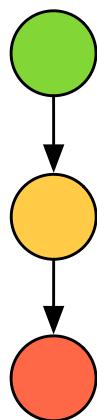


토미의



Git

with 소스트리

저자 계주성

## 저자소개

계주성 (이메일: kyejusung@gmail.com, 트위터: @kyejusung)

Fomola라는 회사에서 창업 멤버로 일했고, 현재 카카오모빌리티에서 iOS 개발자로 일하고 있다.

## 토미의 Git with 소스트리

발행 2022년 4월 29일

저자 계주성

펴낸이 한건희

펴낸곳 주식회사 부크크

출판사 등록 2014.07.15 (제2014-16호)

주소 서울특별시 금천구 가산디지털1로 119 SK트원타워 A동 305호

전화 1670-8316

이메일 [info@bookk.co.kr](mailto:info@bookk.co.kr)

ISBN 979-11-372

[www.bookk.co.kr](http://www.bookk.co.kr)

© 계주성 2022

본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

이 책에 대한 의견이나 조언 그리고 오탈자나 잘못된 내용 등은 위 이메일이나 트위터를 통해 전달해주시면 감사드리겠습니다.

## 저자의 글

제가 Git을 처음 알게 된 것은 2015년입니다. 이후 Git을 쭉 사용해 오다 2019년 Git을 제대로 알고 싶다는 생각을 하게 됐습니다. 당시 저는 Git을 사용한 지 4~5년이 지났지만 실무에서 자주 사용하는 명령어만 사용할 줄 알았을 뿐, Git에 대해 제대로 알지는 못하는 상황이었습니다. 이 상태로 Git을 사용하는 것은 평소에는 문제가 없었지만, 예상치 못한 문제가 발생했을 때 그 문제를 자신 있게 해결하지 못했습니다. 이런 상황이 지속해서 반복되는 것을 보게 돼 제 자신이 부끄러웠습니다. 이래선 안 되겠다 싶어 시간을 내서 Git에 대해 제대로 알고 넘어가야겠다고 결심했습니다.

Git에 대해 공부하기로 결심한 후, 여러 책을 찾아보았지만 제가 원하는 책은 찾을 수 없었습니다. 제가 원했던 책은 명령어에 대한 설명 위주가 아니라, Git의 핵심을 쉽고 짧게 설명한 책이었습니다. 명령의 실행은 CLI(Command Line Interface) 환경이 아니라 소스트리에서 어떻게 사용하는지에 대한 설명이었으면 하는 바람도 있었습니다.

제가 원하는 책을 원하는 사람이 더 있을 것 같아서 이런 책을 제가 직접 써보기로 결심했습니다. Git의 핵심적인 내용을 쉽고 짧게 설명하기 위해 여러 자료를 참고했습니다. Git을 개발한 리누스 토발즈(Linus Torvalds)의 2007년 구글 테크톡에서의 Git 소개 영상을 비롯해 Github의 공동 창업자이자 Pro Git 책의 공동 저자인 스캇 샤콘(Scott Chacon)의 여러 강의와 책, 20권 이상의 Git 책과 강의 슬라이드, Git 개발 코드에 포함된 매뉴얼, 스택오버플로우 등이 참고한 자료입니다. Git을 공부하면서 정확히 이해되지 않는 부분이 생기면 그 부분을 끝까지 추적해 확실히 이해했습니다. 이해한 내용은 어떻게 하면 쉽게 전달할 수 있을지 전달 방법을 고민했습니다. 책에 담은 내용도 자주 사용하고 핵심적인 내용이라고 생각하는 부분을 선별 했습니다. Git에 대한 내용 중 이 책에 담겨있지 않은 부분도 있지만, 이 책에 나와있는 기본적인 지식을 확실히 이해한다면 다른 내용의 이해는 그렇게 어렵지 않을거라 생각합니다.

이 책은 이런 저의 의도와 고민이 담긴 책입니다. 제가 의도한대로 이 책이 Git을 이해하는 데 도움이 된다면 저자로서 더없이 기쁠 것 같습니다. 이 책이 어떤 분께 도움이 되는지에 대한 내용은 바로 뒤에 적어 두었습니다. 아직 이 책을 읽어야 할지 말아야 할지 고민이 되는 분은 몇 분의 시간을 더 사용해 그것을 판단하는 데 사용하시기 바랍니다.

## 이 책의 대상 독자

이 책은 다음에 해당하는 분들을 위해 쓰여졌습니다.

### 1. Git을 처음 접하시는 분

### 2. Git을 제대로 알고 싶으신 분

- 브랜치의 정의
- 머지와 리베이스의 차이
- 리셋과 리버트의 차이
- 머지시 충돌 발생 이유와 해결 방법
- 로컬저장소와 원격저장소
- 원격저장소 추적 브랜치
- 잃어버린 커밋 복원하기

### 3. Git의 내부 구조를 알고 싶으신 분

- Git이 어떤 객체로 구성돼 있는지
- 저장소의 구조와 역할
- 커밋의 해시는 어떻게 생성되는지

### 4. Git을 더 잘 활용하고 싶으신 분

- 레프로그
- 리베이스를 사용한 풀
- 패치
- 어멘드

## 이 책을 읽는 법

이 책은 두 개의 파트로 구성돼 있습니다. 파트I에서는 Git의 기본 개념에 대해 설명 했습니다. 파트2에서는 Git의 여러 명령을 사용하는 방법에 대해 설명했습니다.

이 책을 반드시 처음부터 읽을 필요는 없습니다. 파트의 각 챕터는 독립적으로 구성돼 있어서 이미 잘 알고 있는 내용은 건너뛰고 읽고 싶은 챕터를 선택해 바로 읽어도 이해하시는데 무리가 되지 않습니다. 다만 파트I에서 설명한 Git의 전반적인 구조와 기본 개념은 파트II의 내용을 정확하게 이해하는데 중요한 기반 지식이 되기 때문에 가능하면 파트I은 먼저 읽어보시는 것을 추천 드립니다.

챕터의 구성은 인트로, 개요, 명령의 정의와 설명, 소스트리에서 사용하기, 정리 순으로 돼 있습니다. 설명은 이해가 쉽도록 그림을 최대한 활용했습니다. 소스트리에서 사용하기에서는 명령 실행의 각 단계별로 스크린샷을 첨부해 따라하기 쉽도록 구성했습니다. 마지막으로 정리에서는 챕터에서 알아본 내용을 요약해 챕터의 핵심 내용을 다시 한번 확인하고 넘어갈 수 있도록 했습니다.

챕터의 내용은 크게 이론과 실습으로 나누어집니다. Git을 잘 사용하기 위해서는 이론만 잘 아는 것을 넘어 실제 잘 사용할 수 있어야 합니다. 잘 사용하기 위해서는 많이 써봐야합니다. 그렇기 때문에 가능하면 책에 나와 있는 실습을 직접 따라서 수행해 보시기를 추천드립니다. 테스트용 저장소를 생성해 망가뜨리기도 하고 실수도 하며 여러 명령을 실행하다 보면 어느새 Git을 사용하는데 자신감을 갖게 되시리라 생각합니다. 책의 명령이 너무 많아 전부 따라 해보기가 어렵다면 리베이스, 리버트, 리셋 이 세 가지 명령만이라도 충분히 연습해 사용 방법을 익혀두시길 권해 드립니다. 특히 리버트와 리셋은 변경 사항을 되돌리는 방법으로 이 명령이 무엇이고 어떻게 사용하는지만 확실하게 익혀두신다면 중간에 어떤 실수를 하더라도 대부분의 작업을 안전하게 복원할 수 있을 것입니다.



## 감사의 글

이 책이 나오기까지 많은 분이 도움을 주셨습니다.

먼저 사랑하는 아내 김보미에게 감사를 전합니다. 제 영혼의 친구이자 든든한 반려자인 아내의 도움 없이는 이 책은 나올 수 없었을 것 입니다. 책쓴다고 집안일을 소홀히 할때도 있었고 피곤해하며 힘들어하는 모습을 보일 때도 있었는데, 그럴 때마다 아내는 제 부족함을 감싸주고 지지해 주었습니다. 아내에게 진심으로 고맙다는 말과 아끼고 사랑한다는 말을 전하고 싶습니다.

이 책의 마무리 단계에서 많은 분이 원고를 읽고 의견을 주셨습니다. 김도현, 김태훈, 이종우, 전영진. 이분들의 소중한 의견을 통해 다양한 오자를 수정했고, 책 내용을 일관성 있고 더 이해하기 쉽도록 수정할 수 있었습니다. 바쁘신 가운데서도 제 리뷰 부탁에 흔쾌히 응해주시고 귀한 시간과 마음을 써주신 이분들께 진심으로 감사드립니다.

마지막으로 하나님께 감사드립니다. 이 책을 쓰는 건 하나님께서 제게 주신 사명이었습니다. 능력 없고 부족한 제게 이런 귀한 사명을 주시고 사명을 완수할 수 있도록 지혜와 건강을 허락해 주신 하나님께 감사드립니다.

지금 가장 크게 드는 마음은 감사와 후련함입니다. 무거운 마음의 짐을 1년 넘게 짊어지고 있다가 이제 내려 놓을 수 있다고 생각하니 후련합니다. 이제는 이 짐은 벗어두고 곧 태어날 첫째 예하를 돌보는 데 힘을 쏟을 수 있을 것 같아 기쁩니다.

Git을 이해하며 제대로 사용하고자 하는 분들께 이 책이 도움이 되길 바랍니다.

고맙습니다.

2022. 4. 17 어느 일요일 저녁

저자 계주성



# 목 차

저자의 글-----	3
이 책의 대상 독자 -----	4
이 책을 읽는 법 -----	5
감사의 글-----	7

## Part I. Git 개념 이해하기 (Understanding Git Concepts)

<b>1. 버전 관리 시스템의 역사 (History of Version Control System)-----</b>	<b>15</b>
1.1 개요-----	16
1.2 파일/폴더 시스템-----	16
1.3 로컬 버전 관리 시스템 -----	17
1.4 중앙집중형 버전 관리 시스템-----	18
1.5 분산형 버전 관리 시스템 (DVCS: Distributed Version Control System) -----	20
1.6 정리-----	22
<b>2. 델타 모델과 스냅샷 모델 (Delta Model and Snapshot Model) -----</b>	<b>23</b>
2.1 델타 모델 -----	23
2.2 스냅샷 모델 -----	29
2.3 정리-----	38
<b>3. Git의 객체 (Git Object Types) -----</b>	<b>39</b>
3.1 개요-----	39
3.2 블립 (Blob: Binary large object)-----	40
3.3 트리 (Tree)-----	44
3.4 커밋 (Commit)-----	50
3.5 태그 (Tag) -----	55
3.6 객체 id와 SHA-1 -----	58
<b>4. 저장소 구조 (Repository Structure)-----</b>	<b>63</b>
4.1 개요-----	63
4.2 저장소의 구조 -----	63

4.3 파일의 상태 -----	67
4.4 Git의 동작 -----	69
4.5 소스트리에서 사용하기 -----	74
4.6 정리-----	85

## Part II. Git 사용 하기 (Using Git)

5. Git 사용 환경 (Environment for Git) -----	89
6. 브랜치 (Branch) -----	91
6.1 개요-----	91
6.2 브랜치의 구성 및 동작 -----	92
6.3 브랜치 변경시 내부 동작 -----	99
6.4 소스트리에서 사용하기 -----	101
6.5 정리-----	110
7. 변경 사항 되돌리기 (Undoing Changes)-----	111
7.1 개요-----	111
7.2 리셋 (Reset) -----	112
7.3 리버트 (Revert) -----	120
7.4 정리-----	128
8. 머지 (Merge) -----	129
8.1 개요-----	129
8.2 머지의 선행 조건 -----	130
8.3 머지의 종류-----	130
8.4 소스트리에서 사용하기 -----	135
8.5 정리-----	144
9. 충돌처리 (Resolve Conflicts) -----	145
9.1 개요-----	145
9.2 충돌이 발생했을 때 일어나는 현상 -----	146
9.3 충돌 해결방법 -----	147
9.4 충돌처리 팁-----	149

9.5 전체 저장소 관점에서 총돌 살펴보기 -----	150
9.6 소스트리에서 사용하기 -----	152
9.7 정리-----	167
<b>10. 리베이스 (Rebase) -----</b>	<b>169</b>
10.1 개요-----	170
10.2 리베이스란? -----	172
10.3 리베이스의 동작 -----	173
10.4 리베이스시 주의사항 -----	175
10.5 머지와 리베이스의 차이점 -----	175
10.6 리베이스의 활용 (Interactive Rebase) -----	176
10.7 소스트리에서 사용하기 -----	179
10.8 정리-----	200
<b>11. 원격저장소 (Remote Repository) -----</b>	<b>201</b>
11.1 원격저장소 -----	202
11.2 원격저장소 추적 브랜치 (Remote Tracking Branch)-----	203
11.3 클론 (Clone)-----	204
11.4 푸시 (Push) -----	214
11.5 패치 (Fetch)-----	222
11.6 풀 (Pull)-----	230
11.7 정리-----	238
<b>12. 잃어버린 커밋 복원하기 (Reflog)-----</b>	<b>239</b>
12.1 개요-----	240
12.2 레프로그의 정의 -----	240
12.3 레프로그의 사용 방법 -----	241
12.4 정리-----	251
<b>13. 팁 &amp; 트릭 (Tip &amp; Trick) -----</b>	<b>253</b>
13.1 스탠시 (Stash) -----	253
13.2 풀 수행시 리베이스 옵션 사용하기 (Pull with Rebase) -----	259
13.3 특정 파일의 변경 이력 확인하기 (Log on Specific Commit) -----	265

13.4 패치 (Patch) -----	269
13.5 커밋 템플릿 (Commit Template) -----	277
13.6 가장 최근 커밋 변경하기 (Amend Last Commit) -----	280
13.7 선택적으로 커밋 적용하기 (Cherry Pick) -----	285

## 10. 리베이스 (Rebase)

“토미~ 안녕하세요?”

“네~ 안녕하세요?”

“다음 배포 때, 그동안 개발했던 자율주행 이동 수단을 포함해 나갈 수 있을까요?”

“자율주행 이동 수단이라면 택시, 버스, 배, 비행기, 자전거 말씀하시는 거죠?”

“네 맞아요. 모든 이동 수단을 사용할 수 있도록 서비스에 반영하고 싶어요. 가능할까요?”

“네~ 물론이죠.”

토미는 이동 수단별로 각기 다른 브랜치에서 개발했기 때문에 배포시 필요에 따라 모든 기능을 포함시키거나 일부만 포함시킬 수 있었습니다.

‘자~ 그럼 각 기능 브랜치를 배포할 브랜치에 머지 해볼까?’

‘짜잔~ 머지 완료! 이렇게 배포할 브랜치에 머지가 끝났으니 배포 준비 완료됐네?’

‘역시! 나 토미야! ㅋ’

토미는 스스로 대견해하며 만족스러운 미소를 띠었습니다. 하지만 이내 한 가지가 눈에 들어왔습니다.

‘가만있자! 이거 기능적으로는 문제가 없는데, 여러 브랜치가 하나의 브랜치에 머지되니 히스토리가 복잡해 보이네. 이걸 개선할 수 있는 방법은 없을까?’

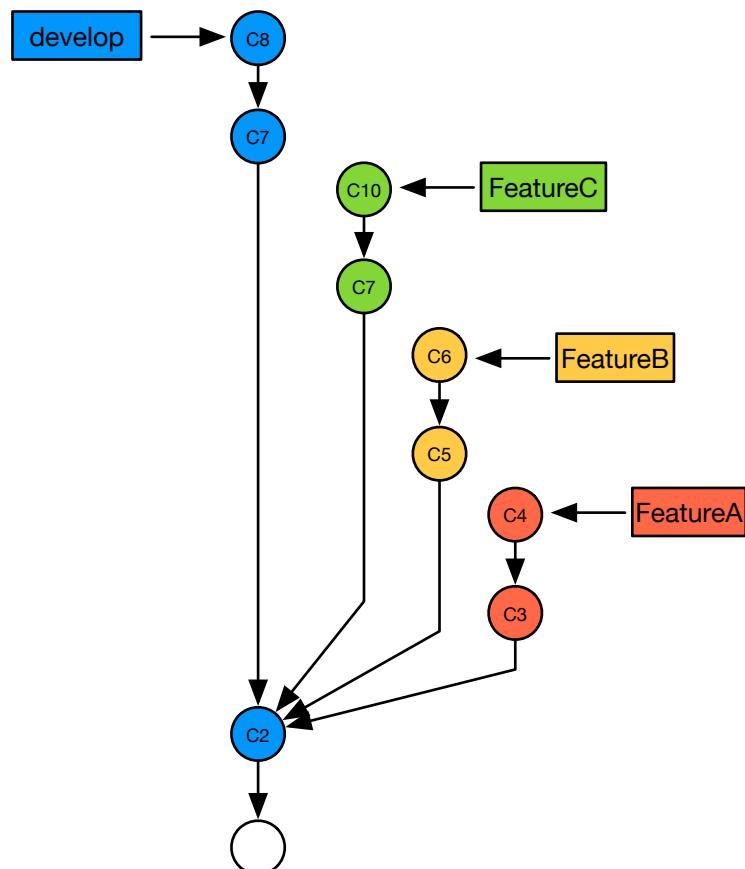
토미는 성공적으로 각 기능을 배포 브랜치에 머지 했지만, 머지된 후 Git의 커밋 히스토리가 마음에 들지 않았습니다.

‘아... 이거 보면 볼수록 복잡해 보인다. 정리하고 싶은데, 이것도 무슨 방법이 있겠지? 어떤 방법이 있는지 한번 알아봐야겠다.’

## 10.1 개요

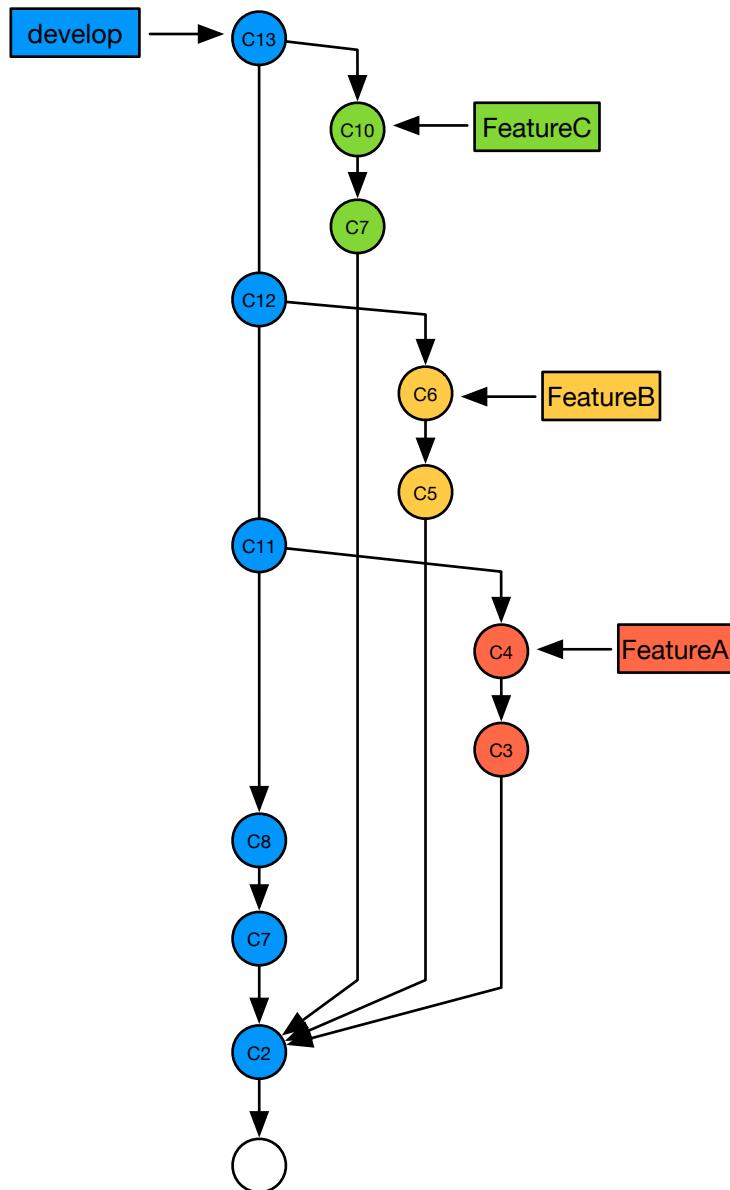
챕터 8에서 브랜치를 합치는 방법 중 하나인 머지에 대해 살펴보았습니다. 때때로 브랜치를 합칠 때 머지대신 다른 방법을 사용해야 할 때가 있습니다. 이번 챕터에서는 리베이스에 대해 살펴보겠습니다. 앞에서 토미가 겪었던 상황을 예를 들어 보겠습니다.

아래 [그림 10-1]과 같이 머지 시킬 브랜치(develop)와 이 브랜치에 여러 기능을 포함시키기 위해 머지할 FeatureA, FeatureB, FeatureC 세 개의 브랜치가 있습니다.



[그림 10-1] develop에 머지 할 세개의 Feature 브랜치

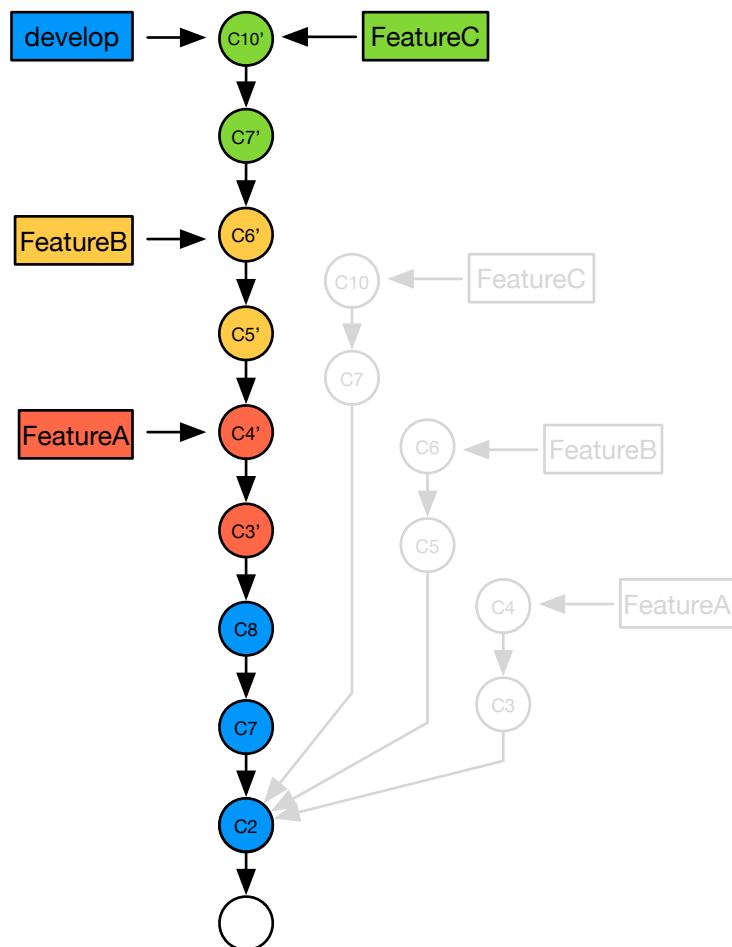
FeatureA, FeatureB, FeatureC 브랜치를 각각 순서대로 develop 브랜치에 머지시키면 Git의 커밋 히스토리는 [그림 10-2]와 같은 모습이 됩니다.



[그림 10-2] develop에 세 개의 Feature 브랜치를 머지한 후 커밋 히스토리

develop 브랜치에는 FeatureA, FeatureB, FeatureC 각 브랜치의 변경사항이 반영된 상태입니다. 그래서 기능상으로는 문제가 없습니다. 하지만 히스토리가 조금 복잡해 보입니다. 만약 develop에 머지된 브랜치가 3개가 아니라 5개 혹은 10개라면 히스토리는 훨씬 더 복잡하게 보일 것입니다.

이런 상황에서 머지 대신 리베이스를 사용하면 아래 [그림 10-3]과 같이 브랜치를 머지커밋 생성없이 일렬로 된 형태로 브랜치를 합칠 수 있어서 Git의 커밋 히스토리를 간결하게 관리할 수 있습니다.

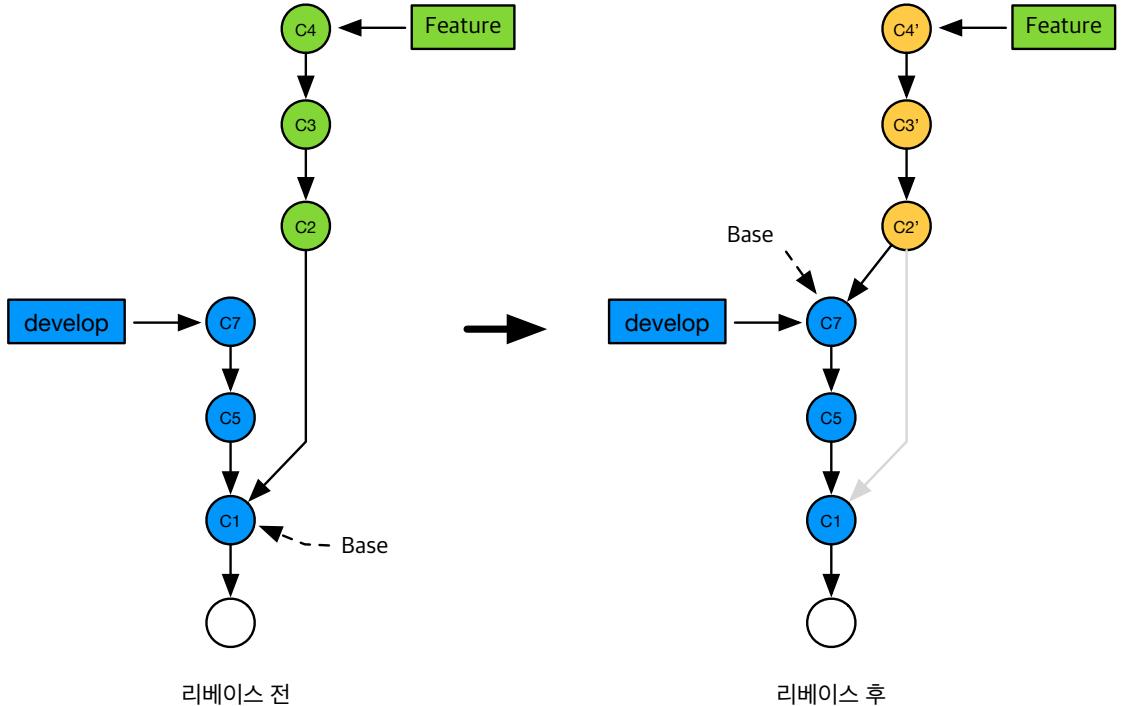


[그림 10-3] 세 개의 Feature 브랜치를 develop 브랜치에 리베이스로 반영 후 커밋 히스토리

이번 챕터에서는 이 리베이스에 대해 알아 보겠습니다.

## 10.2 리베이스란?

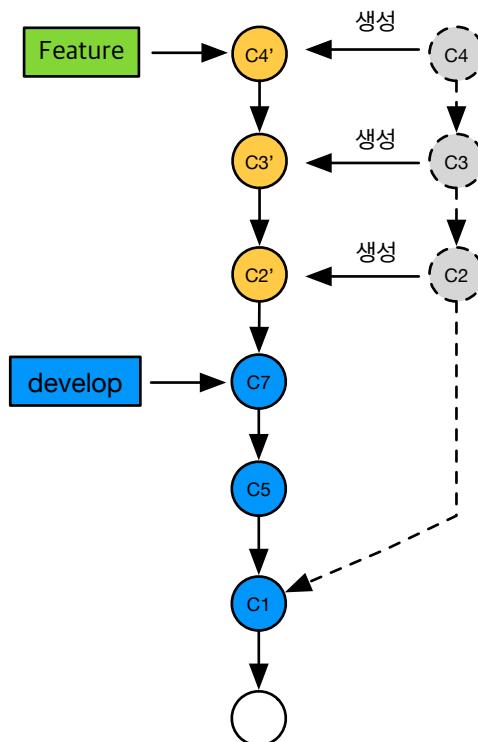
일반적으로 브랜치는 브랜치의 시작점이 되는 base(부모)가 존재 합니다. 리베이스는 브랜치의 base를 변경해 브랜치를 합치는 방법입니다.



위 [그림 10-4]의 왼쪽 리베이스 전 그림에서 Feature 브랜치의 base는 C1 커밋입니다. Feature를 develop와 합치기 위해 리베이스를 수행합니다. 리베이스를 하면 [그림 10-5]의 오른쪽 그림 같이 Feature의 base가 C1에서 C7로 변경되고 두 브랜치가 일렬로된 형태로 합쳐집니다.

## 10.3 리베이스의 동작

리베이스를 수행하면 내부적으로는 각 커밋별로 패치를 생성해 순서대로 다른 브랜치에 적용합니다. 리베이스 후 마치 커밋이 원본 브랜치에서 다른 브랜치로 base만 바뀌어서 이동한 것처럼 보이지만 실제로는 같은 메시지와 변경사항을 포함한 새로운 커밋이 생성돼 재적용된 것입니다.



[그림 10-5] 리베이스의 동작 과정

위 [그림 10-5]는 Feature 브랜치를 develop 에 리베이스하는 과정입니다. Feature 브랜치를 develop에 리베이스 해서 C2, C3, C4 커밋이 develop 에 적용됐습니다. 여기서 주의깊게 보실 것은 C2, C3, C4 커밋이 develop에 그대로 옮겨진 것이 아니라 각각 C2', C3', C4' 로 변경됐다는 것입니다.

앞서 설명드린바와 같이 리베이스를 수행하면 커밋이 그대로 이동하는 것이 아니라 새로 생성됩니다. 그렇기 때문에 리베이스로 생성된 커밋은 원본 커밋과 해시가 다른 커밋이 됩니다. 커밋의 해시가 다르기 때문에 Git이 봤을때 C2, C3, C4는 C2', C3', C4'와 각각 다른 커밋입니다. 커밋을 새로 생성하면 왜 다른 커밋이 되는걸까요? 커밋 메시지나, 커밋 작성자 등 정보는 동일한데 말이죠?

앞의 챕터3.6에서 커밋의 아이디인 SHA-1가 어떻게 만들어지는지 살펴봤었습니다. Git은 커밋을 SHA-1로 구분하는데 커밋의 SHA-1을 생성하는데 사용하는 요소에 커밋 생성 시간과 부모 커밋이 포함된다고 말씀드렸습니다. 위 [그림 10-5] 에서는 리베이스를 수행 해서 커밋이 재생성됐는데 커밋을 재생성 했기 때문에 커밋 생성시간이 변경 되었고, base가 변경 됐기 때문에 C2, C3, C4 커밋의 부모 커밋도 각각 C1, C2, C3에서 C7, C2', C3' 으로

바뀌었습니다. 리베이스 되면서 커밋의 두 가지 정보가 변경 됐기 때문에 리베이스 된 커밋은 이전 커밋과 SHA-1가 다른 완전히 다른 커밋입니다.

## 10.4 리베이스시 주의사항

리베이스시 원본 커밋이 버려지고 새로운 커밋이 생성되기 때문에 리베이스는 일반적으로 개인저장소에서만 수행해야 합니다. 만약 다른 팀원과 함께 사용하는 공용저장소의 커밋을 리베이스하면 공용저장소의 커밋 히스토리가 꼬이게 됩니다.

A라는 사람은 리베이스 전의 커밋에 이어서 커밋을 생성하고, B라는 사람은 리베이스 후의 커밋에 이어서 작업을 했다면 나중에 두 커밋 히스토리는 일반적인 방법으로는 합칠 수 없는 상태가 돼 버립니다. 그렇기 때문에 리베이스는 공용저장소에 푸시된 적이 없는 커밋에만 사용해야 합니다. 그렇다면 공용저장소에 푸시된 커밋은 절대 리베이스를 하면 안되는 걸까요?

꼭 그렇지는 않습니다. 여럿이 협업을 하다보면 공용저장소의 내용을 리베이스 하고 싶을 때가 생기는데 그때는 먼저 동료들에게 리베이스할 브랜치에 아직 푸시하지 않은 내용이 있다면 푸시해 달라고 요청합니다. 그 후 리베이스를 완료하기 전에 그 브랜치에 새로운 커밋을 푸시하지 말라고 요청합니다. 이렇게 서로 합의를 한 후 리베이스를 진행하고, 리베이스가 완료되면 공용저장소에 푸시를 하고 동료들에게 리베이스가 완료됐다고 상황을 공유합니다. 이 후 다른 동료들은 해당 브랜치를 로컬에서 삭제 후 다시 체크아웃해서 사용하면 됩니다.

비록 이렇게 공용저장소에 반영된 커밋을 리베이스 할 수 있지만, 특수한 상황을 제외하고 리베이스는 공용저장소에 푸시된 적이 없는 커밋에 대해서만 수행해야 합니다.

## 10.5 머지와 리베이스의 차이점

머지와 리베이스 두 명령 모두 결국 브랜치를 합치는 일을 하는 것인데, 두 명령어는 어떤 차이가 있을까요?

첫째, 머지는 커밋 히스토리를 변경하지 않습니다. Fast Forward 머지인 경우 커밋을 변경하거나 새로운 커밋을 추가하지 않습니다. Three Way 머지는 새로운 커밋을 추가하지만 이

전 커밋을 변경하지는 않기 때문에 언제든 예전 커밋의 상태로 돌아갈 수 있습니다. 반면 리베이스는 옛날 커밋을 버리고 새로운 커밋을 생성하기 때문에 일반적인 방법으로는 예전의 커밋으로 되돌아 갈 수 없습니다.

둘째, 머지는 한번에 수행되는 명령입니다. 그래서 머지 중 충돌이 발생한 경우 한번의 처리로 충돌 해결이 가능합니다. 반면 리베이스는 리베이스 되는 커밋을 각각 재생성해 적용하기 때문에 리베이스시 충돌이 발생한 경우 충돌 처리를 여러번에 걸쳐 나눠서 처리해야 하는 상황이 발생할 수 있습니다.

셋째, Three Way 머지를 사용하는 경우 리베이스를 했을 때 보다 커밋 히스토리가 복잡합니다. 이 점이 보통 브랜치를 합칠 때 머지 대신 리베이스를 사용하는 가장 큰 이유입니다.

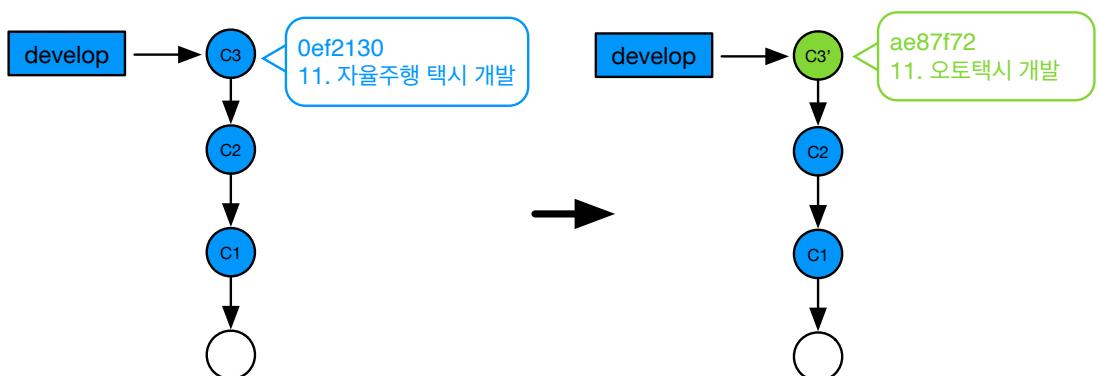
## 10.6 리베이스의 활용 (Interactive Rebase)

앞서 말씀드린 것처럼 리베이스를 수행하면 커밋을 재생성합니다. 인터랙티브 리베이스를 사용하면 리베이스시 커밋이 다시 생성될 때 기존 커밋을 여러 형태로 변경하는 것이 가능합니다.

이번 챕터에서는 인터랙티브 리베이스를 사용해 커밋을 변경하는 방법에 대해 알아 보겠습니다.

### 10.6.1 커밋 메시지 변경하기

리베이스 수행시 커밋 메시지를 변경할 수 있습니다. 아래 [그림 10-6]은 리베이스를 수행해 커밋 메시지를 변경한 모습입니다.



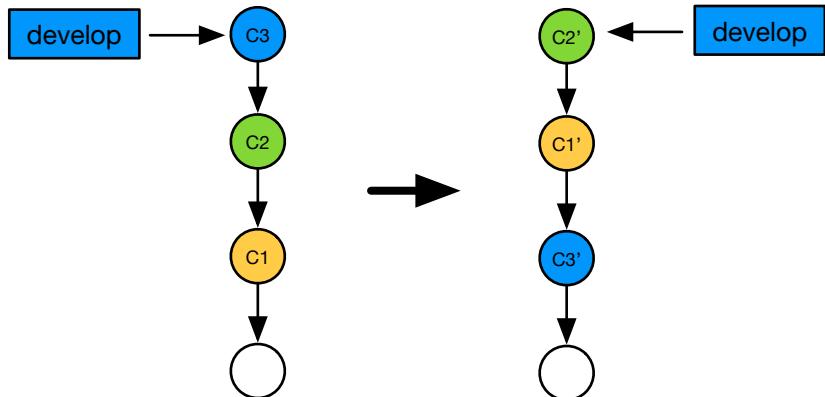
[그림 10-6] 리베이스로 커밋 메시지 변경

그림의 왼쪽은 커밋 메시지 변경 전 히스토리입니다. C3 커밋의 메시지는 “11. 자율주행 택시 개발” 이었는데 리베이스 후 커밋 메시지를 변경해 그림 오른쪽 C3’ 메시지와 같이 “11. 오토택시 개발” 이라는 메시지로 변경했습니다. 이렇게 리베이스시 커밋 메시지를 변경하고 싶을 때 인터렉티브 리베이스를 사용할 수 있습니다.

인터렉티브 리베이스를 사용하는 방법은 뒤에 10.7.2의 소스트리에서 사용하기-커밋 메시지 변경하기에서 확인하실 수 있습니다. 앞으로 설명드릴 모든 리베이스의 실제 사용 방법은 10.7의 소스트리에서 사용하기를 참조하시면 됩니다.

## 10.6.2 커밋 순서 변경하기

리베이스 수행시 커밋 메시지 순서를 변경할 수도 있습니다. 아래 [그림 10-7]은 리베이스를 수행해 커밋 순서를 변경한 모습입니다.

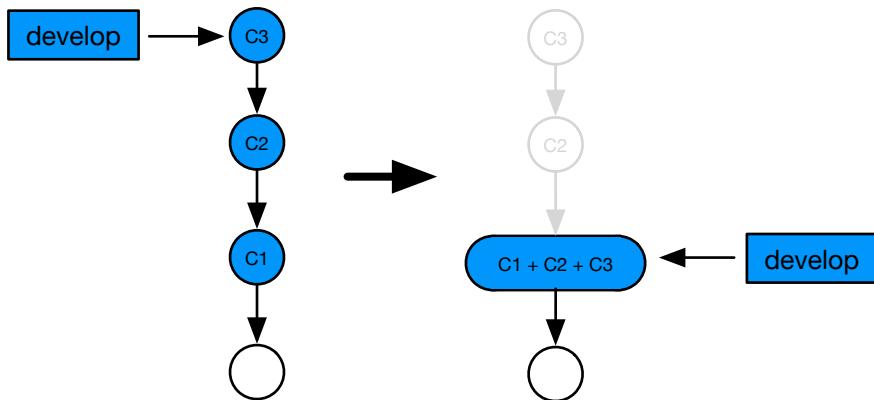


[그림 10-7] 리베이스로 커밋 순서 변경

그림의 왼쪽은 커밋 순서 변경 전 히스토리입니다. 커밋의 순서가 C1, C2, C3 였는데, C3과 C1이 비슷한 성격의 커밋이어서 커밋의 순서를 C3, C1, C2로 변경했습니다. 인터렉티브 리베이스를 사용하면 이렇게 커밋의 순서를 변경할 수 있습니다.

## 10.6.3 커밋 합치기

리베이스 수행시 여러 커밋을 하나로 합칠 수 있습니다. 아래 [그림 10-8]은 리베이스를 수행해 커밋을 합친 모습입니다.

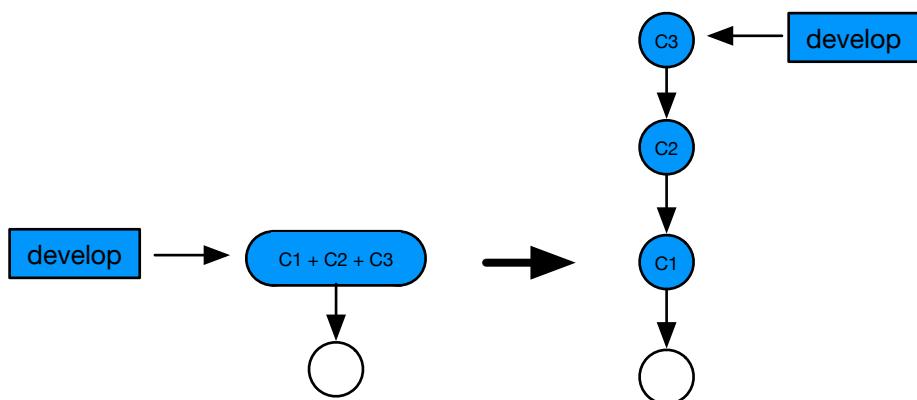


[그림 10-8] 리베이스로 커밋 합치기

그림의 왼쪽은 커밋을 합치기 전 히스토리입니다. 커밋 히스토리가 C1, C2, C3로 나눠져 있었는데, 이 커밋을 C1, C2, C3를 모두 합쳐 하나의 커밋으로 만들었습니다. 인터랙티브 리베이스를 사용하면 이렇게 여러 커밋을 하나로 합칠 수 있습니다.

#### 10.6.4 커밋 분리하기

리베이스 수행시 하나의 커밋을 여러개로 분리할 수 있습니다. 아래 [그림 10-9]는 리베이스를 수행해 커밋을 분리한 모습입니다.

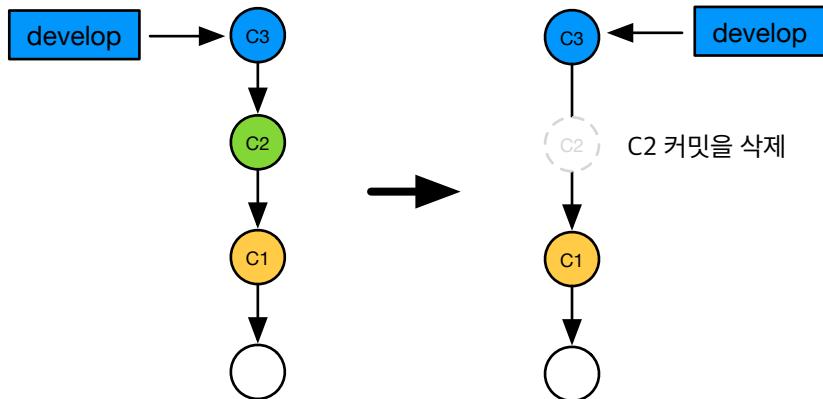


[그림 10-9] 리베이스로 커밋 분리하기

그림의 왼쪽은 커밋을 분리하기 전 히스토리입니다. 커밋이 하나였는데 이 커밋을 C1, C2, C3로 분리했습니다. 이렇게 커밋을 분리하는 방법은 리베이스만으로 가능하진 않습니다. 하나의 커밋을 분리하기 위해선 리베이스와 어멘드(Amend), 리셋 그리고 커밋까지 여러 명령을 수행해야 합니다.

### 10.6.5 커밋 삭제하기

리베이스 수행시 커밋을 삭제할 수 있습니다. 아래 [그림 10-10]는 리베이스 수행시 커밋을 삭제한 모습입니다.



[그림 10-10] 리베이스로 커밋 삭제하기

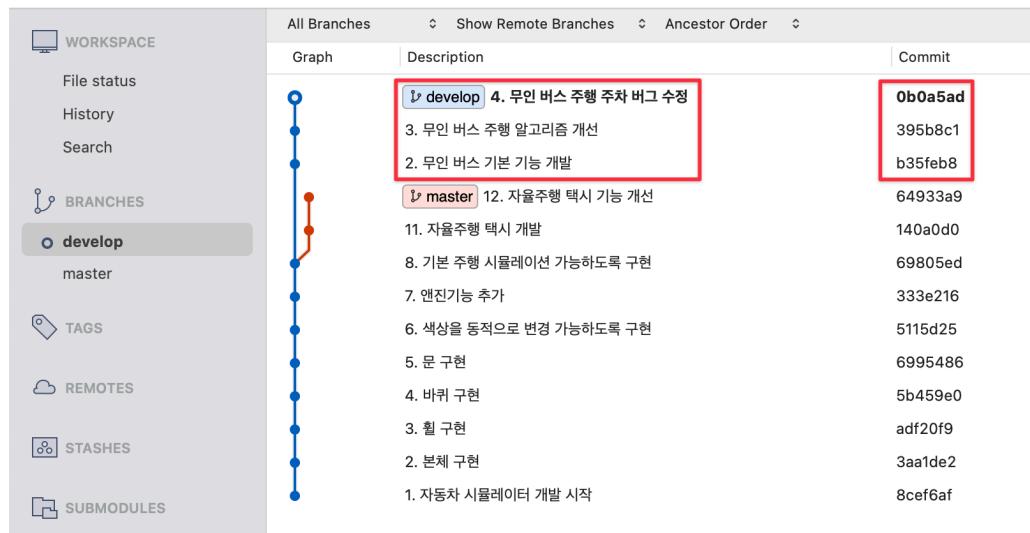
그림의 왼쪽은 커밋을 삭제하기 전 히스토리입니다. 이 히스토리에서 리베이스를 수행시 그림 오른쪽과 같이 C2 커밋을 삭제 했습니다. 인터렉티브 리베이스를 사용하면 이렇게 커밋을 삭제할 수 있습니다.

## 10.7 소스트리에서 사용하기

### 10.7.1 리베이스 기본

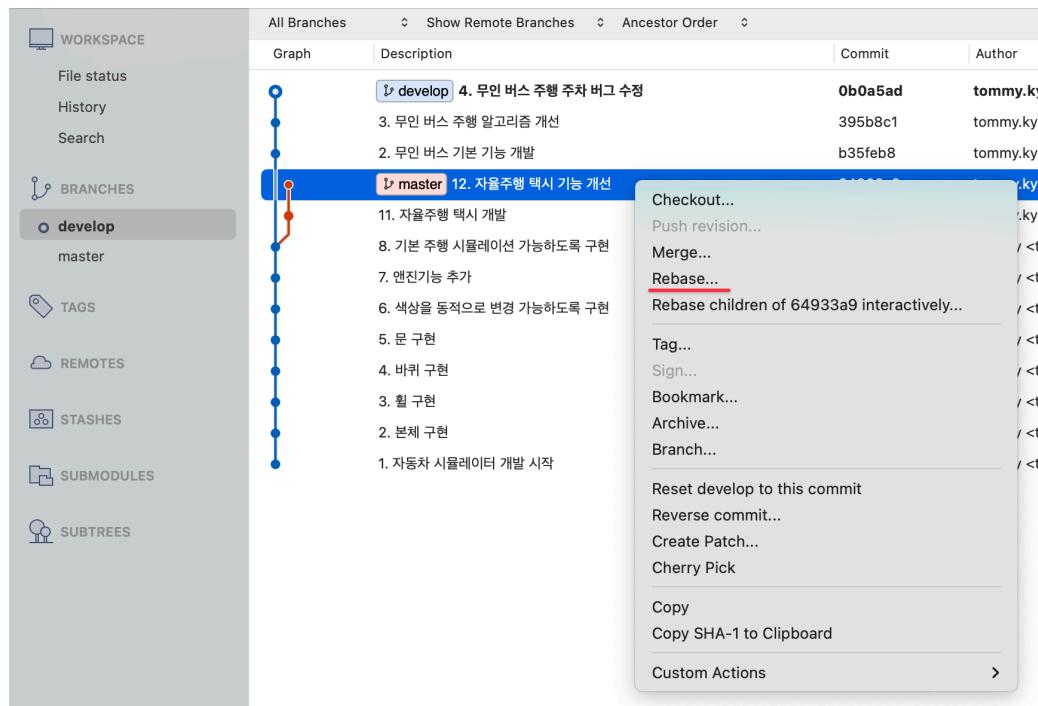
리베이스에 앞서 먼저 현재 커밋 히스토리를 살펴보겠습니다. 아래 [그림 10-11]은 현재 커밋 히스토리입니다. 현재 브랜치는 develop이고 develop과 master 두 브랜치 모두 “8. 기

본 주행 시뮬레이션 가능하도록 구현” 커밋을 공통 조상으로 갖고 있습니다. 이 후 두 브랜치 각각 새 커밋이 생성 됐습니다.



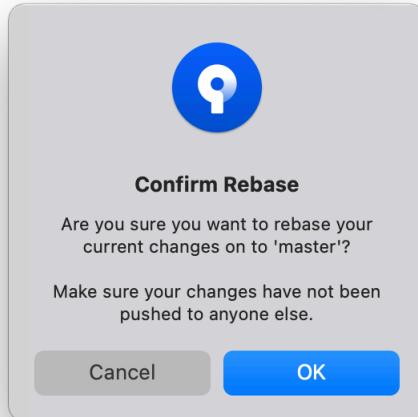
[그림 10-11] 초기 커밋 히스토리

이 상태에서 develop 브랜치를 master 브랜치에 리베이스 해 보겠습니다.



[그림 10-12] 리베이스 수행

리베이스를 수행하기 위해 master 브랜치의 가장 마지막 커밋에서 마우스 오른쪽 버튼을 누른 후 Rebase 메뉴를 선택합니다.



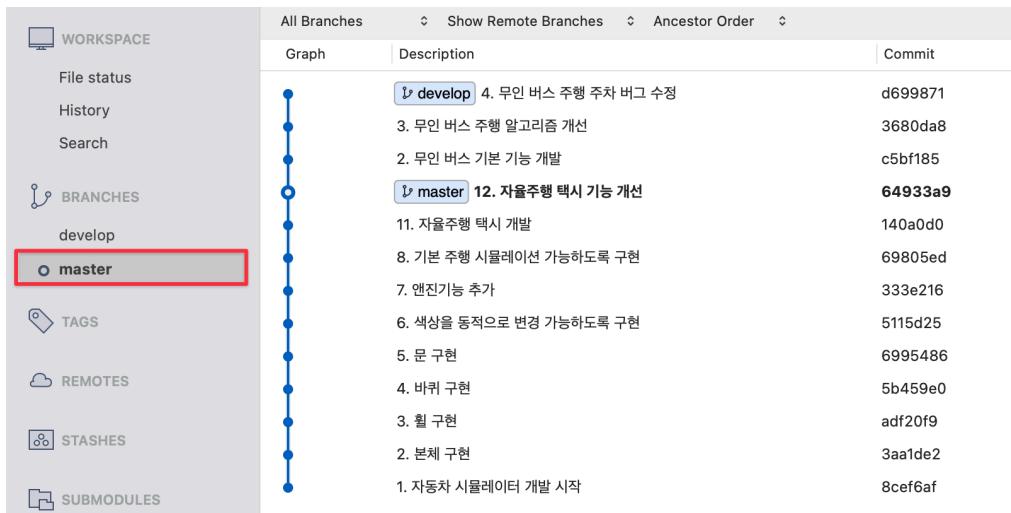
[그림 10-13] 리베이스 확인 팝업

리베이스를 수행하려고 하면 위 [그림 10-13]과 같이 확인 팝업이 나타납니다. 앞에서 설명 드린 것 같이 리베이스 하려는 변경 사항이 원격저장소에 푸시되지 않은 것인지를 확인하는 팝업입니다. OK 버튼을 눌러 리베이스를 완료합니다.

WORKSPACE		All Branches	Show Remote Branches	Ancestor Order	Commit
		Graph	Description		
	File status				
	History				
	Search				
branches	develop	○	▷ develop 4. 무인 버스 주행 주차 버그 수정 3. 무인 버스 주행 알고리즘 개선 2. 무인 버스 기본 기능 개발		d699871
	master	●	▷ master 12. 자율주행 택시 기능 개선 11. 자율주행 택시 개발 8. 기본 주행 시뮬레이션 가능하도록 구현 7. 엔진기능 추가 6. 색상을 동적으로 변경 가능하도록 구현 5. 문 구현 4. 바퀴 구현 3. 휠 구현 2. 본체 구현 1. 자동차 시뮬레이터 개발 시작		3680da8 c5bf185 64933a9 140a0d0 69805ed 333e216 5115d25 6995486 5b459e0 adf20f9 3aa1de2 8cef6af
tags					
remotes					
stashes					
submodules					

[그림 10-14] 리베이스 완료 상태

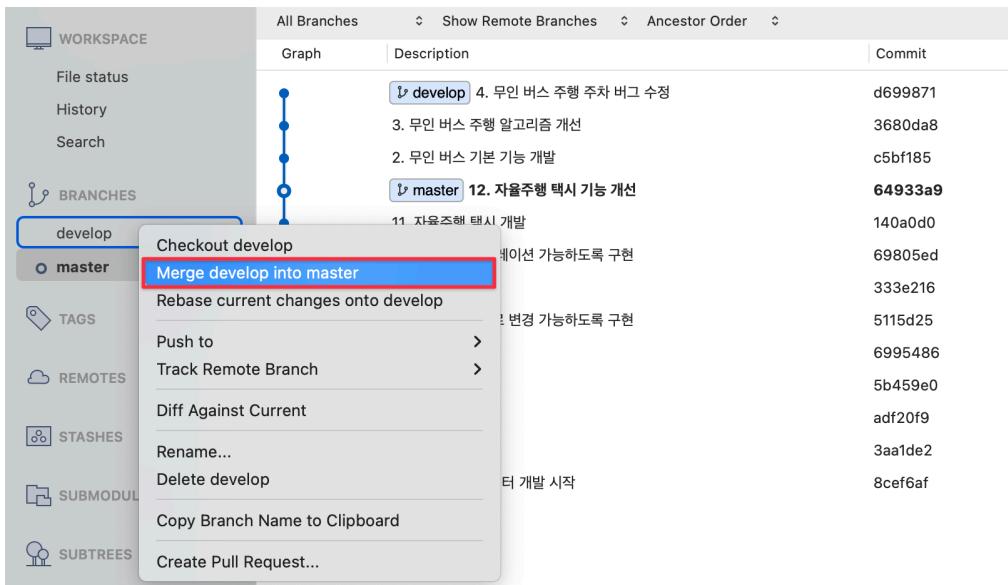
리베이스가 완료되면 위 [그림 10-14]와 같이 develop 브랜치의 내용이 master 브랜치 뒤에 추가 되고 두 브랜치는 일렬로 된 그래프로 변경됩니다. 이 시점에 리베이스는 완료된 것 이지만 아직 두 브랜치가 다 합쳐지지는 않았습니다. 그림의 커밋 히스토리를 보면 master 브랜치와 develop 브랜치가 가리키고 있는 커밋이 다른 것을 확인할 수 있습니다.



[그림 10-15] 현재 브랜치를 master 브랜치로 변경

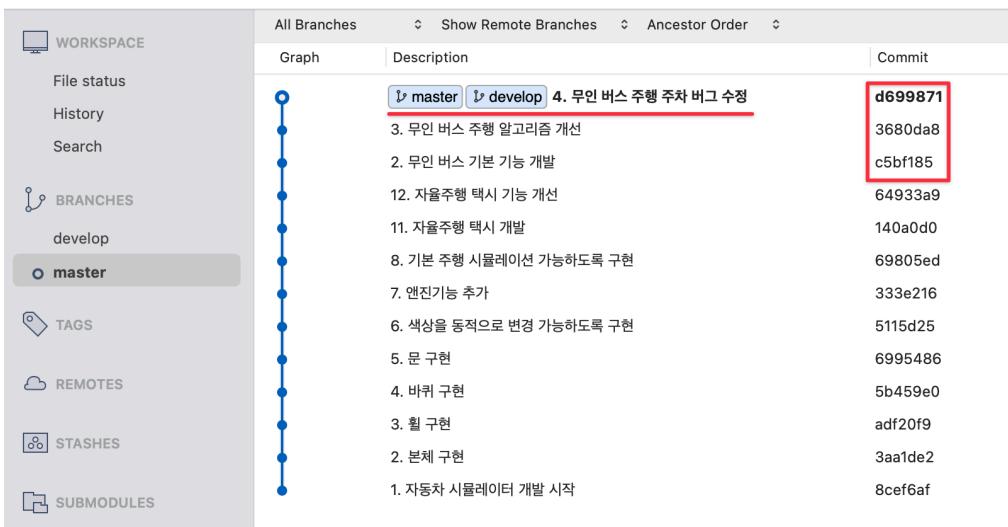
두 브랜치를 완전히 합치기 위해서는 master 브랜치의 head가 develop의 head인 “4. 무인 버스 주행 주차 버그 수정” 커밋을 가리키도록 develop의 변경 사항을 master에 머지해야 합니다. 머지를 수행하기 위해 위 [그림 10-15]와 같이 왼쪽 사이드 바에서 master 브랜치를 마우스로 더블클릭해 현재 브랜치를 master 브랜치로 변경합니다.

브랜치 변경 후 사이드바를 통해 아래 [그림 10-16]과 같이 develop 브랜치를 master 브랜치에 머지 합니다.



[그림 10-16] develop 브랜치를 master 브랜치에 머지

머지를 완료하면 아래 [그림 10-17]과 같이 master 브랜치가 develop 브랜치와 동일한 커밋을 가리키게 됩니다. 이것으로 리베이스와 머지가 모두 완료됐습니다.



[그림 10-17] 머지 후 커밋 히스토리

다시 말씀드리면 리베이스가 수행된 develop의 커밋인 “2. 무인 버스 기본 기능 개발”, “3. 무인 버스 주행 알고리즘 개선”, “4. 무인 버스 주행 주차 버그 수정” 이 세 개의 커밋은 비록 리베이스 하기 전 커밋과 메시지와 변경 사항이 같지만 리베이스 된 커밋은 리베이스 되기 전 커밋과는 완전히 다른 커밋이라는 것입니다.

위 [그림 10-17]에 커밋 아이디에 박스로 표시한 부분(빨강색)을 보면 이 세 커밋의 커밋 아이디를 확인할 수 있습니다. 커밋 아이디는 차례로 c5bf185, 3680da8, d699871입니다. 이 커밋 아이디는 리베이스 되기 전 상태인 [그림 10-11]에서는 각각 b35feb8, 395b8c1, 0b0a5ad였습니다. 리베이스 후 커밋의 해시가 달라진 것을 확인할 수 있습니다. 리베이스 후 커밋이 재생성됐기 때문에 커밋의 해시가 달라진 켰습니다. 이렇게 리베이스를 수행하게 되면 커밋을 재생성하기 때문에 리베이스 된 커밋은 리베이스 전 커밋과 다른 커밋이라는 것을 기억하시면 좋을 것 같습니다.

이상 기본적인 리베이스를 소스트리에서 사용하는 방법을 알아 보았습니다. 다음은 인터랙티브 리베이스를 이용해 리베이스를 다양하게 활용하는 방법에 대해 알아 보겠습니다.

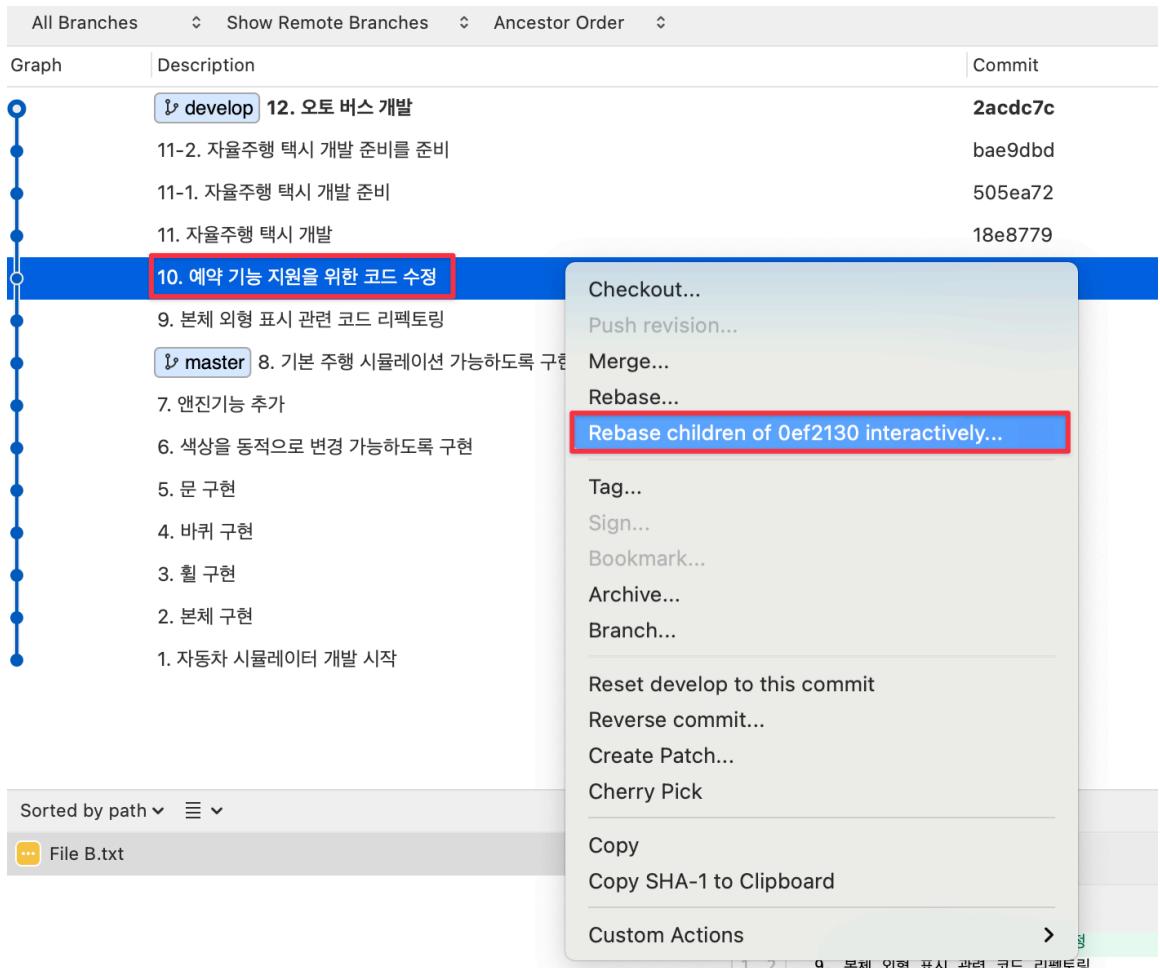
### 10.7.2 커밋 메시지 변경하기

인터랙티브 리베이스를 사용해 커밋 메시지를 변경해 보겠습니다. 변경하고 싶은 커밋은 아래 [그림 10-18]의 “11. 자율주행 택시 개발”이라는 커밋 메시지의 커밋입니다.

All Branches	Show Remote Branches	Ancestor Order	Commit
Graph	Description		
●	develop   12. 오토 버스 개발		2acdc7c
●	11-2. 자율주행 택시 개발 준비를 준비		bae9dbd
●	11-1. 자율주행 택시 개발 준비		505ea72
○	11. 자율주행 택시 개발		18e8779
●	10. 예약 기능 지원을 위한 코드 수정		0ef2130
●	9. 본체 외형 표시 관련 코드 리펙토링		9b14d2d
●	master   8. 기본 주행 시뮬레이션 가능하도록 구현		69805ed
●	7. 앤진기능 추가		333e216
●	6. 색상을 동적으로 변경 가능하도록 구현		5115d25
●	5. 문 구현		6995486
●	4. 바퀴 구현		5b459e0
●	3. 훨 구현		adf20f9
●	2. 본체 구현		3aa1de2
●	1. 자동차 시뮬레이터 개발 시작		8cef6af

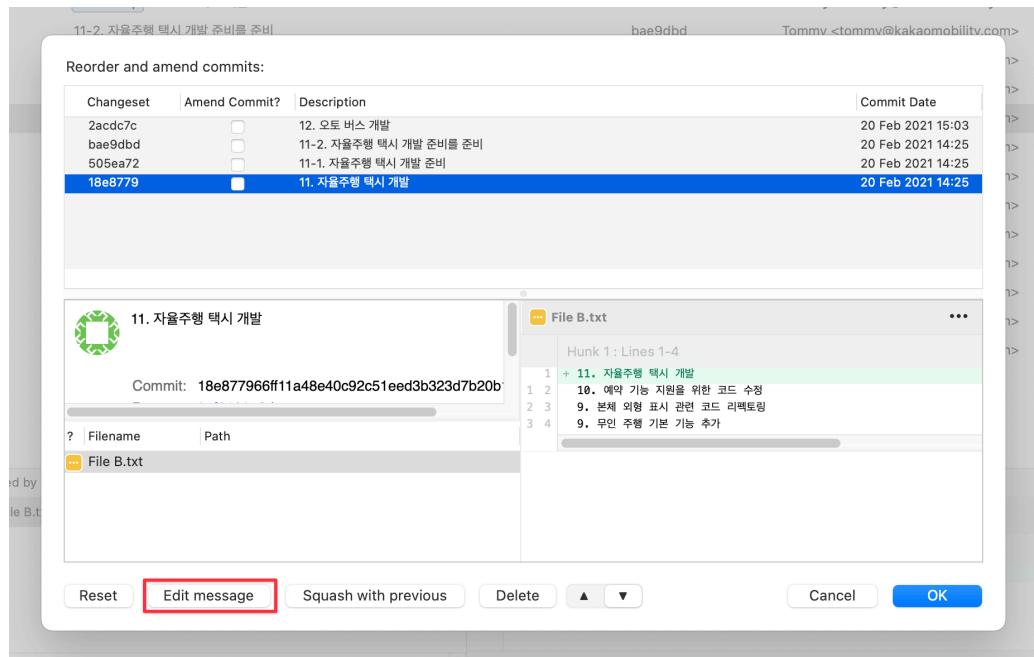
[그림 10-18] 커밋 메시지를 변경할 커밋

이 커밋의 메시지를 수정하기 위해 아래 [그림 10-19]와 같이 이 커밋의 바로 직전 커밋인 부모 커밋을 선택 후 마우스 오른쪽 버튼을 클릭합니다.



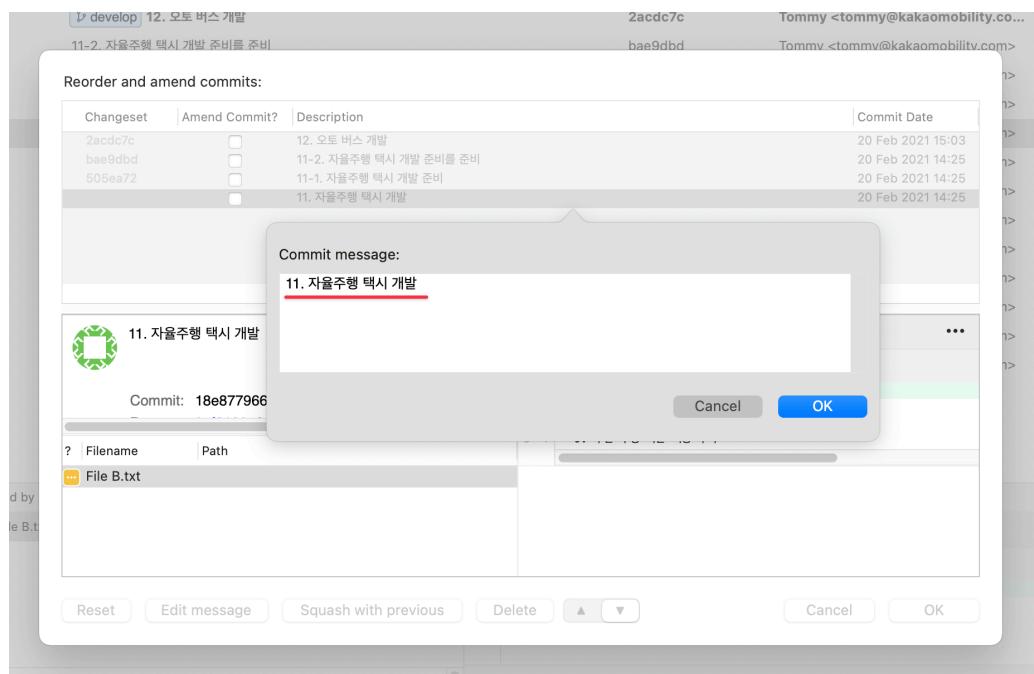
[그림 10-19] 인터렉티브 리베이스 수행

컨텍스트 메뉴에서 “Rebase children of (SHA-1) interactively” 아이템을 선택하면 아래 [그림 10-20]와 같이 인터렉티브 리베이스 창이 나타납니다.



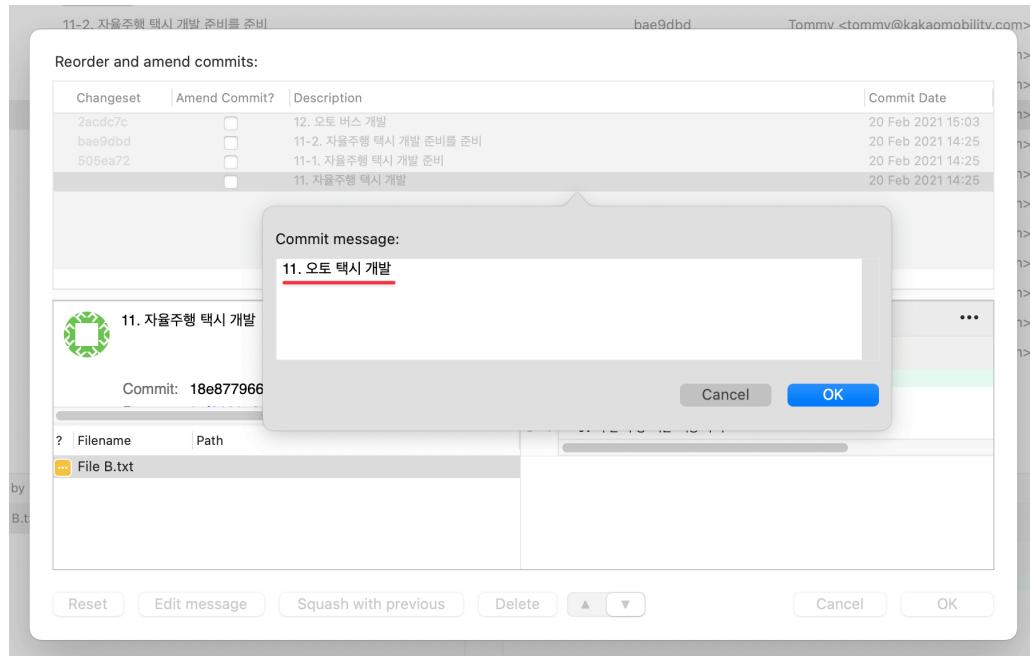
[그림 10-20] 인터랙티브 리베이스 중 메시지 수정 선택

하고 싶은 작업이 “11. 자율주행 택시 개발” 커밋 메시지를 변경하는 것이기 때문에 이 메시지를 선택하고 하단의 Edit message 버튼을 선택합니다.



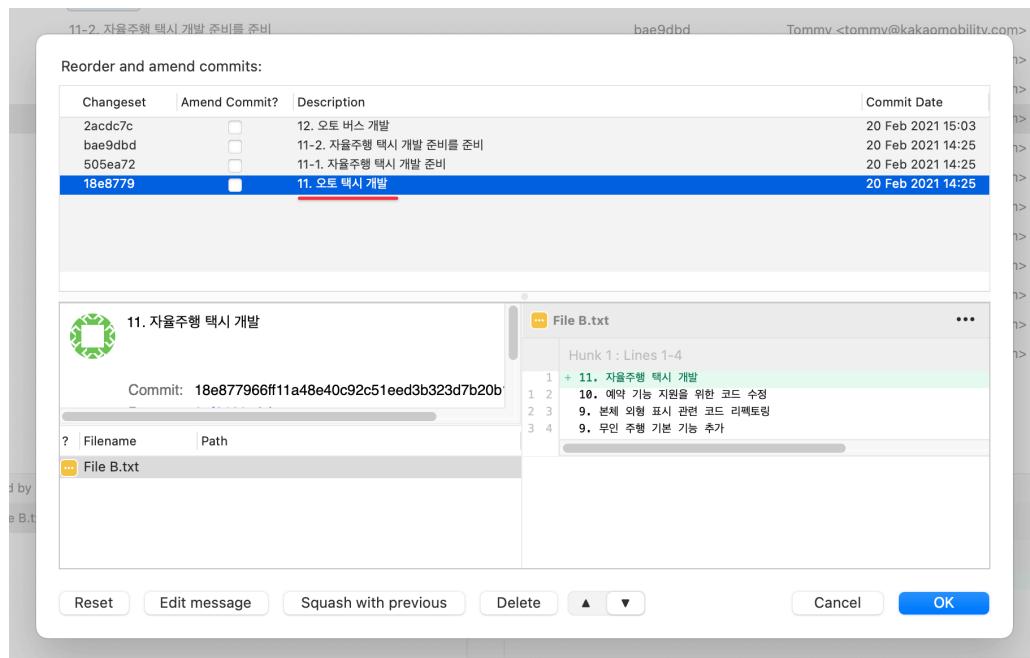
[그림 10-21] 커밋 메시지 변경 팝업

메시지 수정 버튼을 선택하면 위 [그림 10-21]와 같이 선택한 커밋의 현재 메시지를 수정할 수 있는 팝업이 나타납니다.



[그림 10-22] 커밋 메시지 수정

- 이) 메시지를 위 [그림 10-22]와 같이 “11. 오토 택시 개발”이라는 메시지로 변경한 후 OK 버튼을 선택합니다.



[그림 10-23] 변경된 커밋 메시지

메시지 변경 후 인터렉티브 리베이스 창에는 위 [그림 10-23]과 같이 변경한 메시지가 반영된 것을 확인할 수 있습니다. OK 버튼을 눌러 인터렉티브 리베이스를 완료 합니다.

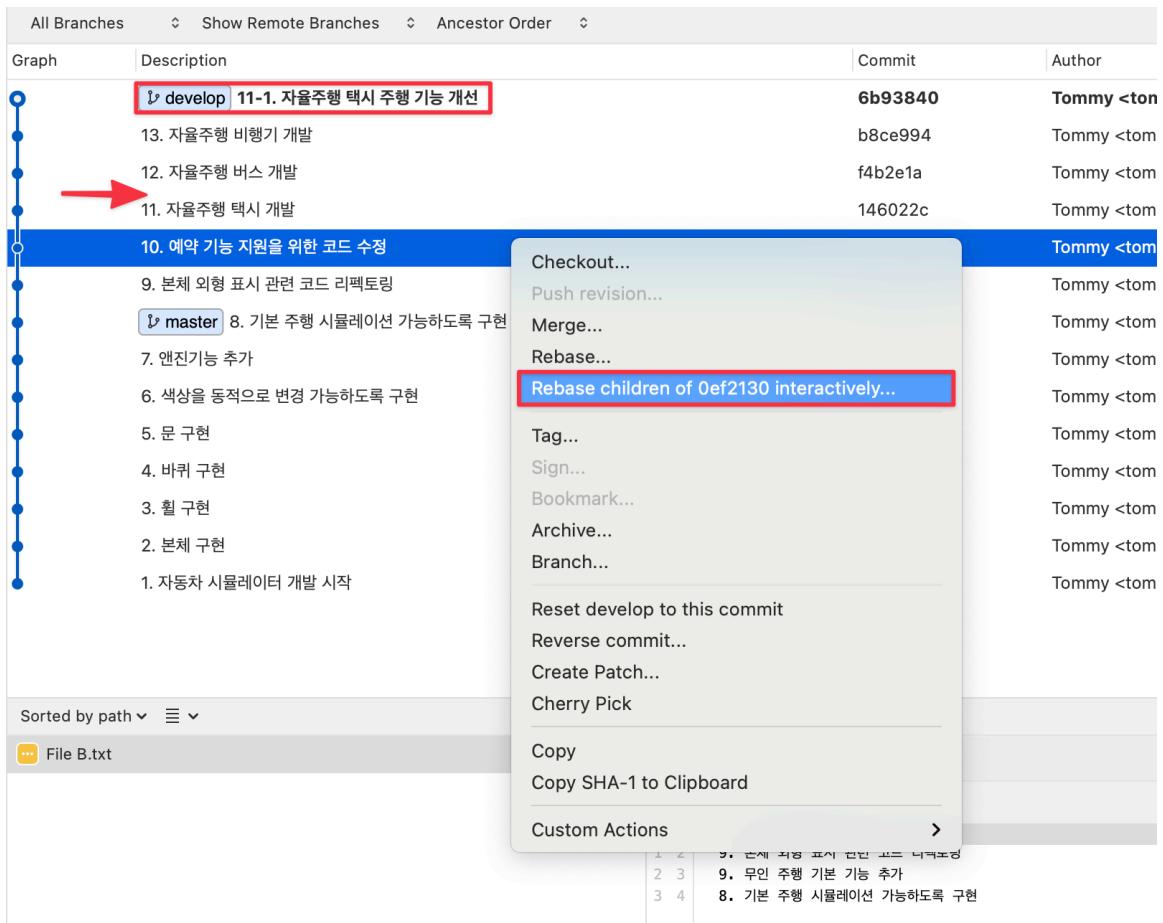
All Branches	Description	Commit
Graph	▶ develop 12. 오토 버스 개발	80ee57b
	11-2. 자율주행 택시 개발 준비를 준비	37c99f3
	11-1. 자율주행 택시 개발 준비	109cdaa
	11. 오토 택시 개발	23f97de
	10. 예약 기능 지원을 위한 코드 수정	0ef2130
	9. 본체 외형 표시 관련 코드 리팩토링	9b14d2d
	▶ master 8. 기본 주행 시뮬레이션 가능하도록 구현	69805ed
	7. 엔진기능 추가	333e216
	6. 색상을 동적으로 변경 가능하도록 구현	5115d25
	5. 문 구현	6995486
	4. 바퀴 구현	5b459e0
	3. 휠 구현	adf20f9
	2. 본체 구현	3aa1de2
	1. 자동차 시뮬레이터 개발 시작	8cef6af

[그림 10-24] 리베이스 후 커밋메시지가 변경된 모습

커밋 히스토리를 확인하면 위 [그림 10-24]와 같이 커밋메시지가 변경된 것을 확인할 수 있습니다. 지금까지 인터렉티브 리베이스를 사용해 커밋 메시지를 변경하는 방법에 대해 살펴봤습니다.

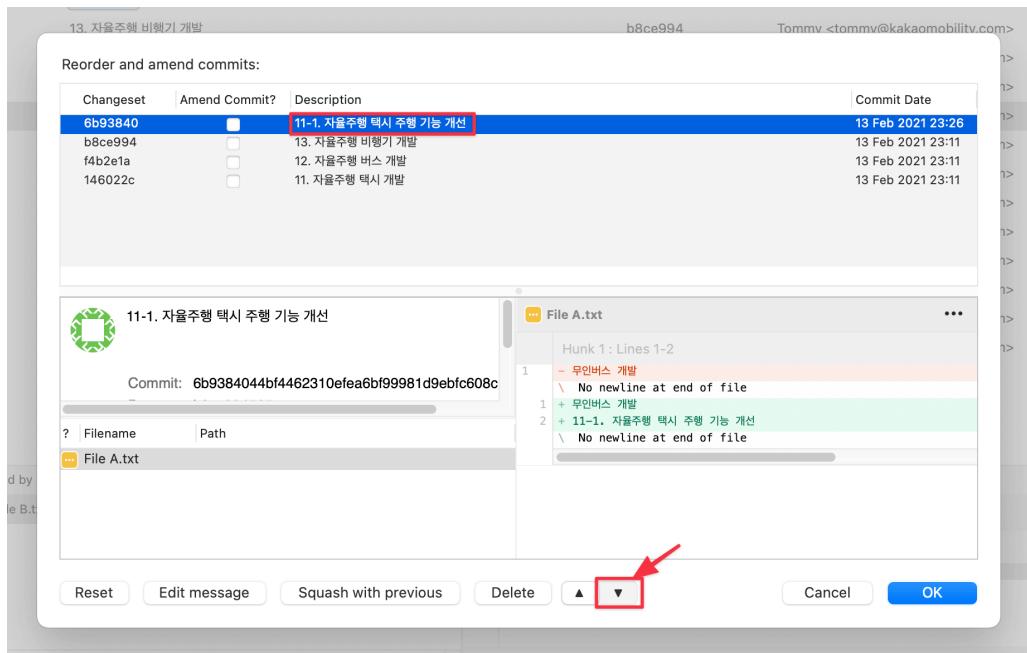
### 10.7.3 커밋 순서 변경하기

인터렉티브 리베이스를 사용해 커밋의 순서를 변경해 보겠습니다. 순서를 변경할 커밋은 아래 [그림 10-25]에서 빨간 사각형으로 표시한 “11-1. 자율주행 택시 주행 기능 개선”이라는 커밋 메시지를 갖는 커밋입니다.



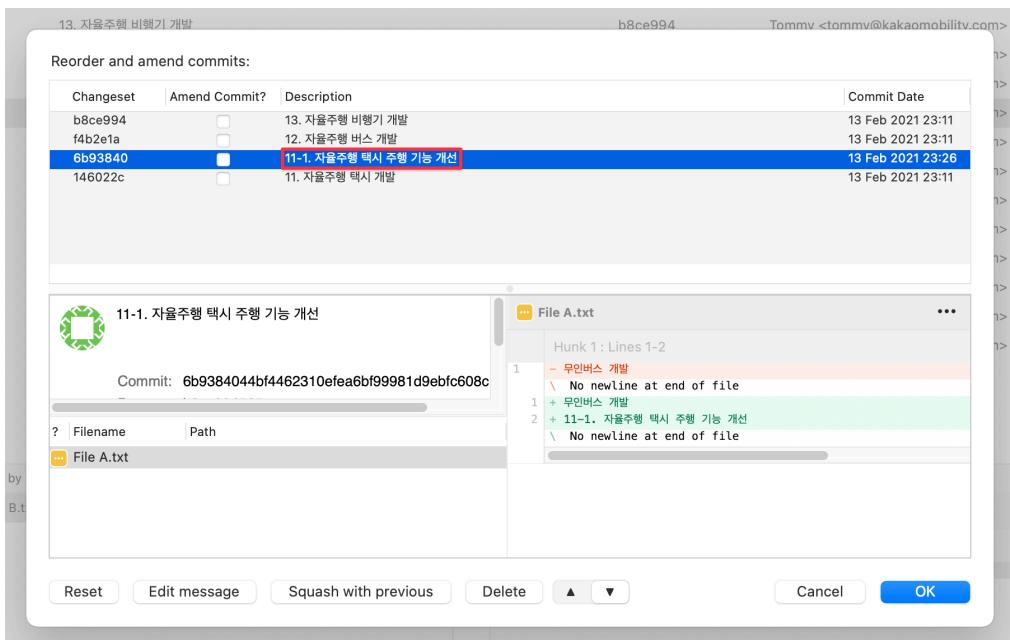
[그림 10-25] 커밋의 순서 변경

이 커밋을 그림의 빨간 화살표가 가리키는 “11. 자율주행 택시 개발” 뒤로 위치를 변경하려고 합니다. 위치를 변경하기 위해 “10. 예약 기능 지원을 위한 코드 수정” 커밋을 선택 후 인터랙티브 리베이스를 수행합니다.



[그림 10-26] 커밋의 순서 변경

이후 나타나는 인터랙티브 리베이스 창에서 위 [그림10-26]과 같이 “11-1. 자율주행 택시 주행 기능 개선” 커밋을 선택한 후 창 하단의 아래 화살표를 반복적으로 클릭해 커밋을 원하는 위치로 이동시킵니다.



[그림 10-27] 커밋의 순서 변경 완료

위 [그림 10-27]와 같이 커밋이 “11. 자율주행 택시 개발”에 위치하면 OK 버튼을 선택해 리베이스를 완료합니다.

All Branches	Show Remote Branches	Ancestor Order	
Graph	Description		Commit
o	develop 13. 자율주행 비행기 개발		f73b6f5
o	12. 자율주행 버스 개발		e13c8c4
o	11-1. 자율주행 택시 주행 기능 개선		27ec862
o	11. 자율주행 택시 개발		146022c
o	10. 예약 기능 지원을 위한 코드 수정		0ef2130
o	9. 본체 외형 표시 관련 코드 리펙토링		9b14d2d
o	master 8. 기본 주행 시뮬레이션 가능하도록 구현		69805ed
o	7. 엔진기능 추가		333e216
o	6. 색상을 동적으로 변경 가능하도록 구현		5115d25
o	5. 문 구현		6995486
o	4. 바퀴 구현		5b459e0
o	3. 휠 구현		adf20f9
o	2. 본체 구현		3aa1de2
o	1. 자동차 시뮬레이터 개발 시작		8cef6af

[그림 10-28] 커밋의 순서가 변경된 모습

커밋 히스토리를 확인하면 위 [그림 10-28]과 같이 커밋의 순서가 변경된 것을 확인할 수 있습니다.

지금까지 인터렉티브 리베이스를 사용해 커밋의 순서를 변경하는 방법에 대해 살펴봤습니다.

#### 10.7.4 커밋 합치기

인터렉티브 리베이스를 사용해 커밋을 합쳐보겠습니다. 합치려고 하는 커밋은 아래 [그림 10-29]에 박스(빨간색)로 표시한 세 개의 커밋입니다.

All Branches	Description	Commit
Graph		
o	develop 13. 오토에어 개발	0d9be90
o	12. 자율주행 버스 개발	09d42e3
o	11. 자율주행 택시 개발	0ac48f9
o	10. 예약 기능 지원을 위한 코드 수정	0ef2130
o	9. 본체 외형 표시 관련 코드 리팩토링	9b14d2d
o	master 8. 기본 주행 시뮬레이션 가능하도록 구현	69805ed
o	7. 앤진기능 추가	333e216
o	6. 색상을 동적으로 변경 가능하도록 구현	5115d25
o	5. 문 구현	6995486
o	4. 바퀴 구현	5b459e0
o	3. 휠 구현	adf20f9
o	2. 본체 구현	3aa1de2
o	1. 자동차 시뮬레이터 개발 시작	8cef6af

[그림 10-29] 합치기 전 커밋 히스토리

All Branches	Description	Commit
Graph		
o	develop 13. 오토에어 개발	0d9be90
o	12. 자율주행 버스 개발	09d42e3
o	11. 자율주행 택시 개발	0ac48f9
o	10. 예약 기능 지원을 위한 코드 수정	0ef2130
o	9. 본체 외형 표시 관련 코드 리팩토링	
o	master 8. 기본 주행 시뮬레이션 가능하도록 구현	
o	7. 앤진기능 추가	
o	6. 색상을 동적으로 변경 가능하도록 구현	
o	5. 문 구현	
o	4. 바퀴 구현	
o	3. 휠 구현	
o	2. 본체 구현	
o	1. 자동차 시뮬레이터 개발 시작	

Checkout...

Push revision...

Merge...

Rebase...

**Rebase children of 0ef2130 interactively...**

---

Tag...

Sign...

Bookmark...

Archive...

Branch...

---

Reset develop to this commit

Reverse commit...

Create Patch...

Cherry Pick

---

Copy

Copy SHA-1 to Clipboard

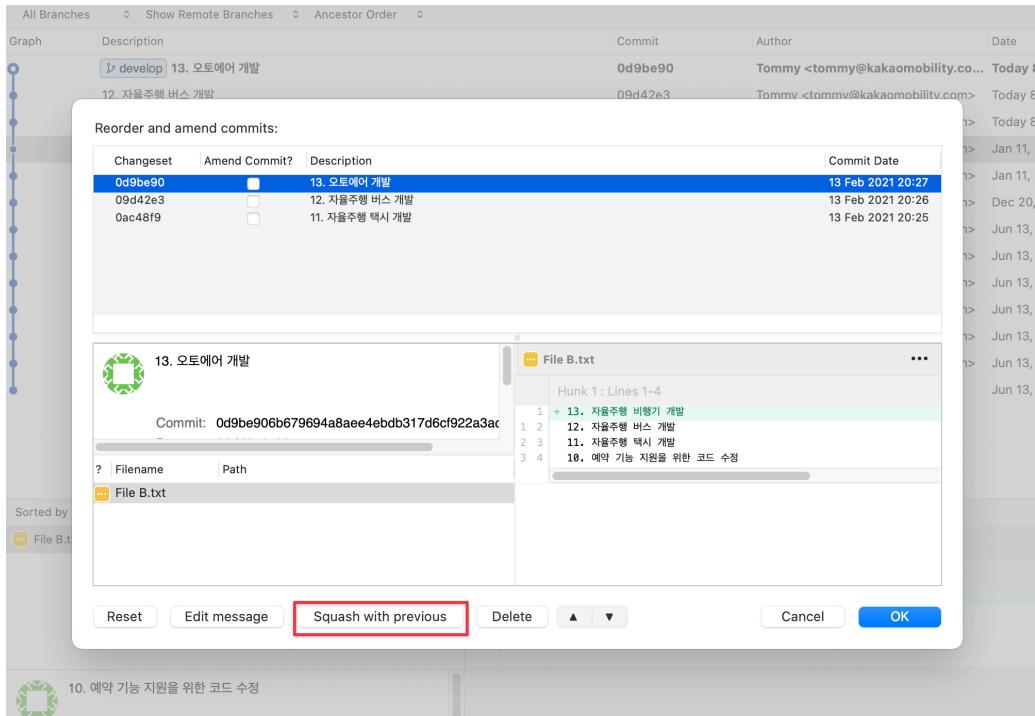
---

Custom Actions >

1	+ 10. 예약 기능 지원을 위한 코드 수정
2	9. 본체 외형 표시 관련 코드 리팩토링
3	9. 무인 주행 기본 기능 추가
4	8. 기본 주행 시뮬레이션 가능하도록 구현

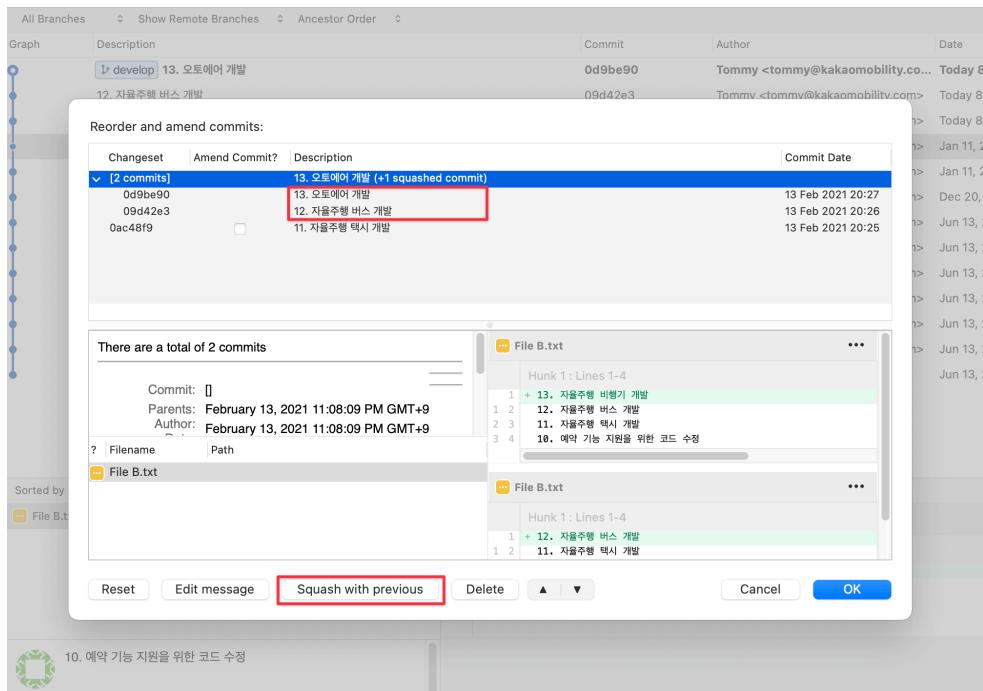
[그림 10-30] 인터랙티브 리베이스 수행

커밋을 합치기 위해 위 [그림 10-30]과 같이 합치려는 커밋 중 가장 앞선 커밋인 10번 커밋의 부모 커밋을 선택후 인터렉티브 리베이스를 수행합니다.



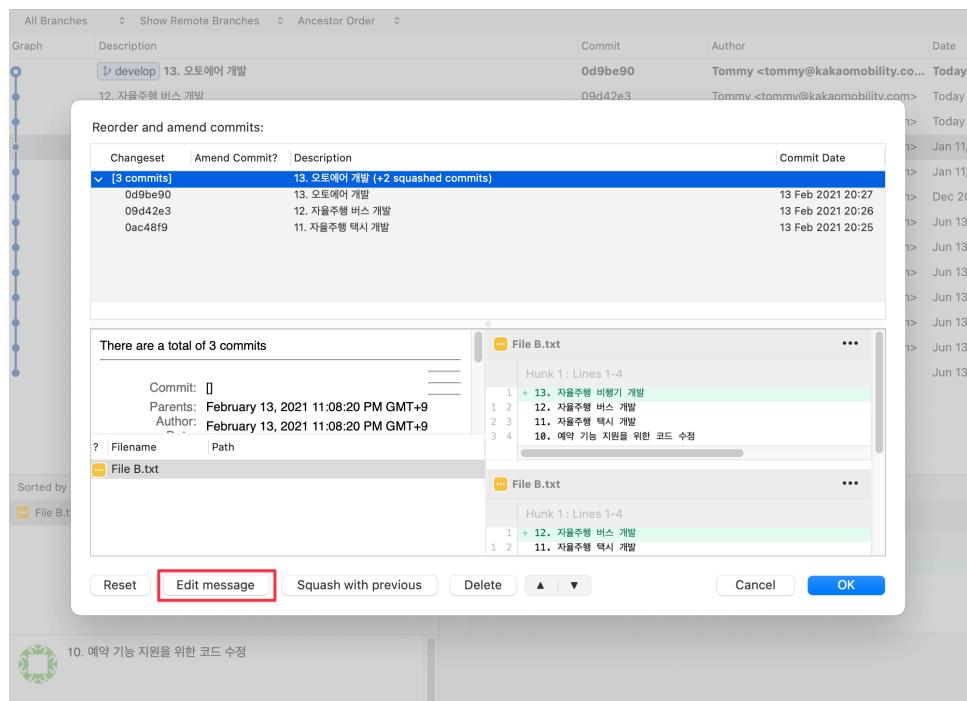
[그림 10-31] Squash with previous 선택

이후 나타나는 인터렉티브 리베이스 창에서 가장 마지막 커밋인 13번 커밋을 선택 후 하단의 Squash with previous 버튼을 선택합니다.



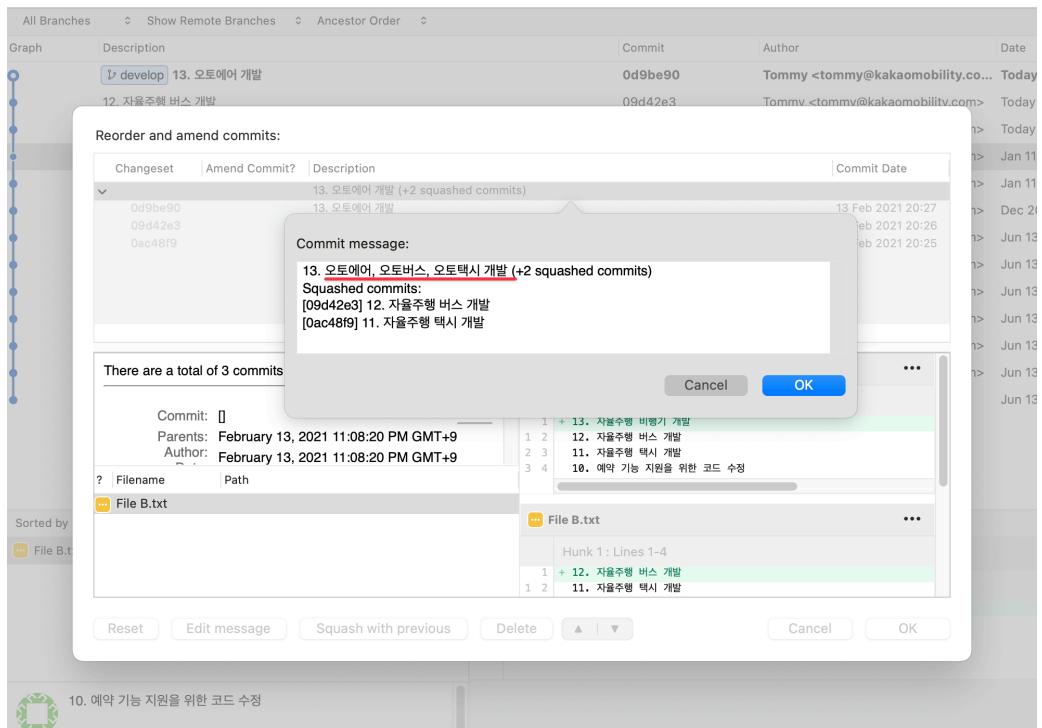
[그림 10-32] 스쿼시 된 커밋

이렇게 스쿼시(Squash)를 수행하면 위 [그림10-32]와 같이 13번 커밋과 이 커밋의 이전 커밋인 12번 커밋이 합쳐지게 됩니다. 11번 커밋도 합쳐야 하기 때문에 Squash with previous 버튼을 한번 더 눌러 11번 커밋도 합칩니다.



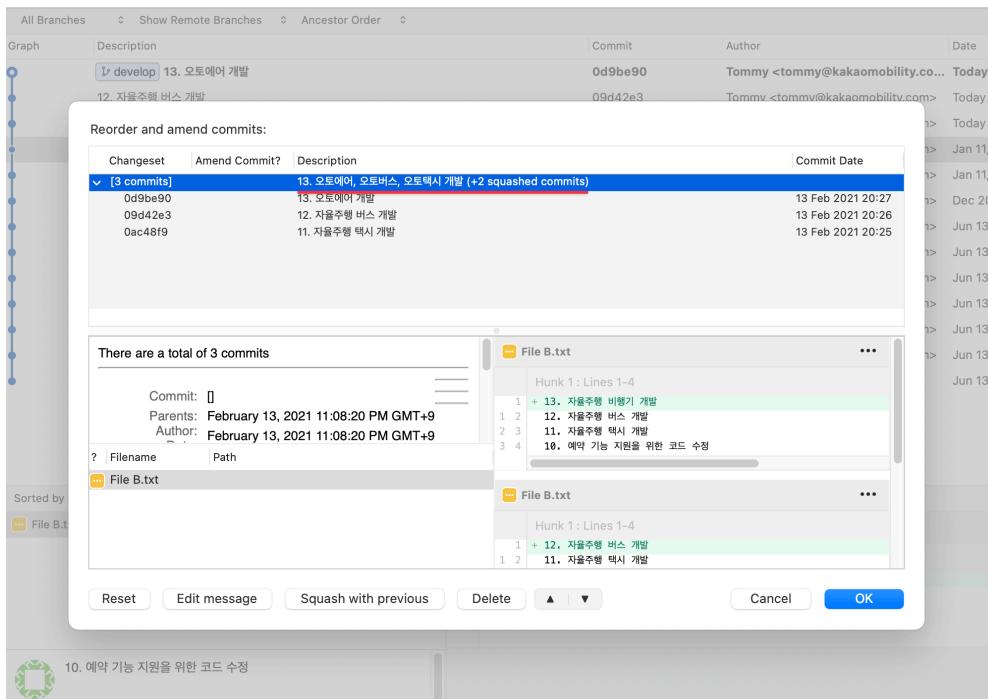
[그림 10-33] 스쿼시 된 커밋의 메시지 변경

위 [그림 10-33]은 11에서 13번 까지 세 커밋이 하나로 모두 합쳐진 모습입니다. 세 커밋을 합친 커밋 메시지 뒤에는 자동으로 (+2 squashed commits)라는 메시지가 추가돼 이 커밋에 추가로 합쳐진 커밋 메시지가 몇 개인지 알려줍니다. 스쿼시로 생성된 커밋의 메시지를 변경하기 위해 Edit message 버튼을 선택합니다. 이 작업은 선택적인 작업으로 커밋 메시지 변경 없이 커밋을 합치기만 하고 싶을 때는 이 단계를 생략해도 됩니다.



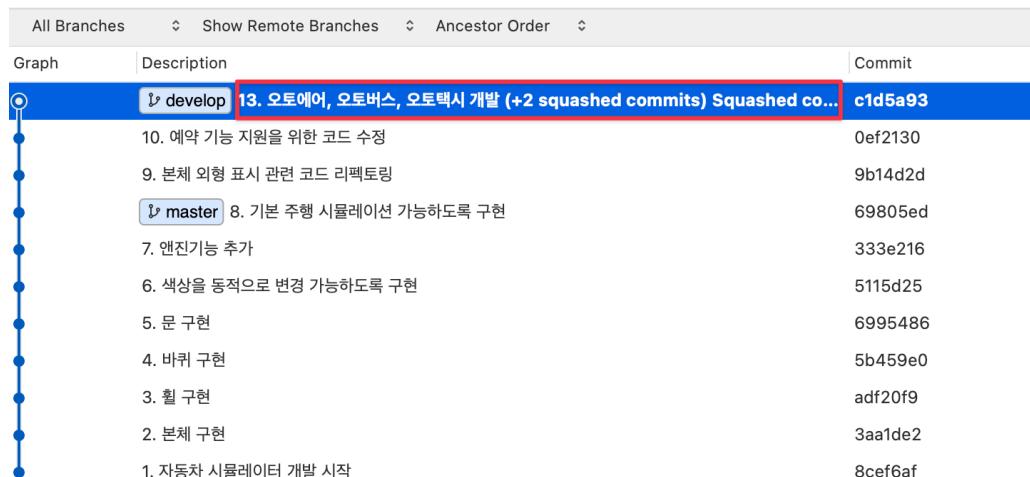
[그림 10-34] 메시지 변경 후 커밋

메시지에 “오토버스, 오토택시 개발” 내용을 추가해 스쿼시 된 커밋의 메시지를 포함하도록 메시지를 변경합니다.



[그림 10-35] 스쿼시와 커밋 메시지 변경이 완료된 모습

메시지 수정이 완료되었고 OK 버튼을 선택해 리베이스를 완료합니다.



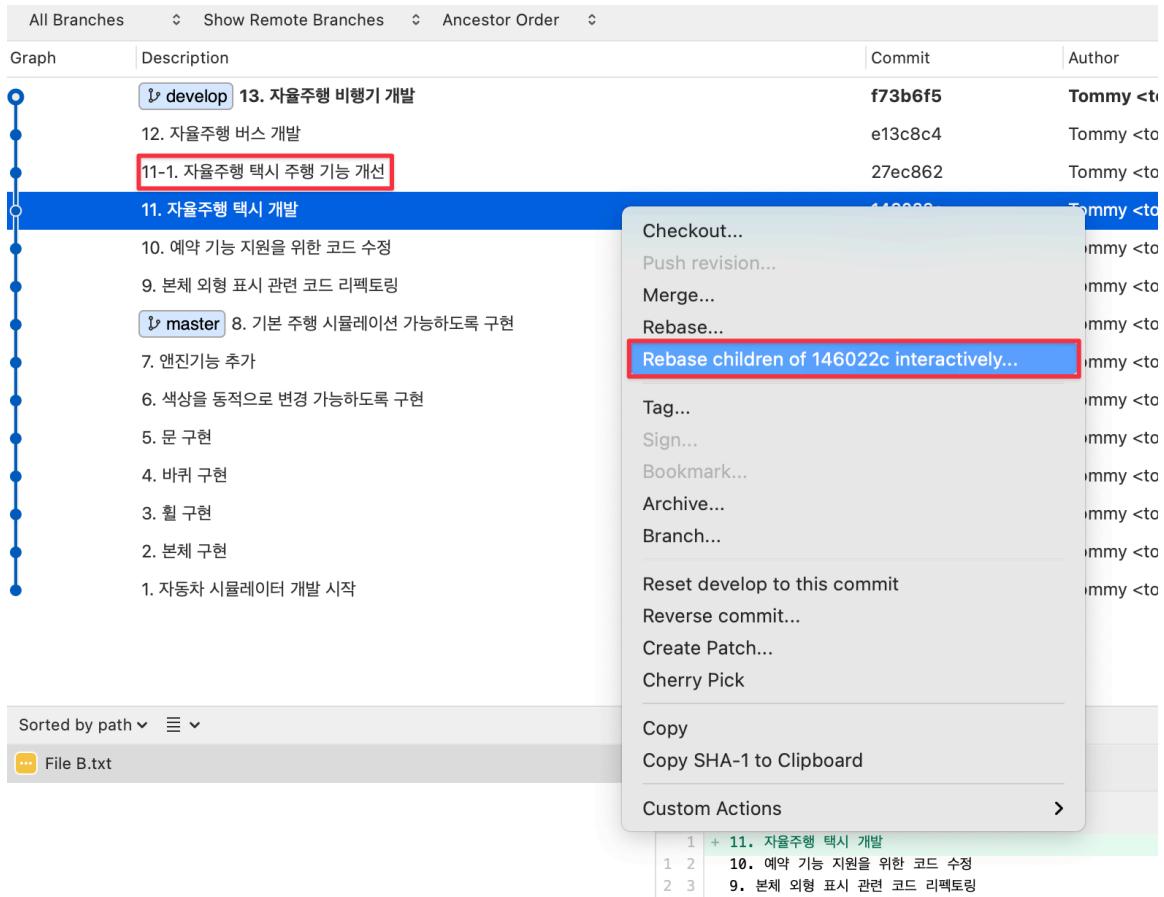
[그림 10-36] 세 개의 커밋이 하나로 합쳐진 모습

리베이스가 완료되면 위 [그림 10-36]과 같이 커밋 히스토리에서 세개였던 커밋이 한 개로 변경된 것을 확인할 수 있습니다.

지금까지 인터레티브 리베이스를 사용해 커밋을 합치는 방법에 대해 살펴봤습니다.

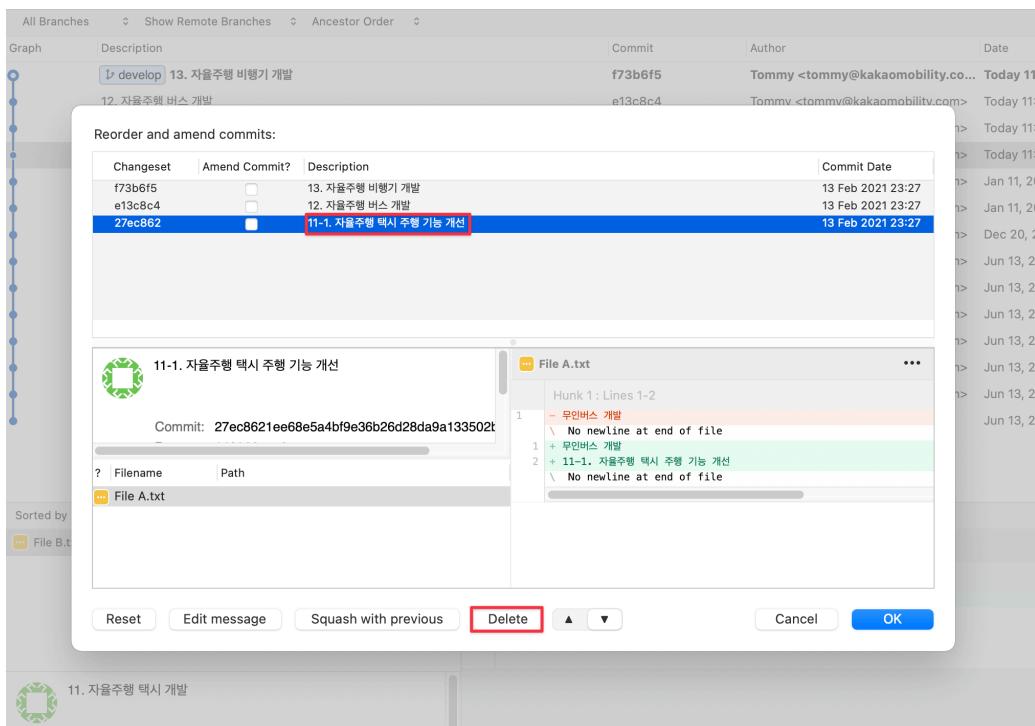
## 10.7.5 커밋 삭제하기

인터렉티브 리베이스를 사용해 커밋을 삭제해 보겠습니다. 삭제할 커밋은 아래 [그림 10-37]에 빨간색 상자로 표시한 “11-1. 자율주행 택시 주행 기능 개선” 커밋입니다.



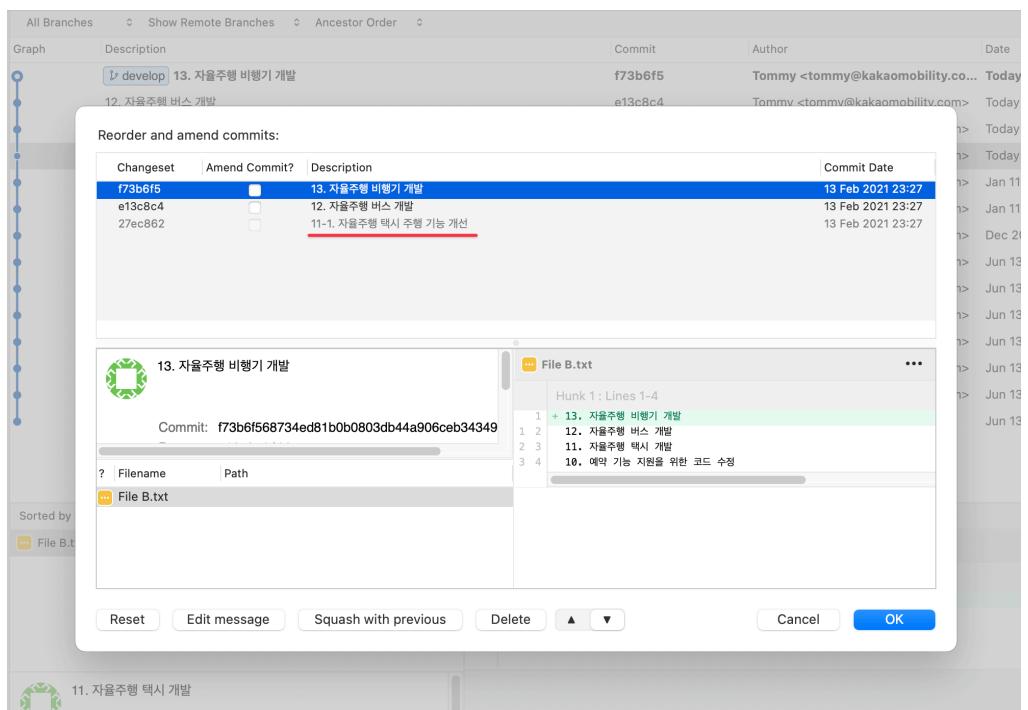
[그림 10-37] 커밋 삭제 전 커밋 히스토리

삭제하려는 커밋의 부모 커밋을 선택 후 인터랙티브 리베이스를 수행합니다.



[그림 10-38] 인터렉티브 리베이스에서 커밋 삭제할 커밋 선택

이후 나타나는 인터렉티브 리베이스 창에서 가장 마지막 커밋인 11-1번 커밋을 선택 후 하단의 Delete 버튼을 선택합니다.



[그림 10-39] 삭제 된 커밋

삭제를 수행하면 위 [그림 10-39]와 같이 리베이스 창에서 커밋이 삭제된 것을 확인할 수 있습니다. OK 버튼을 선택해 리베이스를 완료합니다.

All Branches	Graph	Description	Commit
		▶ develop 13. 자율주행 비행기 개발	186bbd9
		12. 자율주행 버스 개발	236f37c
		11. 자율주행 택시 개발	146022c
		10. 예약 기능 지원을 위한 코드 수정	0ef2130
		9. 본체 외형 표시 관련 코드 리팩토링	9b14d2d
		▶ master 8. 기본 주행 시뮬레이션 가능하도록 구현	69805ed
		7. 엔진기능 추가	333e216
		6. 색상을 동적으로 변경 가능하도록 구현	5115d25
		5. 문 구현	6995486
		4. 바퀴 구현	5b459e0
		3. 훨 구현	adf20f9
		2. 본체 구현	3aa1de2
		1. 자동차 시뮬레이터 개발 시작	8cef6af

[그림 10-40] 리베이스 후 커밋이 삭제된 모습

커밋 히스토리를 확인하면 삭제한 커밋이 커밋 히스토리에서 사라진 것을 확인할 수 있습니다.

지금까지 인터랙티브 리베이스를 사용해 커밋을 삭제하는 방법에 대해 살펴봤습니다.

## 10.8 정리

이번 챕터에서 배운 내용을 정리해 보겠습니다.

- 리베이스란 브랜치의 base를 변경해 브랜치를 합치는 방법이다.
- 리베이스시 원본 커밋과 동일한 변경사항을 갖는 커밋을 다시 생성한다.
- 리베이스 전 커밋과 리베이스 후 커밋은 다른 커밋이기 때문에 일반적으로 리베이스는 공용저장소에 푸시하지 않은 커밋을 대상으로 수행해야 한다.
- 머지와 리베이스의 차이는 리베이스는 머지와 달리 커밋 히스토리를 변경한다. 리베이스 시 충돌이 발생하면 여러 번 나눠서 충돌을 해결해야 하는 경우가 생긴다. 리베이스를 사용하면 머지를 사용해 브랜치를 합치는 것보다 간결한 커밋 히스토리를 만들 수 있다.
- 인터렉티브 리베이스를 활용하면 커밋 메시지를 변경하거나, 커밋의 순서를 변경하거나, 커밋을 합치거나, 커밋을 분리하거나, 커밋을 삭제할 수 있다.

이제 리베이스에 대해 자신감이 생기셨나요? 이것으로 Git의 주요 명령에 대해서는 다 살펴봤습니다.

다음 챕터에서는 협업하는데 필수적인 원격저장소에 대해 알아보겠습니다. 준비되셨나요? 그럼 출발! 😊