

협업을 위한 Github

TMI

Ver.1.0 – 파일 생성

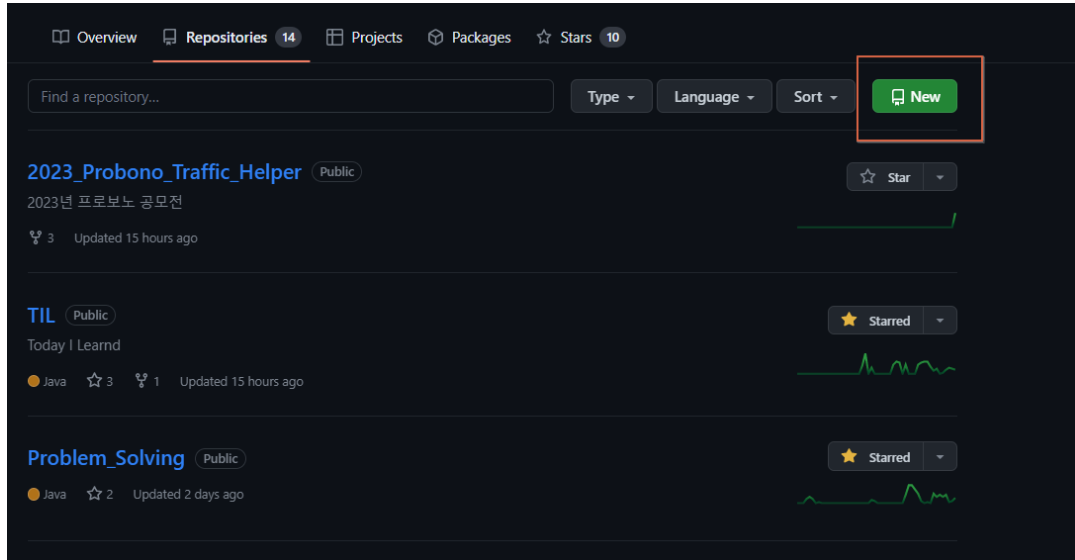
Ver1.1 – 개요 추가, 내용 초안 작성

1. 누구든 이 문서를 보고 도움을 받아도 됩니다.
2. 누구나 변경 가능합니다.
3. 팀장(리더)가 해야 할 항목은 빨간색으로 표시합니다.
4. 팀원이 해야 할 항목은 초록색으로 표시합니다.
5. 명령어는 형광펜으로 표시합니다.

순서

1. 깃 레포지터리 생성
2. Collaborators 추가
3. 깃 초기 설정
4. 깃 클론
5. 브랜치 생성
6. 브랜치 커밋
7. Main merge 요청
8. 필수 명령어

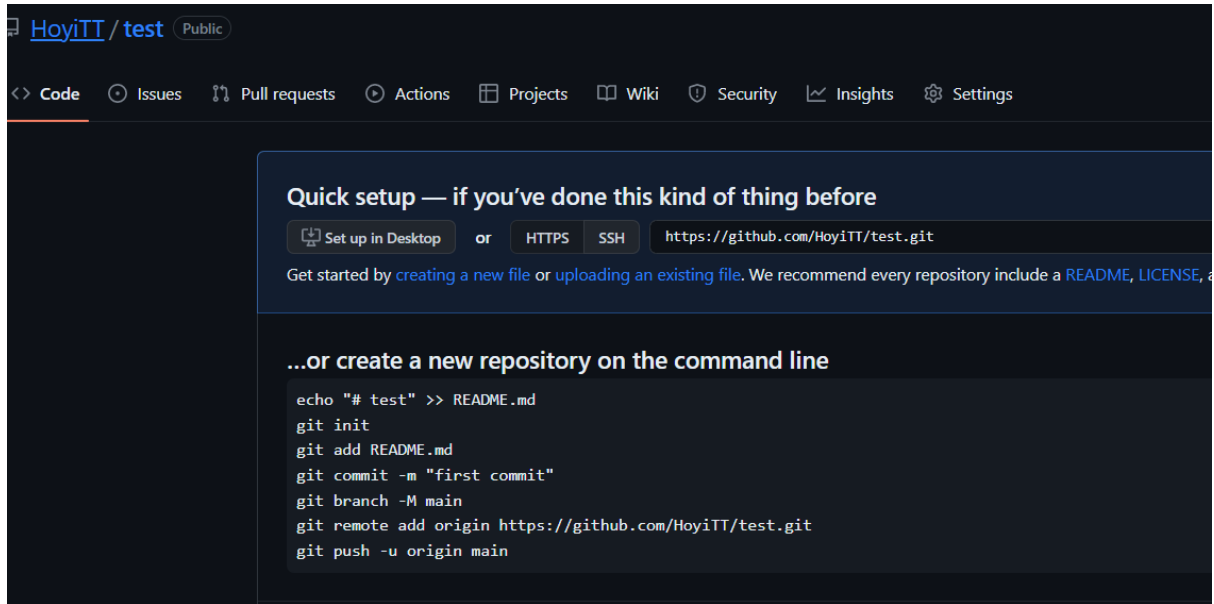
1. 깃 레포지터리 생성



New를 눌러 레포지터리를 생성한다.

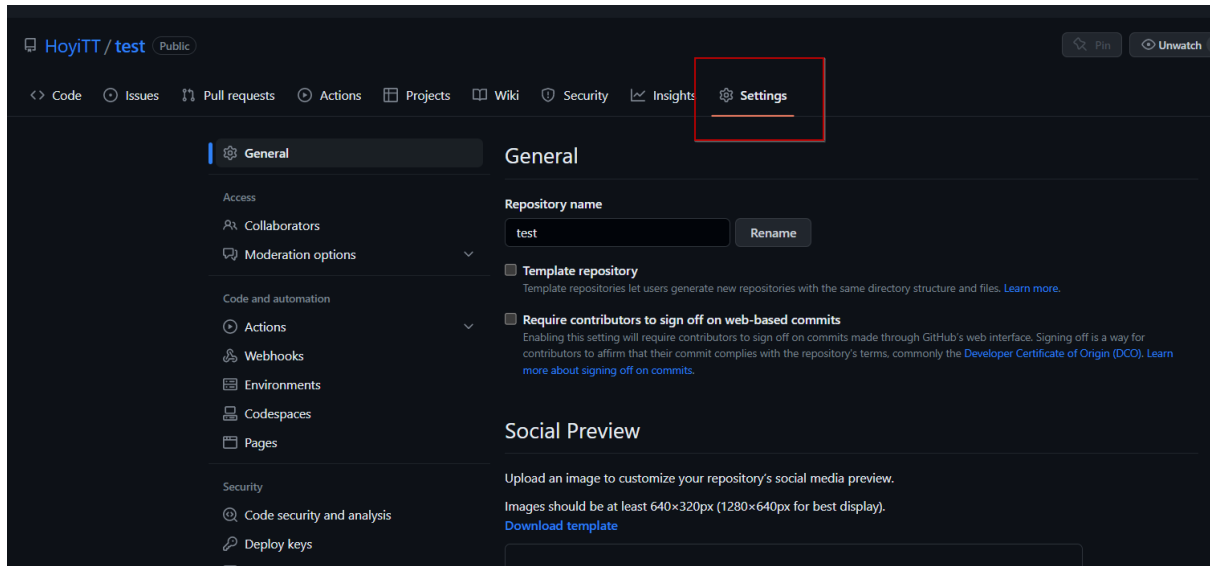
A screenshot of the 'Create a new repository' form in GitHub. The form is titled 'Create a new repository' and includes a subtitle: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. The form fields include: 'Owner' (HoyiTT) and 'Repository name' (test) with a green checkmark; a 'Description (optional)' text area; 'Public' (selected) and 'Private' radio buttons; 'Initialize this repository with:' section with 'Add a README file' (selected) and 'Add .gitignore' (selected); and 'Choose a license' (License: None). A green 'Create repository' button is at the bottom. A note at the bottom states: 'You are creating a public repository in your personal account.'

레포지터리 이름을 정한 뒤 다른 건 누르지 말고 create repository 한다.

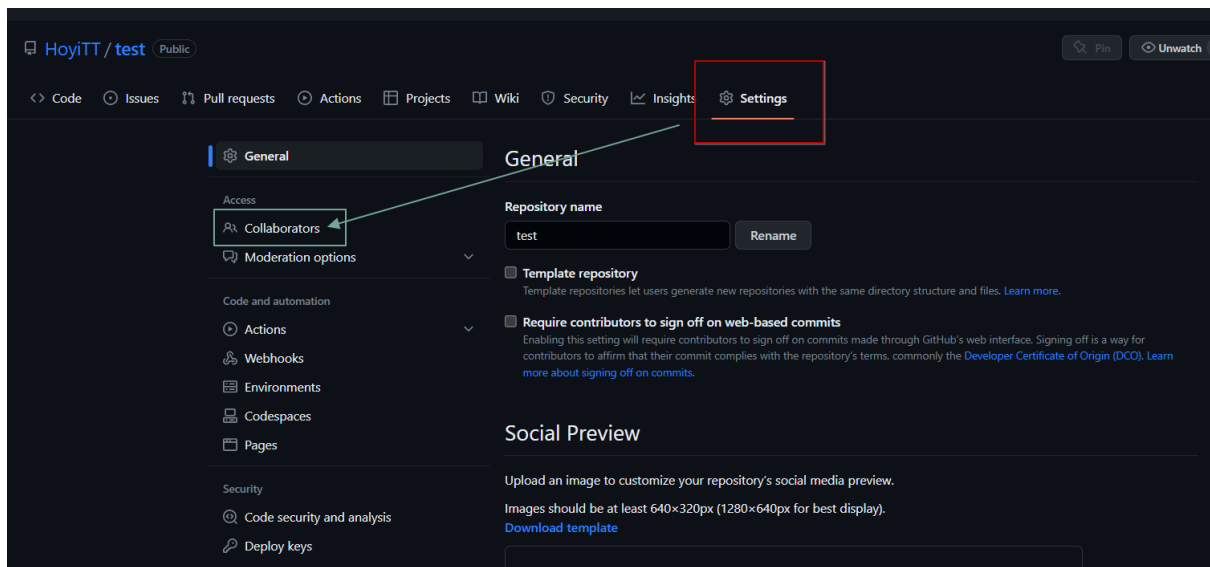


짠! 레포지터리가 생성되었다.

2. Collaborators 추가

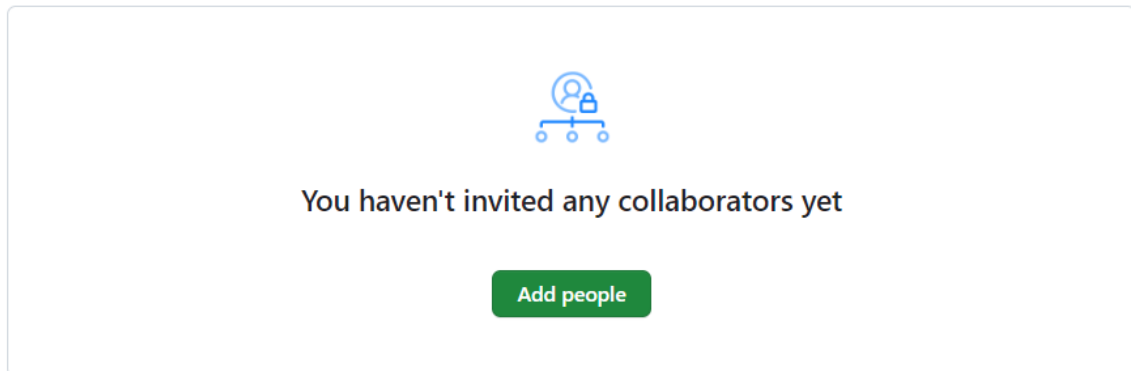


레포지터리 메뉴를 살펴보면 settings가 있다.
클릭해 들어간다.

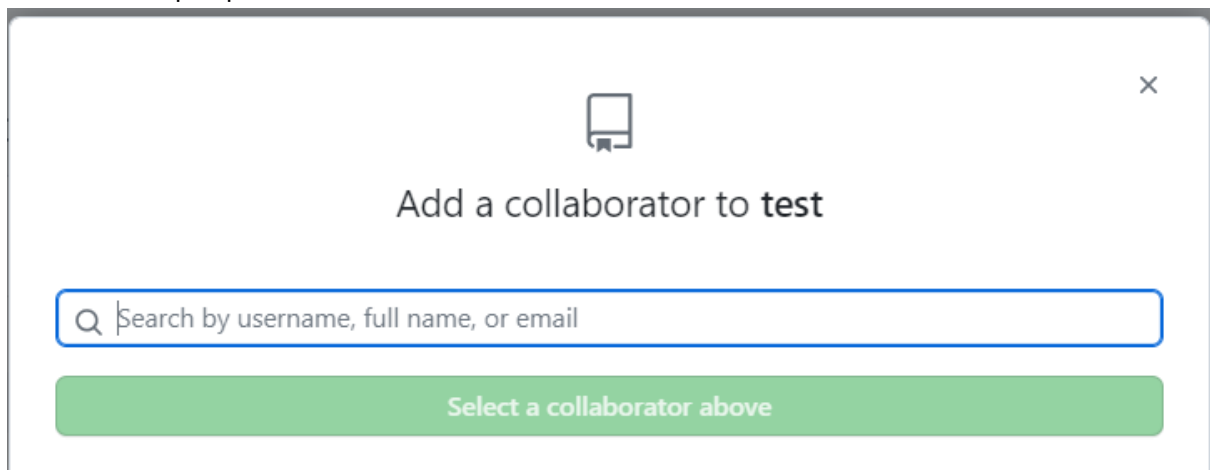


들어가면 맨 위에 Collaborators라는 메뉴가 있다.
클릭해 들어간다.

Manage access



보이는 Add people 클릭



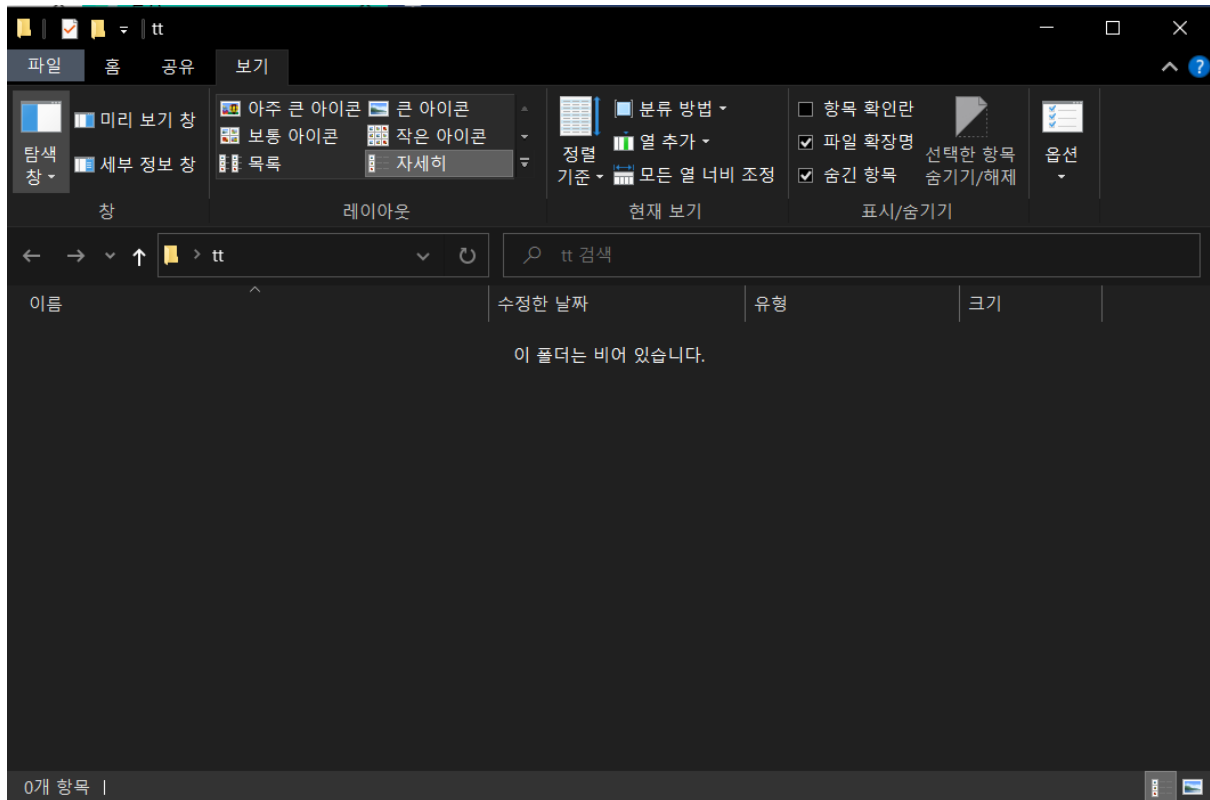
초대할 팀원의 깃허브 username이나 깃허브 가입한 이메일 주소를 입력해 초대한다.

팀원은 이메일을 확인하여 초대를 수락한다.

3. 깃 초기 설정

처음 레포지터리에 들어가면 이상한 알수없는 명령어마 잔뜩 있을 것이다.
겁먹지 말고 차근차근 해보자.

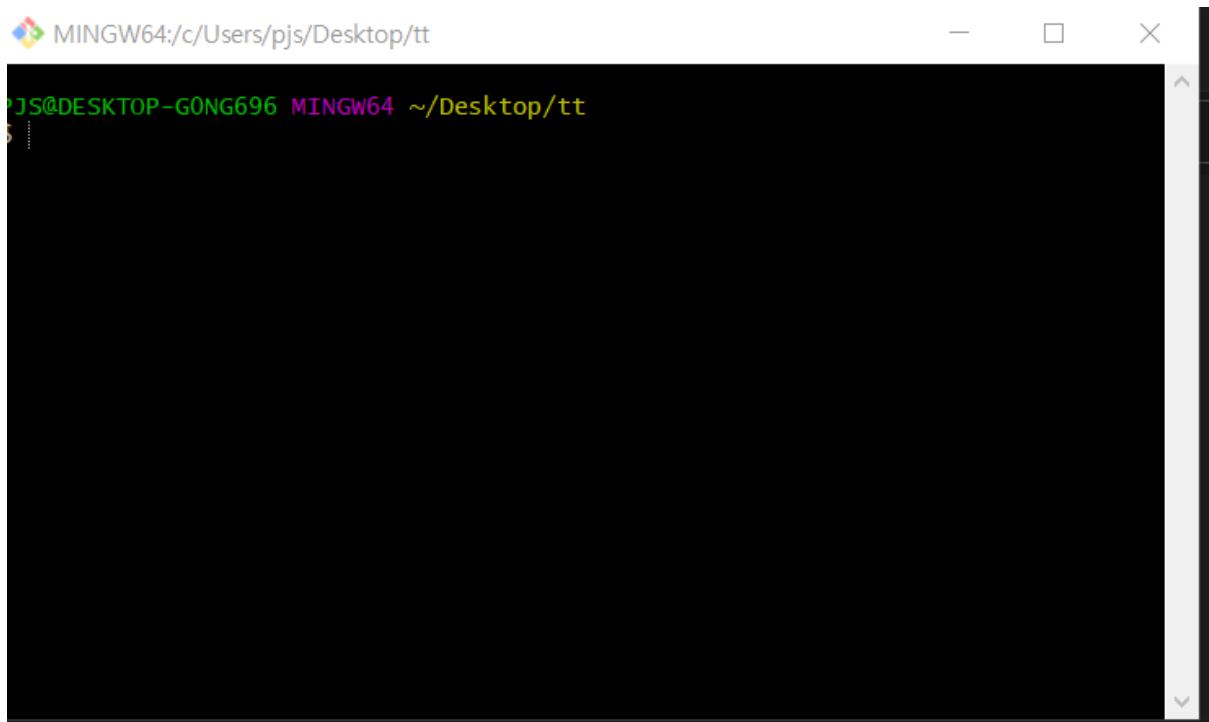
일단 깃허브와 연동해 사용할 폴더에 들어간다.



이 폴더에서 빈공간에 우클릭한다.

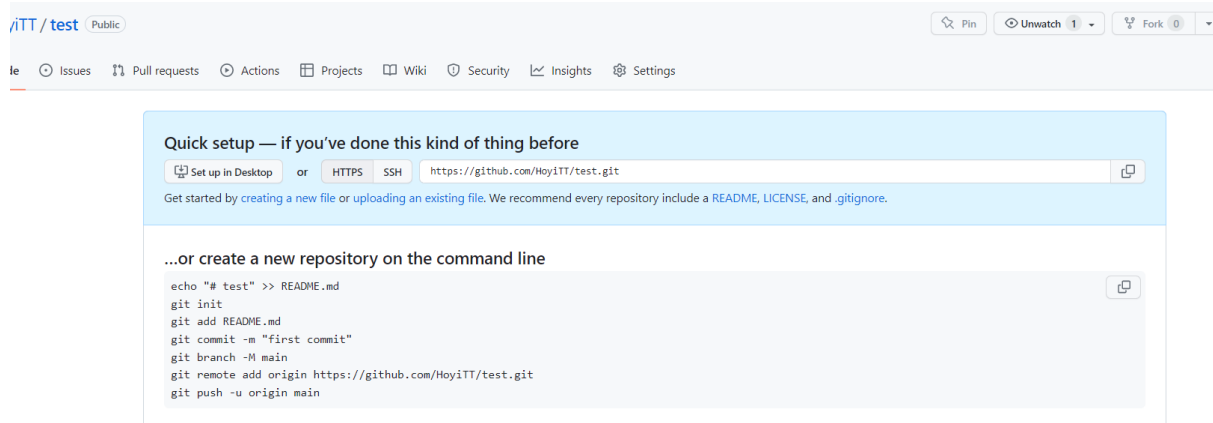


Git이 깔려 있다면 Git Bash Here을 눌러 CLI를 연다.
(다른 터미널이 편하다면 그걸 사용해도 된다.)
만약 보이지 않는다면 Git을 설치하고 다시 시도한다.



이렇게 터미널이 나온다.

이제 이 터미널에 명령어를 친다.



이 화면에 적힌 거 그대로 적으면 된다.
(형관펜 친 거 적으면 된다.)

```
echo "# test" >> README.md
```

➔ # (repository name) 을 README.md에 저장

```
git init
```

➔ 깃을 사용하기 위한 초기 설정

```
git add README.md
```

```
git commit -m "first commit"
```

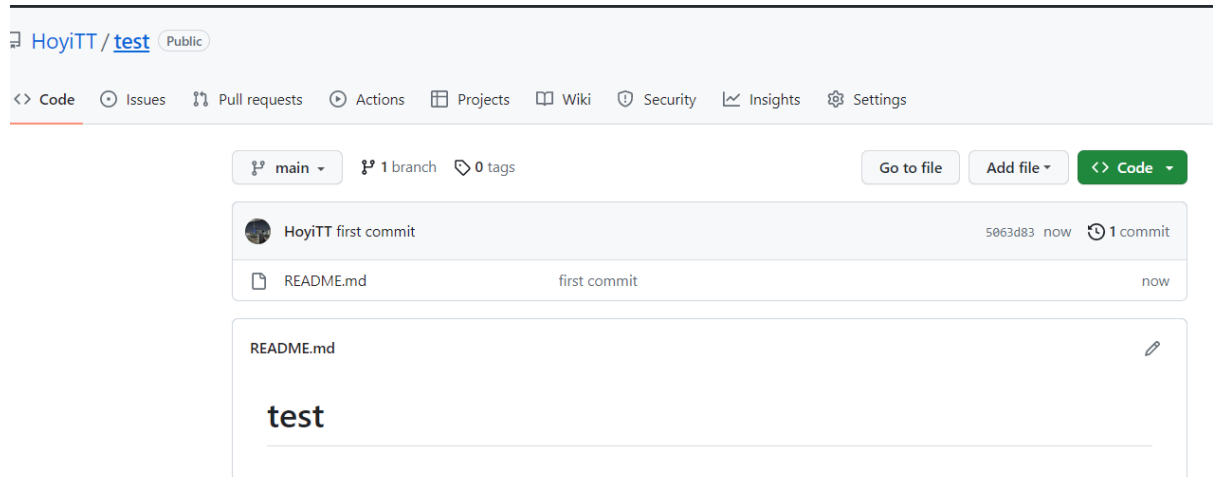
```
git branch -M main
```

```
git remote add origin https://github.com/HoyiTT/test.git
```

```
git push -u origin main
```

(귀찮아서 설명을 나중에 누군가 추가해주길 바란다.)

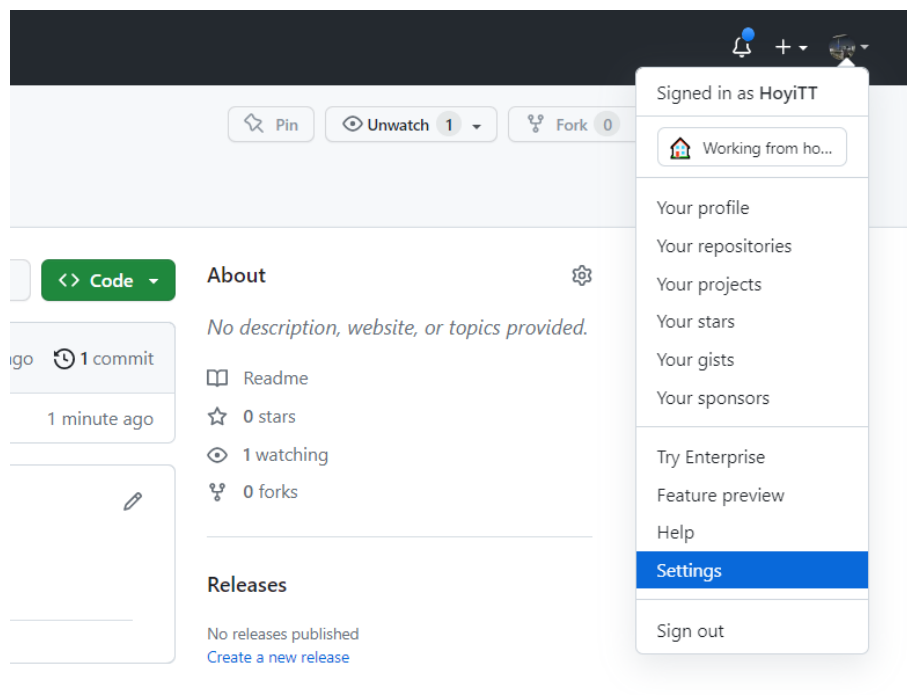
썻든 다 입력하면



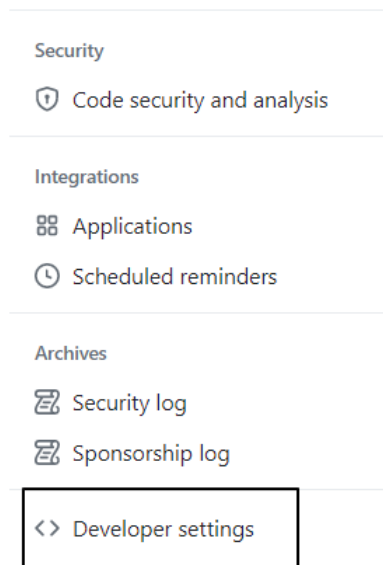
짠 이렇게 우리의 repository가 달라졌다.

만약 여기서 로그인 창이 나타난다면

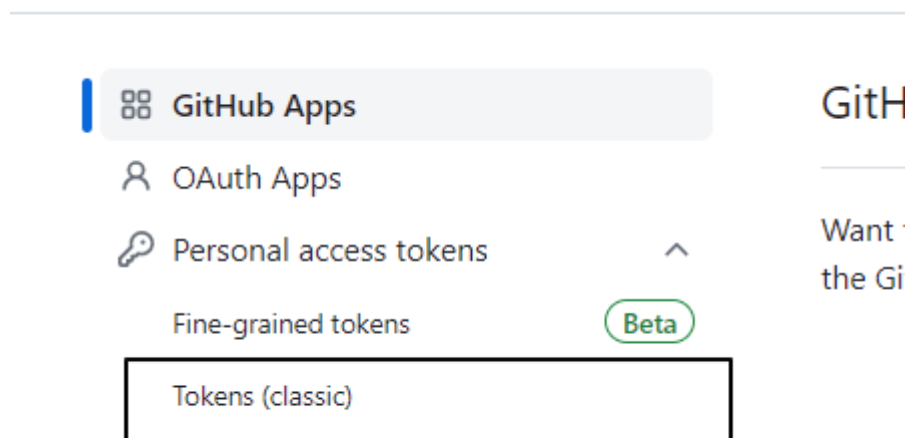
아이디를 입력했는데 비밀번호를 적으라는 창이 나타난다면!?



우측 상단 Settings 클릭



메뉴 제일 아래 Developer Settings 클릭



Personal Access tokens -> Tokens(Classsic) 클릭



Generate new token(Classic) 클릭 후

이름 아무거나 적고

Expiration -> No Expiration 선택 후

여러가지 체크박스 모두 체크 후 맨 아래 generate를 누르면

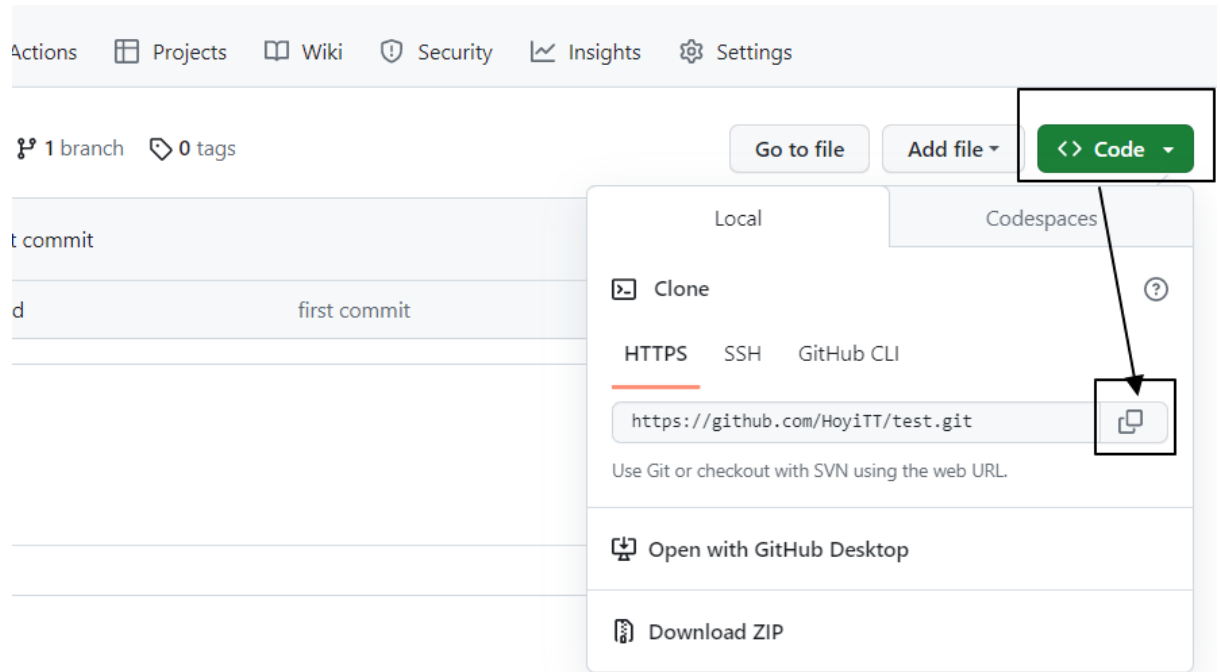
이상하게 긴 난수?문자열이 나오는데

이 문자열을 복사해서 아까 명령어 칠 때 나타난

깃 허브 로그인 창 비밀번호로 입력하면 된다.

4. 깃 클론

초대 받은 팀원들은 이제 인터넷으로 깃허브 레포지토리로 들어간다.



레포지터리에서 Code 버튼을 누르고
그 다음 복사 버튼을 누른다.

그 후 자신의 컴퓨터에 이 레포지토리를 저장할 폴더를 열고

위에처럼 Git Bash를 연 뒤

Git clone 복사한 링크

```
MINGW64:/c/Users/pjs/Desktop/tt/tt1

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 217 bytes | 217.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/HoyiTT/test.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ mkdir tt1

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ ls
README.md  tt1/

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ cd tt1

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt/tt1 (main)
$ git clone https://github.com/HoyiTT/test.git
```

하면 그대로 깃허브에 있는 레포지터리가 자신의 컴퓨터로 복사된다.

5. 브랜치 생성

깃에는 브랜치라는 것이 있다.

음.. 일단은 그냥 쉽게 말하자면

Main 정식버전 올리는 폴더가 있고

내가 작업하고 정식버전 올리기 전에 올리는 폴더가 있다고 생각하자..

이해가 안된다고?

정상이다.

일단 git clone 한 폴더에 들어가 git bash를 열어준다.

그 후

git branch 브랜치 이름
을 입력한다.

브랜치 이름은 팀끼리 규칙을 정해서 만드는게 좋다.

```
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ git branch pjs
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$
```

아무런 반응이 없어 당황했을 수 도 있다.

그럴땐 git branch 를 쳐보자.

```
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ git branch
* main
  pjs
```

Main 브랜치와 pjs 브랜치가 생긴 것을 확인할 수 있다.

6. 브랜치 커밋

이제 나의 브랜치로 커밋을 해보자.

Git checkout 내가 만든 브랜치 이름

```
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (main)
$ git checkout pjs
Switched to branch 'pjs'

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (pjs)
$
```

앞에 붙어있던 (main) 이 (pjs)로 바뀐 것을 알 수 있다.

나의 현재 브랜치가 내가 생성한 브랜치로 바꿨다!

이제 커밋을 해보자.

이 폴더에 파일을 하나만 만들어보자.

이름	수정일 날짜	유형	크기
.git	2023-04-07 오후 4:02	파일 폴더	
README.md	2023-04-07 오후 3:43	MD 파일	1KB
11.txt	2023-04-07 오후 4:04	텍스트 문서	0KB

하나의 파일을 만들고

다시 터미널로 가

git add .

git commit -m "적고 싶은 메시지"

```
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (pjs)
$ git add .

PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (pjs)
$ git commit -m "Add file"
[pjs 1c5bb57] Add file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 11.txt
```

그 후

git push origin 생성한 브랜치 이름

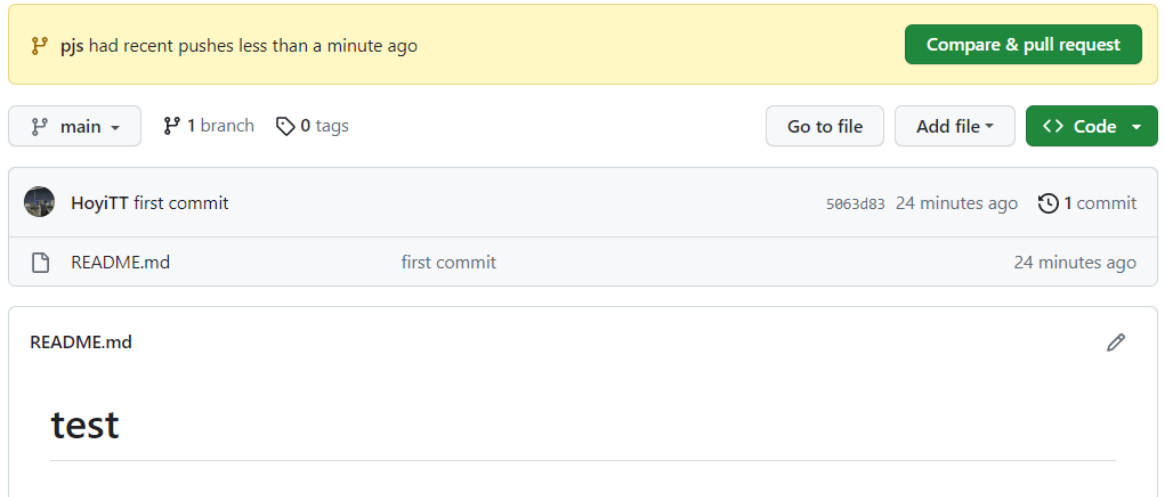
을 하면

```
PJS@DESKTOP-G0NG696 MINGW64 ~/Desktop/tt (pjs)
$ git push origin pjs
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 271 bytes | 135.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'pjs' on GitHub by visiting
remote:   https://github.com/HoyiTT/test/pull/new/pjs
remote:
To https://github.com/HoyiTT/test.git
 * [new branch]      pjs -> pjs
```

뭔가 많은 메시지와 함께 뭔가 된 거 같다.

7. Main merge요청

이제 다시 깃허브 사이트로 돌아가보자.

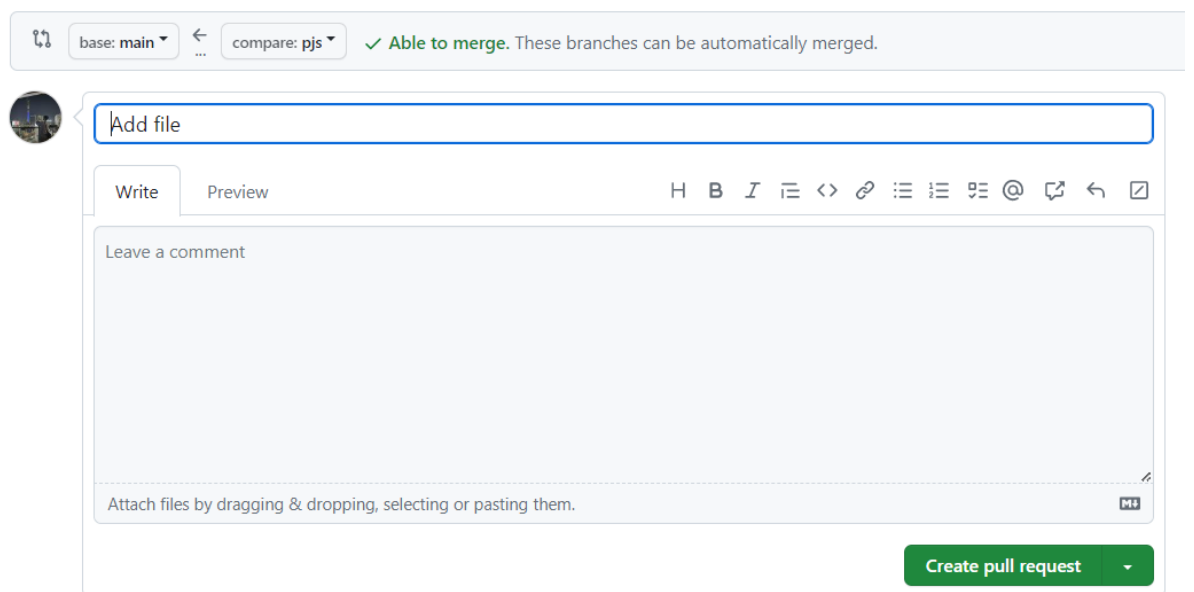


분명 push를 했지만 11.txt가 생성되지 않았다.

그리고 위에는 웬 Compare & pull request 라는 버튼이 생겼다.

Open a pull request

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



Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

누르면 이러한 화면과 함께 Pull request 요청을 할 수 있다.


Pull request는 쉽게 말해


내가 생성한 브랜치에 내용을 Main에 병합해도 되겠냐는 요청을 리더에게 보내는 것이다.

이제 팀장은 레포지토리로 돌아와


Pull request로 들어가


Add more commits by pushing to the `pjs` branch on `HoyiTT/test`.



 **Require approval from specific reviewers before merging**
Branch protection rules ensure specific people approve pull requests before they're merged.

[Add rule](#) ×


 **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](#).

Mergr pull request 를 눌러주면

 `main` ▼


 2 branches

 0 tags


[Go to file](#)


[Add file](#) ▼


[Code](#) ▼


 **Your main branch isn't protected**
Protect this branch from force pushing or deletion, or require status checks before merging. [Learn more](#)


[Protect this branch](#) ×

 **HoyiTT Merge pull request #1 from HoyiTT/pjs** ...

`b548a68` now  3 commits

 11.txt Add file 6 minutes ago

 README.md first commit 28 minutes ago

README.md 

test

Main에 11.txt가 생성된 것을 확인할 수 있다.

8. 필수 명령어

프로젝트를 하다보면 내가 가진 로컬 프로젝트 폴더에는 아직 없는 파일이 Main 브랜치에는 있는 꼬이는 상황이 발생한다.

그런 상황을 방지하기 위해 `git pull origin main`을 입력해주면 현재 내 로컬 프로젝트 폴더에 main에 상황이 덮어씌워진다.

그냥 쉽게 말해 main폴더의 상황이 나에게 적용된다고 생각하면 된다.

- * [Feat] : 새로운 기능 추가
- * [Refact] : 기능상의 변화는 없지만, 코드가 정리 되었을 때
- * [Fix] : 버그 & 오류 & 오타자 수정
- * [Impv] : 성능의 개선
- * [Docs] : 문서 작업
- * [Test] : 테스트 코드 작성 시
- * [Add] : 이미지나 설정파일 등과 같은 파일이 추가되었을 때

커밋메시지를 적을 땐 이러한 규칙을 지켜주면 좋다.

작성 및 수정 : 박준서,