

협업용 가이드

Written by yunarang-dev

Github 협업 가이드

기본 Workflow

(기본적으로 Github Desktop을 기준으로 설명합니다. 원하시는 경우, Fork 등을 사용해도 돼요.)
(버전/로그인 여부 등에 따라 일부 화면이 다를 수 있어요.)

1. main 최신화(Pull)
2. 작업 branch 만들기
3. 작업 branch / 개인 Workspace 폴더에서 작업
 1. 각각의 작업마다 commit
4. 모든 작업이 끝나면 Publish / Push
5. Pull Request(PR) 생성 후 Merge

위 과정을 반복하면 됩니다.

- **main**은 팀이 함께 쓰는 기준 버전이에요.
- **branch**는 각각의 작업을 나누는 공간이에요.
- **commit**은 컴퓨터에 남겨 두는 변경 기록이에요.
- **Push**로 작업한 기록을 Github에 올릴 수 있어요.
- **PR**로 확인을 요청한 뒤, **Merge**로 main에합칠 수 있어요.

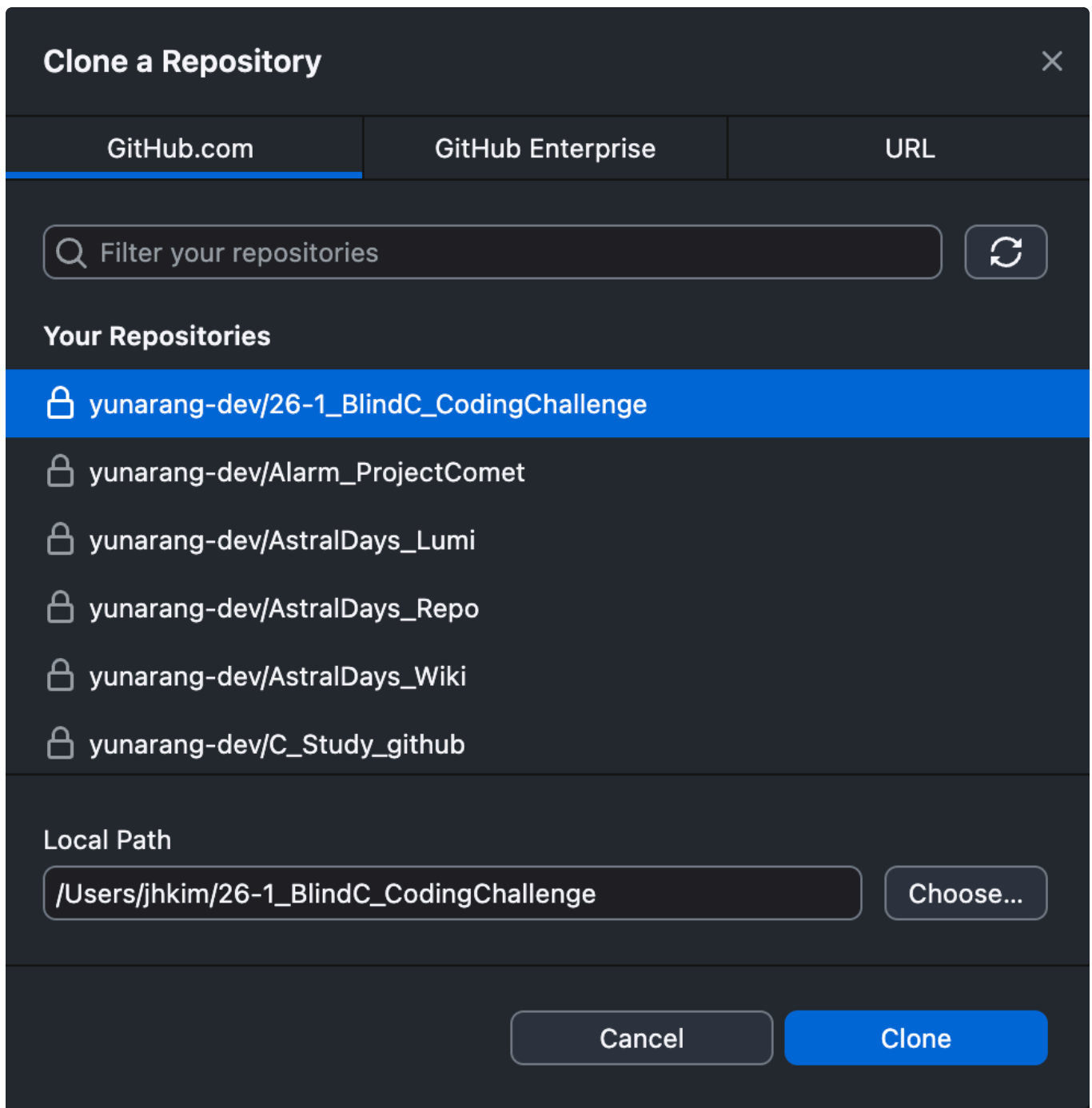
기본적으로 **main**을 직접 수정하지 않습니다.

main은 항상 함께 실행하고 테스트할 수 있는 상태로 유지할 예정이에요

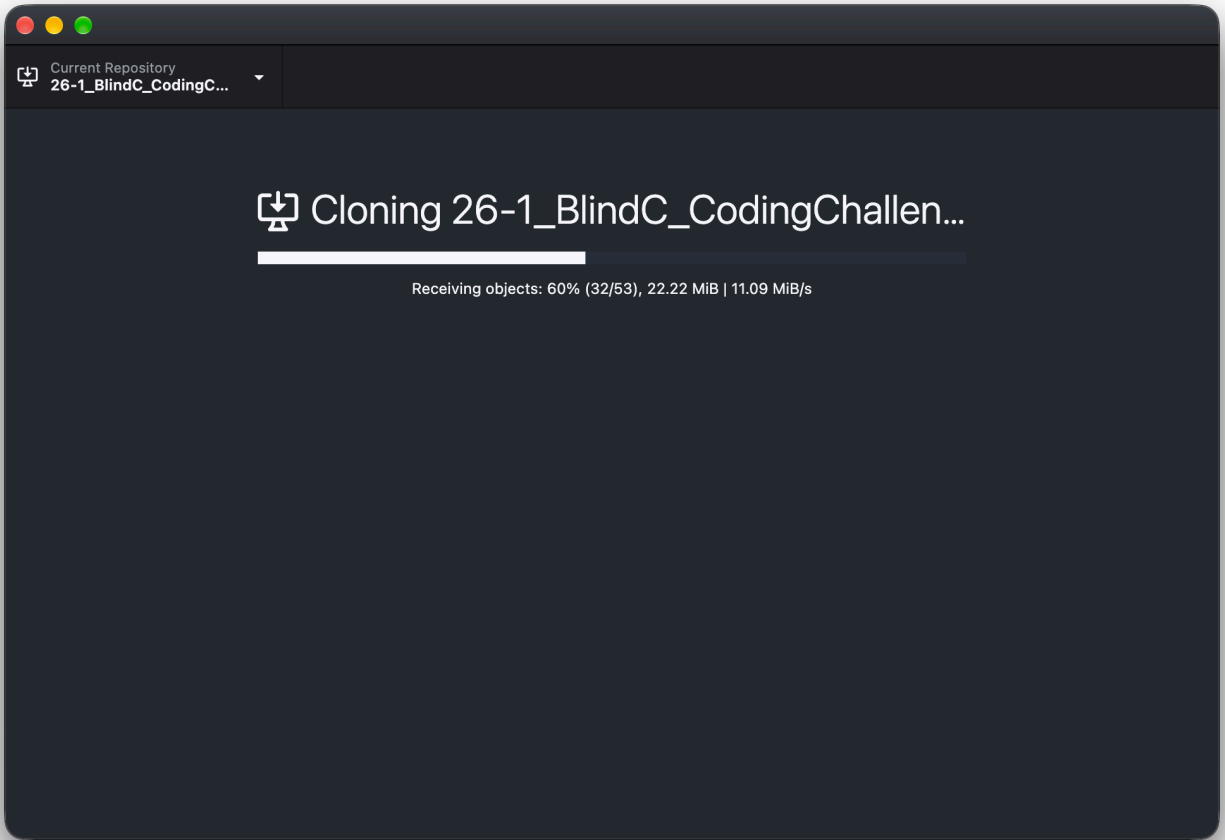
1. Repository Clone

가장 먼저 저장소 초대를 받으면 수락해 주세요.

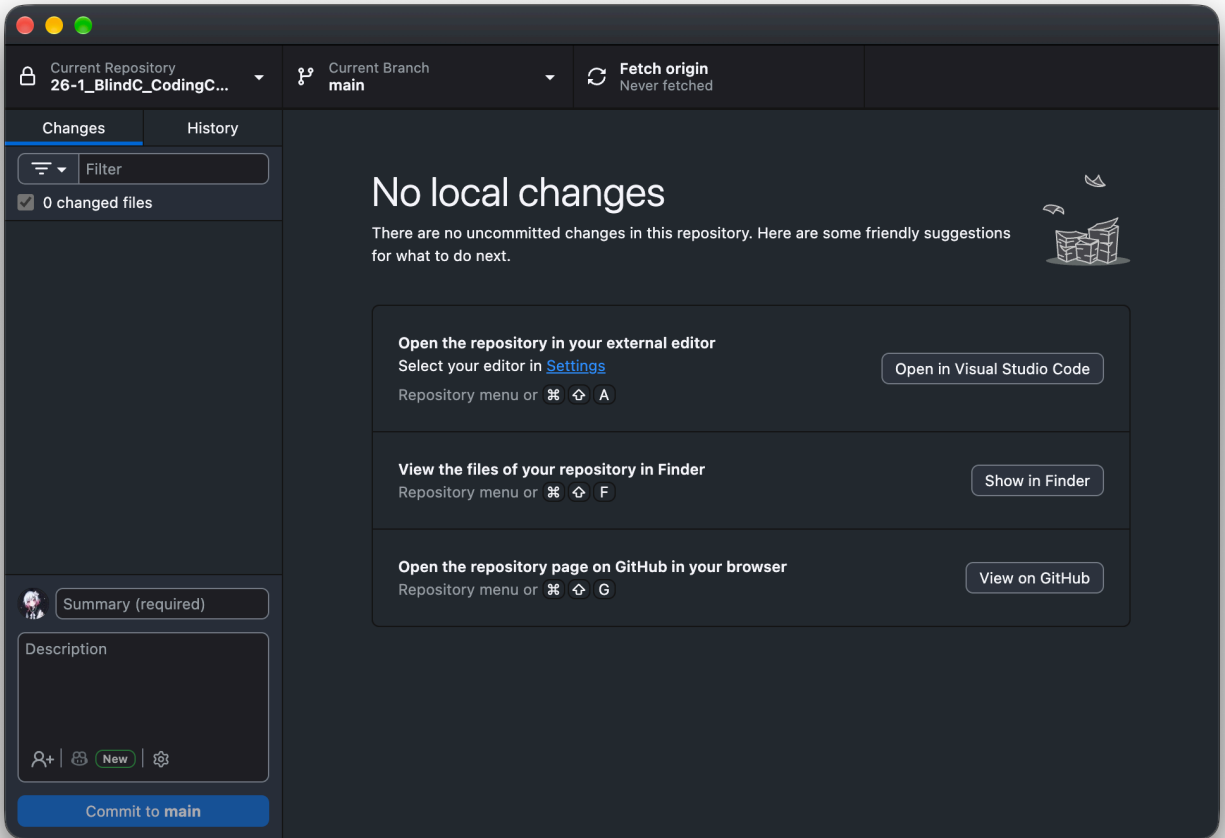
그 후, Github Desktop을 작업 컴퓨터에 설치한 뒤 로그인해요.



(예시 이미지입니다. 실제로는 TRAXXING Repository를 Clone해야 해요.)



(이런 Clone 화면이 뜬 뒤, 아래와 같은 화면이 뜨면 제대로 Clone이 된 거예요.)



2. main 선택 및 최신화

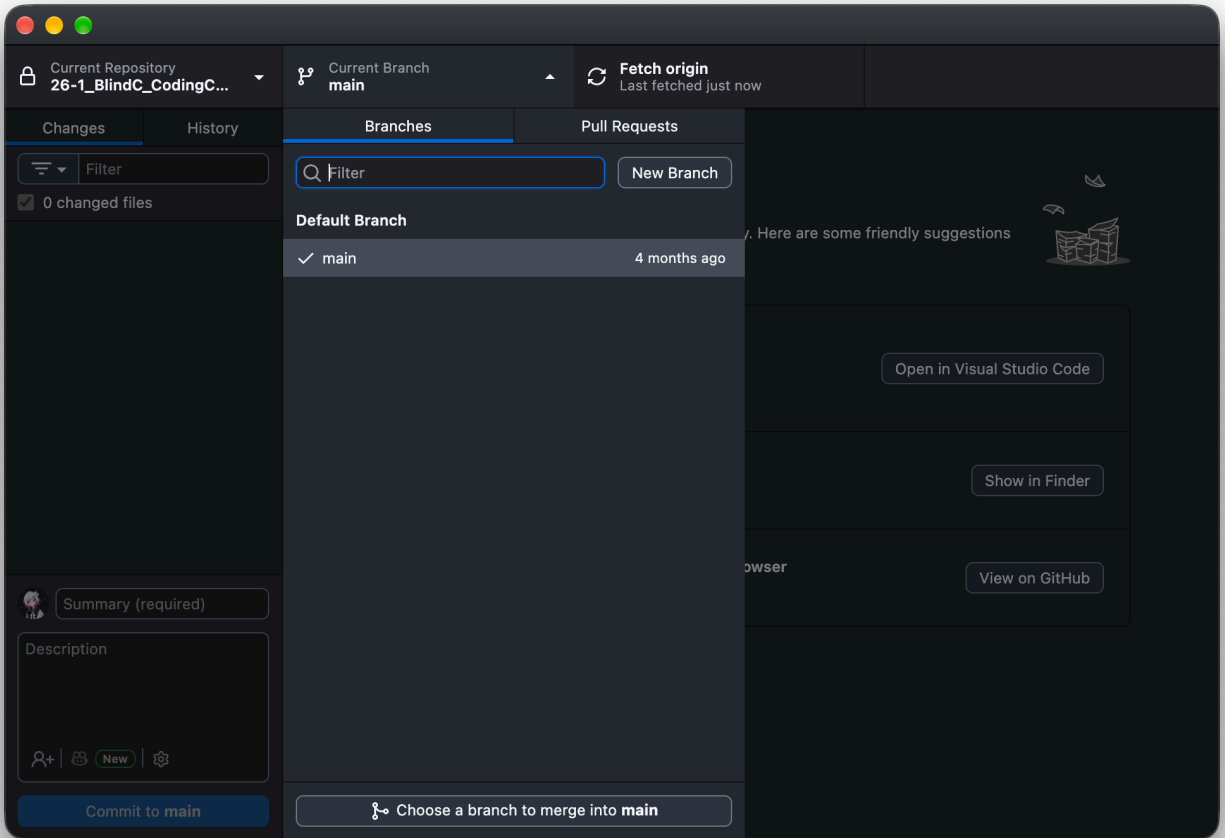
가장 먼저 **Current Branch**가 **main**인지 확인해요. **main**이 아닌 경우, **main**으로 바꿔주세요.

그 후, **Fetch Origin**을 눌러요. 로컬 파일에 최신화가 필요한 경우, **Pull Origin** 버튼이 표시돼요. 이 때는 **Pull**까지 눌러야 파일에 반영돼요.

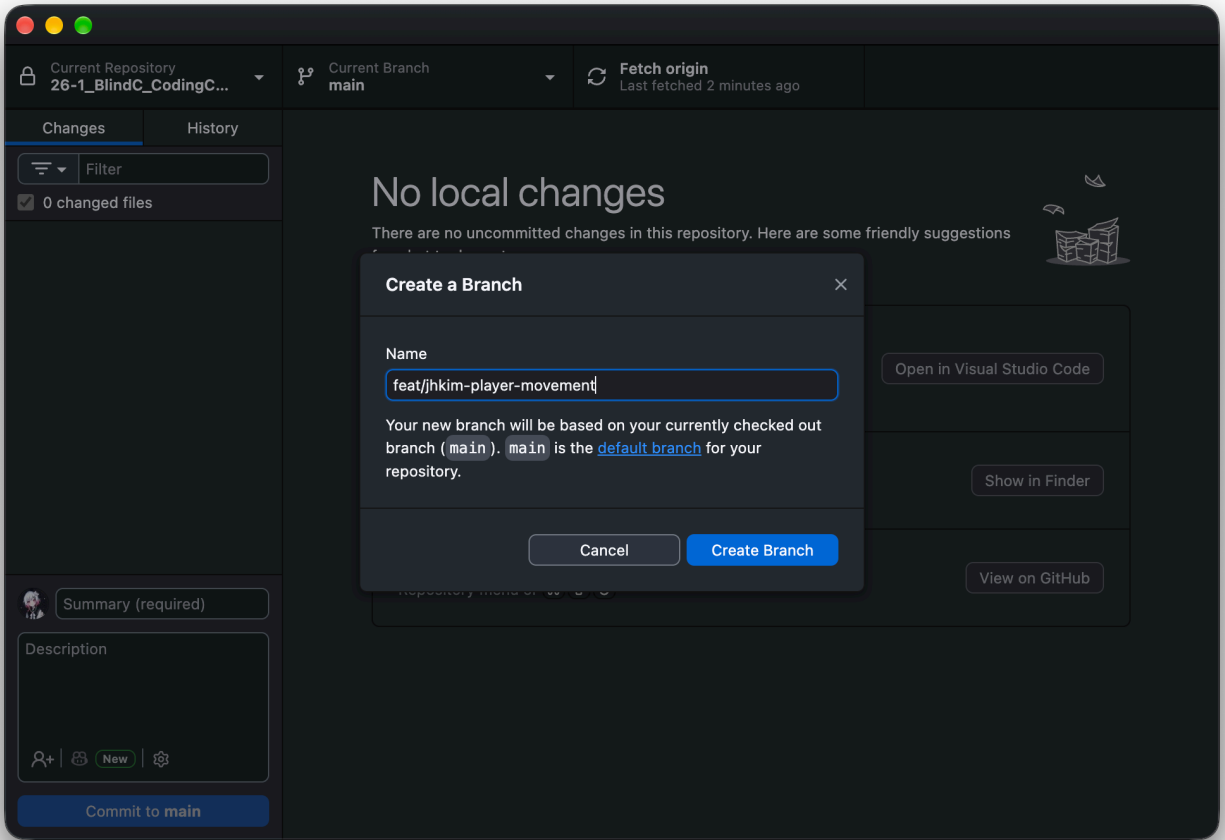
Fetch는 변경 사항 확인 / **Pull**은 실제로 반영하는 단계예요.

3. 작업 브랜치 만들기

Current Branch 버튼을 누르면 여러 Branch가 표시됩니다.



여기서 **New Branch**를 누르면 새 Branch를 만들 수 있어요.



기준 Branch가 **main**인지 확인한 뒤, **Create Branch**를 누르면 돼요. 이 때, Branch의 이름은 다음의 규칙을 따라주세요.

<종류>/<이름>-<작업 내용>

ex.

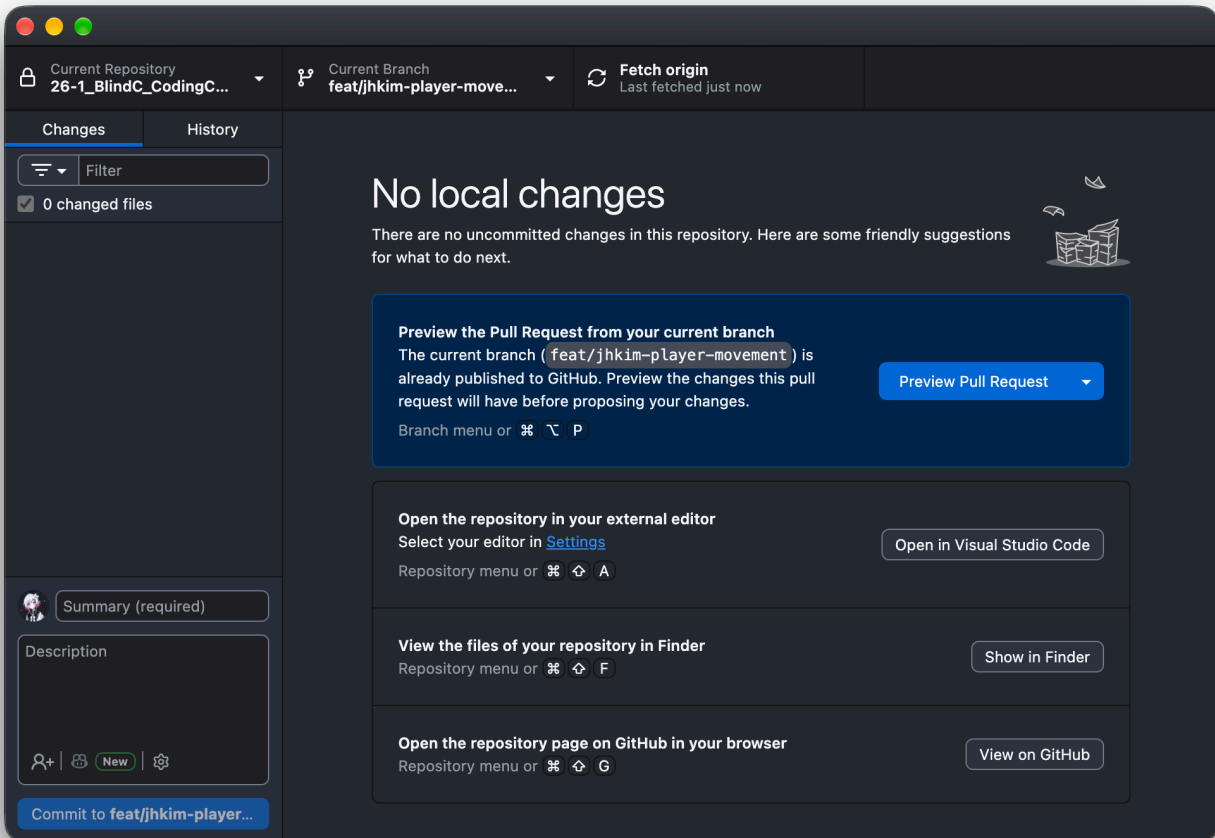
feat/jhkim-player-movement

asset/jhkim-character-modeling

종류	사용 경우
feat	새로운 기능 추가
fix	버그 수정
set	설정/구성/초기 세팅을 수정
docs	문서 추가/수정
refactor	기능 변경 없이 코드 구조만 수정
test	테스트 관련 코드나 내용 수정
perf	성능 개선
build	빌드 시스템/패키지/의존성을 수정

종류	사용 경우
revert	이전 변경을 되돌리기
asset	이미지/모델링/사운드 등을 수정

그 뒤, **Publish Branch**를 누르면 Branch 준비는 완료예요.



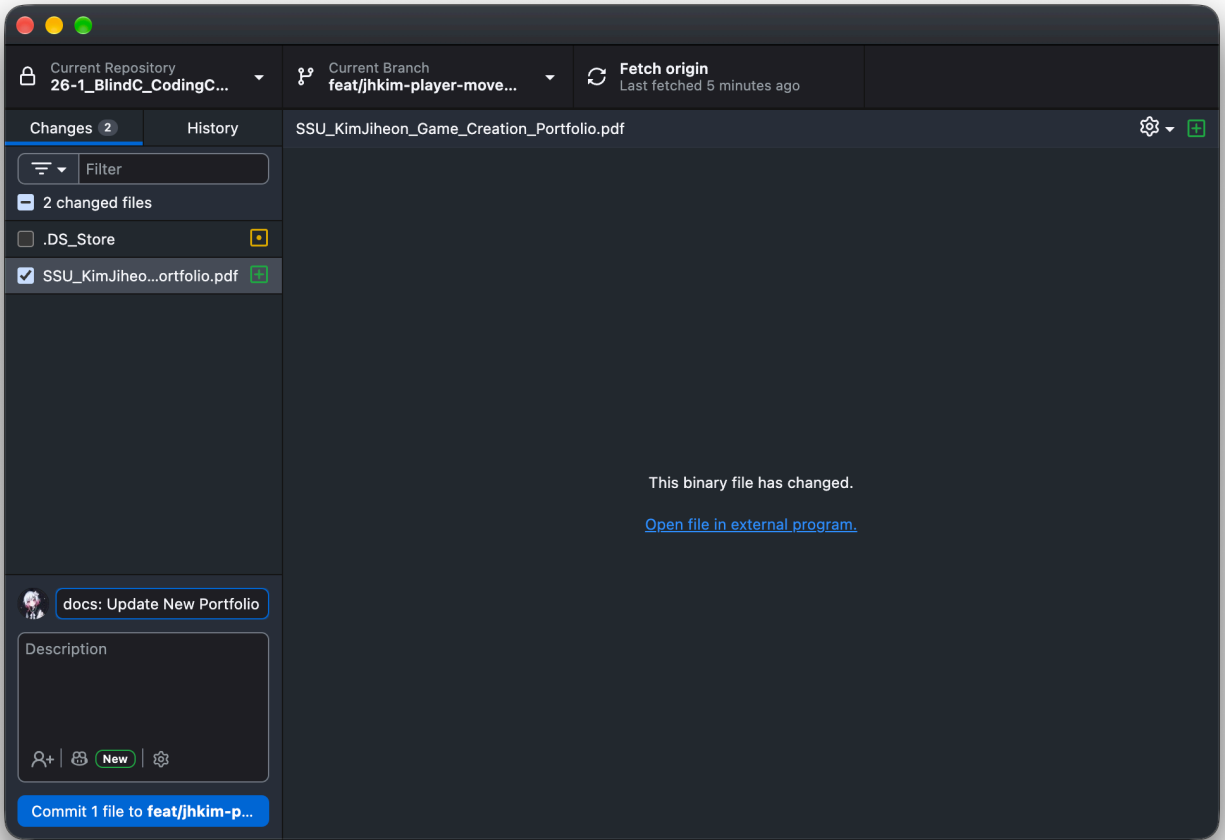
Current Branch가 새 Branch로 변경된 모습

상단에 내 작업 브랜치가 선택된 것을 확인한 뒤 작업을 진행해 주세요.

4. commit 하기

변경된 Branch에서 작업을 진행하면, Changes에 변경 사항들이 표시돼요.

이 중에서 작업과 관련된 파일만 선택한 뒤, **Summary(commit 명)**를 작성해 주세요.



위 사진에서는 두 개의 변경 사항이 감지되었어요. 이 중에서 **.DS_Store**는 필요없는 파일이기에 선택하지 않았어요.

SSU_KimJiheon....pdf 파일을 추가한 것이 메인 변경 사항이기에 **Summary**에 **commit** 규칙에 따라 작성해 줍니다.

commit 규칙은 다음과 같아요.

(commit 명에서는 공백을 사용할 수 있어요)

<종류>: <작업 내용>

ex.

feat: New Movement Code for Character

asset: Character Modeling

종류	사용 경우
feat	새로운 기능 추가
fix	버그 수정
set	설정/구성/초기 세팅을 수정

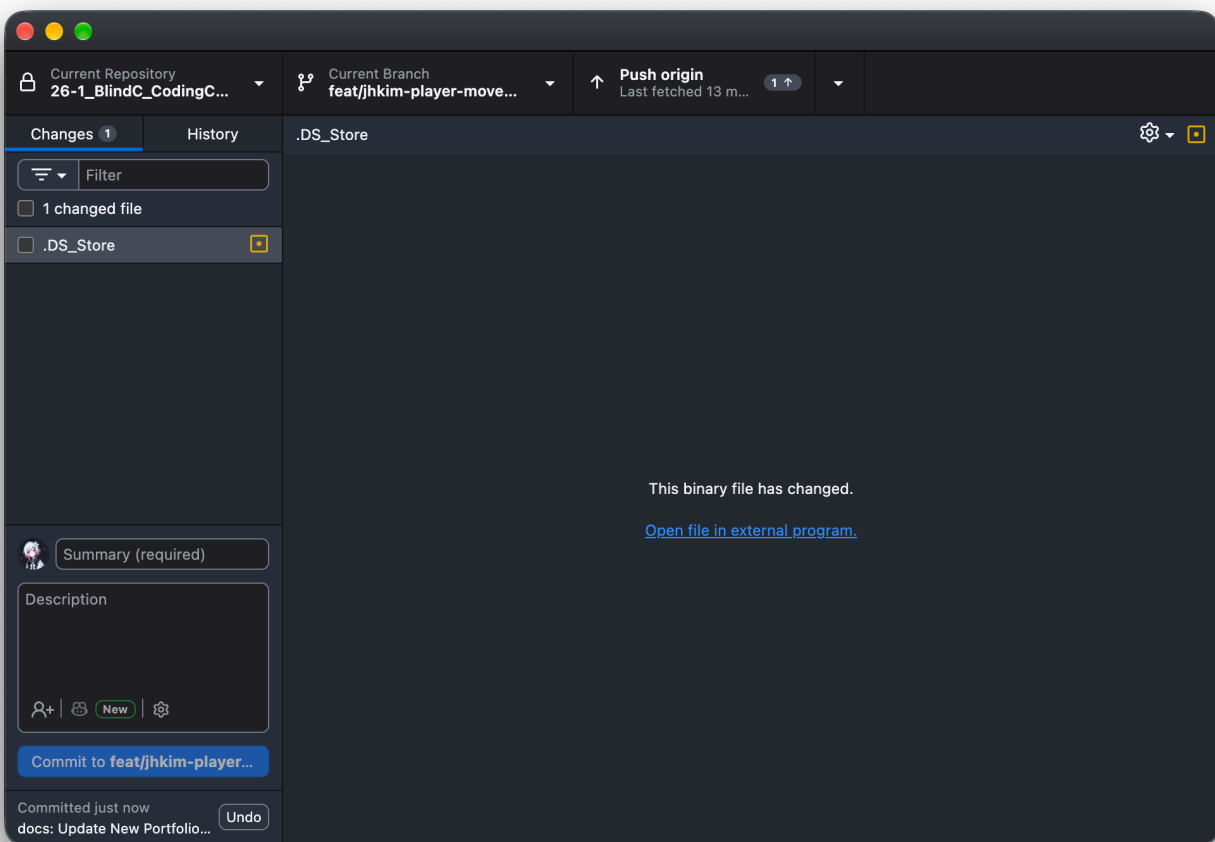
종류	사용 경우
docs	문서 추가/수정
refactor	기능 변경 없이 코드 구조만 수정
test	테스트 관련 코드나 내용 수정
perf	성능 개선
build	빌드 시스템/패키지/의존성을 수정
revert	이전 변경을 되돌리기
asset	이미지/모델링/사운드 등을 수정

기본적으로 Branch와 규칙은 비슷하지만, <이름>이 없고, '/' 대신 ':' 를 사용해요.

종류는 소문자, 영어 설명은 각 단어의 첫 글자를 대문자로 적어주세요. GitHub, UI 같은 표기는 그대로 사용해요.

모든 준비가 다 끝났으면, 하단 파랑 색의 **Commit n Files to <Branch>...** 를 눌러주세요.

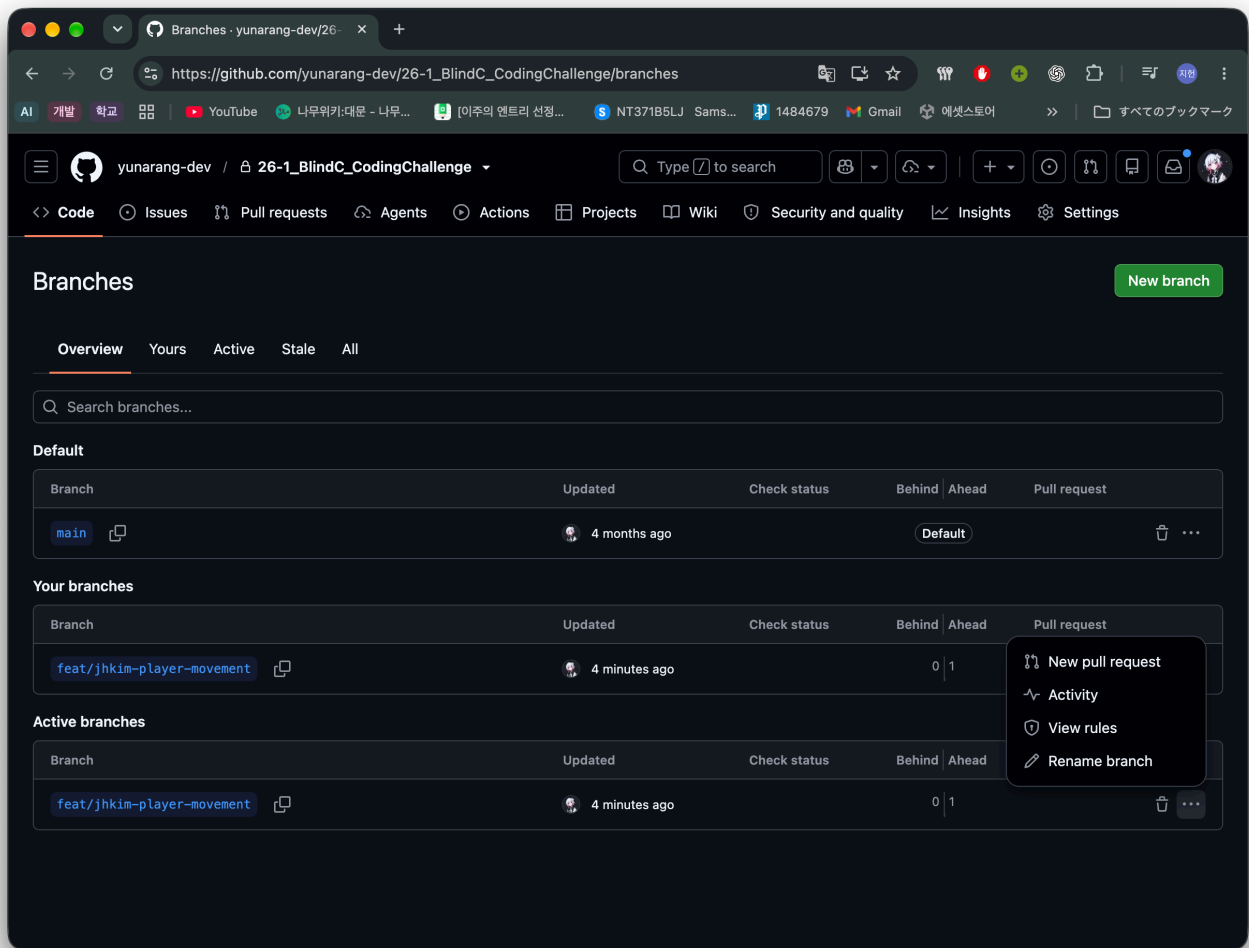
5. Push 및 PR, Merge하기



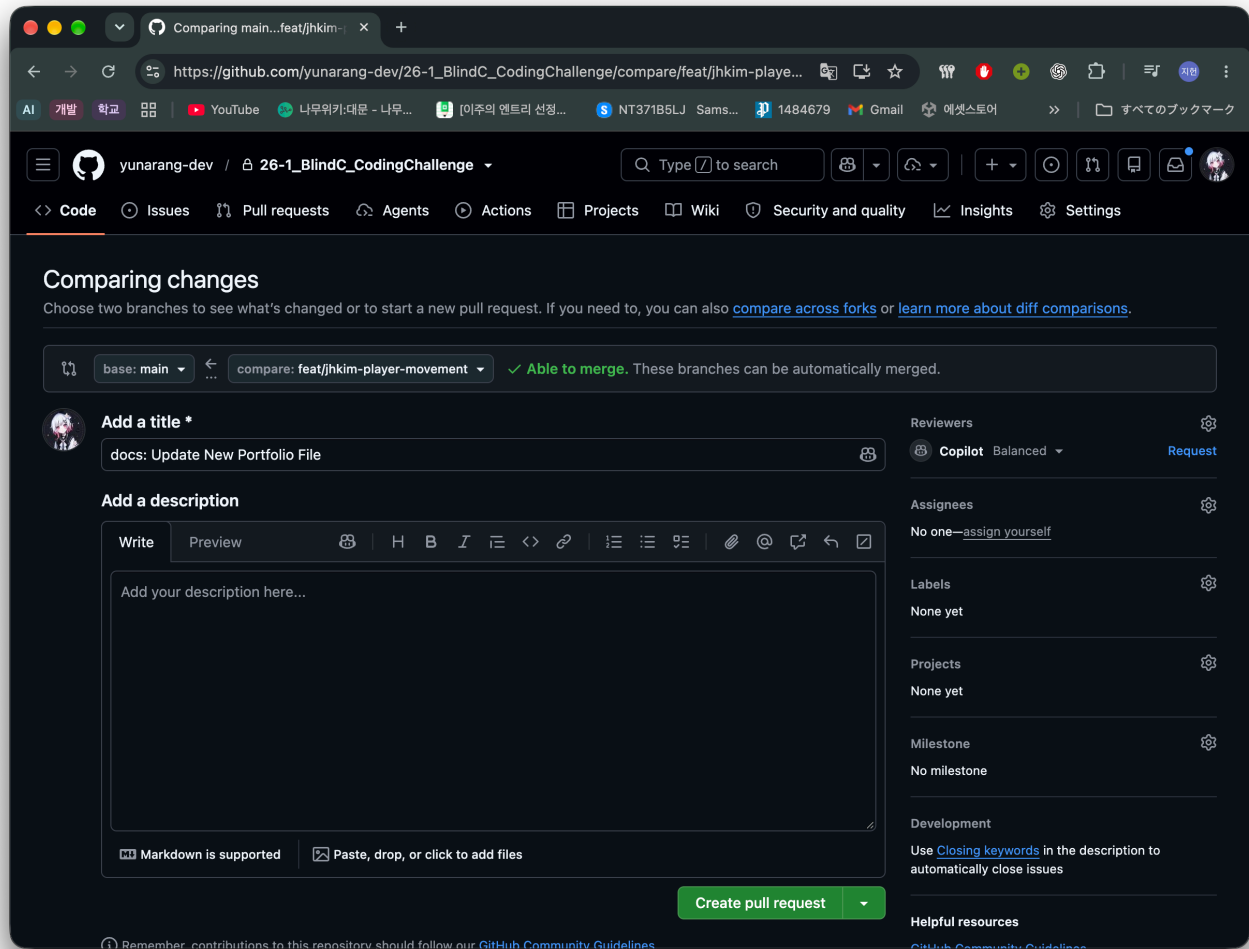
commit을 진행하면, **Push Origin** 버튼이 활성화 됩니다. **Push Origin** 버튼을 누르면 Github Branch에

반영돼요.

Branch에서 모든 작업이 끝난 뒤, Github Repository 페이지를 들어간 뒤, **Branches** 버튼을 눌러 존재하는 Branch들을 볼 수 있어요.



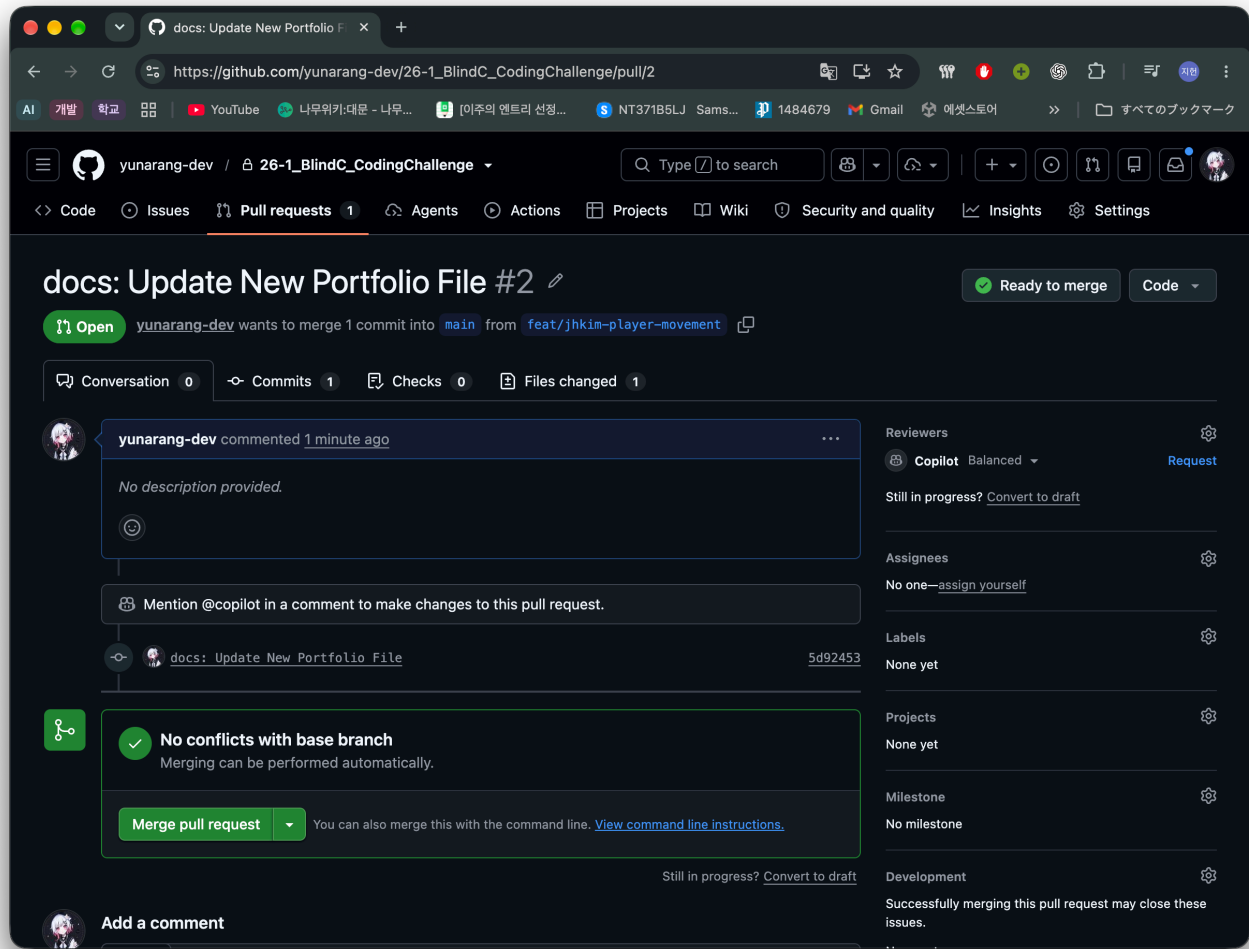
여기서 자신이 작업한 Branch에서 '...' 버튼을 눌러 **New Pull Request**를 눌러주세요.



(PR 시의 이름은 **commit** 규칙을 따라주세요.)

Able to merge가 뜨면 별 다른 조치 없이 바로 **Merge**가 가능하다는 의미예요.
(만약 Merge가 불가능한 경우, 어떤 이유로 안 되는지 확인 후 알려주세요.)

이후 **Create Pull Request** 버튼을 누르면 다음 화면으로 넘어가요.
잠시 기다리면 **Merge Pull Request** 버튼이 활성화돼요.



여러 Merge 방식이 있지만, 저희는 가장 기본인 **Merge Commit**을 사용할 예정이에요.
그냥 **Merge Pull Request** 버튼을 눌러주면 자동으로 돼요.

그 후, **Confirm Merge**를 하면 **main**에 **merge**가 돼요.

Unity 협업 가이드

사용할 Unity 버전: 6000.6.3f1 (예정)

기본 Workflow

Unity는 처음 에디터를 실행하면 Project 창에 Assets 폴더가 열려 있어요.

Unity/
Assets/
Workspace/
jhkim/
Scenes/

Scripts/
Prefabs/
Test/
.../

위와 같은 구조로 폴더에서 작업해 주세요.

1. 기본적으로 **Workspace** 폴더 내에 자신의 이름으로 폴더를 생성해 주세요.
2. 그 후 그 폴더 내에서 자신이 담당하는 기능을 구현하고, 자신이 구현한 기능을 테스트할 **Scene**을 생성해 테스트해 주세요.
3. 이동·리듬 판정·UI 같은 기능은 본인의 테스트 **Scene**에서 먼저 확인해요. 공용 **Scene**을 바로 수정하기 보다는 **Prefab**이나 **Component**로 정리해서 가져가는 방식이에요.

공용 폴더는 **Assets** 폴더 바로 하위에 둘 예정이에요.

공용 폴더로는 다른 팀원이 사용해도 될 만큼 안정화되었을 때 옮겨요. 역할이 정해지고, 테스트를 해봤고, 큰 구조 변경 가능성이 낮은 상태가 기준이에요.

아직 실험 중이면 **Workspace**에 두어도 돼요. **Workspace** 안의 작업도 **PR**로 공유할 수 있지만, **main**에 합칠 때는 프로젝트 실행에 문제가 없어야 해요.

Unity에서 이동하거나 이름을 바꾸면 **.meta**도 함께 처리돼요. **Finder**나 파일 탐색기에서 에셋만 따로 옮기는 말아주세요. 코드에 직접 적은 경로가 있다면 그것도 확인해요.
(**.meta** 파일이 옮겨지지 않을 수 있어요.)

Git LFS는 사용하지 않을 예정이에요.

Assets 폴더 내부의 **.blend**, **.psd**, **.wav**, 영상 같은 원본은 **Commit** 전에 용량과 필요 여부를 확인해요.

GitHub 일반 Git 저장소는 파일 하나가 **100 MiB**를 넘으면 **Push**를 차단해요. 50 MiB를 넘는 파일도 **Commit**하기 전에 먼저 이야기해 주세요.

큰 원본은 팀에서 정한 공유 공간에 두고 파일명·버전·다운로드 위치를 남겨요. 게임에 넣을 파일은 적절한 크기로 내보내고, 필요한 참조가 빠지지 않았는지 확인해요.

용량 때문에 **Push**가 막혔다면 파일을 지우고 다시 **Commit**하는 것으로 해결되지 않을 수 있어요. 이전 기록에 큰 파일이 남아 있을 수 있으니 함께 확인해요.