Git, github 스터디 03

2021-2 KCA

김지환

본 ppt의 자료는 git book, github을 참고하였음을 밝힙니다.

[실습] github fork

- github에서 다른 사람의 repository 자체를 복사해 오는 기능
- 오픈 소스 프로젝트의 출발점이자 안전장치
- 저장소 전체의 분기(branch) 개념

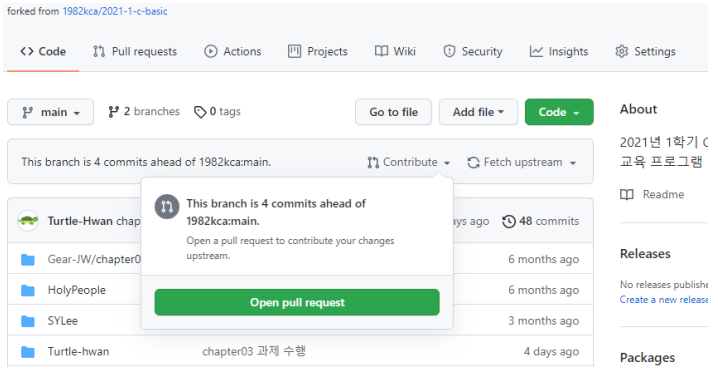
[실습] github pull request PR

merge reques라고도 함. (gitlab)

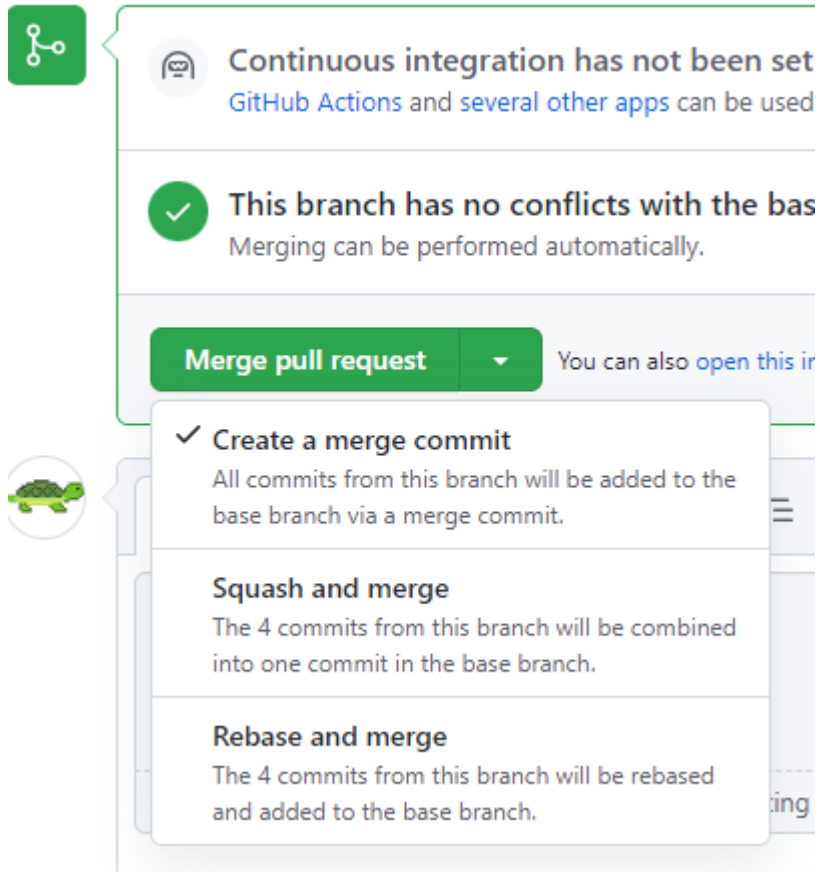
- fork된 저장소 -> 원본 저장소로 코드 통합 요청. (같은 저장소의 branch 간에도 가능하다!)

- 다른 사람이 쓴 코드가 바로 merge되기 전, 한 번 확인하는 절차. 코드 리뷰 등.

- github 기능, github에서 해야 한다!



Pull Request 정책



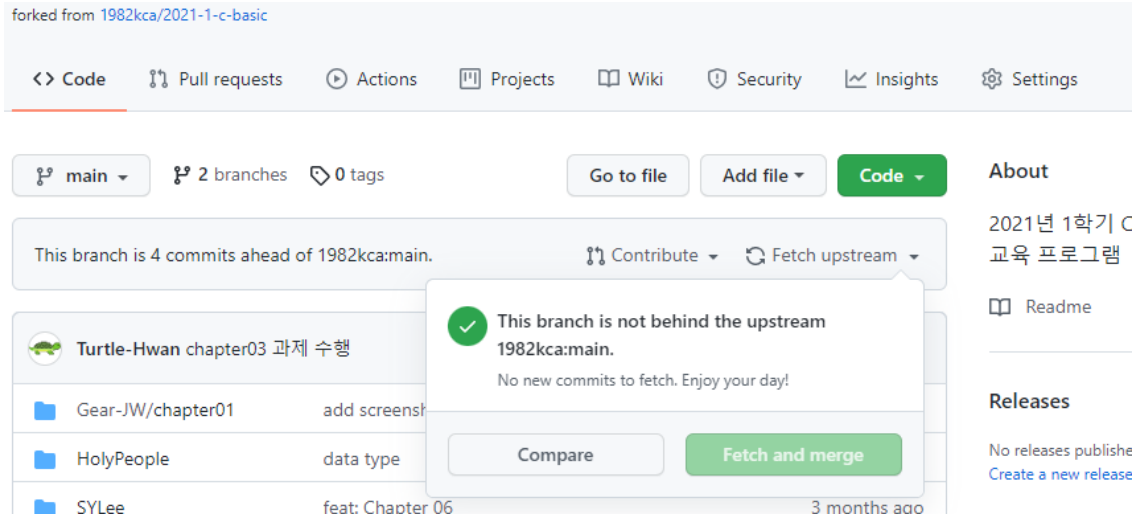
- Create a merge commit : 3-way merge로 새로운 merge commit 생성
- Squash and merge : fork에서 만든 commit들을 하나의 commit으로 압축해서 base에 결합
- Rebase and merge : fork에서 만든 commit들을 rebase해서 붙임.
  새 commit들이 추가만 되었다면, fast-forward merge와 같아진다.

PR 이전에 원본 저장소 fetch & merge

원본 저장소와 forked 저장소를 동기화 해주어야 한다.

1. github에서 하는 방법

   오른쪽 사진의 버튼 위치



2. 로컬 git bash에서 하는 방법

   remote 지정한 후, 여기서 로컬로 fetch & merge

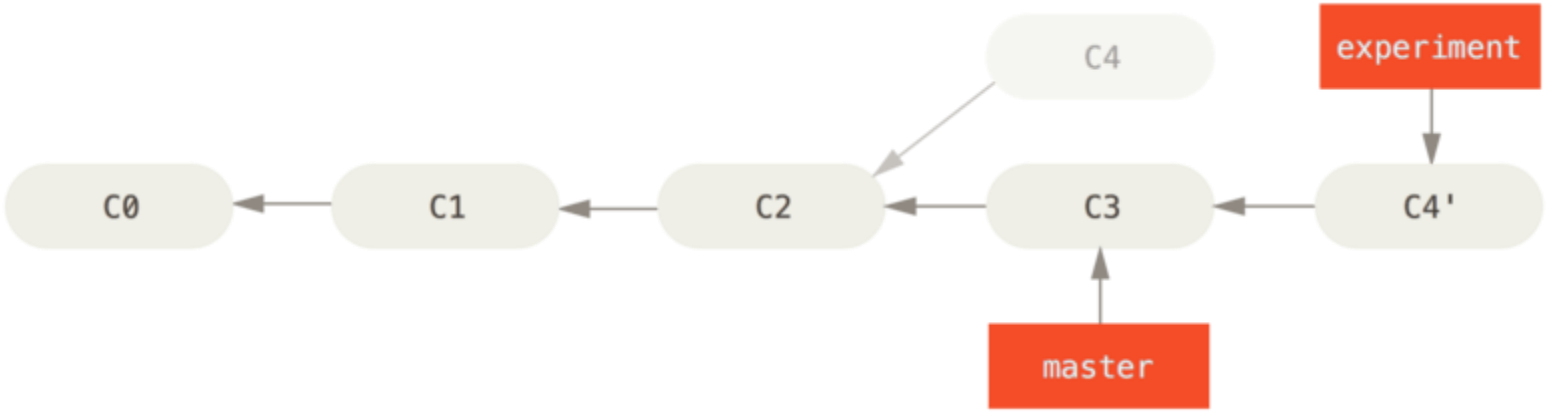
   git remote update
   git merge origin/{브랜치이름}

git remote

- git remote -v : 저장된 원격 저장소 주소 확인.
- git remote add {간편이름} {저장소 주소} : 간편 이름으로 부를 저장소 설정
- git remote rename {이름A} {이름B} : A를 B로 바꿈.
- git remote rm {이름A} : A와 여기 연결된 주소 삭제.
- git remote update == git fetch {이름}

git rebase

- commit 들의 base(조상 commit)을 다시 지정하는 것.



git checkout experiment
git rebase master

- 마치 하나의 branch인 것 처럼 깔끔하게 commit history를 정리 가능하다.
- 하나의 branch 위에서도 rebase 사용 가능하며, commit을 수정한 결과가 나온다.

=> 이미 공개 저장소에 push 된 commit 을 rebase 해서는 안된다!!!

git stash

- commit을 하지 않고 checkout으로 다른 branch로 넘어가면 작성 하던 내용은 사라진다.

- 이때, git stash는 현재 git repository 상태를 임시 저장 해준다.

- git stash pop : 임시 저장된 것 꺼내오기

[추가] git HEAD

- 보통의 HEAD는 브랜치를 통해 커밋을 간접적으로 가리킴 (Attached HEAD)

- Detached HEAD : 브랜치를 통하지 않고 커밋을 가리킴

- git은 브랜치 기반으로 커밋들을 관리해서, Detached HEAD는 브랜치에 연결해 주어야 한다.

issue 기반 개발과 commit message

- github의 issue 기반 개발

issue에 오류 뿐 아니라, 앞으로 개발해야 할 기능 등을 등록한 후에 이를 바탕으로 진행.

- commit message 규칙

처음에 관련 issue, 구현 기능 등 요약 한 줄 이내.

한 줄 띄우고 자세한 내용 적기.

github Tag

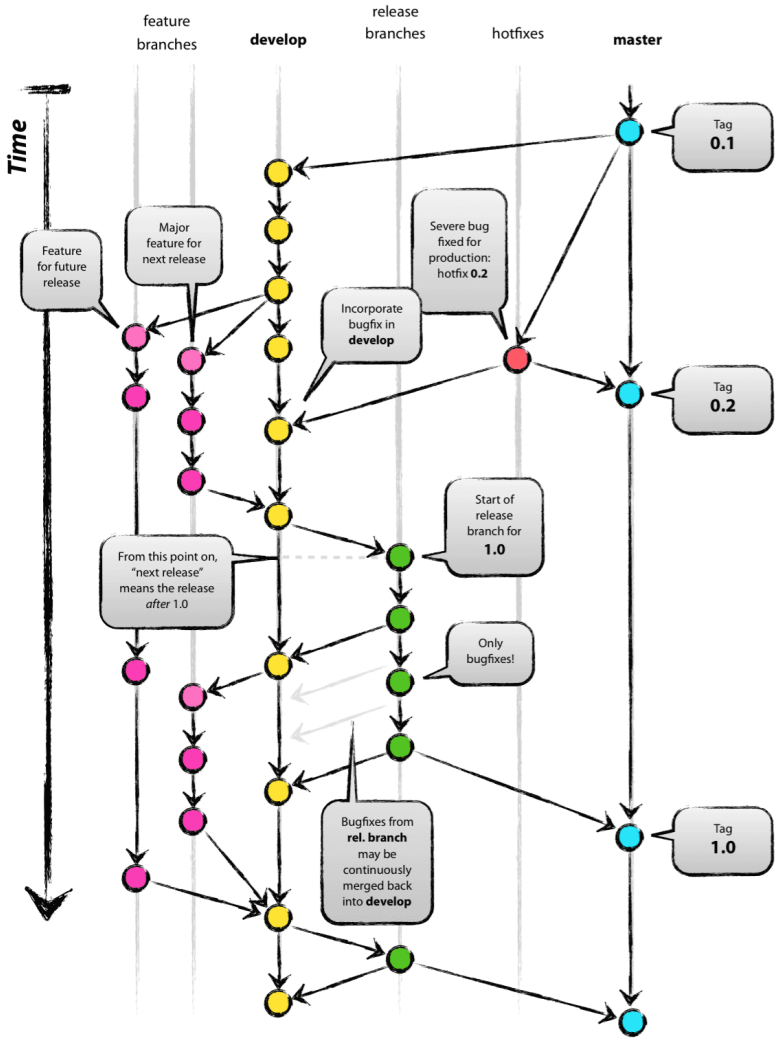
- 커밋에 Tag를 붙여서 관리할 수 있다.

- (주로 배포 버전이 변할 때 붙임)

git flow, github flow

- 실제 개발 시 git branch 관리 흐름, 전략
- github에서 더 간단한 flow 제안한 것이 github flow.
- github flow 이후 제안된 gitlab flow도 있음.

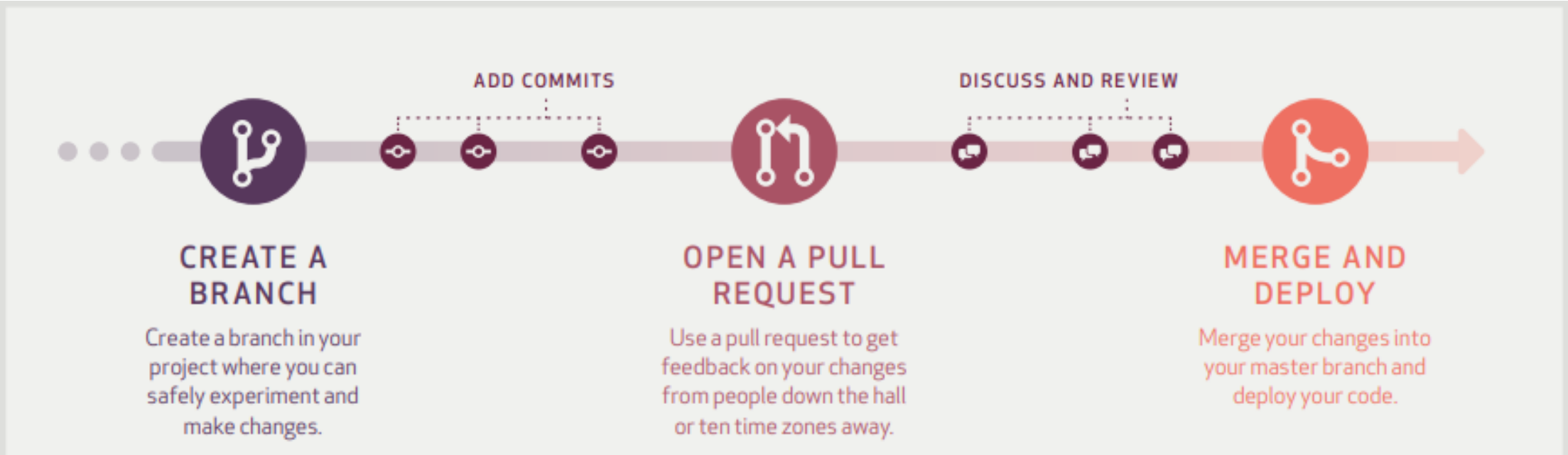
git flow



- Master Branch (항상 유지) : 현재 배포되어 있는 메인 버전이 있는 브랜치
- Develop Branch (항상 유지) : 현재 개발을 진행하는 브랜치, 개발 기준, 시작점.

- Feature Branch (merge) : 새로 개발하는 기능을 위한 브랜치
- Release Branch (merge) : 개발을 마치고 배포 전 테스트를 위한 브랜치 (feature가 develop으로 merge 완료 후, 배포 테스트)
- Hotfix Branch (merge) : Master 브랜치의 버그 픽스를 위한 브랜치

github flow



- master 하나만 지속적으로 존재하며,
- 필요할 때마다 추가 브랜치 파고, 일이 끝나면 PR, Review, Merge 한다.

- branch가 왜 만들어졌는지 의도 파악을 하기 위해 이름을 잘 지어야 한다.

[추가] github 프로필 꾸미기 (markdown)

- 다른 사람의 소스 코드를 적극적으로 확인해 보자!