K-Digital Github 특강

개발자

2020.12.22

현대카드의 초기 digitalization의 실패?

GitHub: what? why? how?

→How가 가장 중요. 면접에서도 많이 물어보고 대답하기 어려움(왜 이걸로 대체했느냐? 등)

Hub: 집합, 모이는곳

SCM: SourceCodeManager

VCS: VersionControlSystem

Git: 버전 관리를 해주는 곳, 실제로 우리 dna에 새겨져있음 (e.g. 최종문서, 최최종)

→사진 혹은 문서가 모여있는 스마트한 폴더

Git =/= github

Github = cloud다

민주당 정치인 Andrew Yang : tech check를 주장

→4차산업혁명 - 어떤 산업이던간에 기술에 집중하는 추세

인공지능 제품 - AI 스피커

echo dot by Amazon(Alexa)이 다른 ai 스피커와 다른 점: 실제로 구매하게 해줌

Google I/O: AI makes an appointment by call (introduced first in 2018)

왜 구매자가 굳이 google ai를 사용하여 예약할까? -> 인터넷 예약이 없는 가게가 많음

→ We have 카카오 예약, 네이버 예약

2019 google I/O: 음성조작에 중심을 둔 AI가 발전

2020 google: 글자를 AI가 읽어줌

결론: 왜 AI 스피커에 투자했는가?

인터페이스의 변화

interface: e.g. UI

사용자와 기계의 접점이 변화함 (손->음성)

Euphonia in 21st century: first AI?

teachable machine: online training machine learning

기술에 대한 관심과 적용에 대한 고민!!

기술 생산 == 기술 활용

즉 생산하는 사람과 활용하는 사람과 같았음

지금은 오픈 소스…

**Git 강의**

Terminal 또는 CLI: C

CLI <-> GU Interface

초기 CLI에서 GUI 발전 – 안전, 편리를 위해

**명령어 1**

mkdir \_\_\_\_ : 폴더 만들기

touch \_\_\_\_ : 파일 생성

rm -r \_\_\_\_ : 삭제

rm -rf (recursive force) \_\_\_\_ : 강제로 지우기 (master도 지움)

ls : 현재 위치의 파일 보여주기

ls -a (list all) : 숨겨진 파일 표시

cd : change directory

cd .. : 상위 폴더로 가기

cd \_\_\_ : 해당 폴더로 가기

mv (기존 파일) (변경파일) : 파일 바꾸기

^c (컨트롤c): 취소하기

~ : 홈폴더, 터미널 베이스 (user)

user@DESKTOP-S4O7RI7 MINGW64 ~

:C/user 에서 시작

user@DESKTOP-S4O7RI7 MINGW64 ~/CLI

:C/user/CLI 에서 시작

마이크로소프트는 다른 os와 다른 독자적임

mac와 linux는 unix 체제를 쭉 사용

예: dir vs. ls

\* le / folder에 띄어쓰기 금지 + 한글 이름 금지 → 오류 가능성

\* ‘tab’을 사용해 자동 완성

**명령어 2**

git init : 리포 (repository)

git status : 현재 git 상태

git add \_\_\_ : 파일을 추적 가능한 상태로 만듬

git commit : identity 계정 설정

:q : 나가기

:q! : 강제로 나가기

git commit -m(message) ‘\_\_\_\_\_/내맘대로 메시지’

git commit –amend : commend message 수정

git log

git log – 2 : 두줄만

git log --pretty=oneline

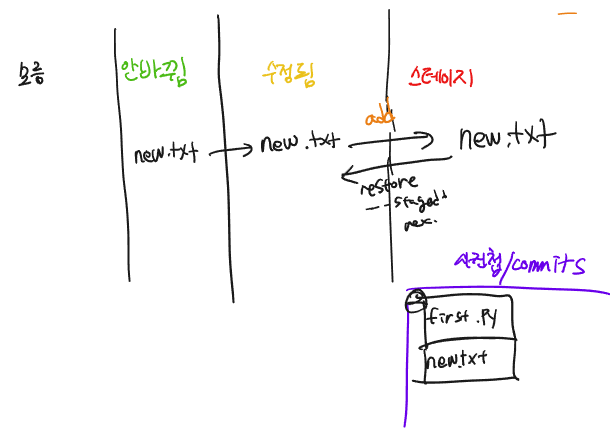
git restore :

rm -rf .git : git init 삭제



1. commit은 stage에 있는 내용을 찍는다.
2. stage에 올리는 명령어는 “$ git add” 다.
3. stage 단계까지 가기위해 add로 stage로 밀어넣고 commit으로 commit.
   1. $ git add new
   2. $ git status
   3. $ git commit -m 'new.txt 수정'

\*git restore --staged new.txt : stage에서 내려옴



\* stage에 올리기 위해 add를 하는 것!

버전을 업데이트 할 때 효율적으로 하는 방법은?

→ 기준을 정해놓기, 옆에 메시지 적기

보고서\_v1.1.1 <= 오탈자 수정

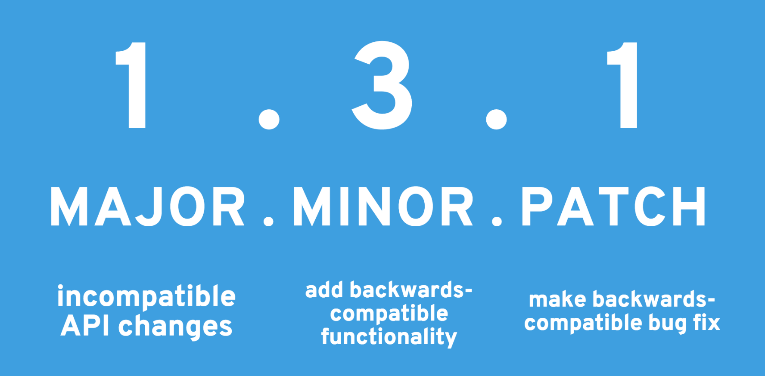
보고서\_v1.1.2 <= 오탈자 수정

보고서\_v1.2.1 <= 슬라이드 수정

보고서\_v.2.0.0 <= 큰 변화

\*메시지에도 스타일 가이드가 있다

- 유의적 버전(semver)을 참고.



버전이 급격히 올라갔을 때 용량 문제를 해결하기 위한 방법?

→ 변경사항만 저장한다.

untracked와 modified 파일이 동시에 존재할 때?

→ git add new.txt diff.txt

→ git add -A

→ git add . (‘.’은 전체를 의미)

**readme (md: markdown)**

마크다운 작성법: <https://gist.github.com/ihoneymon/652be052a0727ad59601> 참고

**12.23**

저장관리를 하고 있지만 클라우드에 업로드 하지 않았음

로컬리포: 내컴퓨터

리모트리포: 온라인 클라우드

따라서 리모트리포 세팅이 필요함

리모트저장소: github, gitlab(주로 사내), bitbucket 등 다양함

origin, 추가리포, 백업리포 등 여러곳에 저장할 수 있음

**명령어 4**

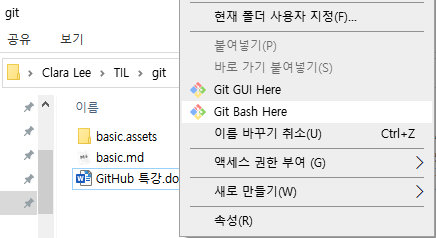
$ git remote add \_\_(리포이름)\_\_ <https://github.com/bibibaba1/TIL> : 리모트저장소로 추가

$ git remote -v :

$ git remote rm \_\_(리포이름)\_\_

$ git push \_\_(리포이름)\_\_ master

$ vim : 터미널에서 쓰이는 문서 편집기



**vim 명령어**

i : insert mode

# 편집 시작

dd : 라인 삭제

:w : 저장

esc : 편집 종료

:wq : 저장후 종료

\* <https://missing.csail.mit.edu/> : git, CLI 등 강의

Branch 에 대해

 이게 뭘까?

**명령어 5**

$ git branch \_\_(이름)\_\_\_ : branch 생성

$ git branch : branch 확인 (초록색 글씨가 현재 head 위치)

$ git switch \_\_(이름)\_\_ : branch 이동

$ git checkout \_\_(이름)\_\_ : 옛날 버전

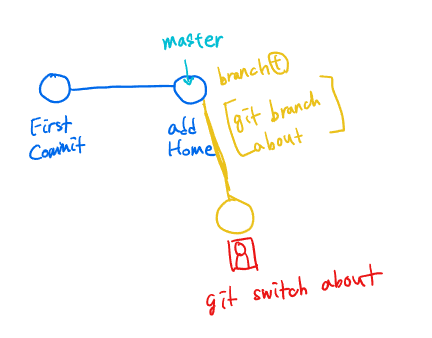
<Branch 이동 과정>

git commit -m 'add home'

git branch about # about 브랜치 생성

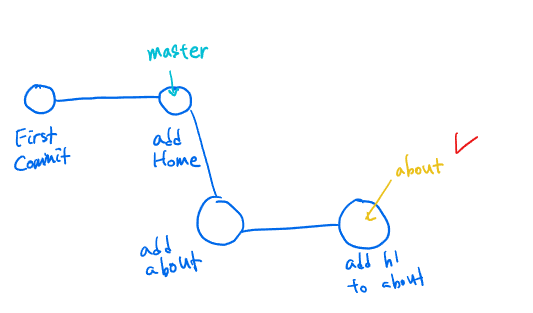
git switch about # about 브랜치로 이동

git branch # 현재 브랜치 현황 확인

****

**이후 commit은 필수!**

**<about page 생성 과정>**

****

$ touch pages/about.md

$ git add .

$ git commit -m 'add about page'

$ touch pages/about.md

$ git add .

$ git commit -m 'add about page'

$ about.md 수정

$ git add .

$ git commit -m 'add h1 to about'

**<Merge 하기>**

* master를 완성본 또는 주체라고 간주, 따라서 그 밑의 branch를 master에 통합(merge)하여야 함

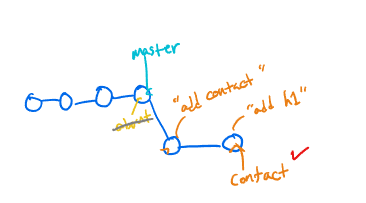
$ git switch master (home으로 이동)

$ git merge about

git branch -d \_\_\_\_\_ : branch 삭제

git checkout -b contact : git branch 만들기 + 이동

git switch -c \_\_\_\_\_



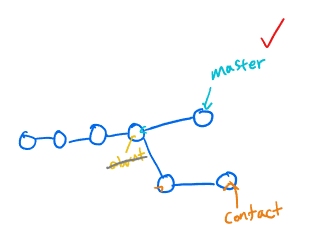
\*수정 후 상위에서 저장하기!

1. 마스터로 이동

2. about.md 수정

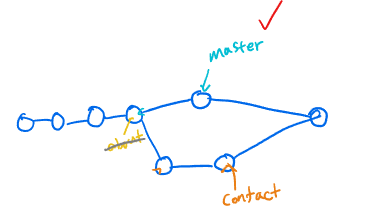
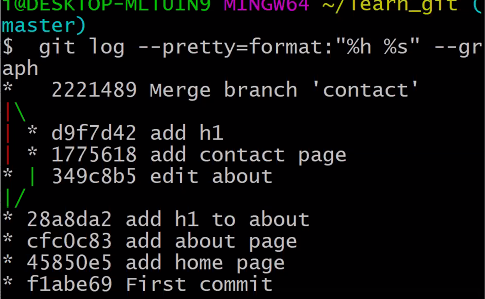
3. ad && commit

<현재 branch 상태>

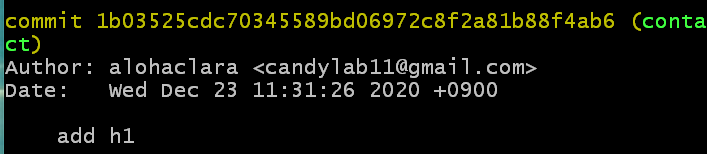


$ git merge contact

vim이 나오고 :wq 로 나가기

merge와 commit이 동시에 완료됨



**<Group Work : 끝말잇기>**

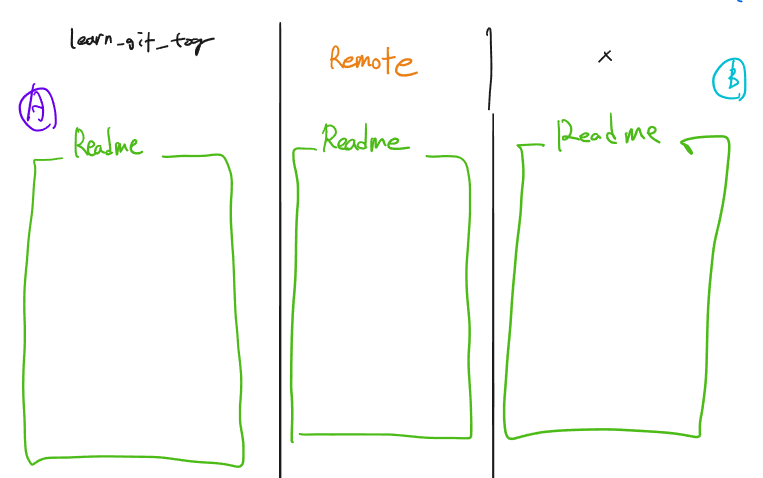
그룹 A:

1. home(/TIL)에서 learn\_git\_together directory 생성
2. READ.md 생성, $ git init && touch README.md
3. github에 리모트리포 생성 및 연결

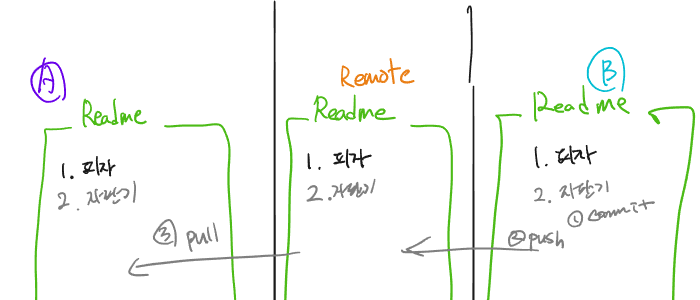
그룹 B:

1. 리포 받기 (pull)
   1. $ git clone \_\_(주소)\_\_

<이때까지 상태>



$ git pull origin master : 주체 -> remote까지 가져갔다면 pull에서 remote에서 끌어오기

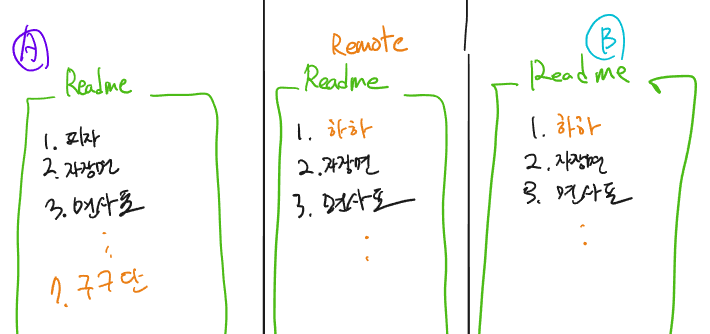


<A와 B끼리 push pull 과정>

가 의미하는 것은?

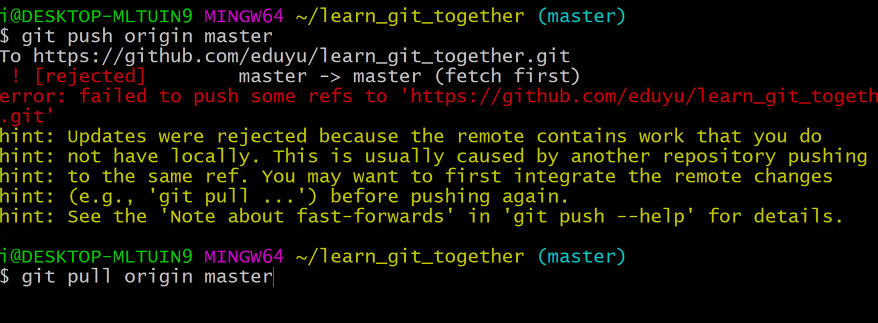
<상황 1>

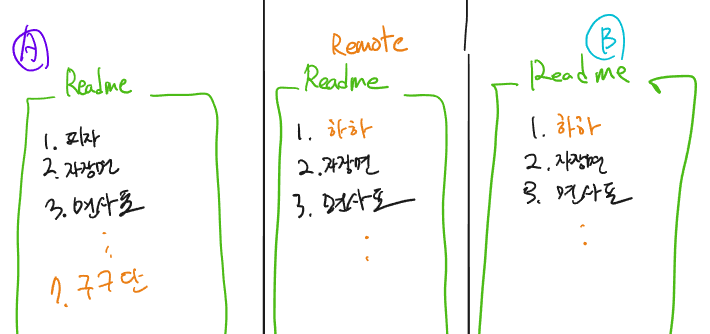
B에서 수정하고 push. 동시에 A가 다른 부분을 수정하고 push 함.



A는 무슨 화면을 보게 될까?

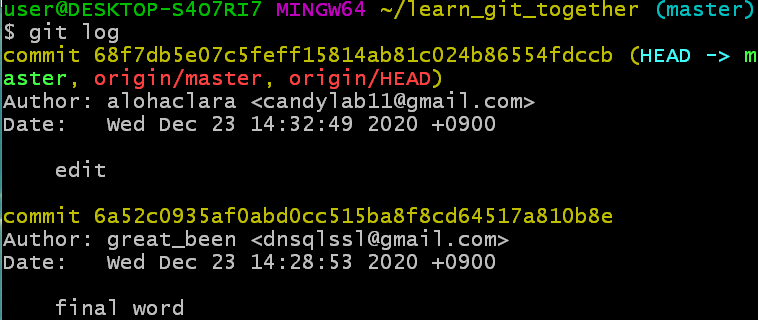
🡪 Error: pull을 먼저 하라고 요청



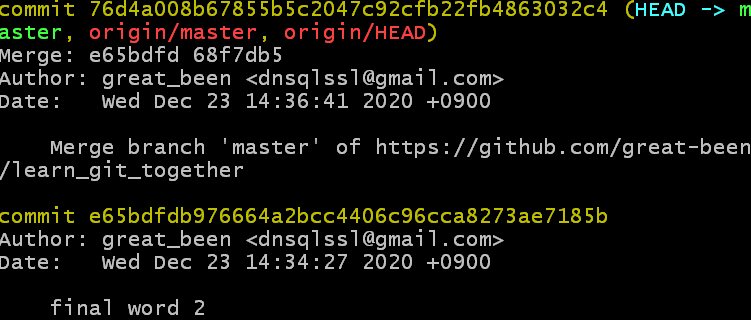


이때 origin/master 는 log의 현재 위치를 말한다.

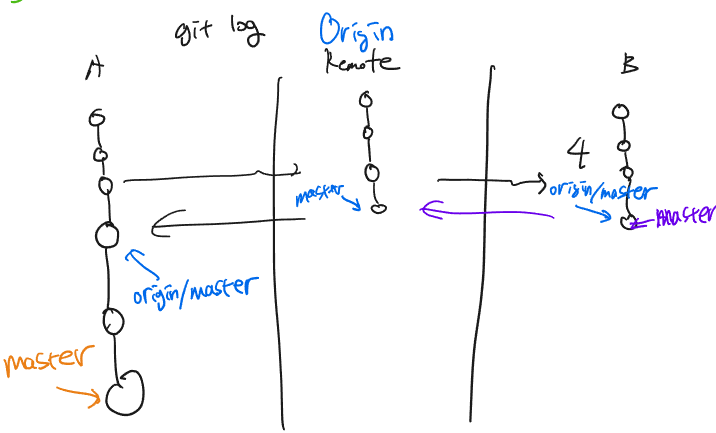
1. A가 1번을 수정했을 때 origin/master 위치:



1. B가 수정해서 push, A가 에러 메시지를 보고 pull, 재수정, push를 함. 이후 B에서 보이는 origin/master:



….일까?

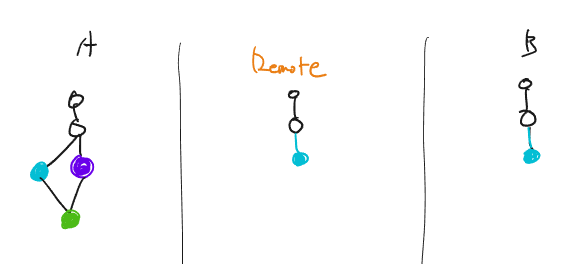


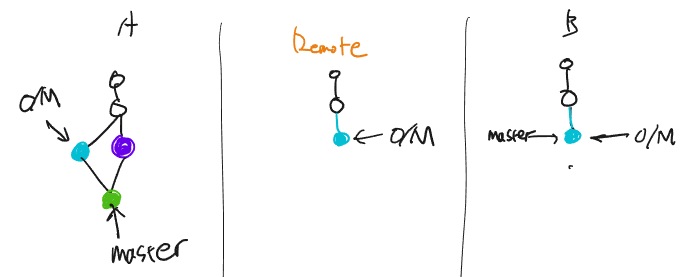
1. 2

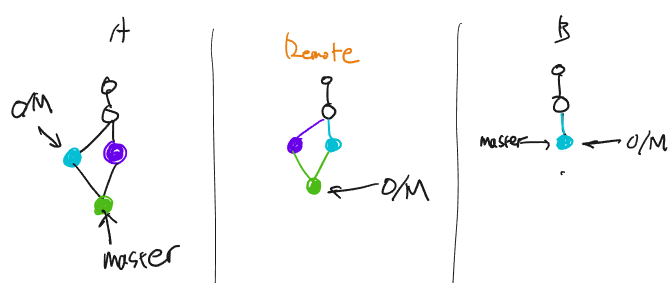
시점

master가 이동하면서 origin의 수정된 시점을 알려주는 것…?

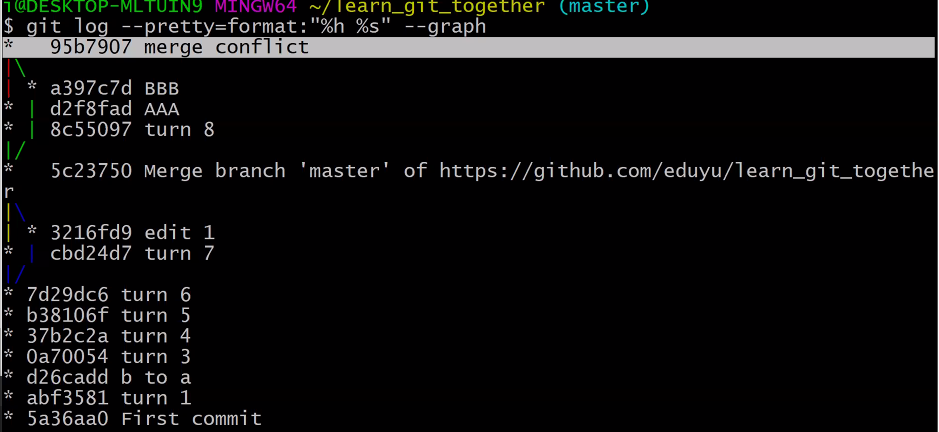
어렵다







$ git log --pretty=format:"%h %s" --graph 로 branch graph 확인하기

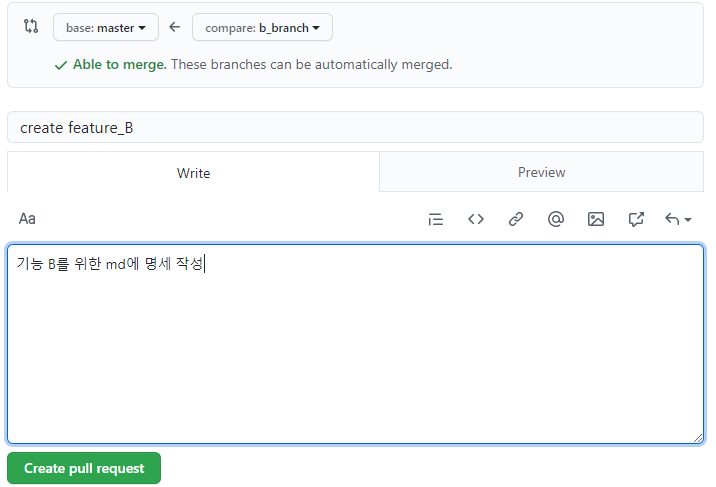


<A와 B 각자만의 branch를 만들어 작업하기>

1. 브랜치 생성, 작업, 저장
2. remote로 푸쉬 하기
   1. git push : error - fatal: The current branch b\_branch has no upstream branch. To push the current branch and set the remote as upstream, use
   2. git push origin b\_branch
3. bithub에 들어가서 리포 확인
   1. ‘compare & pull request’가 나타남

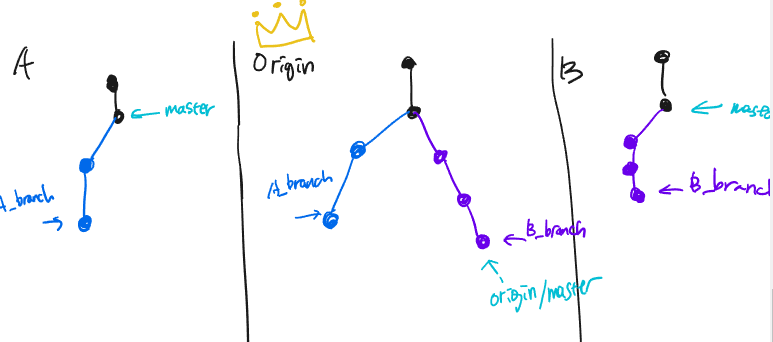


1. 클릭해서 base: master에 병합하길 요청 (마치 결재서류 올리듯)



🡪 즉 이 과정은 remote에서 결정이 나는 것임. (협업!)

1. pull request confirm 하여 merge를 확정함



이때 주의할점! b\_branch가 바뀐 내용이 더 많음. origin이 최종 내용이고 A에서 pull해올 때 필시 a\_branch가 아닌 master에서 pull해와야 함!

 에서 

🡪 git pull origin master

<현재branch상태>

