

git, github

Megabyte School : 프론트엔드 개발 취업 연계 과정

최우영

- Co-founder, CTO(disceptio)
- Solution Architect, Web Developer, Instructor
- Lang&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

Goal

- git을 사용하기 위해 CLI Shell command와 Vim editor를 다룰 수 있다.
- 코드 관리를 위한 git의 정확한 사용법을 이해한다.
- git의 저장소 개념을 이해하고, 원격 저장소 서비스의 차이를 인식한다.
- git을 사용하면서 발생하는 다양한 상황을 해결할 수 있다.
- commit의 보편적인 작성법을 이해하고 이를 활용하여 commit을 작성할 수 있다.
- hexo를 활용하여 github pages에 blog를 생성하고 포스트를 작성할 수 있다.
- git의 branch model을 활용해 능숙하게 코드관리할 수 있다.
- git의 다양한 branch 전략을 이해하고 널리 사용되는 git flow 전략을 활용하여 프로젝트를 수행할 수 있다.
- github projects와 issue로 프로젝트 이슈를 관리할 수 있다.
- git으로 타인과 협업하며, 다른 프로젝트에 기여할 수 있다.

Prerequisite

For windows

- git for windows(<https://gitforwindows.org/>)

For linux, MacOS

- git(installed)

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

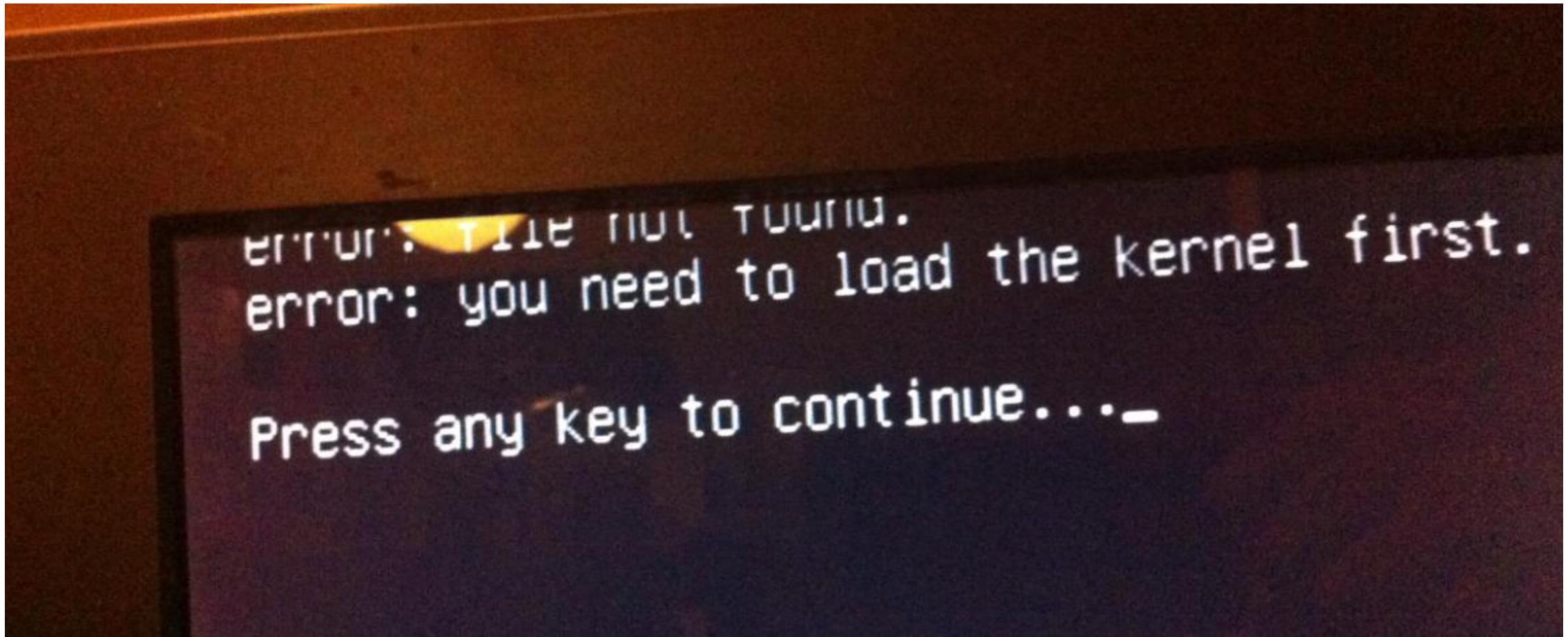
Before Linux(3)

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

Meaning of GNU

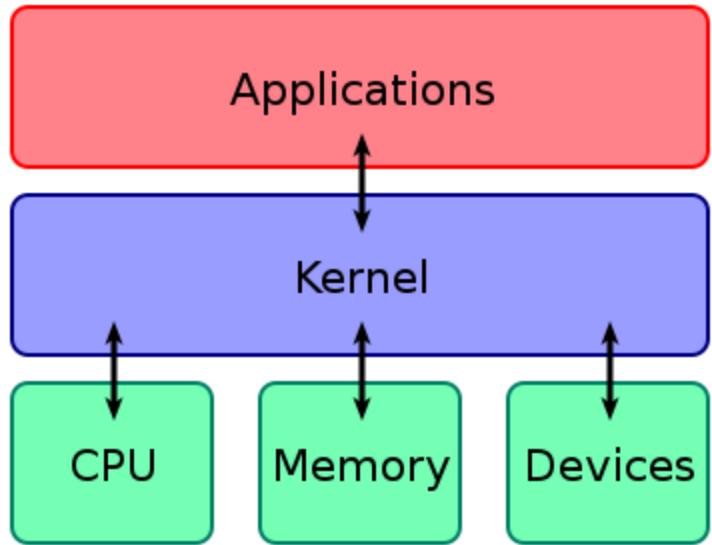
GNU == G NU is N ot U nix

Before Linux(4)



- But, GNU 프로젝트에는 커널이 없었고..

Kernel



- 하드웨어와 응용프로그램을 이어주는 운영체제의 핵심 시스템소프트웨어

Linus Torvalds

<https://www.youtube.com/embed/lVpOyKCNZYw>

- 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 운영 체제
- 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
- 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

```
$ cd Documents/  
$ mkdir dev  
$ cd ..  
$ pwd  
  
$ touch readme.md  
$ mv readme.md bin/  
$ cp readme.md bin/  
$ mv readme.md ./README.txt  
$ rm README.txt  
  
$ rm -rf bin/  
$ chmod 750 readme.md  
$ cat readme.md  
$ vi readme.md
```

Practice(1)

다음의 절차를 수행하여 최종 디렉토리와 파일 구조를 완성하세요.

- cli-practice/ 디렉토리를 생성하세요.
- 생성된 디렉토리에 README.txt 파일을 생성하세요.
- bin/ 디렉토리를 생성하여 README.txt 를 복사하여 introduce.md 로 이름을 바꾸세요.

```
~/Documents/dev
  - cli-practice/
    - bin/
      - introduce.md
      - README.txt
```

Recap - Vim command

h j k l - left, down, up, right
i - insert mode
v - visual mode
ESC - back to normal mode
d - delete
dd - delete a line
y - yank
yy - yank a line
p - paste
u - undo
a - append
A - append from end of line
o - open line(under)
O - open line(upper)
H - move to the top of the screen
L - move to the bottom of the screen

Command mode

```
:q - quit  
:q! - quit discarding all changes  
:w - write  
:wq - write and quit  
:{number} - jump to {number}th line.
```

Bonus - Markdown

```
<!-- 주석표기 -->

<!-- 제목 텍스트 -->
# h1
## h2
### h3
#### h4
##### h5
###### h6

<!-- 순서 없는 리스트( - * + 혼용 가능) -->
- Item1
  - Item1-1
    - Item1-1-1
* Item2
+ Item3

<!-- 순서 있는 리스트 -->
1. Item1
2. Item2
<!-- 하이퍼링크 -->
[링크 텍스트](링크 URL)

<!-- 이미지 -->
![대체 텍스트](이미지 URL)
```

```
<!-- 강조 표기 -->
*Italic*
**Bold**
~Line Break~
_Single underscore_

<!-- 인용문(Blockquote) -->
> 인용할 문장

<!-- Code 입력(문장 내) -->
This is how `code` works.

<!-- Code 입력(블록) -->
```
def say_hello():
 return "hello"
```

<!-- 수평선 -->
-----
```

Page 1

Page 2

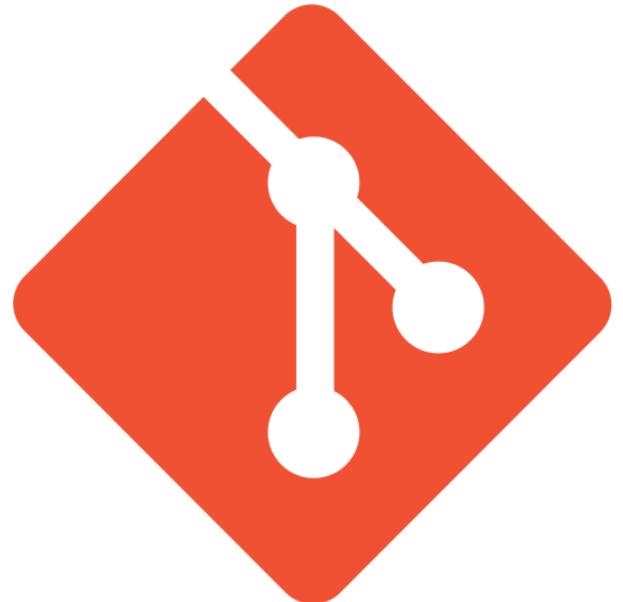
Page 3

Practice(2)

Vim 을 이용해 앞서 생성한 `introduce.md`에 Markdown 문법으로 간단한 자기 소개 글을 작성하세요.

Use Vim in real world!

- vim advanture
- vimium
- vs code vim extension
- intellij vim plugin



git

VCS (Version Control System)

== SCM (Source Code Management)

< SCM (Software Configuration Management: 형상관리)

chronicle of git



git by Linus Torvalds

- Linux Kernel을 만들기 위해 Subversion을 쓰다 화가 난 리누스 토발즈는 2주만에 git이라는 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

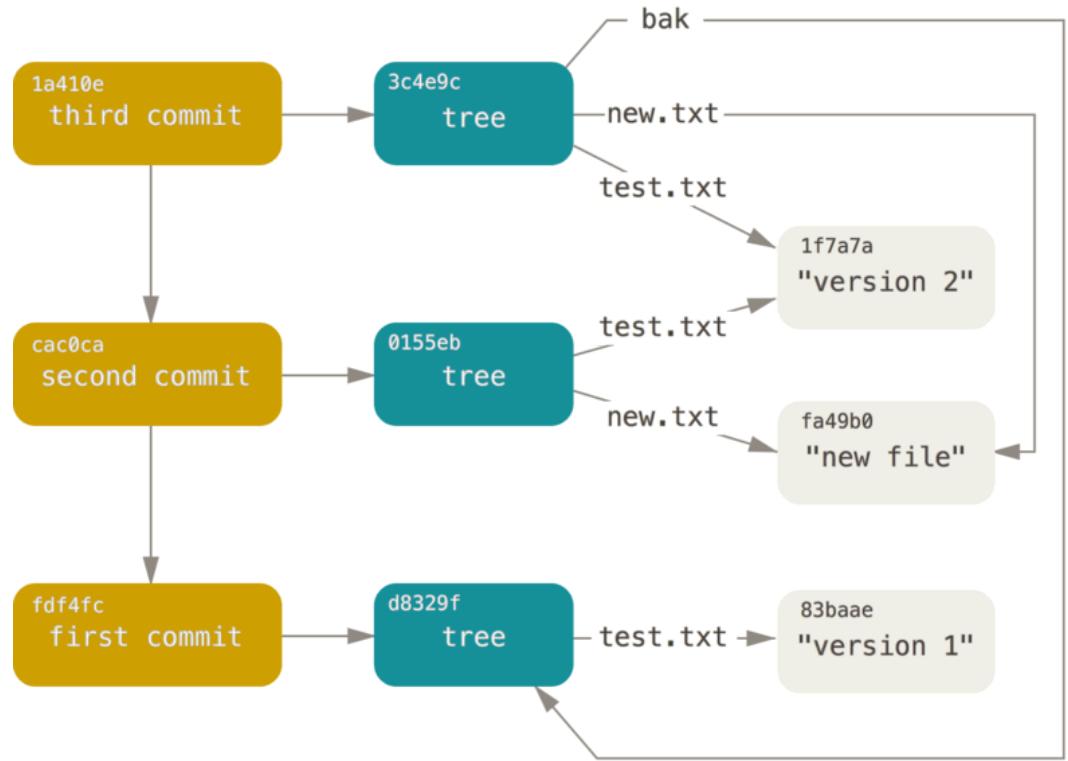
- git GUI
- sourcetree
- kraken
- smartGit

CLI first

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

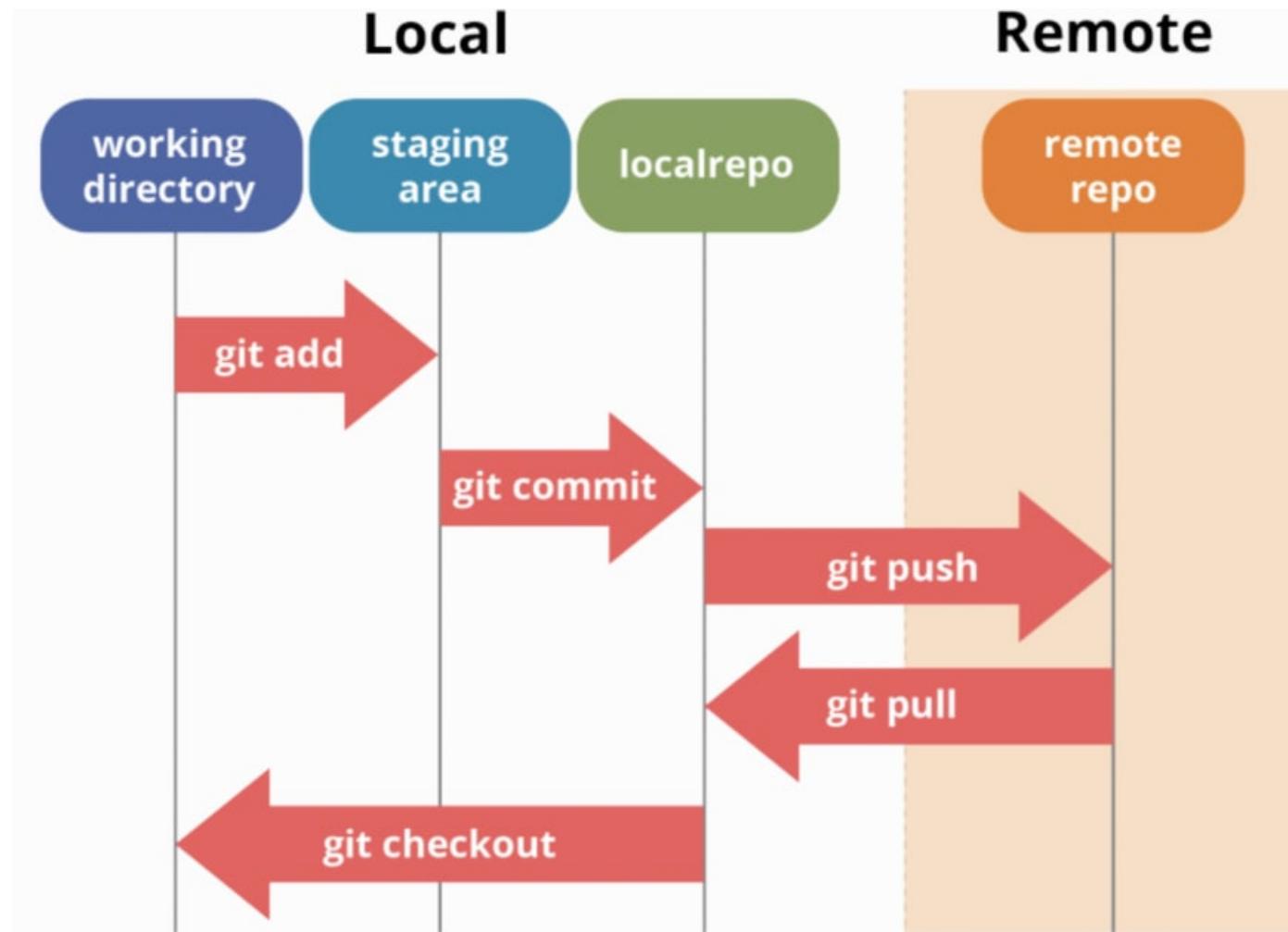
git objects

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



- ref: [git internals - git objects](#)

git Process Flow and Command



git is not equal to github



git



Cloud Remote Repository Services

- Github: 비영리였던, Microsoft에 인수된 가장 유명한 서비스
- Bitbucket: Atlassian이 서비스. jira, confluence, trello 등의 부가도구와 유기적
- GitLab: GitLab이 서비스. 사설 서버 구성이 가능.

Before Start

- git 설치 확인(`$ git -v`)
- git 환경설정

```
$ git config --global user.name "당신의유저네임"  
$ git config --global user.email "당신의메일주소"  
$ git config --global core.editor "vim"  
$ git config --global core.pager "cat"
```

- lg alias 설정: [johanmeiring/gist:3002458](#)
- `$ git config --list` 로 정상 설정 확인
- 수정이 필요할 경우, `$ vi ~/.gitconfig` 에서 수정 가능

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/2019/09/01-git-first/>

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 main
```

Start Project with `git clone`

start project with clone

- github에서 repo를 생성합니다.

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

Conventional Commits

ref: <https://www.conventionalcommits.org/ko/v1.0.0/>

1. commit의 제목은 commit을 설명하는 하나의 구나 절로 완성
2. importanceofcapitalize **Importance of Capitalize**
3. prefix 꼭 달기
 - feat: 기능 개발 관련
 - fix: 오류 개선 혹은 버그 패치
 - docs: 문서화 작업
 - test: test 관련
 - conf: 환경설정 관련
 - build: 빌드 관련
 - ci: Continuous Integration 관련

Conventional Commits - example

Commit Convention은 팀마다 다를 수 있으니 관련 문서를 참조할 것.

```
feat: Add server.py
fix: Fix Typo server.py
docs: Add README.md, LICENSE
conf: Create .env, .gitignore, dockerfile
BREAKING CHANGE: Drop Support /api/v1
refactor: Refactor user classes
```

commit 할 때 기억해야 할 것

- commit은 동작 가능한 최소단위로 자주 할 것.
- 해당 작업단위에 수행된 모든 파일 변화가 해당 commit에 포함되어야 함.
- 모두가 이해할 수 있는 log를 작성할 것.
- Open Source Contribution시 영어가 강제되지만, 그렇지 않을 경우 팀 내 사용 언어를 따라 쓸 것.
- 제목은 축약하여 쓰되(50자 이내), 내용은 문장형으로 작성하여 추가설명 할 것.
- 제목과 내용은 한 줄 띄워 분리할 것.
- 내용은 이 commit의 구성과 의도를 충실히 작성할 것.

First Push

```
$ git push origin main
```

Practice(3)

- 방금 생성한 repository에 다음 commit을 만족하도록 작성하여 push 하세요.
- requirements
 - 언어는 본인이 편한 언어로 작성할 것
 - 프로그래밍 언어 사용이 어려운 경우, .md 파일로 작성 가능

```
- feat: Create adder.{py}
- feat: Add Feature - Return sum of 2 numbers
- fix: Rename Function add to adder
- docs: Create contribute.md to describe how to use these
- refactor: Create main.{py} to run on main function
```

README.md

- 프로젝트와 Repository를 설명하는 책의 표지와 같은 문서
- 나와 동료, 이 repo의 사용자를 위한 문서

```
# Project Name
Abstract your project in few lines.
see [project sample page](project link)

## Documentation

### Installation
To install,
`$ pip install sesame`
and run `\$ python open_sesame.py`

### Supported Python versions
`>=3.6`

### More Information
- [API docs]()
- [Official website]()

### Contributing
Please see [CONTRIBUTING.md]()

### License
Sesame is Free software, and may be redistributed under the terms of specified in the [LICENSE]() file.
```

.gitignore

.gitignore 는 git이 파일을 추적할 때, 어떤 파일이나 폴더 등을 추적하지 않도록 명시하기 위해 작성하며, 해당 문서에 작성된 리스트는 수정사항이 발생해도 git이 무시하게 됩니다. 특정 파일 확장자를 무시하거나 이름에 패턴이 존재하는 경우, 또는 특정 디렉토리 아래의 모든 파일을 무시할 수 있습니다.

```
# 주석을 달기 위한 Hashtag
# MacOS Setup
.DS_Store
# Python cache files
.py[cd]
# Important files
/Important
# AWS key
key.pem
```

LICENSE

오픈소스 프로젝트에서 가장 중요한 License는 내가 만들 때에도, 배포할 때에도 가장 신경써야 하는 일 중 하나입니다.

가장 많이 사용하는 License는 다음과 같습니다.

- MIT License
 - MIT에서 만든 라이센스로, 모든 행동에 제약이 없으며, 저작권자는 소프트웨어와 관련한 책임에서 자유롭습니다.
- Apache License 2.0
 - Apache 재단이 만든 라이센스로, 특허권 관련 내용이 포함되어 있습니다.
- GNU General Public License v3.0
 - 가장 많이 알려져 있으며, 의무사항(해당 라이센스가 적용된 소스코드 사용시 GPL을 따라야 함)이 존재합니다.

git은 습관이 가장 중요!

- TIL(Today I Learned..) repository를 만들고 매일 학습하거나 얻은 지식을 정리
 - commit을 쌓아 commit 하는 습관도 기르고, 나중에 찾아보기 쉬움!
- Github blog
 - [hexo](#) + {username}.github.io repository로 정적 블로그를 만들어 정리하는 습관을 만들고 Markdown과 친해짐!

TIL, Github blog는 github을 이용하여 개인적으로 관리 추천!

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 through 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

```
<!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>

Branch

Branch

- 분기점을 생성하여 독립적으로 코드를 변경할 수 있도록 도와주는 모델

master

```
print('hello' + ' ' + 'world')
```

develop

```
words = ['world', 'hello']
print(' '.join(words[:-1]))
```

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
```

Practice(1)

- Spiderman.md를 생성하고 다음의 정보를 배열을 맵은 배우별로 브랜치를 생성하여 이를 시각화 하세요.
- 완결된 브랜치는 master 브랜치로 merge 해야 합니다.
- 각 commit은 개봉연도 순서대로 존재해야 합니다.
- Format

```
# {Movie Name}
- Year:
- Name:
```

- TobeyMaguire branch

```
# Spider-Man 1
```

- Year: 2002
- Name: Peter Benjamin Parker

```
# Spider-Man 2
```

- Year: 2004
- Name: Peter Benjamin Parker

```
# Spider-Man 3
```

- Year: 2007
- Name: Peter Benjamin Parker

- AndrewGarfield branch

```
# Amazing Spider-Man 1
```

- Year: 2012
- Name: Peter Benjamin Parker

```
# Amazing Spider-Man 2
```

- Year: 2014
- Name: Peter Benjamin Parker

- Tom Holland branch

Captain America: Civil War

- Year: 2016
- Name: Peter Benjamin Parker

Spider-Man: Home Coming

- Year: 2017
- Name: Peter Benjamin Parker

Avengers: Infinity war

- Year: 2018
- Name: Peter Benjamin Parker

Avengers: Endgame

- Year: 2019
- Name: Peter Benjamin Parker

Spider-Man: Far From Home

- Year: 2019
- Name: Peter Benjamin Parker

Additional Practice

- Venom branch
- Into the Spider-verse branch(Miles Morales)

branching models

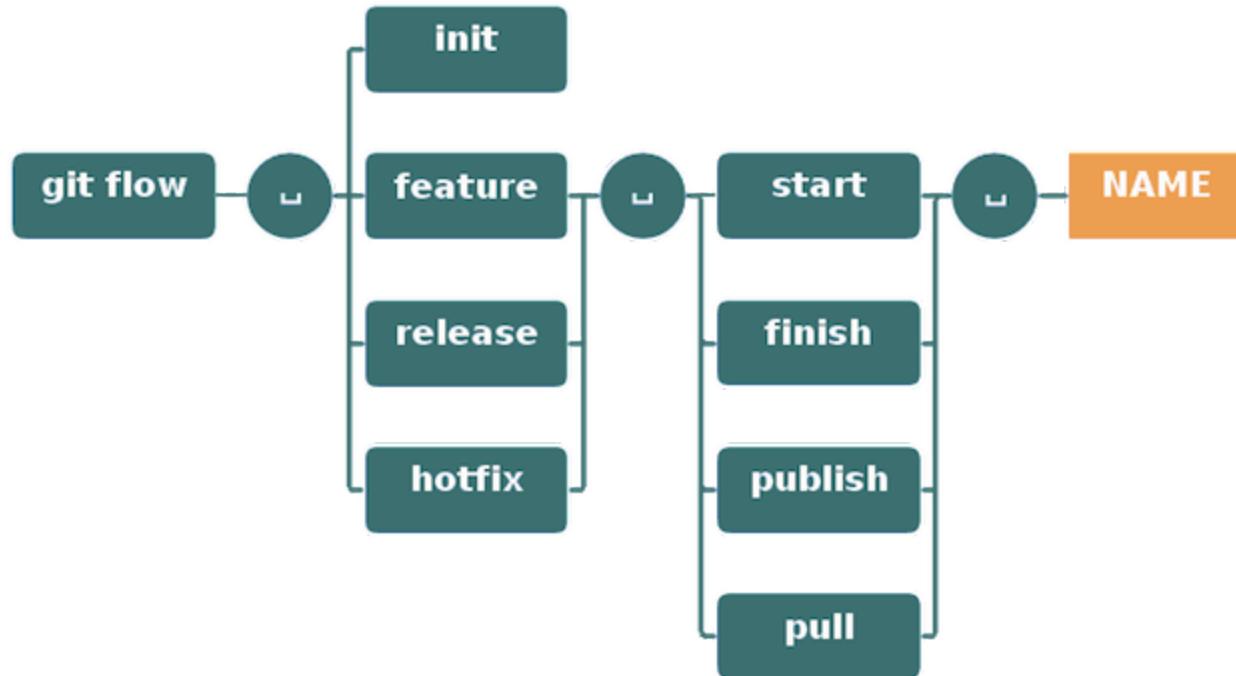
- git flow
 - (hotfix)- master -(release)- develop - feature
 - pros: 가장 많이 적용, 각 단계가 명확히 구분
 - cons: 복잡..
- github flow
 - master - feature
 - pros: 브랜치 모델 단순화, master 의 모든 커밋은 deployable
 - cons: CI 의존성 높음. 누구 하나라도 실수했다간..(pull request로 방지)
- gitlab flow
 - production - pre-production - master - feature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

git flow strategy



use git flow easily!

Link



fibonacci 구현 with git flow

v1.0: recursion(점화식)

v1.1: recursion with memoization(점화식 + 메모리)

v2.0: binet's formula(공식)

꼭 해야 하는 것

1. 새로운 레포
2. 동일한 파일에 대한 버전
3. github issues, projects를 작업 전에 완료하기

Practice(2)

- git flow 전략을 활용하여 어제 작성한 introduce.md를 index.html에 재작성하세요.

Requirements

- develop 브랜치에서 다음 릴리즈를 위한 개발이 끝나야 합니다.
- head, body 등 section별 작업은 각각의 브랜치에서 작업되어야 합니다.
- css, js 작업 또한 각 브랜치를 소유합니다.(선택)
- Semantic Web Elements를 적극 활용하세요.

Revert Everything!

Rename

- Worst

```
$ mv server.py main.py -> deleted, new file
```

- Best

```
$ git mv server.py main.py -> renamed
```

| 파일의 history를 남기기 위해서는 삭제 후 생성이 아닌 이름바꾸기로 추적

Undoing

```
$ git checkout -- . or $ git checkout -- {filename}
```

Unstaging

```
$ git reset HEAD {filename}
```

Unstaging and Remove

```
$ git rm -f {filename}
```

Edit latest commit

```
$ git commit --amend
```

Edit prior commit

```
$ git rebase -i <commit>
```

abort rebase

```
$ git rebase --abort
```

Complete rebase

```
$ git rebase --continue
```

Reset Commit

Worst case: Reset

ex) 직전 3개의 commit을 삭제한 후, remote에 강제 push

```
$ git reset --hard HEAD~3  
$ git push -f origin <branch>
```

- 협업 시 다른 cloned repo에 존재하던 commit log로 인해 파일이 살아나거나, 과거 이력이 깔끔히 사라져 commit log tracking이 힘들어짐.
- solution: 잘못한 이력도 commit으로 박제하고 수정한 이력을 남기자!

Best case: Revert

ex) 현재 HEAD에서 직전의 3개의 commit을 순서대로 거슬러 올라가 해당 내역에 대해 commit, push 수행

```
$ git revert --no-commit HEAD~3..  
$ git commit  
$ git push origin <branch>
```

- 잘못하기 전 과거로 돌아가 최신을 유지하면서 되돌렸다는 이력을 commit으로 남겨 모든 팀원이 이 사항을 공유하고 주지시킬 수 있음.
- commit을 따로 안할땐 `--no-edit`
- merge commit을 되돌릴 땐 `-m ($git revert -m {1 or 2} {merge commit id})`

Collaborate with your teammates

git flow workflow

Collaboration

Add, Commit and Push like you own it.

Method 2: Fork and Merge



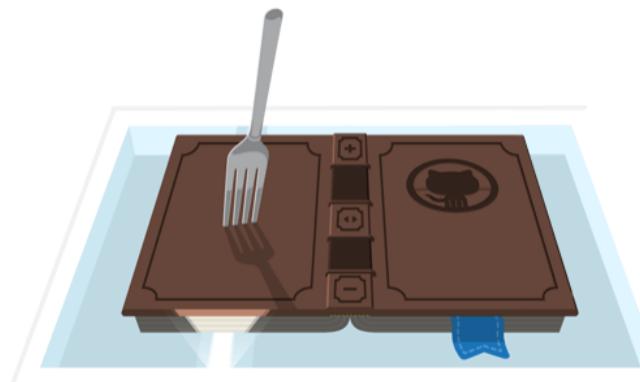
� ▾

Fork and Merge

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

It should only take a few seconds.

 Refresh



Fork and Merge

 [ulgooon / study-of-regression-toyota-corolla](#)
forked from [JKeun/study-of-regression-toyota-corolla](#)

 [Code](#)  [Pull requests 0](#)  [Projects 0](#)  [Wiki](#) 

 Study - Regression Analysis using ToyotaCorolla dataset

[Add topics](#)

 [9 commits](#)

 [1 branch](#)

Branch: [master](#) ▾

[New pull request](#)

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

No description, website, or topics provided.

Edit

Add topics

1 commit

3 branches

0 releases

1 contributor

Your recently pushed branches:

edit-index (less than a minute ago)

Compare & pull request

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.

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 17 seconds ago

Contributor + 

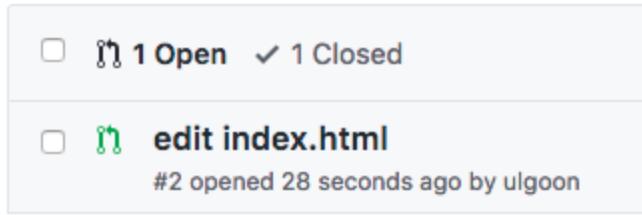
add header, footer tag

 edit index.html ... d81b362

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

  This branch has no conflicts with the base branch
Only those with [write access](#) to this repository can merge pull requests.

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`.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or [view command line instructions](#).

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

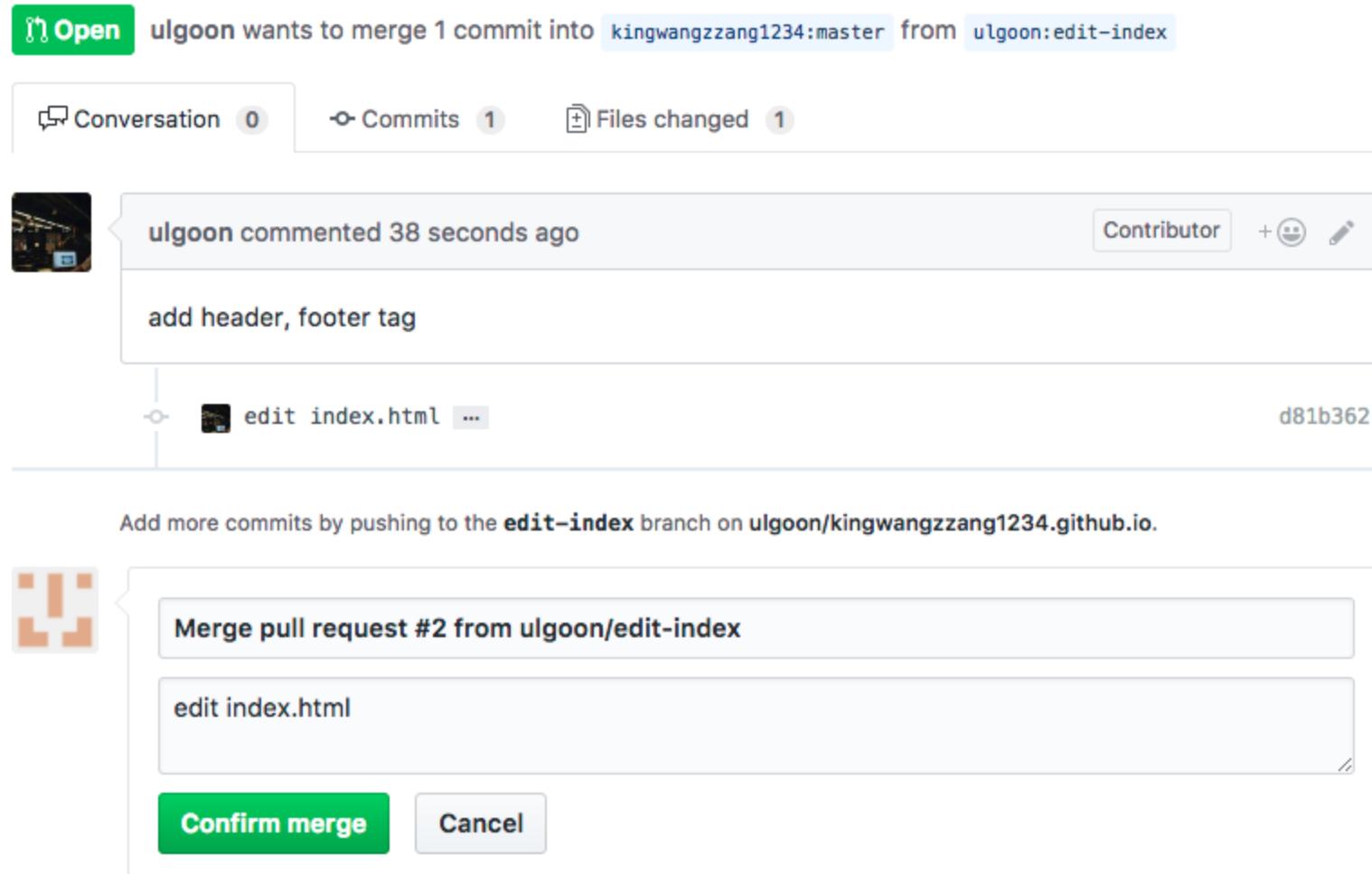
edit index.html ... d81b362

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

edit index.html #2

Merged kingwangzzang... merged 1 commit into kingwangzzang1234:master from ulgoon:edit-index just now

Conversation 0 Commits 1 Files changed 1

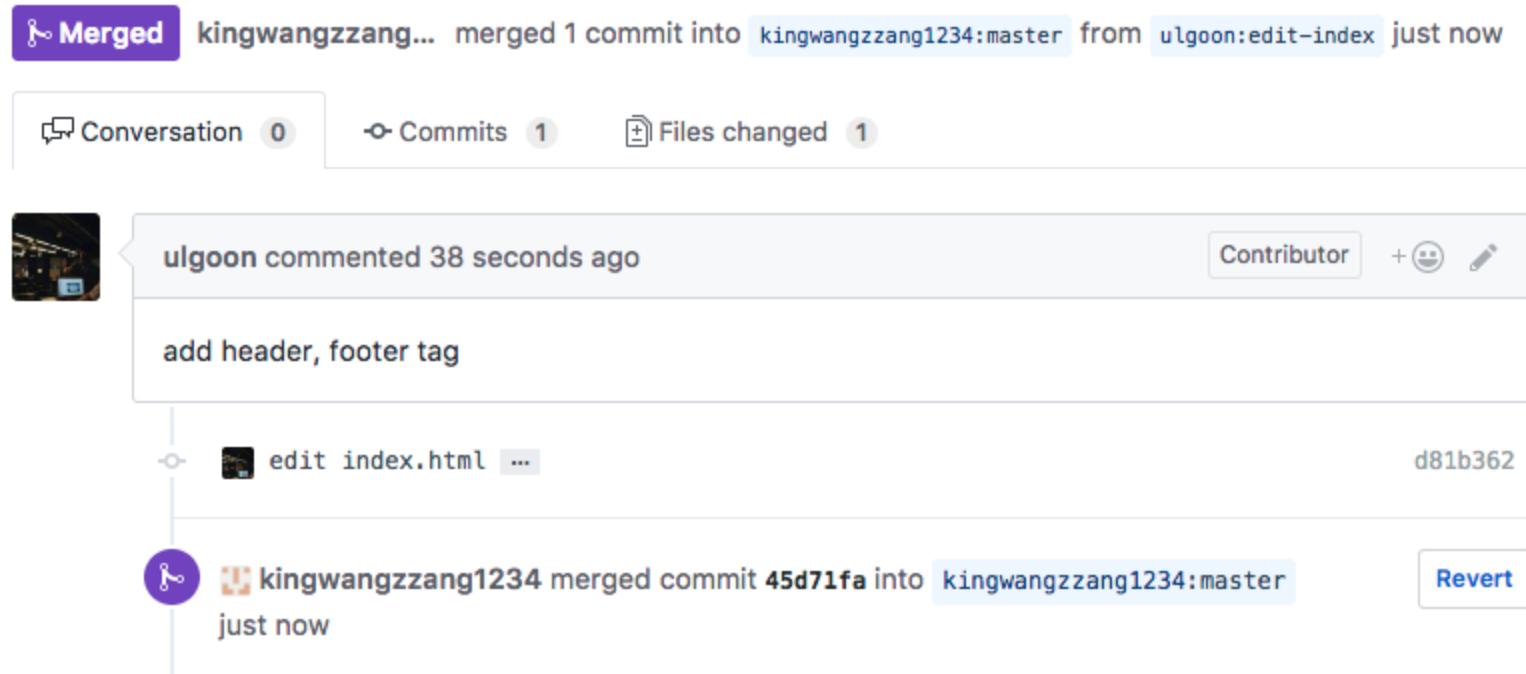
ulgoon commented 38 seconds ago

Contributor + 😊 🖊

add header, footer tag

edit index.html ... d81b362

kingwangzzang1234 merged commit 45d71fa into kingwangzzang1234:master just now Revert



dev2,dev3,devn, .. : Update develop branch

In case of having upstream

```
$ git fetch upstream develop  
$ git merge FETCH_HEAD
```

```
$ git remote add pmorigin {PM repo addr}  
$ git fetch pmorigin develop  
$ git merge FETCH_HEAD
```

Final Practice

- 3~4인이 팀이 되어 프로젝트 수행
- 아래의 과제 중 하나를 수행할 것
 - i. [피보나치킨](#) 클론(치킨과 인원 수에 따라 적절한 맥주의 용량도 출력)
 - ii. 블랙잭 게임
 - iii. [Pig the dice game](#)
 - iv. 인디언포커
- Requirements
 - 타겟 플랫폼, 언어나 Framework는 팀 내 협의 후 결정
 - R&R 분배 -> 구현 -> 평가 순으로 진행

3.Pig the dice game

- n명의 플레이어가 참여($n=1$: PvC, $n<4$ (3+Computer))
- 시계방향으로 턴을 진행하며 각 턴 당 시도횟수는 무제한
- Roll or Stop을 선택
- Roll 시 1이 나오면 해당 턴에서 획득한 모든 점수는 박탈, 턴 종료
- Stop 시 해당 턴에서 획득한 점수 보전
- 가장 먼저 100점을 획득한 플레이어(또는 컴퓨터)가 승리
- 컴퓨터의 게임 진행 알고리듬은 Roll을 1회 이상 진행하는 Random 진행으로 적용

4. 인디언 포커

- 아래의 인디언 포커 설명 문서를 읽고, 이를 구현 하시오
- [https://en.wikipedia.org/wiki/Blind_man's_bluff_\(poker\)](https://en.wikipedia.org/wiki/Blind_man's_bluff_(poker))
- 덱 구성은 [1,2,3,4,5,6,7,8,9,10] * 2로 하되 모두 소진할 때 까지 운용한다.
- 컴퓨터는 오로지 받을 수만 있다.
- PVP일 경우, 베팅을 구현한다.(optional)
- 플레이어는 20개의 칩으로 시작하며, 칩을 모두 소진시키거나(승) 모두 소진할 경우(패) 게임이 종료된다.
- 게임 중 언제든지 :q를 입력하면 게임은 강제종료된다.