12월 17일 (금)

1. Codecademy Learn Git - How to backtrack

- **Backtracking Intro**

프로젝트 중 실수를 지우는 지우개와 같은 역할을 하는 깃의 기능을 이번 레슨에서 배울 것이다.

- **head commit**

HEAD 커밋은 git log command 의 모든 것들을 보여주고, 커밋된 파일의 변경된 내용을 보여준다.

HEAD 는 가장 최근 커밋이다.

```
git show HEAD
```

```
$ git show HEAD
commit 83e09ccc313b0f9ec924b8f9aaa69dbcdcd2c5b7
Author: codecademy <ccuser@codecademy.com>
Date:   Tue Jul 13 06:31:49 2021 +0000

    Ghost change Commit

diff --git a/scene-5.txt b/scene-5.txt
index b12dd97..90b04eb 100644
--- a/scene-5.txt
+++ b/scene-5.txt
@@ -6,7 +6,9 @@ Hamlet:
 Where wilt thou lead me? speak; I'll go no further

 Ghost:
-Mark me.
+My hour is almost come,
+When I to sulphurous and tormenting flames
+Must render up myself.

 Hamlet:
 I will.
$
```

결과값

- **git checkout**

파일을 변경후 다시 이전으로 되돌리기 위한 방법이 있다.

```
git checkout HEAD filename
```

먼저 연습용 텍스트의 내용을 바꾼 후 git diff 로 바뀐 부분을 체크한다.

```
$ git diff
diff --git a/scene-5.txt b/scene-5.txt
index 90b04eb..a691c2c 100644
--- a/scene-5.txt
+++ b/scene-5.txt
@@ -7,8 +7,8 @@ Where wilt thou lead me? speak; I'll go no f
rther

 Ghost:
 My hour is almost come,
-When I to sulphurous and tormenting flames
-Must render up myself.
+When I to sulphurous and tormenting balloons
+Must render up myself.

 Hamlet:
 I will.
```

```
$ git checkout HEAD scene-5.txt
$
```

그 후 위 checkout 커맨드를 실행하면 다시 전 버전으로 돌아가게 된다. 커맨드 입력후 아무것도 출력하지 않지만 해당 파일로 들어가면 내용이 다시 되돌아와있을 것이다.

- **more git add**

```
git add filename_1 filename_2
```

스테이징 영역으로 파일을 추가할 때 이렇게 다수의 파일을 add 할 수 있다.

- **git reset - 1 -**

add 한 파일을 스테이징 영역으로 부터 다시 빼낼 수 있는 기능이다.

```
git reset HEAD filename
```

```
$ git reset HEAD scene-2.txt
Unstaged changes after reset:
M       scene-2.txt
M       scene-5.txt
$ git commit -m "Commit"
```

M 은 (modification)으로 이 출력값은 리셋 후 스테이징되지 않은 변화된 파일들을 보여준다.

- **git reset - 2 -**

프로젝트를 하는 것은 등산하는 것과 같다. 프로젝트 또한 길을 잘못들어서거나 길을 잃는 경우도 있다.

등산처럼 왔던 길을 되짚어보는 것처럼, 깃에서도 가능하다.

```
git reset commit_SHA
```

`5d692065cf51a2f50ea8e7b19b5a7ae512f63`

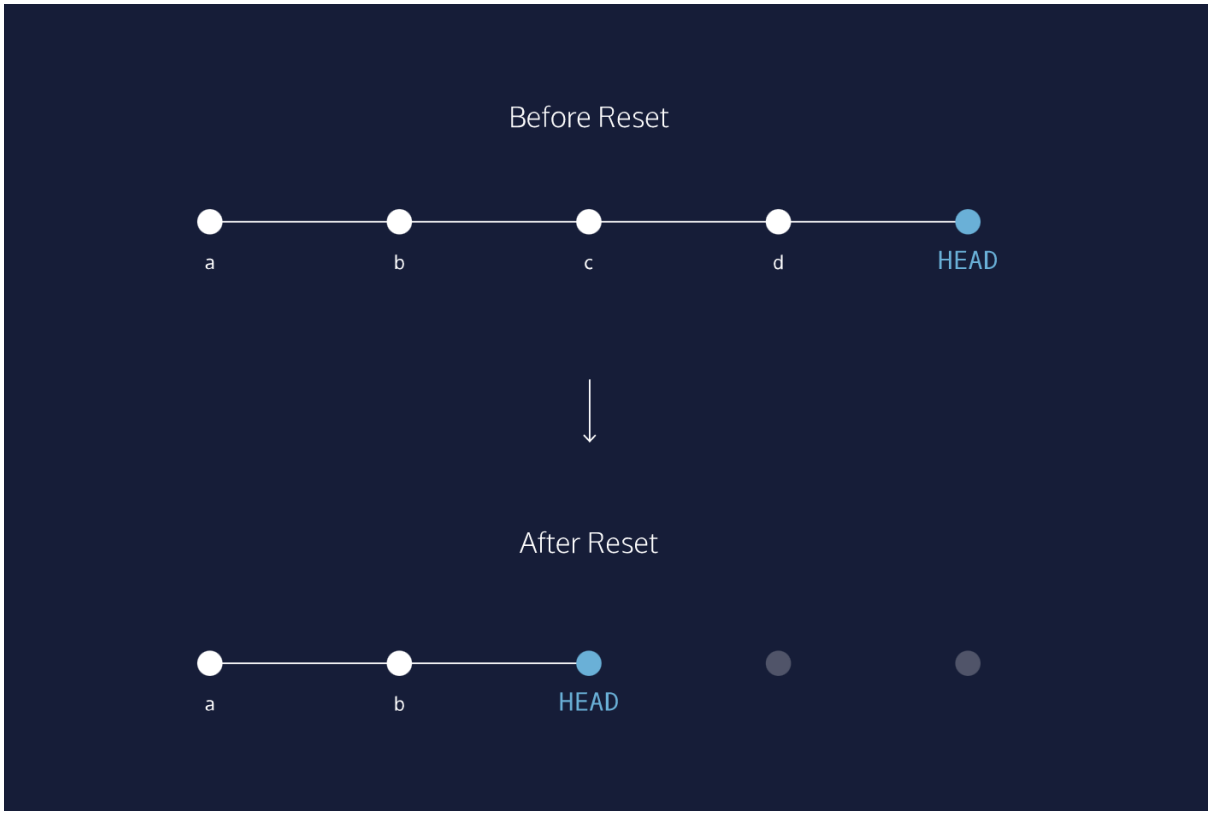
`3ba` , use:

```
git reset 5d69206
```

SHA 코드의 7글자를 따온다.

커맨드 후 HEAD 는 설정한 SHA 코드의 이전 커밋으로 다시 세팅된다.

- **git reset review**



- **Review**

이번 레슨에서 깃에서 백트랙하는 세 가지 방법과 여러 파일을 add 하는 방법을 배웠다.

- `git checkout HEAD filename`: Discards changes in the working directory.
- `git reset HEAD filename`: Unstages file changes in the staging area.
- `git reset commit_SHA`: Resets to a previous commit in your commit history.

Additionally, you learned a way to add multiple files to the staging area with a single command:

```
git add filename_1 filename_2
```

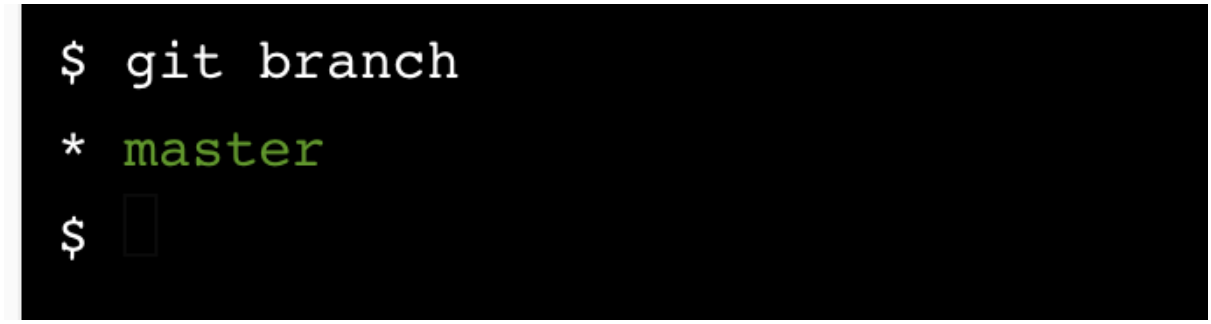
2. Codecademy Learn Git - Git Branching

- **git branch**

지금까지 master 라고 불리는 싱글 git branch 에서만 작업해왔다. git 은 다른 버전을 경험할 수 있는 branch 기능(일명 가지)을 가지고 있다. 이 branch 와 master 을 merge 하지 않는 이상 master에는 아무런 영향을 끼치지 않는다.

이번 레슨에서는 branching 에 대해서 배울 것이다.

내가 현재 어디 브랜치에 있는지 알려주는 커맨드이다.



' * ' 는 너가 현재 있는 위치를 말해준다.

- **branching overview**

- **git branch 2**

```
git branch new_branch
```

새로운 브랜치를 만들기 위한 커맨드이다.

브랜치 이름에 공백은 들어갈 수 없다.

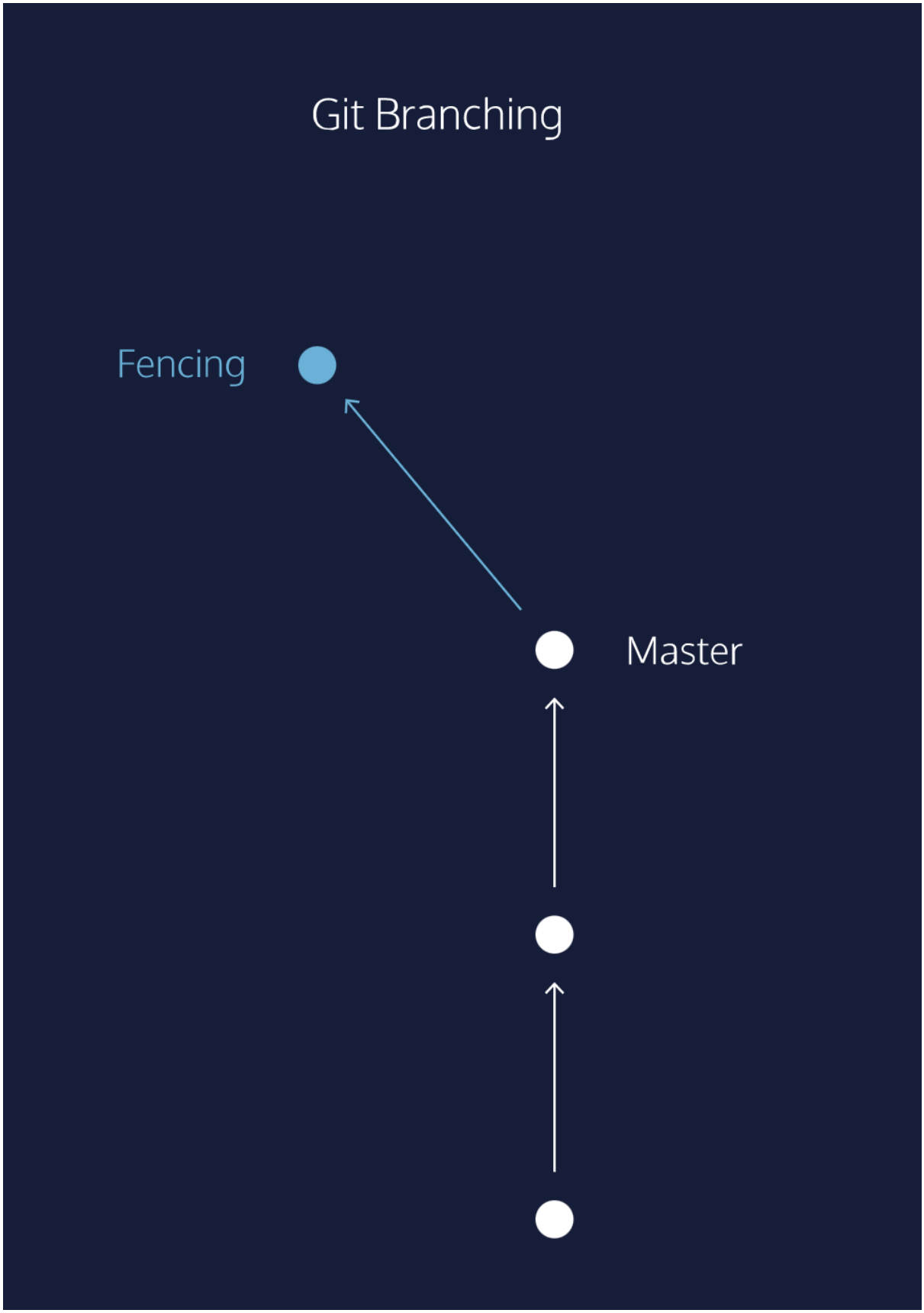
- **git checkout (체크아웃 - 브랜치에서 나온다)**

```
git checkout branch_name
```

다른 브랜치로 이동하는 커맨드이다.

```
$ git checkout fencing
Switched to branch 'fencing'
$ git branch
* fencing
  master
$
```

- **commit on a new branch**



새로운 브런치에서 커밋해보기

- **git merge**

만약 fencing 브런치에서 추가한 내용을 master에 그대로 옮기려면 어떻게 해야할까?
merge 를 사용하면 된다.

```
git merge branch_name
```

master 에 머지하고 싶은 브랜치이름을 넣는다.

fencing 은 변화를 제공하는 giver branch 이고 master 은 receiver branch 이다.

master 로 checkout 후에 merge 를 진행한다.

```
$ git branch
* fencing
  master
$ git checkout master
Switched to branch 'master'
$ git merge fencing
Updating 79a1cc5..0a8d932
Fast-forward
 resume.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
$
```

Fast-forward 는 fencing 브랜치가 가장 최신의 커밋을 포함하고 있다는 것을 깃이 인지하는 것이다.

- **merge conflict - 1 -**

만약 fencing 브랜치를 merge 하기 전에 master 에서 커밋하게 된다면 어떻게 될까? 또 fencing 에서 작업한 내용을 master에서 똑같은 부분을 변경하여 커밋한 상태에서 merge 한다면 어떻게 될까? master 로 돌아온 뒤 깃에게 두 브랜치를 merge 하라고 하면 git 은 어느것을 유지해야할 지 모른다. 이러한 상황을 merge conflict 라고 부른다.

```
$ git branch
  fencing
* master
$ git add resume.txt
$ git commit -m "Third Commit"
[master 4986081] Third Commit
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git checkout fencing
Switched to branch 'fencing'
$ git add resume.txt
$ git commit -m "Forth Commit"
[fencing ed712ad] Forth Commit
 1 file changed, 1 insertion(+), 1 deletion(-)
$
```

resume 파일을 master 브런치에서 내용을 수정후 커밋했다. 몇 주후 fencing 브런치로 체크아웃후 resume 파일을 또 수정한 뒤 커밋한 상황이다.

- **merge conflict - 2 -**

너가 만약 fencing 브런치를 giver brunch로 하고 싶다고 결정한다면, 문제가 발생할 것이다. 위에서 다른 브런치에서 같은 라인이 충돌하는 방식으로 커밋을 했다. fencing 을 giver 로 지정한다면, 깃은 어느 파일을 유지할 것인지 선택하지 못한다.

```
$ git checkout master
Switched to branch 'master'
$ git merge fencing
Auto-merging resume.txt
CONFLICT (content): Merge conflict in resume.txt
Automatic merge failed; fix conflicts and then commit the result.
$
```

fencing 을 머지하려는데 오류가 뜬다.

```
12  <<<<<<< HEAD
13  - Engage in swordfights with
    professional pirates
14  =======
15  - Engage in swordfights with
    professional pirates such as Smee.
16  >>>>>>> fencing
```

resume.txt 파일 내 내용이 이렇게 바뀌었다. 즉, HEAD는 master 의 내용이다. 둘 중 하나를 선택해서 지워야 오류가 안뜬다. HEAD 와 fencing 은 모두 지워야한다.

```
12  - Engage in swordfights with
    professional pirates such as Smee.
13
```

fencing 의 내용만 남기고 모두 지웠다.

지운 뒤 master 에서 커밋하면 문제해결!

- **delete branch**

master로 부터 가지를 친 다른 여러 브런치들은 master에 merge하고 나면 더이상 브런치의 존재에 대한 의미가 없어지게 된다. 이럴 때 보통 브런치를 삭제한다.

```
git branch -d branch_name
```

브랜치 삭제 커맨드

```
$ git branch
  fencing
* master
$ git branch -d fencing
Deleted branch fencing (was c9c1b68).
$ git branch
* master
```

브랜치 삭제 예시

- **review**

git branching 은 사용자들이 분리된 브랜치를 사용하여 다른 버전의 프로젝트를 경험할 수 있도록 한다.

⇒ git branch : 깃 프로젝트의 브랜치 리스트를 보여준다.

⇒ git branch branch_name : 새로운 브랜치를 생성

⇒ git checkout branch_name : 해당 브랜치로 이동한다.

⇒ git merge branch_name(giver) : 변경된 내용을 master에 합친다.

⇒ git branch -d branch_name : 특정 브랜치를 삭제한다.

3. Codecademy Learn Git - Git Teamwork

- **Overview**

지금까지 싱글유저로서의 깃 작업에 대해 배워왔다. 깃은 다른 사람들과 협업할 수 있도록 여러 툴들을 제공한다.

- 프로젝트의 복제파일을 자신의 컴퓨터에 불러오기
- 서로의 작업을 리뷰하고 계속 파악하기
- 최종적인 프로젝트 버전에 접근하기

위 세가지 모두 remote 를 사용하여 구현이 가능하다.

remote 는 공유된 git 레포지토리이다. 이는 다른 지역에 사는 팀원들이 모두 독립적으로 작업이 가능하게 한다.

- **git clone**

프로젝트의 복제파일을 내 컴퓨터에 불러오는 커맨드이다.

```
git clone remote_location clone_name
```

remote_location 은 remote 파일을 어디서 찾아야할지 깃에게 말해준다. 이것은 web address 또는 filepath 가 될 수 있다. clone_name 은 내 컴퓨터에 저장할 이름을 지정하는 것이다.

```
/Users/teachers/Documents/some-remote
```

```
$ git clone science-quizzes my-quizzes
Cloning into 'my-quizzes'...
done.
$
```

- **git remote -v**

```
git remote -v
```

해당 커맨드를 통해 remote 하는 리스트를 모두 볼 수 있다. 이중 master 와 같은 역할을 하는 것은 보통 origin 이라고 칭한다. 이는 바꿀수도 있다.

```
$ ls
my-quizzes  science-quizzes
$ cd my-quizzes
$ git remote -v
origin  /home/ccuser/workspace/curricu
lum/science-quizzes (fetch)
origin  /home/ccuser/workspace/curricu
lum/science-quizzes (push)
$
```

이처럼 origin 의 주소와 함께 나온다.

- **git fetch**

내가 휴식중일 때 다른 사람이 프로젝트의 내용을 즉 origin 을 업데이트 한다면 내 컴퓨터에 있는 remote 파일은 최신버전이 아닌 것이다. 이럴 경우 fetch 를 이용해서 불러온다. 변경이 있었는지 확인할 수 있고, 또 그 변경 내용을 불러올 수 있다.

```
git fetch
```

이 커맨드는 merge 하는 커맨드가 아니다. 변화들은 remote branch 에 따로 불러온다.

```
$ cd my-quizzes
$ git fetch
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
From /home/ccuser/workspace/curriculum-a/science-quizzes
 * [new branch]      master     -> origin/master
$
```

실행 예시이다. origin/master 이라는 새로운 브랜치가 생성되었다.

- **git merge**

```
git merge origin/master
```

origin 의 새로운 커밋은 origin/master 브랜치에 fetch 되었지만, 아직 master 브랜치는 업데이트 되지 않았다. 어떤 커밋이 되었는지 보고 해당 파일을 불러오기 위해서 merge 를 사용해야한다.

해당 커맨드를 사용하자. origin/master는 브랜치이다.

```
$ git merge origin/master
Updating 2fd7d9b..3a29454
Fast-forward
 biology.txt | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 biology.txt
$ git log
commit 3a294546f4a55f02bf37233ef8988d8b9dd7ce59
Author: danasselin <johndoe@example.com>
Date:   Tue Nov 3 12:33:23 2015 -0500

    Add heading and comment to biology quiz

commit 6aa7704a31d05541141fbb529abf946bd2fd416b
Author: danasselin <johndoe@example.com>
Date:   Thu Oct 29 17:04:04 2015 -0400

    Add biology quiz

commit 2fd7d9b248e0b4a3b531b9af3bb61916d42ad45f
Author: danasselin <johndoe@example.com>
Date:   Thu Oct 29 15:42:55 2015 -0400

    Add first question to physics quiz
```

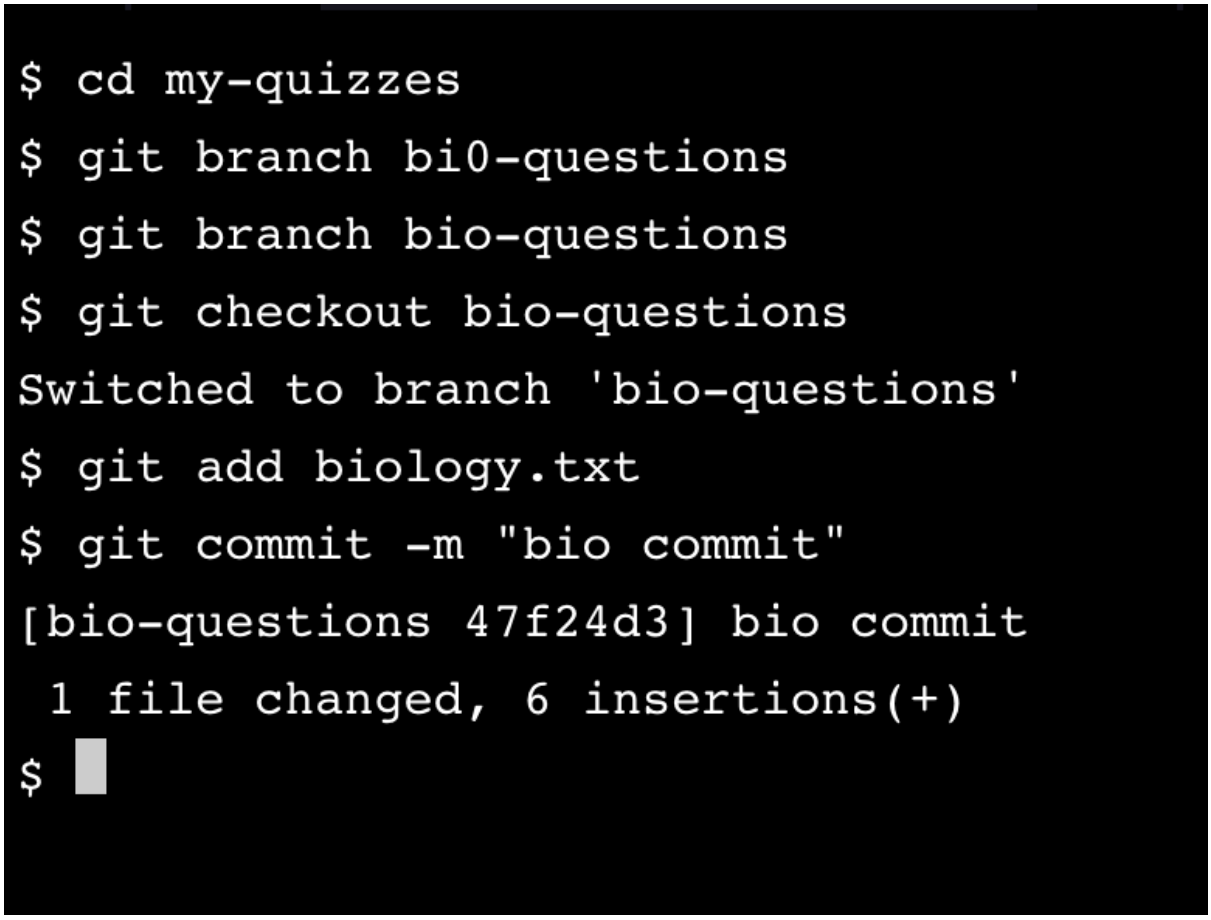
맨 위 커밋 Add heading and ~~ 는 내가 휴식중에 추가된 커밋들이다. 원래 내 컴퓨터의 최신 커밋은 가장 아래 Add first question to ~ 였다.

- **Git workflow**

fetch 를 받았으니 이제 협업에 기여를 할 차례이다. 보통 git 워크플로우는 아래와 같다.

1. fetch and merge - remote 로부터 변경사항을 fetch , merge 한다.
2. 새로운 브런치 생성 - 새로운 프로젝트 버전 위에서 작업하기 위해
3. Develop and Commit - 나의 브런치에서 작업을 마친 후 커밋을 한다.
4. Fetch and Merge again - 내가 작업하는 동안 새로운 커밋이 생겼을 수도 있으니 확인차 fetch 및 merge 를 진행한다.
5. Push - 리뷰를 위해 나의 브런치를 리모트에 push 한다. (git push)

step 1~4 는 merge conflict 를 방지하기 위한 safegaurd 이다.



예시

- **git push**

이제 너의 작업을 공유해보자!

```
git push origin your_branch_name
```

위 커맨드는 너의 브런치를 remote 즉, origin 에 푸쉬한다. 그 후 origin 에서 브런치를 리뷰하고 merge 를 결정한다.

```
$ git push origin bio-questions
Counting objects: 3, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 383 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To /home/ccuser/workspace/curriculum-a/science-quizzes
 * [new branch]      bio-questions -> bio-questions
$
```

bio-questions 브런치는 origin 에 push 되었다고 출력된 내용이다. 이제 origin 을 관리하는 팀원은 나의 코드를 리뷰할 수 있고 remote 인 master 에 merge 할 수 있다.

- **Review**

⇒ remote 는 너의 깃 프로젝트 폴더 밖에서 살아있는 깃 리포지토리이다. remotes 는 공유된 네트워크 위 웹 위 또는 너의 컴퓨터 폴더에 살아 움직인다.

⇒ git clone : 로컬에 remote 의 복제하여 생성한다.

⇒ git remote -v : 깃 프로젝트의 리모트들의 리스트를 보여준다.

⇒ git fetch : 변경된 remote를 로컬의 origin/master 브런치로 불러온다.

⇒ git merge origin/master : master 에 변경된 remote 를 merge 한다.

⇒ git push origin <branch_name> : origin remote 에 로컬 브런치를 push 한다.

git 프로젝트는 보통 github 에서 관리된다. 깃허브를 통해 오늘 배운 기본 워크플로우를 사용하면 전세계 어디에서든 여러 프로젝트에 접근할 수 있다.

참고자료

Git 치트시트

https://education.github.com/git-cheat-sheet-education.pdf

Advanced Git

https://thoughtbot.com/upcase/mastering-git

VS 코드 git 버전관리

https://code.visualstudio.com/docs/introvideos/versioncontrol

오늘의 단어

- Unthinkingly: 생각없이, 경솔하게
- retrace : (왔던 길을) 되짚어 가다.
- rewind : (녹음기나 테이프등을) 되감다.
- rewound : rewind 의 pp 형