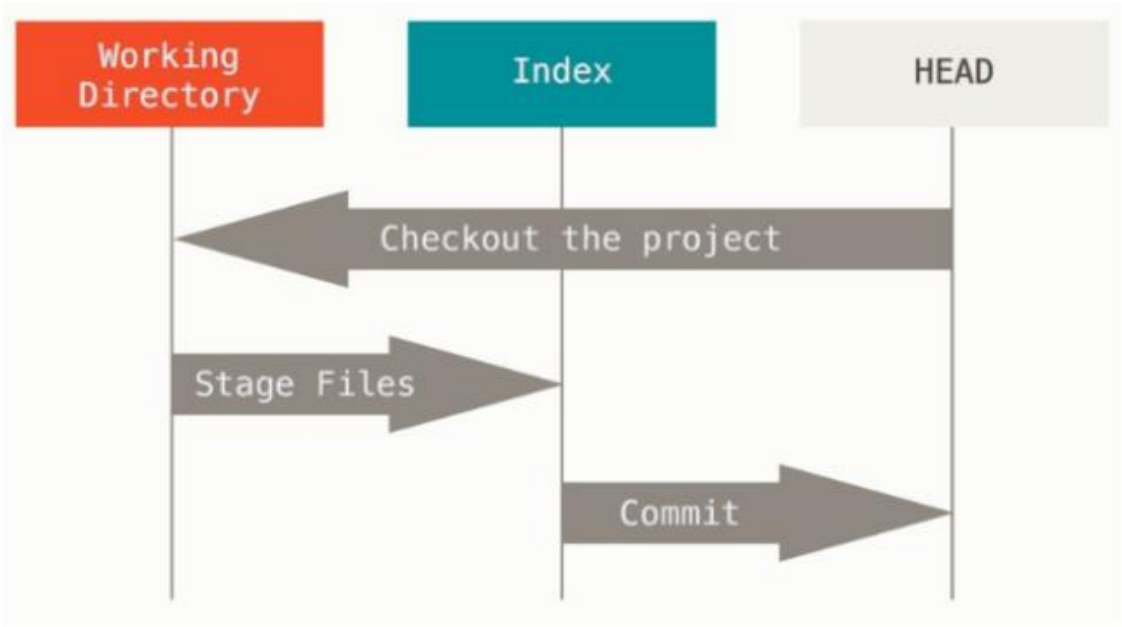
Git Advanced

0. Git 기타 개념

HEAD: 현재 branch 의 포인트
Index: staging area 에 위치한 다음 commit snapshot 이 예정된 것
Working Directory: 수정하고 있는 폴더

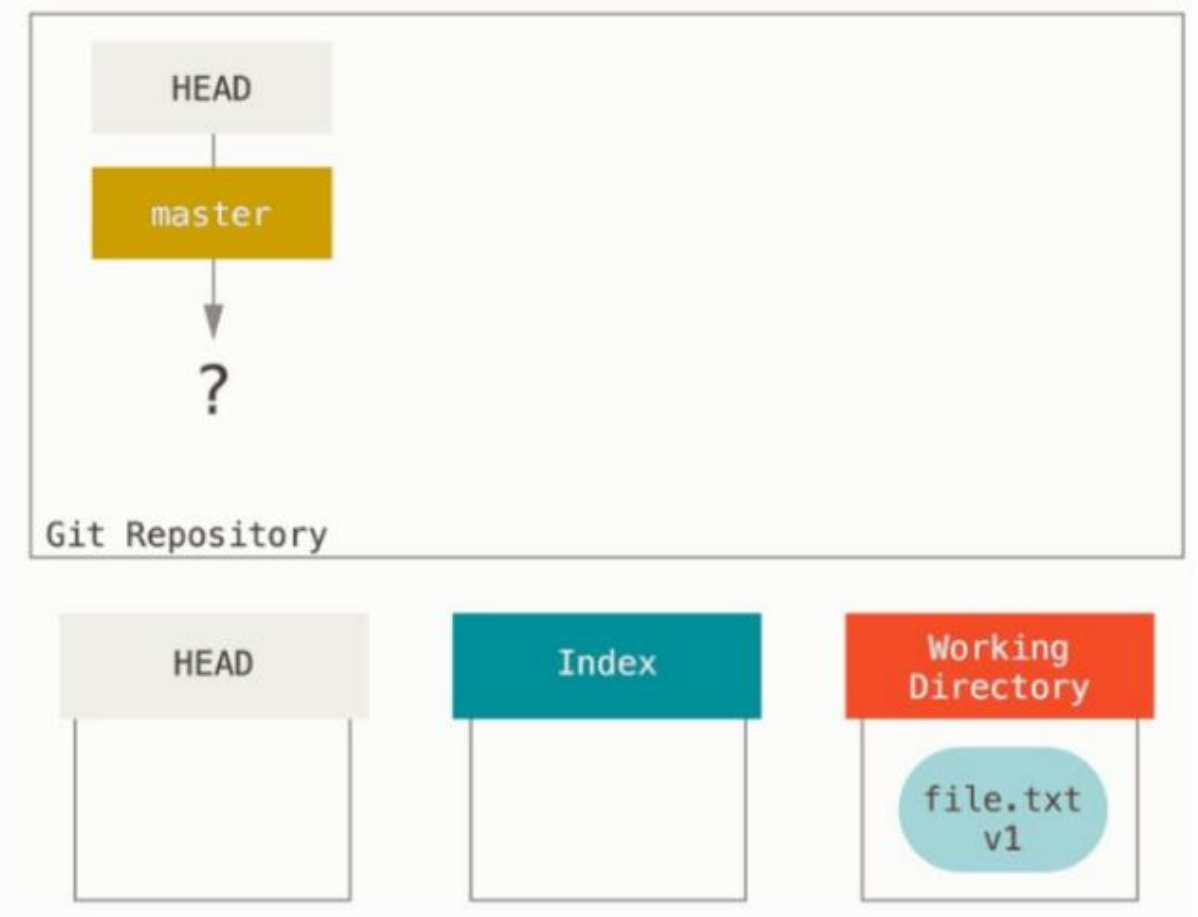
1) Git working flow



전반적인 working flow

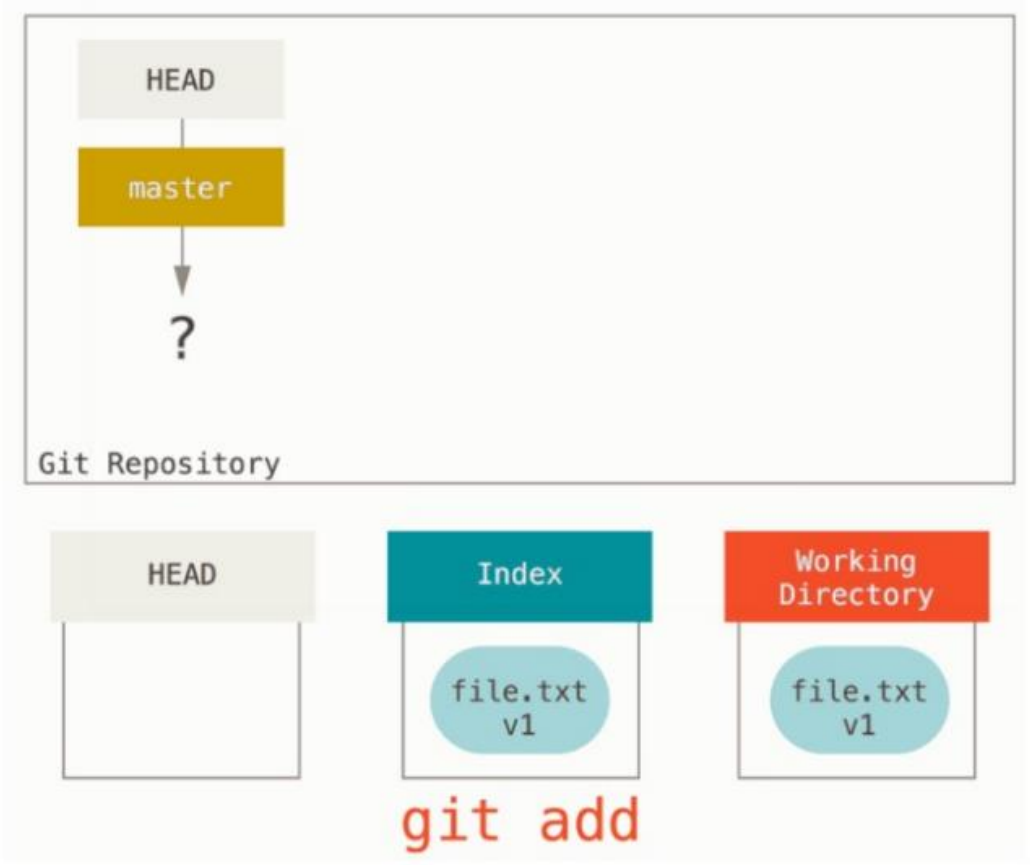
2) add 와 commit 과 관련된 간단한 flow 예시

(1) git init 을 통해 깃 폴더를 생성한다.

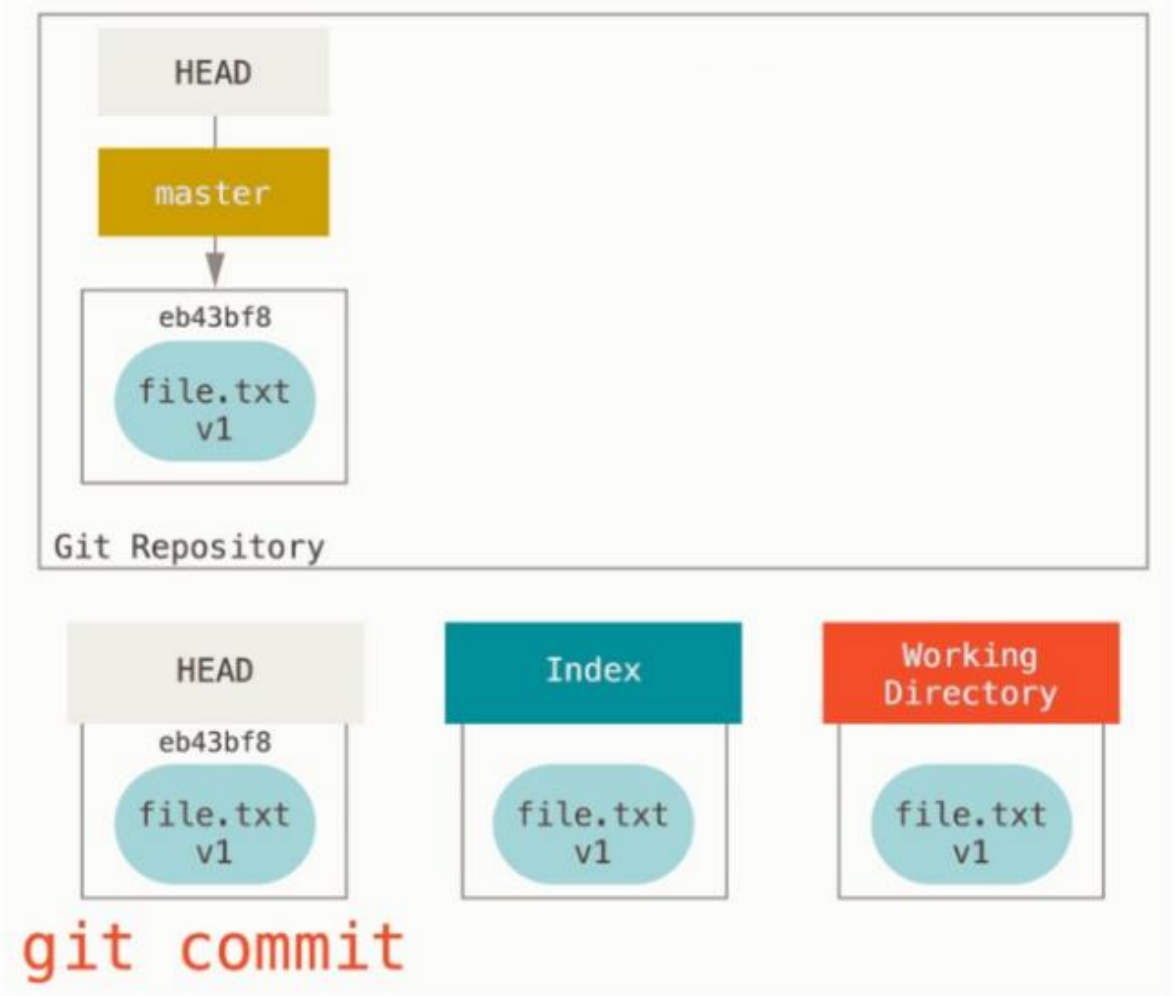


git init

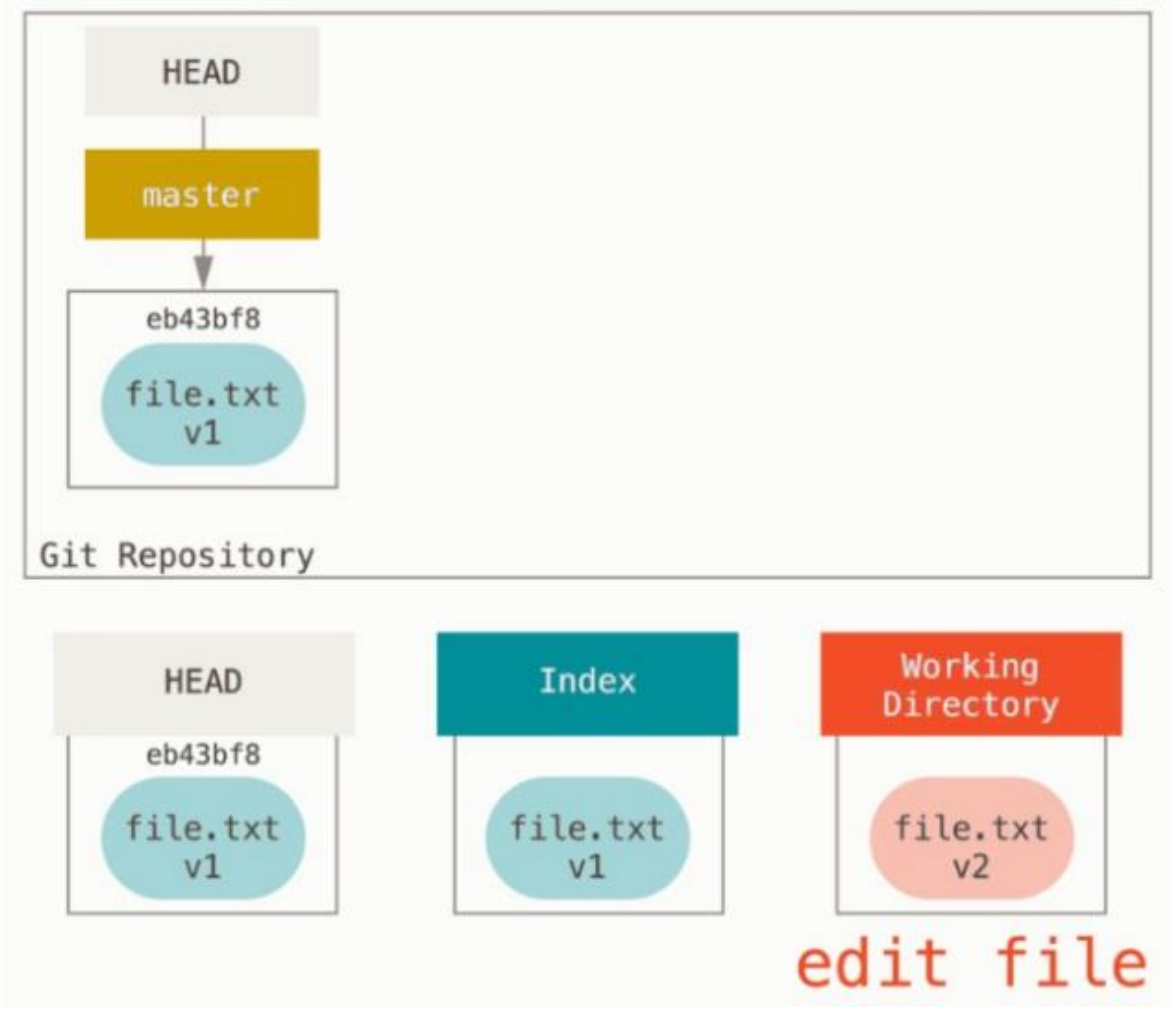
(2) git add 를 통해 파일을 추가한다.



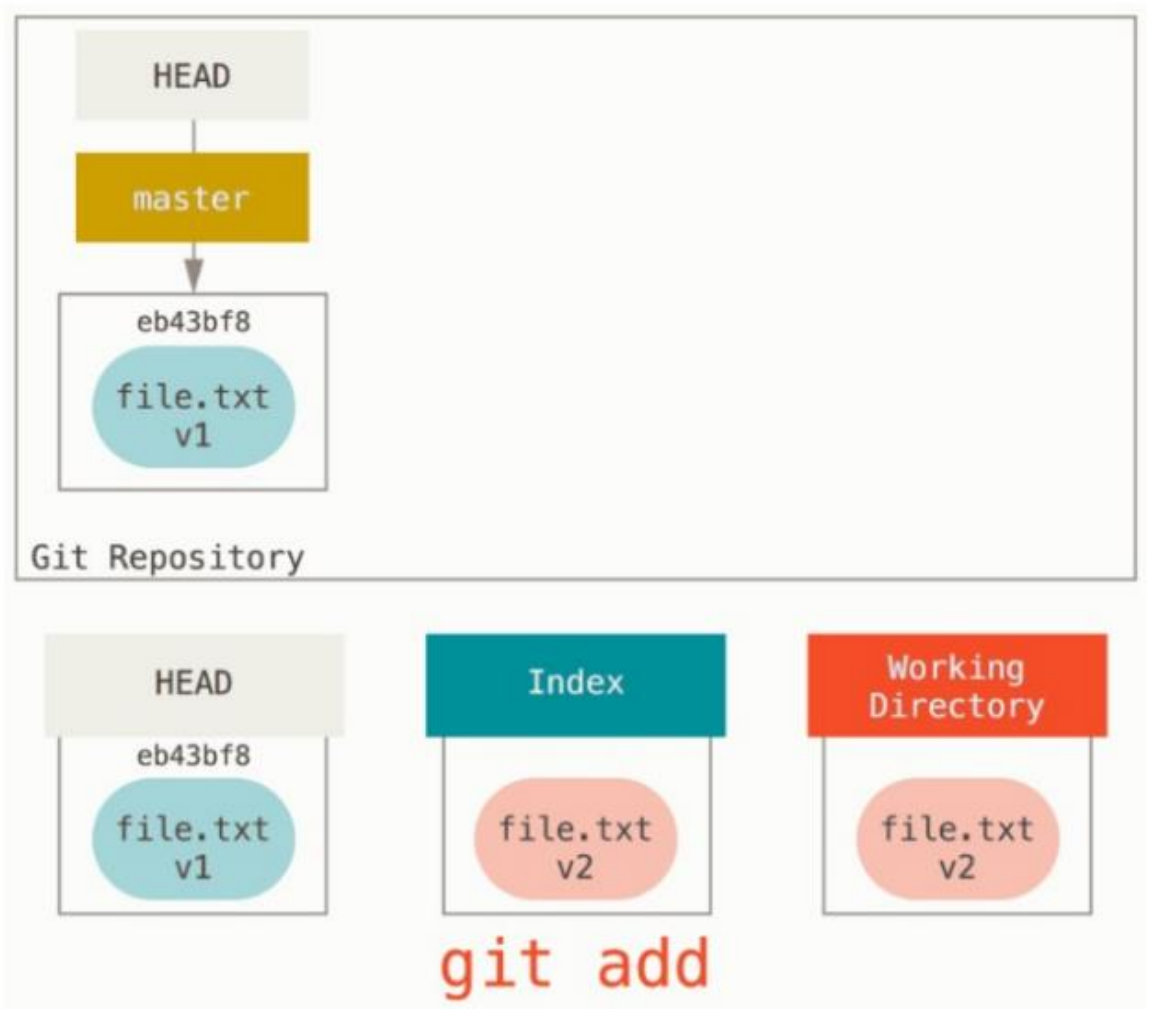
(3) git commit 을 통해 add 로 추가한 파일을 커밋한다. 이때 HEAD 는 commit 한 것을 가리킨다.



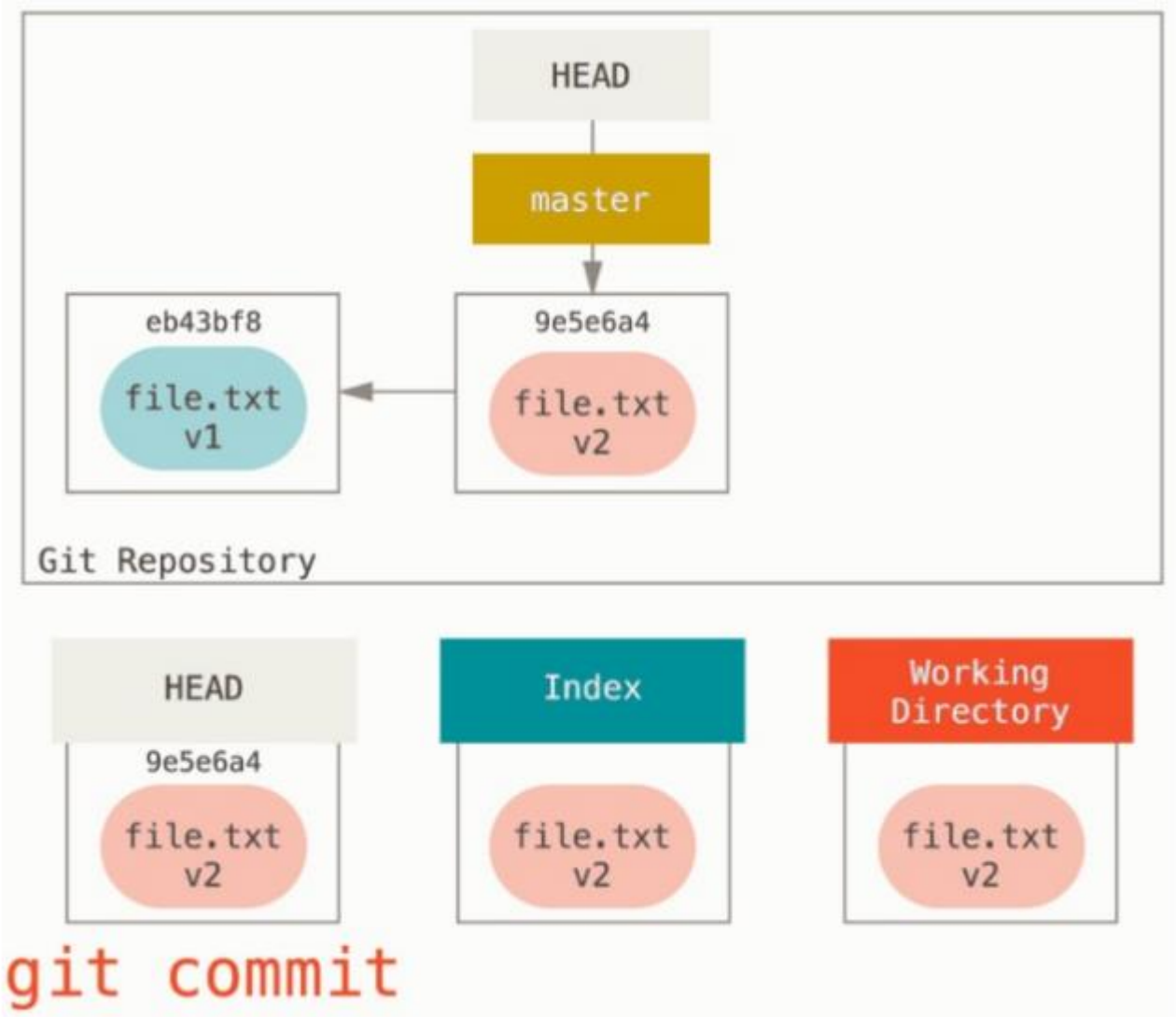
(4) 파일을 수정한다. 그러면 working directory 상에서만 파일이 수정된다.



(5) git add 명령어를 사용하면 Index 의 내용도 바뀐다.



(6) git commit 을 하면 add 가 되었던 파일이 커밋되고 HEAD 는 이를 가리킨다.



1. git reset

옵션에 따라 HEAD, index, Working Directory 중 일부 혹은 모두를 바꿀 수 있다. git reset 은 commit 을 잘못 사용했을 때 사용하는 명령어이다.

1) git reset 의 옵션

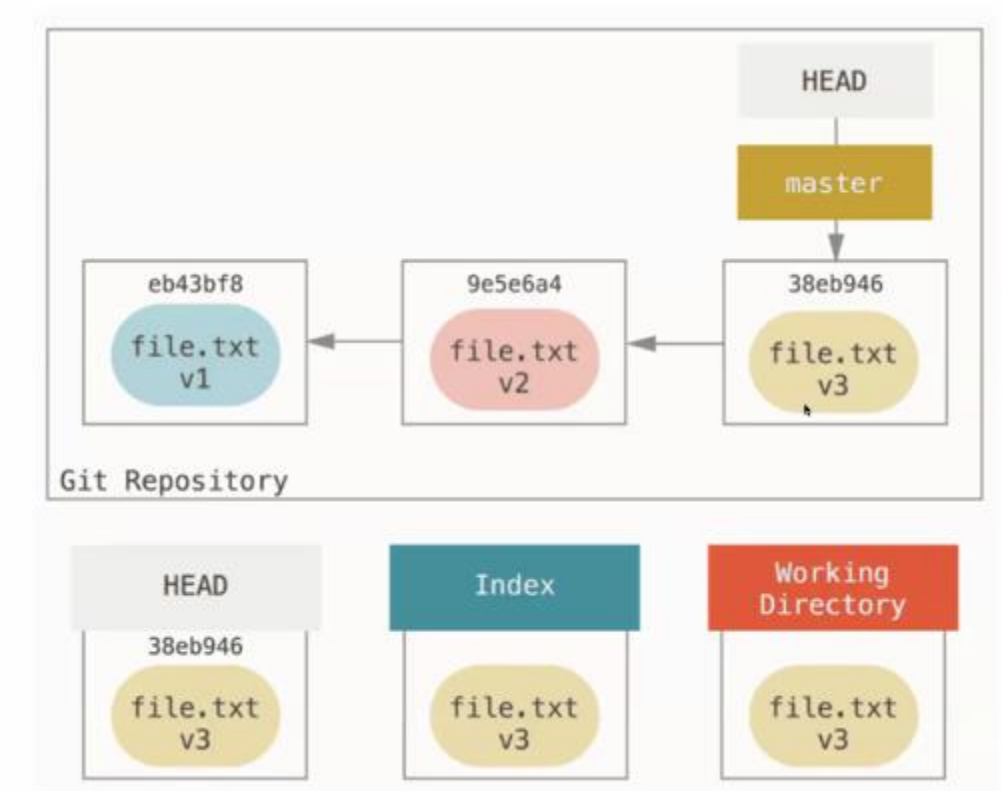
(1) soft: 현재 브랜치와 HEAD 를 이동시킨다.

(2) mixed: 현재 브랜치와 HEAD 를 이동시킨다음 index 도 변경한다.
(3) hard: 현재 브랜치와 HEAD 를 이동시킨다음 index 와 working directory 도 변경한다

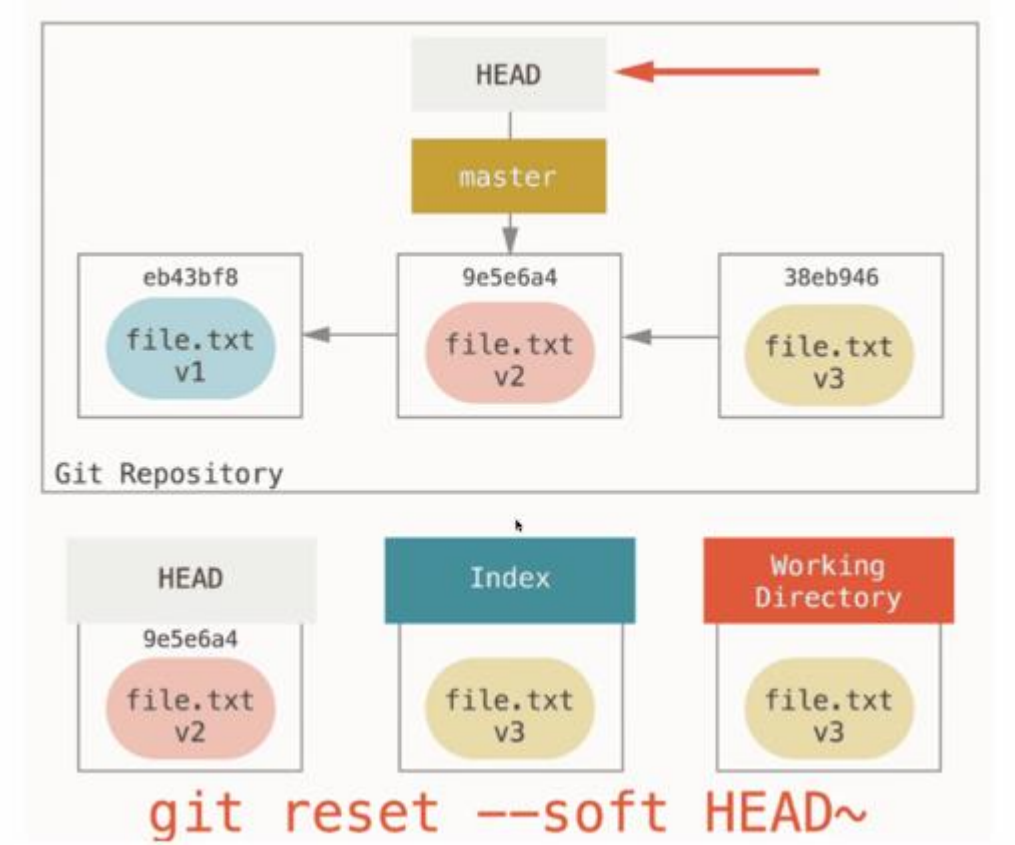
ex) git reset <other branch>

2) 예제

현재 모두 file.txt 를 가리키고 있는 상황이다.

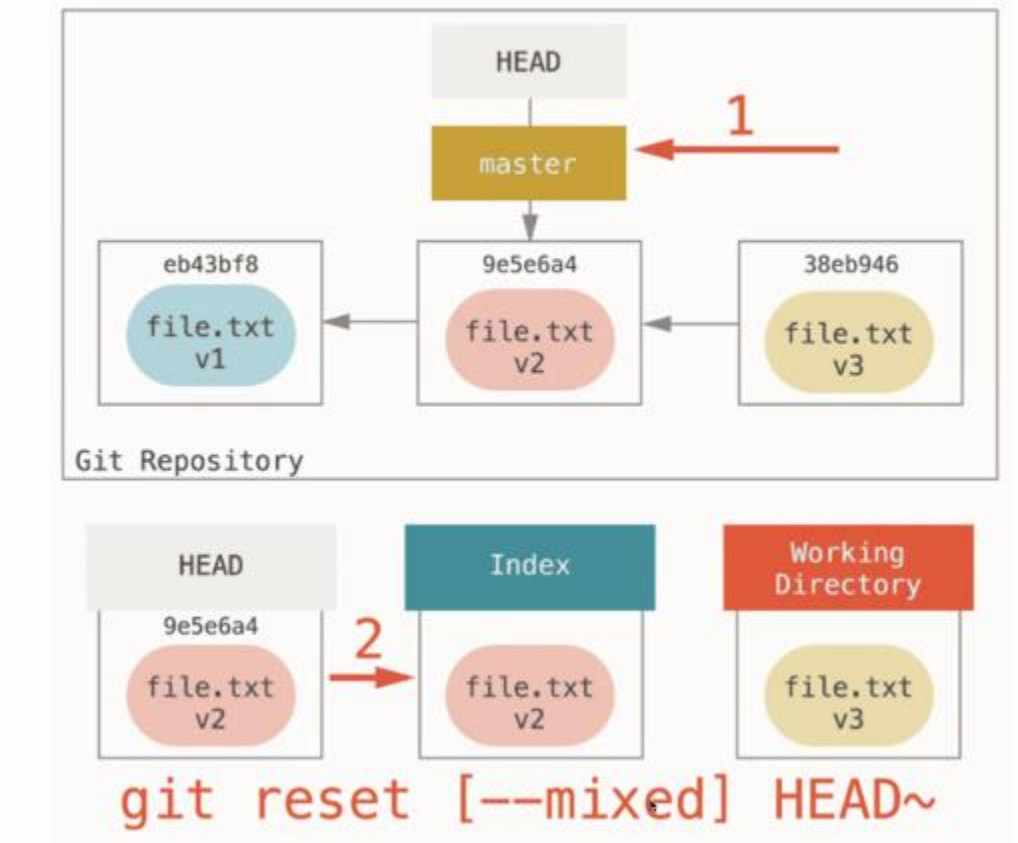


(1) soft



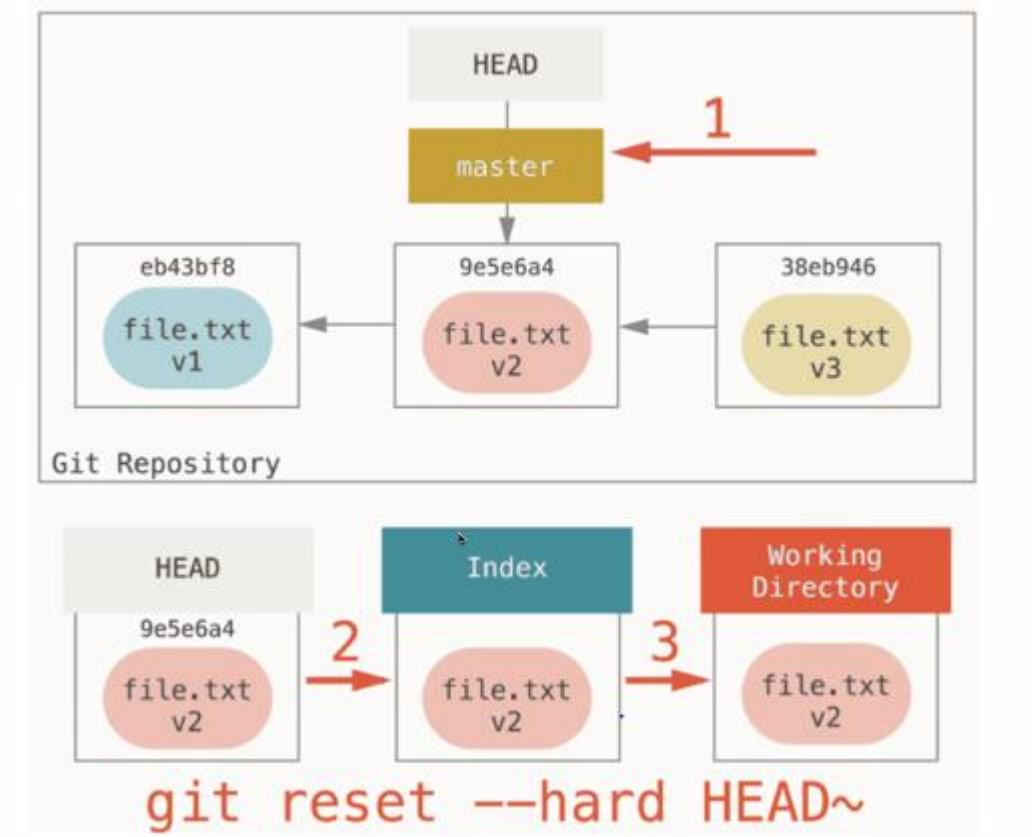
soft 는 commit 만 하지 않은 것처럼 바꿔주는 역할을 한다. 이에 따라 HEAD 가 이전으로 이동하고, 파일 add 까지는 반영이 되어있는 것처럼 된다.

(2) mixed



mixed 는 commit 과 add 를 하지 않은 것처럼 바꿔주는 역할을 한다. 이에따라 HEAD 가 이전으로 이동하고, 파일 add 도 반영되지는 않아 staging area 에는 올라오지 않지만 working directory 에는 남아있도록 한다.

(3) hard



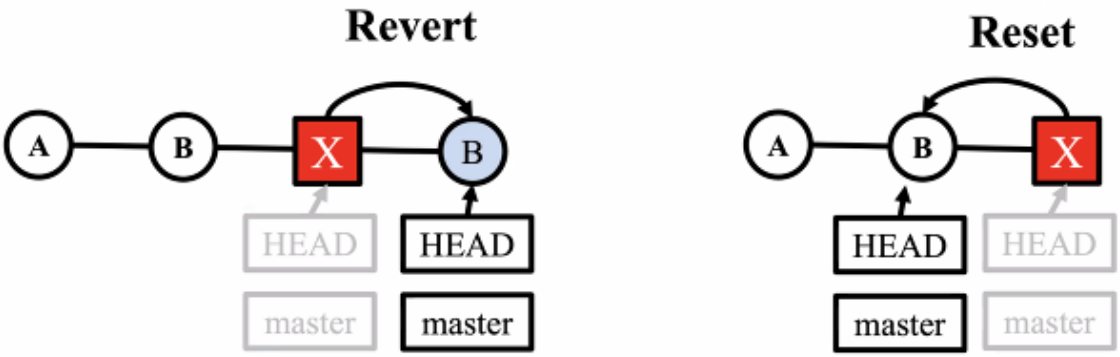
mixed 는 commit 과 add, 작업을 하지 않은 것처럼 바꿔주는 역할을 한다. 이에 따라 HEAD 가 이전으로 이동하고, 파일 add 도 반영되지는 않아 staging area 에는 올라오지 않고 working directory 에는 해당 파일이 없도록 한다. 즉, 작업한 내용이 모두 사라진 것처럼 된다. 하지만 file.txt v3 commit 이 당장 지워지는 것은 아니고 dangling data 가 된다.

이때 명령어 뒤에 번호를 매겨주면 몇 번의 위의 과정을 반복하여 특정한 commit 으로 이동할 수 있다.

2. Revert

새로운 commit 을 만들어서 이동한다.

- **Reverting to a previous revision**
  - **Restores the state of all files at a certain revision to the current working directory**
  - **git revert <commit hash>**
  - **Revert preserves commit history**



git reset 을 함부로 사용하면 다른 사람들과의 협업에 문제가 발생할 여지가 있다. 이에 따라 Revert 를 사용하는 것이 git 의 철학에 맞다고 할 수 있다. Reset 의 경우에는 이전 commit 으로 돌아가지만 Revert 를 할 경우에는 새로운 commit 을 만들어서 이동한다. 이에 따라 Revert 는 잘못 만든 commit 도 이력에 남겨두지만 다른 사람과의 협업 시에 꼬일 일이 없다.

3. 이 외의 부가적인 기능

1) Tag

특정 commit 에 태그를 달 수 있다. 보통 버전을 정할 때 태그를 사용한다. 태그를 사용할 경우에는 checkout 을 할 때 해당 태그가 가리키고 있는 commit 의 파일을 사용할 수 있다.

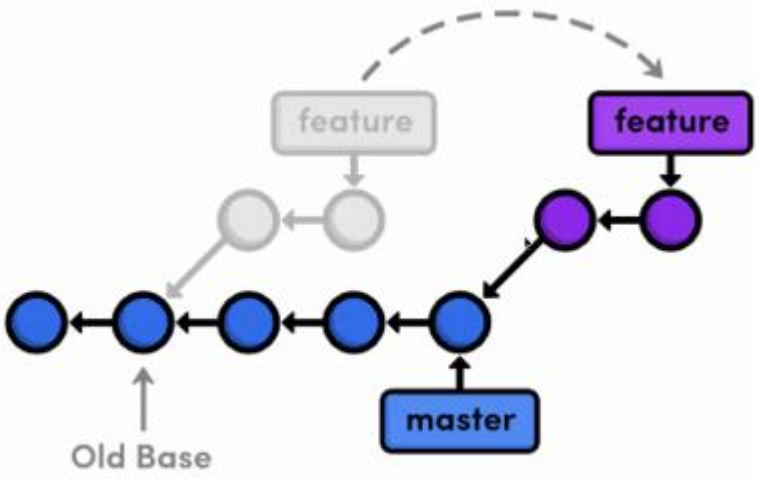
2) Stash

working directory 에서 modified 상태인 파일을 임시로 저장하는 역할을 한다. 이때 다른 branch 로 이동해 checkout 할 수 있기에 종종 사용한다. 만약 stash 를 여러번 사용할 경우에는 stack 형태로 저장된다. 따라서 가장 최근의 stack 을 꺼내오거나 이름을 따로 지정해 이름을 통해서 불러올 수 있다.

3) Rebase

branch 로 나눠져 있는 commit 의 위치를 한 줄로 옮기거나 재배치할 때 사용할 수 있다. branch 가 너무 많아지면 관리하기 어려워지기에 이를 관리하기 위해 재배치할 때 용이하다. 다만 잘못 사용하면 위험할 수 있다.

(1) batch



- **Takes a branch, and modify it to make it look like the branch never existed**
- **Example:**
  - **A-B-C-D master**
    \-E-F-G topic
- **Becomes:**
  - **A-B-C-D-E'-F'-G' topic**
- **From 'topic' branch: 'git rebase master'**

git rebase master 는 master 뒤에 topic 을 붙이라는 명령어이다. 만약 충돌이 발생하는 경우(똑같은 파일이 존재 등)에 이를 수정하면 된다.

(2) Interactive

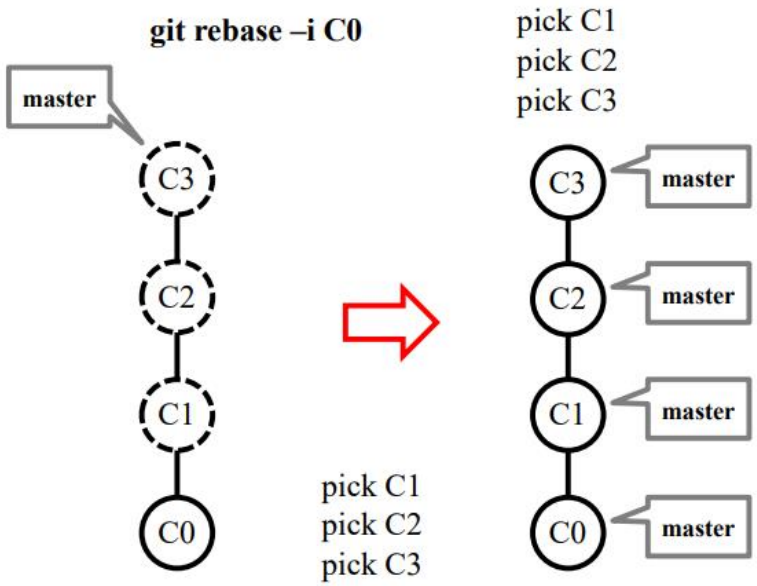
- **git rebase –i <commit>**
  - **<commit> should be the commit BEFORE the first one you wish to alter**

이전에 작업한 commit 정보를 옵션을 전달해서 변경할 때 사용한다. 위의 예시에서는 C0 로 commit 을 설정했기에 C1, C2, C3(현재 commit)에 대한 정보를 바꿀 수 있다.

i. 옵션

ii-1. pick(디폴트): 변경없이 그대로 사용한다.

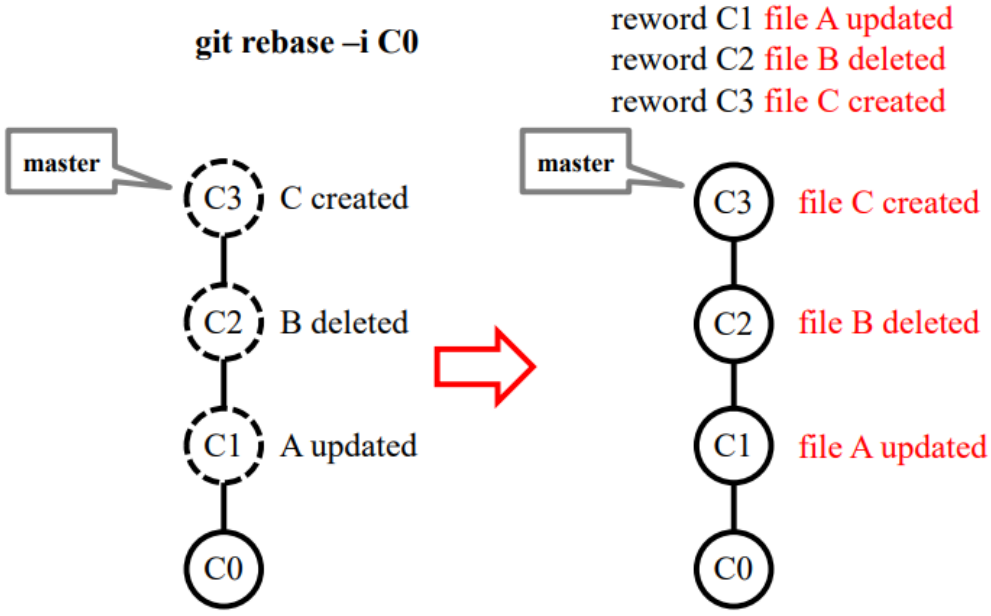
- **pick <commit>**
  - **use commit**



위와 같이 pick 만 사용할 때는 이전에 사용한 것과 완전 동일하다.

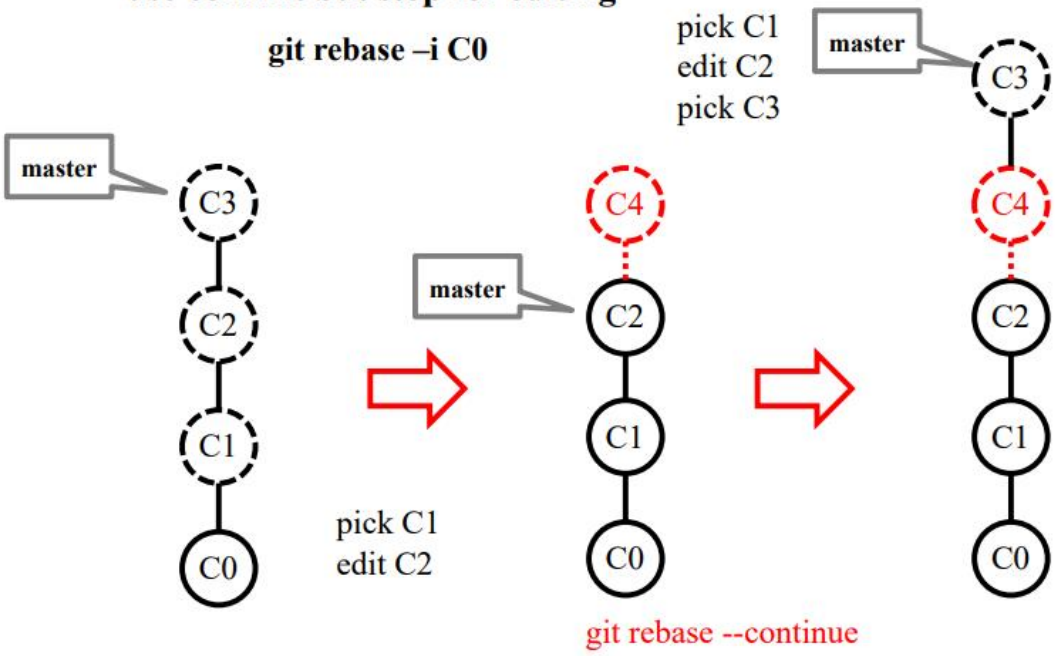
ii-2. reword: commit 은 그대로 사용하지만 commit message 를 바꾼다.

- **reword <commit>**
  - **edit only commit message**



ii-3. edit: commit 을 잠시 멈춰서 새로운 동작을 진행할 수 있다.

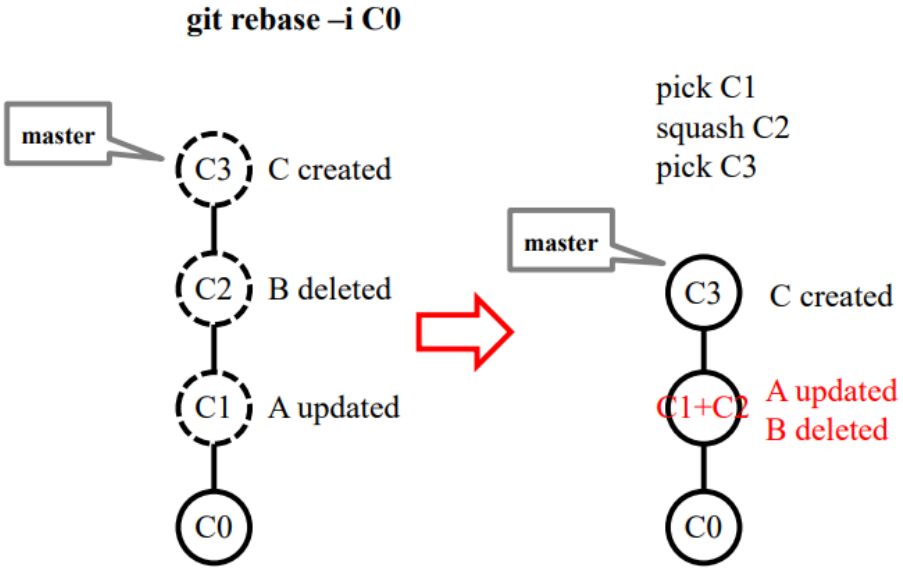
- **edit <commit>**
  - **use commit but stop for editing**



위의 예제에선 새로운 작업을 진행하고 commit 까지하면 C4 가 C2 뒤에 붙는다.

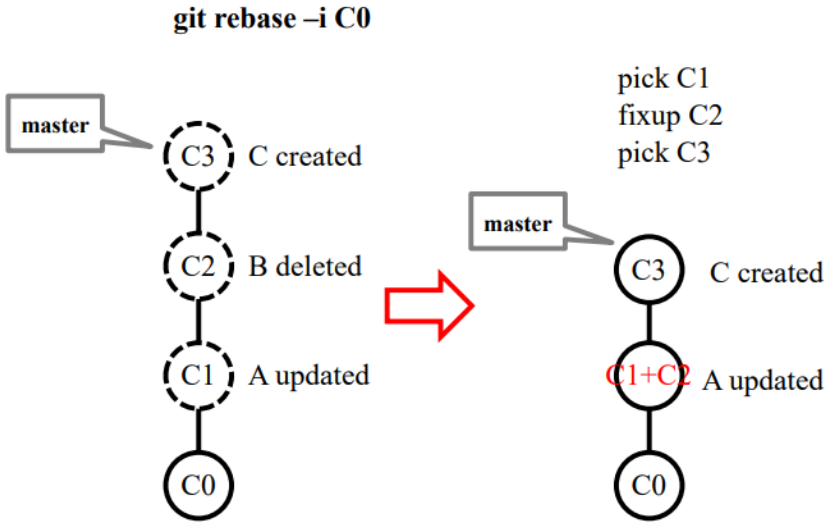
ii-4. squash: commit 을 합친다.

- **squash <commit>**
  - **use commit, but meld into previous commit**



ii-5. fixup: squash 와 비슷하지만 로그 메세지까지 없앤다.

- **fixup <commit>**
  - **like "squash", but discard this commit's log message**



ii-6. 그 외의 명령어

- **exec <command>**
  - **run commit using shell**
- **break**
  - **stop here**
- **drop <commit>**
  - **remove commit**

(3) Rebase 의 충돌

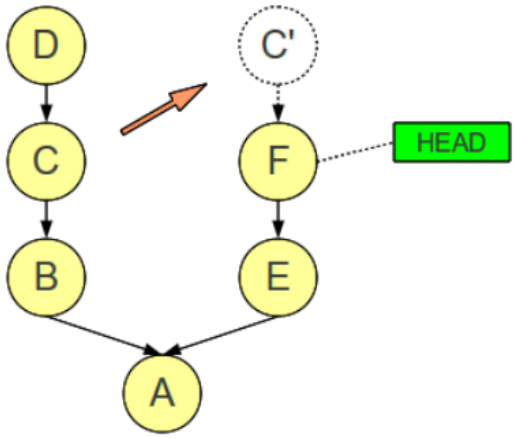
Rebase 는 내부적으로 merge 와 동일하기에 파일을 합칠 때 충돌이 발생할 수 있다. 직접 내용을 수정한 뒤에 continue 하거나 해당 commit 을 무시하기 위해 skip 을 하거나 rebase 를 취소하는 abort 가 있다.

- **Git rebase does an actual merge**
  - **Merges may have conflict**
  - **You have three choices**

- **Solve the conflicts:**
  - **git add the resolutions**
  - **git rebase –continue**
- **Skip this commit**
  - **git rebase –skip**
- **Abort the rebase**
  - **git rebase --abort**

4) cherry pick

- **Get a specific commit from a branch**
- **Merge *only that commit* on a different branch.**
  - **This is a merge operation**
  - **Conflicts may occur and will have to be resolved normally**



특정한 commit 하나만 가져와서 작업중인(HEAD 가 가리키는) commit 뒤에 이를 붙여주는 역할을 한다.

- **git cherry-pick <commit>**
  - **Will merge commit <commit> on the current branch.**

- **If you are merging from a public branch, add "-x", i.e.:**
  - **git cherry-pick –x <commit>**
  - **Reason: Will add a note to the commit message specifying the source of the cherry-pick**

4. git branch 관련 조작 시 주의사항

1) Rebase 나 Reset 은 다른 사람과의 협업을 진행할 때 **함부로 사용한다면 다른 사람이 push 할 때 오류가 발생할 수 있어서 이를 조심해야한다.**

2) remote 레포지토리에 rebase 나 reset 을 사용해 commit 내용을 바꾸었다면 local 레포지토리에 있는 내용을 remote 레포지토리에 push 할 때 에러가 발생할 수 있다. 해당 경우에는 **force 명령어를 사용해** 강제로 이를 저장해야한다.