

---

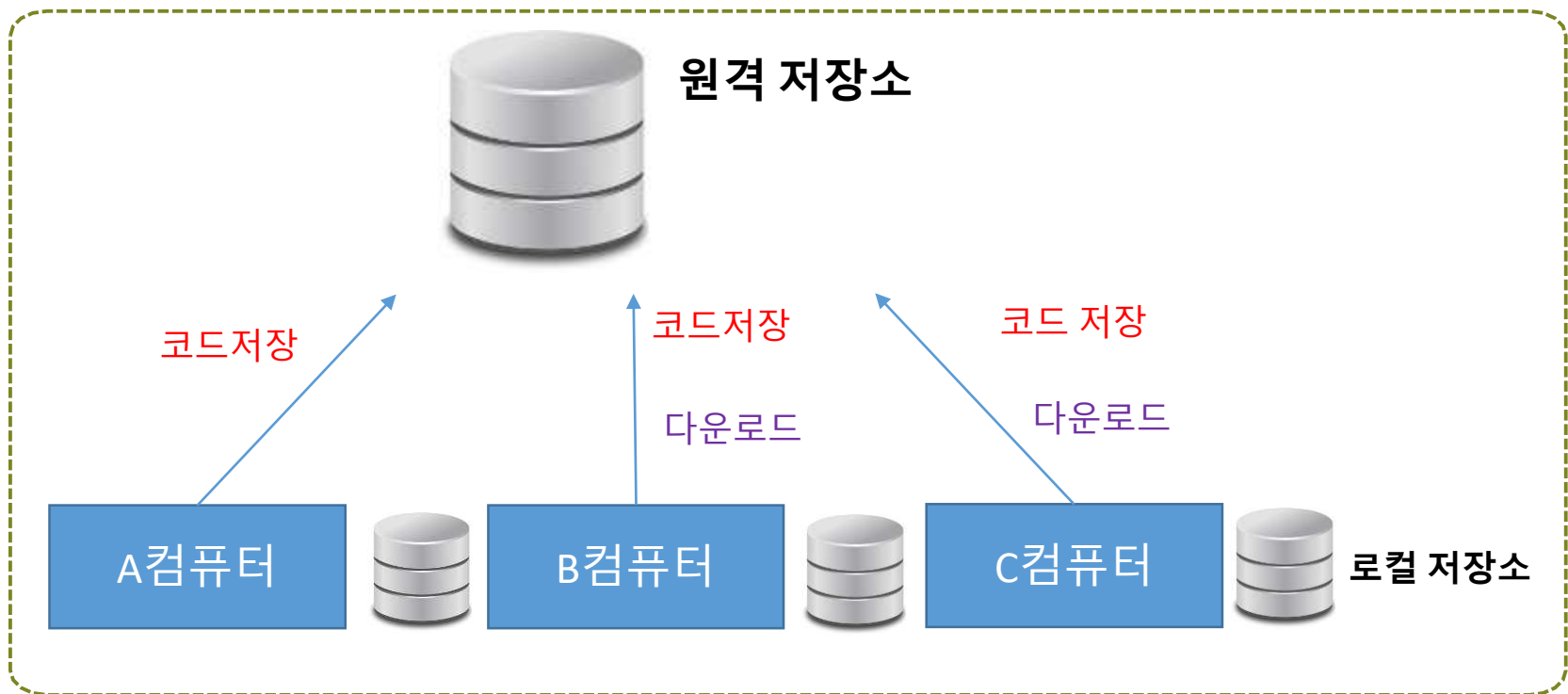
Chapter 1

# 깃 사용하기



# 깃 허브란?

- 깃(git)은 공통으로 관리하는 프로그램 소스코드 관리하기 위한 플랫폼 입니다.
- GitHub를 통해 사람들이 어디서나 프로젝트를 함께 할 수 있습니다.





# 깃 허브란?

카카오톡 2.0

카카오톡 1.2

카카오톡 1.1

카카오톡 1.0

버전별로 유 의미한 변화를 중심으로  
백업해 주는 과정이 필요하다.



혼자서 만들 수 있는가?



# 깃 설치 후 최초 설정

## 사용자 정보 설정

Git은 커밋 할 때마다, 현재 운영체제에 설정 된 아래 정보를 사용한다.  
혼자 사용하는 컴퓨터라면, 한번만 선언하면 되지만,  
매번 다른 환경이라면 사용자 정보를 설정한다.

```
$ git config --global user.name "본인이름영어로"  
$ git config --global user.email 본인이메일
```

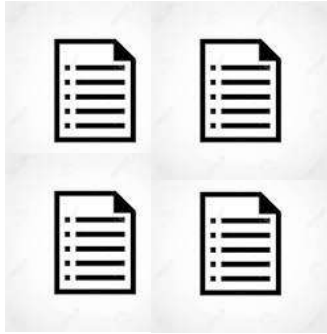
## 사용자 정보 설정 확인

```
$ git config --list
```

# 깃 허브란?

버전이 되기까지 거쳐가는 세 개의 로컬 공간

Working Directory  
(작업공간)



Staging area

Repository  
(실제 저장소)

카카오톡 2.0

카카오톡 1.0

모든 변경사항들이 담기는 공간

변경 사항들 중 후보들이 올라가는 공간

# 깃 명령어

모든 깃 명령어는 시작 되기 전에  
해당 폴더안에서 버전관리를 시작해야 한다!

`git init` :깃 시작

`git status` :현재 깃의 상태

`git add 파일명` :해당 파일을 staging area에 추가

`git rm --cached 파일명` :해당 파일을 staging area에 삭제

`git add .` :파일을 전부다 추가

`git commit -m "메시지"` :깃을 레파지토리에 저장

`git log` :커밋 이후 변경 로그를 확인

파일의 수정이 일어난다면 add부터 다시 진행

여기 까지가 내 컴퓨터의 로컬저장소에 저장하는 명령어 이다.

여기까지 실습

`git commit -am "메시지"` : 추가와 커밋 을 동시에 한다

# 깃 명령어

깃을 원격저장소에 올리는 작업

`git remote add origin 본인의깃계정` :원격저장소를 추가한다

`git push origin master` :원격저장소에 저장한다

여기까지 실습

기존에 해당 컴퓨터에서 다른 깃 계정을 사용하던 경우 에러가 발생한다.  
이때는 window자격 증명관리에서 깃관련 자격증명 관리를 삭제한다

제어판 홈

## 자격 증명 관리

웹 사이트, 연결된 응용 프로그램 및 네트워크에 대해 저장된 로그인 정보를 보고 삭제합니다.



웹 자격 증명



Windows 자격 증명

자격 증명 편집(E) | 자격 증명 복원(R)

Windows 자격 증명

Windows 자격 증명 추가

Windows 자격 증명이 없습니다.

인증서 기반 자격 증명

인증서 기반 자격 증명 추가

인증서가 없습니다.

일반 자격 증명

일반 자격 증명 추가

git:https://github.com

수정된 날짜: 오늘

virtualapp:didlogical

수정된 날짜: 2020-05-04

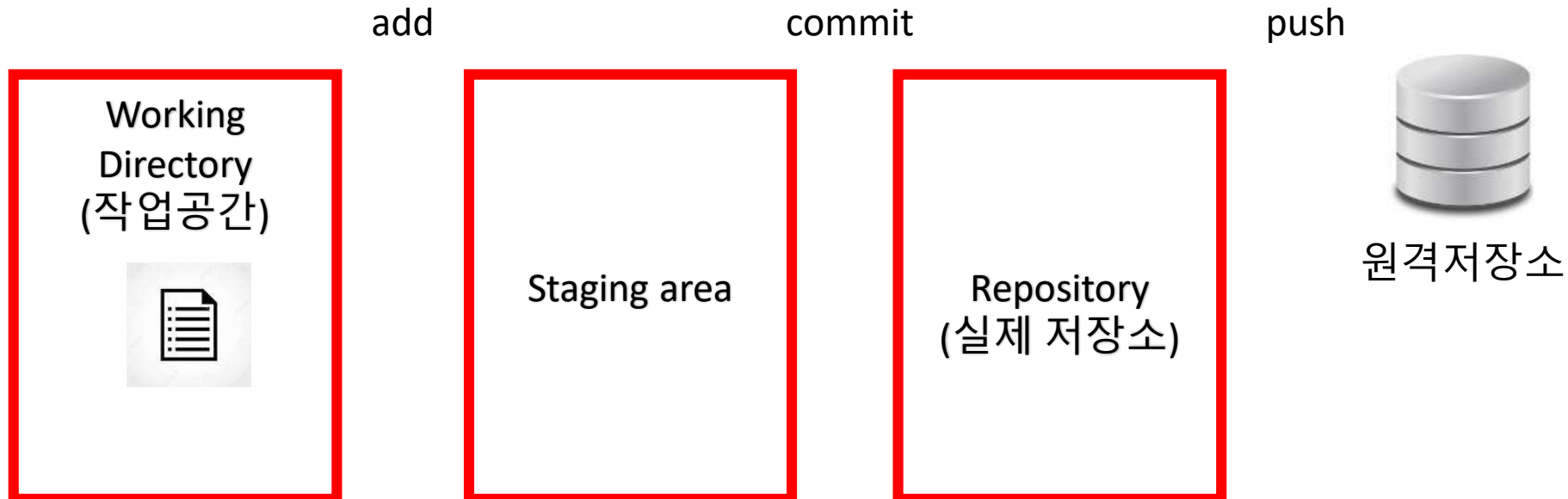
SSO\_POP\_Device

수정된 날짜: 오늘

# 깃 명령어(local repository)

깃 조회, 되돌리기 (working directory, staging area, repository를 되돌린다)

<code>git reset --hard HEAD^</code>	:커밋을 한단계 앞으로 되돌린다(모두 초기화)
<code>git reset --mixed HEAD^</code>	:커밋을 한단계 앞으로 되돌린다(working directory는 유지)
<code>git reset --soft HEAD^</code>	:커밋을 한단계 앞으로 되돌린다(working directory , staging area를 유지)







# 깃 명령어(local repository)

여기까지 현재상태를 맞추어 봅니다

## Working Directory

working dir

## Staging area

add staging

## Repository

second commit

first commit



# 깃 명령어(local repository)

`git reset --hard HEAD^`

모두 리셋이 된다

Working Directory

Staging area

Repository

working dir

add staging

second commit

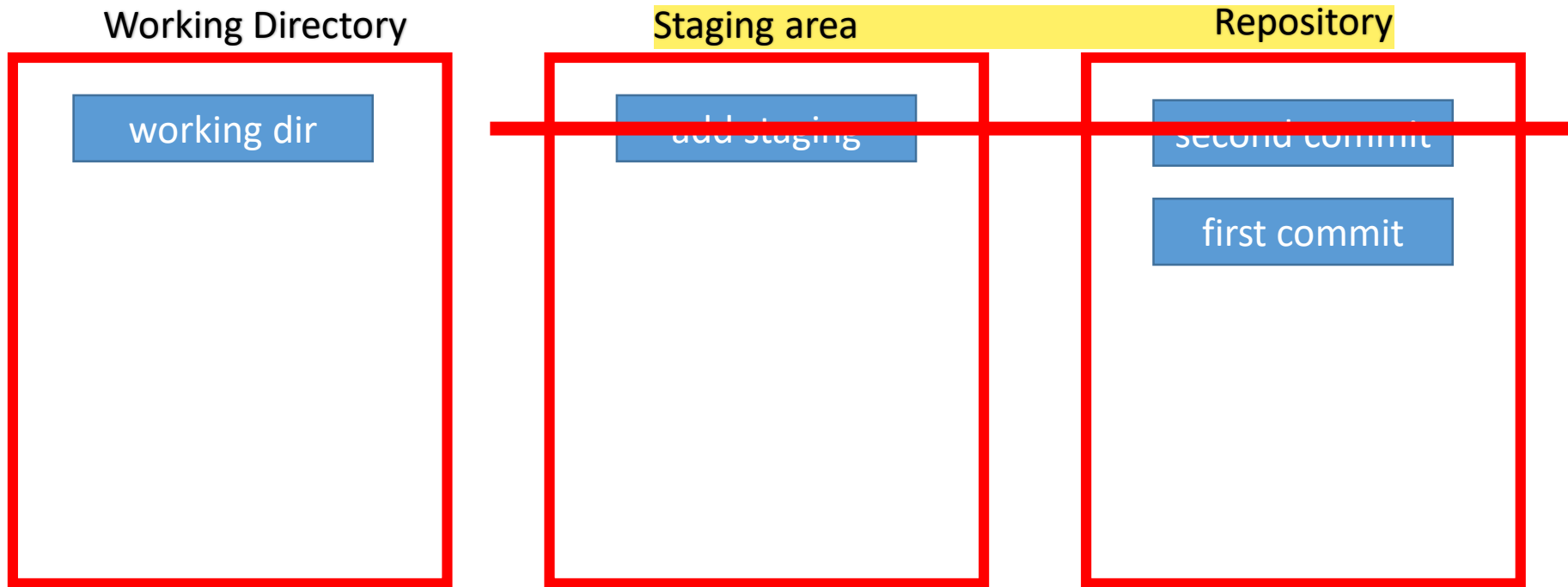
first commit



# 깃 명령어(local repository)

`git reset --mixed HEAD^`

내가 수정하고 있는 파일은 변함이 없다  
reset의 기본 default 옵션 입니다.



# 깃 명령어(local repository)

`git reset --soft HEAD^`

내가 수정하고 있는 파일은 변함이 없다  
staging area에도 변함이 없다

Working Directory

Staging area

Repository

working dir

add staging

~~second commit~~

first commit

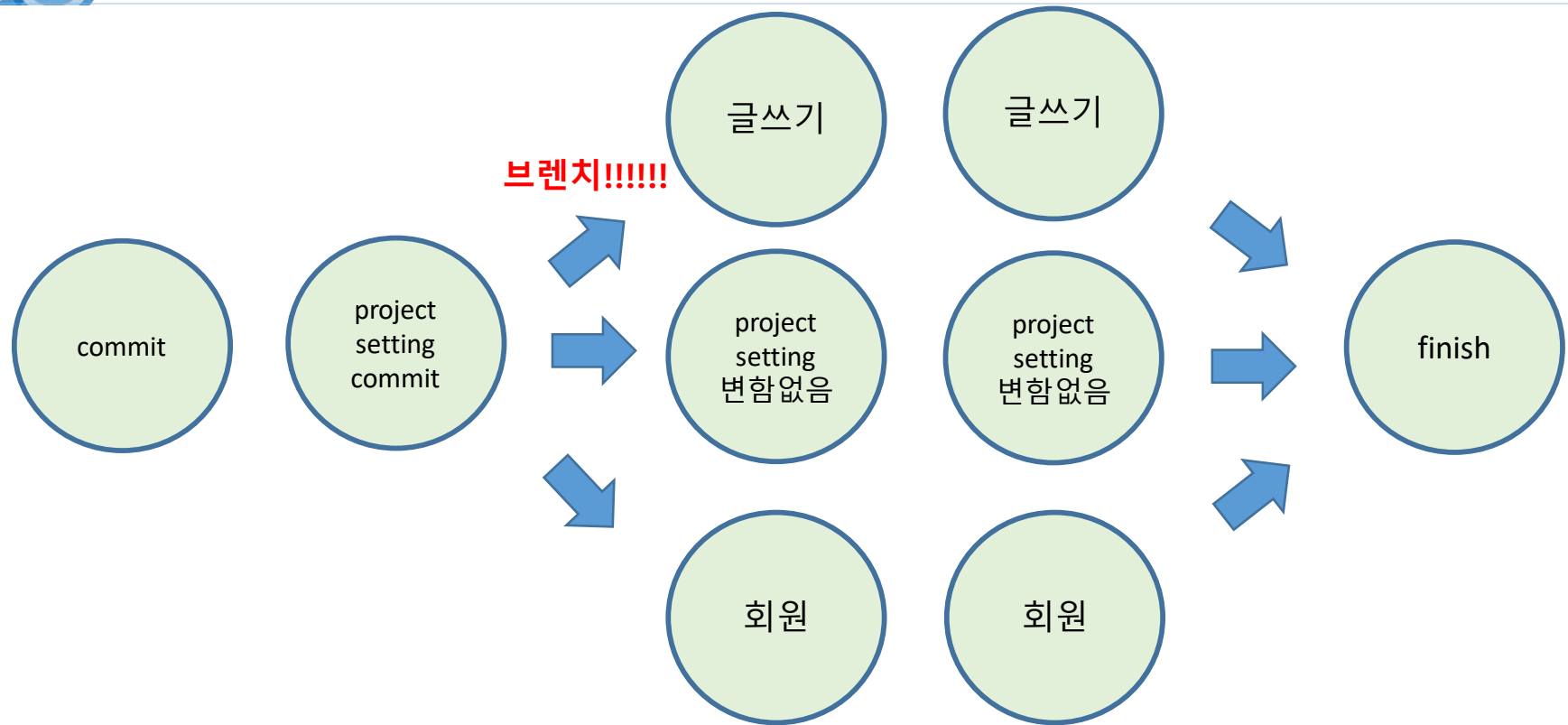
# 깃 명령어

프로젝트 개발에 협업은 필수 인가? 협업 시나리오????

아래와 같은 시나리오 라면????



# 깃 명령어



브랜치를 나누기 전에 최소한 한번이라도 **commit**이 이루어 져야함(프로젝트 초기세팅)  
깃 나누기 branch

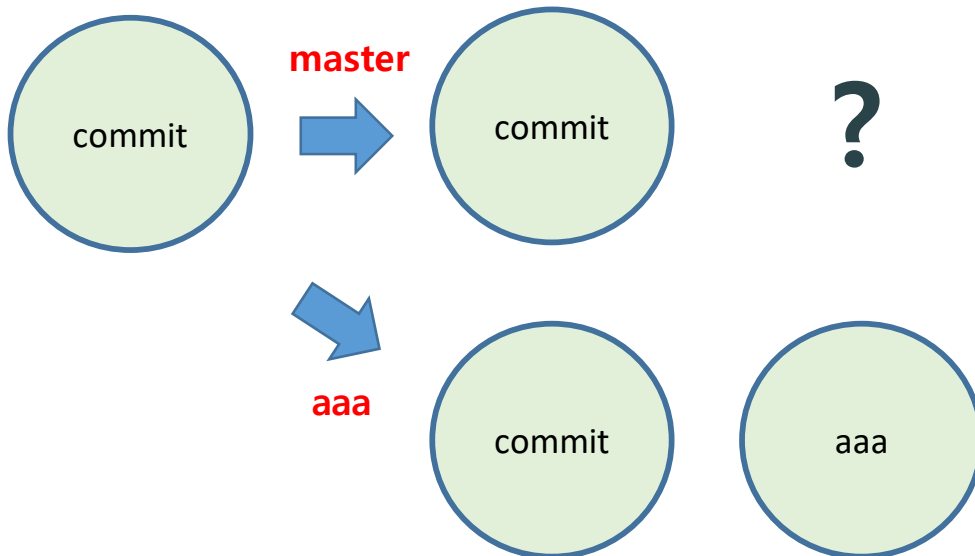
git branch 이름	:브랜치 생성
git branch	:브랜치 확인
git checkout 이름	:브랜치 변경
git branch -D 이름	:브랜치 삭제

git merge 합치는브랜치명	:병합
-------------------	-----

# 깃 명령어

실습

```
git branch aaa  
git checkout aaa  
파일생성  
git add .  
git commit -m "aaa가 생성"
```



**master브랜치에 aaa가 생성한 결과를 합치고 싶으면?  
합치고 싶은 브랜치에 들어가서 merge명령문을 사용한다**

실습

```
git checkout master  
git merge aaa
```

# 깃 명령어

원격저장소와 상호작용(실질적인 협업) – 원격저장소는 또다른 repository일 뿐입니다

원격저장소에 조회(추가)하기	remote
원격저장소에 밀어넣기	push
원격저장소에서 얻어와서 합치기	pull
원격저장소에서 얻기	fetch
원격저장소에 복사하기	clone

1. 원격저장소에 조회(추가)하기

**\$ git remote add <단축이름> 주소**

git remote -v :깃의 현재 원격 저장소를 확인한다  
git remote remove 단축이름:깃의 현재 원격 저장소를 삭제한다

2. 원격저장소에 조회(추가)하기

(나의 레파지토리의 master브랜치를 origin의 master브랜치로 push해라)

**\$ git push origin master**

3. 원격저장소에서 얻어와서 합치기

(origin을 나의 레파지토리 master브랜치를 갖고와라)

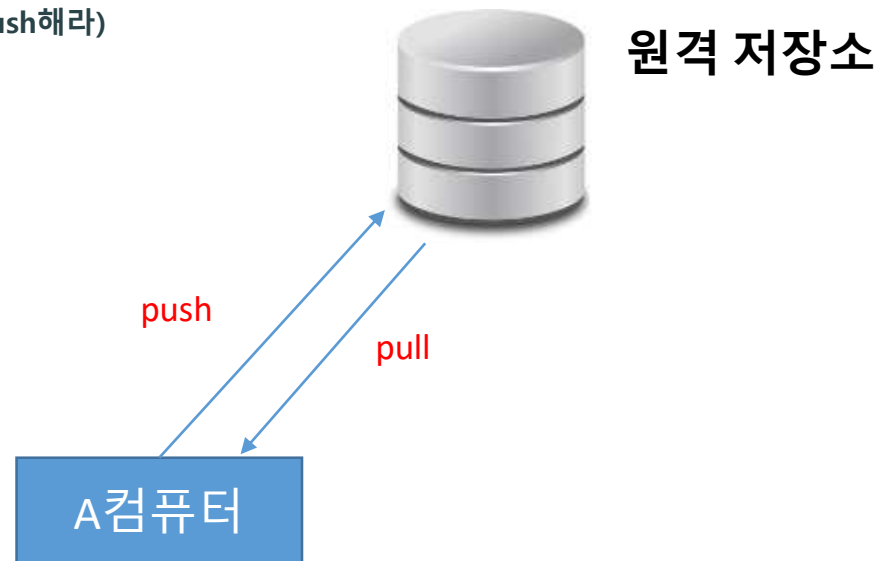
**\$ git pull (origin master)**

4. 원격 저장소에서 얻기

**\$ git fetch (origin master)**

5. 원격 저장소에서 복사하기

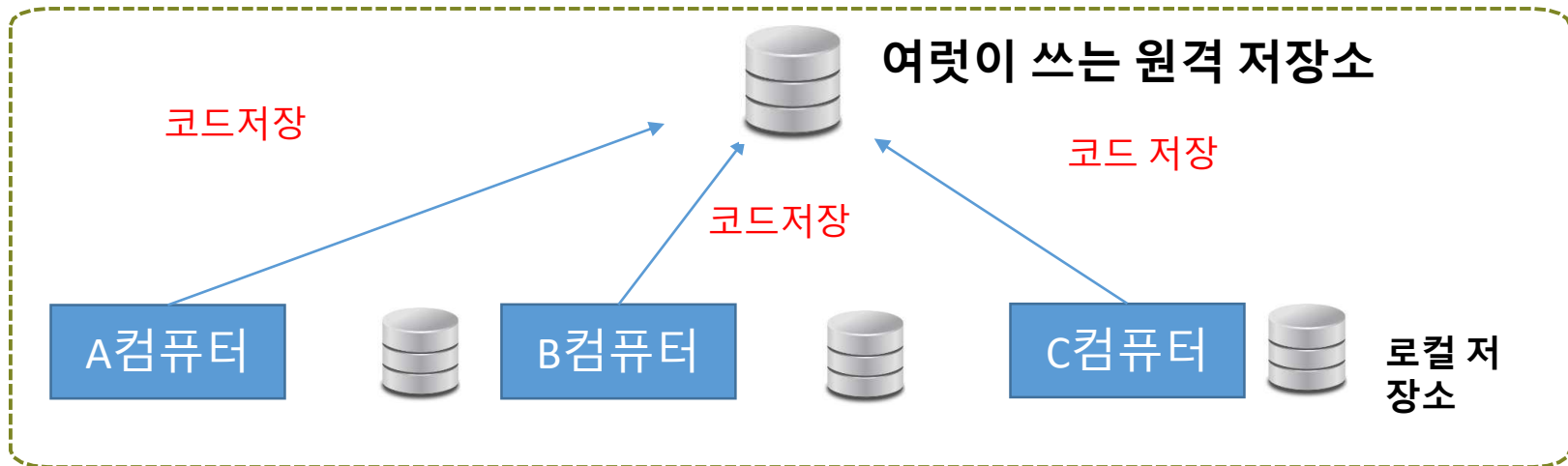
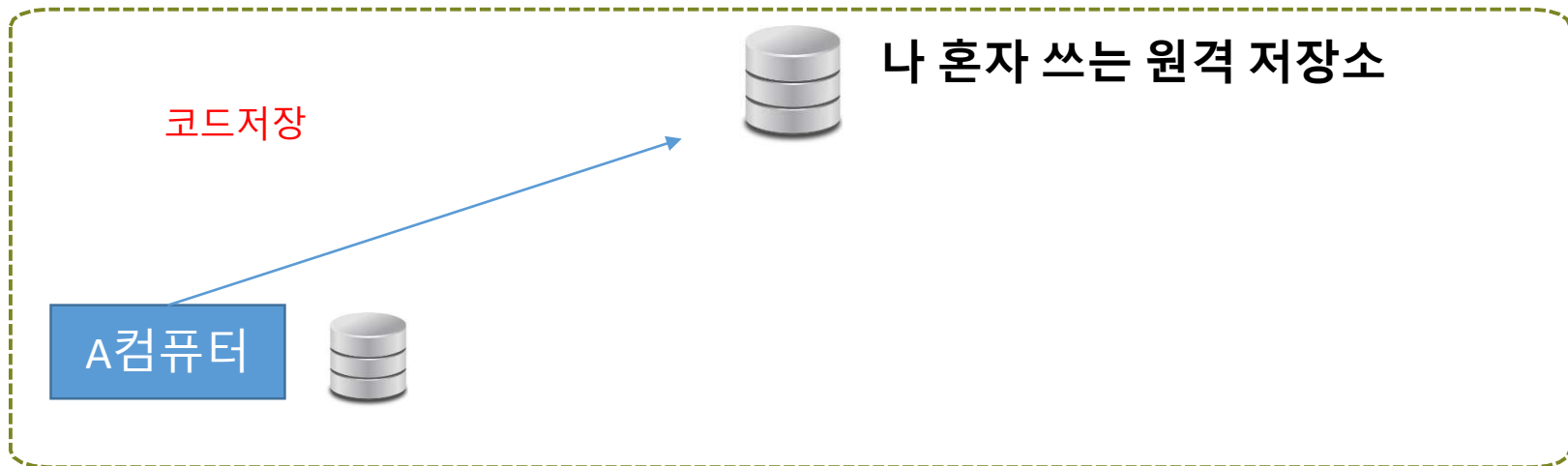
**\$ git clone 주소**





# 깃 명령어

원격저장소와 상호작용(협업시나리오)



# 깃 명령어

원격저장소와 상호작용(협업 시나리오)

내 로컬 저장소는 변했는데 원격 저장소는 변함 없는 경우

내 로컬 저장소는 변함 없는데 원격 저장소는 변한 경우

내 로컬 저장소도 변했는데 원격 저장소도 변한 경우

← git push origin master 사용

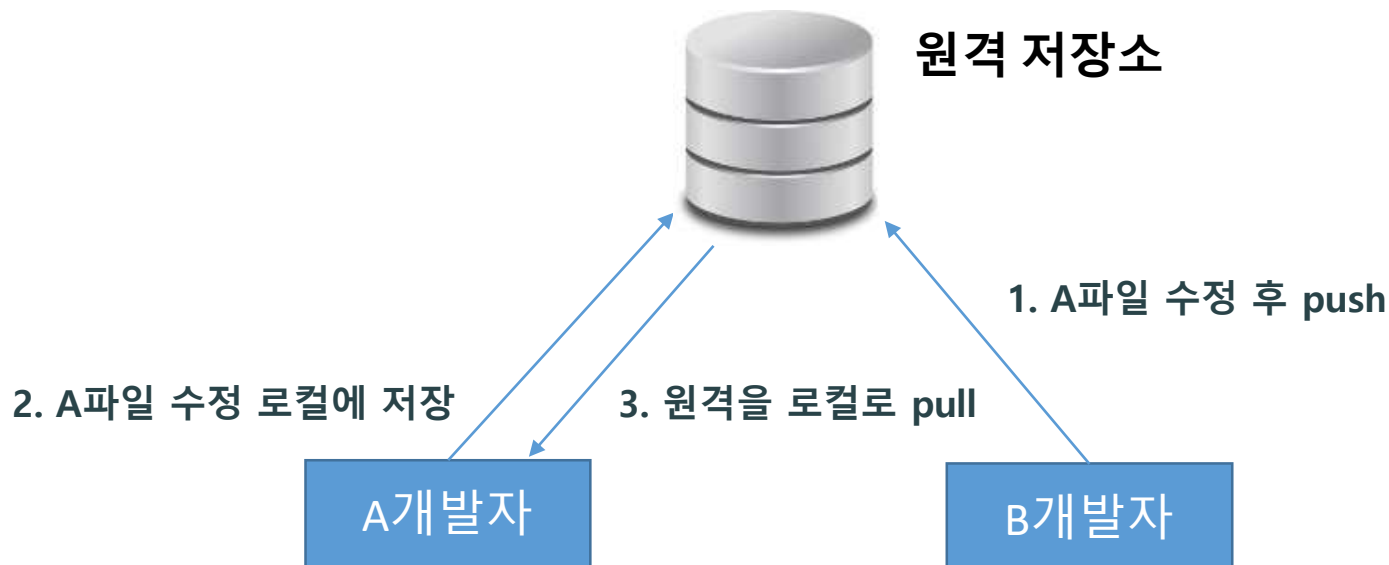
← git pull origin master 사용

← conflict 시나리오  
collaborator 사용

or

pull request 사용

conflict 시나리오



git: 누구 파일이 맞는 거야? 나는 모르겠다~ (conflict 발생)

# 깃 명령어

## conflict해소절차

- > 같은 파일을 동시에 수정할 때 발생한다.
- > 1. pull을 당겨오면 conflict가 발생한다 (git status로 확인가능)
- > 2. 일단 내가 쓰던 코드를 add -> commit한다(commit을 안하는 경우 단순 에러만 발생)
- > 3. 다시 한번 pull을 한다 (conflict발생)
- > 4. conflict 파일 목록을 확인하고 파일을 확인하고 코드를 수정한다
- > 5. 다시 한번 add -> commit 후 push한다

```
28: public void setBoolean(boolean value)
29: {
30: <<<<<< HEAD
31: .....
32: * 추후 플러그 변경시 이벤트 발생 기능 추가 필요
33: .....
34: .....
35: .....
36: * 플러그 변수값 변경
37: .....
38: >>>>>> branch 'master' of http://almdemo.cuvvc.com:48081/sampleapp
39: this.flagValue = value;
40: }
```

## conflict이 발생한 부분을

<<<<<<

내가쓴코드

=====

남이쓴코드

>>>>>>

로 표시한다

적절하게 알맞은 코드로 수정 후

특수문자를 제거후 commit을 진행 하면 됩니다

# 깃 명령어

## pull request

-> 1.메인 원격저장소에서 fork하기

pull request란 내가 복사해온 코드를 원본이 되는 원격저장소에 반영해 달라는 요청이다

-> 2.fork해온 곳에서 clone하기(중요)

-> 3.브랜치생성후 작성하고자 하는 코드(add , commit)작성

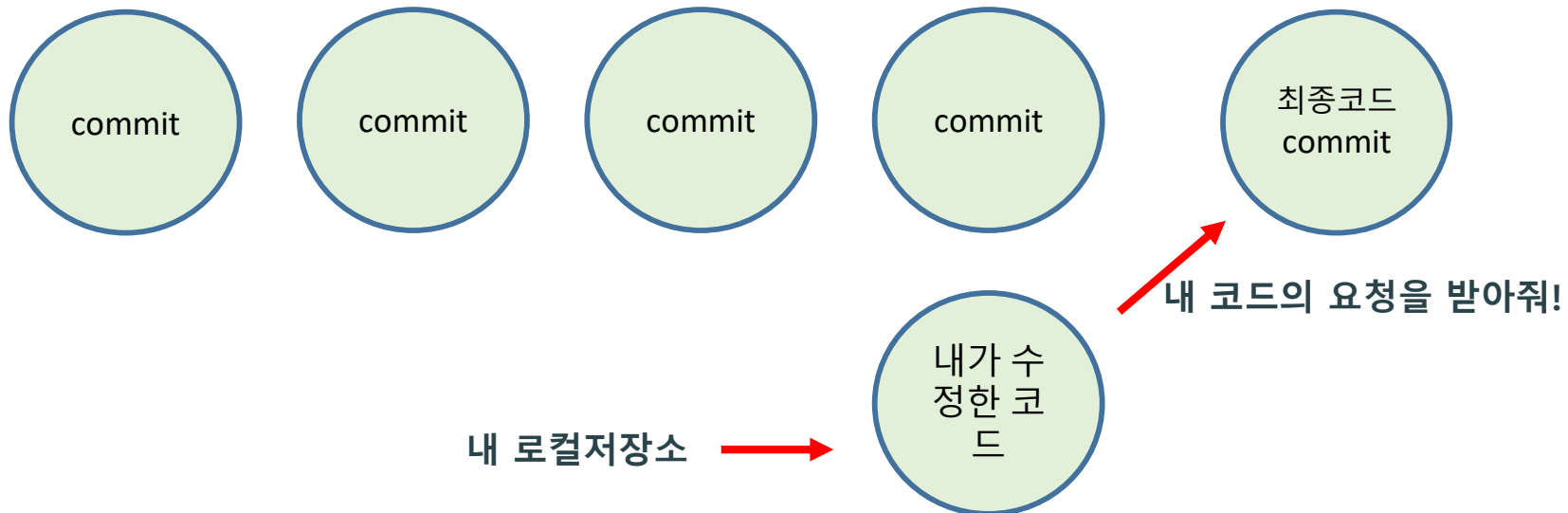
-> 4.브랜치에 있는 내용을 origin으로 push해라 (git push origin 브랜치명)

-> 5.fork계정으로 들어가면 풀리퀘스트를 날릴 수 있다.

-> 6.본계정(관리자)는 풀리퀘스트를 확인하고 merge를 받아준다

-> 프로젝트완료까지 4번부터 반복한다

## pull request시나리오





# 깃 명령어

관리자의 git이 업데이트 되어갈때 중간중간, 부계정의 git도 업데이트 시켜준다

pull request

-> remote를 추가한다 (git remote add collabo 주소)

-> pull을 당긴다 (git pull collabo master)(콜라보의 마스터 브랜치를 당긴다)

1. 자동으로 merge창이 열리는 경우

-> merge창이 자동으로 열리기도 한다(i로 수정하고 :q로 종료) - merge완료

2. pull이 거절되어 가져오지 못하는 경우

-> 2. FETCH\_HEAD로 진입(git checkout FETCH\_HEAD)

-> 새로운 브랜치로 따준다 (git switch -c me)

-> 파일을 조금 수정하고 add commit한다

-> master로 이동해서 merge한다 (git merge me)

-> 브랜치를 지워준다 (git branch -D 브랜치명)

# 유용한 깃 명령어

## 유용한 깃 사용법

### stash 시나리오

커밋없이 막 코드를 작성하다가~ 원격에 업데이트가 올라온 경우 **comflict없이** 빠르게 pull하고 local에서 merge한다.

#### 로컬 저장소에서 하던 일 잠시 되돌리기

- **git stash** 란?아직 마무리하지 않은 작업을 안전하게 임시 저장할 수 있도록 하는 명령어이다.
- 불필요한 **commit**을 피할 수 있다. (branch이동시)
- 이를 통해 아직 완료하지 않은 일을 **commit하지 않고** 나중에 다시 꺼내 마무리할 수 있다.
- **pull 명령시 comflict가 날 때 유용하게 사용하는 명령**

#### 1. 하던 작업 임시로 되돌리기

**\$ git stash**

#### 2. stash목록 확인하기

**\$ git stash list**

#### 3. stash 적용하기(하던 작업 다시 가져오기)

**\$ git stash apply [stash이름]**

#### 4. stash 제거하기

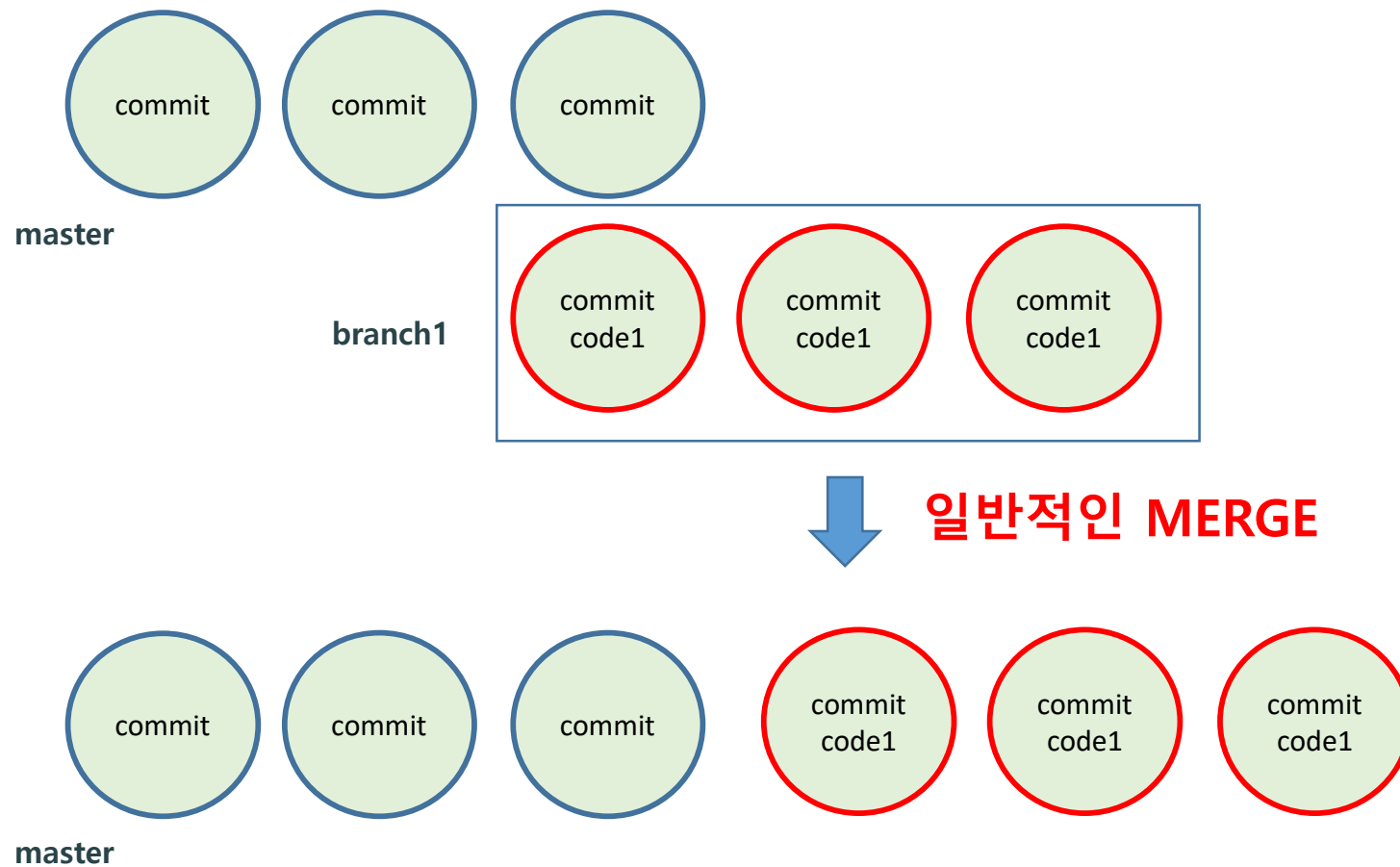
**\$ git stash drop [stash이름]**

#### 5. stash 전부 제거하기

**\$ git stash clear**

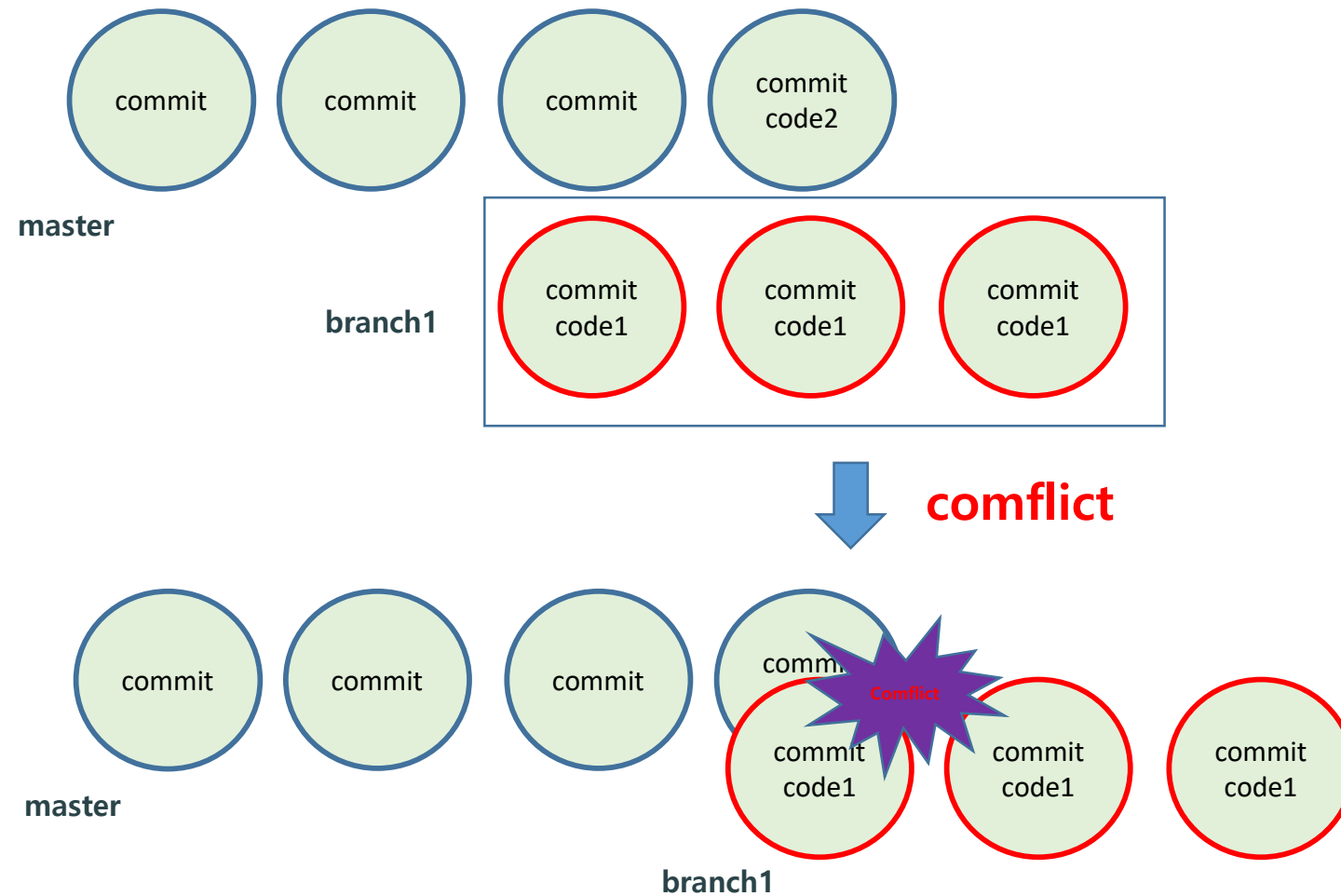
# 깃 명령어

## MERGE와 Conflict



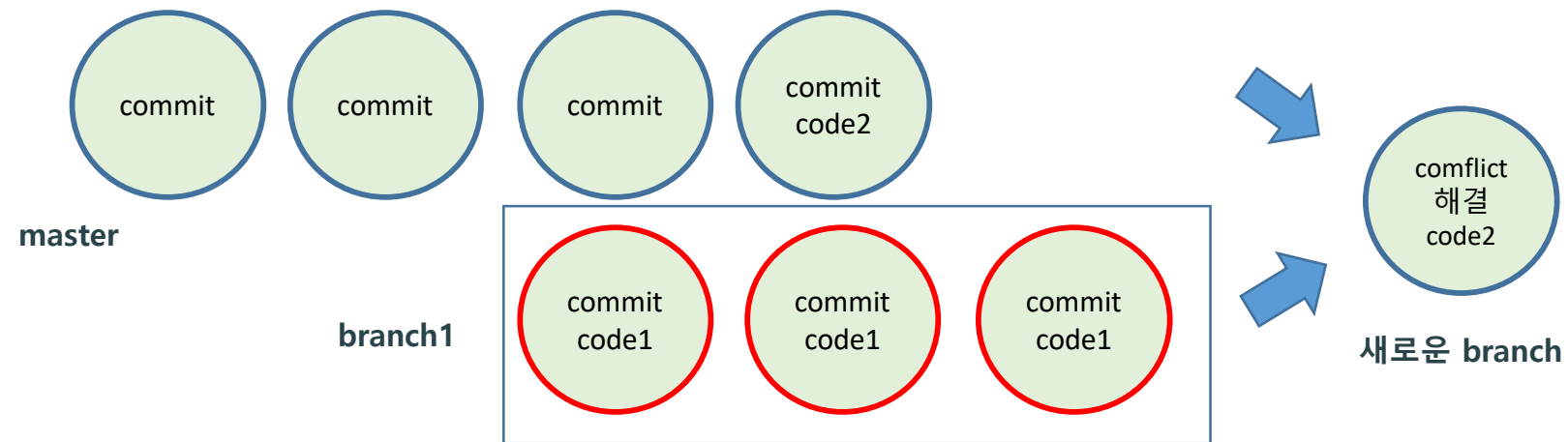
# 깃 명령어

## MERGE와 Conflict



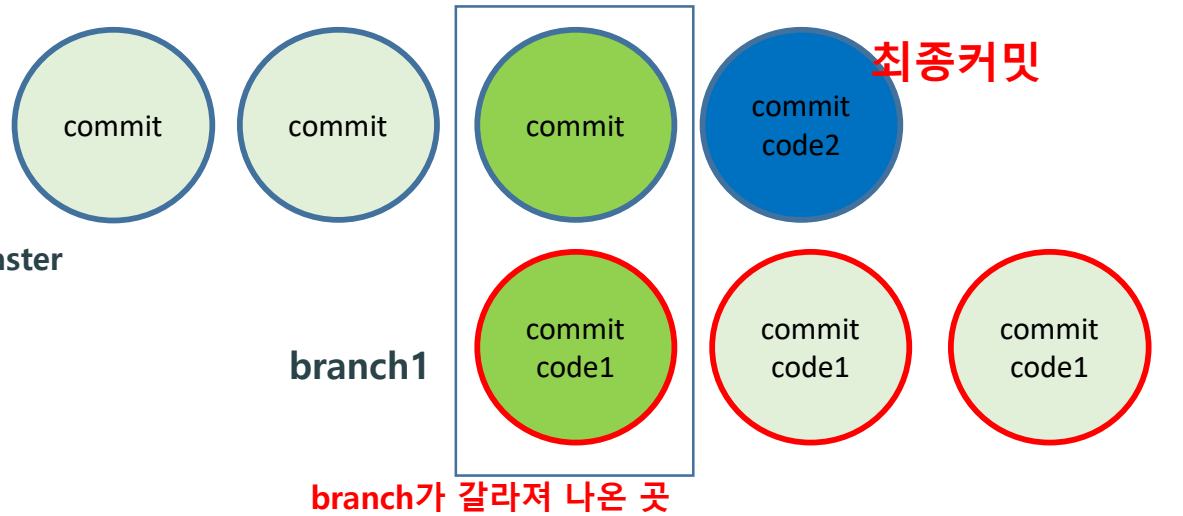


## MERGE와 Conflict

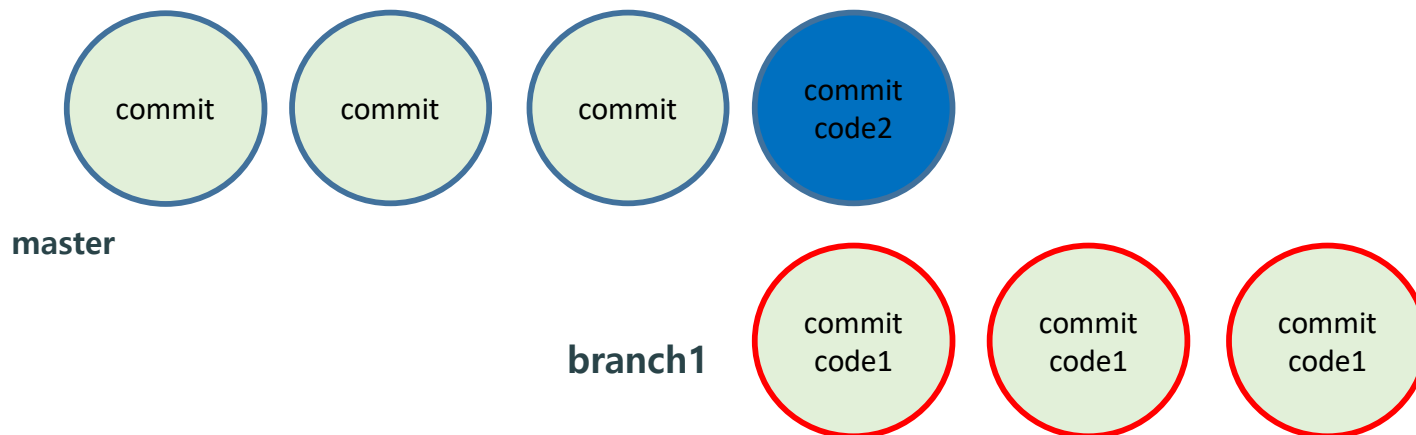


# 깃 명령어

**rebase** - branch의 base가 되는 최신 커밋으로 base를 맞추는 작업



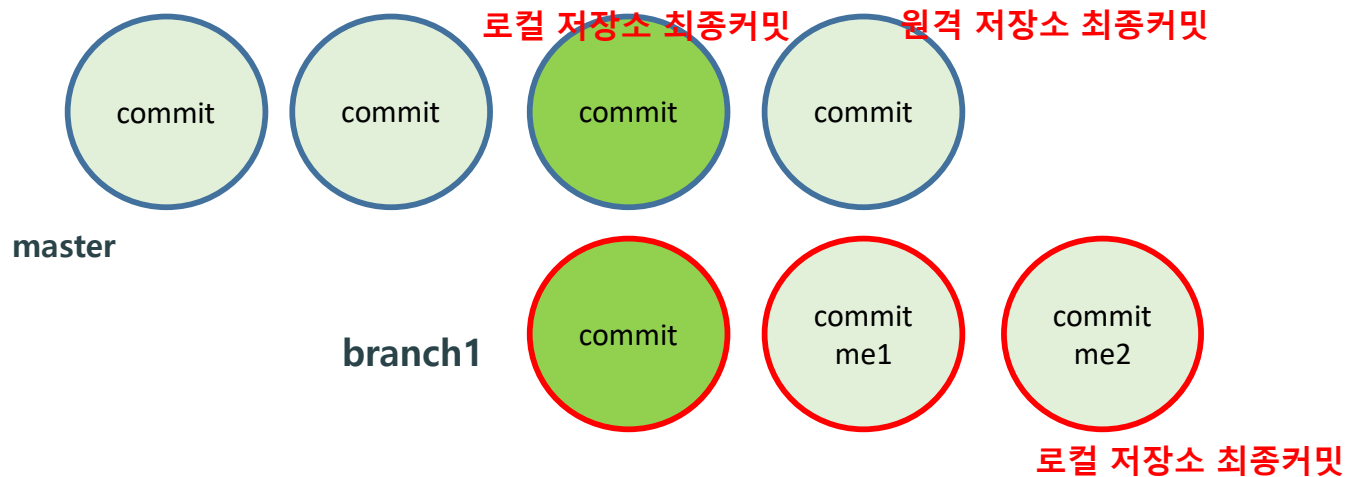
rebase 실행시



# 깃 명령어

## rebase실습

- 1.git\_rebase원격저장소 생성
- 2.t1커밋, t2커밋 추가
- 3.원격저장소에 푸시
- 4.브랜치 생성 (me)
- 5.b1커밋, b2커밋 추가
- 6.이 상황에서 원격저장소 새 파일추가



이런 경우 master브랜치에서 pull을 안전하게 당겨주고 me브랜치에서 rebase를 실행한다



# 깃 명령어

---


수고하셨습니다

# 이클립스에서 깃 사용하기



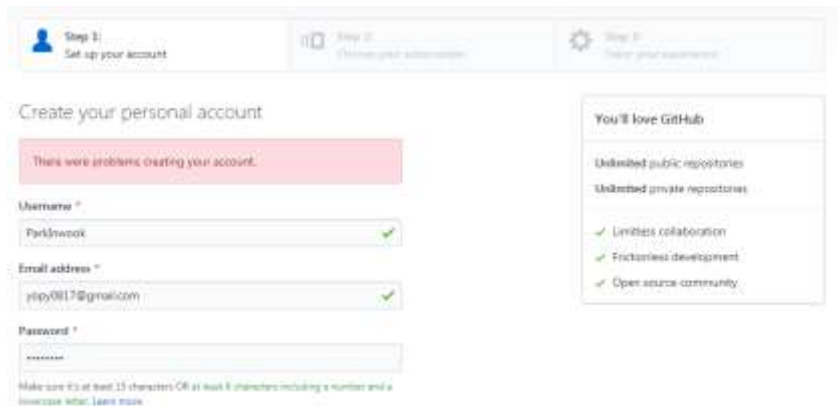
# 깃 허브 가입하기

<https://github.com/>



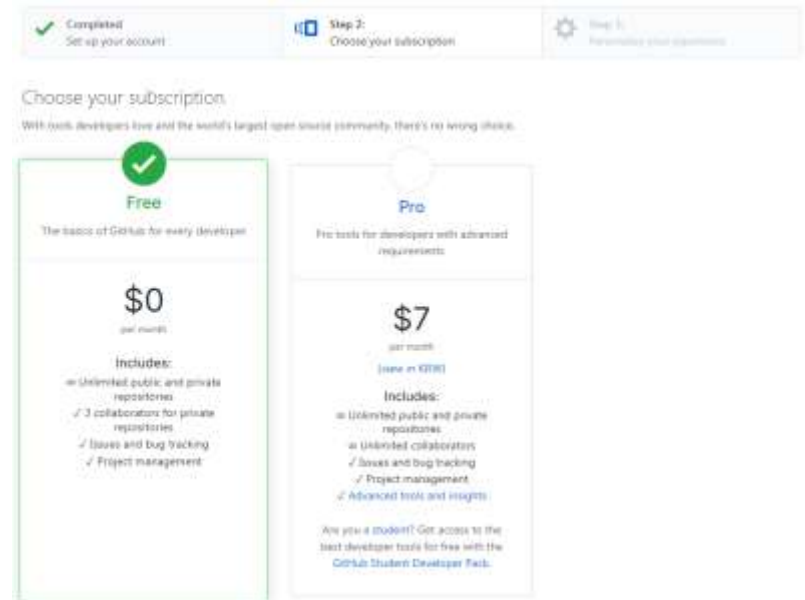
The image shows the GitHub homepage with a sign-up form. The form has three input fields: "Pick a username", "Your email", and "Create a password". Below these fields is a green button labeled "Sign up for GitHub". To the left of the form, the text "Where software is built" is displayed, followed by a description of GitHub's features and a note that private plans start at \$7/mo.

sign up클릭



The image shows the GitHub account creation form. It has three steps: Step 1: Set up your account, Step 2: Choose your subscription, and Step 3: Personalize your experience. The first step is active. The form has a red error message: "There were problems creating your account." Below this, there are input fields for "Username" (ParkInwook), "Email address" (yipy0817@gmail.com), and "Password". To the right, there is a section titled "You'll love GitHub" with a list of features: Unlimited public repositories, Unlimited private repositories, Limitless collaboration, Frictionless development, and Open source community.

이름 이메일 비밀번호 작성 후  
다음 클릭



The image shows the GitHub subscription selection screen. It has two options: "Free" and "Pro". The "Free" option is selected, indicated by a green checkmark. The "Free" option is labeled "The basis of GitHub for every developer" and has a price of "\$0 per month". It includes: Unlimited public and private repositories, 3 collaborators for private repositories, Issues and bug tracking, and Project management. The "Pro" option is labeled "Pro tools for developers with advanced requirements" and has a price of "\$7 per month (same as GDR)". It includes: Unlimited public and private repositories, Unlimited collaborators, Issues and bug tracking, Project management, and Advanced tools and insights. Below the "Pro" option, there is a note: "Are you a student? Get access to the best developer tools for free with the GitHub Student Developer Pack."

free로 지정하고 다음을 누르면  
작성한 이메일로 메일이 옵니다.  
확인후 start a project를 클릭하세요


# 깃 허브 가입하기

-원하는 Repository(저장소) 이름을 입력하고 public으로 생성합니다.  
private은 유료입니다.

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 jwgye ▾

Repository name

Great repository names are short and memorable. Need inspiration? How about **solid-potato**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

# 깃 허브 가입하기

- 완성된 repository 입니다. https주소 URL로 이클립스와 연동해 사용할 수 있습니다

ParkInwook / MyProject

Unwatch 1 Star 0 Fork 0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Security

Insights

Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop

 or 

HTTPS

SSH

https://github.com/ParkInwook/MyProject.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# MyProject" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/ParkInwook/MyProject.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/ParkInwook/MyProject.git
git push -u origin master
```

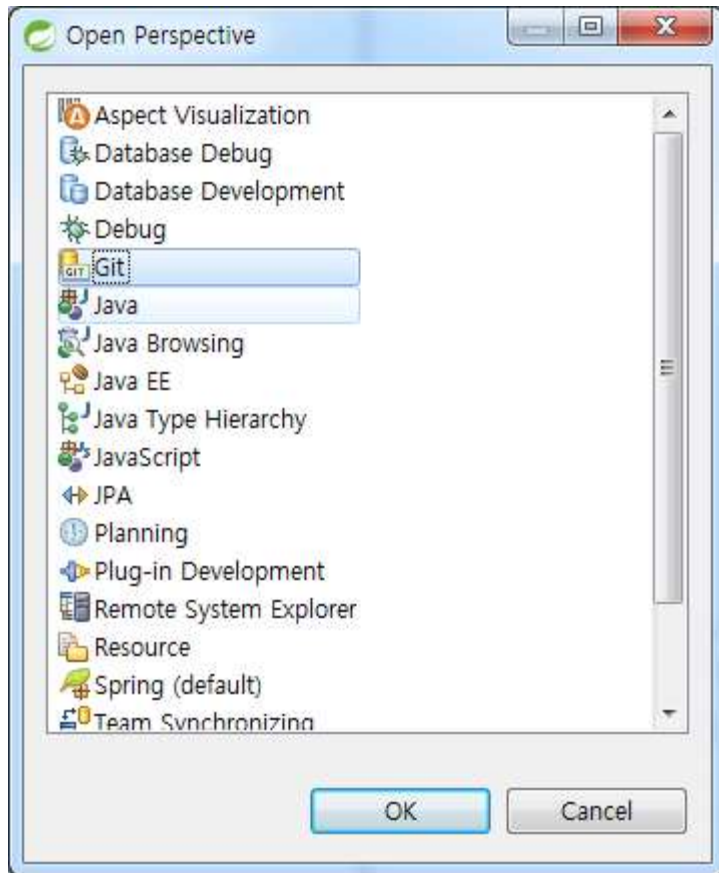
...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code



- STS(스프링) 을 설치했다면 별도의 다운로드 없이 곧바로 사용할 수 있습니다.
- 우측 상단 **Perspective**에서 **Git**을 선택합니다.

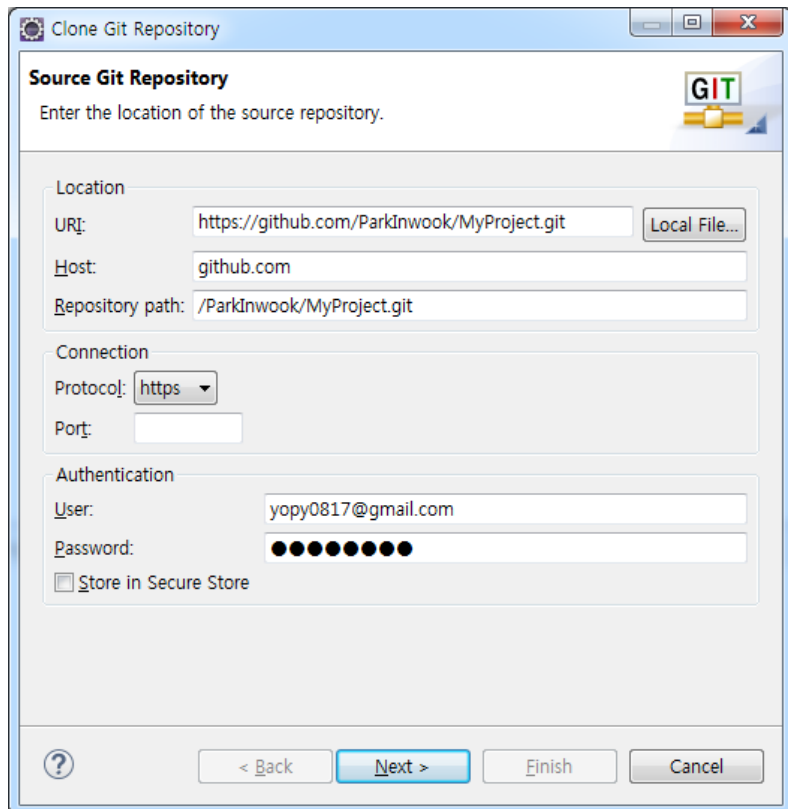


Select one of the following to add a repository to this view:

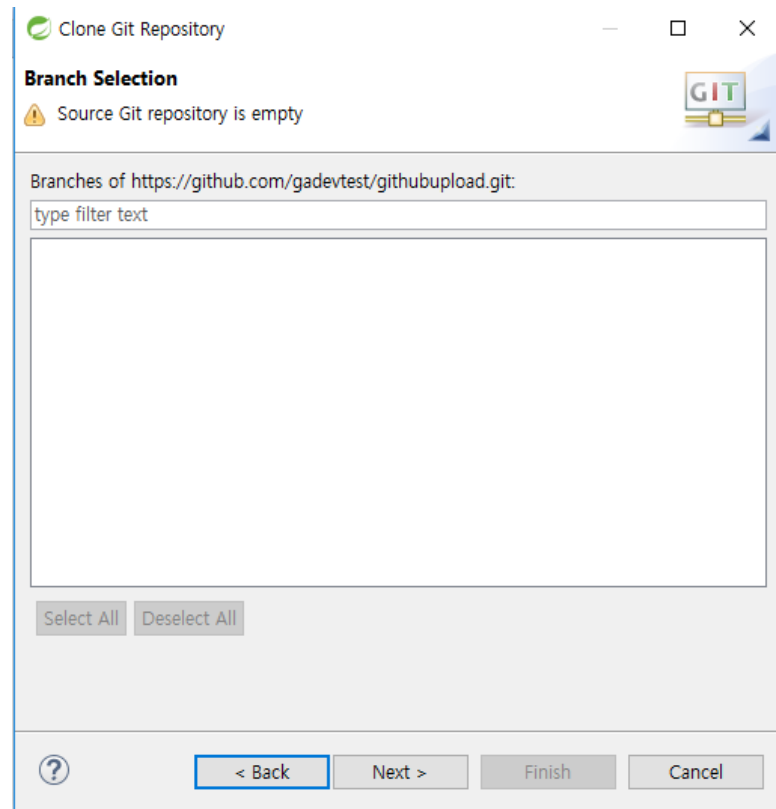
-  [Add an existing local Git repository](#)
-  [Clone a Git repository](#)
-  [Create a new local Git repository](#)

-3개의 항목중 **Clone a Git repository** 선택

# 깃 이클립스에 연동하기

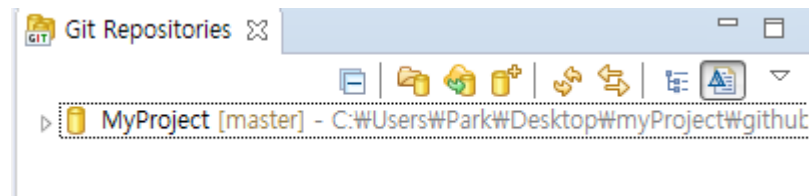
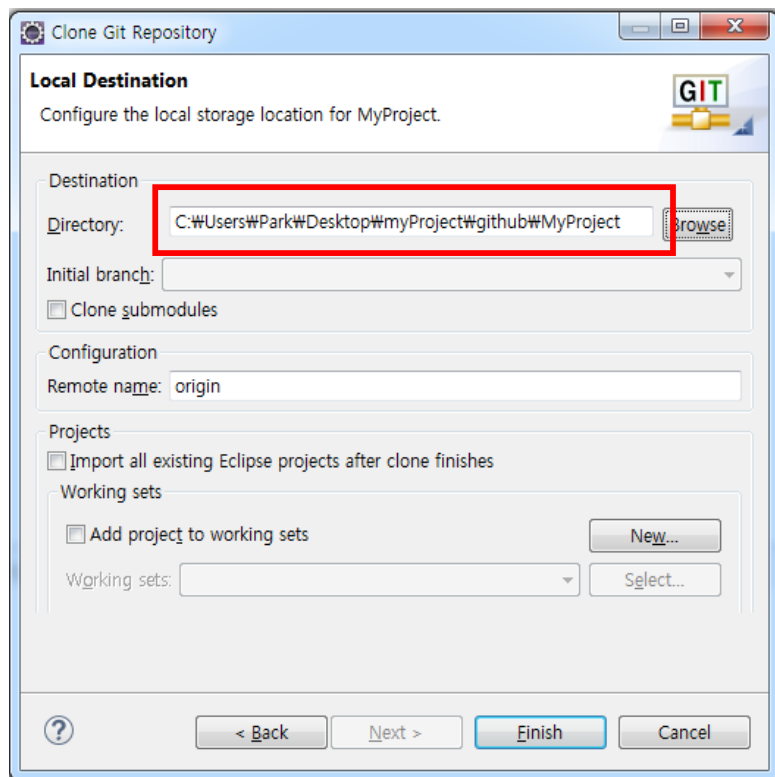


URL에 본인 원격저장소 URL를 적고  
아이디 비번을 적고 next



아직 아무것도 없으니 next

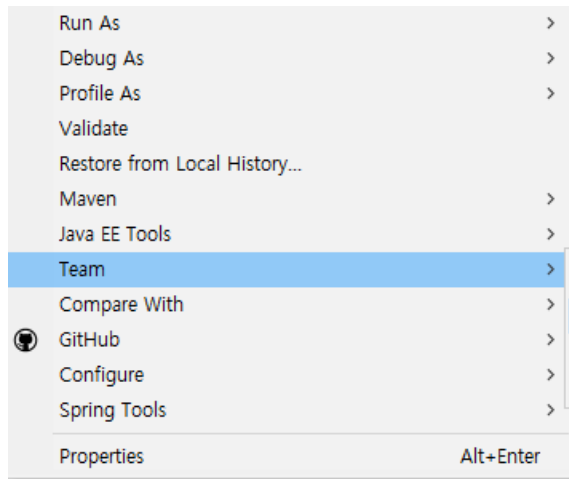
# 깃 이클립스에 연동하기



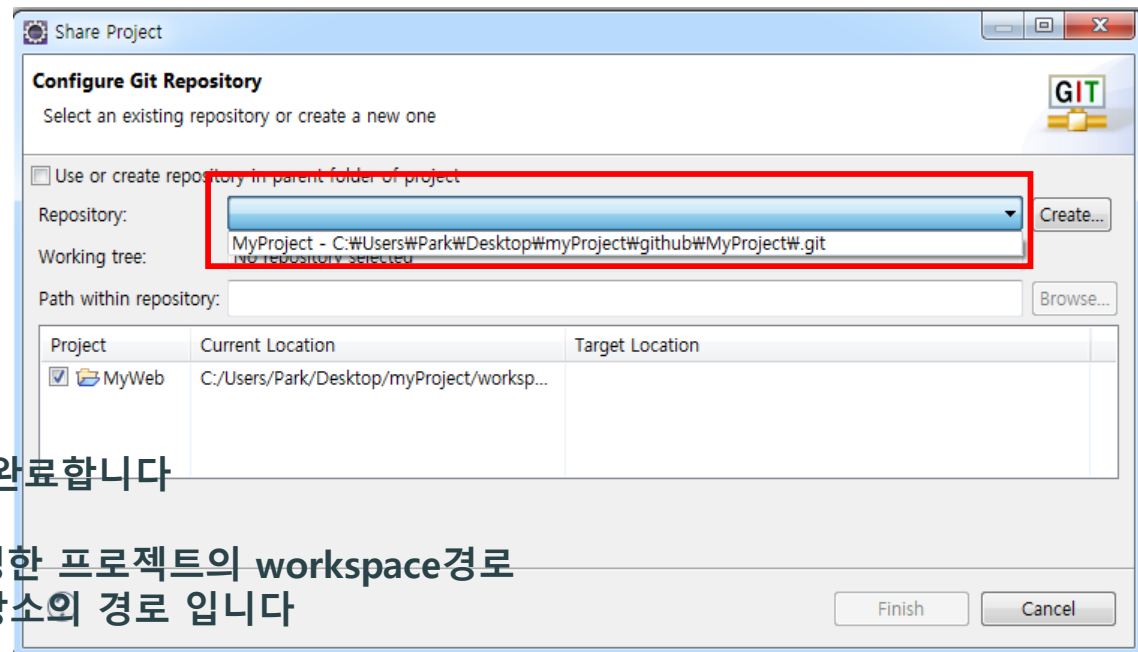
- 다음 본인 로컬 repository(저장소)가 생성되었습니다.
- STS로 돌아가서 업로드할 프로젝트를 선택.

- 프로젝트가 임시로 저장된 로컬저장소를 본인 폴더에 만들고 finish를 누릅니다

# 깃 이클립스에 연동하기



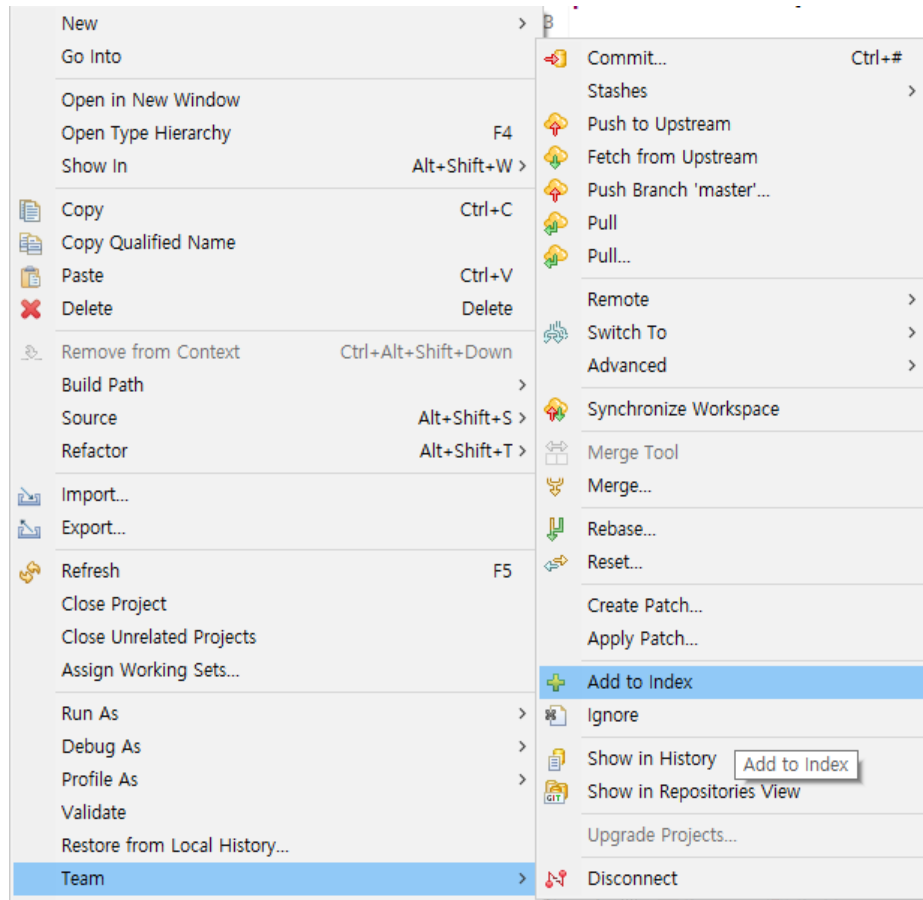
프로젝트 우클릭 Team -> Share Project클릭



- 아까 생성한 로컬저장소를 클릭해서 완료합니다

**Current Location**은 이클립스에서 생성한 프로젝트의 workspace경로  
**Target Location**은 Git에서 받아올 저장소의 경로 입니다

# 깃 이클립스에 연동하기

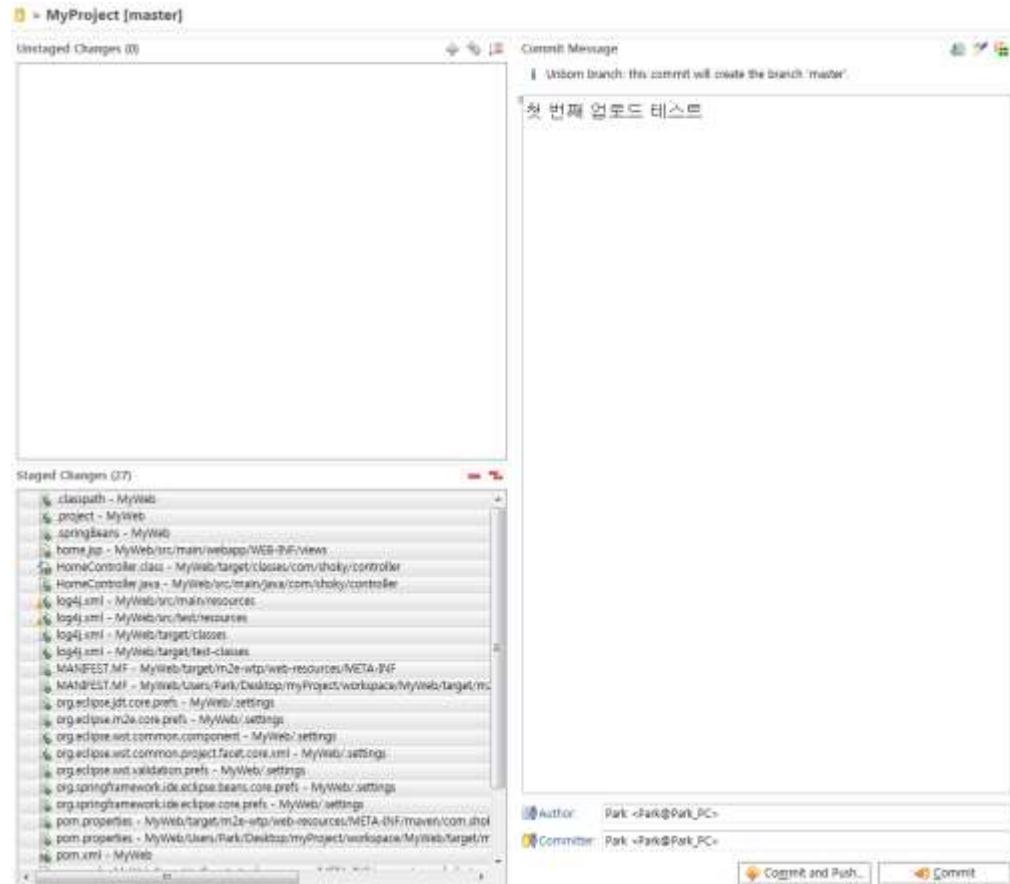
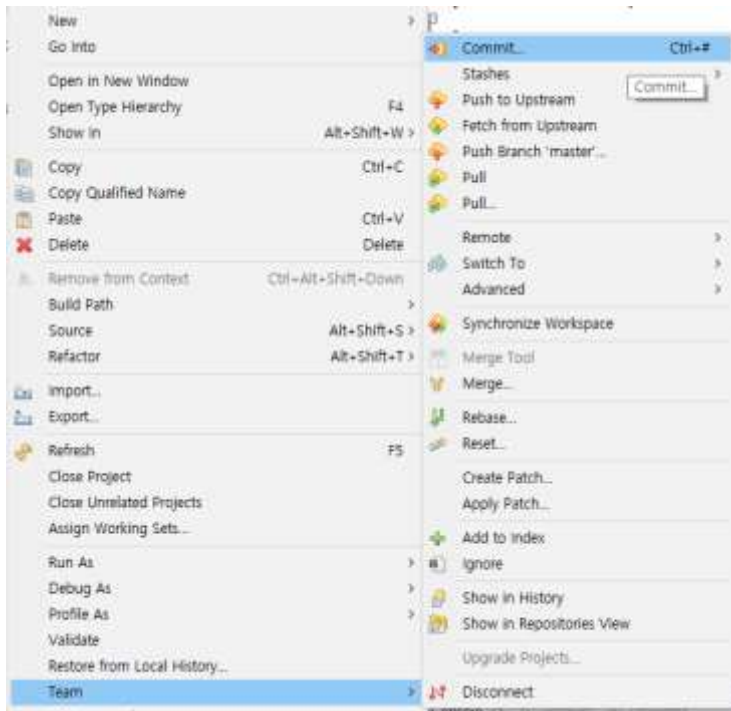


-다시 해당 프로젝트를 우 클릭해서 -> Add to Index를 선택

하면 프로젝트의 물음표가 사라질 것입니다

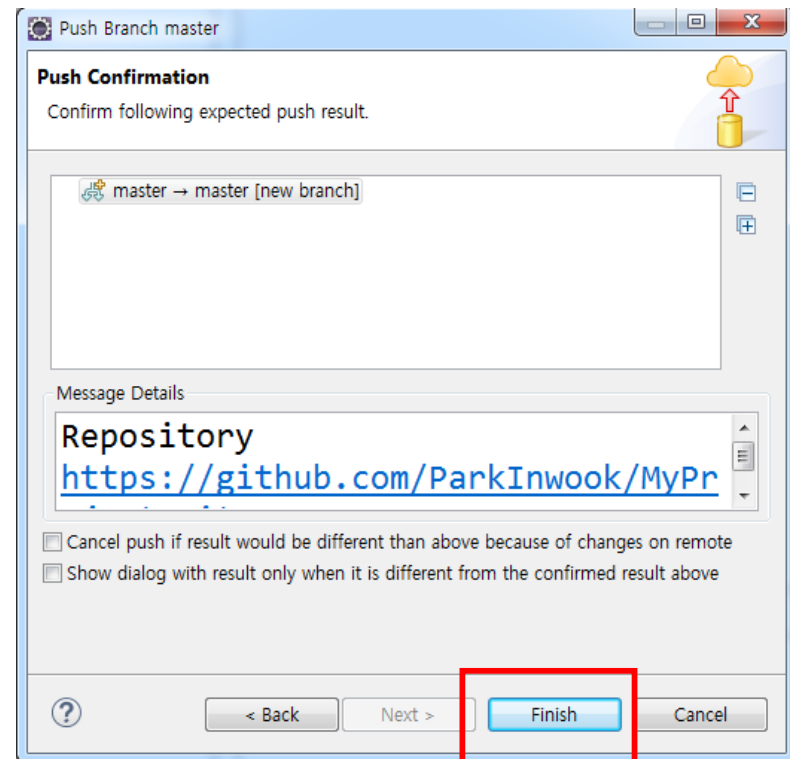
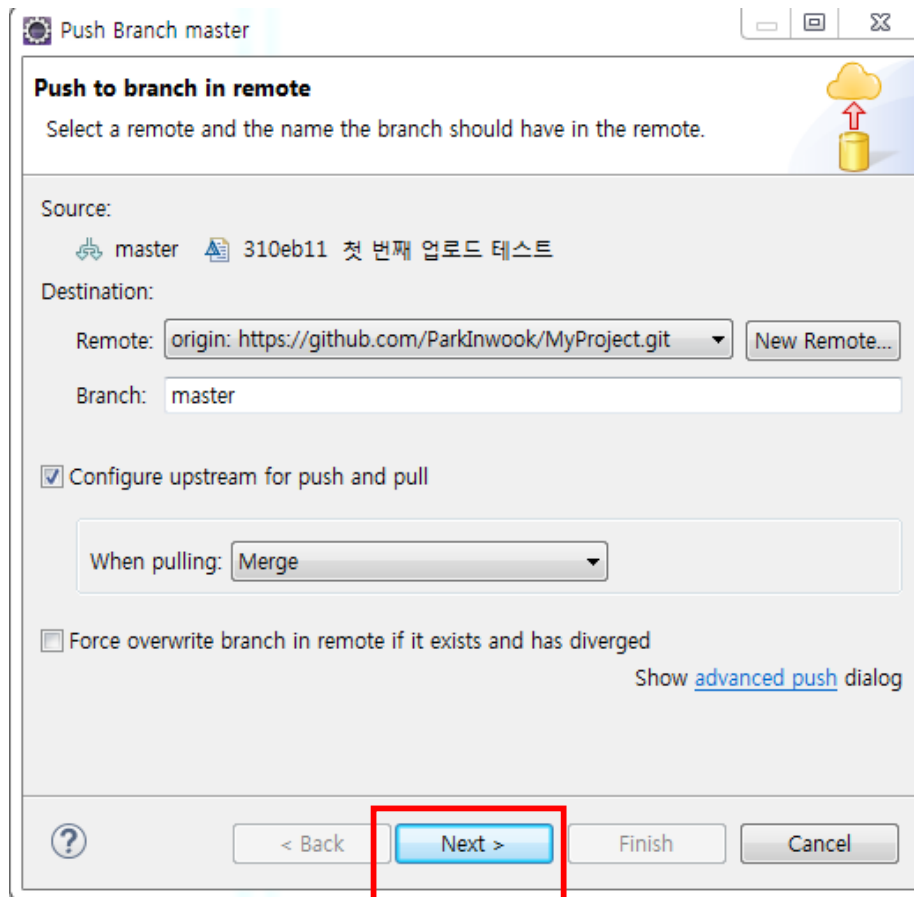
# 깃에 프로젝트 업로드 하기

- 프로젝트 우클릭 -> commit 클릭 (원격저장소로 업로드 하는 것 입니다)



- 코멘트 써주고 Commit and push클릭

# 깃에 프로젝트 업로드 하기



finish를 누르면 업로드가 완료됩니다

-업로드 정보가 나타나고 next 클릭

# 깃에 프로젝트 업로드 하기

ParkInwook / MyProject

Unwatch 1Star 0Fork 0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Security

Insights

Settings

No description, website, or topics provided.

Edit

Manage topics

1 commit

1 branch

0 releases

0 contributors

Branch: master


New pull request

Create new file


Upload files

Find File

Clone or download

 Park and Park 첫 번째 업로드 테스트

Latest commit 310eb11 7 minutes ago

 MyWeb

첫 번째 업로드 테스트

7 minutes ago

Help people interested in this repository understand your project by adding a README.

Add a README

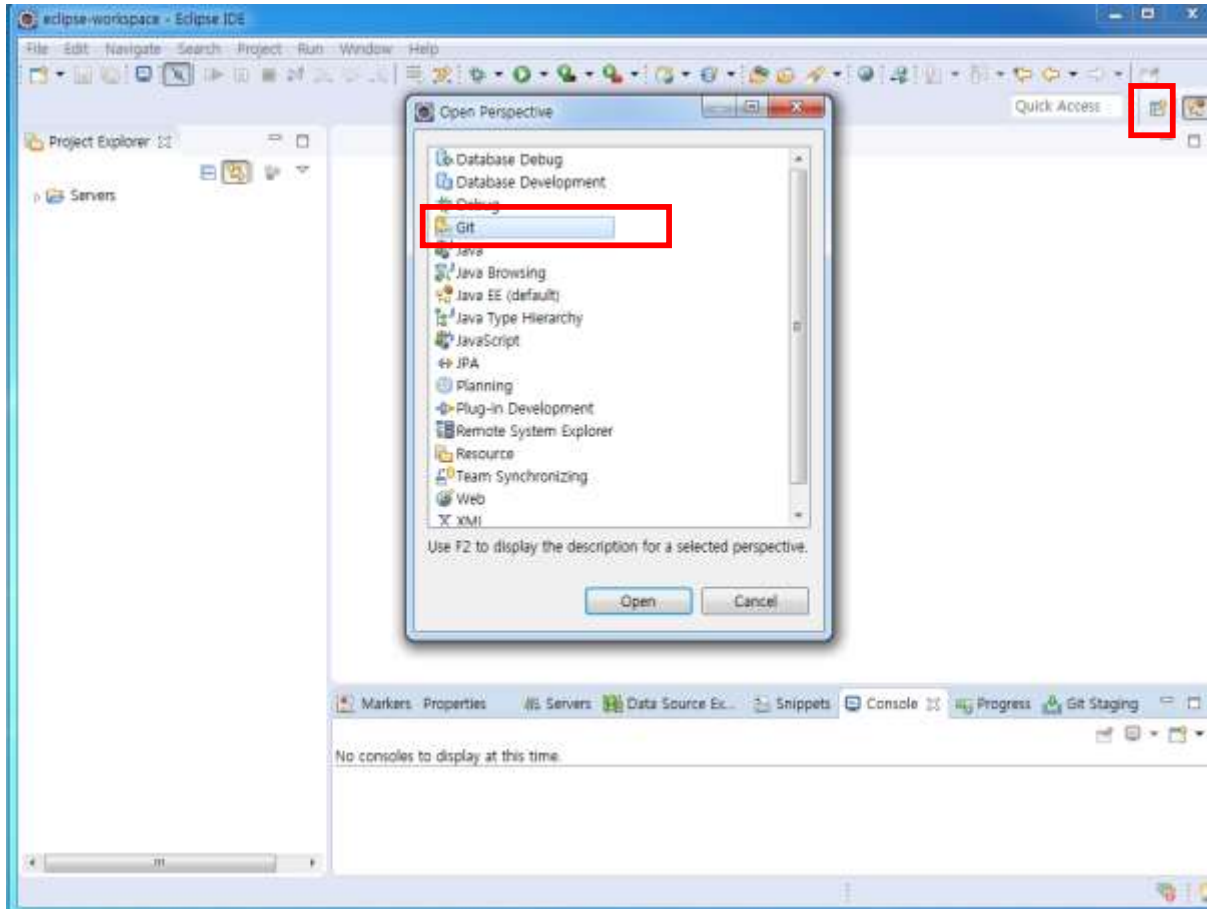
나의 Git저장소에 업로드가 완료된 모습!



# 깃 허브 본인 프로젝트로 가져오기



# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기



아무것도 없는 본인의 이클립스 환경에 우측 perspective클릭 git추가

# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기

ParkInwook / MyProject

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. Edit

Manage topics

1 commit 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find File Clone or download

Park and Park 첫 번째 업로드 테스트

MyWeb 첫 번째 업로드 테스트

Help people interested in this repository understand your project by adding a README.

Clone with HTTPS Use SSH

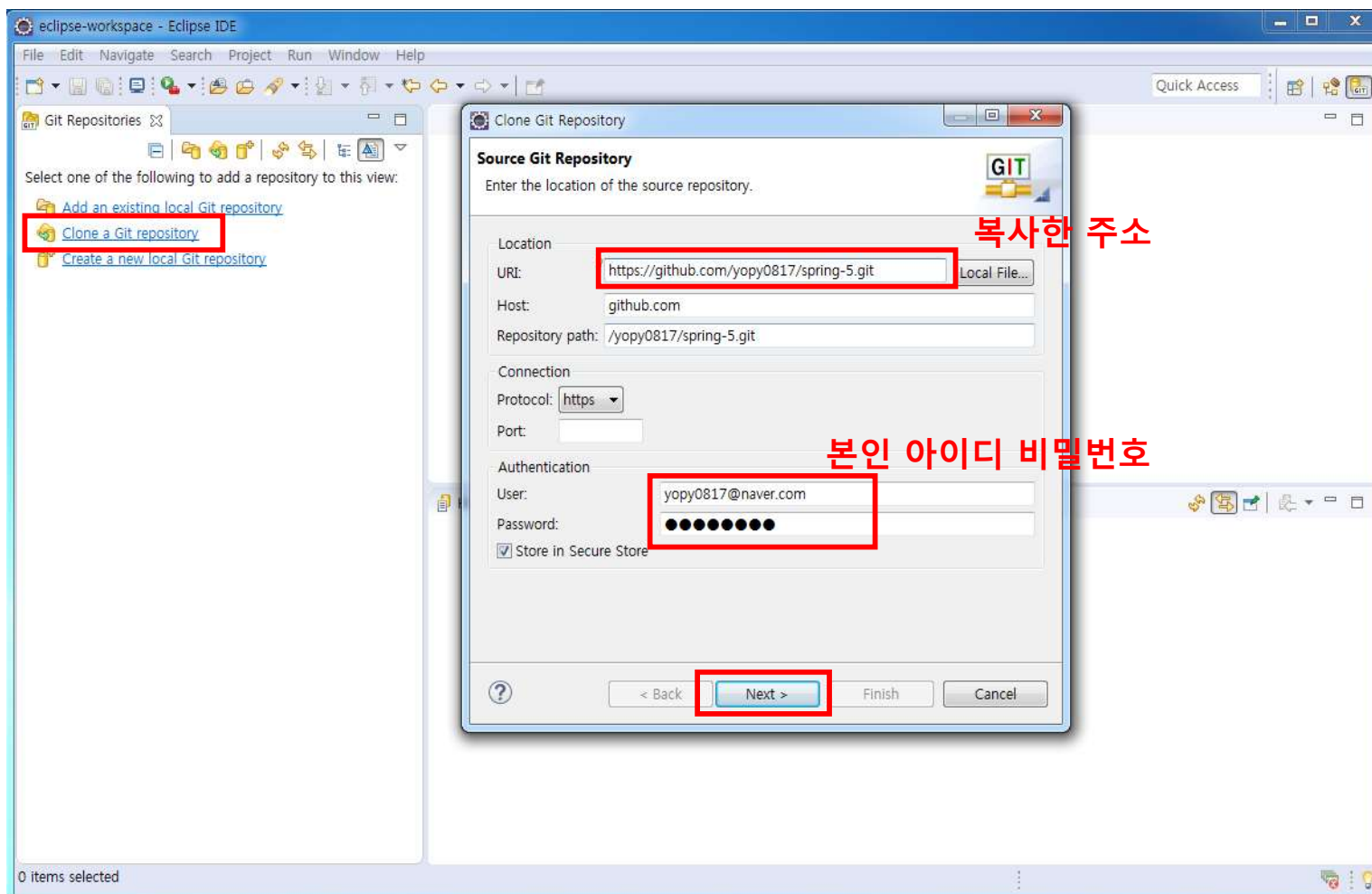
Use Git or checkout with SVN using the web URL.

https://github.com/yopy0817/spring-5.git

Open in Desktop Download ZIP

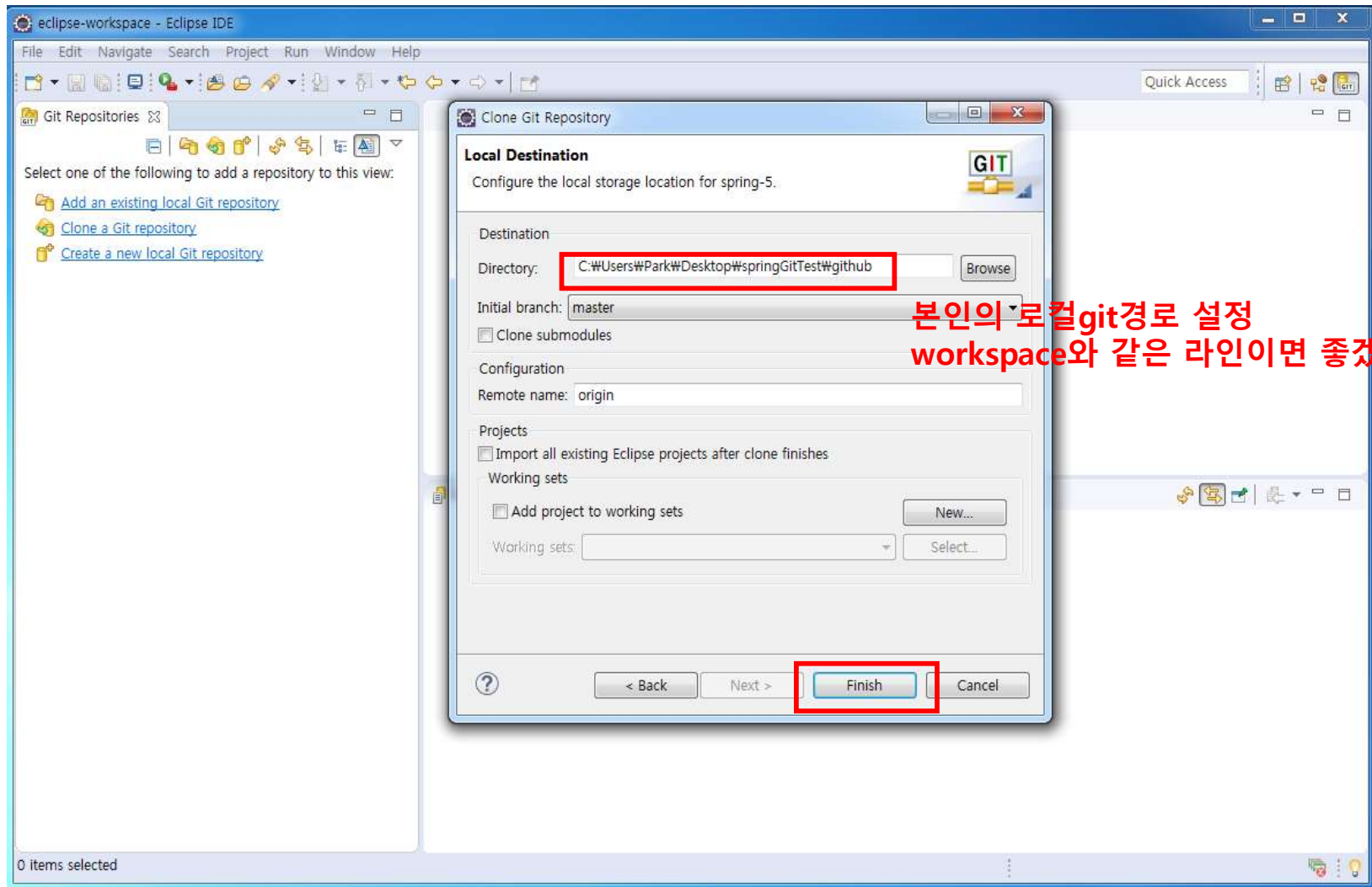
본인 깃에 접속 클론 클릭 uri주소 복사

# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기

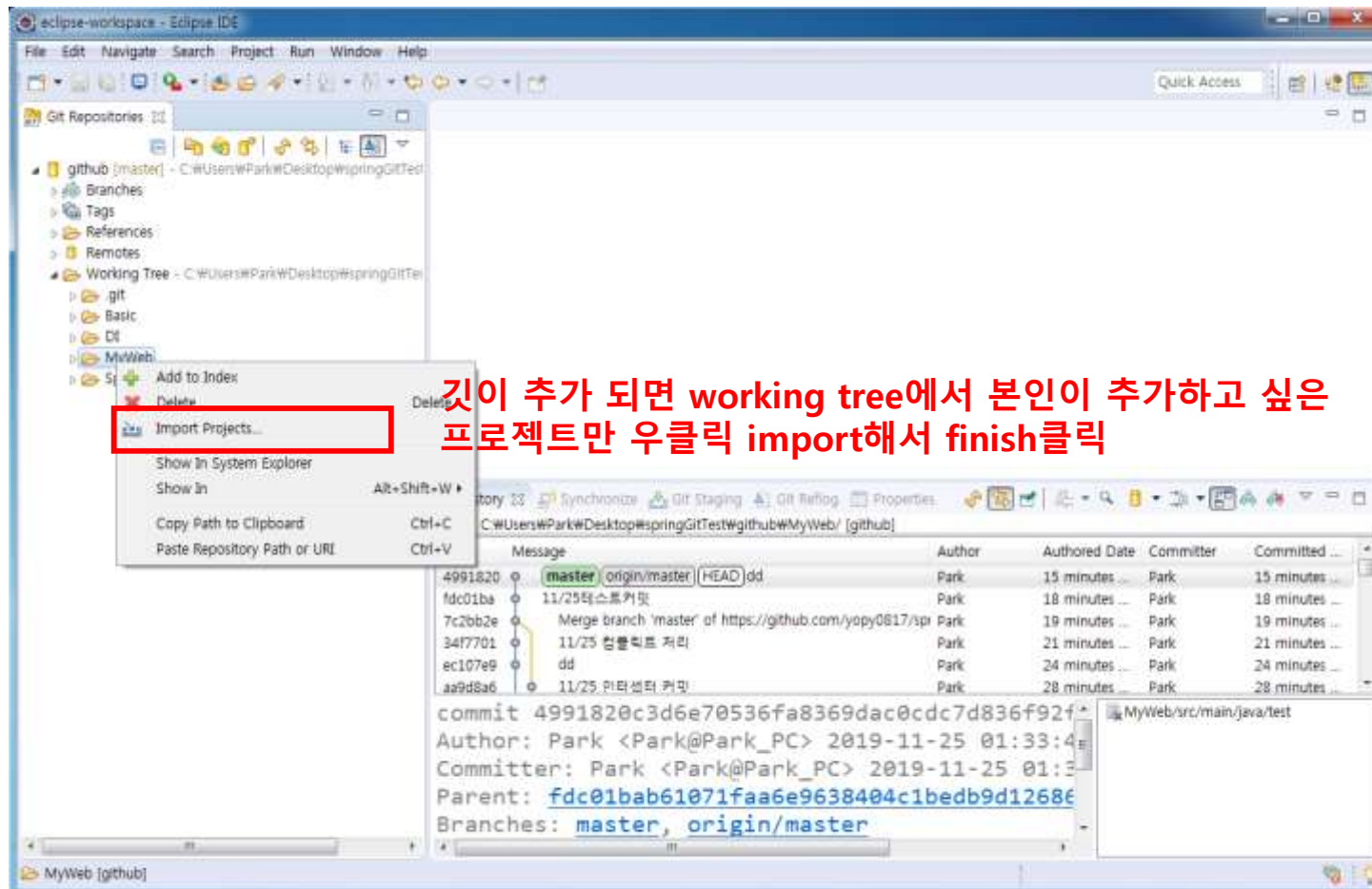


아이디 비밀번호가 다르면 나중에 push, pull이 안되니 주의

# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기

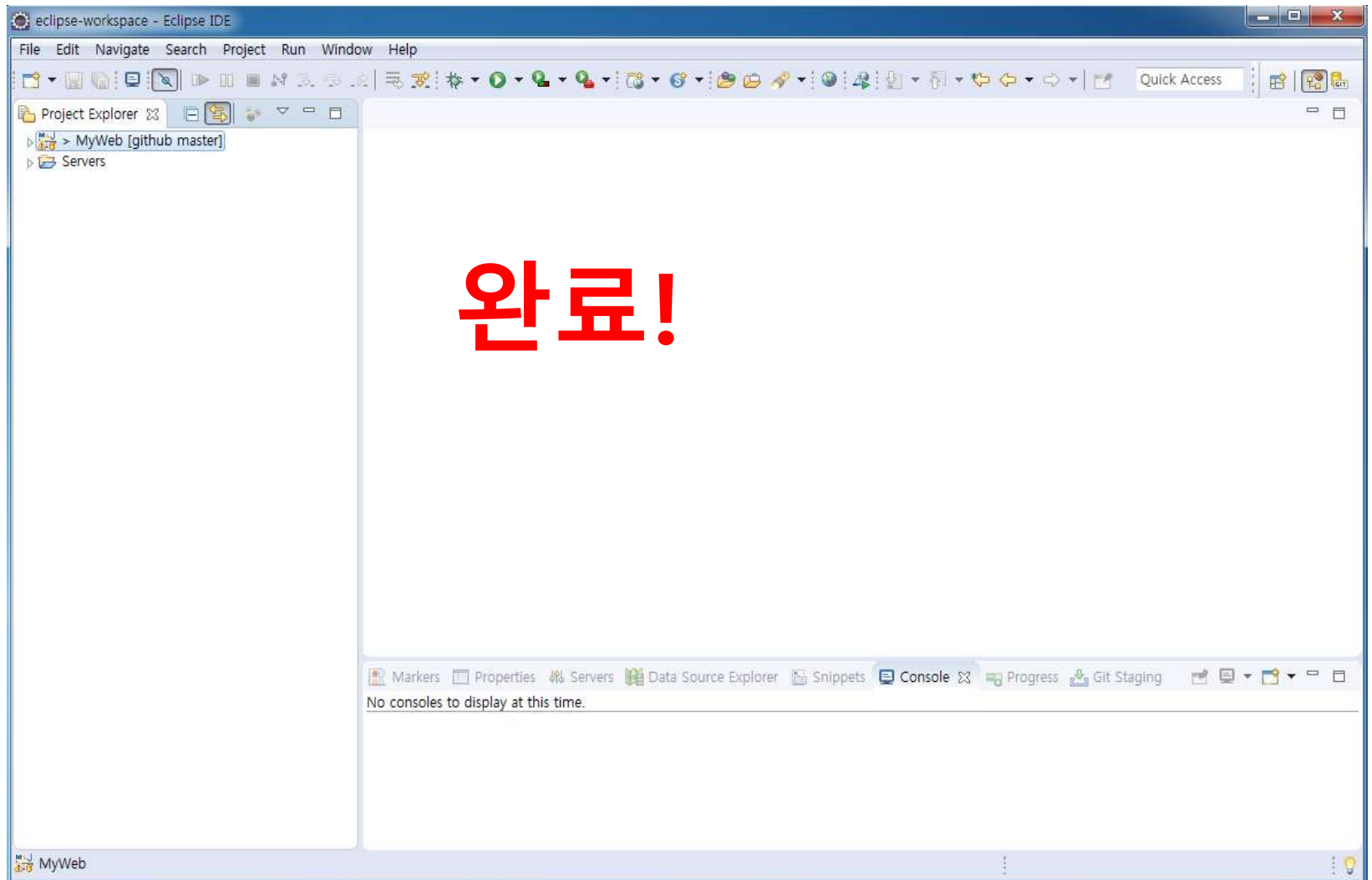


# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기



깃이 추가 되면 working tree에서 본인이 추가하고 싶은 프로젝트만 우클릭 import해서 finish클릭

# 가장 처음 깃을 본인의 프로젝트 환경으로 import하기

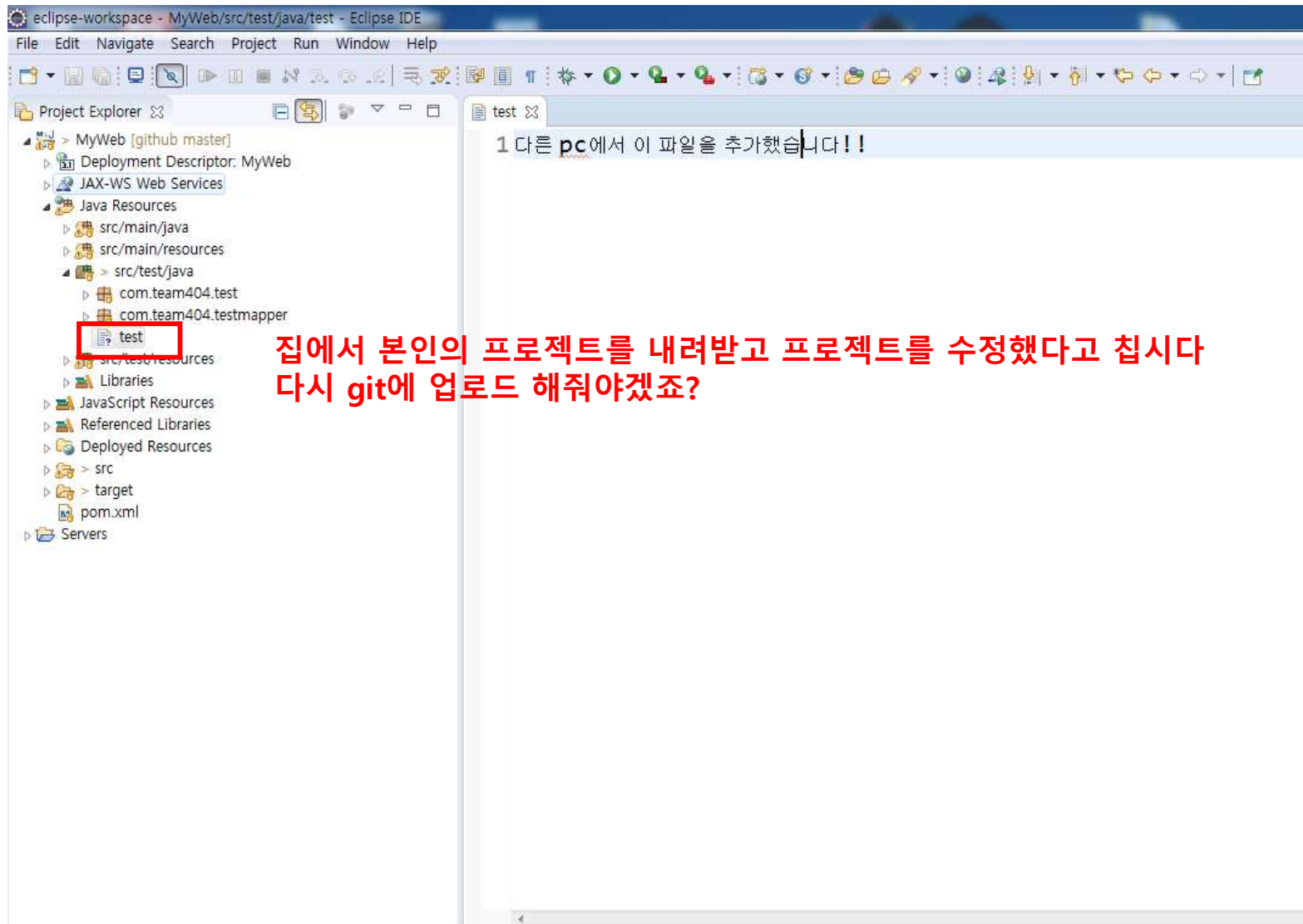


# 깃 허브 수정 후 project push and commit





# 깃 허브 수정 후 project push and commit

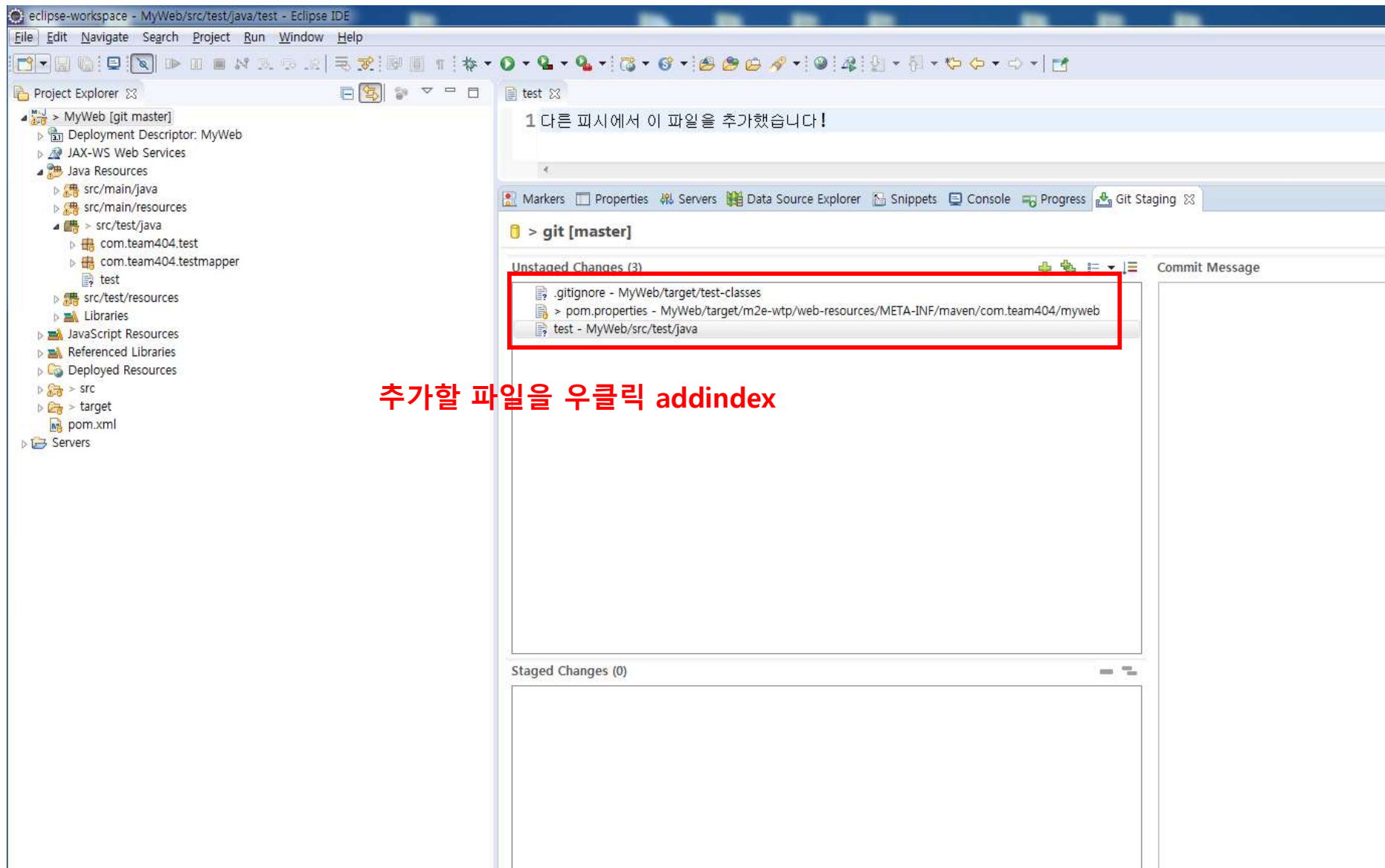


# 깃 허브 수정 후 project push and commit

The screenshot shows the Eclipse IDE interface. The 'Team' menu is open, and the 'Commit...' option is highlighted with a red box. Below the menu, the 'Git Staging' view is visible, showing 'Unstaged Changes (4)' and 'Staged Changes (0)'. The 'Commit Message' field is also visible.

프로젝트 우클릭 team -> commit

# 깃 허브 수정 후 project push and commit



The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer showing the project structure. The main area on the right is the Git Staging view, which displays files ready to be committed. A red rectangle highlights the 'Unstaged Changes (3)' section, which includes:

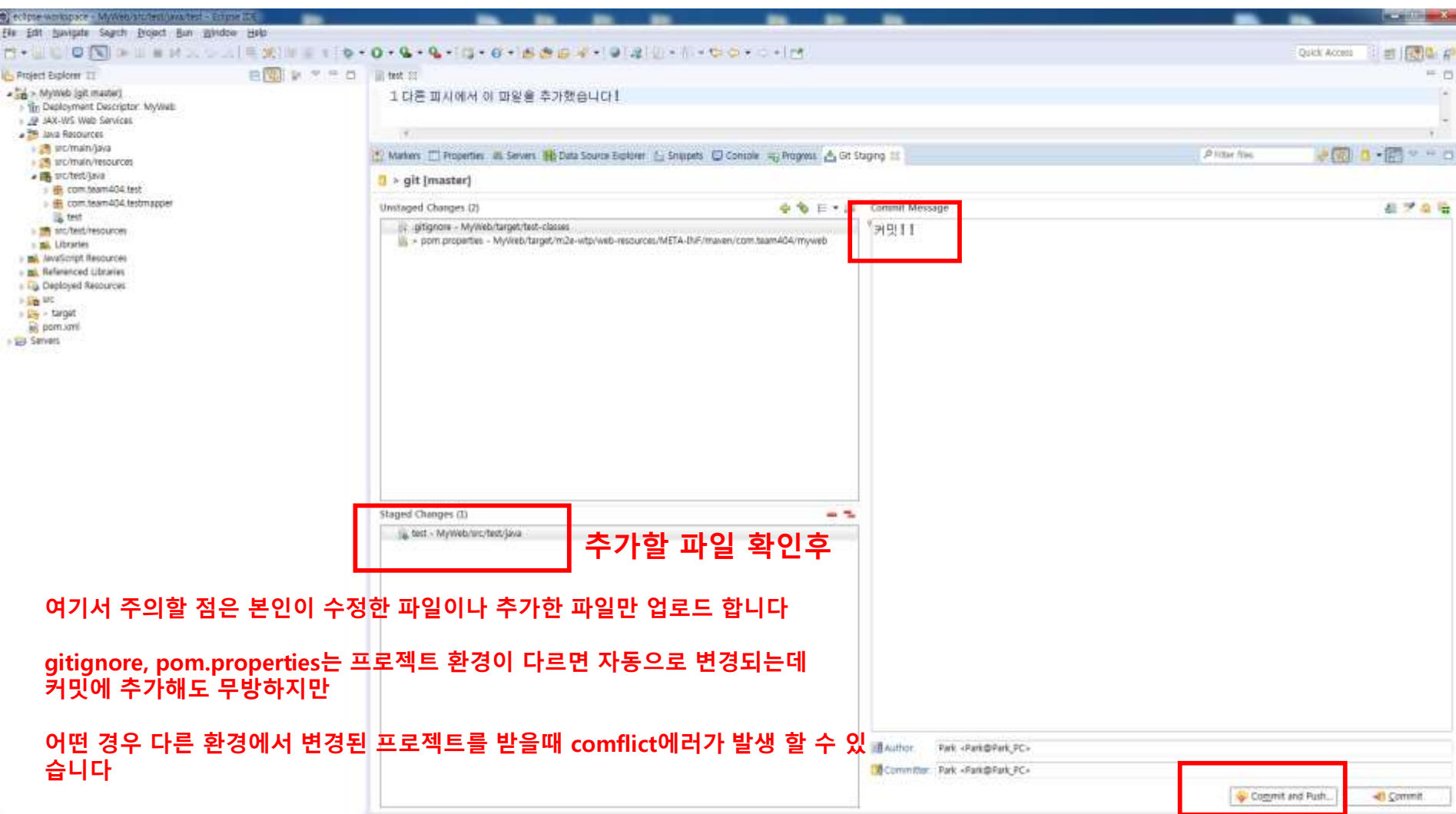
- .gitignore - MyWeb/target/test-classes
- > pom.properties - MyWeb/target/m2e-wtp/web-resources/META-INF/maven/com.team404/myweb
- test - MyWeb/src/test/java

Below the 'Unstaged Changes' section is the 'Staged Changes (0)' section, which is currently empty. To the right of the Git Staging view is the 'Commit Message' field.

1 다른 피스에서 이 파일을 추가했습니다!

추가할 파일을 우클릭 addindex

# 깃 허브 수정 후 project push and commit



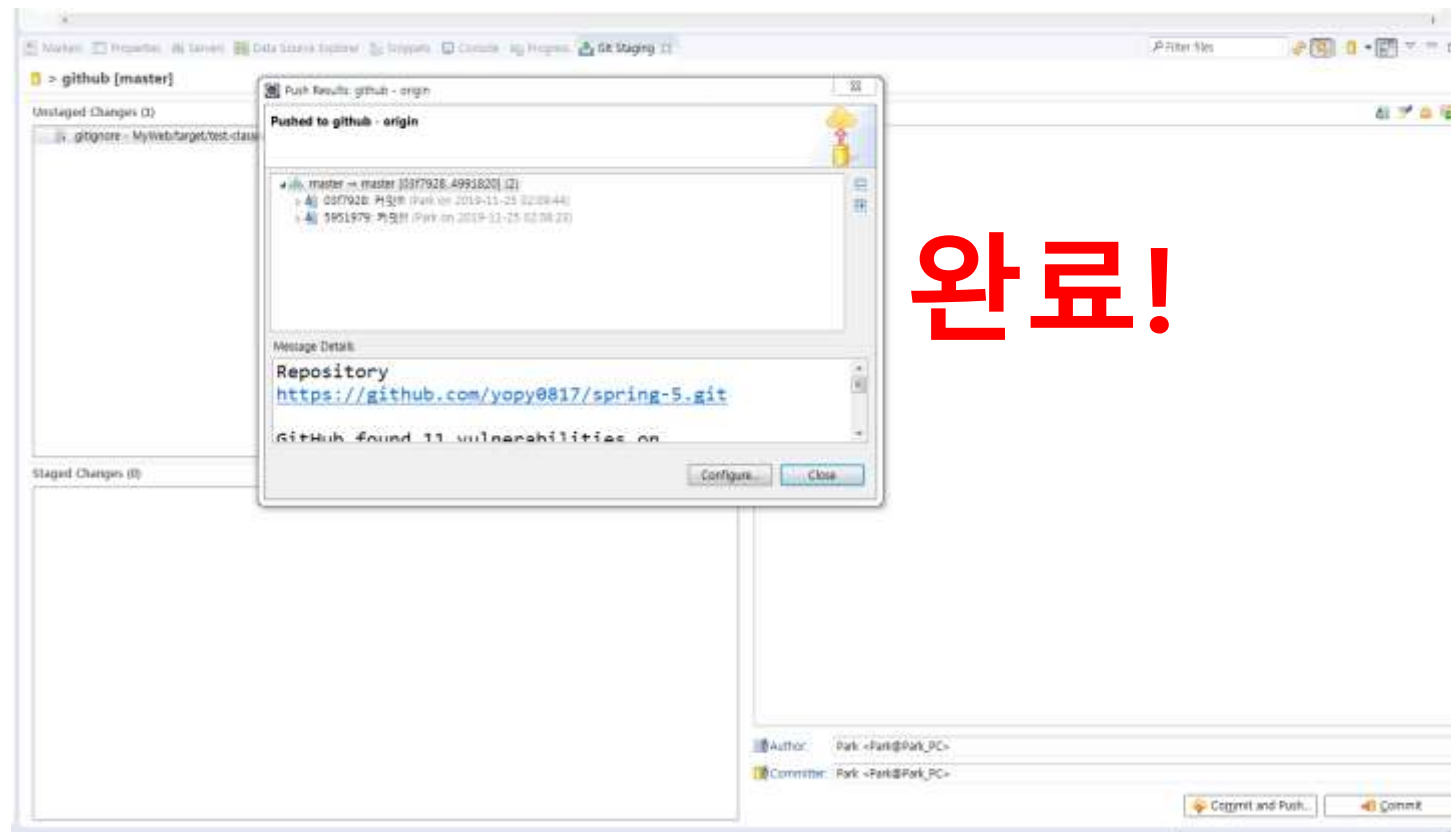
여기서 주의할 점은 본인이 수정한 파일이나 추가한 파일만 업로드 합니다

gitignore, pom.properties는 프로젝트 환경이 다르면 자동으로 변경되는데 커밋에 추가해도 무방하지만

어떤 경우 다른 환경에서 변경된 프로젝트를 받을때 conflict에러가 발생 할 수 있습니다

커밋엔 푸시!!

# 깃 허브 수정 후 project push and commit



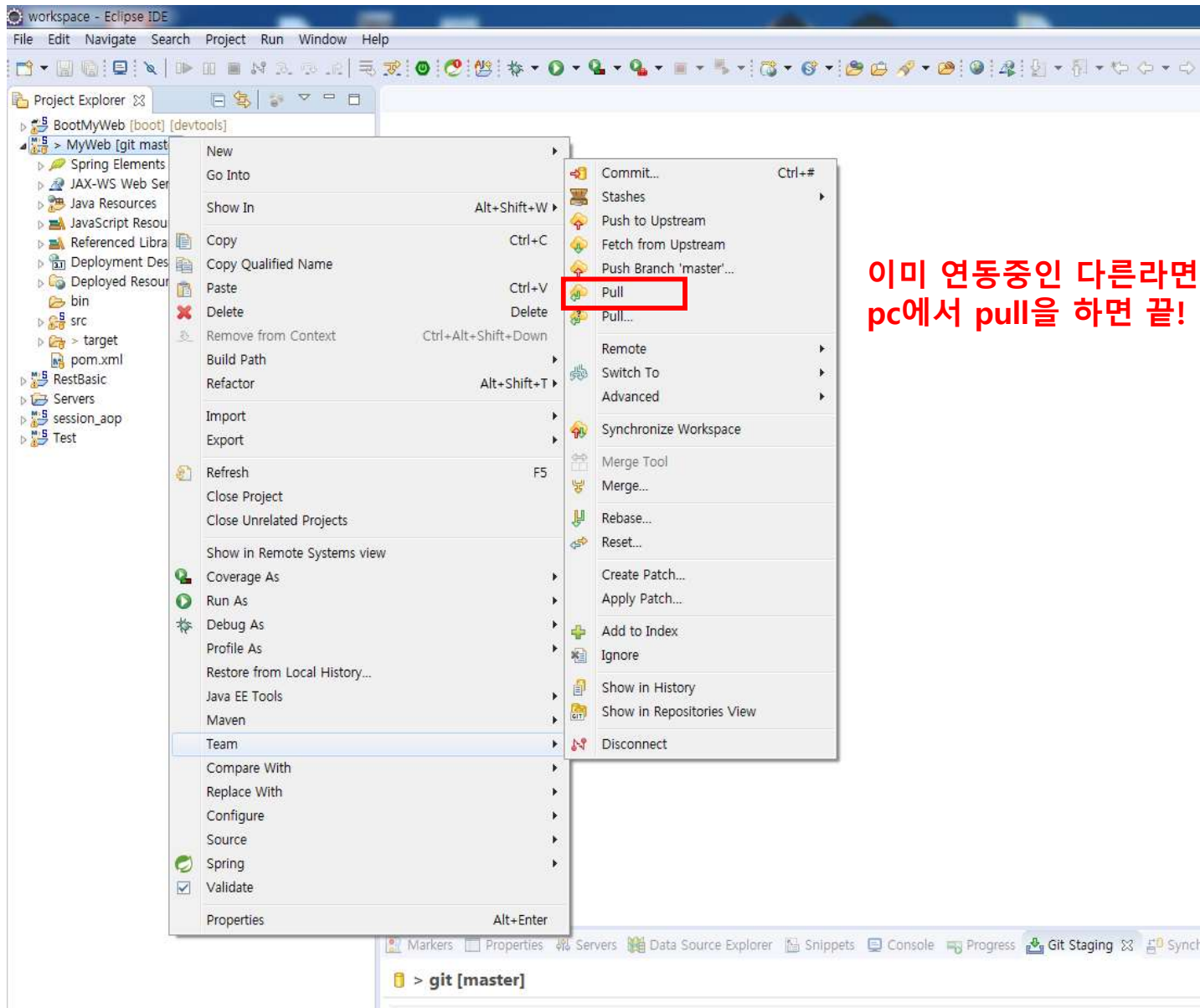
완료!

본인의 깃 사이트로 가서 업로드를 확인하세요!

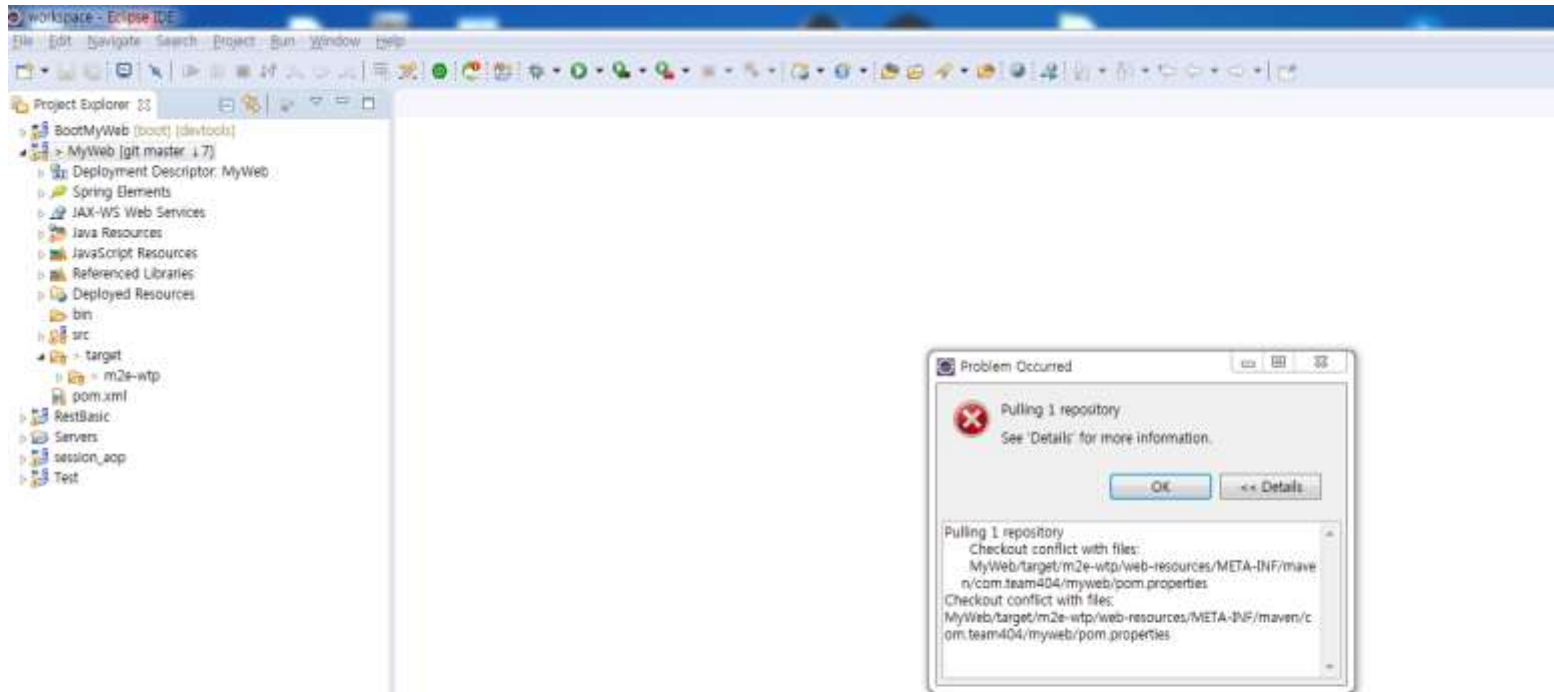
# 수정된 깃허브를 다른 pc에서 받기 pull



# 수정된 깃허브를 다른 pc에서 받기 pull



# 수정된 깃허브를 다른 pc에서 받기 pull



잘 당겨지면 끝입니다.

하지만, 에러가 나는 경우가 많습니다.

에러 내용을 가만히 읽어보면 당기는 도중 **comflict**충돌이 발생했다네요.

협업을 하며, 서로다른 pc에서 같은 파일들이 수정될 때, 내 로컬에 존재하는 커밋 이력과 저장소에 커밋이력이 다르거나, 파일이 둘다 수정된 경우

에러를 발생하는 것입니다.

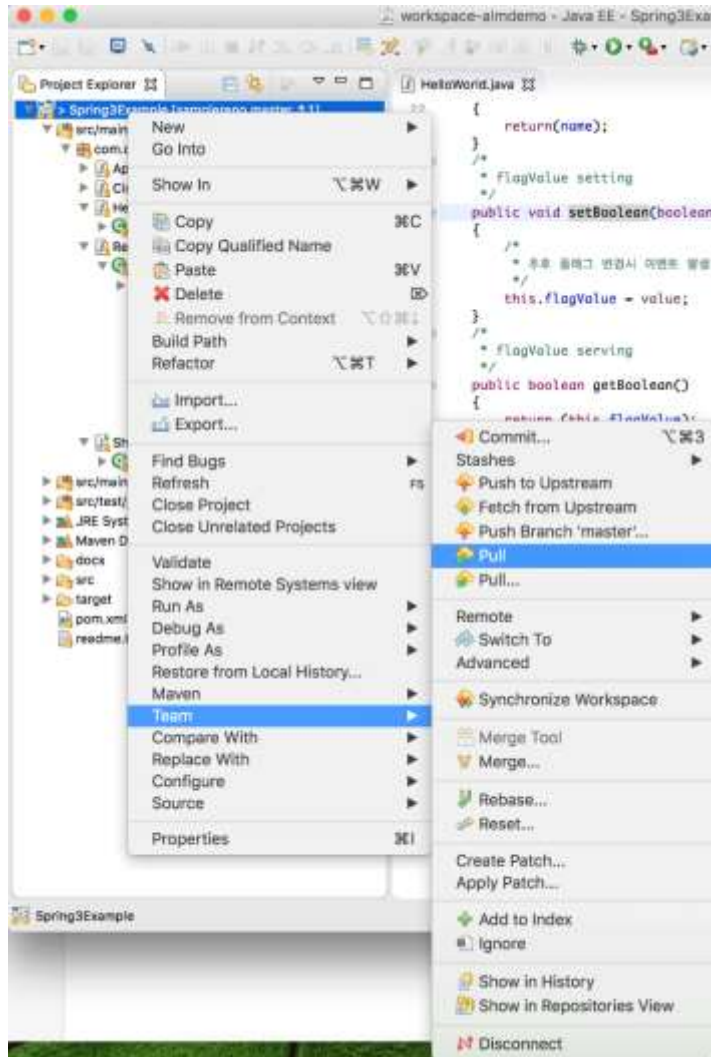
이럴 때 개발자는 충돌된 두 파일 중 무엇을 받을 것인지 병합 작업(merge)를 해주셔야 합니다



# 수정된 깃허브를 다른 pc에서 받기 pull

## Conflict 해소 절차

1) 프로젝트 선택하여 마우스 우클릭 >> Team >> Pull 선택



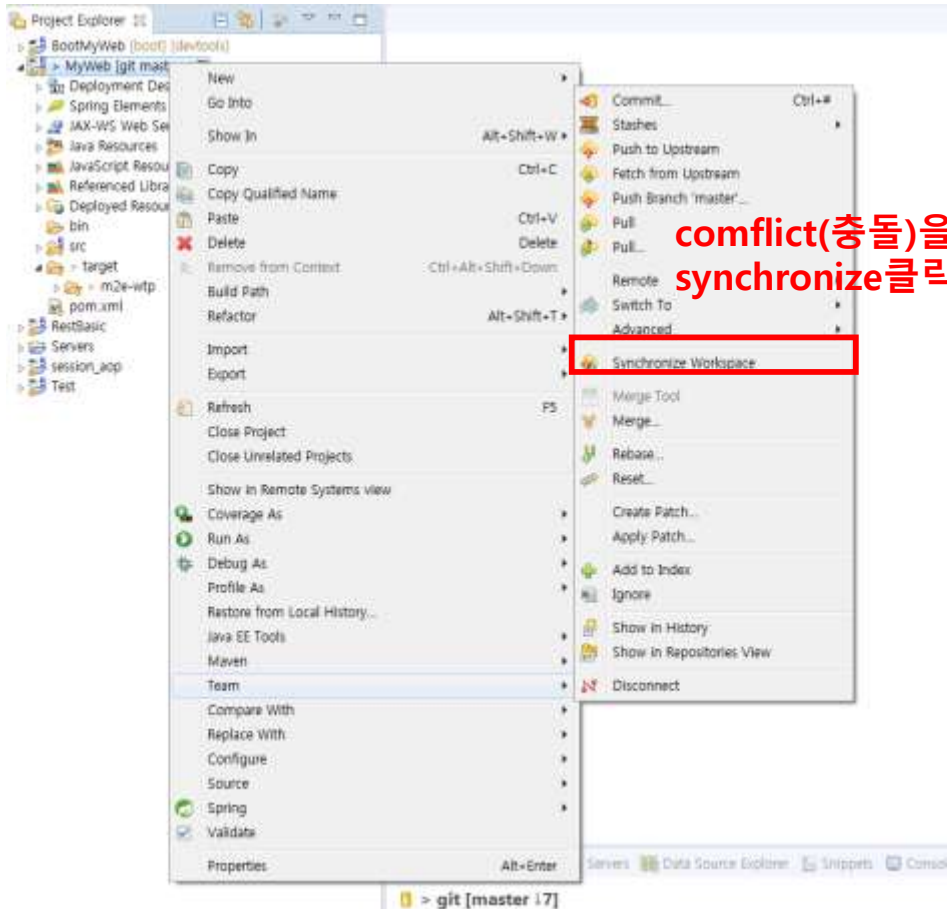
Conflicting이 발생한 commit 목록이 표시된다.



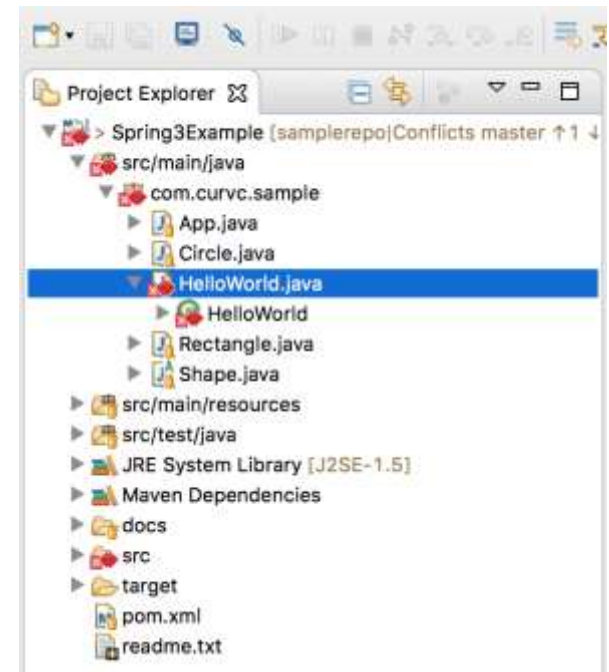
# 수정된 깃허브를 다른 pc에서 받기 pull

## 2) Conflict 위치 찾기 및 수정

Eclipse의 Project Explorer에 conflict이 발생한 파일을 확인하다.



conflict(충돌) 파일



# 수정된 깃허브를 다른 pc에서 받기 pull

```
28 public void setBoolean(boolean value)
29 {
30 <<<<<< HEAD
31 /*
32 * 추후 플러그 변경시 이벤트 발생 기능 추가 필요
33 */
34
35 /*
36 * 플러그 변수값 변경
37 */
38 >>>>>> branch 'master' of http://almdemo.curvc.com:48081/samplerapp
39 this.flagValue = value;
40 }
```

Git은 컴파일 오류를 유발고 눈에 띄게 할 목적으로 **conflict**이 발생한 부분을

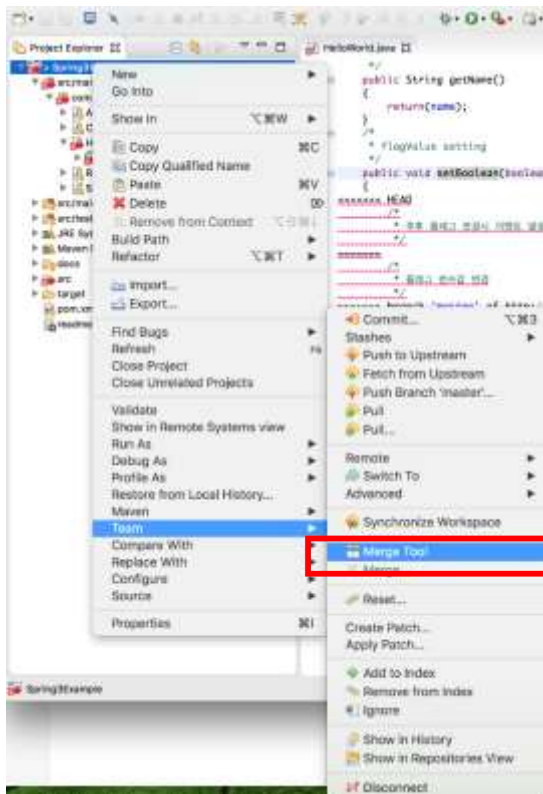
<<<<<<<

=====

>>>>>>> 로 표시한다

적절하게 알맞은 코드로 수정 후

특수문자를 제거후 **commit**을 진행 하면 됩니다

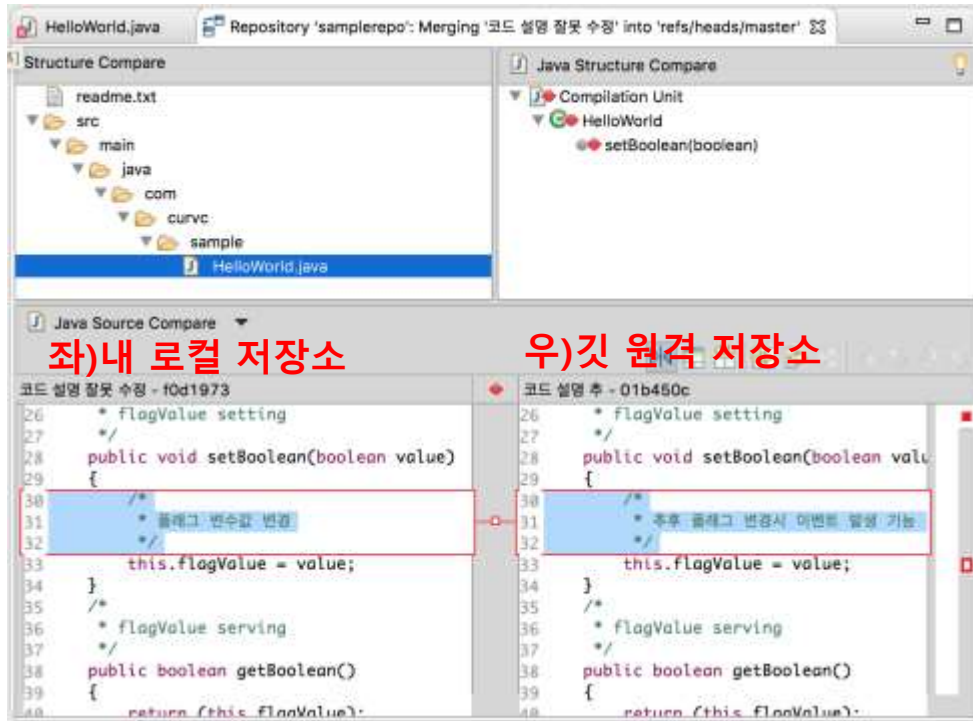


복잡한 **conflict**이 발생한 경우 Eclipse에서 제공하는 **merge** 기능이 유용하다.

**Project (또는 conflict이 발생한 파일 선택) >> Team >> Merge Tool**

# 수정된 깃허브를 다른 pc에서 받기 pull

Merge 도구를 통해 수동 merge가 필요한 파일과 위치를 찾을 수 있다.



수정을 완료하고 파일을 저장한다.

**Merge 도구 메뉴**

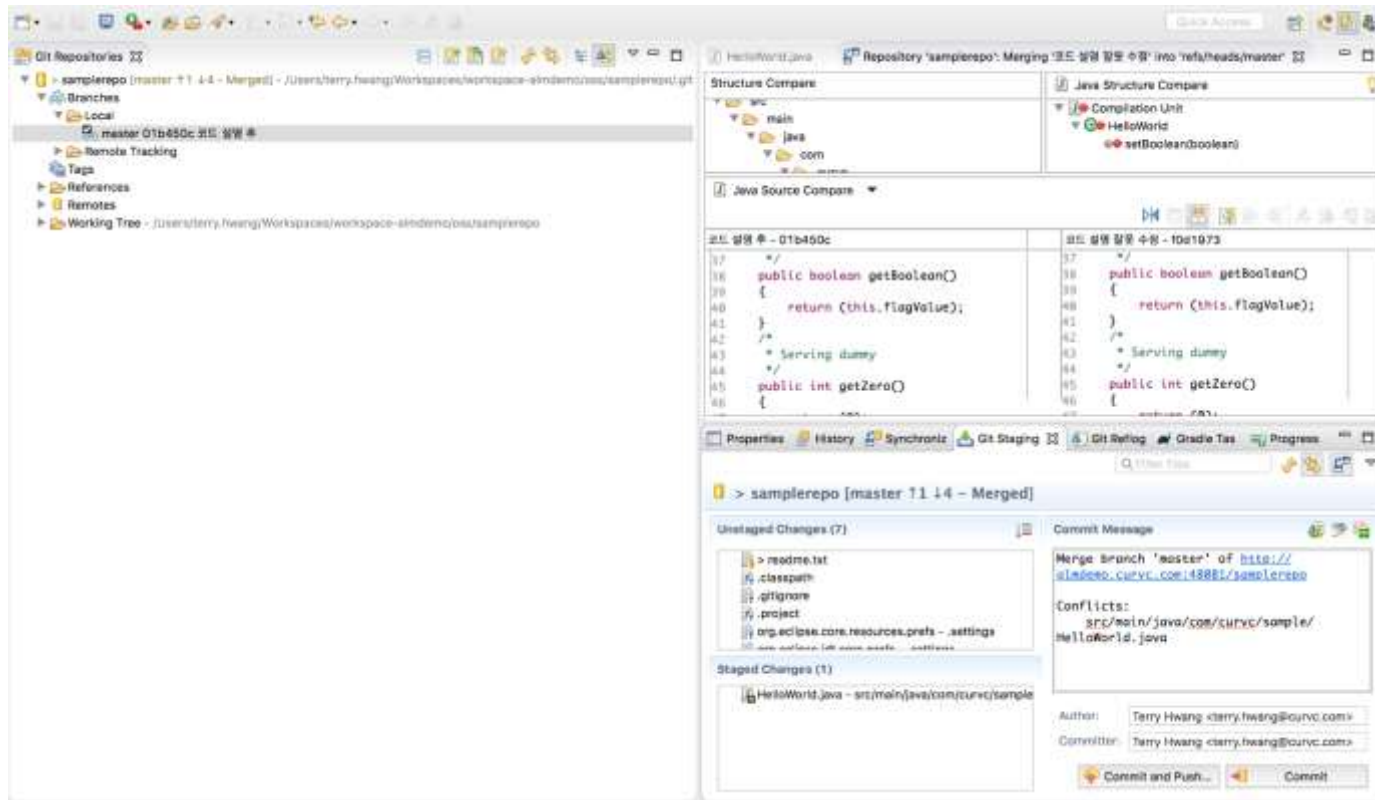


- 좌/우 view 치환
- 두 변경의 공통 원본 표시
- Three way compare: 공통 원본과 함께 두 변경 비교
- Conflict가 발생하지 않는 오른쪽의 내용을 모두 왼쪽에 반영
- 왼쪽 변경을 오른쪽에 반영
- 오른쪽의 변경을 왼쪽에 반영
- 다음 차이점 위치
- 이전 차이점 위치
- 다음 변경
- 이전 변경

# 수정된 깃허브를 다른 pc에서 받기 pull

## 3) Merge 결과 Commit & Push

Merge 한 파일을 Staged Changes 창으로 이동시키고 "Commit and Push" 또는 Commit 한다



정상적으로 커밋이 진행될 것이고, 깃에 저장이 될 것입니다.  
나머지 다른 상세한 부분은 인터넷이나 교본을 참조하시기 바랍니다