

**git, github**

**K-Digital Training**

# 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](mailto:me@ulgoon.com)
4. discord: @ulgoon

# Goal

- 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.
- git의 branch model을 활용해 능숙하게 코드관리할 수 있다.
- git branching strategy 중 git flow를 이해하고 사용할 수 있다.
- github projects와 issue로 프로젝트 이슈를 관리할 수 있다.
- git으로 타인과 협업하며, 다른 프로젝트에 기여할 수 있다.

# Before Start

## MacOS로 개발할 때 꼭 필요한 도구와 설정

1. [iTerm](#) => 기본 터미널의 대체재. tmux와 호환 Good
2. Xcode Command Line Tools( `$ xcode-select --install` ) => 터미널 환경에서 다양한 도구 지원을 위해 필요한 도구
3. [OhMyZsh](#) => zsh Customize
4. [Homebrew](#) => Missing Package Manager for MacOS

## Windows 로 개발할 때 꼭 필요한 도구와 설정

1. WSL(Windows Subsystem for Linux) - Linux 가상환경을 편하게 사용
2. winget - choco 이후 MS에서 공식 지원하는 Package Manager

# Prerequisite

## For windows

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

## For linux, MacOS

- git(installed)



# VCS (Version Control System)

== SCM (Source Code Management)

< SCM (Software Configuration Management: 형상관리)



# chronicle of git



## git by Linus Torvalds

- Linux Kernal을 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 편한 실험이 가능하며, 성공적으로 개발이 완료되면 **Merge**하여 반영
- 인터넷이 연결되지 않아도 개발할 수 있음

## git GUI Clients

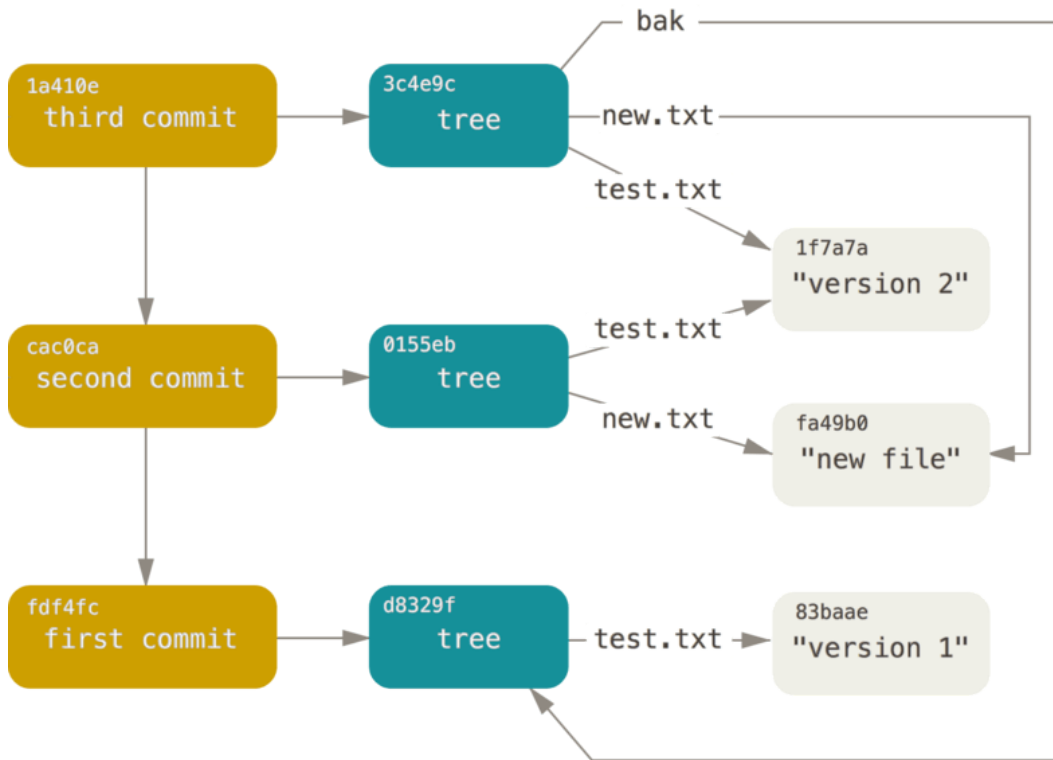
- git GUI
- sourcetree
- kraken
- smartGit

## 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

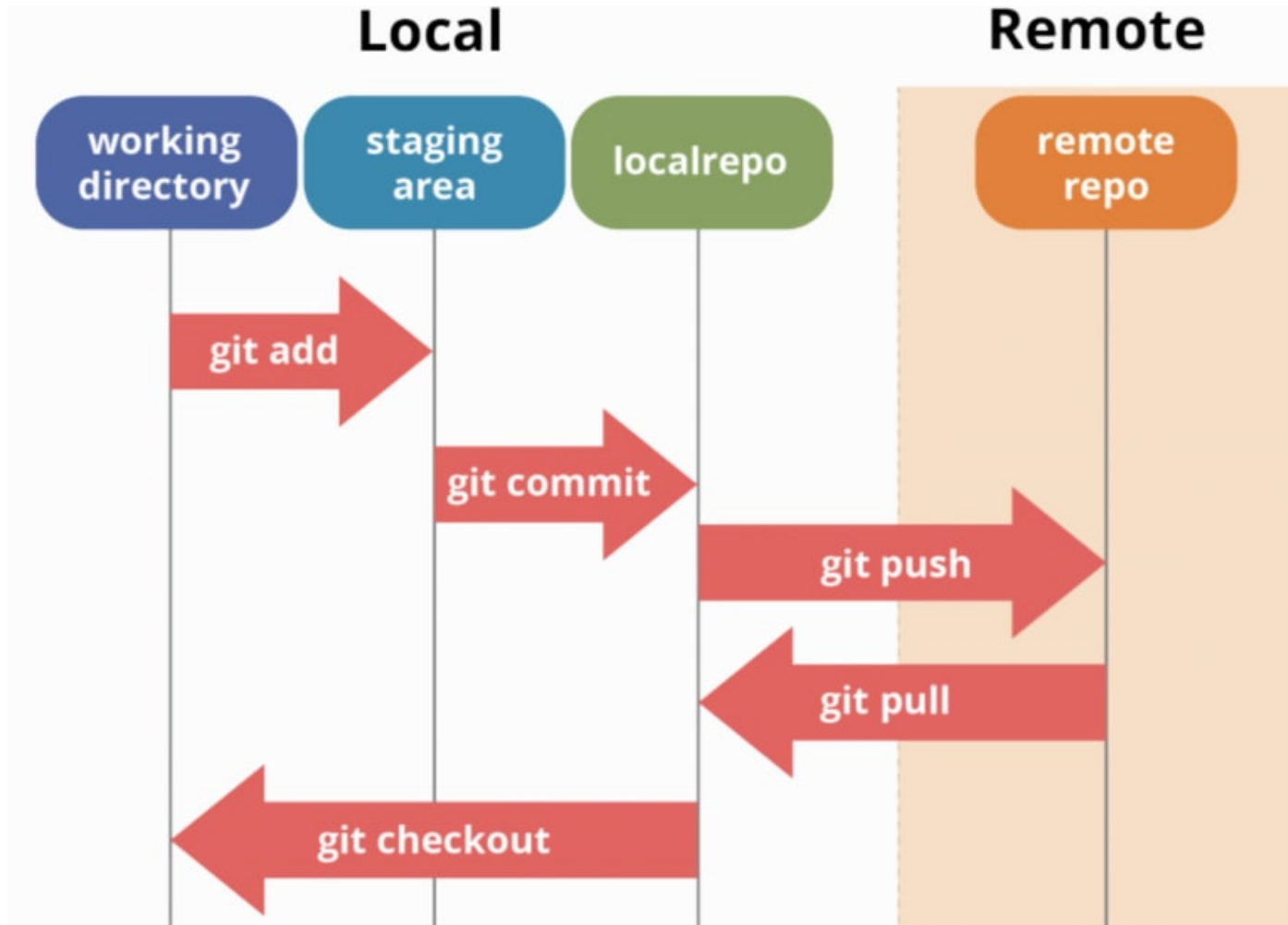
- Blob: 파일 하나의 내용에 대한 정보
- Tree: Blob이나 subtree의 메타데이터(디렉토리 위치, 속성, 이름 등)
- Commit: 커밋 순간의 스냅샷



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



# git Process Flow and Command



# git is not equal to github



## 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](https://gist.github.com/johanmeiring/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/>

**Start Project with `git clone`**

## start project with clone

- github에서 repo를 생성합니다.

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



## First Push

```
$ git push origin main
```

## 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.

# 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: 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
```

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

# 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(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[cdo]
# 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 라이선스로, 모든 행동에 제약이 없으며, 저작권자는 소프트웨어와 관련한 책임에서 자유롭습니다.
- Apache License 2.0
  - Apache 재단이 만든 라이선스로, 특허권 관련 내용이 포함되어 있습니다.
- GNU General Public License v3.0
  - 가장 많이 알려져있으며, 의무사항(해당 라이선스가 적용된 소스코드 사용시 GPL을 따라야 함)이 존재합니다.

# 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)

- multiplication.py를 생성하고 Ethiopian Multiplication을 구현합니다.
- 구현이 끝난 브랜치는 main 브랜치로 merge 해야 합니다.
- branches
  - `def-operation` : 연산 정의
  - `ui` : 사용자 입출력 및 연산 수행
  - `def-lambda` : function을 lambda로 구현

## 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!

# .gitignore and .gitattributes

**.gitignore:** 특정파일 추적을 하고 싶지 않을 경우

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

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

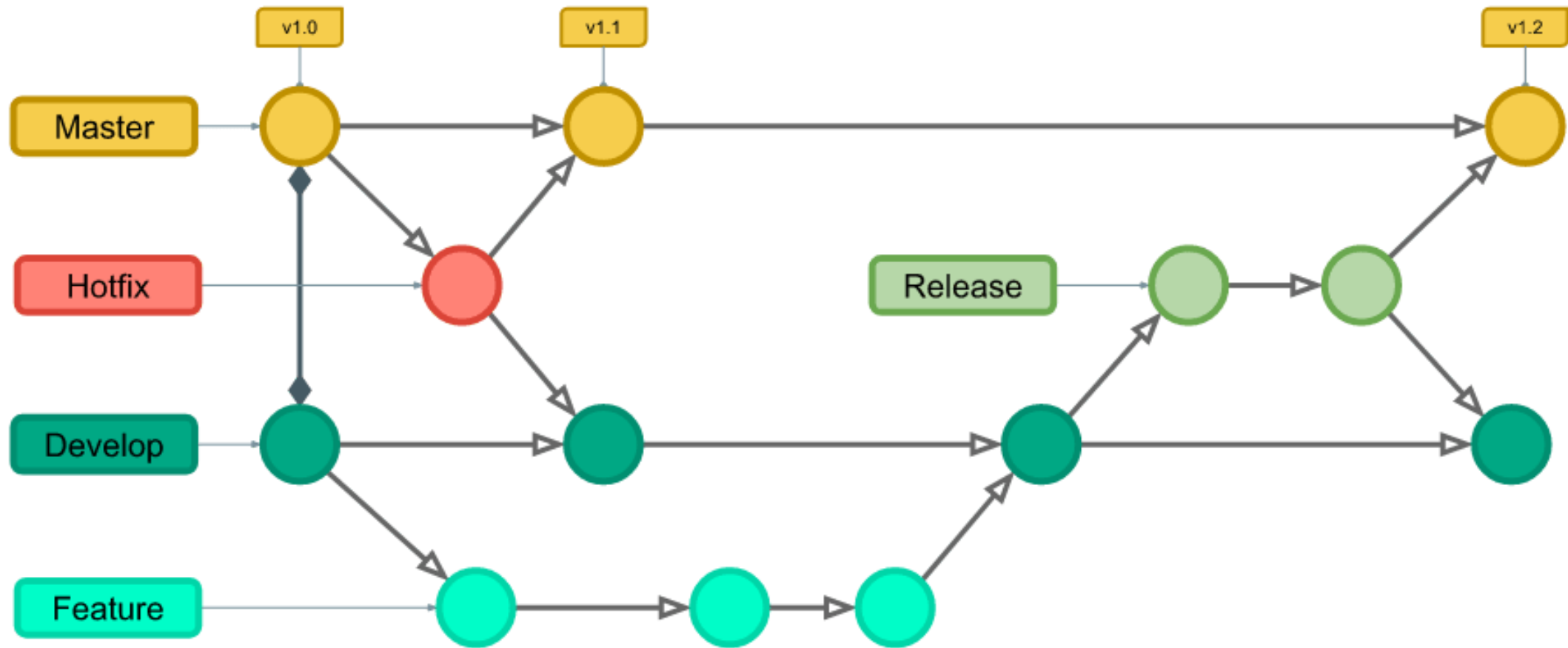
```
# Avoid conflicts in pbxproj files  
*.pbxproj binary merge=union  
  
# Always diff strings files as text  
*.strings text diff
```

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

# branching models

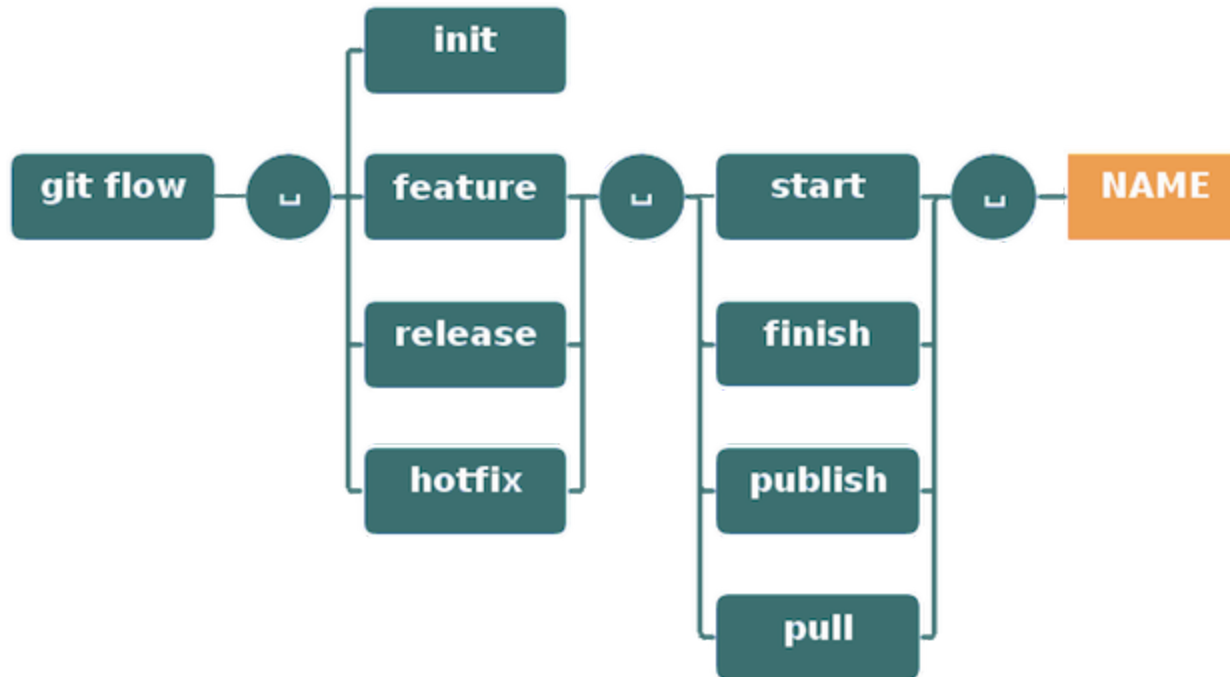
- git flow
  - (hotfix)- `master` -(release)- `develop` - feature
  - pros: 가장 많이 적용, 각 단계가 명확히 구분
  - cons: 복잡..
- github flow
  - `master` - feature
  - pros: 브랜치 모델 단순화, `master` 의 모든 커밋은 deployable
  - cons: CI 의존성 높음. 누구 하나라도 실수했다간..(pull request로 방지)
- gitlab flow
  - `production` - `pre-production`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

v1.1: recursion with memoization

v2.0: binet's formula

## 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 Elements](#)를 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: 잘못된 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.

# github issue and projects

## Issue & Projects

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

Projects: 해야할 일의 진도에 따른 구성과 우선순위 지정



# Issue(1)

## Created Sample Issue #3



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)

### Created Sample Issue #3



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

**Collaborate with your teammates**

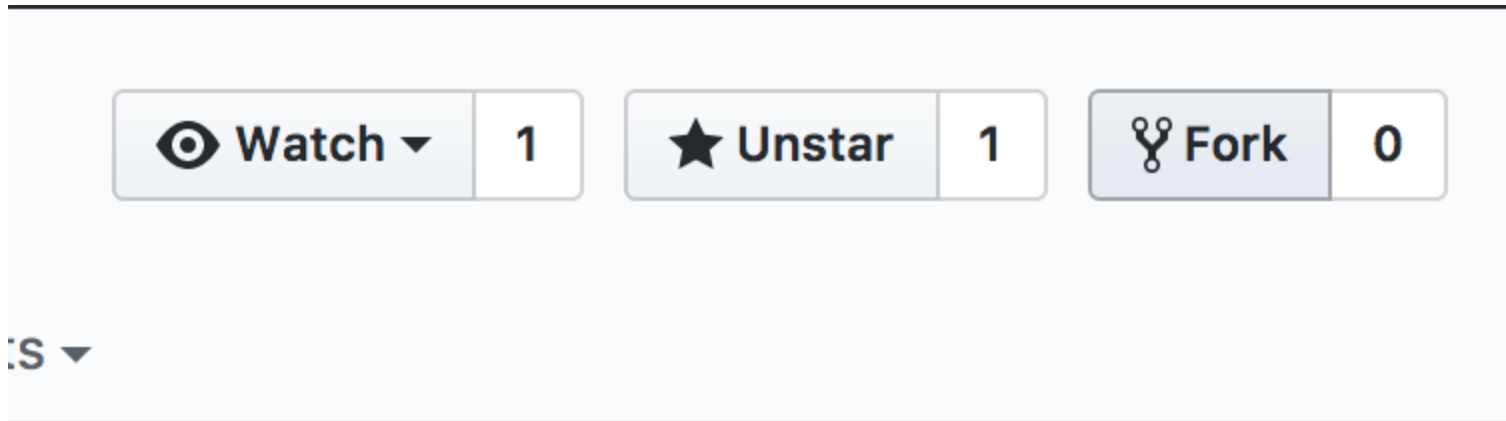
## Forking workflow



# Collaboration

Add, Commit and Push like you own it.

## Method 2: Fork and Merge

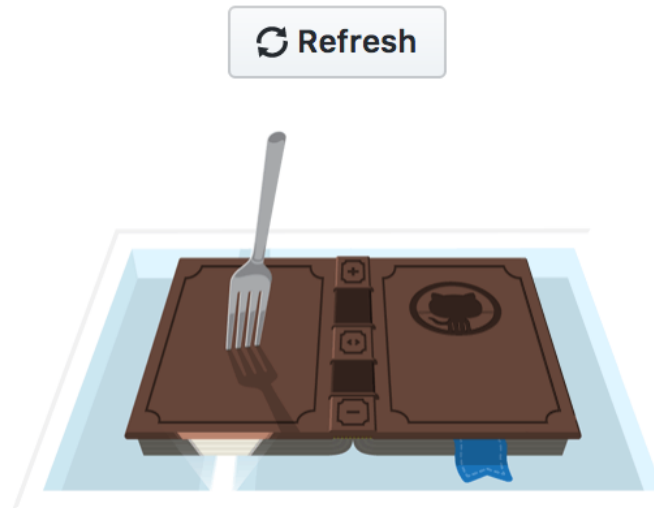


# Fork and Merge

---

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

It should only take a few seconds.



# Fork and Merge

 **ulgoon / 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](#).

 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

AA ▼ B i “ <> 🔗 ☰ ☷ ✓☰ ↶ @ 🚩

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

## 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](https://github.com/ulgoon/kingwangzzang1234).




**This branch has no conflicts with the base branch**

Only those with [write access](#) to this repository can merge pull requests.


# Fork and Merge



☐  1 Open ✓ 1 Closed

☐  **edit index.html**  
#2 opened 28 seconds ago by ulgoon

# 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](https://github.com/ulgoon/kingwangzzang1234).



**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
```

## Practice(3)

- 3~4인이 팀이 되어 프로젝트 수행
- 아래의 과제 중 하나 이상을 수행할 것
  - i. 100 prisoners problem
  - ii. 24 game
  - iii. Monty Hall Problem Simulation
- Process
  - 서비스 기획 -> backlog 작성(issue, projects) -> 개발 -> 평가 순으로 진행



# 1. 100 prisoners problem

## The Problem

- 100 prisoners are individually numbered 1 to 100  
A room having a cupboard of 100 opaque drawers numbered 1 to 100, that cannot be seen from outside.
- Cards numbered 1 to 100 are placed randomly, one to a drawer, and the drawers all closed; at the start.

- Prisoners start outside the room
  - They can decide some strategy before any enter the room.
  - Prisoners enter the room one by one, can open a drawer, inspect the card number in the drawer, then close the drawer.
  - A prisoner can open no more than 50 drawers.
  - A prisoner tries to find his own number.
  - A prisoner finding his own number is then held apart from the others.
  - If all 100 prisoners find their own numbers then they will all be pardoned. If any don't then all sentences stand.

## 2. 24 game

- Given any four digits in the range 1 to 9, which may have repetitions, Using just the +, -, \*, and / operators; and the possible use of brackets, (), show how to make an answer of 24.
- An answer of "q" will quit the game.
- An answer of "!" will generate a new set of four digits.
- Otherwise you are repeatedly asked for an expression until it evaluates to 24
- Note: you cannot form multiple digit numbers from the supplied digits, so an answer of  $12+12$  when given 1, 2, 2, and 1 would not be allowed.

### 3. Monty Hall Problem

- Suppose you're on a game show and you're given the choice of three doors.
- Behind one door is a car; behind the others, goats.
- The car and the goats were placed randomly behind the doors before the show.
- Run random simulations of the Monty Hall game. Show the effects of a strategy of the contestant always keeping his first guess so it can be contrasted with the strategy of the contestant always switching his guess.
- Simulate at least a thousand games using three doors for each strategy and show the results in such a way as to make it easy to compare the effects of each strategy.