

오픈소스sw의 이해

Lecture #5: Branch

Software Engineering laboratory
Yeungnam University



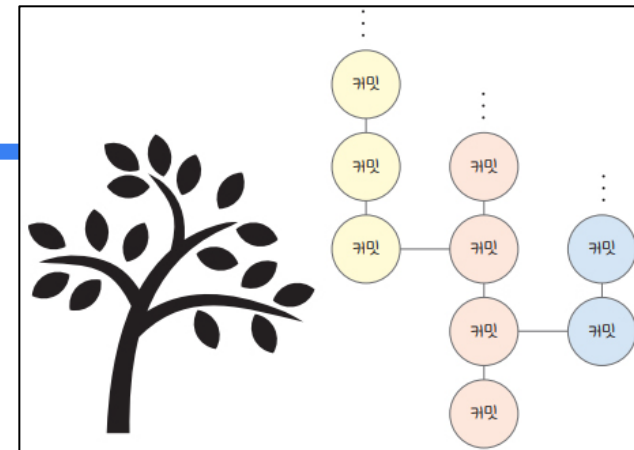
git branch

■ branch 개념 (branch: 나뭇가지?)

● SW 개발시 독립적인 개발 라인을 생성하고 관리하는 방법.

- 개발을 하다 보면 코드를 여러 개로 복사해야 하는 일이 자주 발생.
- 코드를 통째로 복사하고 나서 원래 코드와는 상관없이 독립적으로 개발을 진행할 수 있는데, 이렇게 독립적으로 개발하는 것이 브랜치.
 - 독립적인 작업 공간 생성: 각 브랜치는 다른 브랜치의 영향을 받지 않으므로, 각기 다른 작업을 동시에 진행.
 - 병합 기능: 브랜치는 필요에 따라 다른 브랜치와 병합 될수 있어, 다양한 작업을 하나로 통합하는데 유용.
- Git에서는 기본적으로 master 브랜치 (main이라는 용어를 사용하기도 함)를 생성함.

- 고객과 협의를 거쳐 웹사이트를 개발하고 완성했는데 새로운 기능을 추가해달라고 고객이 요구했다고 가정해보자.
 - 단순히 기존 파일에 새로운 기능을 위한 소스를 추가해서 버전을 새로 만들어도 될까?
 - 만약 새로운 기능을 추가했을때 오류없이 완벽하게 동작한다고 보장할 수 없다면 어떤 방식으로 해야할까?
- 제대로 동작하는 소스는 그대로 둔 채 소스 추가 버전을 따로 만들어 관리하고, 완벽하게 완성한 다음 원래 소스에 더할 수 있다면 편리할 것이다.

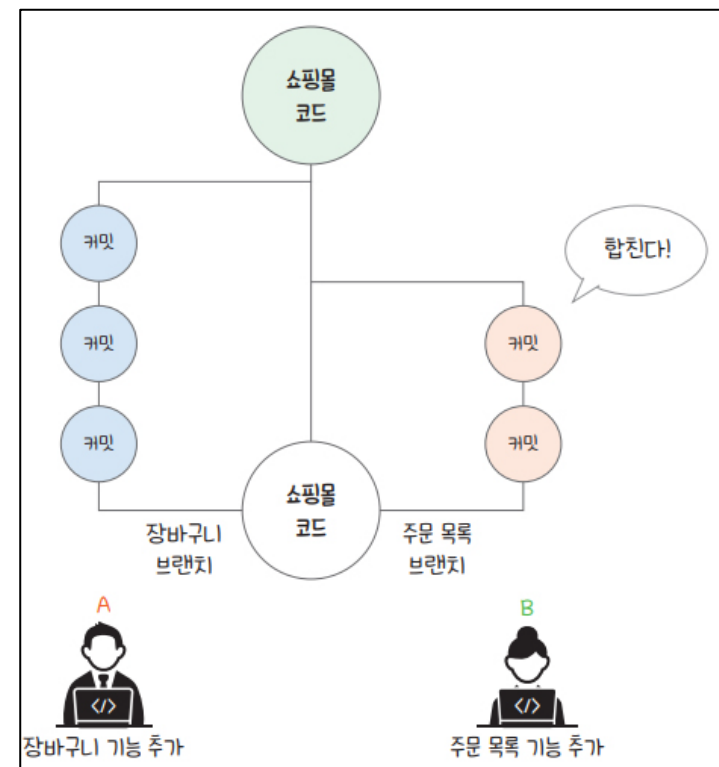
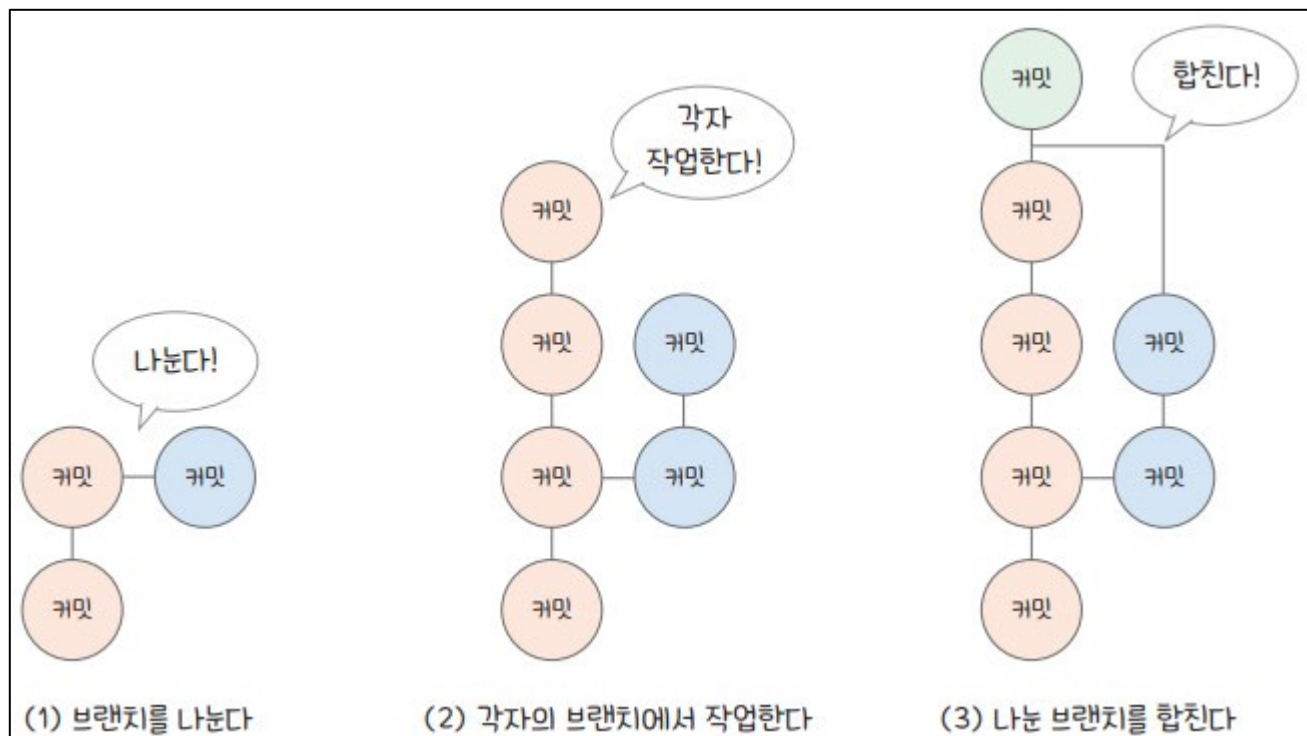


출처: 강민철, 모두의 깃&깃허브

git branch

■ branch의 기본 활용

- 신규 기능 개발, 버그 수정 작업 등을 수행할 때 브랜치를 사용하여 다른 팀 구성원의 작업과 분리하여 진행함.
 - 하나의 SW 프로젝트에서 여러 기능을 동시에 개발하고, 다른 브랜치의 영향을 받지 않으면서 작업 가능.



git branch

- Master branch(Main branch)에서만 협업한다면?
 - 내가 작업중인 파일을 누군가 건드릴 수 있게 됨.
 - 기획 변경으로 개발중인 기능이 필요 없어졌을 때 혹은 문제가 발생했을 때 원하는 시점으로 롤백하기도 어려워짐.
 - 여러 기능을 개발하면서 남겨진 커밋 히스토리가 메인 브랜치에 뒤죽박죽 섞이게 될 것이기 때문.

git branch

■ branch 기능을 사용한다면?

- 다른 브랜치에 영향을 받지 않는 독립적인 환경에서 기능 개발 및 버그 수정 가능.
 - 즉, 여러 기능을 여러 사람이 손쉽게 병렬적으로 개발할 수 있는 환경 구축 가능.
 - 마치 프로젝트 폴더를 복사해서 복사한 폴더에서 각자 따로 작업하는 것과 같음.
- 일반적으로 기능을 개발할 때 브랜치를 생성하고, 코드를 작성하며 커밋을 남김.
- 이후 기능 개발이 완료된 경우에 메인 브랜치에 merge(병합)하면 안전하게 기능을 개발할 수 있음.
- 이런 브랜치를 사용하여 새로운 기능을 개발하다, 기획이 변경되어 기능이 필요 없어졌을 때도 간단하게 브랜치만 삭제해버리면 끝.
- 또한 실험적인 것들을 맘편하게 시도해보고, 안전하게 삭제할 수도 있을 것.

git branch

■ HEAD와 checkout

● HEAD

- HEAD는 기본적으로 현재 작업 중인 브랜치의 최신 커밋을 가리키는 일종의 표시.
- 보통은 현재 작업 중인 브랜치의 최신 커밋을 가리키지만, 브랜치를 나누고 합치는 과정에서 HEAD의 위치를 자유자재로 바꿀 수 있음.

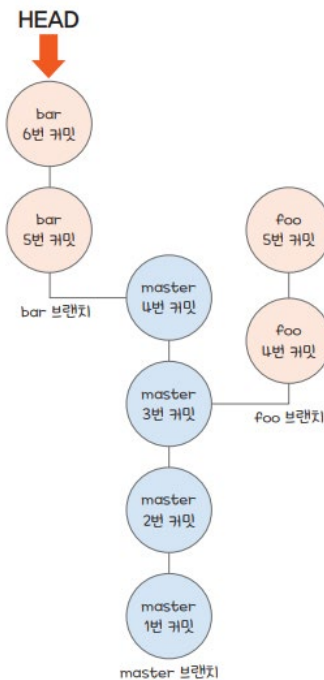
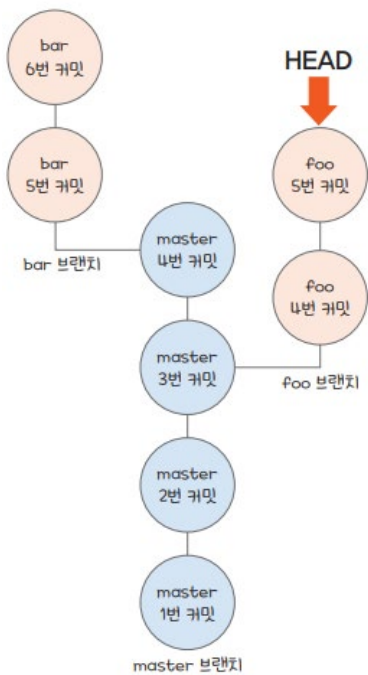
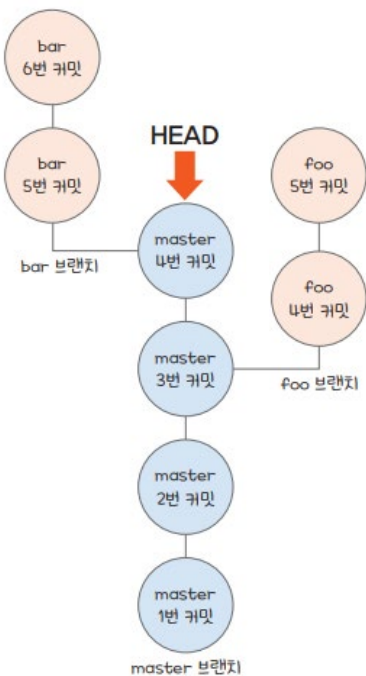
● checkout

- 특정 브랜치에서 작업할 수 있도록 작업 환경을 바꾸는 것을 의미.
- 특정 브랜치로 checkout하게 되면 HEAD의 위치가 해당 브랜치의 최신 커밋을 가리키고, 작업 디렉터리는 checkout한 브랜치의 모습으로 바뀌게 됨.

git branch

■ HEAD와 checkout 예

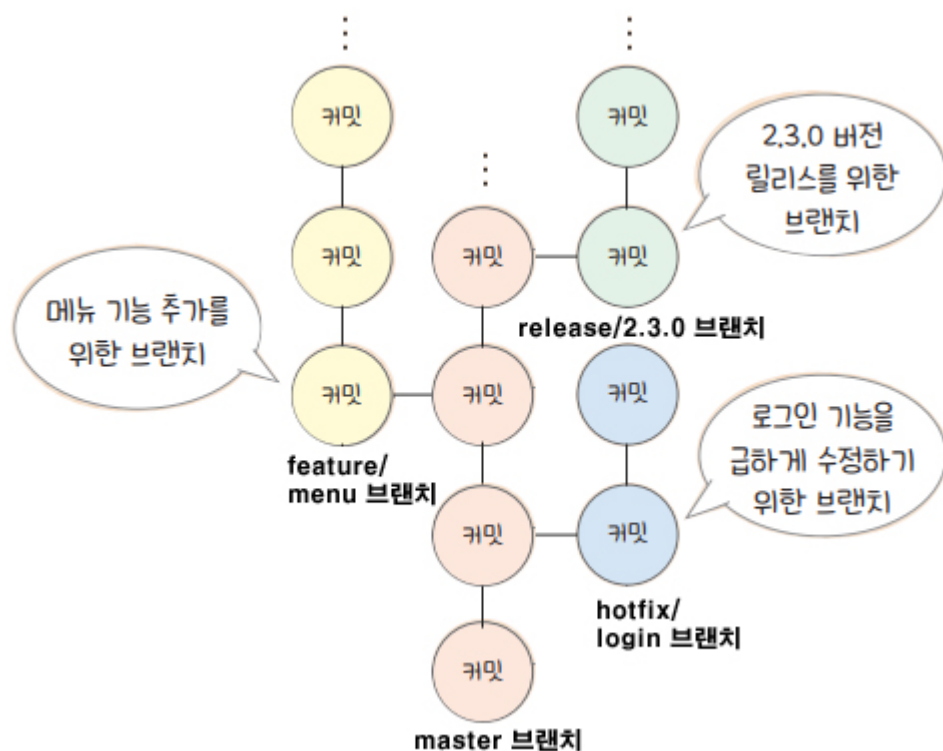
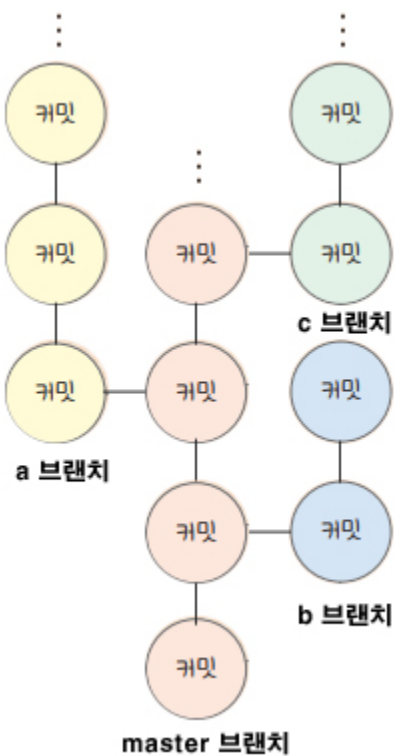
- HEAD가 master 브랜치의 최신 커밋을 가리킬 경우, 다시 말해 master 브랜치로 체크아웃할 경우 working directory에는 총 네 개의 커밋이 만들어진 직후의 모습으로 바뀌게 됨.
- HEAD가 foo 브랜치의 최신 커밋을 가리킬 경우, 다시 말해 foo 브랜치로 체크아웃할 경우 작업 디렉터리는 총 다섯 개의 커밋이 만들어진 직후의 모습을 띄게 됨.
- HEAD가 bar 브랜치를 가리킬 경우, 즉 bar브랜치로 체크아웃할 경우 작업 디렉터리는 총 여섯 개의 커밋이 만들어진 직후의 모습을 띄게 됨



git branch

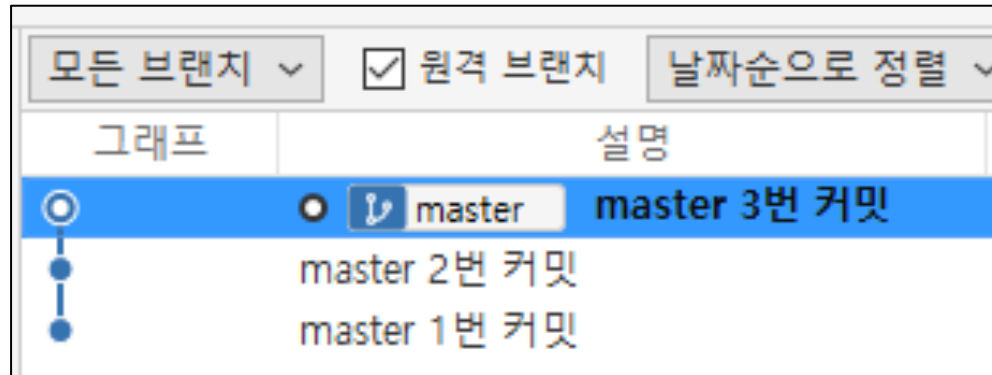
■ branch 나누기 전략

- 실무에서는 branch 이름 정하고 관리하는 전략들이 존재함.
 - branch 이름을 마구잡이로 지으면 해당 branch가 무엇을 위해 만들어졌는지 도무지 알 수가 없음.



(실습) git branch 만들기

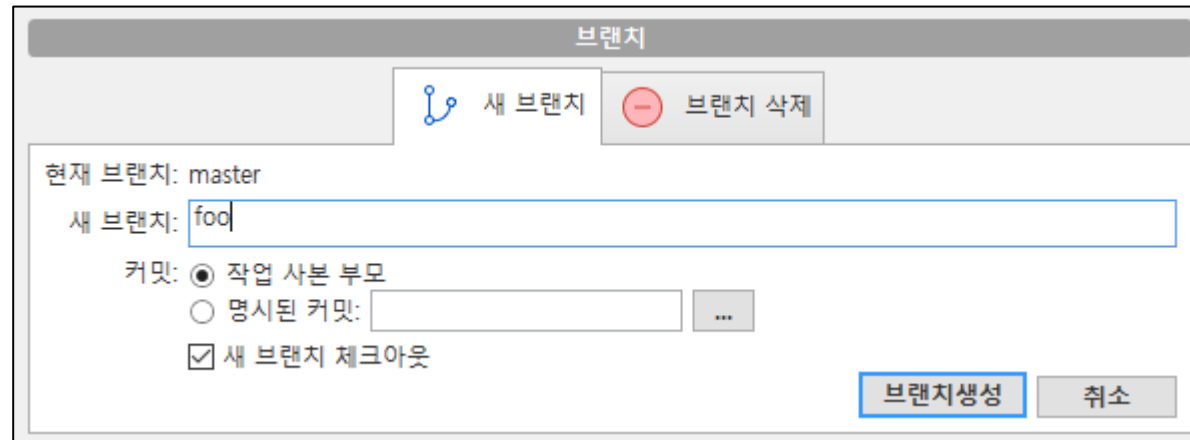
- 비어있는 로컬저장소 생성 후,
 - a.txt 파일 만들고 커밋. 커밋 메시지는 “master 1번 커밋”.
 - b.txt 파일 만들고 커밋. 커밋 메시지는 “master 2번 커밋”.
 - c.txt 파일 만들고 커밋. 커밋 메시지는 “master 3번 커밋”.



(실습) git branch 만들기

■ branch 생성하기

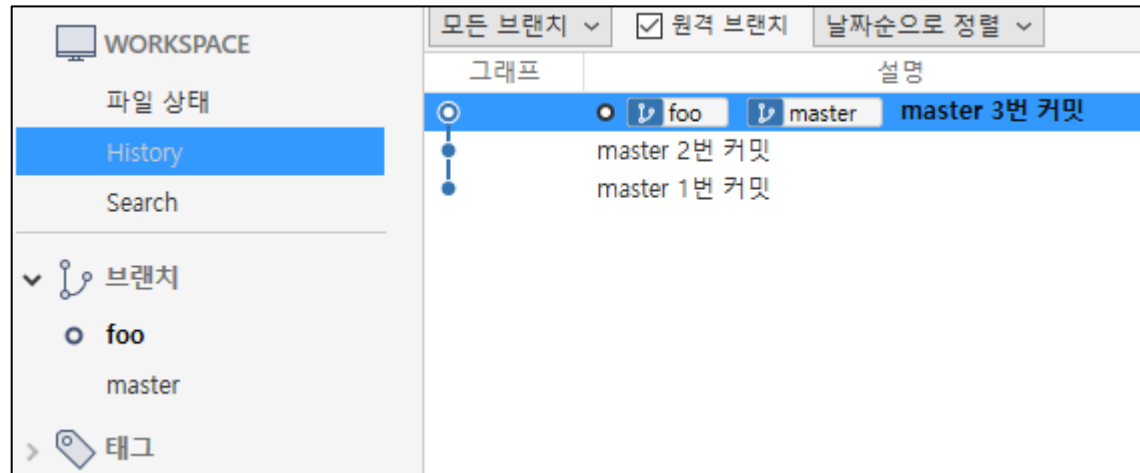
- 상단에 브랜치 클릭 후 foo 라는 이름의 브랜치 생성.
- “새 브랜치 체크아웃” 클릭 확인.
 - 작업환경을 새롭게 생성된 브랜치로 변경한다는 의미.
 - 즉, 작업환경을 foo 브랜치로 변경함.



(실습) git branch 만들기

■ branch 생성하기

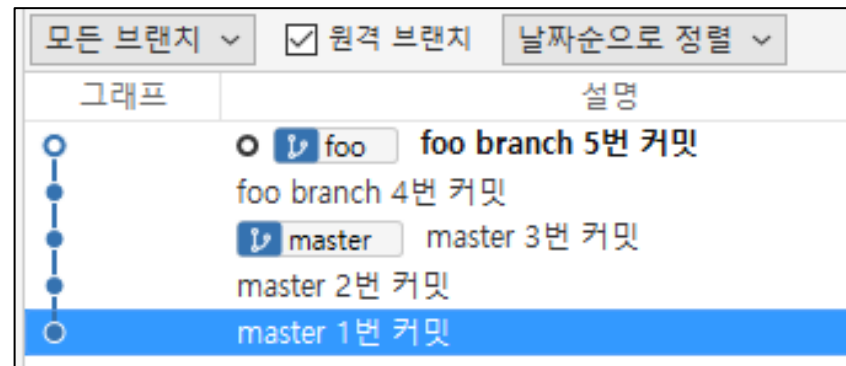
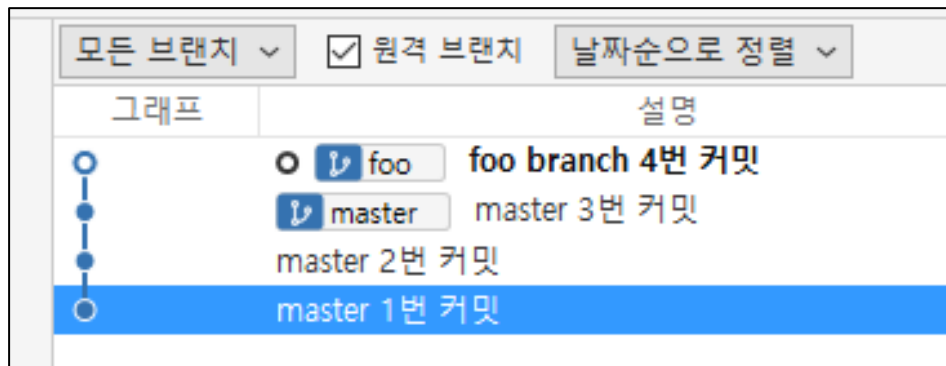
- 좌측의 브랜치 메뉴에 굵게 표기된 foo 확인 가능.
- 굵게 표기된 브랜치 이름이 현재 체크아웃된 브랜치를 나타냄.



(실습) git branch 만들기

■ branch 생성하기

- foo 브랜치에 파일 추가하여 커밋 쌓아 올리기.
 - d.txt 파일 만들고 커밋. 커밋 메시지는 “foo branch 4번 커밋”.
 - e.txt 파일 만들고 커밋. 커밋 메시지는 “foo branch 5번 커밋”.
 - master branch가 아닌 foo branch에 추가됨.
 - 현재 checkout된 branch가 foo branch이기 때문.



(실습) git branch 만들기

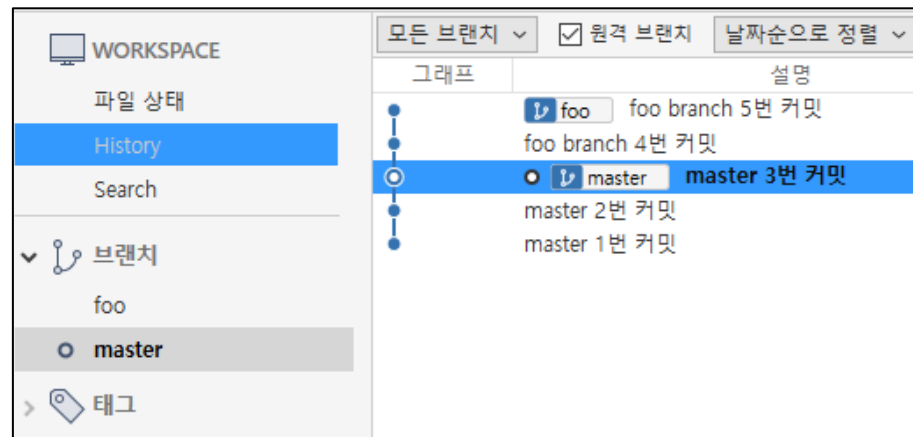
■ foo branch 파일 내용 확인

- 탐색기 클릭하여 현재 working directory에 txt파일 5개 확인 가능 (현재는 foo branch에 checkout 되어 있기 때문)
 - master branch에는 커밋이 3개
 - master branch로부터 파생된 foo branch에는 커밋이 5개

■ master branch로 checkout 해보기

- 좌측 브랜치 메뉴에서 master를 더블클릭.
- 탐색기 클릭하여 현재 working directory에 txt파일 3개 확인 가능

(현재는 master branch에 checkout 되어 있기 때문)

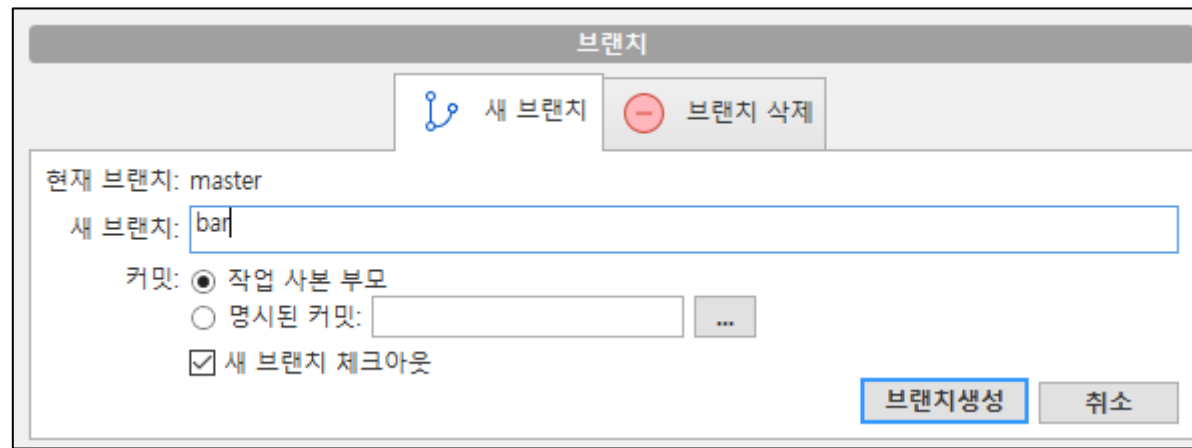


(실습) git branch 만들기

■ 새로운 branch 생성하기

● bar 브랜치 생성하기

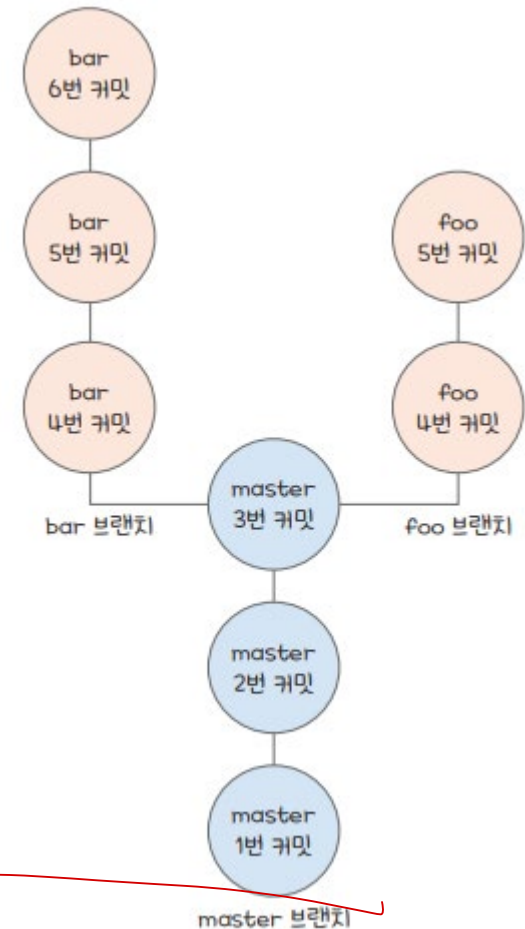
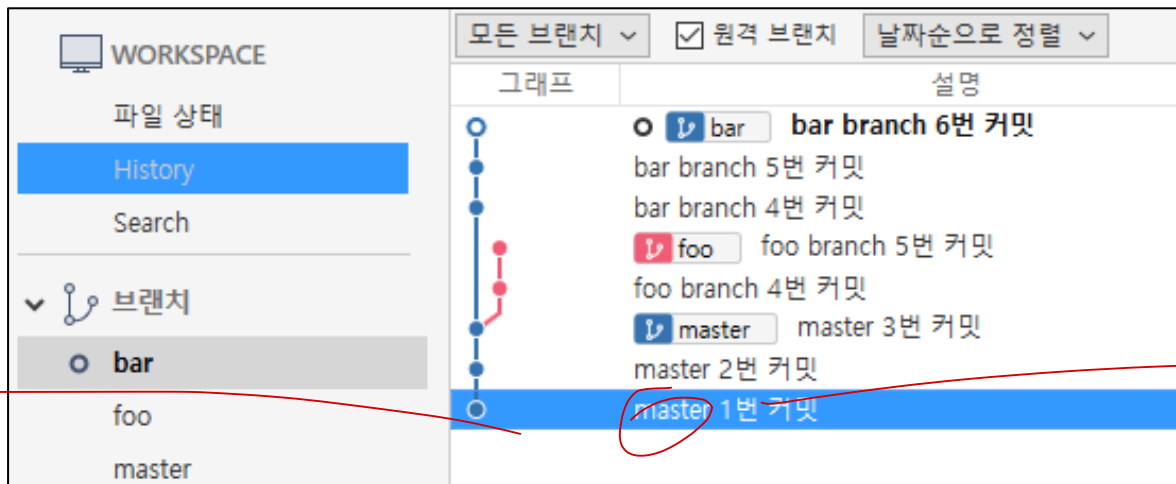
- 현재 master 브랜치에 체크아웃되어 있는 상태임.
- 상단에 브랜치 클릭.
- 작업환경을 bar 브랜치로 변경하기 위해 '새 브랜치 체크아웃' 클릭 확인.



(실습) git branch 만들기

■ 새로운 branch 생성하기

- bar 브랜치에 파일 추가하여 커밋 쌓아 올리기
 - bar_d.txt 파일 만들고 커밋. 커밋 메시지는 “bar branch 4번 커밋”.
 - bar_e.txt 파일 만들고 커밋. 커밋 메시지는 “bar branch 5번 커밋”.
 - bar_f.txt 파일 만들고 커밋. 커밋 메시지는 “bar branch 6번 커밋”.
 - master branch가 아닌 bar branch에 추가됨.
 - 현재 checkout된 branch가 bar branch이기 때문.



git flow

- git 브랜치를 효과적으로 관리하기 위한 워크플로우 (대표적인 branch 전략)
 - 각 SW개발업체별로 직접 브랜치 전략을 만들어 사용해도 되겠지만, 세상에는 브랜치를 효과적으로 관리하기 위한 모범 사례들이 존재함.
 - 좋은 브랜치도 규칙 없이 마구잡이로 사용하면 혼란 발생: 명확한 기준 필요.
 - ‘이 브랜치는 어떤 목적으로 생성된거지?’,
 - ‘이 브랜치는 어떤 커밋에서 분기된거지?’,
 - ‘어떤 브랜치에서 내 브랜치를 생성해야하지?’,
 - ‘내 브랜치는 어디에 병합해야지?’,
 - ‘어떤 브랜치가 최신이지?’,
 - ‘어떤 브랜치가 배포된 버전이지?’ 등등 수 많은 의문점이 생길것이고 프로젝트는 엉망이될 것 이다.

git flow

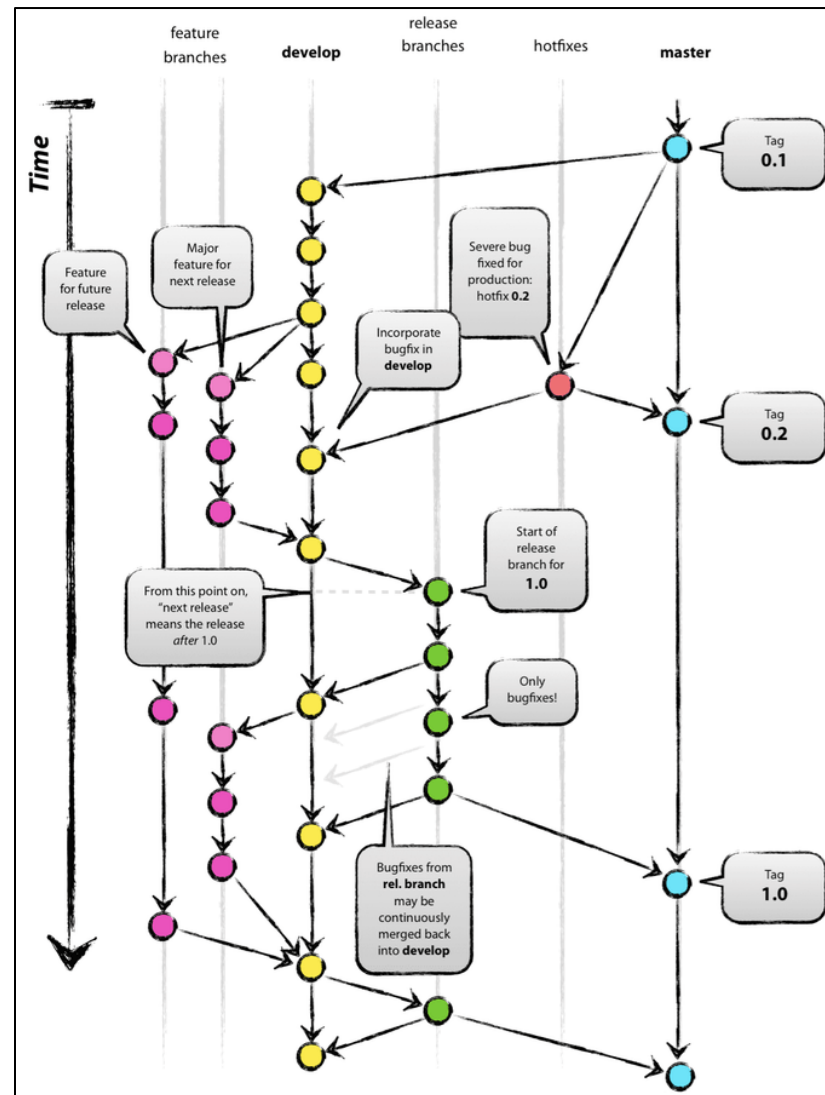
■ git flow의 브랜치 전략 (Main, Develop, Supporting branches)

● Main branch

- 프로젝트 시작 시 생성되며, 개발 프로세스 전반에 걸쳐 유지.
- 출시 가능한 프로덕션 코드를 모아두는 브랜치.
- 배포된 각 버전을 Tag를 이용해 표시.

● Develop branch

- 개발 프로세스 전반에 걸쳐 항상 유지되는 브랜치.
- 다음 버전 개발을 위한 코드를 모아두는 브랜치.
- 개발이 완료되면, Main 브랜치로 merge 됨.



git flow

■ git flow의 브랜치 전략

● Supporting branch

■ Feature branch,

- 하나의 기능을 개발하기 위한 브랜치.
- Develop 브랜치에서 생성하며, 기능이 개발 완료되면 다시 Develop 브랜치로 merge 됨.
- (주의점) Fast-Forward로 merge하지 않고, Merge Commit을 생성하며 머지를 해주어야 한다. 이렇게해야 히스토리가 특정 기능 단위로 묶이게 됨.
- 네이밍은 feature/branch-name 과 같은 형태로 생성.

■ Release branch,

■ Hotfix branch.

git flow

■ git flow의 브랜치 전략

● Supporting branch

■ Feature branch,

■ Release branch,

- 소프트웨어 release를 준비하기 위한 브랜치.
- Develop 브랜치에서 생성하며, 버전 이름 등의 소소한 데이터를 수정하거나 배포전 사소한 버그를 수정하기 위해 사용.
- 배포 준비가 완료되었다면 Main과 Develop 브랜치에 둘다 merge 함.
- 이때, Main 브랜치에는 태그를 이용하여 버전을 표시.
- Release 브랜치를 따로 운용함으로써, 배포 업무와 관련없는 팀원들은 병렬적으로 Feature 브랜치에서 이어서 기능을 개발할 수 있게 됨.
- 네이밍은 release/v1.1 과 같은 형태로 생성.

■ Hotfix branch.

git flow

■ git flow의 브랜치 전략

● Supporting branch

- Feature branch,
- Release branch,
- Hotfix branch.

- 이미 배포된 버전에 문제가 발생했다면, Hotfix 브랜치를 사용하여 문제를 해결.
- Main 브랜치에서 생성하며, 문제 해결이 완료되면 Main과 Develop 브랜치에 둘다 merge 함.
- Release 브랜치와 마찬가지로 Hotfix 브랜치를 따로 운용함으로써, 핫픽스 업무와 관련없는 팀은 병렬적으로 기능 개발 가능.
- 네이밍은 hotfix/v1.0.1 과 같은 형태로 생성.

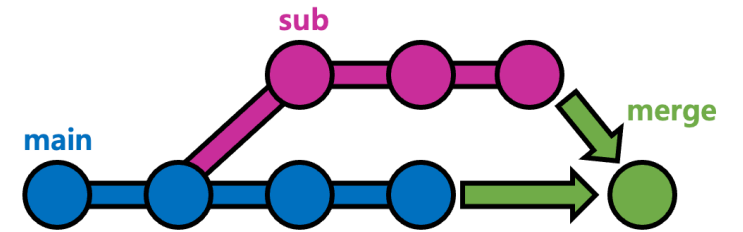
branch 합치기

■ branch를 합치는 방법은 크게 2가지

● (1) Merge

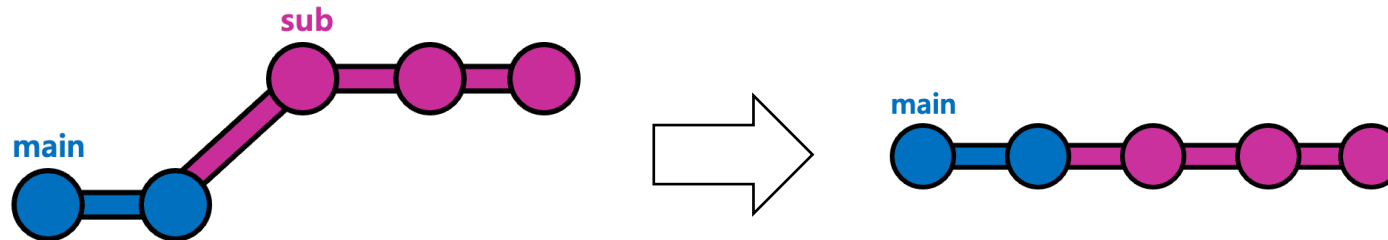
■ 3-way merge

- 각 branch에 새로운 commit이 하나씩 존재하는 경우 수행되는 merge
- 새로운 commit에 2개의 branch를 합치는 것
- 가장 기본적인 형태



■ fast-forward merge

- 3-way와 다르게 새로운 branch에만 commit이 존재하고 기존 branch에서는 commit을 하지 않은 경우에는 자동으로 fast-forward merge가 수행



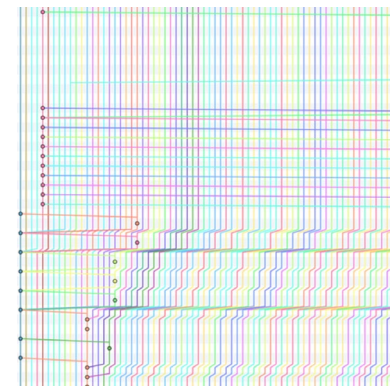
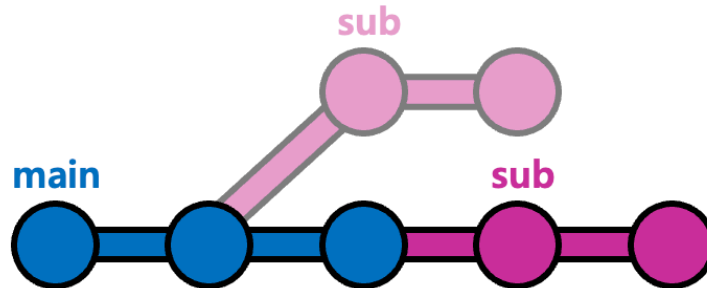
branch 합치기

■ branch를 합치는 방법은 크게 2가지

● (2) Rebase

■ 브랜치의 시작점을 다른 commit으로 옮겨주는 것

- rebase 후 두 브랜치의 병합은 sub브랜치의 시작점을 main의 최근 commit으로 옮겨준 후 (rebase)
- fast-forward merge를 수행

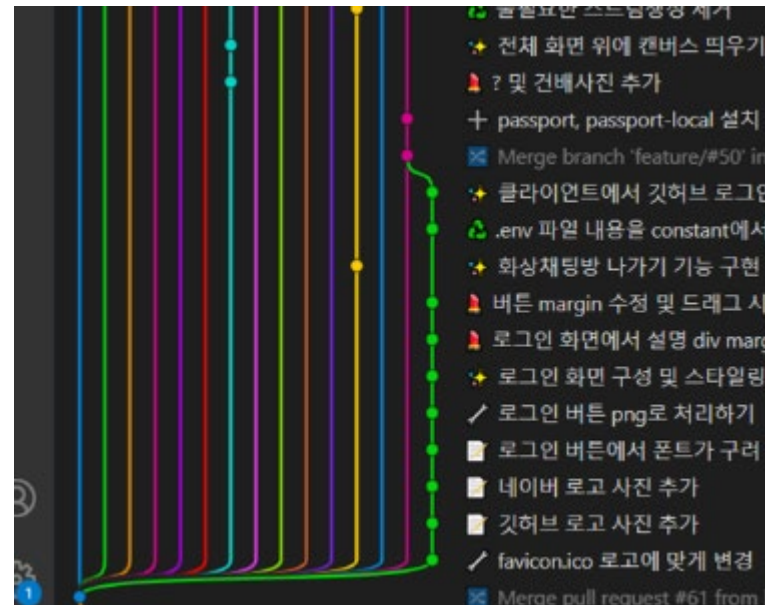
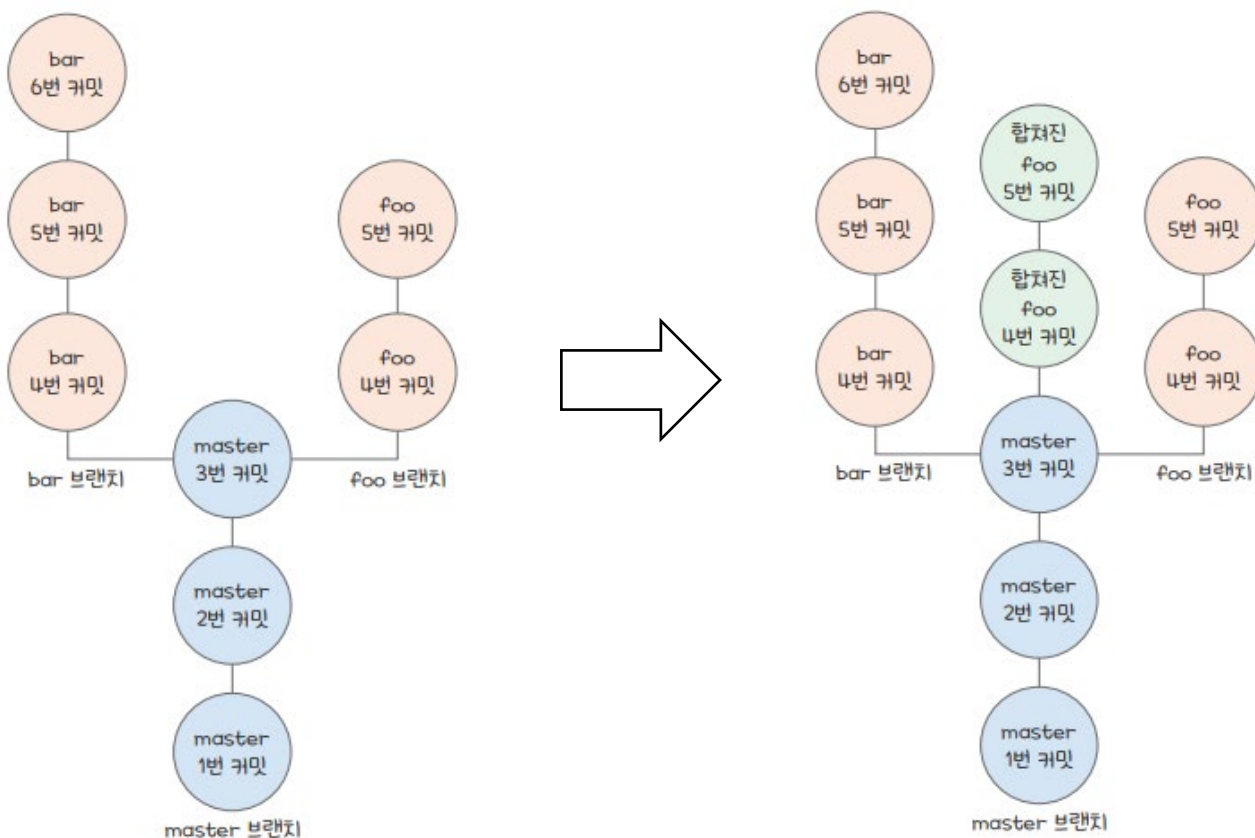


- 복잡한 브랜치 구조에서 3-way merge를 하다보면 git log가 복잡해지는데 간단하고 짧은 브랜치에 대해서 rebase를 사용하면 깔끔하게 정돈할 수 있음.
- (단점) conflict 발생 확률이 높기 때문에 이를 하나씩 해결해주어야 할 수도 있다.

git merge

■ merge (병합)개념

- 나누어진 브랜치를 하나로 통합하는 것.
- 앞선 실습에서 foo 브랜치를 master 브랜치로 병합하면 어떻게 될까?

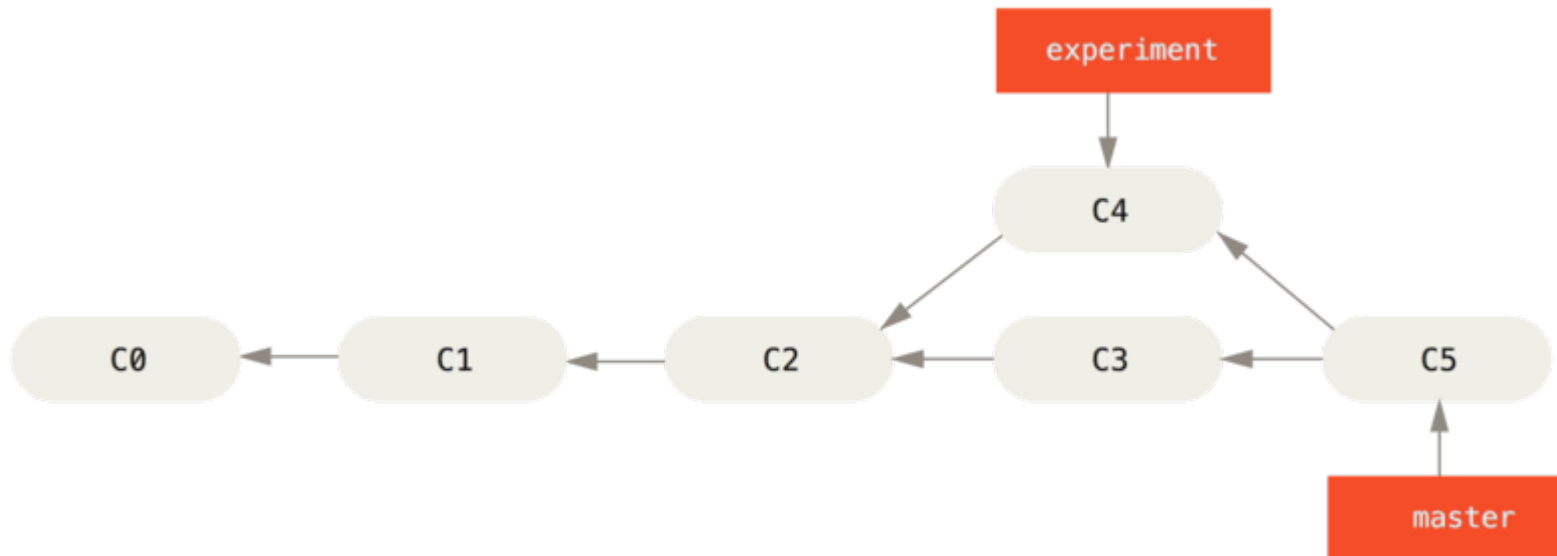


Merge가 없다면?

git merge

■ 3-way merge

- 각 브랜치의 마지막 커밋 두 개와 공통 조상의 총 3개의 커밋을 이용하는 merge 방식
 - 3-way merge를 수행하여 새로운 커밋을 만들어 냄
- (예) 두 브랜치의 마지막 커밋 두 개(C3, C4)와 공통 조상(C2)을 사용하는 3-way Merge로 새로운 커밋을 만들어 냄.



git merge

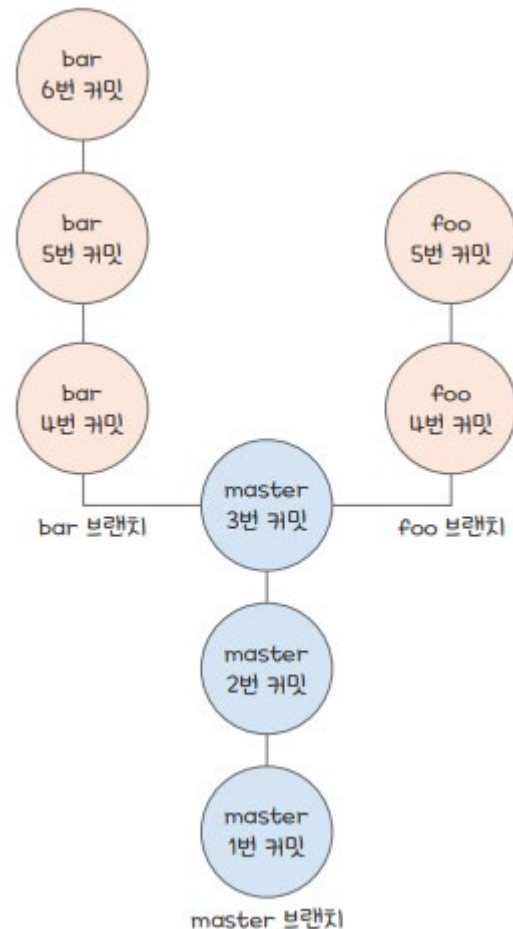
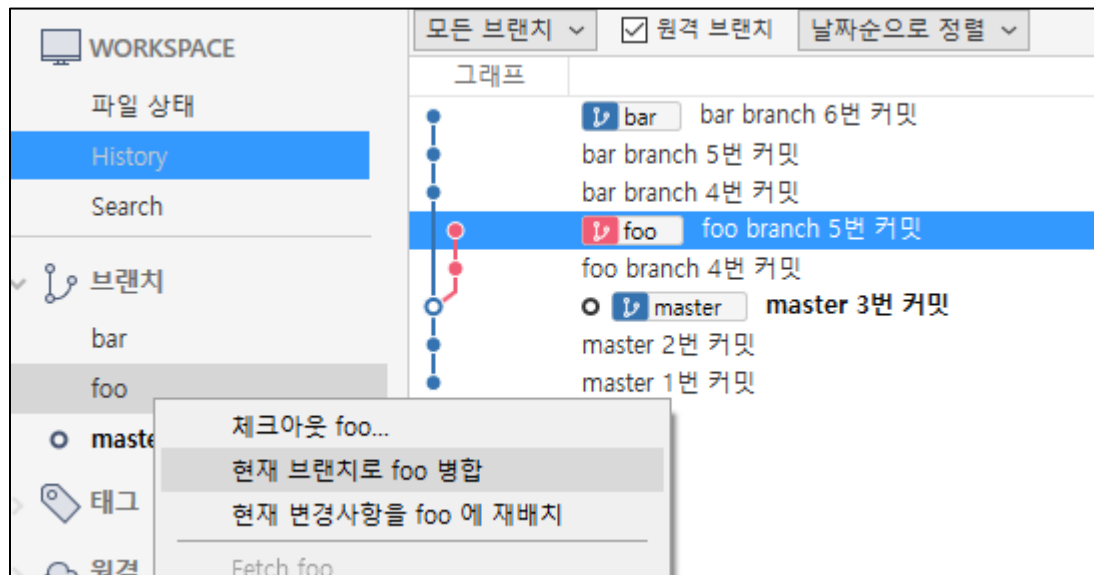
■ Fast-forward merge

- 변화가 없었던 브랜치에 특정 브랜치가 병합되는 작업 (HEAD만 변경)
- (예) master 브랜치 입장에서는 자신의 브랜치가 마치 빨리 감기 하듯이 foo 브랜치와 동일하게 업데이트됨.
 - master 브랜치가 빨리 감기 하듯 foo와 동일해질 수 있었던 이유는, foo 브랜치가 master 브랜치에서부터 뺄어나온 시점부터 병합되는 순간까지 master 브랜치에 어떤 변화도 없었기 때문임.
 - 다시 말해, foo 브랜치는 master 브랜치에서 뺄어나온 이후로 여러 커밋이 쌓였지만, 그동안 master 브랜치는 어떤 새로운 커밋도 없이 그저 가만히 있었음.
 - 그렇기 때문에 foo 브랜치를 master 브랜치로 병합할 적에 master 브랜치는 그저 foo 브랜치에 새롭게 쌓인 커밋을 반영만 하면 됨.
- 상위 branch에 변경사항이 없을 때에만 fast-forward merge 가능

(실습) git merge

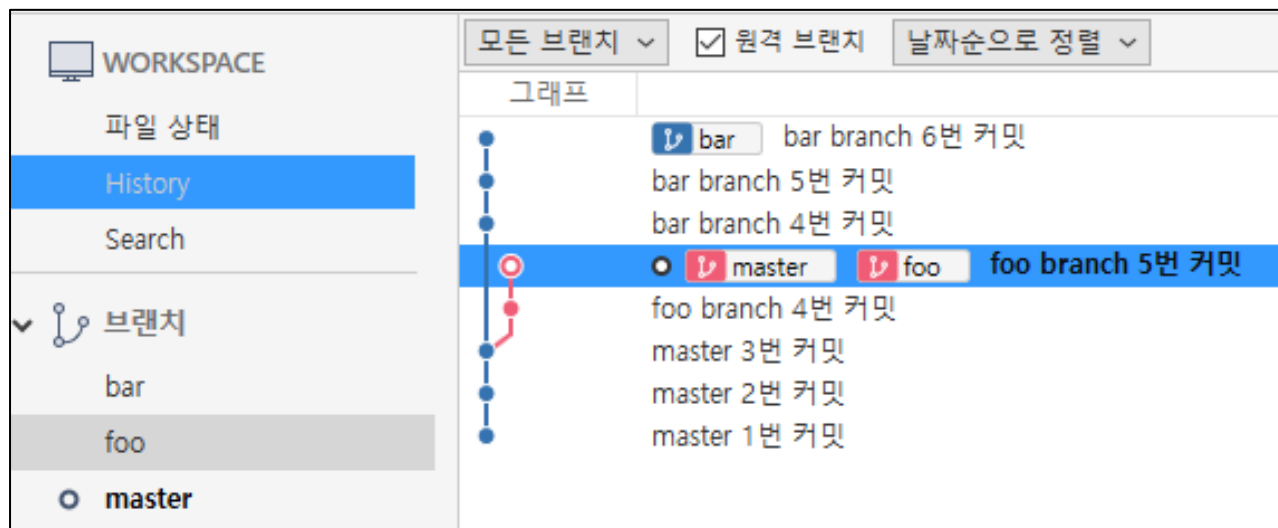
■ foo 브랜치를 master 브랜치로 merge 하기

- 먼저 master브랜치로 병합하려면 master 브랜치로 체크아웃해야 함.
- foo 브랜치에서 마우스 오른쪽 버튼을 클릭.
 - 현재 브랜치로 foo 병합이 있음
 - '현재 브랜치'는 현재 체크아웃한 브랜치, 즉 master 브랜치를 말함
 - master 브랜치로 foo 브랜치를 병합해야 하니 이를 클릭.



(실습) git merge

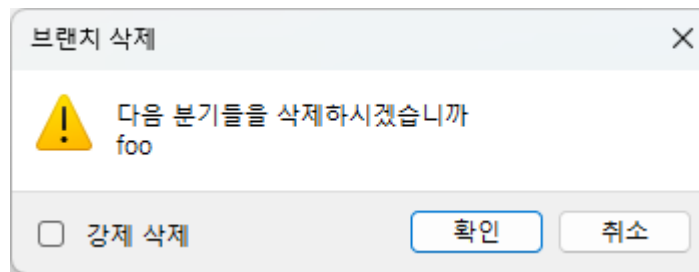
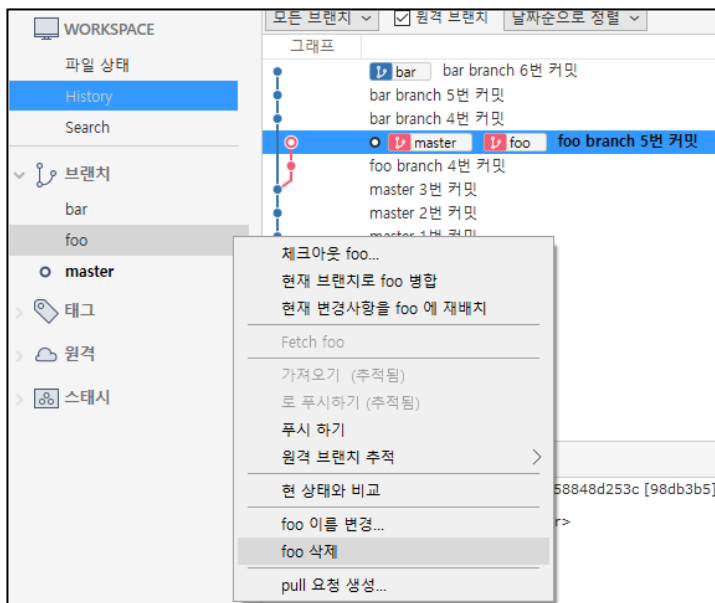
- foo 브랜치를 master 브랜치로 merge 하기
 - '병합 확정' 창이 뜨면 확인을 클릭
 - fast-forward가 가능해도 새 커밋으로 생성 항목은 체크하지 않음
 - foo 브랜치가 master 브랜치에 병합 완료
 - master 브랜치는 foo 브랜치와 같아졌음.



(실습) git merge

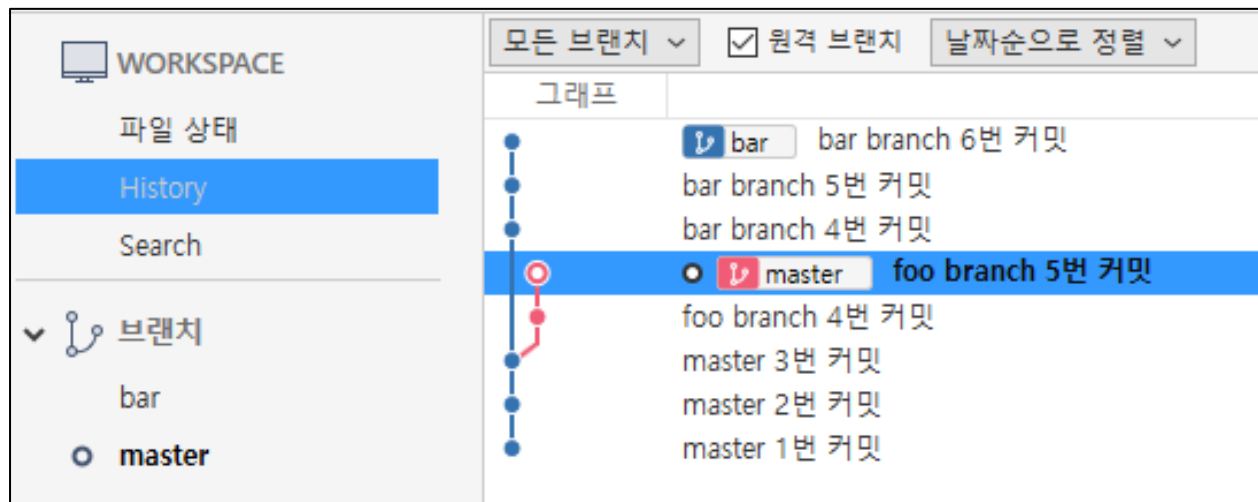
■ foo 브랜치를 master 브랜치로 merge 하기

- 브랜치를 병합하고 나서 더 이상 브랜치에 남은 작업이 없다면 삭제 권장.
 - foo 브랜치를 master 브랜치와 병합한 뒤 foo 브랜치에서 더는 작업하지 않을 예정이라면 병합 후 foo 브랜치를 삭제하는 것을 권장
- foo 브랜치를 삭제하기 위해서는 master 브랜치에 체크아웃되어 있는 상태로 foo 브랜치에서 마우스 오른쪽 버튼을 클릭한 후 foo 삭제를 클릭.



(실습) git merge

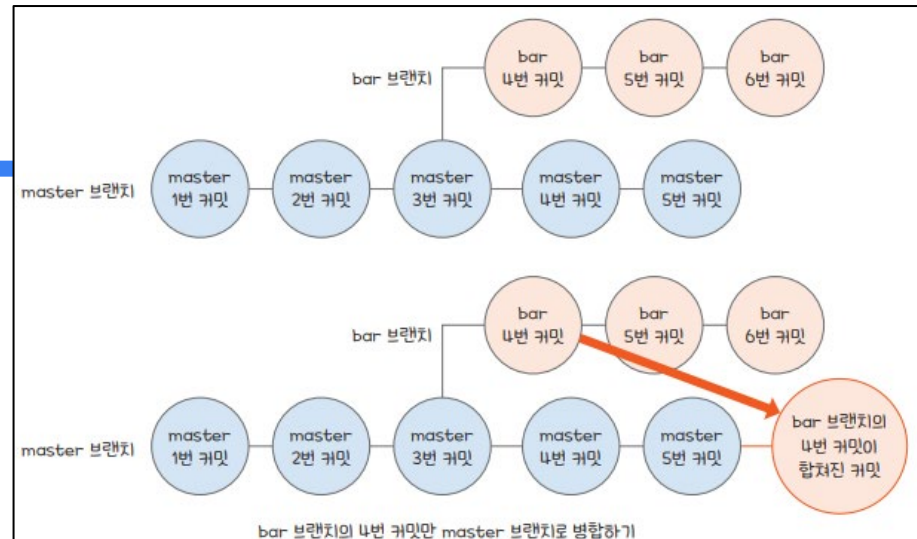
- foo 브랜치를 master 브랜치로 merge 하기
 - foo 브랜치 삭제 완료 결과 화면



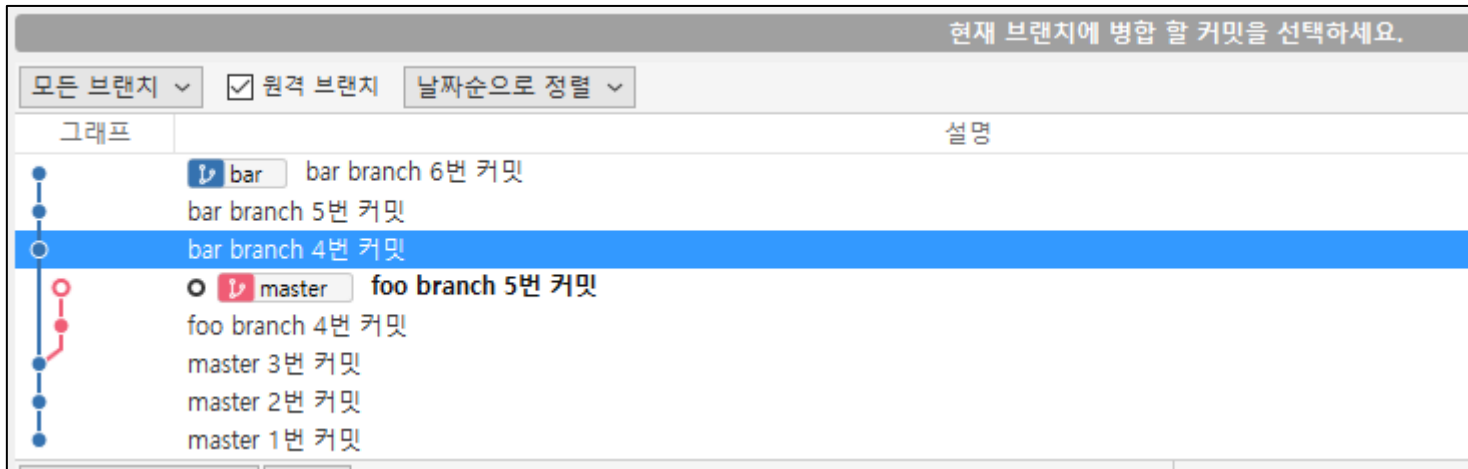
(실습) git merge

■ bar 브랜치의 4번 커밋을 master로 merge 하기

- 메뉴바 상단에 “병합” 메뉴도 이용 가능.
- master 브랜치로 체크아웃한 뒤 상단의 병합을 클릭.
- 여기서 master 브랜치에 병합할 브랜치의 커밋을 클릭한 뒤 확인을 누르면 해당 커밋이 병합

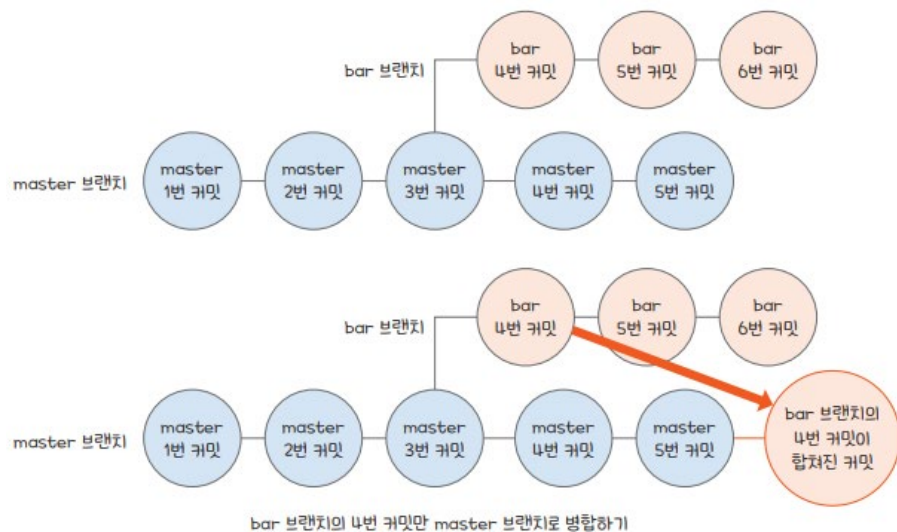
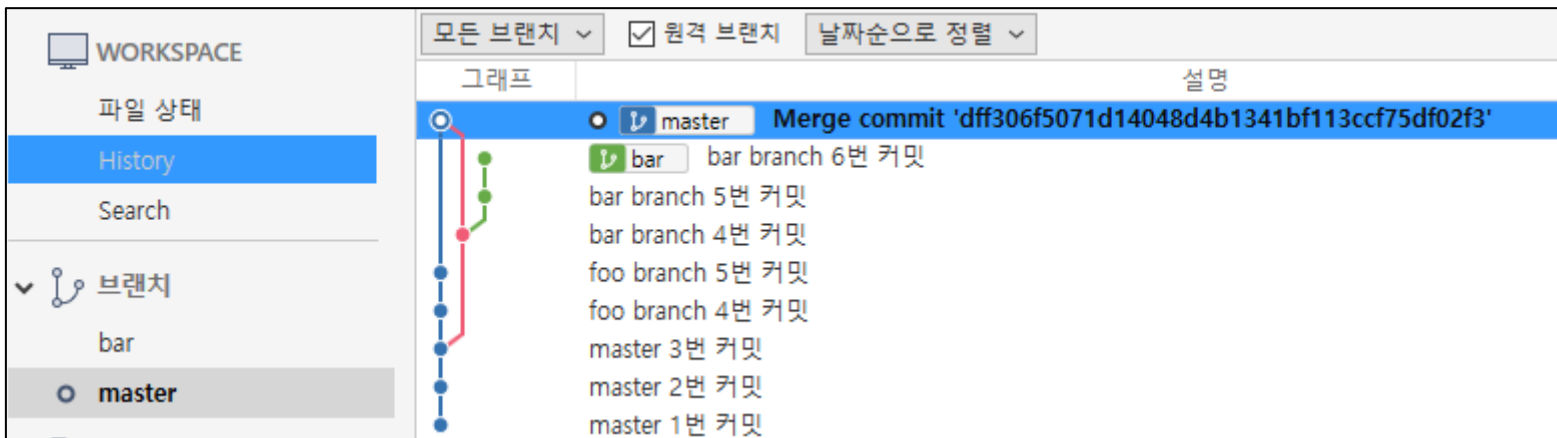


- bar 브랜치의 4번 커밋을 master 브랜치로 병합하려면 4번 커밋을 클릭한 뒤 확인을 클릭



(실습) git merge

- bar 브랜치의 4번 커밋을 master로 merge 하기
 - master 브랜치에 bar 브랜치 4번 커밋이 병합됨.
 - Merge commit '<commit 해시>' 라는 새로운 커밋이 생성됐다는 것을 볼 수 있음.
 - Fast-forward merge가 아닌 병합에서는 항상 새로운 commit이 생성됨.
 - 다음 그림과 같이 bar 브랜치에는 없는 커밋이 master브랜치에 있고 master 브랜치에는 없는 커밋이 bar 브랜치에 있는 상태에서 두 브랜치를 병합할 때는 이렇듯 새로운 커밋이 생성됨.



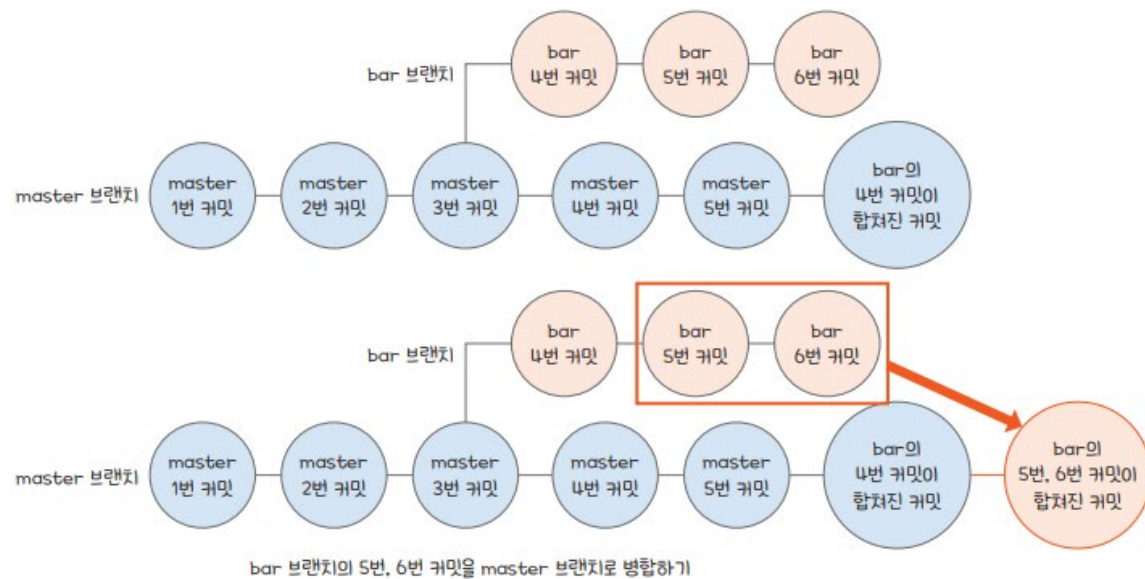
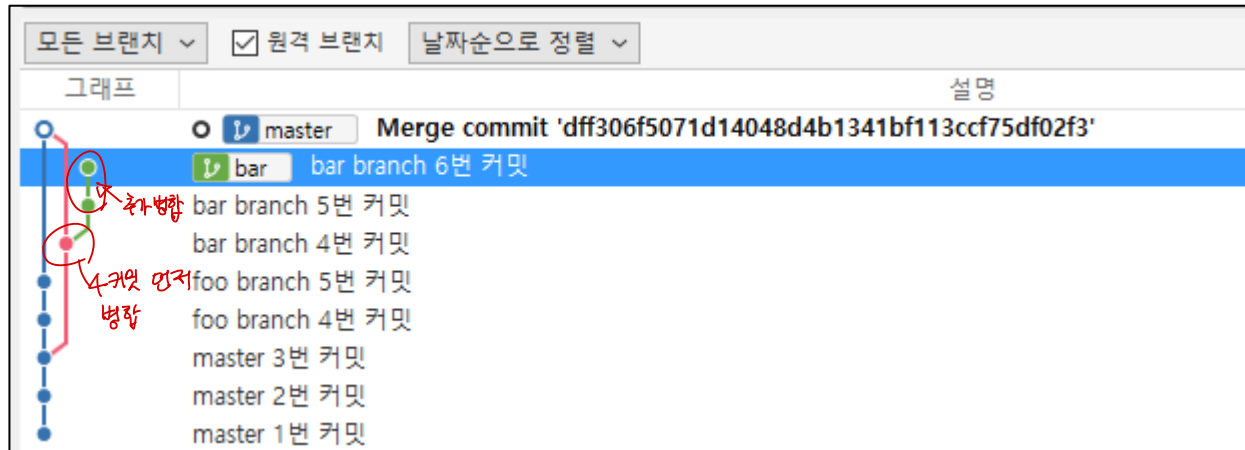
(실습) git merge

- bar 브랜치의 4번 커밋을 master로 merge 하기
 - 탐색기의 파일 현재 파일 상태를 확인해보기
 - bar 브랜치의 네 번째 커밋이었던 bar_d.txt 파일이 master 브랜치에 생성



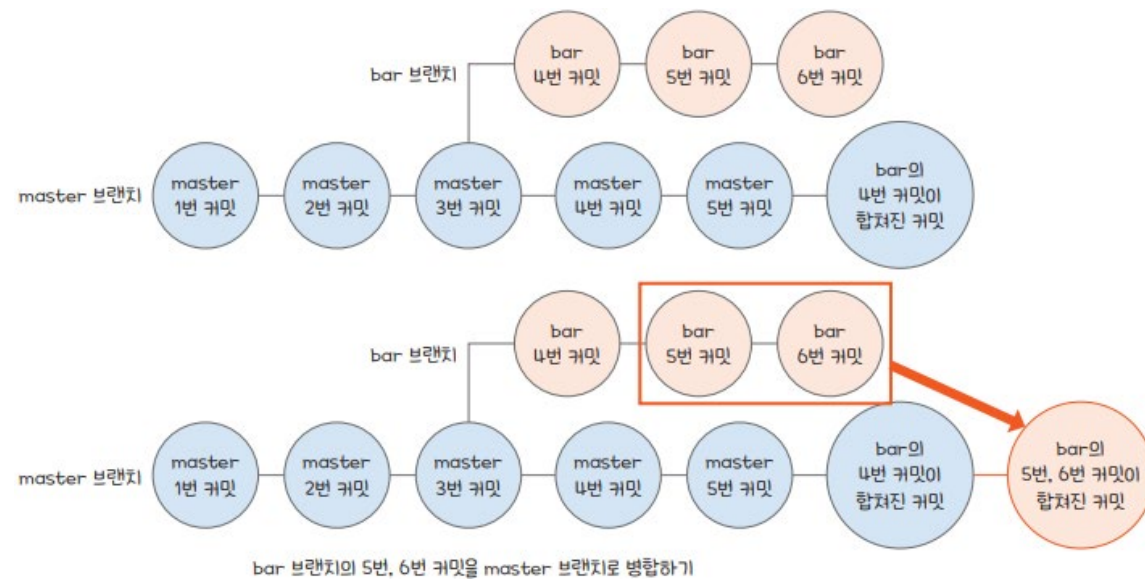
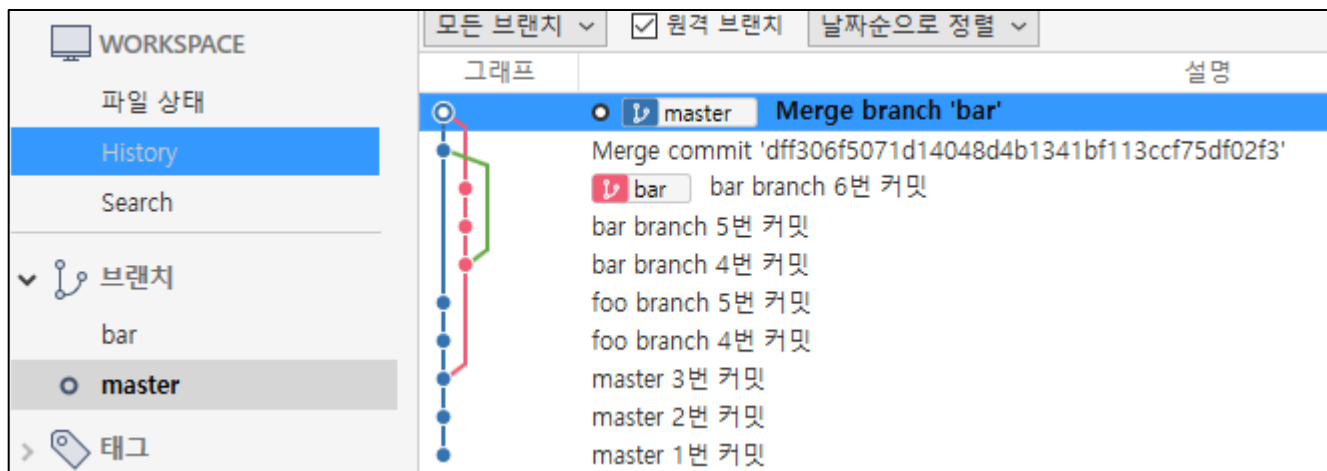
(실습) git merge

- bar 브랜치의 5번 및 6번 커밋을 master로 merge 하기
 - master 브랜치로 체크아웃한 뒤 상단의 병합을 클릭.
 - bar 브랜치의 6번 커밋을 클릭한 뒤 확인을 클릭 (6번 커밋에 5번 커밋도 포함).



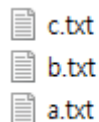
(실습) git merge

- bar 브랜치의 5번 및 6번 커밋을 master로 merge 하기
 - master 브랜치에 bar 브랜치 5번 및 6번 커밋이 병합됨.
 - Merge branch 'bar' 라는 새로운 커밋이 생성.

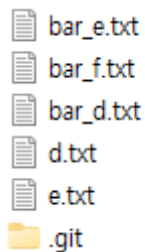


(실습) git merge

- bar 브랜치의 5번 및 6번 커밋을 master로 merge 하기
 - 탐색기의 파일 현재 파일 상태를 확인해보기
 - bar 브랜치의 6번째 커밋이었던 bar_f.txt 파일이 master 브랜치에 생성
 - bar 브랜치의 5번째 커밋이었던 bar_e.txt 파일이 master 브랜치에 생성



c.txt
b.txt
a.txt



bar_e.txt
bar_f.txt
bar_d.txt
d.txt
e.txt
.git

- foo 브랜치에서 생성했던 파일들과 bar브랜치에서 생성했던 파일들이 모두 잘 추가되어 있음을 확인 가능.

git merge에서 충돌 해결하기

■ Resolve the Merge Conflicts

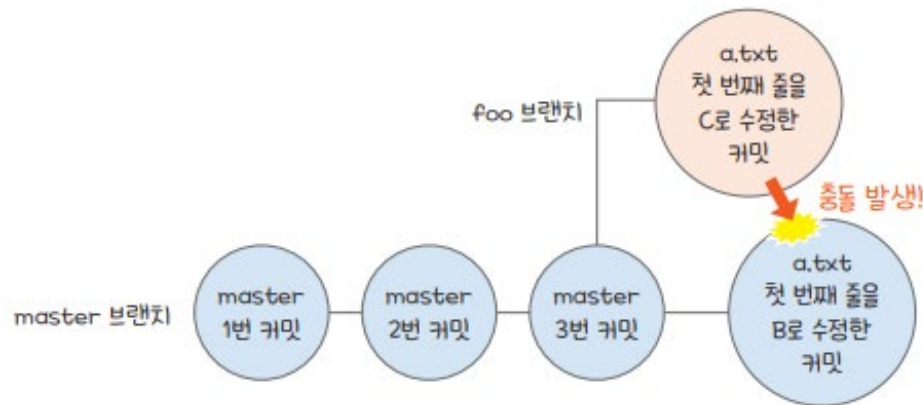
- 충돌이란 병합하려는 두 브랜치가 서로 같은 내용을 다르게 수정한 상황을 의미.
 - 실제 SW 개발환경에서 브랜치를 병합하는 과정은 생각보다 순탄치 않음.
 - 앞에서는 브랜치가 한 번에 성공적으로 합쳐졌지만 그렇지 못한 상황, 즉 충돌이 발생하는 경우도 있기 때문임.
- 충돌이 발생하면 브랜치가 한 번에 병합되지 못함.
- 충돌은 여럿이 협업하여 개발할 때 빈번히 발생함.

git merge에서 충돌 해결하기

■ Resolve the Merge Conflicts

● (예제)

- master 브랜치에서 foo 브랜치가 생성된 경우를 생각해보자.
- master 브랜치는 a.txt 파일의 첫 번째 줄을 B로 수정한 다음 커밋했고, foo 브랜치는 a.txt 파일의 첫 번째 줄을 C로 수정한 다음 커밋했다고 가정하자.
- 이런 상황에서 foo 브랜치를 master 브랜치에 병합한다면? Conflict 발생!
 - a.txt 파일에는 어떤 내용을 저장해야 할까?
 - master 브랜치를 따라 B라고 저장해야 할까? 아니면 foo 브랜치를 따라 C라고 저장해야 할까?
 - 결국, 개발자가 어떤 브랜치를 반영할지 직접 선택해야 함.



git merge에서 충돌 해결하기

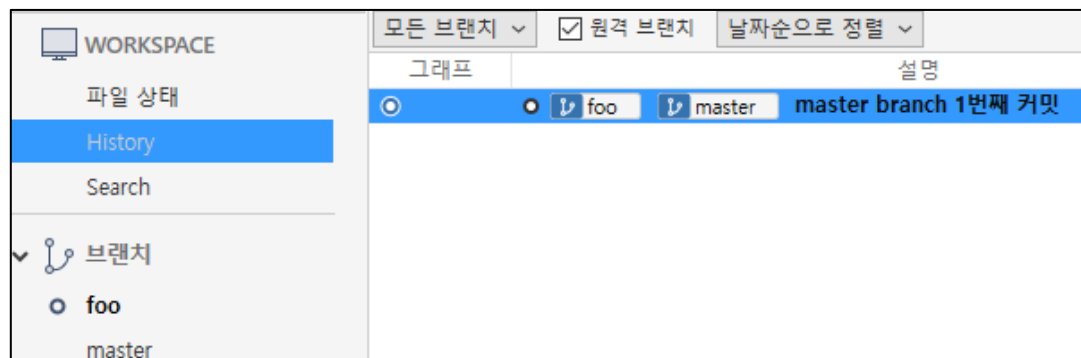
■ Resolve the Merge Conflicts

- 브랜치를 병합하는 과정에서 충돌이 발생했을 경우, **충돌이 발생한 파일들의 충돌을 해결한 뒤 다시 커밋**해야만 브랜치가 올바르게 병합.
- 충돌을 해결한다는 의미는 어떤 브랜치 내용을 최종적으로 반영할지를 직접 선택하는 것.
 - 충돌이 발생한 이유는 병합하려는 두 브랜치가 같은 내용을 서로 다르게 수정했기 때문임.
 - 따라서 충돌이 발생하면 충돌이 발생한 두 브랜치 중 어떤 브랜치의 내용을 병합 결과에 반영할지를 개발자가 직접 선택.

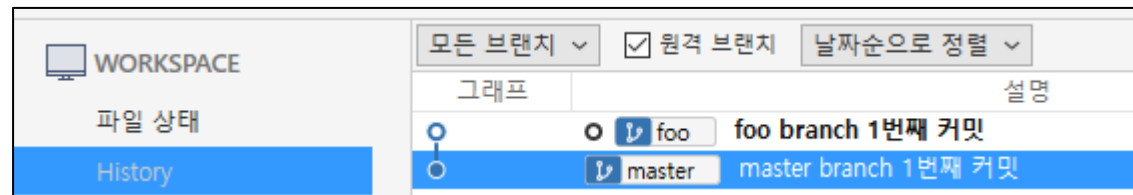
(실습) Merge conflict

■ 브랜치 충돌 상황 만들기

- 임의의 로컬 저장소를 만들기
- master 브랜치에 A가 저장된 a.txt 파일을 만든 뒤 이를 커밋
 - 커밋 메시지: master branch 1번째 커밋
- 새로운 브랜치 foo를 생성하기



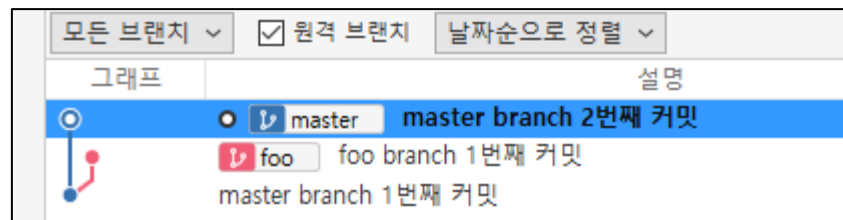
- 생성한 foo 브랜치로 체크아웃 후 a.txt 파일에 적힌 A를 foo로 변경한 뒤 커밋.
 - 커밋 메시지: foo branch 1번째 커밋



(실습) Merge conflict

■ 브랜치 충돌 상황 만들기

- master 브랜치로 체크아웃 후 a.txt 파일에 적힌 A를 master로 변경한 뒤 커밋.
 - 커밋 메시지: master branch 2번째 커밋



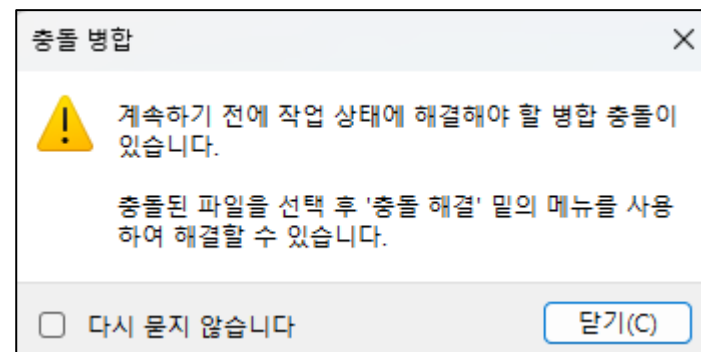
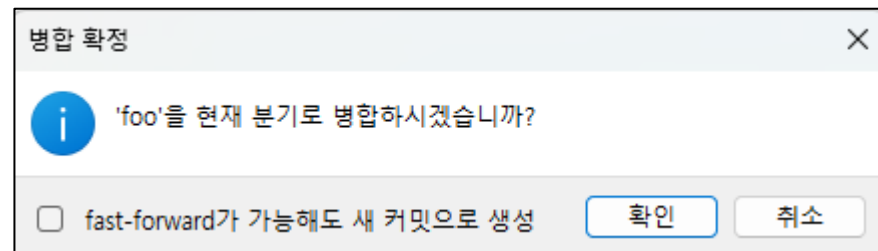
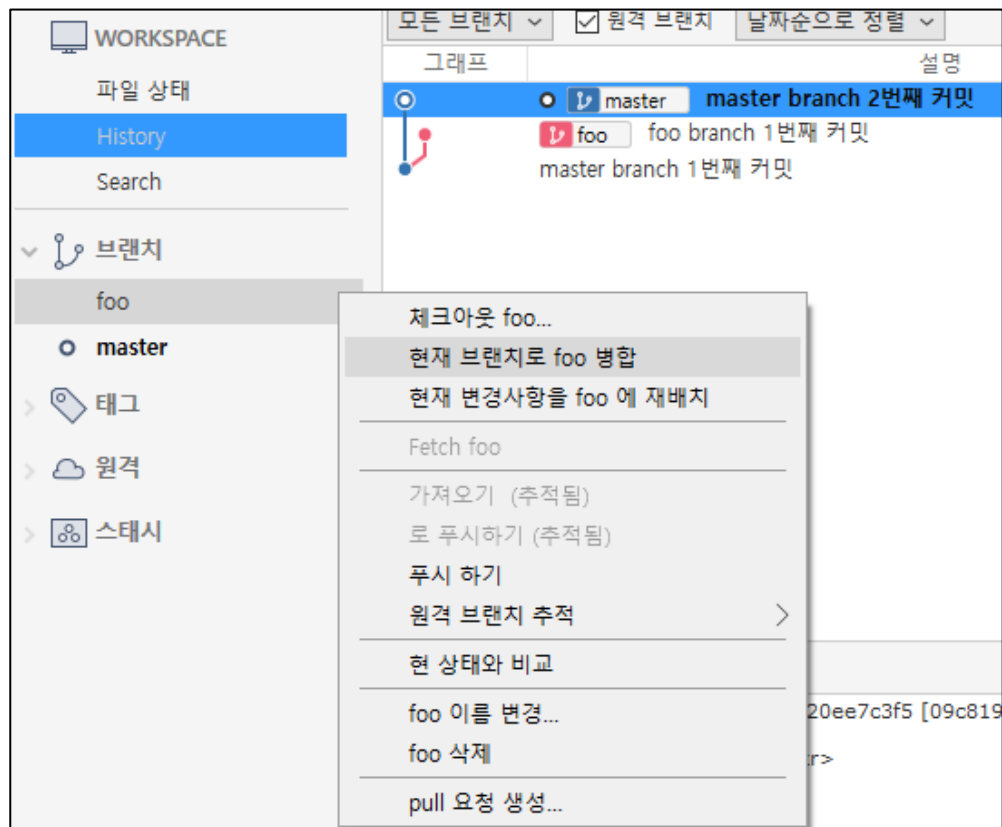
- 현재 foo 브랜치와 master 브랜치는 같은 내용을 다르게 수정한 상태
 - foo 브랜치는 a.txt 파일을 foo로 변경한 뒤 커밋
 - master 브랜치는 a.txt 파일을 master로 변경한 뒤 커밋
- 이 상태에서 두 브랜치를 병합하면 충돌이 발생!

(실습) Merge conflict

■ 브랜치 충돌 상황 만들기

● 충돌 확인을 위해 merge 진행해보기

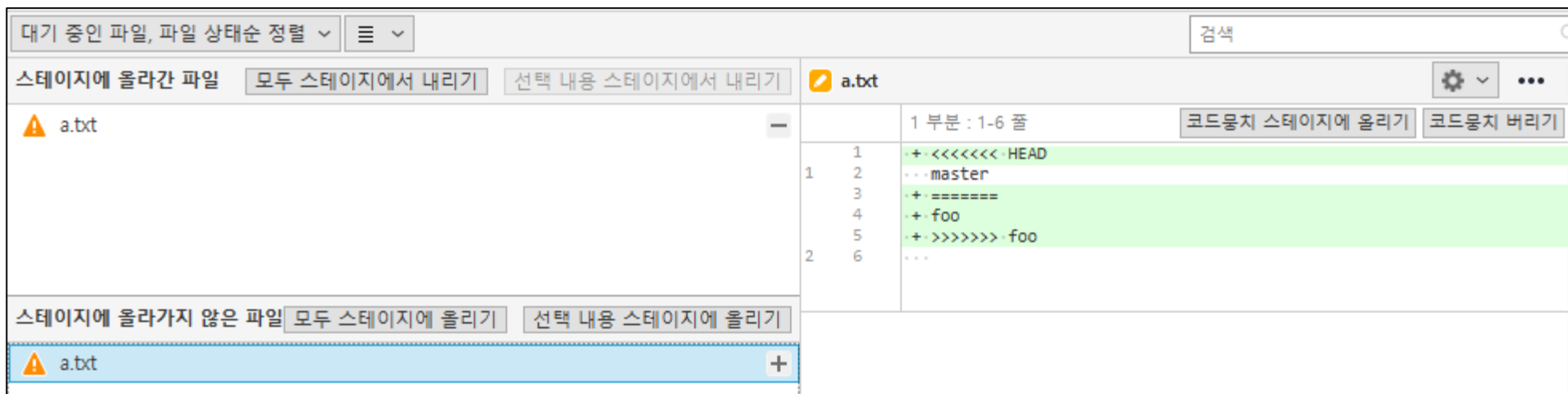
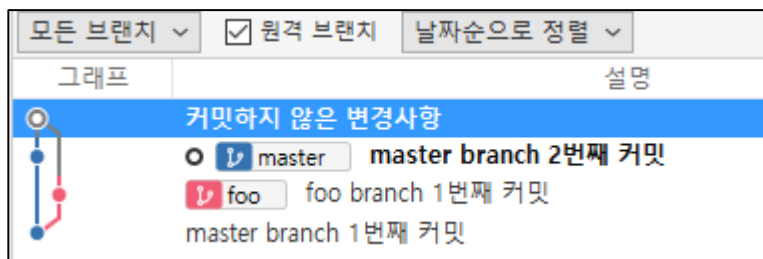
- master 브랜치로 체크아웃한 상태로 foo 브랜치에서 마우스 오른쪽 버튼을 클릭한 후 현재 브랜치로 foo 병합을 클릭.



(실습) Merge conflict

■ 브랜치 충돌 상황 만들기

- 충돌이 발생하면 다음과 같이 커밋하지 않은 변경사항이 생기고, 스테이지에 올라가지 않은 파일과 스테이지에 올라간 파일 항목에는 충돌이 발생한 파일이 추가



(실습) Merge conflict

■ 브랜치 충돌 상황 만들기

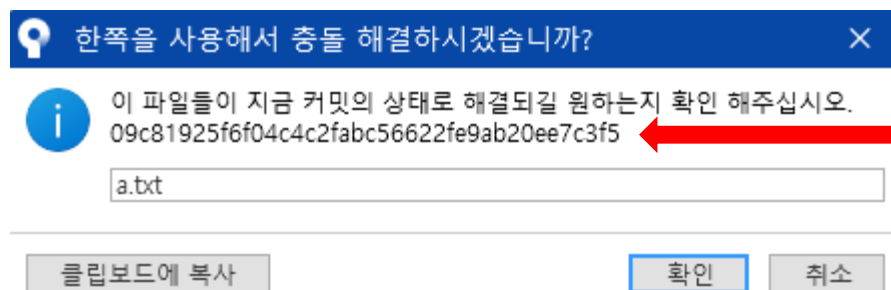
● 충돌한 파일 내용 확인

- 충돌이 발생한 파일에는 <<<<<<, >>>>>>, ===== 기호: 일종의 영역 표기
- ===== 기호를 기준으로 위부분은 HEAD가 가리키는 브랜치, 즉 현재 체크아웃한 브랜치의 내용이 적혀 있고, 아랫부분은 병합하려는 브랜치, 즉 foo 브랜치의 내용이 적혀 있음.
- 이는 <<<<<< 기호와 ===== 기호 사이의 내용을 선택할지, ===== 기호와 >>>>>> 기호 사이의 내용을 선택할지 고르라는 표기
- 여러분은 이 두 영역 중 반영할 부분을 직접 선택해 충돌을 해결해야 함

(실습) Merge conflict

■ 브랜치 충돌 해결해보기: master 브랜치 내용 반영

- 충돌이 발생한 파일, 즉 스테이지에 올라가지 않은 파일 항목에 있는 a.txt 파일에서 마우스 오른쪽 버튼을 클릭 - 충돌 해결 클릭
 - '내것'을 이용해 해결
 - 현재 체크아웃된 브랜치(HEAD, master 브랜치)의 내용을 병합에 반영하겠다는 의미
 - '저장소'것을 사용하여 해결
 - 병합하려는 브랜치(foo 브랜치)의 내용을 병합에 반영하겠다는 의미
- master 브랜치를 병합 결과로 반영할 예정이므로 '내것'을 이용해 해결을 클릭.

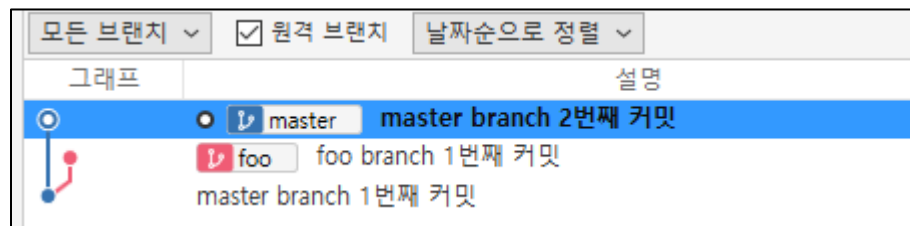


master 브랜치의 최신 커밋 해시

(실습) Merge conflict

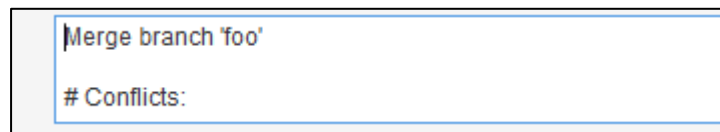
■ 브랜치 충돌 해결해보기: master 브랜치 내용 반영

- 충돌이 발생한 파일을 담고 있던 커밋하지 않은 변경사항이 사라짐: 충돌 해결!



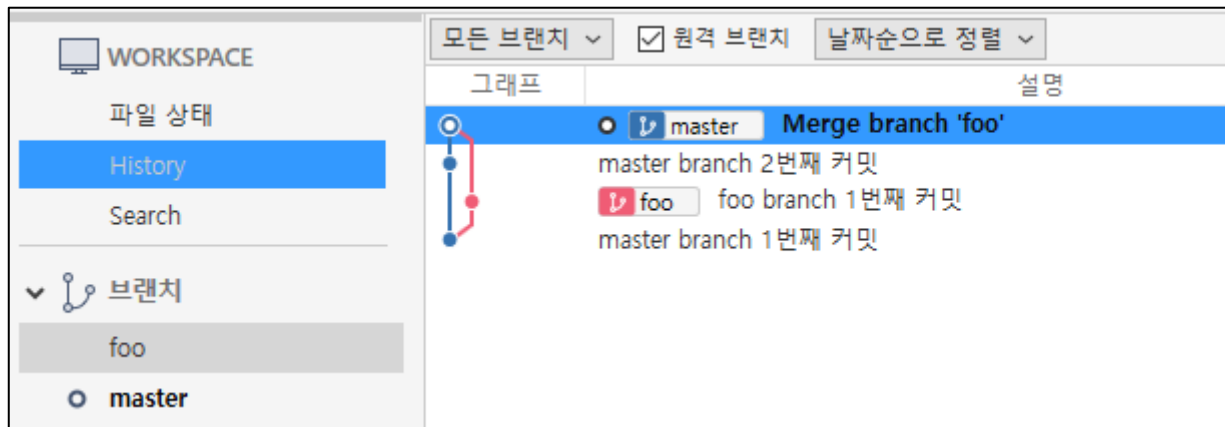
● 브랜치 병합을 끝내려면 충돌을 해결한 뒤 다시 커밋해야 함

- 충돌을 해결했다고 해서 브랜치 병합이 끝난 것이 아님
- 파일 상태로 들어가 보면, 하단부에 다음과 같이 커밋 메시지가 자동으로 기입되어 있고, 커밋이 활성화되어 있음
- 커밋 클릭하기



(실습) Merge conflict

- 브랜치 충돌 해결해보기: master 브랜치 내용 반영
 - History에서 충돌이 발생했던 foo 브랜치가 성공적으로 병합된 것을 확인할 수 있음.
 - a.txt 파일 내용도 master 브랜치의 내용으로 업데이트

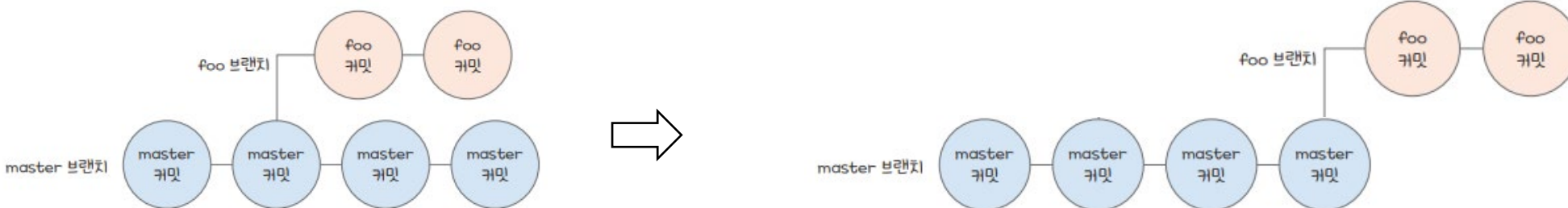


- 충돌 해결하기 요약
 - 같은 내용을 다르게 수정한 브랜치를 병합하면 충돌이 발생
 - 충돌이 발생하면 ===== 기호를 기준으로 나뉘어진 <<<<<< 기호와 >>>>>> 기호 사이의 코드 중 무엇을 반영할지 선택
 - 그런 다음 다시 커밋하면 성공적으로 병합할 수 있음.

git rebase

■ Rebase 개념

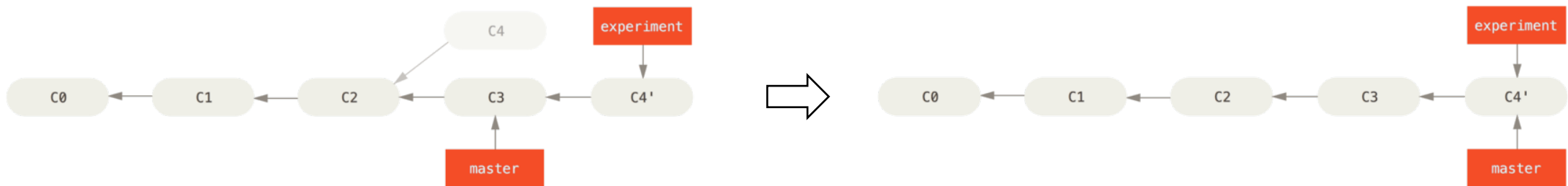
- 브랜치의 재배치
- 브랜치가 뺀어나온 기준점을 변경하는 것



git rebase

■ Rebase 동작

- 일단 두 브랜치가 나뉘기 전인 공통 커밋으로 이동하고 나서 그 커밋부터 지금 Checkout 한 브랜치가 가리키는 커밋까지 diff를 차례로 만들어 어딘가에 임시로 저장
- Rebase 할 브랜치(experiment)가 합칠 브랜치(master)가 가리키는 커밋을 가리키게 하고
아까 저장해 놓았던 변경사항을 차례대로 적용
- 적용 후 master 브랜치를 fast-forward merge 시킴.

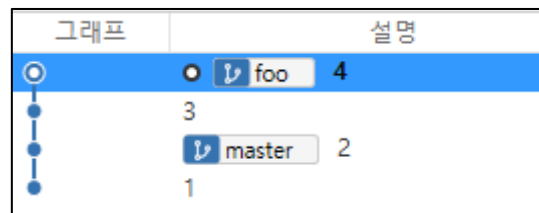
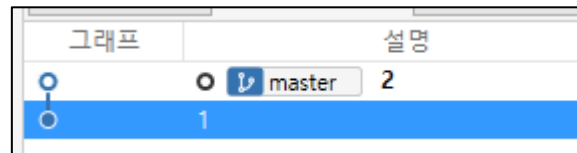


- Rebase를 하든지, Merge를 하든지 최종 결과물은 같고 커밋 히스토리만 다르다는 것이 중요하다.
- Rebase 의 경우는 브랜치의 변경사항을 순서대로 다른 브랜치에 적용하면서 합치고 Merge의 경우는 두 브랜치의 최종결과만을 가지고 합친다.

(실습) rebase

■ 예제 브랜치 만들기

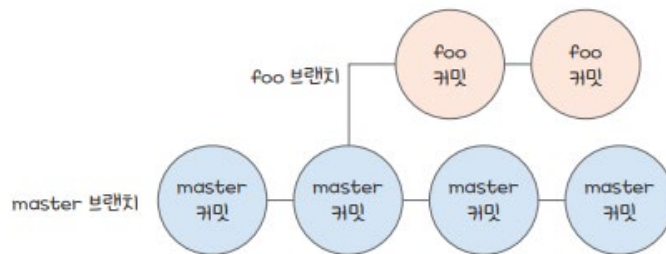
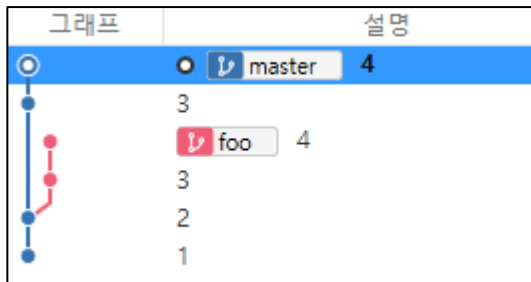
- master 브랜치에서,
 - A가 저장된 a.txt 파일을 만들어 첫 번째 커밋으로 만들기 (커밋메시지: 1)
 - B가 저장된 b.txt 파일을 만들어 이를 두 번째 커밋으로 만들기 (커밋메시지: 2)
- foo라는 새로운 브랜치를 만든 뒤 체크아웃
- foo 브랜치에서,
 - C가 저장된 foo_c.txt 파일을 만들어 이를 세 번째 커밋으로 만들기 (커밋메시지: 3)
 - D가 저장된 foo_d.txt 파일을 만들어 이를 네 번째 커밋으로 만들기 (커밋메시지: 4)



(실습) rebase

■ 예제 브랜치 만들기

- master 브랜치로 체크아웃 후,
 - C가 저장된 c.txt 파일을 만들어 이를 master 브랜치의 세 번째 커밋으로 만들기 (커밋 메시지: 3)
 - D가 저장된 d.txt 파일을 만들어 이를 네 번째 커밋으로 만들기 (커밋 메시지: 4)

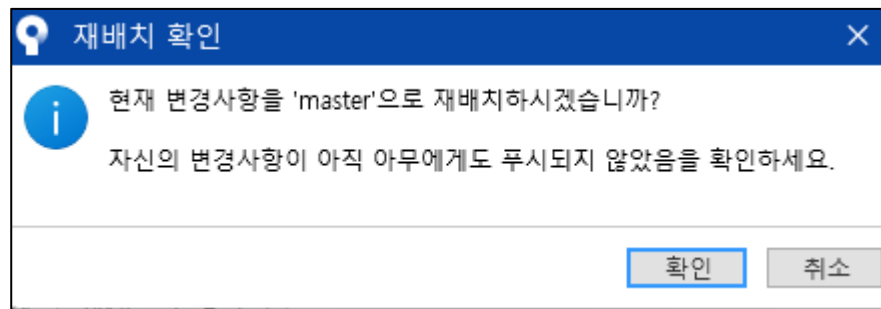
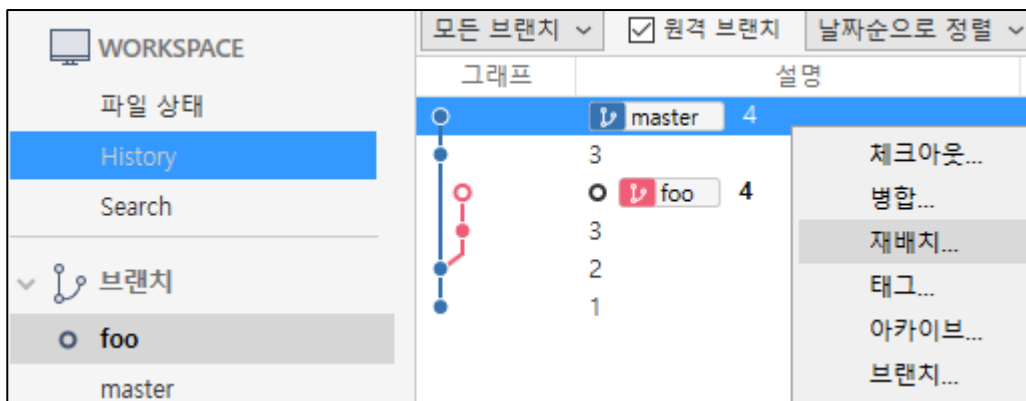


(실습) rebase

■ rebase 해보기



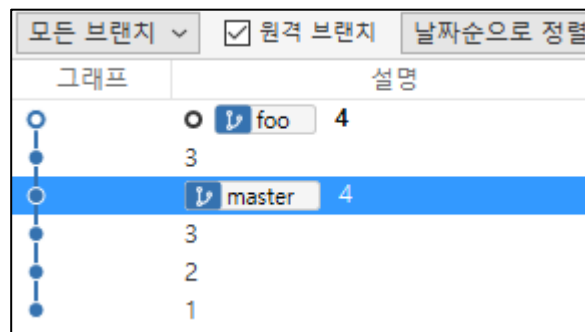
- 브랜치를 재배치하려면 재배치하려는 브랜치로 체크아웃해야 함.
 - foo브랜치로 체크아웃 !
- master 브랜치의 4번 커밋으로 브랜치를 재배치할 예정이므로 master의 네 번째 커밋에서 마우스 오른쪽 버튼을 클릭 후 재배치를 클릭



(실습) rebase

■ rebase 해보기

- 다음과 같이 foo 브랜치가 master 브랜치의 네 번째 커밋으로 재배치
- 기존에는 master 브랜치의 2번 커밋에서 뿔어나왔던 foo 브랜치가 이제는 master의 4번 커밋에서 뿔어나오게 기준점이 이동한 것



• 충돌 발생 유의

- 브랜치를 재배치하는 과정에서도 충돌이 발생할 수 있음
- 충돌이 발생한다면 당황하지 말고 앞에서 학습한 대로 충돌을 해결하면 됨