발제자 : 유영재

추가 커밋 생성 + commit -am

소스코드를 추가하고, 커밋을 새로 하나 생성하도록 한다
추적된 모든 파일에 대해서는 add와 commit 명령을 am 옵션을 합할 수 있다.
반대로 untracked 파일은 항상 add를 먼저 한 후 commit 명령을 해야 한다

$ git commit -am “add contents template C file”

주의할 점은 원격에 push한 이후에는 --amend를 자제해야 한다. 커밋이 꼬이기 때문에 원격에 push --force를 해야만 한다

Git HEAD 이해하기

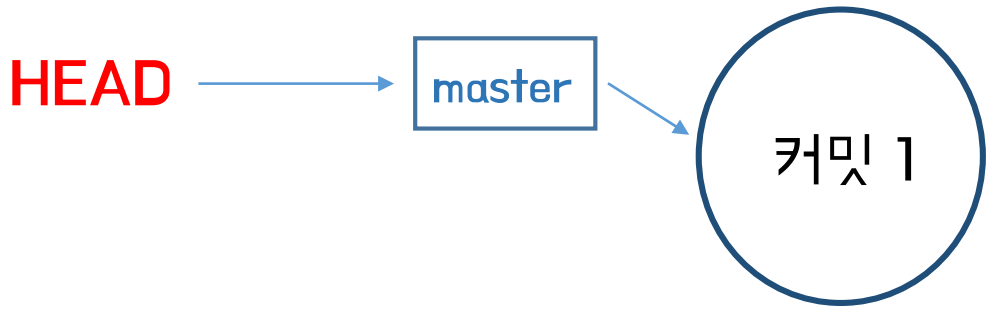
```
yeong@DESKTOP-ES5CT3G MINGW64 ~/Desktop/git-work (master)
$ git log
commit 170581a4b5f87535225137ce60144dcb7af360a5 (HEAD -> master)
Author: ingbeeedd <yeongjae8066@hanmail.net>
Date:   Wed May 20 21:56:48 2020 +0900

    add contents - template C file

commit 1c5b80677e3a05afaca3f0d98050f3e6287f3019
Author: ingbeeedd <yeongjae8066@hanmail.net>
Date:   Wed May 20 21:38:54 2020 +0900

    first commit - add main.c file
```

실제 checkout은 브랜치를 이동하는 것으로 알고 있지만 실제로는 커밋을 가리키는 용도다. 브랜치는 결국 커밋을 가리키기 때문에, checkout을 통해 직접적으로 커밋을 가리킬 수 있다. 그 가리키는 친구가 HEAD 포인터다



stage -> unstage

스테이지에 올라간 파일을 커밋에서 제외하고 싶어 unstage 하는 경우

$ git restore --staged <파일>

모든 파일을 unstage하기 위해서는 .을 붙이면 된다

수정한 파일 최근 커밋으로 되돌리기

파일을 수정하고 나서 undo(ctrl + z)를 하고 싶은데, 일일이 지우기 힘들 때 사용한다

$ git checkout -- <파일>

커밋 덮어쓰기 commit --amend

개인 혹은 협업에 있어 커밋 단위는 의미를 가져야만 나중에 원하는 커밋을 손쉽게 찾을 수 있다. 현재 커밋으로부터 발생하는 단순한 라인 추가나 오타 수정에 관련한 추가 커밋 발생은 바람직하지 않다. 따라서 수정사항을 이전 커밋에 포함시킬 수 있다

```
$ git add <수정 파일>
$ git commit --amend
```

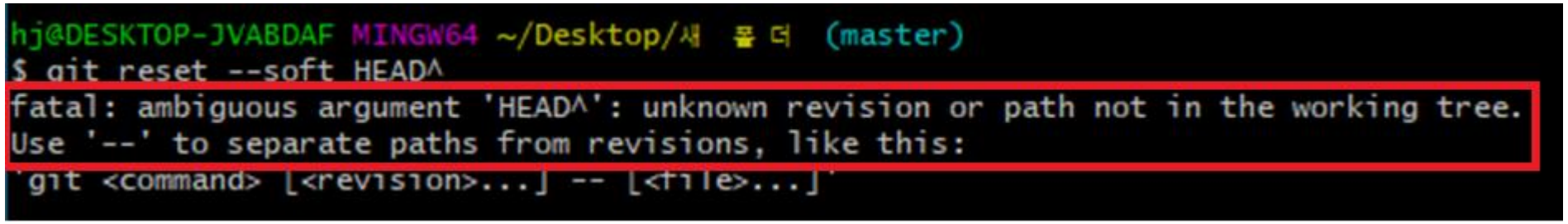
또한 직전 커밋 메시지가 마음에 들지 않아도 사용해도 된다

커밋 취소하기 (1) git reset

커밋을 파괴시킨다. 개인이 로컬에서 작업한다면 사용해도 좋지만, 협업에 있어서는 사용을 지양해야 한다. 근본을 파괴할 수 있기 때문이다

```
$ git reset --hard
$ git reset --mixed (default)
$ git reset --soft
```

첫 커밋을 지울 때 에러가 발생한다. 어떻게? --soft, --hard, --mixed



첫 커밋을 지울 때 에러가 발생한다. 어떻게? 다음을 가리킬 커밋이 없기 때문에 에러가 발생한다. .git 파일을 지우는 방법밖에 없다

커밋 취소하기 (2) git revert

기존 커밋을 유지하면서 내용을 수정할 수 있는 명령어다. 바로 이전 커밋은 --amend 옵션으로는 가능하지만, 뒤에 있는 커밋을 지우기 위해서는 revert로 log를 남기면서 지워야 한다. log를 남겨야만 다른 협업자도 그 커밋을 받을 수 있다

지우고 싶은 커밋을 $ git revert <해쉬>

충돌(confilct)이 발생하는 경우를 방지하려면 한 번에 하나씩 이전 커밋을 revert 해야 한다는 주의사항이 있다

Git 로그 옵션

$ git log --stat --all --oneline --graph -p

```
yeong@DESKTOP-ES5CT3G MINGW64 ~/Desktop/gitwork (master)
$ git log --all --graph --oneline --stat
* d0534e0 (HEAD -> master) add files
|  util.c | 4 ++++
|  1 file changed, 4 insertions(+)
* 3129c95 add util.c
|  util.c | 0
|  1 file changed, 0 insertions(+), 0 deletions(-)
* 8ba505e edit main.c, add main.h
|  main.c | 1 +
|  main.h | 0
|  2 files changed, 1 insertion(+)
* d24ea9c add main.c template
|  main.c | 6 ++++++
|  1 file changed, 6 insertions(+)
* 814ef00 add main.c
   main.c | 0
   1 file changed, 0 insertions(+), 0 deletions(-)
```

브랜치

브랜치는 특정 커밋을 기준으로 줄기를 나누어 작업할 수 있는 기능을 일컫는다. 아래 명령을 통해 현재 커밋에서 브랜치를 생성해보도록 하자

1. 브랜치 생성
$ git branch <branch name>
2. 브랜치 생성 후 바로 checkout
$ git checkout -b <branch name>
3. 모든 브랜치 정보 확인
$ git branch -a

브랜치 왜 만드나요?

1. 새로운 기능 추가 (가장 흔한 이유)
2. 버그 수정
3. 기존 코드 개선 (이전 커밋으로 돌아가서 브랜치 생성하는 경우)
4. 병합과 리베이스 테스트

현재 브랜치에서 커밋 생성

```
$ vim main.c
$ git commit -am “edit main.c | branch feature”
$ git log --all --graph
```

다른 커밋에서 브랜치 생성

HEAD 포인터를 움직여 브랜치를 생성할 수 있다

```
$ git checkout <커밋 Hash>
$ git switch -c <브랜치 이름>
$ vim main.c
$ git commit -am “edit main.c | branch feature”
$ git log --all --graph
```

체크아웃 시 주의사항 stash

체크아웃 할 때는 반드시 stage의 내용이 커밋이 되어야만 한다. 그래야만 체크아웃이 가능하다. 그리고 unstage의 내용은 checkout 때 사라진다. 따라서 Git에서는 stash라는 명령을 통해 staged 내용을 임시로 보관하게 해준다

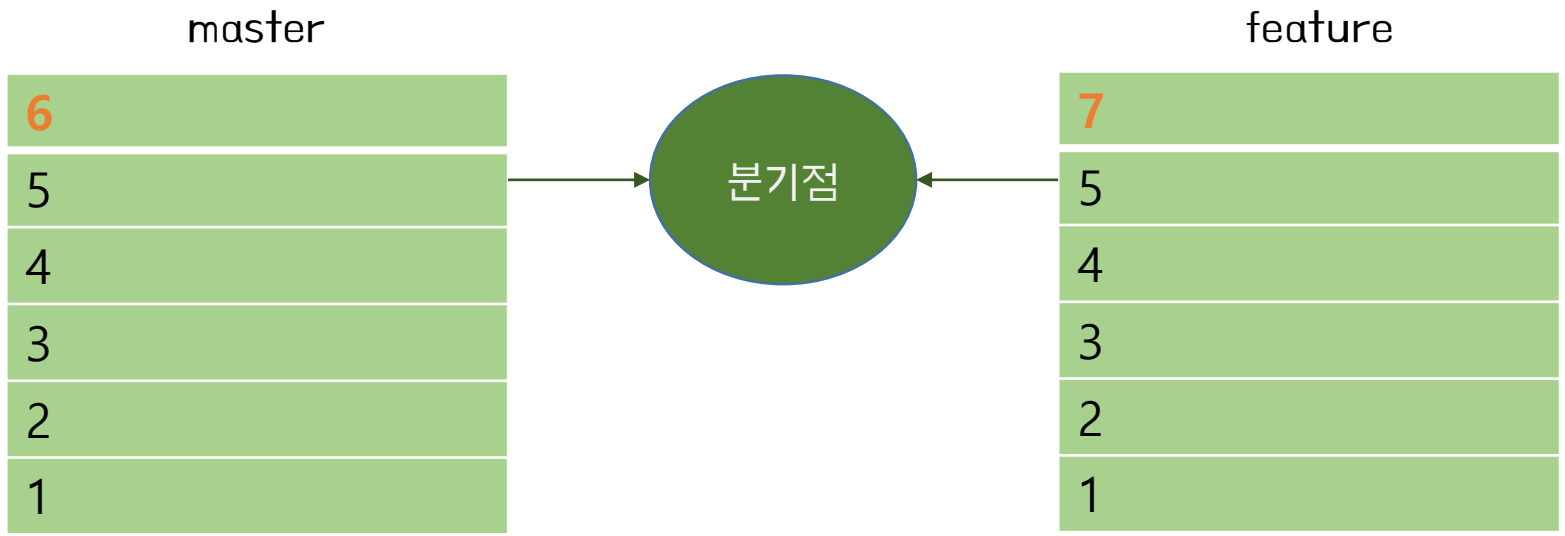
임시저장소 만들기
$ git stash save
임시저장소 리스트 확인
$ git stash list
임시저장소 내용 꺼내기
$ git stash apply
임시저장소 내용 지우기
$ git stash drop

병합 (merge)

두 브랜치의 커밋을 합하는 행위. 브랜치를 만드는 대부분의 이유가 새로운 기능 추가에 있다. 그래서 메인 브랜치와 서브(기능 개발) 브랜치를 합할 때 merge를 사용한다

병합 (merge)

두 브랜치의 커밋을 합하는 행위. 브랜치를 만드는 대부분의 이유가 새로운 기능 추가에 있다. 그래서 메인 브랜치와 서브(기능 개발) 브랜치를 합할 때 merge를 사용한다. 그리고 병합을 설명할 때 충돌(conflict)이라는 단어는 항상 언급된다. 충돌이 생기는 원인은 두 브랜치 사이에 커밋 이력이 같지 않기 때문이다



3 way merge

Git에서 merge를 할 때 내부적으로 동작되는 merge 기법으로, 대부분의 version control system에서는 3 way merge 기법을 사용한다

base (조상)	master	feature	2-way-merge	3-way-merge
A	A	A	A	A
B	B	C	충돌	C
C	K	C	충돌	K
D	A	B	충돌	충돌
E	E	H	충돌	H
F	A	Q	충돌	충돌

Fast forward merge

Fast forward merge는 base가 두 브랜치 중 하나와 똑같을 경우다. 이때는 굳이 병합 커밋을 생성할 필요가 없기 때문에 병합 커밋을 생성하지 않는다.

base (조상)	master = base	feature	3-way-merge(=feature)
A	A	A	A
B	B	C	C
C	C	C	C
D	D	B	B
E	E	H	H
F	F	Q	Q

질문 하나

master와 feature 브랜치가 병합하려고 하는데 두 파일의 내용이 같다면 머지 커밋은 생기는가?

base (조상)	master	feature	3-way-merge(=feature)
A	Z	Z	Z
B	Y	Y	Y
C	X	X	X
D	W	W	W
E	U	U	U
F	V	V	V

mergetool p4merge 설정

충돌이 났을 경우에는 Git이 직접 해결하지 않고, 사용자에게 충돌사항만 알려준다. 따라서 사용자는 충돌사항을 보고 직접 해결해야만 한다. 하지만 프로젝트가 방대해질 때는 Git에서 기본적으로 제공되는 에디터로는 충돌사항을 쉽게 확인하기가 어렵다.

따라서 서드파티 mergetool 중 하나인 p4merge를 사용하려고 한다. 모든 플랫폼에서 사용할 수 있다는 장점이 있고 기본적으로 3-way-merge 기법으로 merge를 한다

설치 https://www.perforce.com/downloads/visual-merge-tool

유용한 사이트 소개

통합검색

1. 생활코딩 (Git - CLI, Git - SourceTree, SVN, Github, Gitlab 등 제공)
2. 인프런 (코드스쿼드 정호영 - https://www.inflearn.com/course/git-and-github/dashboard)
3. 브랜치 가지고 놀기 (https://learngitbranching.js.org/?locale=ko)

감사합니다

PPT 템플릿 자료는 http://pptbizcam.co.kr/에서 이용했습니다.