

비전공자

웹프로그램을 위한

git 문법 책

발행일 : 2021.01.01

변경일 : 2021.09.23 (2본 시작일)

저자 : 박수민

이메일 : foxman12@hanmail.net

발행처 : 휴먼교육

git -

목 차

01. git

> 01. git 이란?

3

3

1. 3개 설치
2. bash 이용한 프로필 설정 sourcetree와 bash 같은 것이다.
3. 간단한 sourcetree 코드 일기 쓰는 것과 같다 간단한 소스트리 사용
4. 작업디렉터리, 스테이지, 저장소, 원격저장소 설명, add, commit
5. gitignore에 대한 설명
6. 태그에 대한 설명
7. 폐기와 제거
9. history를 이용해서 과거 상태를 선택해서 커밋당시 내용을 확인할 수 있다.
10. revert과 reset 이해와 사용
11. 충돌해결법 폐기하자.
12. 브랜치 사용하기

최초의 브랜치를 master라 한다.

head란 현재 작업중인 브랜치를 의미한다.

cheak out이란? 다른 브랜치로 작업환경을 변경하는 것을 의미한다.
13. 브랜치 병합
14. 브랜치 삭제
15. 재배치
5. add, commit, branch, push, fetch, pull, clone, fork에 대한 설명
16. 원격저장 설정
17. 클론clone 원격저장소를 복제하기 및 삭제하기

`origin` 원격저장소별명

`main master`와동일

`head` 현재작업중인 브랜치

18. `push` 원격 저장소에 밀어 넣기

19. `fetch` 원격저장소를 일단 가져만오기

20. `pull` 원격저장소를 가져와서 합치기

21. `eclipse`연결하기

22. 콜라보

01. git

> 01. git 이란?

Git은 소스 코드를 관리하고 추적하는 도구로, 여러 개발자가 함께 작업할 때 유용합니다. 변경 사항을 기록하여 이전 상태로 돌아갈 수 있고, 충돌을 방지하여 협업을 용이하게 합니다. 원격 저장소 서비스와 통합되어 프로젝트를 온라인으로 백업하고 협업할 수 있습니다. 브랜치 기능을 통해 여러 작업을 동시에 관리할 수 있어 개발과 수정이 효율적으로 이루어집니다.

The screenshot shows the Naver homepage with the following sections:

- Top News:** Headlines from Naver TV, VIBE, and other sources.
- Entertainment:** Headlines from Naver TV, VIBE, and other sources.
- Sports:** Headlines from Naver TV, VIBE, and other sources.
- Economy:** Headlines from Naver TV, VIBE, and other sources.
- Weather:** Current temperature (19.2°C) and a 5-day forecast for Incheon (17°C to 13°C).
- Finance:** Stock market information for Kospi (2,692.06), Oiichulhoen (12,640), Eswo (59,800), and Daksaneo (38,400).
- Cooking:** Recipe categories like Desserts, Korean Food, and International Food.
- Calendar:** A weekly calendar for the week starting Monday, April 30, 2018.

git은 소스관리 프로그램 source-tree는 GUI를 git-hub는 원격저장소이다.

git설치



--distributed-is-the-new-centralized

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.



About

The advantages of Git compared to other source control systems.



Downloads

GUI clients and binary releases for all major platforms.



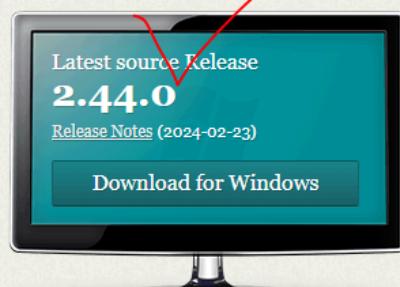
Documentation

Command reference pages, Pro Git book content, videos and other material.



Community

Get involved! Bug reporting, mailing list, chat, development





git-scm.com/downloads

-distributed-even-if-your-workflow-isnt

Downloads



macOS



Windows



Linux/Unix

[Older releases](#) are available and the [Git source repository](#) is on GitHub.

t book
iacon and
able to [read](#)

Download for Windows

[Click here to download](#) the latest (2.44.0) 64-bit version of Git for Windows. It was released about 2 months ago, on 2023-07-10.

Other Git for Windows downloads

[Standalone Installer](#)

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup](#).

[Portable \("thumbdrive edition"\)](#)

[32-bit Git for Windows Portable](#).

[64-bit Git for Windows Portable](#).

깃 설치하기

- 19 잘 설치됐는지 확인해 봅시다. 바탕화면 등 편한 공간에 폴더를 만들고 그 안에서 마우스 오른쪽 버튼을 클릭하면 Git Bash Here 항목이 생겼을 것입니다. Git Bash Here를 클릭합니다.

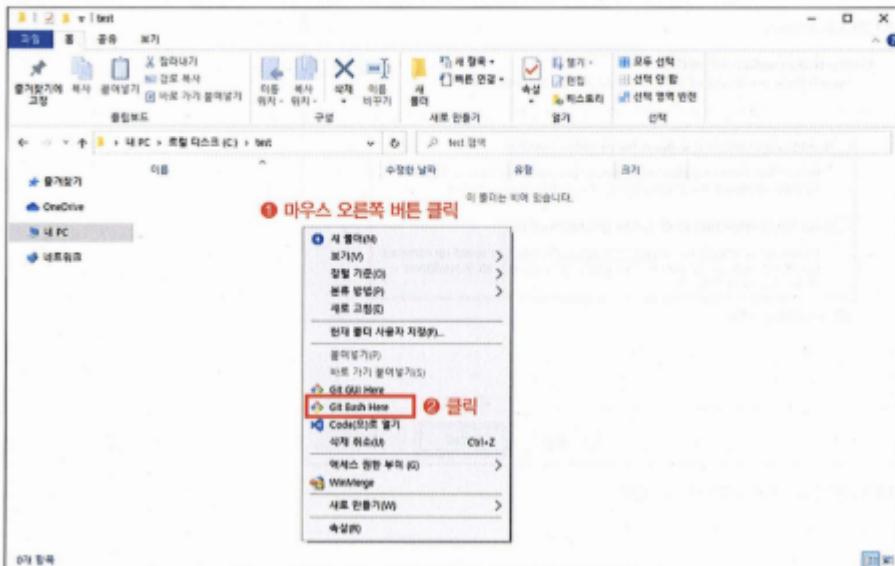


그림 1-37 | Git Bash Here 항목 확인하기

- 20 다음과 같이 명령어를 입력할 수 있는 공간, 즉 깃 배시(git bash)가 나옵니다. 여기에 깃 명령어를 직접 입력할 수 있습니다.



그림 1-38 | 깃 배시

TIP
참고로 다음 그림에서 박스 친 부분이 현재 내가 명령어를 입력하고 있는 작업 공간, 다시 말해 현재 작업 공간을 의미합니다. 여기서는 C 드라이브의 test 폴더입니다. 내가 지금 어디에서 명령어를 사용하고 있는지는 매우 중요하므로 이 현재 작업 공간에 각별히 유의해야 합니다.

minchu1@DESKTOP-9KULGUE MINGW64 /c/test (master)
\$ |

그림 1-39 | 현재 작업 공간 확인하기

- 21 git 명령어를 입력해 잘 설치됐는지 확인합니다. 다음 그림처럼 git과 관련한 명령어 목록이 잘 뜨는 걸 확인했다면 현재 깃이 잘 설치된 것입니다.

```
minchu1@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ git
usage: git [--version] [--help] [-C <path>] [-c <name=value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common git commands used in various situations:
start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one
```

2 깃 설정하기

- ① 깃을 잘 설치했다면 이제 본인 컴퓨터에 사용자 이름과 이메일을 등록하는 간단한 초기 설정을 해봅시다. 앞으로 깃을 이용해 만드는 모든 버전에는 '만든 사람', '지은이'와 같은 개념으로 지금부터 설정할 이름과 이메일이 함께 명시될 것입니다. 다음과 같이 명령을 입력해 봅시다(이름은 가급적 영어를 사용할 것을 권장합니다).

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --global user.name "Kang Minchul"
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --global user.email tegongkang@gmail.com
```

- ② 설정한 이름과 이메일은 다음과 같은 명령으로 확인할 수 있습니다.

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config user.name
Kang Minchul

minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config user.email
tegongkang@gmail.com
```

- ③ 이처럼 git config 명령으로 깃과 관련한 내용을 설정하거나 설정한 값을 확인할 수 있습니다. 다음 명령은 이름과 이메일뿐 아니라 다른 설정 값도 보여주는 명령인데, 자세히 살펴보면 우리가 초기 설치 과정에서 선택한 항목들도 설정된 값으로 출력되는 걸 확인할 수 있습니다(지금 시점에서 설정 값의 의미 하나하나를 모두 알 필요는 없습니다).

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
```

git config --list 명령어는 현재 Git 설정을 확인하는 데 사용됩니다. 이 명령어를 실행하면 Git의 설정된 모든 항목과 해당 값이 출력

window 콘솔과 window는 같은 것이다.

git bash 랑 sourcetree는 같은 작업을 하는 것이다.

git관련 작업을 지금 한것 처럼 콘솔로 작업 할 수 있지만 gui프로그램을 사용하면 간단하게 git작업을 할 수 있다. 가장 대표적인 git gui프로그램 소스트리를 살펴볼 예정이다.

```
add : staging 명령어
      - git add .
commit : staging된 파일들을 확정
      - git commit -m "메시지"
push : commit된 파일을 원격 저장소로 업로드
      - git push origin master
fetch : 원격 저장소의 정보로 업데이트
      - git fetch
pull : 원격 저장소의 변경된 파일을 다운로드해서 로컬 파일과 동기화
      - git pull

clone : 원격 저장소의 모든 내용을 내 저장소로 다운로드
branch : push가 따로 관리되는 복사본
```

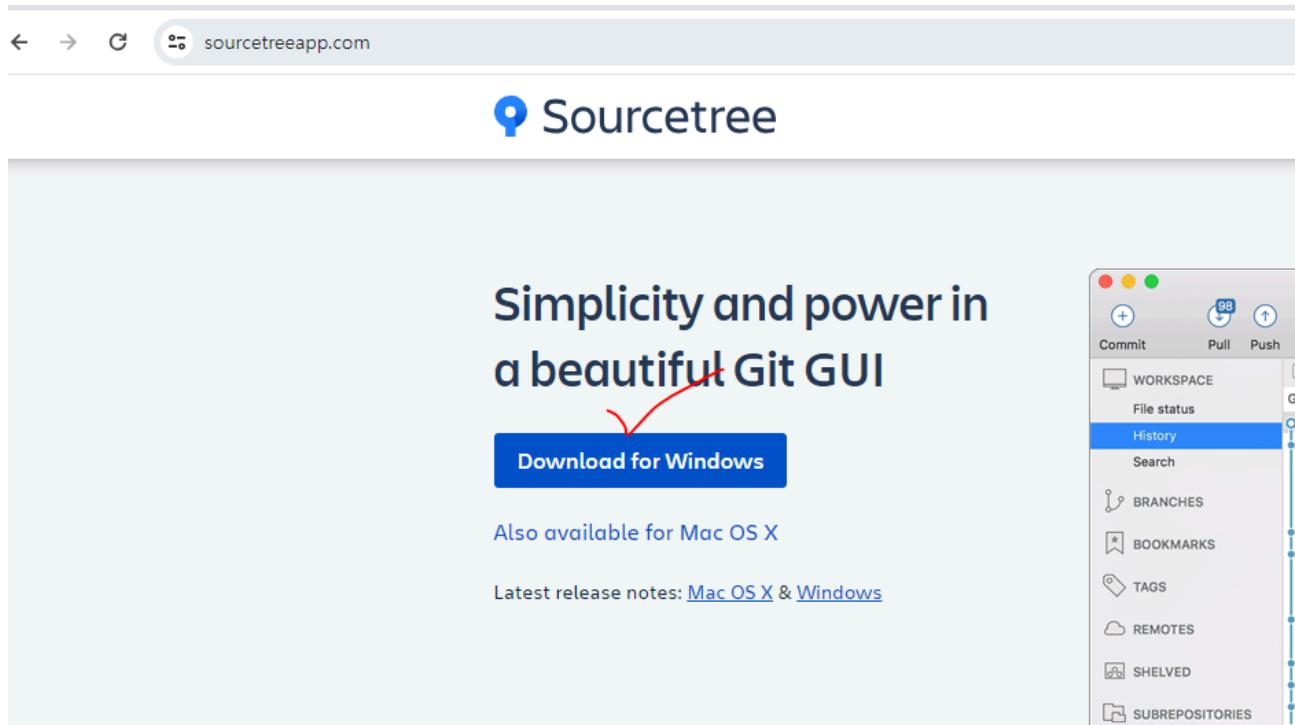
```
commit : staging된 파일들을 확정
      - git commit -m "메시지"
push : commit된 파일을 원격 저장소로 업로드
      - git push origin master
fetch : 원격 저장소의 정보로 업데이트
      - git fetch
pull : 원격 저장소의 변경된 파일을 다운로드해서 로컬 파일과 동기화
      - git pull
clone : 원격 저장소의 모든 내용을 내 저장소로 다운로드
```

```
branch : push가 따로 관리되는 복사본
checkout : branch 변경
merge : branch 병합
fork : branch이긴 하지만 push가 즉각 반영되지 않음
pull request : fork의 내용을 승인해달라는 요청 pull req 풀리퀘
```

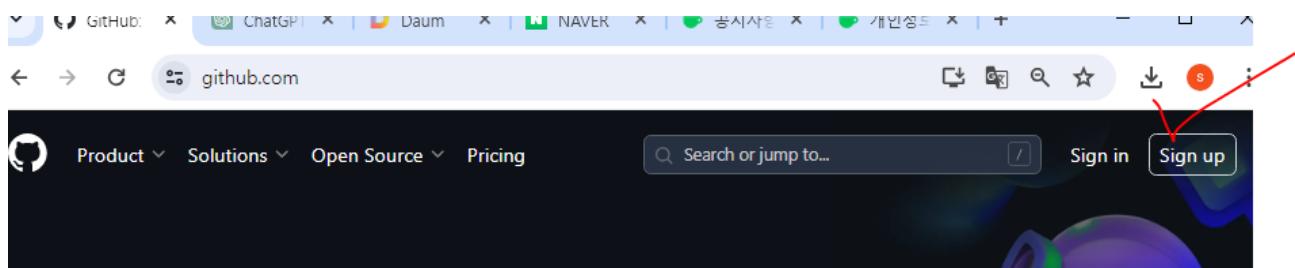
git fetch 명령어는 저장소의 최신 변경 사항을 로컬 저장소로 가져오는 데 사용됩니다.

이 명령어는 원격 저장소의 최신 정보를 확인하고 로컬 저장소를 업데이트할 때 유용합니다.

소스트리 설치법



git-hub

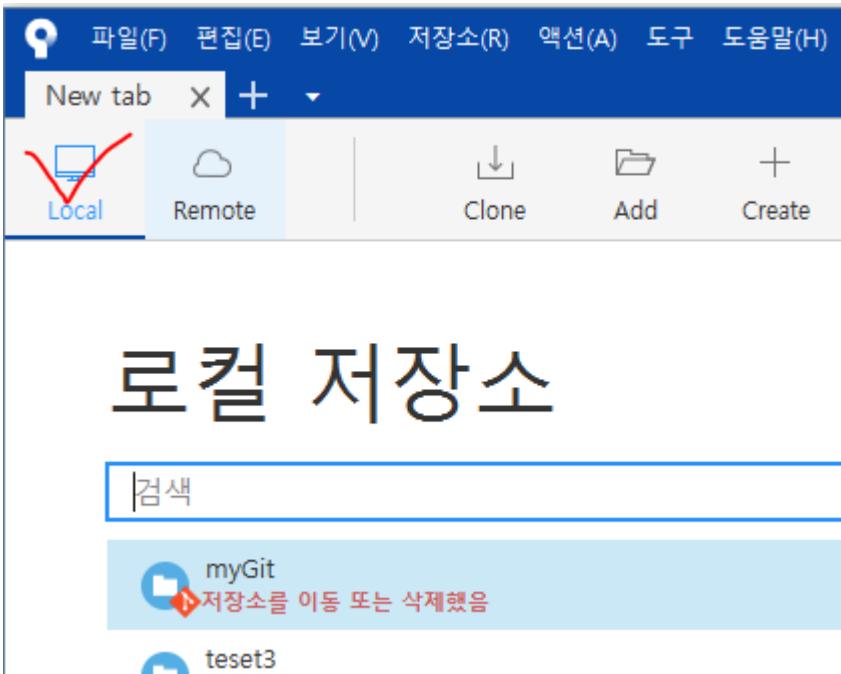


로그인 하자.

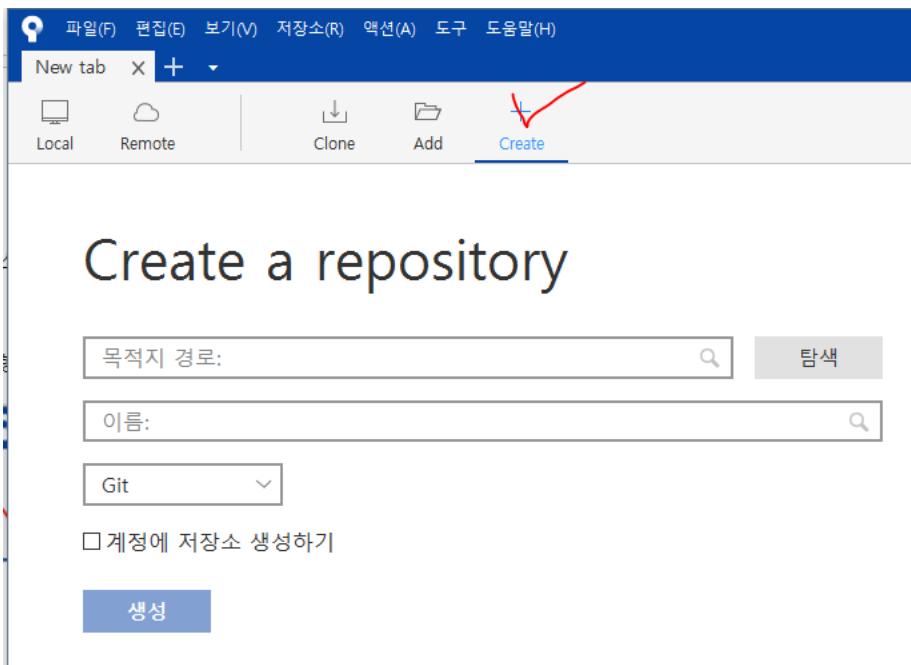
git은 소스관리 프로그램 source-tree는 GUI를 git-hub는 원격저장소이다.

소스트리로 로컬 저장소 연결하기

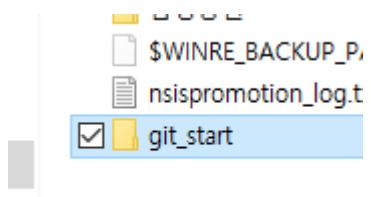
실행후 local선택



새로 만들려면 create 선택



c에 git_start 풀더 생성



탐색하거나 입력 해서 해당 폴더 연결



Create a repository

c:\git_start

탐색

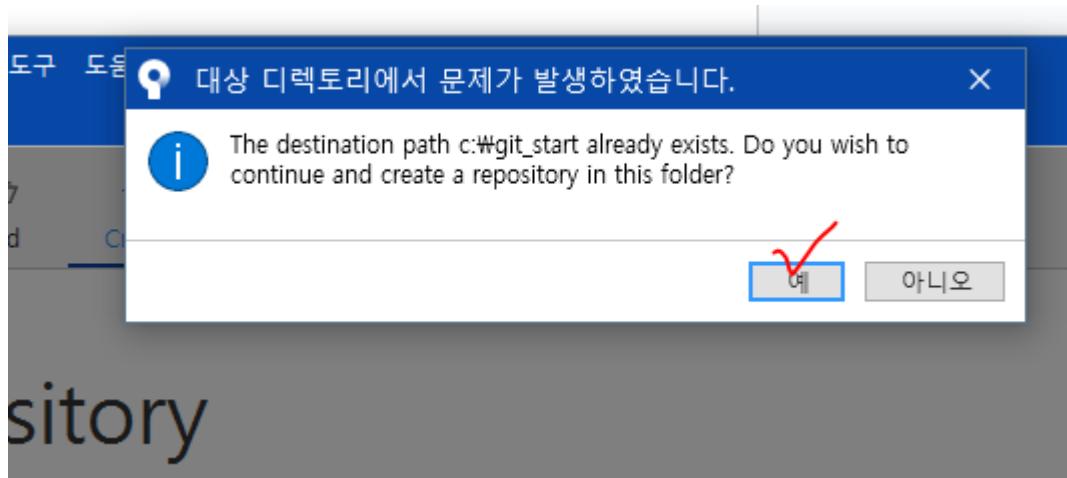
git_start

Git

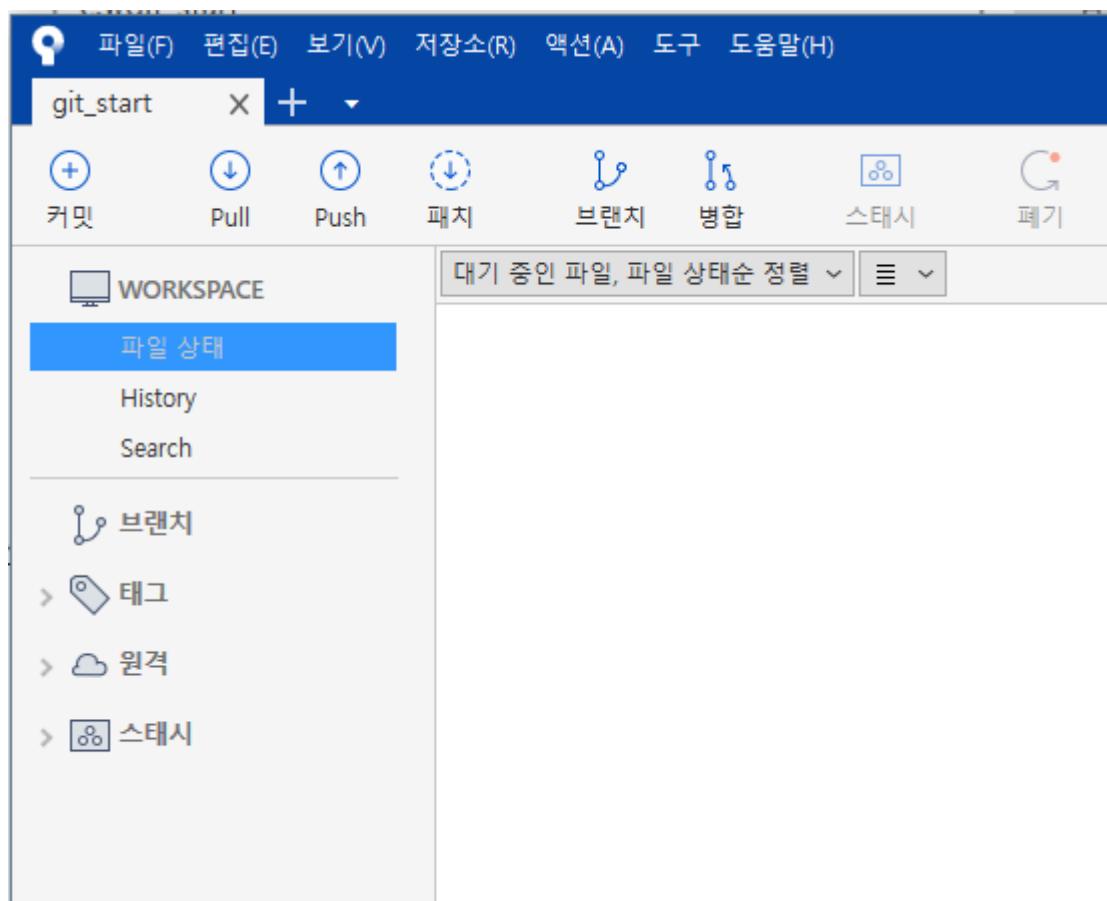
계정에 저장소 생성하기

생성

생성 버튼을 누르고 yes를 선택 한다.

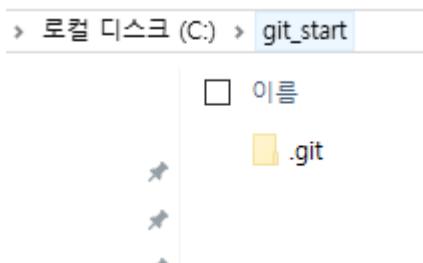


다음과 같은 화면이 뜬다.



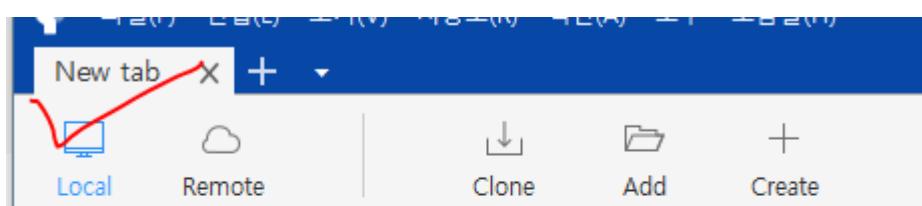
해당 폴더에 들어가면 .git 폴더가 생긴것을 확인할 수 있다.

종종 생성되지 않을때도 있는데 무시하고 작업을 이어나가면 도중에 생기기도 하고 안생기면 해당 폴더를 지우고 다시 실행해 보자.



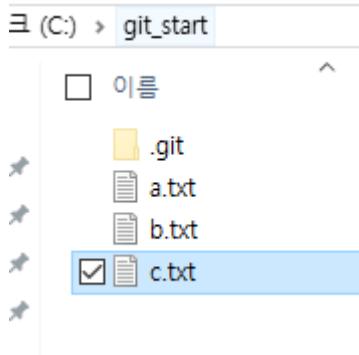
확장자가 보이지 않으면 확장자 보기 를 선택한다.

기존 만든걸 다시 열 경우 local 클릭한다음 기존 프로젝트를 검색 혹은 다시 선택한다.



로컬 저장소

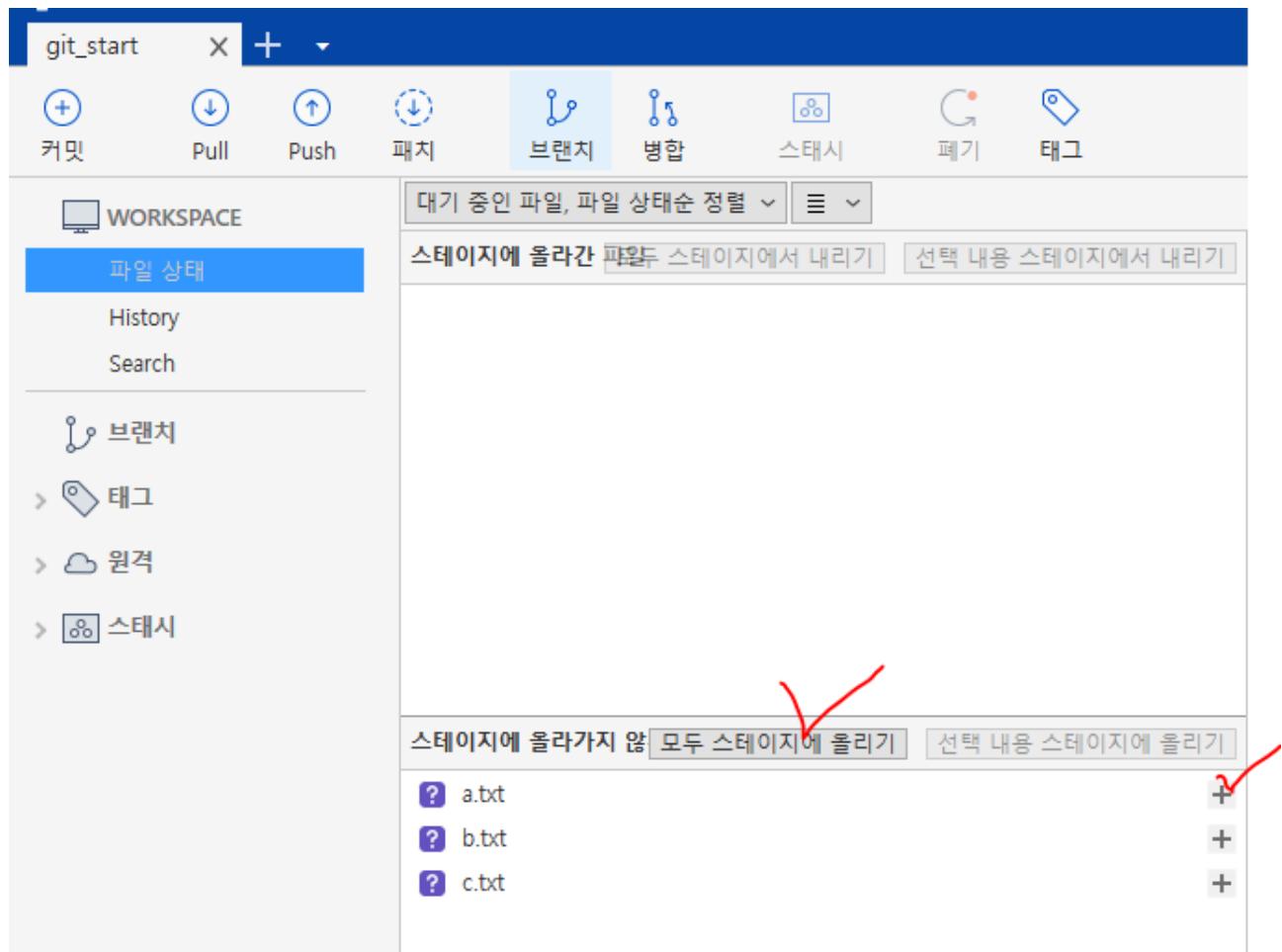
Repository	Status	Path
git_start	Normal	c:\git_start
myGit	Deleted	
teset3	Deleted	



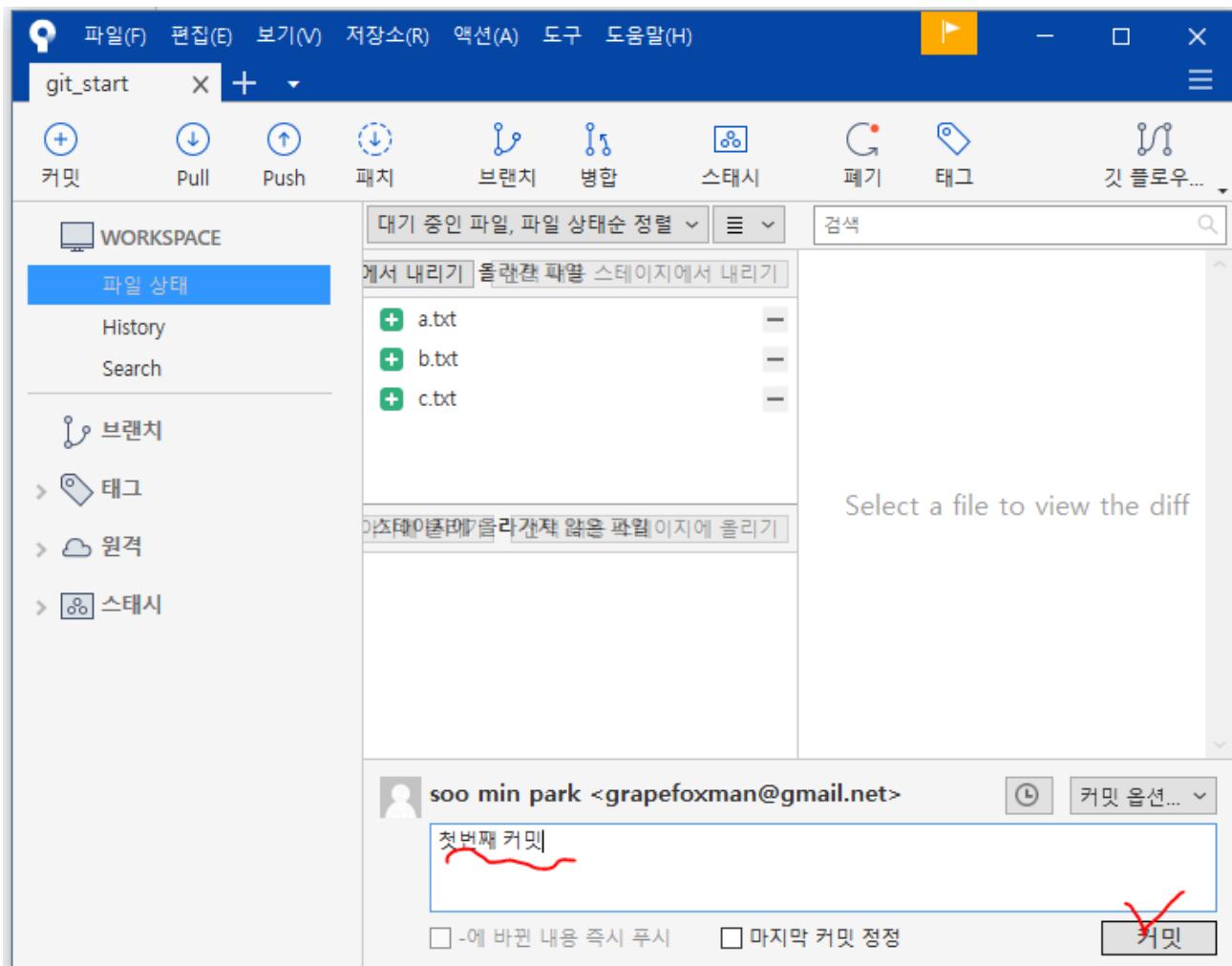
해당 폴더에 a,b,c파일을 만들고 각각 a,b,c,문자를 기입한다.

해당 폴더가 작업 디렉토리 이다.

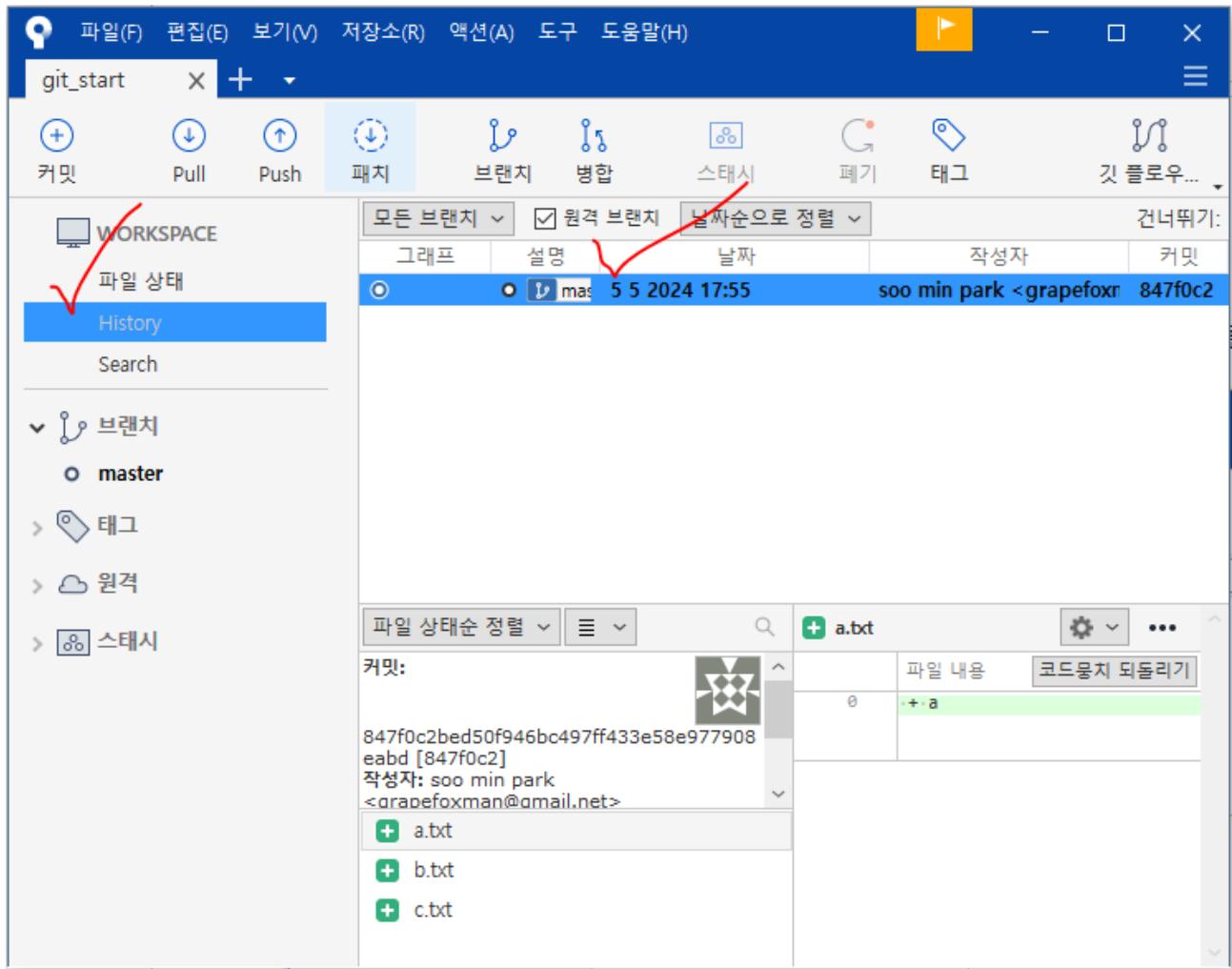
모두 스테이지에 올리기를 선택하거나 + 버튼을 눌러서 스테이지에 올린다.



커밋을 위해서 반드시 버전을 설명하는 메시지를 작성하고 하단의 커밋 버튼을 누른다.



왼쪽 탭에서 history 오른 쪽 장에서 해당 커밋을 선택한 커밋을 확인할 수 있다.

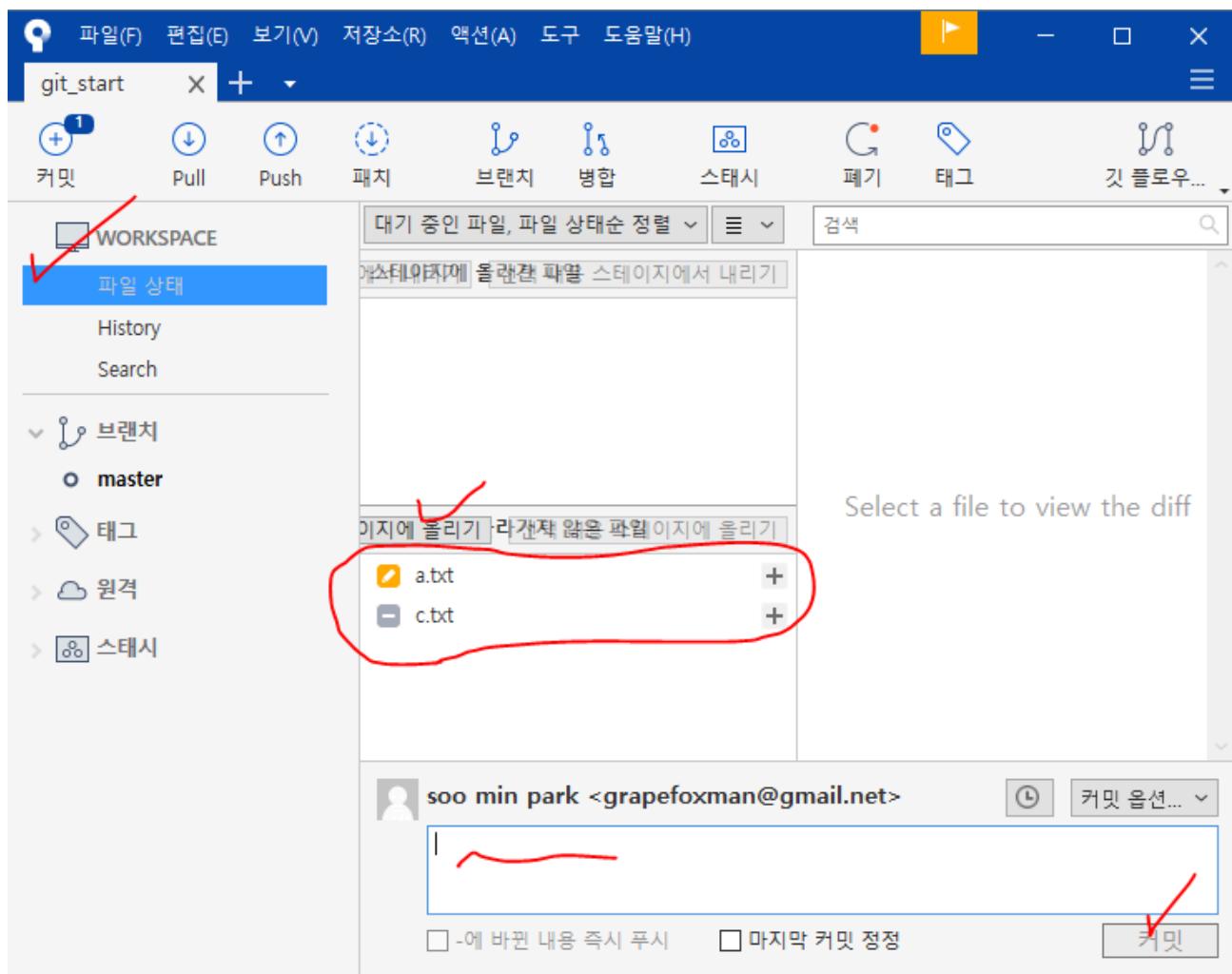


커밋 추가하기

a.txt를 수정하고

c.txt를 삭제해보자.

수정 파일을 스테이지에 올린 다음 커밋을 작성하고 커밋하자.



history에 두번째 커밋을 확인할 수 있다.

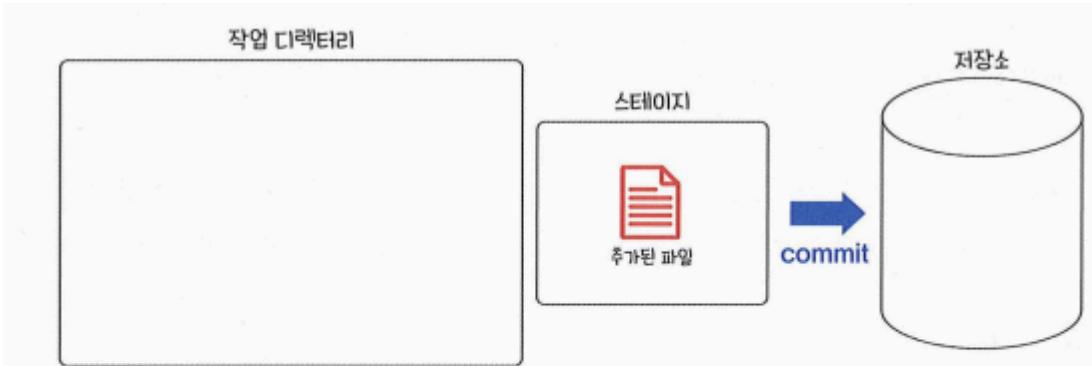


그림 2-7 | 스테이지에 있는 변경 사항을 커밋하여 버전 만들기

정리하자면, 작업 디렉터리의 파일은

1 | 변경 사항 생성

2 | add

3 | commit

의 과정을 통해

1 | 작업 디렉터리

2 | 스테이지

3 | 저장소

순으로 이동하며 새로운 버전으로 만들어집니다.

작업 디렉터리 (Working Directory):

- 작업 디렉터리는 실제 파일이 존재하는 디렉터리이며, 사용자가 파일을 수정하고 새로운 파일을 만드는 곳입니다.
- 변경사항을 추적하고 버전 관리하는데 사용됩니다.

스테이징 영역 (Staging Area 또는 Index):

- 스테이징 영역은 작업 디렉터리와 저장소 사이의 중간 영역으로, 사용자가 준비된 변경사항을 임시로 저장하는 곳입니다.
- 변경사항이 스테이징 영역에 추가되면, 이 변경사항들이 다음 커밋에 포함될 것임을 나타냅니다.

저장소 (Repository):

- 저장소는 프로젝트의 모든 파일과 해당 파일의 변경 이력을 저장하는 곳입니다.
- 일반적으로 로컬 저장소와 원격 저장소 두 가지가 있습니다. 로컬 저장소는

개발자의 컴퓨터에 위치하며, 원격 저장소는 인터넷상의 서버에 위치합니다.

Git을 사용할 때, 작업 디렉터리에서 파일을 수정하고 저장소에 커밋하기 전에 변경사항을 스테이징 영역에 추가합니다. 이를 통해 여러 개의 변경사항을 논리적으로 그룹화하여 커밋할 수 있습니다.

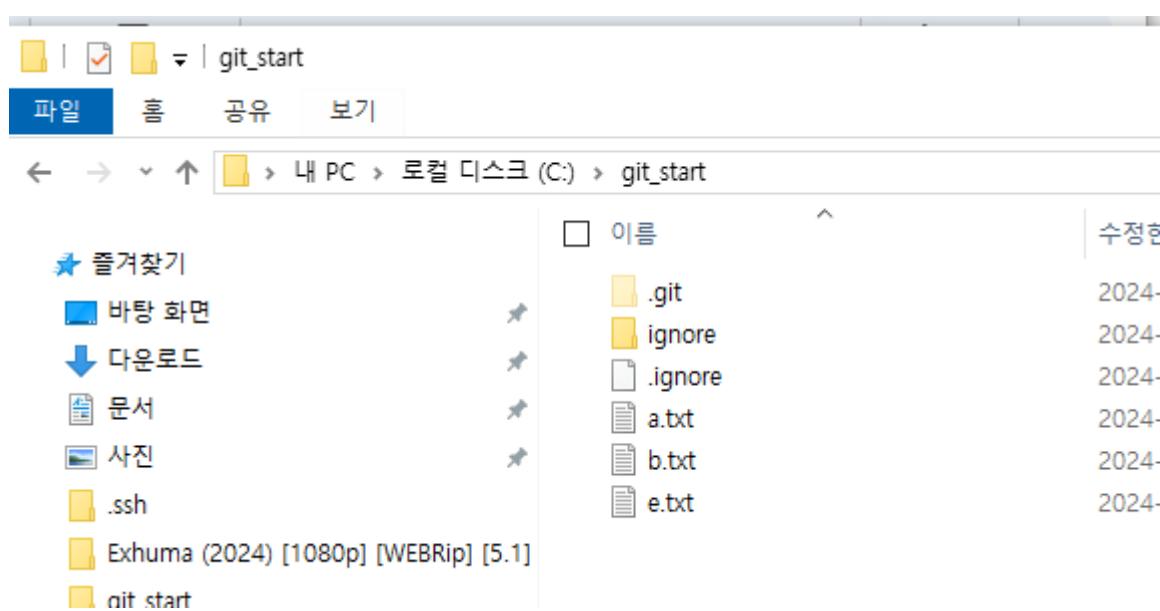
```
add : staging 명령어  
      - git add .  
commit : staging된 파일들을 확정  
      - git commit -m "메시지"  
push : commit된 파일을 원격 저장소로 업로드  
      - git push origin master  
fetch : 원격 저장소의 정보로 업데이트  
      - git fetch  
pull : 원격 저장소의 변경된 파일을 다운로드해서 로컬 파일과 동기화  
      - git pull  
  
clone : 원격 저장소의 모든 내용을 내 저장소로 다운로드  
branch : push가 따로 관리되는 복사본
```

```
commit : staging된 파일들을 확정  
      - git commit -m "메시지"  
push : commit된 파일을 원격 저장소로 업로드  
      - git push origin master  
fetch : 원격 저장소의 정보로 업데이트  
      - git fetch  
pull : 원격 저장소의 변경된 파일을 다운로드해서 로컬 파일과 동기화  
      - git pull  
clone : 원격 저장소의 모든 내용을 내 저장소로 다운로드  
  
branch : push가 따로 관리되는 복사본  
checkout : branch 변경  
merge : branch 병합  
fork : branch이긴 하지만 push가 즉각 반영되지 않음  
pull request : fork의 내용을 승인해달라는 요청 pull req 풀리퀘
```

e.txt

ignore/

.gitignore파일에 상위 처럼 기술하면 e.txt와 ignore/폴더의 모든 내용이 git작업에서 무시되어 스테이비에 올라가지 않는다.

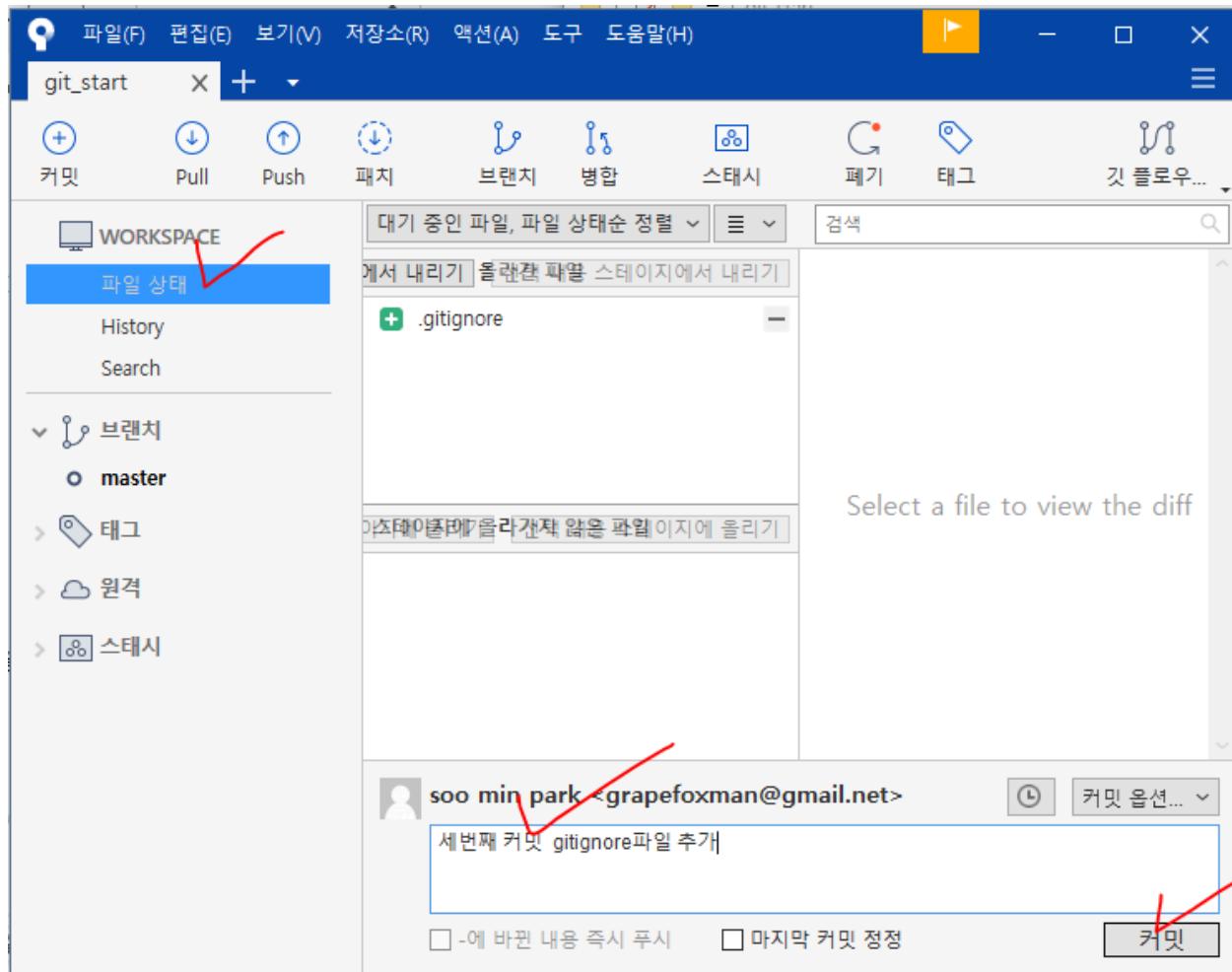


gitignore파일 확인하기.

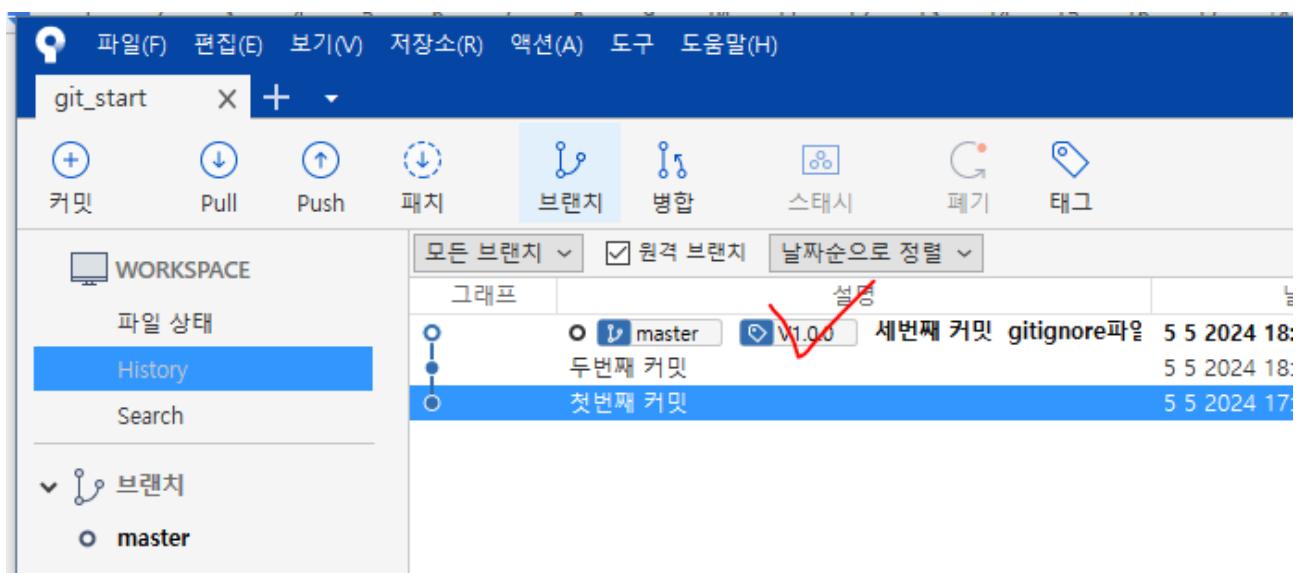
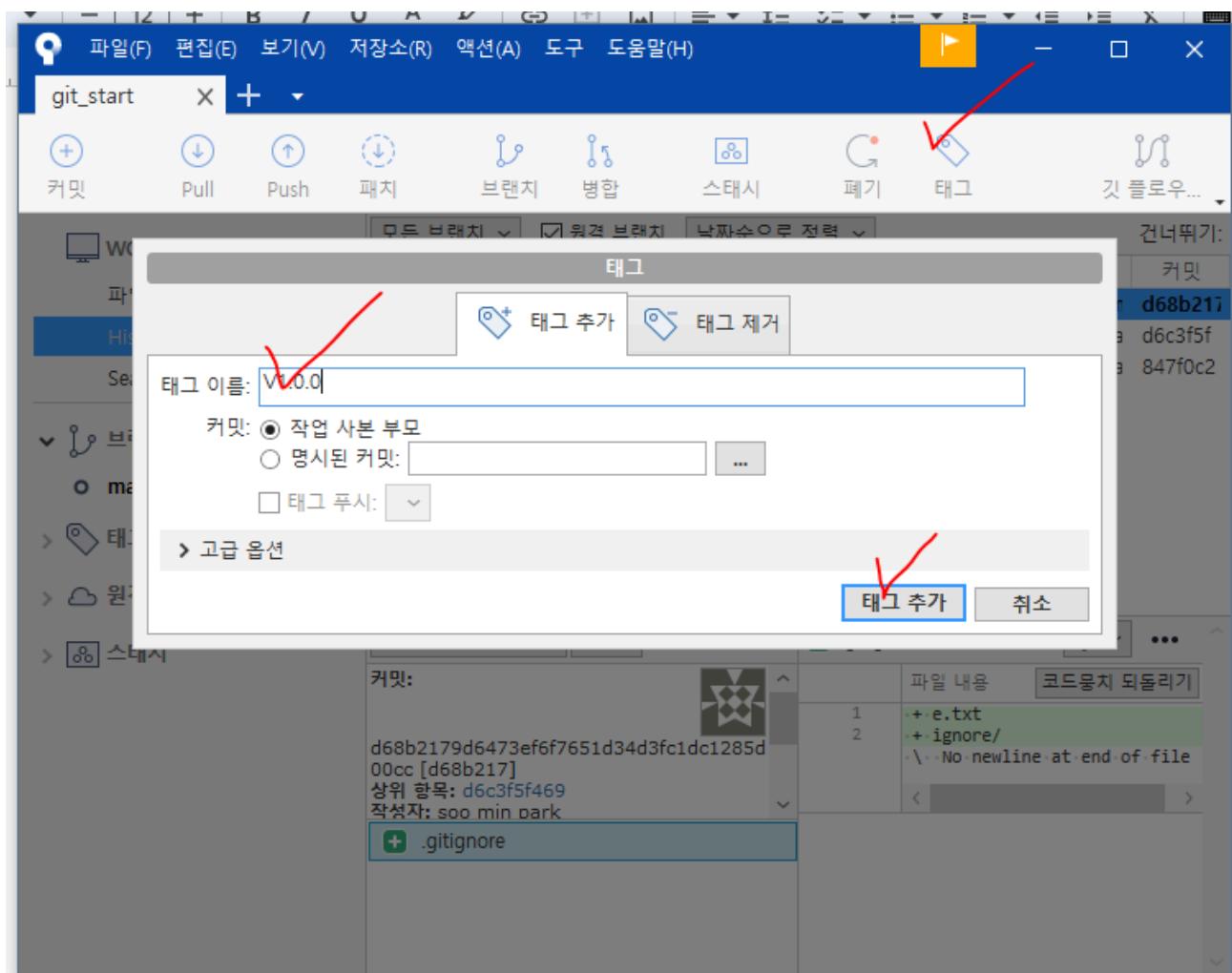
파일 상태 선택

스테이지에 올릴 gitignore파일 선택

커밋내용 입력 및 커밋



Git 태그는 프로젝트의 특정 지점을 식별하고 마킹하는 데 사용됩니다.



스테이지에 올라간 파일 폐기 삭제 하기

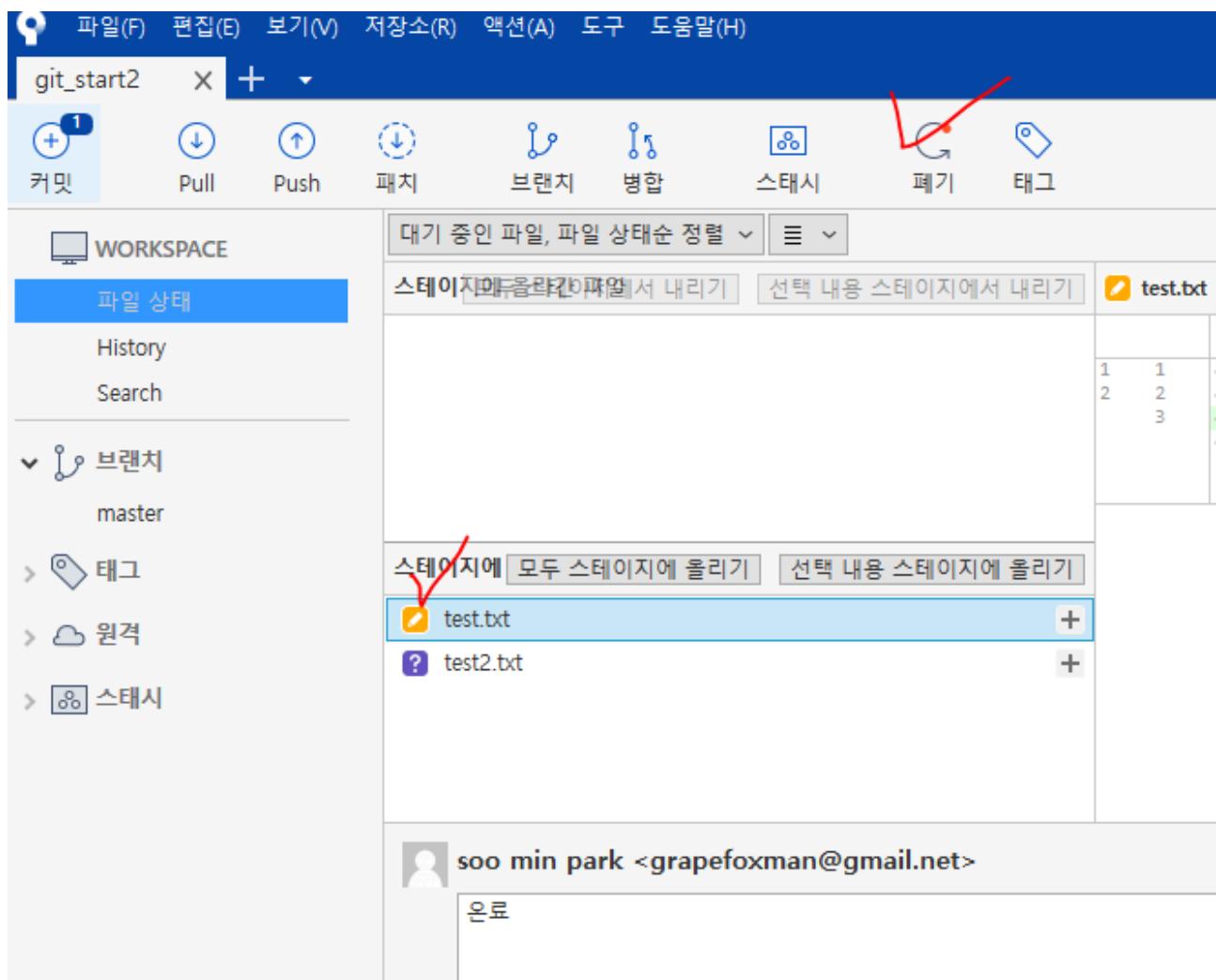
"제거"는 파일이나 디렉토리를 로컬 저장소에서 삭제하는 것을 의미합니다

"폐기"는 변경 사항을 롤백하여 이전 상태로 되돌리는 것을 의미합니다.

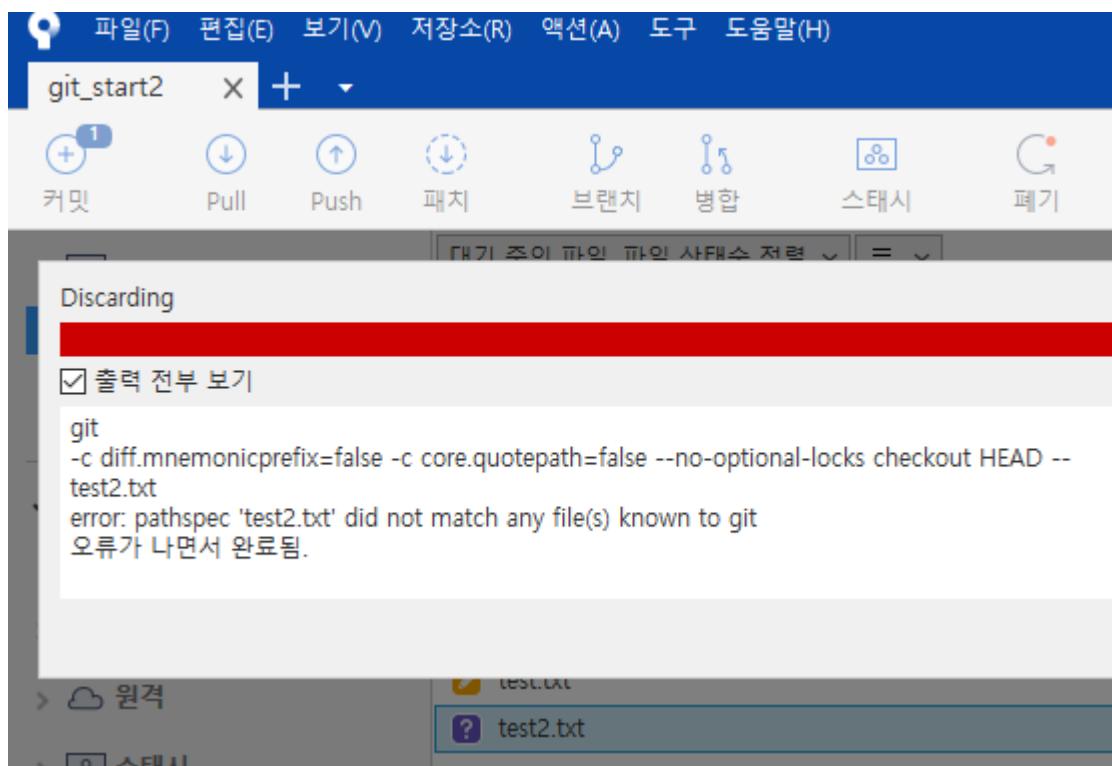
test.txt에 문자열 c추가

test2.txt파일에 a추가

다음 처럼 test.txt 파일을 생성한 다음 상단 폐기 버튼을 누르면 해당 파일에 지금까지 작업한 내용이 취소 되고 test.txt파일의 이전 커밋 작업까지 복귀된다.

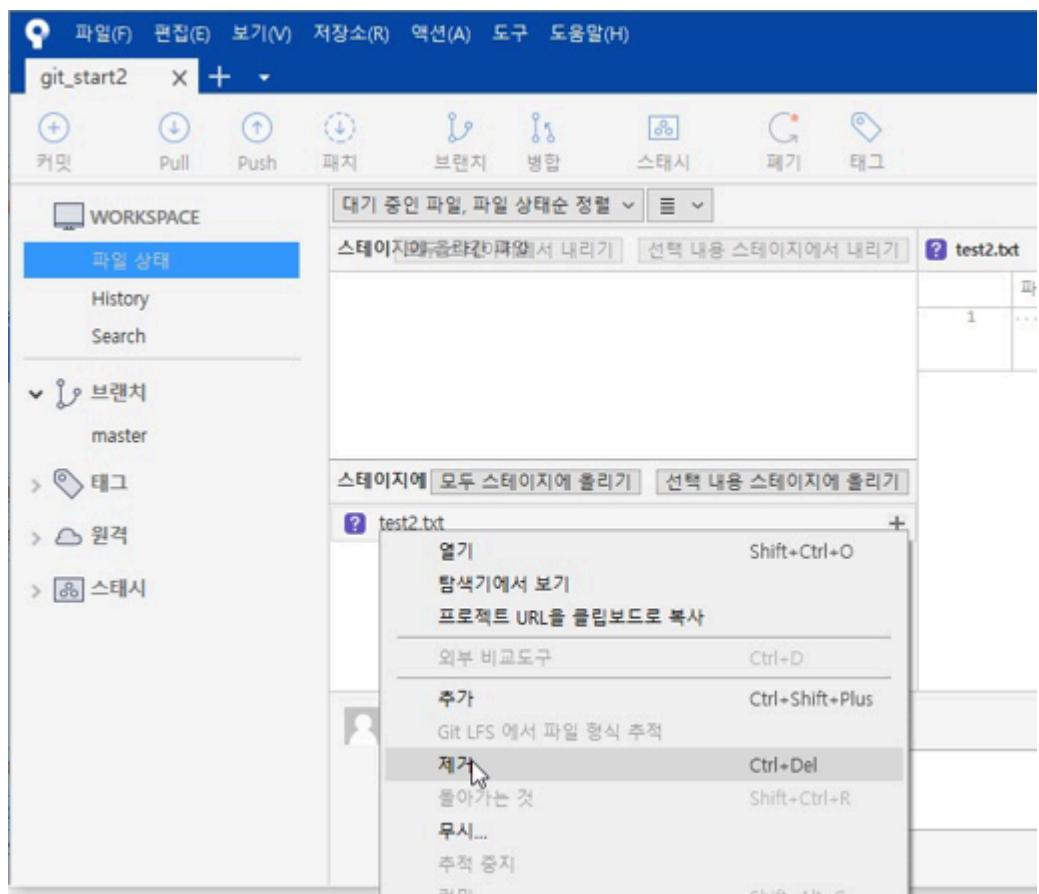


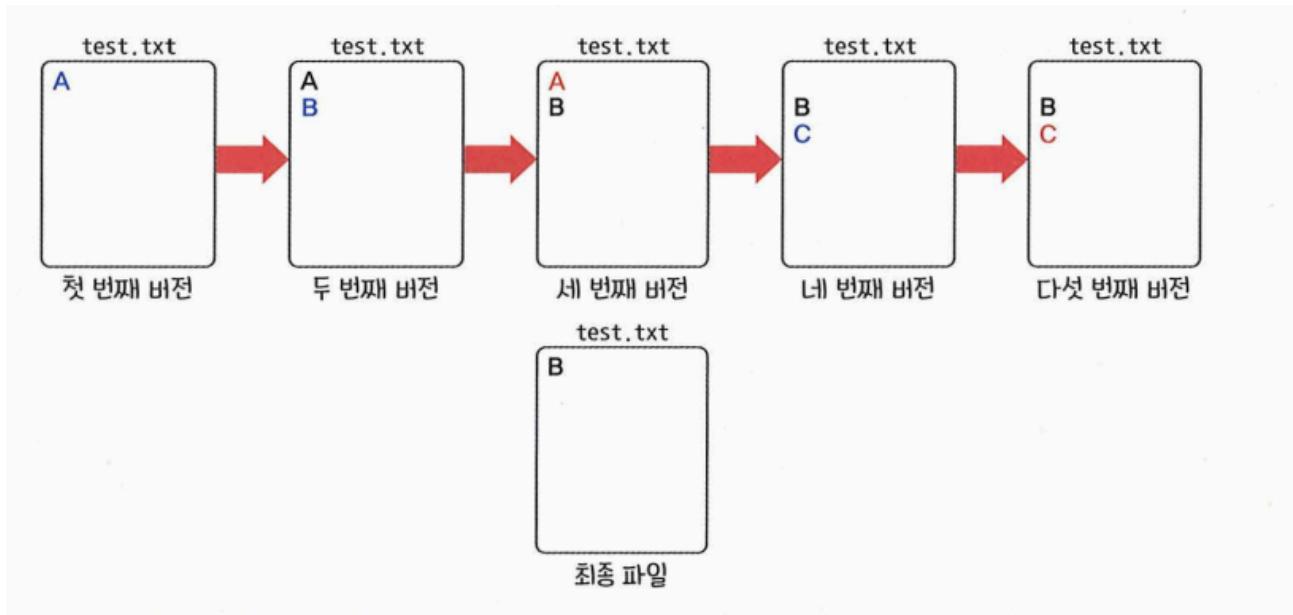
test2.txt처럼 처음 만든 파일은 폐기가 불가능해서 다음과 같은 에러 메시지가 발생한다.



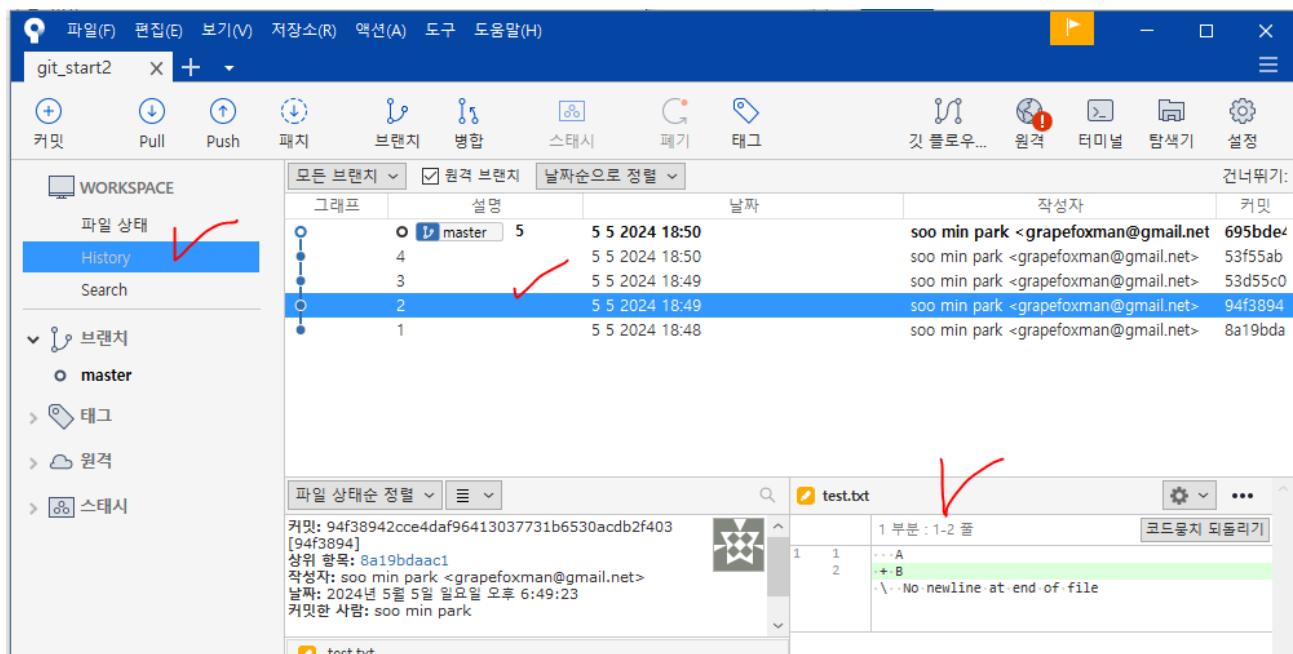
삭제하는 방법은 다음 이미지처럼 파일에서 오른쪽 마우스 클릭한 다음에 삭제를 선택하면 된다.

삭제하게 되면 선택된 파일이 완전히 사라지게 된다.





각각의 파일 내용을 알고 싶으면 커밋 지점의 항목을 선택하면 오른쪽 창에 커밋 시점의 파일을 확인할 수 있다.



커밋 되돌리기

revert과 reset

revert는 기존의 버전은 삭제되지 않는다.

reset은 기존 버전은 삭제되고 이전버전으로 이동한다.

- "Revert"는 특정한 커밋을 선택하여 그 커밋에서의 변경 사항을 취소하는 것입니다.
- "Revert"를 사용하면 이전 상태로 돌아갈 수 있지만, 이전 커밋은 그대로 유지됩니다.

"Reset"은 현재 브랜치의 위치를 이전 상태로 이동시키는 것입니다.

"Reset"은 세 가지 모드로 사용할 수 있습니다: 소프트(Soft), 믹스드(Mixed), 하드(Hard).

- 소프트 리셋은 커밋만 이동시키고 변경 사항은 그대로 남깁니다.
- 믹스드 리셋은 커밋과 인덱스를 이동시키고 변경 사항은 작업 디렉토리에 남깁니다.
- 하드 리셋은 커밋, 인덱스, 작업 디렉토리의 변경 사항을 모두 제거합니다.

복잡하니 초보자는 hard로 익숙해질때 까지 사용하자.

"Reset"을 사용하면 선택된 커밋 상태로 변환하고 사이에 있던 커밋 정보는 삭제된다.

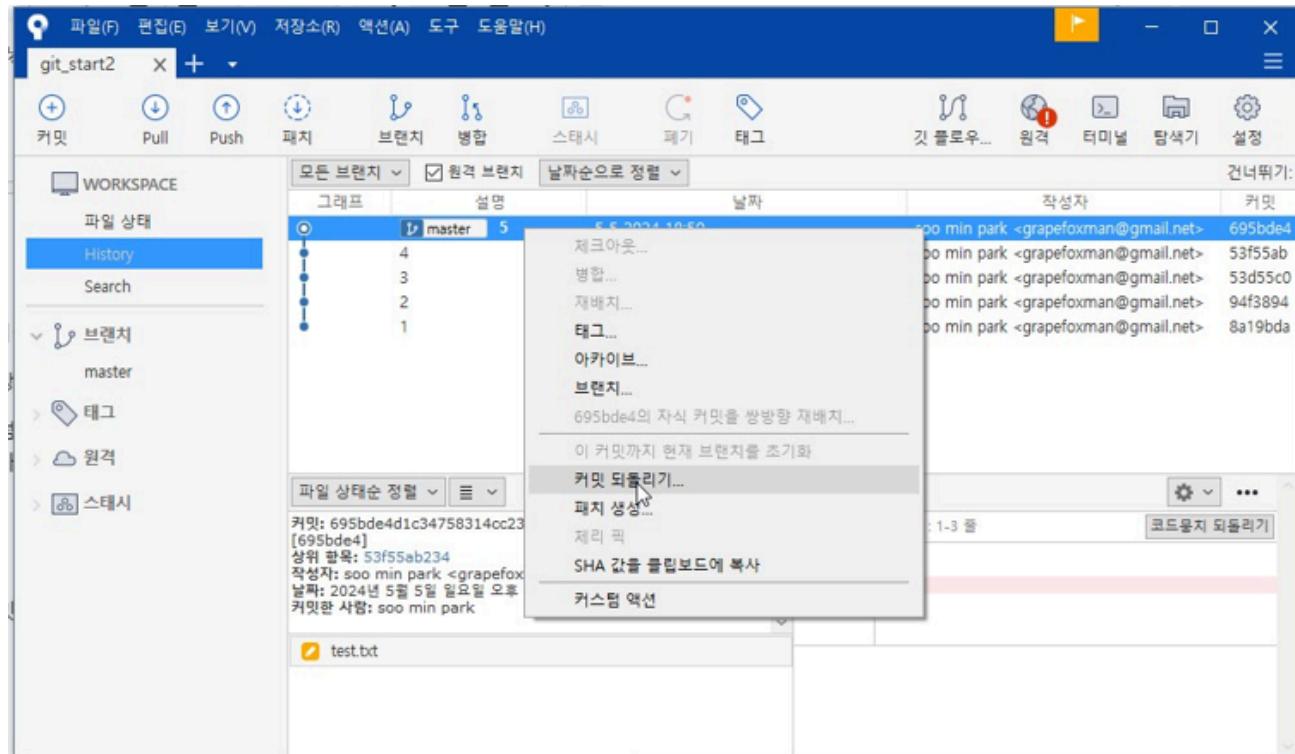
한번에 여러단개의 커밋을 뛰어 넘어 적용하려고 하면 충돌 문제가 발생할 수 있으니
되도록 하나씩 이전 단계로 돌아가자.

revers

기록을 남기고 이전 상태로 돌아가고자 할때 사용한다.

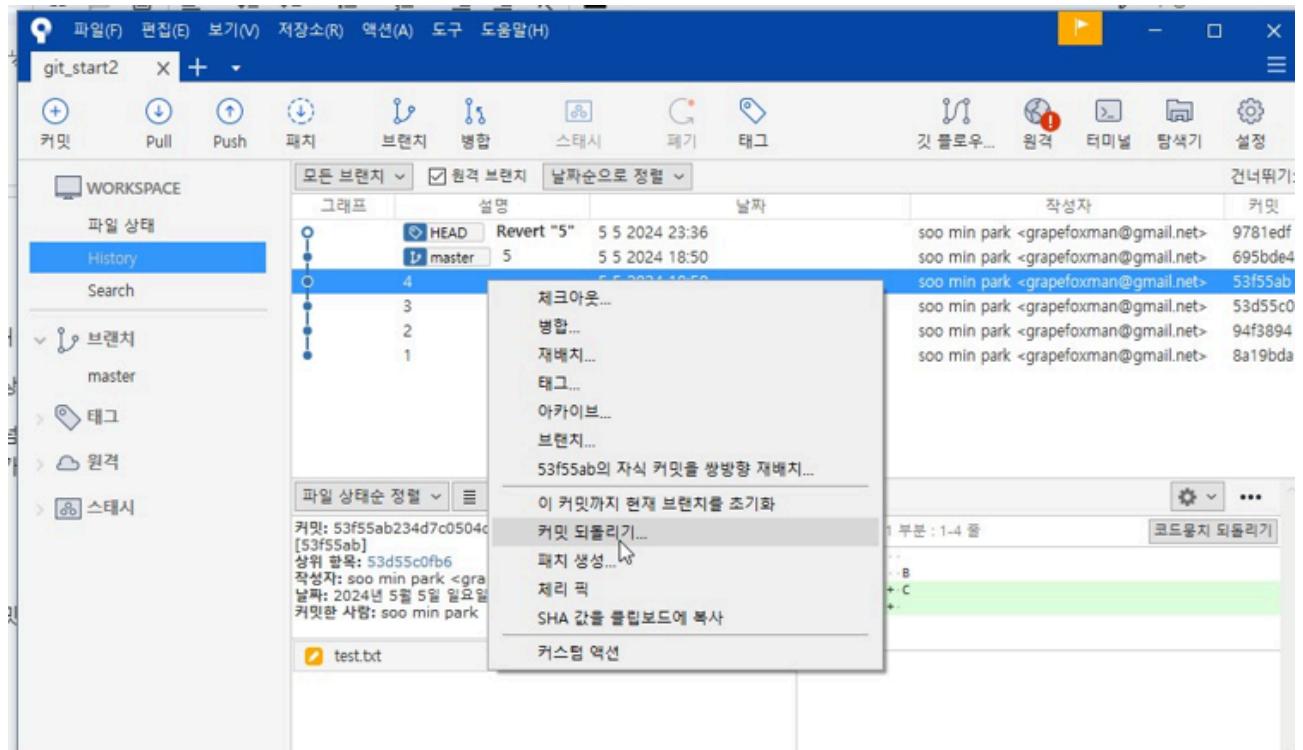
커밋에서 오른쪽 마우스 클릭한다음 이커밋 까지 현재 브랜치를 초기화를 선택한다.

5를 선택한 다음 오른쪽 클릭 커밋되돌리기를 선택한다.

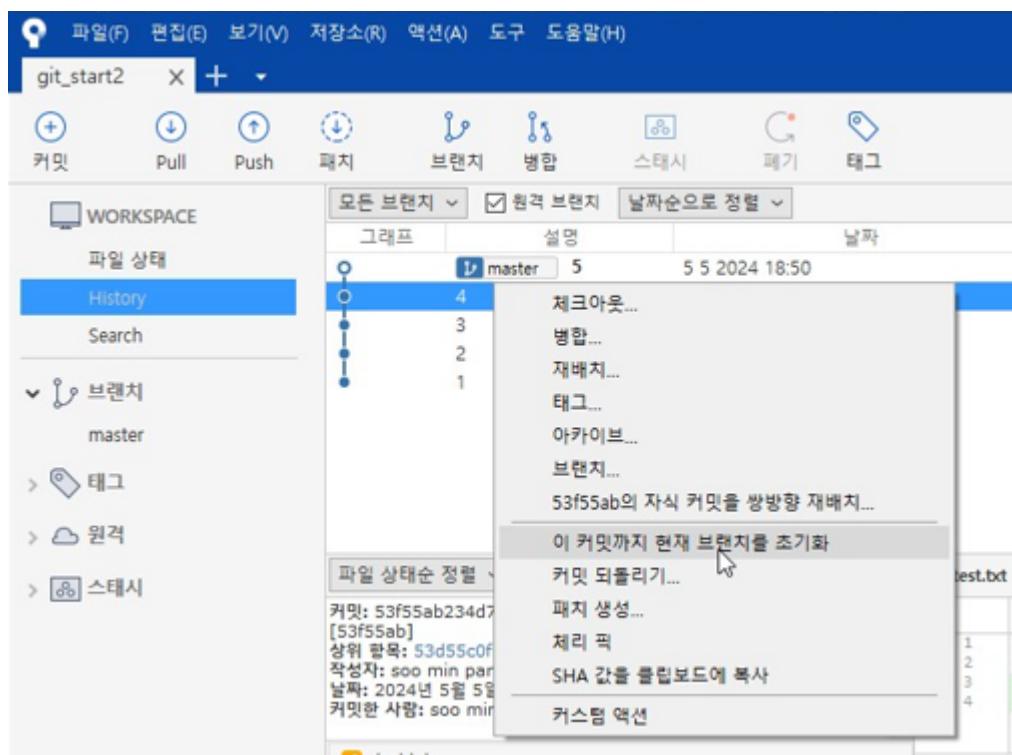


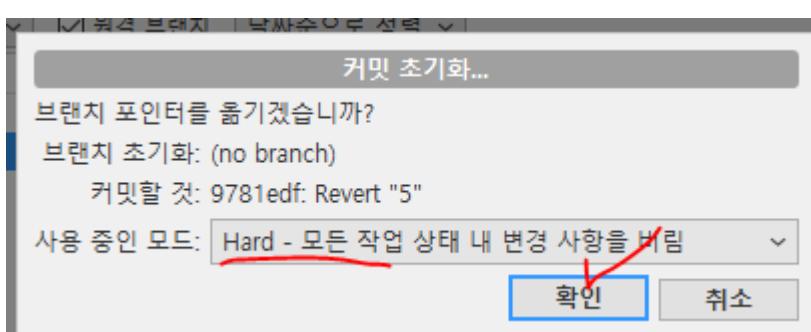
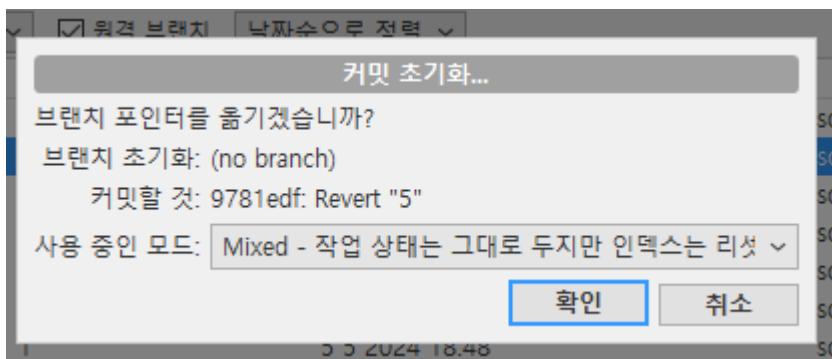
새로 만들어진 커밋은 커밋4와 동일해진다.

커밋 3으로 복귀하고 싶다면 4를 선택한 다음 오른쪽 마우스를 클릭해서 커밋 되돌리기를 선택한다.



특정 커밋으로 reset하는 방법은 돌아가고 싶은 커밋에 가서 오른쪽 마우스 클릭 이 커밋 까지 현재 브랜치를 초기화를 선택하면 된다. 충돌 나지 않도록 하나씩 진행하자.

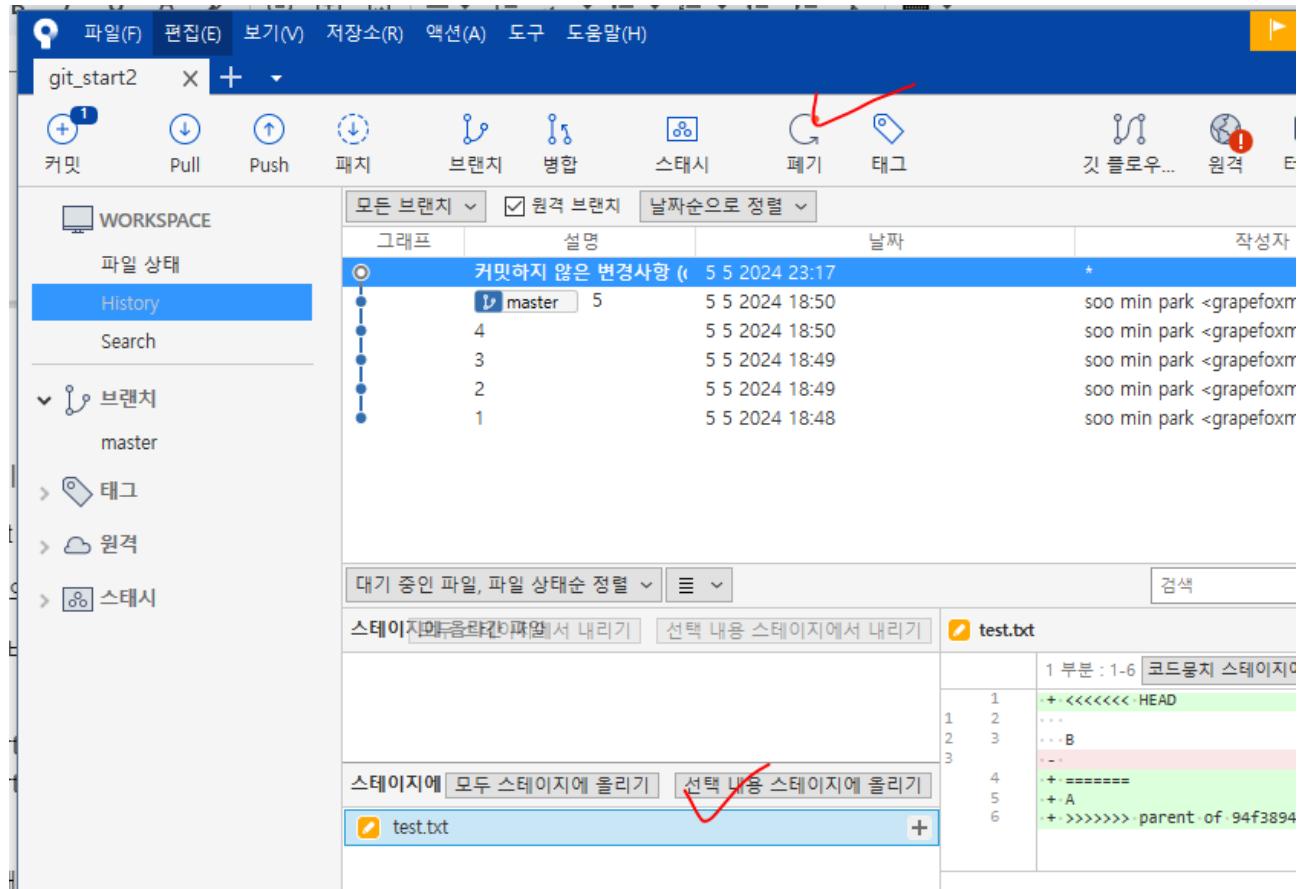




되도록 hard로 하고 확인을 누르자.

돌아 갈때 충돌이 생기면 다음과 같지 커밋되지 않은 정보가 남을수 있다.

커밋되지 않은 정보를 클릭해서 폐기하거나 다시 커밋해서 적용시키면 된다.



충돌 해결 방법

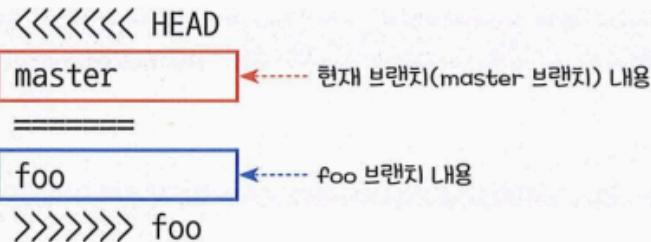


그림 4-68 | a.txt 파일의 충돌

충돌이 발생한 파일에는 <<<<<, ===== 기호가 표기됩니다.

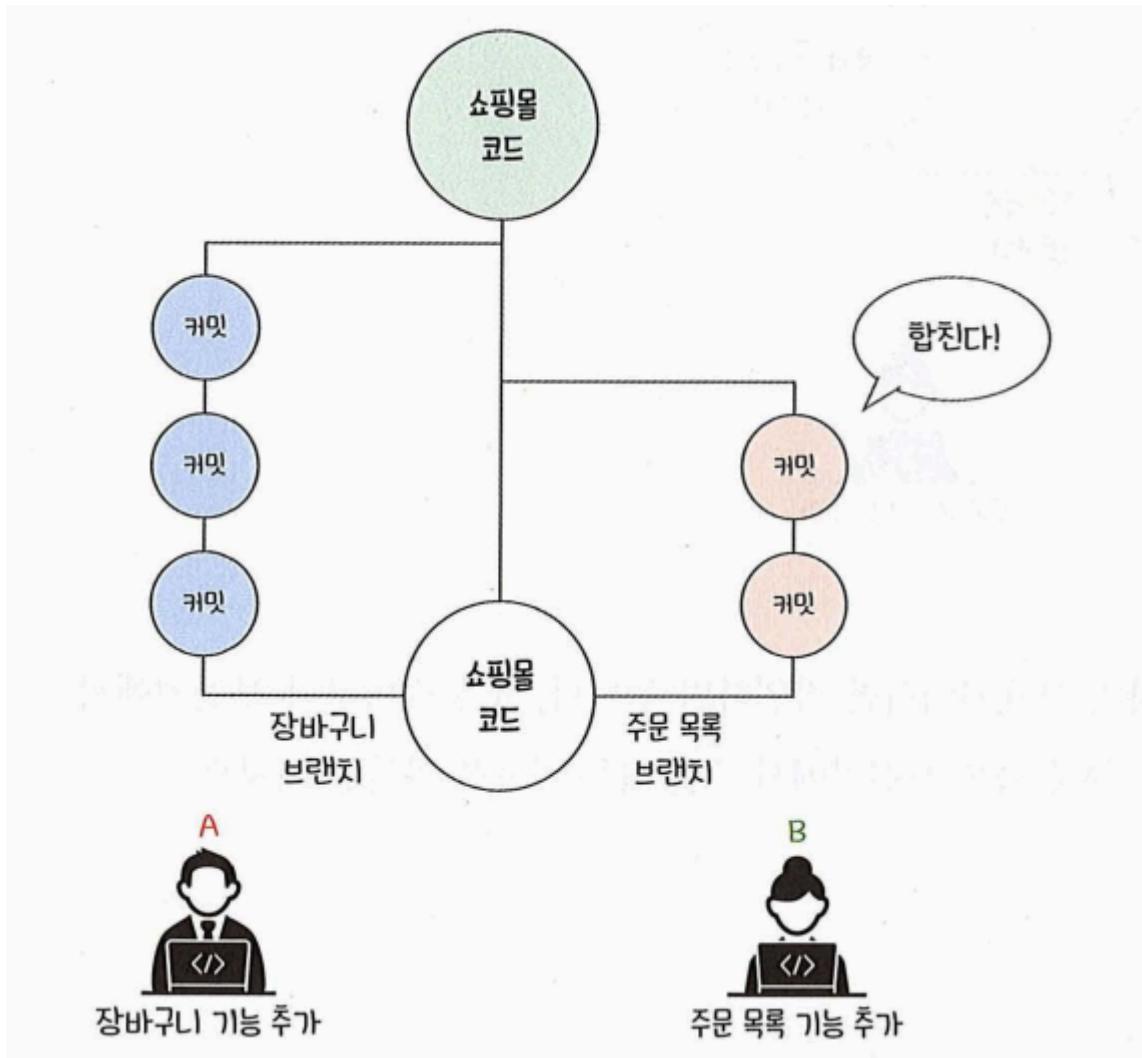
이 기호는 일종의 영역 표기입니다. ===== 기호를 기준으로 윗부분(그림 4-68에서 붉은색 박스)은 HEAD가 가리키는 브랜치, 즉 현재 체크아웃한 브랜치의 내용이 적혀 있고, 아랫부분(그림 4-68에서 파란색 박스)은 병합하려는 브랜치, 즉 foo 브랜치의 내용이 적혀 있습니다.

이는 <<<<< 기호와 ===== 기호 사이의 내용을 선택할지, ===== 기호와 >>>>> 기호 사이의 내용을 선택할지 고르라는 표기입니다. 여러분은 이 두 영역 중 반영할 부분을 직접 선택해 충돌을 해결해야 합니다.

브랜치 사용하기

하나의 프로젝트를 분리된 상태에서 작업하고 싶을 때 사용한다.

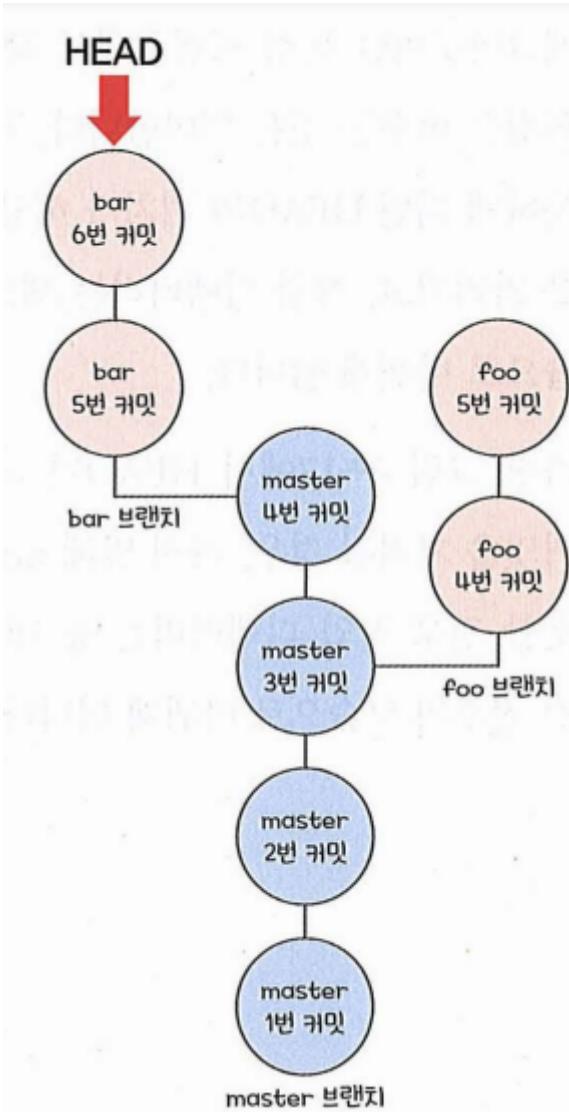
브랜치는 Git에서 코드를 분리하여 개발하고 관리하는 방법입니다. 각 브랜치는 독립적인 작업 영역이며, 여러 작업을 병행할 수 있게 해줍니다. 주로 사용되는 브랜치에는 "master"라는 메인 브랜치가 있습니다. 개발자는 새로운 기능을 추가하거나 버그를 수정할 때마다 새로운 브랜치를 만들어 작업하고, 작업이 완료되면 메인 브랜치로 병합합니다.



최초의 브랜치를 master라 한다.

head란 현재 작업중인 브랜치를 의미한다.

cheak out이란? 다른 브랜치로 작업환경을 변경하는 것을 의미한다.



a.txt 1

b.txt 2

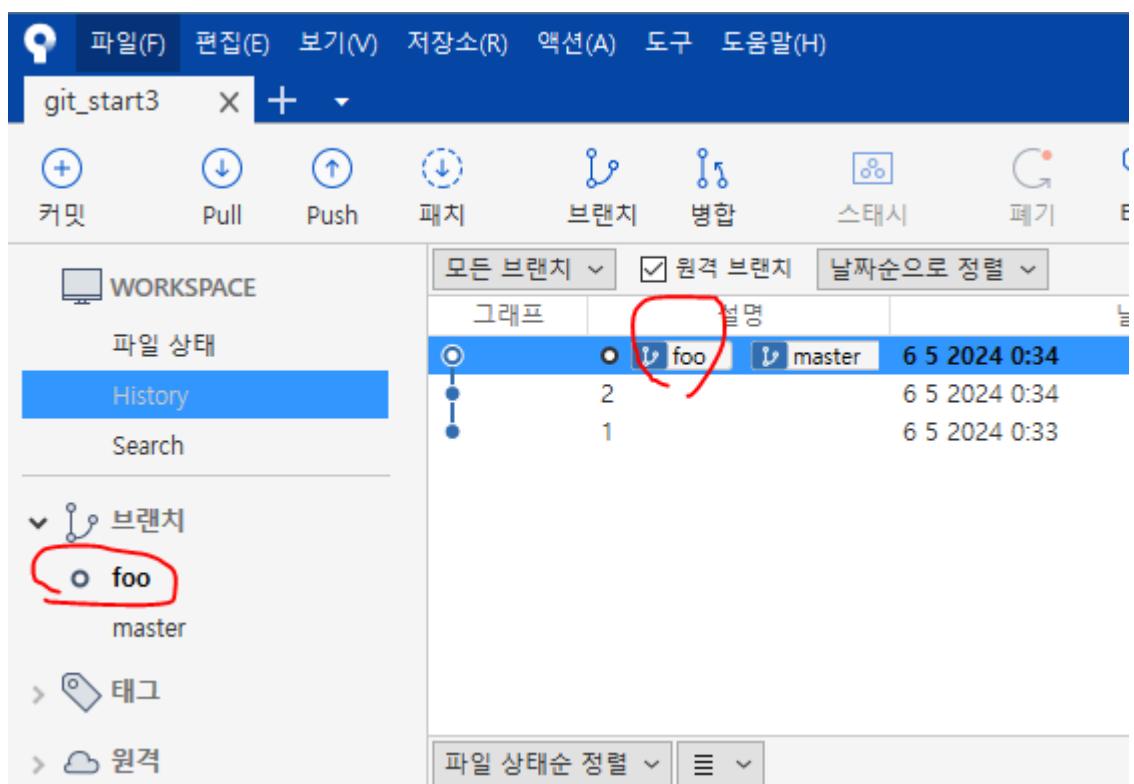
c.txt 3 파일을 만들어 커밋 3개를 만든다.

The screenshot shows a Git client interface with the following details:

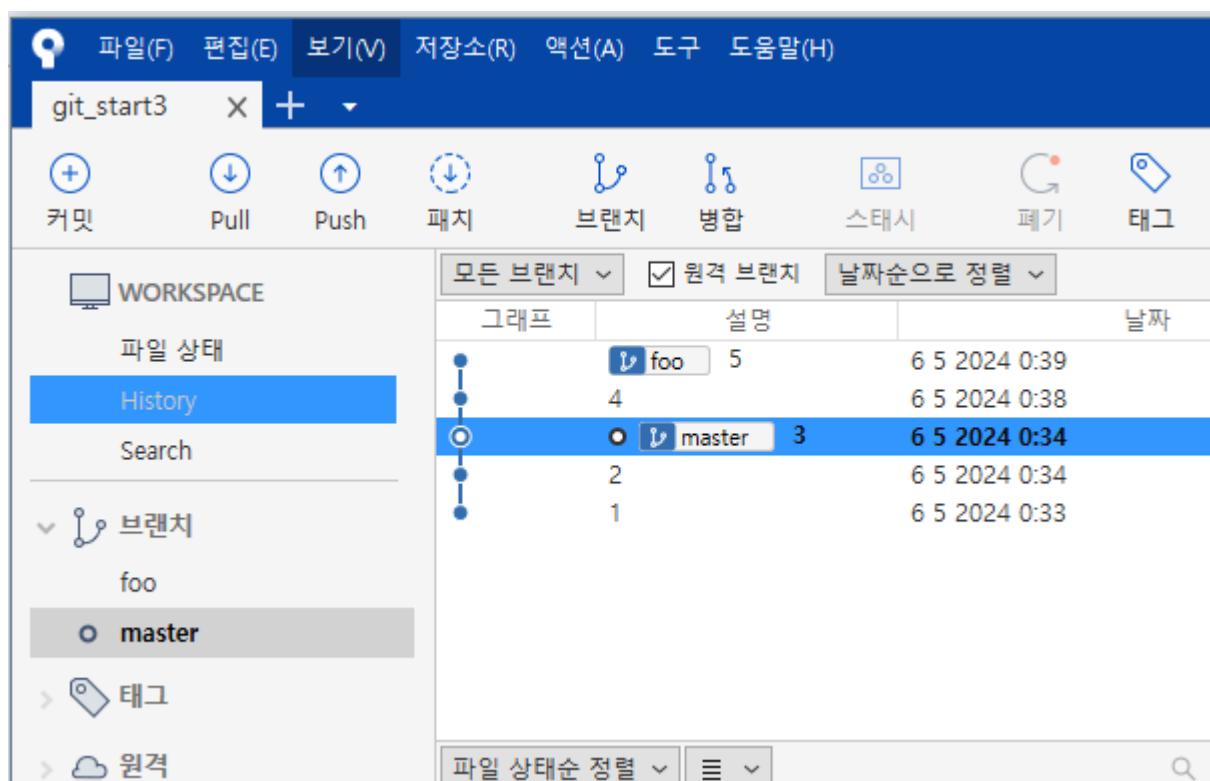
- Toolbar:** 파일(F), 편집(E), 보기(V), 저장소(R), 액션(A), 도구, 도움말(H).
- Branches:** 커밋, Pull, Push, 패치, **브랜치** (highlighted with a red checkmark), 병합, 스테이, 폐기, 태그.
- Left Sidebar:** WORKSPACE, 파일 상태, History (selected), Search, 브랜치 (master selected), 태그, 원격, 스테이.
- History View:** 모든 브랜치, 원격 브랜치, 날짜순으로 정렬. Branch: master, Commit ID: cc3e25afc326c09a6997c3eeb74892c5d285f29f [cc3e25a], Author: soo min park <grapefoxman@gmail.net>, Date: 2024년 5월 6일 월요일 오전 12:34:51, Committer: soo min park.
- File View:** 파일 상태순 정렬, 파일 목록: c.txt.

The screenshot shows a branch creation dialog window:

- Buttons:** 새 브랜치 (highlighted with a red circle), 브랜치 삭제.
- Current Branch:** 현재 브랜치: master.
- New Branch:** 새 브랜치: **foo** (highlighted with a red circle).
- Commit Type:** 커밋: 작업 사본 부모, 명시된 커밋: ..., 새 브랜치 체크아웃.
- Buttons:** 브랜치생성 (highlighted with a red checkmark), 취소.

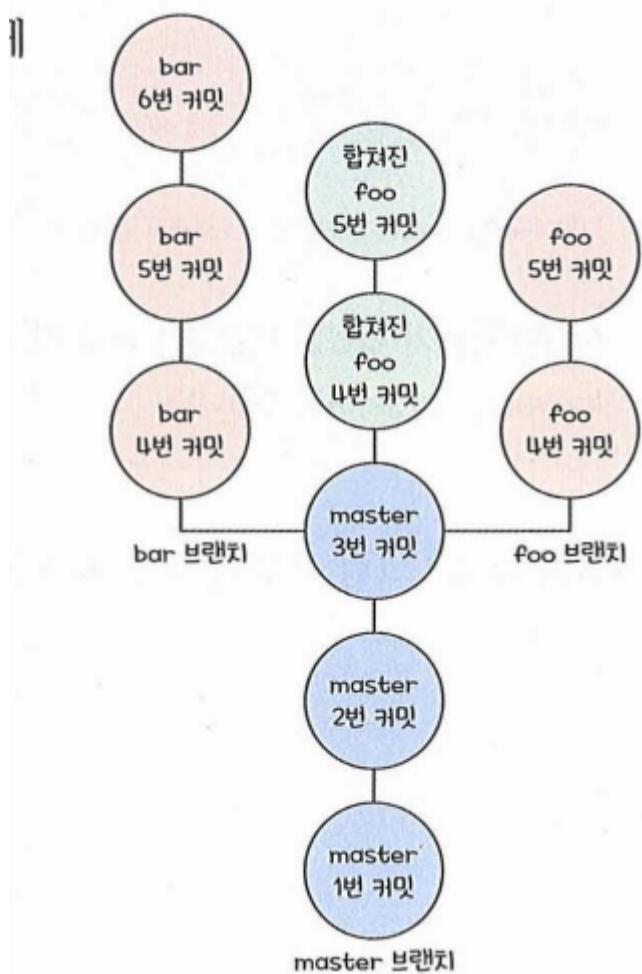


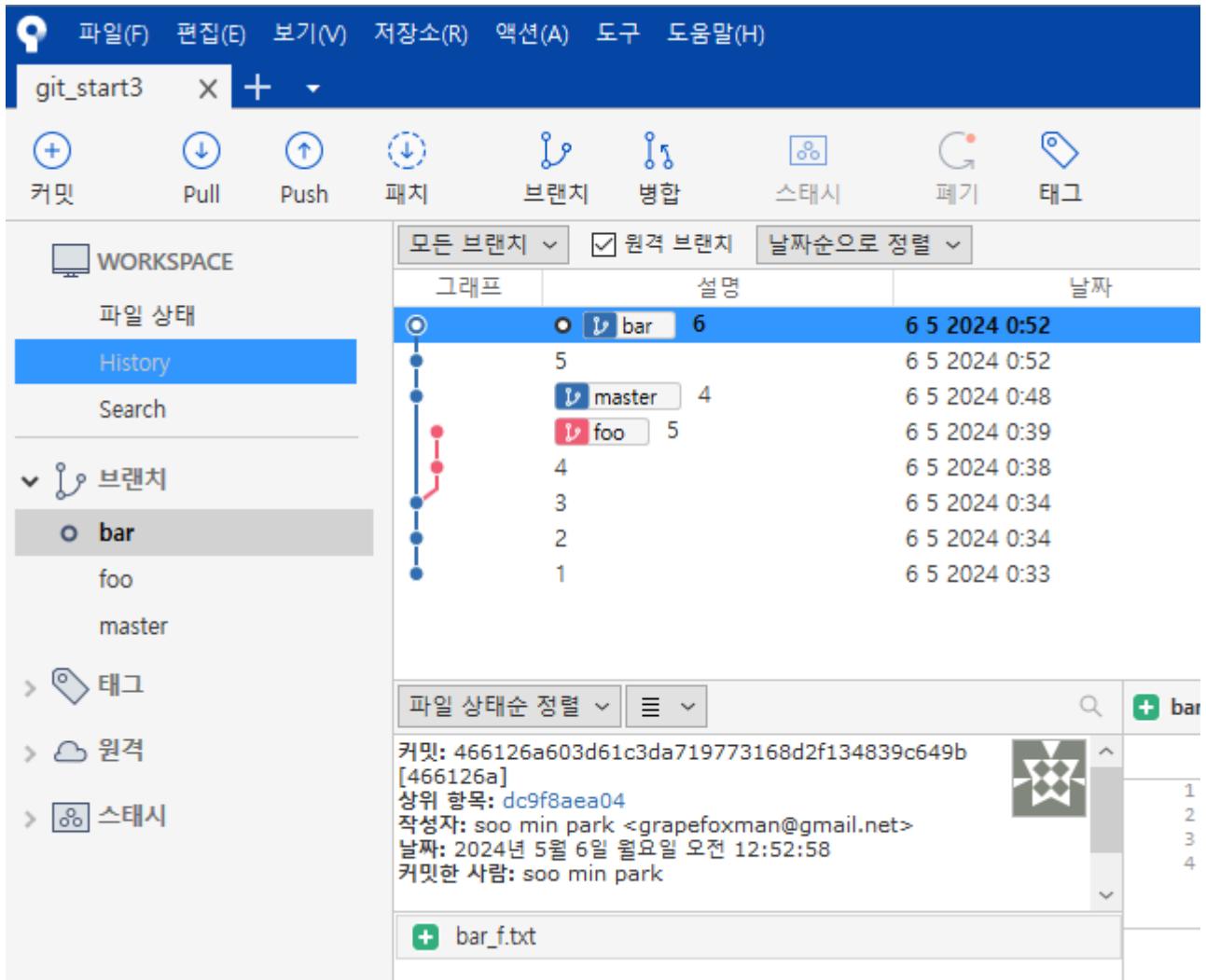
브랜치 foo에서 foo_e.txt 커밋4 추가 foo_e.txt 커밋5 추가



master 브랜치 이동

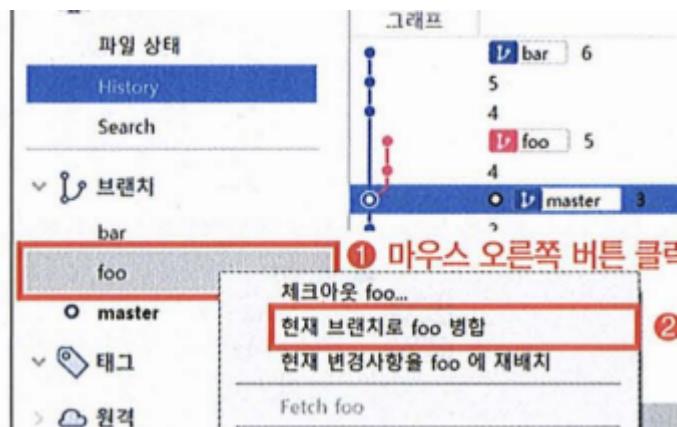
파일 d.txt 추가



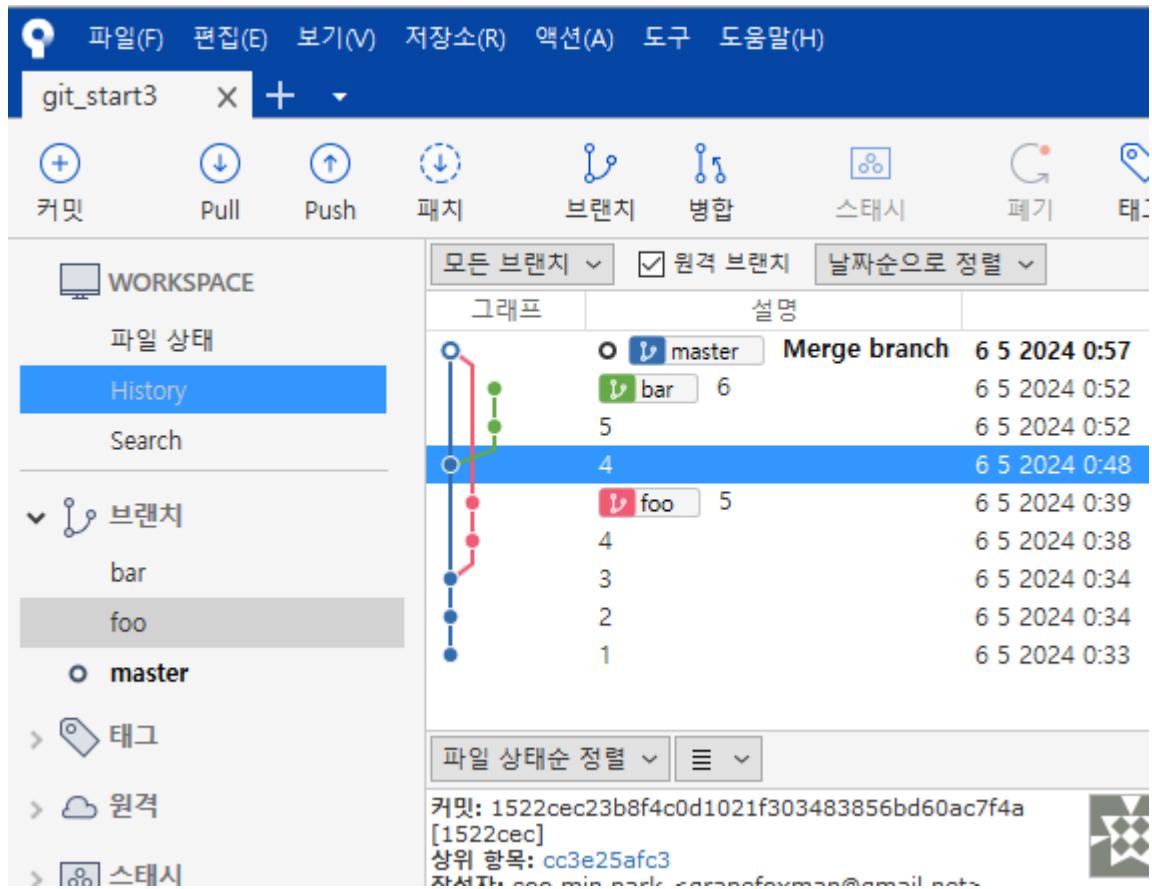


브랜치 병합하는 방법은 다음과 같다.

master를 체크아웃한다음 왼쪽 foo 브랜치에서 마우스 오른쪽 클릭 현재 브랜치로 foo 병합을 선택하면된다.



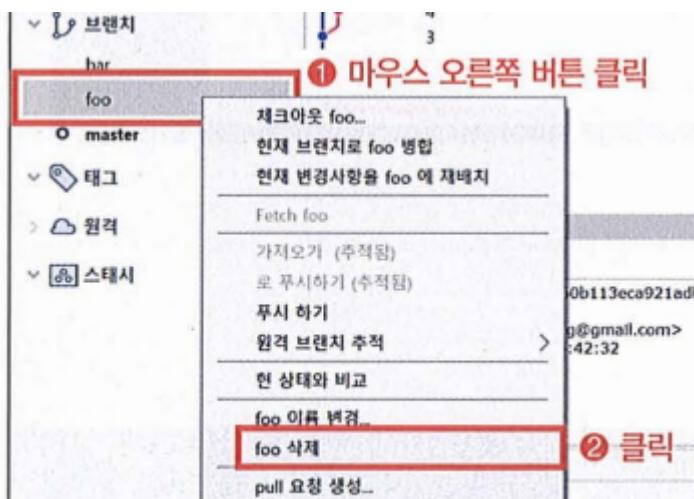
병합되었다.



브랜치 삭제하는 방법

master로 체크아웃한다음에 foo 브랜체에서 마우스 오른쪽 클릭해서 foo 삭제를 선택한다.

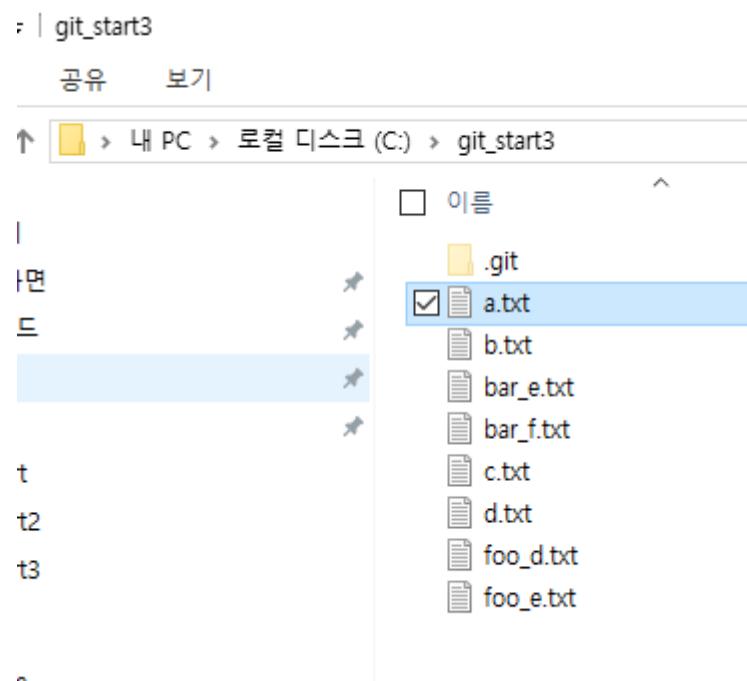
foo선택 상황에서 작업을 진행하면 에러가 발생한다.

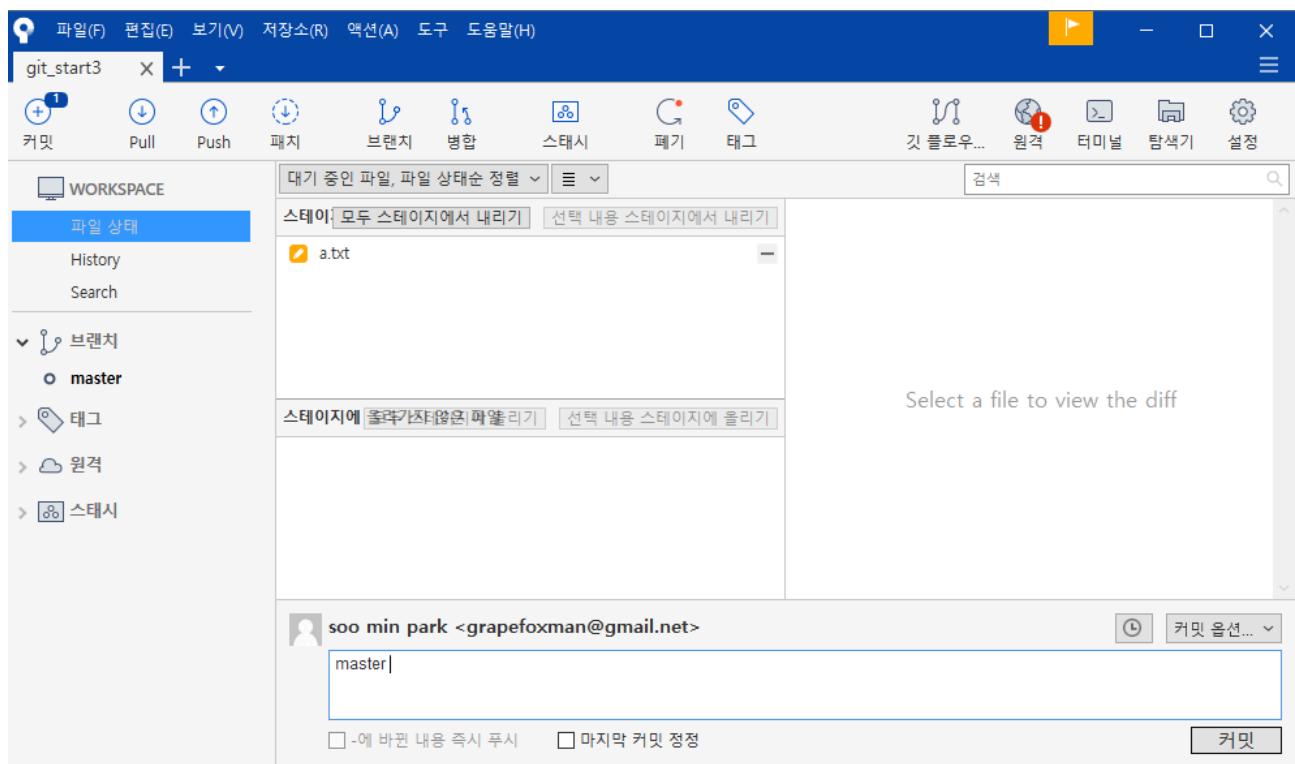


충돌 해결하기

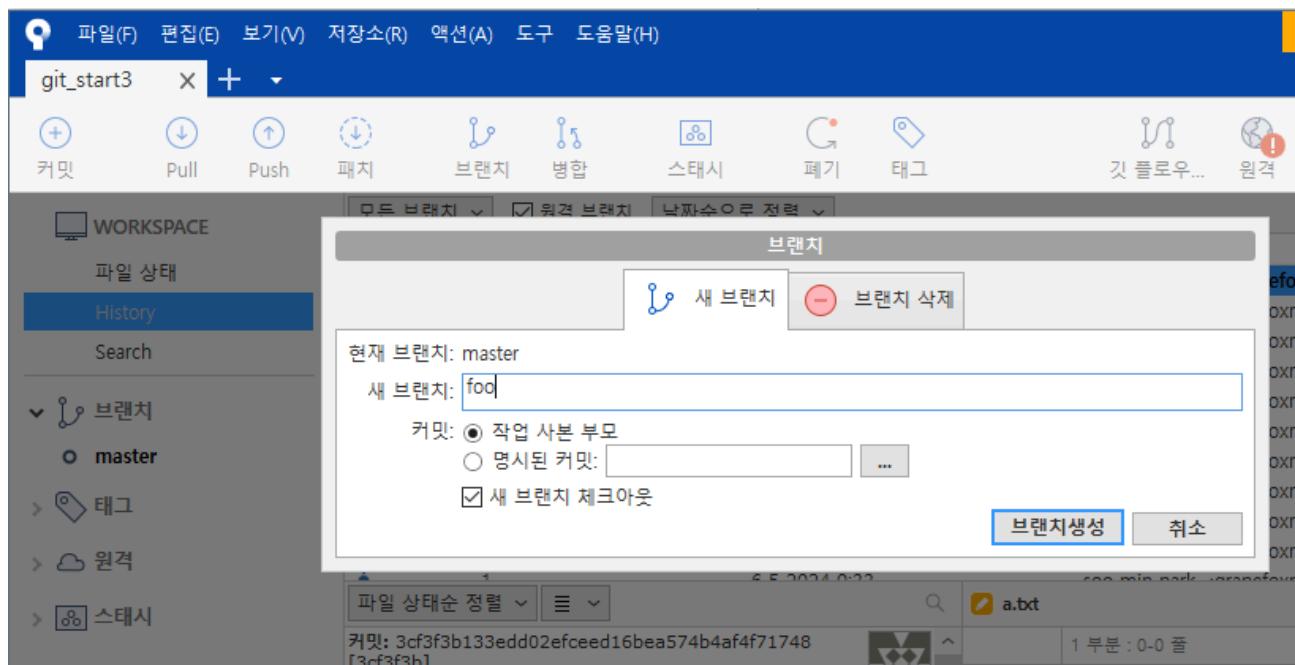
foo 브랜치를 만들어 a.txt를 foo로 변경한다.

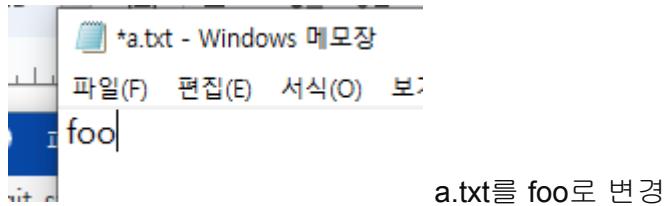
master 브랜치에서 a.txt를 master로 변경한다.





foo 브랜치 생성





파일 작업중 브런치를 꼭 확인할것

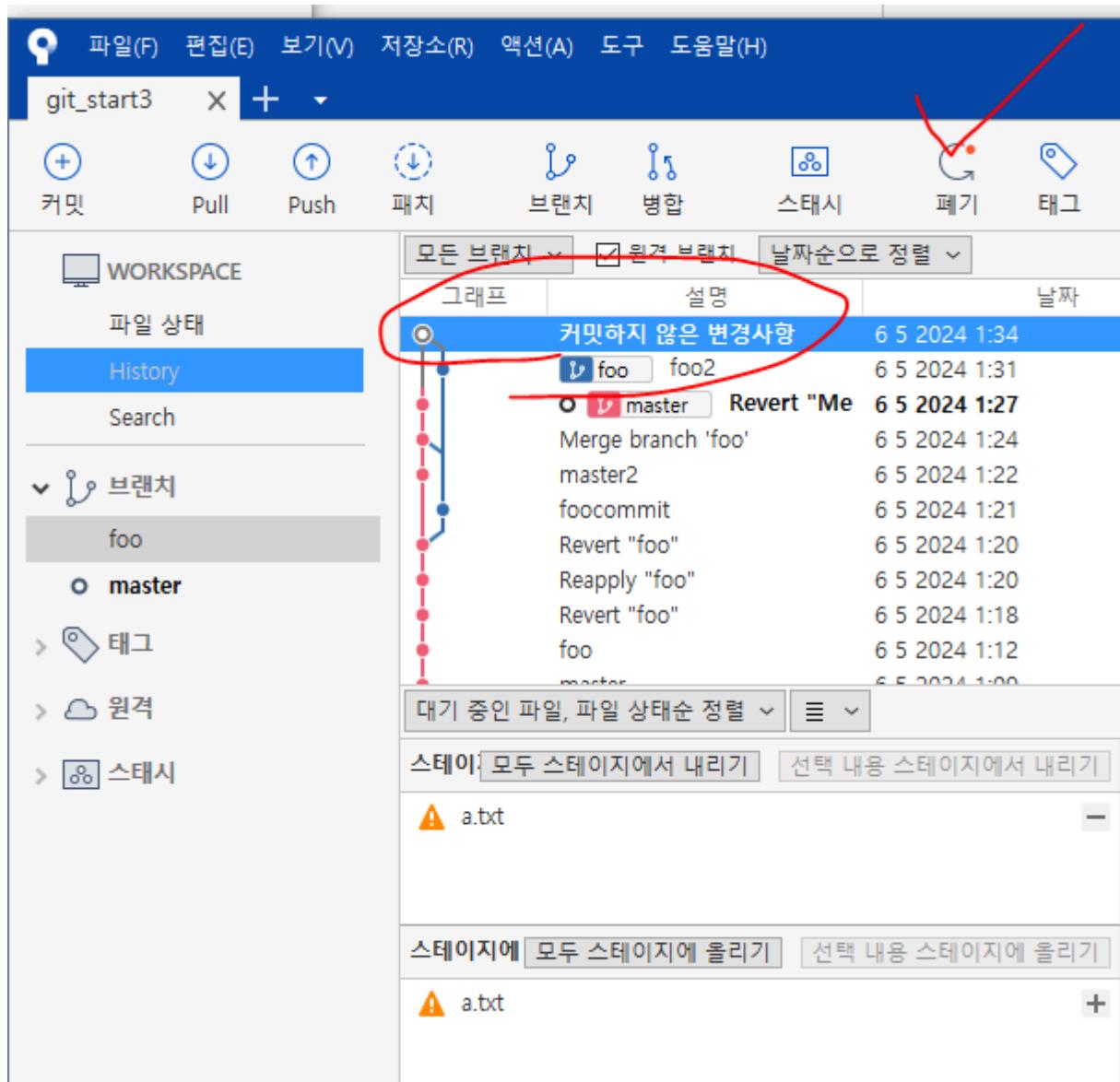
master에서 a.txt는 master이고

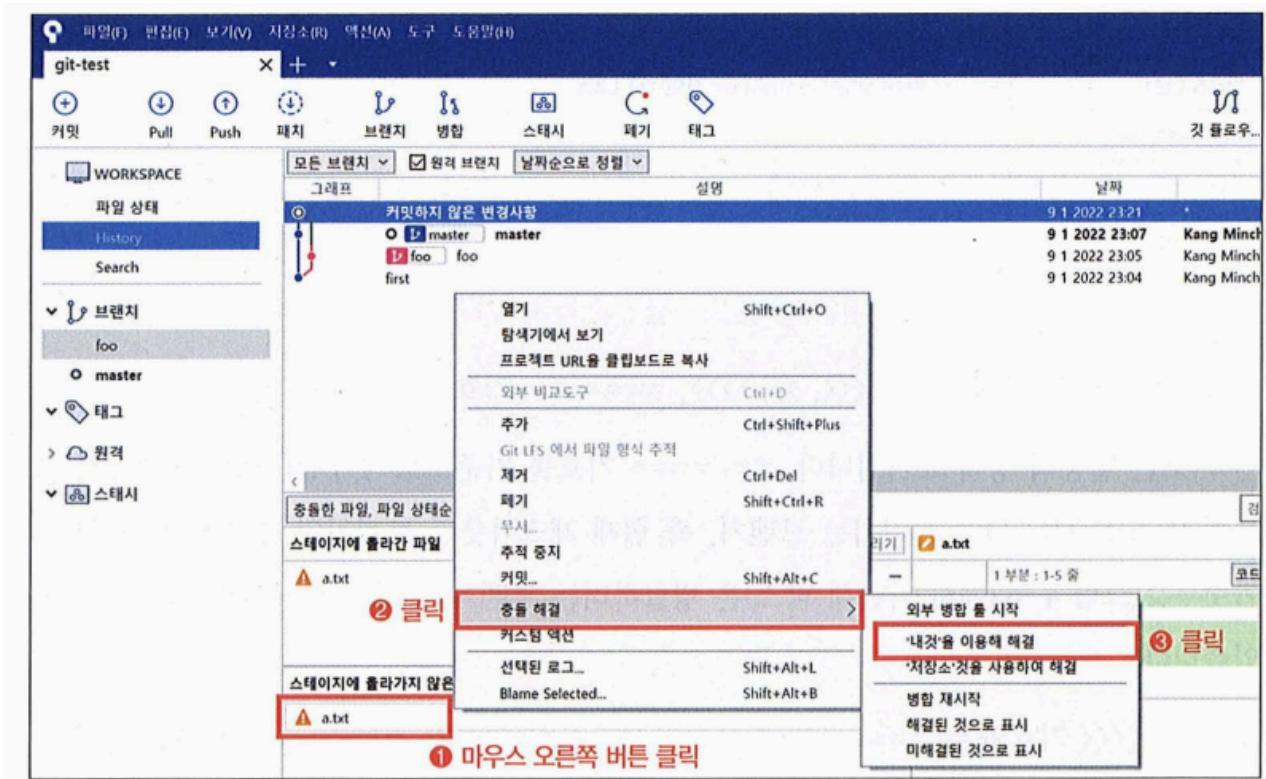
foo에서 a.txt는 foo이다.

합치면 충돌 문제가 발생한다.

master선택후 foo병합하면 에러 발생

병합 실패시 폐기하자. 그렇지 않으면 문제가 발생해서 괴일수 있다.





TIP
foo 브랜치의 내용을 병합 결과로 반영하고 싶다면 “저장소”들을 사용하여 해결’을 클릭하면 됩니다.

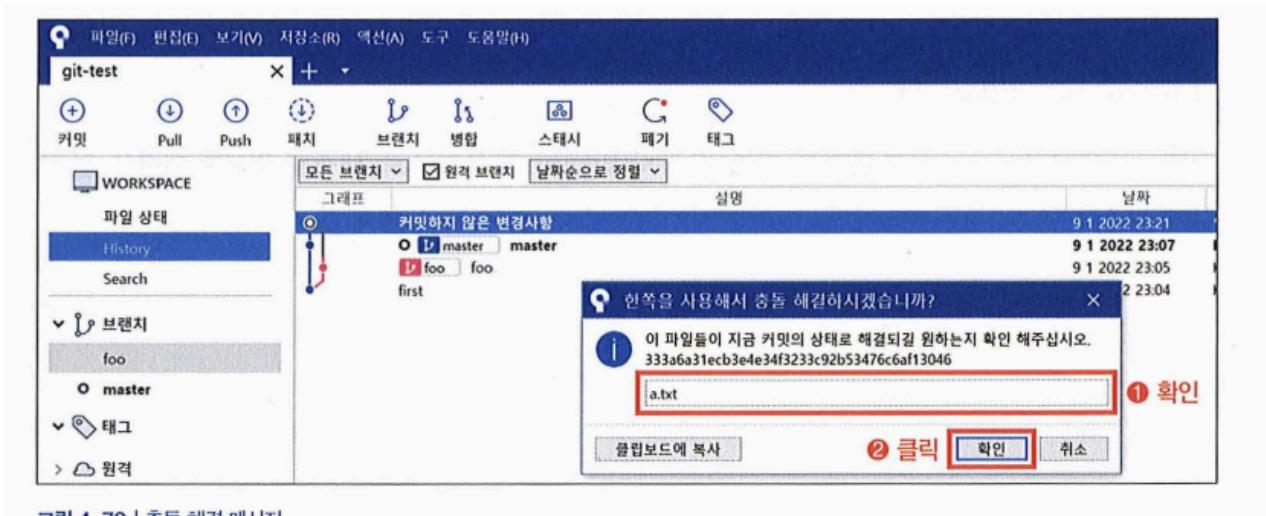


그림 4-70 | 충돌 해결 메시지

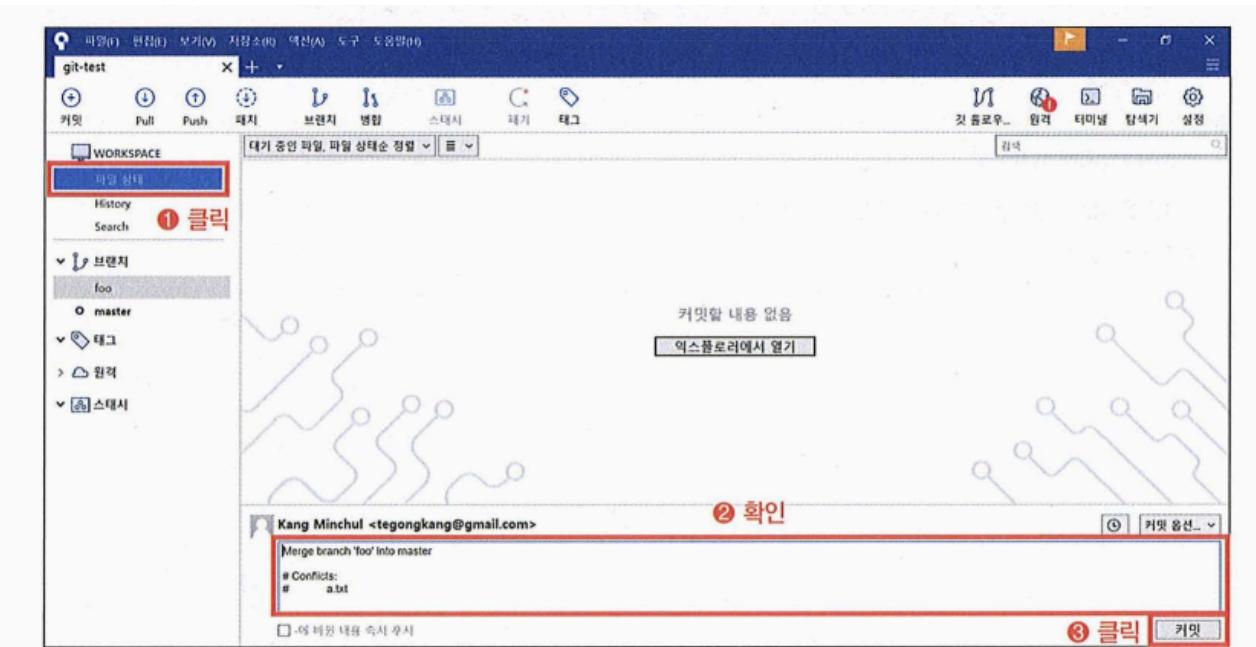


그림 4-72 | 충돌 해결 후 커밋하기

- ⑤ 그러면 History에서 충돌이 발생했던 foo 브랜치가 성공적으로 병합된 것을 확인할 수 있습니다. a.txt 파일 내용도 master 브랜치의 내용으로 업데이트됐지요.

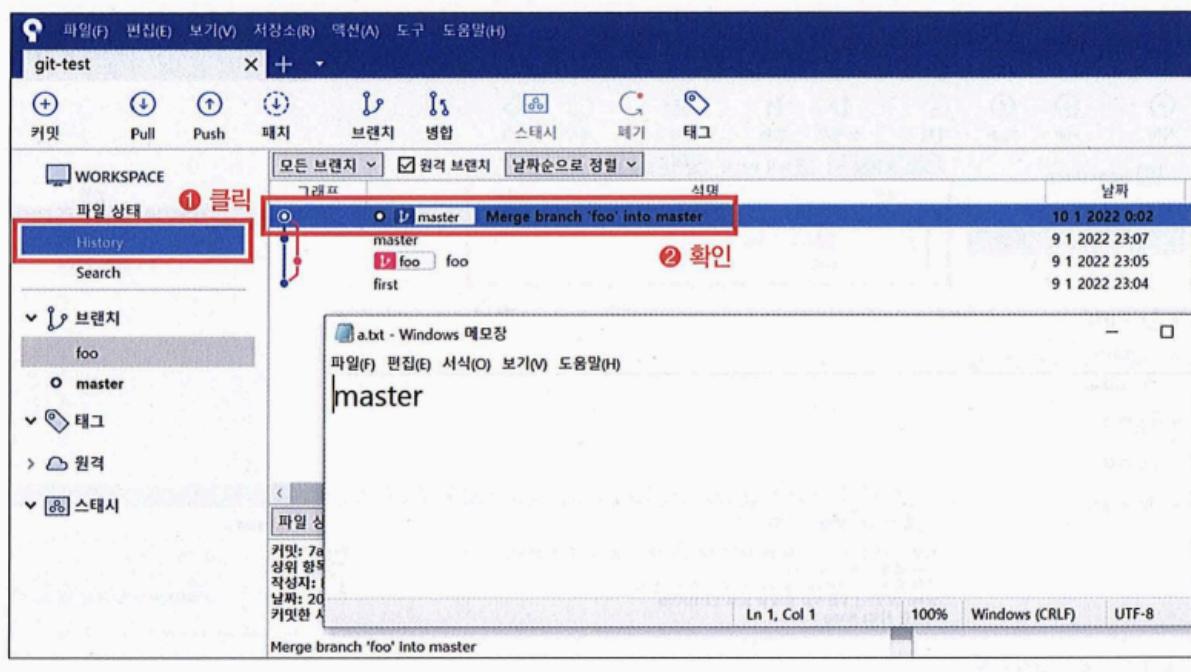


그림 4-73 | 충돌 해결 후 내용 확인

재배치로 올리기

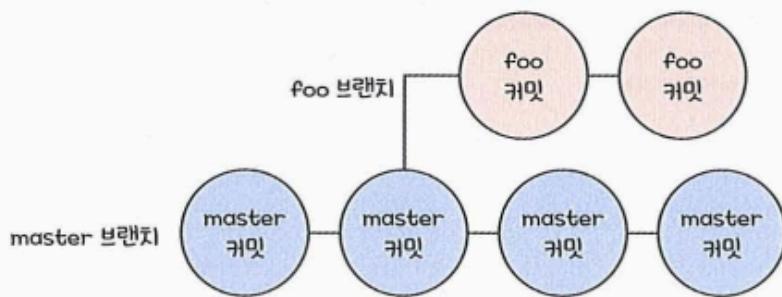


그림 4-74 | 브랜치 재배치 전 master 브랜치와 foo 브랜치

이 상황에서 그림 4-75와 같이 foo 브랜치를 네 번째 커밋에서 뺀어나오도 이처럼 브랜치가 뺀어나온 기준점을 변경하는 것을 브랜치의 재배치, rebase라

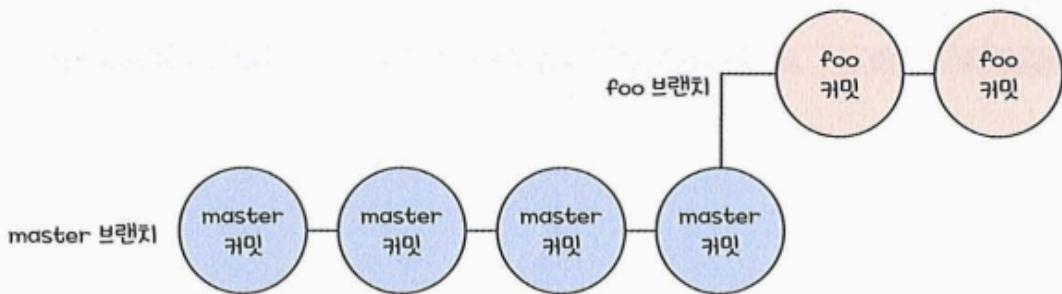
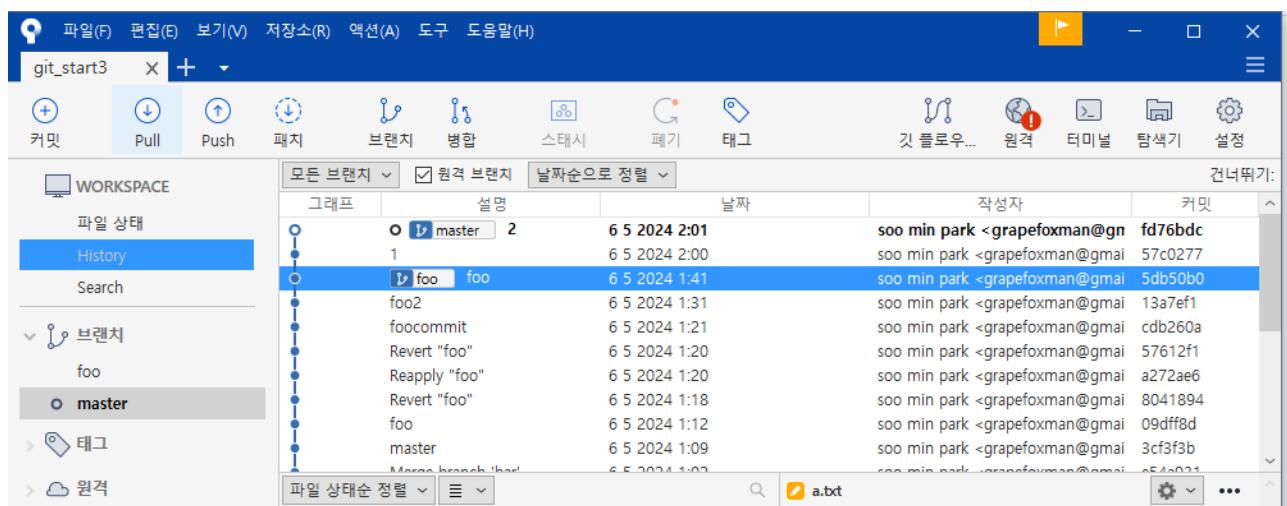
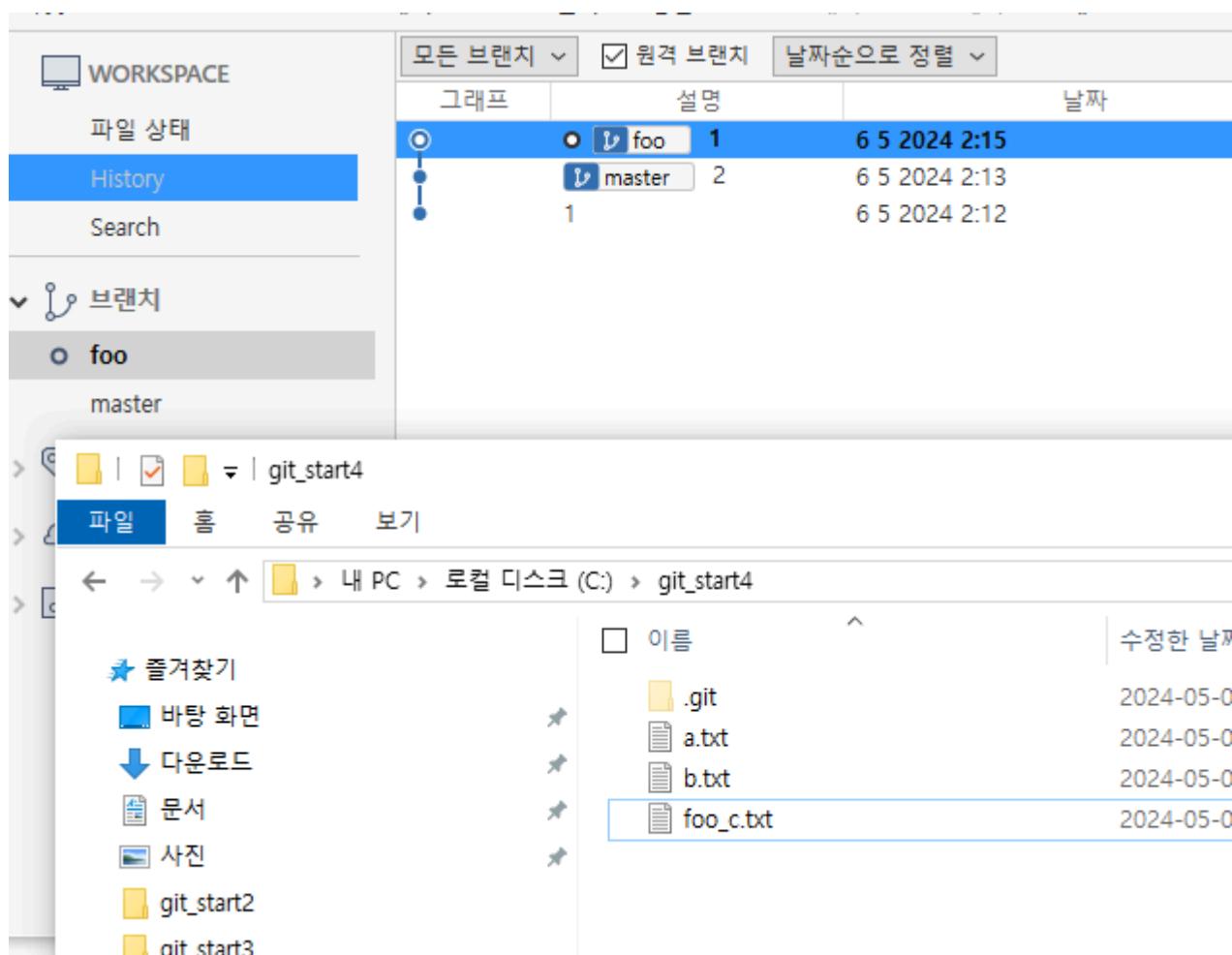
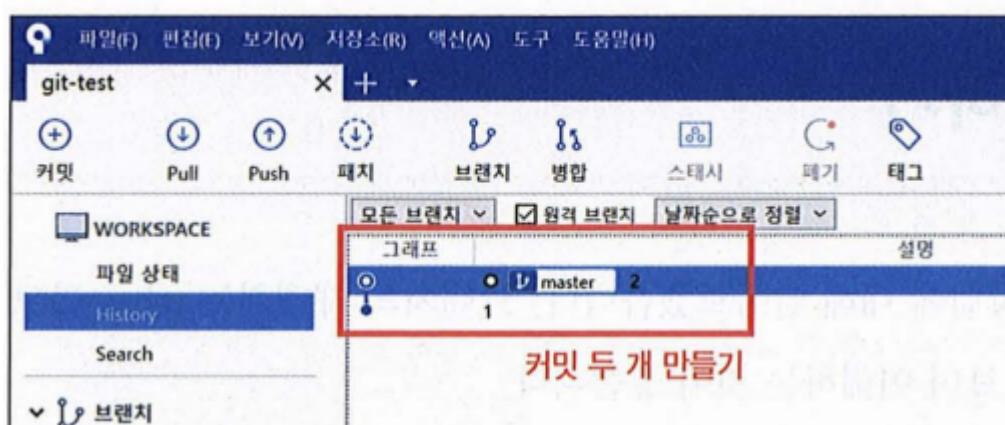
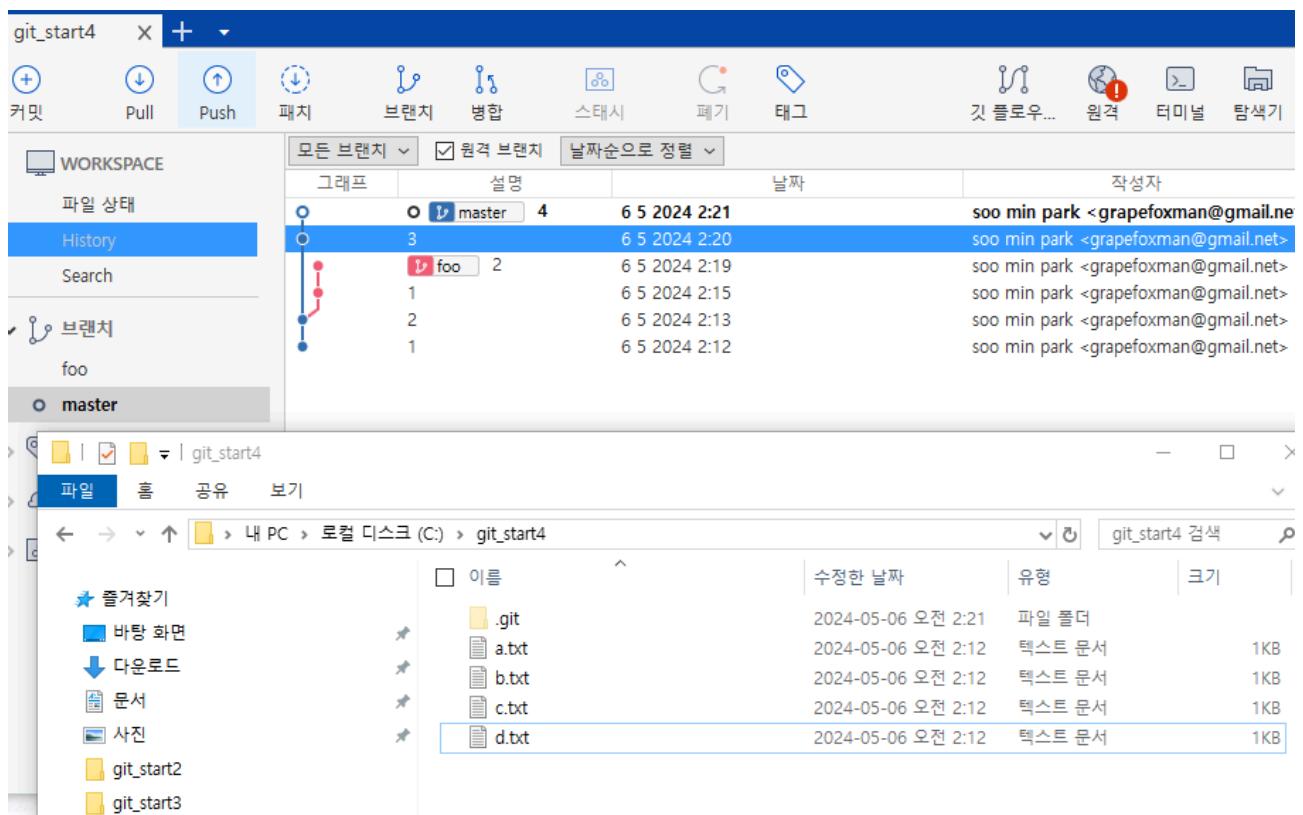


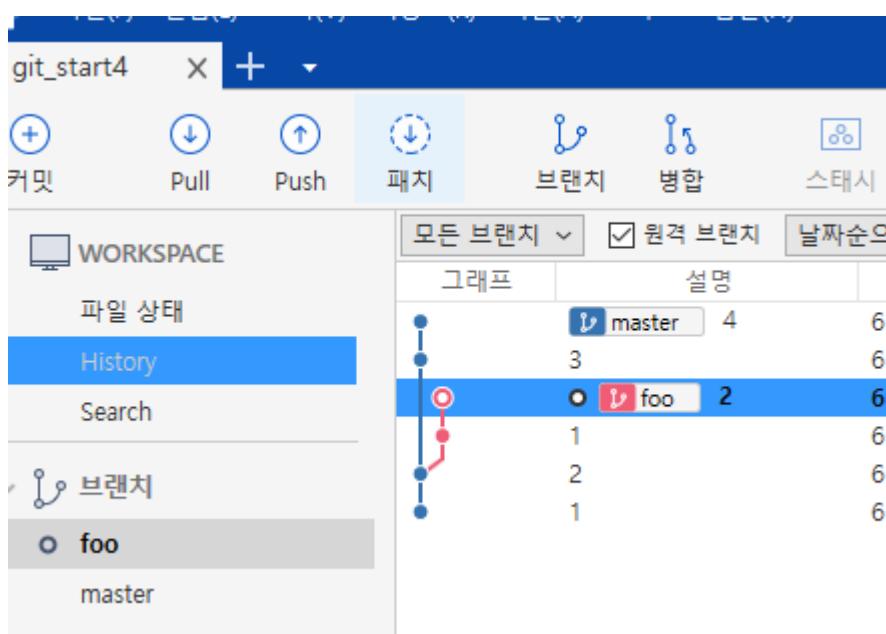
그림 4-75 | 브랜치 재배치 후 master 브랜チ와 foo 브랜치



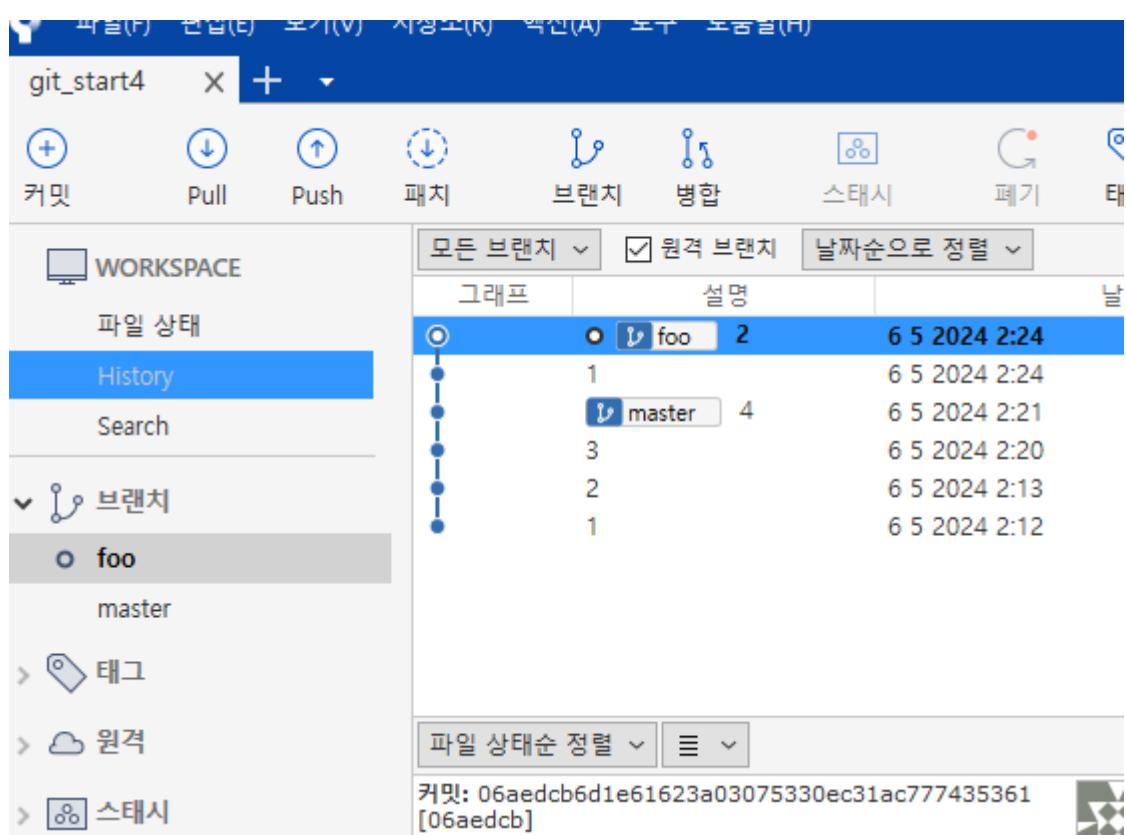
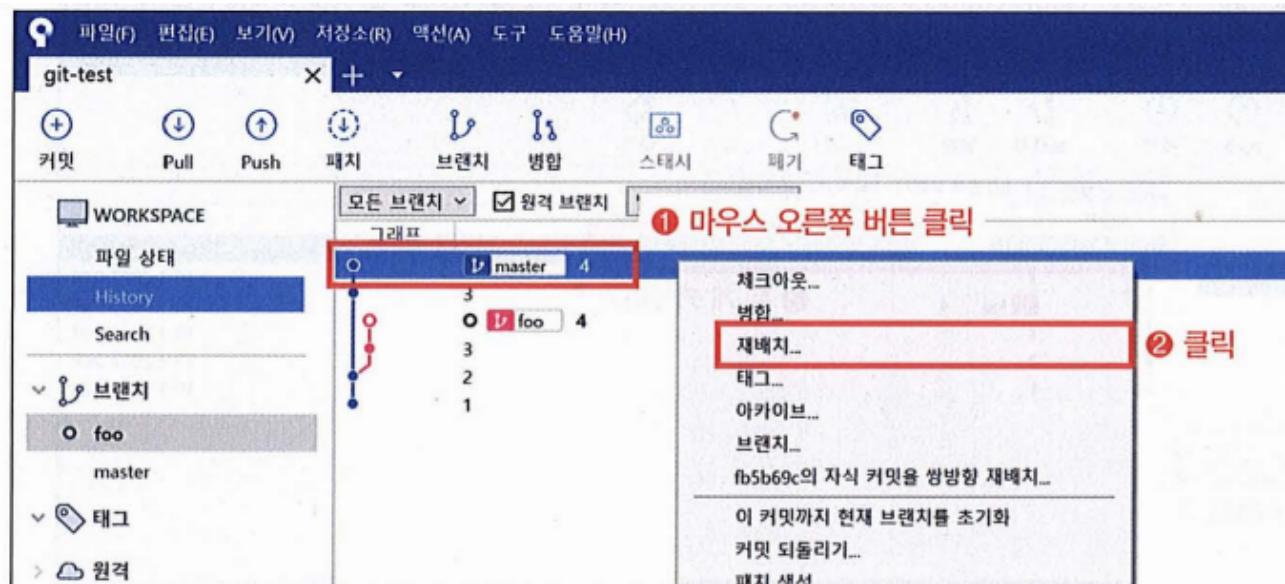




foo를 재배치하려면 foo를 선택한다



재배치할 마스터 위치를 선택해서 마우스 오른쪽 클릭 재배치를 선택한다.



git hub 사용방법

git hub는 원격 저장소이다. 인터넷을 통해 다른 컴퓨터에 프로젝트 정보를 저장하는 것입니다.

로그인하고 왼쪽 상단의 이미지를 클릭해서 세팅 메뉴에 들어가자.

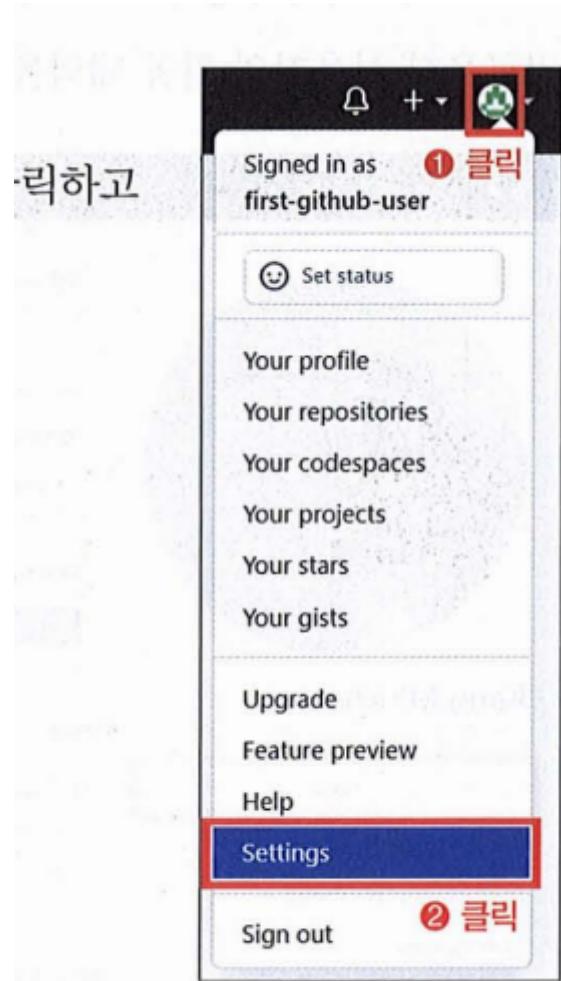


그림 5-8 | 'Settings' 클릭하기

The screenshot shows the GitHub user settings interface for a user named 'first-github-user'. The 'Profile' tab is highlighted with a red border. On the left sidebar, other tabs like 'Account', 'Appearance', 'Accessibility', and 'Notifications' are visible. The main area is titled 'Public profile' and contains fields for 'Name' (with a note about visibility), 'Public email' (with a dropdown menu and a note about privacy), and 'Bio'. A 'Profile picture' section shows a placeholder image and an 'Edit' button.

③ 필자는 다음 그림처럼 정보를 채웠습니다. 모든 정보를 다 입력할 필요는 없습니다. 여러분도 이름, 이메일 등을 입력해 보세요.

This screenshot shows the GitHub profile settings page with sample data entered. The 'Name' field contains 'Kang Minchul'. The 'Public email' field contains 'minchulkang.46@gmail.com'. The 'Bio' field contains 'Software Developer'. To the right, there is a placeholder profile picture with an 'Edit' button. A red box highlights the entire form area, and the word '입력' (Input) is written in red at the bottom right corner of the highlighted area.

그림 5-10 | 수정한 필자의 프로필

- ④ 다 입력했다면 스크롤을 내려 Update profile을 클릭하세요.

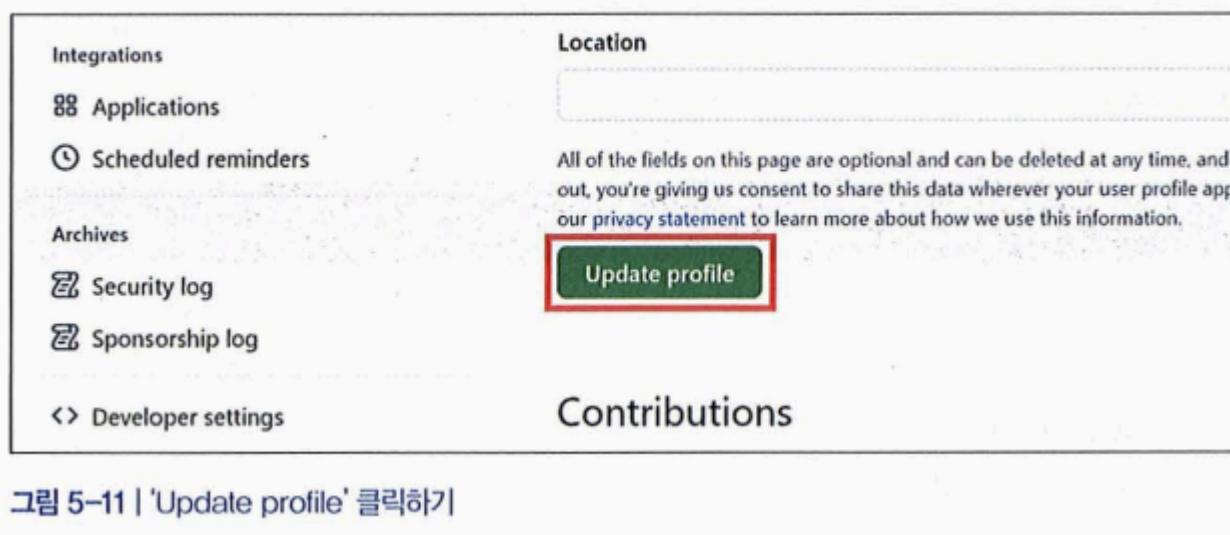


그림 5-11 | 'Update profile' 클릭하기

■ 원격 저장소 만들기

- ① 우측 상단의 +를 클릭한



그림 5-26 | 'New repository' 클릭하기

Create a new repository

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

Owner * Repository name * ① 입력

Great repository names are short and memorable. Need inspiration? How about jubilant-guacamole?

Description (optional) ② 입력

Public Anyone on the internet can see this repository. You choose who can commit. ③ 선택

Private You choose who can see and commit to this repository.

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. ④ 그대로 두기

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.

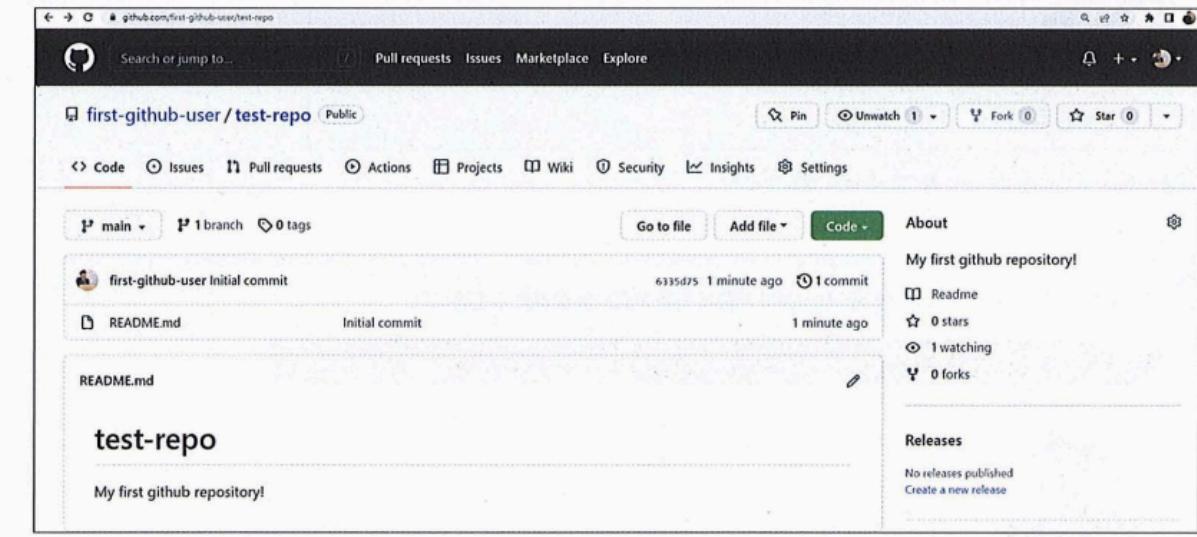
This will set **main** as the default branch. Change the default name in your settings.

Create repository ⑤ 클릭

그림 5-28 | 원격 저장소 생성 화면

<https://github.com/grapecfoxman/test-repo2> 아까 만든 조소를 이용해서 해당 프로젝트에 접근할 수 있다.

③ 생성된 원격 저장소의 모습입니다. 다음 절부터 여러분은 이 저장소와 상호 작용하며 코드를 업로드하고, 다운로드할 예정입니다. URL을 한번 확인해 볼까요? 여러분이 방금 만든 저장소 URL은 <https://github.com/계정 이름/저장소 이름> 형식으로 되어 있습니다. 이와 같은 URL로 특정 사용자의 저장소에 접근할 수 있다는 것도 알아두기 바랍니다.



저장소 삭제 방법

TIP

이렇게 생성한 원격 저장소는 'Settings' 메뉴의 최하단에 있는 'Delete this repository'를 클릭해 삭제할 수도 있습니다.

Danger Zone

Change repository visibility
This repository is currently public.

[Change visibility](#)

Transfer ownership
Transfer this repository to another user or to an organization where you have the ability to create repositories.

[Transfer](#)

Archive this repository
Mark this repository as archived and read-only.

[Archive this repository](#)

Delete this repository
Once you delete a repository, there is no going back. Please be certain.

[Delete this repository](#)

그림 5-30 | 원격 저장소 삭제하기

1 | 클론(clone): 원격 저장소를 복제하기

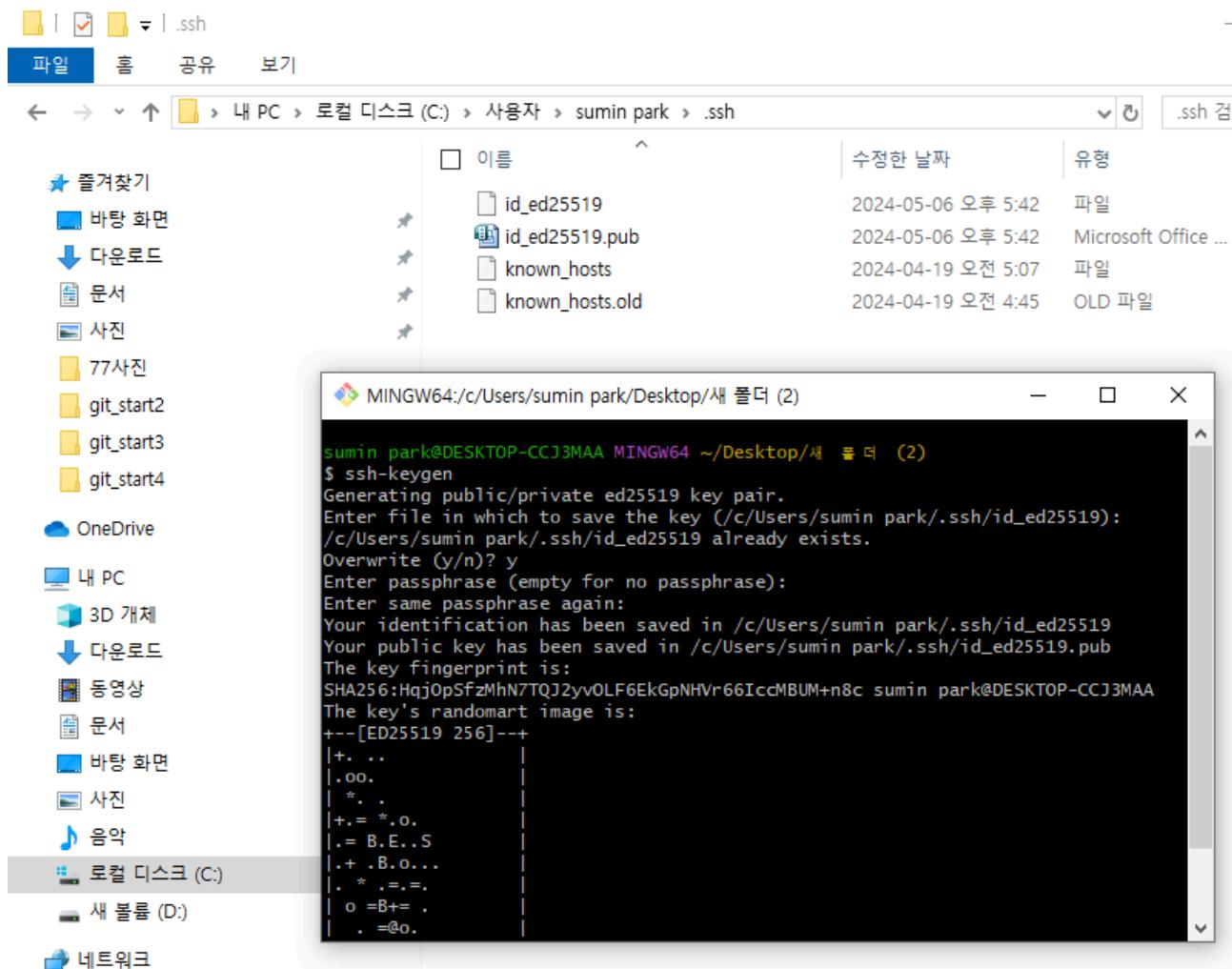
2 | 푸시(push): 원격 저장소에 밀어넣기

3 | 패치(fetch): 원격 저장소를 일단 가져만 오기

4 | 풀(pull): 원격 저장소를 가져와서 합치기

SSH 통신은 암호화된 통신 방법이므로 여러분과 (공개 키를 전달받은) 깃허브 사이에 주고 받는 대회는 암호화되어 전송됩니다

SSH 키는 `ssh-keygen`이라는 간단한 명령으로 생성할 수 있습니다. 배치 콘솔에서 `ssh-keygen`을 입력하고 엔터를 친다. 다음 이미지처럼 특정 폴더에 `ssh` 키가 생성된다.



8 도구 > SSH 키 추가를 클릭하세요.

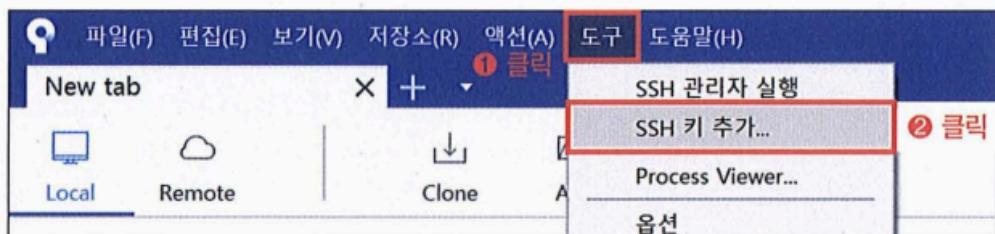


그림 5-39 | 'SSH 키 추가' 선택하기

9 앞서 두 키가 저장된 경로에서 개인 키, 즉 id_rsa를 선택한 후 열기를 클릭하세요.

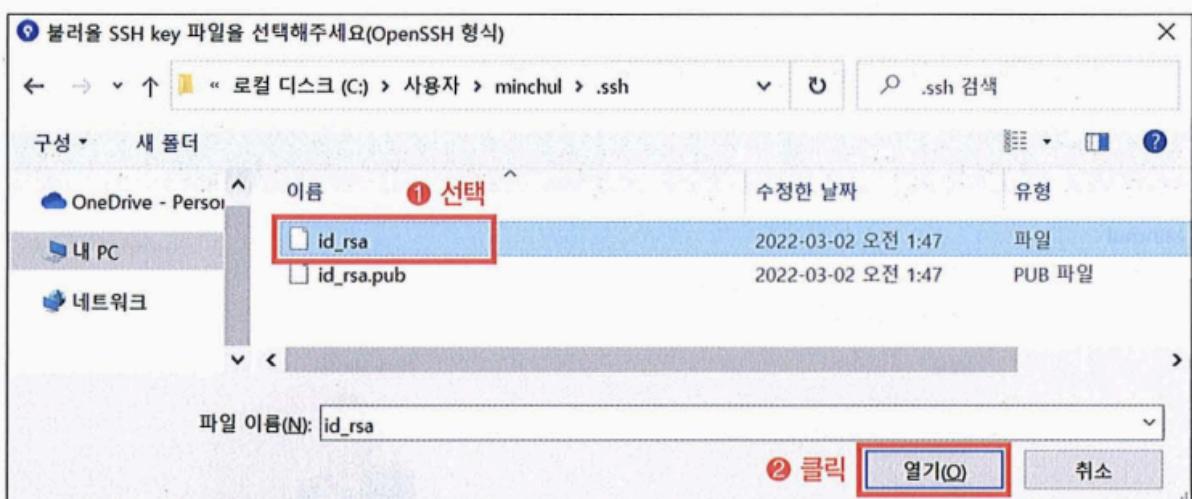
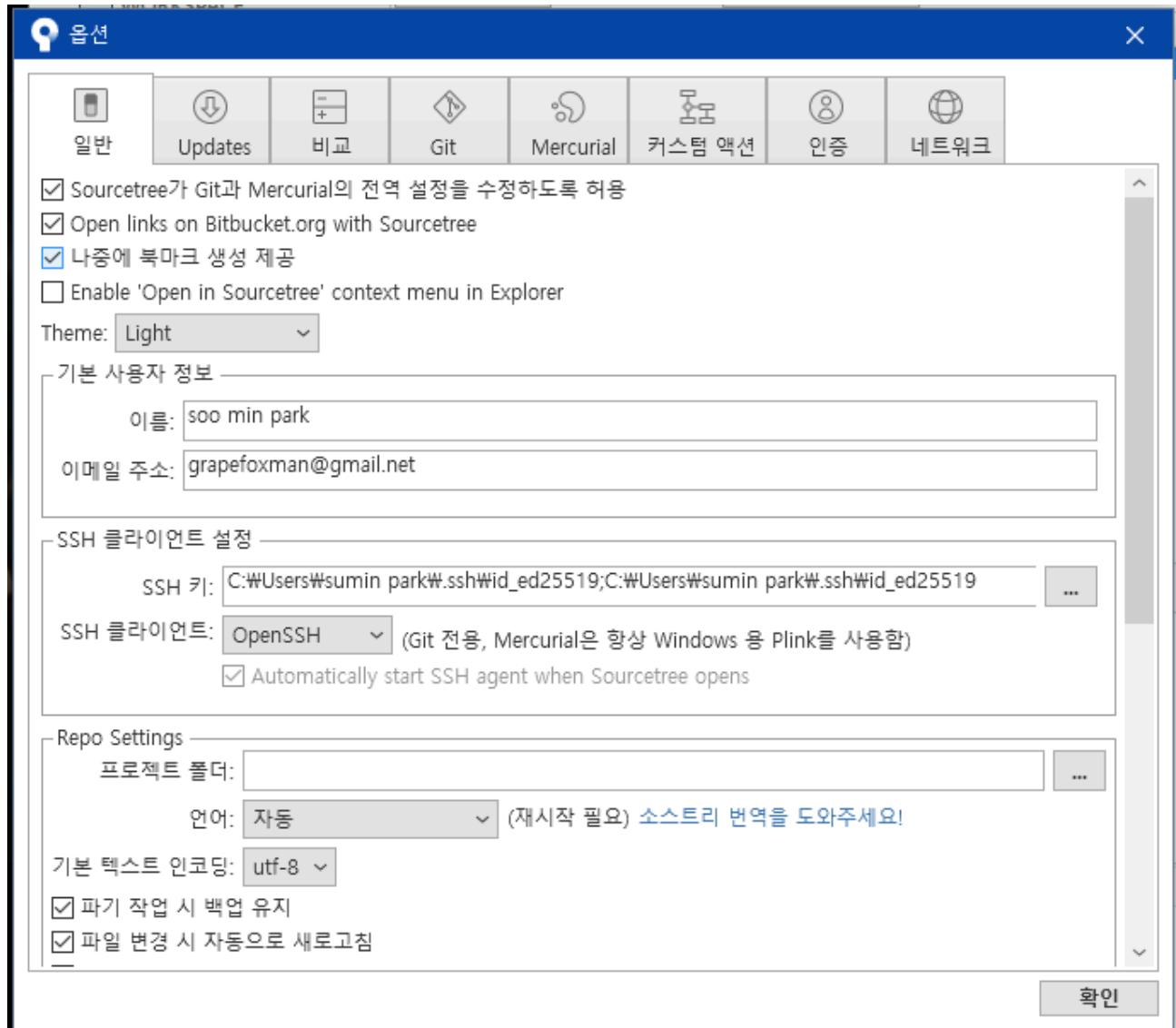


그림 5-40 | 생성한 개인 키 등록하기

제대로 등록되어 있는지 확인한다.



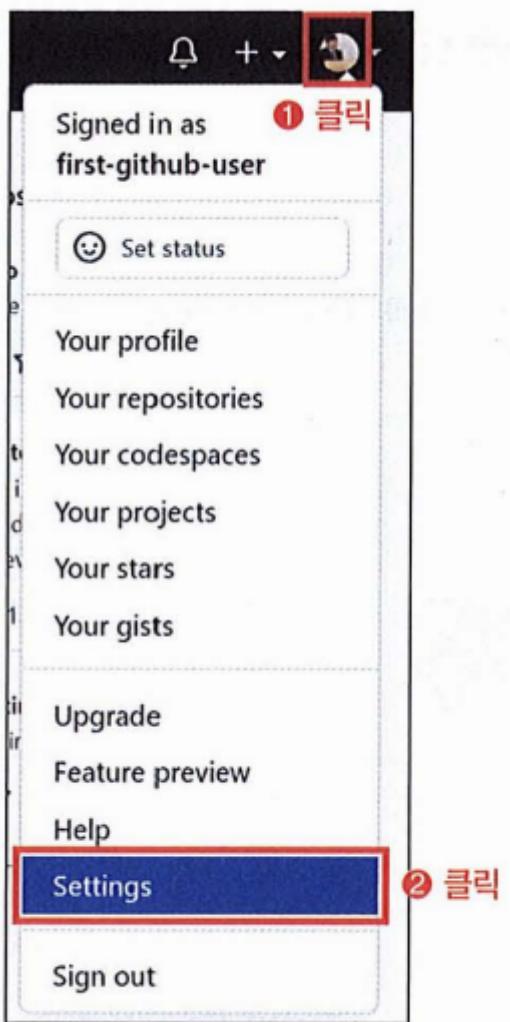


그림 5-42 | 'Settings' 클릭하기

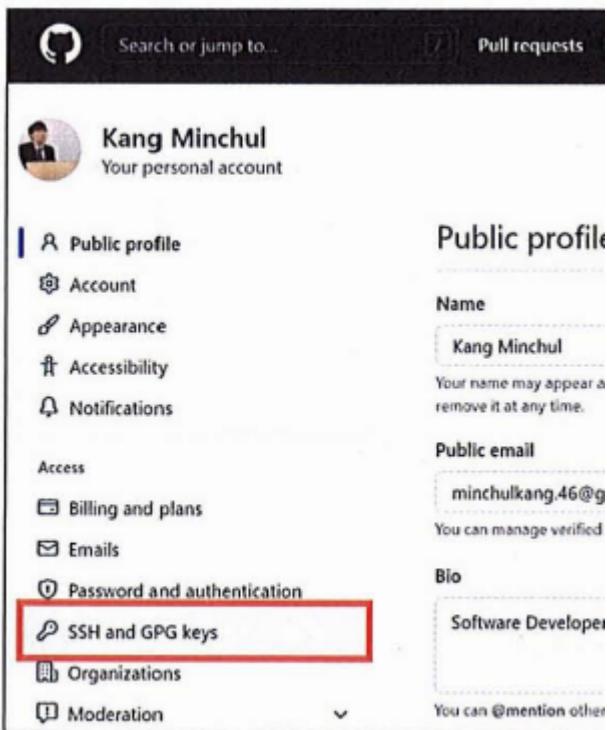


그림 5-43 | 'SSH and GPG keys' 클릭하기

⑯ 여기에 여러분의 공개 키를 등록하면 됩니다. Title 항목에는 여러분의 임의대로 키의 제목을 쓰고, Key 항목에는 공개 키 파일, 즉 id_rsa.pub 안에 적힌 내용을 넣으면 됩니다. id_rsa.pub 파일을 열어 복사한 후 Key 항목에 붙여넣습니다. 그리고 Add SSH key를 눌러주세요.

The screenshot shows the 'SSH keys / Add new' form. It has a 'Title' input field containing 'My computer' and a 'Key' input field containing a long string of characters representing a public RSA key. At the bottom, there are two buttons: 'Add SSH key' (highlighted with a red box) and '② 붙여넣기' (Paste).

그림 5-45 | 생성한 공개 키 붙여넣기

oxman)

[Go to your personal profile](#)

[New SSH key](#)

SSH keys

There are no SSH keys associated with your account.

Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

[New GPG key](#)

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

Vigilant mode

Flag unsigned commits as unverified

This will include any commit attributed to your account but not signed with your GPG or S/MIME key.

Note that this will include your existing unsigned commits.

[Learn about vigilant mode](#).

Add new SSH Key

Title

Key type

Authentication Key

Key

Begins with 'ssh-rsa' | 'ecdsa-sha2-nistp256' | 'ecdsa-sha2-nistp384' | 'ecdsa-sha2-nistp521' | 'ssh-ed25519' | 'sk-ecdsa-sha2-nistp256@openssh.com' or 'sk-ssh-ed25519@openssh.com'

[Add SSH key](#)



그림 5-47 | '계정 추가' 클릭하기

17 호스팅 서비스는 GitHub를, 선호 프로토콜은 SSH를 선택한 뒤 OAuth 토큰 새로고침을 눌러보세요.

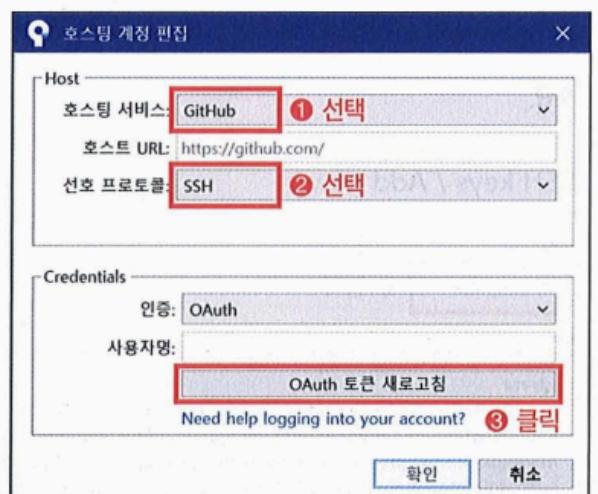


그림 5-48 | 깃허브 계정 추가하기

인증이 안될때는 브라우저를 바꿔서 주소를 복사해서 크롬으로 옮긴다.

19 이 경우 스크롤을 내려 Authorize Atlassian을 누르세요.

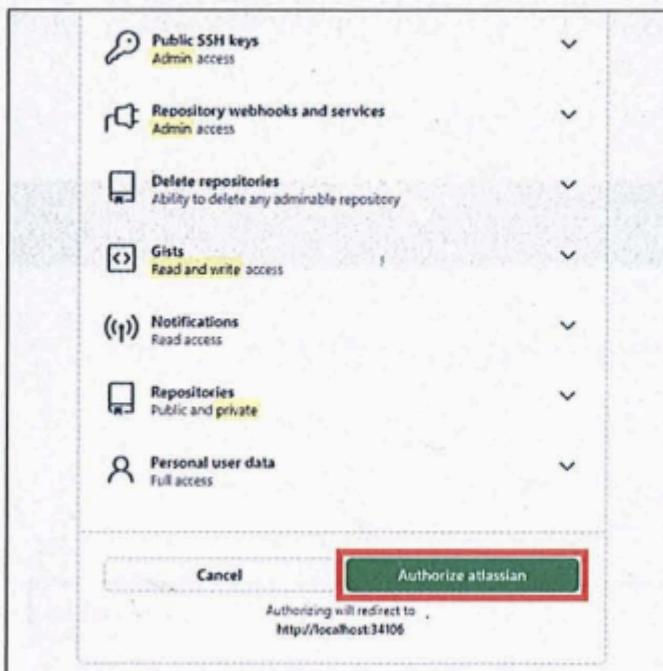


그림 5-50 | 'Authorize Atlassian' 클릭하기

21 인증 성공이라는 표시가 뜨면 성공적으로 연동된 것입니다. 확인을 클릭합니다.



그림 5-52 | 깃허브 계정 인증 성공

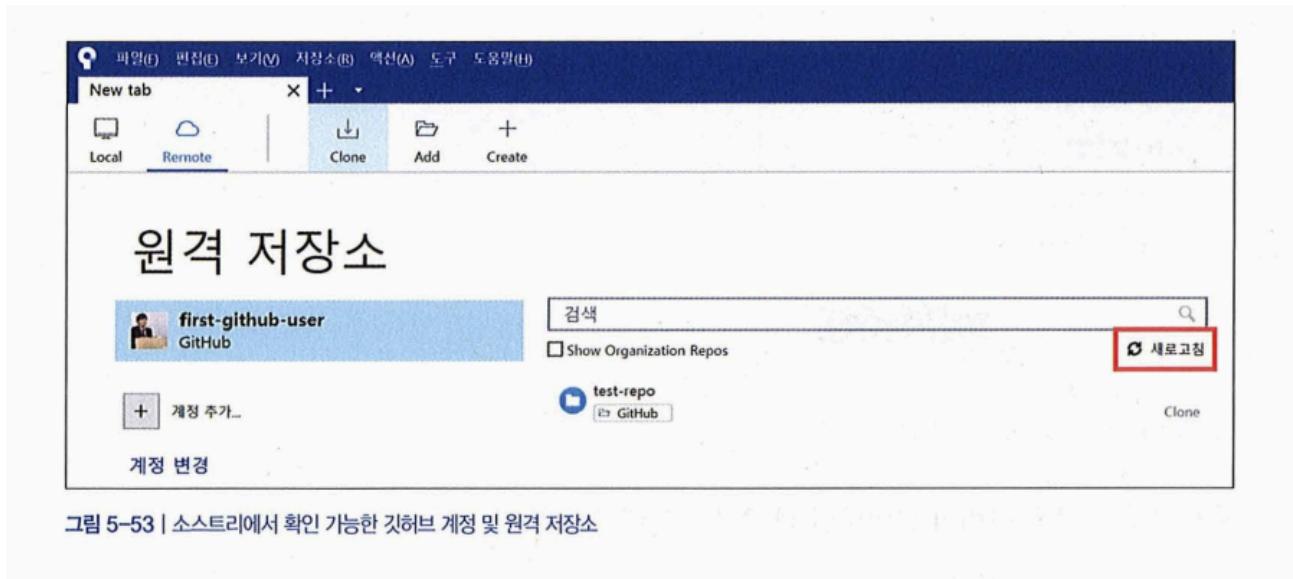


그림 5-53 | 소스트리에서 확인 가능한 깃허브 계정 및 원격 저장소

main 브랜치는 master 브랜치와 같습니다. 깃허브에서는 기본 브랜치를 main 브랜치라고 부릅니다.

origin/HEAD; origin/main은 원격 저장소 origin과 기본 브랜치를 지칭합니다.

다시 소스트리로 돌아옵니다. 여기서 브랜치의 이름에 주목해 주세요. main, origin/main, origin/HEAD, 이렇게 브랜치 세 개가 보입니다.

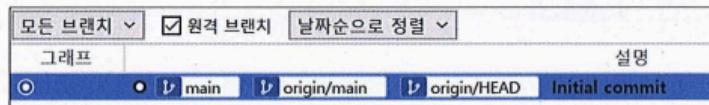


그림 5-64 | main, origin/main, origin/HEAD 브랜치

좌측에서도 브랜치에 main, 원격에 origin/HEAD, origin/main 브랜치를 볼 수 있지요.



origin/HEAD, origin/main은 원격 저장소 origin의 HEAD와 기본 브랜치를 지칭합니다. 여기서 origin은 원격 저장소 경로에 붙은 일종의 별명입니다. 소스트리 우측 상단의 설정을 클릭해 볼까요?

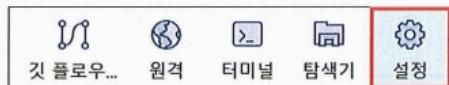


그림 5-66 | '설정' 클릭하기

원격 저장소 경로를 보면 이름에 origin, 경로는 원격 저장소를 클론할 때 사용한 경로가 적혀 있습니다. 원격 저장소를 지칭할 때 매번 경로(git@github.com:...)를 사용하는 것은 번거로우니 단순히 origin이라고 부르기로 한 것입니다.

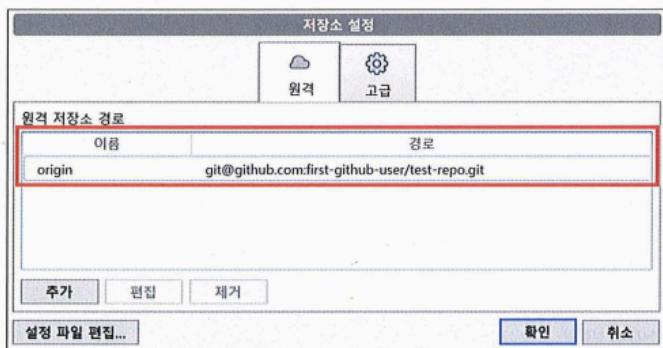
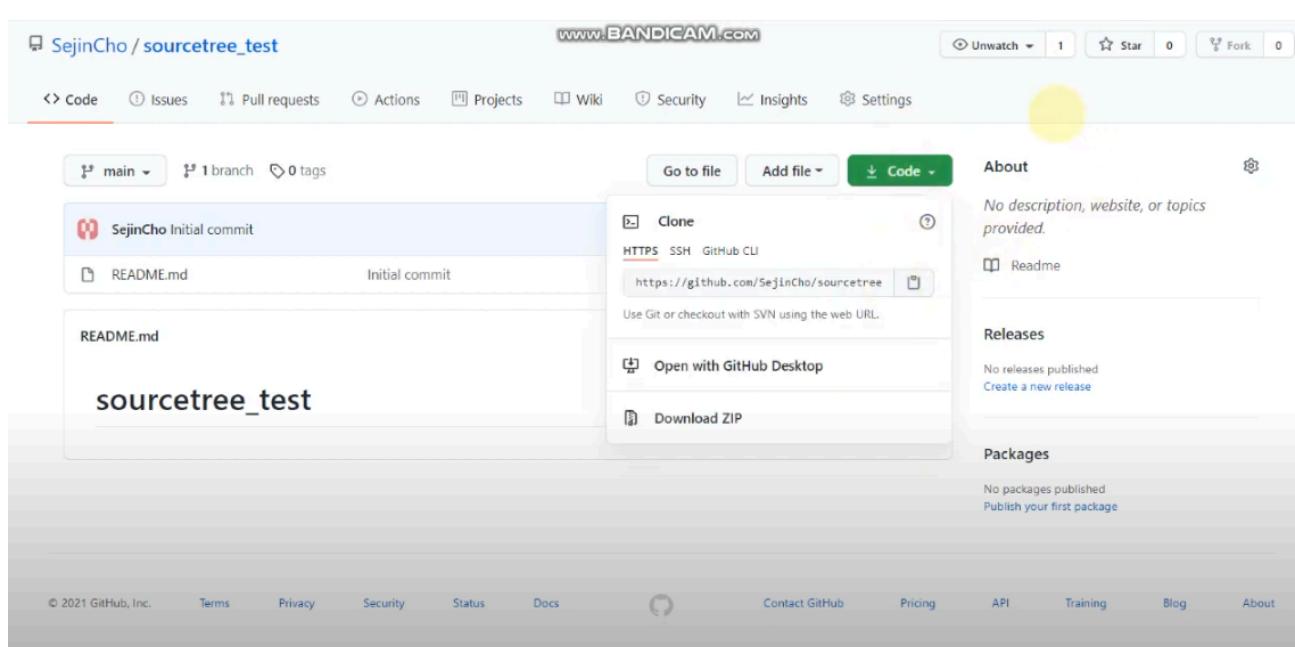
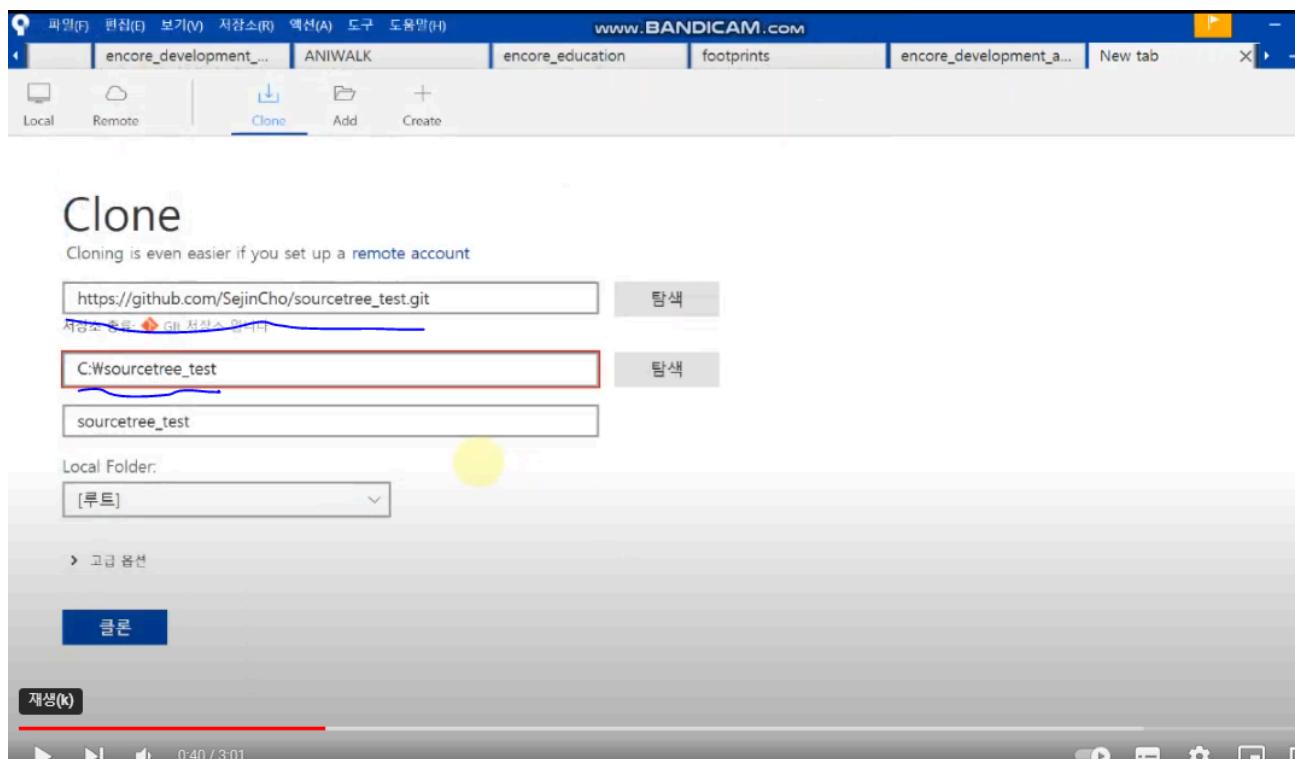
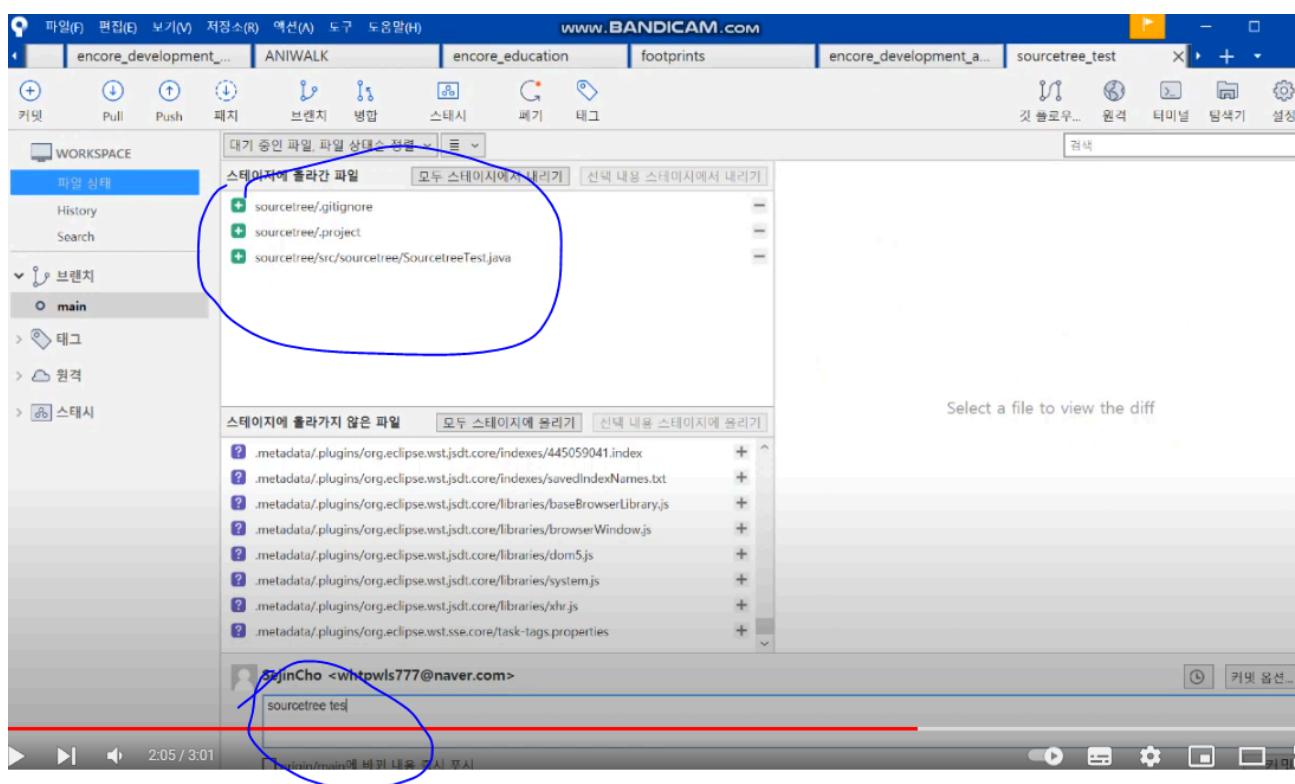


그림 5-67 | 원격 저장소의 이름과 경로 확인하기





eclipse에서 work 폴더로 연 다음 프로젝트를 만들어 보자.



프로젝트를 만들고 나면 생성된 소스를 소스트리를 이용해서 커밋 풋시한다.

