

git -

목차

>etc	2
>branch	3
>3/26일	10
>4/17일	11
✓ git revert란?	12
🔄 예를 들어 설명해볼게요	12
커밋 내역:	12
✓ 질문에 대한 답	13
▶ 여러 커밋을 한 번에 revert 하는 방법:	13
✓ revert vs reset 차이	13
요약	14
완전 콘솔로만 하는 Git 협업 Test (혼자 실습 가능)	15
🌱 1. 폴더 준비 & Git 초기화	15
👤 2. main 브랜치에서 초기 파일 만들기	15
👤 3. 협업자 A가 작업 (브랜치 생성 → 작업 → 병합)	15
👤 4. 협업자 B가 작업	16
📦 결과 확인	16
📌 요약 명령어	17
✓ GitHub Collaborator 방식 협업 전체 예제	18
👤 전제 조건	18
📌 [1단계] 저장소 주인(userA)이 초대하기	18
👤 [2단계] 협업자(userB)의 작업 흐름	18
📦 1. 저장소 클론	18
🌱 2. 브랜치 생성 및 전환	19
🔧 3. 작업 및 커밋	19
🚀 4. 브랜치 푸시	19
🔄 [3단계] PR(Pull Request) 생성	19
✓ 전체 흐름 요약	20
🔍 "Compare & pull request" 뜻	21
🖋 예시 흐름	21
✓ 언제 쓰나?	21

나의 말:	22
ChatGPT의 말:	22
✓ 전체 흐름 요약	22
📝 요약	23
👁️ 참고로...	23
✓ Fork + Pull Request 방식 (간접 push) 쉽게 이해하기	24
📌 전체 흐름 요약	24
🔧 단계별 상세 설명	24
① Fork (포크) - 저장소 복사하기	24
② Clone (클론) - 내 컴퓨터로 가져오기	24
③ 작업 후 Push (푸시) - 변경사항 업로드	25
④ Pull Request (풀 리퀘스트) - 병합 요청하기	25
⑤ 관리자 검토 및 병합	25
💡 팁	26
🔒 GitHub에서 원본 저장소의 쓰기 권한 제한 방법	27
1. 개인 저장소(Personal Repository)인 경우	27
2. Organization 소속 저장소인 경우 (더 세밀한 권한 제어 가능)	27
3. 모든 경우에 적용 가능한 추가 보안 설정	27
4. Fork된 저장소의 동기화 방지 (Optional)	28
💡 결론: 원본 저장소를 철저히 보호하는 방법	28
🔍 문제 원인	29
✓ 해결 방법: Fork 저장소를 통해 PR 보내기	29
1. 현재 원격 저장소 확인	29
2. Fork 저장소를 원격 저장소로 설정	30
3. 변경 사항을 내 Fork 저장소에 푸시	30
4. GitHub에서 PR 생성	31

>etc

Git 커밋 및 변경 이력 확인 명령어 정리

1. git log

커밋 이력을 확인할 수 있는 기본 명령어

사용법:

bash

복사편집

git log

-
- 확인 가능한 정보:
 - 커밋 ID
 - 작성자
 - 작성 날짜
 - 커밋 메시지
- 화면 종료: q 키를 누르면 빠져나올 수 있음

2. git log -p

각 커밋에서 어떤 변경이 있었는지(diff) 함께 확인

사용법:

bash

복사편집

```
git log -p
```

- - 각 커밋에 대해 변경된 파일 내용(diff 결과)을 함께 보여줌
-

3. git diff

두 상태(예: 워킹 디렉터리 vs 스테이지, 이전 커밋 vs 현재 등) 사이의 차이점 확인

기본 사용법:

bash

복사편집

```
git diff
```

- → 워킹 디렉터리와 스테이징 영역의 차이
 - 예시:
 - `git diff HEAD`
→ 최신 커밋과 현재 작업 디렉토리의 차이
 - `git diff 커밋1 커밋2`
→ 두 커밋 간의 차이
-

요약

명령어

설명

`git log` 커밋 이력 확인

`git log` 커밋별 코드 변경 내용(`diff`) 확인
`-p`

`git diff` 두 상태(커밋, 파일, 브랜치 등) 간
차이 확인

`q` 로그 화면에서 빠져나올 때 사용

git reset 원 하는 커밋으로 돌아갈수 있다. 하지만 원격저장소 배포후에 많은 문제가 생길수 있다. 원격저장소에 배포되지 않은 커밋상황에서 만 되도록 사용하자.

돌아가고자 하는 커밋을 **git log**로 찾는다.

```
git reset 441c339cdds8b3e72701eb36fc5f515d2c3c86bc --hard
```

상위와 같이 입력 한다.

-hard 옵션도 기존 정보들이 기록에서 사라져 많은 문제가 발생할 수 있다.

```
git reset 441c339cdds8b3e72701eb36fc5f515d2c3c86bc --hard
```

로 설정한 커밋은 남고 이전 커밋은 사라진다.

>branch

커밋이 실행되어야 브랜치를 시작할 수 있다.
새로 하나 파일을 만들고 다음 처럼 커밋한다.

```
git commit -m "첫 커밋"
```

branch이름 변경하는 방법은 다음과 같다.

```
git branch -m master main
```

Git 브랜치와 병합(Merge), 충돌(Conflict) 개념은 처음엔 다소 어렵지만, 예제와 함께 하나씩 이해하면 충분히 익힐 수 있습니다. 아래에서 차근차근 설명하겠습니다.

✓ Git 브랜치 이해하기

1. 브랜치 생성 및 확인

```
git branch          # 현재 존재하는 브랜치 목록 확인
```

```
git branch exp      # 'exp'라는 새로운 브랜치 생성
```

```
git branch
```

git log를 하면 main 브랜치와 동일하다는것을 확인할 수 있다.

```
git log -- branches -- decorate -- graph -- online
```

브랜치의 모든 로그를 보여준다.

- * main : 현재 위치한 브랜치를 표시
- exp : 새로 만든 브랜치 (아직 이동하지 않았음)

2. 브랜치 이동

```
git checkout exp
```

- 이제부터는 exp 브랜치에서 작업하게 됩니다.

✅ 브랜치별 작업 내용 구분

예를 들어 **exp**에서 새로운 파일을 만듭니다:

```
echo "파일 내용" > exp.txt
```

```
git add exp.txt
```

```
git commit -m "exp 브랜치에서 파일 추가"
```

그 후, **main** 브랜치로 이동해 봅니다:

```
git checkout main
```

- **exp.txt** 파일은 보이지 않음 → 브랜치별 작업 상태가 분리되어 있기 때문입니다.
- 다시 **exp**로 이동하면 해당 파일이 다시 보입니다.

```
git log -- branches -- decorate -- graph -- online
```

 브랜치의 모든 로그를 보여준다.

```
git diff main..exp
```

 두 브랜치 사이의 차이를 확인할 수 있다.

✓ 브랜치 병합 (merge)

1. exp → main 병합

```
git checkout main      # main 브랜치로 이동
```

```
git merge exp          # exp 브랜치 내용을 병합
```

```
# 강제 커밋 하고 싶다면
```

```
git merge --no-ff exp -m "exp 브랜치 병합"
```

- 병합되면 **exp**의 변경사항이 **main**에 반영됩니다.
- **wq!**는 **Vim**에서 메시지가 뜰 경우 빠져나오는 방법입니다 (커밋 메시지 입력 시).

2. main → exp 병합

반대로 **main**에서 작업한 것을 **exp**에도 반영하고 싶다면:

```
bash
```

```
복사편집
```

```
git checkout exp
```

```
git merge main
```

이제 `main`과 `exp` 브랜치가 같은 내용을 가지게 됩니다.

`add commit`없이 브랜치를 왔다갔다 하면 관리되지 않는 파일이 되어서 복잡하게 된다.

✅ 브랜치 상태 확인

```
git log --branches --decorate --graph --oneline
```

- 브랜치들의 커밋 히스토리를 그래픽 형태로 한눈에 보여줍니다.
-

✅ 브랜치 삭제

`exp` 브랜치가 더 이상 필요 없다면:

```
bash
```

복사편집

```
git branch -d exp
```

- 병합 후 브랜치를 정리할 때 사용합니다.
-

✅ 브랜치 간 차이 확인

bash

복사편집

```
git diff main..exp
```

- 두 브랜치의 차이점을 확인할 수 있습니다.

⚠ 충돌 (Conflict) 예제

1. 충돌 상황 만들기

main에서 파일 작성

bash

복사편집

```
git checkout main
```

```
echo "Hello from main" > test.txt
```

```
git add test.txt
```

```
git commit -m "main에서 test.txt 작성"
```

exp에서 같은 파일 수정

```
git checkout exp
```

```
echo "Hello from exp" > test.txt
```

```
git add test.txt
```

```
git commit -m "exp에서 test.txt 수정"
```

2. 충돌 발생

```
git checkout main
```

```
git merge exp
```

- 같은 파일의 같은 부분을 수정했기 때문에 충돌 발생!

3. 충돌 해결 방법

- `test.txt` 열어 보면 이런 식으로 표시됩니다:

text

복사편집

```
<<<<<< HEAD
```

```
Hello from main
```

```
=====
```

```
Hello from exp
```

```
>>>>>> exp
```

- 원하는 대로 내용을 수정하고, 아래 명령어로 충돌 해결을 마칩니다:

```
git add test.txt
```

```
git commit -m "충돌 해결"
```

작업

명령어

브랜치
생성

`git branch exp`

브랜치
이동

`git checkout exp`

브랜치
병합

`git merge exp` (main에서)

차이 비교

`git diff main..exp`

충돌 해결

파일 수정 후 `git add`, `git commit`

브랜치
삭제

`git branch -d exp`

필요하다면 위 내용을 다이어그램 형태로 그려서 보여드릴 수도 있습니다.
원하시나요?

>3/26일

<https://www.toptal.com/developers/gitignore/>

```
git init
git add .
git commit -m "2"
git push origin main --force
git remote add origin https://github.com/grapecfoxman/start01.git
git push origin main --force
git branch -m master main #이름 변경
git push origin main --force
git branch exp
git push origin main
git push origin exp

git push origin exp
```

>4/17일

```
git reset --hard 6bdf881e760950359dab6f10476b162f0850f489
```

--hard 파일들도 해당 상태로 복귀 시킨다.

reset은 관련 로그를 지운다. push후사용시 문제가되니 조심해서 사용하자.

```
git revert 6bdf881e760950359dab6f10476b162f0850f489
```

해당 커밋을 복귀시킨다. 중간에 있는 커밋은 살린다.

하나씩 삭제 해서 모든 작업을 복귀 시킬수 있다.

```
git log 확인
```


✓ git revert란?

`git revert <커밋 해시>`는 **해당 커밋을 "취소하는 새로운 커밋을 생성"합니다.

- 기존 커밋은 남아 있고, 되돌린 변경 사항만 반영한 새 커밋이 생성됩니다.
 - 그래서 **중간 커밋을 살리면서도 복귀(취소)**할 수 있어요.
-

🔄 예를 들어 설명해볼게요

커밋 내역:

mathematica

복사편집

A - B - C - D - E - F (HEAD)

- 지금 HEAD는 F입니다.
- 그런데 C에서 했던 변경사항을 되돌리고 싶다면?

bash

복사편집

`git revert <C 커밋 해시>`

- 그러면 이런 식으로 됩니다:

mathematica

복사편집

A - B - C - D - E - F - G (G는 "C를 되돌리는 커밋")

- 즉, C는 삭제되지 않고, "C의 반대 변경 내용"이 G로 들어감.

✓ 질문에 대한 답

특정 부분으로 복귀하고 싶다면 커밋을 일일이 하나씩 이동해야 하는거야?

✗ 꼭 하나씩 하지 않아도 됩니다.

▶ 여러 커밋을 한 번에 **revert** 하는 방법:

bash

복사편집

git revert A^..F

- 위 명령은 A 이후부터 F까지의 커밋을 순차적으로 모두 **revert**합니다.
- A^..F는 A 다음부터 F까지의 범위를 의미합니다.

✓ revert vs reset 차이

명령어	설명	안전성	협업
revert	기존 기록은 살리고 반대 커밋을 새로 만듦	안전	✓
reset	커밋 자체를 삭제하거나 이동	위험	✗ 공동작업 브랜치에서는 주의

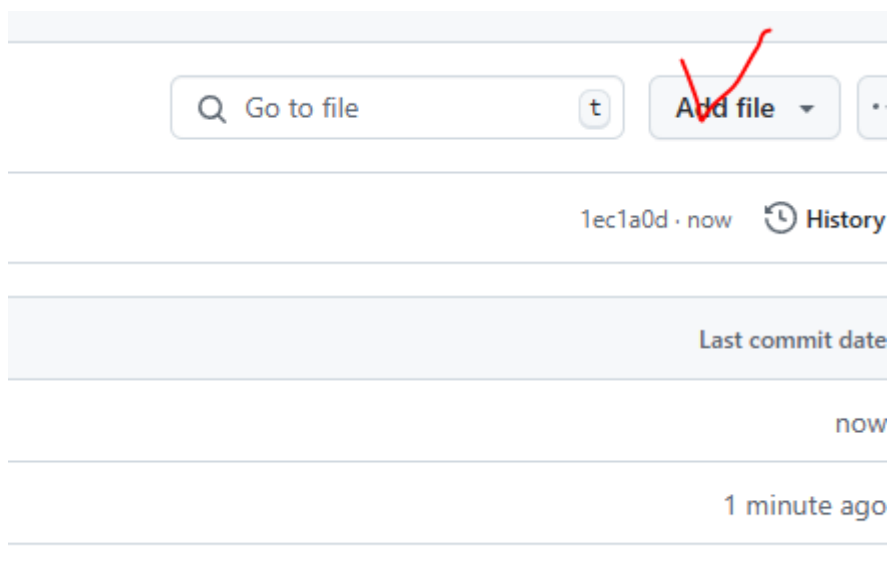
요약

- `git revert`는 원하는 커밋 하나만 골라서 되돌릴 수 있다.
- 커밋들을 하나씩도, 여러 개도 되돌릴 수 있다.
- 중간 커밋을 살리면서 복귀할 수 있는 안전한 방법이다.

필요하다면 그래픽으로 시각적으로 설명해줄 수도 있어요. 원하시나요?

> Collaborator

1. first user로 github에 저장소를 만든다. foxman12@naver.com
2. [colaborator01](#) 저장소 이름 파일이름 1.txt 추가 및 커밋



[1단계] 저장소 주인(**userA**)이 초대하기

1. GitHub에서 **userA/cool-project**로 이동
2. **Settings** → **Collaborators** 메뉴 클릭
3. **userB**의 GitHub 계정을 검색 → 초대 (invite)
4. **userB**는 이메일이나 알림으로 초대 수락

Issues Pull requests Actions Projects Wiki Security Insights **Settings**

an has been added as a collaborator on the repository.

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Models

Webhooks

Environments

Codespaces

Pages

Security

Collaborators and teams

Public repository
This repository is public and visible to anyone [Manage visibility](#)




Direct access
1 user has access to this repository.
[0 collaborators](#), [1 invitation](#).

Manage access

[Add people](#)

☐ Select all Type ▾

Find a collaborator...


<input type="checkbox"/>	 soo min park Awaiting grapefoxman's response	Pending Invite 	
--------------------------	--	--	---



추가해서 초대한사람의 이메일로 다음과 같은 허락 이메일이 전송된다.
추가한사람 foxman12@hanmail.net

답장 전체답장 전달 삭제 스팸신고 | 이동 ▾ 추가 기능 ▾

soominpkr invited you to soominpkr/colaborator01

보낸사람 soominpkr<noreply@github.com> 주소추가 | 수신자단
받는사람 foxman12<foxman12@hanmail.net> 주소추가



 + 

@soominpkr has invited you to collaborate
soominpkr/colaborator01 repository

You can [accept](#) or [decline](#) this invitation. You can also head over to <https://github.com/soominpkr/colaborator01> to check out the repository or visit @soominpkr to more about them.

This invitation will expire in 7 days.

[View invitation](#)

Note: This invitation was intended for foxman12@hanmail.net. If you were not ex

[2단계] 협업자(userB(foxman12@hanmail.net))의 작업 흐름

1. 저장소 클론

```
mkdir cool-project #폴더 생산
git clone https://github.com/userA/cool-project.git .
cd cool-project
```

2. 브랜치 생성 및 전환

```
git checkout -b feature/login
```

`feature/login`은 로그인 기능을 개발하려는 브랜치 이름입니다.

```
git branch #현재 brnch를 확인할 수 있다.
```

3. 작업 및 커밋

예: `index.html` 수정했다고 가정

```
git config user.email #foxman12@hanmail.net
git add index.html #git add .
git commit -m "Add login form UI"
```

4. 브랜치 푸시

#foxman12@hanmail.net 개정으로 foxman12@naver.com에 push한다.
userB → userA

```
git remote -v
git push origin feature/login
```

이게 바로 **Collaborator**이기 때문에 가능한 직접 **push**입니다.

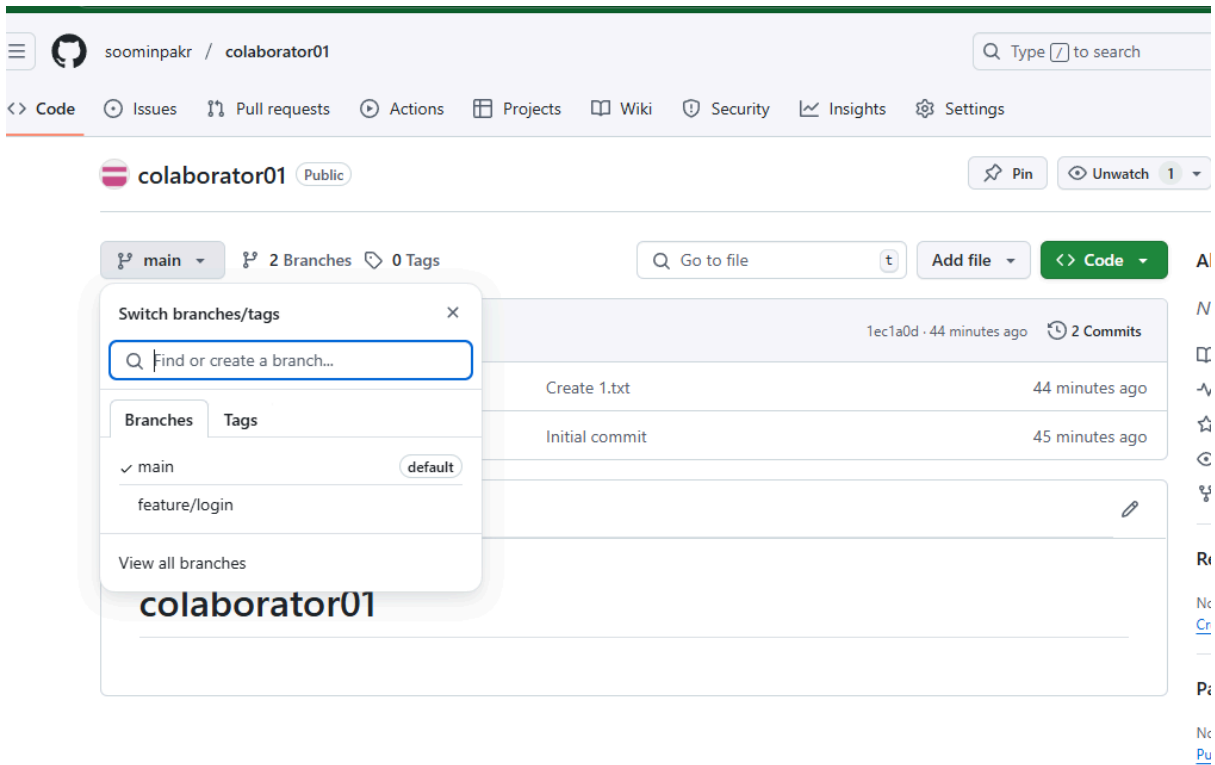
> 과거 버전

[3단계] PR(Pull Request) 생성

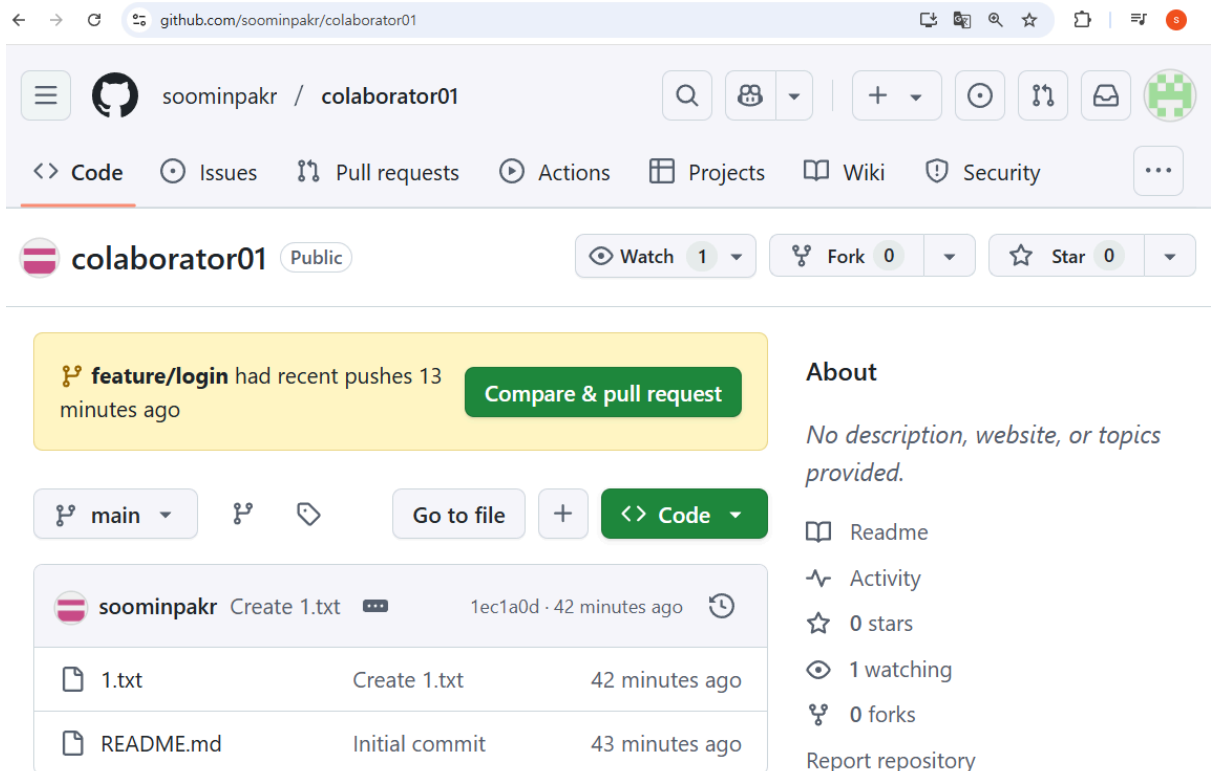
userB([@hanmail.net](mailto:hanmail.net))가 userA깃허브 주소로 간다.

1. GitHub 웹사이트에서 **userA/cool-project**에 접속
2. **feature/login** 브랜치가 푸시된 것을 확인
실제 계정에는 이미 추가되어 있는것을 확인할 수 있다.

userA(@naver.com)



3. userB로 (@hanmail.net)으로 로그인 한 상태에서 userA링크에 접속하면 상단에 다음과 같은 문구가 있다. "Compare & Pull Request" 클릭



4. 설명 작성 후 PR 생성 및 Merge 작업을 할 수 있다.

팀원들과 코드 리뷰 후 `main` 브랜치로 병합(Merge) 가능

✓ 전체 흐름 요약

단계	설명
1	저장소 주인이 협업자 초대
2	협업자가 <code>clone</code> 후 브랜치 생성
3	작업하고 <code>commit</code> 후 <code>push</code> (<code>git push origin</code> <code>브랜치</code>)
4	GitHub에서 Pull Request 생성
5	리뷰 후 Merge (or 수정 반복)

필요하다면 이걸 PDF나 README.md 형식으로도 정리해드릴 수 있어요.

또는 이걸 실제 연습해보고 싶다면 테스트용 **GitHub** 저장소를 만들어 같이 진행도 가능합니다.

userB는 마무리 해서 병합을 원함으로 `compare pull request` 눌러서 병합 요청을한다.

The screenshot shows the GitHub interface for the repository 'colaborator01' by user 'soominpokr'. The repository is public and has 1 watch, 0 forks, and 0 stars. A yellow banner at the top indicates that the 'feature/login' branch has recent pushes 13 minutes ago, with a 'Compare & pull request' button. Below this, the 'main' branch is selected, and there are buttons for 'Go to file' and 'Code'. A table of recent commits is shown, with the latest commit '1ec1a0d' from 'soominpokr' titled 'Create 1.txt' made 42 minutes ago. The commit details show a file '1.txt' created 42 minutes ago and a 'README.md' file committed 43 minutes ago. On the right side, the 'About' section is empty, and the 'Readme', 'Activity', '0 stars', '1 watching', and '0 forks' sections are visible.

File	Commit	Time
1.txt	Create 1.txt	42 minutes ago
README.md	Initial commit	43 minutes ago

userA로 로그인한 사용자가 아래처럼 `pull requests` 메뉴를 통해서 추가 할 수 있다.

soominpkr / collaborator01

Type to search

Code

Issues

Pull requests

1

Actions

Projects

Wiki

Security

Insights

Settings

로기add login #1

Open

grapefoxman wants to merge 1 commit into main from feature/login

Conversation 0

Commits 1

Checks 0

Files changed 1

grapefoxman commented now

Collaborator

로그인 완료

add login

7b5381a

No conflicts with base branch

Merging can be performed automatically.

Merge pull request

You can also merge this with the command line. [View command line instructions.](#)

Add a comment

Write

Preview

H B I

되도록 userA로 하는 것이 좋은데 userB가 머지 할 수도 있다.

"Compare & pull request" 버튼은 **GitHub**에서 브랜치 작업을 마친 후, 그 변경사항을 **main** 브랜치(또는 다른 브랜치)에 병합하려고 할 때 사용하는 기능이에요.

"Compare & pull request" 뜻

- **Compare:** 현재 브랜치의 변경사항을 **main** 브랜치와 비교합니다.
 - **Pull request (PR):** 비교한 변경사항을 **main** 브랜치에 병합해달라고 요청합니다.
-

예시 흐름

1. 당신이 **feature/login** 브랜치에서 로그인 기능을 개발함
 2. **git push origin feature/login** 으로 **GitHub**에 업로드함
 3. **GitHub** 저장소에 들어가면 → **Compare & pull request** 버튼이 뜸
 4. 클릭하면 다음 화면이 나옴:
 - "이 브랜치를 **main**에 병합할까요?"
 - 변경된 파일 목록, 커밋 내역 등 확인 가능
 5. 설명을 쓰고 **"Create pull request"** 클릭
 6. 팀원들이 코드 리뷰 후 → ****"Merge pull request"***로 병합
-

전체 흐름 요약

1. "Compare & pull request" 클릭

- 🔍 병합할 내용을 비교하고 설명 작성하는 ****Pull Request** 생성 화면**으로 이동
- 이 단계에서는 아직 **main**에 병합되지 않음

2. "Create pull request" 클릭

- Pull Request(PR)가 생성됨
- 팀원이 리뷰하거나, 본인이 직접 병합 가능

3. "Merge pull request" 클릭

-  이 때 브랜치 내용이 **main**에 병합됨

4. (선택) 브랜치 삭제

- 병합 후 **GitHub**이 "브랜치 삭제할래?" 라고 물어봄
- 이때 삭제하면 브랜치가 **GitHub**에서 사라짐

요약

단계	main에 병합?	브랜치 삭제됨?
Compare & pull request 클릭	✗	✗
Create pull request 클릭	✗	✗
Merge pull request 클릭	✓	✗

브랜치 삭제 선택

-



👁️ 참고로...

로컬(`git`)에 있는 브랜치는 `GitHub`에서 삭제해도 사라지지 않아요.

직접 `git branch -d feature/login` 명령으로 삭제해줘야 합니다.

필요하면 로컬-원격 브랜치 삭제 전체 예제도 만들어드릴게요.

✅ Fork + Pull Request 방식 (간접 push) 쉽게 이해하기

GitHub에서 다른 사람의 저장소에 기여할 때 사용하는 방법입니다. 직접 권한이 없어도 코드에 기여할 수 있게 해주는 시스템이에요.

📌 전체 흐름 요약

1. 원본 저장소를 내 계정으로 **Fork** (복사)
2. Fork한 저장소를 내 컴퓨터에 **Clone** (다운로드)
3. 코드 수정 후 내 Fork 저장소에 **Push** (업로드)
4. 원본 저장소에 **Pull Request** (병합 요청)
5. 원본 저장소 관리자가 검토 후 병합

🔧 단계별 상세 설명

포크 저장소 `colaborator02`로 생성한다.

① Fork (포크) - 저장소 복사하기

- GitHub에서 기여하고 싶은 원본 저장소로 이동
- 오른쪽 상단의 **"Fork"** 버튼 클릭
- 내 GitHub 계정에 동일한 저장소가 생성됨 (예: `내계정/저장소명`)

💡 포크는 원본 저장소의 사본을 내 계정에 만드는 것입니다. 원본과는 독립적이지만, 나중에 동기화할 수 있습니다.

② Clone (클론) - 내 컴퓨터로 가져오기

- 내 계정에 생성된 포크 저장소로 이동
- **"Code"** 버튼 클릭 후 HTTPS/SSH 주소 복사
- 터미널/CMD에서 다음 명령어 실행:
-
- `git clone` [복사한 주소]
- 이제 내 컴퓨터에서 작업할 준비 완료!

③ 작업 후 Push (푸시) - 변경사항 업로드

1. 새 브랜치 만들기 (권장):
2. `git clone` [복사한 주소]
- 3.
4. `git branch`
5. `git checkout -b feature/new-feature`
6. 파일 수정/추가 후 저장
7. 변경사항 스테이징:
8. `git branch`
- 9.
10. `git add .`
11. 커밋 생성:
- 12.
13. `git commit -m "설명하는 커밋 메시지"`
- 14.
15. `git remote -v`
16. `git remote remove origin`
17. `git remote add origin https://github.com/yourAccount/the001.git`
- 18.
- 19.
20. 내 포크 저장소에 푸시:
21. `git push origin feature/new-feature`

④ Pull Request (풀 리퀘스트) - 병합 요청하기

UserB

1. 내 GitHub 포크 저장소로 이동
2. **"Pull request"** 버튼이 나타남 (또는 **"Contribute"** → **"Open Pull Request"**)
3. 원본 저장소와 내 변경사항 비교 화면이 나타남
4. 제목과 설명을 작성 후 **"Create Pull Request"** 클릭
- 5.

5 관리자 검토 및 병합

userA

- 원본 저장소 관리자가 내 PR을 검토
- 필요시 코멘트나 수정 요청이 올 수 있음
- 관리자가 승인하면 **"Merge"** 버튼으로 병합 완료!



팁

- 작업 전 항상 원본 저장소와 동기화하는 것이 좋음
- 하나의 PR에는 하나의 기능/수정사항만 포함시키기
- 커밋 메시지는 명확하고 간결하게 작성
- PR 설명에 변경 이유와 내용을 상세히 기술

이 방식은 오픈소스 프로젝트에 기여할 때 가장 일반적으로 사용되는 방법입니다! 😊

`git checkout main`

```
git merge feature/new-feature  
git branch -d feature/new-feature
```