 Staging Area로 추가

Staging Area 는 변경 사항을 추적하고 잠깐 모아두는 공간

Local Repository

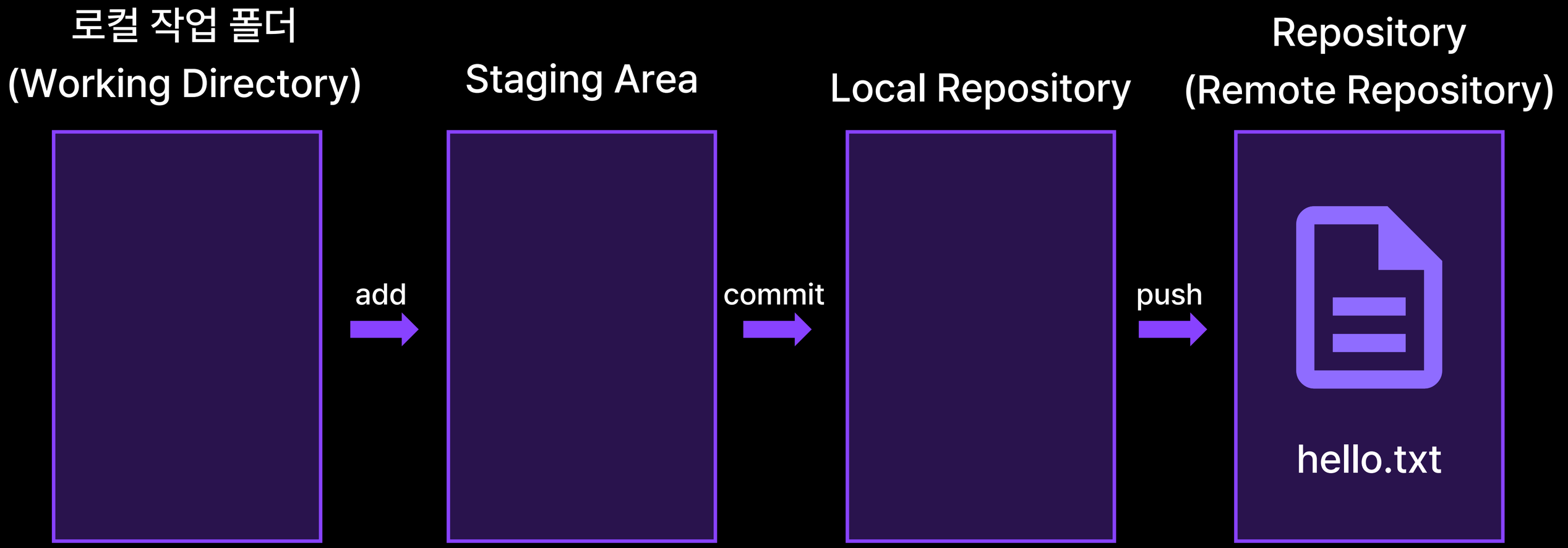


hello.txt의 변경 사항을 Github 상에 올리기 위해 로컬에서 변경 이력을 잠깐 관리
실제 Remote Repository에는 변경 사항이 반영되지 않음

Remote Repository

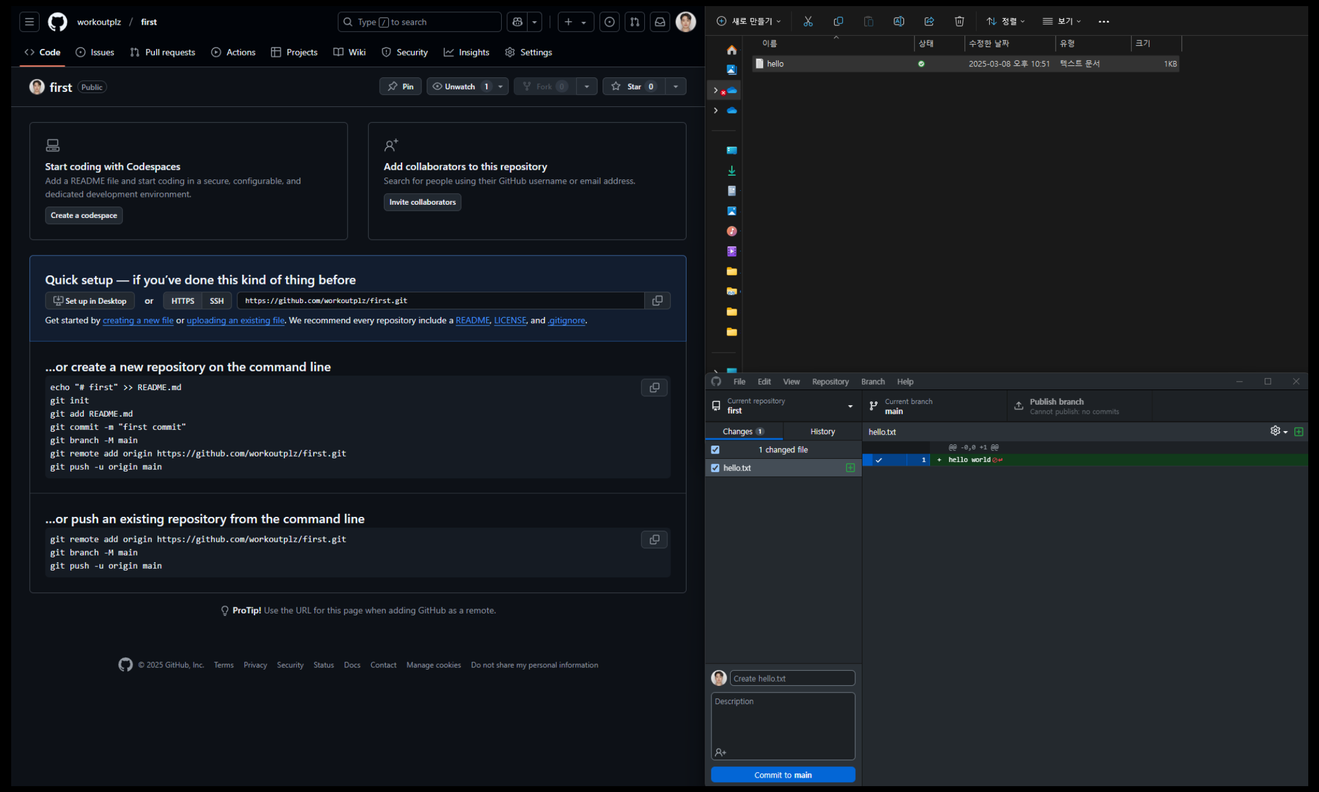
Local Repository 영역에 모아진 변경 사항(commit)을 Remote Repository에 업로드

GitHub의 4개 영역



Git add, commit, push 의 절차를 거쳐야
비로소 로컬의 변경 사항이 Remote Repository에 적용

Commit 과 Push



- Clone 받았던 로컬 경로에 hello.txt를 저장해도 Github 내에는 미반영
- 하지만 Github Desktop은 인식

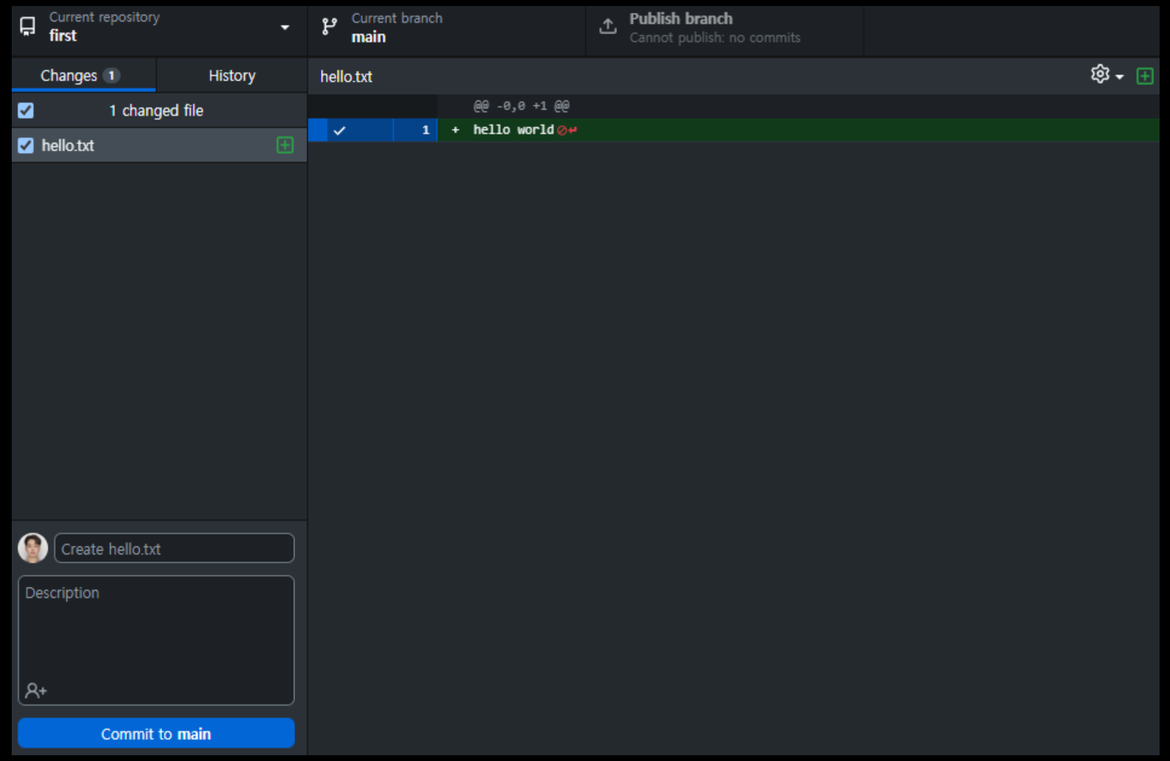
⇒ Staging Area에 있는 상태

현 상황에서 Github에 반영하려면?

= Remote Repository에 올리려면?

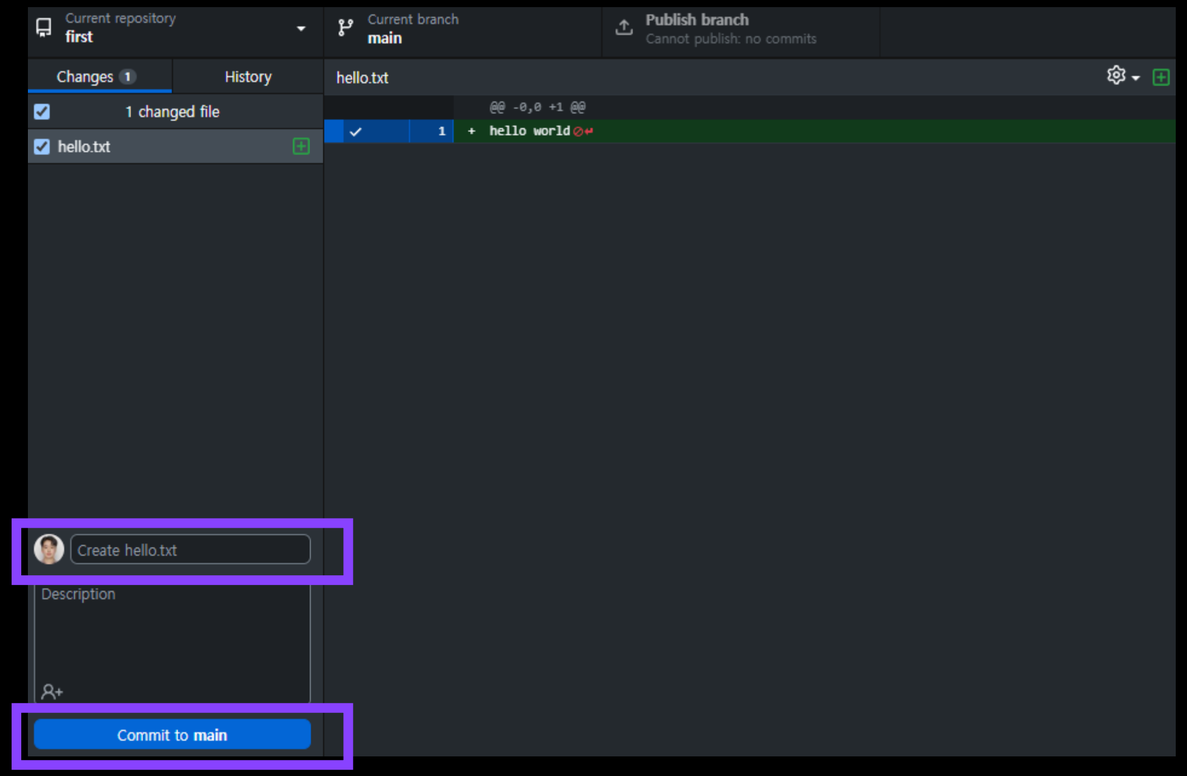
Commit

Staging에 있는 변경 사항을
Local Repository 영역으로 이동
⇒ Commit 필요

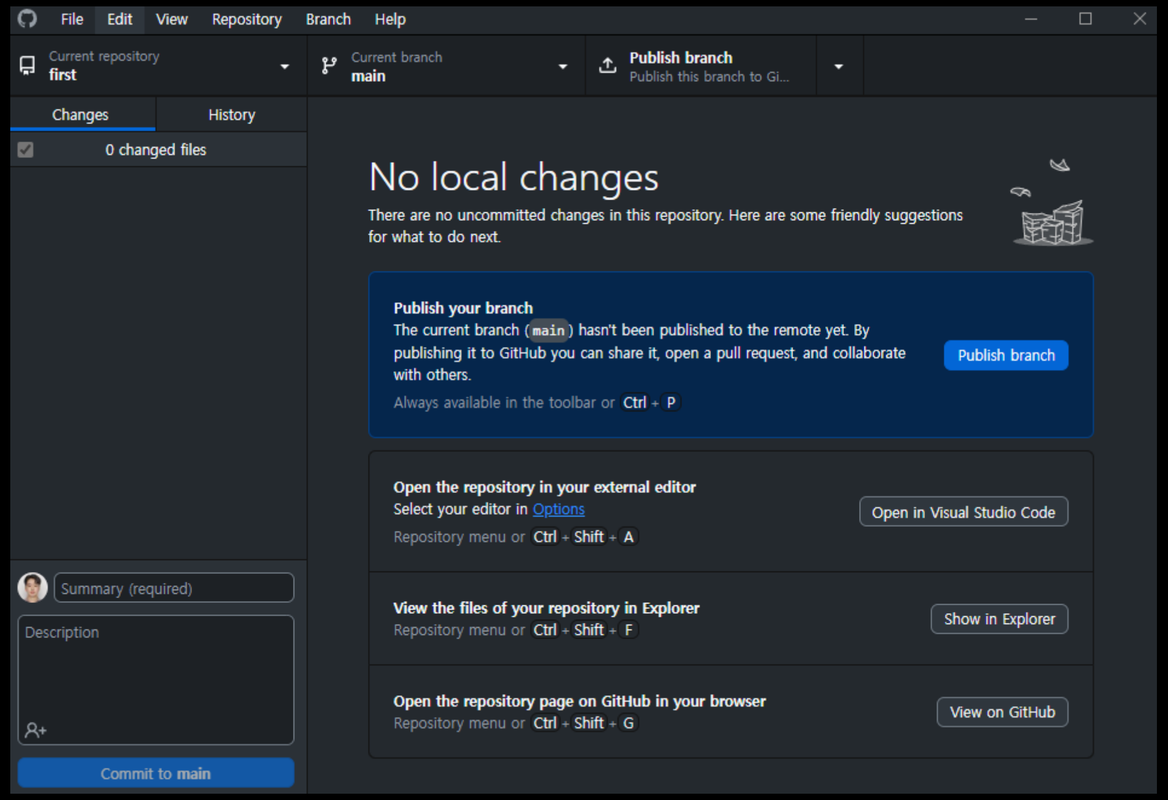


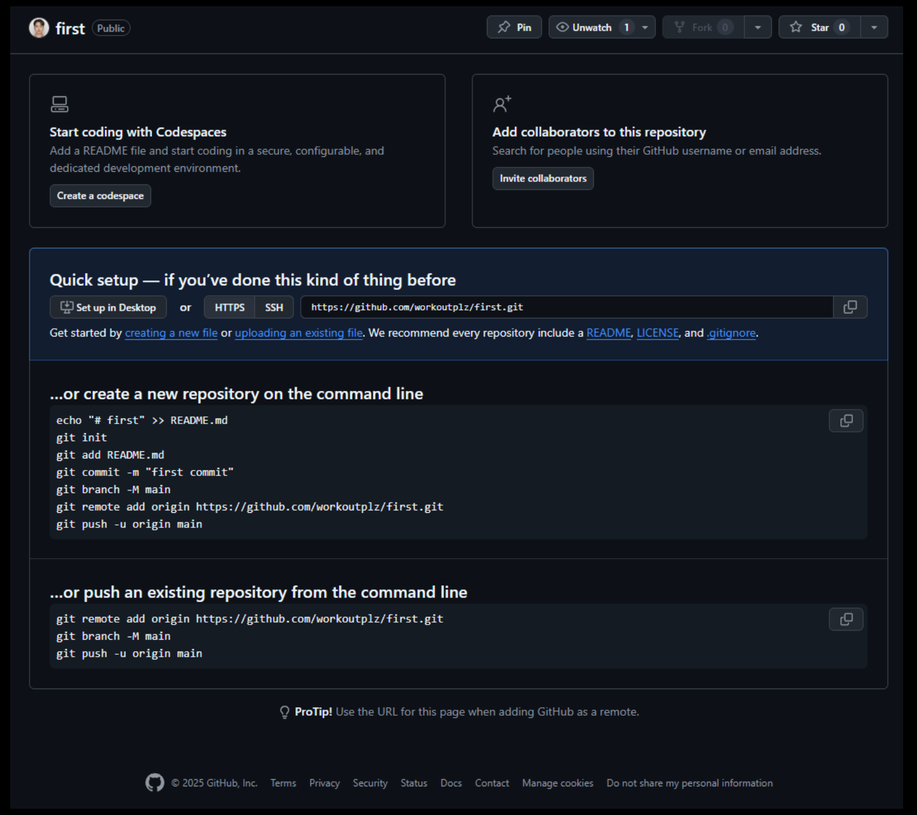
Commit

Commit 시에는 변경 내용 요약 메시지,
‘commit message’ 작성 후 commit



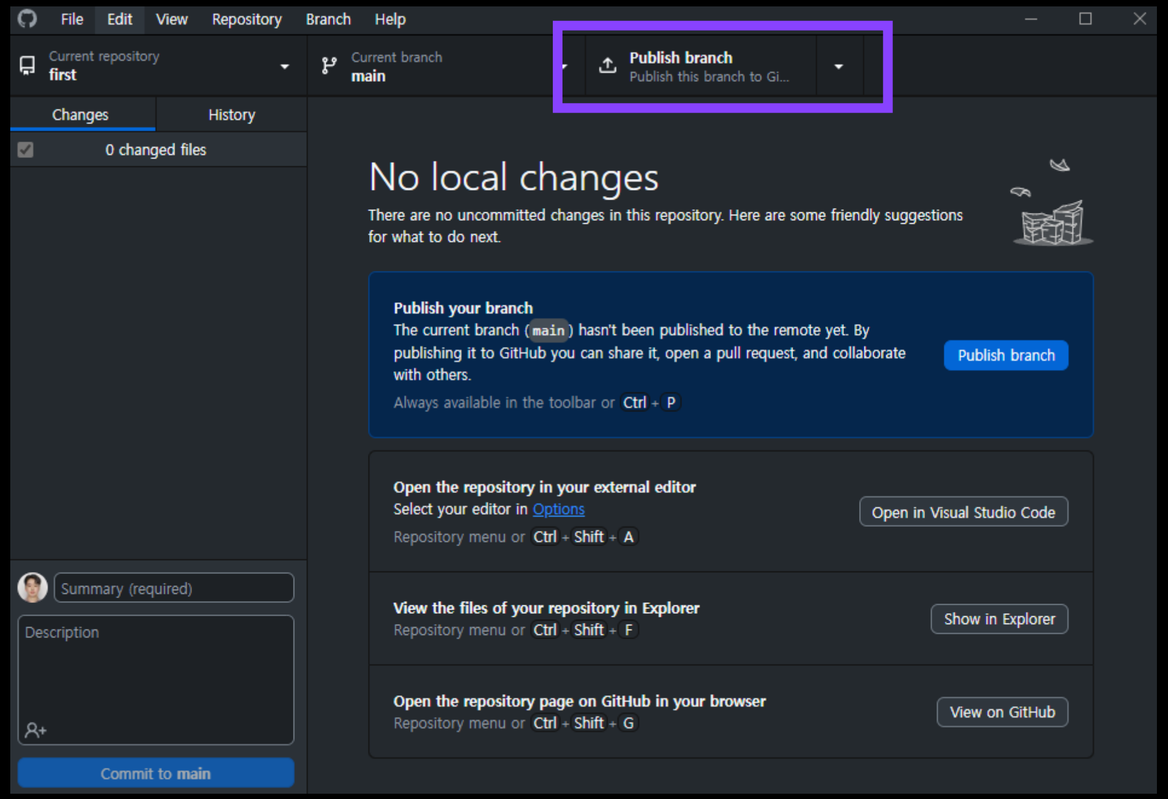
Push





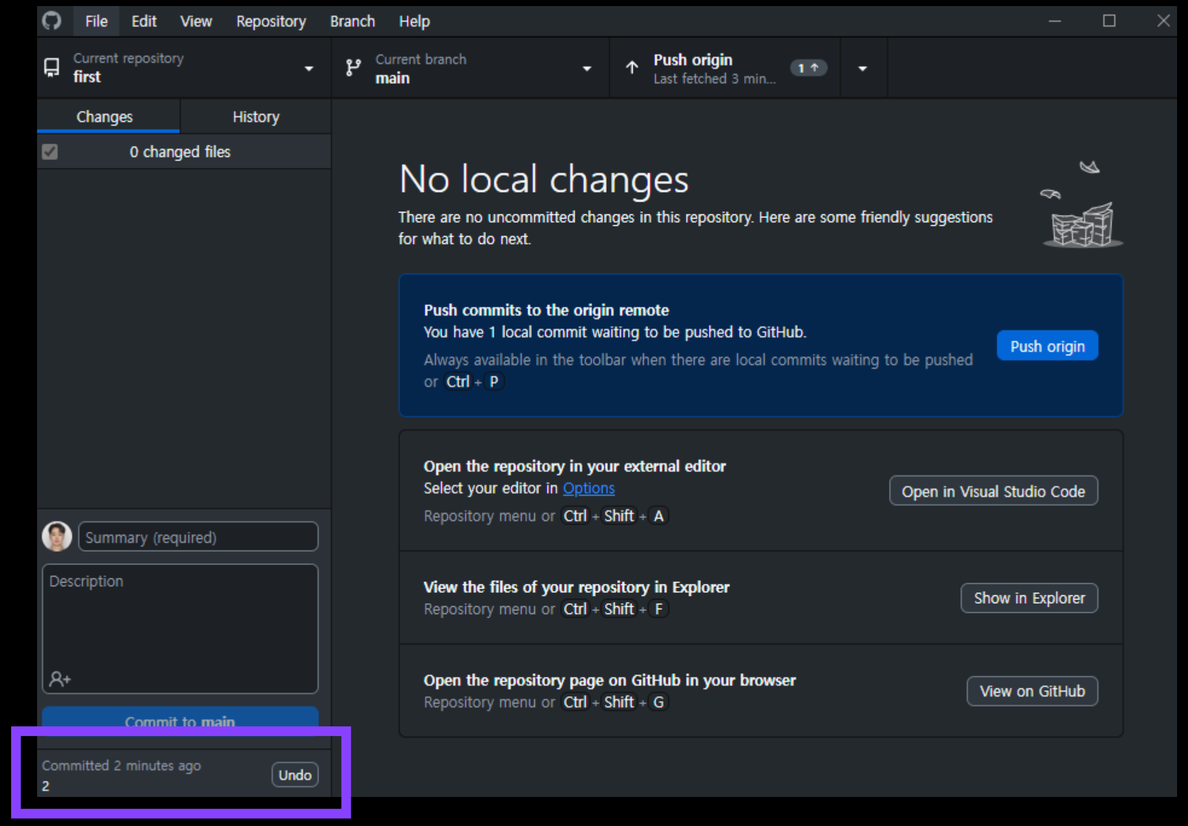
Local Repository에 옮겨 간 상황이므로 Staging Area에는 보이지 않으나,
Remote Repository에도 존재하지 않음
⇒ Remote Repository에 Push 필요

Push



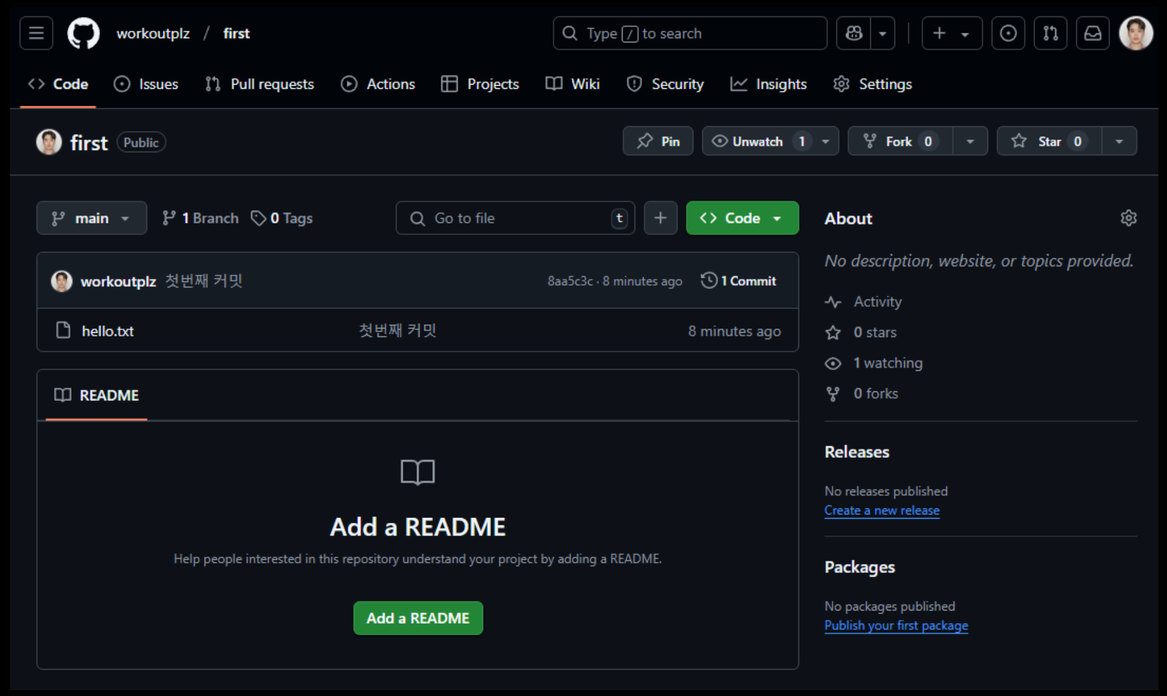
- 최초 push 시,
  Publish branch 버튼 클릭
- 이후 push 시에는
  ‘Push origin’ 버튼으로 변경 됨
- Push 완료 시에는
  Remote Repository 도 변경

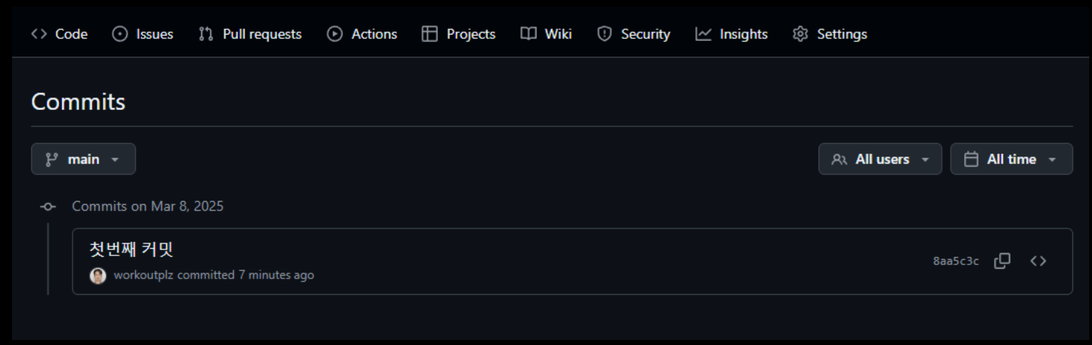
Undo

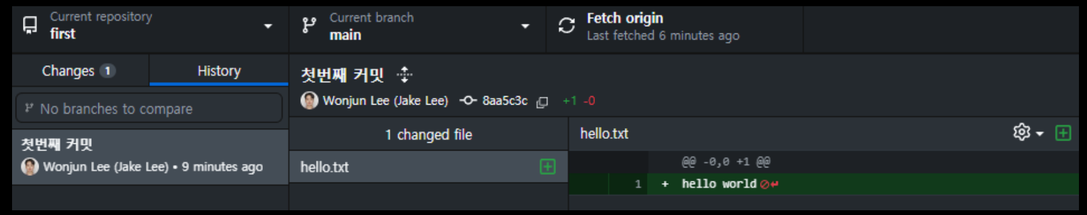


만약 잘못 commit을 했을 경우,
push 이전에 Undo 가능

Commit 과 Push



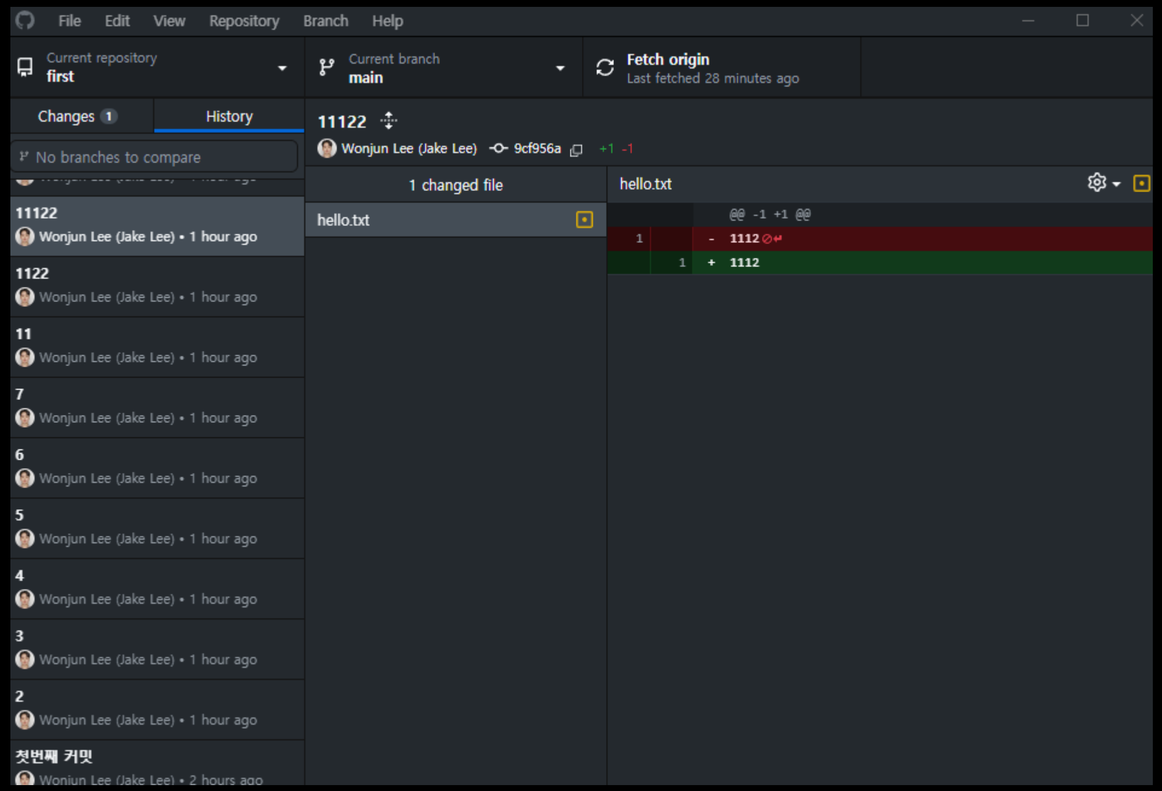




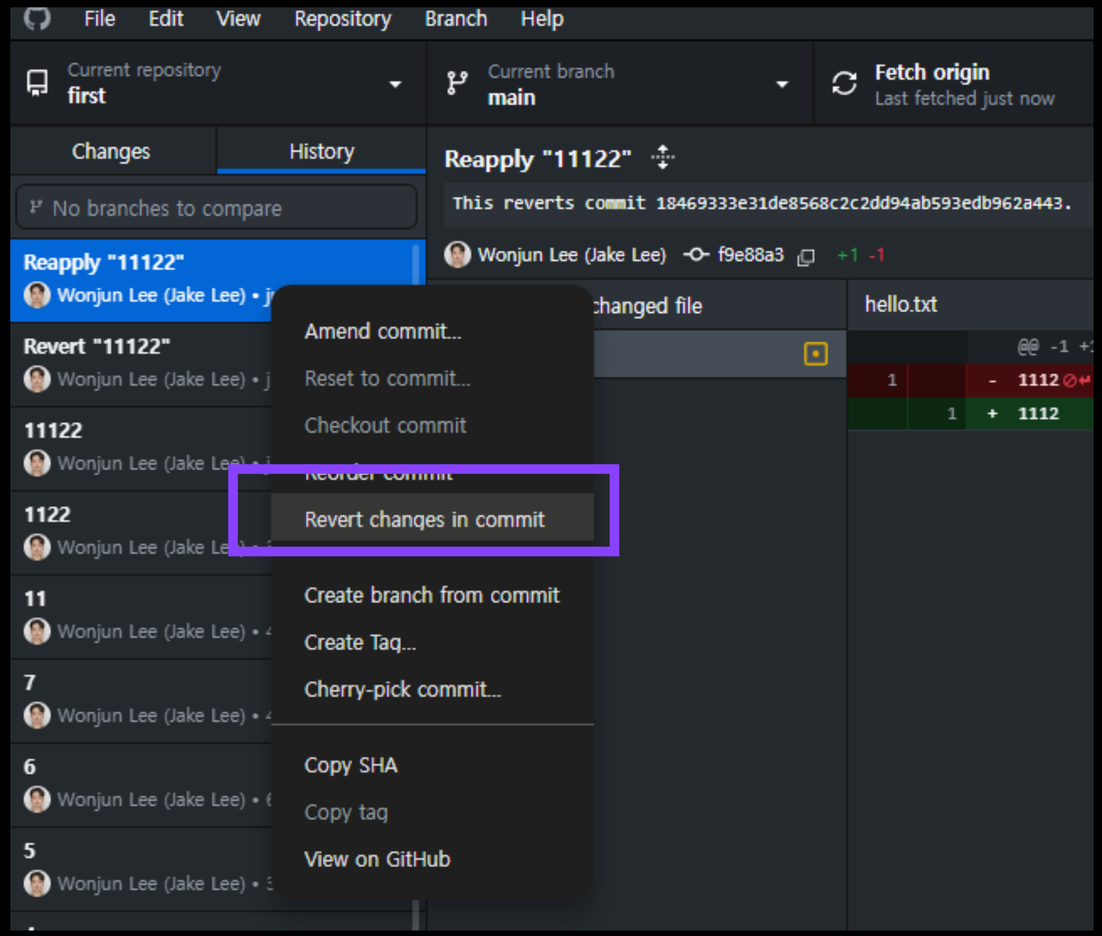
Remote Repository와 Local Repository 모두
변경 내용이 정상적으로 기록, 적용 된 모습을 볼 수 있음

[실습4] Commit & Push 5번

- Clone 받은 Repository를 대상으로 README.md를 push 해보세요.
- 추가로 최소 5번 이상의 commit과 push를 해보세요.
- 이전에 작성했던 코드들을 올려보세요.

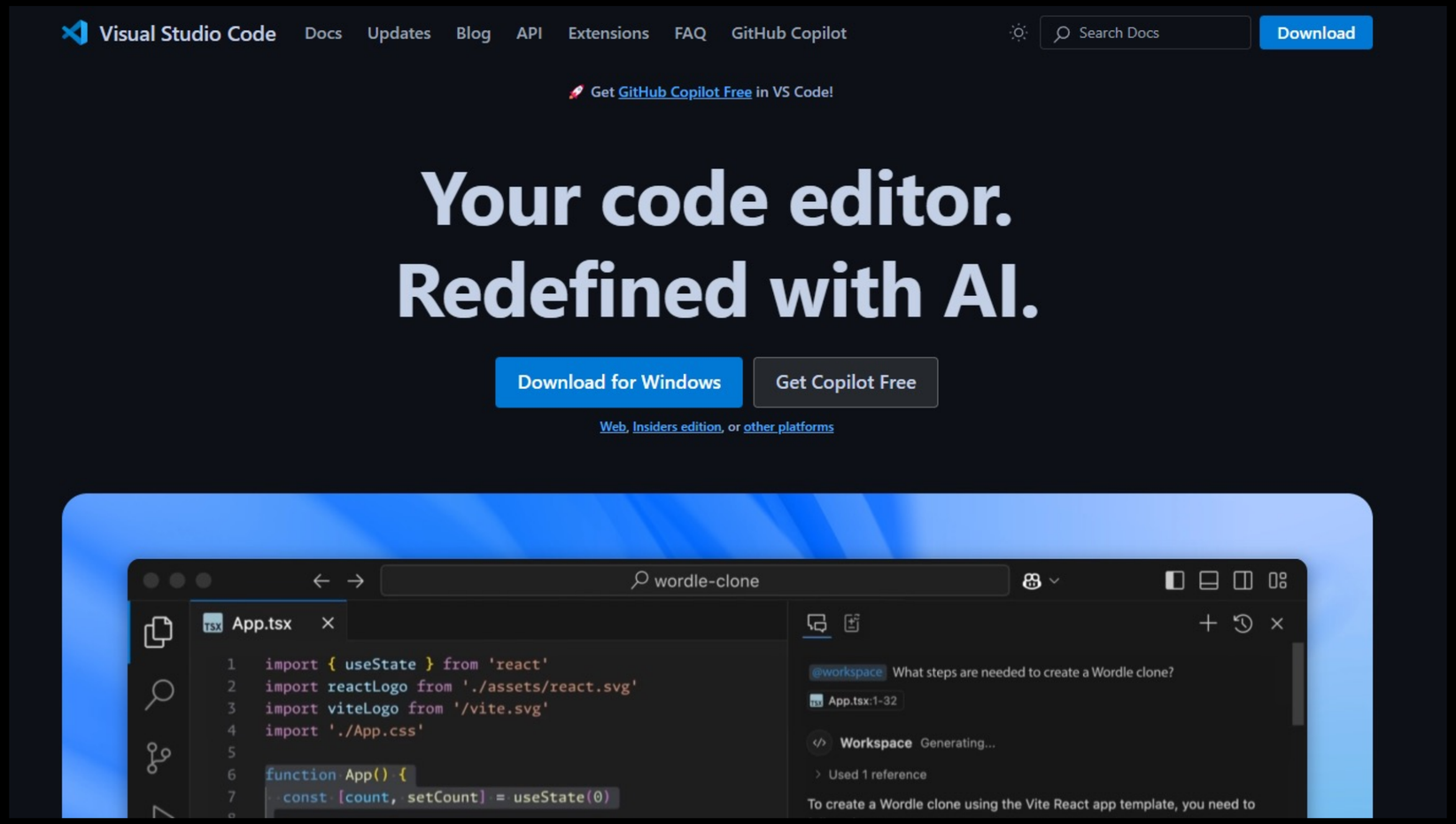


Revert



- Github Desktop 내 History 탭에서 이전 커밋 기록 확인 가능
- ‘Revert changes in commit’을 통해 이전 커밋으로 복구 가능
  (단, 직전의 커밋으로만 가능)

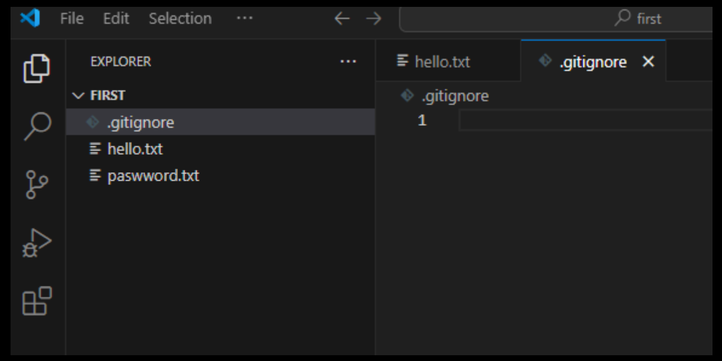
VS Code 다운로드

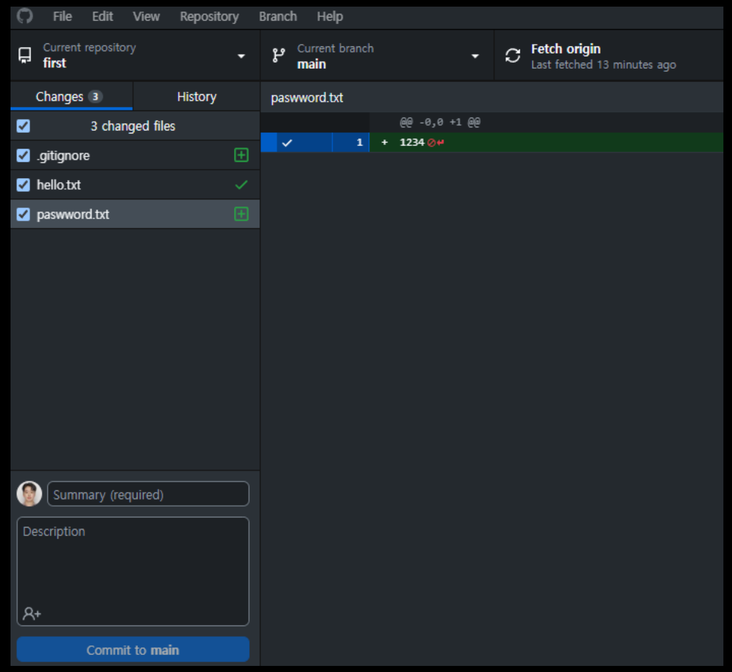


https://code.visualstudio.com/ 에서 다운 후 설치

개인적으로 사용하는 IDE가 있다면 해당 IDE 사용 가능

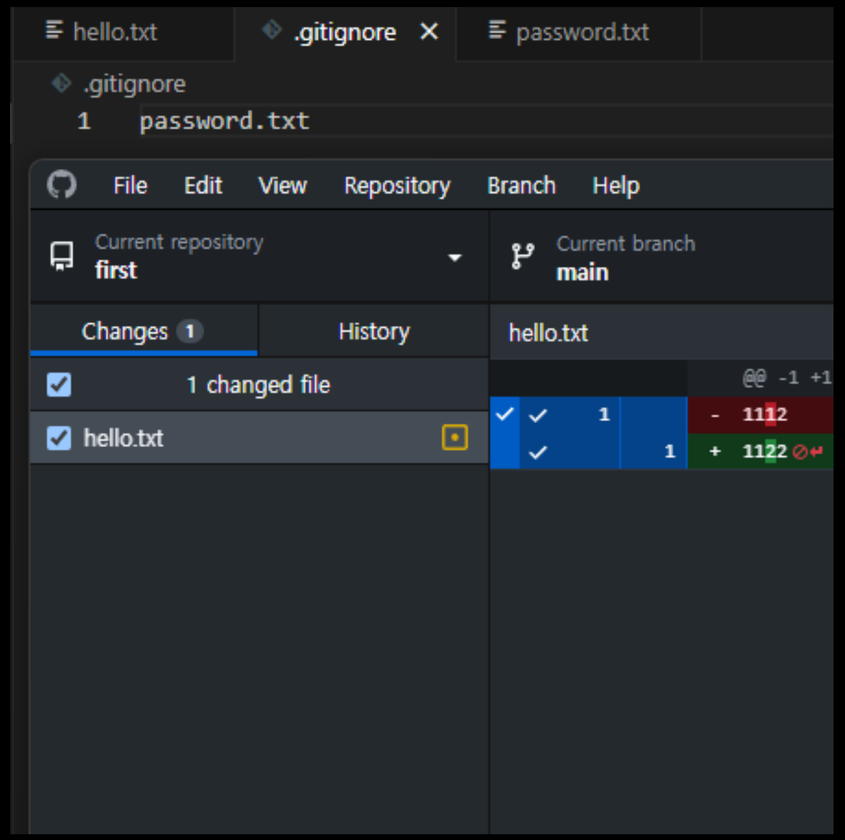
.gitignore





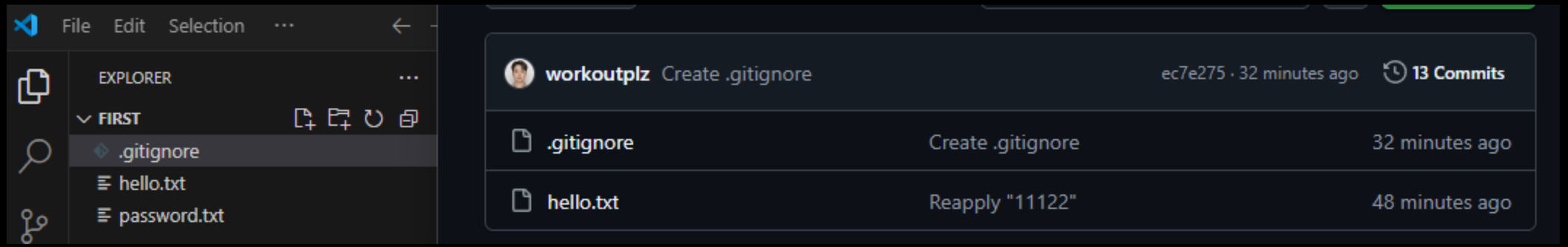
- 왼쪽 상단 사진과 같이 폴더가 구성되어 있다고 할 때, 보안 상 password.txt는 Git에서 추적하면 안됨
- 이 때 사용 하는 것이 .gitignore 파일
- 보안 관련 파일 뿐만 아니라 불필요한 빌드 파일, 라이브러리 관련 폴더를 주로 지정

.gitignore



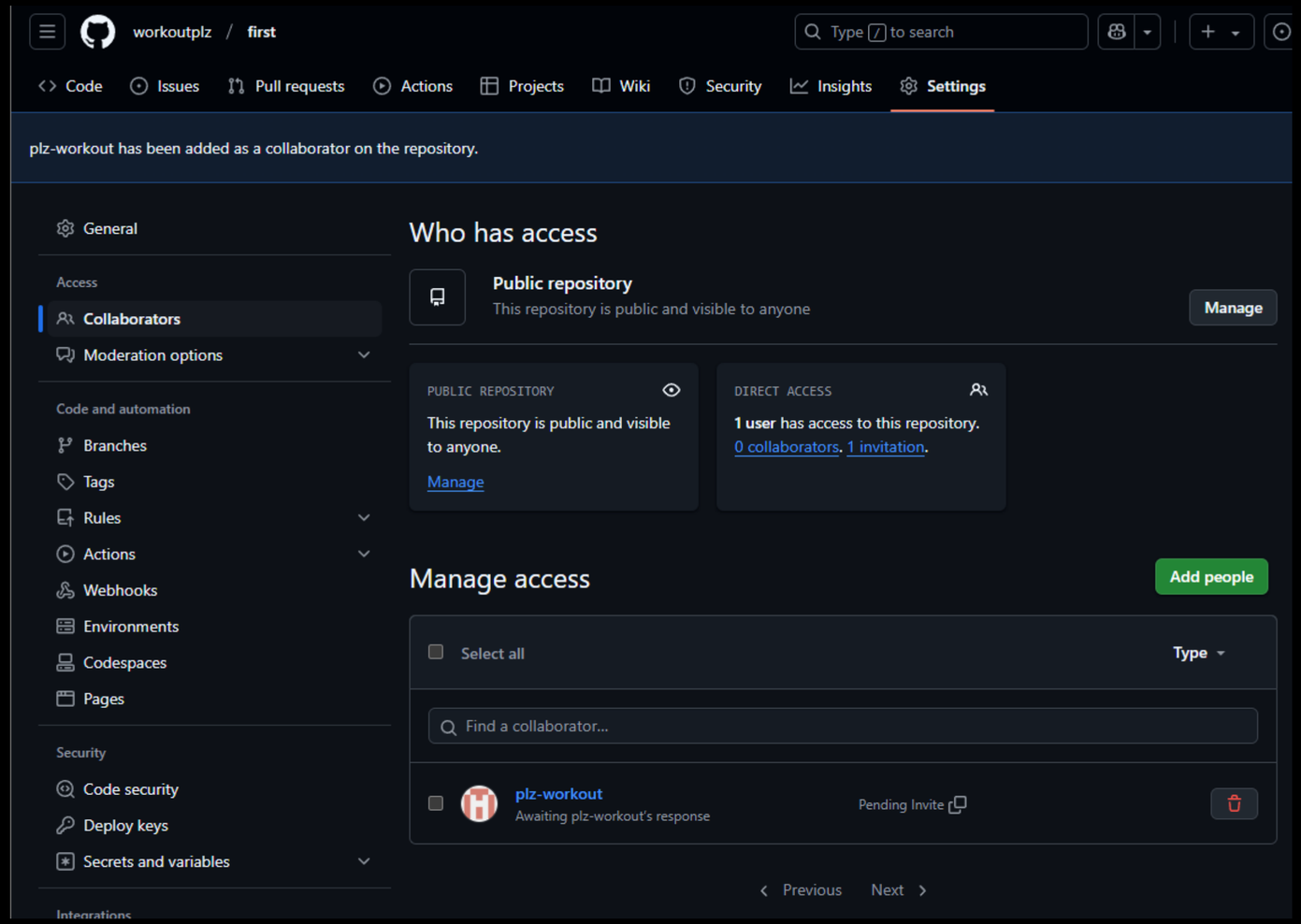
- 다음과 같이 .gitignore 파일에
  업로드가 되면 안되는 파일명 혹은 폴더명을 입력하면
  더 이상 git이 추적하지 않음
- 계정에 대한 정보, 내부 IP 정보 등
  보안 상 민감한 정보가 담겨 있는 파일은
  반드시 .gitignore 파일에 명시

[실습5] .gitignore 활용



로컬 환경에 password.txt 파일을 하나 생성하고,
리모트 환경에는 계속해서 password.txt 파일이 업로드 되지 않도록 만들어 보세요.

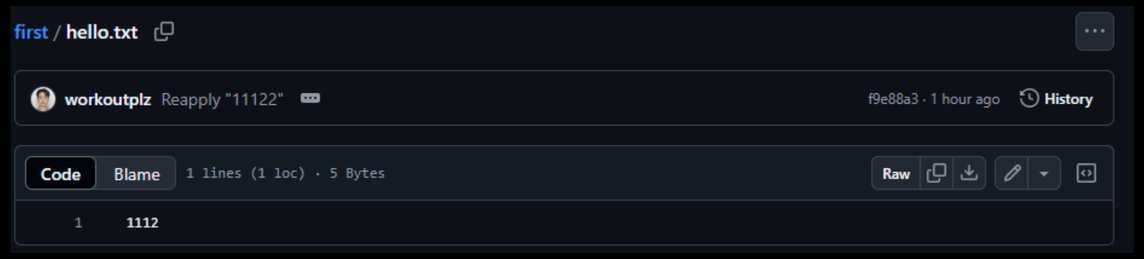
다른 사람과 협업하기(1) - Collaborator

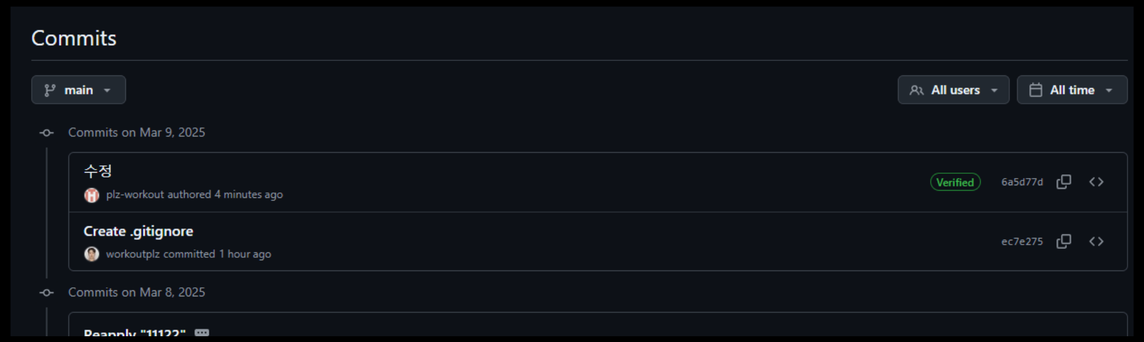


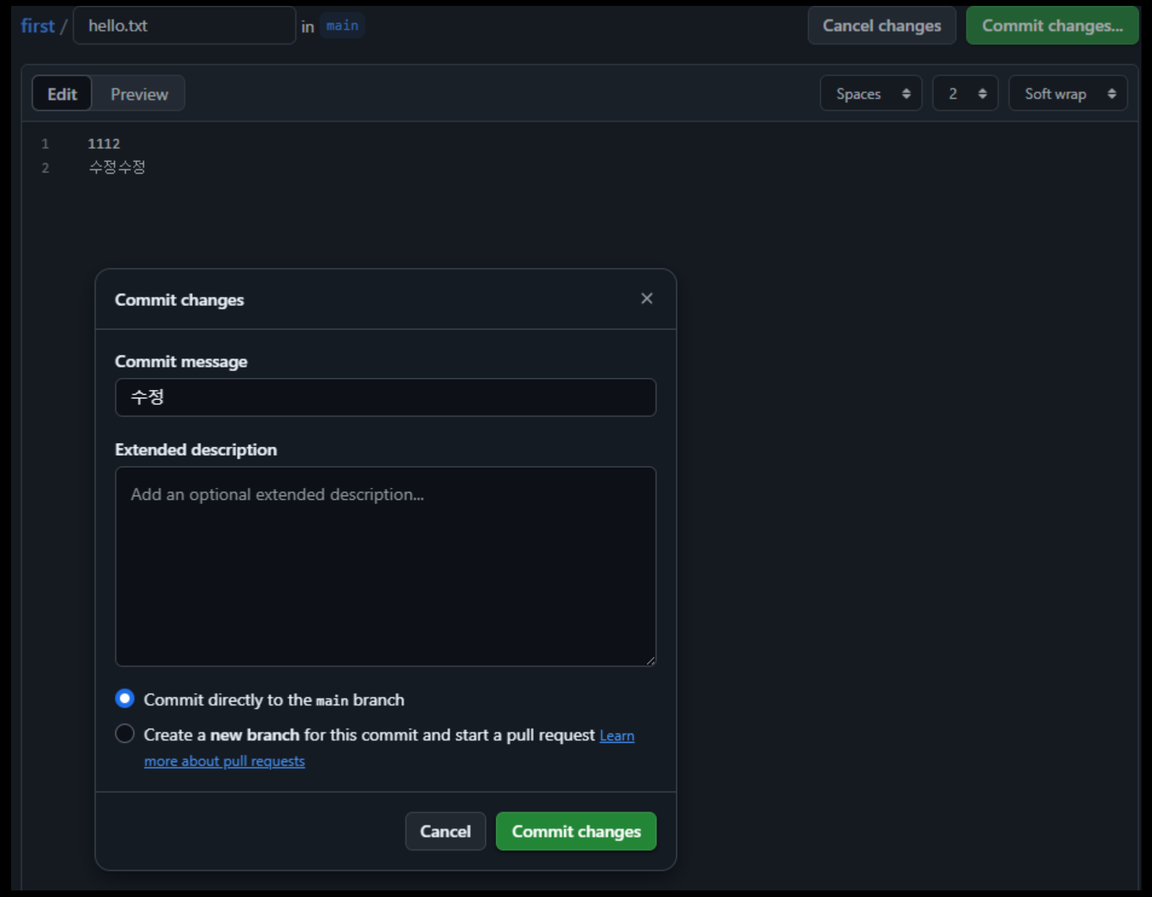
Collaborator(팀원) 등록방법

- Repository -> Settings
  -> Colaboratorsㅇ서 타 수강생 초대
- 초대 받은 사람이 수락을 해야 함

다른 사람과 협업하기(1) – Collaborator



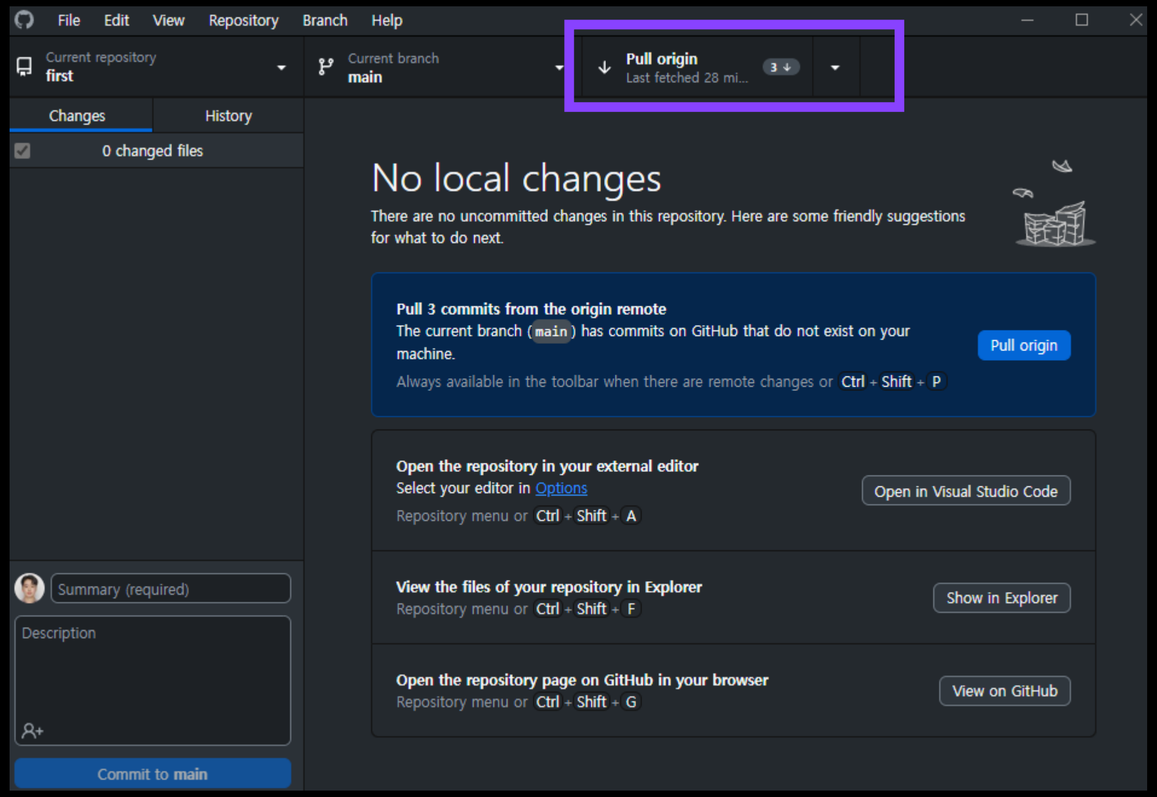




Collaborator 등록 시

- Repository를 직접 수정 가능
  (연필 모양 버튼)
- 웹 상에서 바로 commit, push 가능

다른 사람과 협업하기(1) - Collaborator

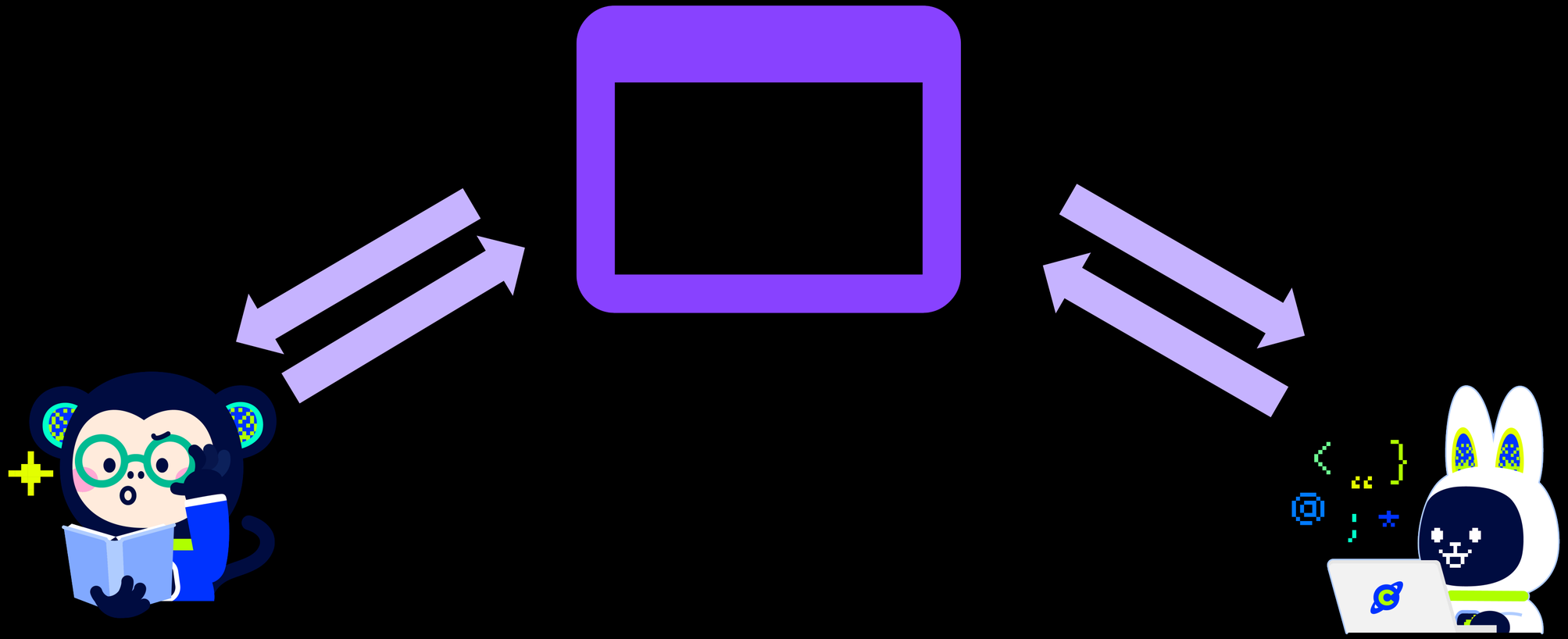


최신 버전 유지를 위해서,
Remote Repository의 변경 사항 로컬 적용 필수

➔ Pull 필수

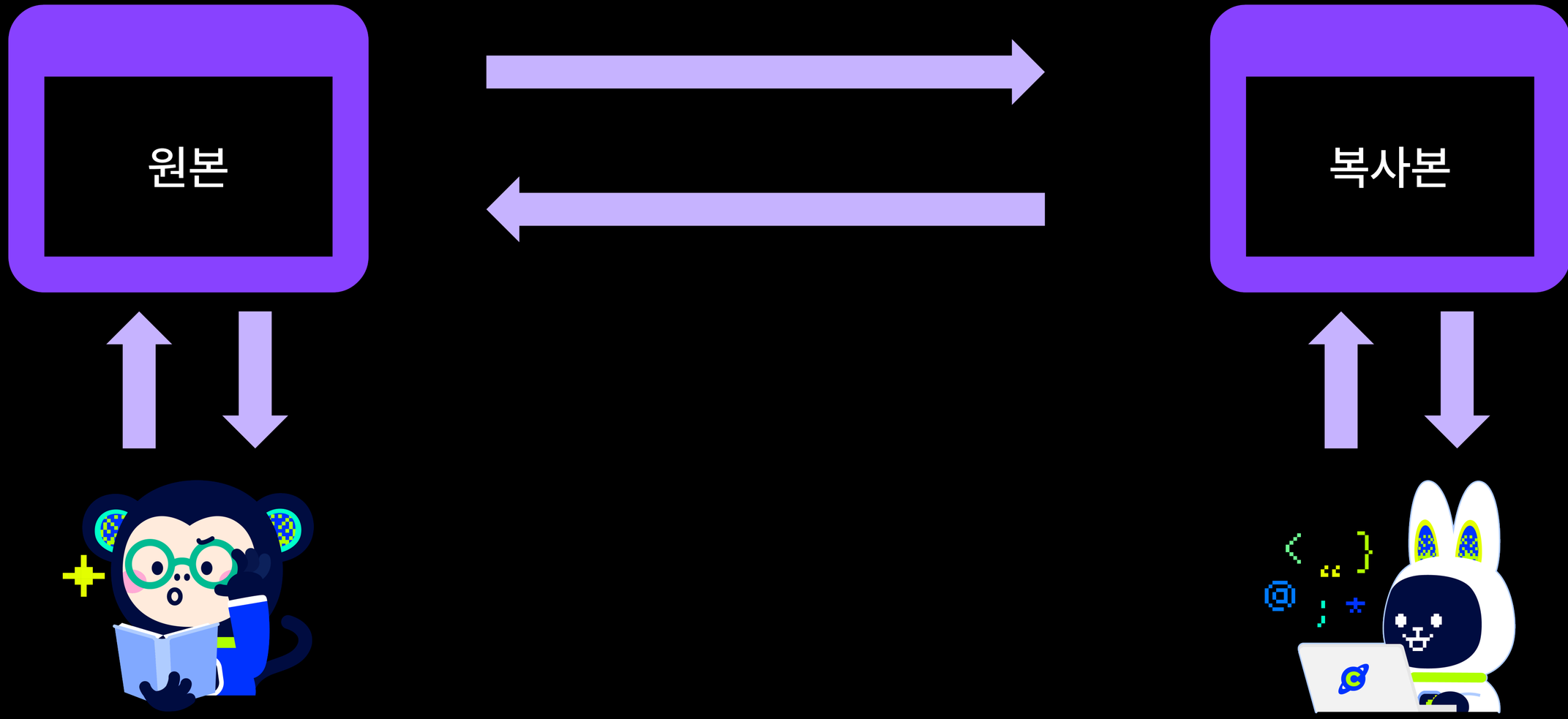
최신 버전이 유지되지 않을 때,
충돌이 발생하여 원활한 협업 불가능

Collaborator의 협업 개념



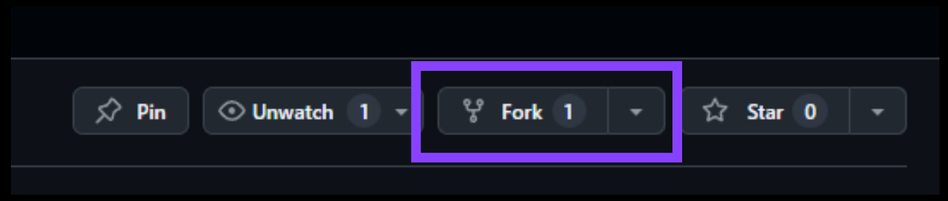
같은 Remote Repository를 직접 수정

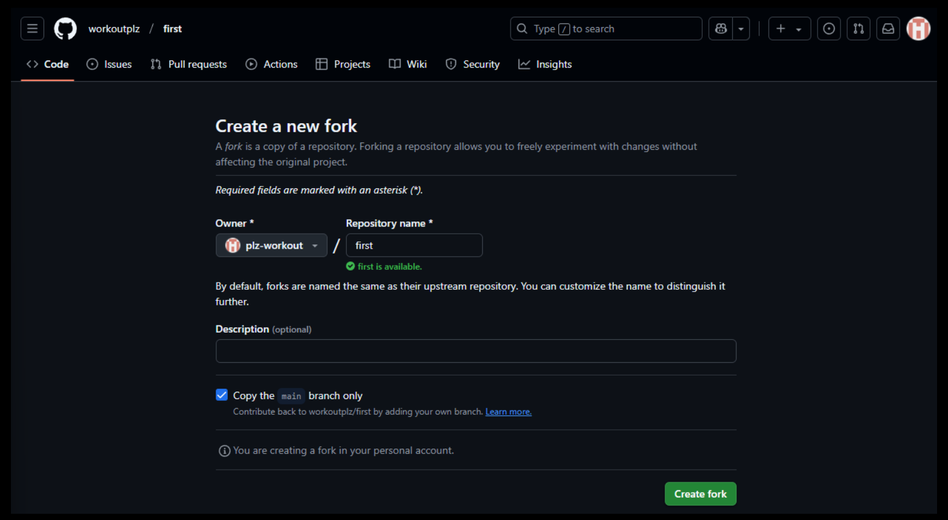
다른 사람과 협업하기(2) – Pull Request

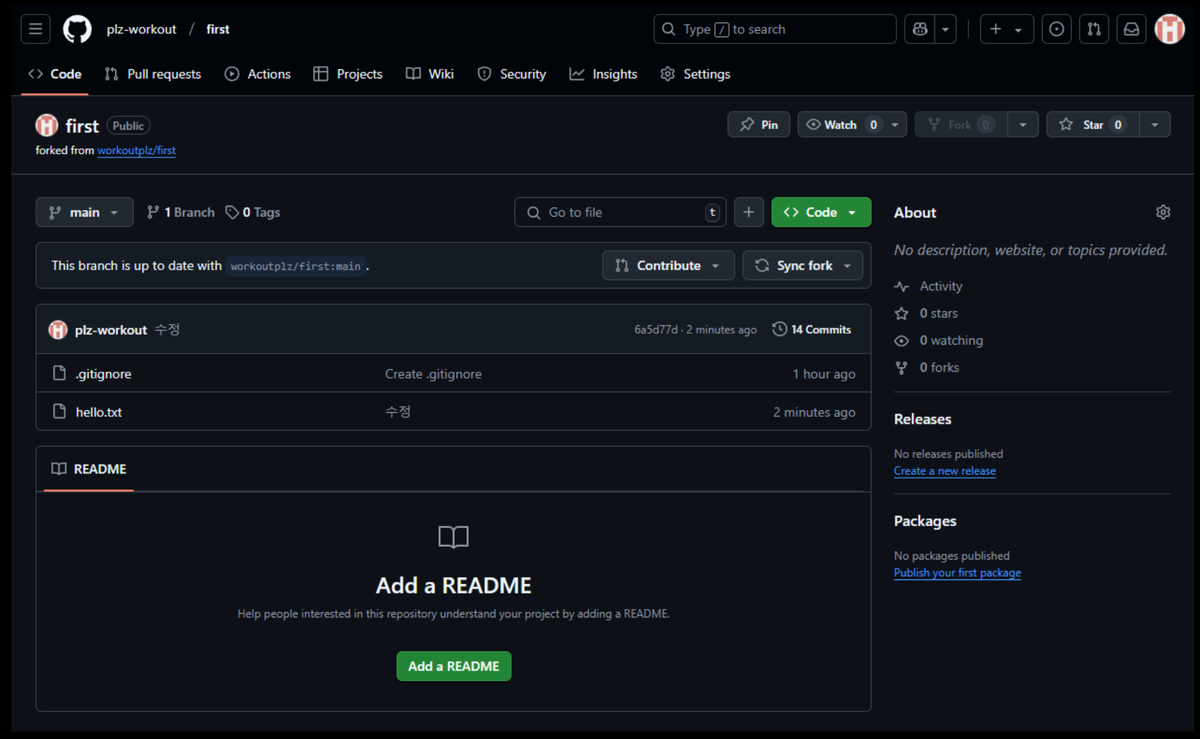


내용은 같으나 따로 존재하는 Remote Repository를 각자 직접 수정, 통합

다른 사람과 협업하기(2) – Pull Request







Fork = 내용이 동일한 Repository를 내 계정으로 복사

다른 사람과 협업하기(2) – Pull Request

원본

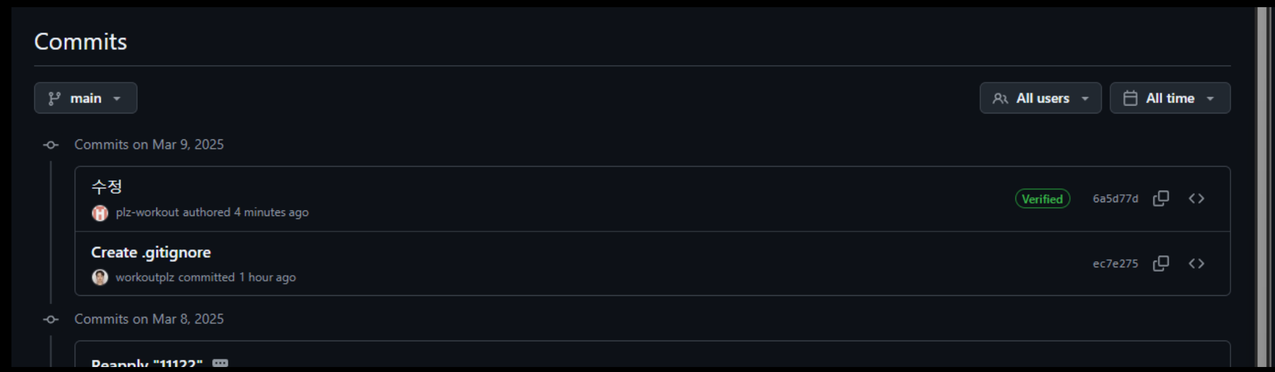
```
print('hi')
```

복사본

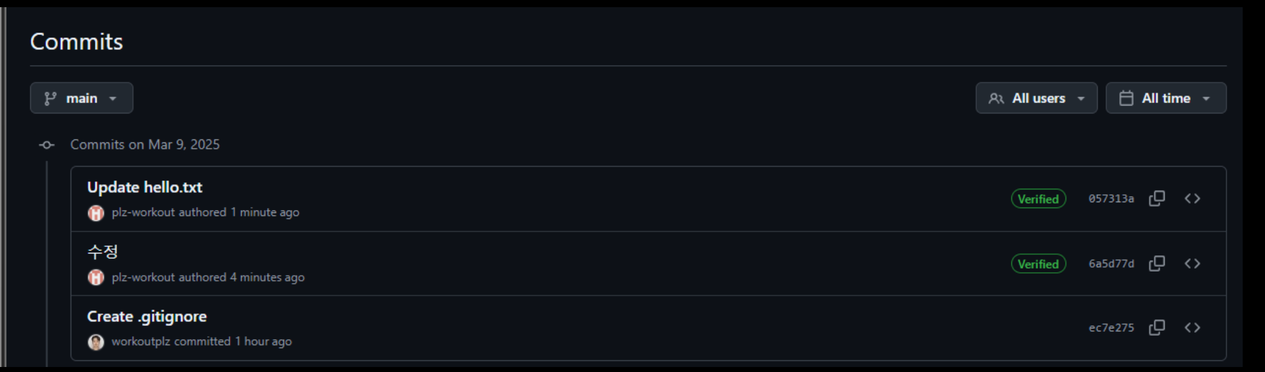
```
print('hi')
print('bye')
```

각각의 개발자는 본인의 Repository를 수정하므로 서로의 수정 내용이 반영되지 않음

다른 사람과 협업하기(2) – Pull Request



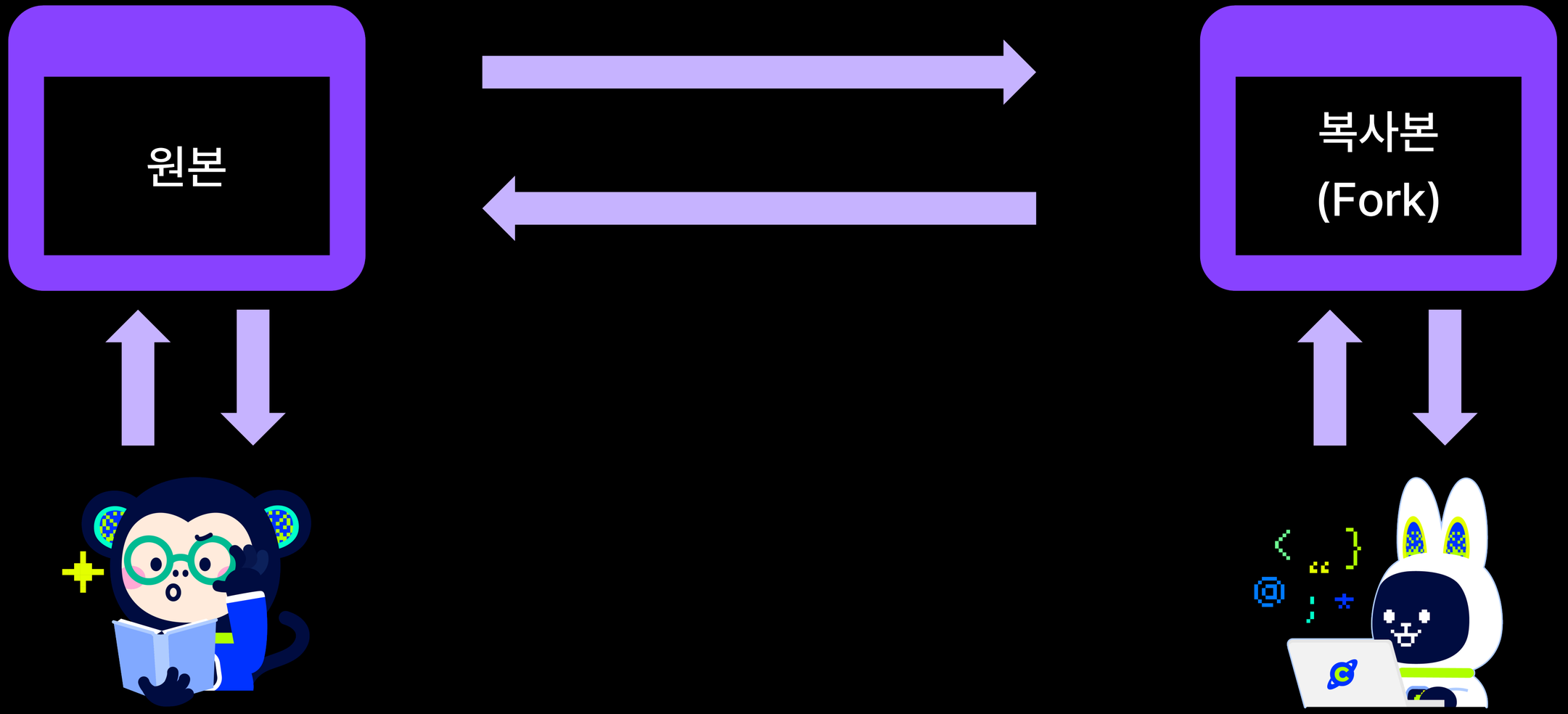
기존 Repository



Forked Repository

Fork 한 Repository에서 수정을 해도 원래의 Repository에는 반영이 안되는 모습

다른 사람과 협업하기(2) – Pull Request



그리고 이 둘을 동기화 시켜 주는 것이 Pull Request와 Merge

다른 사람과 협업하기(2) – Pull Request

원본

```
print('hi')
```

Pull Request

복사본

```
print('hi')
print('bye')
```

복사본에서 통합 요청을 하는 것 = Pull Request (PR)

다른 사람과 협업하기(2) – Pull Request

원본

```
print('hi')
print('bye')
```

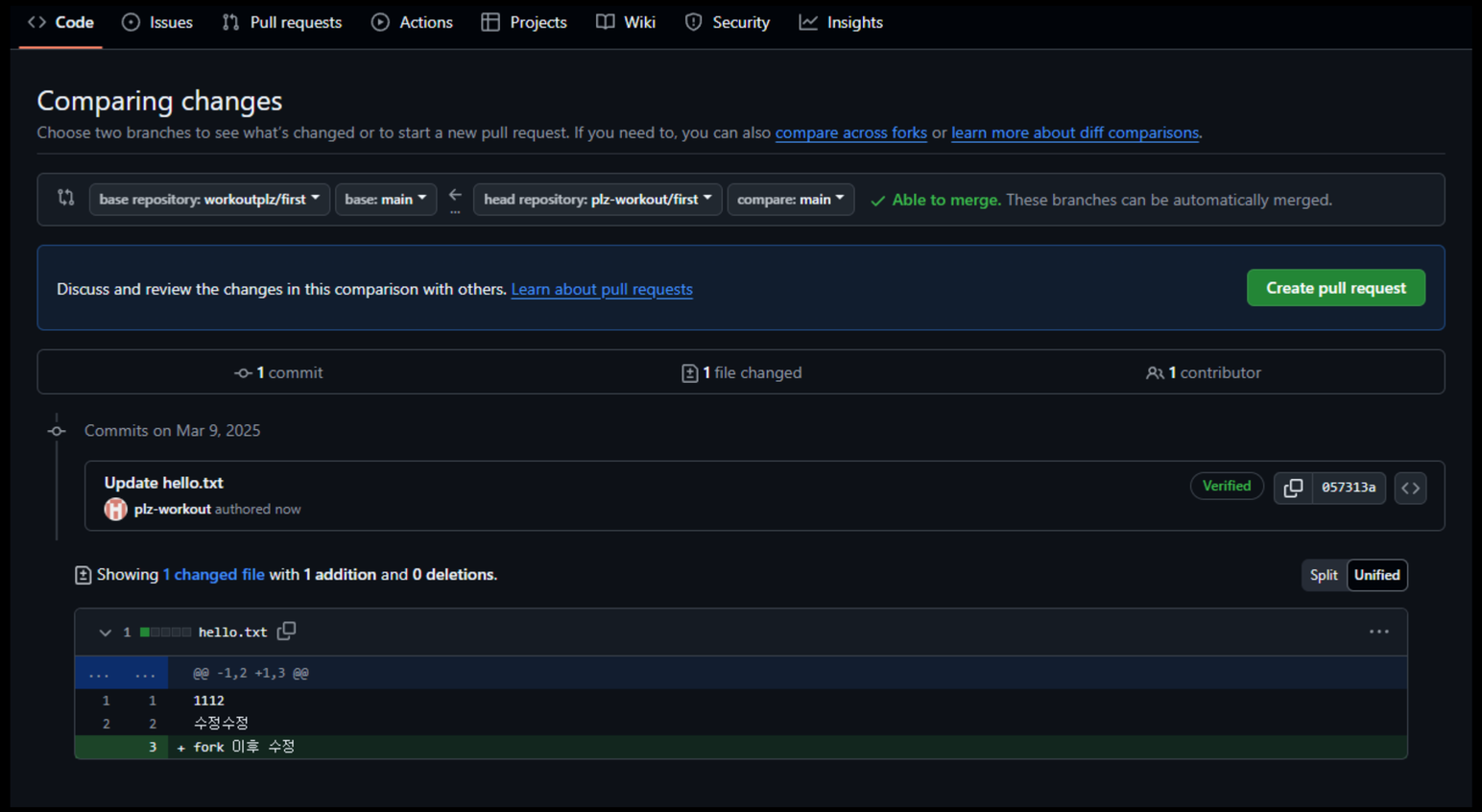
Pull Request

복사본

```
print('hi')
print('bye')
```

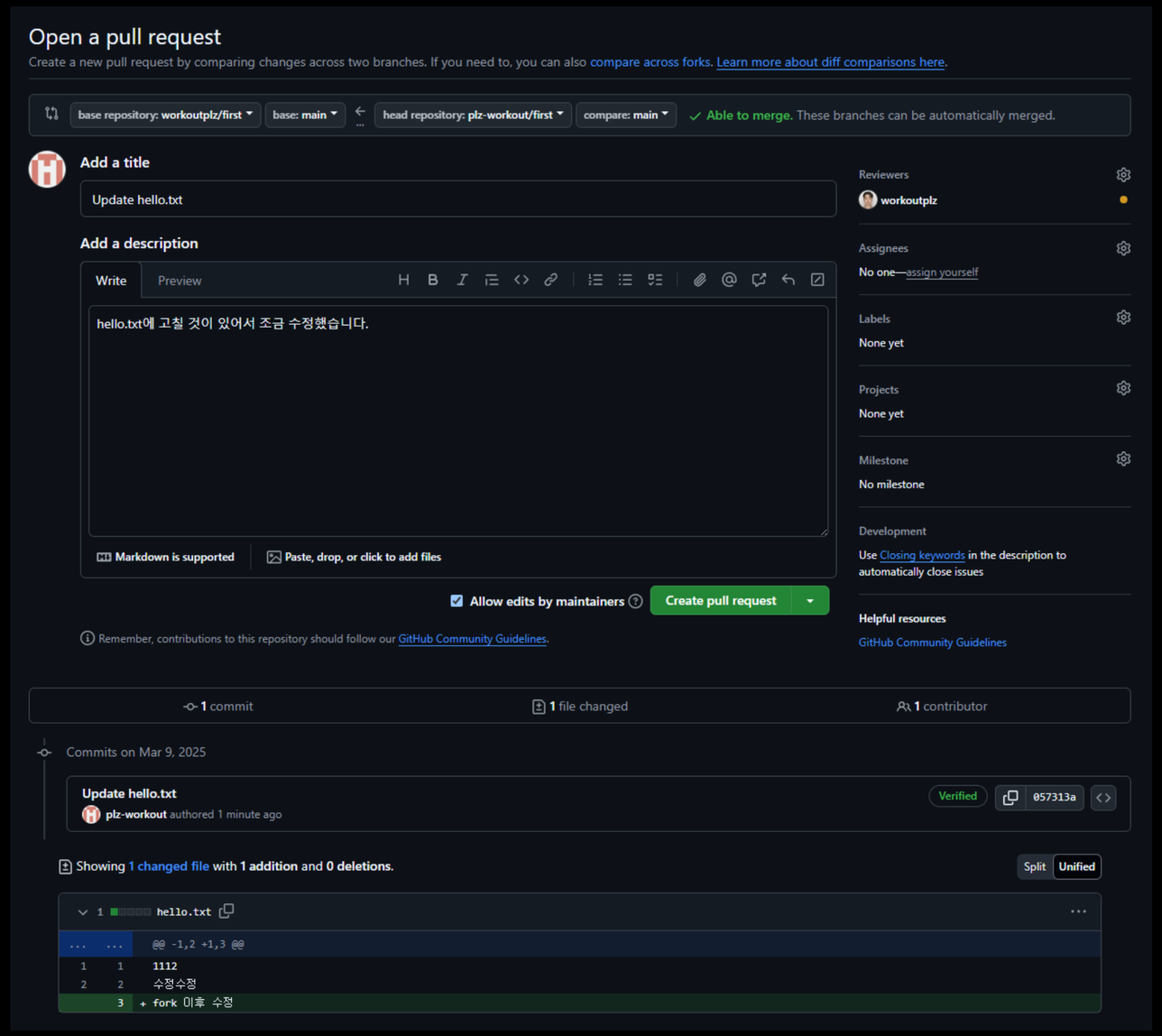
Pull Request를 승인 시 Merge 되어 원본 또한 바뀜

다른 사람과 협업하기(2) – Pull Request



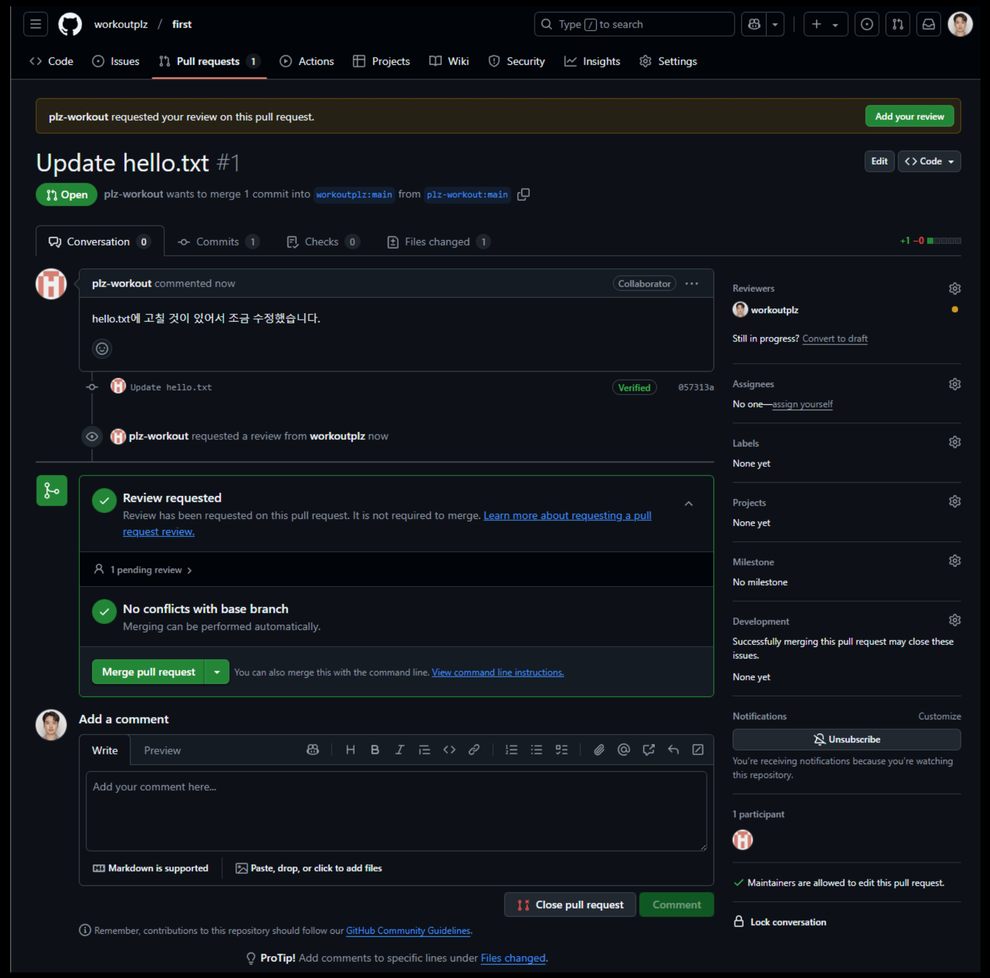
commit을 기준으로
Pull Request를 생성할 수 있음

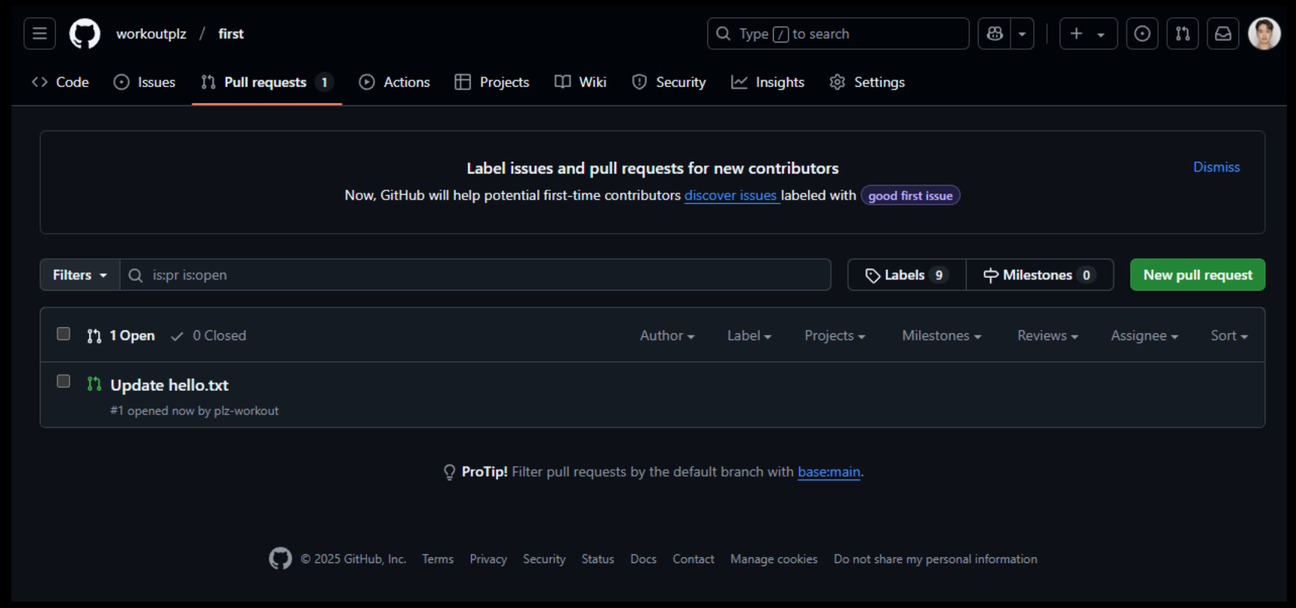
다른 사람과 협업하기(2) – Pull Request



- 어떤 변경이 있는지에 대한 설명을 작성
- 리뷰어가 필요한 경우 별도 지정

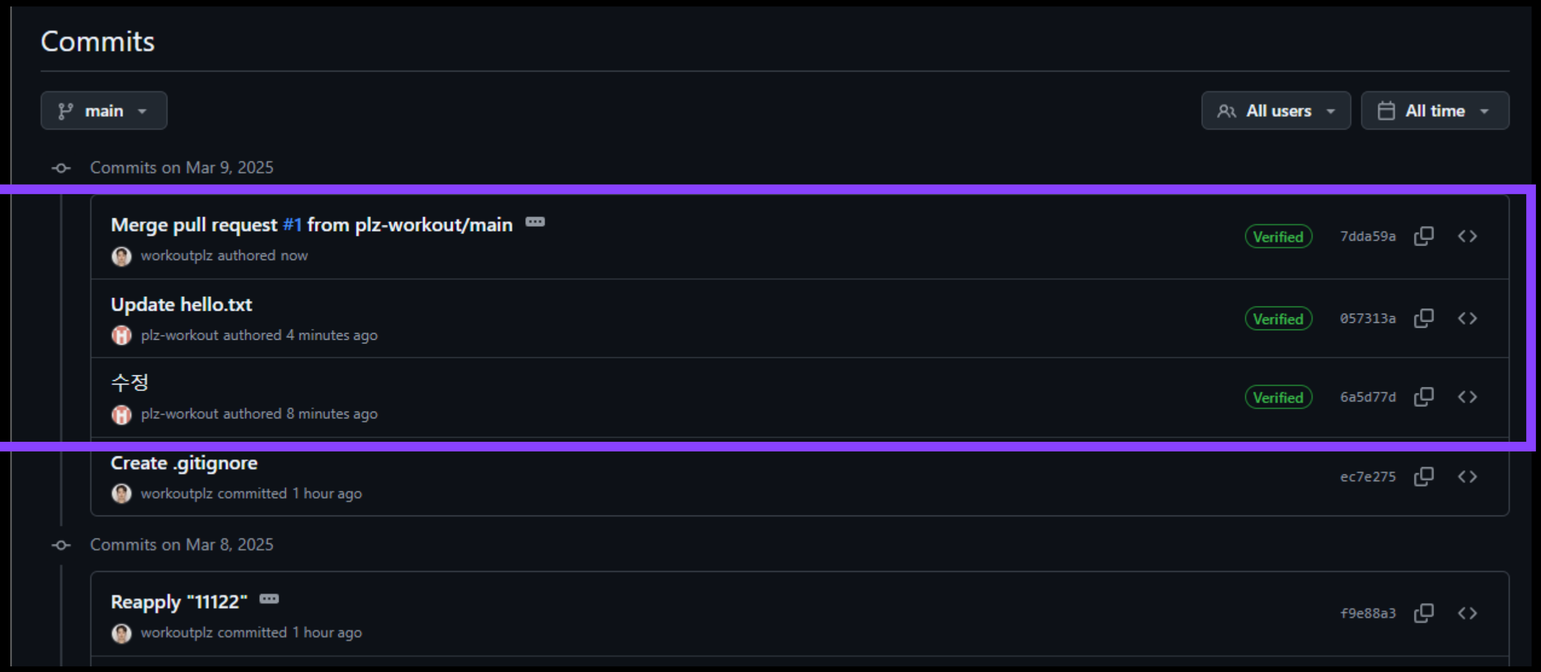
다른 사람과 협업하기(2) – Pull Request





Pull Request가 오면 원작자는 해당 내용을 리뷰 후 Commit 내용을 Merge 할 수 있음

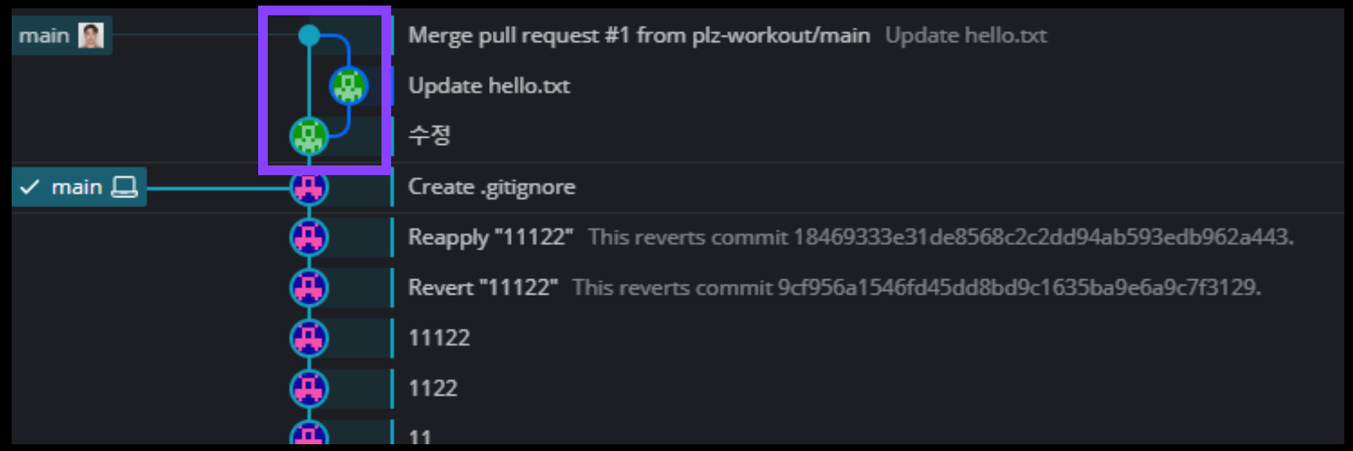
다른 사람과 협업하기(2) – Pull Request



Merge가 되면 정상적으로 Commit 내용들이 모두 정상적으로 반영 됨

다른 사람과 협업하기(2) – Pull Request

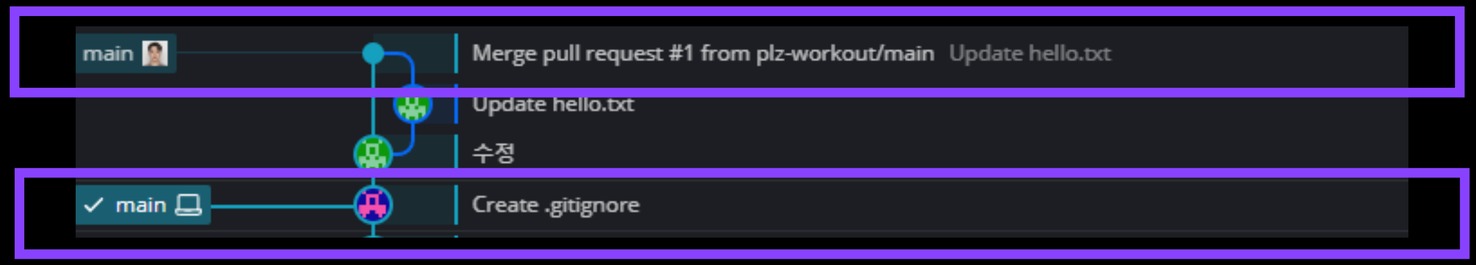
Pull Request & Merge 시점



변경 내용을 시각화 하면 위와 같이 나타낼 수 있음
(위 프로그램은 GitKraken)

다른 사람과 협업하기(2) – Pull Request

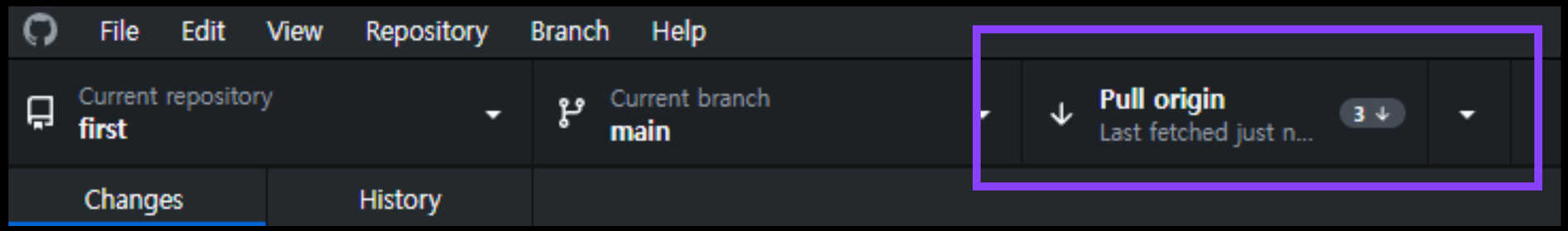
현재 Remote Repository의 버전



현재 Local Repository의 버전

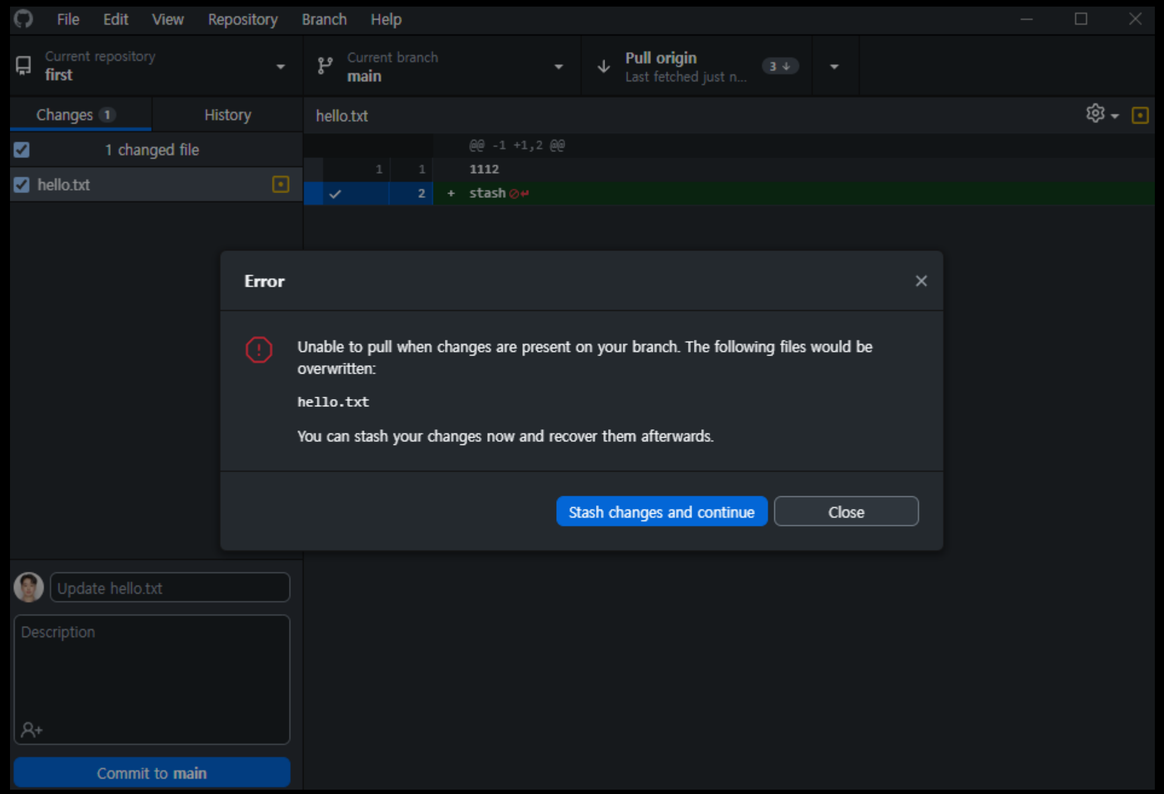
Merge 이후 Remote Repository와 Local Repository의 버전이 차이가 나면,
이는 곧 버전 충돌로 이어질 수 있으므로 반드시 동기화(Pull) 필요

다른 사람과 협업하기(2) – Pull Request



코드 작업시 Github Desktop에서 동기화(Pull) 할 것이 있는지
반드시 먼저 확인하고, Pull 이후 작업을 해야 함.

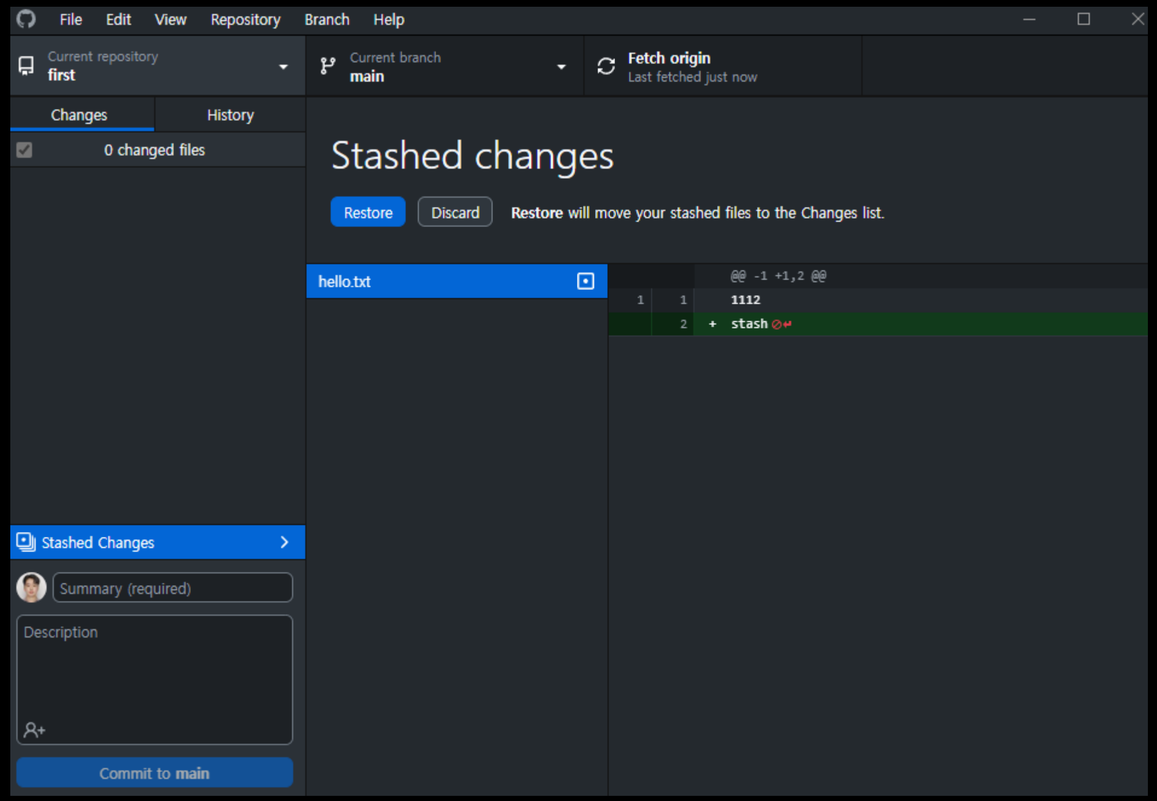
다른 사람과 협업하기(2) – Pull Request



로컬의 버전과 리모트의 버전 차이가 있을 경우,
Conflict가 발생

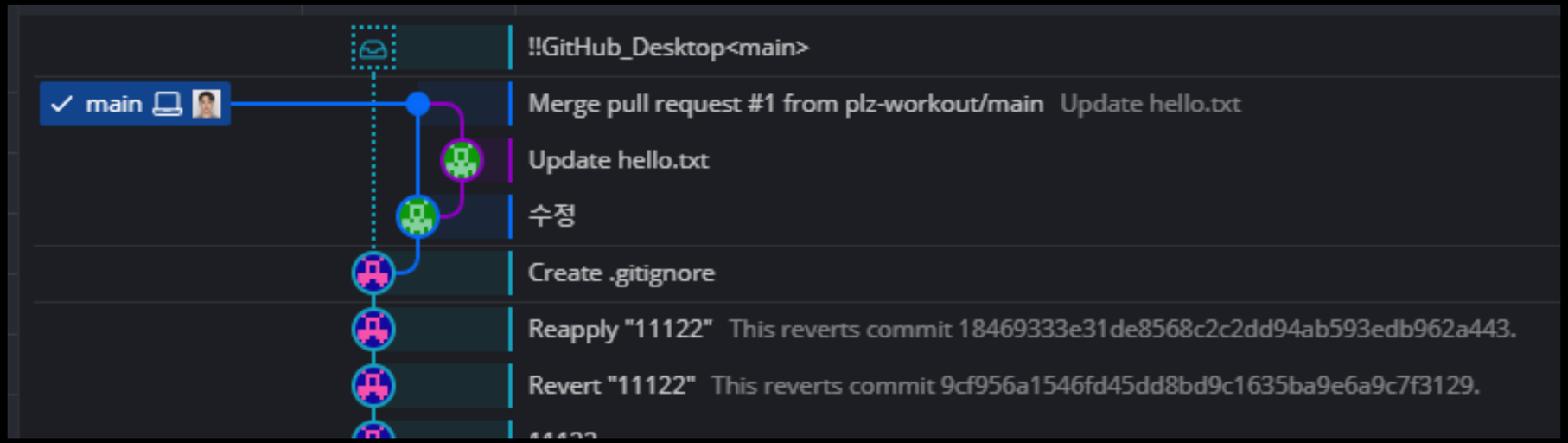
이 때 로컬의 수정 내용을
임시 저장하는 기능이 'stash'

다른 사람과 협업하기(2) – Pull Request



- 로컬의 변경 내용을
  임시 저장(stash)하고, Remote와 동기화
- 동기화 이후 Stashed changes를 확인하여
  이어서 작업 가능

다른 사람과 협업하기(2) – Pull Request



Remote Repository와 Local Repository가 동기화되고,
작업 내용 중 일부가 Stash 된 모습

다른 사람과 협업하기(2) – Pull Request

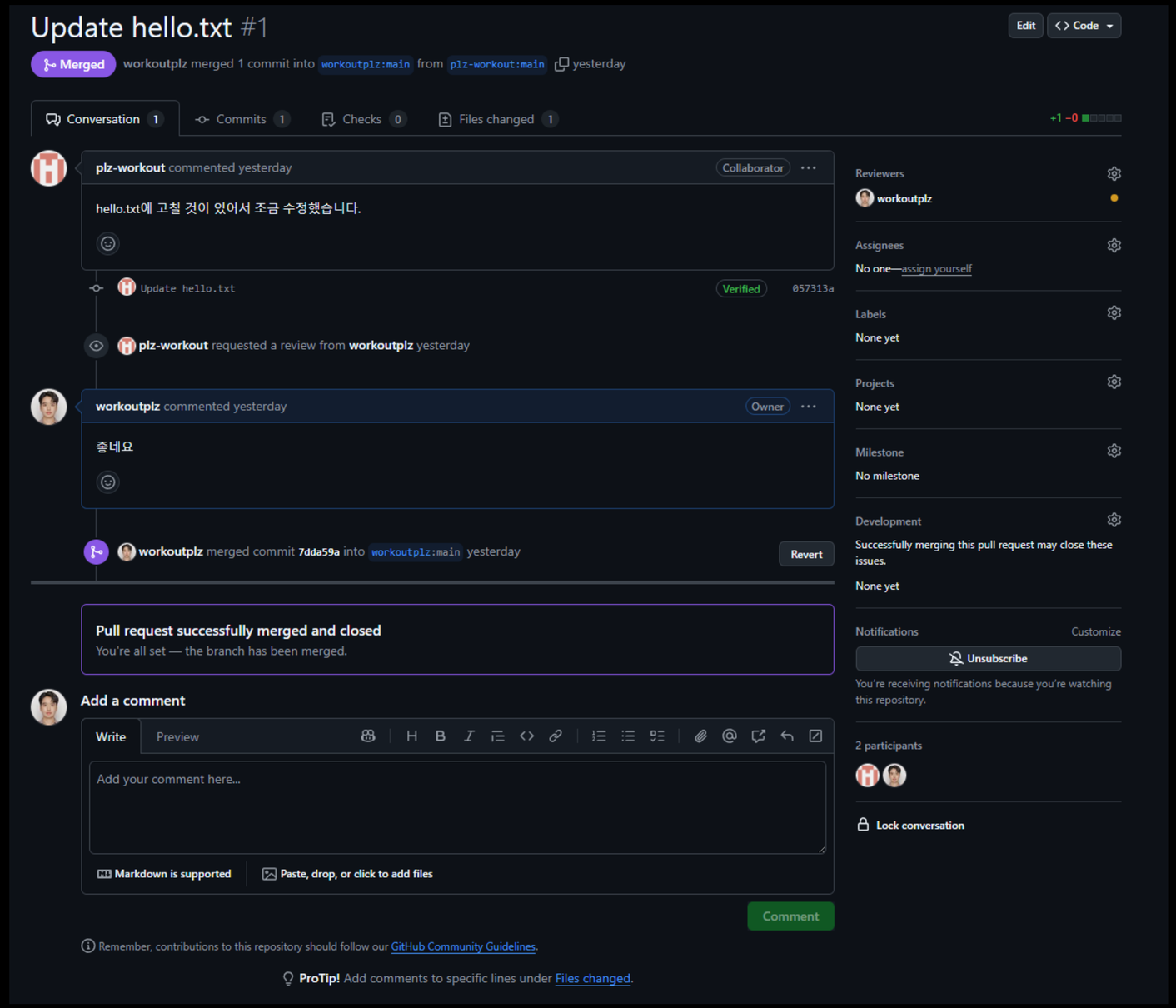
Pull Request(PR) & Merge 시 주의점

- PR 작성 시 내용을 명확히 전달하는 것이 중요
- 적절한 코드 리뷰를 통해 무분별한 수정과 Merge를 방지

Stash 사용시 주의점

- 편리하다고 남발하면 작업 내용이 꼬일 수 있으니, 남발해서는 안됨
- Stash는 어디까지나 임시 저장용이므로 중요한 변경 사항은 반드시 commit
- Stash가 무적은 아님. Stash 이후에도 충돌이 날 수 있으므로 Pull을 습관화
- 현재 branch의 변경 사항만 stash 됨 (branch 개념은 이후 학습 예정)

[실습6] PR & Merge



- 다른 수강생의 Repository에 Pull Request를 날려보세요.
- Pull Request를 받은 수강생 분들은 Comment를 남기고 Merge 해보세요.
- 최소 2번 이상의 Pull Request와 Merge를 해보세요.