Git 실습 - 2021.10.16(토)

지난 시간 못한 이론 짚고 넘어가기

- 커밋 수정하기 part

실습 1. 개인 프로젝트

[remote & local] 기본 세팅

1. remote 저장소 생성 (with `README.md`)
2. local 저장소로 clone

```
git clone {저장소 url}
```

3. git 기본 설정

```
# set name and email
git config --global user.name "Eunbi"
git config --global user.email "eunbi@gmail.com"# show all configs
git config --list
```

[remote] 이슈 생성

1. github.com > 생성한 저장소 > Issue 메뉴
2. **Issue 생성**
    a. 제목 : Fix readme.md
    b. 내용 : 자유롭게 🙂

[local] 실습용 브랜치 생성

```
git checkout master # or git checkout main (본인 기본 브랜치에 따라 다름)
git branch

git branch fix-readme
git checkout fix-readme
```

[local] git 명령어를 사용하여 커밋 추가해보기!

1. README.md 파일 수정하기
2. 아래 명령어로 커밋 추가하기

```
git diff
git status

git add
git status

git commit

git log
# log with options
git log --graph --oneline
```

[local → remote] 저장소에 올리기

```
git branch
git fetch origin
git push origin fix-readme
```

[remote] Pull Request & Merge

1. github.com > 생성한 저장소 > Pull Request 메뉴
2. Pull Request 생성
    a. base : `master`
    b. compare : `fix-readme`
3. Pull Request에 comment 달아보기
4. Merge : 아래 세 가지 방법 중 택 1
    a. Merge pull request = `git merge --squash`
        - 현재 코드를 그냥 머지
    b. Squash and merge = `git merge --squash`
        - 여러 커밋을 하나의 커밋으로 만들고(squash) 머지
    c. Rebase and merge = `git rebase && git merge`
        - 현재 코드를 base branch 기준으로 rebase 후 머지
5. github.com > 생성한 저장소 > Issue 메뉴
6. Close Issue

[local] fetch & merge (pull)

1. 로컬 저장소에서 `README.md` 파일 수정
2. 변경사항 임시 저장해보기 (WIP = Work in progress)

```
# 변경 사항 확인
git status
git diff # README.md 파일 변경된 것이 보이겠죠?

# stash 명령으로 변경사항 잠깐 저장
git stash
```

3. 원격 저장소의 변경사항 로컬에 반영하기

```
git fetch origin
### origin(원격 저장소)가 변경되었다는 로그가 나올거에요
```

```
git merge
### 코드 머지
```

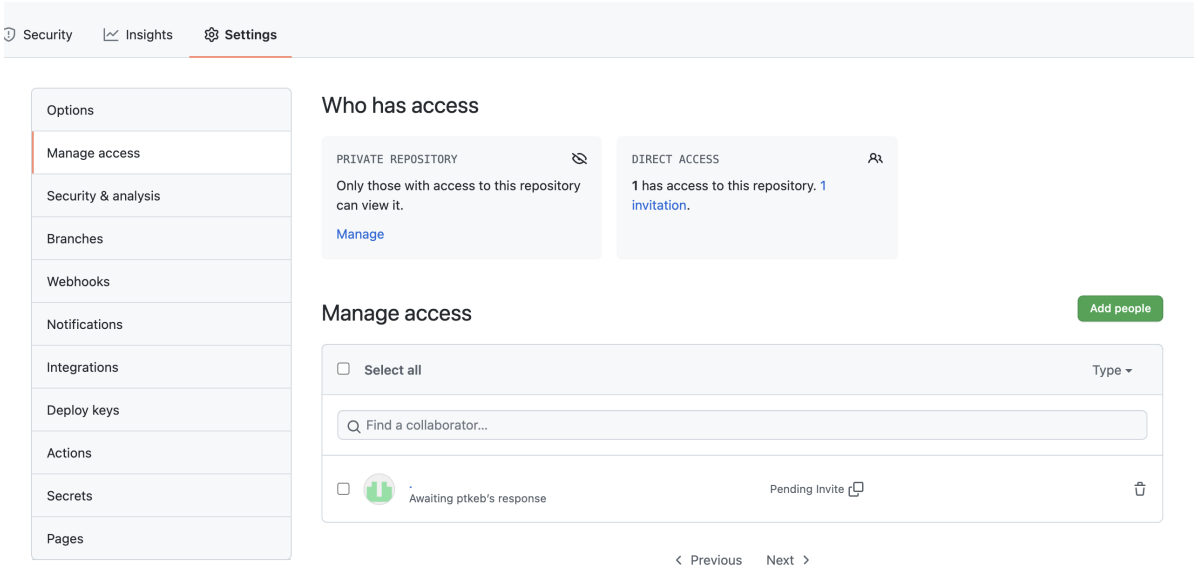
4. 다시 stash로 저장한 변경사항 가져오기

```
git stash list

git stash pop 0
# 아마 충돌이 날 거에요!

vi README.md # 로 충돌을 해결합니다

# 확인
git status
git diff
```

실습 2. 협업 프로젝트

사전작업 : 조 편성

- drive 파일에 수강생 리스트 작성 + 본인 이메일 작성
- 조에서 맨 위에 있는 사람이 조장
  - github.com에 저장소 생성 후, 조원 초대 해주시면 됩니다!
  - github.com > 생성한 저장소 > **Settings > Manage access > Add people**



[remote & local] 기본 세팅

1. 🔥조장(=팀원1)만 해주세요🔥 github.com 에서 remote 저장소 생성 (with `README.md`)
2. 조장 & 팀원 모두 local 저장소로 clone

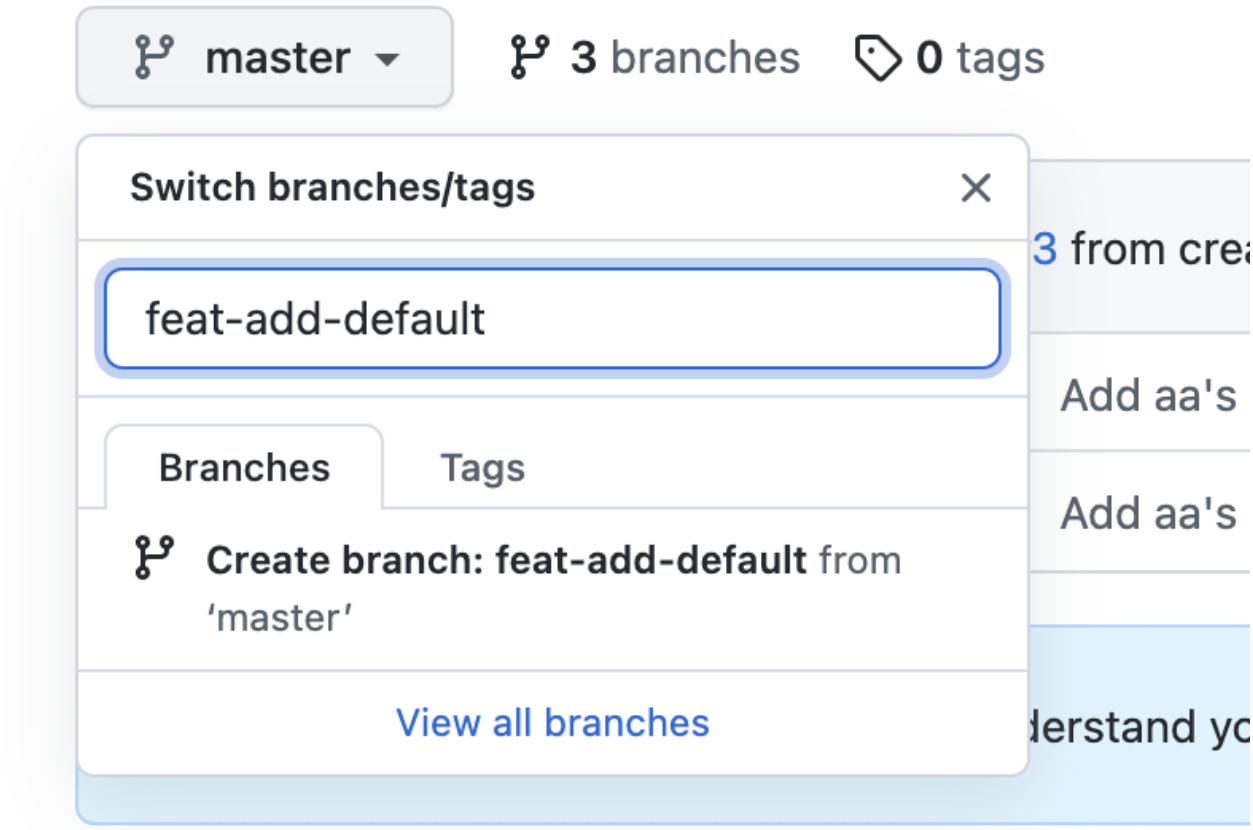
```
git clone {저장소 url}
```

3. git 기본 설정 (개인 실습 때 설정하였으면 생략 가능)

```
# set name and email
git config --global user.name "Eunbi"
git config --global user.email "eunbi@gmail.com"# show all configs
git config --list
```

[remote] 충돌용 파일 생성

- 🔥팀원2만 해주세요🔥
- github.com > 협업용 저장소 > 브랜치 버튼 > 새로운 브랜치 생성 아래와 같이 해주세요!



- github.com > 협업용 저장소 > `Add file` > `Create new file` 메뉴에서 `hobby.md` 파일 생성하고 아래 내용 복사해서 붙여넣기 해주세요.

```
# 조원과 취미목록
## 조원 목록
1. {조원번호} - {닉네임}

## 조원의 취미
1. {닉네임} : {자기 취미1}, {자기 취미2}, .... {자기 취미n}

## 마지막 편집자
- default
```

[remote] 이슈 생성

1. github.com > 생성한 저장소 > Issue 메뉴
2. **Issue 생성**
    a. 제목 : [이름] Git 협업 연습용 자료
    b. 내용 : 자유롭게 🙂

[local] 실습용 브랜치 생성

```
git checkout master # or git checkout main (본인 기본 브랜치에 따라 다름)
git branch

git branch feat-{nickname} # feat = feature
git checkout feat-{nickname}
```

[local] 파일 수정하고 커밋하기

1. `hobby.md` 만들고 내용 채워넣기 & 각자 만들고 싶은 파일 아무거나 만들기

```
# hobby.md
# ================

# 조원과 취미목록
## 조원 목록
1. {조원번호} - {닉네임}

## 조원의 취미
1. {닉네임} : {자기 취미1}, {자기 취미2}, .... {자기 취미n}

## 마지막 편집자
- {닉네임}
```

2. 아래 명령으로 커밋 작성

```
# 변경사항 확인
git diff
```

```
# 상태 확인
git status
git add .

# 커밋
git commit
# 제목 : Add {nickname}'s hobby list
# 내용 : 한 줄 띄고!
#       - 하고싶은 말
#       - Related Issue : #1 ### 아까 생성한 이슈 번호 등록!

# 상태 확인
git status

# 로그 확인
git log
```

[local → remote] 저장소에 올리기

```
git branch
git fetch origin
git push origin feat-{nickname}
```

[remote] 충돌용 파일 Pull Request & Merge

- 🔥팀원3만 해주세요🔥

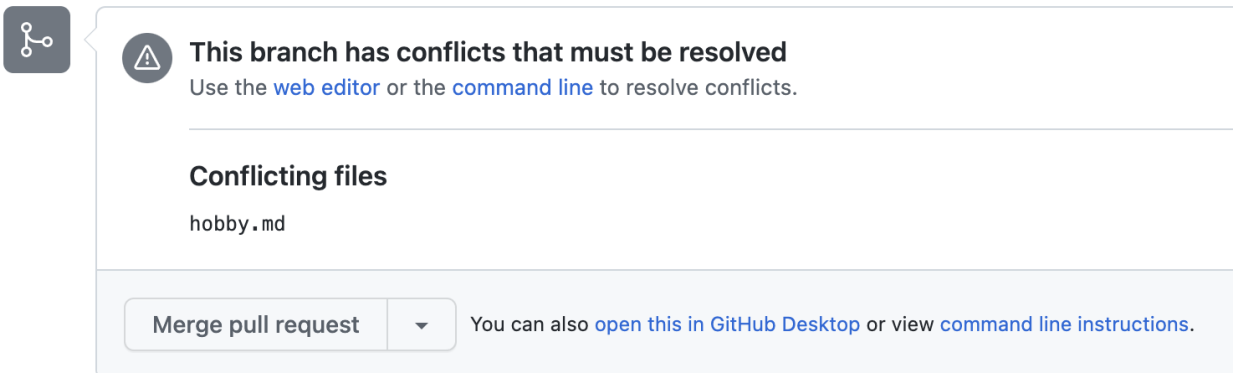
1. github.com > 생성한 저장소 > Pull Request 메뉴
2. Pull Request 생성
    a. base : `master`
    b. compare : `feat-add-default`
3. 내용은 자유롭게 작성하고 Pull Request를 생성합니다
4. `Merge Pull Request` 버튼으로 master에 머지 진행!

[remote] github.com 에서 브랜치 확인하기 & Pull Request 생성

1. github.com > 협업용 저장소

- 본인이 올린 브랜치가 반영이 되었는지 확인! 정상적으로 push가 되었는지 확인하고 Pull Request를 생성

1. github.com > 생성한 저장소 > Pull Request 메뉴
2. Pull Request 생성
    a. base : `master`
    b. compare : `feat-add-default`
3. 내용은 자유롭게 작성하고 Pull Request를 생성
4. 생성 후에 아래와 같은 conflict이 발생한다면 성공 🙂



[local] 충돌 해결하기

1. 로컬 개발환경으로 돌아옵시다
2. 원격 저장소의 변경사항을 로컬에 반영하기

```
git checkout master
git fetch origin
### 변경되었다는 로그가 나오겠죠?
git merge
# or git pull 로 한번에 해결 가능!
```

3. 로컬 저장소의 feature 브랜치로 이동

```
git checkout feat-{nickname}
```

4. feature 브랜치를 master 브랜치 기준으로 rebase
    - 코드의 기준(base)를 최신 master 기준으로 변경하는 작업

```
git rebase master
```

5. 로컬 저장소에서도 충돌이 발생했을 것
    - 지난시간에 말씀드린 것 처럼, 로그에는 많은 설명이 담겨 있으니 주의깊게 봐주세요

```
$ git rebase master
Auto-merging hobby.md
CONFLICT (content): Merge conflict in hobby.md
error: could not apply 6eed5ee... Add eb's hobby and files
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 6eed5ee... Add eb's hobby and files
```

6. `hobby.md` 파일을 아래와 같은 형태로 잘 수정
    - 이전 작업자의 내용이 사라지지 않고 잘 어우러지게 해주세요

```
# 조원과 취미목록
## 조원 목록
1. {조원번호} - {닉네임1}
1. {조원번호} - {닉네임2}
....

## 조원의 취미
1. {닉네임1} : {자기 취미1}, {자기 취미2}, .... {자기 취미n}
1. {닉네임2} : {자기 취미1}, {자기 취미2}, .... {자기 취미n}
```

7. 수정한 사항을 Staging area에 등록

```
git diff --color
git status
git add hobby.md
```

8. git status명령어로 잘 반영이 되었는지 확인

```
interactive rebase in progress; onto 73d9639
Last command done (1 command done):
   pick 6eed5ee Add eb's hobby and files
No commands remaining.
You are currently rebasing branch 'feat-hobby-eb' on '73d9639'.
  (all conflicts fixed: run "git rebase --continue")

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   hobby.md
```

9. rebase 완료

```
git rebase --continue
```

10. 원격 저장소로 **강제** 푸시

```
git push -f origin feat-{nickname}
```

[remote] Pull Request & Merge

1. github.com > 생성한 저장소 > Pull Request 메뉴
2. Pull Request 속 내용 사용해보기
    a. **comment 작성**
    b. **이슈 본인에게 assign**
    c. **reviewer 지정**
3. 팀원들과 코드리뷰 진행

[remote] 코드리뷰가 완료된 branch는 master 브랜치로 merge

1. Merge : 아래 세 가지 방법 중 택 1
    a. Merge pull request = `git merge --squash`
        - 현재 코드를 그냥 머지
    b. Squash and merge = `git merge --squash`
        - 여러 커밋을 하나의 커밋으로 만들고(squash) 머지
    c. Rebase and merge = `git rebase && git merge`
        - 현재 코드를 base branch 기준으로 rebase 후 머지
2. github.com > 생성한 저장소 > Issue 메뉴
3. Close Issue

[local] stash, reset, rebase, cherry-pick 등 사용해보기

1. 로컬 저장소로 복귀
2. 머지가 완료된 feature 브랜치를 로컬 저장소에서 삭제

```
git branch -d feat-{nickname}
```

3. master 브랜치로 이동

```
git checkout master
```

4. 파일 하나를 임의로 수정함 (README.md이든, 다른 파일이든 기존에 있던 파일이면 아무거나 상관 없음)
5. 원격의 최신 코드를 로컬에 반영

```
git fetch origin
### 변경되었다는 로그가 나오겠죠?
git merge
# or git pull 로 한번에 해결 가능!
```

5. 아래와 같은 로그가 뜸

```
Updating 73d9639..9438fd2
error: Your local changes to the following files would be overwritten by merge:
  hobby.md
Please commit your changes or stash them before you merge.
Aborting
```

6. **git stash** 명령어로 작업하던 코드 임시 저장

```
git stash
```

7. 다시 pull 진행

```
git pull
```

8. git stash 창고에서 임시 코드 꺼내오기

```
git stash pop 0
```

9. 충돌이 났으면 해결하고, 그게 아니라면 개발을 마저 진행하면 됨 😂

알면 좋은 유용한 명령어

```
# 원격 저장소로 강제 푸시
# **주의사항** : 남들과 공유하는 master, develop 브랜치는 강제푸시 금지!!!
git push -f origin {branch}

# HEAD기준 n개의 커밋 interactive rebase
git rebase -i HEAD~n

# 특정 커밋을 현재 브랜치에 추가
git cherry-pick {commit id}
```

```
# working space의 변경사항을 버리고 특정 커밋 기준으로 저장소의 HEAD 변경
git reset --hard origin/master
# or
git reset --hard {commit id}
```

추천 사이트 & 정보

git 자습

- https://backlog.com/git-tutorial/kr/
- https://git-scm.com/book/ko/v2

git 브랜칭 연습

- https://learngitbranching.js.org/?locale=ko

git 브랜칭 전략

- https://www.atlassian.com/git/tutorials/comparing-workflows

커밋 메시지 컨벤션 설정

- https://overcome-the-limits.tistory.com/entry/협업-협업을-위한-기본적인-git-커밋컨벤션-설정하기
- 커밋 템플릿 만들기
  - `.gitcommit-template.txt`
  - `git config --global commit.template <경로>`

```
# <타입> : <제목> 형식으로 작성하며 제목은 최대 50글자 정도로만 입력
# 제목을 아랫줄에 작성, 제목 끝에 마침표 금지, 무엇을 했는지 명확하게 작성

################
# 본문(추가 설명)을 아랫줄에 작성

################
# 꼬릿말(footer)을 아랫줄에 작성 (관련된 이슈 번호 등 추가)

################
# feature : 새로운 기능 추가
# fix : 버그 수정
# docs : 문서 수정
# test : 테스트 코드 추가
# refactor : 코드 리팩토링
# style : 코드 의미에 영향을 주지 않는 변경사항
# chore : 빌드 부분 혹은 패키지 매니저 수정사항
################`
```