Shell, vim command for git

Fastcampus Programming SCHOOL

최우영

- Co-founder, CTO(disceptio)
- Solution Architect, Web Developer, Instructor
- Skills: Python, Golang, Julia, Node.js, Google tag manager ...

Contacts

1. blog: https://ulgoon.github.io/
2. github: https://github.com/ulgoon/
3. email: me@ulgoon.com
4. discord: @ulgoon

Introduce

Goal(1)

- Linux의 역사를 이해한다
- CLI에 대한 공포를 극복하고 Shell과 친구가 된다
- Linux Shell 커맨드를 학습하여 능숙하게 이를 활용할 수 있다
- Vim 텍스트 에디터를 이용하여 커밋메시지를 작성하고, 파일 수정을 할 수 있다

Goal(2)

- git을 이해하고, git과 github이 다름을 인지한다
- git을 활용하여 나의 소스코드를 관리할 수 있다
- 나의 커리어를 Swag 할 수 있는 블로그를 git을 활용하여 관리할 수 있다.
- git의 branch model을 활용해 능숙하게 코드관리할 수 있다
- git으로 타인과 협업하며, 다른 프로젝트에 기여할 수 있다

Contents

- Introduction to Linux, Shell
- Shell commands
- Vim commands

Before Linux(1)

- 1965년 데니스 리치, 켄 톰슨 외 x명이 AT&T Bell 연구소에서 PDP-7 기반 어셈블리어로 작성한 UNIX를 개발

Before Linux(2)

- 1973년 데니스 리치와 켄 톰슨이 C를 개발한 뒤, C 기반 UNIX 재작성

Before Linux(3)

- 1984년 리차드 스톨먼이 오픈 소프트웨어 자유성 확보를 위한 GNU 프로젝트 돌입

Meaning of GNU

GNU == `G` NU is `N` ot `U` nix

Before Linux(4)

- But, GNU 프로젝트에는 커널이 없었고..

Kernel

- 하드웨어와 응용프로그램을 이어주는 운영체제의 핵심 시스템소프트웨어

Linus Torvalds

https://www.youtube.com/embed/IVpOyKCNZYw

- 헬싱키 대학생이던 리누스 토발즈는 앤디 타넨바움의 MINIX를 개조한 Linux를 발표
- 0.1 - bash(GNU Bourne Again SHell), gcc(UNIX 기반 C 컴파일러)

Linux

- 리누스 토발즈가 작성한 커널 혹은 GNU 프로젝트의 라이브러리와 도구가 포함된 운영체제
- PC와 모바일, 서버, 임베디드 시스템 등 다양한 분야에서 활용
- Redhat, Debian, Ubuntu, Android 등 다양한 배포판이 존재

Shell

- 운영체제의 커널과 사용자를 이어주는 소프트웨어
- sh(Bourne Shell): AT&T Bell 연구소의 Steve Bourne이 작성한 유닉스 쉘
- csh: 버클리의 Bill Joy가 작성한 유닉스 쉘(C언어랑 비슷한 모양)
- bash(Bourne Again Shell): Brian Fox가 작성한 유닉스 쉘
    - 다양한 운영체제에서 기본 쉘로 채택
- zsh: Paul Falstad가 작성한 유닉스 쉘
    - sh 확장형 쉘
    - 현재까지 가장 완벽한 쉘

Let's learn bash

Shell Command Basic(1)

```
$ cd documents

$ mkdir dev # - make directory dev
$ cd dev # - change directory
$ cd .. # - go up
$ pwd # - print working directory

$ ls
$ ls -al

$ touch hello.py # - create hello.py
$ exit # - terminate shell
```

chmod

> 파일의 권한을 설정할 때 사용

`drwxr-xr-x`

`d` or `-` : directory or file

(user)(group)(other)

`r` : read

`w` : write

`x` : execute

`-` : no permission

chmod

```
$ chmod [옵션] (8진수) (파일명)
```

8진수

0: 000

1: 001

2: 010

3: 011

4: 100

5: 101

6: 110

7: 111

Shell Command Basic(2)

```
$ mv hello.py dev
$ cp hello.py dev

$ rm hello.py
$ rm -rf dev/

$ cat README.md
$ head README.md
$ tail -20 README.md
$ cat README.md > README.txt
$ open README.md # use explorer instead open on windows
```

Vim

TODO

1. documents/dev에서 fastcampus 디렉토리를 만듭니다
2. 새파일 index.html을 만들어주세요
3. vim으로 내부 작업을 한뒤 저장하여 cat 명령어로 저장을 확인합니다.
4. index.html을 style.css로 이름을 변경합니다.

version 1.1
April 1st, 06
vi / vim 단축키 모음
Esc 명령 모드
~ 대소문자 전환
` 마크로 이동
! 외부 명령
@ 매크로 실행
# 이전 검색
$ 줄끝으로 이동
% 일치하는 괄호찾기
^ 줄의 첫 글자
& :s 반복
* 다음 검색
(문장 시작
) 문장 끝
_ 아래줄로 이동
+ 다음 줄
0 줄의 처음
- 이전 줄
= 자동 들여쓰기
Q 실행 모드
q 매크로 기록
W 다음 WORD
w 다음 단어
E 끝 WORD
e 단어 끝
R 수정 모드
r 한 문자 교체
T 뒤로 검색
t 한 문자 검색
Y 줄단위 복사
y 복사
U 줄 단위 실행취소
u 실행취소
I 줄 시작에서 삽입
i 편집 모드
O 행 위에 삽입
o 행 아래에 삽입
P 커서 이전에 붙여넣기
p 커서 이후에 붙여넣기
{ 문단 시작
[기타
} 문단 끝
] 기타
A 줄 끝에 덧붙이기
a 덧붙이기
S 줄 삭제후 편집모드
s 단어 삭제후 편집모드
D 줄 끝까지 삭제
d 삭제
F 뒤로 검색
f 한 문자 찾기
G 파일끝/줄로 이동
g 확장 명령
H 화면 상단
h
J 줄 합치기
j
K 도움말
k
L 화면 하단
l
: ex 명령줄
; t/T/f/F 명령 반복
" 레지스터 지정
' 마크로 이동
| 열 이동
\ 사용 안함
Z 종료
Z 확장 명령
X 백스페이스
x 글자 삭제
C 줄 끝까지 바꾸기
c 바꾸기
V 줄단위 비주얼모드
v 비주얼 모드
B 이전 WORD
b 이전 단어
N 이전 (찾기)
n 다음 (찾기)
M 화면 가운데
m 마크 설정
< 내어쓰기
, t/T/f/F 역순 검색
> 들여쓰기
. 명령 반복
? 찾기 (뒤로)
/ 찾기
동작 커서를 이동하거나, 연산자가 동작할 범위를 지정합니다.
명령 바로 동작하는 명령, 빨간색은 편집 모드로 변경됩니다.
연산자 이동 관련 문자(숫자나 커서 이동)와 함께 사용하여야 하며, 커서의 위치부터 목적지까지 연산합니다.
확장 특별한 키 함수로, 추가적인 키 입력이 필요합니다.
q 입력후 (숫자를 제외한 .으로 끝날수 있는) 글자를 입력하여야 합니다.
words: 구분자로 공백, 특수기호 모두 사용
WORDs: 구분자로 공백 문자만 사용
words: quux(foo, bar, baz);
WORDs: quux(foo, bar, baz);
주요 명령행 명령 ('ex'):
:w (저장), :q (종료), :q! (저장하지 않고 종료)
:e f (파일 f 열기),
:%s/x/y/g (파일 전체에서 'x' 를 'y' 로 교체),
:h (vim 도움말), :new (새 파일)
그외 중요한 명령들:
CTRL-R: 재실행 (vim),
CTRL-F/-B: 페이지 위로/아래로,
CTRL-E/-Y: 줄 스크롤 위로/아래로,
CTRL-V: 블럭-비주얼 모드 (vim 전용)
비주얼 모드:
커서를 움직여 지정한 범위에 연산자를 적용합니다. (vim 전용)
참고:
(1) 복사/붙여넣기/지우기 명령어를 사용하기 전에 "x를 입력하여 레지스터(클립보드)를 지정하세요. (x는 a에서 z 또는 * 을 사용할 수 있음) (예: "ay$ 를 입력하면 현재 커서에서 라인 끝까지의 내용을 레지스터 'a'에 저장합니다.)
(2) 어떤 명령을 입력하기 전에 횟수를 지정하면, 횟수만큼 반복하게 됩니다.(예: 2p, d2w, 5i, d4j)
(3) 연속으로 입력하는 명령은 현재의 라인에 반영됩니다. 예시: dd(현재 라인 지우기), >>(들여쓰기)
(4) ZZ 는 저장후 종료, ZQ는 저장하지 않고 종료.
(5) zt : 커서가 위치한 곳을 제일위로 올리기, zb : 바닥으로, zz : 가운데로
(6) gg : 파일의 처음으로(Vim 전용), gf : 커서가 위치한 곳의 파일 열기(Vim 전용)
vi/vim 에 대한 더 많은 강좌나 팁을 얻으려면 www.viemu.com (ViEmu, MS 비주얼 스튜디오를 위한 vi/vim 에뮬레이션)을 방문하십시오.

Vim Basic

`h,j,k,l - move cursor`

`i - insert mode`

`v - visual mode`

`d - delete`

`y - yank`

`p - paste`

`u - undo`

`r - replace`

`$ - move end of line`

`^ - move start of line`

`:q - quit`

`:q! - quit w/o write(no warning)`

`:wq - write and quit`

`:{number} - move to {number}th line`

write `hello.py` with Vim

```
$ vim
```

```
$ vim hello.py
```

```
i
```

```
-- insert --
```

type `print("hello python!")`

press `esc` to escape

```
:wq
```

```
$ python hello.py
```

copy & paste

`$ vim hello.py`

`v`

`-- visual --`

블록지정 후 `y`

`p`

press `esc` to escape

`:wq`

`$ python hello.py`

Use Vim in real world!

- vim advanture
- vimium
- vs code vim

git (1)

Fastcampus Programming SCHOOL

git

VCS (Version Control System)

== SCM (Source Code Management)

< SCM (Software Configuration Management: 형상관리)

chronicle of git

chronicle of git

- Linux Kernal을 만들기 위해 Subversion을 쓰다 화가 난 리누스 토발즈는 2주만에 git 이라는 버전관리 시스템을 만듦
  git official repo

Characteristics of git

- 빠른속도, 단순한 구조
- 분산형 저장소 지원
- 비선형적 개발(수천개의 브랜치) 가능

Pros of git

- **중간-발표자료_최종_진짜최종_15-4(교수님이 맘에들어함)_언제까지??_이걸로갑시다.ppt**
- 소스코드 주고받기 없이 동시작업이 가능해져 생산성이 증가
- 수정내용은 **commit** 단위로 관리, 배포 뿐 아니라 원하는 시점으로 **Checkout** 가능
- 새로운 기능 추가는 **Branch**로 개발하여 편안한 실험이 가능하며, 성공적으로 개발이 완료되면 **Merge**하여 반영
- 인터넷이 연결되지 않아도 개발할 수 있음

git GUI Clients

- git GUI
- sourcetree
- kraken
- smartGit

CLI first

- Source code를 Cloud Platform에서 사용할 경우, CLI 커맨드로 버전관리를 수행해야 합니다.
- CLI 커맨드로 git을 사용할 줄 알면, GUI 도구가 제공하는 기능에 대한 이해가 빠릅니다.
- 확인용도로 GUI를 참고하는 것은 Good^^

git objects

- Blob: 파일 하나의 내용에 대한 정보
- Tree: Blob이나 subtree의 메타데이터(디렉토리 위치, 속성, 이름 등)
- Commit: 커밋 순간의 스냅샷

git Process and Command

Useful package manager for mac

http://brew.sh/index_ko.html

install git

- for windows
  https://gitforwindows.org/
- for MacOS
  `$ brew install git`
- for Ubuntu
  `$ sudo apt-get install git`

`$ git --version` 으로 정상적으로 설치되었는지를 확인

Optional

- github CLI
  https://cli.github.com/

git is not equal to github

sign up github

https://github.com/

important!!

- 가입할 `email` 과 `username` 은 멋지게
- ~~private repo를 원한다면 $7/month~~

Useful blog post

https://ulgoon.github.io/2019/09/09/01-git-first/

Important github User Interface

Star

watch

Set configuration

terminal

```
$ git config --global user.name "{github username}"
$ git config --global user.email "{github email address}"
$ git config --global core.editor "vim"
$ git config --global core.pager "cat"
$ git config --list
```

- optional

  `$ git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"`

My First Repo

Let's make your first repo with github

My First Repo(1)

Let's Create New repo

My First Repo(2)

```
$ mkdir first-repo && cd first-repo
$ git init
$ git remote add origin https://github.com/{username}/{reponame}.git
$ touch README.md
$ git add README.md
$ git commit -m "docs: Create README.md"
$ git push -u origin master
```

> Caution: Do not `git init` on any other directories

Second push to My First Repo

```
# make some change on README.md
$ git add .
$ git commit
$ git push origin master
```

Commit Convention

- 커밋 제목은 50자 이내로 요약하여 작성한다
- 제목과 내용사이 한 칸
- prefix를 사용하여 한 눈에 커밋의 용도를 알기 쉽게 한다

```
feat: features
docs: documentations
conf: configurations
test: test
fix: bug-fix
refactor: refactoring
ci: Continuous Integration
build: Build
perf: Performance
```

Commit Convention - example

```
feat: Create server.py to start flask project
docs: Create README.md
conf: poetry init
test: User model CRUD test complete
```

Start Project with `git clone`

start project with clone

- github에서 repo를 생성합니다.

```
$ git clone {repo address}
$ git add .
$ git commit
$ git push
```

My First Github Pages

github 저장소를 활용해 정적인 사이트 호스팅이 가능

`username`.github.io
http://tech.kakao.com/
https://spoqa.github.io/

sample index page

After create new repo throuch github,

```
$ git clone https://github.com/username/username.github.io.git
```

Create New file `index.html`

```
$ git add .
$ git commit -m "first page"
$ git push origin master
```

sample index page

```html
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>My first gh page</title>
 </head>
 <body>
  <h1>Home</h1>
  <p>Hello, there!</p>
 </body>
</html>
```

Static Site Generator

- Jekyll: Ruby 기반 정적인 블로그 생성기
  - 설치와 사용이 쉬움
  - 사용자가 많았음
- Hugo: Golang 기반 정적인 블로그 생성기
  - 빠른 속도로 사이트를 생성
  - 사용자 증가 중
- Hexo: Node.js 기반 정적인 블로그 생성기
  - Node.js를 안다면 커스터마이즈가 쉬움
  - 빠른 속도로 사용자 증가 중

Recommand

`Hexo` > `Jekyll` > `Hugo`

Let's use Hexo

Requirements

1. git
2. node.js(https://nodejs.org/en/)

```
$ npm install -g hexo-cli
```

Init hexo project

```
$ hexo init <folder>
$ cd <folder>
$ npm install
```

clean && generate static files

```
$ hexo clean && hexo generate
```

Run hexo server

```
$ hexo server
```

deploy

```
$ npm install hexo-deployer-git --save
```

```
deploy:
  type: git
  repo: <repository url>  branch: [branch] #published
  message:
```

.gitignore and .gitattributes

.gitignore: 특정파일 추적을 하고 싶지 않을 경우

```
*.java
*.py[cod]
```

.gitattributes: 파일단위, 디렉토리 별 다른 설정을 부여하고 싶을 경우

```
# Avoid conflicts in pbxproj files
*.pbxproj binary merge=union

# Always diff strings files as text
*.strings text diff
```

- reference: https://thoughtbot.com/blog/xcode-and-git-bridging-the-gap

git (2)

Fastcampus Programming SCHOOL

main

develop

단장님 repo

main

develop

조하경

feature

develop

Branch

What is branch?

What is branch?(1)

What is branch?(2)

분기점을 생성하고 독립적으로 코드를 변경할 수 있도록 도와주는 모델

ex)

master branch

```
print('hello world!')
```

another branch

```
for i in range(1,10):
    print('hello world for the %s times!' % i)
```

branch

m main

i iterate

create hello.py

add print function

Add iteration to repeat print() for 5 times

if-init

Merge

recreate function

merge

if

else

stash 기능? 갑안보여... 뽑내보여... 안정... 임시로 숨겨둔 임시 내용. if-init 가는 merge

if you try to checkout to another branch, it will abort since local change (commit) would be overwritten

staged but not commited.

if-init을 작성했음. vi hello.py 이런 구구 쓰는 수동 명령어들 자동화하자.

merge commit

1. 수정 + 저장 하고
2. git status 확인 - add, commit 하면됨
3. git add.
4. git commit ☆ merge는 계획이 있었음 내용은 간단하지 않았던 것임.

rebasing branch

merge 하기 전에 rebase를 통해
conflict를 먼저 해결하고
main에 가서 merge 하면
문제없이 merge 된다.

깔끔하고 보관하기 좋은 만든다.

git rebase main

- take care of conflicts
- add - stage에 올리고 ☆ commit을 안한다!!
- git rebase --continue

Branch(1)

Show available local branch

```
$ git branch
```

Show available remote branch

```
$ git branch -r
```

Show available All branch

```
$ git branch -a
```

Branch(2)

Create branch

```
$ git branch stem
```

Checkout branch

```
$ git checkout stem
```

Create & Checkout branch

```
$ git checkout -b new-stem
```

make changes inside readme.md

```
$ git commit -a -m 'edit readme.md'
```

```
$ git checkout master
```

merge branch

```
$ git merge stem
```

Branch(3)

delete branch

```
$ git branch -D stem
```

push with specified remote branch

```
$ git push origin stem
```

see the difference between two branches

```
$ git diff master stem
```

Branch asciinema

```
for _ in range(10):
    print("Fizzybuzzy")
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ vi fizzbuzz.py
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ python3 fizzbuzz.py
Fizzybuzzy
Fizzybuzzy
Fizzybuzzy
Fizzybuzzy
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ vi fizzbuzz.py
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ python3 fizzbuzz.py
Fizzybuzzy, 0
Fizzybuzzy, 3
Fizzybuzzy, 6
Fizzybuzzy, 9
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ vi fizzbuzz.py
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ python3 fizzbuzz.py
Fizzybuzzy, 3
Fizzybuzzy, 6
Fizzybuzzy, 9
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ git status
On branch feature/fizz
Changes not staged for commit:
  (use "git add <file>..." to update what will be commit
  (use "git checkout -- <file>..." to discard changes in  directory)

        modified:   fizzbuzz.py

no changes added to commit (use "git add" and/or "git commit -a")
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ git add fizzbuzz.py
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ git commit -m "feat: Add fizz solution
> print fizzybuzzy while i is times of 3"
[feature/fizz 3e2b305] feat: Add fizz solution print fizzybuzzy while i is times of 3
 1 file changed, 3 insertions(+), 2 deletions(-)
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ cat fizzbuzz.py
for i in range(1,10):
    if i%3==0:
        print(f"Fizzybuzzy, {i}")
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ cat fizzbuzz.py
for _ in range(10):
    print("Fizzybuzzy")
vagrant@ubuntu-bionic:~/Documents/dev/upgraded-fortnight$ vi
```

branching models 전략

- 예) 4X4 기능을 뺀다.
- git flow
  - (hotfix)- `master` -(release)- `develop` - feature
  - pros: 가장 많이 적용, 각 단계가 명확히 구분
  - cons: 복잡..
- github flow
  - `master` - feature
  - pros: 브랜치 모델 단순화, `master` 의 모든 커밋은 deployable
  - cons: CI 의존성 높음. 누구 하나라도 실수했다간..(pull request로 방지)
- gitlab flow
  - `production` - `pre-production` - `master` - feature
  - pros: deploy, issue에 대한 대응이 가능하도록 보완
  - cons: git flow와 반대 (`master` -develop, `production` -master)

main 사용자가 보게될

life cycle 있음

다음버전 개발은 여기서

기능별로 다음버전을 수정. branch 에서.

unit test

최종 test를 하는 road test

보완하는것이 목표. 목표치를 정해놓기

위험. 간단

git flow strategy

Git flow

use git flow easily!

Link

Collaborate with your Co-worker

Method 1: Collaboration

Add Collaborator

Collaboration

Add, Commit and Push like you own it.

Method 2: Fork and Merge

Fork and Merge

Forking JKeun/study-of-regression-toyota-corolla

It should only take a few seconds.

Refresh

A
fork
pull request
local
B
C
D

A
B
E
C
D

Fork and Merge

Fork and Merge

```
$ git clone https://github.com/username/forked-repo.git
```

Fork and Merge

```
$ git branch -a
```

```
$ git checkout -b new-feature
```

Fork and Merge

Make some change

```
$ git add file
```

```
$ git commit -m "commit message"
```

```
$ git push origin new-feature
```

Fork and Merge

Fork and Merge

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base fork: kingwangzzang1234/kingwa... | base: master | ... | head fork: ulgoon/kingwangzzang1234... | compare: edit-index

✓ Able to merge. These branches can be automatically merged.

edit index.html

Write | Preview

add header, footer tag

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

☑ Allow edits from maintainers. Learn more

Create pull request

Fork and Merge

Fork and Merge

Fork and Merge

Fork and Merge

edit index.html #2

Open ulgoon wants to merge 1 commit into `kingwangzzang1234:master` from `ulgoon:edit-index`

Conversation 0 | Commits 1 | Files changed 1

ulgoon commented 38 seconds ago — Contributor

add header, footer tag

edit index.html ··· d81b362

Add more commits by pushing to the **edit-index** branch on **ulgoon/kingwangzzang1234.github.io**.

Merge pull request #2 from ulgoon/edit-index

edit index.html

Confirm merge | Cancel

Fork and Merge

1. 팀장이 자기 소유의 repo를 만든다
2. clone 하여 약간의 작업 후 add, commit, push
3. notice 받은 팀원은 fork 하여 자신의 소유가 된 repo를 clone
4. 작업할 내용을 미리 issue 등록
   4-1. or 팀장이 강제 issue 생성 후 작업지시
5. `$ git remote add pmorigin {pm의 repo 주소}` 하여 미리 등록
6. feature branch 생성 후 작업, add, commit(with issue no)
7. master로 merge 한 후, 나의 master로 push
8. pull request 생성
9. 팀장은 확인 후 merge(conflict, file change 확인 필수)

volvocars main page clone with git flow

1. role 분배(40분안에 클론하려면 어떻게 역할을 나눠야 할까)
2. 위의 과정 반복

continuous pull

continuous pull

```
$ git remote add upstream https://github.com/anotheruser/original-repo.git
```

```
$ git fetch upstream
```

```
$ git merge upstream/master
```

How to move files

Don't

```
$ mv style.css static/css
```

-> deleted:
-> untracked files:

Do

```
$ git mv style.css static/css
```

-> renamed:

Issue & Projects

Issue: 프로젝트, 레포와 관계된 모든 해야할 일과 버그, 개선사항 등을 기록

Projects: 해야할 일의 진도에 따른 구성과 우선순위 지정

Issue(1)

Createt Sample Issue #3

Open **ulgoon** opened this issue 29 seconds ago · 0 comments

Write Preview H

We need to make issue for sample

TODO:
- [ ] #1
- [ ] #2

Attach files by dragging & dropping, selecting or pasting them.

Issue(2)

Createt Sample Issue #3

Open **ulgoon** opened this issue 1 minute ago · 0 comments

ulgoon commented 1 minute ago • edited

We need to make issue for sample

TODO:

- [ ] #1
- [ ] #2

Issue(3-1)

Assignees

ulgoon

Labels

enhancement

Projects

backend

Awaiting triage

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

Issue(3-2)

- Assignees: 이 이슈에 대한 책임인원
- Labels: 이슈의 종류
- Projects: 이슈를 배당할 프로젝트
- Milestone: 이슈에 해당하는 중요 시점 지정

Projects

Projects(2)

backend
Updated on Mar 16
Filter cards
Add cards
Fullscreen
Menu
1 Backlog
Create Sample Issue
0 of 2
#3 opened by ulgoon
enhancement
0 TODO
0 Doing
0 Review
1 Done
Project init
#2 opened by ulgoon
enhancement