Git rebase를 이용한 커밋 수정 (Interactive Rebase)

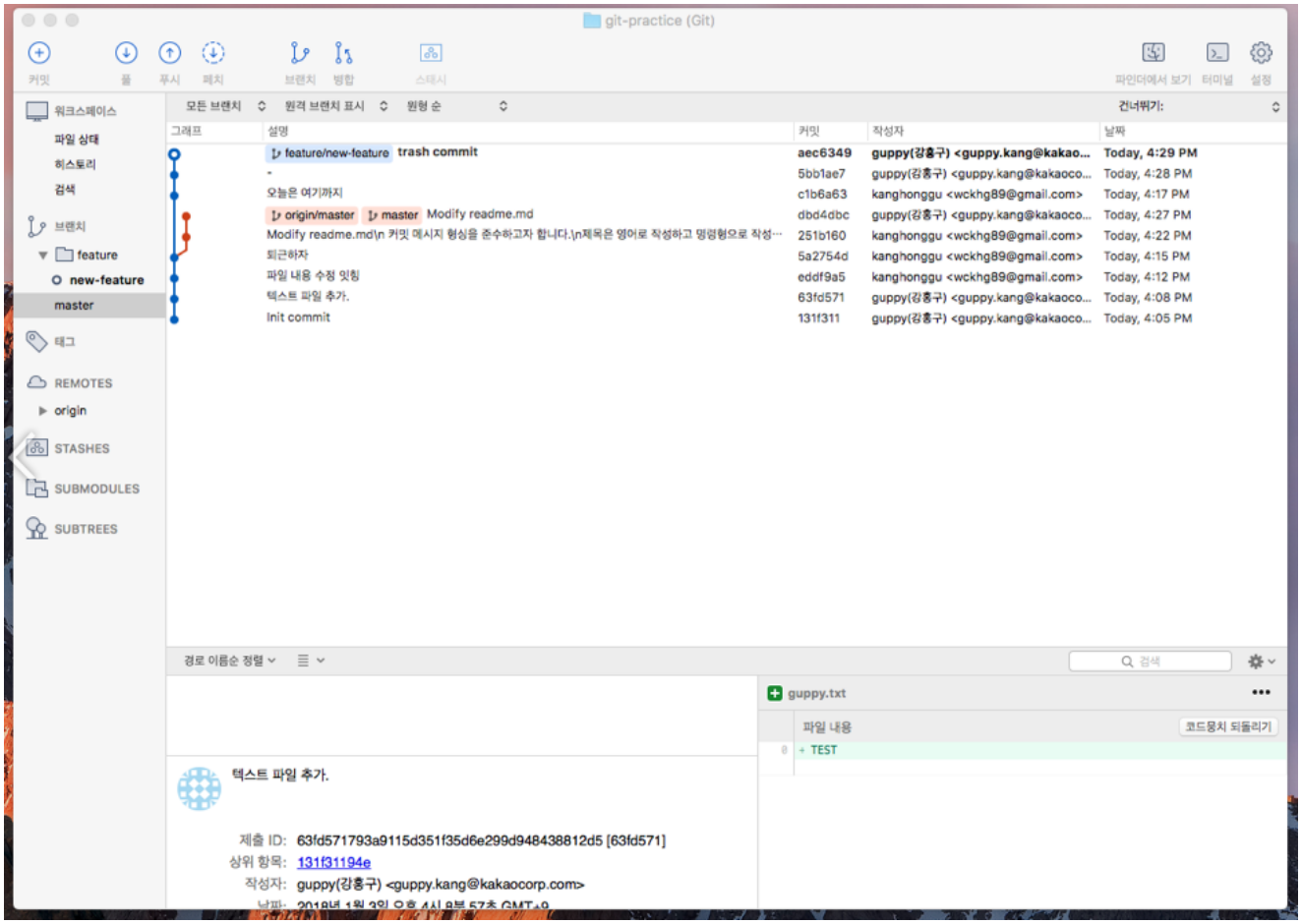
Jan 4, 2018

오늘 회사에서 기술 공유시간을 가지며 `git rebase` 의 강력한 기능에 대해 강의를 들었습니다. `rebase` 를 통해서 깃 커밋로그를 보다 깔끔하게(커밋 합치기, 커밋 지우기, 커밋 메시지 수정하기) 만들 수 있는 것이 정말 강력한 기능이라는 생각을 했습니다. 듣기만하고 '와 좋구나~'하면 결국 언젠가는 다 까먹을것 같아 간단한 깃 연습용 샘플 리파지토리를 만들어서 바로 연습을 해보았습니다.

본론

이번에 소개해드릴 내용은 `git interactive rebase` 기능을 사용하여 커밋 합치기(`Squash`), 지우기 (`drop`), 수정하기 (`reword`)를 사용해보려 합니다. CLI를 사용하여 커밋을 고쳐보고, 많이들 사용하고 계실 SourceTree를 이용해서도 커밋을 고쳐보도록 하겠습니다.

시뮬레이션을 위해서 아래와 같은 브랜치 상황을 만들어 두었습니다.



커밋 메시지를 수정하고 불필요한 커밋은 없애 버리기 위해 의도적으로 **나쁜 커밋 로그** 를 남겨두었습니다. (좋은 커밋 로그 남기기에 대한 내용은 이번 내용과 관련이 적어 링크로 남기도록 하겠습니다.)

`feature/new-feature` 브랜치에서는 CLI를 이용하여 커밋을 수정해보고, `master` 브랜치에서는 SourceTree를 이용하여 커밋을 수정해보도록 하겠습니다.

(CLI를 이용한) INTERACTIVE Rebase 를 이용한 커밋 수정

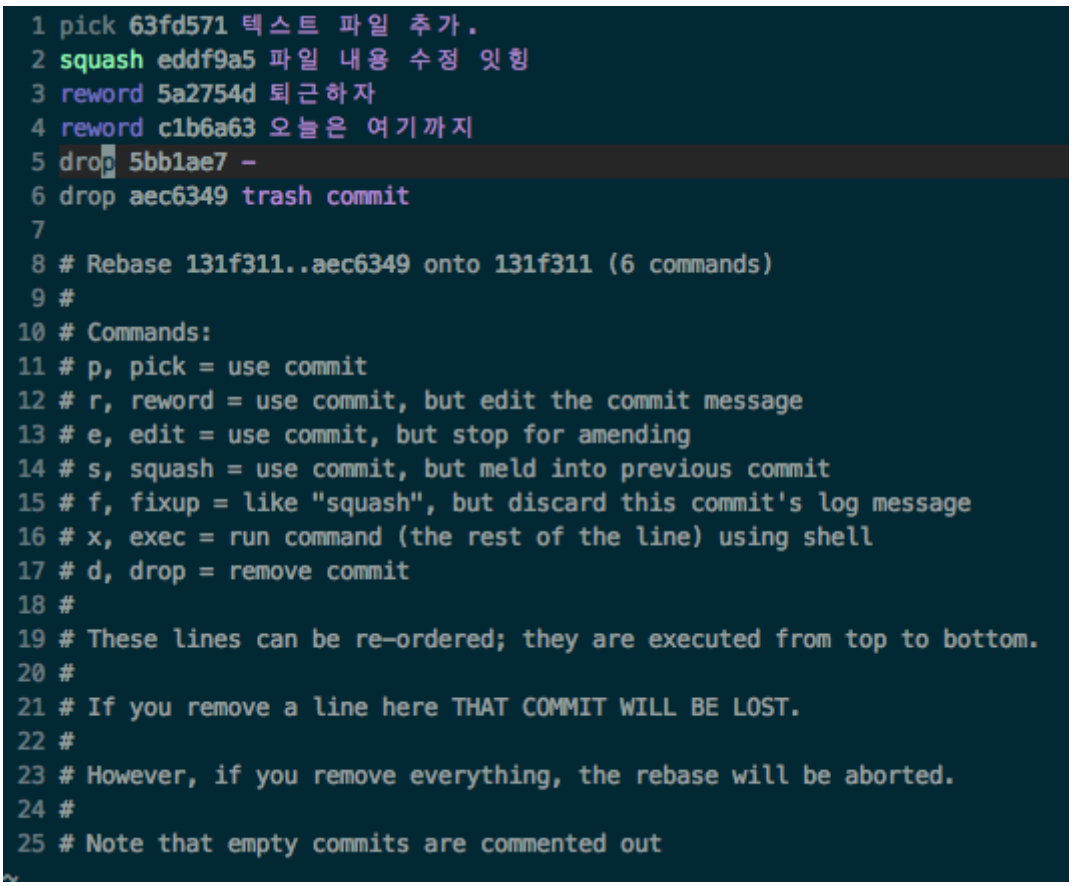
interactive rebase 를 이용하려면 다음과 같은 명령어를 통해 가능합니다.

```
> git rebase -i {hash-value}
```

지정한 포인트 윗부분 부터 커밋의 수정이 가능해집니다. 저는 '텍스트 파일 추가.'부터 커밋을 수정해보겠습니다.

```
> git rebase -i 131f311
```

위와 같이 명령어를 넣으면 아래와 같은 그림을 볼 수 있습니다.



> 처음에는 모두 pick 으로 되어 있습니다.

보시는 것 처럼 Init commit (131f311) 이후의 커밋에 대해 나와 있습니다.

커밋 합치기

먼저 커밋을 합쳐보겠습니다. 제가 남긴 커밋중 '파일 내용 수정 잇힝' 과 '텍스트 파일 추가' 이 두가지 커밋은 거의 동일한 내용의 커밋입니다. 따라서 이 두개의 커밋을 합치려고 합니다.

> Squash : 이 명령어는 두개 또는 그 이상 커밋을 단일 커밋으로 합칠 수 있습니다. 사용할 커밋을 선택한 뒤에 이전 커밋으로 수정됩니다. Git은 rebase를 일시 정지하고 다중 커밋으로부터 커밋 메시지를 포함하여 텍스트 에디터를 엽니다. 만족스럽게 메시지를 수정하고 파일에 저장한 후에 에디터를 닫습니다. Git은 rebase를 재개합니다.

`Squash` 를 이용하면 적용한 커밋을 기준으로 하위의 커밋(두개 또는 그 이상의 커밋)을 하나의 커밋으로 합칠 수 있습니다.

```
1 pick 53fd571 텍스트 파일 추가
2 squash eddf9a5 파일 내용 수정 잇힝

... 생략
```

위와 같이하면 '파일 내용 수정 잇힝'과 그 하위 커밋인 '텍스트 파일 추가' 이 두가지 커밋을 합치겠냐고 묻게됩니다. 그러면 알맞은 커밋 메시지로 수정을 해주고 저장후 종료 (:wq!)를 해주면 두개의 커밋이 단일 커밋으로 합쳐지게 됩니다.

커밋 메시지 수정하기

두번째로 커밋 메시지를 수정해보겠습니다.

> reword : pick과 유사하지만 rebase 진행을 일시 정지하고 커밋 메시지를 변경할 기회가 주어집니다. 커밋 내용은 변경되지 않습니다.

```
3 reword 5a2754d 퇴근하자
4 reword c1b6a63 오늘은 여기까지

... 생략
```

이 두가지 커밋은 커밋 메시지가 마음에 들지 않습니다. `reword` 키워드로 바꾸어준 후 저장종료(:wq!)를 해주면 커밋 메시지 수정창이 나오며, 적당한 커밋메시지로 수정해주면 됩니다.

커밋 지우기

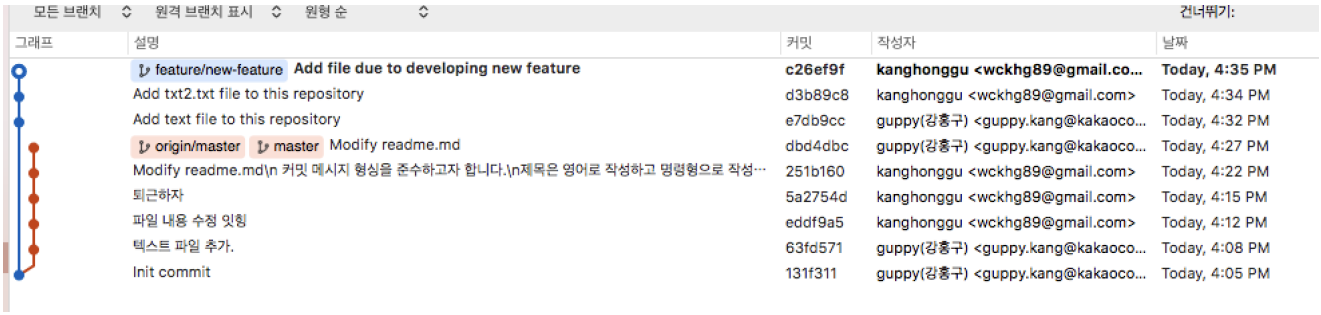
마지막으로 불필요한 커밋을 삭제해 보겠습니다. 저는 '-'과 'trash commit'이라는 두개의 불필요한 커밋을 지워보겠습니다.

> drop : 커밋을 삭제한다.

```
5 drop 5bb1ae7 -
6 drop aec6349 trash commit

... 생략
```

위와 같이하면 해당 커밋은 사라지게 됩니다.

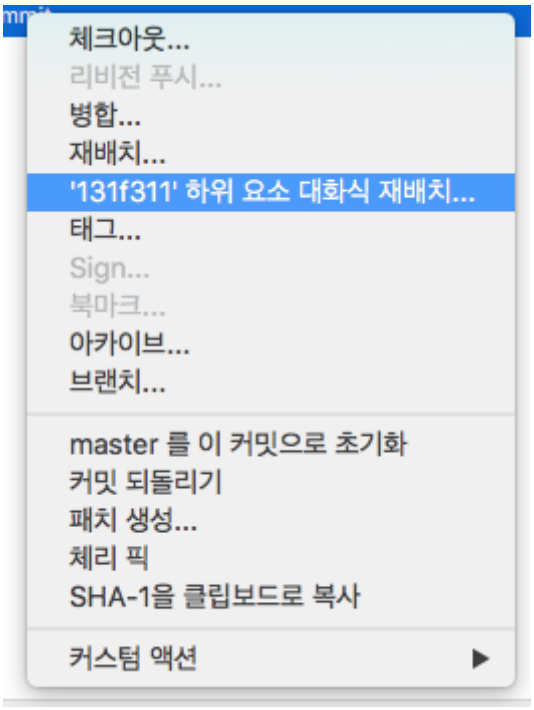


모든 커밋을 수정한후 위와 같이 커밋이 깔끔해진 것을 확인할 수 있습니다. 아직 지저분해 보이는 master 브랜치는 소스트리를 이용하여 수정을 진행해보겠습니다.

(SourceTree를 이용한) INTERACTIVE Rebase 를 이용한 커밋 수정

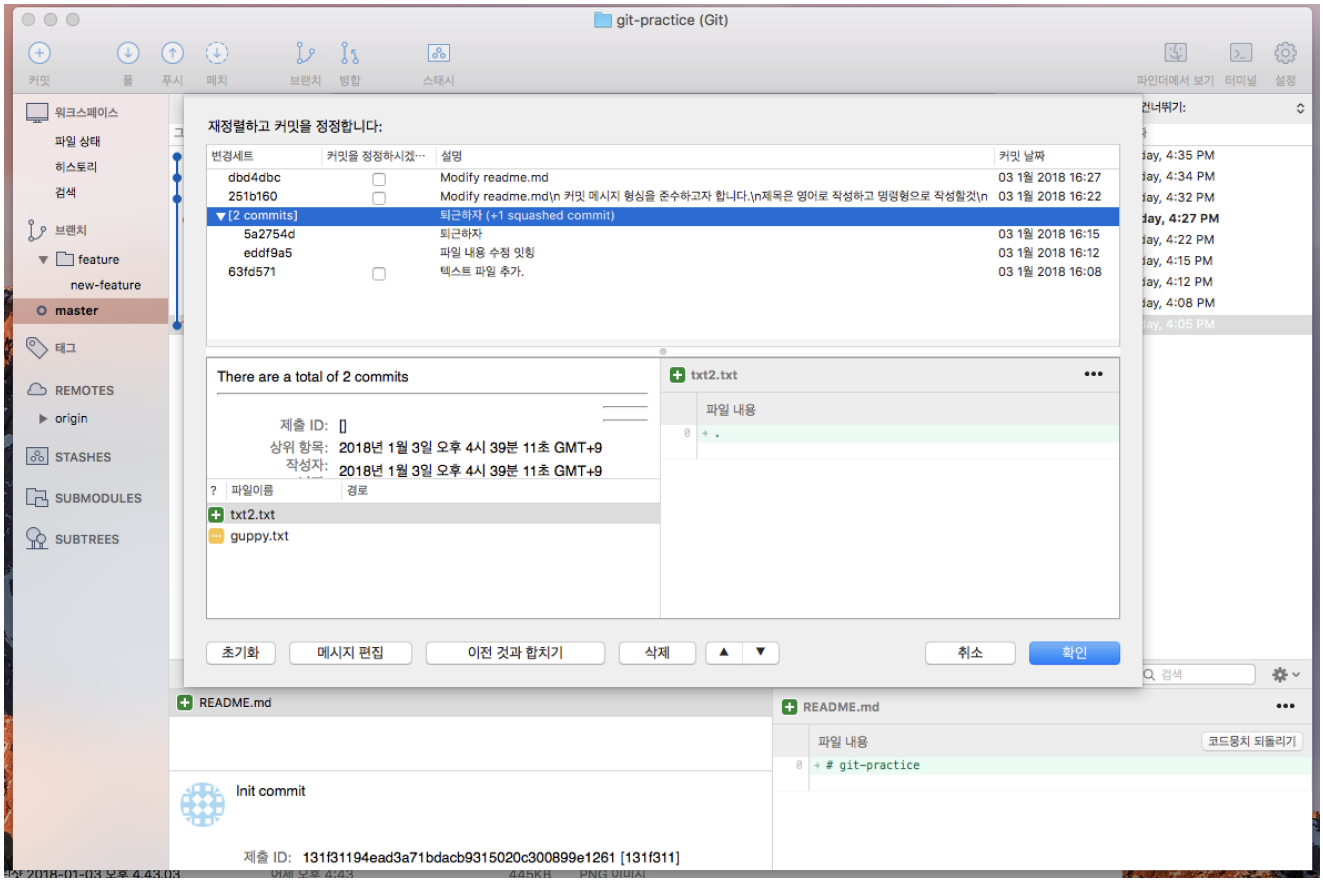
저도 아직은 CLI를 이용한 깃 관리 보다는 SourceTree를 이용한 관리가 더 편리하여 소스트리를 주로 사용하고 있습니다. 많은 분들이 그러실 것 같은데 이어서 마스터 브랜치에 대한 커밋은 SourceTree 툴을 이용해서 진행해 보겠습니다. (~~아무래도 Git은 CLI 환경이 더 편리한 부분이 많은것 같아 CLI도 익숙해지고자 노력을 해야겠습니다.~~)

마스터 브랜치는 'InitCommit' 상위 커밋인 '텍스트 파일 추가' 부터 커밋을 수정해보겠습니다. 소스트리를 이용해서 `Interactive rebase` 를 이용하고 싶다면, 수정을 원하는 지점의 한단계 아래의 커밋에 우클릭을 하시면



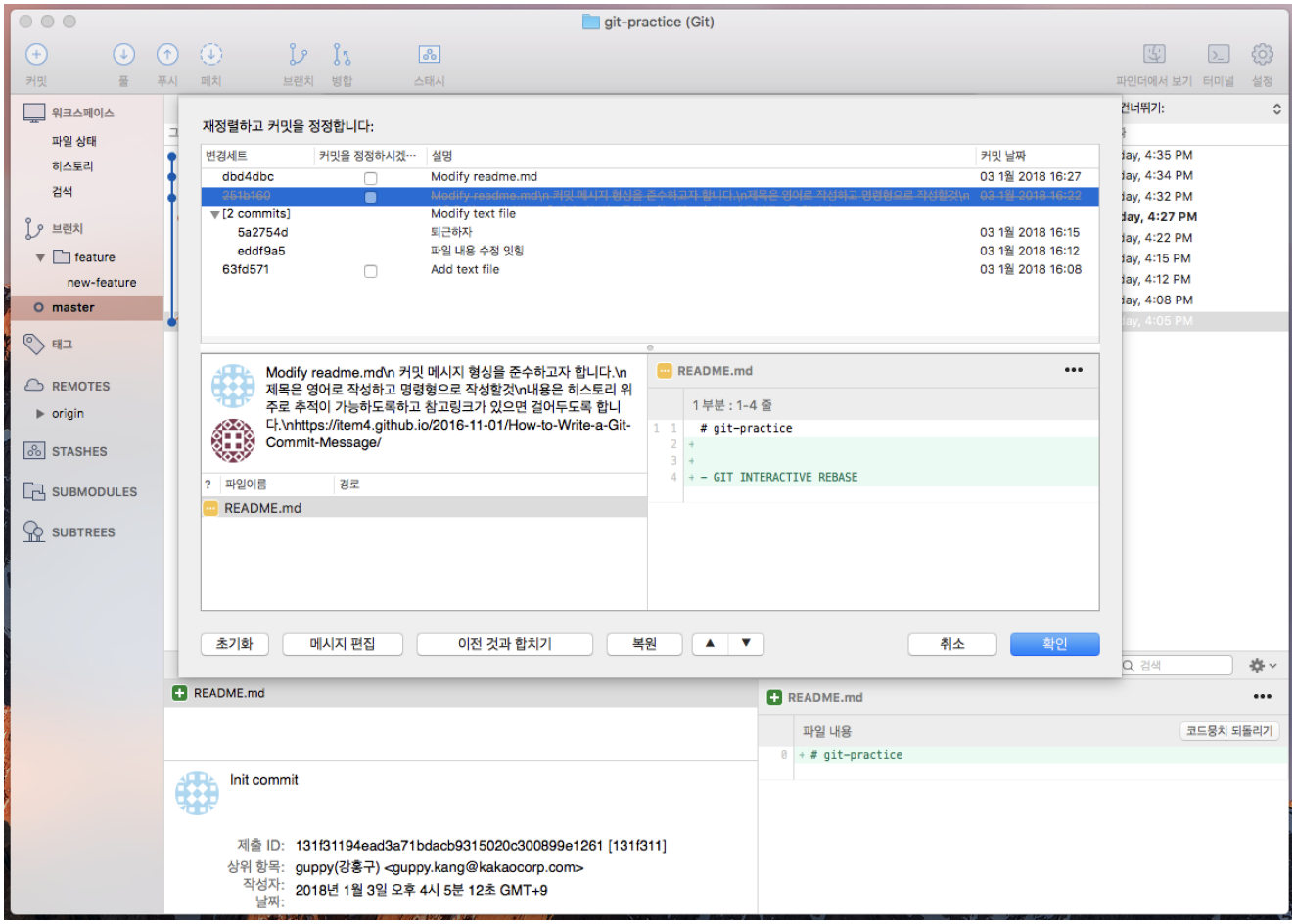
위와 같이 `[Hash 값] 하위 요소 대화식 재배치` 항목이 보입니다. 해당 항목을 선택하시면 하위 커밋인 '텍스트 파일 추가' 부터 커밋을 재편집 할 수 있는 창이 나오게 됩니다.

커밋 합치기 및 커밋 메시지 수정하기



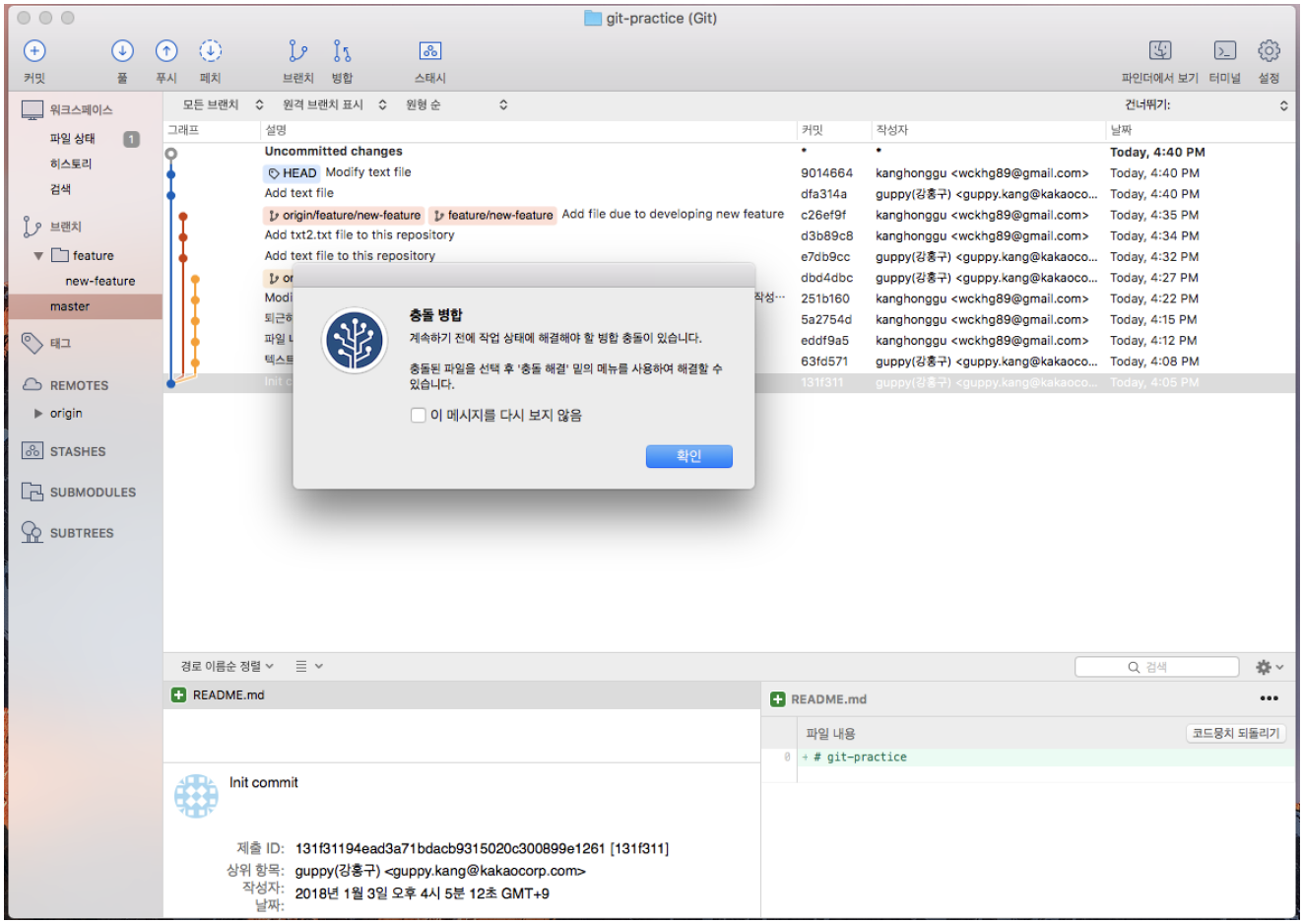
먼저 '퇴근하자', '파일 내용 수정 잇힝' 이 두개의 커밋을 합치고자 합니다. 위의 그림과 같이 '퇴근하자' 커밋을 선택 후 이전 것과 합치기 를 눌러주면 두개의 커밋이 위의 그림과 같이 합쳐지게 됩니다. 또한 합쳐진 (Squash) 커밋 묶음에 대한 커밋 메시지를 수정하고 싶다면 해당 커밋(합쳐진 커밋)을 더블 클릭 또는 메시지 편집 을 통해서 원하는 커밋 메시지로 수정 할 수 있습니다.

커밋 지우기

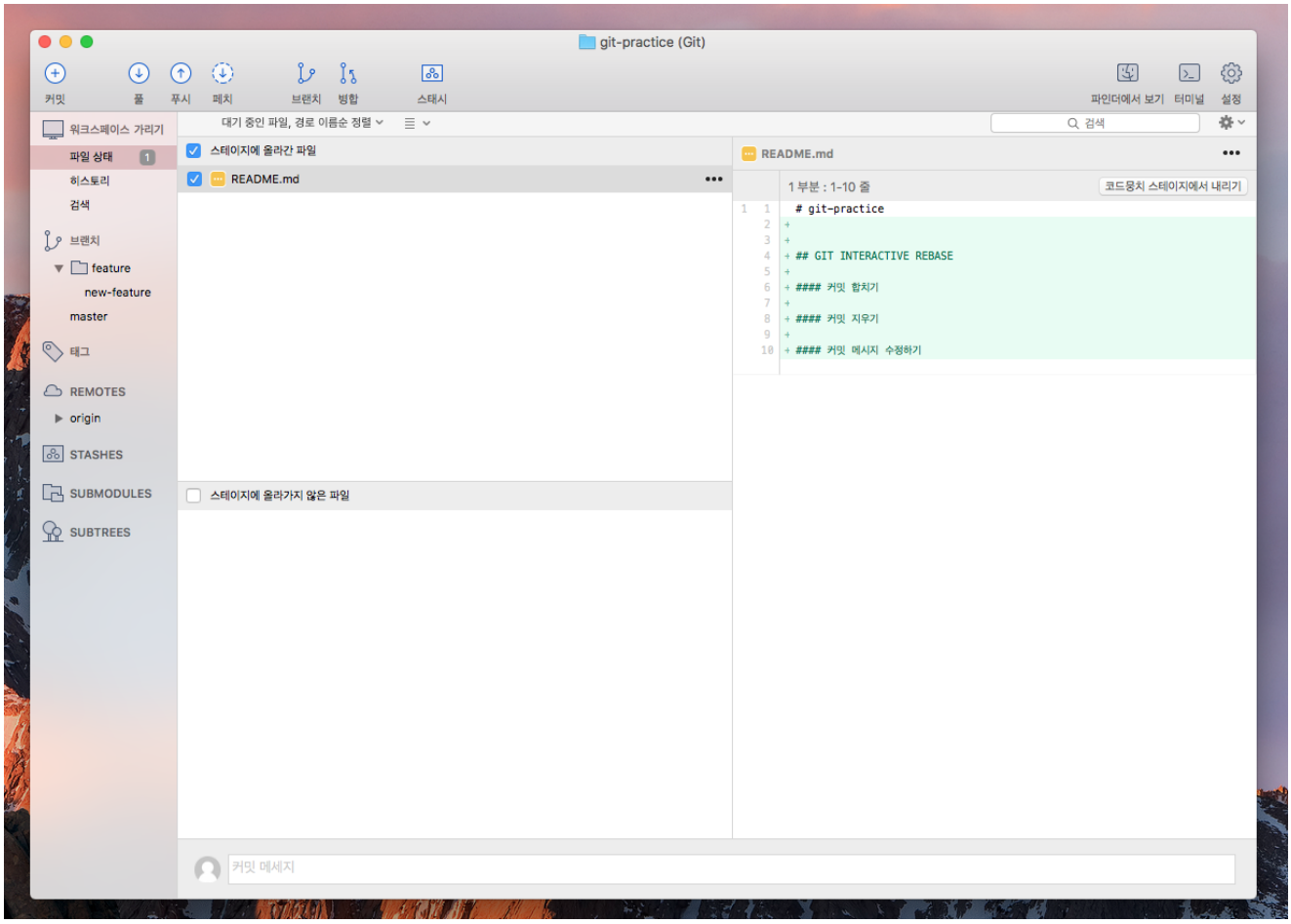


이번에는 커밋 메시지를 지워보겠습니다. 위의 그림에서 보는 것 처럼 지우고자 하는 커밋을 선택한 후에 삭제 버튼을 눌러주시면 위의 그림과 같이 ~~취소선~~ 이 해당 커밋에 나타나며 커밋이 지워지게 됩니다. (위 그림에서는 삭제된 상태라 삭제 버튼이 복원 버튼으로 보입니다.)

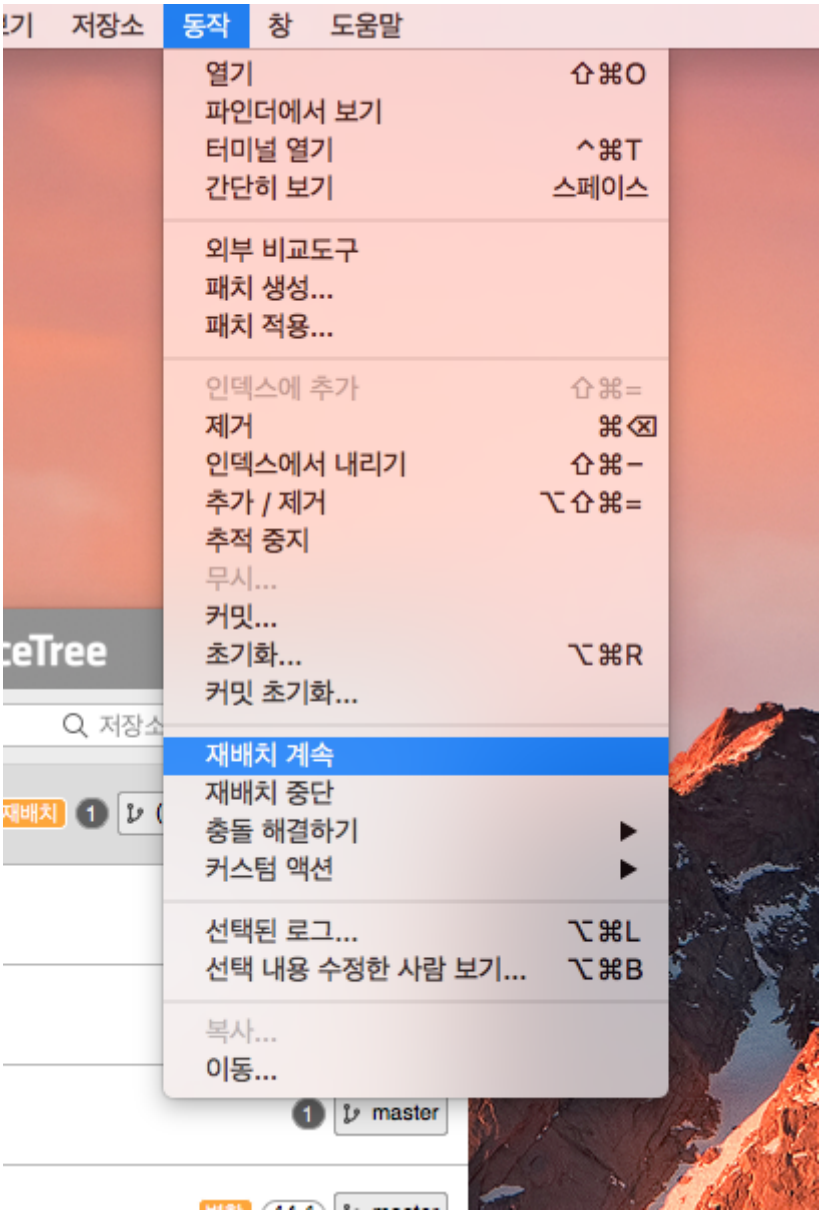
리베이스중 충돌 병합



리베이스과정을 완료하면 **종종 (꾀 자주..)** 충돌이 일어나게 됩니다. 충돌이 일어나면 당황을 하게 되어 `rebase --abort` 가 막 떠오르며 급격히 소심해지게 되는것 같습니다.(제가 그랬습니다..) 리베이스 과정이 실패하더라도 언제든지 `abort` 명령어를 이용해서 되돌릴 수 있으니 걱정하지 말고 병합을 계속 진행하시면 됩니다. 명령어로는 `git rebase --continue` 를 이용하면 되지만 저는 SourceTree를 이용해 보겠습니다.

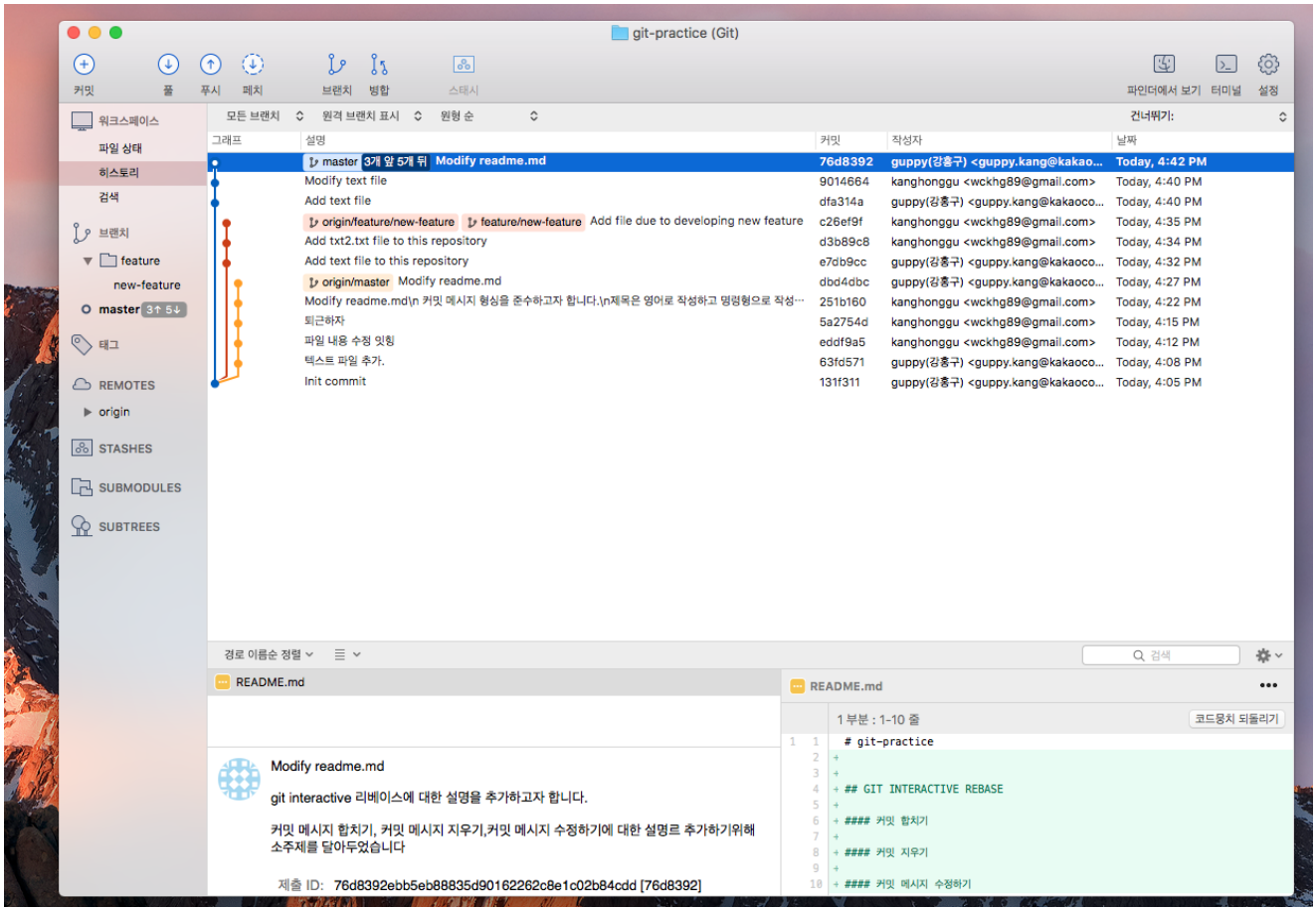


충돌이 난 부분을 위의 그림처럼 해결한 후에 스테이지에 올리기 (`git add .`)를 해준 후에 아래의 그림처럼 재배치 계속 버튼을 눌러 병합을 계속 진행합니다.



> 상단 네비게이션 바의 동작 > 재배치 계속

예제는 간단한 샘플이어서 한번의 충돌 병합으로 쉽게 해결되었지만 실제 복잡한 코드에서는 재배치 계속 을 여러번 해주어야 완전히 리베이스가 완료됩니다. 리베이스가 완료되게 되면 아래의 그림과 같은 트리 모양이 나오게 됩니다.

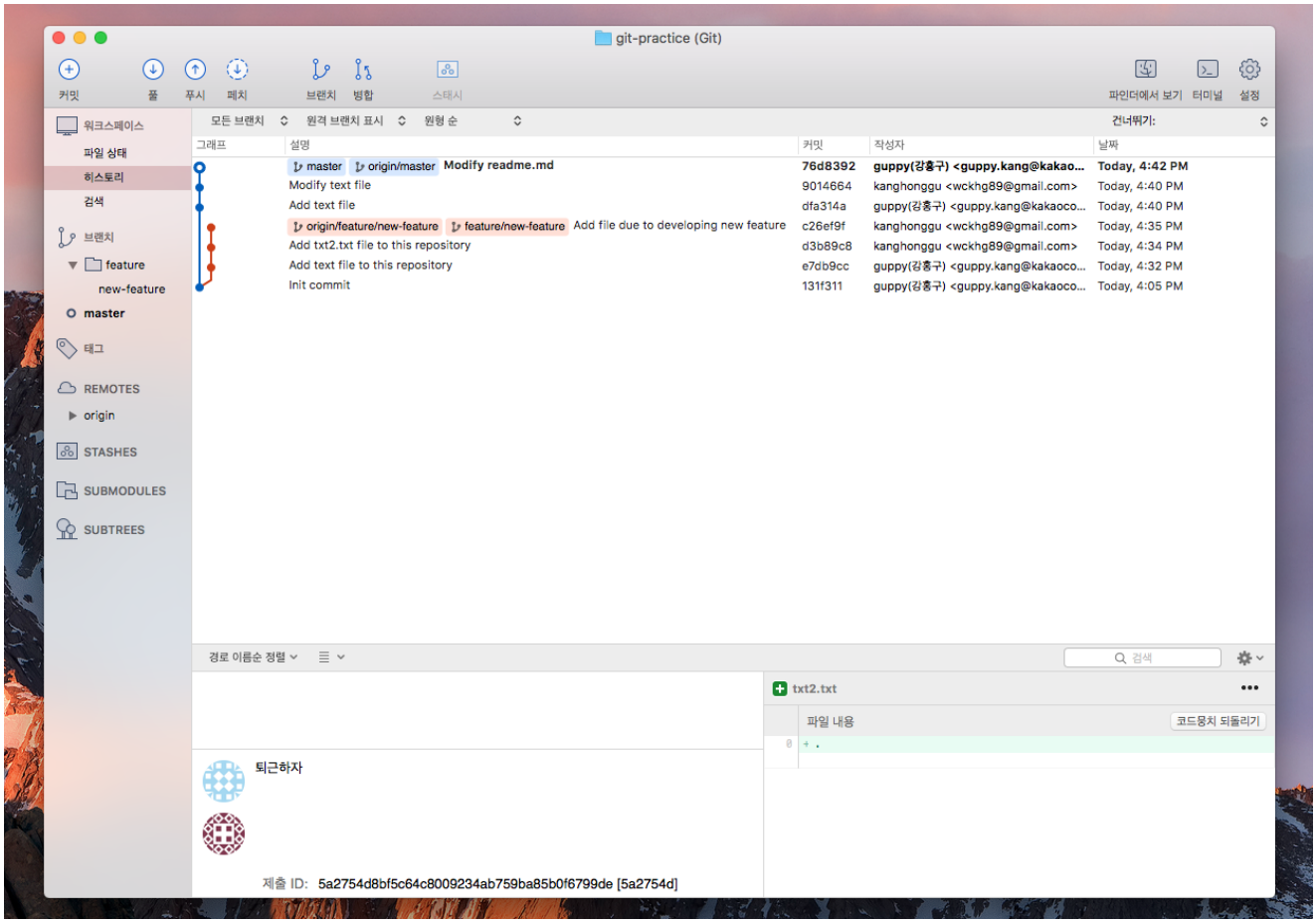


> 로컬 리파지토리에 새로운 master 브랜치가 생겼습니다.

이렇게 커밋 메시지를 수정해주면 새로운 브랜치가 생성되게 됩니다. 그러면 기존 원격 마스터 브랜치와 합쳐주어 커밋을 정리해야 합니다.

```
> git push -f origin head:master
```

위 명령어를 통해서 강제로 푸시를 해주면 아래의 그림과 같이 커밋이 정리되며 리베이스과정이 완료되게 됩니다. (소스트리를 이용한 강제 푸시기능을 사용하는 법을 몰라 저는 강제 푸시는 명령어를 사용하고 있습니다.)



> 최종 완성된 레파지토리의 트리 모양

결론

깃을 사용하며 커밋을 수정하다가 혹시나 리파지토리를 망칠까바 넘어갔던 경험이 많이 있습니다. 이번에 `Interactive rebase` 기능을 사용해보면서 그 기능이 상당히 편리하고 유용하다는 생각을 했습니다.

참고 문서를 읽어보던 중 이미 원격 저장소에 올라간 브랜치는 `rebase` 를 하지 않는게 좋다는 말을 많이 보았습니다. 앞으로는 원격 저장소에 푸시를 하기전에 항상 커밋을 살펴보고 협업자가 더 알아보기 쉬운 커밋의 형태로 수정을 거친 후에 원격 저장소에 푸시를 하는 습관을 가져볼 생각입니다.

참고 문서

Git 커밋 메시지 작성법: https://item4.github.io/2016-11-01/How-to-Write-a-Git-Commit-Message/

[번역]GitHub / Advanced Git / 대화형 Rebase : http://minsone.github.io/git/github-advanced-git-interactive-rebase