2023년 2월 8일
Git과 Github 개념
보고서

Git과 Github 개념

임원호

-목차-

Ⅰ. Git이란?

1. Git의 탄생

'Linux 커널은 굉장히 규모가 큰 오픈소스 프로젝트이다. Linux 커널의 삶 대부분은(1991~2002) Pacth와 단순 압축 파일로만 관리했다. 2002년에 드디어 Linux커널은 BitKeeper라고 불리는 사용 DVSC(분산 버전 관리 시스템)를 사용하기 시작했다.'[1] 하지만 오픈소스를 추구하는 Linux 커널과 이익을 추구하는 BitKeeper와의 관계는 틀어지게 되었고, 이 사건으로 인해 Linus Torvalds가 자체 도구를 만들게 되었으며, Git은 BitKeeper를 사용하게 되었다.

2. Git과 Github의 차이

'Git은 기본적으로 독립적인 공간에서 개인이 사용하는 것이다.'[2] 개인이 코드와 그에 따른 기록을 관리하며, 버전을 관리할 수 있게 한다. 다만 Git은 팀과의 협업에서 독립적인 공간에서만 이루어지기 때문에, 실시간으로 또는 후에 다른 작업자의 공간에 가기 전까지, 서로 작업하는 내용을 비교할 수 없다.

---

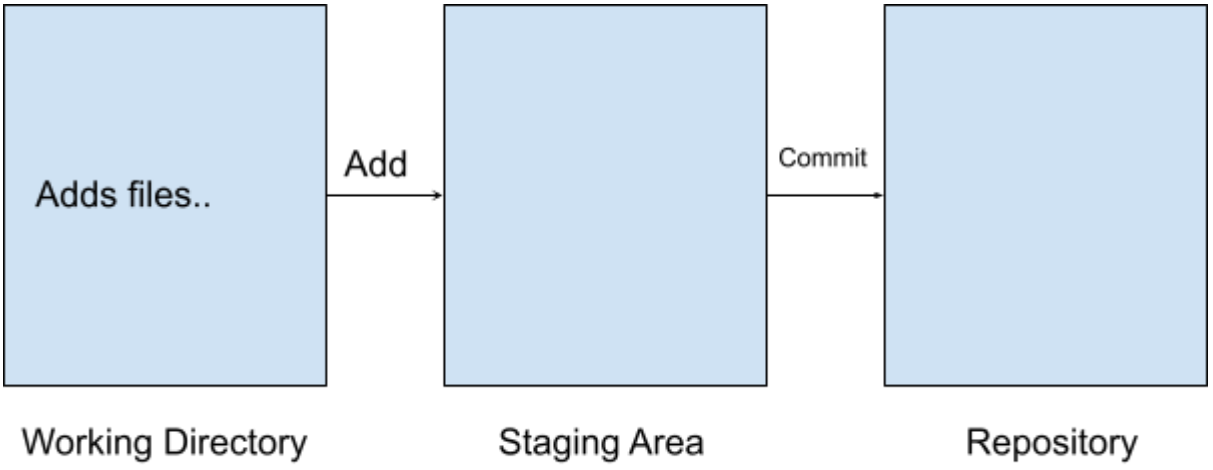
[1] Git, "Git의 역사", https://git-scm.com/book/ko/v2/, (2023.02.08.최종검색).
[2]escapefromcoding, "Git과 Github의 차이", https://escapefromcoding.tistory.com/281, (2023.02.08.최종검색).

Github은 Git의 단점을 보완하여, 클라우드를 기반으로 호스팅하는 서비스로 Git 저장소를 클라우드에 올려 개인도 클라우드에 접근하고, 팀 단위 협업도 클라우드로 들어가기 때문에, 실시간으로 버전을 관리할 수 있다. 쉽게 말하면, Github은 온라인 데이터베이스로 생각하면 이해하기 더 쉬울 수 있다. Github은 현재 마이크로소프트가 인수하였고, Github외에도 클라우드를 기반으로 호스팅하는 Git과 같은 서비스는 GitLab, BitBucket, SourceForge 등이 있다. 웹으로 접근이 가능해, CLI환경보다 쉬운 유저 인터페이스를 제공하는 게 특징이다.

Ⅱ. Github에서 자주 사용하는 용어

1. Add

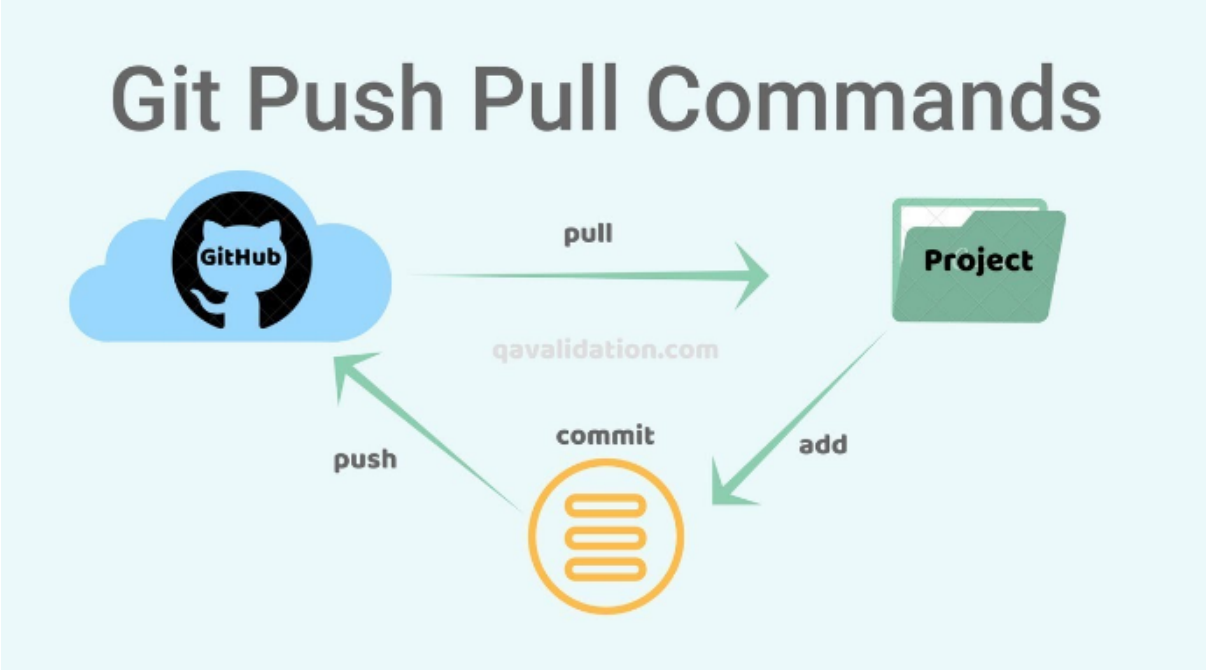
일반적으로 Github Repository에 본인이 수정하거나 작업했던 내용들을 올릴려면, Repository에 가기 전, Commit 전에 git add 올릴파일을 먼저 쳐야한다. git은 어떻게 작업 중인 공간과 Repository를 구분할까? 먼저, Working Directory, Staging area, Repository의 3개의 공간으로 분리가 된다. git add를 하게 되면 Working Directory에 있는 파일 중, add 했던 파일 들만 Staging Area로 이동하게 된다. 3개의 독립적인 부분은, git status를 입력하게 되면 알 수 있다. Staging Area는 8번에서 더 자세하게 다루겠다.



2. Commit

Commit은 Git에서 add한 파일들을 Repository로 이동하기 전까지, 준비해 놓는 상태이다. 준비해 놓는 상태이므로, add를 잘못했거나, 다른 것들을 추가하는 것 또한 가능하다. 예를 들면, 친구들과 여행을 갈 때, 차에 친구들을 불러 태우는 것까지, add라고 한다면, Commit은 차에 키를 꽂고 시동을 거는 것과 같다. 목적지까지 출발하면 내릴 수 없을 때, 시동만 걸었으므로 친구들을 언제든 변경할 수 있다. Commit에 대한 기록은 git log를 통해 확인 할 수 있다. 누가 언제 보냈는지, 확인할 수 있다.

3. Push



Push는 말 그대로 Repository로 내가 add하고, commit까지 거쳐 완벽하게 솎아낸 파일들만 보내겠다는 의미이다. Push를 하게 된다면, Repository에 있는 파일 중 add 된 파일에서 수정이 된 것들이 update가 된다. push를 하게되면, 생각보다 많은 오류를 발견할 수 있는데, 먼저 push 할 때 사용자의 이름과 토큰의 비밀번호가 맞지 않는 경우도 있다. 또한 Repository주소를 잘 못 적어 upstream을 하라는 엉뚱한 대답을 얻을 때도 있다. 이런 오류는 미리 공부하며 피할 수 있지만, 나는 오류가 나면 그때마다 인터넷을 찾아보며 해결하는 것을 추천한다. 암기하듯이 대처하면, 오류를 무서워하게 된다.

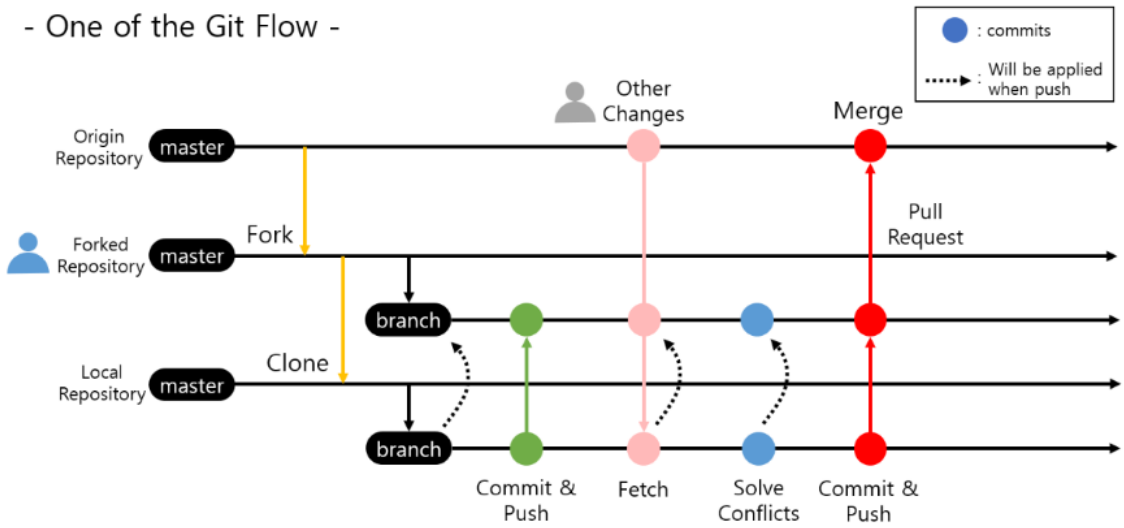
4. Pull request

Pull request는 기본적으로 팀 단위 프로젝트에서 협업할 때 사용된다. 하나의 Repository를 가지고, 여러 사람이 단체로 Push를 하게 되는 경우 (그럴 리는 없지만..)오류가 나거나 난잡하게 일이 벌어질 수 있다. 예를 들면 100명이 모여서 하나의 컴퓨터를 잡고 보고서를 작성하는 것과 같다.. 생각만해도 끔찍하다. Git에서는 그럴 일이 없다. 왜냐하면, Fork라는 개념이 존재하기 때문이다. 쉽게 생각하면, 기존 원본 Repository는 그대로 두고, 원본 Repository를 포크로 찍어와, 나의 Github Repository로 끌고 오는 것이다. 나는 Fork 된 Repository를 내 컴퓨터로 Clone 하여, 작업을 진행하게 된다. 작업이 모두 끝난 후 우리는 Fork된 Repository에 Push를 한 뒤, Pull request를 하게 되면, Origin Repository 관리자의 승인 후 Origin Repository로 병합되게 된다.

5. Merge

Fork를 통해 생성된 자기 자신만의 영역을 Branch라고 해보자, 이제 이 Branch를 Pull request를 통해 Origin Repository로 합치고 싶을 때, Merge를 한다. Merge는 병합이라는 의미로 말 그대로 작업이 끝난 내용을 합칠 것이냐 말 것이냐를 묻는 것이다. 아주 좋은 예시가 있다.

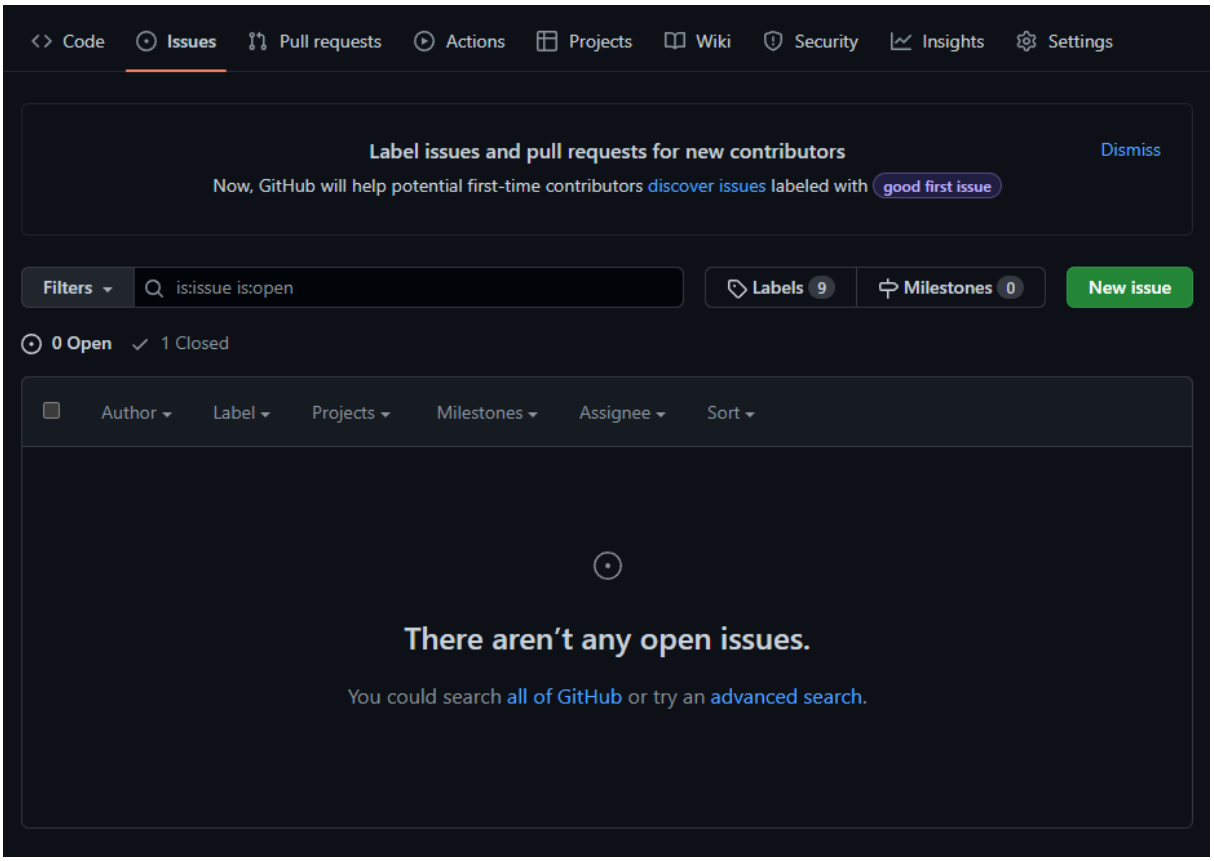
pull request 사용 단계



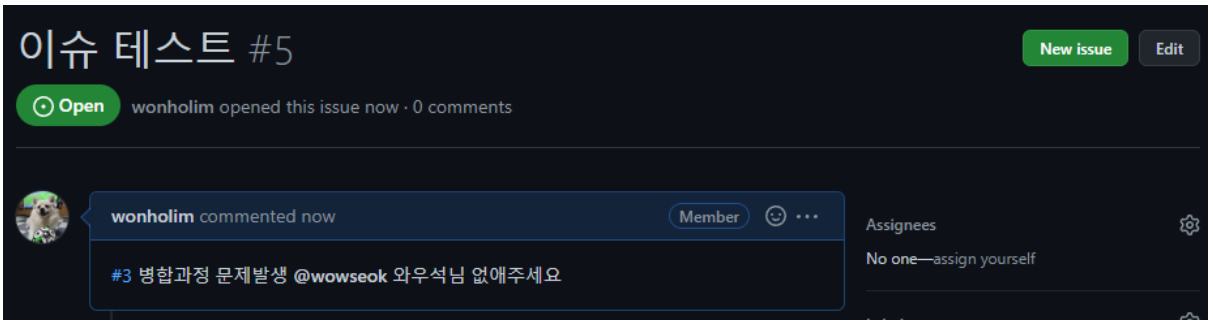
출처 - ([GIT] ⚡ 깃헙 Pull Request 보내는 방법 - 알기 쉽게 정리)

6. Issue

이슈는 말 그대로, 작업한 내용이 잘못되었거나, 새로 해야 할 내용들을 부른다. 이거 누가 작성하셨죠?? 이거 당장 고쳐주세요! 와 같이하거나, 팀에서 대응해야 할 이슈가 있다면, 이슈에 올린 뒤 해당하는 Github에 닉네임을 적어주면 된다. 이슈가 해결되면 Complete를 눌러주면 해결된다.

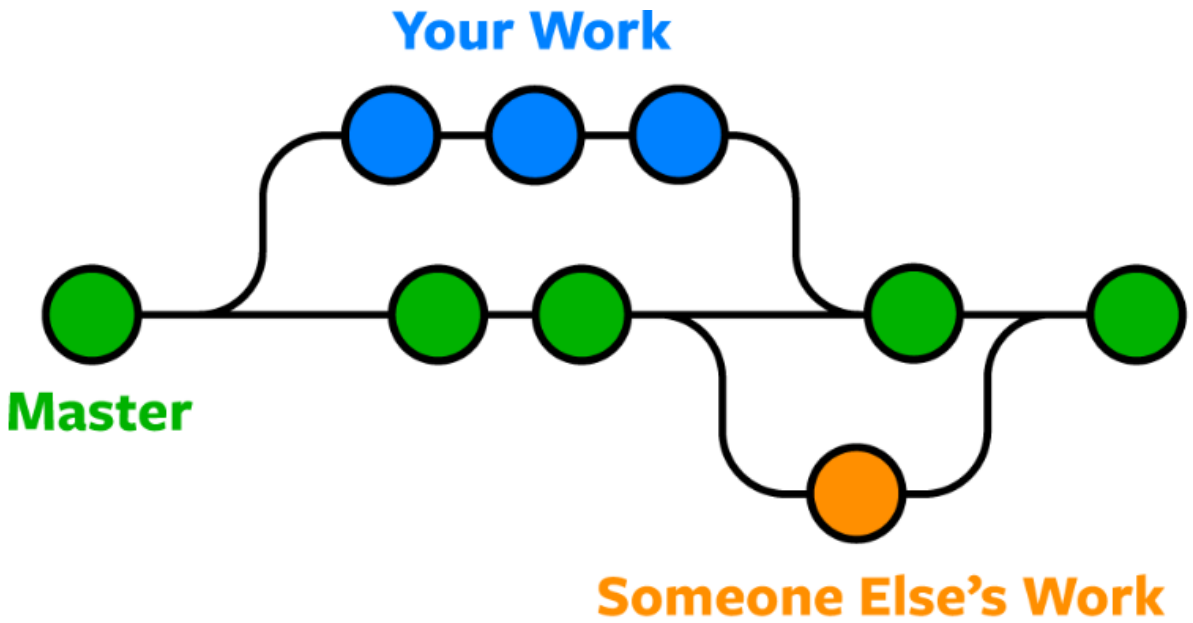


(이슈가 없는 경우)



(이슈가 있는 경우 닉네임 태그와 Pull request한 것 까지 체크가 가능하다.)

7. Branch



출처 - (https://velog.io/@gil0127/Git-branch)

Git 명령어에 대한 기본적인 지식은 모두 알게 되었다! 근데 Branch는 왜 필요한 거지? 라고 생각할 수 있다. 팀과 협업을 할 때는 항상 잘되는 일만 존재하는 게 아니다. 모든 것이 완벽하고 잘 돌아가더라도, 실제로 사람들에게 배포해보면 이전 버전이 더 나은 경우가 생긴다. 그때에는 branch를 이용해 뒷 버전으로 돌아갈 수 있다. (물론 백업데이터로 하겠지만..) 일단 먼저 팀과 협업할 때 절대 Origin Repogitory에서 작업하면, 안된다. 무조건 자신만의 브런치를 내어 클론을 하거나, 포크하여 클론한 뒤 Origin Repogitory에서 Pull request를 하면 된다. 요약하자면, 당신이 팀과 협업할 때, 작업을 하게 된다면, 브런치를 내서 작업을 진행해야 한다. 당신이 잘못해도 브런치를 없애거나, 이미 Merge가 된 상태여도, 다시 브런치 이전의 내용으로 복구가 반드시 가능하다. 조금 더 자세한 내용이 필요하다면, 링크를 ..
(https://git-scm.com/book/ko/v2/Git-%EB%B8%8C%EB%9E%9C%EC%B9%98-%EB%B8%8C%EB%9E%9C%EC%B9%98%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80)

8. 그 외 다양한 용어

Pull 명령어 외에도, Fetch 명령어가 존재하는데.. Fetch는 내가 작업한 데이터와 Origin Repogitory의 내용을 병합하지 않고, 저장소의 내용만을 확인할 때 사용한다. 허나 Fetch를 하게 되면, Branch가 Fetch_Head의 이름으로 되게 되는데, 이 상태에서 Branch를 Merge 하게 되면, 같은 이력으로 남게 된다. 즉 git Pull 명령어와 git fetch + merge는 서로 같다는 의미이다. 말이 좀 어렵다면, Fetch는 Origin Repogitory의 최신 이력을 알기 위해 사용하는 명령어로 알고 있어도 무방하다. 내가 사용할 때의 fetch는 이런 용도로 사용한다. 팀과 협업할 때, 누군가가 이미 업데이트를 진행했는지 확인하기 위해 받아오는 것이다. A, B, C라는 파일을 Branch로 가져왔을 때, 누군가가 D 파일을 Origin Repogitory에 올렸다면, 나는 Fetch를 통해 다시 받아와 내 Branch를 Origin Repogitory와 동일하게 최신상태로 유지하기 위해 사용한다. pull과 fetch의 차이점 꼭 알아야한다.

[참고 문헌]

Git, “Git의 역사”, https://git-scm.com/book/ko/v2/, (2023.02.08.최종검색).

escapefromcoding, “Git과 Github의 차이”, https://escapefromcoding.tistory.com/281, (2023.02.08.최종검색)

출처 - ([GIT] ⚡ 깃헙 Pull Request 보내는 방법 - 알기 쉽게 정리)

출처 - (https://velog.io/@gil0127/Git-branch)

(https://git-scm.com/book/ko/v2/Git-%EB%B8%8C%EB%9E%9C%EC%B9%98-%EB%B8%8C%EB%9E%9C%EC%B9%98%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80)