DevKor Back-End Study #1
- OT, git -

스터디장 소개

컴퓨터학과 20학번 노정훈

경력

- 컴퓨터학과 알고리즘 학회 ALPS 부회장(2021)
- DevKor 운영진 (2022.09~)
- DevKor 주최 2022 KU SUMMER HACKATHON 플랫폼 서버 개발

기술 스택

- React, JavaScript, TypeScript, C, Express, PostgreSQL, MongoDB

github: https://github.com/overthestream

스터디원 자기 소개 시간

스터디 진행 방식

1. 스터디장의 강의 형식

- 최소한 1주일 전에 강의 자료 ppt를 완성해 공유드릴 예정
- 건의 사항 or 돌아가면서 진행하는 세미나 형식이 괜찮다면 말씀해주세요!

2. 실습와 과제

- 노션에 기재한 커리큘럼에 따라 진도를 나갈 예정
- 매 시간마다 실습, 과제를 통해 실제로 작동할 수 있는 express web server를 만들고 배포까지 진행할 예정!

DevKor 규정 상 결석 or 과제 미제출 4진 out

커리큘럼 : https://devkor.notion.site/Backend-aff5617da10a448a95b810e259a7cca6

VCS

개요

- GIT은 VCS의 일종
- VCS = Version Control System
- 소프트웨어나 문서 등은 여러 번 수정사항을 거치며 개발되기에 필요

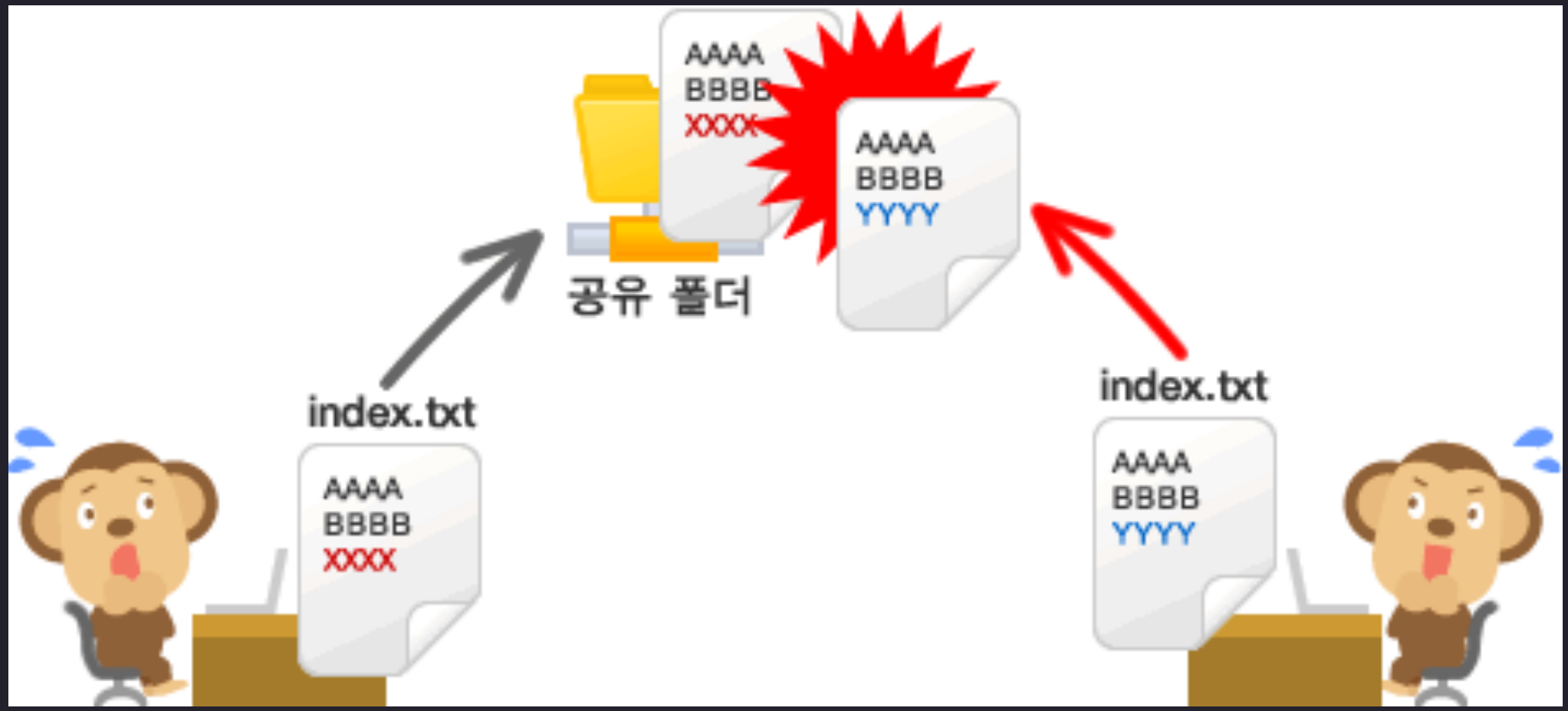
사용 시 이점

- 소프트웨어 (code)의 수정사항, 변경사항을 관리하는 것은 당연히 중요
- 어떤 내용을, 어떤 시점에, 누가 바꿨는지 알 수 있다.
- 버그, 에러, 데이터가 날아간 경우 등 문제 발생 시 과거 버전으로 돌릴 수 있음
- branch 기능을 통한 협업, 여러 버전 개발 가능 (develop/release)

git, subversion 등 여러 툴이 존재하며 그 중 git의 점유율이 최고

Name

- 120525_문서_업데이트.txt
- 120604_문서.txt
- 120605_문서_수정판.txt
- 120605_문서_수정판2.txt
- 120605_문서_최신 복사.txt
- 120605_문서_최신.txt
- 120605_문서.txt
- 1200602_문서.txt
- 문서_회의용.txt


AAAA
BBBB
XXXX
AAAA
BBBB
YYYY
공유 폴더
index.txt
AAAA
BBBB
XXXX
index.txt
AAAA
BBBB
YYYY

GIT

주요 기능

- commit
- push
- branch
- merge
- pull
- clone

- tag
- release
- stash
- rebase

…

GIT: commit

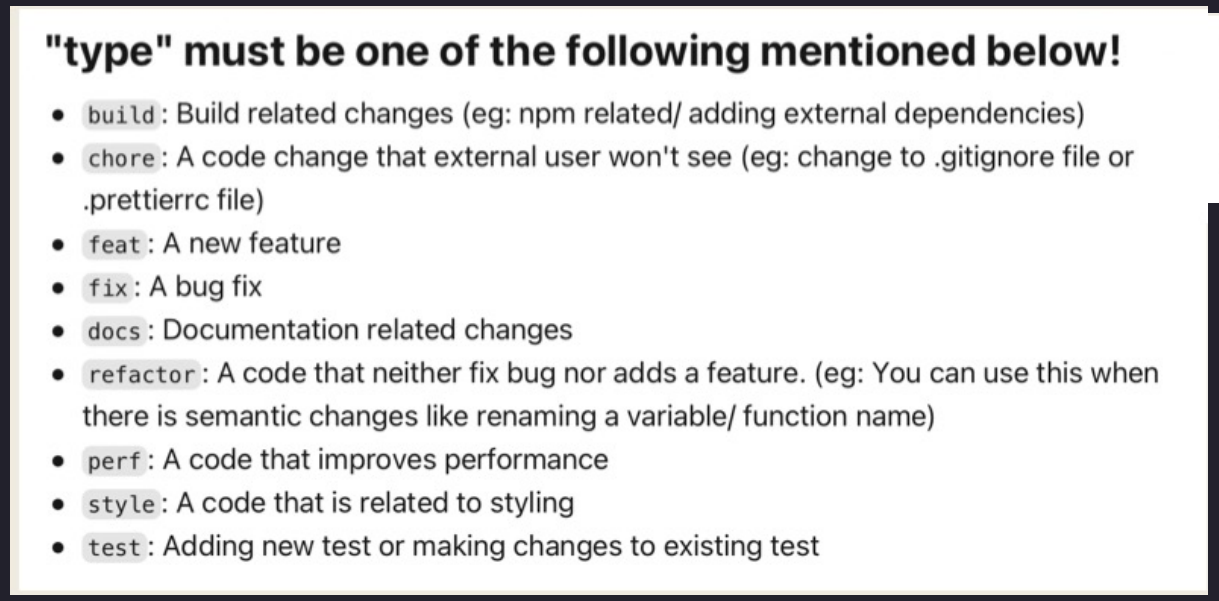
- 코드가 변경된 기록의 최소 단위 (atomic, 원자적 단위)
- 커밋 하나하나가 코드의 수정 기록이며, 하나의 버전이라고 할 수 있음
- 커밋마다 id, 메세지를 붙일 수 있다
- 한 커밋에 여러 파일의 변경 기록이 포함될 수 있으며, 각 파일에서 어떤 부분이 삭제되고, 추가되고, 수정되었는지 포함
- 커밋이 쌓여서 하나의 repository (코드 폴더?) 를 구성

```
feat(project): controller & routes

feat(project): test code for project/temp services

feat(project): add service

feat(project): add likes in model
```

GIT: commit

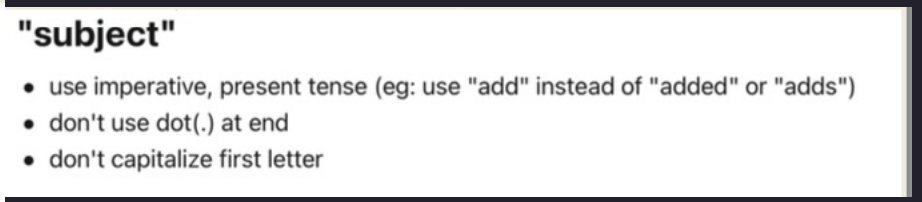
commit message

- 각각의 커밋 (변경 사항)이 어떤 내용을 포함하고 있는지 알려주는 설명문
- 하나의 task에 대하여 코딩할 때, 한 번에 커밋하는 것보다, 각각의 수정 단위(ex: 기능 추가, model 변경 등)마다 커밋을 하는 것이 에러 대응 등 vcs의 이점을 최대화할 수 있는 방안



"type" must be one of the following mentioned below!

- `build`: Build related changes (eg: npm related/ adding external dependencies)
- `chore`: A code change that external user won't see (eg: change to .gitignore file or .prettierrc file)
- `feat`: A new feature
- `fix`: A bug fix
- `docs`: Documentation related changes
- `refactor`: A code that neither fix bug nor adds a feature. (eg: You can use this when there is semantic changes like renaming a variable/ function name)
- `perf`: A code that improves performance
- `style`: A code that is related to styling
- `test`: Adding new test or making changes to existing test



"subject"

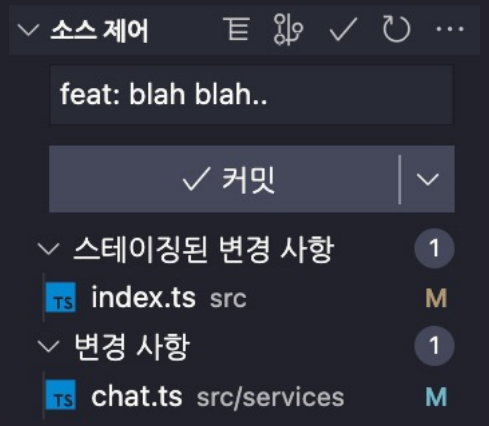
- use imperative, present tense (eg: use "add" instead of "added" or "adds")
- don't use dot(.) at end
- don't capitalize first letter

커밋 메세지를 쓰는 관례 commit message convention
각 수정 사항이 어떤 내용을 포함하는지 잘 알려주면 좋다.

GIT: commit

How Works?

로컬 저장소 == git이 설치된 본인의 pc.
로컬 저장소의 구성



- 우측 사진은 vscode text editor의 built-in git gui
- 변경 사항 : 현재 working directory에서 추적되는 이전 커밋 기준 변경 사항
- 스테이징: ‘git add’ 명령어를 통해 변경 사항을 index에 스테이징. message를 적은 후, 커밋 하면 local 저장소의 HEAD에 저장됨

GIT: commit

How Works?

로컬 저장소의 git log를 ‘GitHub’이라는 원격 저장소에 올리는 것

GitHub은 원격 저장소로서 여러 로컬 저장소의 git 파일들을 저장하며, 각 로컬 저장소가 내려받고, 수정하여 commit한 것을 업로드할 수 있도록 도와줌.

이를 통해 여러 로컬 저장소에 각각 복사본이 생기고,
각각의 로컬 저장소의 커밋들을 branch기능을 통해 나누고 merge하는 등 협업이 가능

더욱 자세한 내용은 후술

GIT: push

로컬 저장소의 HEAD에 존재하는 커밋들을, 원격 저장소에 올리는 기능

원격 저장소 더하기

```
$ git remote add origin <주소>
      : Ex) https://github.com/overthestream/devkor-repository
```

:origin이라는 이름으로, <주소>에 있는 remote repository를 추가

원격 저장소에 push

```
$ git push origin <branch>
$ git push origin main
```

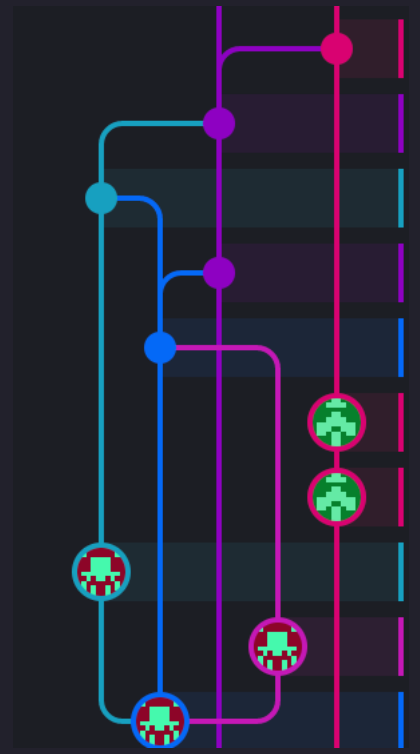
해당 사항들은 실습해볼 예정~!

GIT: branch

branch: 가지치기
커밋들을 가지치는 것

여러 버전을 개발하기 위하여 원래의 줄기에서 가지를 치고,
다시 병합하여 하나의 소프트웨어를 개발하는 것

GitHub 의 default 줄기 branch 이름: main
: 보통 $ git push origin main 으로 push하는 이유



가지치기와 병합
branch & merge

GIT: branch

branch기능을 통하여 협업이 가능

branch 미사용 시 :

[REMOTE] : Github main branch version
1 -> 2 -> 3

[LOCAL] for user A
1 -> 2 -> 3 -> a

push -> remote main

[REMOTE]
1 -> 2 -> 3 -> a와 b의 충돌

[LOCAL] for user B
1 -> 2 -> 3 -> b

GIT: branch

branch를 사용한 협업

[REMOTE] : Github main branch version : main branch
1 -> 2 -> 3

[LOCAL] for user A
: feat/a branch
1 -> 2 -> 3 -> a

push -> remote feat/a

[REMOTE]
main: 1 -> 2 -> 3
feat/a: 1 -> 2 -> 3 -> a

merge feat/a into main
main: 1 -> 2 -> 3 -> a

[LOCAL] for user B
: feat/b branch
1 -> 2 -> 3 -> b

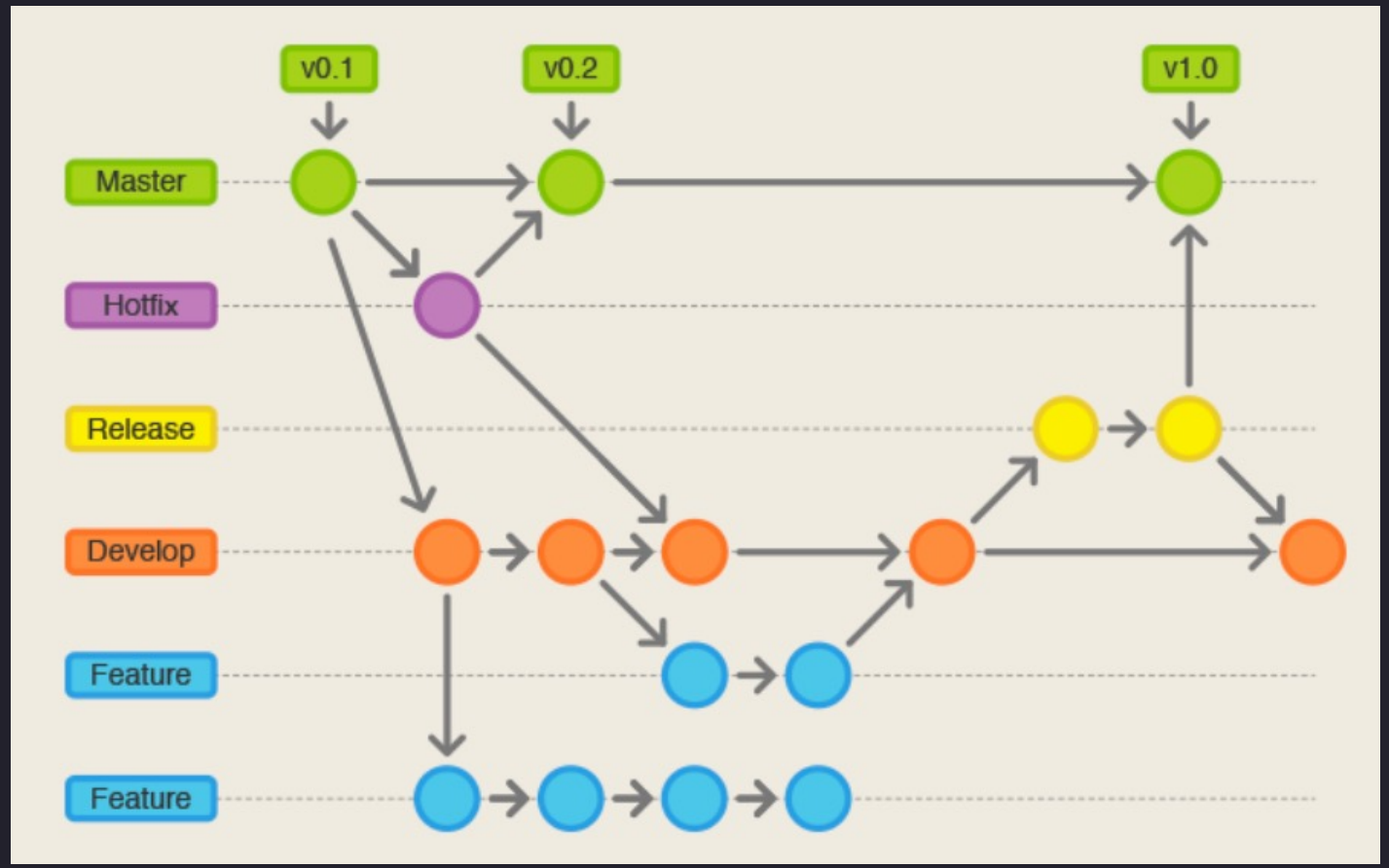
push -> remote feat/b

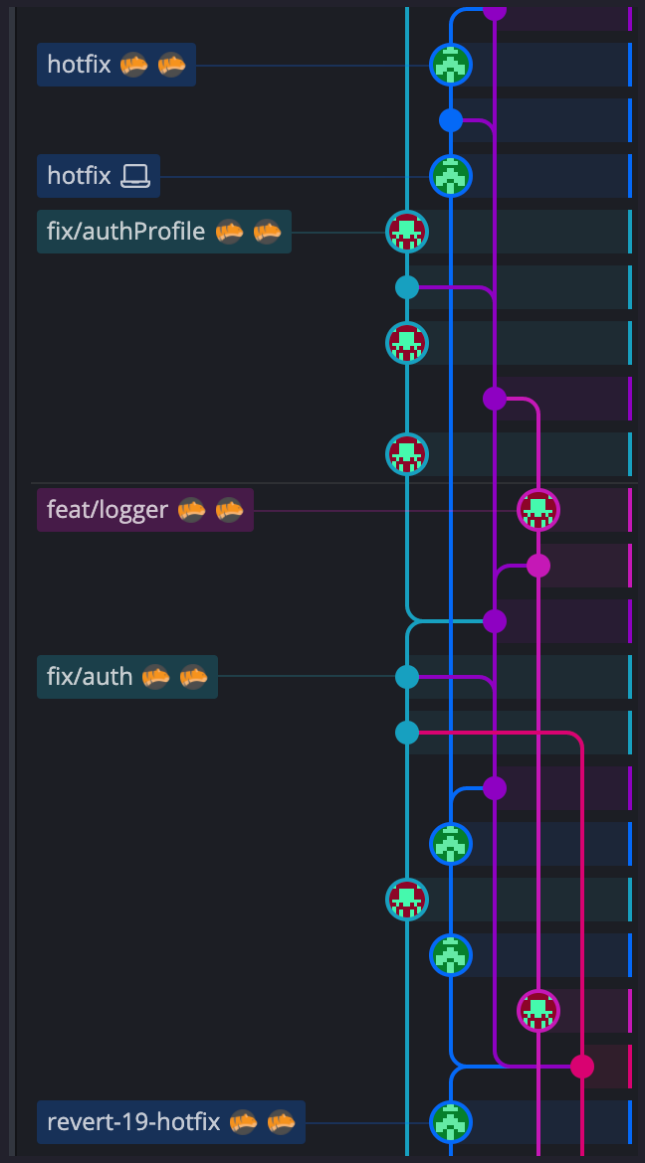
[REMOTE]
main: 1 -> 2 -> 3 -> a
feat/b: 1 -> 2 -> 3 -> b

merge feat/b into main
충돌이 발생하지만 병합할 수 있음!
main: 1 -> 2 -> 3 -> a -> b (rebase)
main: 1 -> 2 -> 3 -> 4(a+b) (merge)

GIT: branch

branch를 사용한 여러 버전 관리





GIT: branch

branch 사용하기

```
$ git branch -m <branch name>
$ git checkout <branch name>
```

GIT: merge

main 이나 develop, release 등의 branch에 branch를 병합하는 과정

어떤 파일을 수정했냐에 따라 충돌(conflict)이 발생할 수도, 아닐 수도 있음

충돌이 발생한다면 사용자가 직접 어떤 변경사항을 수용하고 하지 않을 것인지, 둘 다 수용할 것인지를 결정해야 함

branch의 merge

1. GitHub을 통한 merge:
각자가 기능을 개발한 후 main, develop 등의 branch에 merge 시 많이 사용 (pr이라고도 함)
2. local merge:
다른 브랜치에서 작업하다가, main 브랜치가 수정되었을 때 main branch를 pull하는 과정에서 사용

GIT: pull

로컬 저장소에 원격 저장소의 파일들을 받아오는 과정

```
$ git pull origin <branch name>
$ git pull origin // default: 현재의 branch
```

GIT: clone

로컬 저장소에 원격 저장소의 파일들을 받아오는 과정

```
$ git clone <원격 저장소>
```

pull과의 차이점

- pull은 이미 로컬 저장소가 있을 때 원격 저장소에 새로 생긴 변경사항을 받아오는 개념

- clone은 아무것도 없을 때 원격 저장소의 내용들을 그대로 새로운 working directory에 받아오는 과정

GIT: clone

로컬 저장소에 원격 저장소의 파일들을 받아오는 과정

```
$ git clone <원격 저장소>
```

pull과의 차이점

- pull은 이미 로컬 저장소가 있을 때 원격 저장소에 새로 생긴 변경사항을 받아오는 개념

- clone은 아무것도 없을 때 원격 저장소의 내용들을 그대로 새로운 working directory에 받아오는 과정

GitHub

쉽게 말하면 원격 저장소 서버

원격 저장소를 public / private으로 설정 가능

- 각자 프로필과 개인 repository(:저장소)를 이용한 포트폴리오로서의 기능
- 방대한 public repository를 통한 오픈소스 플랫폼으로서의 기능
  - fork, pull, push, pr을 이용하여 나도 linux, react등의 코드를 수정할 수 있다!
- PR, Merge, issue, code review를 이용한 협업 플랫폼으로서 기능

Git/GitHub 실습

1. Github 가입 : https://github.com
2. git 설치
   - windows: http://git-scm.com/download/win
   - $ git –version
3. git configure
   - $ git config --global user.name "overthestream"
   - $ git config --global user.email njhnjh02@naver.com
4. github repository 만들기
5. 로컬 저장소 만들기:
   - git clone
   - git init / remote add, pull
6. branch
7. readme 변경
8. add, commit, push
9. merge

과제

어떤 서비스의 서버를 만들고 싶은지 생각해 옵시다~

ex) 친구 기능이 있는 todo list, 채팅 서비스, …