

Git Advanced

Git, Github seminar

Microsoft Student Partner
김다영

목차

1. 커밋 되돌리기

- git commit --amend
- git reset
- git revert
- git cherry-pick
- git rebase -i

2. Tag

3. 브랜치 합치기

- merge
- rebase

4. 더 알아보면 좋을 것들

Commit 되돌리기 : git commit --amend

--amend

--amend 옵션을 지정하여 커밋을 수행하면 같은 브랜치 상에서 직전에 커밋했던 내용에 새로운 내용을 추가하거나 설명을 추가할 수 있음.

- 누락된 파일을 새로 추가하거나 기존의 파일을 업데이트 해야할 때 사용
- 이전의 커밋 내용을 변경하고 싶을 때 사용

```
dayoung@gimdayeong-ui-MacBook-Pro ~ /Documents/msp > cd test
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > git init
Initialized empty Git repository in /Users/dayoung/Documents/msp/test/.git/
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master > vi a.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master > git add a.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master + git commit -m 'add a.txt'
[master (root-commit) 7cfe3a8] add a.txt
1 file changed, 1 insertion(+)
create mode 100644 a.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master > git log
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master > git commit --amend -m 'first commit'
[master 5919620] first commit
Date: Thu Aug 16 02:42:58 2018 +0900
1 file changed, 1 insertion(+)
create mode 100644 a.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test > master > git log
```

Commit 되돌리기 : git commit --amend

--amend

```
commit 7cfe3a843240be3e50cbaece193d3e89e9b0e3cb (HEAD -> master)
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:42:58 2018 +0900

    add a.txt
(END)
```



```
commit 5919620e9bd63adb97f38020910cf4ed71881365 (HEAD -> master)
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:42:58 2018 +0900

    first commit
(END)
```

Commit 되돌리기 : git reset

\$ git reset <옵션> <commit ID>

Checkout 명령시계를 다시 돌리듯이 이력(커밋)을 그 당시로 돌려놓음.

<옵션>

• --hard

- 돌아가려는 커밋 이후의 모든 작업디렉토리와 인덱스 모두 유지하지 않고 이전 커밋으로 HEAD를 되돌림
- 작업하던 내용을 의도적으로 모두 삭제하고자 할 때만 사용

• --soft

- 현재 인덱스 상태와 작업 디렉토리 내용을 그대로 보전한 채 커밋만 되돌릴 경우 사용.
- add까지 진행된 상태, \$git status로 확인하면 남아있다.
- 바로 commit 될 수 있는 상태

• --mixed

- default
- 모든 작업디렉토리는 유지하면서 인덱스를 HEAD와 함께 돌아가고자하는 커밋으로 돌아감.

모드명	HEAD의 위치	인덱스	작업 트리
Soft	변경함	변경 안 함	변경 안 함
Mixed	변경함	변경함	변경 안 함
Hard	변경함	변경함	변경함

Commit 되돌리기 : git reset Practice

Git reset 옵션 Practice

----- 앞서 만든 commit에 commit을 하나 이상 더 만든다.

1. soft

```
$ git log
```

```
$ git reset --soft <돌아가고싶은 커밋 ID>
$ git log      // HEAD 위치와 커밋 취소 확인
$ git status   // 인덱스가 변경되었는지 확인
```

```
$ git reset --hard ORIG_HEAD // 직전 reset 내역 취소 (reset하기 전으로 돌아감)
```

2. Mixed

```
$ git log
```

```
$ git reset (--mixed) <돌아가고싶은 커밋 ID>
$ git log      // HEAD 위치와 커밋 취소 확인
$ git status   // 인덱스가 변경되었는지 확인
```

```
$ git reset --hard ORIG_HEAD // 직전 reset 내역 취소 (reset하기 전으로 돌아감)
```

Commit 되돌리기 : git reset

\$ git reset <옵션> <commit ID>

\$ git reset --hard HEAD~

- : 가장 최근 커밋내용 삭제하고 되돌아감
- : (git 연습시 많이 사용)

\$ git reset HEAD~5

- : 현재로부터 5개 이전 커밋으로 돌아감

\$ git reset HEAD^

- : 최신 커밋을 취소함
- : 커밋은 했지만, push 는 하지 않은 경우에 유용

\$ git reset --hard ORIG_HEAD

- : reset 전의 커밋은 ORIC_HEAD라는 이름으로 참조할 수 있음.
- : 실수로 reset 을 한 경우에는 ORIC_HEAD로 reset하여 reset 실행 전으로 돌아갈 수 있음

git add 한 것을 commit 전에 취소하기

\$ git reset HEAD <file이름>

\$ git status

- : add 확인

Commit 되돌리기 : git revert

\$ git revert <commit ID >

이전 이력은 그대로 두고, 그 되돌릴 커밋의 코드만 원복시킴.



revert VS reset

Reset : 시계를 다시 돌리듯이 이력(커밋)을 그 당시로 돌려놓음.

Revert : 이전 이력은 그대로 두고, 그 되돌릴 커밋의 코드만 복구 시키는 커밋을 하나 만듦

Reset이 이전 커밋으로 돌아갔다면, revert는 돌이키고자 하는 커밋을 복구시키고 새로운 커밋으로 덮어씌움.

Revert는 commit을 이미 push해서 원격 저장소에 저장해버린 경우 많이 사용

Practice : reset, revert

1. Revert practice

1. 현재는 \$git log 를 확인해보면 'first commit' 만 존재.
2. b.txt 를 추가하여 'b=30' 내용을 입력
3. 새로 추가한 파일을 add, 'add b.txt' 메세지로 commit
4. b.txt 내용에 b'=50 추가
5. 새로 추가한 파일을 add, 'edit b.txt' 메세지로 commit
6. 'edit b.txt' 커밋을 취소. (맨 마지막 커밋 취소)
7. git log 확인
8. 커밋이 취소되었는지 b.txt 파일 확인

2. Reset practice

1. 방금 취소했던 커밋을 다시 실행
2. git log로 log를 확인하여
3. commit add b.txt 이후 커밋 모든 작업디렉토리는 유지하면서 인덱스를 HEAD와 함께 돌아가고자하는 커밋으로 돌아감

Practice : reset, revert

1. Revert practice

```
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test master vi b.txt
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test master git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    b.txt

nothing added to commit but untracked files present (use "git add" to track)
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test master git add b.txt
x dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test master + git commit -m 'add b.txt'
[master 136f5dd] add b.txt
 1 file changed, 1 insertion(+)
 create mode 100644 b.txt
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test master git log
commit 136f5dda81320390432f24a1d84b5d15408c6ee1 (HEAD -> master)
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:59:11 2018 +0900

  add b.txt

commit 5919620e9bd63adb97f38020910cf4ed71881365
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:42:58 2018 +0900

  first commit
(END)
```

Practice : reset, revert

1-6. 'edit b.txt' 커밋을 취소. (맨 마지막 커밋 취소)

```
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master vi b.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master git add b.txt
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

modified:   b.txt

dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master git reset 136f5dda81320
[master cf1eb7e] dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master git status
1 file changed
On branch master
dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master nothing to commit, working tree clean
commit cf1eb7ee2e dayoung@gimdayeong-ui-MacBook-Pro ~/Documents/msp/test master git log
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 03:03:21 2018 +0900

edit b.txt

commit 136f5dda81320390432f24a1d84b5d15408c6ee1
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:59:11 2018 +0900

add b.txt

commit 5919620e9bd63adb97f38020910cf4ed71881365
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:42:58 2018 +0900

first commit
(END)
```

b.txt
b=30
b'=50

Git

Commit 되돌리기

1_6. 맨 마지막 커밋 취소

```
dayoung@gimdayeong-ui-MacBook-Pro ~ /Documents/msp/test master git revert HEAD
```

```
Revert "edit b.txt"
```

```
This reverts commit cf1eb7ee2eff2855fc1415bc8842e1762221acc.
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   b.txt
#
```

1-7. git log 확인

```
commit 9ae08dcd0fc8e2b0582d3ec8c767c368d75361a (HEAD -> master)
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 03:08:33 2018 +0900
    Revert "edit b.txt"
    This reverts commit cf1eb7ee2eff2855fc1415bc8842e1762221acc.

commit cf1eb7ee2eff2855fc1415bc8842e1762221acc
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 03:03:21 2018 +0900
    edit b.txt

commit 136f5dda81320390432f24a1d84b5d15408c6ee1
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:59:11 2018 +0900
    add b.txt

commit 5919620e9bd63adb97f38020910cf4ed71881365
Author: ekdud2412 <dayoung9650@gmail.com>
Date:   Thu Aug 16 02:42:58 2018 +0900
    first commit
(END)
```

Commit 을 revert 했다고
commit 이 생김

1_8. 커밋이 취소되었는지 b.txt 파일 확인

```
dayoung@gimdayeong-ui-MacBook-Pro ~ /Documents/msp/test master gedit b.txt
dayoung@gimdayeong-ui-MacBook-Pro ~ /Documents/msp/test master vi b.txt
```

```
b=30
~
```

Practice : reset, revert

2.

```
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test [master] git reset 136f5dda81320  
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test [master] git status  
On branch master  
nothing to commit, working tree clean  
dayoung@gimdayeong-ui-MacBook-Pro ~Documents/msp/test [master] git log  
commit 136f5dda81320390432f24a1d84b5d15408c6ee1 (HEAD -> master)  
Author: ekdud2412 <dayoung9650@gmail.com>  
Date: Thu Aug 16 02:59:11 2018 +0900  
add b.txt  
  
commit 5919620e9bd63adb97f38020910cf4ed71881365  
Author: ekdud2412 <dayoung9650@gmail.com>  
Date: Thu Aug 16 02:42:58 2018 +0900  
first commit  
(END)
```

Mixed 옵션은 워킹 트리도 이전 커밋으로 돌아감

Defualt 옵션: --mixed

* 만약 soft 옵션이었다면 add 한 상태까지 남아있을 것.

지정한 커밋으로 되돌아감.

Commit 되돌리기 : git cherry-pick

\$ git cherry-pick <옵션> <commit ID >

다른 브랜치에서 지정한 커밋을 복사하여 현재 브랜치로 가져올 수 있음

- 특정 브랜치에 잘못 추가한 커밋을 올바른 브랜치로 옮기고 싶을 때
- 다른 브랜치의 커밋을 현재 브랜치에도 추가하고 싶을 때

<옵션>

-n : add까지만 수행 commit은 수행하지 않음.

<3개의 커밋을 하나의 커밋으로 합쳐서 반영>

```
$ git cherry-pick -n asd123  
$ git cherry-pick -n fghdf45  
$ git cherry-pick -n tji3545
```

```
$ git commit -m 'cherry commit'
```

Commit 되돌리기 : git rebase -i

\$ git rebase -i <commit ID >

1. 커밋 통합

커밋들을 합치고 싶을 때 사용
작은 커밋들을 합쳐서 하나의 큰 커밋으로 만들

\$ git rebase -i HEAD~~
: 이전의 두 커밋을 합침

Pick 문자를 **squash**로 변경

```
pick 009a258 edit sample.txt and add sample2.txt
pick ba711e7 add revert.txt

# Rebase be9996e..ba711e7 onto be9996e (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
#       However, if you remove everything, the rebase will be aborted.
#
#
# Note that empty commits are commented out
```

Tag

\$ git tag <옵션> <tag name>

Commit을 참조하기 쉽도록 알기 쉬운 이름을 붙이는 것.

한 번 붙인 태그는 브랜치처럼 위치가 이동하지 않고 고정됨.

1. Lightweight tag (일반 태그)

: 이름만 붙일 수 있다.

2. Annotated tag (주석 태그)

: 이름

: 태그에 대한 설명

: 서명

: 이 태그를 만든 사람의 이름, 이메일과 날짜 정보

릴리즈 브랜치에서는 보통 주석태그를 사용하여 설명이나 서명을 넣은 태그를 사용하고,

토픽 브랜치처럼 로컬에서 일시적으로 사용할 땐 이름만 붙이는 태그 사용

Tag

```
$ git tag <옵션> <tag name>
```

1. 일반 태그

```
$ git tag one  
: 현재 HEAD 가 가리키고 있는 커밋에 one라는 태그를 붙임  
$ git tag  
: tag 목록 조회  
$ git log --decorate  
:태그 정보를 포함한 이력 조회
```

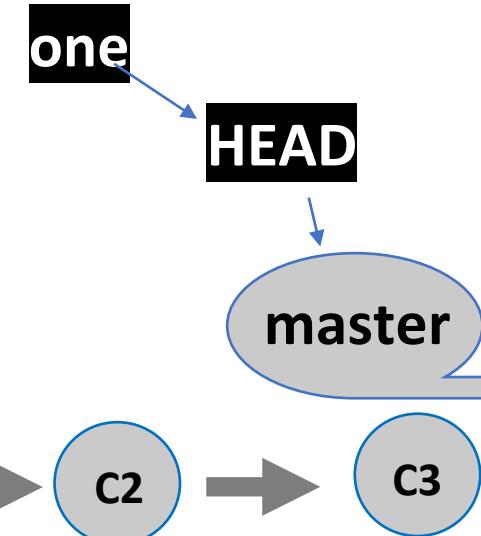
2. 주석태그

```
$ git tag -am "git tag teat" two  
: 현재 HEAD 가 가리키고 있는 커밋에 two라는 태그와 설명을 붙임
```

```
$ git tag -n  
: 태그 목록과 주석 내용 확인
```

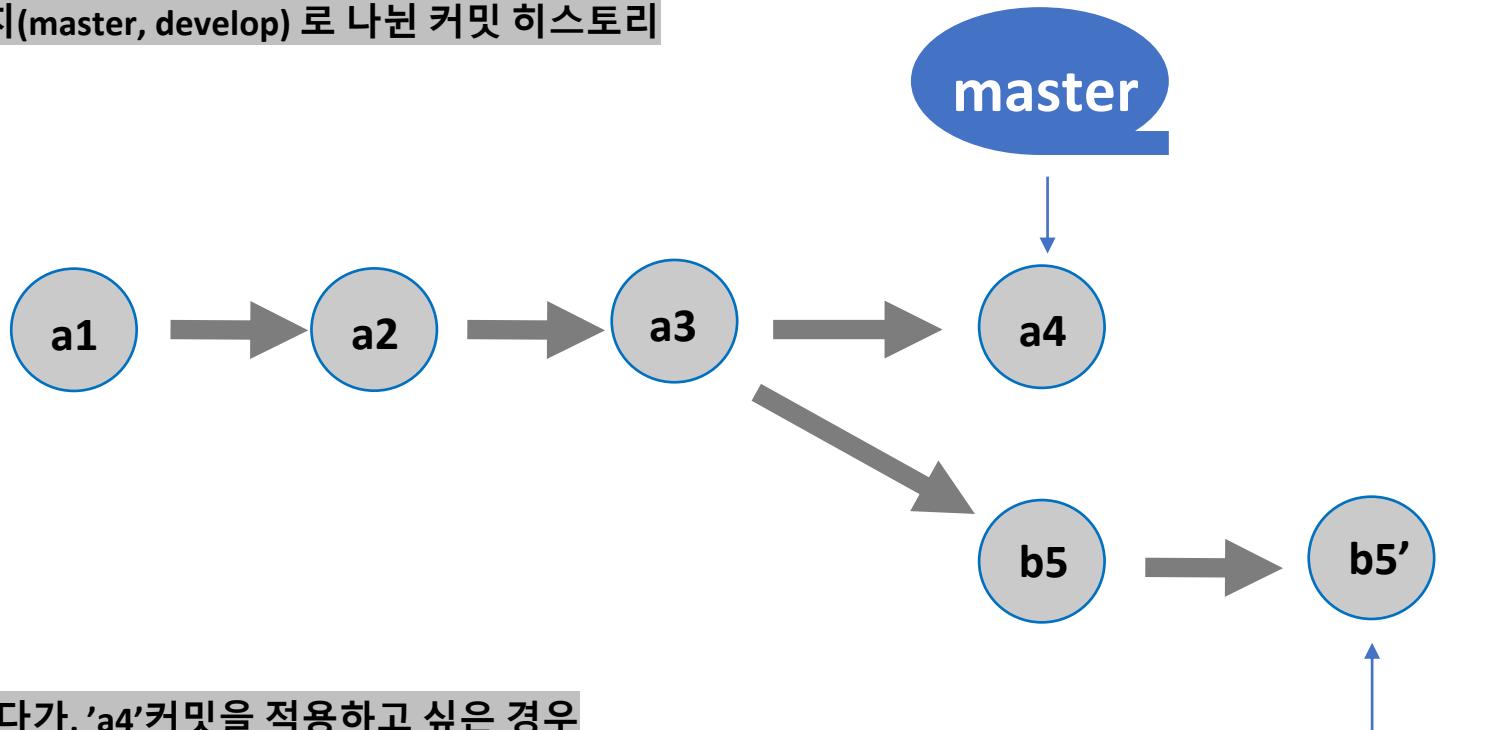
3. 태그 삭제

```
$ git tag -d <tag name>
```



브랜치 합치기: git merge < commit | branch >

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



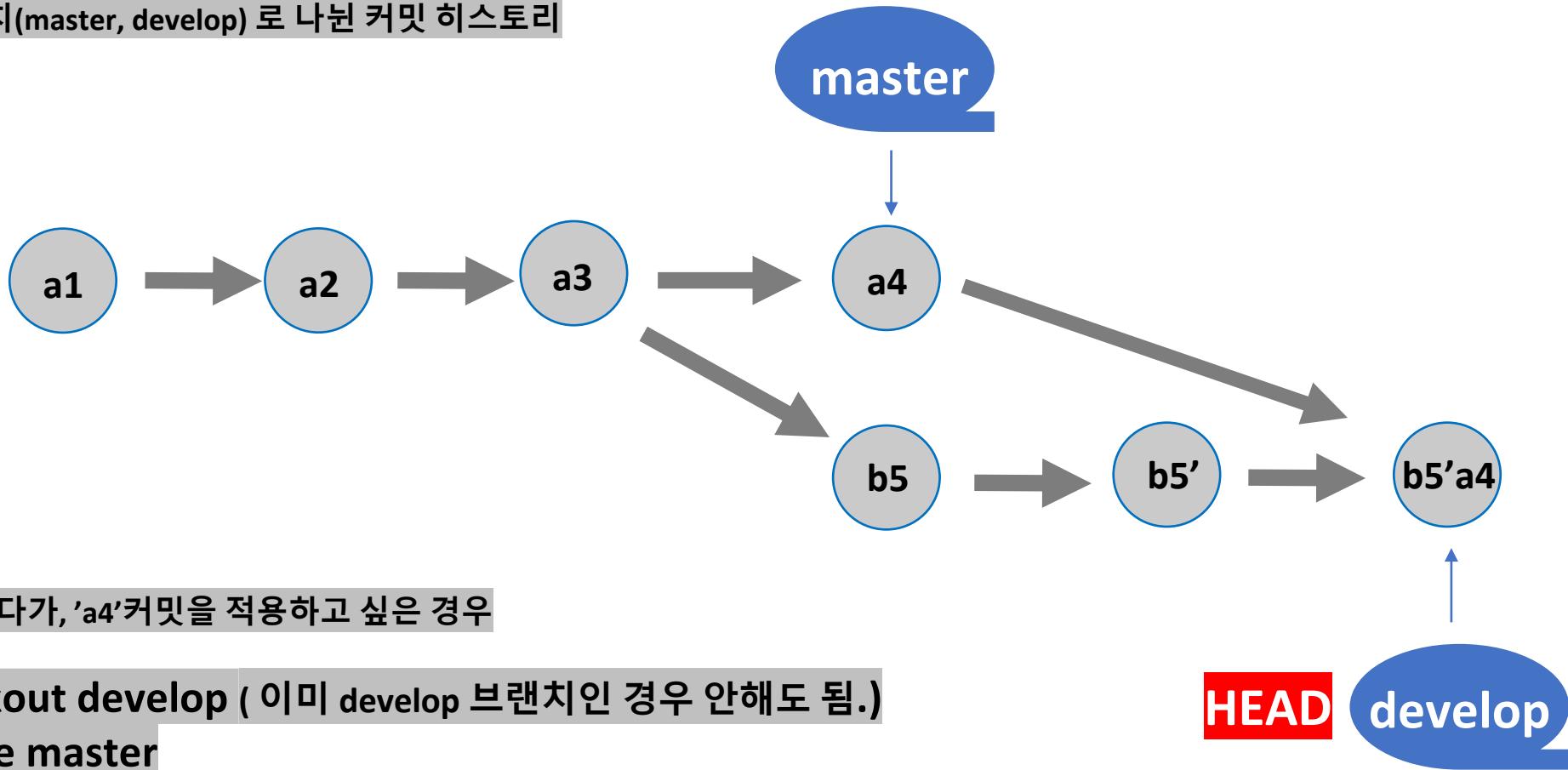
'b5'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

```
$ git checkout develop ( 이미 develop 브랜치인 경우 생략)  
$ git merge master
```

HEAD **develop**

브랜치 합치기: git merge < commit | branch >

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



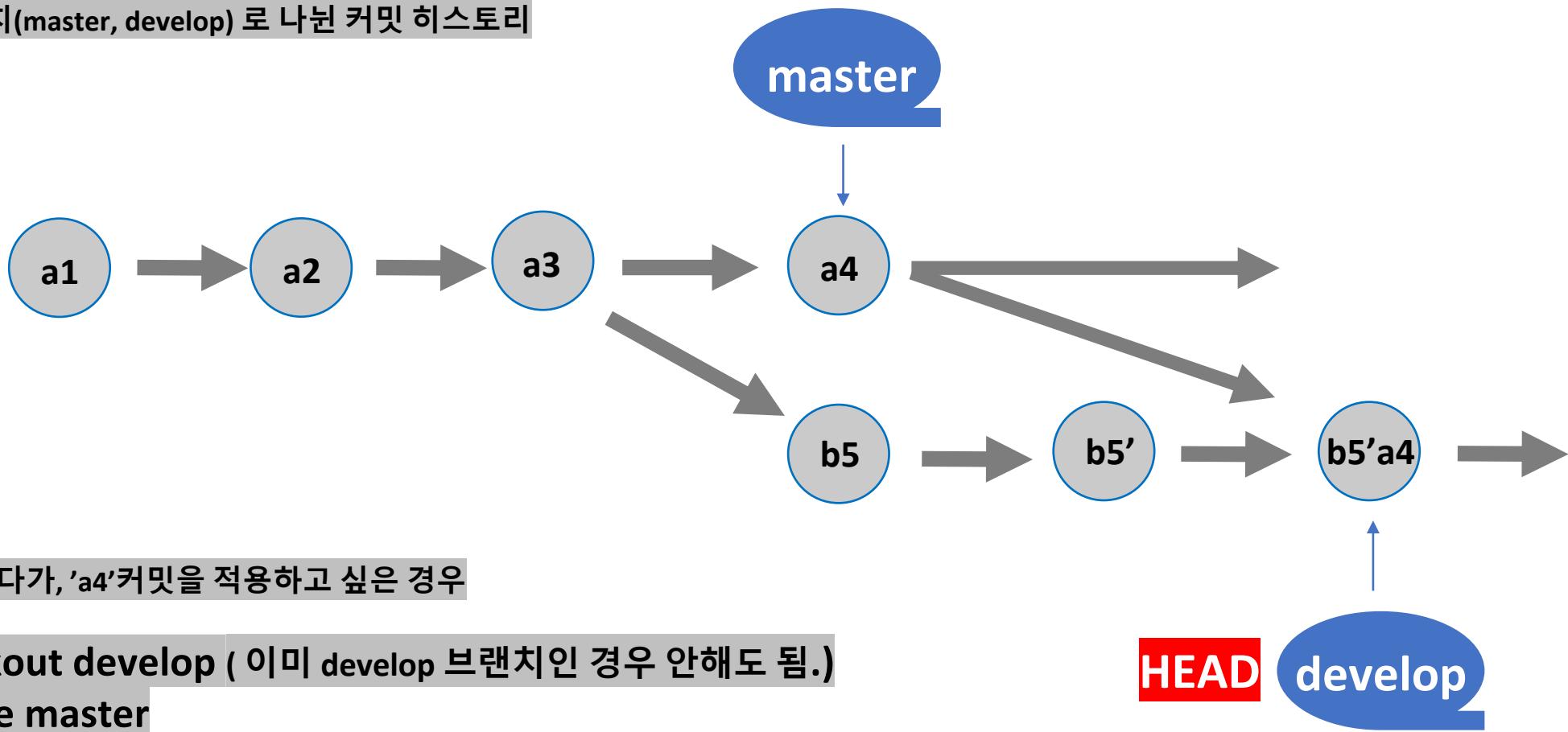
'b5'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

```
$ git checkout develop ( 이미 develop 브랜치인 경우 안해도 됨.)  
$ git merge master
```

HEAD
develop

브랜치 합치기: git merge < commit | branch >

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



'b5'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

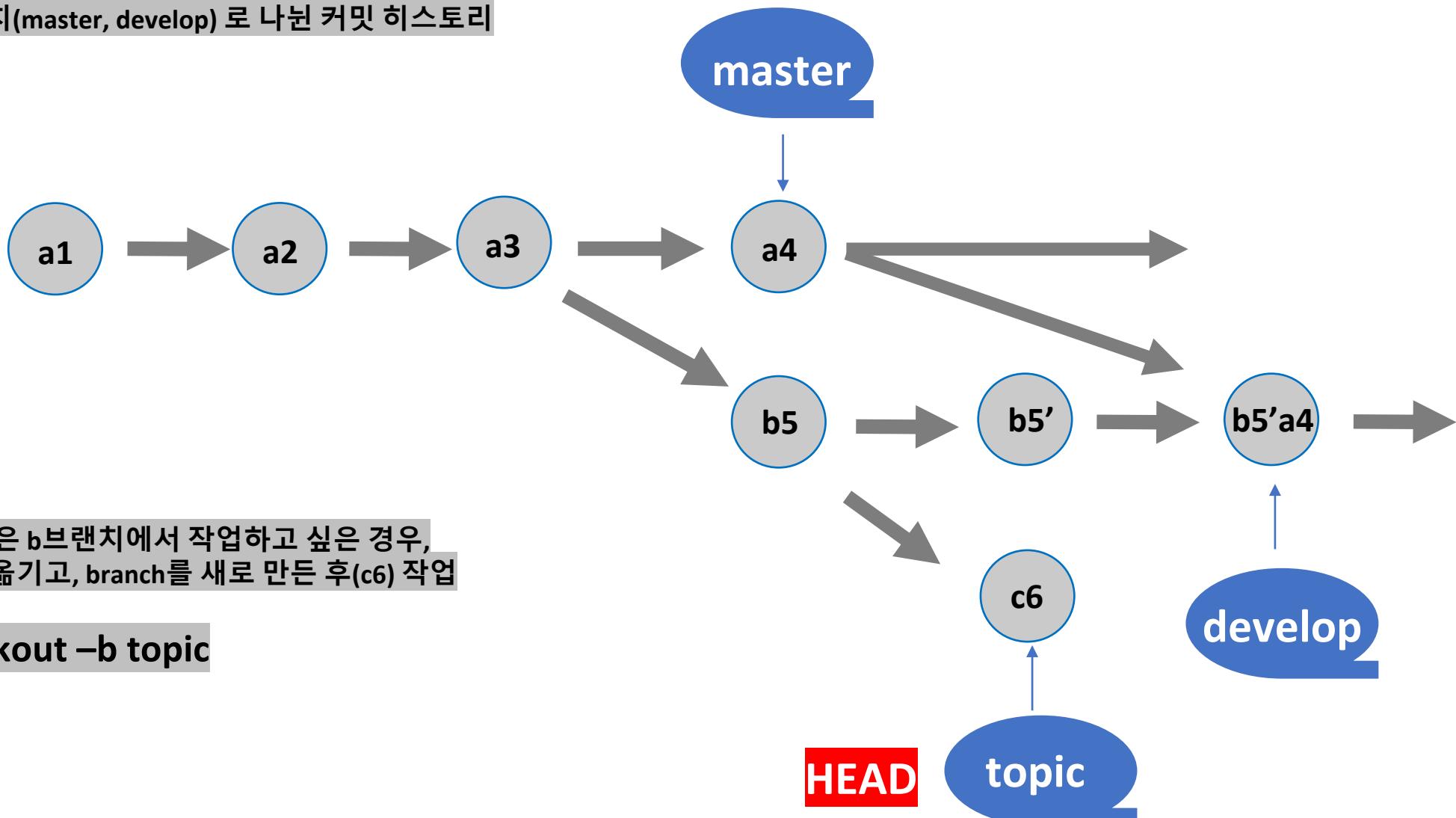
">\$ git checkout develop (이미 develop 브랜치인 경우 안해도 됨.)

```
$ git merge master
```

HEAD develop

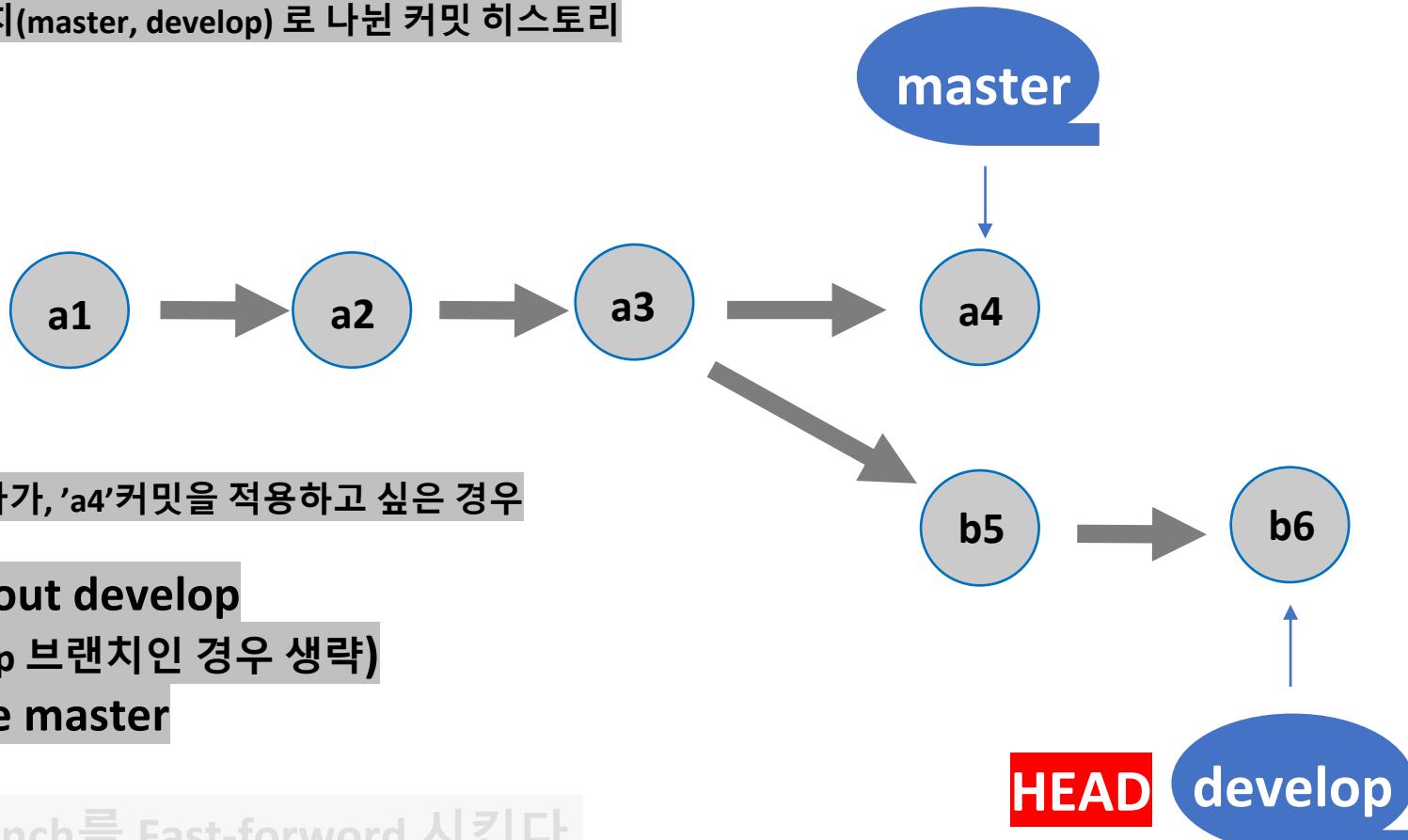
브랜치 합치기: git merge < commit | branch >

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



브랜치 합치기: rebase

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



'b6'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

\$ git checkout develop

(이미 develop 브랜치인 경우 생략)

\$ git rebase master

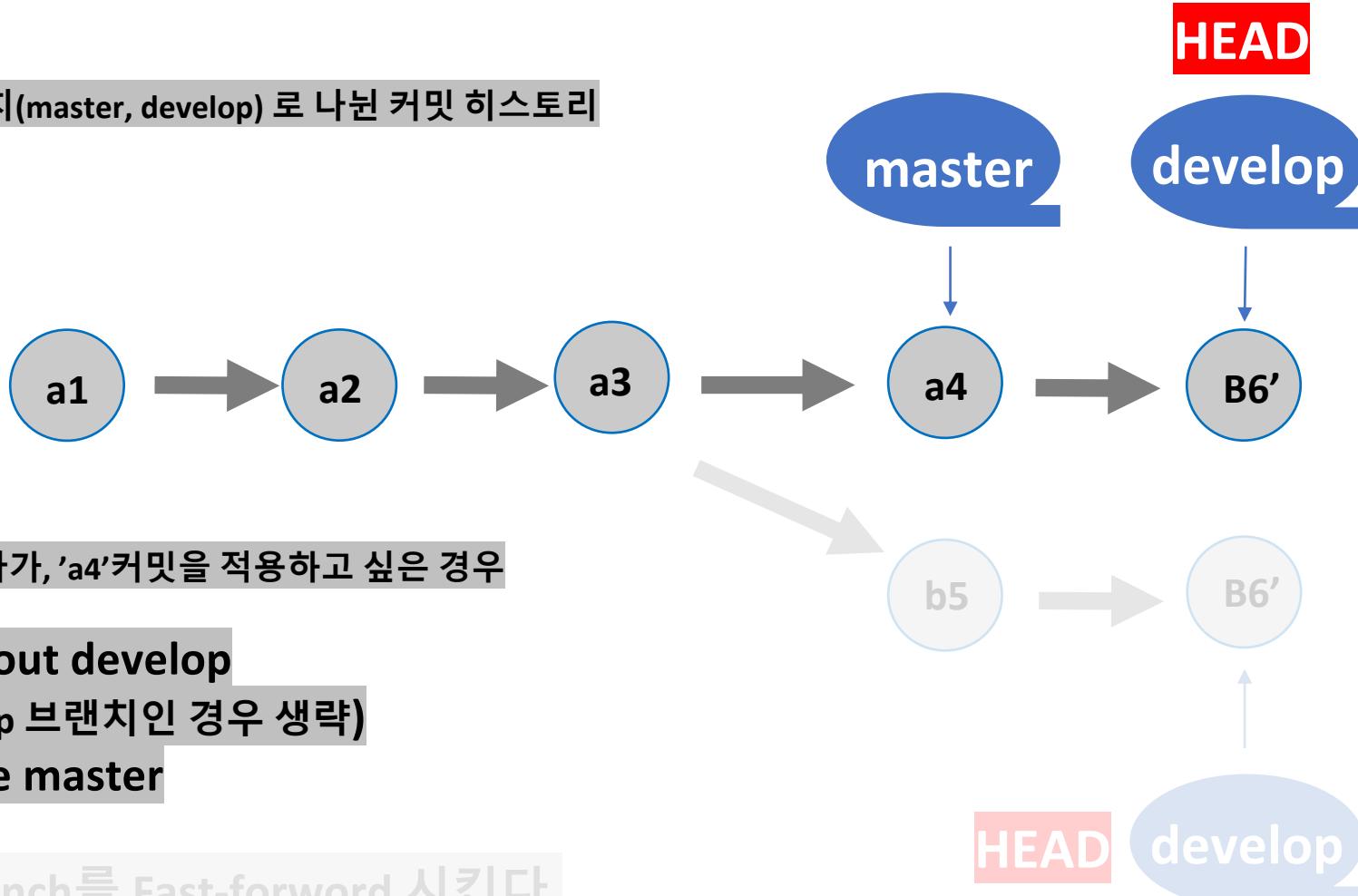
Master branch를 Fast-forward 시킨다.

\$ git checkout master

\$ git merge develop

브랜치 합치기: rebase

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



'b6'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

\$ git checkout develop

(이미 develop 브랜치인 경우 생략)

\$ git rebase master

Master branch를 Fast-forward 시킨다.

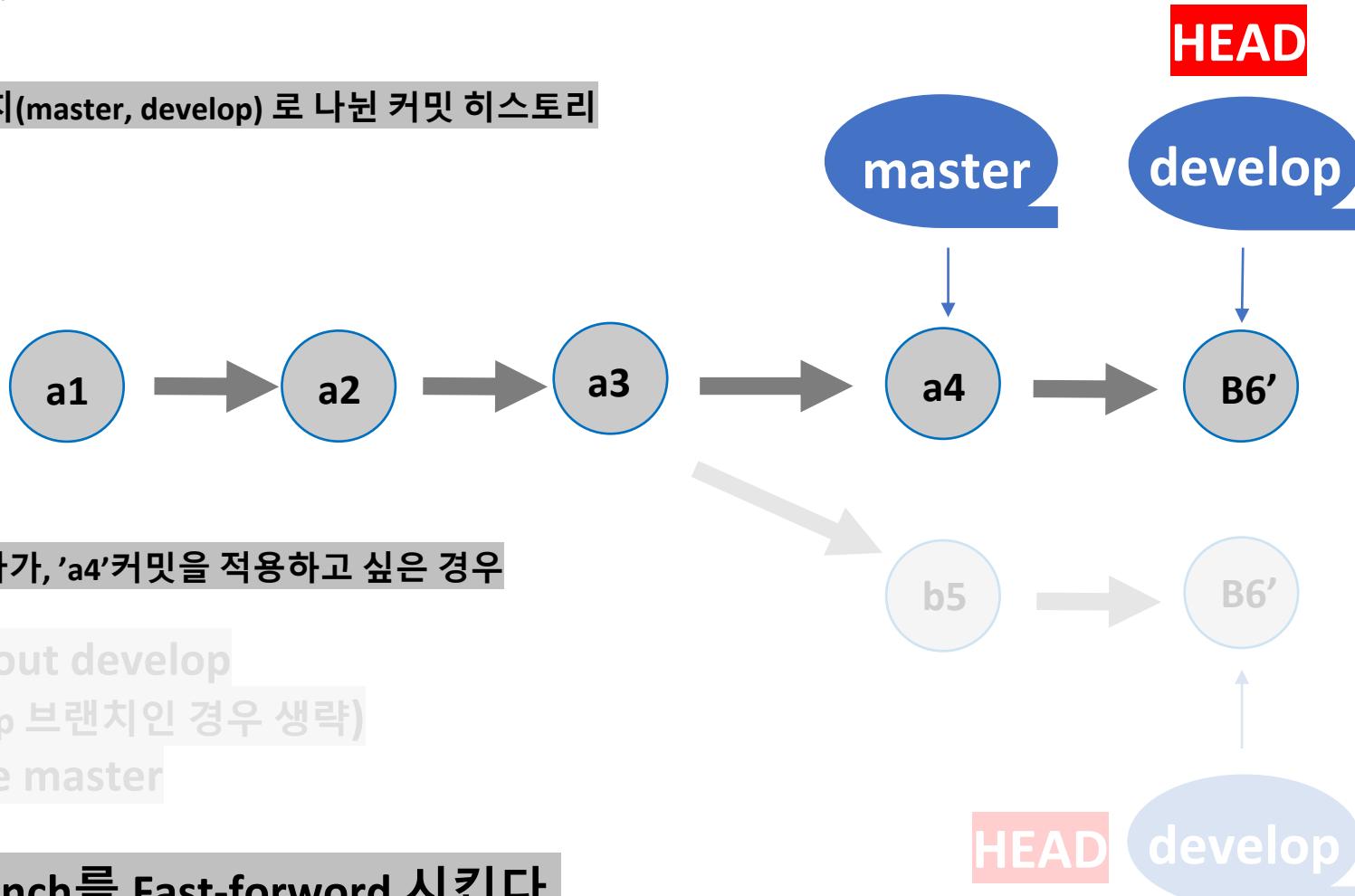
\$ git checkout master

\$ git merge develop

HEAD
develop

브랜치 합치기: rebase

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



Master branch를 Fast-forward 시킨다.

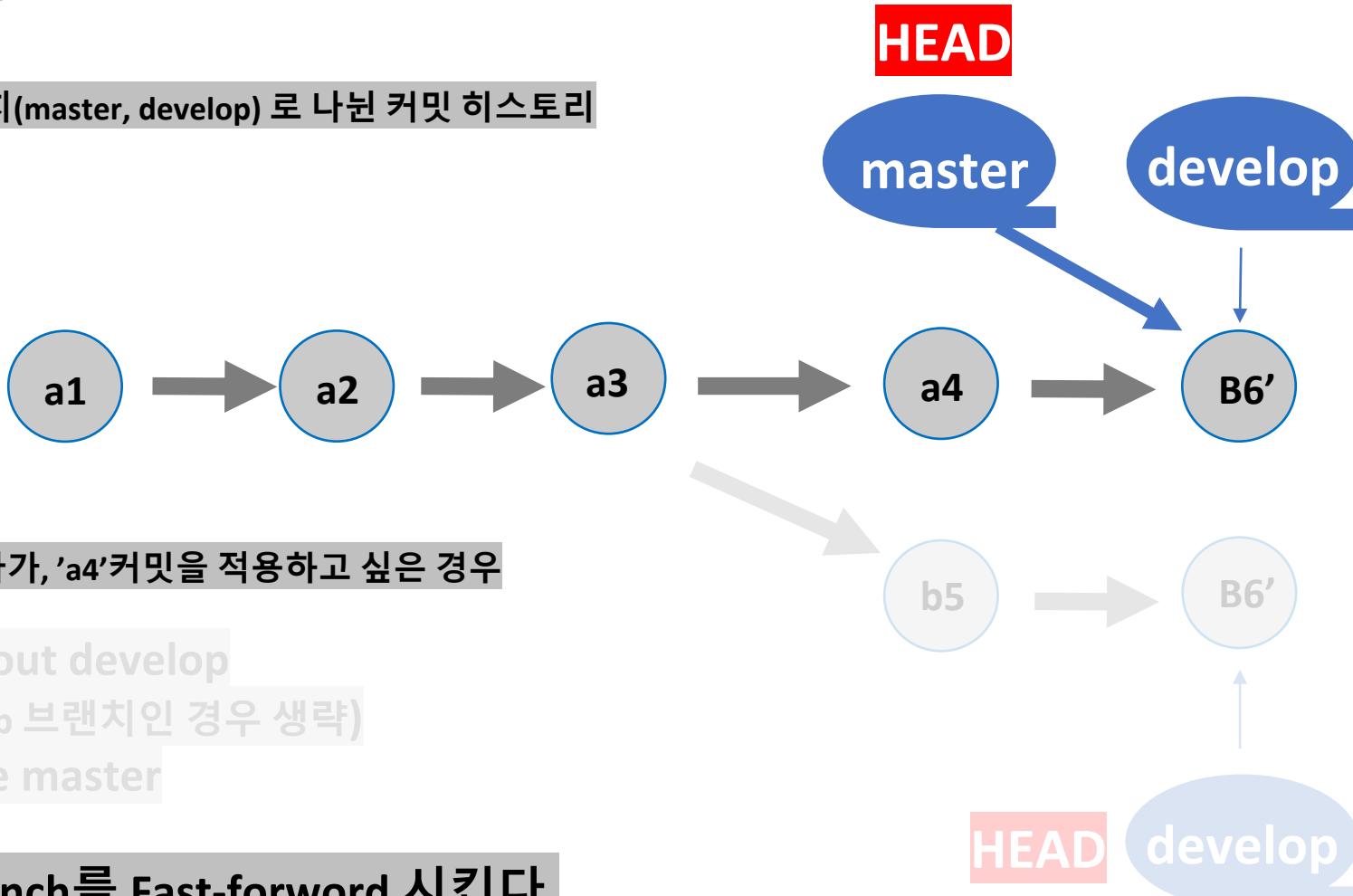
```
$ git checkout master
```

```
$ git merge develop
```

HEAD
develop

브랜치 합치기: rebase

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



'b6'를 진행하다가, 'a4'커밋을 적용하고 싶은 경우

```
$ git checkout develop
```

(이미 develop 브랜치인 경우 생략)

```
$ git rebase master
```

Master branch를 Fast-forward 시킨다.

```
$ git checkout master
```

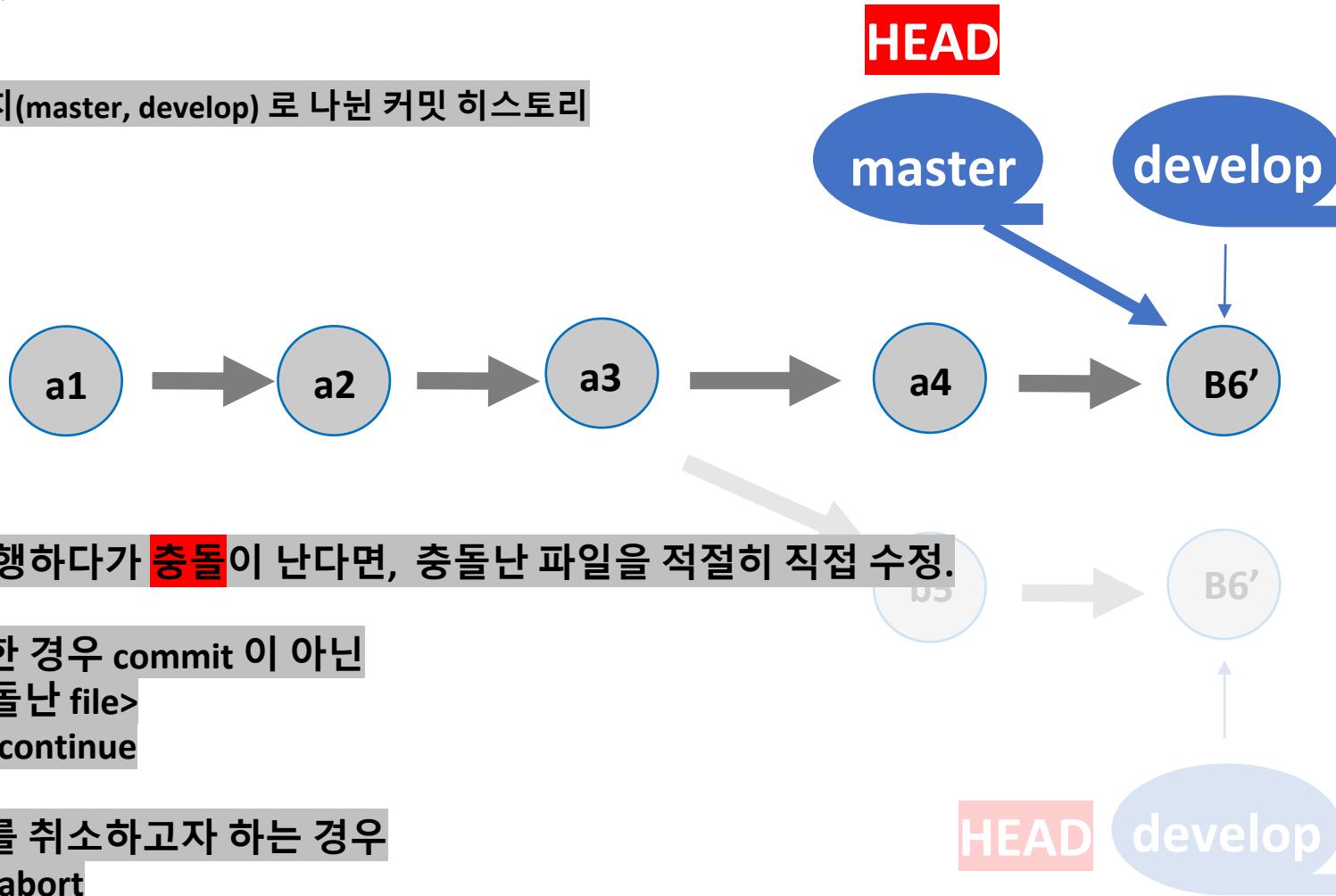
```
$ git merge develop
```

HEAD

develop

브랜치 합치기: rebase

두 개의 브랜치(master, develop)로 나눈 커밋 히스토리



브랜치 합치기: merge 와 rebase 의 차이점

브랜치에서 하던 일을 완전히 마치고 origin/master 로 **rebase** 한다.

- : 1. 기존 브랜치의 커밋 diff를 차례로 만들어 어딘가에 저장
- : 2. rebase 할 브랜치는 rebase할 브랜치로 (origin/master) 커밋을 가리키게 하고 저장해놓았던 변경사항을 차례로 적용하며 커밋을 새로 만든다.
- : 3. **master** 브랜치에서 merge (Fast-forward) 한다. -> 프로젝트 관리자는 어떠한 통합작업도 필요없이 이 작업만 하면 됨

차이점

최종 결과물은 같고 커밋 히스토리만 다르다는 것이 중요. Rebase의 경우는 브랜치의 변경사항을 순서대로 다른 브랜치에 적용하면서 합치고 Merge의 경우 두 브랜치의 최종본만 가지고 합친다.

브랜치 합치기: merge 와 rebase 의 차이점

그렇다면 언제 rebase를 사용하고, 언제 merge를 사용할까 ?

상황에 따라 다르게 사용

완료된 기능 브랜치를 다시 합칠 때는 merge 사용

Rebase 는 보통 리모트 브랜치에 커밋을 깔끔하게 적용하고 싶을 때 사용

메인 프로젝트에 패치를 보낼 준비가 되면 하는 것

기능 브랜치에 부모 브랜치 변경 내용을 반영하고 싶을 때

1. rebase

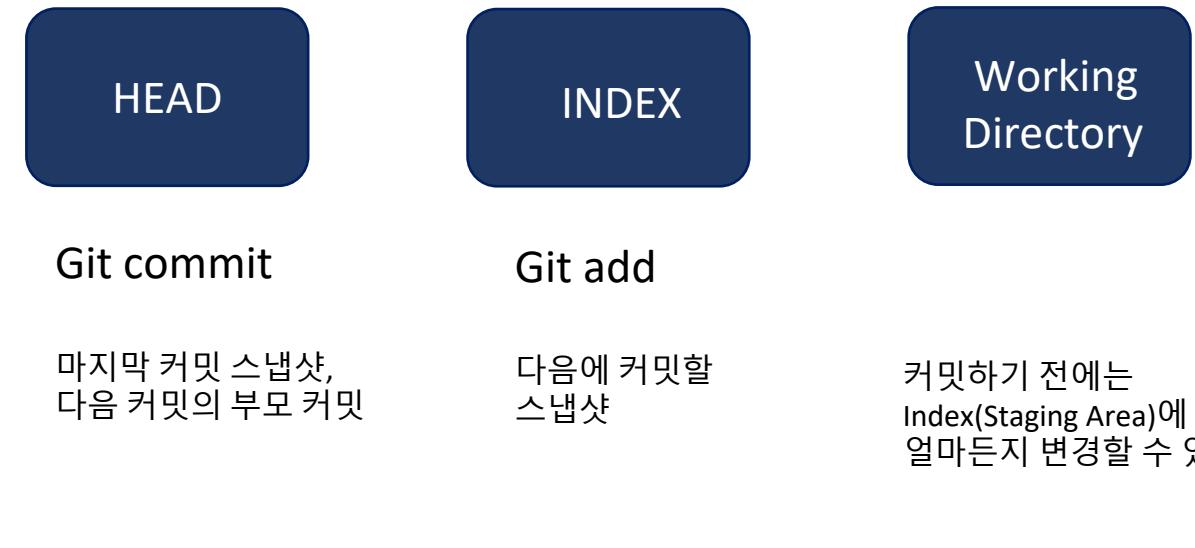
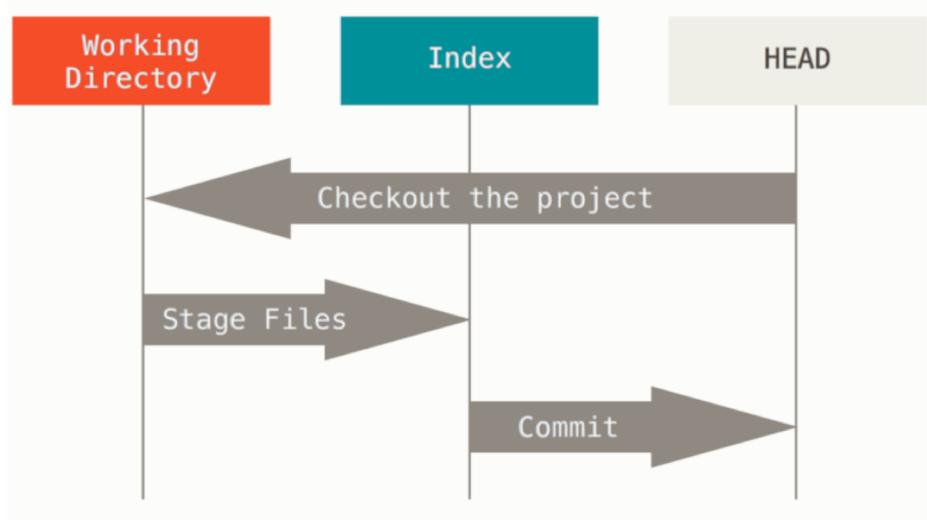
- 이 브랜치를 다른 곳에 푸시한 적이 없는 경우
- 다른 사람이 이 기능 브랜치를 체크아웃 할 일이 없을 것이라 확신하는 이유

2. merge

- 이외의 상황은 merge

Reset 과 checkout에 대하여

이 두 명령은 깃을 처음 접해보는 사람들이 가장 헷갈려 하는 것 중 하나.



감사합니다.