Github로 협업하기

이 글은 팀프로젝트를 할 때 처음 Git, Github을 사용해서 협업하는 분들을 대상으로 합니다.
작업하는 과정에서 있을 수 있을 법한 일들을 시나리오로 만들었습니다.

상황

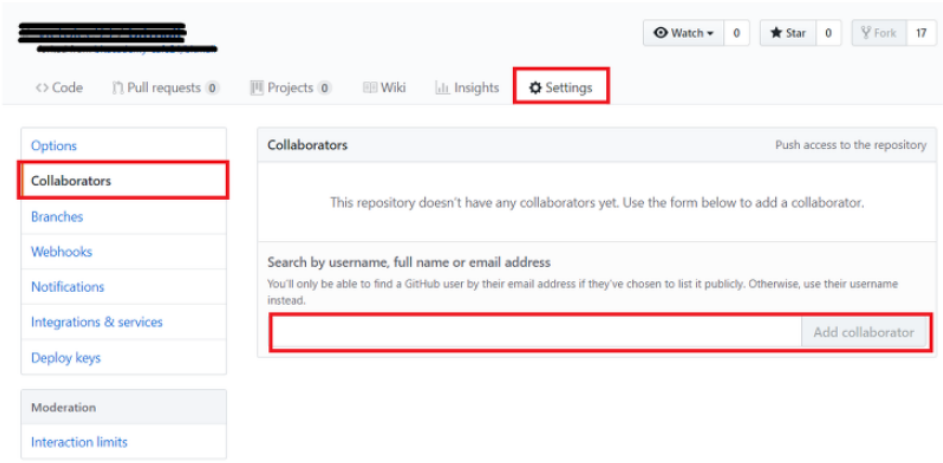
A, B, C 3명이 팀을 이뤄 프로젝트를 진행한다고 하겠습니다.
아래는 시나리오이며, 이 글의 목차이기도 합니다.

1. A는 환경 설정을 마친 프로젝트 파일을 Github에 올립니다.
2. B, C는 프로젝트 파일을 자신의 PC로 가져옵니다.
3. A, B, C는 각각 브랜치를 생성하여 작업을 진행합니다.
4. B가 기능 구현을 마치고 이 파일을 master 브랜치에 병합해서 Github에 올립니다. (master 브랜치에 푸시)
5. A, C는 작업 진행 중에 B가 올린 최신 버전을 갖고 와서 이어서 작업을 진행합니다. (최신버전 받아오기, 충돌 해결)
6. 그런데 C의 작업에서 버그가 발생하여 이전 버전으로 돌아가야 합니다. (revert)

브랜치 전략은 A, B, C 각각 브랜치를 생성해서 독립된 작업을 하고, master 브랜치에서 A, B, C 코드를 병합해서 관리하는 메인 브랜치입니다.

1. A는 환경 설정을 마친 프로젝트 파일을 Github에 올립니다.

A는 Github에 repository를 생성하여 B, C를 collaborator로 추가합니다.
collaborator로 추가해야 B, C가 해당 프로젝트에 pull, push 할 권한이 생깁니다.



그리고 A는 각종 기본 라이브러리들을 추가해서 초기 프로젝트를 셋팅합니다.
스프링이라면 pom.xml, node면 npm, RoR이면 gem 등을 설치하는 것을 의미합니다.
프로젝트에 필요한 모든 라이브러리를 미리 설치하는 것은 알 수 없으므로, 꼭 필요하다는 몇 가지 라이브러리만 설치하면 됩니다.

환경설정을 마쳤으면 B, C가 프로젝트 파일 받을 수 있도록 깃헙에 올립니다.

```
# git init
# git add .
# git commit -m "프로젝트 시작"
# git remote add origin 깃헙주소
# git push origin master
```

2. B, C는 프로젝트 파일을 자신의 PC로 가져옵니다.

B, C는 A가 작업한 프로젝트 파일을 clone해서 가져옵니다.

git clone을 하면 자동으로 remote repository가 등록됩니다.

```
# git clone 깃헙주소
```

3. A, B, C는 각각 브랜치를 생성하여 작업을 진행합니다.

A, B, C는 각각 자신의 PC에서 brchA, brchB, brchC 이름의 branch를 생성해서 독립된 작업 공간을 마련합니다. (**참고**)

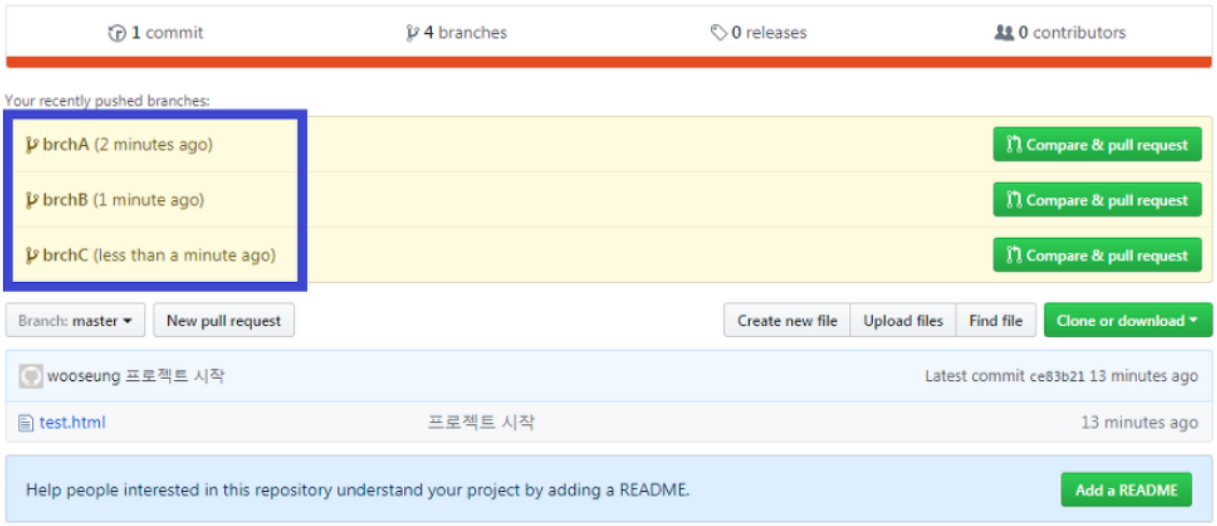
```
# git branch brchA
# git branch brchB
# git branch brchC
```

이제 A, B, C는 깃헙을 공유하고 있는 상황이며, 독립적인 local repository를 갖고 있습니다.

A, B, C는 각자 구현할 기능이 정해져 있으며, 각 기능을 끝낼 때 마다 깃헙에 자신의 브랜치 작업본을 push 합니다.

병합이 필요할 때 master 브랜치에 merge하며, 주기적으로 하는 것이 merge conflict가 발생했을 때 대처가 쉬울 것입니다.

물론 충돌이 발생하지 않도록 작업 범위를 나눠야겠죠.



A, B, C는 자신의 브랜치에서 기능을 구현하여 push하고 있음을 보여줍니다.

4. B가 기능 구현을 마치고 이 파일을 master branch에 병합합니다.

제일 먼저 B가 기능 구현을 마쳤다고 가정하겠습니다.

이제 B는 master 브랜치에 자신의 파일을 올릴 것입니다.

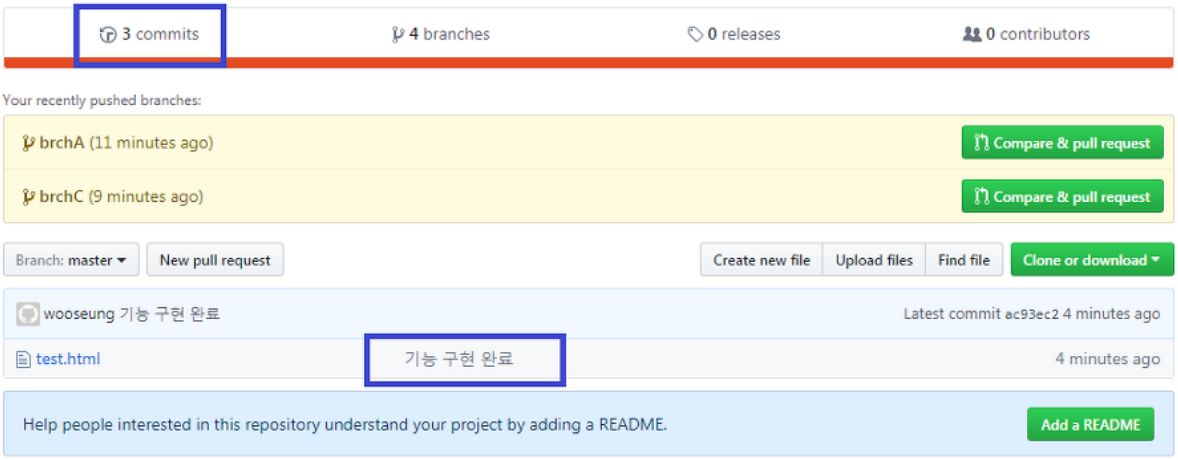
```
// 현재 브랜치는 brchB
# git add .
# git commit -m "기능 구현 완료"

# git checkout master
# git merge brchB
# git push origin master
```

B는 깃헙의 master 브랜치에 push를 하기 전에, 먼저 자신의 local repository에 있는 master 브랜치에 brchB 브랜치를 merge합니다.

그리고 local master 브랜치에서 깃헙 master 브랜치로 push합니다.

자신의 local master 브랜치는 최신 코드를 push 하고 pull하는 용도로만 사용하는 것이 관리면에서 편합니다.

즉, **master 브랜치는 local이든 remote이든 관리만 한다는 것이 핵심**입니다.



깃헙의 master 브랜치에서 확인 결과 B의 작업 결과가 반영된 것을 확인했습니다.

B가 기능 구현 과정에서 commit을 1번 밖에 안했다고 가정했을 때, master 과정에서 commit이 1번 더 발생했으니, 총 2개의 커밋이 추가된 상황입니다.

5. A, C는 작업 진행 중에 B가 올린 최신 버전을 갖고 와서 이어서 작업을 진행합니다.

A, C는 B가 push한 최신 작업본을 사용하기 위해 깃헙에서 master 브랜치를 pull합니다.

위에서 말씀드린 대로 A, C는 pull할 때 master 브랜치로 이동한 후, pull하도록 합니다.

브랜치를 이동할 때, 작업을 마무리 짓고 commit을 한 후 이동해야 합니다. (working directory에 작업내용이 있다면 브랜치 이동이 안됩니다.)

```
# git checkout master
# git pull origin master
```

최신 버전을 가져왔으면, 자신의 작업본에 반영해야 합니다.

A를 기준으로 말씀드리면, brchA 브랜치로 이동 후 master 브랜치를 merge하면 됩니다.

```
# git checkout brchA
# git merge master
```

이 때, 같은 곳을 수정했다면 충돌이 발생할 수 있습니다. (**참고**)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div>
Use ne                                                    our changes
<<<<<<< HEAD
    <span>브랜치A 구현 중</span>
  </div>
=======
    <span>브랜치B 기능 구현 완료됨 !</span>
  </div>

  <span>기능1</span>
  <table>
    <tr>
      <td>기능2</td>
    </tr>
  </table>
>>>>>>> ac93ec295a9e156eda6b5a5478589ad521f3ea14
```

위의 예는 충돌이 발생한 경우입니다.

현재 브랜치는 brchA니까 <<<<<< HEAD는 brchA의 영역을 뜻합니다.

본인이 수정하지 않았는데 충돌이 발생한거면 상대방의 소스를 반영하면 되지만, 그래도 B와 상의하는 것이 좋을 것 같습니다.

어떤 코드를 적용할지 결정이 되면, 수동으로 작업을 해주면 됩니다.

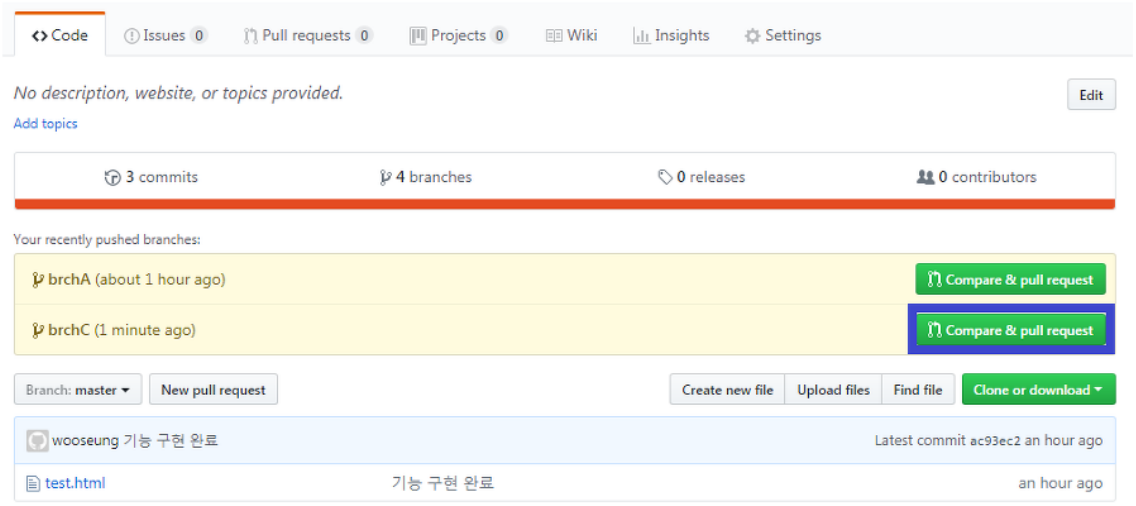
6. C의 코드에서 버그가 발생하여 이전 버전으로 돌아가야 합니다.

Github을 통해 A, B, C가 push, pull을 수행하면서 작업이 원할하게 진행되고 있었습니다.

그런데 C가 테스트 도중 버그가 발생해서 한참 전의 상태로 돌아가야 하는 상황이 발생했습니다.

C가 버전을 local repository에서만 관리했다면 reset, revert 명령어 둘 중 하나를 사용할 수 있지만,
push를 하여 깃헙에도 올라간 상황이라면, revert 명령어만 사용할 수 있습니다. (**참고**)

여기서는 push를 한 상황이라 가정을 하고, 버그가 발생하기까지 총 5번의 commit을 했다고 가정하겠습니다.



Github에서 버튼을 클릭하시면 commit 이력을 보여주는 아래의 사진을 볼 수 있습니다.

현재 마지막 커밋 내용은 "C에서 5번째 기능 구현" 버전 (6d5ed94)입니다.
버그가 발생했기 때문에 안전하다 생각되는 곳은 "C에서 2번째 기능 구현" 버전이라 생각되어, f37a6b7 버전으로 돌아가려 합니다.
그러면 "C에서 3번째 기능 구현" 버전의 commit 번호(1d9184f)로 revert를 하면 됩니다.

참고로 깃헙에서 보는 방법 말고 log를 통해 확인할 수도 있습니다.

```
# git log --online -10
```

해당 버전의 commit 번호를 복사해서 revert 명령어를 실행합니다.

```
# git revert commit번호
```

그러면 brchC 브랜치의 파일에서 충돌이 발생한 것을 확인할 수 있습니다. (충돌 발생은 안할수도 있습니다.)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <div>
    <span>브랜치C 구현 중</span>
  </div>

  <span>기능 1 완료</span>
  <span>기능 2 완료</span>
Use me ... our changes
<<<<<<< HEAD
  <span>기능 3 완료</span>
  <span>기능 4 완료</span>
  <span>기능 5 완료</span>
=======
>>>>>>> parent of 1d9184f... C에서 3번째 기능 구현
Use me ... their changes
</body>
</html>
```

기능 2까지 완료된 버전으로 돌아가고 싶었던 것이므로 아래의 부분을 제거합니다.

~~<<<<<<< HEAD~~

~~기능 3 완료~~

~~기능 3 완료~~

~~기능 3 완료~~

~~==========~~

다음으로 이전 버전으로 돌아갔으니 커밋을 해서, 이력을 남기는 것이 좋습니다.

```
# git add .
# git commit -m "기능2로 백업"
# git push origin brchC
```

revert는 이력을 남기고 버전을 되돌리는 명령어이므로 기능3, 4, 5에 대한 커밋 이력은 남아있을 것입니다.