



## 성신여대 오픈소스 팀 프로젝트 실습

**Summary:** Web 페이지를 프로젝트화 하여 서로 협업하면서 작업을 진행, 웹 프로젝트 소스를 서로 수정함으로 발생하는 문제를 강사와 함께 해결한다.



### 목표

- Webpage 프로젝트를 서로 협업하면서 작업을 진행.
- Markdown으로 구성된 페이지 수정 작업을 통해 git/github 사용법을 완벽 숙지한다.

### PDF 문서

- [다운로드 클릭](https://github.com/oss-sungshin/git-lecture/raw/master/pdf/mydoc.pdf)  
(<https://github.com/oss-sungshin/git-lecture/raw/master/pdf/mydoc.pdf>)

### 사용 언어

- 개발 프로젝트가 아닌 관계로 모든 작업은 Markdown을 기반으로 작성.
- Markdown 사용방법은 아래 사이트를 참고하세요.

- <https://guides.github.com/features/mastering-markdown>  
(<https://guides.github.com/features/mastering-markdown>)
- Markdown viewer 활용 <https://jbt.github.io/markdown-editor/>  
(<https://jbt.github.io/markdown-editor/>)

## 실습 진행 방법

- 모든 실습자들은 웹 프로젝트(<https://github.com/oss-sungshin/git-lecture> (<https://github.com/oss-sungshin/git-lecture>))의 소스를 수정하면서 발생하는 문제를 2명의 강사와 함께 해결한다.
- 실습 진행 도중 여러 이벤트(브랜치 작업, 개발 도중 머지) 등의 미션이 발생되며, 강사와 함께 문제점을 같이 해결한다.

## Git 내용 참고 사이트

- <https://git-scm.com/book/ko/v1> (<https://git-scm.com/book/ko/v1>)

## 2월 9일(목)시간표

- 10:00 ~ 11:50 : 실습 진행 방법 설명, 팀 빌딩, 실습 진행
- 11:00 : 브랜치 후 작업 진행 (강사 개별 지도)
- Event #1(11:20) : 1차 머지(팀원 push -> 팀장 pull request -> 서버)
  - 개별 브랜치 및 머지 충돌을 실습한다.
- Event #2(15:10) : 2차 머지(팀원 push -> 팀장 pull request -> 서버)
  - Stash 사용 방법을 실습한다.
- Event #3(16:00) : 커밋 가져오기
  - 최근 patch 파일 1개를 추출 후 각 팀에 전달(git format-patch -1)
  - 최근 oss-sungshin 커밋을 팀장 github 저장소에 적용하기(git cherry-pick)
- Event #4(16:30) : 팀별 커밋 내용 공유 및 설명(강사 진행)
  - 내용 공유

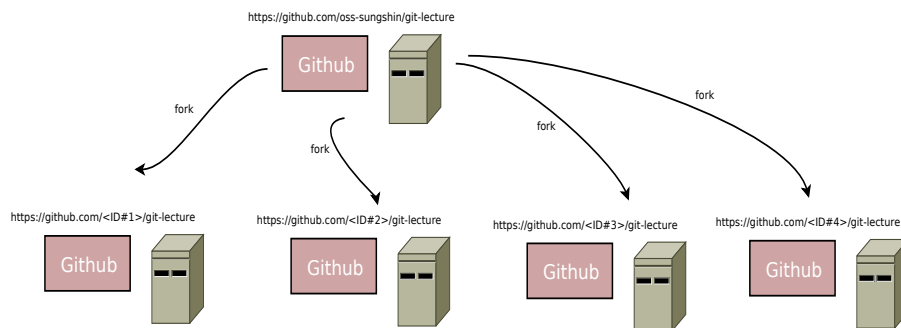
## 2월 10일(금) 시간표

- 13:00 ~ 15:50 : 실습 진행 및 시상

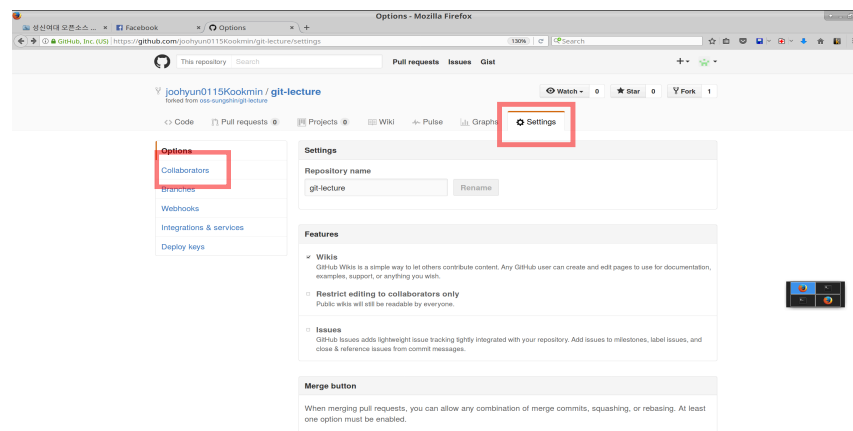
- Event #5(13:20) : 4차 머지(팀원 push -> 팀장 pull request -> 서버)
  - 머지 문제 해결
- Event #5(14:00) : reset 실습
  1. 현재 커밋 백업 (git format-patch -2)
  2. 과거 커밋으로 이동 (git reset XXXXX)
  3. 백업 커밋 적용 (git am 이용)
- Event #5(15:00) : 최종 머지 및 작업 내용 발표
  - 작업 한 내용 서로 공유
- Event #5(15:30) : 종합 및 시상식

## 팀장이 해야 할 일

- 자신의 github에 oss-sungshin 프로젝트 fork 하기



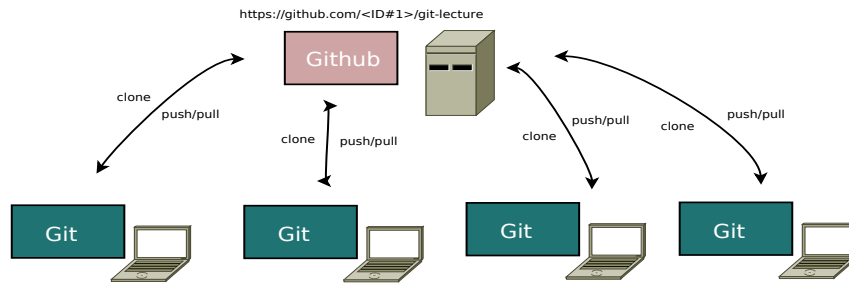
- 팀원 github 계정을 프로젝트에 등록하기



- 통합은 pull request를 사용하여 병합 할 예정

## 팀원이 해야 할 일

- 팀장의 github 프로젝트를 local PC에 clone 하기



## 팀별 결과 공유 및 시상

최종 결과물은 아래 페이지에 영구 보존되며, 우수 팀은 팀별 경품이 있습니다.

## 소스 트리 구조

**Summary:** 문서 소스 트리 구조에 대한 설명

## 소스 트리 구조

## 팀 빌딩

### Overview

실습 수업 진행을 위한 팀 빌딩: 한 조당 5명, 5분동안 정하세요.

team 빌딩 결과

팀명	조원이름
A	name, name, name, name and name
B	name, name, name, name and name
C	name, name, name, name and name
D	name, name, name, name and name

# 문서 구조

## 참고 자료

### Git Cheat Sheet

[Back - Command Quick Reference](#)

**Create**

```

From existing files
git init
git add .
git commit
From remote repository
git clone --old .../new
git clone git://...
git clone ssh://...
        
```

**Browse**

```

git status
git diff oldref newref
git log [-p] [file|dir]
git blame file
git show ref[:file]
git branch (shows list, * = current)
git tag -l (shows list)
        
```

**Change**

**Track Files**

```

git add files
git mv old new
git rm files
git rm --cached files
(Stop tracking but keep files in working dir)
        
```

**Revert**

```

In Git, reverting usually means adding a commit
that undoes changes in previous commits.
git reset --hard (NO UNDO)
(throw away all pending changes)
git revert ref
git commit -a --amend
(replace previous commit)
git checkout ref file
        
```

**Update**

```

git fetch (from default upstream)
git fetch ref
git pull (= fetch + merge)
git am -3 patch mbox
git apply patch.diff
        
```

**Branch**

```

git checkout branch
(switch working dir to branch)
git merge branch
(merge into current)
git branch branch
(branch current)
git checkout -b new other
(branch new from other and
switch to it)
        
```

**Record**

```

In Git, commit only respects changes that have
been marked explicitly with add.
git commit [-a]
(-a: add changed files automatically)
git push [remote]
(push to origin or remote)
git tag foo
(mark current version)
        
```

**Publish**

```

git push
git push remote
git format-patch origin
(create set of diffs)
        
```

**Resolve Conflicts**

```

Use add to mark files as resolved.
git diff [-base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
        
```

**Configuration**

Change options using `git config [--global] varname value`. The following variable names are useful:

- `core.bare` True for repositories without a working tree (usually public repositories).
- `core.sharedRepository` Set to group or all to make the repository contents writable for the file group or everybody.
- `core.compression` A zlib compression level for objects (0-9, 9 = best compression) or -1 to use zlib's default.
- `color.branch` Color-code list of branches (true = always, auto = only when outputting to a terminal)
- `color.diff` Color-code diffs (true, auto)
- `color.status` Color-code output of `git status` (true, auto).
- `user.email` Your e-mail address (used in commits).
- `user.name` Your name (used in commits).

**Object refs**

```

master default devel branch
origin default upstream branch
HEAD~ current branch
HEAD~ parent of HEAD
HEAD~4 great-great grandp. of HEAD
foo..bar from ref foo to ref bar
        
```

**Commit Messages**

Some of Git's viewing tools need commit messages in the following format:

A brief one-line summary  
 <blank line>  
 Details about the commit

There's a little bit of room for your own notes here. This is your chance to customize this cheat sheet!

---

This is version 2.0 of Jan Krüger's Git cheat sheet. You can contact the author by e-mail at: <jk@jk-gs>. Partially based on work by Zack Rusin, <http://zrusin.blogspot.com/>

## 문서구조

1. create (page 9)
2. browse (page 10)
3. branch (page 11)
4. track\_files (page 12)
5. record (page 13)
6. revert (page 14)
7. publish (page 15)
8. update (page 16)

- 9. [resolve\\_conflict](#) (page 17)
- 10. [useful\\_command](#) (page 18)



## git create-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git browse-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git branch-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git track-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git record-related command

**Summary:** Writing git manual

### Writing git manual!

Writing git manual!

## git revert-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git publish-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

## git update-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!



## git resolve-related command

**Summary:** Writing git manual

Writing git manual!

Writing git manual!

# git useful command

## Summary: Writing git manual

## git tags

## 태그 조회하기

git tag 명령으로 이미 만들어진 태그가 있는지 확인할 수 있다.

```
$ git tag
v0.1
v1.3
```

# Writing git manual!

## Writing git manual!

안녕하세요

- 리스트 입니다.

소스는 아래와 같이 출력하세요

```
#include <stdio.h>
```

이미지 링크는 아래와 같이 하세요

2L 0 2L 2L 0 2L 0 2

LOZL

## Sidebar Navigation

**Summary:** github 사용법에 대한 문서

github 사용법

# Travis CI 사용법

**Summary:** Travis CI 사용법에 대한 문서

## Travis CI 사용법

## Release notes 0.1

**Summary:** Version 0.1 of the Documentation

Relative links