

## 2025 깃협 스터디

서강대학교 Release 학회





수고하셨습니다~!

모두 시험 보느라 고생하셨어요~!



## 지난 시간 문제

만약 f-string 오류가 났다면.. 파이썬 버전 때문입니다!  
파이썬 버전 상관없이 작동되도록 고쳐놓았습니다~  
(에러사항은 언제든지 말해주세요!!)



기억을 되살려봅시다.

git add : 커밋하기 전, 파일을 스테이징 하는 명령어  
(나 커밋할 준비가 되어있다)

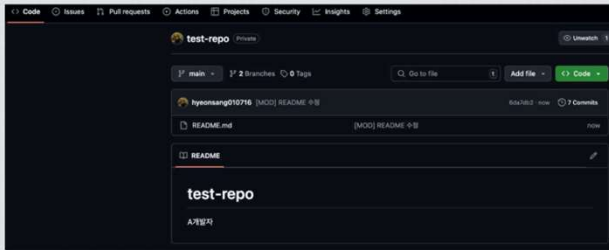
git commit : 리모트 깃헙에 파일을 보내기 전, 파일 변경 사항을 설명하는 명령어  
(스테이징한 애들은 ~~ 이렇게 고쳤어)

git push : 리모트 깃헙에 파일을 보내는 명령어  
(커밋한 애들을 리모트에 보낼게~~)

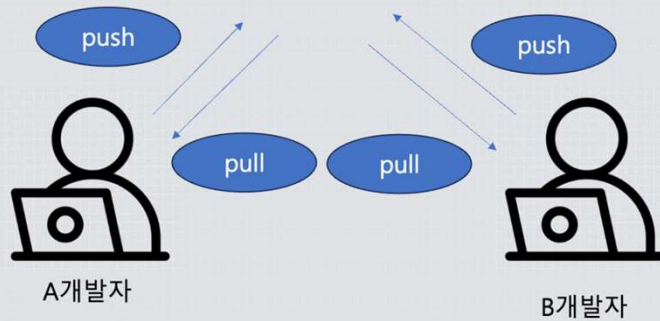


기억을 되살려봅시다.

한 브랜치 안에서 여러 명에서 작업을 할 때, 발생하는 문제.

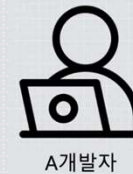
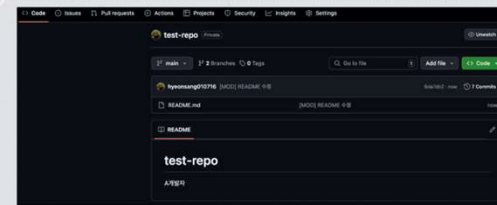


그렇다면 계속 pull, push, pull, push... 해야할까..?



충돌이 발생하지 않게 pull, push를 계속 해야 함.

충돌이 발생하는 이유는?



히스토리

[MOD] 수정1  
[MOD] README  
수정  
[FEAT] hello ~

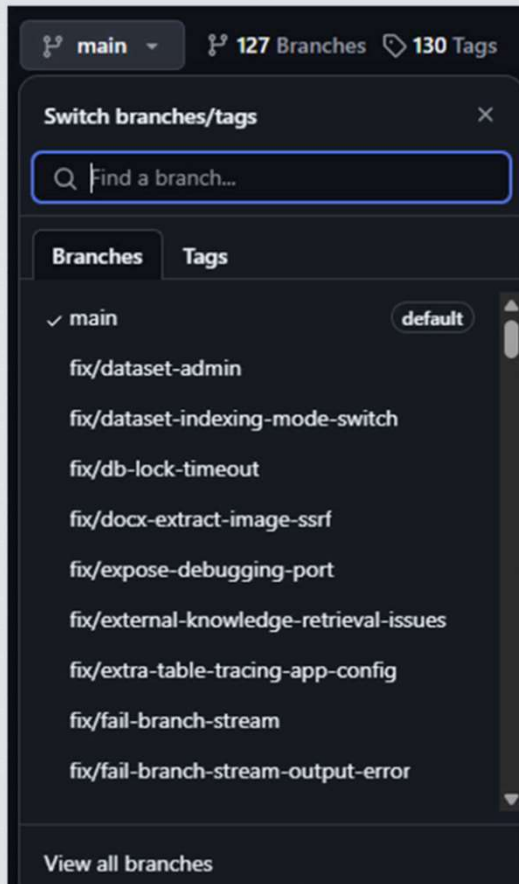
A

B

히스토리 충돌이 나서!

기억을 되살려봅시다.

그러면 무한 pull/push를 하지 않기 위해선?



브랜치의 장점: 히스토리를 자동 정리 해주기 때문!  
덕분에 무한 pull/push에서 벗어날 수 있음.

여럿이서 협업 할 때, 브랜치가 매우 유용함!



기억을 되살려봅시다.

## 브랜치 관련 명령어!

git branch : 현재 로컬에서 내 브랜치가 어디인지 알려줌. ( + 로컬에서 내가 생성한 브랜치들을 보여줌)

git branch -d 브랜치이름 : 해당 로컬 브랜치 삭제. (주의할점: 리모트 브랜치까지 삭제되는 건 아님, 주기적으로 정리해야 함)

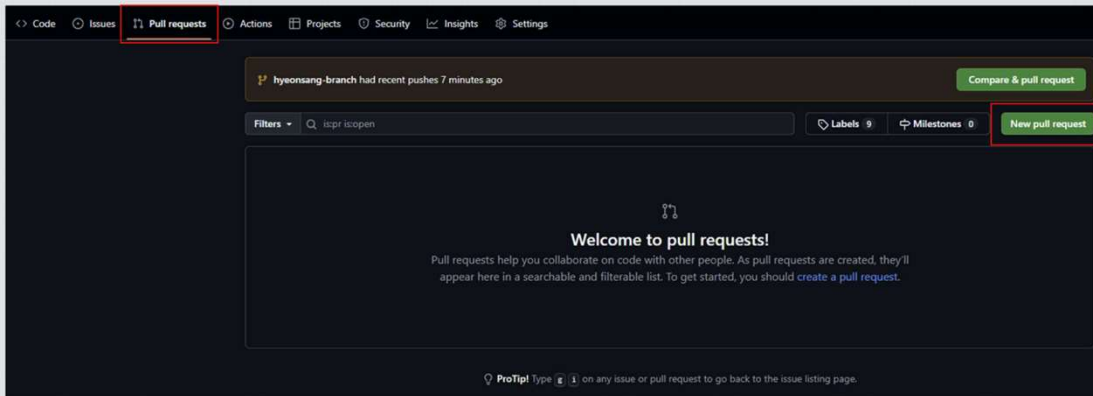
git checkout 브랜치이름 : 해당 브랜치로 이동

git checkout -b 브랜치이름: 새로운 브랜치 생성

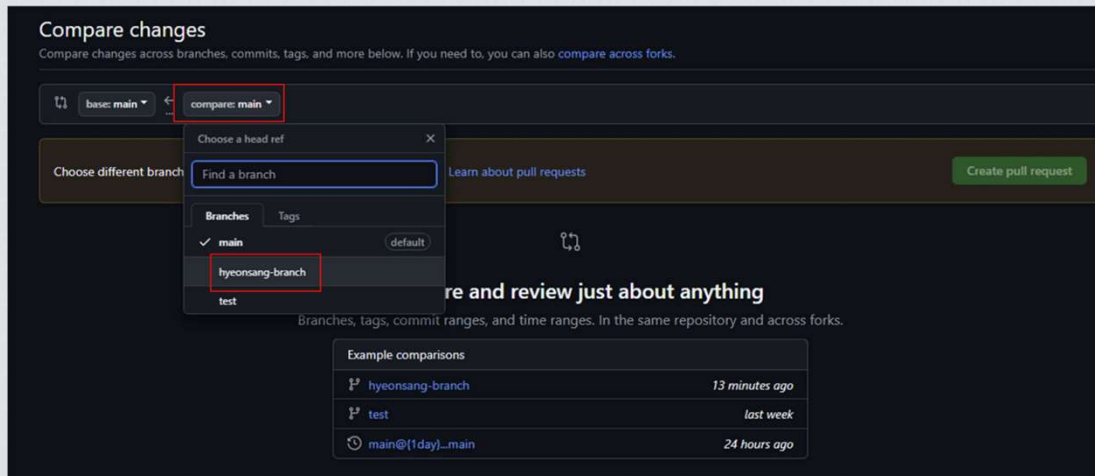


기억을 되살려봅시다.

## 브랜치로 작업을 했을 때, 풀리퀘 보내는 방법



1. Pull requests에 들어간다.
2. New pull request 버튼을 누른다.



1. 반영할 브랜치 <- 작성한 브랜치 로 세팅한다.
2. Create pull request를 연다.





기억을 되살려봅시다.

스프린트가 완료 되었을 경우(= 브랜치들이 머지 되었을 경우)

- |  |   |
|--|---|
| <code>git checkout main</code>         | - main으로 이동한다. (현재 로컬 main은 아직 최신 상태가 아님) |
| <code>git pull origin main</code>      | - 리모트에 최신 업데이트 된 히스토리를 가져온다.              |
| <code>git checkout -b 다음스프린트브랜치</code> | - 그리고 나서 다음 스프린트 브랜치를 만든다.                |

## 기억을 되살려봅시다.

항상 1인 1브랜치를 하면 좋겠지만.. 두 명이서 한 브랜치에서 작업할 경우도 생길 수 있음..  
언제 어디서 충돌이 발생할지 모름..

그래서 우리는 충돌에 관해서도 배웁니다.

우리가 배운 충돌 상황 두 가지

1. 히스토리 충돌 (같은 파일이 아닐 때)
2. 같은 파일 충돌

```
hint: You have divergent branches and need to specify how to reconcile them.  
hint: You can do so by running one of the following commands sometime before  
hint: your next pull:  
hint:  
hint:   git config pull.rebase false  # merge  
hint:   git config pull.rebase true   # rebase  
hint:   git config pull.ff only       # fast-forward only  
hint:  
hint: You can replace "git config" with "git config --global" to set a default  
hint: preference for all repositories. You can also pass --rebase, --no-rebase,  
hint: or --ff-only on the command line to override the configured default per  
hint: invocation.
```

이렇게 뜬다면?

git config 설정이 안 되었기 때문!

git config pull.rebase false

git config pull.rebase true. 택 중 1

기억을 되살려봅시다.

git config pull.rebase false

A 히스토리와 B히스토리가 충돌 났을 때, 새로운 C히스토리를 만들어서 충돌을 해결하겠다.

Commit Message	Author	Time	Hash	Label
Merge branch 'main' of <a href="https://github.com/hyeonsang010716/test-repo">https://github.com/hyeonsang010716/test-repo</a>	hyeonsang010716	committed 4 minutes ago	a2bdcd1	C
[FEAT] world 기능 구현	hyeonsang010716	committed 2 hours ago	4d986e8	B
Create hello.py	hyeonsang010716	authored 2 hours ago	c4ec838	A



기억을 되살려봅시다.

git config pull.rebase false

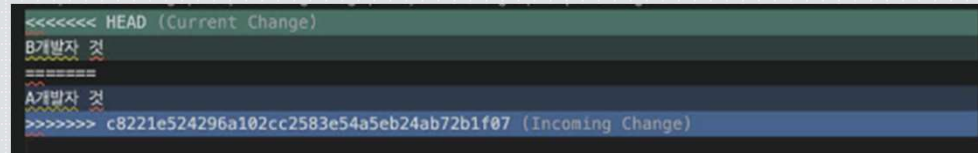
단순 히스토리 충돌일 경우 (같은 파일 충돌 X)

git pull origin main을 해주고, vi 커밋 메시지를 작성한 후, 푸시를 한다.

같은 파일 충돌이 날 경우

git pull origin main을 해주고

충돌난 코드를 고쳐준 다음



```
<<<<<< HEAD (Current Change)
B개발자 것
=====
A개발자 것
>>>>>> c8221e524296a102cc2583e54a5eb24ab72b1f07 (Incoming Change)
```

git add 충돌났던 코드 (충돌을 해결했다고 알려줌)

git commit -m "새로운 히스토리 C 작성"

git push




기억을 되살려봅시다.


git config pull.rebase true

A , B 순서로 히스토리를 정리하겠다. (새로운 히스토리 없이)


[FEAT] world 구현

 hyeonsang010716 committed 1 minute ago

B


84978de  <>

Create hello.py

 hyeonsang010716 authored 2 minutes ago

A

Verified

ac82e76  <>



## 기억을 되살려봅시다.

```
git config pull.rebase true
```

단순 히스토리 충돌일 경우 (같은 파일 충돌 X)

git pull origin main을 해주고, 푸시를 한다.

같은 파일 충돌이 날 경우

git pull origin main을 해주고

충돌난 코드를 고쳐준 다음

git add 충돌났던 코드 (충돌을 해결했다고 알려줌)

git rebase --continue (rebase 모드 해제 해주는 거 잊으면 안 됨)

git push



## 기억을 되살려봅시다.

git config pull.rebase false

단순 히스토리 충돌일 경우 (같은 파일 충돌 X)

git pull origin main을 해주고

vi 커밋 메시지를 작성한 후, 푸시를 한다.

같은 파일 충돌이 날 경우

git pull origin main을 해주고

충돌난 코드를 고쳐준 다음

git add 충돌났던 코드

git commit -m "새로운 히스토리 C 작성"

git push

git config pull.rebase true

단순 히스토리 충돌일 경우 (같은 파일 충돌 X)

git pull origin main을 해주고, 푸시를 한다.

같은 파일 충돌이 날 경우

git pull origin main을 해주고

충돌난 코드를 고쳐준 다음

git add 충돌났던 코드

git rebase --continue

git push





아직도 충돌이 조~금 남아있습니다.

브랜치 전략 알아보기!

깃 태그(tag) 란 무엇인가



## 브랜치 이동 충돌

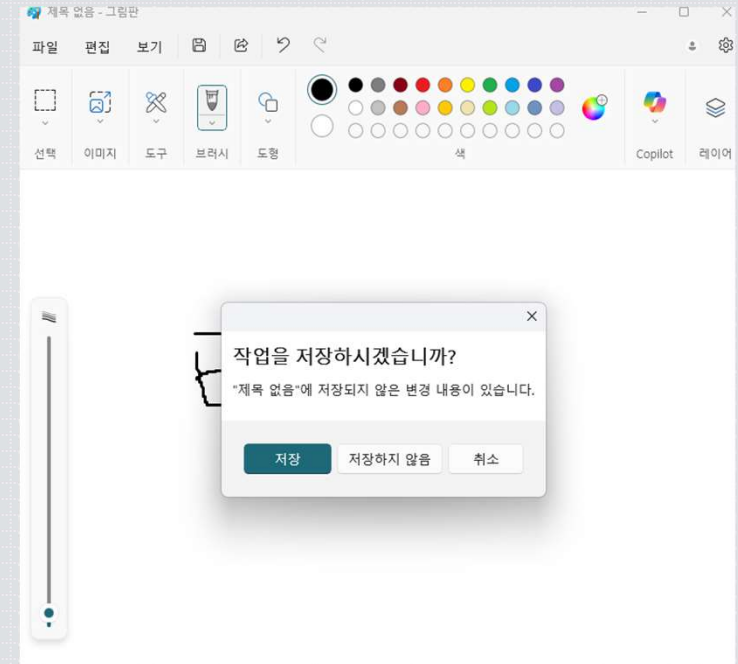
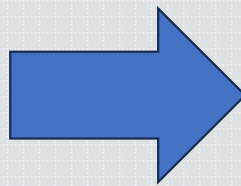
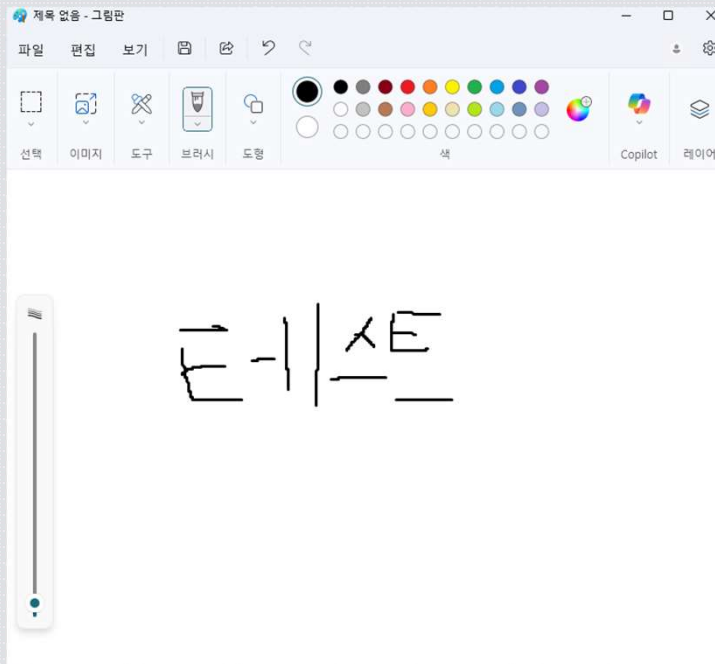
코드 수정 중, 브랜치를 이동할 수 없습니다!

깃은 그림판과 같습니다.

코드 수정 중, 브랜치를 이동할 경우

진행 중이던 코드를 저장할 수 없기 때문에 브랜치 이동을 막습니다.

```
kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/test-repo (main)
$ git checkout test
error: Your local changes to the following files would be overwritten by checkout:
    func.py
Please commit your changes or stash them before you switch branches.
Aborting
```





## git status 란

git status : 현재 브랜치 안에서 작업 중인 코드가 있는지 확인하는 명령어

작업 중(수정 중)인 파일이 없을 때

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

작업 중(수정 중)인 파일이 있을 때

```
$ git status
On branch main
Your branch is up to date with 'origin/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:   func.py

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

하지만 git checkout -b "새로운브랜치" 는 사용 가능합니다.

이유: 해당 작업 중인 상태를 새로운 브랜치로 가져가겠다는 의미이기 때문.



## git restore 이란

git restore 파일 이름 : 해당 파일의 변경 사항을 지운다는 명령어

<하수>

아.. 코드를 엄청 수정했는데, 수정하기 전 코드가 더 좋은데..?

컨트롤 + z

컨트롤 + z

컨트롤 + z

컨트롤 + z

컨트롤 + z

컨트롤 + z

<고수>

git restore 파일이름





한 번쯤 들어본 적이 있을 것이다.

"아 롤 25.7 버전 망했는데..?" -> 버전

장기적으로 프로젝트가 진행될 경우  
1년 전 코드..., 1달 전 코드..., 1주일 전... 코드까지도 기억이 안 날 수 있다.

<예시 상황>

아주 심각한 버그가 발생함..

책임 개발자: 야 주니어 개발자 1주일 전 코드로 복구 시켜놔

나: ??? 1주일 전 코드..? 그게 뭐였지





## git tag 란

git tag란 깃헙에서 버전을 관리하는 도구다!

해당 시점에 코드를 전부 세이브 해주고

버전 관리를 하는데 매우 유용하다.

리모트 깃헙에 들어가서, branches 옆 tags를 눌러보면 옆 사진처럼 볼 수 있음!

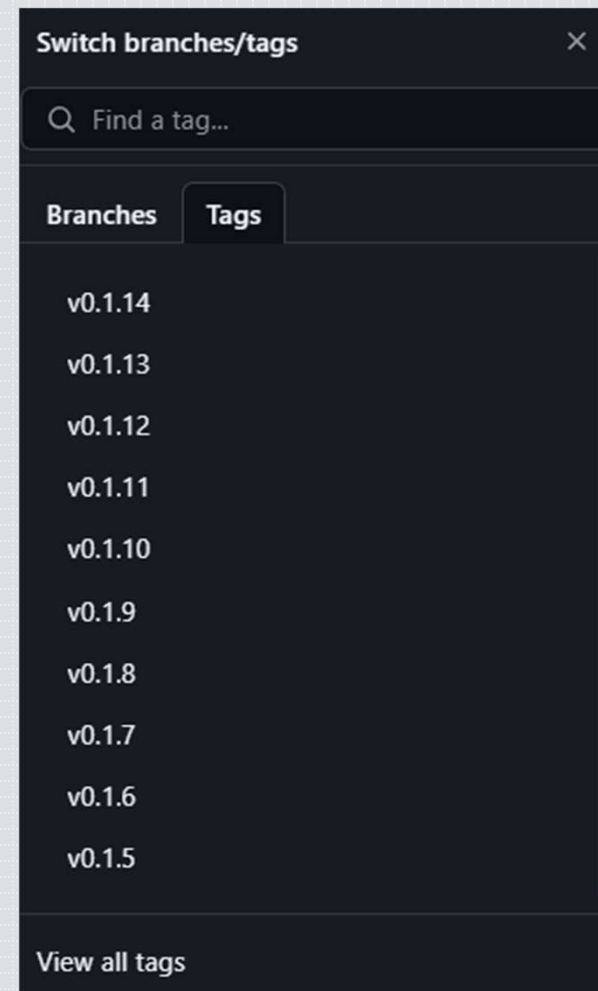
git tag를 사용한다면

<예시 상황>

아주 심각한 버그가 발생함..

책임 개발자: 야 주니어 개발자 1주일 전 코드로 복구 시켜놔

나: 네~ git tag 쓱~



## git tag 란

### git tag 사용하는 방법

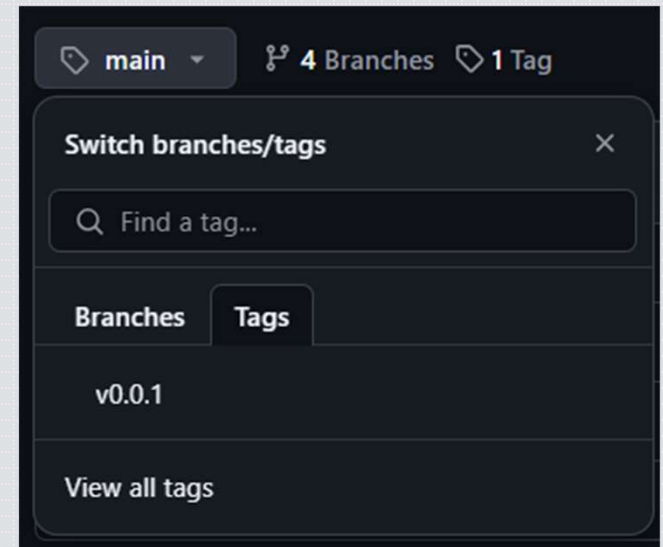
git tag 태그이름 : 로컬에서 태그 이름을 만들어 놓는다.

git push origin tag 태그이름 : 만들어 놔던 태그 이름으로 등록한다.  
(현재 코드 시점이 태그 기준이 됨.)

```
kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/test-repo (main)
• $ git restore func.py

kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/test-repo (main)
• $ git tag v0.0.1

kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/test-repo (main)
• $ git push origin v0.0.1
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/hyeonsang010716/test-repo.git
* [new tag]          v0.0.1 -> v0.0.1
```



보통 tag는 목표했던 기능까지 스프린트가 끝나면 tag를 달아서 관리를 해줘요~

롤 같은 경우, 실서버와 pbe 테스트 서버가 나눠져 있는 것을 볼 수 있다.

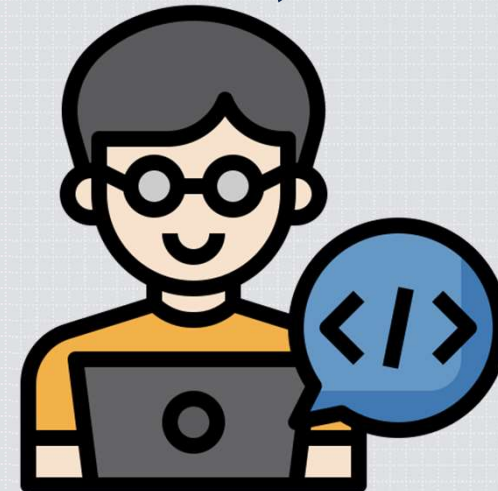
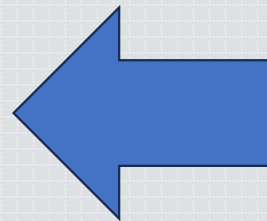
나눠진 이유: 테스트 기능을 반영했을 때, 버그와 민심?을 보기 위해서

만약 pbe 테스트 서버가 없었다면..?



플레이어

아 롤 망했네.  
발로란트 ㄱ



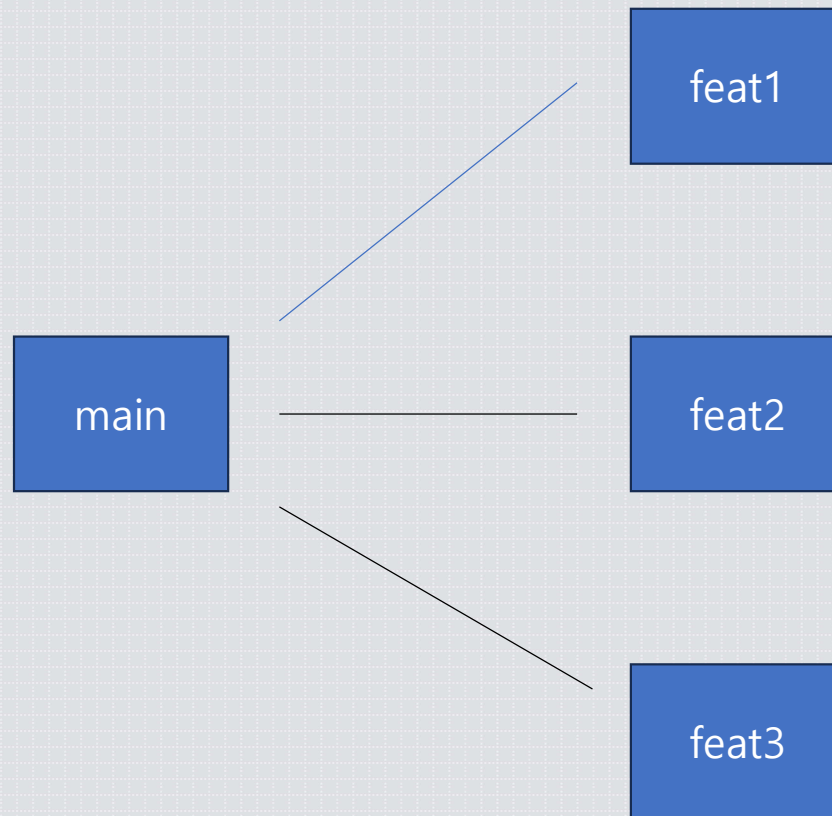
개발자

ㅎㅎ 개발 끝났다.  
Push 해야지~  
(BF소드 130원)



실수로 발생한 버그가 실서버에 반영되면 매우 큰일..!!!

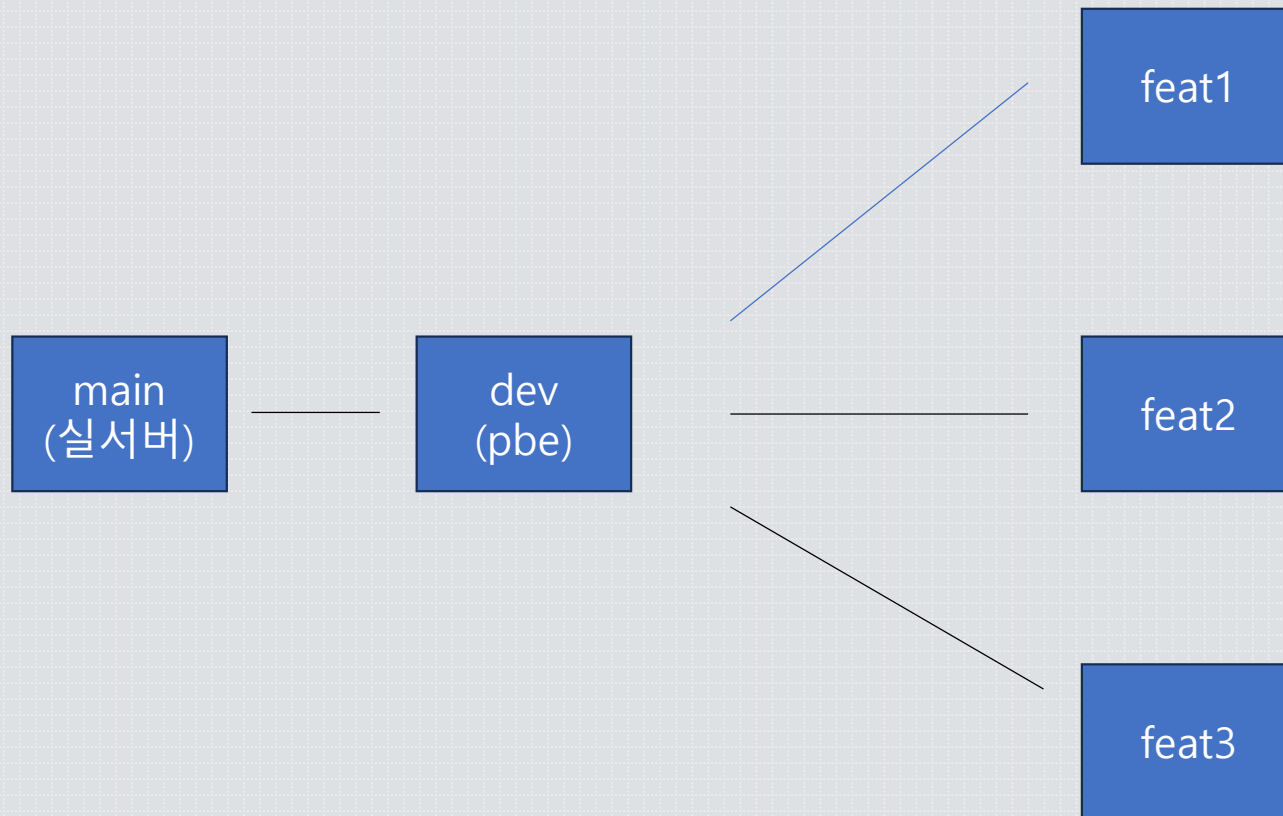
그래서 테스트 서버(깃 브랜치 전략)이 꼭 필요합니다.



현재 우리의 깃 브랜치 모습



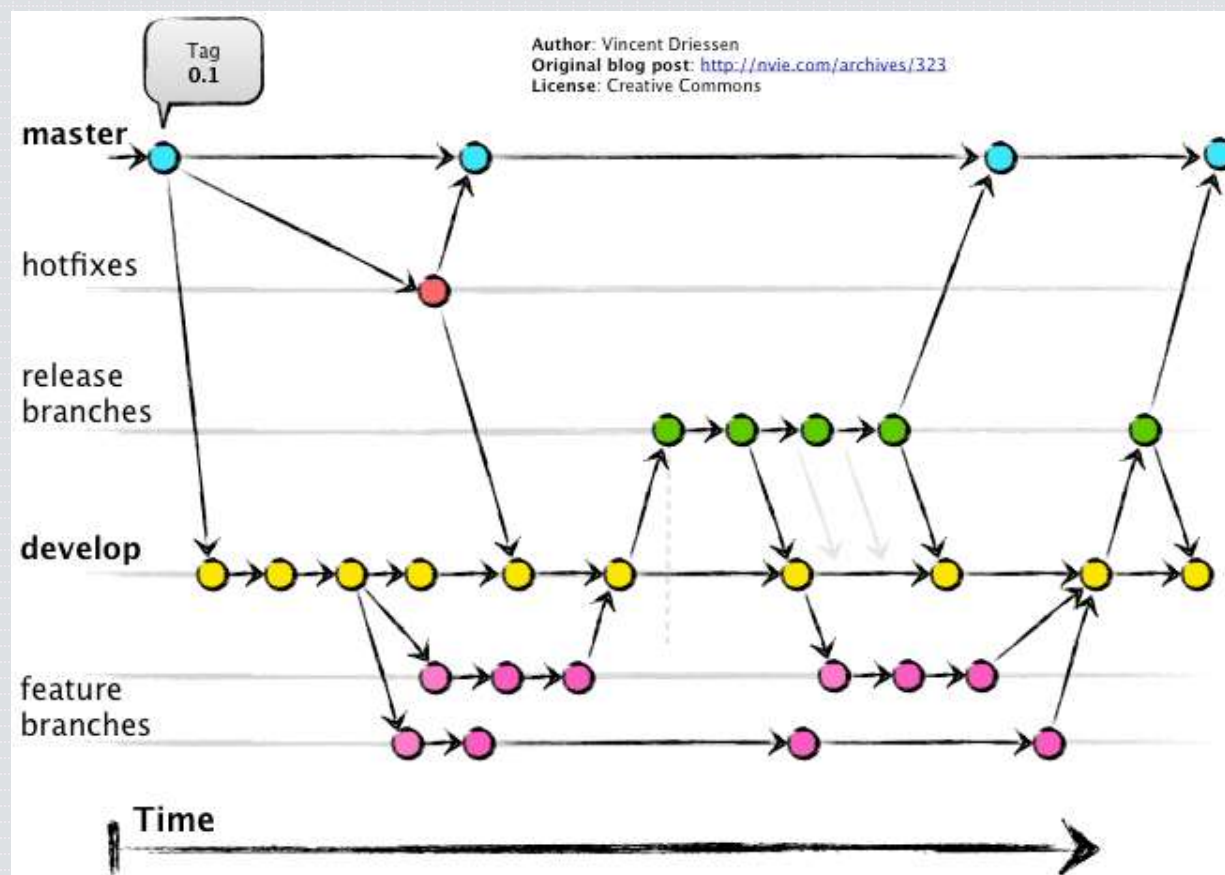
실서버에 반영하기 전, dev 서버에 먼저 적용한 후  
테스트 검증이 끝나면 실서버에 반영 되어야 한다.



이상적인 깃 브랜치 전략



(중요한 점은 절대 main에 푸쉬하면 안 됨..)





## 4주차 실습

1. 간단한 알고리즘을 가지고 깃 협업 해보기!
2. 주의사항) 커밋 컨벤션, 브랜치 네이밍 컨벤션, 풀리퀘 컨벤션 맞추기 (실습 목적)

- FEAT: 새로운 기능 추가
- FIX: 버그 수정
- DOCS: 문서 수정
- STYLE: 스타일 관련 기능(코드 포매팅, 세미콜론 누락, 코드 자체의 변경이 없는 경우)
- REFACTOR: 코드 리팩토링
- TEST: 테스트 코드 추가
- CHORE: 빌드 업무 수정, 패키지 매니저 수정(ex .gitignore 수정 같은 경우)

커밋 컨벤션: [FEAT] , [MOD] , [FIX] ~

브랜치 컨벤션: 깃헙 계정 아이디/백준 문제 유형-백준 문제 번호  
(만약 hyeonsang010716이 math 유형 백준 1000번을 풀었을 경우 → hyeonsang010716/math-1000)

풀리퀘 컨벤션: [백준 문제 유형]문제 번호-최초 풀리퀘를 보낸 날짜  
(만약 2025.3.16일날 math 1000번을 풀어서 풀리퀘를 보낼 경우 → [math]1000-2025.3.16)

그리고 풀리퀘 상세 설명에 성공한 화면 사진 붙여넣기

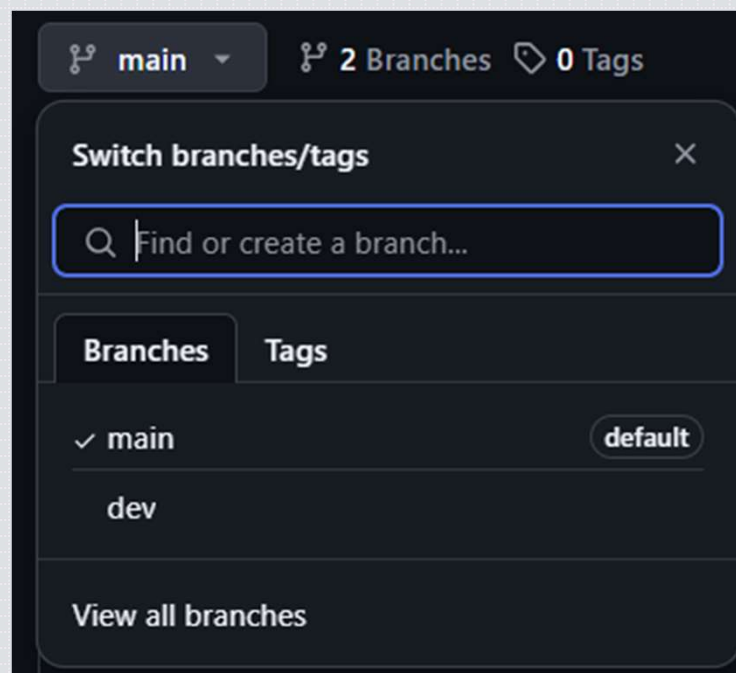
(+ dev 브랜치에 풀리퀘를 보내기!)





## 4주차 실습

dev 브랜치를 만들어봤어요!



먼저 git checkout dev 로 브랜치를 이동해줍니다. (만약 dev가 없다고 뜨면, git remote update를 해주세요!)

그 다음, git pull origin dev 로 리모트에 있는 dev 최신 내용을 가져옵니다.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/git-study (main)
• $ git checkout dev
Switched to branch 'dev'
Your branch is up to date with 'origin/dev'.

kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/git-study (dev)
• $ git pull origin dev
From https://github.com/SGCS-Release-Git-Project/Git-Study
 * branch          dev          -> FETCH_HEAD
Already up to date.
```

## 4주차 실습

브랜치 네이밍 컨벤션에 맞춰서, 브랜치를 만들어줍니다.

```
kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/git-study (dev)
$ git checkout -b hyeonsang010716/math-8393
Switched to a new branch 'hyeonsang010716/math-8393'

kyr76@WIN-659R041E9C0 MINGW64 ~/OneDrive/바탕 화면/git-study (hyeonsang010716/math-8393)
$
```

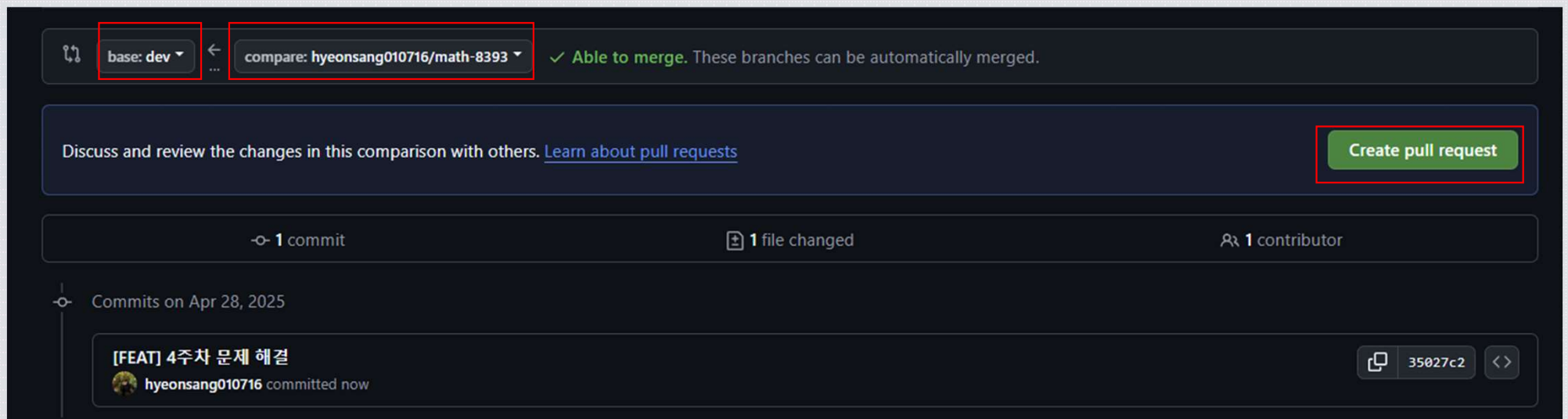
이번 주 문제를 풀어줍니다.

```
main.py M X Home.py
1 def solve(inputs):
2     answers = []
3
4     for input in inputs:
5         ans = 0
6         for i in range(1, int(input)+1):
7             ans += i
8         answers.append(str(ans))
9
10    return answers
```



## 4주차 실습

문제를 해결 및 푸쉬 후, 풀리퀘를 다음과 같이 설정해줍니다.



## 4주차 실습

폴리퀘 네이밍 컨벤션 룰도 잊지 말 것!

