GIT 세부 설명

Subject	GIT
index	5
날짜	@2024/01/03
배운 내용 요약	GIT을 과거로 돌리는 법과 브랜치 생성 및 삭제 방법, 브랜치 합병 후 충돌 해결
상태	완료

Git에서 과거로 돌아가는 두 방식

- reset : 원하는 시점으로 돌아가고 원하는 시점 이후의 내역들을 전부 지움
- revert : 되돌리기를 원하는 시점의 커밋을 거꾸로 실행함(해당 커밋이 수행한 작업을 반대로 하는 커밋을 생성함)

두 가지 방법 모두 Sourcetree에서 GUI로 하게 되면 쉽게 됨

그렇기에 여기선 터미널로 하는 방법을 위주로 정리하고 소스트리로 하는 법은 교안을 그대로 가져옴

reset 사용해서 과거로 돌아가기

1. git log 명령어로 커밋 내역을 확인함
2. 되돌아가길 시점에 해당하는 커밋의 커밋 해시를 복사
3. :q로 git log에서 나옴
4. git reset —hard (돌아갈 커밋 해시) 수행해서 과거로 돌아감

- reset 명령어 : git reset —hard (돌아갈 커밋 해시)

revert 로 과거의 커밋 되돌리기

1. git log 명령어로 커밋 내역을 확인함
2. 취소하길 원하는 커밋에 해당하는 커밋의 커밋 해시를 복사
3. :q로 git log에서 나옴
4. git revert (되돌릴 커밋 해시) 수행해서 과거로 돌아감

- revert 명령어 : git revert (되돌릴 커밋 해시)
- 만약 커밋하지 않고 revert 하려고 하면 다음과 같이 사용
  - git revert —no-commit (되돌릴 커밋 해시)

SourceTree로 git reset, revert 실행해보기

1. 변경사항 만들고 커밋하기

1. `A.txt` 추가

```
a
```

- 커밋 메시지: `commit one`

2. `A.txt` 수정

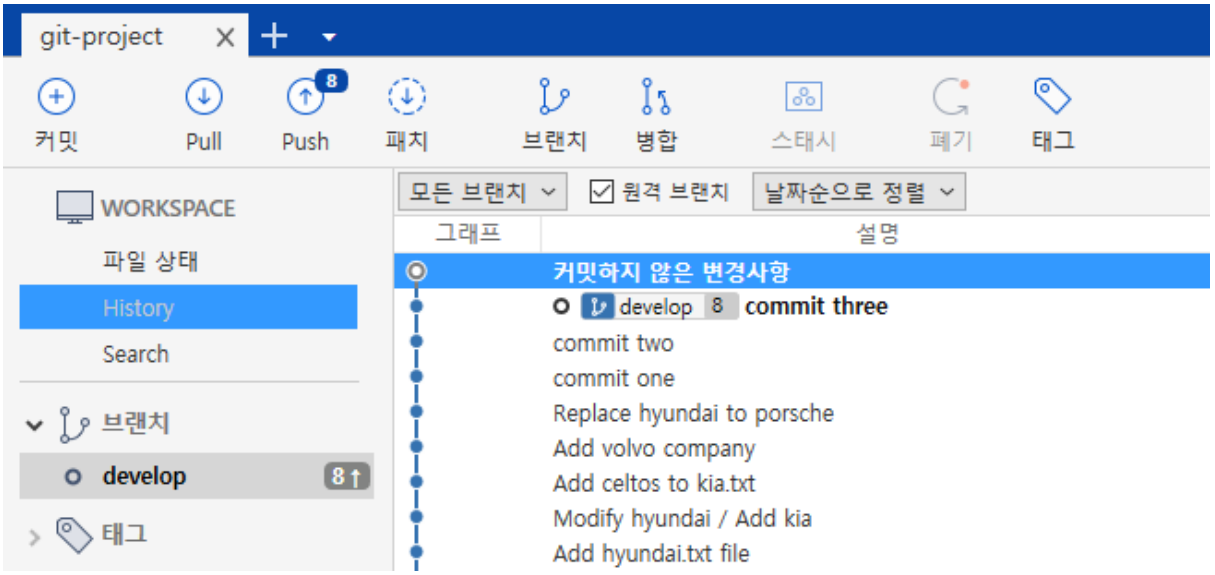
```
ab
```

- 커밋 메시지: `commit two`

3. `A.txt` 수정

```
abc
```

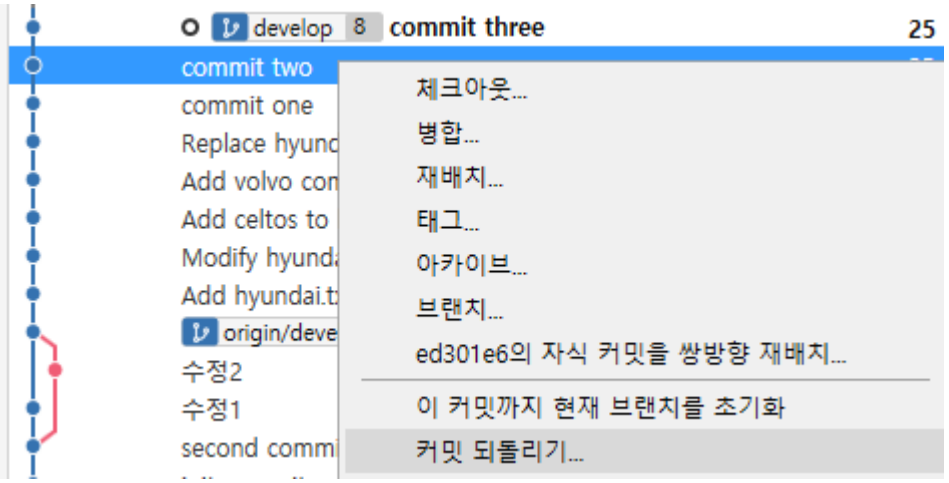
- 커밋 메시지: `commit three`

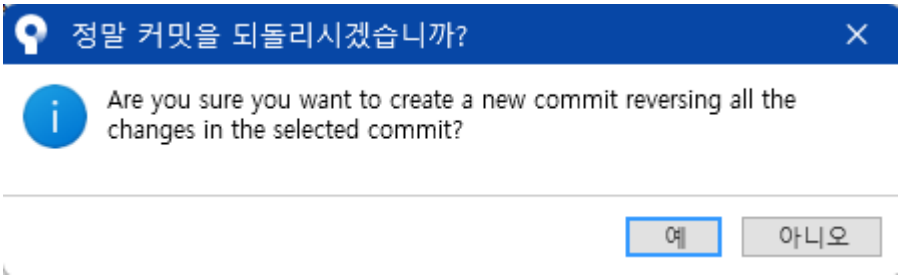


2. revert

- `commit two`의 수정사항 되돌려보기

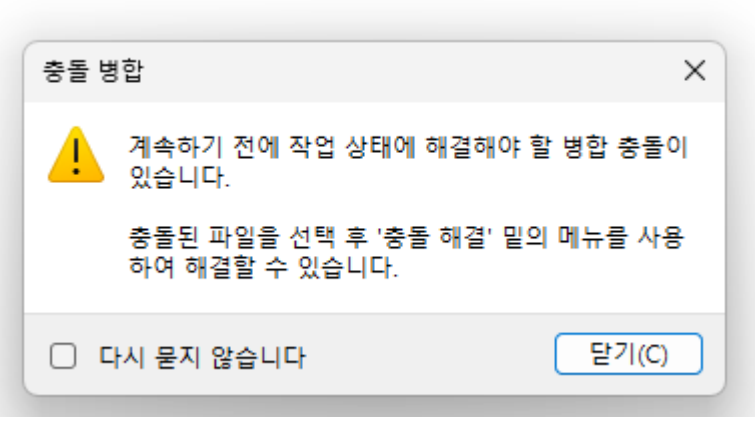
1. 해당 커밋에 마우스 우클릭 - `커밋 되돌리기`



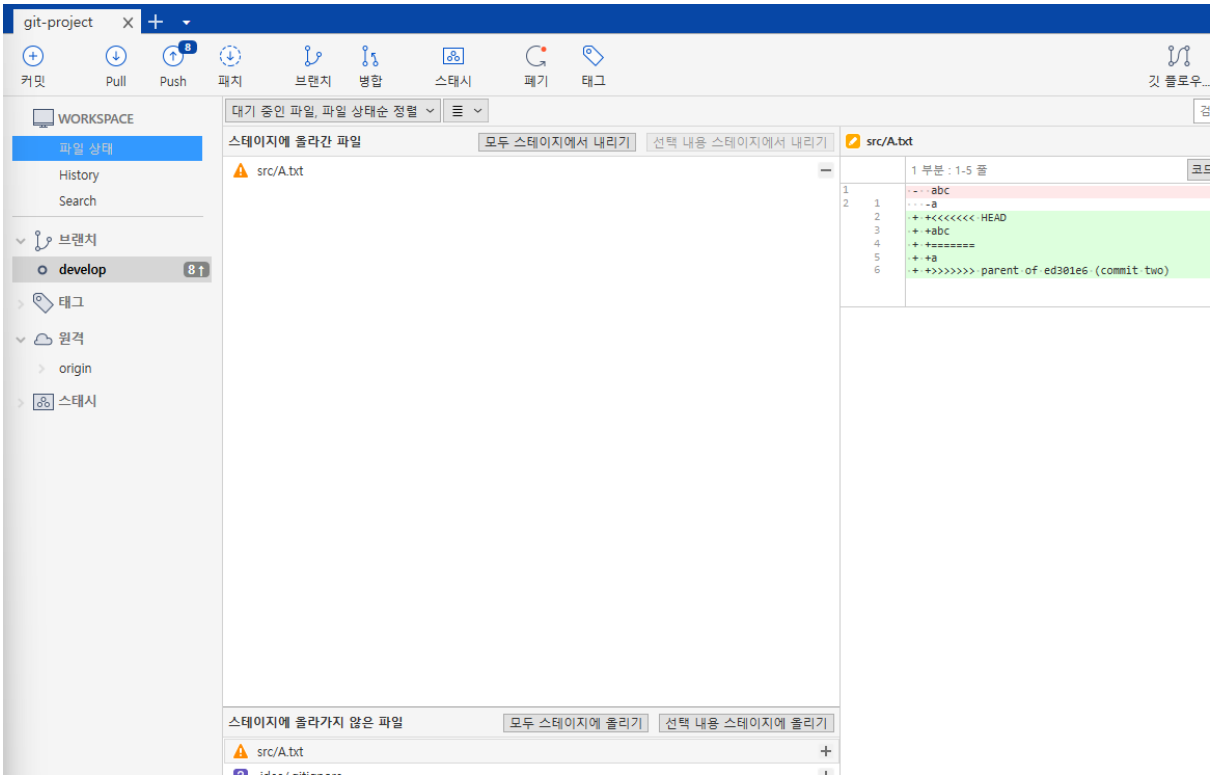


⭐Revert시 충돌 해결

그럼 다음과 같이 충돌이 일어남을 알리는 팝업창이 뜹니다.



A파일에는 다음과 같은 충돌 상황 메시지가 출력됩니다.

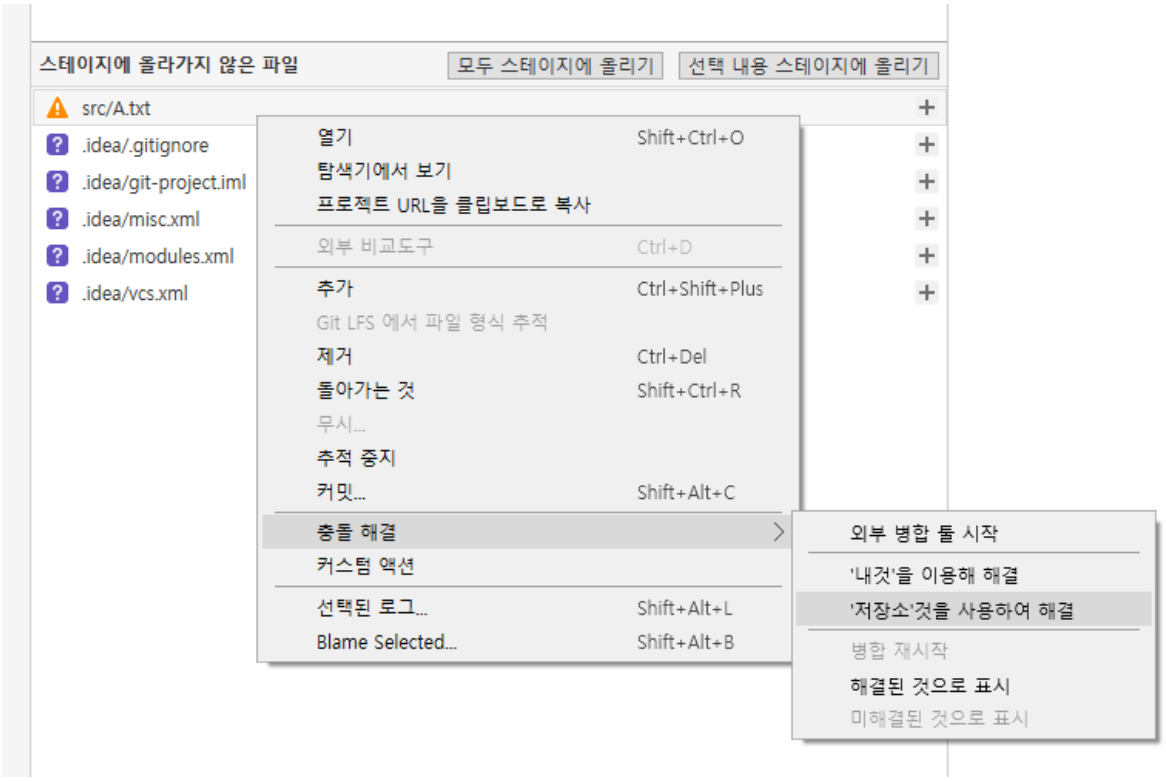


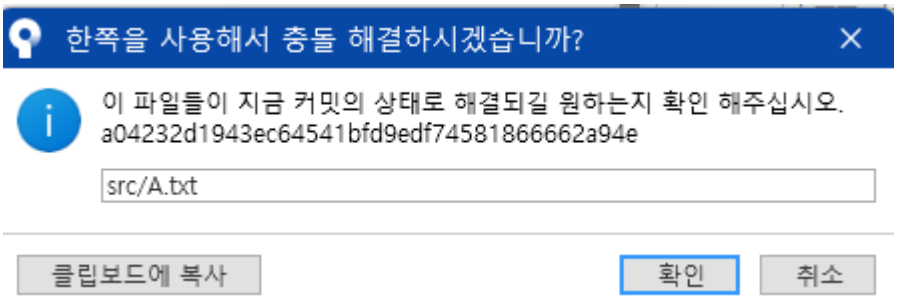

```
<<<<<<< HEAD
abc
=======
a
>>>>>>> parent of ed301e6 (commit two)
```

<<< HEAD 부터 === 까지는 현재 저장소에 작성되어있는 내용을 보여주고

=== 부터 >>> parent 까지는 revert 대상 브랜치에 작성되어 있는 내용을 보여줍니다.

2. 스테이지에 올라가지 않은 파일 src/A.txt를 우클릭 하고 `충돌해결 > '저장소'것을 사용하여 해결` 을 선택합니다.





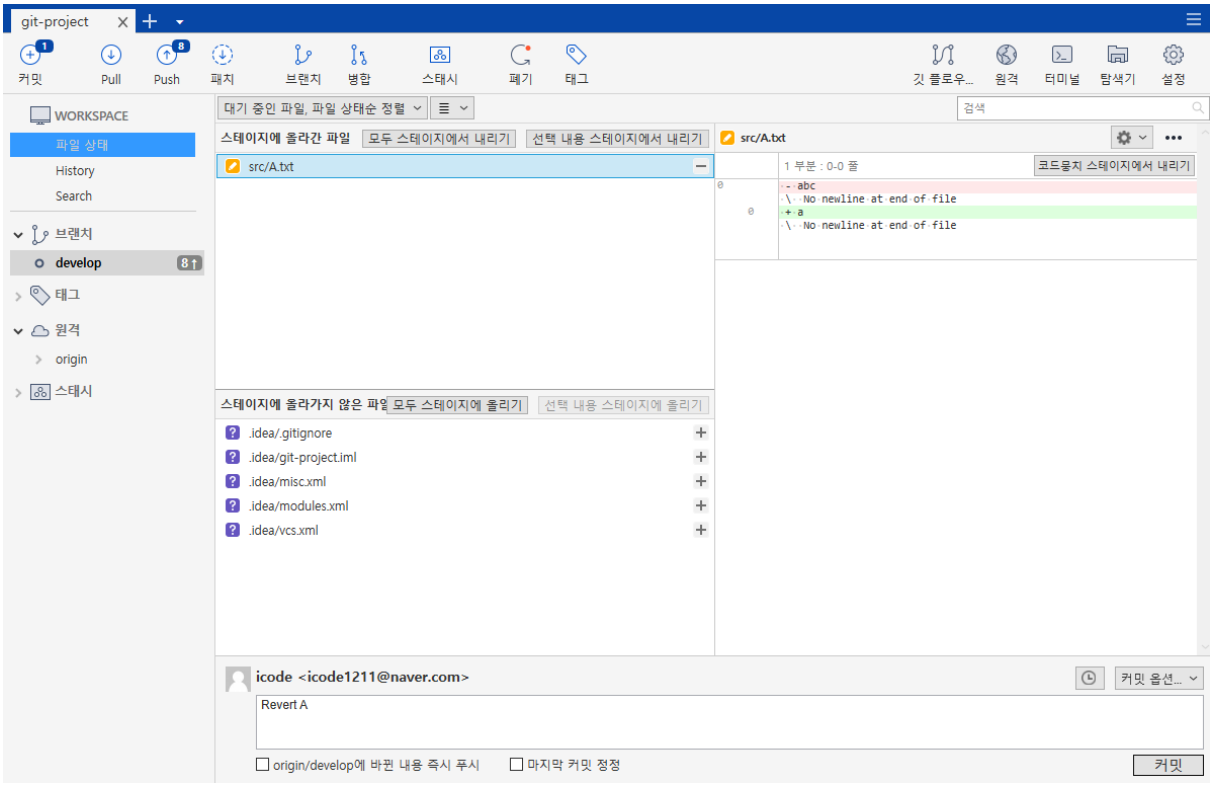
단, '저장소'것을 사용하지 않고, 직접 수정할경우 src/A.txt 파일을 직접 열어서 수정합니다.

원하는 내용으로 수정한 후 **SourceTree에서 스테이지에 올립니다.** (아래 명령어와 동일)

> git add A.txt

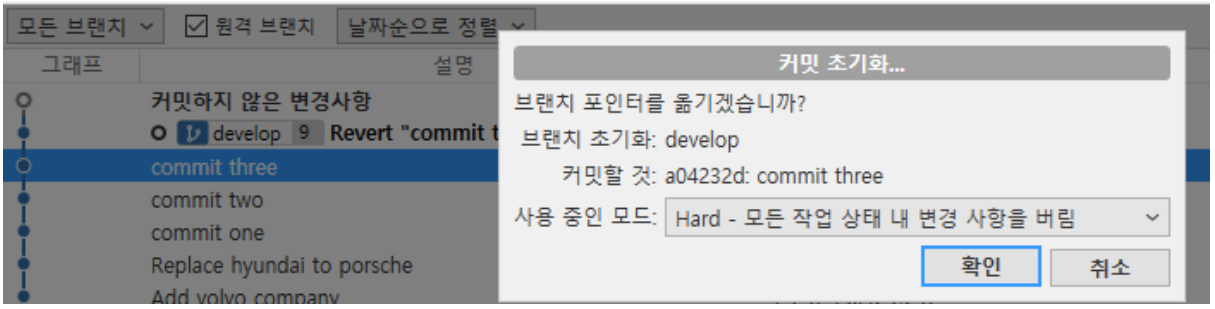
> git revert --continue

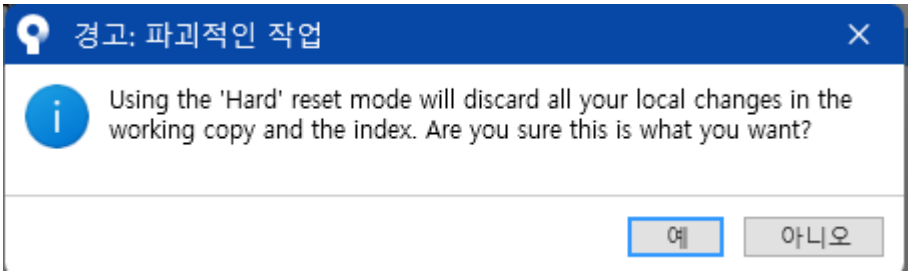
3. 커밋 메시지를 입력하고 커밋을 하면 완료됩니다.

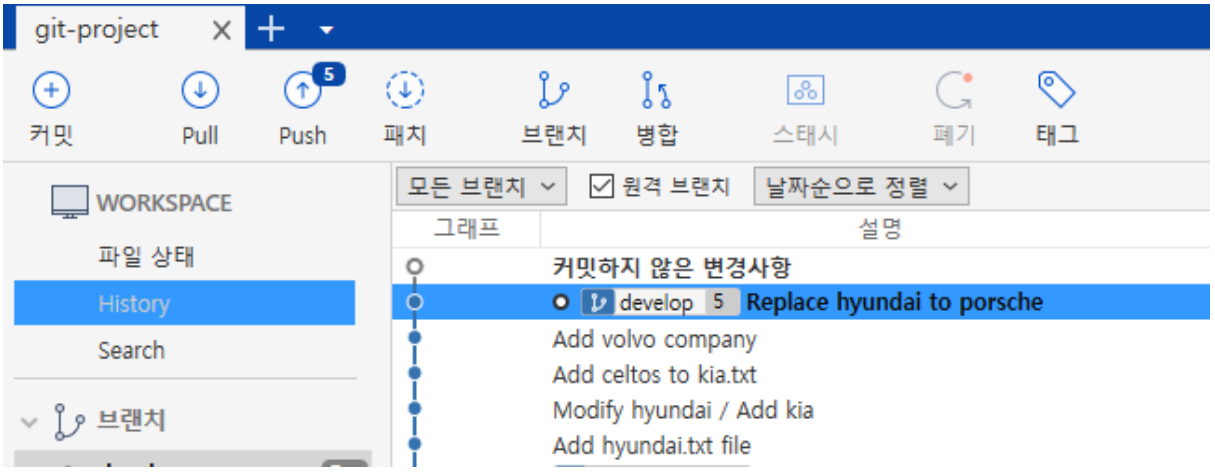


3. reset

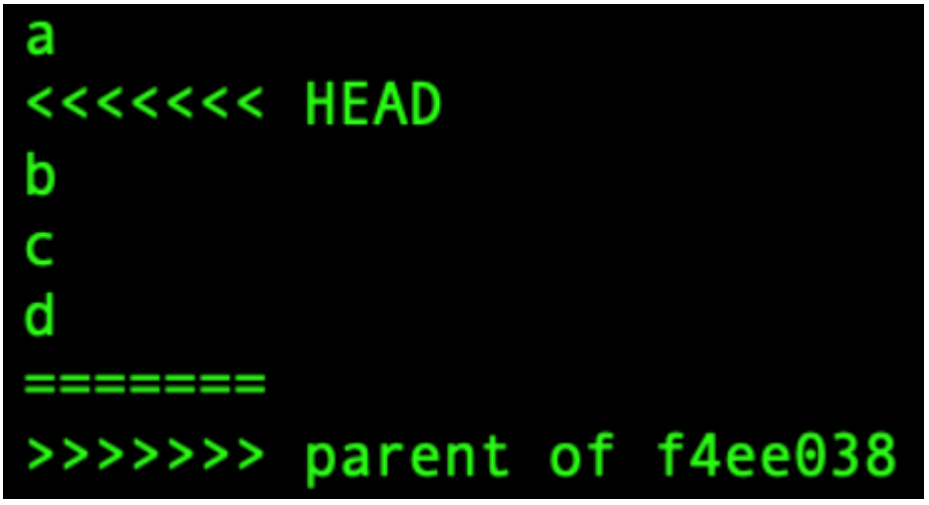
- `Replace hyundai to porsche` 시점으로 되돌려보기
- 해당 커밋에 마우스 우클릭 - `... 이 커밋까지 현재 브랜치를 초기화`
- 선택지에서 `Hard` 선택







(번외) 충돌 파일 메시지 해석하기



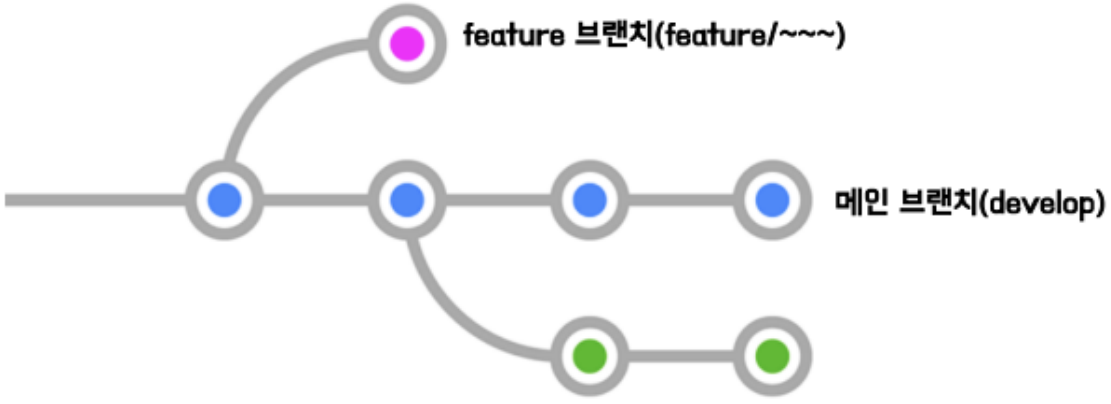
<< HEAD == >> parent of f4ee038 …

<< HEAD 부터 == 까지는 현재 저장소에 작성되어 있는 소스코드를 의미,

== 부터 >> parent of xxxx까지는 merge 대상인 브랜치에 작성되어 있는 소스코드를 의미

브랜치를 쓰는 이유

- Branch는 분기된 가지임
- 프로젝트를 하나 이상의 모습으로 관리해야 할 필요가 있을 때 사용
- 여러 작업들이 각각 독립되어 진행될 때 사용



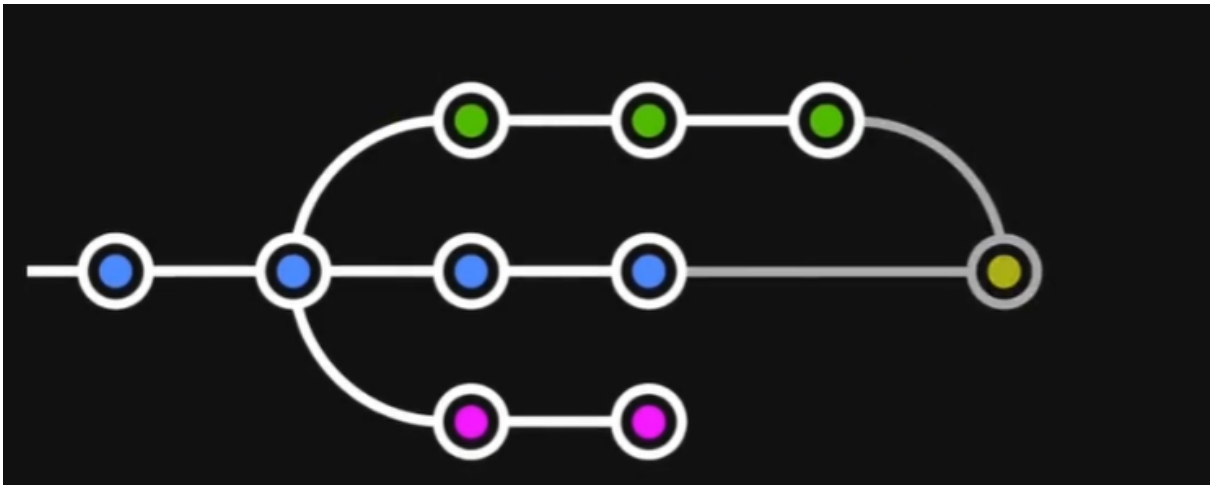
브랜치 관련 명령어

- git branch 브랜치이름 : 브랜치 생성 명령어
- git branch : 브랜치 목록 확인 명령어
- git switch 브랜치이름 : 해당하는 브랜치로 이동하는 명령어
- git switch -c 브랜치이름 : 브랜치 생성과 동시에 이동하는 명령어
- git branch -d (삭제할 브랜치명) : 브랜치 삭제
- git branch -D (강제 삭제할 브랜치명) : 브랜치 강제 삭제
    - 지우려고 하는 브랜치에 커밋이 있을 경우 지워지지 않기에 강제 삭제 해줘야함
- git branch -m (기존 브랜치명) (새 브랜치 명) : 브랜치 이름 바꾸는 명령어

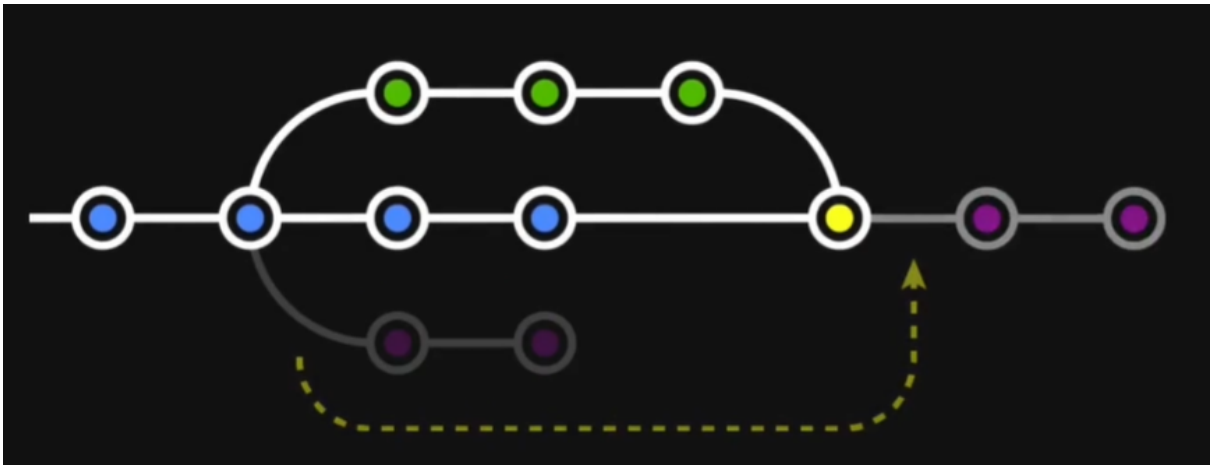
- git merge 브랜치이름 : merge로 브랜치를 합치는 명령어
- git rebase 브랜치이름 : rebase로 브랜치를 합치는 명령어

브랜치를 합치는 2가지 방법

- merge
  - 두 브랜치를 한 커밋에 이어붙임
  - 브랜치 사용 내역을 남길 필요가 있을 때 적합한 방식
  - 붙임 당하는 브랜치에서 수행함

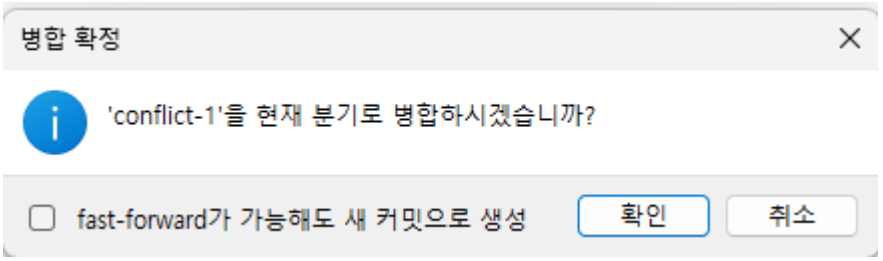


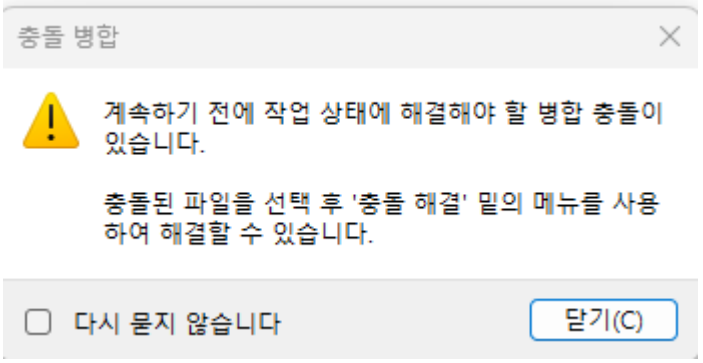
- rebase
  - 브랜치를 다른 브랜치에 이어붙임
  - 한 줄로 깔끔히 정리된 내역을 유지하기 원할 때 적합함
  - 이미 팀원과 공유된 커밋들에 대해서는 사용하지 않는 것이 좋음
  - 붙임 당하는 브랜치에서 수행함



브랜치 간의 충돌을 해결하는 법

파일의 같은 위치를 서로 다른 브랜치에서 서로 다른 내용을 입력 및 수정하여 충돌할 수 있음





이러한 충돌을 해결 하기 위해 소스 트리에서 충돌 해결 밑의 메뉴를 사용해서 원하는 것을 선택해서 충돌을 해결 할 수 있음

Push

- 커밋한 것을 GitHub로 밀어 올리는 동작
- push 명령어 : git push(GUI로 하는 것이 더 쉬움)
- push가 끝나면 GitHub에서 push한 것이 올라간 것을 확인할 수 있음

Pull

- GitHub에서 수정된 내용이 있을 때 그것을 가져오는 동작
- pull 명령어 : git pull(GUI로 하는 것이 더 쉬움)
- pull이 끝나면 로컬 저장소에서 수정된 내용이 반영된 것을 확인할 수 있음