

# 명령어로 깃 다루기

1

2

# 8장 명령어로 깃허브로 다루기

- 8.1 | 원격 저장소와 상호 작용하기
- 8.2 | 깃 명령으로 풀 리퀘스트 보내기
- 8.3 | 더 나아가기

# 8.1 원격 저장소와 상호 작용하기

## » 원격 저장소와 상호 작용하기

- 그럼 명령어로 깃허브와 상호 작용하는 방법에 대해 알아보자
- 5장에서 깃허브와 상호 작용하는 방법에는 다음 네 가지가 있다고 했음

1 | 클론(clone): 원격 저장소를 복제하기

2 | 푸시(push): 원격 저장소에 밀어넣기

3 | 패치(fetch): 원격 저장소를 일단 가져만 오기

4 | 풀(pull): 원격 저장소를 가져와서 합치기

# 8.1 원격 저장소와 상호 작용하기

## » 원격 저장소와 상호 작용하기

- 이번에는 여기에 하나를 더 추가해 보자
- 두 번째에 ‘리모트’가 추가
- 이게 무엇인지에 대해서는 해당 절에서 자세히 설명하겠음

1 | 클론(clone): 원격 저장소를 복제하기

2 | 리모트(remote): 원격 저장소를 추가하고, 조회하고, 삭제하기

3 | 푸시(push): 원격 저장소에 밀어넣기

4 | 패치(fetch): 원격 저장소를 일단 가져만 오기

5 | 풀(pull): 원격 저장소를 가져와서 합치기

# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

- 먼저 명령어로 원격 저장소를 클론해 보자
- 클론이란 원격 저장소를 복제하는 방법이라 설명한 바 있음
- 원격 저장소를 클론하는 명령은 git clone <원격 저장소>
- 가령 <https://github.com/kangtegong/collaboration> 링크에 접속하면 볼 수 있는 collaboration 저장소를 여러분의 컴퓨터로 클론해 보자

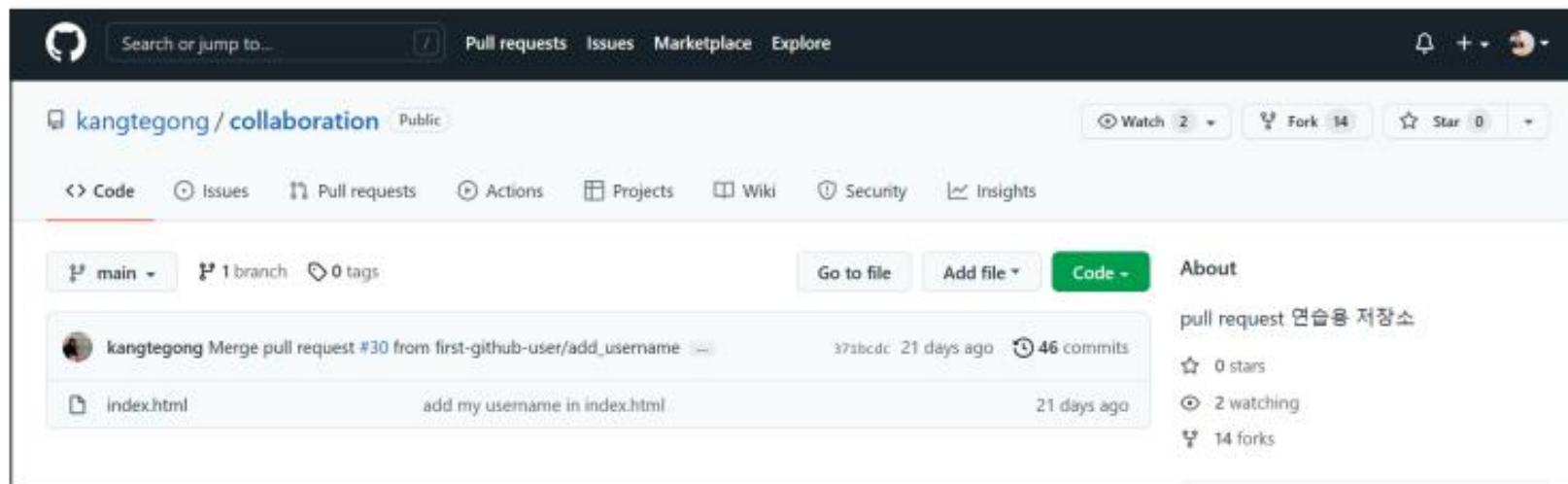
# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

- ① 원격 저장소 링크에 들어가 보자

URL <https://github.com/kangtegong/collaboration>

### ▼ 그림 8-1 | 클론받을 원격 저장소

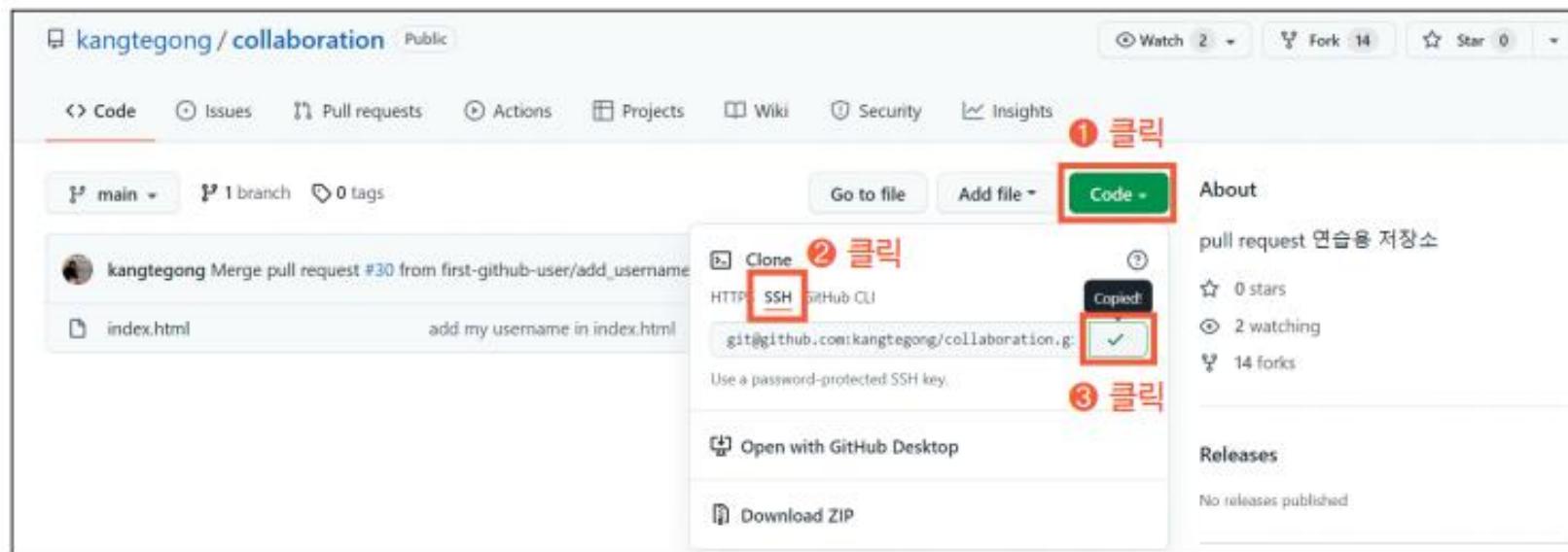


# 8.1 원격 저장소와 상호 작용하기

» git clone: 원격 저장소를 복제하기

- ② 다음처럼 Code를 클릭한 다음 원격 저장소 경로를 복사

▼ 그림 8-2 | 원격 저장소 경로 복사하기



# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

- ③ 이제 이 저장소를 클론받을 위치에서 깃 배시를 열어줌
- 여기에서는 C:\에 클론받기 위해 C:\ 경로에서 깃 배시를 열겠음

### ▼ 그림 8-3 | 깃 배시 열기



The screenshot shows a terminal window titled "MINGW64:/c". The title bar includes the MinGW logo, the title "MINGW64:/c", and standard window control buttons. The main area of the terminal displays a command-line interface. The prompt shows the user's name "minchul" followed by the host name "DESKTOP-9KULGUE" and the environment "MINGW64 /c". A dollar sign (\$) is displayed at the end of the line, indicating where the user can type commands.

# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

- ④ 이제 원격 저장소를 클론
- 원격 저장소를 클론하는 명령은 git clone <원격 저장소>
- git clone을 입력하고, 복사한 경로를 붙여넣으면 됨

```
minchul@DESKTOP-9KULGUE MINGW64 /c
$ git clone git@github.com:kangtegong/collaboration.git
Cloning into 'collaboration'...
remote: Enumerating objects: 102, done.
remote: Counting objects: 100% (102/102), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 102 (delta 16), reused 65 (delta 13), pack-reused 0
Receiving objects: 100% (102/102), 23.35 KiB | 97.00 KiB/s, done.
Resolving deltas: 100% (16/16), done.
```

# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

### 특정 경로로 클론받기

- git clone <원격 저장소>는 암묵적으로 현재 경로(이 예시의 경우에는 C:\)에 원격 저장소를 클론
- 특정 경로에 원격 저장소를 클론받기 위해서는 git clone <원격 저장소> <클론받을 경로> 형식으로 명령을 입력
- 가령 다음과 같이 입력한다면 원격 저장소는 C:\test에 collaboration 저장소를 클론

```
minchul@DESKTOP-9KULGUE MINGW64 /c
$ git clone git@github.com:kangtegong/collaboration.git /C/test
Cloning into 'C:/test/collaboration'...
remote: Enumerating objects: 106, done.
remote: Counting objects: 100% (106/106), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 106 (delta 17), reused 68 (delta 14), pack-reused 0
Receiving objects: 100% (106/106), 24.20 KiB | 6.05 MiB/s, done.
Resolving deltas: 100% (17/17), done.
```

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-4 | 지정한 경로(C:\test)에 클론한 원격 저장소



# 8.1 원격 저장소와 상호 작용하기

## » git clone: 원격 저장소를 복제하기

- ⑤ 클론받은 원격 저장소를 확인해 보자
- 확인했다면 이 저장소는 삭제해도 좋음

### ▼ 그림 8-5 | 성공적으로 클론된 원격 저장소



# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

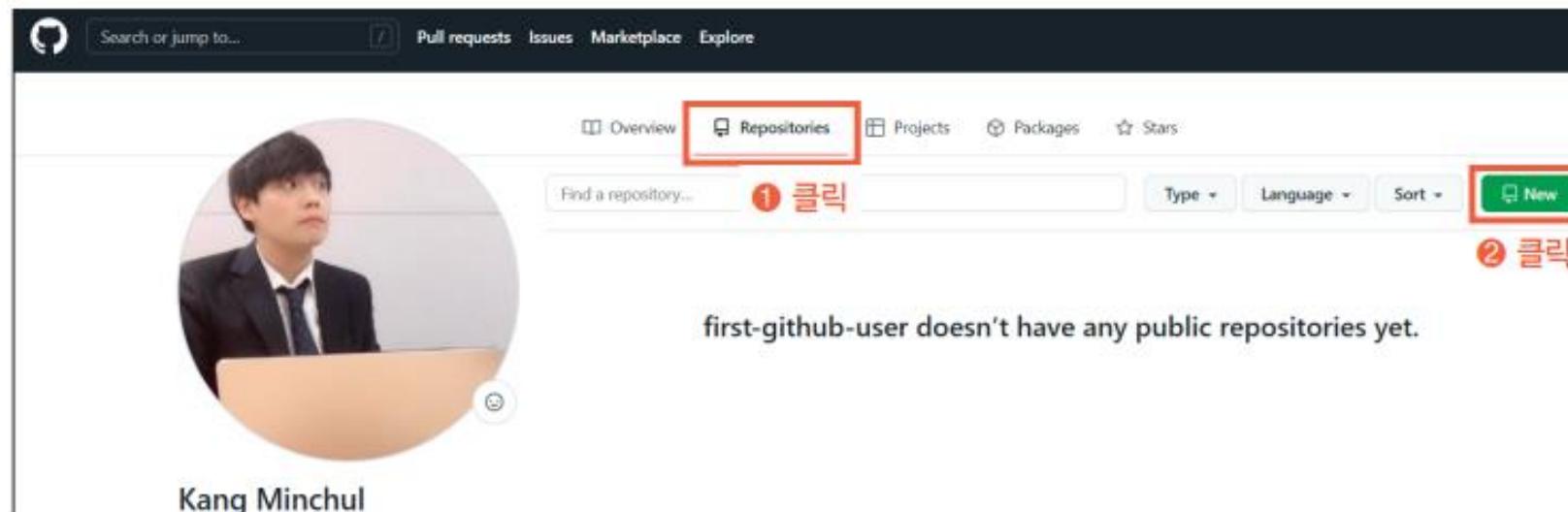
- 이번에는 원격 저장소와의 두 번째 상호 작용, 리모트에 대해 학습
- git remote는 원격 저장소를 추가하고, 조회하고, 삭제할 수 있는 명령
- 이 명령을 자세히 학습하기 앞서 우선 상호 작용할 원격 저장소를 만들어 보자
- 깃허브로 가보자

# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ① 원격 저장소를 생성하는 방법은 5장에서 설명했음
- 여러분의 프로필 페이지에서 Repositories의 New를 클릭

▼ 그림 8-6 | ‘Repositories’의 ‘New’ 클릭하기



## 8.1 원격 저장소와 상호 작용하기

» git remote: 원격 저장소를 추가, 조회, 삭제하기

- ② 저장소 이름(Repository name)은 test-repo로 하겠음
- 이때 원활한 실습을 위해 그림 속 박스 친 부분은 체크하지 말고 Create repository를 클릭

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-7 | 원격 저장소 생성하기

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner \* first-github-user Repository name \* test-repo ① 입력

Great repository names are short and memorable. Need inspiration? How about [expert·guacamole?](#)

Description (optional)

② 체크 해제

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file  
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore  
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license  
A license tells others what they can and can't do with your code. [Learn more.](#)

**Create repository** ③ 클릭

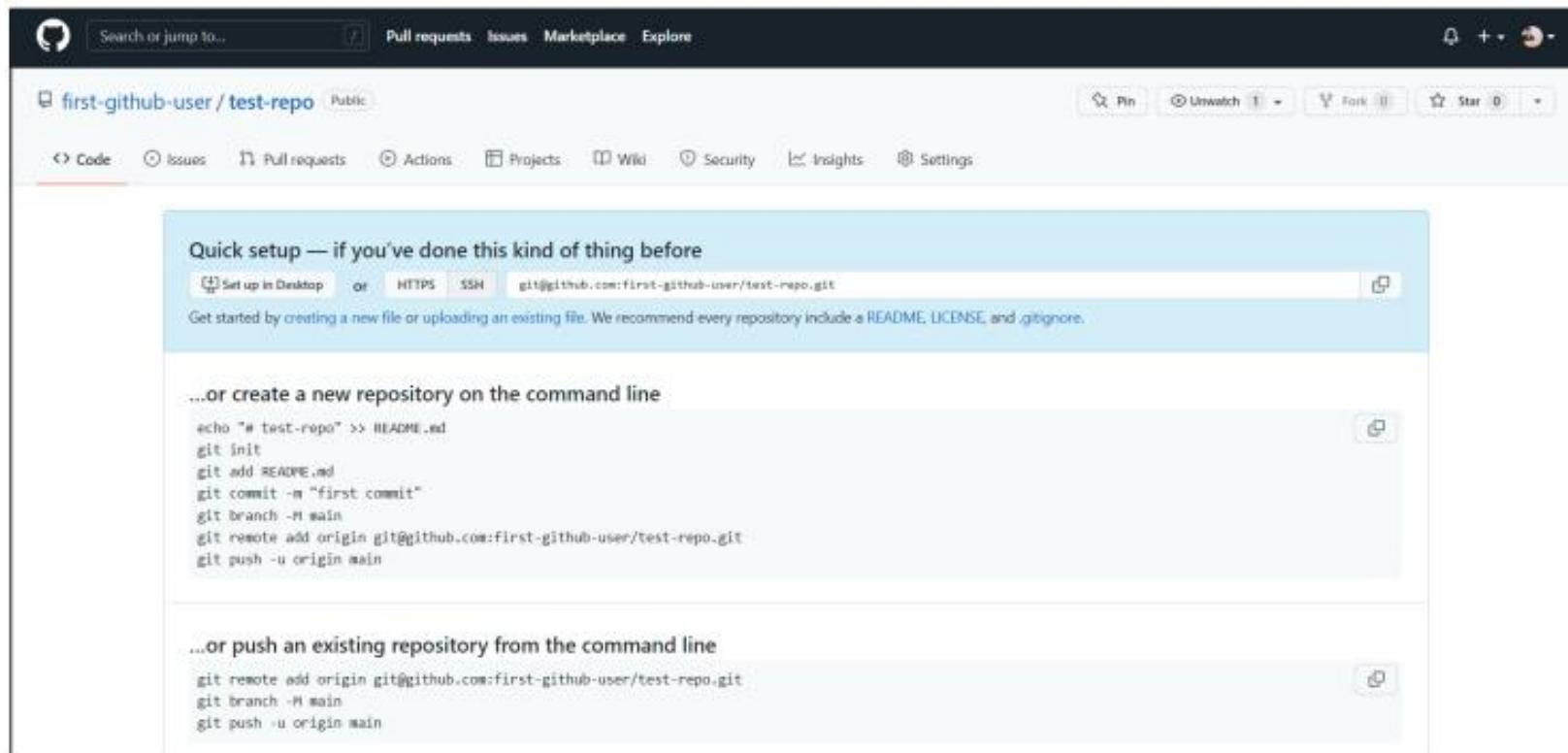
# 8.1 원격 저장소와 상호 작용하기

» git remote: 원격 저장소를 추가, 조회, 삭제하기

- ③ 생성된 원격 저장소의 모습
- 이제부터 여러분은 이 절 전체에 걸쳐 이 저장소와 상호 작용하는 연습을 할 예정

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-8 | 생성된 원격 저장소의 모습



# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ④ 원격 저장소를 만들었다면 이제 여러분의 컴퓨터에 로컬 저장소를 만들어 보자
- test-repo라는 폴더를 만들고, 그 안에서 깃 배시를 열어 git init 명령으로 로컬 저장소를 만들어 보자
- C:\ 경로에 test-repo 폴더를 만들고, 그 안에서 git init 명령을 입력

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo
$ git init
Initialized empty Git repository in C:/test-repo/.git/
```

# 8.1 원격 저장소와 상호 작용하기

» git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑤ 생성된 로컬 저장소의 모습
- 아직 버전을 관리할 어떤 대상도 없음

▼ 그림 8-9 | 생성된 로컬 저장소의 모습



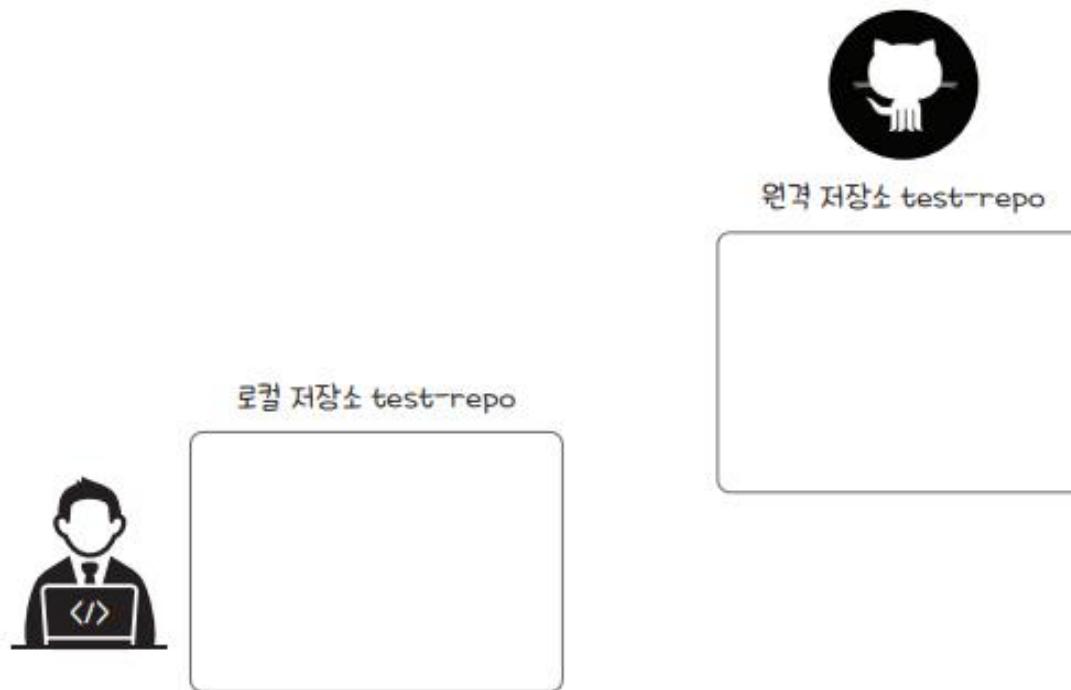
## 8.1 원격 저장소와 상호 작용하기

### » git remote: 원격 저장소를 추가, 조회, 삭제하기

- 현재 상황을 그림으로 표현하면 다음과 같음
- 현재 로컬 저장소 test-repo는 원격 저장소 test-repo의 존재를 알지 못함
- 로컬 저장소 test-repo는 원격 저장소 testrepo와 상호 작용할 수 없음
- 로컬 저장소 test-repo가 원격 저장소 test-repo와 상호 작용하려면 원격 저장소 test-repo를 로컬 저장소 test-repo에 추가

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-10 | 현재 로컬 저장소와 원격 저장소의 상황



# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑥ 로컬 저장소에 원격 저장소를 추가하는 명령은 git remote add <원격 저장소 이름><원격 저장소 경로>
- 앞서 생성한 원격 저장소의 경로는 (원격 저장소에 들어가보면 확인할 수 있듯)  
git@github.com:first-github-user/test-repo.git

### ▼ 그림 8-11 | 원격 저장소의 경로 확인하기



## 8.1 원격 저장소와 상호 작용하기

### » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑦ 이 원격 저장소를 origin이라는 이름으로 로컬 저장소에 추가
- 명령은 git remote add origin <원격 저장소 경로>를 입력하면 됨
- 이렇게 원격 저장소를 추가하면 추후 origin이라는 이름으로 git@github.com:first-github-user/test-repo.git과 상호 작용할 수 있음

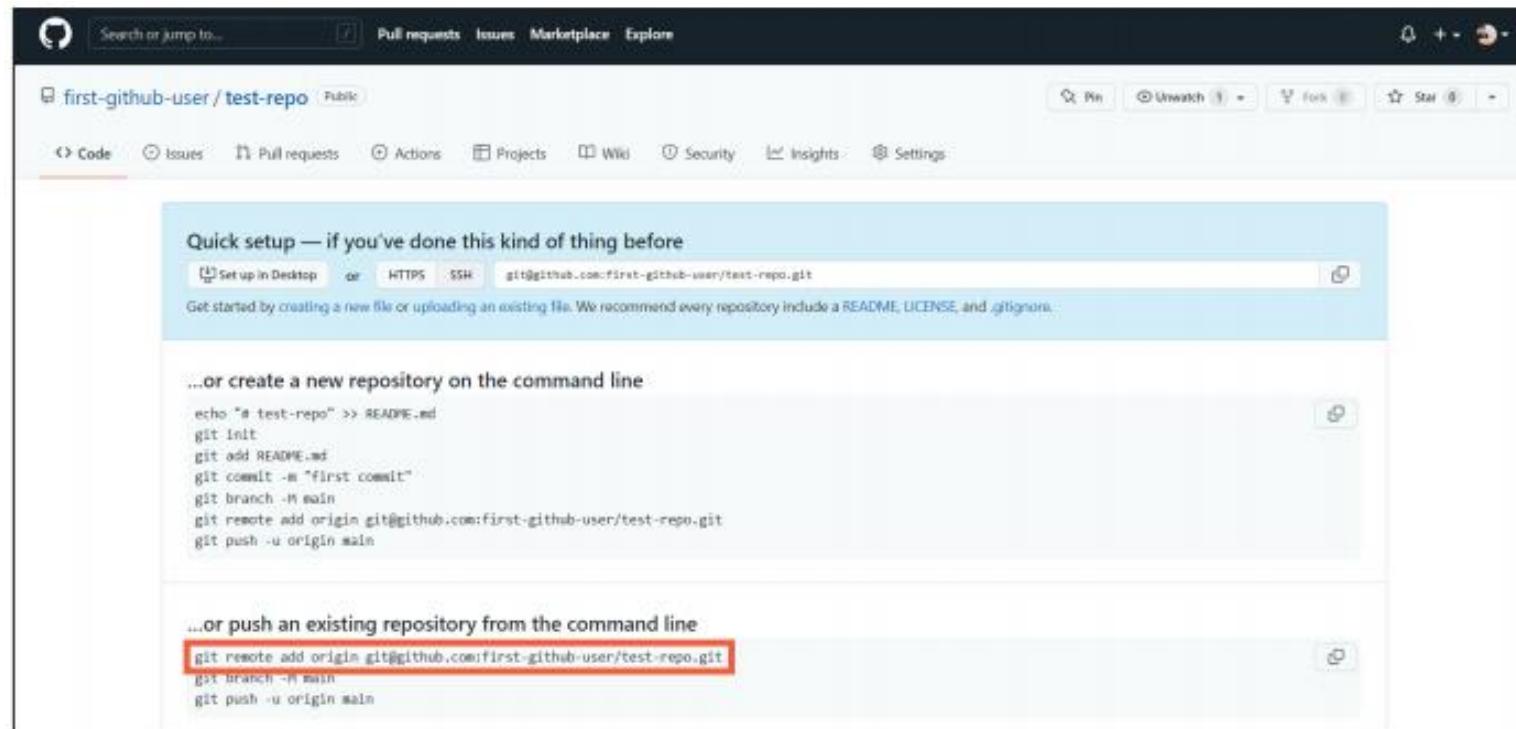
```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote add origin git@github.com:first-github-user/test-repo.git
```

# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

- 원격 저장소에서 다음 그림 속 박스를 그대로 복사-붙여넣기 해도 무방함

▼ 그림 8-12 | 원격 저장소 추가하기



## 8.1 원격 저장소와 상호 작용하기

### » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑧ 추가된 원격 저장소 목록은 git remote 명령으로 확인할 수 있음
- 방금 추가했던 원격 저장소의 이름(origin)이 보임

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote
origin
```

## 8.1 원격 저장소와 상호 작용하기

### » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑨ git remote -v 또는 git remote --verbose 명령을 입력하면 원격 저장소의 이름과 경로를 함께 확인할 수 있음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote -v
origin  git@github.com:first-github-user/test-repo.git (fetch)
origin  git@github.com:first-github-user/test-repo.git (push)
```

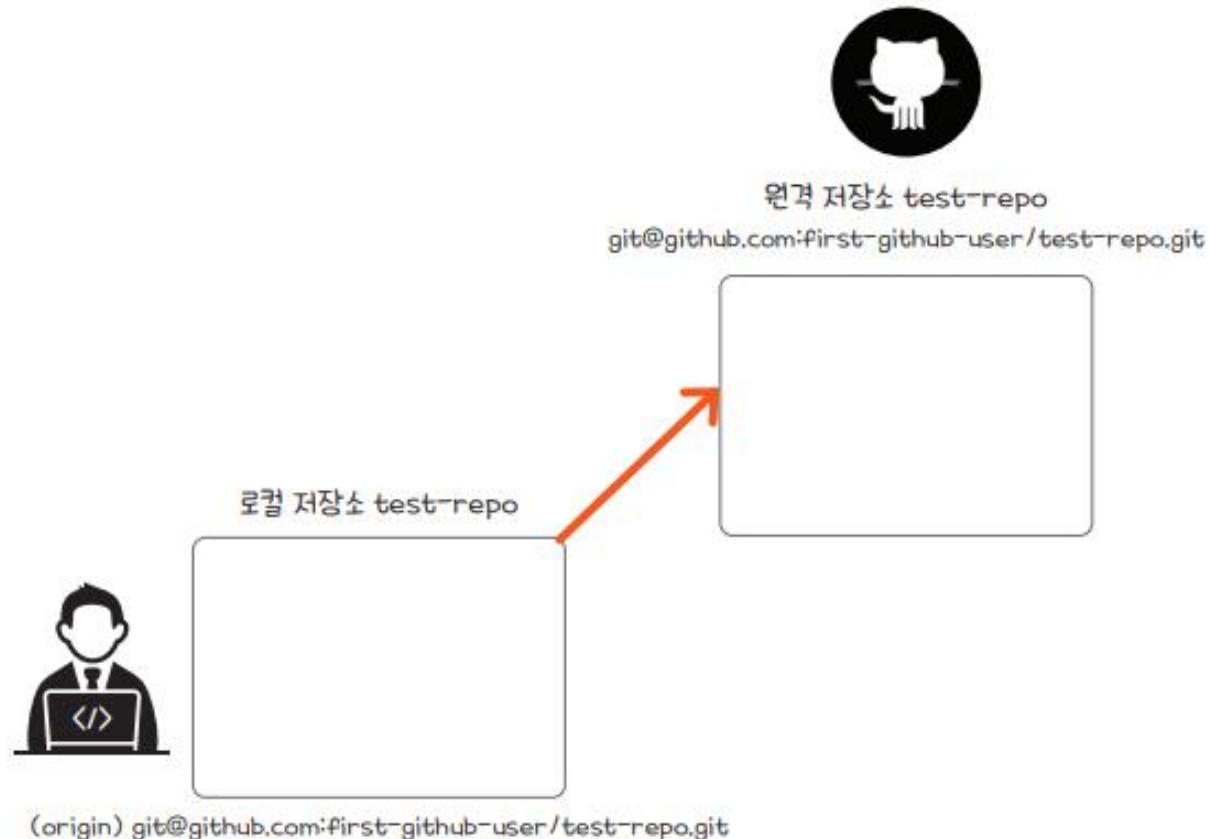
## 8.1 원격 저장소와 상호 작용하기

» git remote: 원격 저장소를 추가, 조회, 삭제하기

- 지금까지의 상황을 그림으로 표현하면 그림 8-13과 같음
- 로컬 저장소에 원격 저장소를 추가했기 때문에 이제 로컬 저장소는 이 원격 저장소와 상호 작용할 수 있게 됐음

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-13 | 현재 로컬 저장소와 원격 저장소의 상황



## 8.1 원격 저장소와 상호 작용하기

### » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑩ origin이라는 이름은 얼마든지 바꿀 수 있음
- 원격 저장소의 이름을 바꾸는 명령은 git remote rename <기존 원격 저장소 이름> <바꿀 원격 저장소 이름>
- 원격 저장소의 이름을 origin에서 changed로 바꾼 뒤 git remote 또는 git remote -v 명령으로 확인해 보자

## 8.1 원격 저장소와 상호 작용하기

» git remote: 원격 저장소를 추가, 조회, 삭제하기

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote rename origin changed
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote
changed
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote -v
changed git@github.com:first-github-user/test-repo.git (fetch)
changed git@github.com:first-github-user/test-repo.git (push)
```

# 8.1 원격 저장소와 상호 작용하기

## » git remote: 원격 저장소를 추가, 조회, 삭제하기

- ⑪ 추가한 원격 저장소를 삭제하는 방법도 알아보자
- 추가한 원격 저장소를 삭제하는 명령은 간단함
- git remote remove <원격 저장소 이름>
- 가령 원격 저장소 changed를 삭제하는 명령은 git remote remove changed
- 삭제한 후 git remote 명령을 쳤을 때 아무것도 나오지 않는다면 성공적으로 삭제된 것

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote remove changed
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git remote
```

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

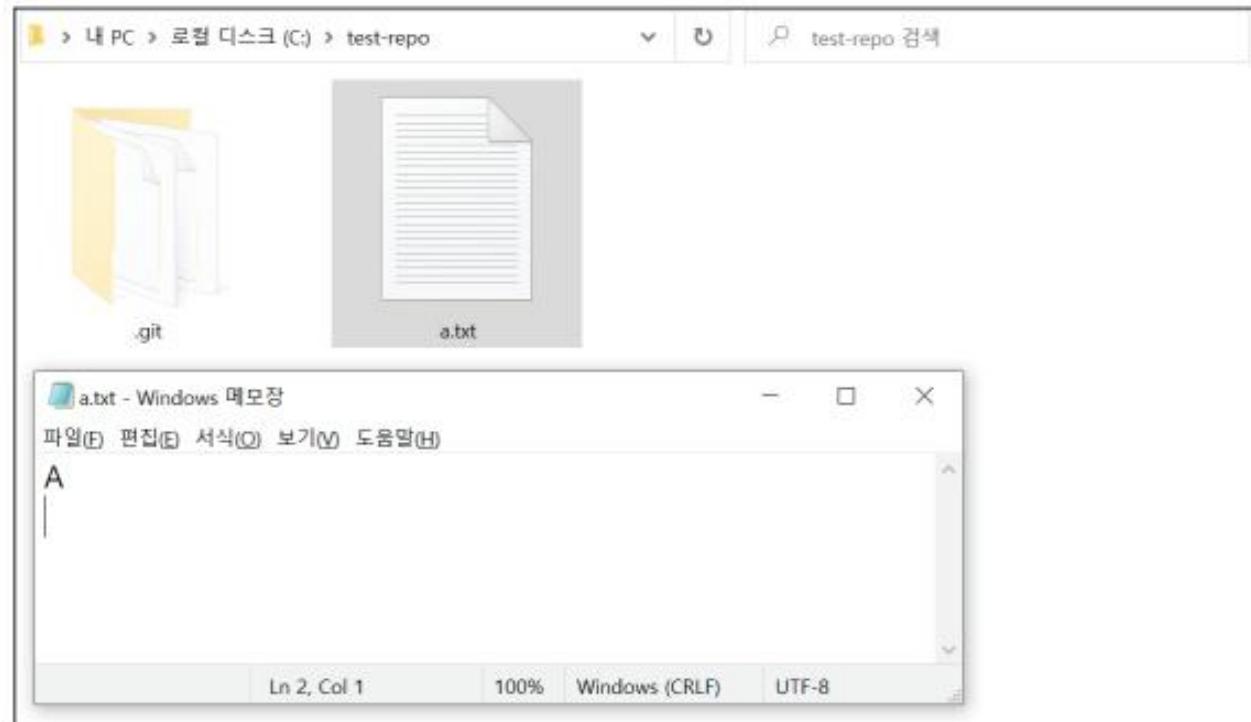
- 이제 원격 저장소와의 세 번째 상호 작용, git push에 대해 알아보자
- 푸시는 로컬 저장소의 변경 사항을 원격 저장소에 밀어넣는 방법이라고 했음
- 실습을 통해 익혀보자

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- ① 우선 버전을 관리할 대상을 만들어 보자
- 로컬 저장소 test-repo에 문자 A가 적힌 a.txt 파일을 생성한 뒤 저장

### ▼ 그림 8-14 | a.txt 파일 만들기



## 8.1 원격 저장소와 상호 작용하기

» git push: 원격 저장소에 밀어넣기

- ② 이를 스테이지에 추가하고, first commit이라는 커밋 메시지로 커밋

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git add a.txt
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (master)
$ git commit -m "first commit"
[master (root-commit) afd753a] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
```

## 8.1 원격 저장소와 상호 작용하기

### » git push: 원격 저장소에 밀어넣기

- 현재까지의 상황을 그림으로 표현하면 다음과 같음
- 원격 저장소에는 어떤 커밋도 없고, 로컬 저장소에는 하나의 커밋이 추가
- 이 커밋을 원격 저장소에 반영하려면 원격 저장소로 푸시

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-15 | 현재 로컬 저장소와 원격 저장소의 상황



# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- ③ 그럼 명령어로 푸시하는 방법을 익혀보자
- 사실 로컬 저장소의 커밋(들)을 처음 원격 저장소로 푸시하는 경우, 그림 8-16의 박스 친 명령들을 그대로 복사–붙여넣기 하면 됨
- 우선 이 명령들을 복사–붙여넣기 한 뒤 각 명령들의 의미를 하나씩 설명하겠습니다

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-16 | 처음 푸시하는 경우 복사–붙여넣기 할 명령들

The screenshot shows the GitHub 'Create Repository' wizard. At the top, it says 'Quick setup — if you've done this kind of thing before' with options to 'Set up in Desktop' or 'HTTPS / SSH'. A URL 'git@github.com:first-github-user/test-repo.git' is shown. Below this, a note says 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.'

Under '...or create a new repository on the command line', there is a block of terminal commands:

```
echo "# test-repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:first-github-user/test-repo.git
git push -u origin main
```

Under '...or push an existing repository from the command line', there is another block of terminal commands:

```
git remote add origin git@github.com:first-github-user/test-repo.git
git branch -M main
git push -u origin main
```

This second block is highlighted with a red rectangular box.

At the bottom, there is a section '...or import code from another repository' with the note 'You can initialize this repository with code from a Subversion, Mercurial, or TFS project.' and a 'Import code' button.

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- ④ 명령들을 복사-붙여넣기 한, 성공적으로 푸시된 모습은 다음과 같음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git remote add origin git@github.com:first-github-user/test-repo.git
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git branch -M main
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 214 bytes | 214.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:first-github-user/test-repo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- 그럼 이 명령들의 의미를 하나씩 알아보자
  - git remote add origin git@github.com:first-github-user/test-repo.git  
이 명령은 앞선 절에서 학습했음  
원격 저장소 <git@github.com:first-github-user/testrepo.git>을 origin이라는 이름으로 추가하는 명령
  - git branch -M main  
git branch -M <브랜치 이름>은 현재 브랜치 이름을 <브랜치 이름>으로 바꾸는 명령  
즉, 이 명령은 현재 브랜치(master) 이름을 main으로 변경하는 명령(브랜치 이름을 바꾸는 경우는 비교적 드물기 때문에 앞선 장에서 따로 설명하지는 않았음)  
깃허브에서는 기본 브랜치의 이름을 master 브랜치가 아닌 main 브랜치로 지칭  
로컬 저장소의 기본 브랜치는 master이지만 깃허브의 기본 브랜치는 main이기 때문에  
로컬 저장소의 기본 브랜치(master)에서 만든 변경 사항을 깃허브의 기본  
브랜치(main)로 푸시하기 위해서는 이와 같이 브랜치 이름을 main으로 변경

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- `git push -u origin main`

`git push <원격 저장소 이름> <브랜치 이름>`은 <원격 저장소 이름>으로 <브랜치 이름>을 푸시하는 명령

다시 말해, `git push origin main`은 원격 저장소 `origin`으로 로컬 저장소 `main` 브랜치의 변경 사항을 푸시하는 명령

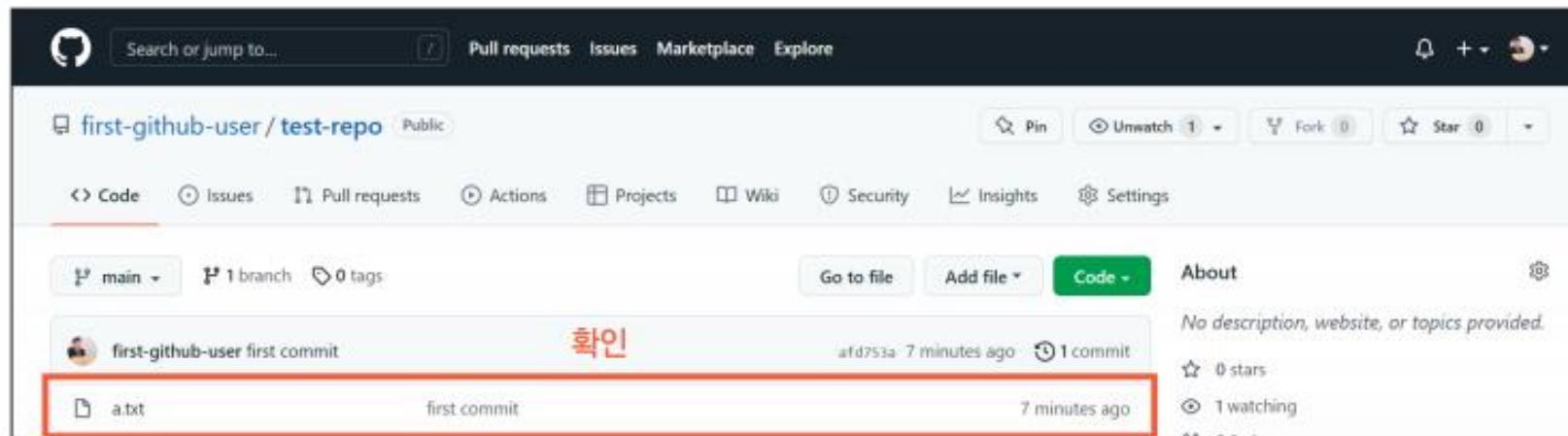
`-u` 옵션은 처음 푸시할 때 한 번만 사용하면 되는데, 이 옵션과 함께 푸시하면 추후 간단히 `git push`(또는 `git pull`) 명령만으로 `origin`의 `main` 브랜치로 푸시(또는 풀)할 수 있음

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- ⑤ 원격 저장소를 확인하면 로컬 저장소에서 커밋한 내용을 볼 수 있음

▼ 그림 8-17 | 성공적으로 푸시된 모습

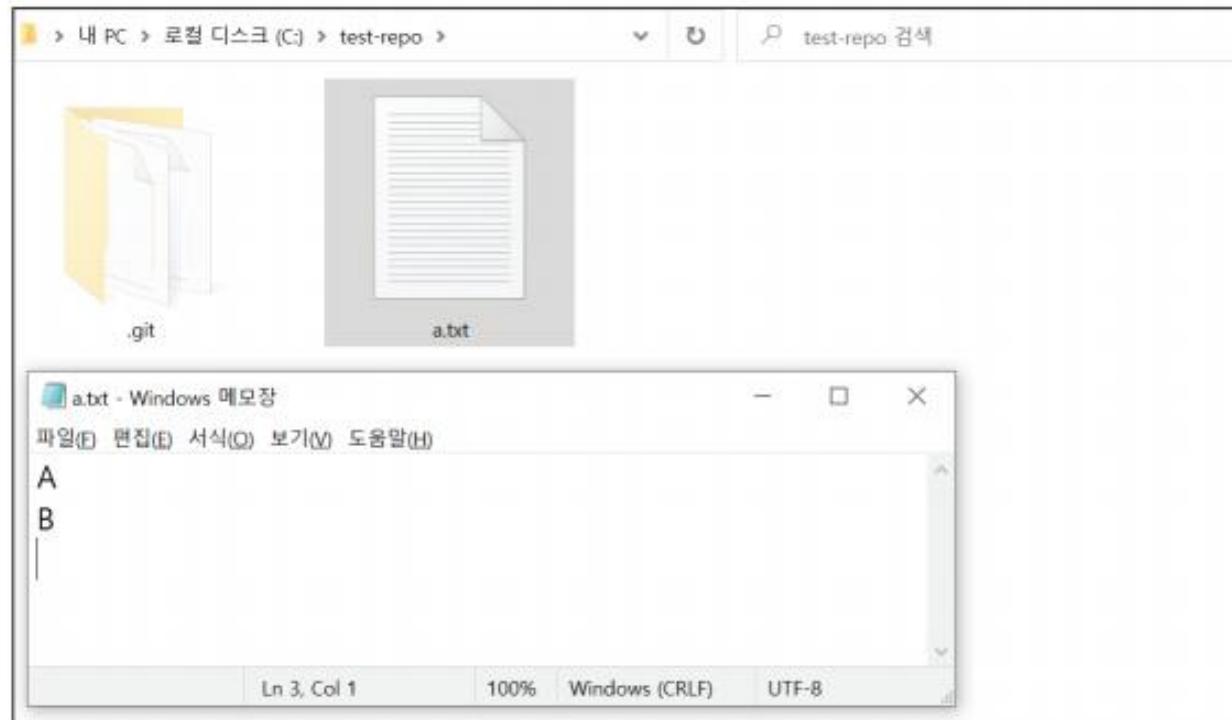


# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- 그럼 하나만 더 푸시해 보자
- ① a.txt 파일에 문자 B를 추가한 뒤 저장

### ▼ 그림 8-18 | 문자 B 추가 후 저장하기



## 8.1 원격 저장소와 상호 작용하기

### » git push: 원격 저장소에 밀어넣기

- ② 이를 스테이지에 추가하고, 커밋해 보자
- 커밋 메시지는 second commit으로 하겠음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git commit -am "second commit"
[main ec7531a] second commit
 1 file changed, 1 insertion(+)
```

## 8.1 원격 저장소와 상호 작용하기

### » git push: 원격 저장소에 밀어넣기

- ③ 이제 git push를 입력해 보자
- 앞서 -u 옵션과 함께 푸시했기 때문에 단순히 git push만 입력해도 됨

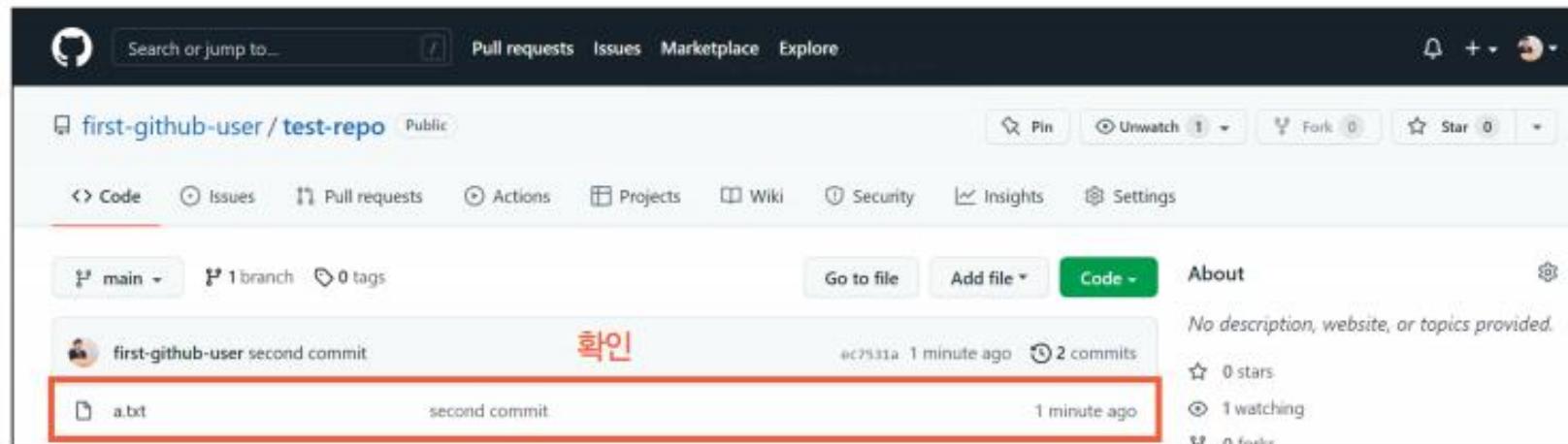
```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 247 bytes | 247.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:first-github-user/test-repo.git
  af753a..ec7531a  main -> main
```

# 8.1 원격 저장소와 상호 작용하기

## » git push: 원격 저장소에 밀어넣기

- ④ 성공적으로 푸시된 것을 확인해 보자
- 두 번째 커밋이 원격 저장소에 잘 반영

▼ 그림 8-19 | 성공적으로 푸시된 모습



## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- 패치란 원격 저장소의 변경 사항을 로컬 저장소에 병합하지 않고 ‘일단 가져만 오는’ 방법이라고 했음
- 이 또한 실습하며 익혀보겠음

## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- ① 우선 git log 명령으로 지금까지의 로컬 저장소 상황을 파악해 보자
- 다음 그림에서 박스 친 부분을 보면 알 수 있듯, 로컬 저장소의 main 브랜치와 원격 저장소의 main 브랜치(origin/main)에는 모두 동일하게 커밋 두 개가 쌓여 있음
- 달리 말해, 현재 로컬 저장소와 원격 저장소의 상태는 동일한 것

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git log
commit 1ac7f2fe1bdbbb8ff737af76ff0b502286c476f43 (HEAD -> main, origin/main)
Author: Kang Minchul <tegongkang@gmail.com>           확인
Date:   Thu Mar 24 02:44:44 2022 +0900

    second commit

commit abf6d7de46d0585bc02e293ad1657f0bcea2b945
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:44:28 2022 +0900

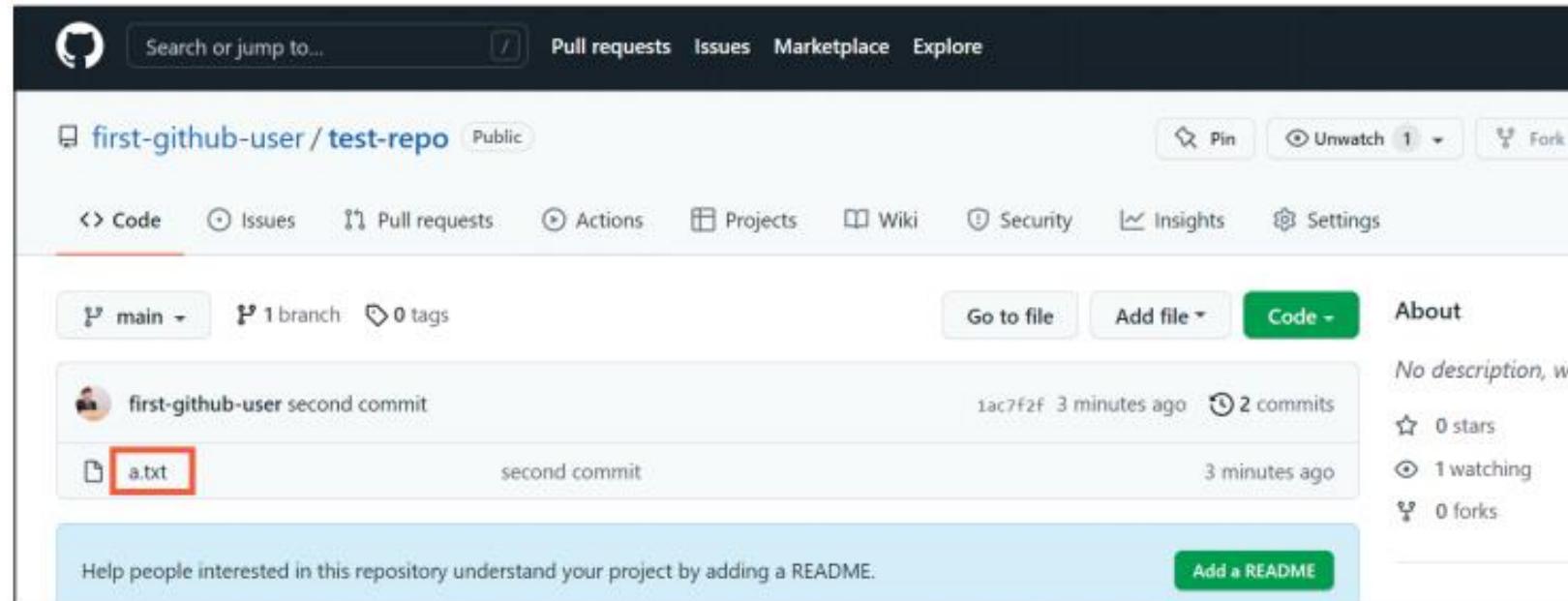
    first commit
```

# 8.1 원격 저장소와 상호 작용하기

## » git fetch: 원격 저장소를 일단 가져만 오기

- ② 깃허브에서 직접 세 번째 커밋을 추가하고, 이를 로컬 저장소로 패치해 보자
- 원격 저장소로 접속한 뒤 a.txt 파일을 클릭

### ▼ 그림 8-20 | a.txt 파일 클릭하기

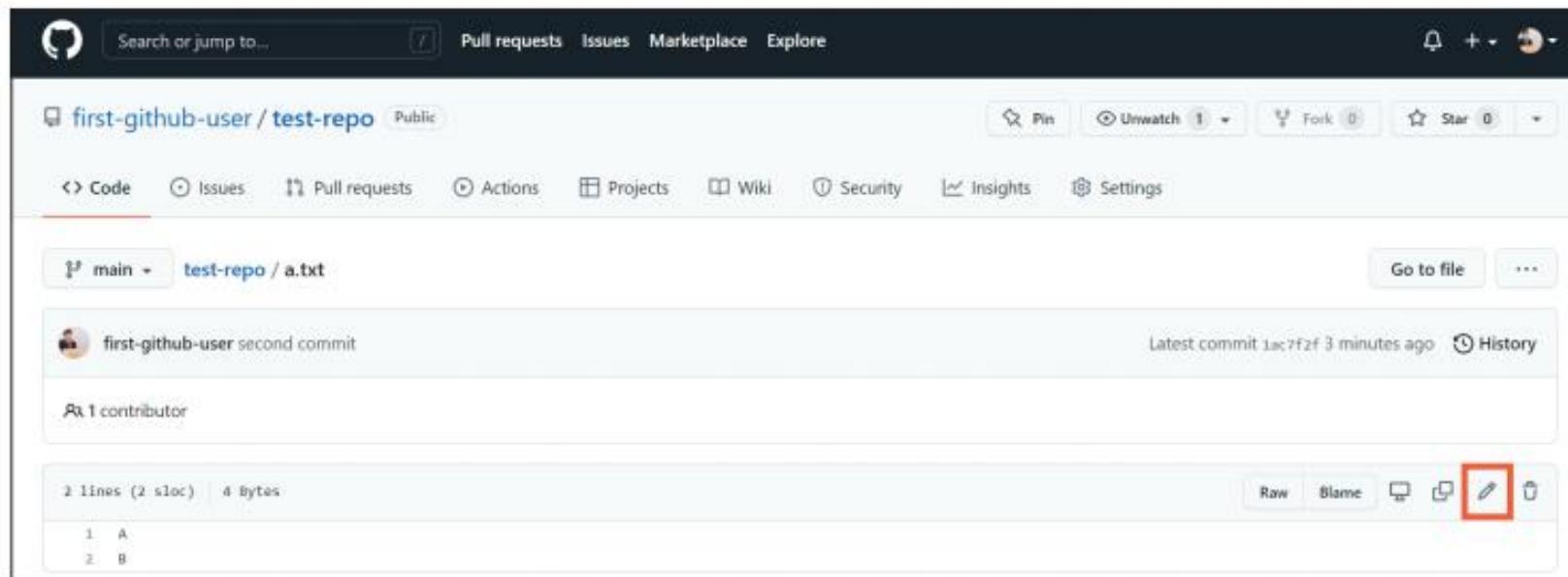


# 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ③ 연필 아이콘을 클릭

▼ 그림 8-21 | 연필 아이콘 클릭하기



# 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ④ 세 번째 줄에 문자 C를 입력

▼ 그림 8-22 | 문자 C 입력하기

first-github-user / test-repo Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

test-repo / a.txt in main

Edit file Preview changes Spaces 2

1	A
2	B
3	C
4	

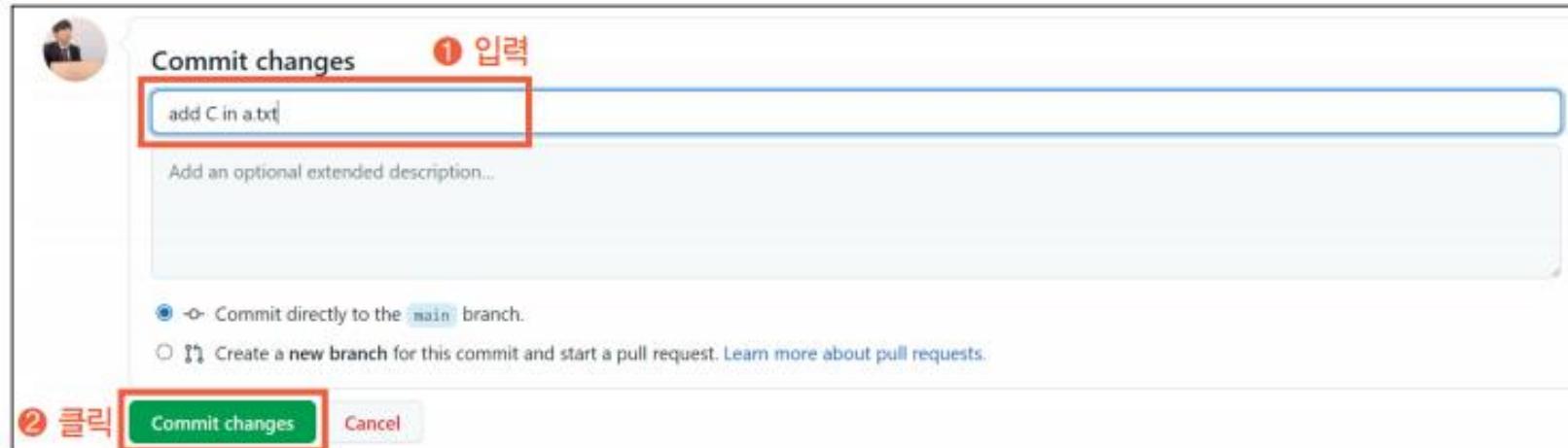
입력

# 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ⑤ 스크롤을 내려 커밋
- 커밋 메시지는 add C in a.txt로 하겠음
- 커밋 메시지를 입력했다면 Commit changes를 클릭

▼ 그림 8-23 | 커밋 메시지 입력 후 커밋하기

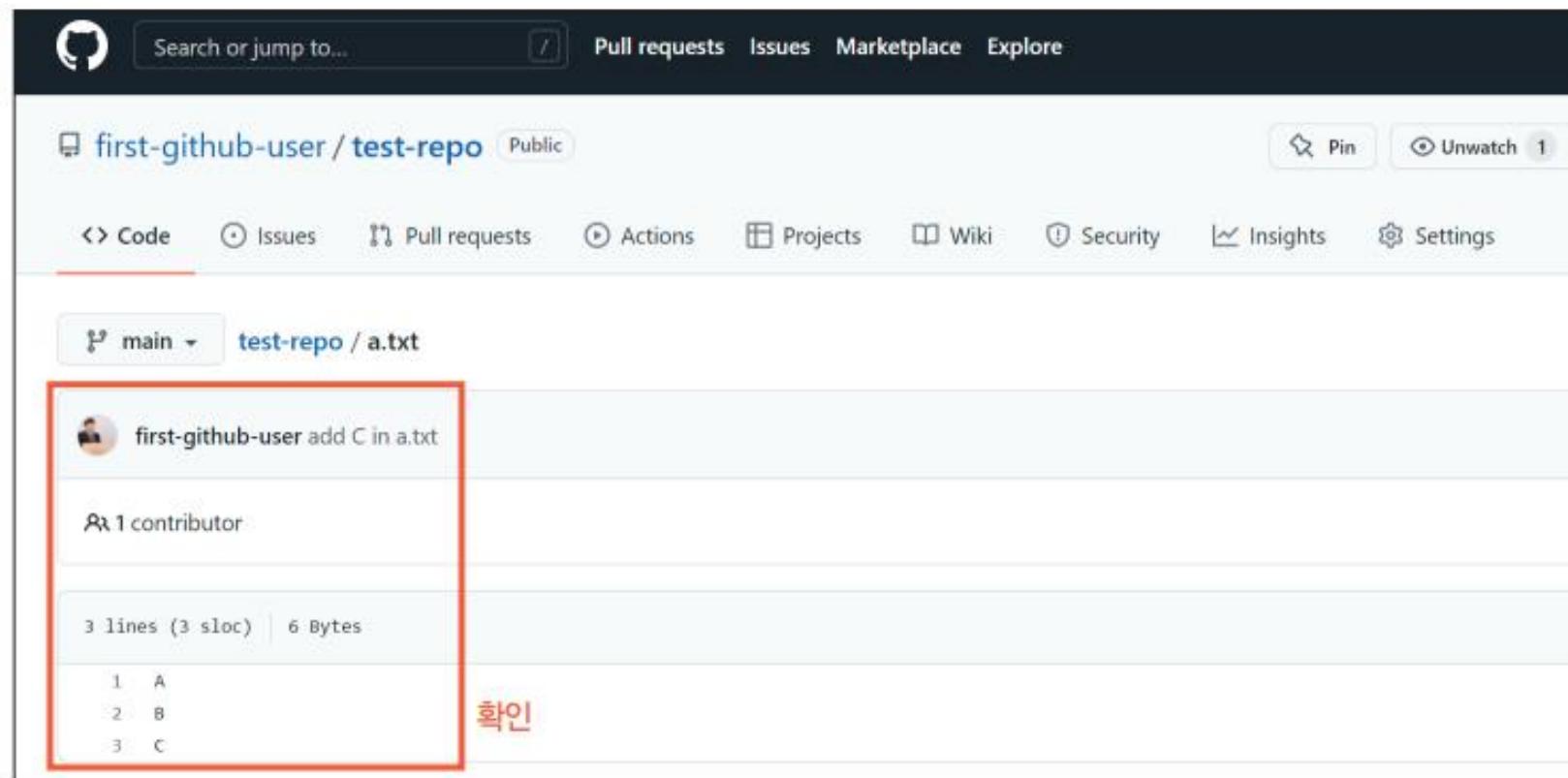


# 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- 6 자, 이렇게 문자 C를 추가하는 새로운 커밋이 추가

▼ 그림 8-24 | 새로운 커밋이 추가된 모습



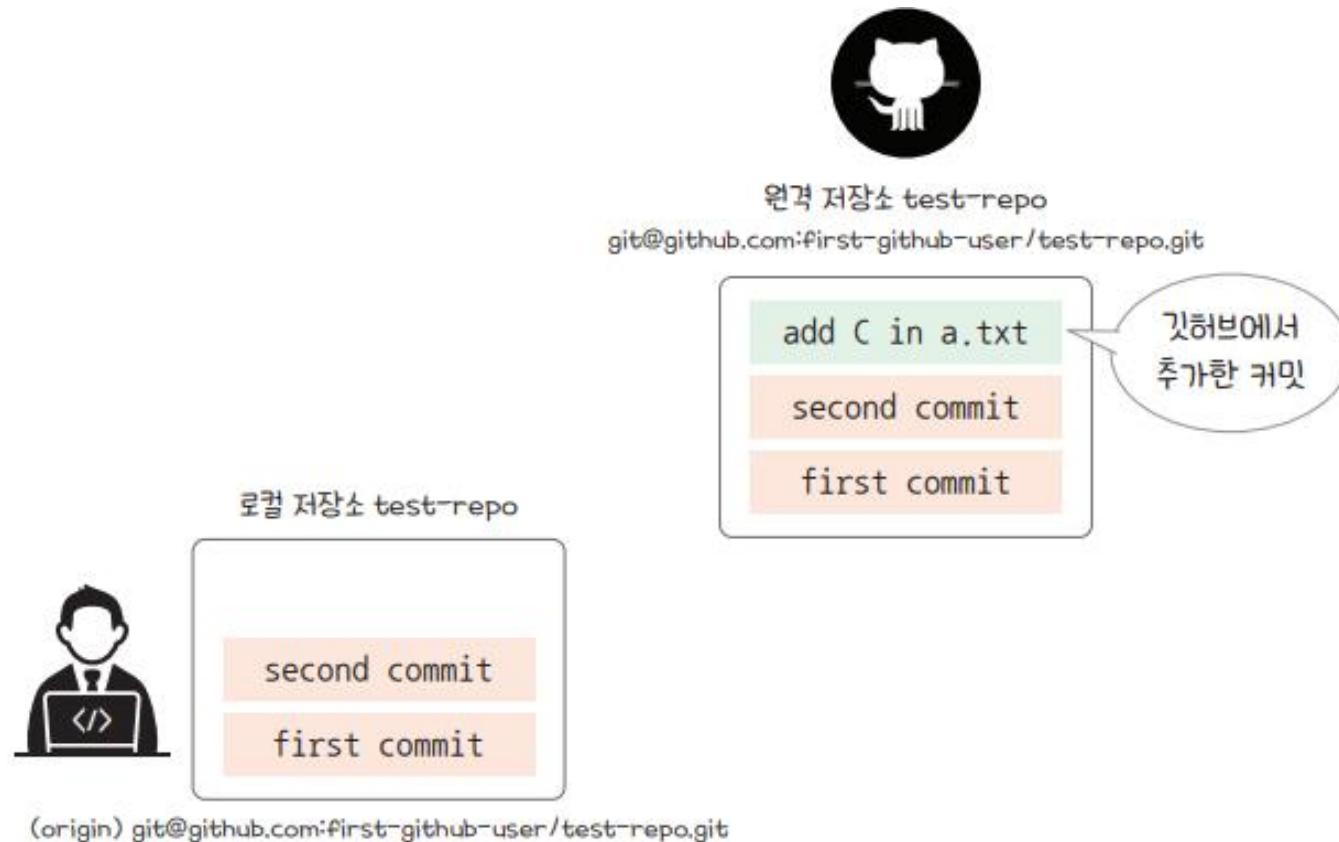
## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- 현재까지의 상황을 그림으로 표현하면 그림 8-25와 같음
- 로컬 저장소에는 커밋이 두 개 쌓여 있지만, 원격 저장소에는 세 개가 쌓여 있음

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-25 | 현재 로컬 저장소와 원격 저장소의 상황



## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- 7 이런 상황에서 패치하면 어떻게 될까?
- 패치하는 명령은 git fetch 또는 git fetch <원격 저장소 이름>
- git fetch를 입력해 보자

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 617 bytes | 25.00 KiB/s, done.
From github.com:first-github-user/test-repo
  1ac7f2f..644b54a  main      -> origin/main
```

## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- ⑧ git status를 입력해 보자
- 잘 읽어보면 Your branch is behind 'origin/main' by 1 commit이라는 메시지를 볼 수 있음
- 현재 브랜치는 origin/main 브랜치에 비해 커밋 하나가 뒤쳐진다는 의미
- 달리 말해, origin/main 브랜치는 현재 main 브랜치에 비해 한 커밋 앞서 있다는 의미

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git status
On branch main                                         확인
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)

nothing to commit, working tree clean
```

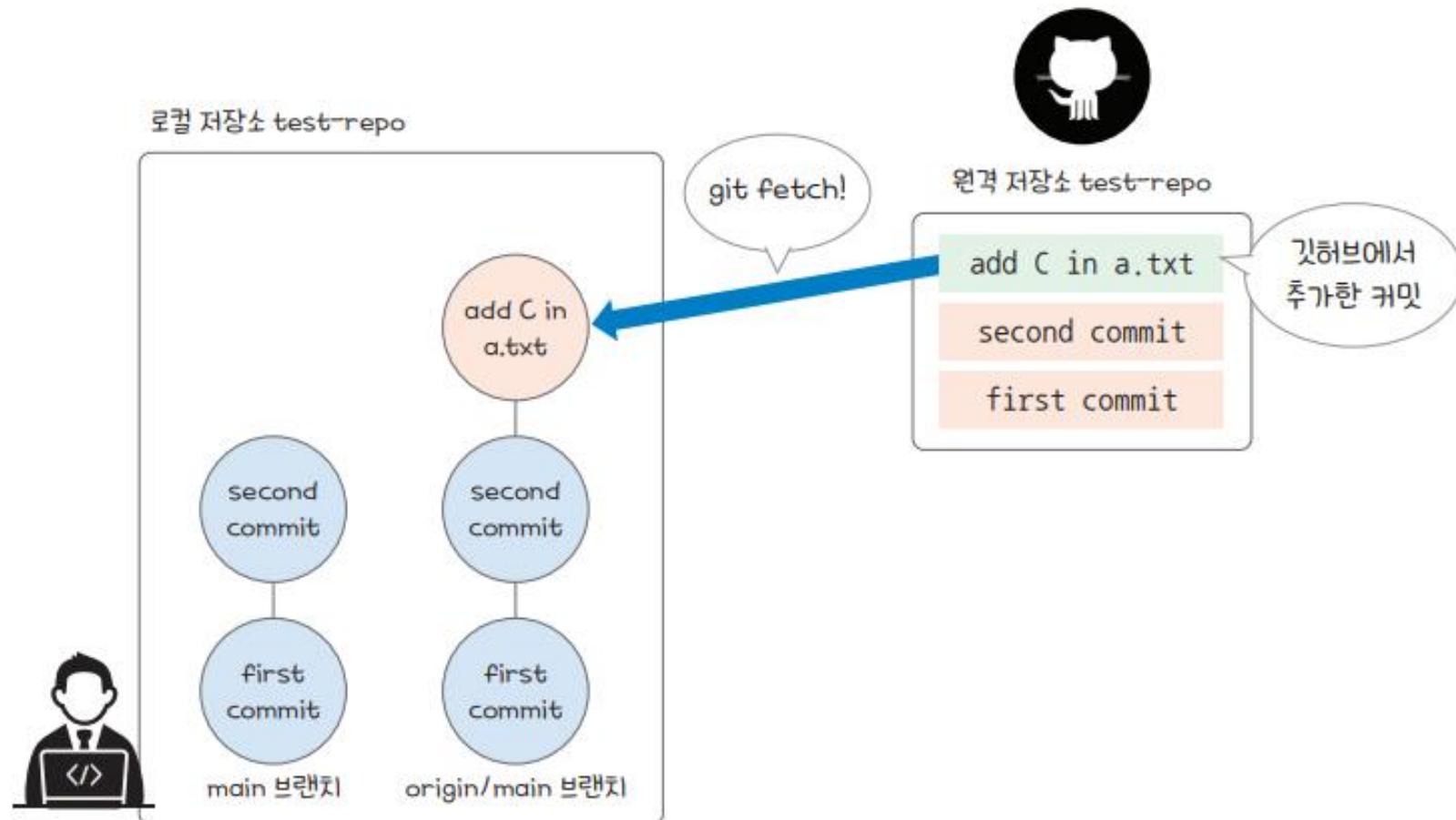
## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- 패치 직후 main 브랜치가 origin/main 브랜치에 비해 커밋 하나가 뒤쳐지는 이유는 패치는 원격 저장소의 변경 사항을 origin/main 브랜치로 가져올 뿐 main 브랜치는 변함이 없기 때문임
- 즉, 패치는 원격 저장소의 변경 사항을 ‘일단 가지고 올 뿐’ 로컬 저장소의 브랜치로 병합하지는 않는다는 걸 알 수 있음

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-26 | git fetch 직후 main 브랜치는 한 커밋 뒤쳐짐



# 8.1 원격 저장소와 상호 작용하기

## » git fetch: 원격 저장소를 일단 가져만 오기

- ⑨ 6장에서 브랜치 간 차이를 비교하는 명령은 git diff라고 했음
- git diff main origin/main 명령으로 로컬 저장소의 main 브랜치와 원격 저장소 origin의 main 브랜치를 비교해 보자
- 깃허브에서 추가한 변경 사항, 즉 +C가 보인다면 성공

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git diff main origin/main
diff --git a/a.txt b/a.txt
index 35d242b..b1e6722 100644
--- a/a.txt
+++ b/a.txt
@@ -1,2 +1,3 @@
A
B
+C 확인
```

## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- ⑩ 그럼 원격 저장소의 브랜치로 체크아웃해 보자
- git checkout origin/main 명령을 입력

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git checkout origin/main ..... origin/main 브랜치로 체크아웃
Note: switching to 'origin/main'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

```
Or undo this operation with:
```

```
git switch -
```

```
Turn off this advice by setting config variable advice.detachedHead to false
```

```
HEAD is now at 644b54a add C in a.txt
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

패치한 결과를 가리키는 FETCH\_HEAD

- git checkout FETCH\_HEAD 명령으로도 체크아웃할 수 있음
- FETCH\_HEAD은 최근에 패치한 브랜치의 최신 커밋을 가리키기 때문임
- 이 예시의 경우 FETCH\_HEAD은 깃허브에서 추가한 커밋(add C in a.txt)을 가리킴

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git checkout FETCH_HEAD
Note: switching to 'origin/main'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

```
git switch -c <new-branch-name>
```

Or undo this operation with:

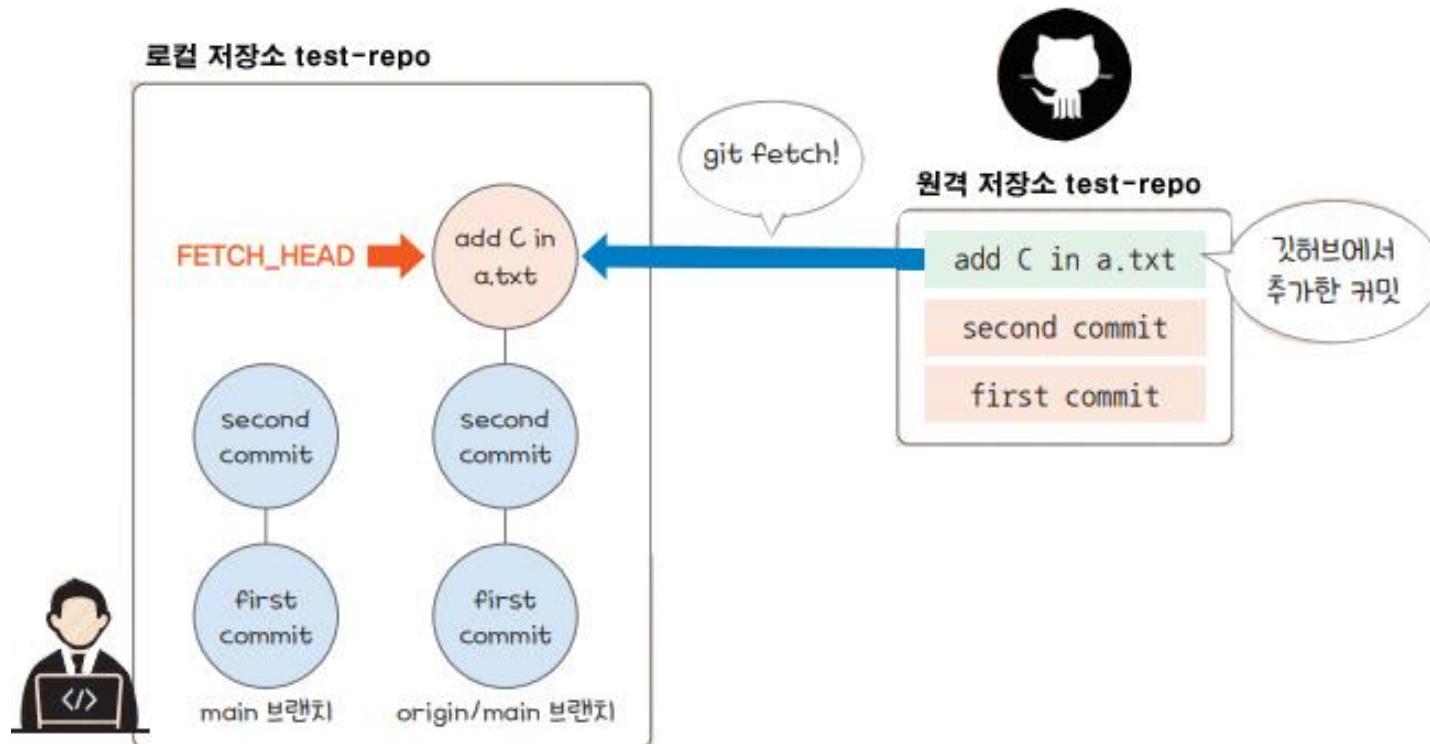
```
git switch -
```

Turn off this advice by setting config variable advice.detachedHead to false

```
HEAD is now at 644b54a add C in a.txt
```

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-27 | FETCH\_HEAD가 가리키는 커밋



## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ⑪ git log로 origin/main의 커밋 목록을 보자
- 깃허브에서 추가한 세 번째 커밋을 확인할 수 있음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo ((644b54a...))
$ git log
commit 644b54adb03696add430eb1b7d71ef60f2a7abaf (HEAD, origin/main) 확인
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:48:34 2022 +0900

    add C in a.txt
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

```
commit 1ac7f2fe1bdbbb8f737af76ff0b502286c476f43 (main)
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:44 2022 +0900
```

second commit

```
commit abf6d7de46d0585bc02e293ad1657f0bcea2b945
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:28 2022 +0900
```

first commit

## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- ⑫ 자, 이제 패치한 결과를 로컬 저장소의 main 브랜치로 병합해 보자
- 다시 main 브랜치로 체크아웃

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo ((644b54a...))
$ git checkout main
Previous HEAD position was 644b54a add C in a.txt
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ⑯ git merge origin/main 명령으로 원격 저장소 origin/main 브랜치를 로컬 저장소의 main 브랜치로 병합

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git merge origin/main
Updating 1ac7f2f..644b54a
Fast-forward
 a.txt | 1 +
 1 file changed, 1 insertion(+)
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- git merge FETCH\_HEAD 명령을 이용해도 무방함

## 8.1 원격 저장소와 상호 작용하기

### » git fetch: 원격 저장소를 일단 가져만 오기

- 14 성공적으로 병합했다면 git log 명령 결과는 다음과 같음
- 깃허브에서 추가한 세 번째 커밋을 main 브랜치로 병합

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git log
commit 644b54adb03696add430eb1b7d71ef60f2a7abaf (HEAD -> main, origin/main)
Author: Kang Minchul <tegongkang@gmail.com>           main 브랜치로 병합
Date:   Thu Mar 24 02:48:34 2022 +0900

    add C in a.txt
```

## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

```
commit 1ac7f2fe1bdbbb8f737af76ff0b502286c476f43
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:44:44 2022 +0900
```

second commit

```
commit abf6d7de46d0585bc02e293ad1657f0bcea2b945
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:44:28 2022 +0900
```

first commit

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$
```

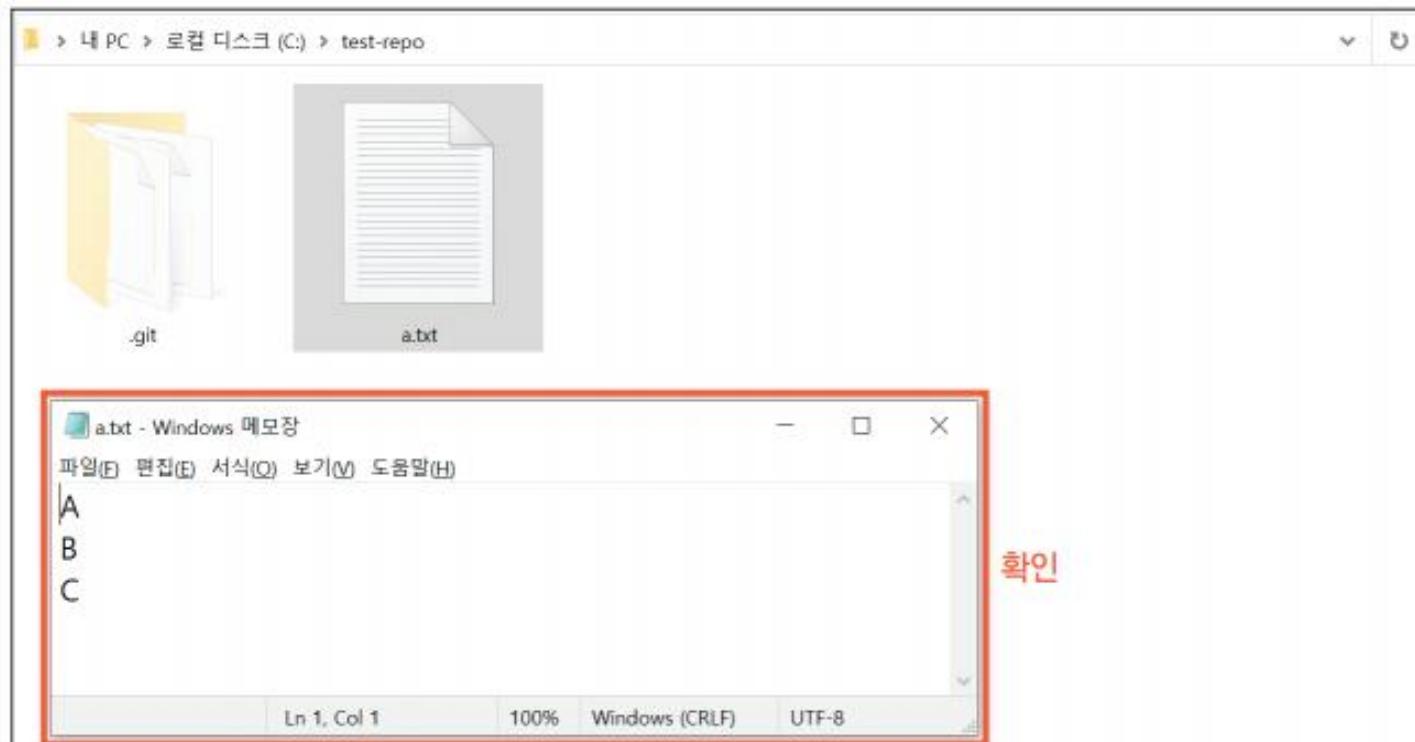
## 8.1 원격 저장소와 상호 작용하기

» git fetch: 원격 저장소를 일단 가져만 오기

- ⑯ a.txt 파일을 직접 열어 깃허브에서 만든 변경 사항을 확인해 보자
- 잘 반영됨

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-28 | 병합 결과 확인하기



# 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- 원격 저장소와의 마지막 상호 작용, 풀에 대해 학습해 보자
- 풀은 앞서 설명한 패치와 병합을 동시에 하는 방식
- 어려운 개념이 아닌 만큼, 바로 실습해 보자

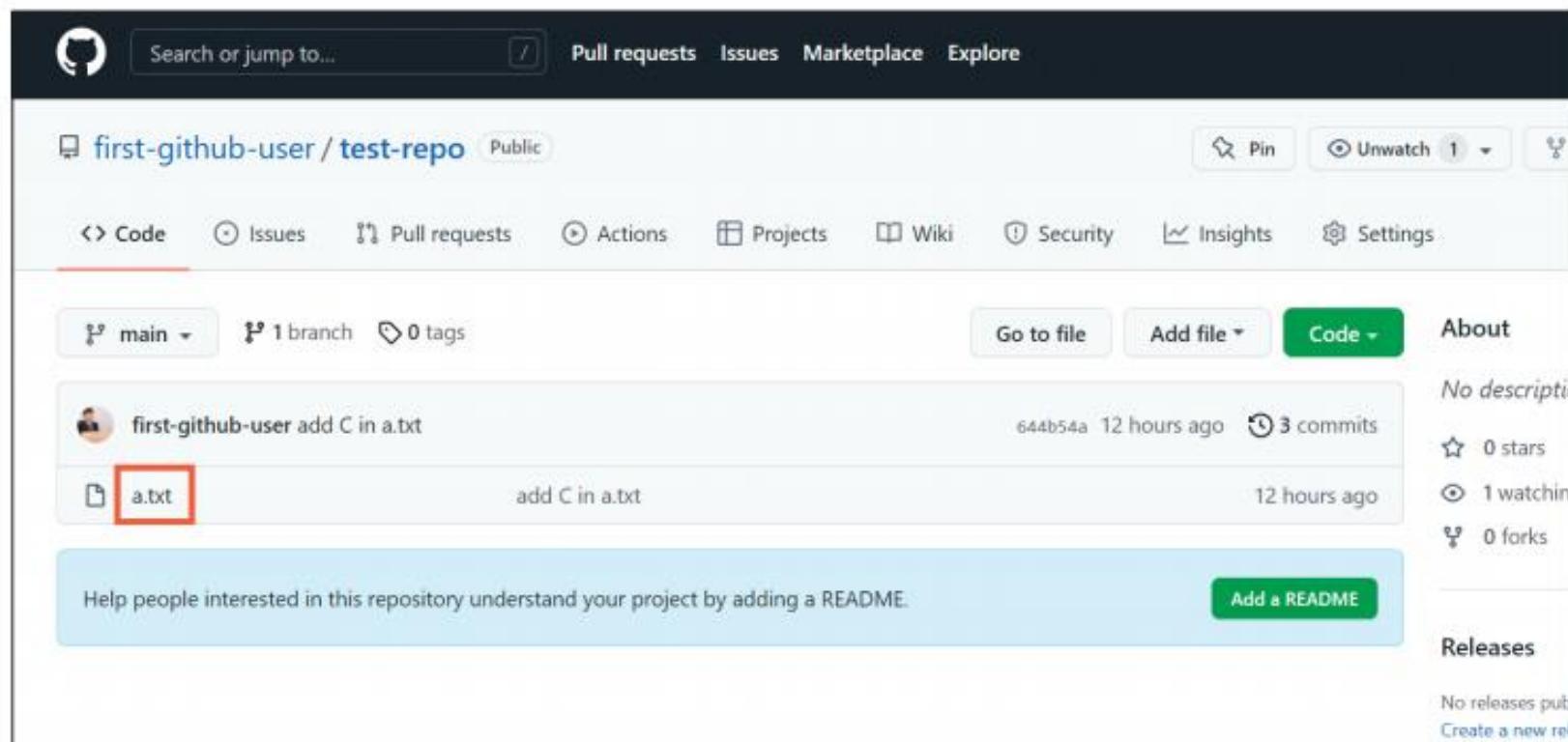
## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- ① 깃허브에서 새로운 커밋을 추가
- 깃허브에서 a.txt 파일을 클릭해 보자

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-29 | a.txt 파일 클릭하기

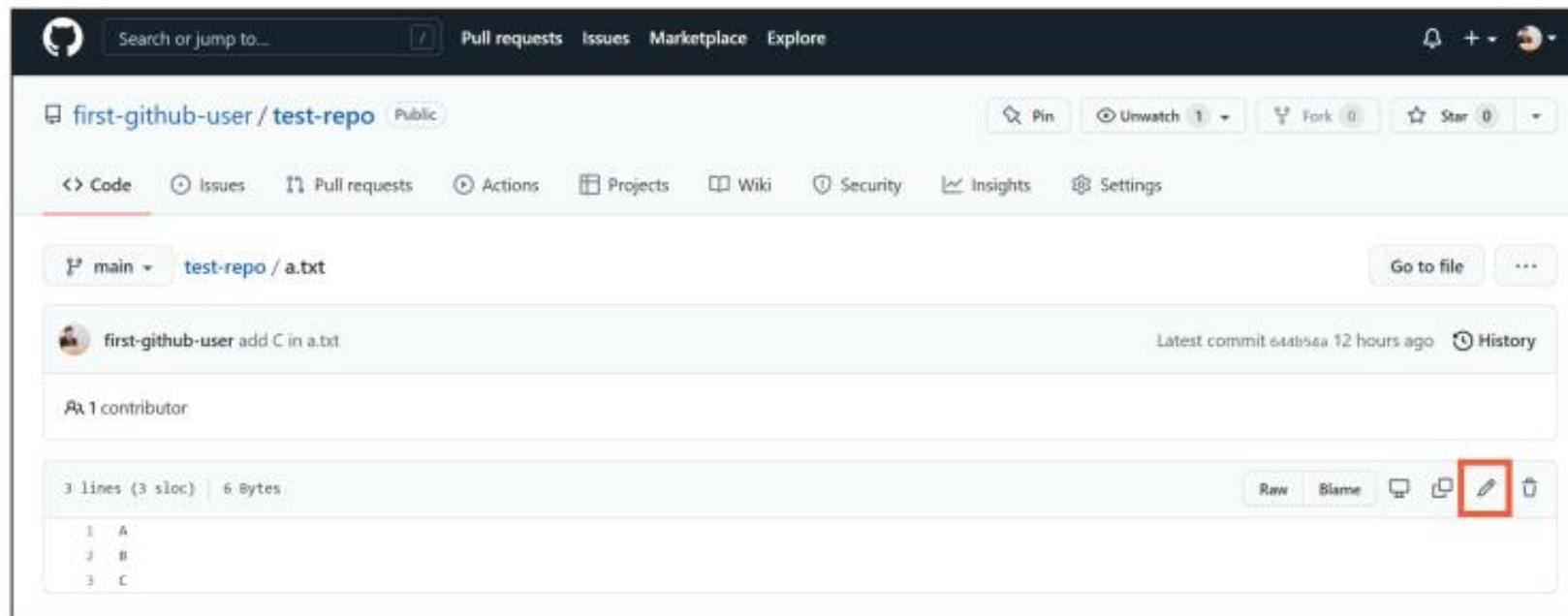


# 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- ② 파일을 깃허브에서 직접 수정하기 위해 연필 아이콘을 클릭

▼ 그림 8-30 | 연필 아이콘 클릭하기



# 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- ③ 마지막 줄에 다음과 같이 문자 D를 추가

▼ 그림 8-31 | 문자 D 추가하기

The screenshot shows a GitHub repository named 'first-github-user/test-repo'. The 'Code' tab is selected. In the 'a.txt' file, the content is:

```
1 A
2 B
3 C
4 D
5 |
```

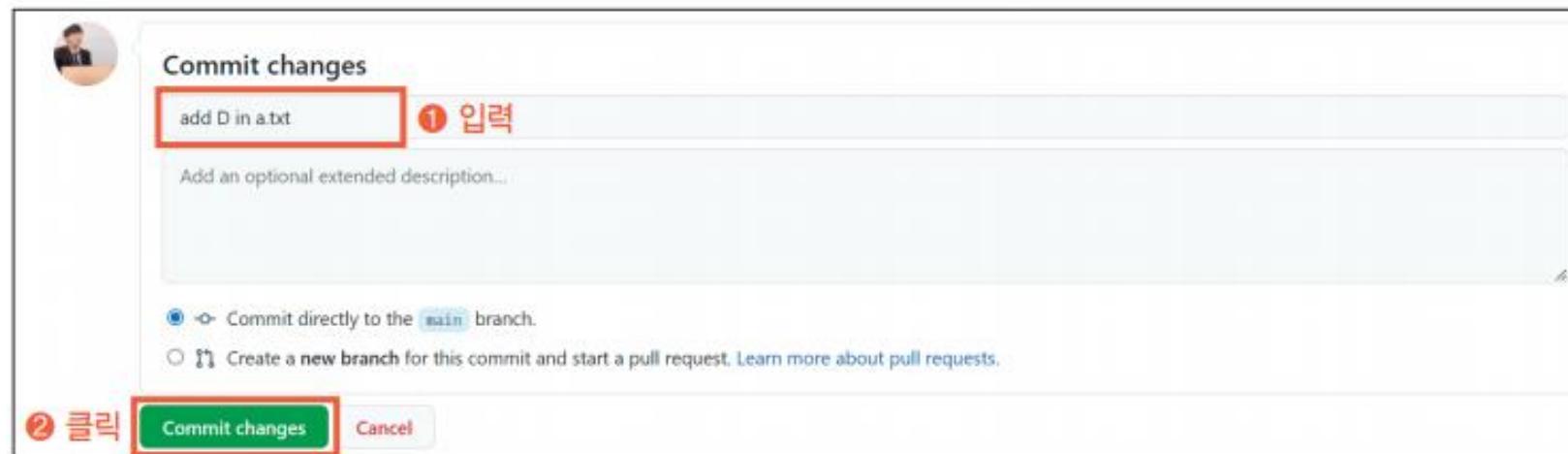
The line '4 D' is highlighted with a red box, and the word '입력' is written next to it in red, indicating where the user should input the character 'D'.

# 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- ④ 스크롤을 내려 커밋
- 커밋 메시지로 add D in a.txt를 입력하고 Commit changes를 클릭

▼ 그림 8-32 | 커밋 메시지 입력 후 커밋하기



# 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- 5 이렇게 문자 D를 추가하는 새로운 커밋을 추가

▼ 그림 8-33 | 새로운 커밋이 추가된 모습

The screenshot shows a GitHub repository page for 'first-github-user/test-repo'. The 'Code' tab is selected. A commit history is displayed for the 'main' branch, showing a single commit from 'first-github-user' with the message 'add D in a.txt'. The commit shows four lines of text: 'A', 'B', 'C', and 'D'. The line 'D' is highlighted with a red box and the word '확인' (Check) is written next to it.

first-github-user / test-repo Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main test-repo / a.txt

first-github-user add D in a.txt

1 contributor

4 lines (4 sloc) 8 Bytes

1	A
2	B
3	C
4	D 확인

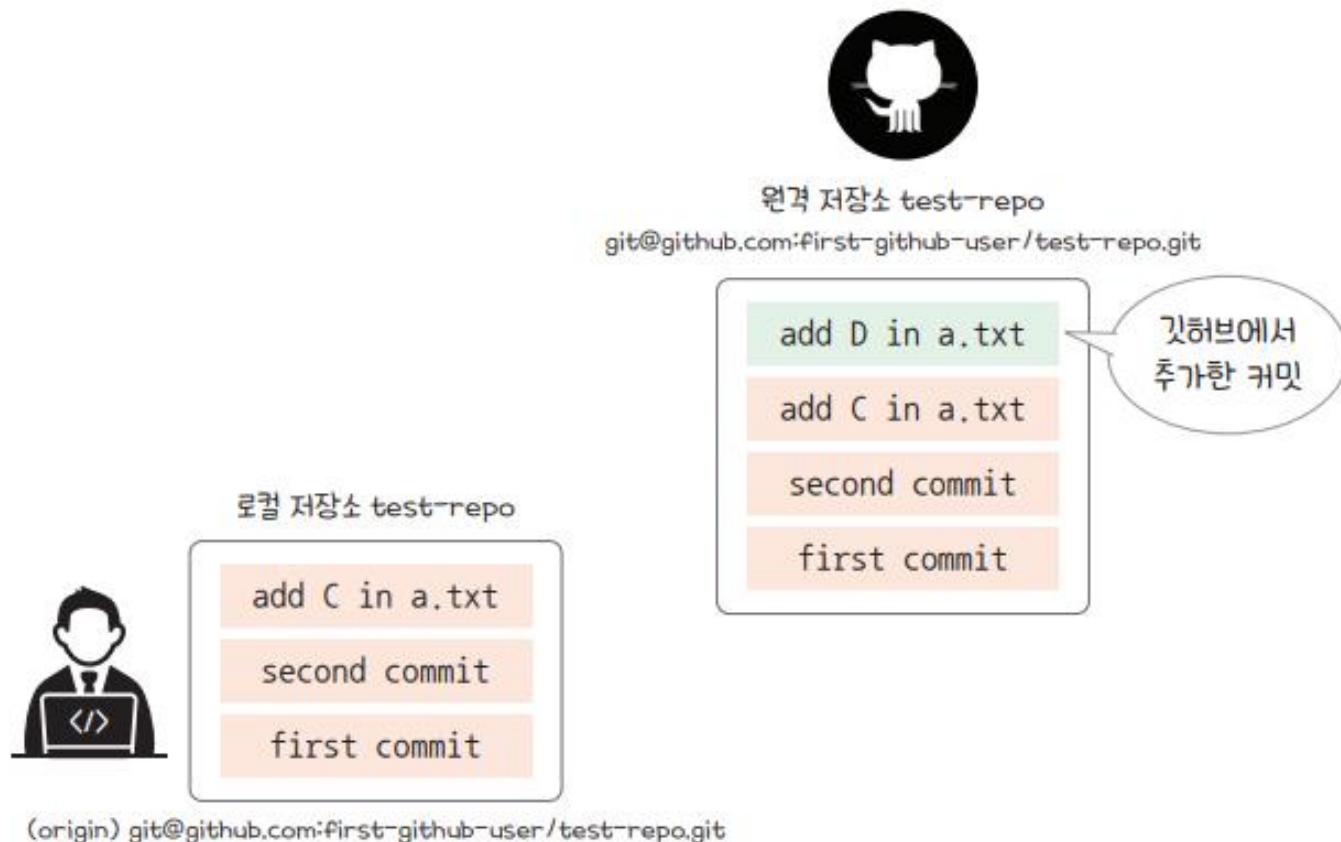
## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- 현재 상황 그림으로 표현하면 다음과 같음
- 원격 저장소에는 로컬 저장소에는 없는 커밋을 추가

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-34 | 현재 로컬 저장소와 원격 저장소의 상황



## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- ⑥ 원격 저장소의 변경 사항을 풀하기 전에 git log 명령으로 현재 로컬 저장소의 커밋 목록을 확인해 보자

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git log
commit 644b54adb03696add430eb1b7d71ef60f2a7abaf (HEAD -> main, origin/main)
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:48:34 2022 +0900

    add C in a.txt
```

## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

```
commit 1ac7f2fe1bdbbb8f737af76ff0b502286c476f43
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:44 2022 +0900
```

```
second commit
```

```
commit abf6d7de46d0585bc02e293ad1657f0bcea2b945
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:28 2022 +0900
```

```
first commit
```

# 8.1 원격 저장소와 상호 작용하기

## » git pull: 원격 저장소를 가져와서 합치기

- 7 자, 이제 원격 저장소의 변경 사항을 풀 해보자
- 명령은 git pull 또는 git pull<원격 저장소 이름>
- git pull을 입력해 보자

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 617 bytes | 30.00 KiB/s, done.
From github.com:first-github-user/test-repo
  644b54a..5bf1925  main      -> origin/main
Updating 644b54a..5bf1925
Fast-forward
  a.txt | 1 +
    1 file changed, 1 insertion(+)
```

## 8.1 원격 저장소와 상호 작용하기

### » git pull: 원격 저장소를 가져와서 합치기

- ⑧ 다시 git log 명령으로 로컬 저장소의 커밋 목록을 확인해 보자
- 앞서 깃허브에서 추가한 커밋이 main 브랜치에 병합된 것을 확인할 수 있음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test-repo (main)
$ git log
commit 5bf19256408ff3329432e03b2f09012a0a588e23 (HEAD -> main, origin/main)
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 14:50:11 2022 +0900

    add D in a.txt
```

확인

```
commit 644b54adb03696add430eb1b7d71ef60f2a7abaf
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Thu Mar 24 02:48:34 2022 +0900
```

```
    add C in a.txt
```

## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

```
commit 1ac7f2fe1bdbbb8f737af76ff0b502286c476f43
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:44 2022 +0900
```

second commit

```
commit abf6d7de46d0585bc02e293ad1657f0bcea2b945
```

```
Author: Kang Minchul <tegongkang@gmail.com>
```

```
Date: Thu Mar 24 02:44:28 2022 +0900
```

first commit

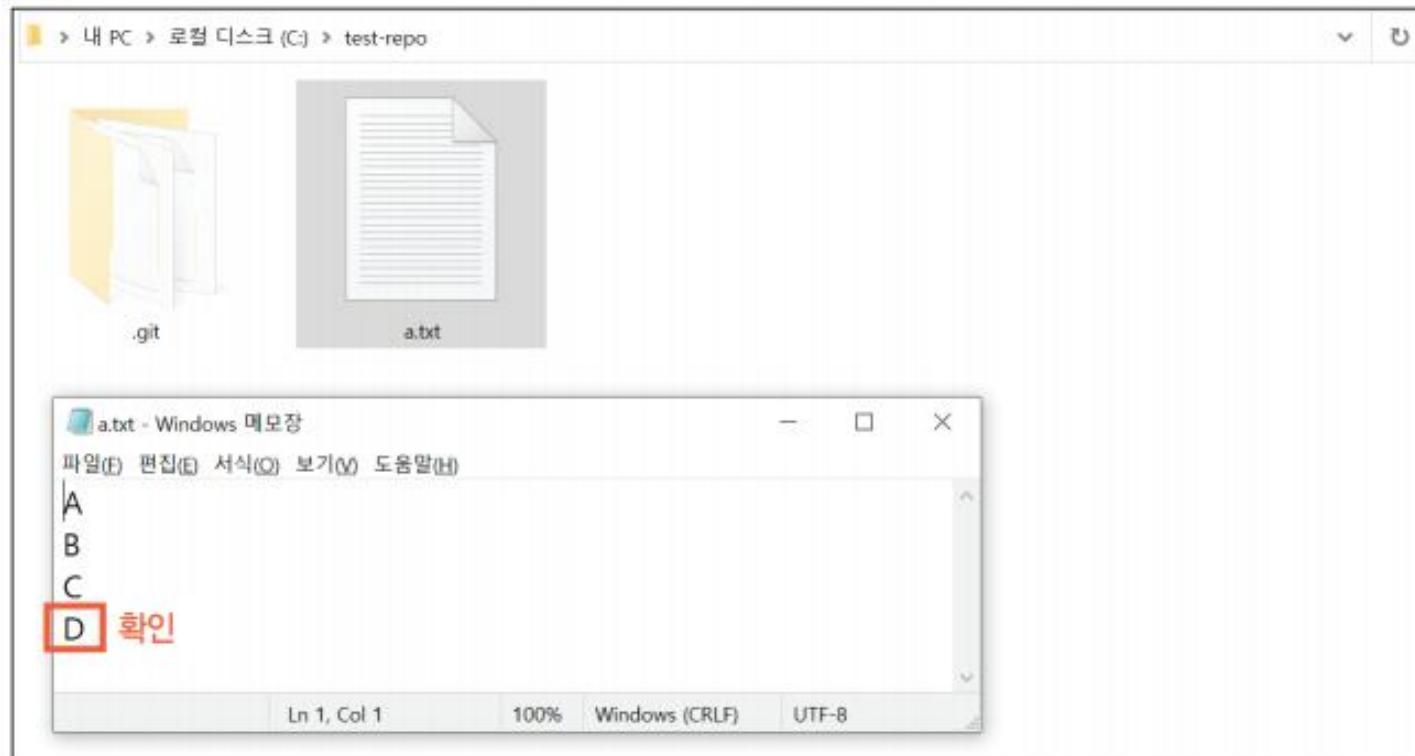
## 8.1 원격 저장소와 상호 작용하기

» git pull: 원격 저장소를 가져와서 합치기

- 9 직접 a.txt 파일을 확인해 보자
- 깃허브에서 추가한 문자 D를 확인할 수 있음

# 8.1 원격 저장소와 상호 작용하기

▼ 그림 8-35 | 풀 결과 확인하기



## 8.1 원격 저장소와 상호 작용하기

### » git pull: 원격 저장소를 가져와서 합치기

- 즉, 풀은 원격 저장소의 변경 사항을 로컬 저장소로 가져옴과 동시에 병합까지 해주는 방식임을 알 수 있음
- 다시 말해, 풀은 패치와 달리 원격 저장소를 ‘가져와서 합치는’ 방식이라는 점을 알 수 있음

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 5장에서 풀 리퀘스트를 통해 협업하는 방법을 학습
- 풀 리퀘스트를 보내는 과정은 다음과 같았음

1 | 기여하려는 저장소를 본인 계정으로 포크하기

2 | 포크한 저장소를 클론하기

3 | 브랜치 생성 후 생성한 브랜치에서 작업하기

4 | 작업한 브랜치 푸시하기

5 | 풀 리퀘스트 보내기

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 깃 명령으로 풀 리퀘스트를 보내는 방법도 이와 다르지 않음
- 이 과정을 그대로 따라 하면 됨
- 이 절에서는 5장의 풀 리퀘스트 실습을 명령어로 다시 한번 해보자

## 8.2 깃 명령으로 풀 리퀘스트 보내기

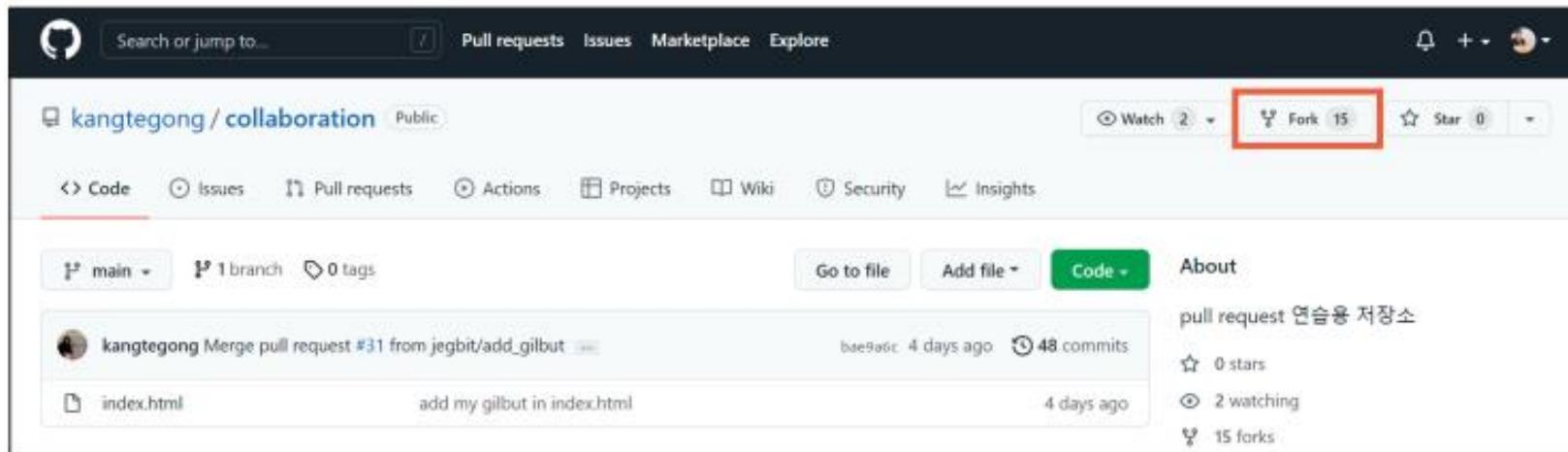
### » 깃 명령으로 풀 리퀘스트 보내기

- ① 풀 리퀘스트의 첫 번째 단계는 ‘기여하려는 저장소를 본인 계정으로 포크하기’
- 5장과 마찬가지로 kangtegong 계정의 collaboration 저장소로 풀 리퀘스트를 보낼 예정이므로 <https://github.com/kangtegong/collaboration>에 접속한 후 Fork를 눌러보자

URL <https://github.com/kangtegong/collaboration>

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-36 | 리퀘스트를 보낼 저장소 포크하기



## 8.2 깃 명령으로 풀 리퀘스트 보내기

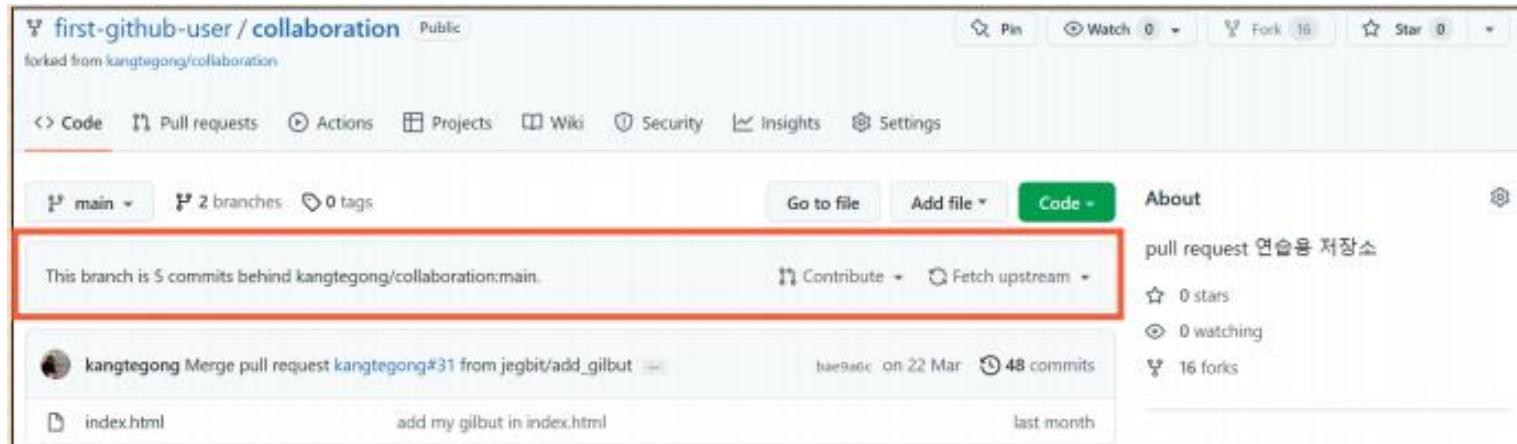
### » 깃 명령으로 풀 리퀘스트 보내기

만일 이미 collaboration 저장소가 포크되어 있다면?

- 5장에서 이미 collaboration 저장소를 여러분의 계정으로 포크했을 경우, 포크한 저장소를 클론하기 전에 잊지 말고 확인해야 하는 단계가 있음
- 여러분의 계정으로 포크된 저장소로 들어가보면, 다음 그림 속 박스처럼 This branch is X commits behind kangtegong/collaboration:main 메시지가 떠 있을 수 있음

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-37 | 원본 저장소에 비해 뒤쳐진 포크된 저장소

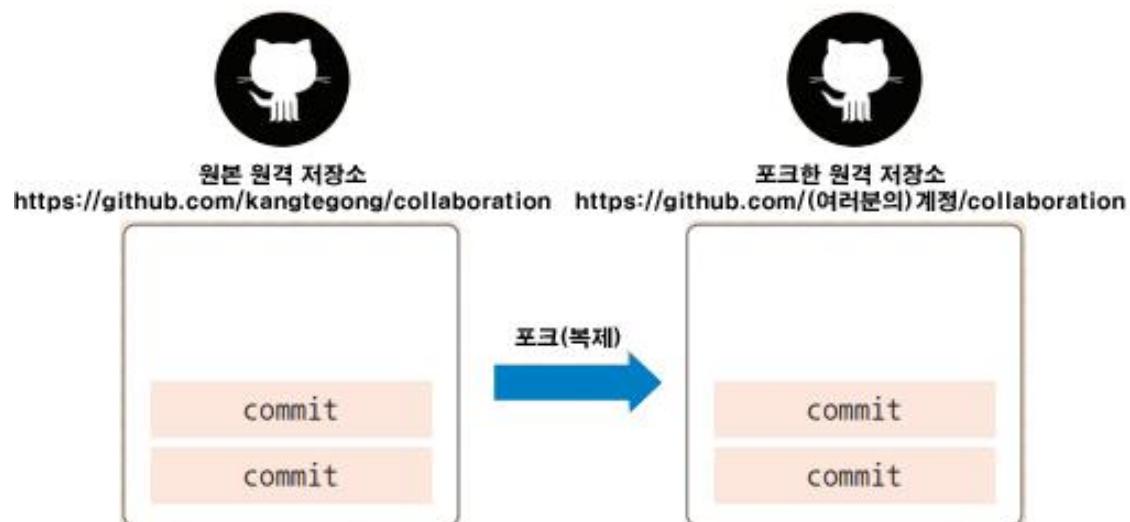


## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 이 메시지는 포크된 저장소가 원본 저장소에 비해 몇 커밋 뒤쳐져 있다는 것을 의미
- 포크는 여러분의 계정으로 원격 저장소를 복제하는 것이라 했음

▼ 그림 8-38 | 원격 저장소를 복제하는 포크



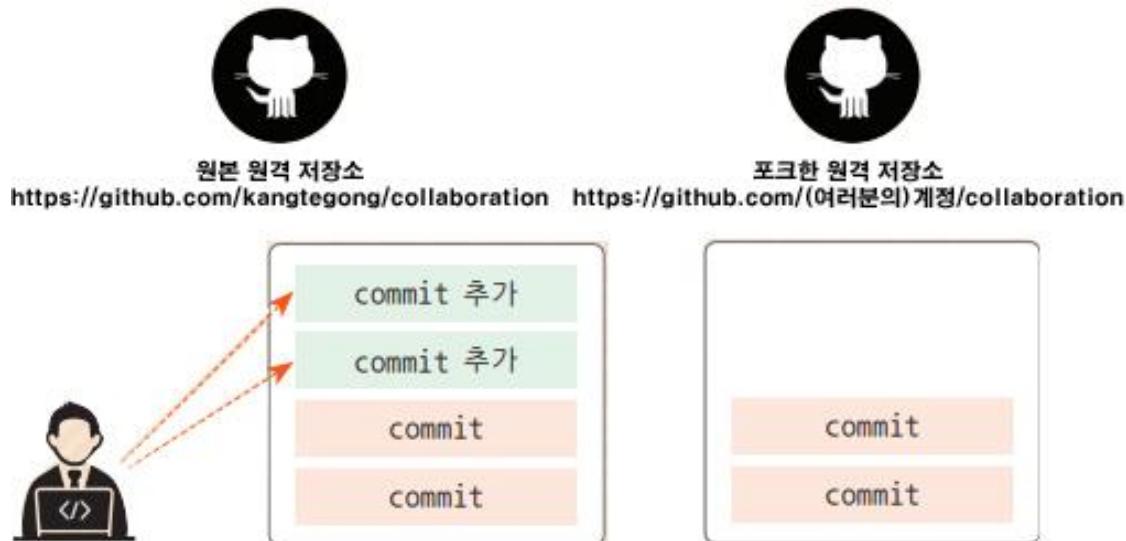
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 여러분의 계정으로 원격 저장소를 포크(복제)한 이후로도 원래의 원격 저장소(kangtegong/collaboration)에는 계속해서 새로운 커밋이 쌓일 수 있음
- 이 경우 여러분의 계정으로 포크한 원격 저장소는 원본 저장소에 비해 몇 커밋 뒤쳐지게 됨

## 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-39 | 원본 저장소에 비해 뒤쳐지게 되는 포크된 저장소



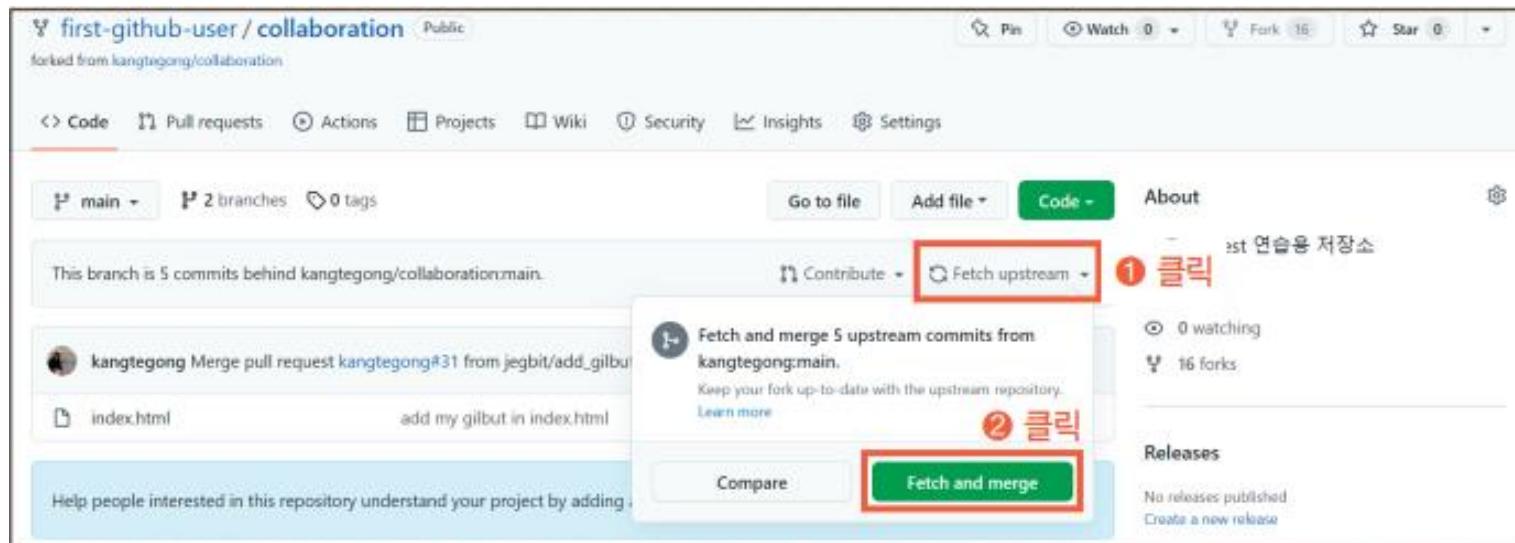
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 이 경우 여러분의 계정으로 포크한 원격 저장소가 원본 저장소에 비해 뒤쳐지지 않도록 맞춰주어야 함
- 이는 Fetch upstream을 클릭한 후 Fetch and merge를 클릭
- 참고로 upstream은 포크했던 원본 저장소, 이 예제의 경우에는 kangtegong 계정의 collaboration 저장소를 의미

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-40 | ‘Fetch and merge’ 클릭하기



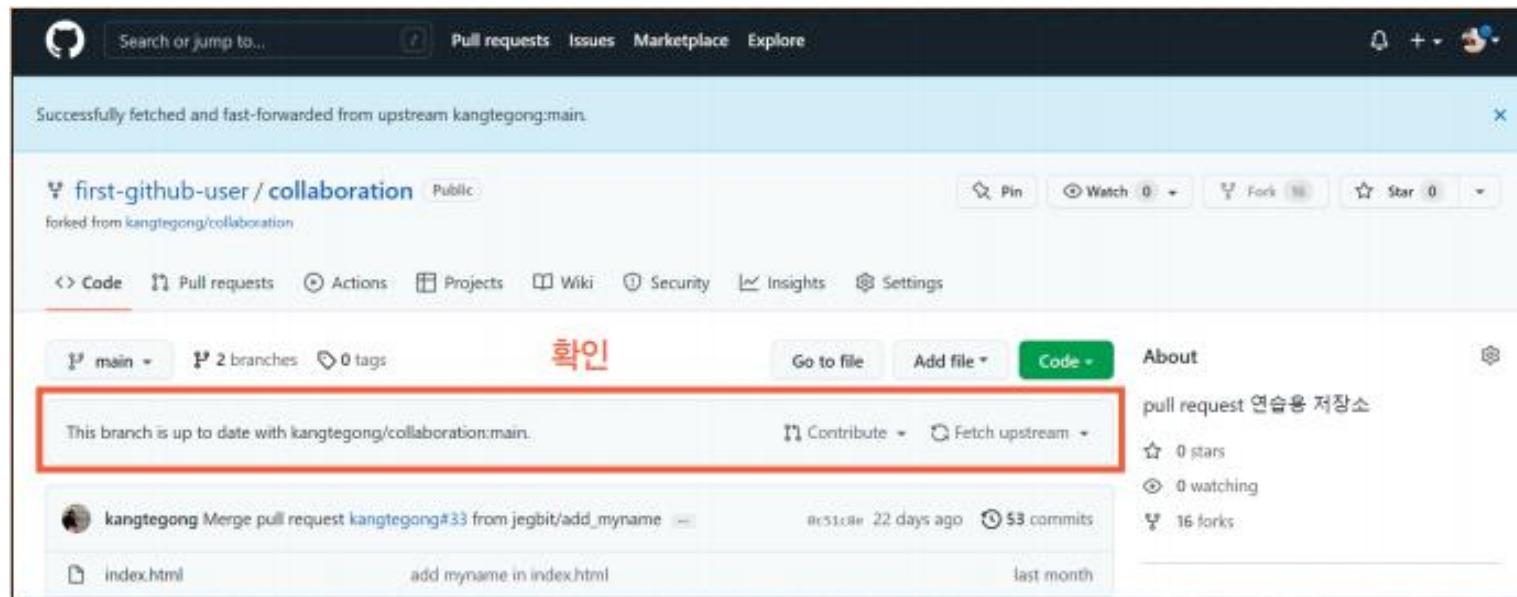
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- This branch is up to date with kangtegong/collaboration:main 메시지가 뜣다면 여러분의 계정으로 포크한 저장소와 kangtegong 계정의 collaboration이 동일하게 맞춰진 것
- 이제 포크한 여러분의 저장소를 클론받으면 됨

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-41 | 원본 저장소의 동일하게 맞춰진 포크된 저장소



# 8.2 깃 명령으로 풀 리퀘스트 보내기

## » 깃 명령으로 풀 리퀘스트 보내기

- ② 포크된 저장소의 모습
- 이제 이 저장소를 여러분의 컴퓨터로 클론
- 클론하는 방법은 앞선 절에서 설명했음
- Code를 클릭하고 SSH를 클릭한 후 원격 저장소 경로를 복사

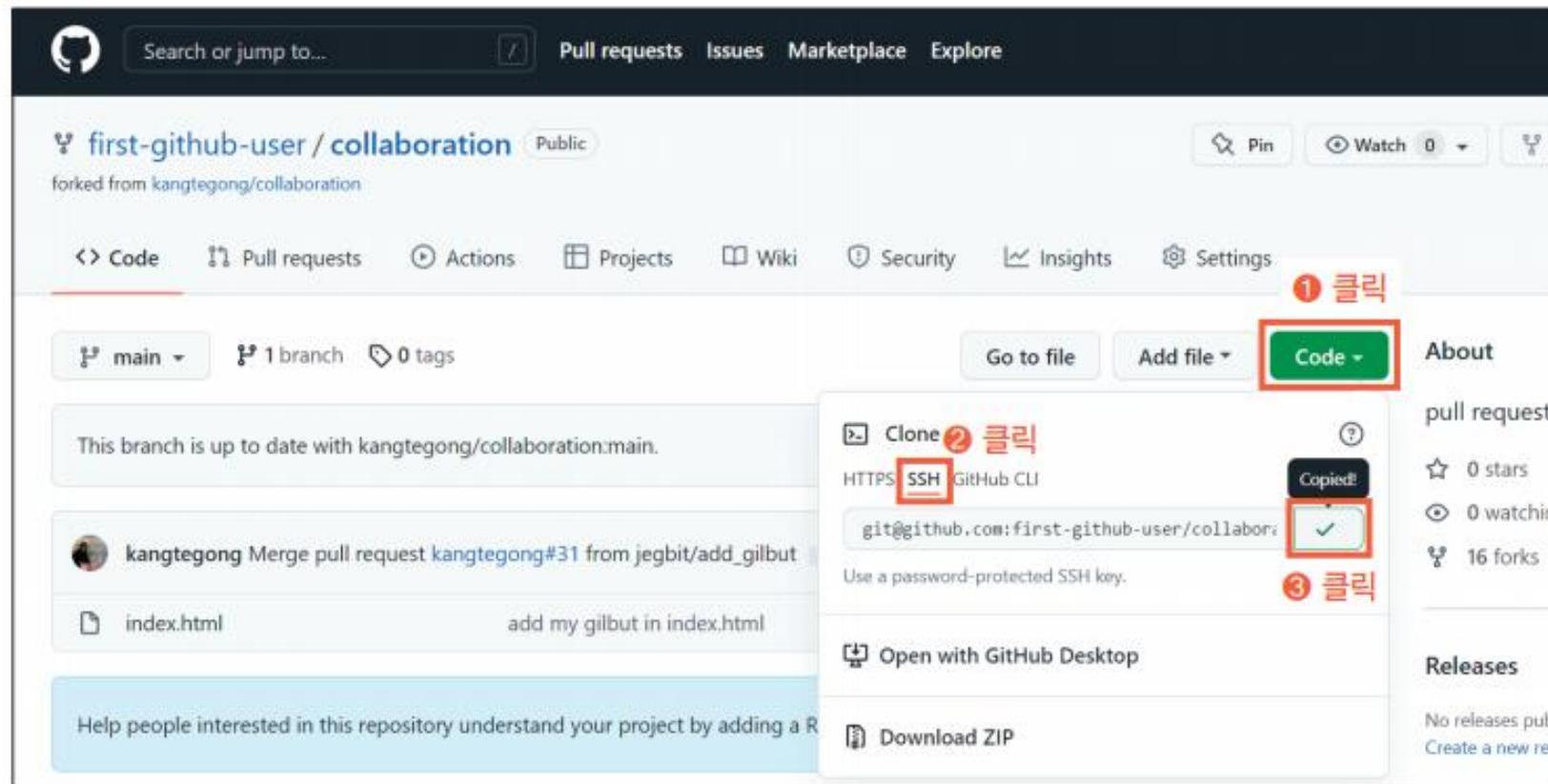
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 이때 kangtegong 계정의 collaboration 저장소가 아닌 여러분 계정으로 포크한 collaboration 저장소를 클론

## 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-42 | 포크된 저장소 클론하기



# 8.2 깃 명령으로 풀 리퀘스트 보내기

## » 깃 명령으로 풀 리퀘스트 보내기

- ③ 클론받으려는 경로에서 깃 배시를 열어주자
- 여기에서는 C:\ 경로에 클론받기 위해 이 경로에서 깃 배시를 열겠음
- 그런 다음 git clone을 입력한 후 복사한 경로를 붙여 넣음
- 원격 저장소를 클론하는 명령은 git clone <원격 저장소>였음

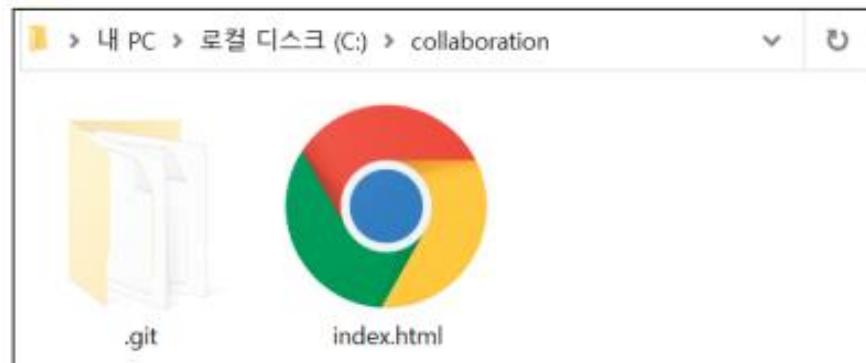
```
minchul@DESKTOP-9KULGUE MINGW64 /c
$ git clone git@github.com:first-github-user/collaboration.git
Cloning into 'collaboration'...
remote: Enumerating objects: 106, done.
remote: Counting objects: 100% (106/106), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 106 (delta 17), reused 67 (delta 14), pack-reused 0
Receiving objects: 100% (106/106), 24.20 KiB | 136.00 KiB/s, done.
Resolving deltas: 100% (17/17), done.
```

# 8.2 깃 명령으로 풀 리퀘스트 보내기

## » 깃 명령으로 풀 리퀘스트 보내기

- ④ 자, 이렇게 포크한 원격 저장소를 여러분의 컴퓨터로 클론받았음

▼ 그림 8-43 | 클론받은 원격 저장소



## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑤ cd <경로> 명령은 <경로>로 이동하는 명령
- cd collaboration 명령을 입력하여 클론받은 원격 저장소로 이동

```
minchul@DESKTOP-9KULGUE MINGW64 /c  
$ cd collaboration/
```

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑥ 풀 리퀘스트의 세 번째 단계는 ‘브랜치 생성 후 생성된 브랜치에서 작업하기’
- 브랜치를 만드는 명령은 앞선 장에서 배운 적이 있었음
- git branch <브랜치 이름>이었음
- 브랜치 이름은 자유롭게 정해도 무방함
- 가령 add\_myname이라는 브랜치를 만들고 싶다면 git branch add\_myname 명령을 입력

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (main)
```

```
$ git branch add_myname
```

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑦ 브랜치를 만들었다면 이제 만들어진 브랜치로 체크아웃

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (main)
$ git checkout add_myname
Switched to branch 'add_myname'
```

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 브랜치를 만들고 동시에 체크아웃하는 명령도 배웠음
- git checkout -b <브랜치 이름>이었음
- 즉, git branch add\_myname 명령과 git checkout add\_myname 명령을 연이어 입력하는 대신 git checkout -b add\_myname 명령을 입력해도 됨

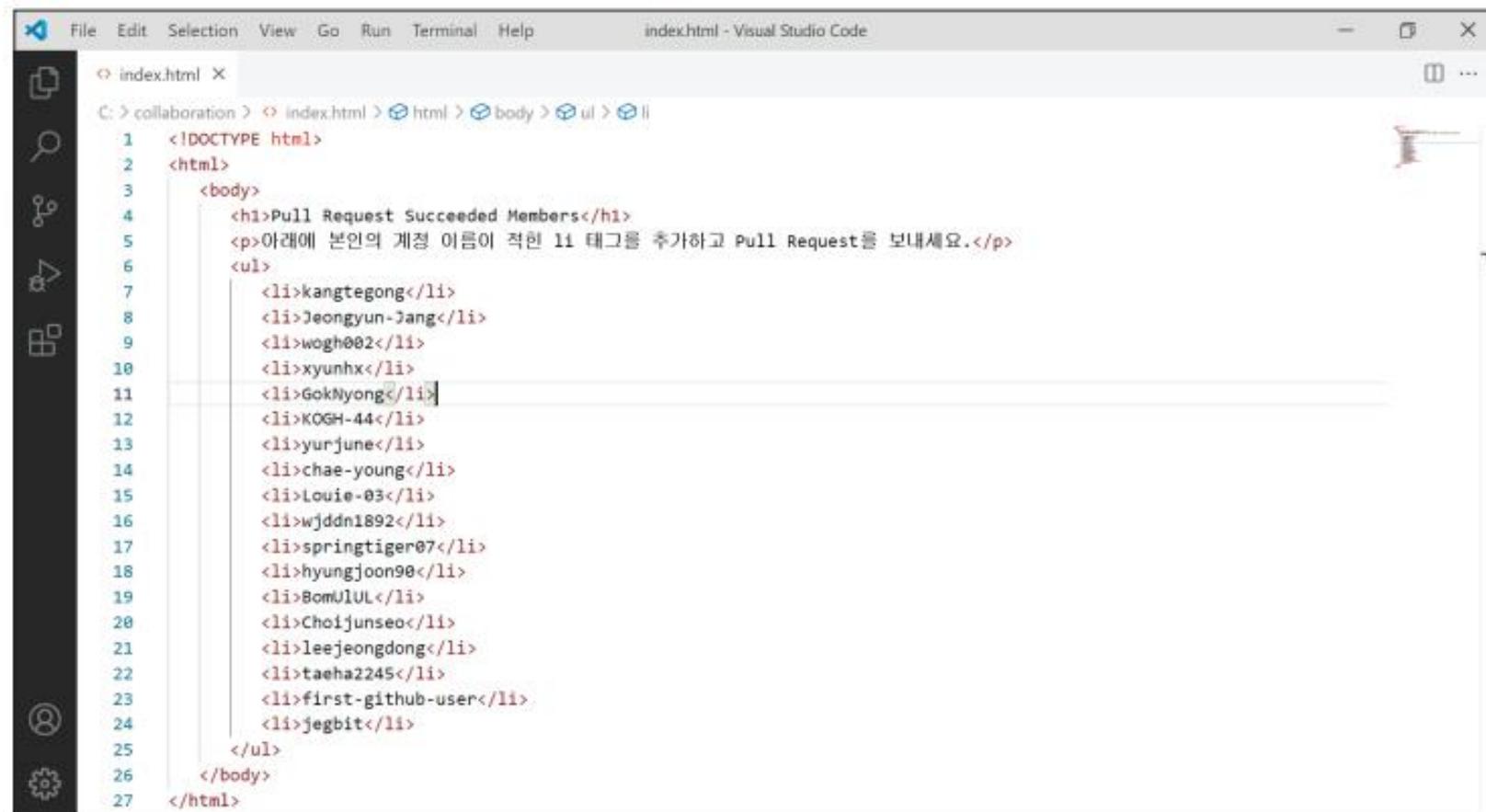
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑧ 지금까지 여러분은 포크한 저장소에서 브랜치를 만들고, 해당 브랜치로 체크아웃까지 했음
- 이제 여기서 여러분이 하려던 작업(<li> 태그로 깃허브 아이디 추가하기)을 하면 됨
- 메모장이나 코드 편집기로 index.html을 열어줌
- 현재 index.html의 모습은 다음과 같음

## 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-44 | index.html의 모습



The screenshot shows the Visual Studio Code interface with the file 'index.html' open. The code editor displays the following HTML content:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Pull Request Succeeded Members</h1>
    <p>아래에 본인의 계정 이름이 적힌 11 태그를 추가하고 Pull Request를 보내세요.</p>
    <ul>
      <li>kangtegong</li>
      <li>Jeongyun-Jang</li>
      <li>woghh002</li>
      <li>xyunhx</li>
      <li>GokNyong</li>
      <li>KOGH-44</li>
      <li>yurjune</li>
      <li>chae-young</li>
      <li>louie-03</li>
      <li>wjddn1892</li>
      <li>springtiger07</li>
      <li>hyungjoon90</li>
      <li>BomJUL</li>
      <li>ChoiJunseo</li>
      <li>leejeongdong</li>
      <li>taeha2245</li>
      <li>first-github-user</li>
      <li>jegbit</li>
    </ul>
  </body>
</html>
```

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑨ </ul> 태그 위에 다음과 같이 줄을 맞춰 여러분의 깃허브 아이디를 적어줌

## 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-45 | 깃허브 아이디 입력 후 저장하기

```
File Edit Selection View Go Run Terminal Help • index.html - Visual Studio Code  
index.html ●  
C:\collaboration>index.html > html > body > ul > li  
6   <ul>  
7     <li>kangtegong</li>  
8     <li>Jeongyun-Jang</li>  
9     <li>mogh002</li>  
10    <li>xyunhx</li>  
11    <li>GokNyong</li>  
12    <li>KOGH-44</li>  
13    <li>yurjune</li>  
14    <li>chae-young</li>  
15    <li>Louie-03</li>  
16    <li>wjddn1892</li>  
17    <li>springtiger07</li>  
18    <li>hyungjoon90</li>  
19    <li>BomULUL</li>  
20    <li>Chojunseo</li>  
21    <li>leejeongdong</li>  
22    <li>taeha2245</li>  
23    <li>first-github-user</li>  
24    <li>jiehit</li>  
25    <li>first-github-user</li> 입력  
26  </ul>  
27 </body>  
28 </html>  
29
```

Ln 25, Col 36 Spaces: 3 UTF-8 CRLF HTML ⌂ ⌂

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑩ 이제 이 변경 사항을 커밋하고 푸시하면 됨
- 커밋하기 전, git diff 명령으로 변경 사항이 올바르게 만들어졌는지 확인해 보자
- 다음과 같이 <li>여러분의 깃허브 아이디</li> 한 줄이 추가됐다면 올바르게 작업한 것

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (add_myname)
$ git diff
diff --git a/index.html b/index.html
index 032d3be..07002ab 100644
--- a/index.html
+++ b/index.html
@@ -22,6 +22,7 @@
      <li>taeha2245</li>
      <li>first-github-user</li>
      <li>jegbit</li>
+      <li>first-github-user</li> 확인
    </ul>
  </body>
</html>
```

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑪ 이제 index.html을 커밋
- 커밋 메시지를 add myname in index.html로 하겠음

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (add_myname)
$ git add index.html
```

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (add_myname)
$ git commit -m "add myname in index.html"
[add_myname 01779fc] add myname in index.html
 1 file changed, 1 insertion(+)
```

# 8.2 깃 명령으로 풀 리퀘스트 보내기

## » 깃 명령으로 풀 리퀘스트 보내기

- ⑫ 이제 풀 리퀘스트의 네 번째 단계, ‘작업한 브랜치 푸시하기’
- add\_myname이라는 브랜치를 원격 저장소 origin에 푸시하겠다는 의미로 git push origin add\_myname 명령을 입력하면 add\_myname 브랜치가 원격 저장소 origin에 푸시

## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (add_myname)
$ git push origin add_myname
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 298 bytes | 298.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'add_myname' on GitHub by visiting:
remote:     https://github.com/first-github-user/collaboration/pull/new/
remote:     add_myname
remote:
To github.com:first-github-user/collaboration.git
 * [new branch]      add_myname -> add_myname
```

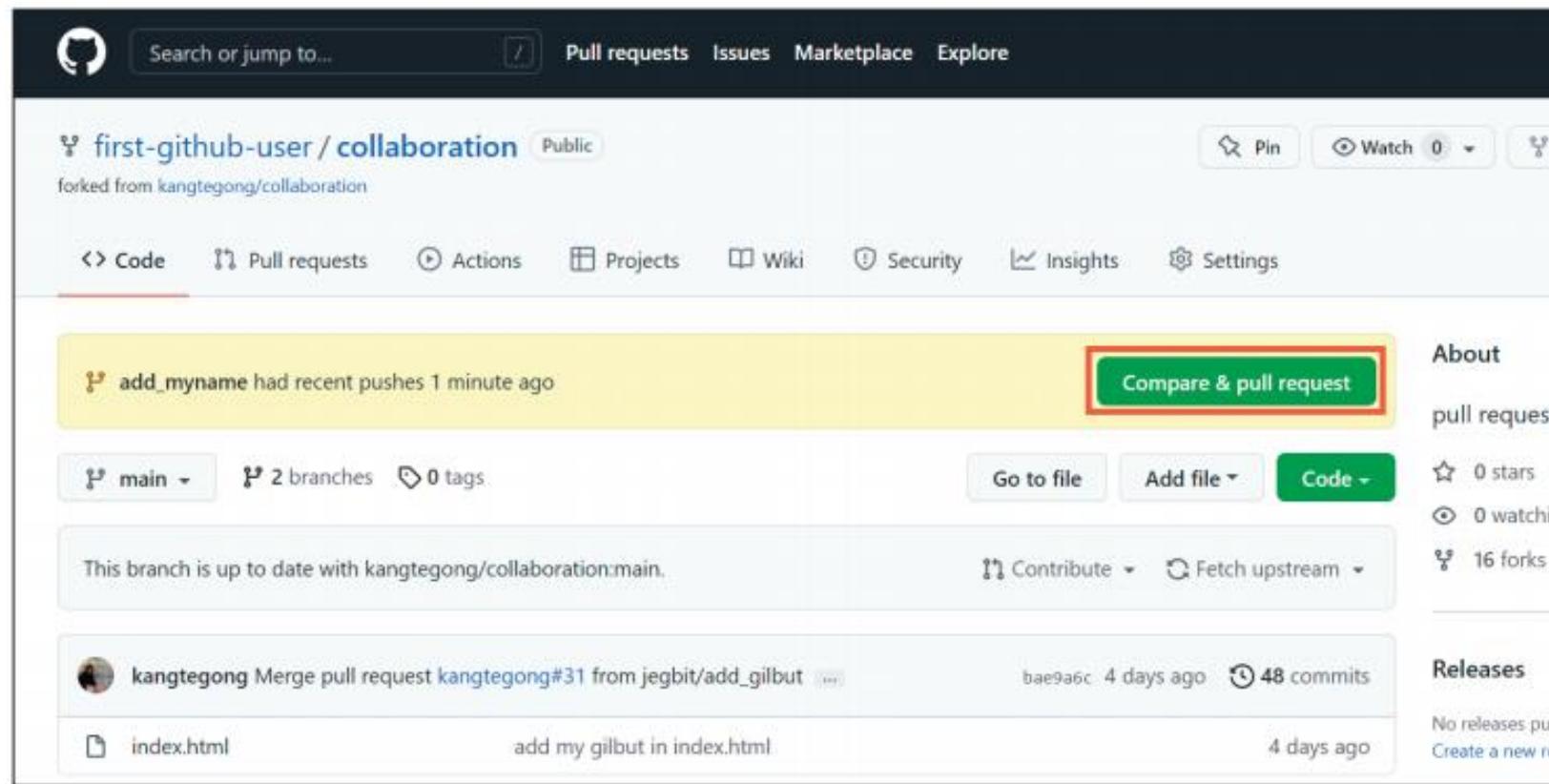
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑯ 13 자, 마지막으로 풀 리퀘스트를 보내자
- 포크한 원격 저장소로 돌아가보면 Compare & pull request 버튼이 생겼을 것
- 이를 클릭해 보자

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-46 | ‘Compare & pull request’ 클릭하기



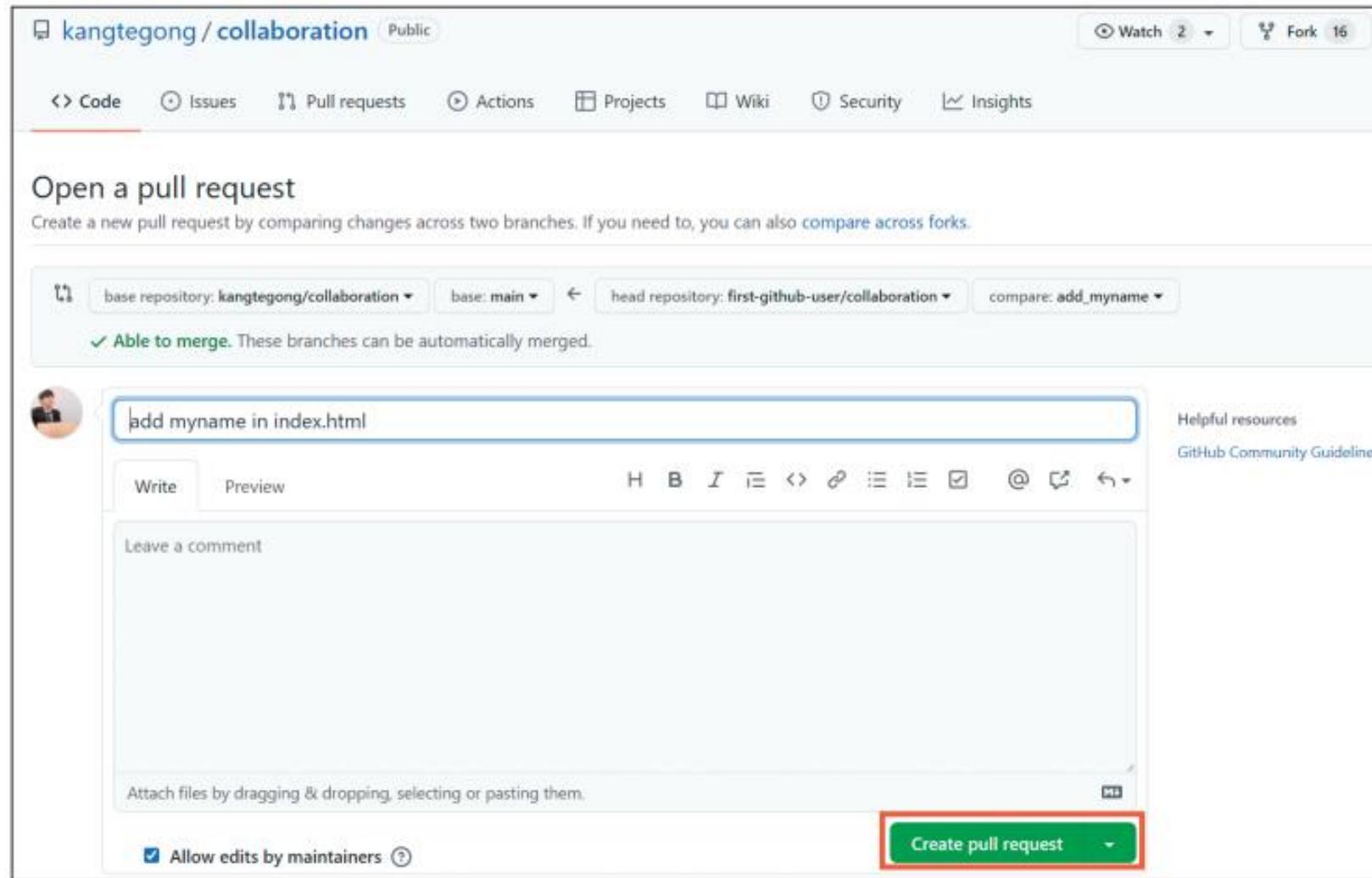
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- 14 풀 리퀘스트를 보내는 화면
- Create pull request 버튼을 클릭

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-47 | ‘Create pull request’ 클릭하기



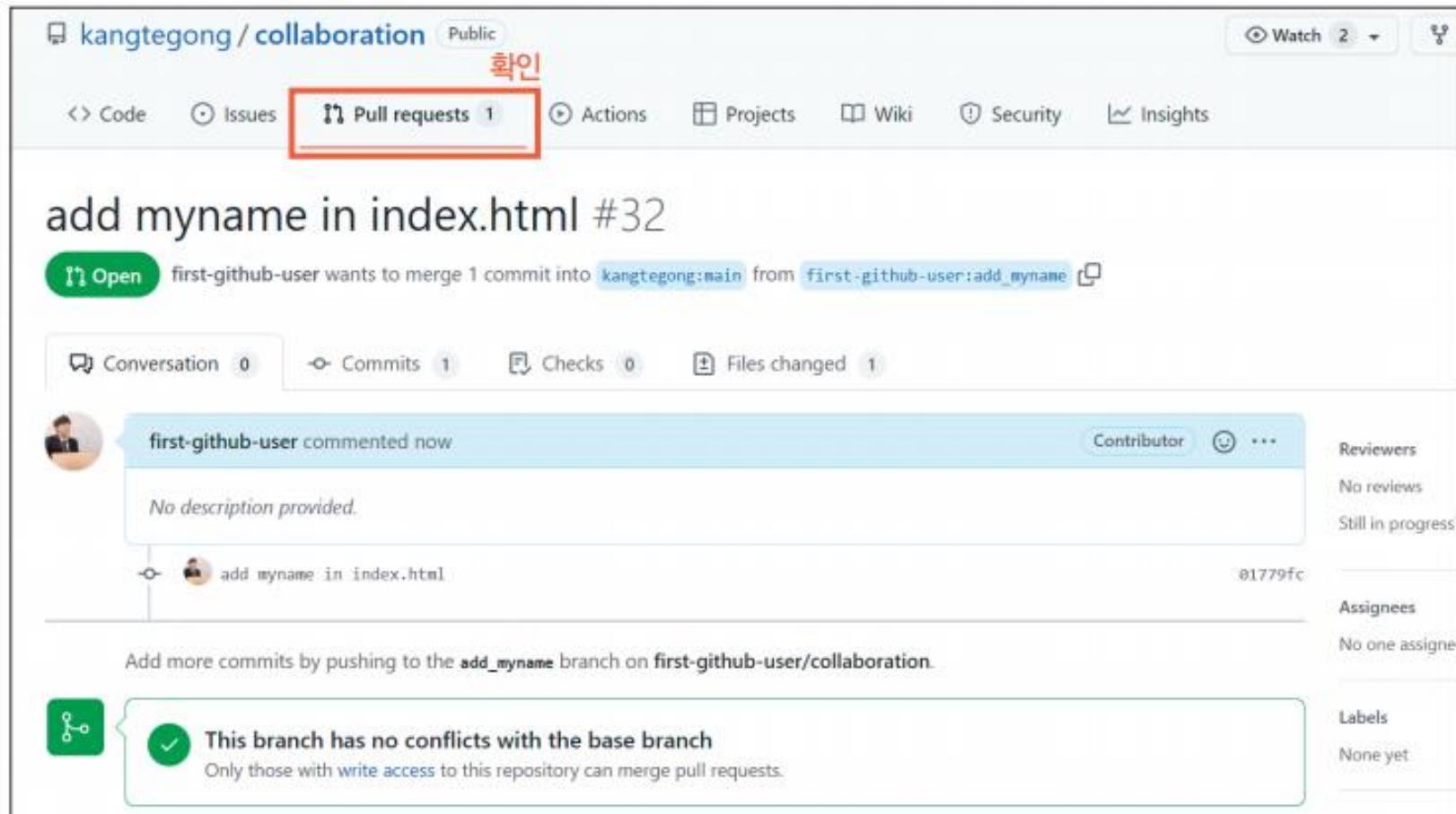
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑯ 깃 명령으로 풀 리퀘스트 보내기

## 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-48 | 성공적으로 생성된 풀 리퀘스트



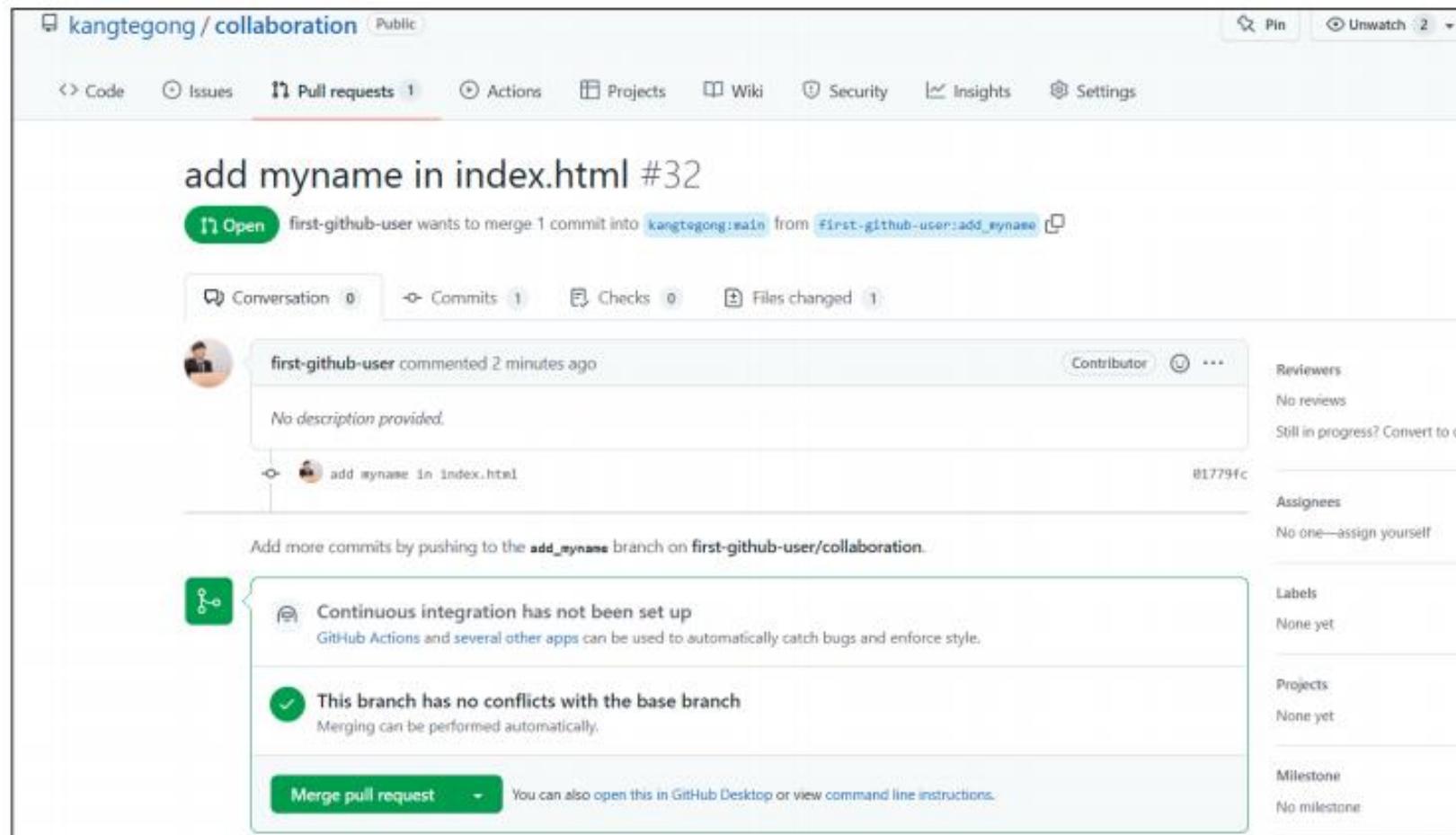
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑯ 여기까지 잘 했다면 여러분의 할 일은 끝
- 이제 collaboration 저장소의 소유자인 kangtegong이 여러분의 풀 리퀘스트를 병합해 주거나, 댓글을 달아줄 것
- 다음 사진은 collaboration 저장소의 소유자 시점에서 바라본 여러분의 풀 리퀘스트

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-49 | 저장소 소유자 시점에서 본 풀 리퀘스트



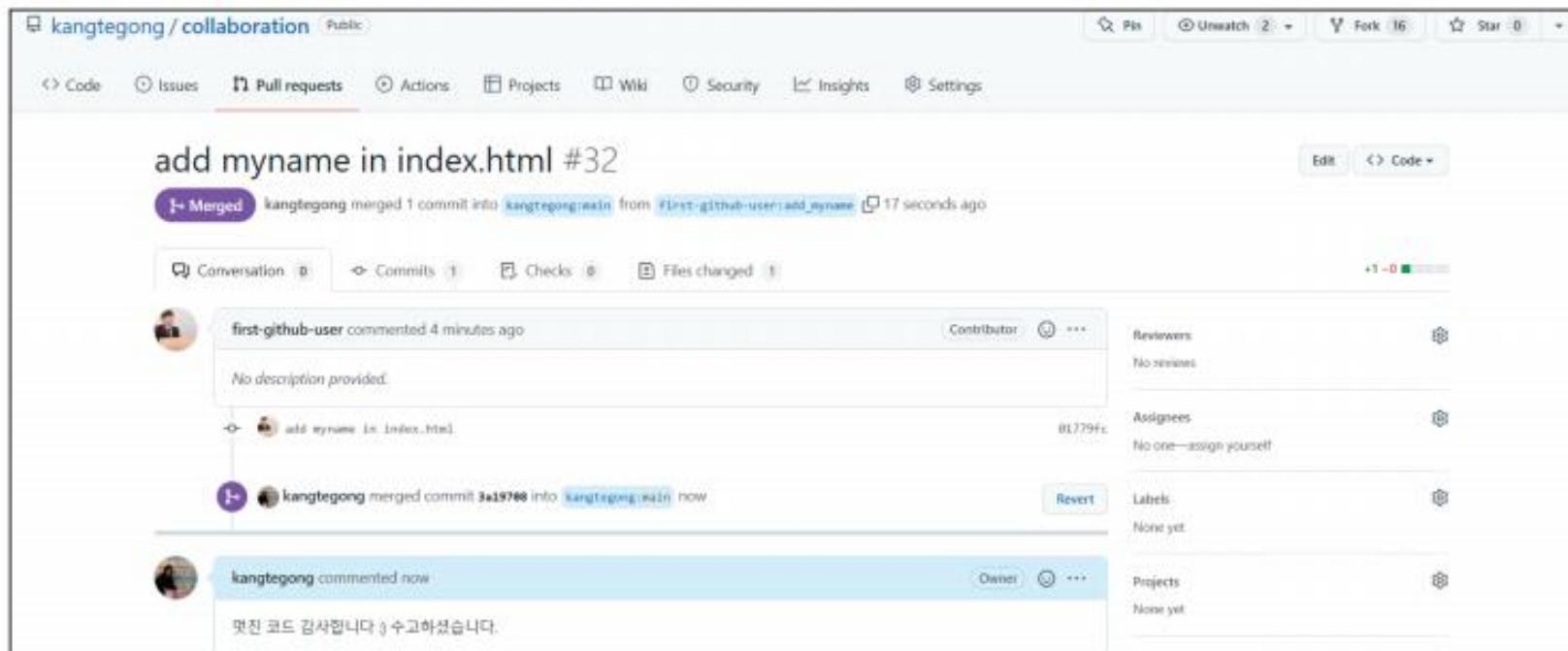
## 8.2 깃 명령으로 풀 리퀘스트 보내기

### » 깃 명령으로 풀 리퀘스트 보내기

- ⑯ 풀 리퀘스트가 병합된 모습과 간단한 댓글이 달린 모습은 다음과 같음
- 풀 리퀘스트가 병합됐다면 이제 kangtegong의 collaboration 저장소에서 여러분의 변경 사항을 확인할 수 있음

# 8.2 깃 명령으로 풀 리퀘스트 보내기

▼ 그림 8-50 | 풀 리퀘스트가 병합된 모습과 댓글이 달린 모습



# 8.3 더 나아가기

## » 더 나아가기

- 지금까지 명령어로 깃을 다루는 방법에 대해 알아보았음
- 분량상 깃의 수많은 명령과 옵션을 전부 다루지는 않았지만, 지금까지 설명한 명령들만 제대로 숙지해도 여러분이 추후 깃을 활용하는 데는 큰 지장이 없음

### ▼ 그림 8-51 | 수많은 깃 명령어

```
minchul@DESKTOP-9KULGUE MINGW64 /c/collaboration (add_myname)
$ git
Display all 68 possibilities? (y or n)
add           config          log             restore
am            credential-helper-selector maintenance
apply          describe         merge
archive        diff             mergetool
askpass        difftool
askyesno      fetch
bisect         flow
blame          format-patch
branch        fsck
bundle        gc
checkout       gitk
cherry        grep
cherry-pick   gui
citoool       help
clean          init
clone          instaweb
commit         lfs
                           merge
                           mv
                           notes
                           prune
                           pull
                           push
                           range-diff
                           rebase
                           reflog
                           remote
                           repack
                           replace
                           request-pull
                           reset
                           send-email
                           shortlog
                           show
                           show-branch
                           sparse-checkout
                           stage
                           stash
                           status
                           submodule
                           switch
                           tag
                           update-git-for-windows
                           whatchanged
                           worktree
```

## 8.3 더 나아가기

» git <명령> --help: 매뉴얼 페이지 보기

- 가장 추천하는 방법
- 깃 명령 뒤에 --help를 치면 해당 명령에 대한 자세한 설명이 담긴 매뉴얼 페이지가 브라우저로 열림

## 8.3 더 나아가기

» git <명령> --help: 매뉴얼 페이지 보기

- ① 가령 git commit 명령에 대한 자세한 설명을 보기 위해 git commit --help를 입력해 보자

```
minchul@DESKTOP-9KULGUE MINGW64 /c  
$ git commit --help
```

## 8.3 더 나아가기

» git <명령> --help: 매뉴얼 페이지 보기

- ② 브라우저에 다음과 같은 페이지가 열림

▼ 그림 8-52 | git commit 매뉴얼 페이지

git-commit(1) Manual Page

**NAME**

git-commit - Record changes to the repository

**SYNOPSIS**

```
git commit [-a | --interactive | --patch] [-s] [-v] [-u<mode>] [--amend]
           [--dry-run] [(-c | C | --squash)] <commit> | --fixup [(--amend | reword);]<commit>]
           [-F <file> | -m <msg>] [--reset-author] [--allow-empty]
           [--allow-empty-message] [--no-verify] [-e] [--author=<author>]
           [--date=<date>] [--cleanup=<mode>] [--[no-]status]
           [-i | -o] [--pathspec-from-file=<file>] [--pathspec-file-nul]
           [(-trailer <token>[=] ;<value>)]... [-S<keyid>]
           [-] [<pathspec>...]
```

**DESCRIPTION**

## 8.3 더 나아가기

» git <명령> --help: 매뉴얼 페이지 보기

- 이 매뉴얼 페이지에는 git commit 명령에 대한 자세한 설명이 담겨 있음
- git commit 명령은 무엇인지, 어떻게 사용해야 하는지, 이 명령과 함께 사용할 수 있는 옵션에는 어떤 것들이 있는지 등 자세한 설명이 적혀 있음
- 영어로 되어 있지만, 이보다 신뢰성 높고 자세한 자료는 없다고 보아도 무방함

## 8.3 더 나아가기

### » git <명령> --help: 매뉴얼 페이지 보기

- ③ 스크롤을 내려보면 여러분들이 학습한 -m 옵션도 보임
- -m 옵션은 간단한 커밋 메시지를 통해 커밋할 때 사용한다고 했음
- git commit -m "커밋 메시지"

#### ▼ 그림 8-53 | git commit에서 사용할 수 있는 옵션

`-m <msg>`

`--message=<msg>`

Use the given <msg> as the commit message. If multiple `-m` options are given, their values are concatenated as separate paragraphs.

The `-m` option is mutually exclusive with `-c`, `-C`, and `-F`.

## 8.3 더 나아가기

» git <명령> --help: 매뉴얼 페이지 보기

- ④ 다른 명령어도 마찬가지
- 이번에는 git log 명령에 대한 매뉴얼 페이지도 볼까?

```
minchul@DESKTOP-9KULGUE MINGW64 /c
$ git log --help
```

# 8.3 더 나아가기

## » git <명령> --help: 매뉴얼 페이지 보기

- ⑤ 마찬가지로 git log와 관련한 정보를 자세히 담고 있는 매뉴얼 페이지가 나타남

♥ 그림 8-54 | git log 매뉴얼 페이지



git-log(1) Manual Page

**NAME**

git-log - Show commit logs

**SYNOPSIS**

*git log [<options>] [<revision-range>] [<-->] <path>...*

---

**DESCRIPTION**

Shows the commit logs.

List commits that are reachable by following the `parent` links from the given commit(s), but exclude commits that are reachable from the one(s) given with a `^` in front of them. The output is given in reverse chronological order by default.

You can think of this as a set operation. Commits reachable from any of the commits given on the command line form a set, and then commits reachable from any of the ones given with `^` in front are subtracted from that set. The remaining commits are what comes out in the command's output. Various other options and paths parameters can be used to further

## 8.3 더 나아가기

### » git <명령> --help: 매뉴얼 페이지 보기

- 매뉴얼 페이지에 나오는 모든 내용을 알아야 한다는 부담감은 갖지 않아도 됨
- 앞서 설명했듯 설명한 명령만 잘 파악해도 기본적인 깃 사용에는 큰 지장이 없기 때문임
- 여기까지 읽었다면 이미 깃에 대한 입문은 잘 끝마친 셈이니, 매뉴얼 페이지에서 생소한 내용을 접하더라도 부담 갖지 말고 ‘깃의 고수가 되기 위한 과정’ 정도로 생각해 주자

## 8.3 더 나아가기

### » git-scm.com: 공식 사이트

- 두 번째로 추천하는 학습 방법
- 깃을 처음 내려받고 설치했던 사이트를 기억해보자
- 다시 한번 접속해 볼까?
- ① 다음 URL에 접속한 다음 Documentation을 클릭해 보자

URL <https://git-scm.com>

# 8.3 더 나아가기

▼ 그림 8-55 | 깃 공식 페이지 접속 화면



# 8.3 더 나아가기

» git-scm.com: 공식 사이트

- ② Reference를 클릭

▼ 그림 8-56 | ‘Reference’ 클릭하기

The screenshot shows the official Git documentation website at [git-scm.com](https://git-scm.com). The navigation bar includes links for About, Documentation (with sub-links for Reference, Book, Videos, and External Links), Downloads, and Community. The main content area is titled "Documentation". Under "Documentation", there is a section titled "Reference" which is highlighted with a red box. This section contains a "Reference Manual" card featuring a book icon and text about the official man pages included in the Git package. Below this, there is a "Book" card featuring a book icon and text about the "Pro Git" book by Scott Chacon and Ben Straub, noting that it is available online for free and in print on Amazon.com. A search bar is located at the top right of the page.

# 8.3 더 나아가기

## » git-scm.com: 공식 사이트

- ③ Reference에서는 --help 옵션으로 볼 수 있었던 매뉴얼을 볼 수 있음

▼ 그림 8-57 | 깃 공식 페이지의 ‘Reference’ 페이지

The screenshot shows the 'Reference' page of the git-scm.com website. At the top left is the git logo and the tagline '--everything-is-local'. A search bar is at the top right. On the left, there's a sidebar with links for 'About', 'Documentation' (which is currently selected), 'Downloads', and 'Community'. The main content area is titled 'Reference' and contains several sections: 'Quick reference guides' (GitHub Cheat Sheet, Visual Git Cheat Sheet), 'Complete list of all commands' (with a link to 'Setup and Config'), 'Setup and Config' (listing 'git', 'config', 'help', 'bugreport'), 'Getting and Creating Projects' (listing 'init', 'clone'), 'Basic Snapshotting' (listing 'add', 'status', 'diff', 'commit', 'notes'), 'Guides' (listing 'gitattributes', 'Command-line interface conventions', 'Everyday Git', 'Frequently Asked Questions (FAQ)', 'Glossary', 'Hooks', 'gitignore', 'gitmodules', 'Revisions', 'Submodules', 'Tutorial', 'Workflows', 'All guides...'), and an 'Email' link.

## 8.3 더 나아가기

### » git-scm.com: 공식 사이트

- ④ --help 명령을 쳤을 때와 비슷하게 대부분의 문서가 영어로 되어 있지만, 간혹 한국어로 번역된 문서를 볼 수도 있음

# 8.3 더 나아가기

## ▼ 그림 8-58 | Reference 페이지의 다국어 지원

The screenshot shows the official Git documentation page for the command `git init`. The page includes a navigation bar with links to About, Documentation (Reference, Book, Videos, External Links), Downloads, and Community. The main content area displays the command's NAME, SYNOPSIS, and DESCRIPTION. A red box highlights the right sidebar, which contains a "Topics" dropdown set to English, a search bar, and a list of localized versions of the manual in various languages: Deutsch, English, Français, Íslenska, 日本語, Português (Brasil), and Română. Below this list is a note encouraging users to help translate the page.

git --distributed-is-the-new-centralized

Version 2.35.1 • git-init last updated in 2.35.1

About Documentation Reference Book Videos External Links Downloads Community

**NAME**

git-init - Create an empty Git repository or reinitialize an existing one

**SYNOPSIS**

```
git init [-q | --quiet] [--bare] [--template=...] [--separate-git-dir <git-dir>] [--shared[=<permissions>]] [<directory>]
```

**DESCRIPTION**

This command creates an empty Git repository - basically a `.git` directory with subdirectories for `objects`, `refs/heads`, `refs/tags`, and template files. An initial branch without any commits will be created (see the `--initial-branch` option below for its name).

If the `$GIT_DIR` environment variable is set then it specifies a path to use instead of `./.git` for the base of the repository.

Topics • English •

Localized versions of `git-init` manual

Deutsch  
English  
Français  
Íslenska  
日本語  
Português (Brasil)  
Română

Want to read in your language or fix typos?  
You can help translate this page.

## 8.3 더 나아가기

### » git-scm.com: 공식 사이트

- ⑤ 이번에는 Book을 클릭해 보자
- 깃을 자세히 학습할 수 있는 『Pro Git』이라는 책을 소개하는 페이지로 이동
- 이는 웹 사이트에서도 읽을 수 있고, pdf 파일로 내려받아 읽어볼 수도 있음

# 8.3 더 나아가기

▼ 그림 8-59 | ‘Book’ 클릭하기

The screenshot shows the Pro Git website ([git-scm.com/book](http://git-scm.com/book)). The left sidebar has a red box around the 'Book' link under the 'Documentation' section. The main content area is titled 'Book'. It contains a summary of the book, its authors, and availability, followed by two sections of table of contents: '1. Getting Started' and '2. Git Basics'. To the right is an image of the book 'Pro Git' and download links for PDF, EPUB, and MOBI formats.

git --everything-is-local

Search entire site...

About

Documentation

Reference

Book

Videos

External Links

Downloads

Community

This book is available in English.  
Full translation available in  
azərbaycan dili,  
български език,  
Deutsch,  
Español,  
Français,  
Ελληνικά,  
日本語,  
한국어,

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).

**1. Getting Started**

- 1.1 About Version Control
- 1.2 A Short History of Git
- 1.3 What is Git?
- 1.4 The Command Line
- 1.5 Installing Git
- 1.6 First-Time Git Setup
- 1.7 Getting Help
- 1.8 Summary

**2. Git Basics**

- 2.1 Getting a Git Repository
- 2.2 Recording Changes to the Repository

2nd Edition (2014)

Download Ebook

pdf    epub    mobi

## 8.3 더 나아가기

» git-scm.com: 공식 사이트

- ⑥ 좌측을 보면 한국어가 있음
- 이를 클릭해 보자

# 8.3 더 나아가기

▼ 그림 8-60 | ‘한국어’ 클릭하기

The screenshot shows the official Pro Git website ([git-scm.com/book](http://git-scm.com/book)). The main navigation bar includes links for 'About', 'Documentation' (with 'Reference', 'Book' (highlighted in red), 'Videos', and 'External Links'), 'Downloads', and 'Community'. On the left, a sidebar lists available translations: English, Azerbaijani, Bulgarian, Deutsch, Español, Français, Ελληνικά, 日本語, and 한국어 (highlighted with a red box). The central content area is titled 'Book' and describes the availability of the entire Pro Git book in PDF and print formats. It features an image of the 'Pro Git' book cover (2nd Edition, 2014) and a 'Download Ebook' section with icons for PDF, EPUB, and Amazon Kindle.

**About**

**Documentation**

- Reference
- Book**
- Videos
- External Links

**Downloads**

**Community**

This book is available in [English](#).  
Full translation available in  
[azərbaycan dili](#),  
[български език](#),  
[Deutsch](#),  
[Español](#),  
[Français](#),  
[Ελληνικά](#),  
[日本語](#),  
**한국어**.

## Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).

**1. Getting Started**

- 1.1 About Version Control
- 1.2 A Short History of Git
- 1.3 What is Git?
- 1.4 The Command Line
- 1.5 Installing Git
- 1.6 First-Time Git Setup
- 1.7 Getting Help
- 1.8 Summary

**2. Git Basics**

- 2.1 Getting a Git Repository
- 2.2 Recording Changes to the Repository

Search entire site...

2nd Edition (2014)

**Download Ebook**

pdf   epub   kindle

## 8.3 더 나아가기

» git-scm.com: 공식 사이트

- 7 한국어판 『Pro Git』을 볼 수 있음

# 8.3 더 나아가기

▼ 그림 8-61 | 한국어판 Pro Git

The screenshot shows the Pro Git website's "Book" page. The left sidebar includes links for "About", "Documentation" (with "Reference", "Book", "Videos", and "External Links" options), "Downloads", and "Community". A note states that the book is available in English and provides links for full translations in various languages. The main content area features the title "Book" and a brief description of the book's availability. It includes an image of the "Pro Git" book cover (2nd Edition, 2014) and a "Download Ebook" section with PDF and EPUB download icons.

**About**

**Documentation**

Reference  
Book  
Videos  
External Links

**Downloads**

**Community**

This book is available in English.  
Full translation available in:  
azərbaycan dilü,  
български език,  
Deutsch,  
Español,  
Français,  
Ελληνικά,  
日本語,  
한국어,

**Book**

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the Creative Commons Attribution Non Commercial Share Alike 3.0 license. Print versions of the book are available on Amazon.com.

**1. 시작하기**

- 1.1 버전 관리란?
- 1.2 꽂게 보는 Git의 역사
- 1.3 Git 기초
- 1.4 CLI
- 1.5 Git 설치
- 1.6 Git 최초 설정
- 1.7 도움말 보기
- 1.8 요약

**2. Git의 기초**

- 2.1 Git 저장소 만들기
- 2.2 수천하고 저작소에 저장하기

Search entire site...

2nd Edition (2014)

Download Ebook

pdf    epub