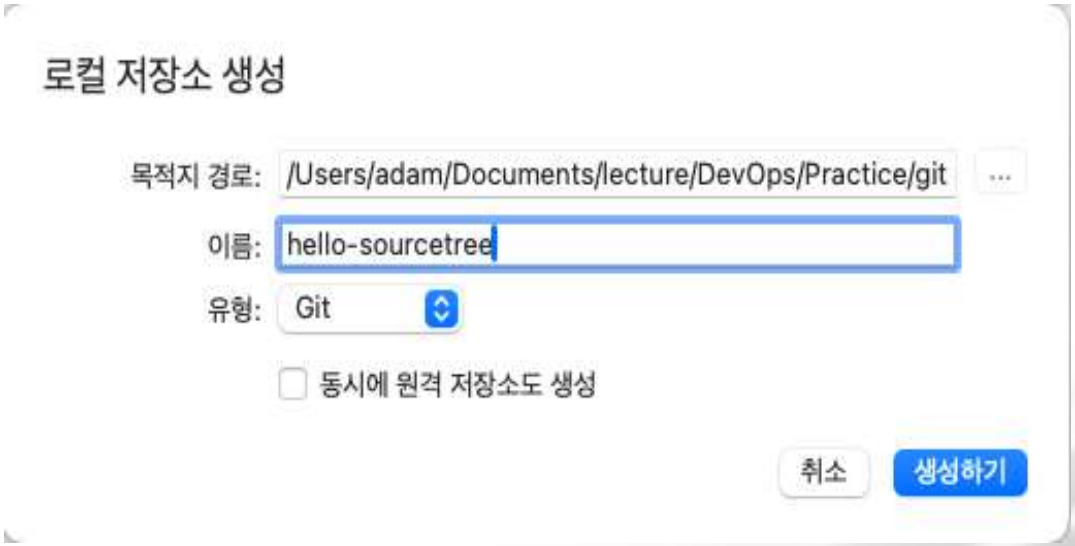
Source Tree

Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - ● 로컬 저장소 생성
      - ◆ 메뉴에서 로컬 저장소 생성 버튼 클릭

Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - ● 로컬 저장소 생성
      - ◆ 저장소 정보를 설정하고 [생성하기] 클릭



        - ▪ 디렉토리 안에 .git 이라는 디렉토리가 생성되었다면 로컬 저장소 생성 성공

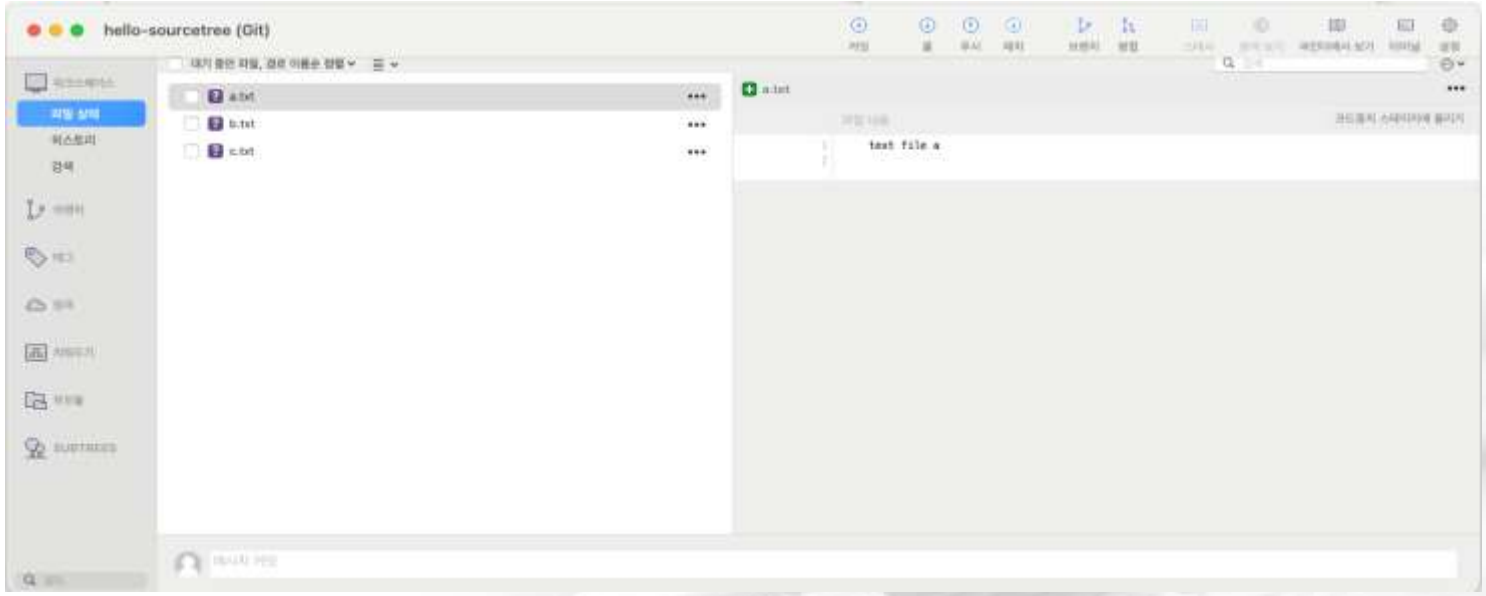
Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - ● 버전을 관리할 대상 만들기
      - ◆ 작업 디렉토리에 a.txt, b.txt, c.txt 라는 텍스트 파일로 만들고 각각의 파일에 텍스트 작성
        - ▪ a.txt: text file a
        - ▪ b.txt: text file b
        - ▪ c.txt: text file c
      - ◆ 저장을 할 때 줄 바꿈을 하지 않고 저장하면 스테이지에 올릴 때 경고 메시지가 출력됨

        ₩ No newline at end of file

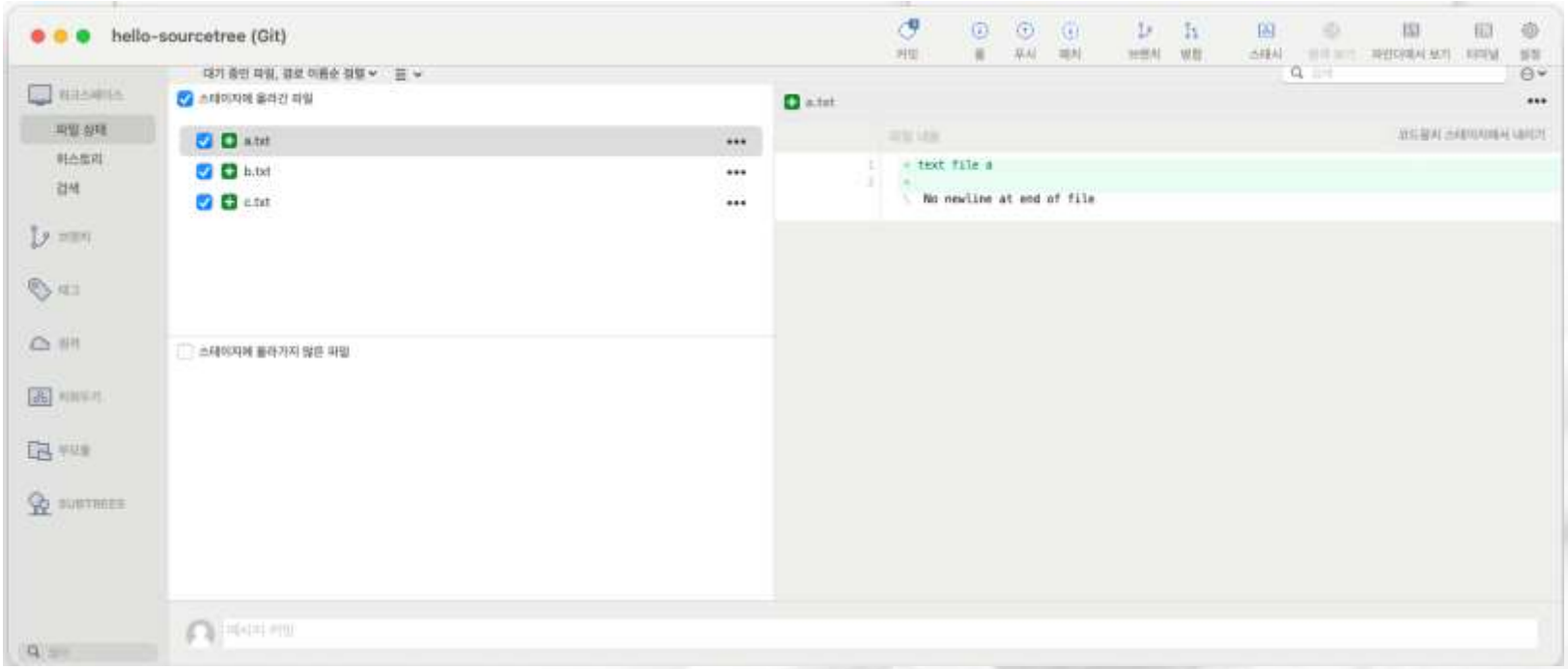
Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - ● 버전을 관리할 대상 만들기
      - ◆ 현재 상태: 소스 트리의 파일 상태에서 스테이지에 올라가지 않은 파일 항목에 생성한 파일들이 추가됨



Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - ● 스테이지에 올리기
      - ◆ 하단에서 파일 옆의 체크박스를 선택하면 스테이지에 올라감

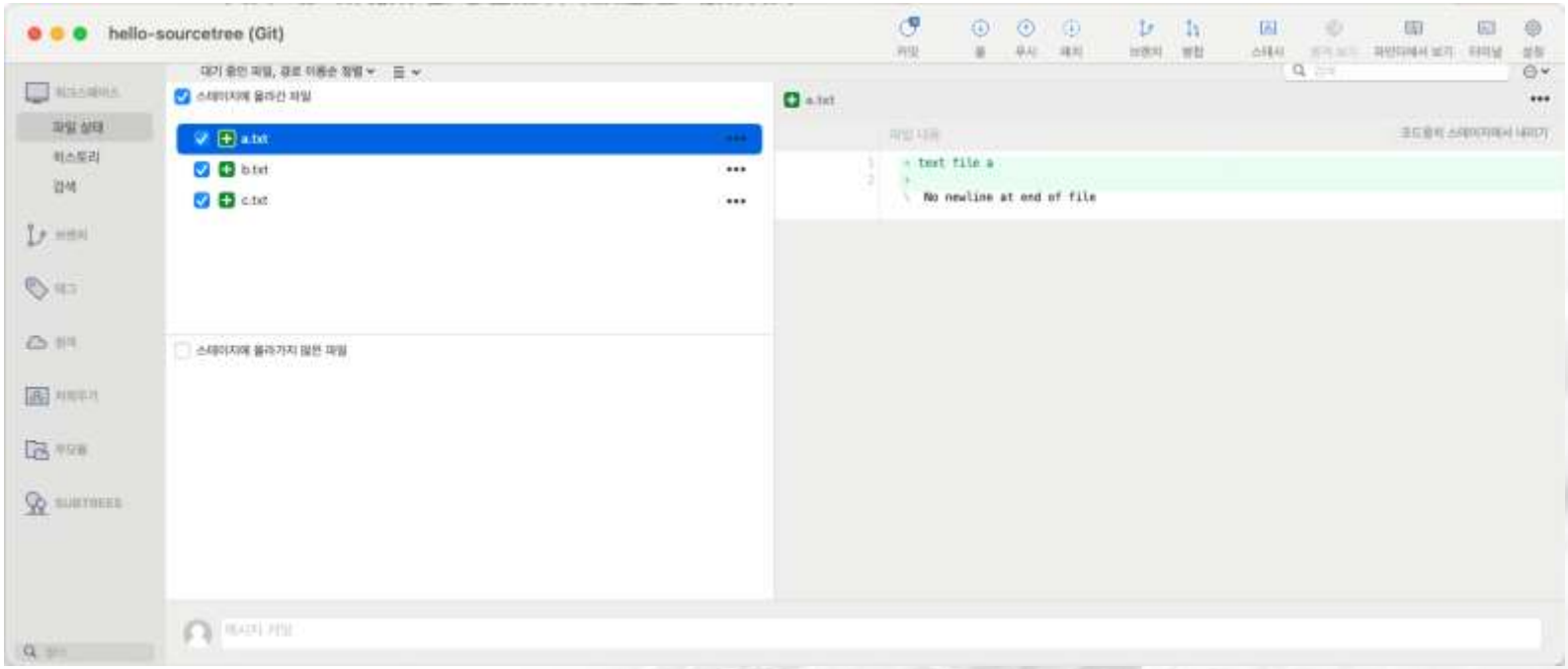


Source Tree

- 버전 관리 시작
  - ✓ 첫 버전 생성
    - 스테이지에 올리기
      - 스테이지에 올라가지 않은 파일 전체를 올리는 방법: 모두 스테이지에 올리기를 클릭
      - 파일 별로 올리는 방법: 파일 이름 우측의 +를 클릭합니다
      - 선택한 파일만 올리는 방법: Ctrl 을 누른 채 스테이지에 올리려는 파일을 클릭한 후 선택 내용 스테이지에 올리기를 클릭
      - 스테이지에 올라간 파일 항목에서 파일 이름 우측의 -를 클릭하면 해당 파일은 스테이지에서 내려옴

Source Tree

- 버전 관리 시작
  - ✓ 첫 버전 생성
    - ◆ 스테이지에 올라가 있는 파일을 선택하면 변경 내역 확인 가능: 초록색 표시 와 + 표시는 각 파일에서 새롭게 추가된 내용을 의미

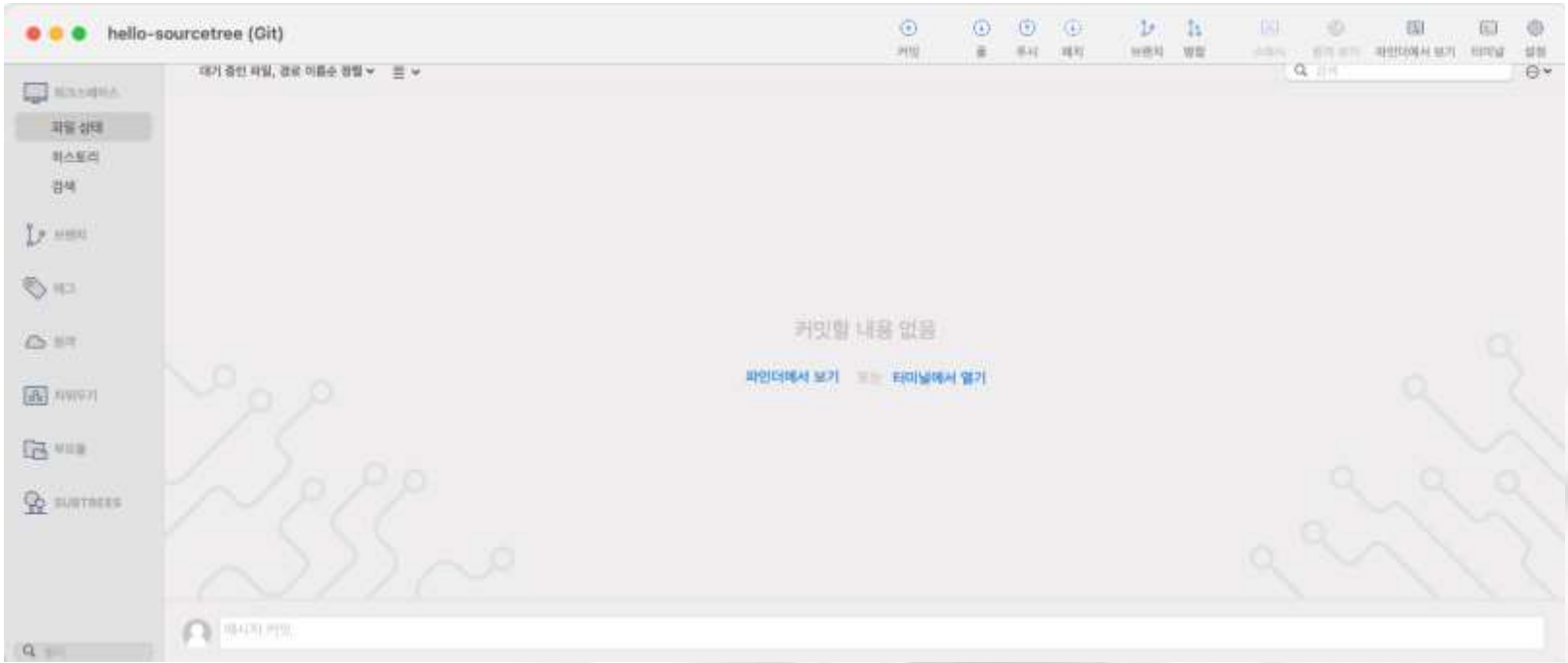


Source Tree

- 버전 관리 시작
  - ✓ 첫 버전 생성
    - Commit
      - Commit 메시지를 작성하고 파일을 선택한 후 Commit 버튼을 클릭
      - Commit Message
        - Commit 메시지는 버전을 설명하는 메시지
        - Commit 메시지는 크게 제목과 본문으로 이루어져 있으며 본문은 생략할 수 있음
        - 첫 줄에는 제목을 쓰고 한 줄 띄고 다음 줄에는 본문을 작성
        - 깃을 통해 새로운 버전을 만들 때 Commit 메시지를 작성을 권장
        - Commit 메시지를 적지 않는다면 프로젝트의 규모가 커지고 수많은 Commit이 쌓였을 때 다른 누군가가 누가, 언제, 어떤 변경 사항을 만들었는지, 이 변경 사항이 의미하는 것은 무엇인지 파악하기 어려움

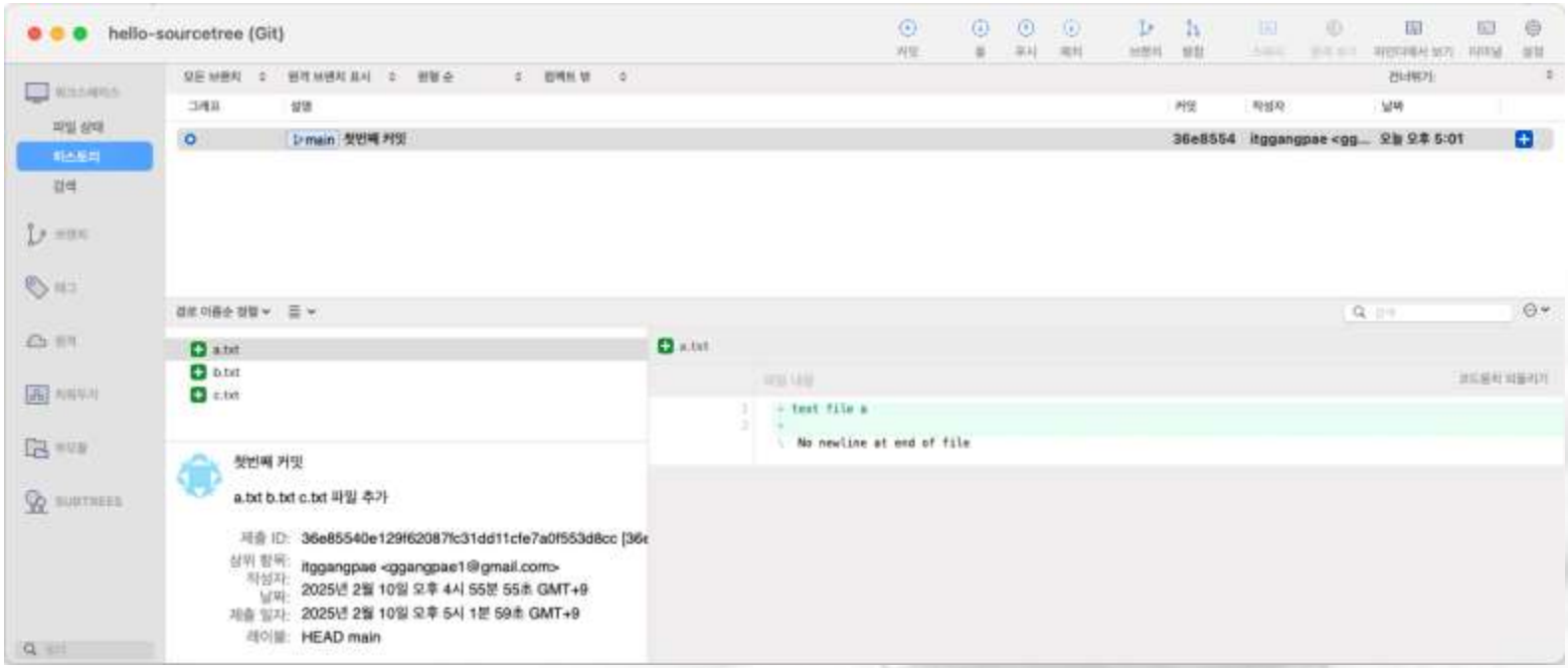
Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - Commit
      - ◆ Commit이 완료된 현재 상태: 대기 중인 파일이 없음



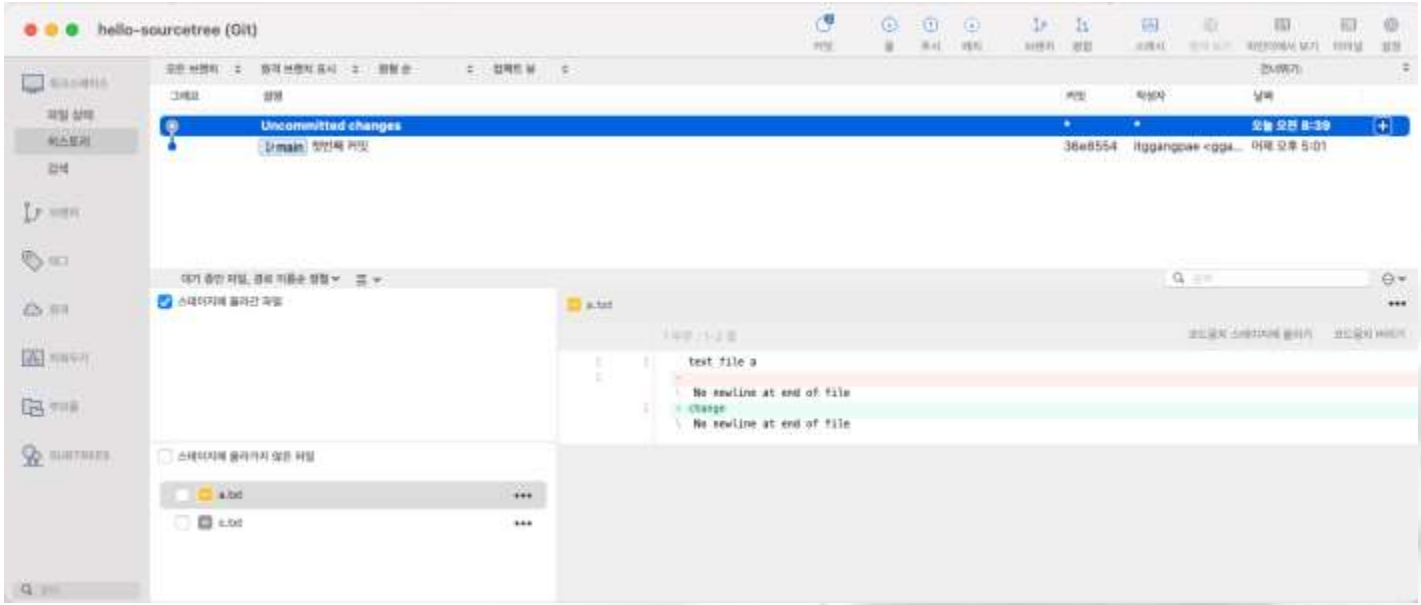
Source Tree

- ❖ 버전 관리 시작
  - ✓ 첫 버전 생성
    - Commit
      - ◆ Commit 내역은 History에서 확인 가능



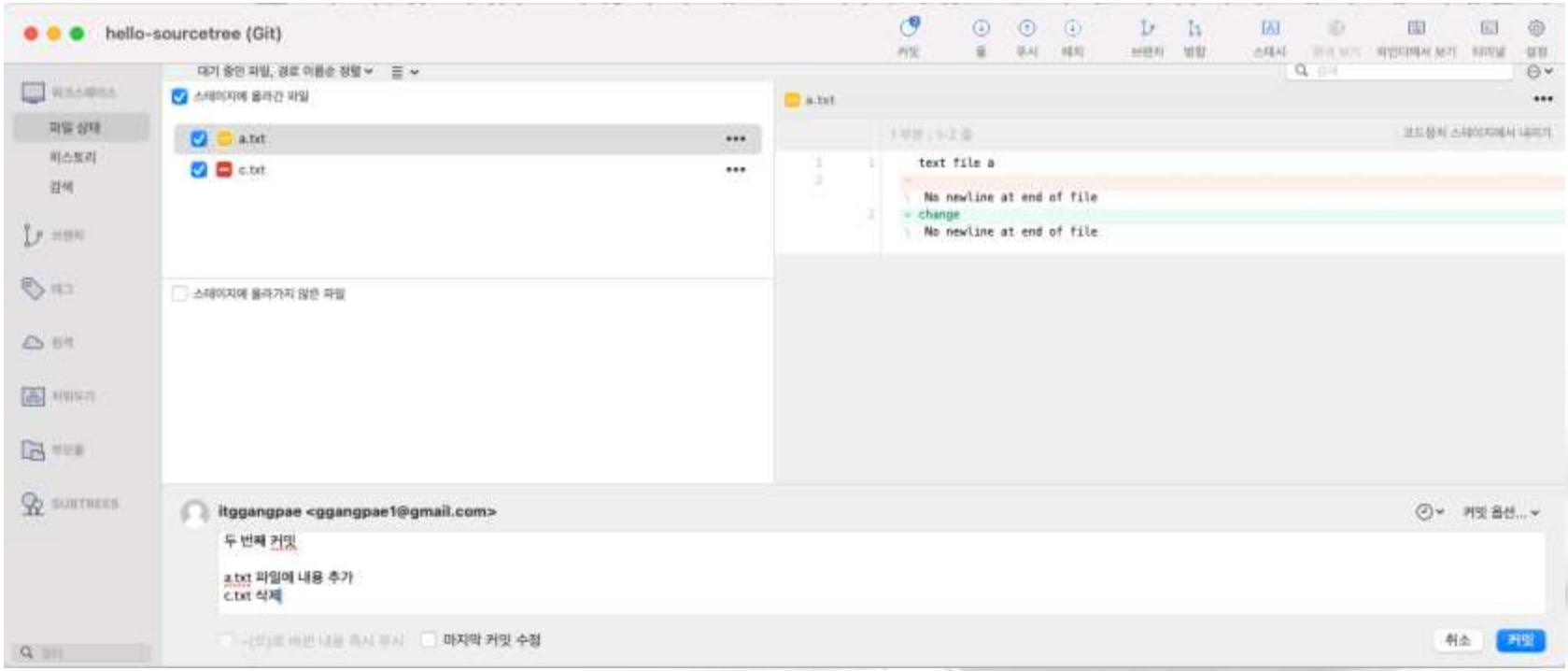
Source Tree

- ❖ 버전 관리 시작
  - ✓ Commit 쌓아 올리기
    - 새로운 Commit을 만들기 위해서 a.txt를 수정(changed를 추가)
    - c.txt 파일은 삭제
    - History에서 Commit하지 않은 변경 사항 확인



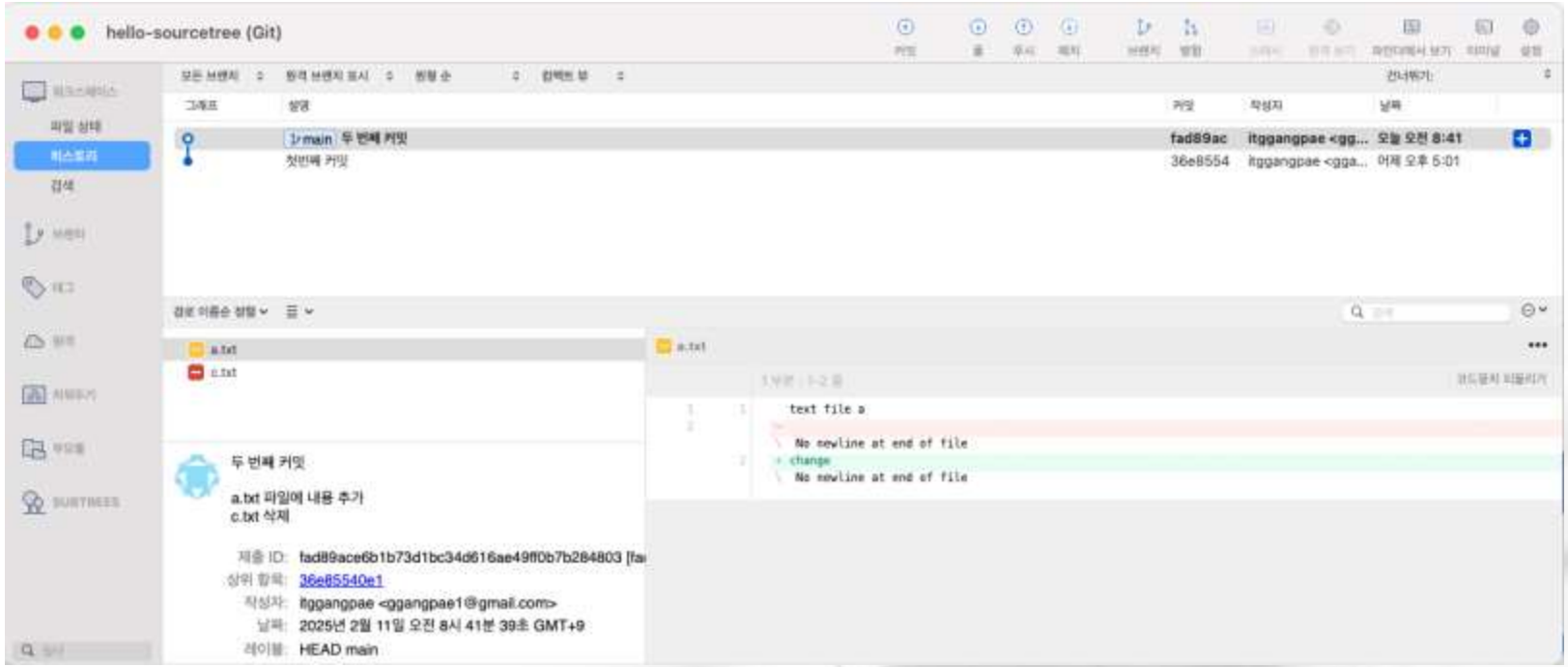
Source Tree

- ❖ 버전 관리 시작
  - ✓ Commit 쌓아 올리기
    - Staging Area 에 2개의 파일을 추가하고 Commit 수행



Source Tree

- 버전 관리 시작
  - ✓ Commit 쌓아 올리기
    - History를 눌러서 Commit 확인

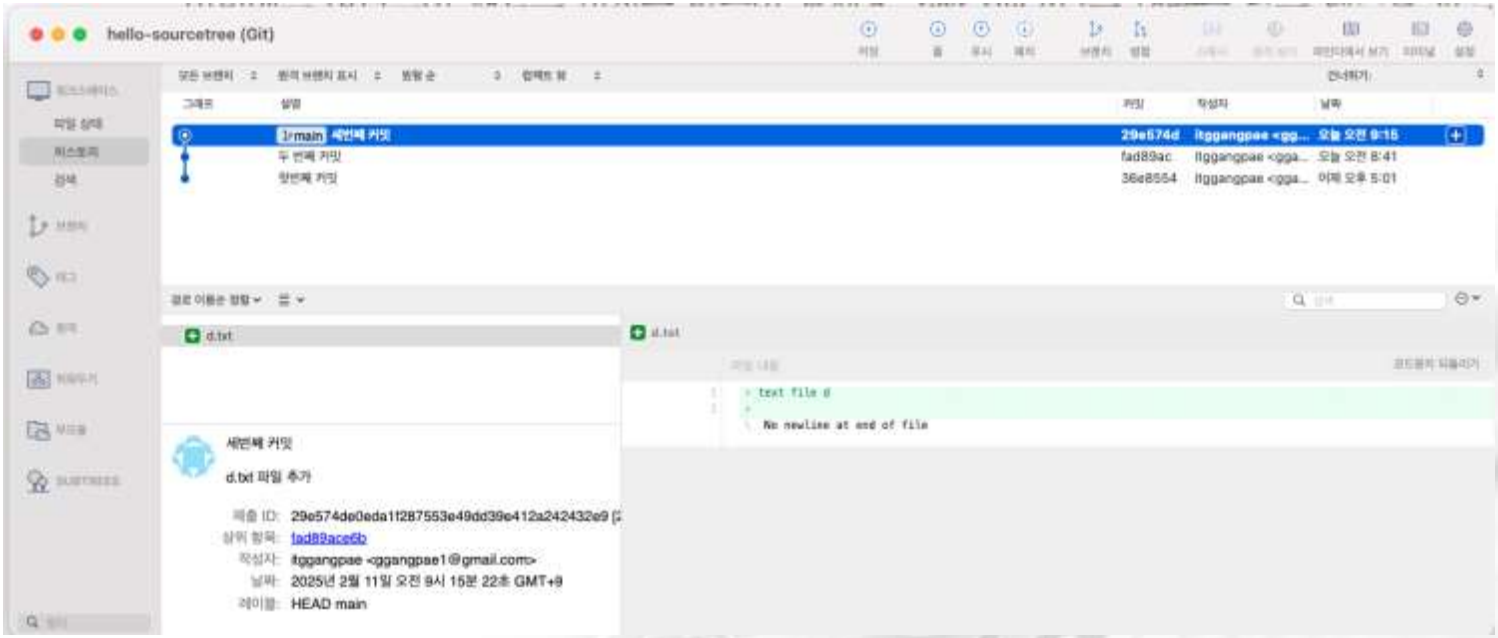


Source Tree

- ❖ 버전 관리 시작
  - ✓ Commit 쌓아 올리기
    - ● History를 눌러서 Commit 확인
      - ◆ Commit한 뒤 History로 들어가 보면 두 번째 Commit이 생성된 것을 확인할 수 있음
      - ◆ 그래프 항목을 보면 동그라미 두 개가 연결되어 있는 것을 볼 수 있는데 동그라미 하나는 Commit 하나 즉 하나의 버전을 나타내는데 두 번째 Commit'의 동그라미가 첫 번째 Commit 의 동그라미와 연결되어 있는 것은 두 번째 Commit이 첫 번째 Commit에서부터 만들어진 버전 임을 의미

Source Tree

- ❖ 버전 관리 시작
  - ✓ Commit 쌓아 올리기: 연습
    - text file d 라는 내용을 가진 d.txt 파일을 생성하고 스테이지에 올린 뒤 Commit하기
      - ◆ 메시지 제목: 세 번째 Commit
      - ◆ 메시지 내용: d.txt 파일 추가



Source Tree

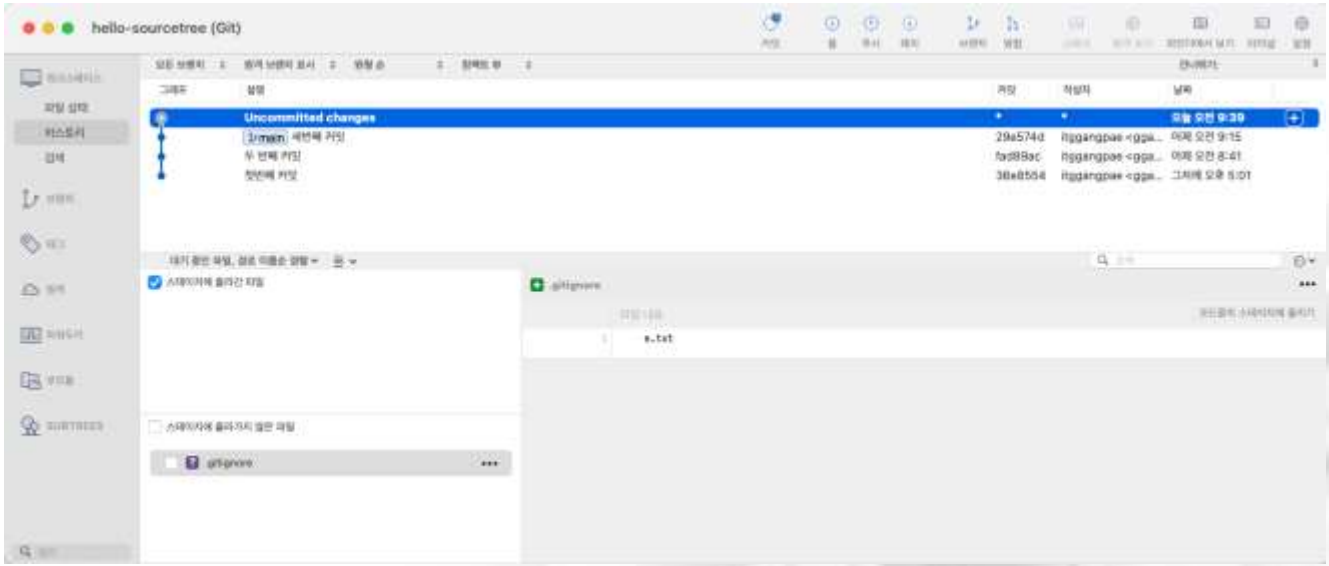
- 버전 관리 시작
  - ✓ 관리 대상(tracked) 파일과 관리 대상이 아닌(untracked) 파일
    - a.txt 파일을 수정하고 c.txt 파일을 삭제한 두 번째 버전을 만들 때는 펜 모양의 아이콘과 빼기 모양의 아이콘이 뜨고 새롭게 d.txt 파일을 추가한 세 번째 버전을 만들 때는 물음표 모양의 아이콘이 보임
    - 물음표 모양의 아이콘은 깃이 기존에 변경 사항을 추적하지 않았던 새로운 파일을 의미하는데 이런 종류의 파일 즉 기존에 깃이 관리하지 않았던 파일을 untracked 상태에 있는 파일
    - 깃이 변경 사항을 추적하고 있는 파일은 스테이지에 올라왔거나 한 번이라도 Commit 된 적 있는 파일로 이런 파일을 tracked 상태에 있는 파일이라 함
    - tracked 상태의 파일이 삭제된 경우에는 c.txt처럼 빼기 모양 아이콘으로 나타내고 수정된 경우에는 a.txt처럼 펜 모양 아이콘으로 나타남

Source Tree

- 버전 관리 시작
  - ✓ .gitignore
    - 개요
      - 깃으로 변경 사항을 추적하고 싶지 않은 파일이나 디렉토리가 있을 수 있는데 이러한 파일이나 디렉토리는 .gitignore 파일로 무시할 수 있음
      - .gitignore 파일은 무시할 파일/디렉토리 목록을 적은 파일
      - 깃은 .gitignore 파일에 적은 파일이나 디렉토리에 변경 사항이 생겨도 이를 무시

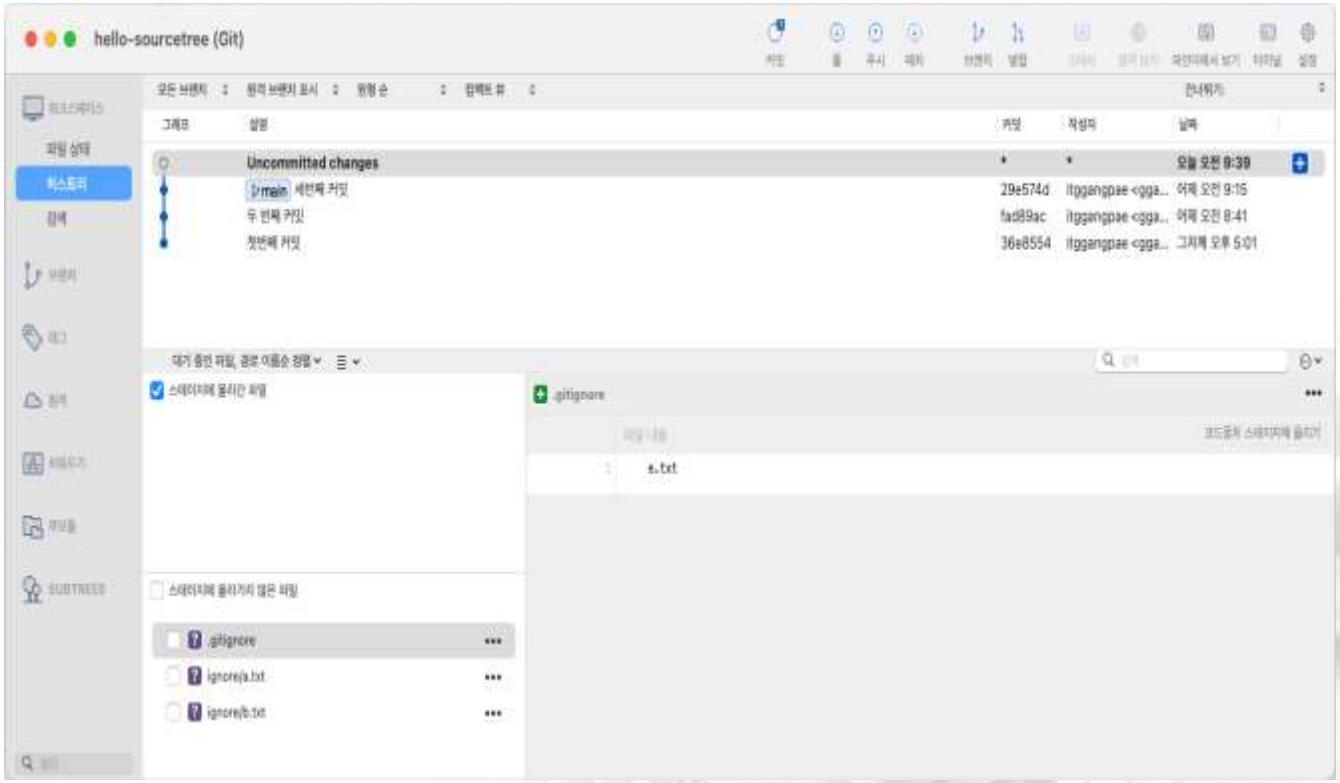
Source Tree

- ❖ 버전 관리 개요
  - ✓ .gitignore
    - ● 파일 제외
      - ◆ e.txt 파일을 생성하고 text file e 로 내용을 작성
      - ◆ .gitignore 파일을 생성하고 e.txt 라고 작성
      - ◆ 작업 디렉터리에 e.txt 파일이 새롭게 추가됐는데도 소스 트리의 스테이지에 올라가지 않은 파일 항목에 e.txt 파일이 생성되지 않는 것을 확인할 수 있음



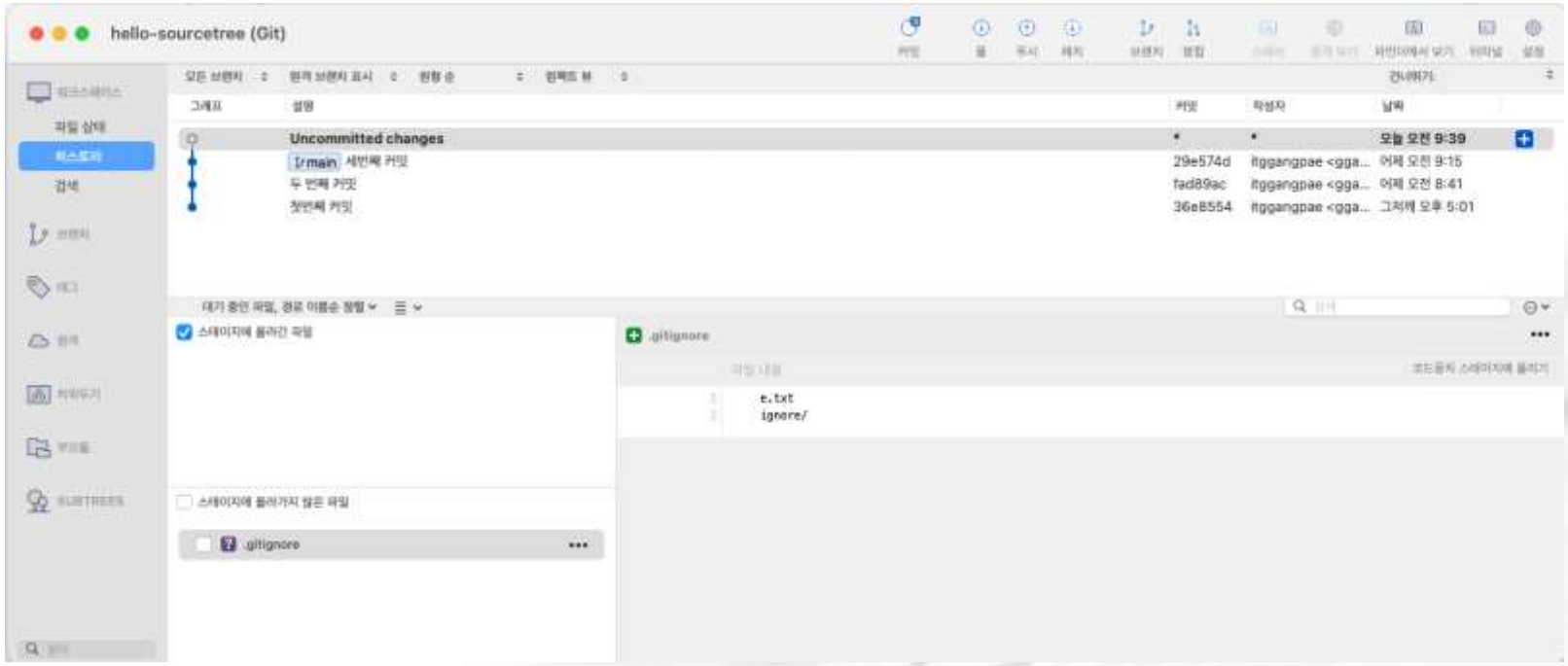
Source Tree

- ❖ 버전 관리 개요
  - ✓ .gitignore
    - ● 디렉토리 제외
      - ◆ ignore 디렉토리를 생성하고 파일을 복사



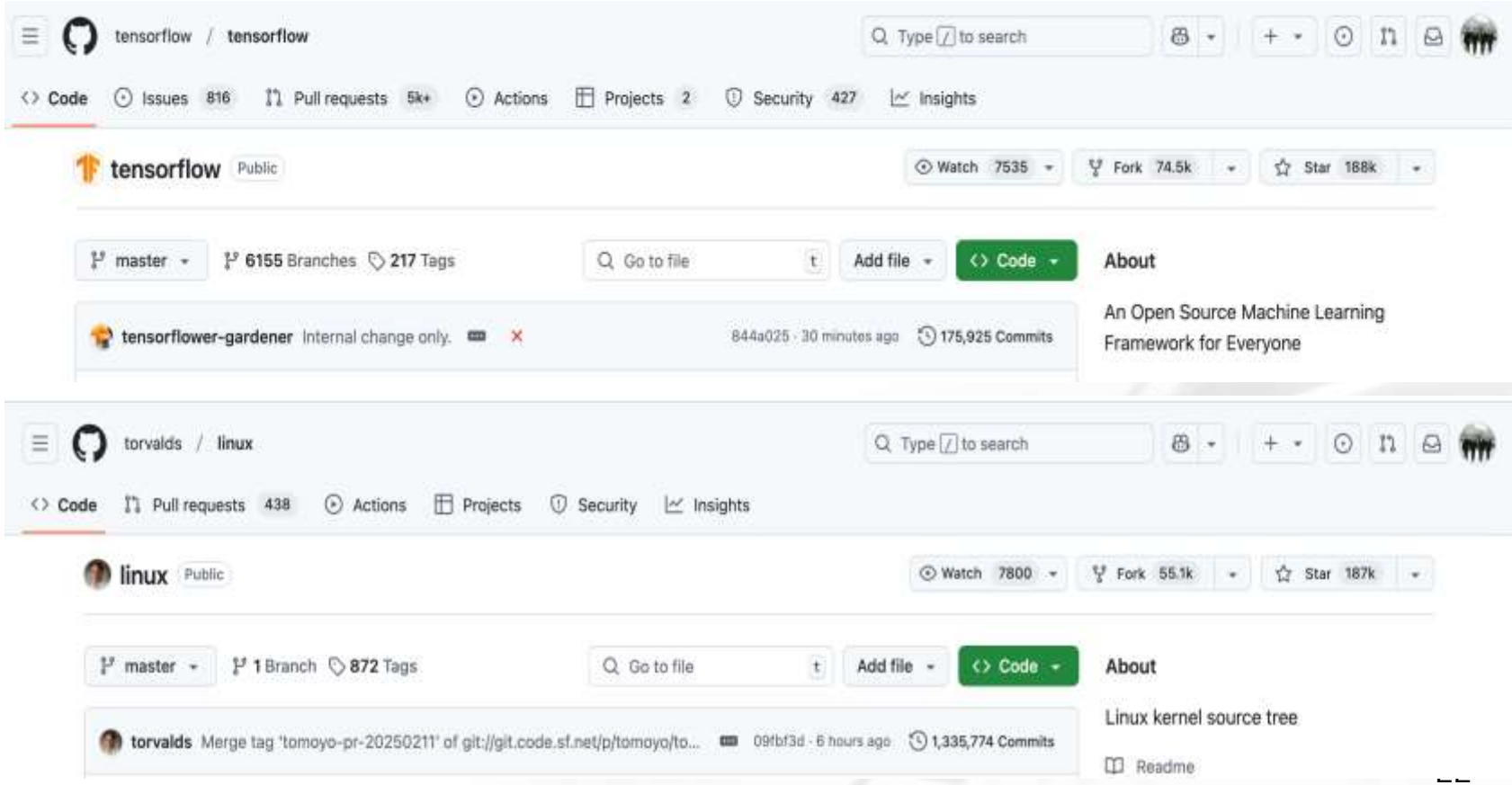
Source Tree

- ❖ 버전 관리 개요
  - ✓ .gitignore
    - ● 디렉토리 제외
      - ◆ .gitignore 파일을 생성하고 ignore/ 라고 내용을 추가
      - ◆ 작업 디렉터리에 ignore 디렉토리가 새롭게 추가됐는데도 소스 트리의 스테이지에 올라가지 않은 파일 항목에 디렉토리가 보이지 않음



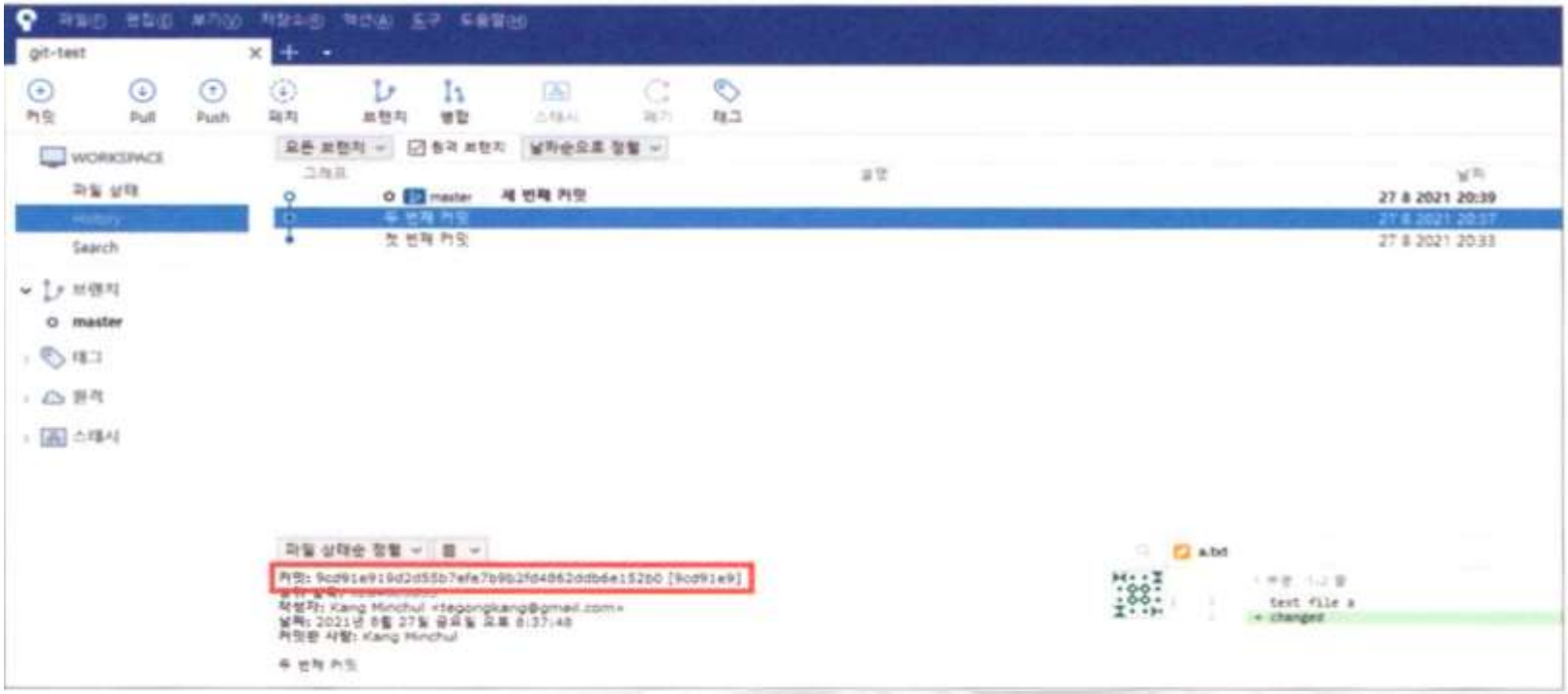
Source Tree

- ❖ 버전 조회
  - ✓ Commit 자세히 보기
    - 커다란 건물이 작은 벽돌 하나하나가 모여 만들어지듯 커다란 프로젝트는 Commit들이 모이고 모여 만들어지며 딥러닝에 사용되는 텐서플로는 Commit이 10만 개 이상 모여 만들어졌고 리눅스 운영체제는 100만 개 이상 모여 만들어 짐



Source Tree

- ❖ 버전 조회
  - ✓ Commit 자세히 보기
    - ● Commit Hash
      - ◆ 각 Commit에는 고유한 Commit Hash라고 하는 고유한 ID가 부여됨

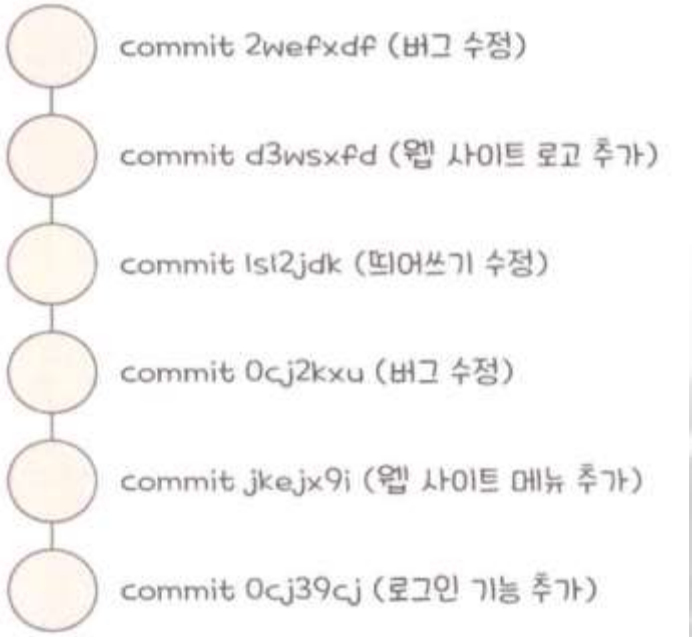


Source Tree

- ❖ 버전 조회
  - ✓ Commit 자세히 보기
    - Commit Hash
      - ◆ Hash 값의 길이가 너무 길어서 Hash 값의 앞부분 일부만 활용 가능
      - ◆ 소스 트리의 Commit 항목(O)은 특정 Commit을 지칭하기 위해 Commit Hash의 앞부분을 사용
      - ◆ 파일 상태 순 정렬 부분에서 상위 항목(@) 또한 특정 Commit을 지칭하기 위해 짧은 Commit Hash를 사용
      - ◆ 상위 항목이란 이 Commit이 어떤 Commit을 변경해서 만들어진 Commit인지를 나타냄

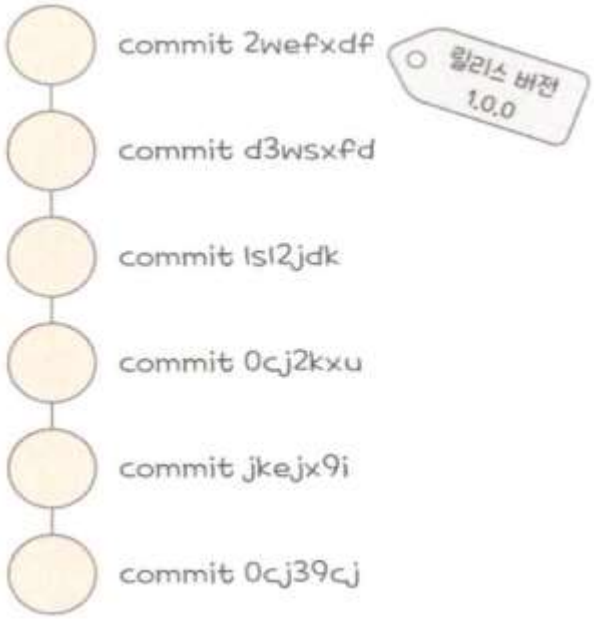
Source Tree

- ❖ 버전 조회
  - ✓ 태그
    - 웹 서비스를 만드는 과정에서 여러 Commit이 쌓이게 되는데 생성된 Commit 중에는 로그인 기능, 글쓰기 기능과 같이 새로운 기능을 추가하는 Commit도 있을 것이고 버그를 수정하는 Commit도 있을 것이고 아주 사소한 변경 사항만 담은 Commit도 있을 것



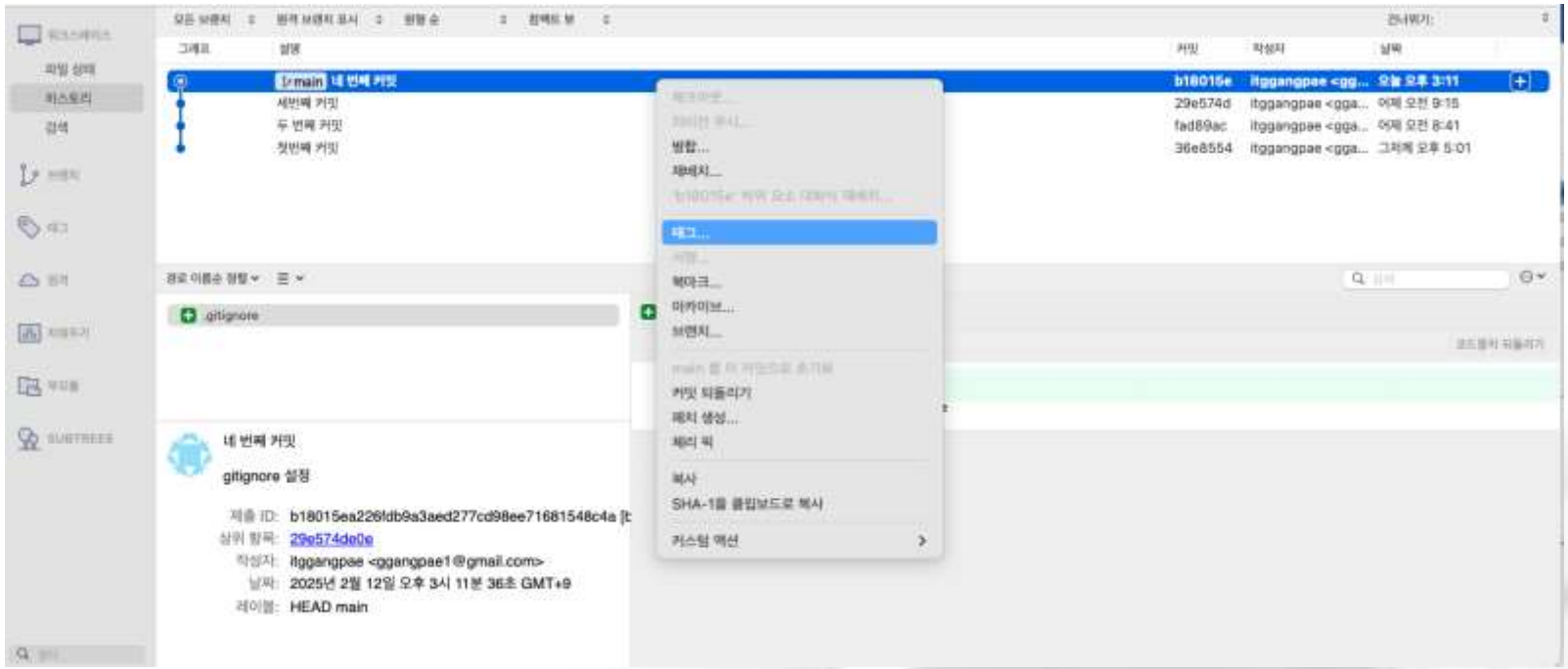
Source Tree

- ❖ 버전 조회
  - ✓ 태그
    - 사용자가 사용할 수 있도록 배포하는 작업을 Release라고 하는데 이 경우 버전을 어떻게 표시할 건인가 하는 문제가 발생하는데 Commit Hash는 무작위 문자열이라 가독성이 좋지 못함
    - 태그는 특정 Commit에 붙일 수 있는 꼬리표와 같은 것
    - 릴리스되는 Commit에 태그를 붙인다면 Commit이 여러 개 있는 상황에서도 의미 있는 Commit이 무엇인지 한눈에 알아보기 쉬움



Source Tree

- ❖ 버전 조회
  - ✓ 태그
    - 태그 생성: 태그 Commit을 선택하고 마우스 오른쪽을 눌러서 [태그]를 클릭



Source Tree

- ❖ 버전 조회
  - ✓ 태그
    - ● 태그 생성: 태그 입력

Source Tree

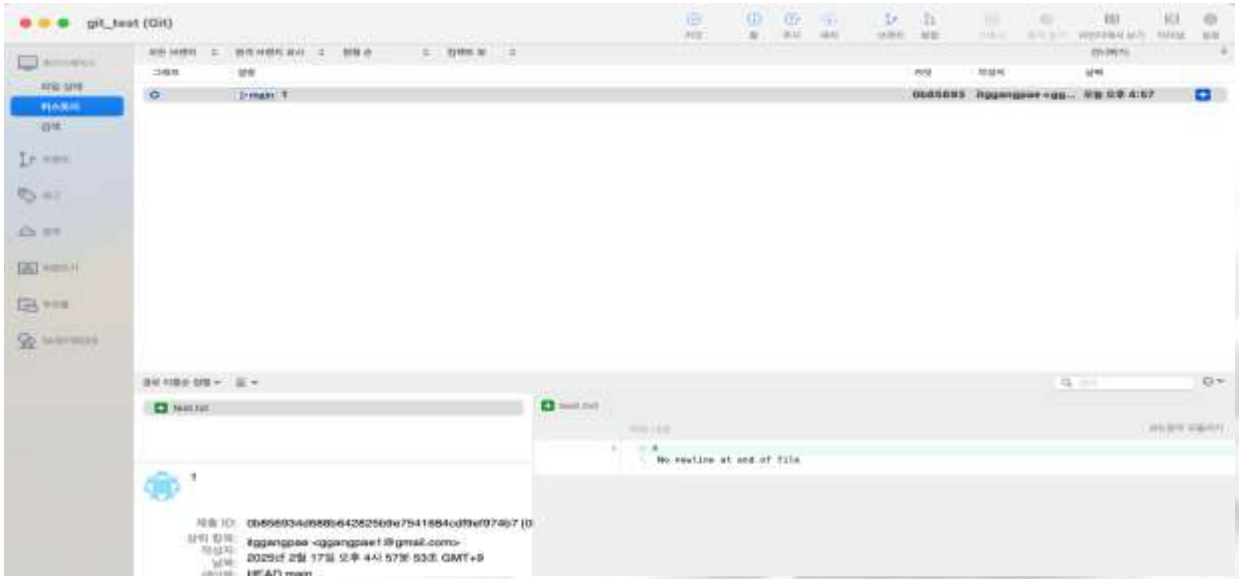
- ❖ 버전 조회
  - ✓ 태그
    - ● 가장 일반적인 버전 표기법

      vX.Y.Z

      - ◆ 가장 앞에 나오는 숫자는 Major 버전이라고 부르는데 이는 가장 중요한 버전으로 일반적으로 새롭게 내놓은 버전이 기존에 내놓은 버전과 호환되지 않을 정도로 큰 변화가 있을 때 증가
      - ◆ 두 번째 숫자는 Minor 버전이라고 부르는데 일반적으로 새롭게 내놓은 버전이 기존에 내놓은 버전과 문제없이 호환되지만 새로운 기능을 추가했을 때 증가
      - ◆ 마지막 숫자는 Patch 버전이라 부르는데 일반적으로 기존에 내놓은 버전과 문제없이 호환되며 버그를 수정한 정도의 작은 변화가 있을 때 증가

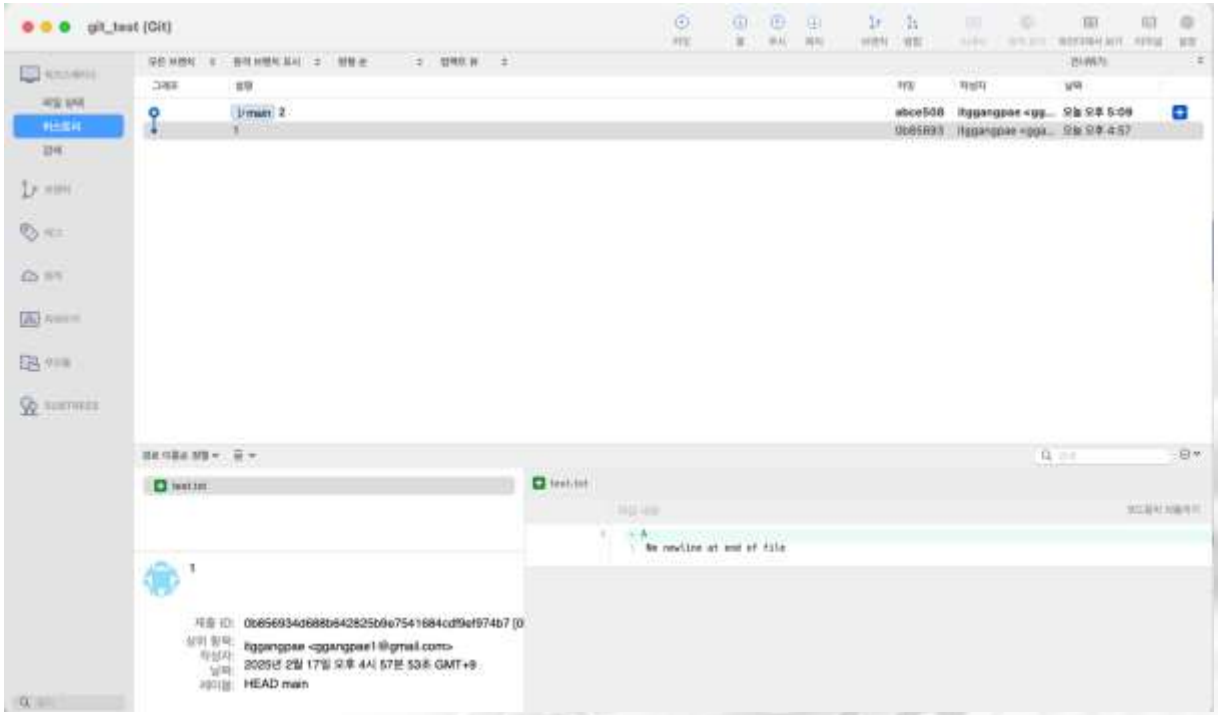
Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - ● 초기 설정
      - ◆ 디렉토리를 생성하고 그 디렉토리를 로컬 저장소로 설정
      - ◆ test.txt 파일을 생성하고 A를 추가
      - ◆ test.txt 파일을 스테이지에 올리고 Commit 메시지를 1로 해서 Commit



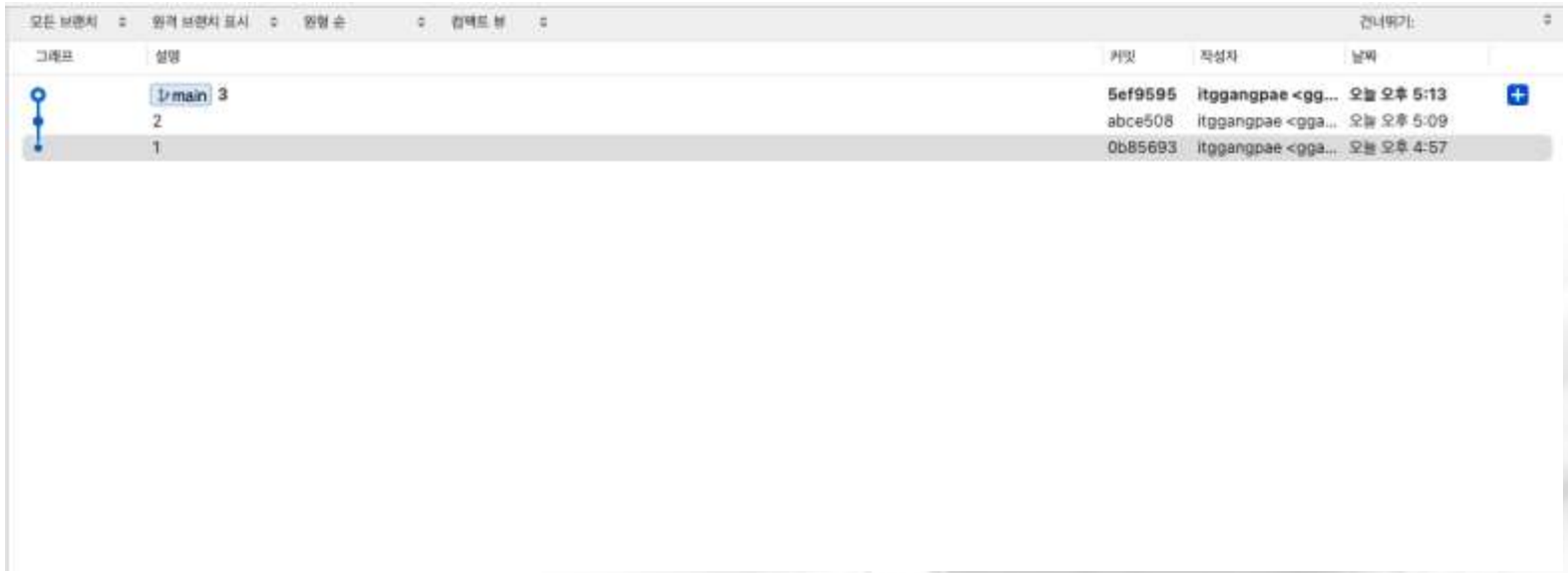
Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - Commit 추가
      - ◆ test.txt 파일에 B를 추가
      - ◆ test.txt 파일을 스테이지에 올리고 Commit 메시지를 2로 해서 Commit



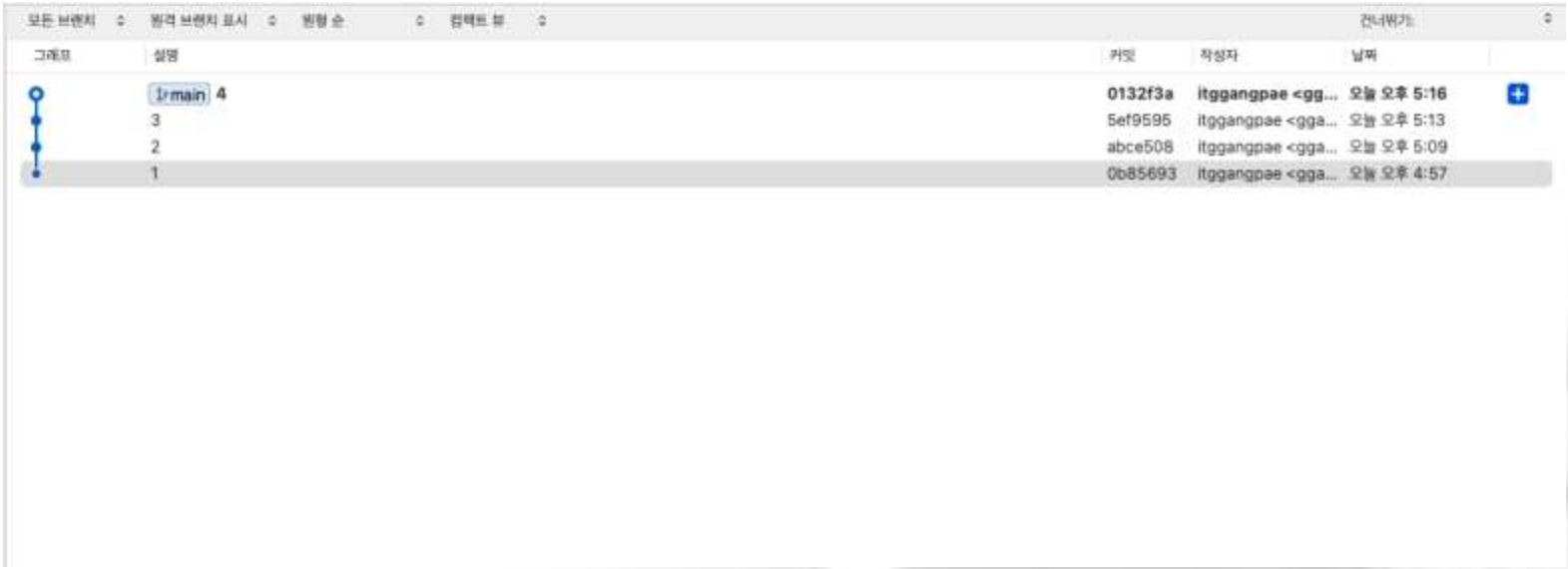
Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - ● Commit 추가
      - ◆ test.txt 파일에서 A를 제거
      - ◆ test.txt 파일을 스테이지에 올리고 Commit 메시지를 3로 해서 Commit



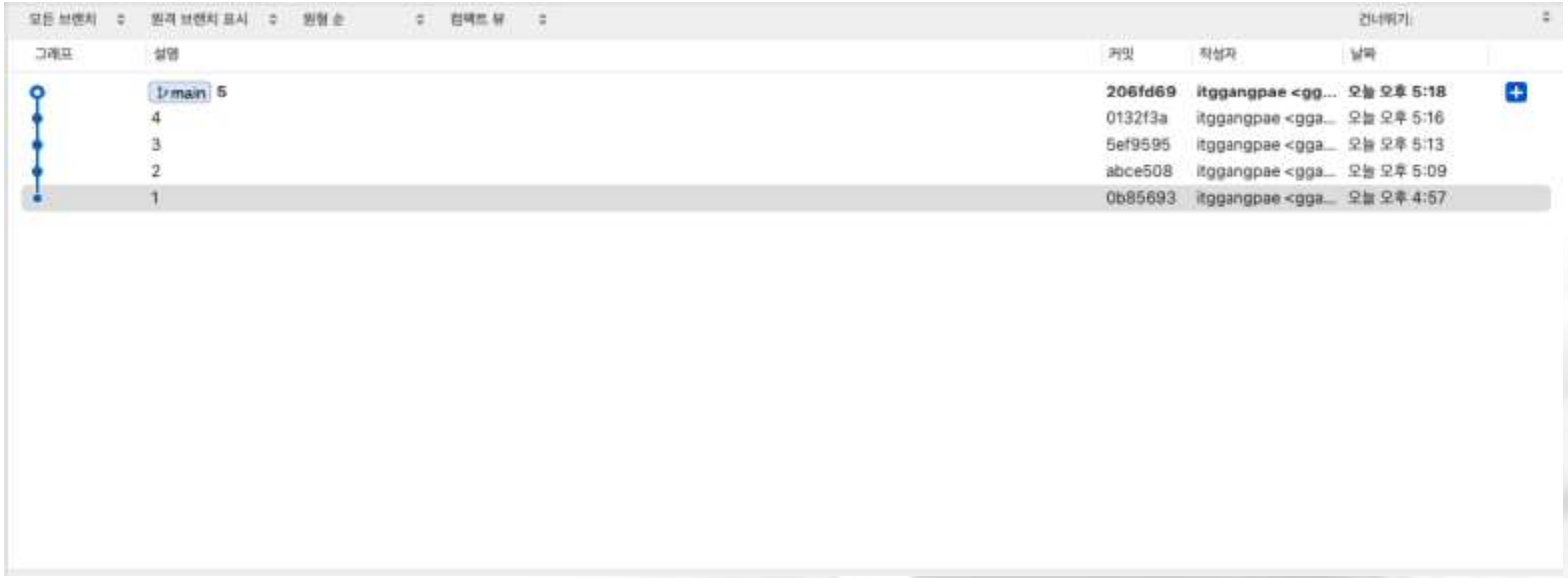
Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - Commit 추가
      - ◆ test.txt 파일에서 C를 추가
      - ◆ test.txt 파일을 스테이지에 올리고 Commit 메시지를 4로 해서 Commit



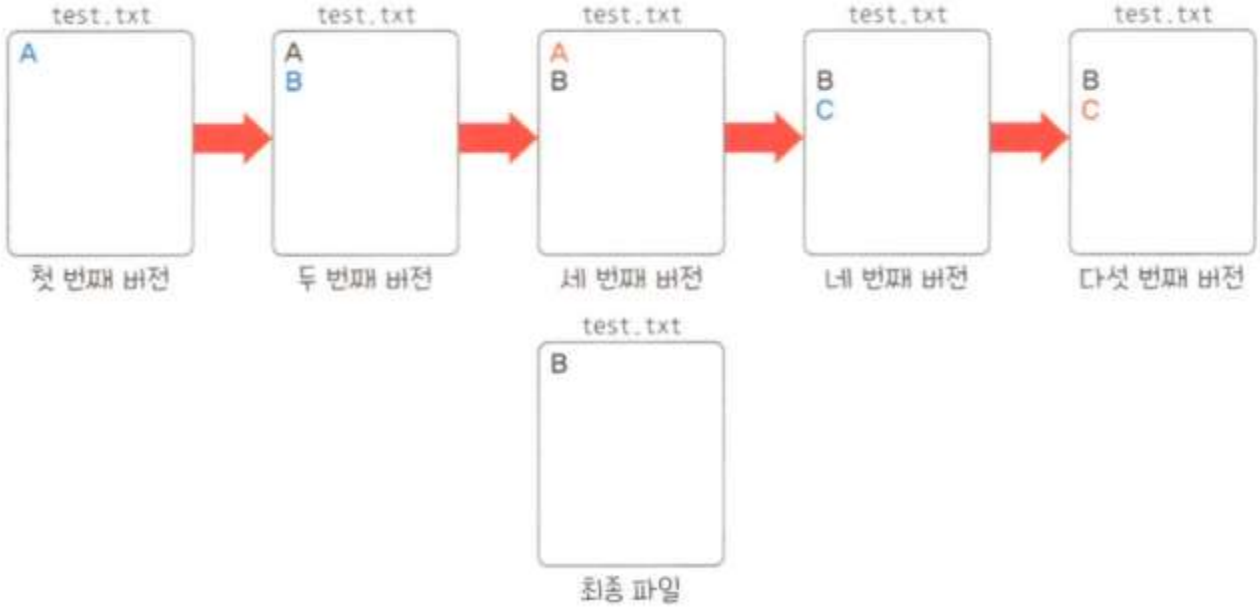
Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - Commit 추가
      - ◆ test.txt 파일에서 C를 제거
      - ◆ test.txt 파일을 스테이지에 올리고 Commit 메시지를 5로 해서 Commit



Source Tree

- ❖ 버전 관리 활용
  - ✓ 실습 환경 만들기
    - ● 현재 상태

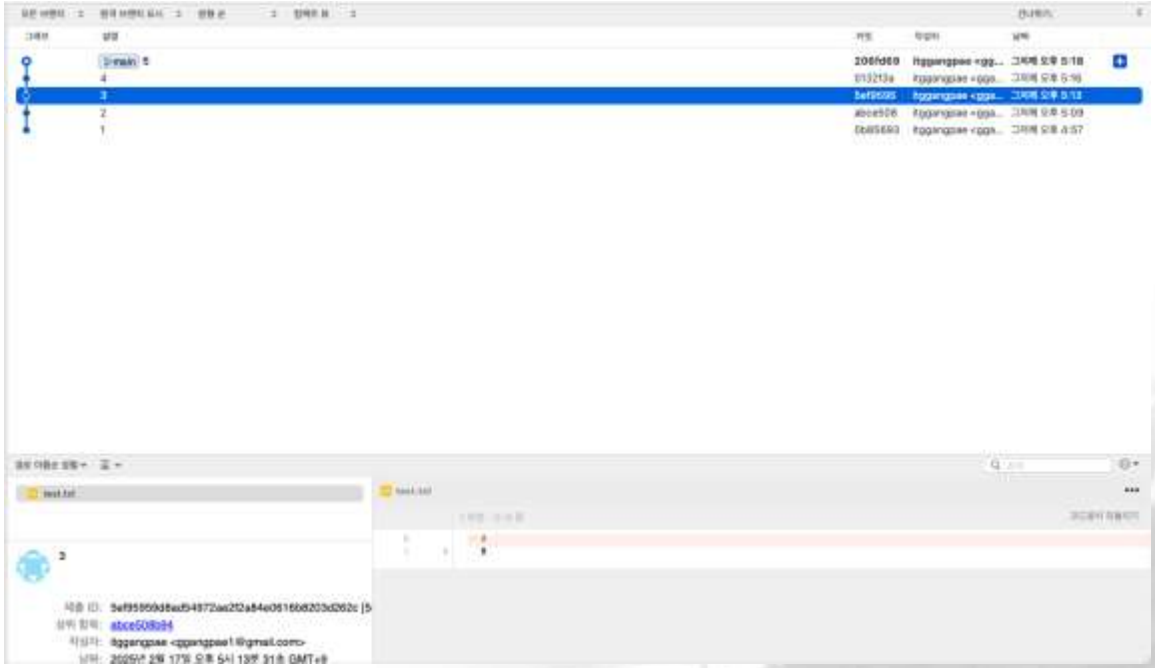


Source Tree

- ❖ 버전 관리 활용
  - ✓ 버전 비교
    - ● 직전 버전과 비교: 개별 Commit을 클릭해서 확인
      - ◆ 첫번째 버전 확인: A가 추가

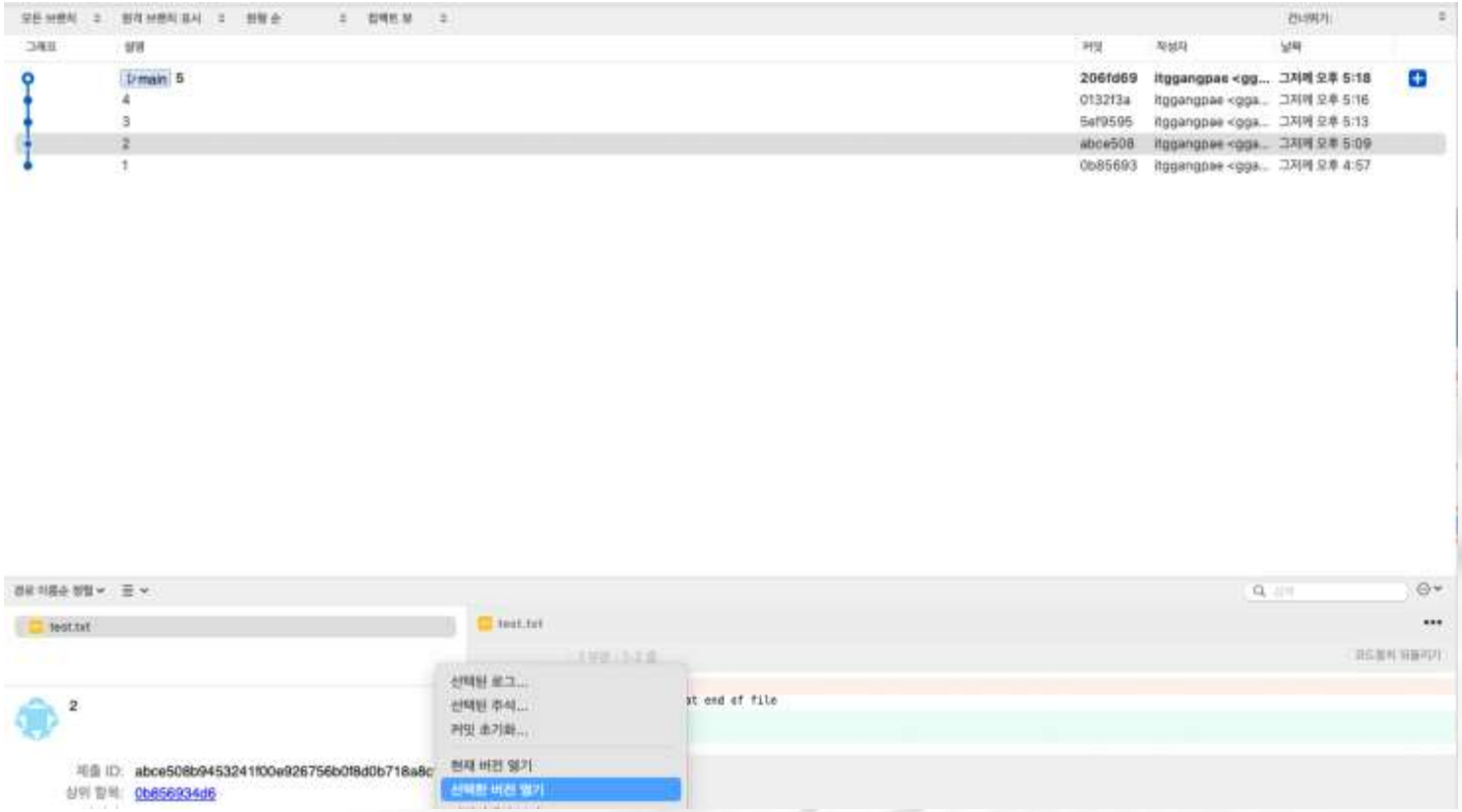
Source Tree

- ❖ 버전 관리 활용
  - ✓ 버전 비교
    - ● 직전 버전과 비교: 개별 Commit을 클릭해서 확인
      - ◆ 세번째 버전 확인: A 삭제



Source Tree

- ❖ 버전 관리 활용
  - ✓ 버전 비교
    - ● 버전 별 비교: Commit을 선택하고 마우스 오른쪽을 눌러서 [선택한 버전 열기]를 클릭
      - ◆ 두번째 Commit을 확인

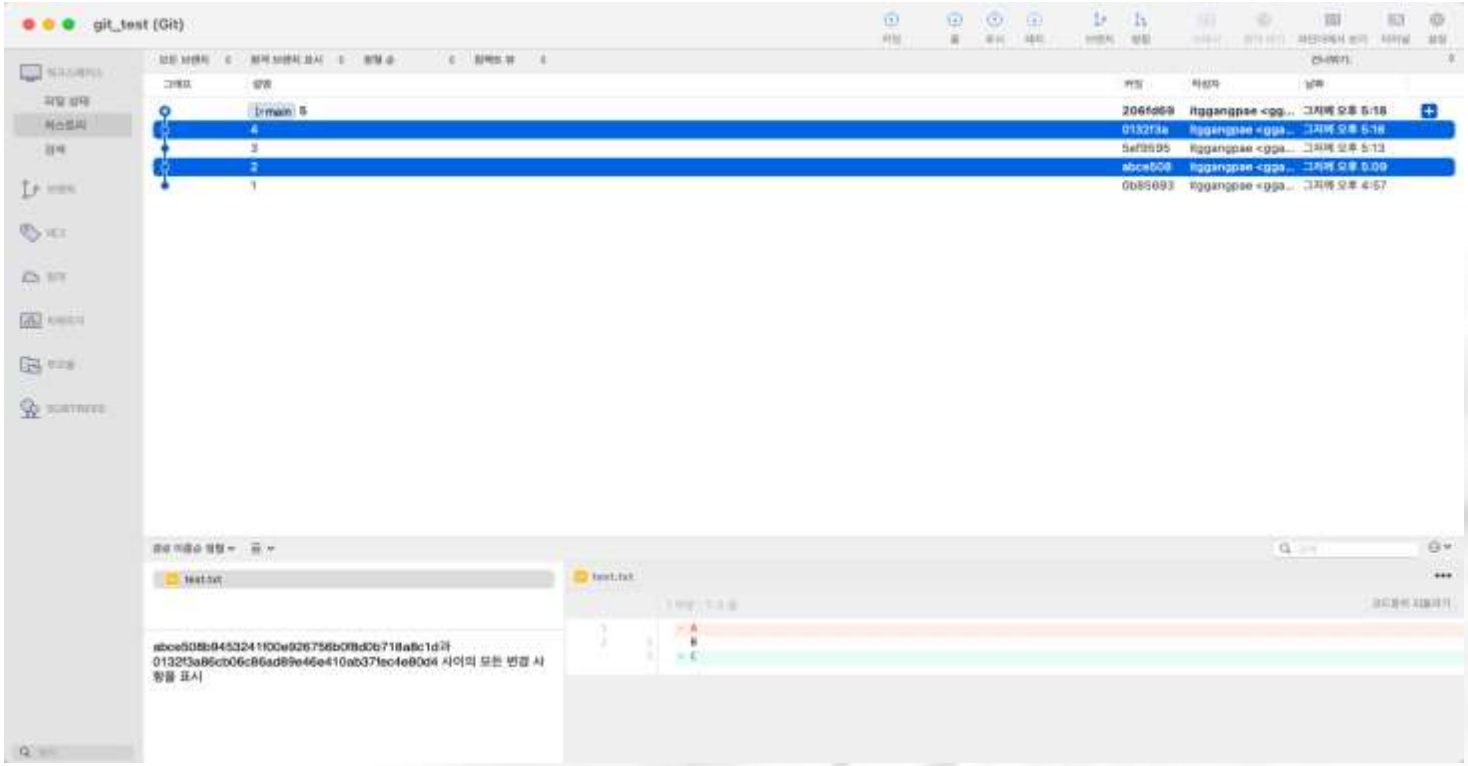


Source Tree

- ❖ 버전 관리 활용
  - ✓ 버전 비교
    - ● 버전 별 비교: Commit을 선택하고 마우스 오른쪽을 눌러서 [선택한 버전 열기]를 클릭
      - ◆ 네번째 Commit을 확인

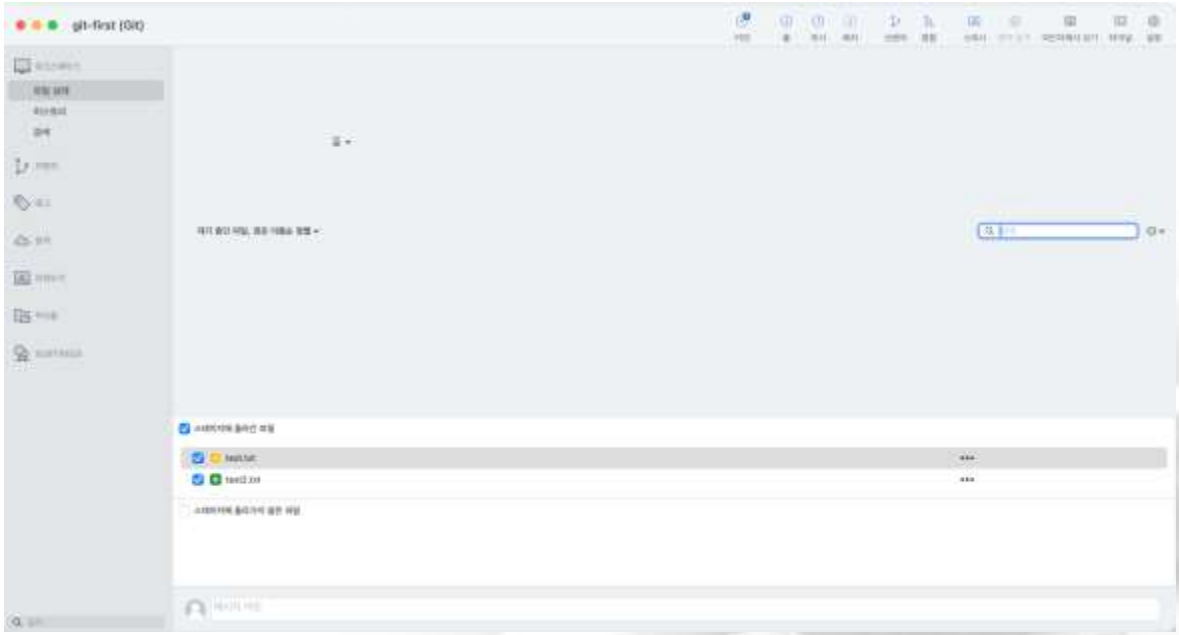
Source Tree

- ❖ 버전 관리 활용
  - ✓ 버전 비교
    - ● 버전 별 비교
      - ◆ 두번째 버전을 선택하고 CTRL 키를 누른 채 네번째 버전을 클릭하고 하단에서 test.txt 파일을 선택



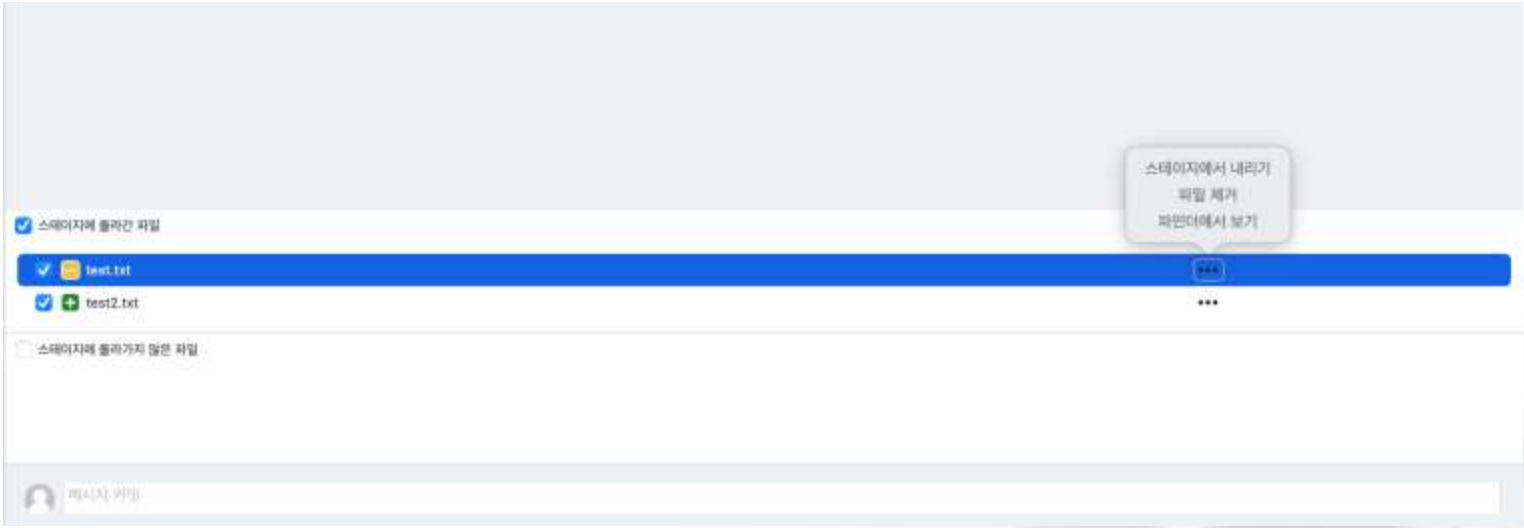
Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 스테이지에 올라간 파일 되돌리기
      - ◆ test.txt 파일에 C를 추가하고 A가 적힌 test2.txt 파일을 추가로 만든 뒤 저장
      - ◆ 2개의 파일 모두를 Stage로 이동



Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 스테이지에 올라간 파일 되돌리기
      - ◆ test.txt 파일을 선택하고 [선택 내용 스테이지에서 내리기] 버튼 클릭



Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 스테이지에 올라간 파일 되돌리기
      - ◆ test2.txt 파일을 선택하고 [선택 내용 스테이지에서 내리기] 버튼 클릭

Source Tree

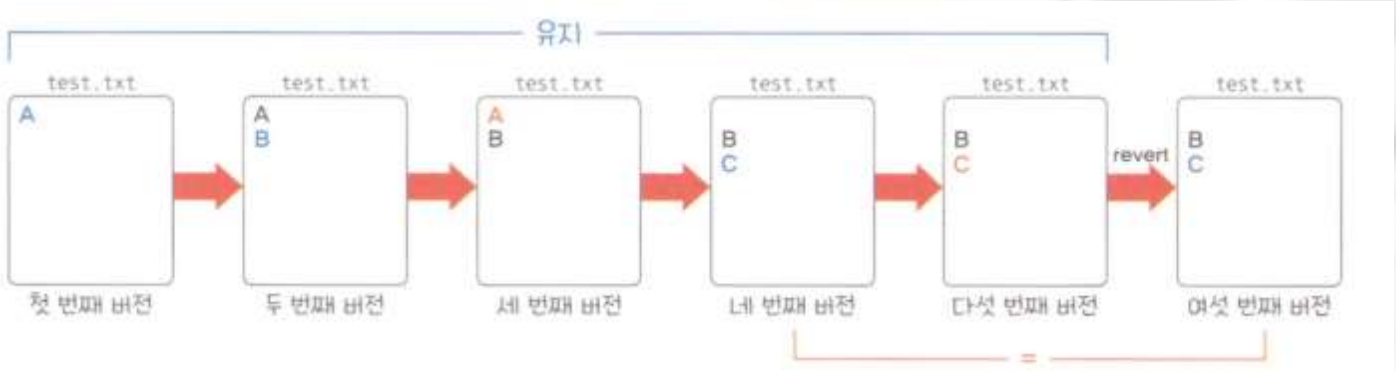
- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 스테이지에 올라가지 않은 파일 되돌리기
      - ◆ test.txt 파일을 선택하고 [초기화] 나 [폐기] 메뉴를 클릭

Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 스테이지에 올라가지 않은 파일 되돌리기
      - ◆ test2.txt 파일을 선택하고 [제거] 메뉴를 클릭

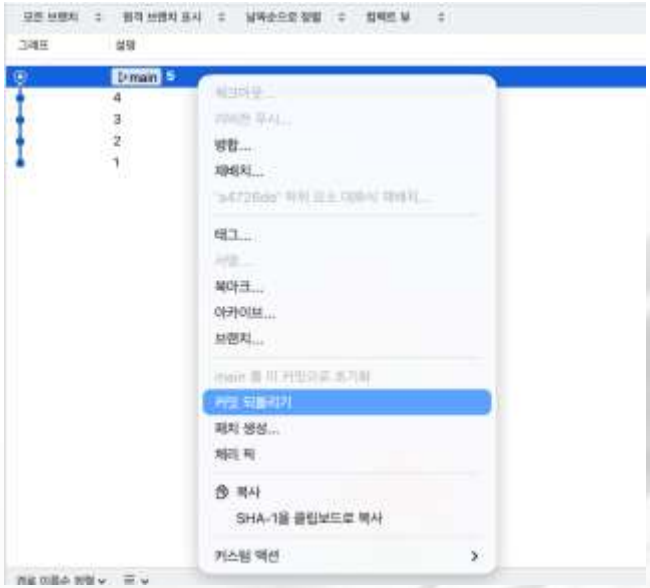
Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ revert
        - ▪ revert는 버전을 되돌리되 되돌아간 상태에 대한 새로운 버전을 만드는 방식
        - ▪ 기존의 버전은 삭제되지 않는다는 점



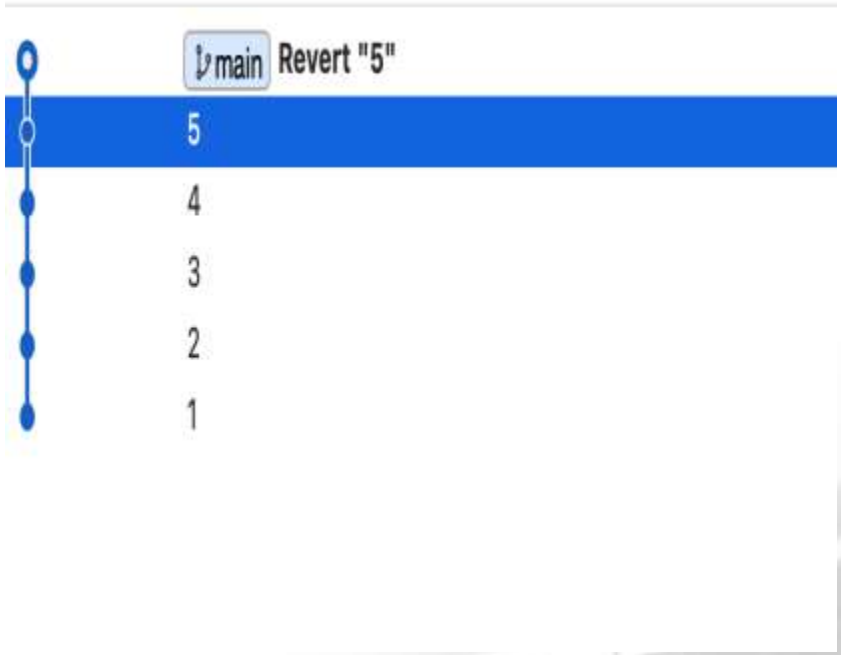
Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ revert
        - ▪ revert는 버전을 되돌리되 되돌아간 상태에 대한 새로운 버전을 만드는 방식
        - ▪ 기존의 버전은 삭제되지 않는다는 점
        - ▪ 다섯번째 버전을 선택하고 마우스 오른쪽 버튼을 클릭해서 [커밋 되돌리기]를 선택



Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ revert



Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ reset
        - ▪ 돌아갈 버전의 시점으로 완전하게 되돌아가는 방식
        - ▪ 되돌아 갈 버전 이후의 버전은 삭제되는 방식
        - ▪ 종류
          - soft: 작업 디렉토리 내 변경 사항과 스테이지에 추가된 변경 사항은 유지하되 커밋했다는 사실만 되돌리는 reset

Source Tree

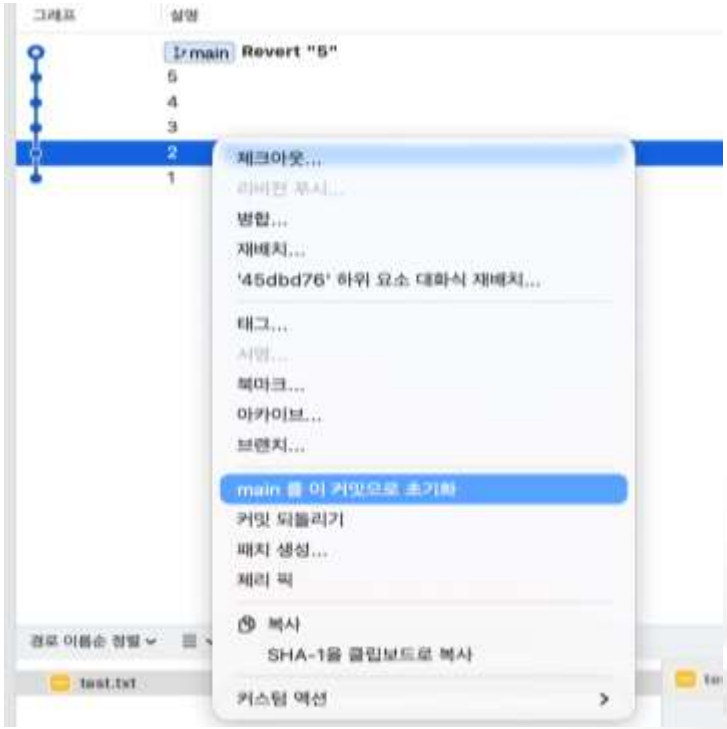
- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ reset
        - ▪ 종류
          - mixed: 스테이지 와 커밋을 되돌리는 reset

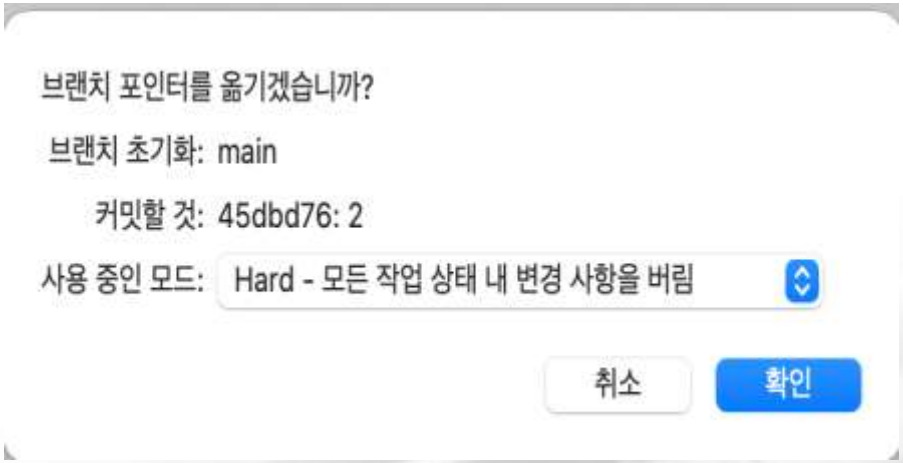
Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ reset
        - ▪ 종류
          - hard: 작업 디렉토리 내 변경 사항까지 통째로 되돌리는 reset

Source Tree

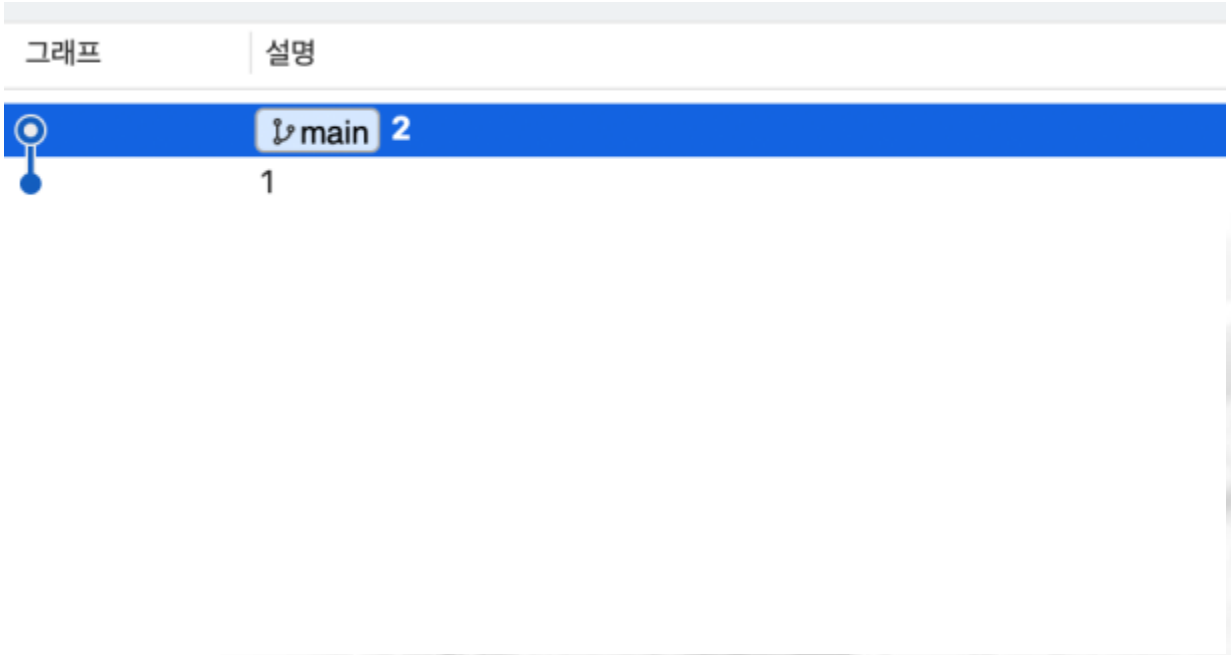
- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ reset
        - ▪ 두번째 커밋을 선택하고 마우스 오른쪽을 클릭해서]이 커밋까지 현재 브랜치를 초기화]를 선택





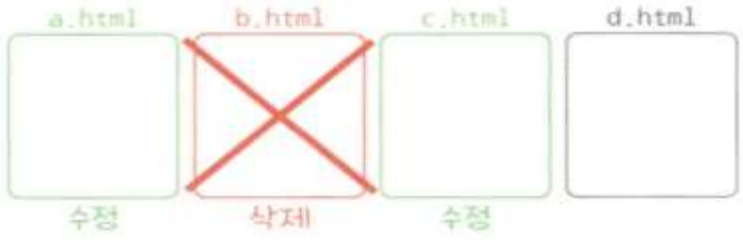
Source Tree

- ❖ 버전 관리 활용
  - ✓ 작업 되돌리기
    - ● 커밋한 내용 되돌리기
      - ◆ reset
        - ▪ 두번째 커밋을 선택하고 마우스 오른쪽을 클릭해서]이 커밋까지 현재 브랜치를 초기화]를 선택



Source Tree

- 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - stash를 하게 되면 작업 디렉토리에서 생성한 모든 변경 사항이 임시 저장되고 작업 디렉토리는 변경 사항이 생기기 전의 깨끗한 상태로 돌아감 – 깃이 변경 사항을 추적하는 파일에만 사용 가능
    - 작업 디렉토리에 있는 a.html, b.html, c.html. d.html 중에서 a.html과 c.html은 수정하고 b.html을 삭제하는 변경 사항을 생성했다고 가정
      - 현재 작업 디렉토리



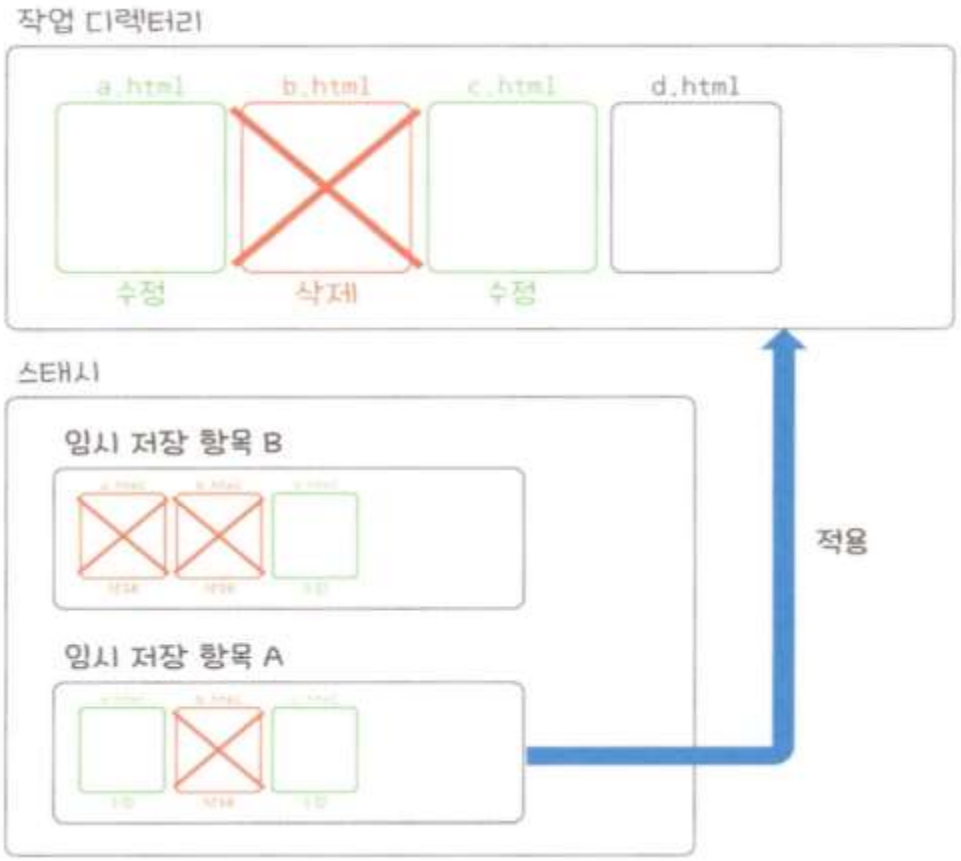
      - stash 명령 수행

Source Tree

- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 여러 개 저장 가능

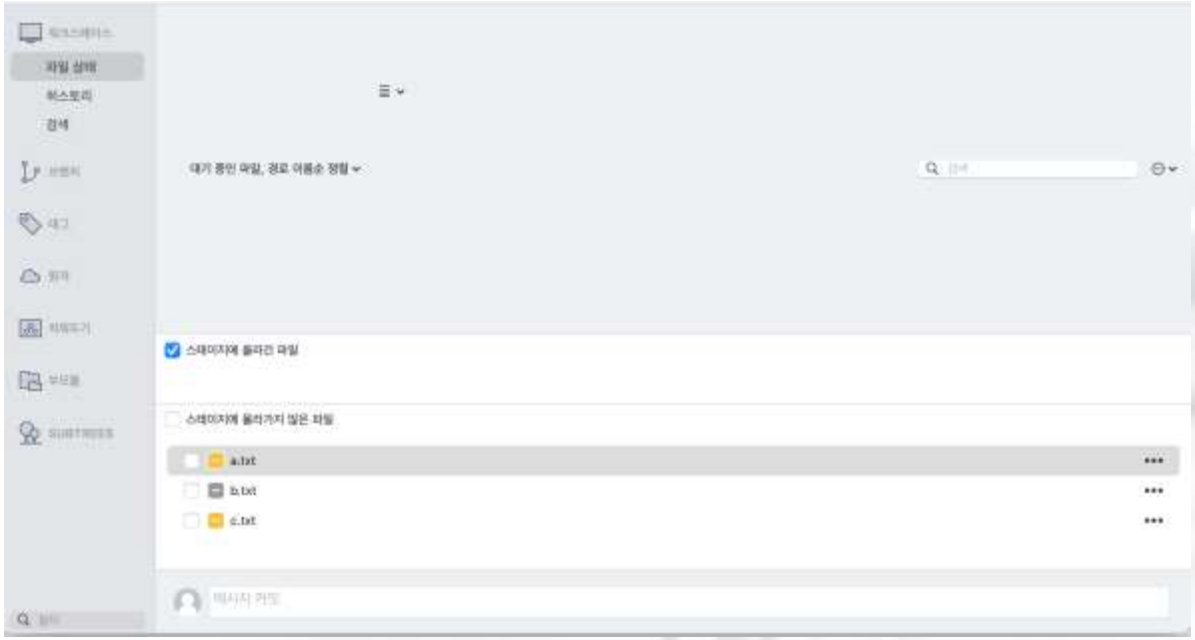
Source Tree

- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 임시 저장된 변경 사항 적용



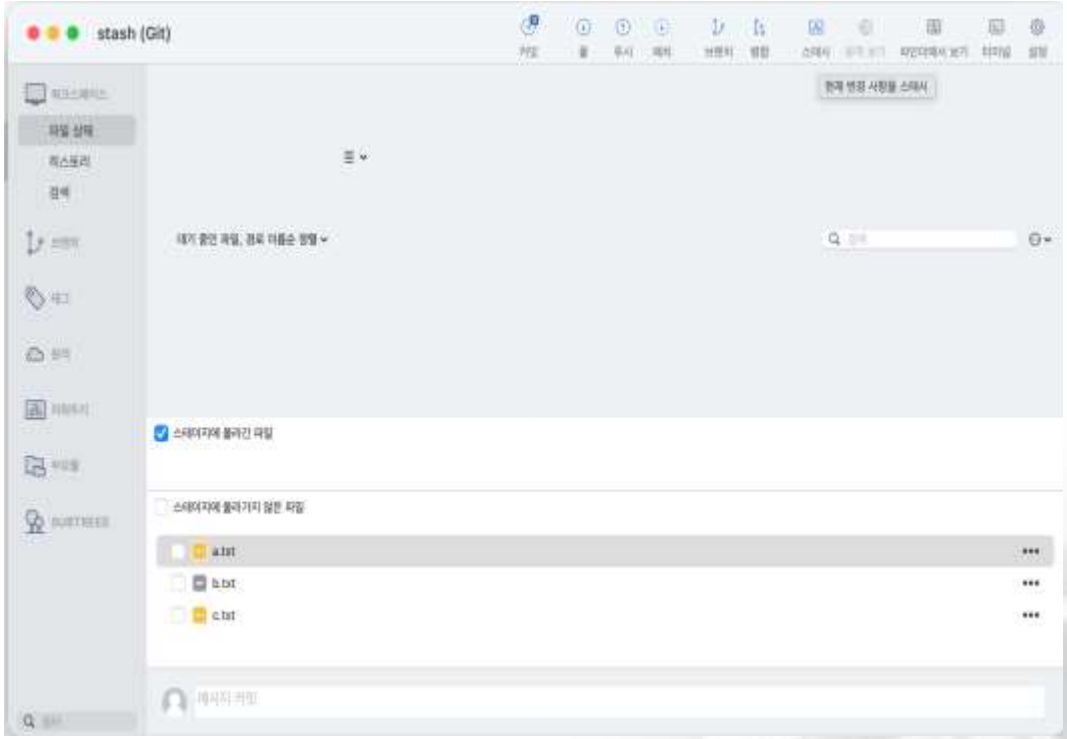
Source Tree

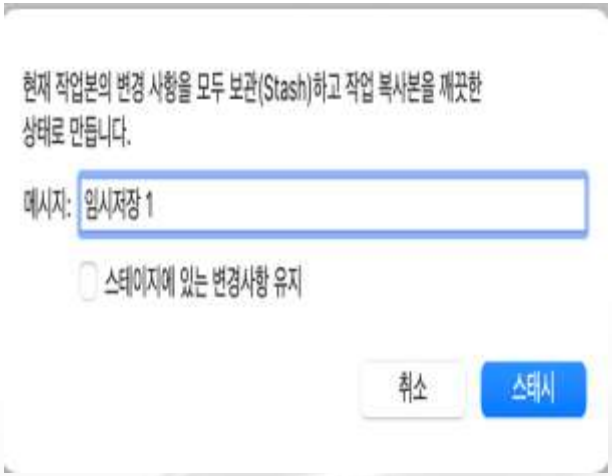
- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 실습
      - ◆ 임의의 로컬 저장소에 a.txt, b.txt, c.txt, d.txt, e.txt 파일을 만들고 5개의 파일에 A, B, C, D, E를 저장
      - ◆ 모두 stage에 올리고 커밋 메시지를 1로 해서 커밋
      - ◆ a.txt 파일에 B를 추가하고 b.txt 파일을 삭제한 후 c.txt 파일에 D를 추가
      - ◆ 현재 상태



Source Tree

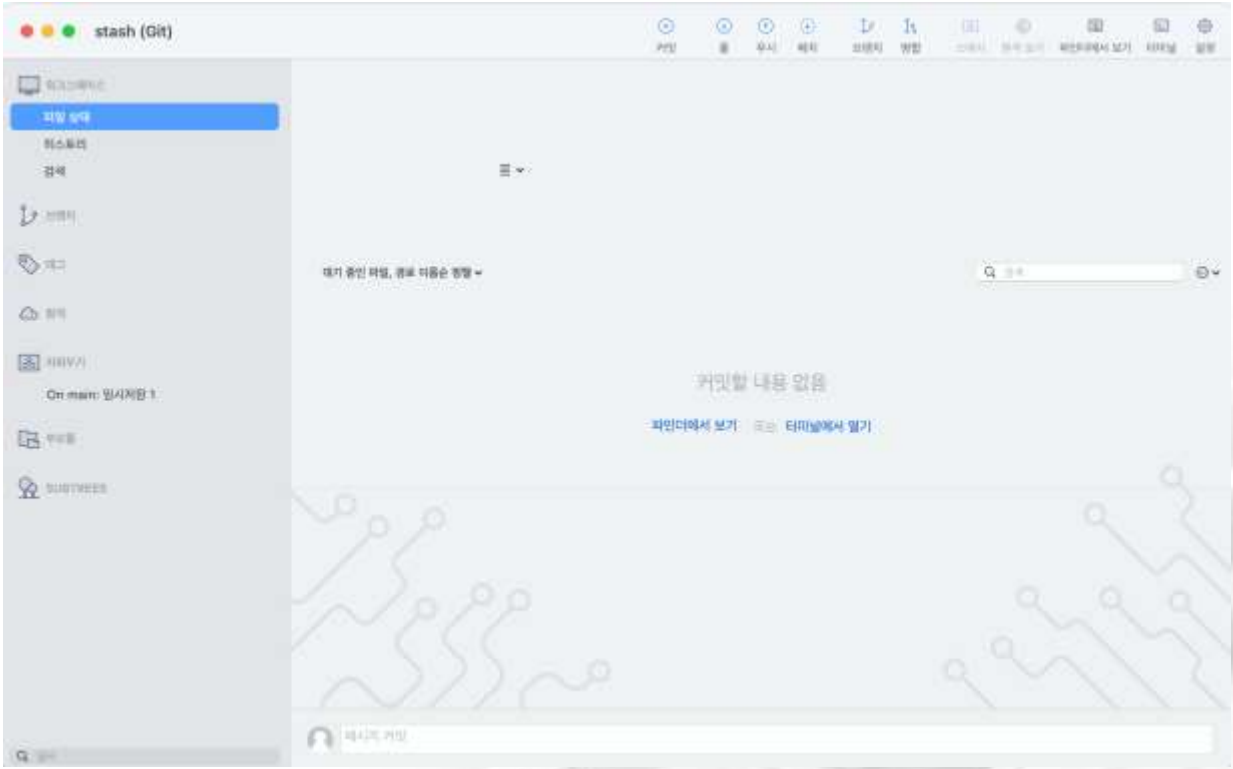
- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 실습
      - ◆ 임시 저장





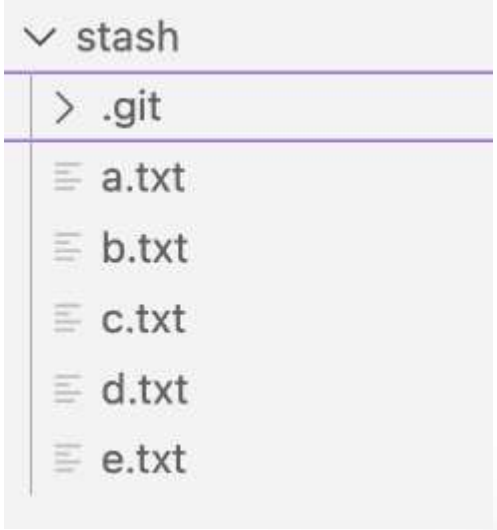
Source Tree

- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 실습
      - ◆ 현재 파일 상태



Source Tree

- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 실습
      - ◆ 현재 파일 상태

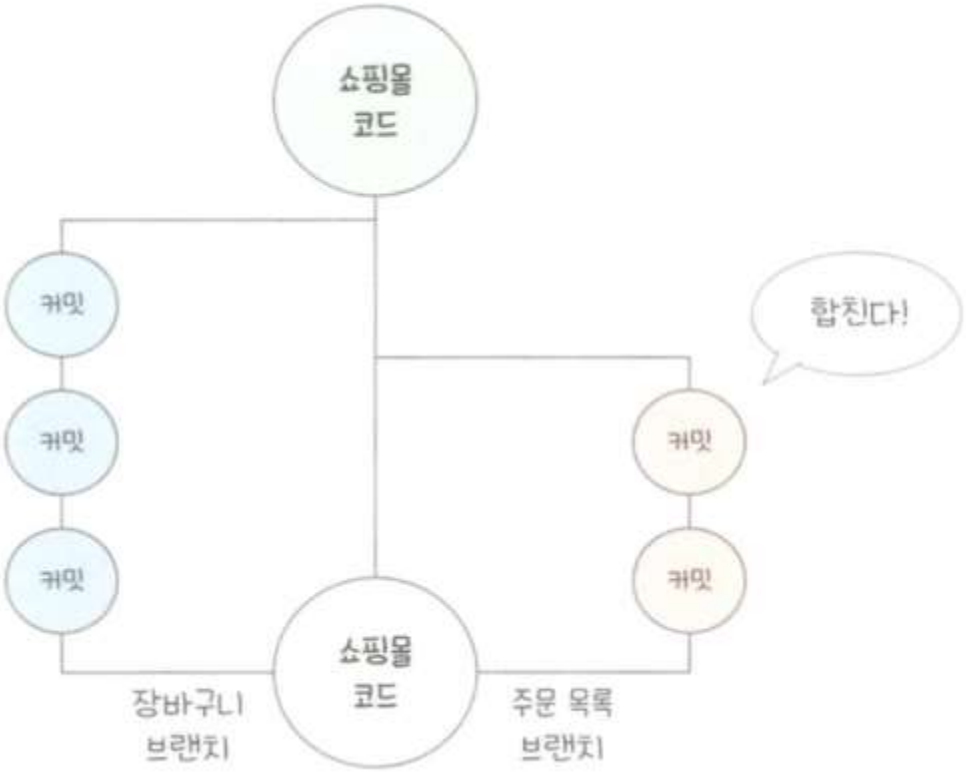


Source Tree

- ❖ 버전 관리 활용
  - ✓ Stash로 작업 임시 저장하기
    - ● 실습
      - ◆ d.txt 와 e.txt 파일 삭제 한 후 삭제한 변경 사항을 임시 저장
      - ◆ 임시저장1을 선택한 후 마우스 오른쪽을 클릭한 후 [스태시 적용]을 선택

Source Tree

- branch
  - ✓ 개요
    - 버전을 여러 흐름으로 나누어 관리하는 방법
    - branch는 버전의 분기

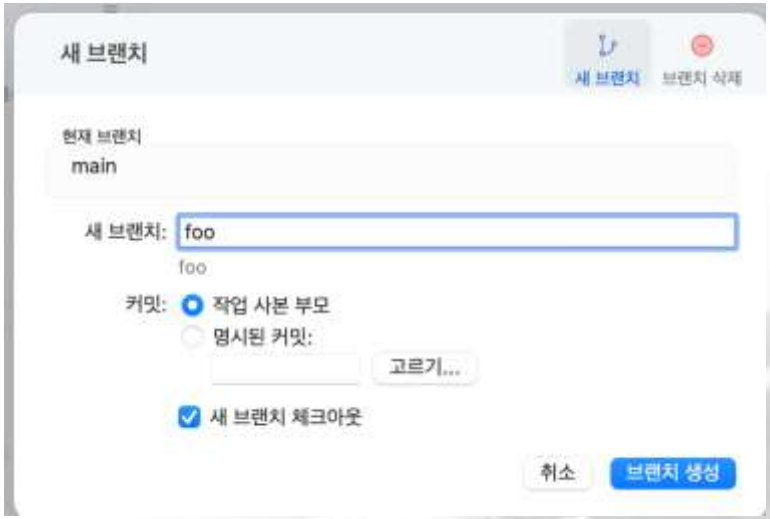


Source Tree

- ❖ branch
  - ✓ HEAD 와 Checkout
    - HEAD
      - ◆ 현재 작업 중인 브랜치의 최신 커밋을 가리키는 일종의 표시
      - ◆ 브랜치를 나누고 합치는 과정에서 HEAD의 위치를 자유자재로 바꿀 수 있음
    - Checkout
      - ◆ 체크아웃이란 특정 브랜치에서 작업할 수 있도록 작업 환경을 바꾸는 것
      - ◆ 특정 브랜치로 체크아웃하게 되면 HEAD의 위치가 해당 브랜치의 최신 커밋을 가리키고 작업 디렉토리는 체크아웃한 브랜치의 모습으로 바뀜

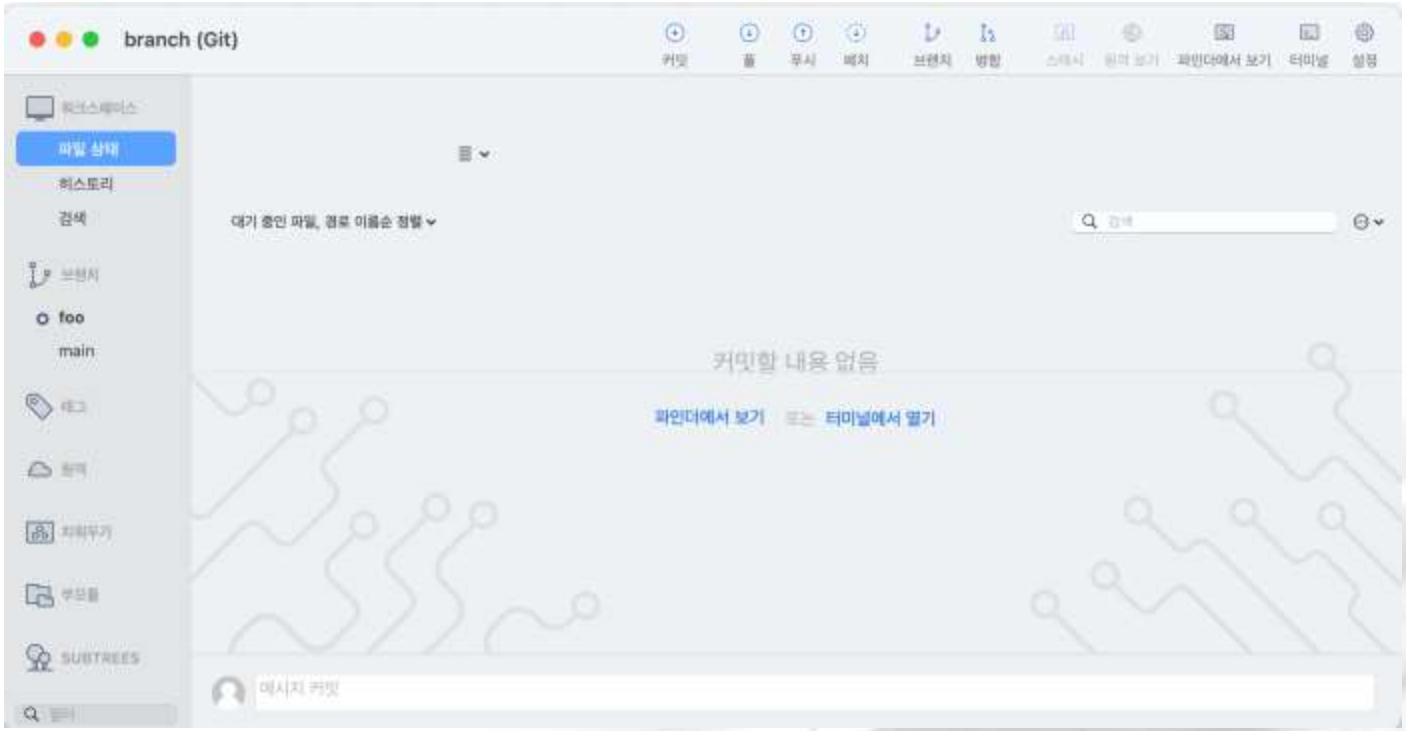
Source Tree

- ❖ branch
  - ✓ 실습
    - 로컬 저장소를 생성
    - a.txt 파일을 생성하고 A를 추가하고 커밋 메시지를 1로 해서 커밋
    - b.txt 파일을 생성하고 B를 추가하고 커밋 메시지를 2로 해서 커밋
    - c.txt 파일을 생성하고 C를 추가하고 커밋 메시지를 3로 해서 커밋
    - 새로운 브랜치 foo 생성 – 상단의 브랜치 버튼을 클릭



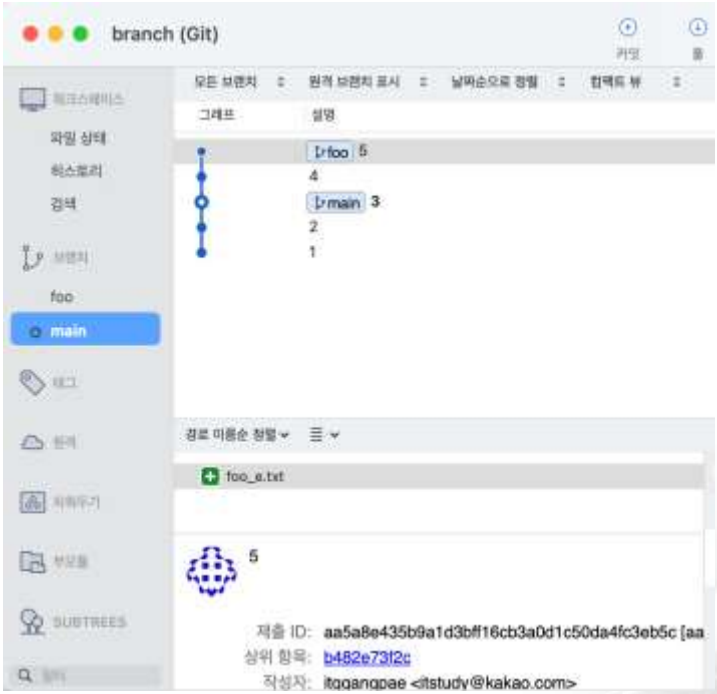
Source Tree

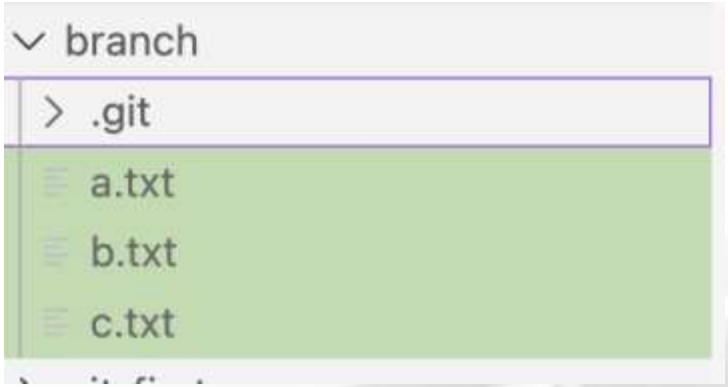
- branch
  - ✓ 실습
    - 현재 브랜치 확인



Source Tree

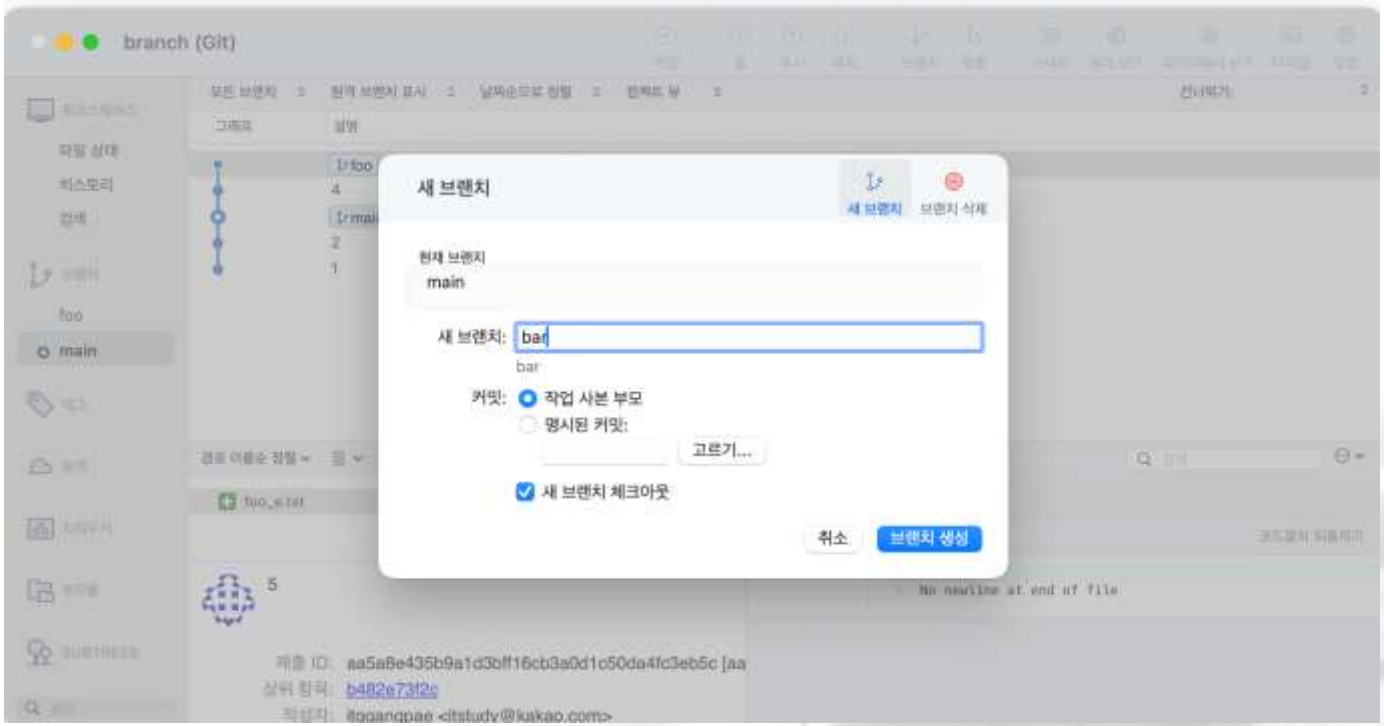
- branch
  - ✓ 실습
    - foo_d.txt 파일을 추가하고 D를 적은 후 커밋하는데 메시지는 4
    - foo_e.txt 파일을 추가하고 E를 적은 후 커밋하는데 메시지는 5
    - main 브랜치를 더블 클릭해서 main 브랜치로 체크아웃





Source Tree

- branch
  - ✓ 실습
    - main 브랜치에 체크아웃 된 상태에서 bar 라는 새로운 브랜치를 생성



Source Tree

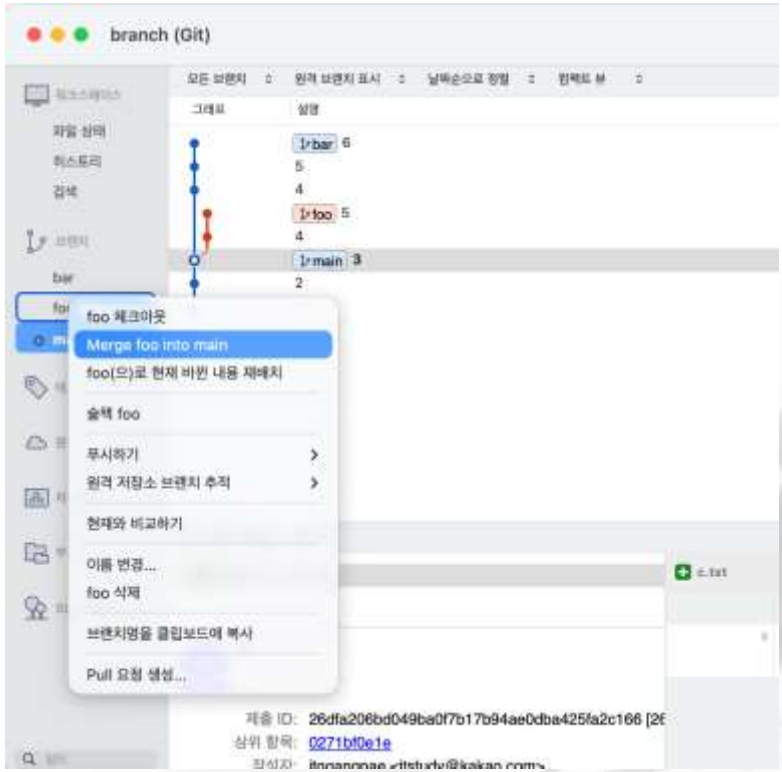
- ❖ branch
  - ✓ 실습
    - bar 브랜치에서 bar_d.txt 파일을 생성하고 D를 적은 후 커밋하고 메시지는 4
    - bar 브랜치에서 bar_e.txt 파일을 생성하고 E를 적은 후 커밋하고 메시지는 5
    - bar 브랜치에서 bar_f.txt 파일을 생성하고 F를 적은 후 커밋하고 메시지는 6

Source Tree

- branch
  - ✓ 실습
    - 현재 브랜치 상태

Source Tree

- ❖ branch
  - ✓ 실습
    - ● 브랜치 병합: 브랜치를 하나로 통합하는 것
      - ◆ main 브랜치로 Checkout
      - ◆ foo 브랜치를 선택 한 후 마우스 오른쪽 버튼을 클릭해서 main 브랜치에 병합을 선택



Source Tree

- branch
  - ✓ 실습
    - 브랜치 병합: 브랜치를 하나로 통합하는 것
      - foo 브랜치를 main 브랜치에 병합한 모습

Source Tree

- ❖ branch
  - ✓ 실습
    - ● 브랜치 병합: 브랜치를 하나로 통합하는 것
      - ◆ foo 브랜치 와 main 브랜치가 동일한 내용을 가리키므로 foo 브랜치 삭제

Source Tree

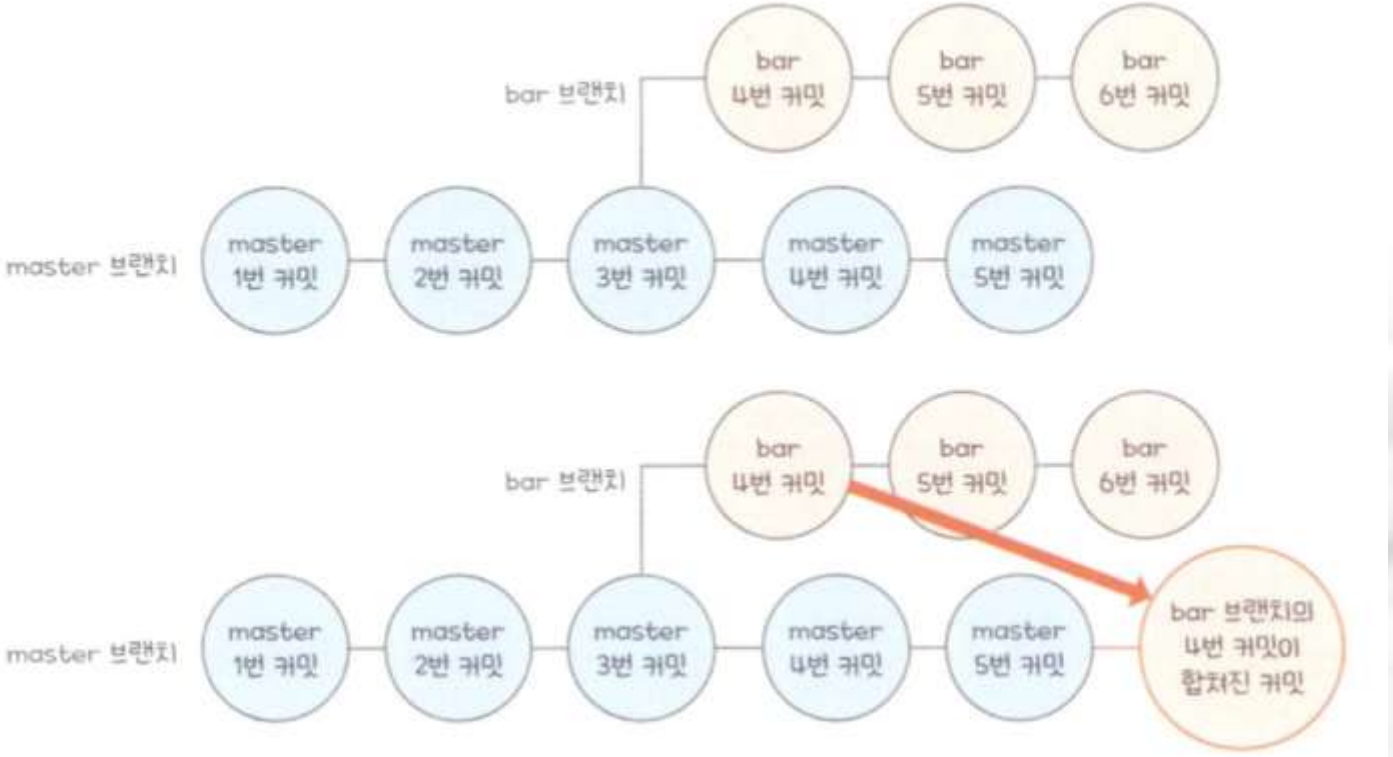
- ❖ branch
  - ✓ 실습
    - ● 브랜치 병합: 브랜치를 하나로 통합하는 것
      - ◆ fast-forward merge
        - ▪ 하나의 브랜치에서 뻗어나온 시점부터 병합되는 순간까지 원본 브랜치에 어떤 변화도 없는 상태에서의 병합
        - ▪ foo 브랜치는 main 브랜치에서 뻗어나온 이후로 여러 커밋이 쌓였지만 그동한 main 브랜치는 새로운 커밋도 없이 그저 가만히 있었는데 이런 경우는 main 브랜치로 병합할 때 foo 브랜치에 새롭게 쌓인 커밋을 반영만 하면 됨

Source Tree

- ❖ branch
  - ✓ 실습
    - ● 브랜치 병합: 브랜치를 하나로 통합하는 것
      - ◆ 특정 커밋으로 병합: main 브랜치로 체크아웃 한 후 상단의 병합을 클릭한 후 bar 브랜치를 선택하고 4번 커밋을 선택하면 새로운 커밋이 만들어 짐

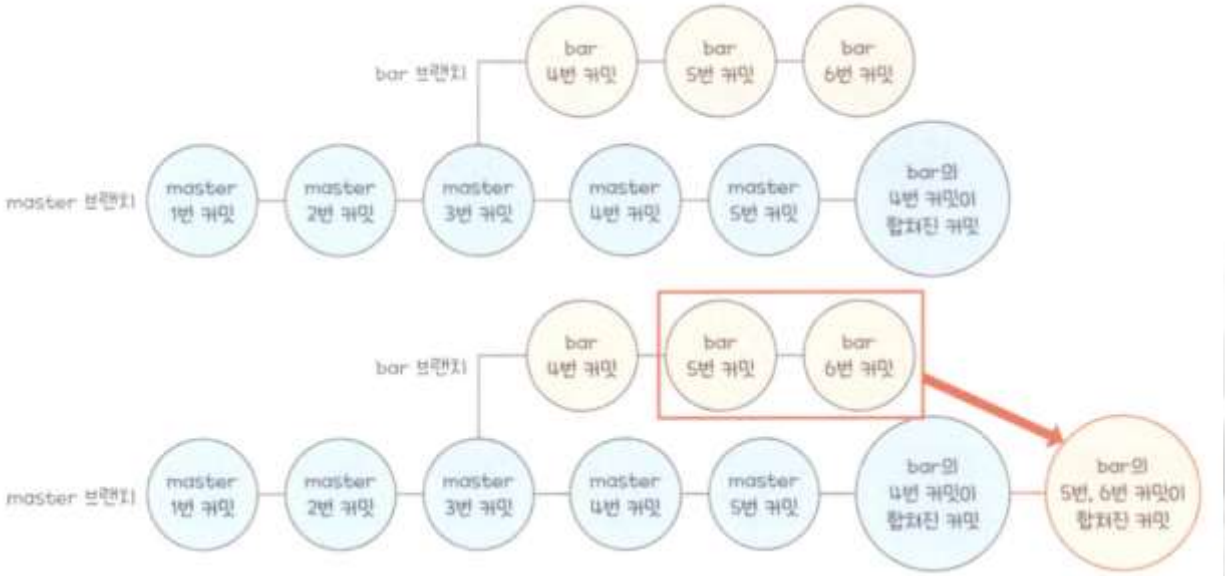
Source Tree

- ❖ branch
  - ✓ 실습
    - ● 브랜치 병합: 브랜치를 하나로 통합하는 것
      - ◆ 현재 상태



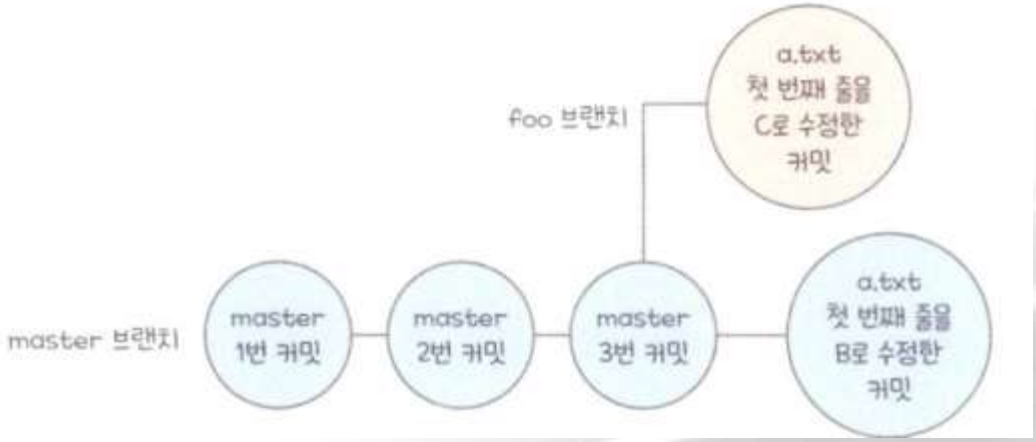
Source Tree

- branch
  - ✓ 실습
    - 브랜치 병합: 브랜치를 하나로 통합하는 것
      - bar의 6번 커밋을 병합: Merge branch bar 라는 커밋이 생성됨



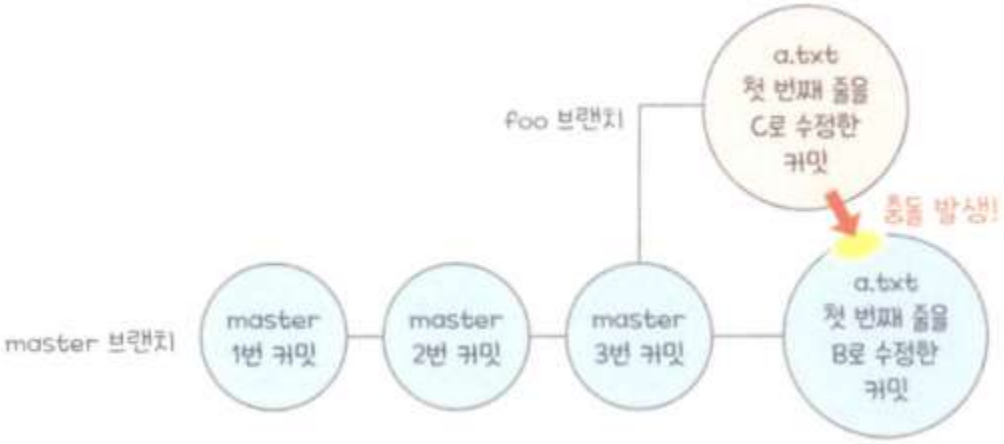
Source Tree

- branch
  - ✓ 충돌
    - 충돌이란 병합하려는 두 브랜치가 서로 같은 내용을 다르게 수정하는 상황으로 충돌이 발생하면 브랜치가 한 번에 병합되지 않음
    - 예를 들면 main 브랜치에서 foo 브랜치가 뻗어나온 경우 main 브랜치는 a.txt 파일의 첫 번째 줄을 B로 수정한 다음 커밋했고 foo 브랜치는 a.txt 파일의 첫 번째 줄을 C로 수정한 후 커밋을 한 경우



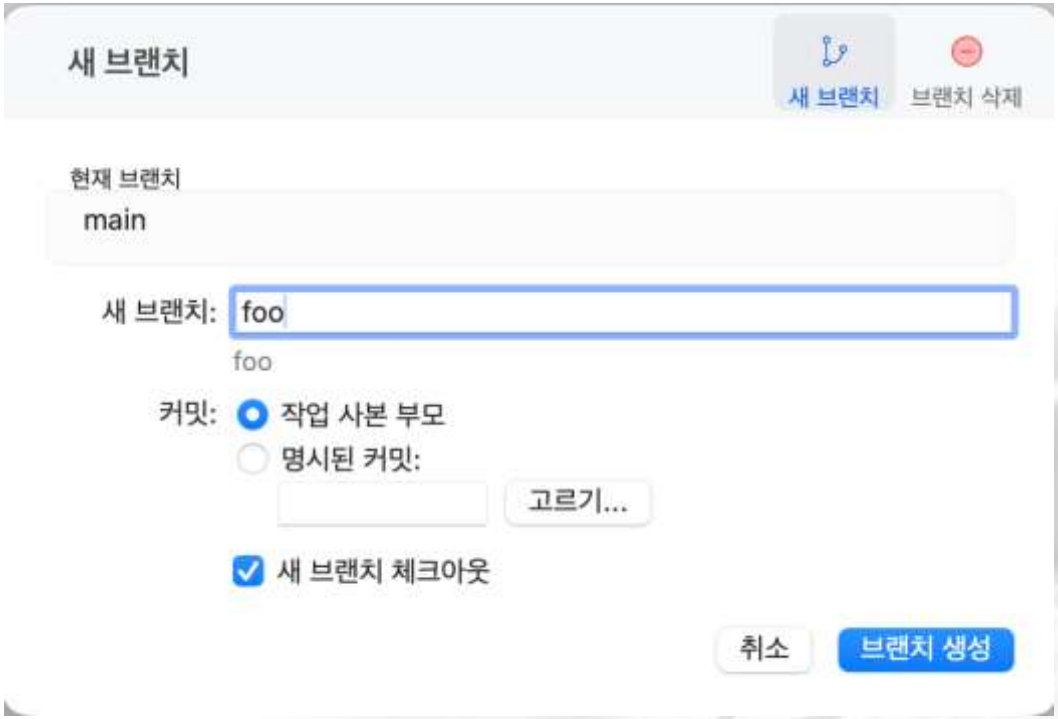
Source Tree

- branch
  - ✓ 충돌
    - 이런 경우 깃은 어떤 브랜치의 내용을 반영해야 할 지 판단할 수 없음
    - 충돌이 발생하면 최종적으로 어떤 브랜치의 내용을 반영할지 직접 선택



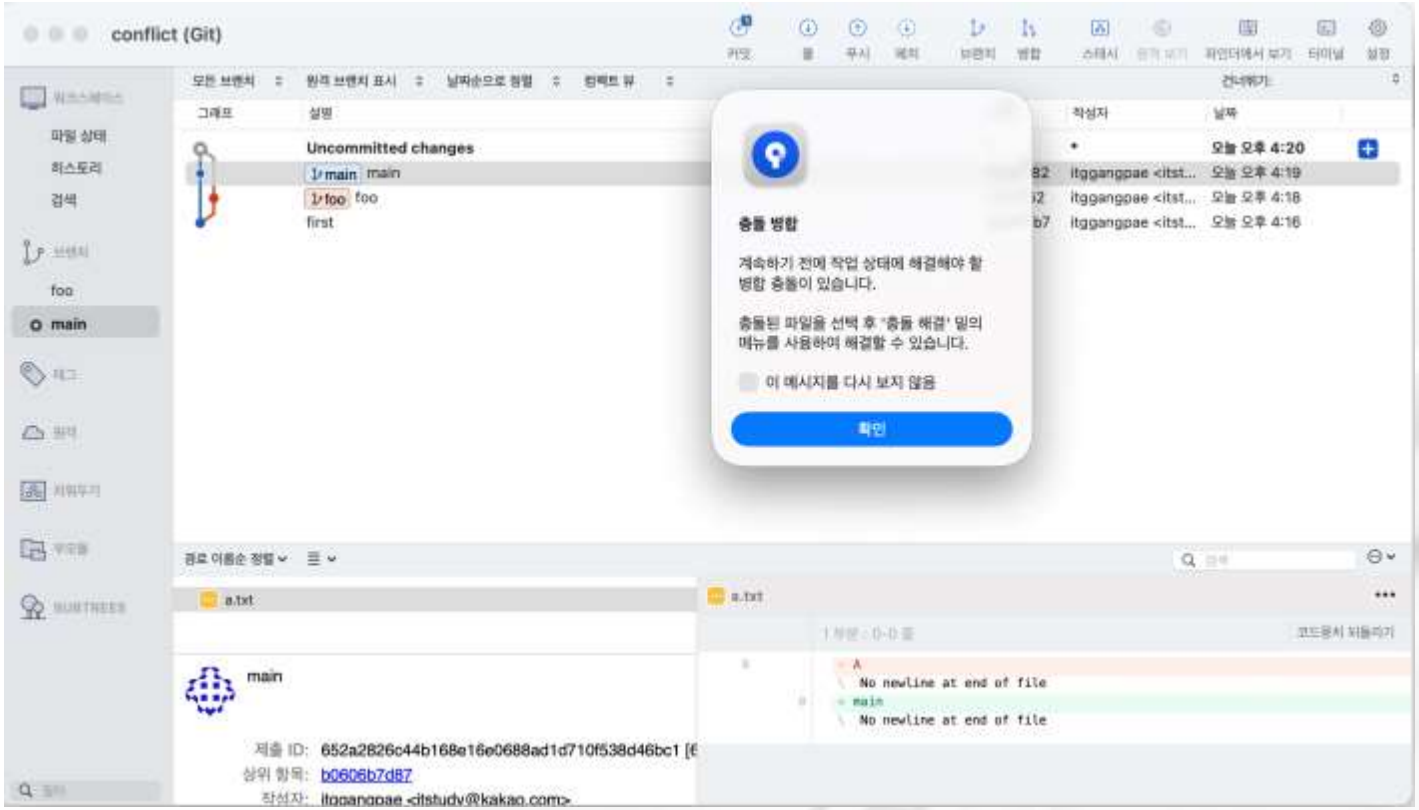
Source Tree

- ❖ branch
  - ✓ 실습
    - 로컬 저장소 생성
    - main 브랜치에 A가 저장된 a.txt 파일을 만들고 커밋(메시지는 first)
    - main 브랜치를 이용해서 foo 브랜치를 생성하고 체크아웃 한 후 a.txt 파일의 내용을 foo로 변경한 뒤 커밋(메시지는 foo)



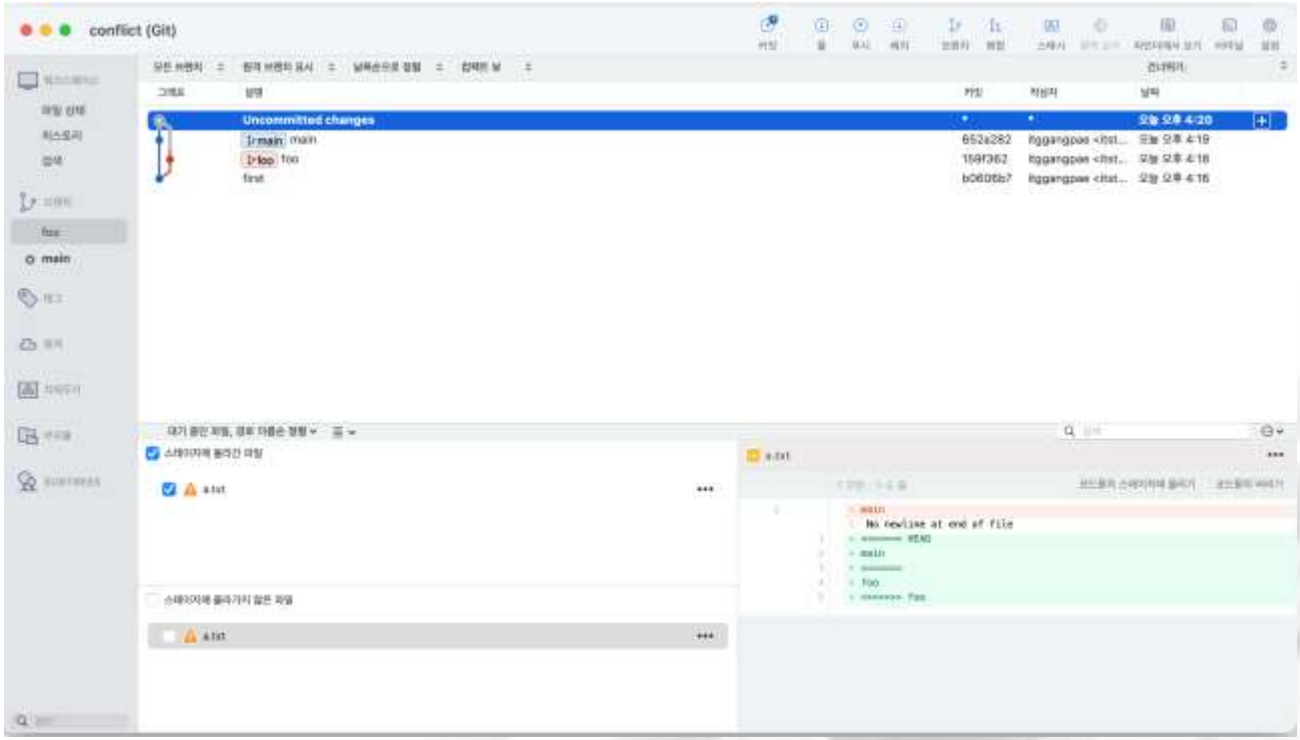
Source Tree

- branch
  - ✓ 실습
    - main 브랜치로 체크아웃해서 a.txt 파일의 내용을 main로 변경한 뒤 커밋(메시지는 main)
    - 현재 상태에서 foo 브랜치를 main에 병합



Source Tree

- branch
  - ✓ 실습
    - 충돌이 발생하면 커밋하지 않은 변경사항이 생기고 스테이지에 올라가지 않은 파일 과 스테이지에 올라간 파일 항목에는 충돌이 발생한 파일이 추가됨



Source Tree

- branch
  - ✓ 충돌 해결
    - 충돌이 발생하면 충돌이 발생한 두 브랜치 중 어떤 브랜치의 내용을 병합 결과에 반영할지를 여러분이 직접 선택해야 함
    - 같은 내용을 다르게 수정한 브랜치 중 어떤 브랜치 내용을 최종적으로 반영할 지를 직접 선택하는 것을 충돌 해결이라 함
    - 스테이지에 올라가지 않은 파일의 내용

      +<<<<<<< HEAD
      +main
      +=======
      +foo
      +>>>>>>> foo

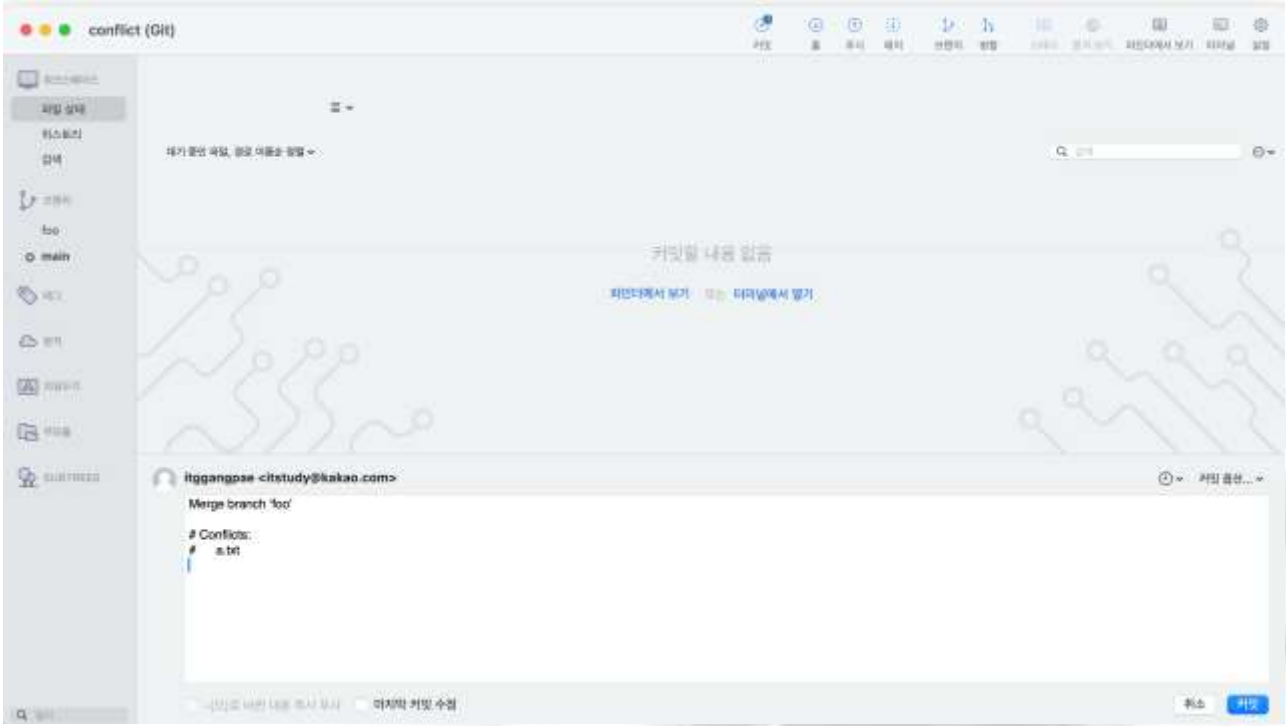
      - 충돌이 발생한 파일에는 <<<<<<, >>>>>>, ======= 기호가 표기되는데 이 기호는 일종의 영역 표기
      - ======= 기호를 기준으로 윗부분은 HEAD가 가리키는 브랜치로 현재 체크아웃한 브랜치의 내용이 적혀 있고 아랫부분은 병합하려는 브랜치로 foo 브랜치의 내용이 적혀있음
      - <<<<<< 기호와 ====== 기호 사이의 내용을 선택할지 ====== 기호와 >>>>>> 기호 사이의 내용을 선택할지 고르라는 표기
      - 두 영역 중 반영할 부분을 직접 선택해 충돌을 해결해야 함

Source Tree

- branch
  - ✓ 충돌 해결
    - 스테이지에 올라가지 않은 파일을 선택한 후 마우스 오른쪽을 클릭한 후 [충돌 해결] 메뉴에 보면 [ResolveUsingMineBaseFormat – 내것을 이용해서 해결] 과 [ResolveUsingTheirsBaseFormat – 저장소것을 이용해서 해결]이 보임
    - 내것을 이용해서 해결을 선택
    - 충돌이 발생한 파일을 담고 있던 커밋하지 않은 변경사항이 사라짐

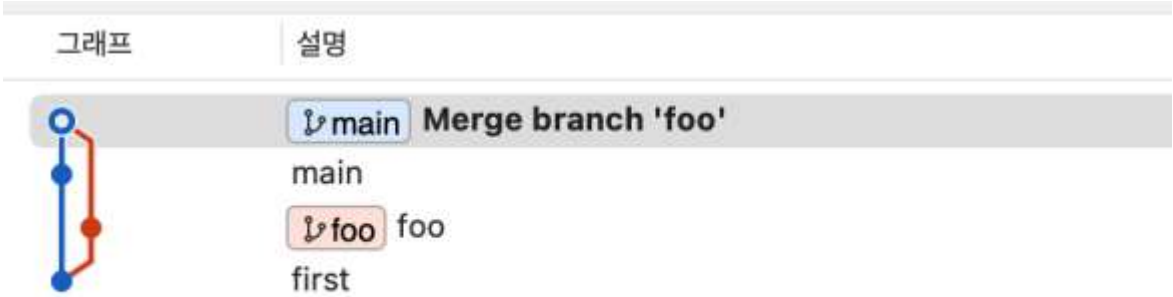
Source Tree

- branch
  - ✓ 충돌 해결
    - 브랜치 병합을 위해서는 커밋을 작성해야 함



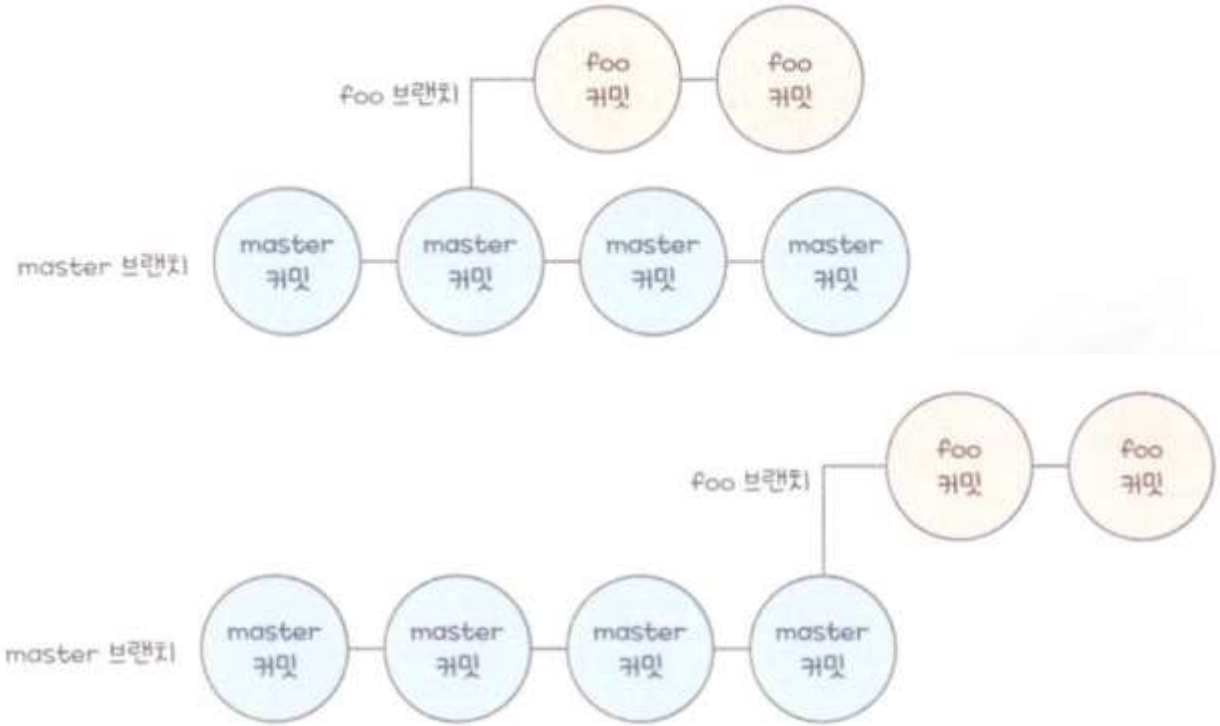
Source Tree

- ❖ branch
  - ✓ 충돌 해결
    - ● 결과



Source Tree

- branch
  - ✓ 브랜치 재배치(rebase)
    - 브랜치가 뻗어나온 기준점을 변경하는 것이 rebase

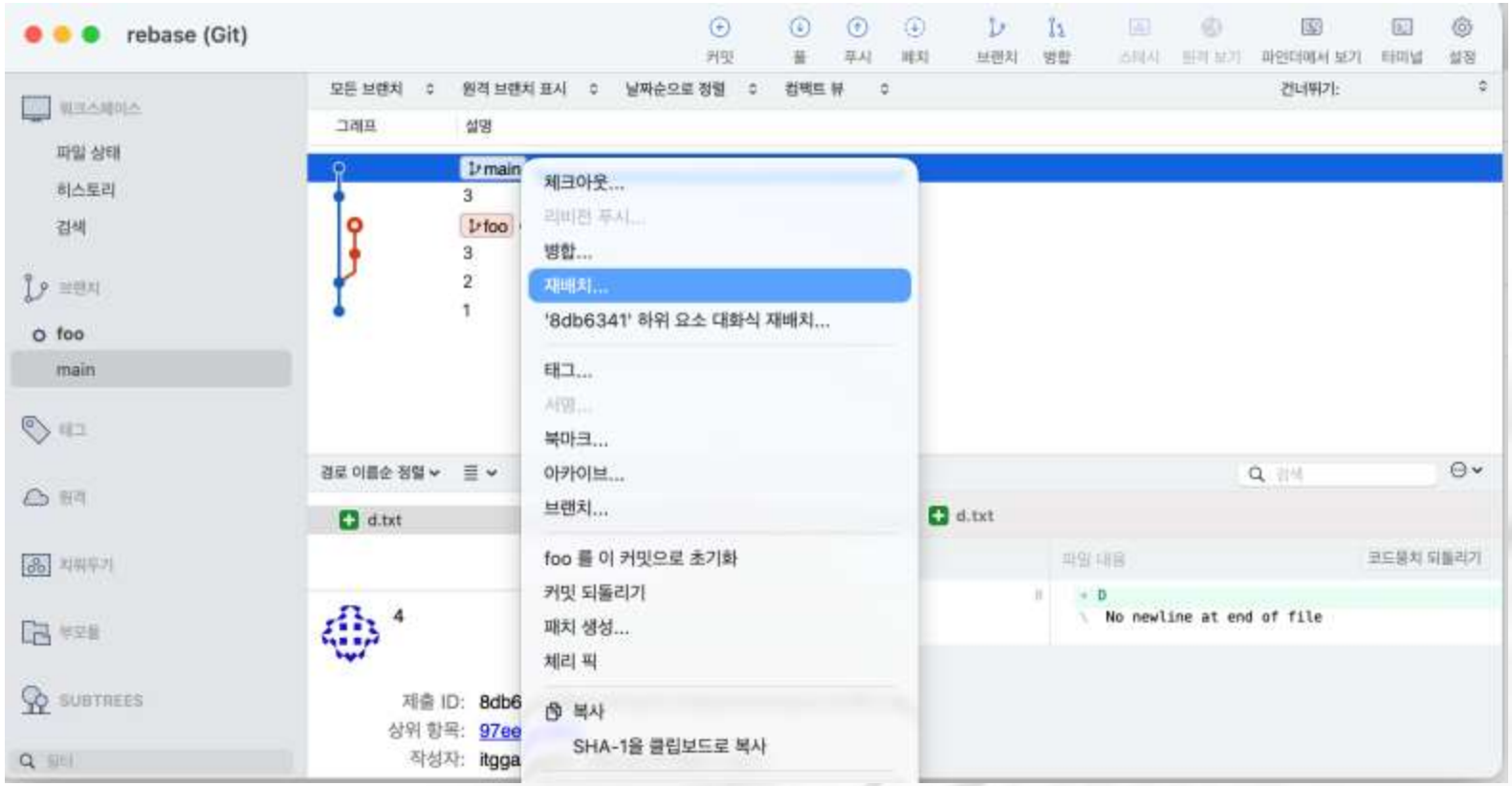


Source Tree

- branch
  - ✓ 브랜치 재배치(rebase)
    - 로컬 저장소 생성
    - main 브랜치에 A가 저장된 a.txt 파일을 만들고 커밋(1)
    - main 브랜치에 B가 저장된 b.txt 파일을 만들고 커밋(2)
    - 현재 상태를 가지고 foo 라는 브랜치를 생성
    - foo 브랜치에서 C가 저장된 foo_c.txt 파일을 만들고 커밋(3)
    - foo 브랜치에서 D가 저장된 foo_d.txt 파일을 만들고 커밋(4)
    - main 브랜치에 C가 저장된 c.txt 파일을 만들고 커밋(3)
    - main 브랜치에 D가 저장된 d.txt 파일을 만들고 커밋(4)

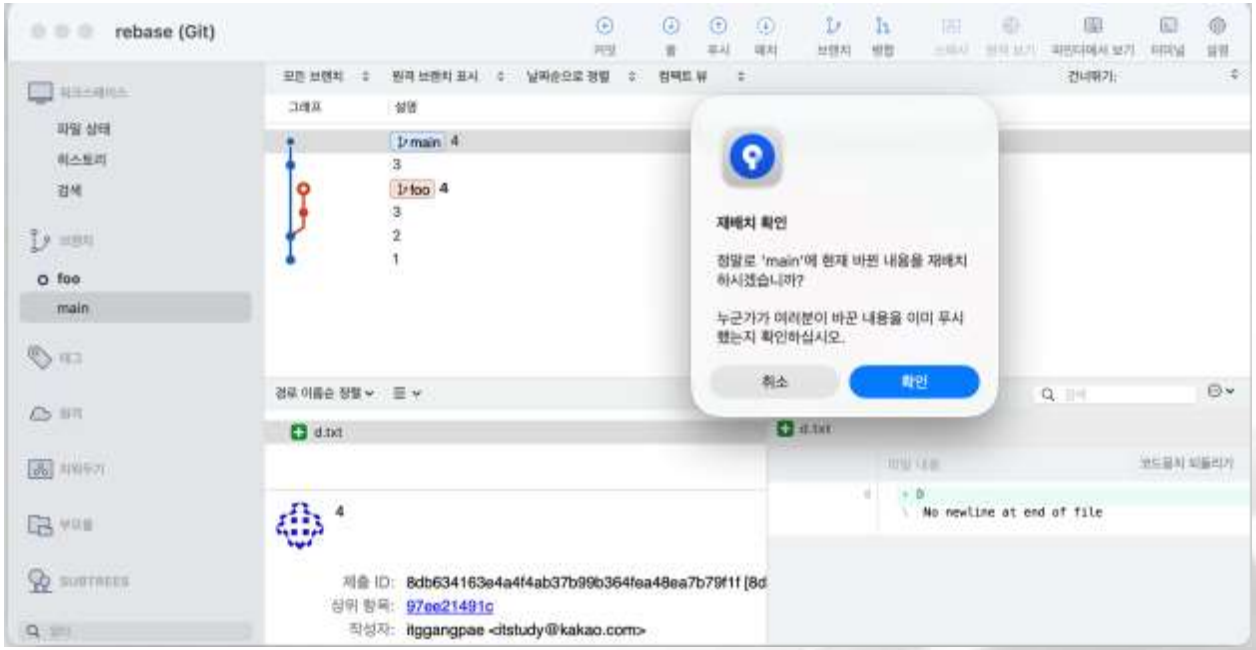
Source Tree

- branch
  - ✓ 브랜치 재배치(rebase)
    - foo 브랜치로 체크아웃
    - 재배치하고자 하는 커밋에서 마우스 오른쪽을 클릭해서 [재배치]를 클릭



Source Tree

- branch
  - ✓ 브랜치 재배치(rebase)
    - 재배치하고자 하는 커밋에서 마우스 오른쪽을 클릭해서 [재배치]를 클릭



Source Tree

- branch
  - ✓ 브랜치 재배치(rebase)
    - 재배치하고자 하는 커밋에서 마우스 오른쪽을 클릭해서 [재배치]를 클릭