eclipse와 github 연동

2017-03-06

이승진

**학습목표**

github에 리포지토리 만들기

eclipse 프로젝트를 github 리포지토리에 등록하기

github 리포지토리에 등록된 프로젝트를 팀원 PC에 가져오기

프로젝트 소스 코드에서 수정된 내용을 github 리포지토리에 등록하기

github 리포지토리에 등록된 소스 코드 수정 내용을 팀원 PC에 가져오기

충돌(conflict) 해결

**목차**

[1. github 2](#_Toc523155602)

[1) github에 계정 생성 2](#_Toc523155603)

[2) github에 repository 생성 2](#_Toc523155604)

[3) 팀원 초대 3](#_Toc523155605)

[2. EGit 설치 5](#_Toc523155606)

[1) EGit 설치 5](#_Toc523155607)

[2) EGit 업데이트 7](#_Toc523155608)

[3. local repository 생성 8](#_Toc523155609)

[1) 프로젝트 생성 8](#_Toc523155610)

[2) 프로젝트에 local repository 생성 8](#_Toc523155611)

[3) .gitignore 파일 생성 10](#_Toc523155612)

[4) local repository에 commit 11](#_Toc523155613)

[4. remote repository에 push 12](#_Toc523155614)

[1) Push 12](#_Toc523155615)

[2) Push 결과 확인 15](#_Toc523155616)

[5. 프로젝트 가져오기 16](#_Toc523155617)

[1) Import 16](#_Toc523155618)

[6. 프로젝트 설정 파일 무시 20](#_Toc523155619)

[7. 수정된 내용 push 21](#_Toc523155620)

[1) 소스 코드 수정 21](#_Toc523155621)

[2) commit 21](#_Toc523155622)

[3) push 22](#_Toc523155623)

[8. 수정된 내용 pull 22](#_Toc523155624)

[9. push 실패 23](#_Toc523155625)

[1) push 실패 23](#_Toc523155626)

[2) pull 먼저 23](#_Toc523155627)

[10. merge conflict 25](#_Toc523155628)

[1) merge conflict 25](#_Toc523155629)

[2) merge conflict가 발생한 이유 27](#_Toc523155630)

[3) 충돌 해결 28](#_Toc523155631)

[4) 다시 commit & push 29](#_Toc523155632)

[11. 바람직한 습관 30](#_Toc523155633)

[1) 잘 실행되는 것을 확인하고 push 해야 함 30](#_Toc523155634)

[2) 가급적 자주 push 30](#_Toc523155635)

[3) 가급적 자주 pull 30](#_Toc523155636)

[12. eclipse에서 git 아이콘 31](#_Toc523155637)

[1) commit 전과 후 아이콘 차이 31](#_Toc523155638)

[2) Assume Unchanged 32](#_Toc523155639)

# github

## github에 계정 생성

github 사이트에 회원 가입(sign up)을 하자.

팀원 전부 회원 가입 해야 한다.

gmail 주소로 가입하는 것을 추천.

https://github.com/

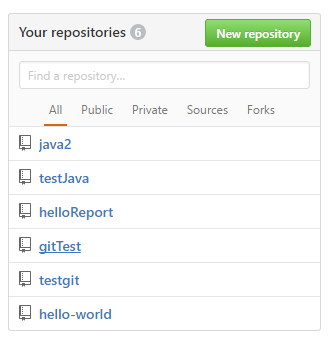
자세한 절차는 생략.

## github에 repository 생성

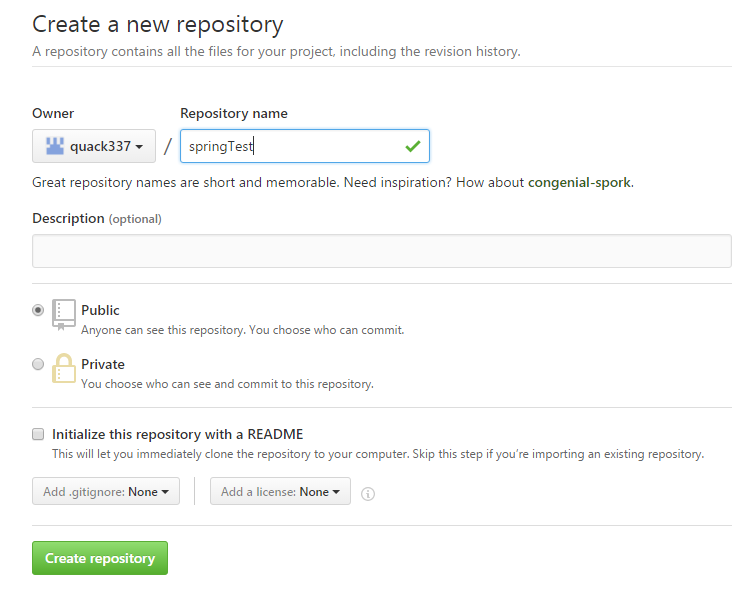
소스 코드를 저장할 저장소를 git repository라고 부른다.

프로젝트 팀장이 github 에 repository를 생성하자.

New Repository



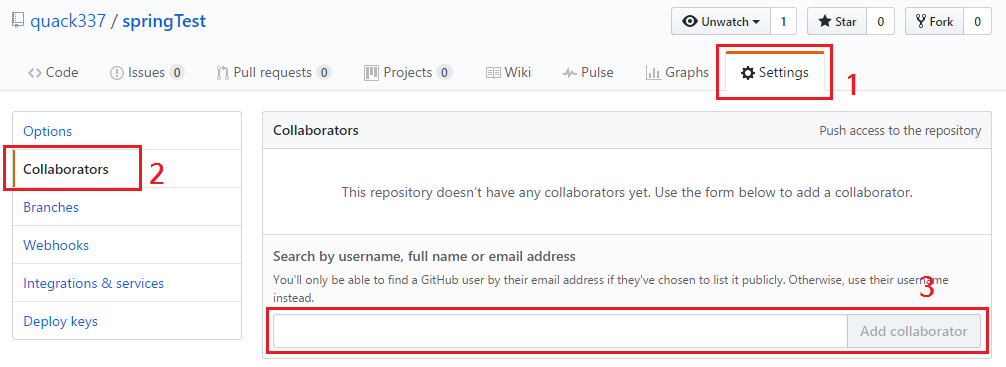
[new repository] 버튼 클릭



[Create repository] 버튼을 클릭하면, Github 서버에 리포지토리가 생성된다.

이것을 remote repository라고 부른다.

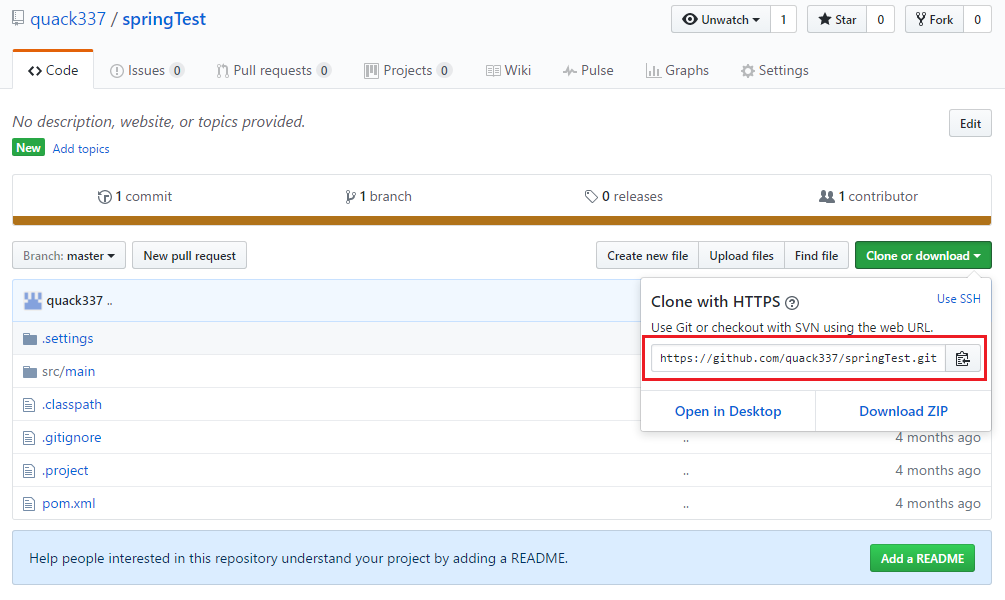
## 팀원 초대



팀원이 github에 등록한 이메일 주소를 입력하고 Add 버튼을 클릭하자.

그 이메일 주소로 초대 이메일이 전송된다.

팀원들은 초대 이메일을 열고 [Accept Invitation] 버튼을 클릭하자.



github에 생성된 remote repository의 URL을 위 화면에서 확인할 수 있다.

위 화면에서 [Download ZIP] 링크를 클릭하면 등록된 전체 소스코드를 다운받을 수 있다.

# EGit 설치

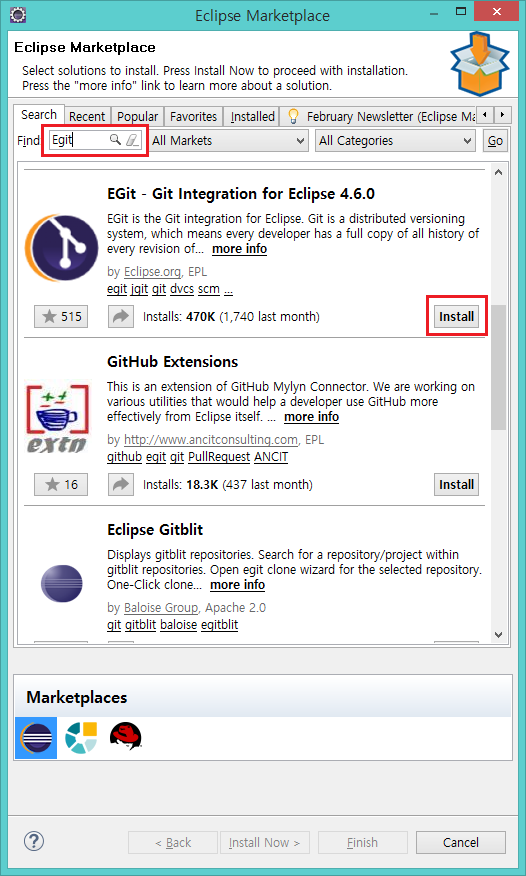
eclipse에 git 클라이언트 도구를 설치하자.

이것을 EGit 이라고 부른다.

모든 팀원이 EGit을 설치해야 한다.

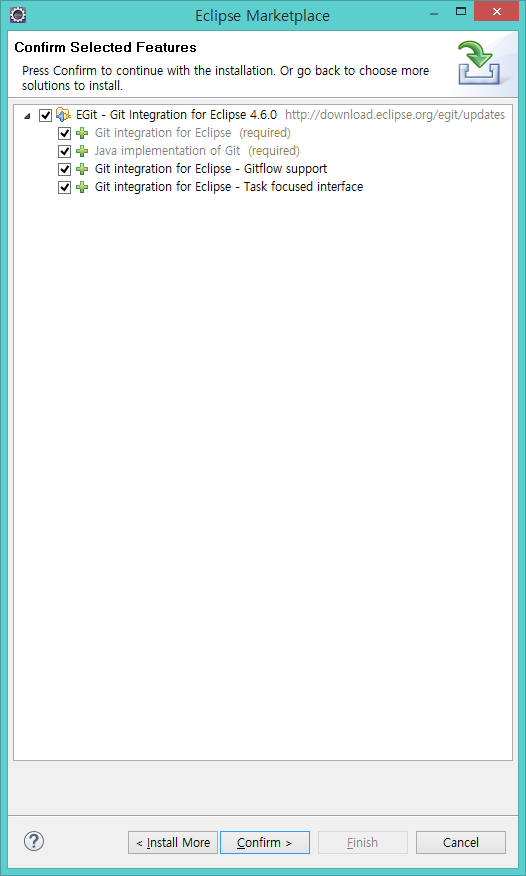
## EGit 설치

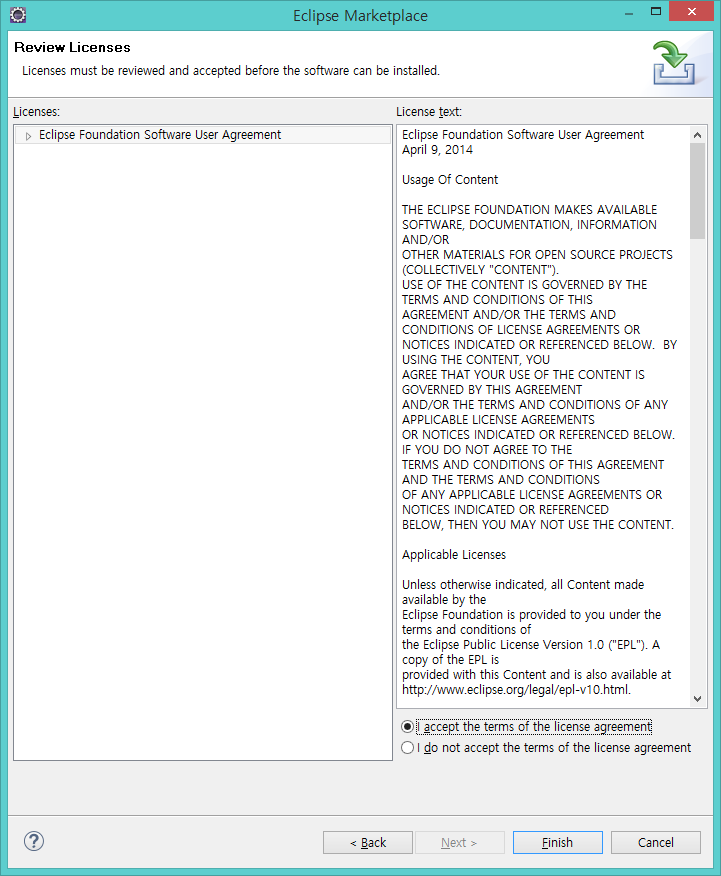
eclispe 메뉴 - Help - Eclipse Marketplace



검색에 EGit을 입력하고 엔터키

EGit 항목의 Install 버튼 클릭





## EGit 업데이트

EGit을 설치한지 오래되었다면 업데이트 하자.

eclipse 메뉴 - Help - Check for Updates

# local repository 생성

아래 절차는 프로젝트를 github repository에 처음 등록할 때 필요한 작업이다.

프로젝트 팀장이 최초로 한 번만 수행하면 된다.

## 프로젝트 생성

아직 프로젝트를 생성하지 않았다면 eclipse에서 프로젝트를 생성하자.

## 프로젝트에 local repository 생성

프로젝트 소스 코드를 바로 remote repository에 저장하는 것이 아니고,

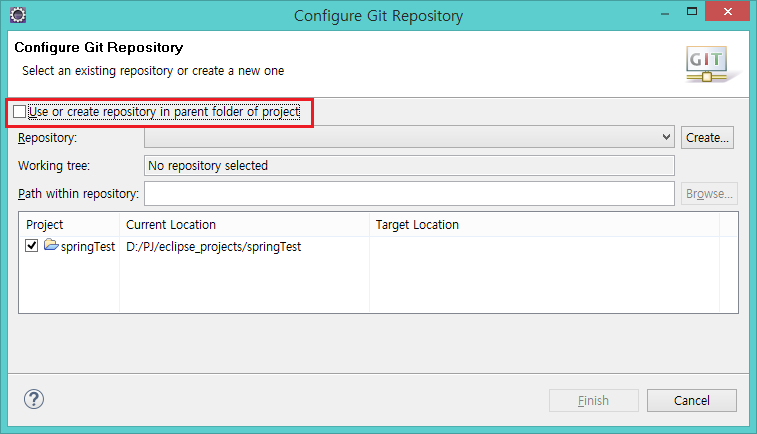
먼저 내 PC의 local repository에 저장한 후에

remote repository에 업로드 해야 한다.

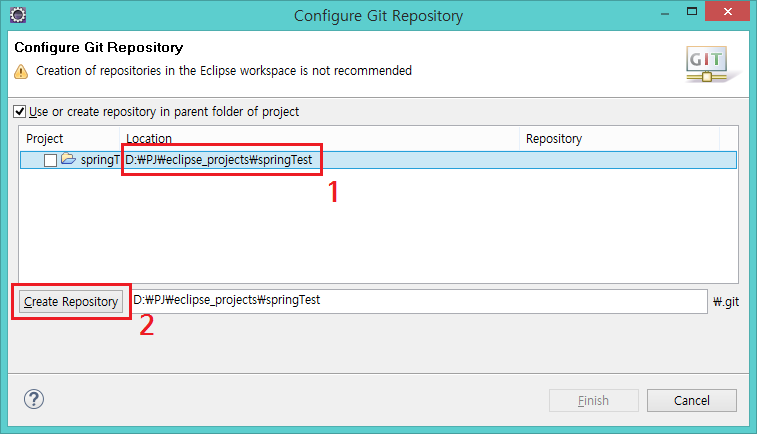
그래서 내 PC에 local repository를 만들어야 한다.

이클립스의 project explorer 창에서

프로젝트를 마우스 오른쪽 버튼으로 클릭 - Team - Share Project

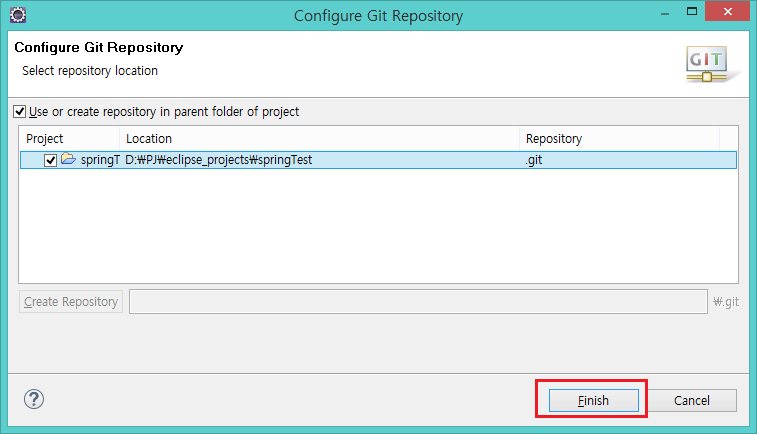


[Use or create repository in parent folder of project] 체크박스를 켠다.



1번 항목을 클릭하여

2번 버튼을 클릭한다.



Finish 버튼을 클릭하면

프로젝트 폴더 아래에 .git 폴더가 생성된다.

이 폴더가 local repository 이다.

## .gitignore 파일 생성

.gitignore 파일에 폴더나 파일의 경로명을 적어주면,

그 폴더와 파일은 repository에 등록되지 않고 무시된다.

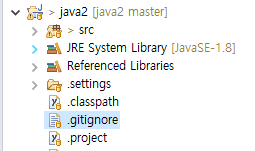
### java 프로젝트인 경우

.gitignore 파일 생성

|  |
| --- |
| /bin/ |

/bin/ 폴더는 Java 소스를 컴파일하여 생성된 \*.class 파일이 저장되는 곳이다.

이 파일들을 repository에 등록할 필요없다.



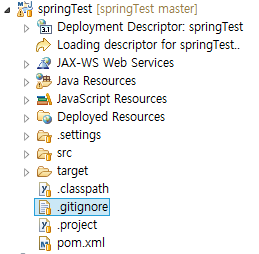
### spring 프로젝트의 경우

.gitignore 파일 생성

|  |
| --- |
| /target/ |

/target/ 폴더는 Java 소스를 컴파일하여 생성된 \*.class 파일이 저장되는 곳이다.

이 파일들을 repository에 등록할 필요없다.

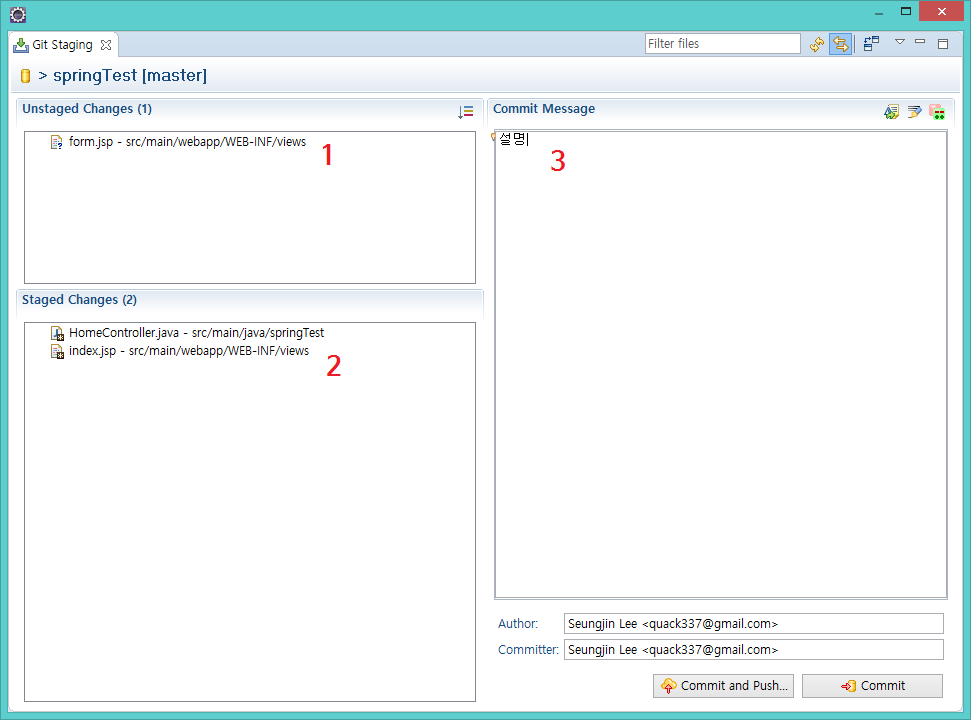


## local repository에 commit

수정된 소스 파일들을 local repository에 등록하는 것을 commit 라고 한다.

이클립스 Project Explorer 창에서

프로젝트를 마우스 오른쪽 버튼으로 클릭 - Team - Commit



파일의 내용이 수정되었기 때문에 repository에 등록해야할 파일 목록이

영역1에 나열된다.

영역2은 repository에 등록할 파일 목록이다.

repository에 등록할 파일들을 영역1에서 영역2로 드래그 드롭해야 한다.

등록하지 않을 특별한 이유가 없다면, 영역1의 모든 파일을 영역2에 드래그 드롭하자.

repository에 등록하지 않아야할 대표적인 파일은

서버 IP 주소에 계정 비밀번호가 등록된 설정파일이다.

영역2에 간단한 설명을 적는다.

[Commit] 버튼을 클릭하면, 영역2에 나열된 파일들이 local repository에 등록된다.

수정된 파일들을 local repository에 등록하는 것을 commit 라고 한다.

[Commit and Push] 버튼을 클릭하면, local repository에 등록되고,

바로 이어서 remote repository에 업로드 된다.

repository에 등록된 내용을 remote repository에 업로드 하는 것을 push 라고 한다.

local repository, remote repository, commit, push 개념을 외우자.

# remote repository에 push

아래 절차는 프로젝트를 github repository에 처음 등록할 때 필요한 작업이다.

프로젝트 팀장이 최초로 한 번만 수행하면 된다.

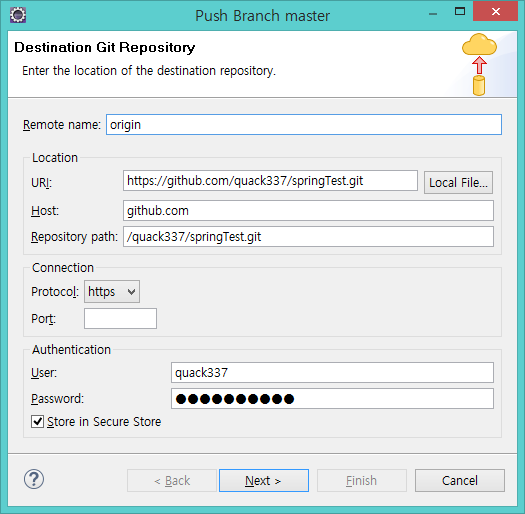
## Push

앞 단계에서 [Commit] 버튼만 클릭했다면, 수정된 파일이 local repository에만 등록(commit)되었을 것이다.

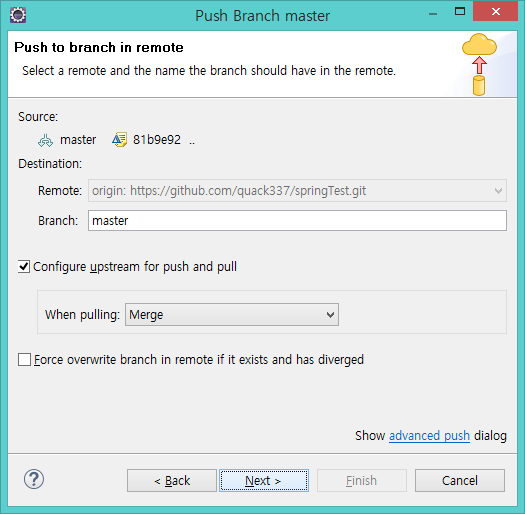
local repository에 등록(commit)된 내용을 remote repository에 등록(push) 하려면

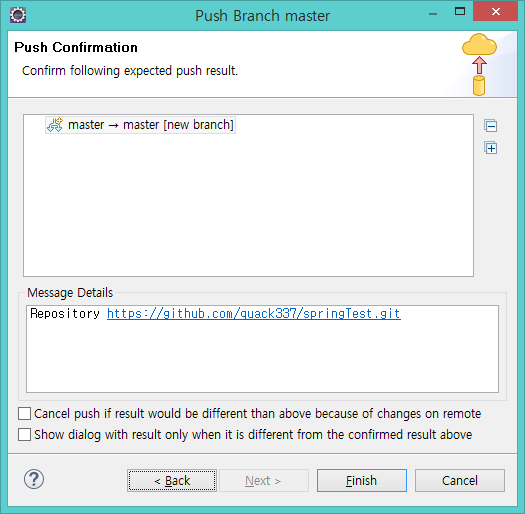
프로젝트를 마우스 오른쪽 버튼으로 클릭 - Team - Remote - Push Branch master 클릭

push를 처음 실행하는 경우에는 아래 화면이 나타난다.



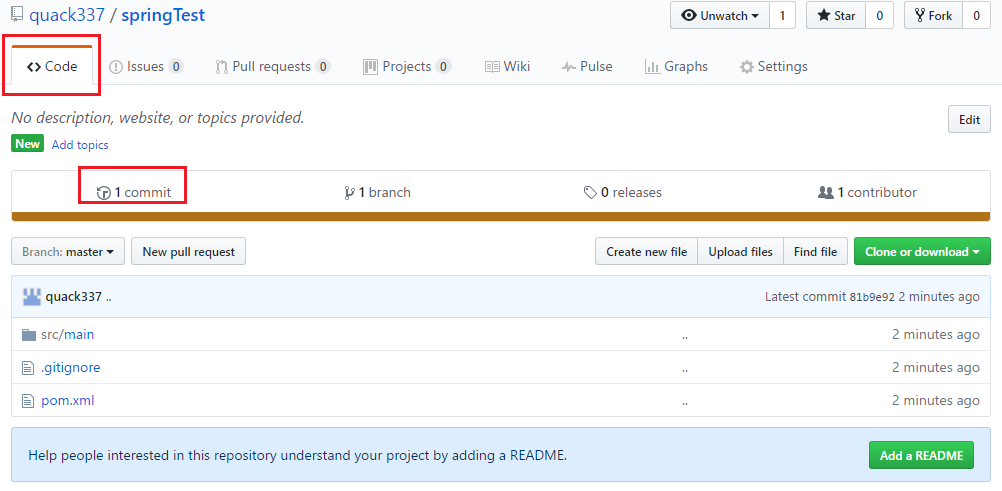
github에 생성한 remote repository의 URL을 입력하자.





Finish 버튼을 클릭하면 remote repository에 업로드가 완료된다.

## Push 결과 확인



위 화면의 Code 탭에서 소스 코드를 탐색할 수 있다.

그리고 위 화면에서 [1 commit] 링크를 클릭하면, remote repository에 push된 commit 목록을 확인할 수 있다.

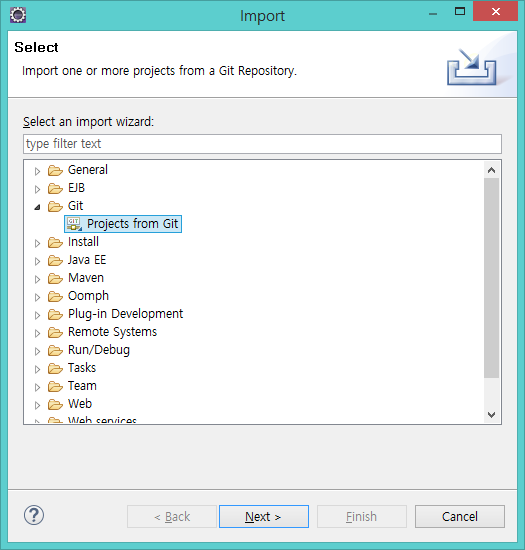
# 프로젝트 가져오기

아래 절차는 팀장이 remote repository에 등록한 프로젝트 소스 코드를 팀원 PC로 가져오는 절차이다.

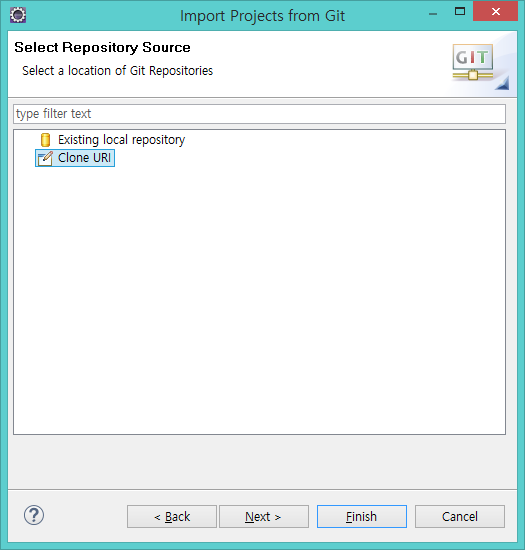
팀장을 제외한 나머지 팀원이 아래의 절차를 수행해야 한다.

## Import

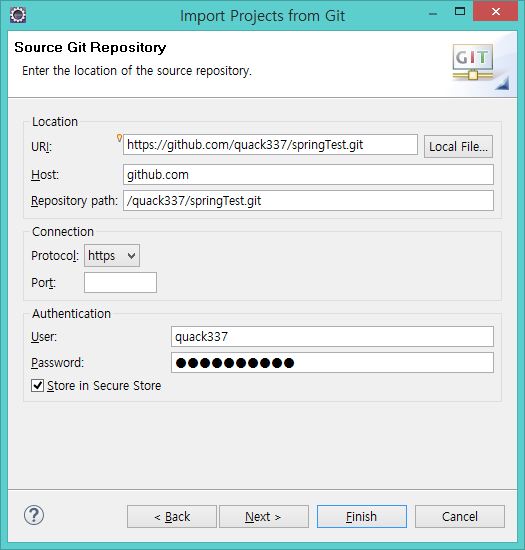
eclipse 메뉴 File - Import



Git - Projects from Git

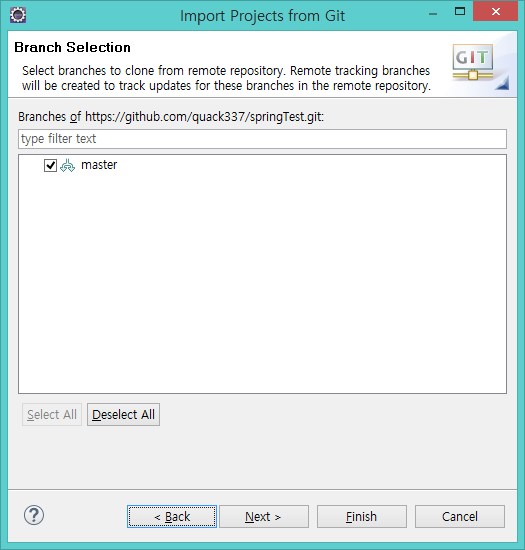


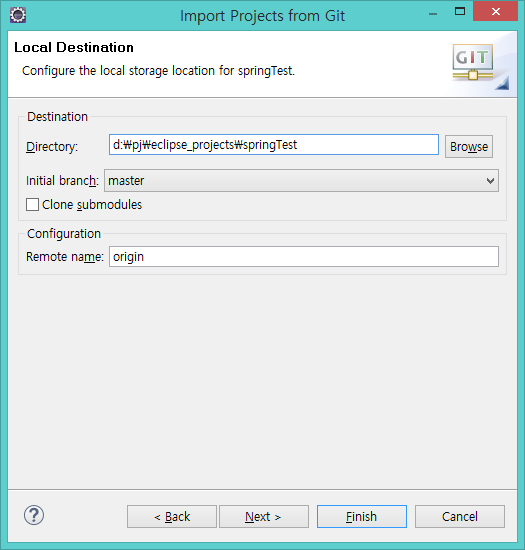
Clone URL



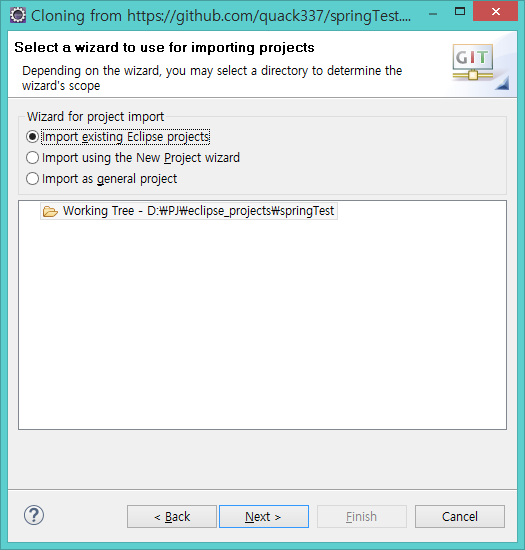
github repository의 URL을 입력한다.

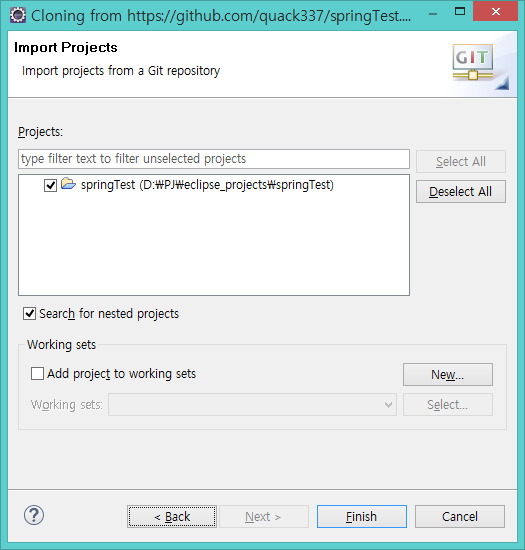
자신의 github 계정을 입력한다.





프로젝트를 저장할 폴더 경로명을 입력한다.





완료!

# 프로젝트 설정 파일 무시

/.settings/ 폴더

/.classpath 파일

/.project 파일

위 폴더와 파일들은 소스 코드가 아니고 프로젝트 설정 파일이다.

팀장이 remote repository에 등록한 프로젝트를 팀원이 처음 가져올 때는 이 설정 파일도 가져와야 하지만,

이제는 이 설정 파일들을 팀원이 서로 공유하지 않는 것이 좋다.

팀원 PC 마다 이 설정이 달라질 수 있기 때문이다.

팀원 PC의 jdk 버전, tomcat 버전 등 개발 환경이 모두 동일하다면,

프로젝트 설정 파일을 공유해도 된다.

이 파일들을 지금부터 팀원들과 공유하지 않으려면,

이 파일 각각에 대해서

파일을 마우스 오른쪽 버튼으로 클릭 - Team - Advanced - Assume Unchanged 메뉴를 클릭한다.

모든 팀원들이 이렇게 해야 한다.

# 수정된 내용 push

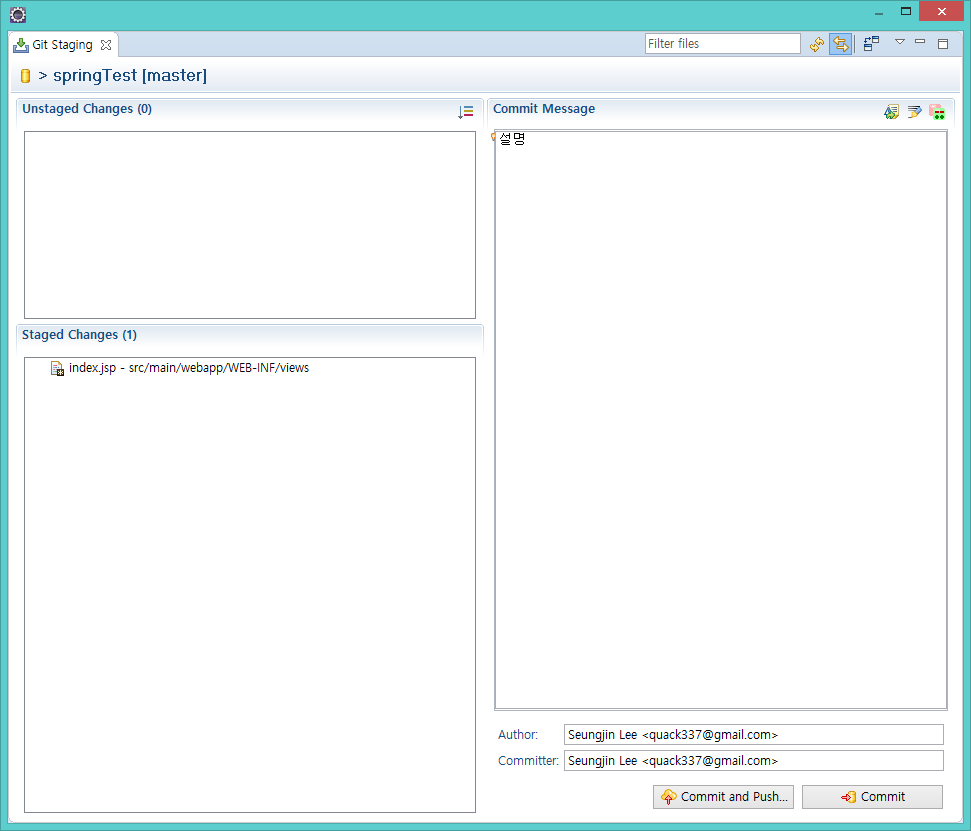
프로젝트 소스 코드를 수정하고, 수정된 내용을 remote repository에 등록해 보자.

## 소스 코드 수정

소스 코드 파일을 생성하거나 수정하자.

## commit

프로젝트 - Team - Commit



Staged Changes 영역의 파일들은 repository에 등록할 파일이다.

Unstaged Changes 영역의 파일은 repository에 등록되지 않기 때문에

Staged Changes 영역으로 드래그 드롭하여 옮겨야 한다.

Commit Message 영역에 간단한 설명을 입력한다.

[Commit] 버튼을 클릭하면 local repository에 등록(commit)된다.

[Commit and Push] 버튼을 클릭하면 먼저 local repository에 등록되고,

이어서 remote repository에 업로드(push) 된다.

commit 할 때마다 push 해야하는 것은 아니다.

어떤 기능을 구현 중일 때는 가끔 commit만 하다가,

구현이 완료되어 팀원들에게 완성된 소스를 공유하고 싶을 때 한 번 push 하면

그동안 commit 되었던 것들이 한 번에 remote repository에 push 된다.

## push

프로젝트 - Team - Push Branch master

# 수정된 내용 pull

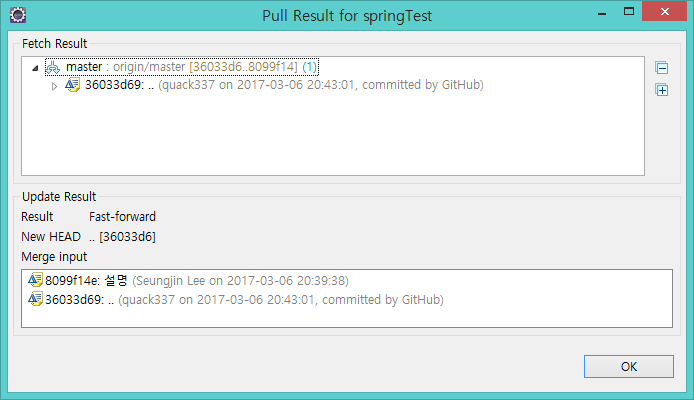
다른 팀원이 remote respository에 등록한 수정 내용을, 내 PC에 가져와야 한다.

remote repository에 등록된 수정 내용을 내 PC에 가져오는 것을 pull 이라고 한다.

프로젝트 - Team - Pull

프로젝트 - Team - Pull...

[Pull] 메뉴와 [Pull...] 메뉴의 기능이 동일하기 때문에 아무거나 선택해도 된다.



pull 된 내용은, 내가 수정한 소스 코드와, 대부분의 경우 자동으로 잘 합쳐진다(merge).

# push 실패

프로젝트 소스 코드를 수정하고, 수정된 내용을 remote repository에 등록하는 것을 push 라고 한다.

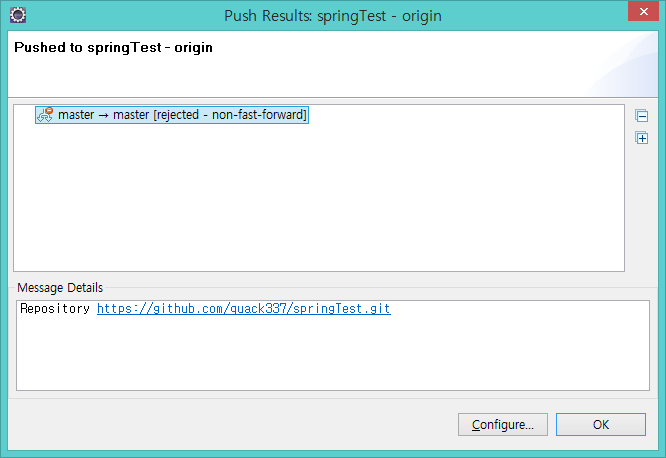
push 작업이 언제나 성공하는 것은 아니다.

내가 프로젝트 소스 코드를 수정하는 사이에,

다른 팀원이 먼저 수정 내용을 remote repository에 push 했다면,

그 뒤를 이어 내가 push 하는 것은 실패한다.

## push 실패



push 실패 화면이다.

위 화면에서 [rejected - non-fast-forward] 부분을 주목하자.

push 작업이 rejected 되었다는 뜻이다.

## pull 먼저

위와 같이 push가 실패하면,

pull을 먼저 하고, 그 다음에 다시 push 해야 한다.

자세히 설명하자면,

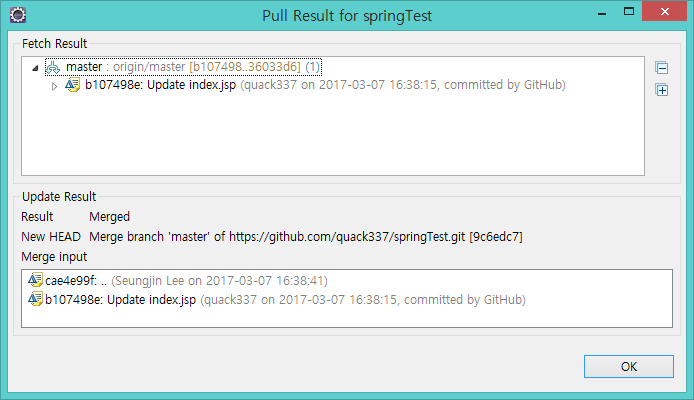
내가 프로젝트 소스 코드를 수정하는 사이에,

다른 팀원이 remote repository에 push 한 수정 내용을

먼저 pull 해야 한다. (가져와야 한다)

그래야 그 다음에 내가 수정한 내용을 push 할 수 있다.

프로젝트 - Team - Pull 메뉴 클릭



pull 성공 화면이다.

위 화면에서 [Result Merged] 부분에 주목하자.

remote repository에서 가져온 수정 내용과, 내가 수정한 내용이 자동으로 잘 합쳐졌다는 뜻이다. (merge 성공)

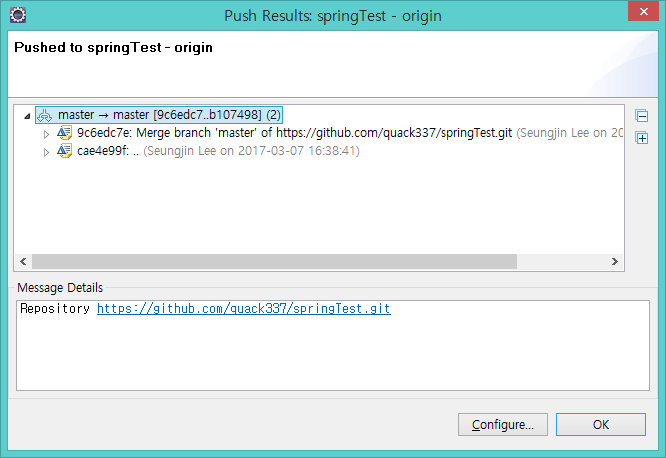
pull을 성공한 다음에

프로젝트 - Team - Push Branch master 메뉴

프로젝트 - Team - Push to upstream 메뉴

위 메뉴 중에 하나를 클릭한다.

위 메뉴 둘 다 push 이다.



push 성공 화면이다.

push 하기 전에 언제나 pull 먼저 하는 것이 좋다.

# merge conflict

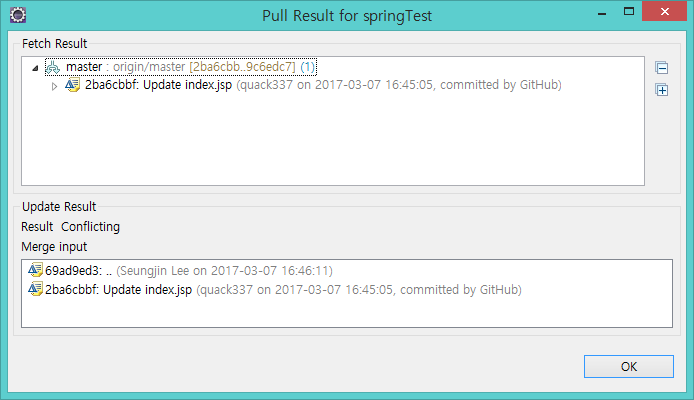
다른 팀원이 remote repository에 push 한 수정 내용을

내 PC에 가져오는 것을 pull 이라고 한다.

pull 된 내용은, 내가 수정한 소스 코드와, 대부분의 경우 자동으로 잘 합쳐진다(merge).

그런데 merge 과정에서 가끔 충돌이 발생하기도 한다. (merge conflict)

## merge conflict



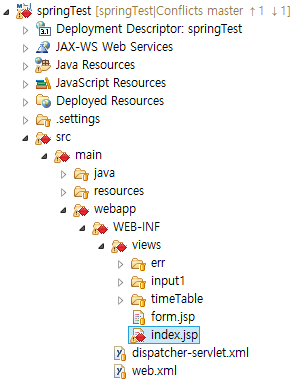
merge conflict 화면이다.

위 화면에서 [Result Conflicting] 부분에 주목하자.

remote repository에서 가져온 수정 내용과, 내가 수정한 내용이 충돌한다는 뜻이다. (merge conflict)

위 화면에서 [Result Merged] 이라고 표시되었다면

remote repository에서 가져온 수정 내용과, 내가 수정한 내용이 자동으로 잘 합쳐졌다는 뜻이다. (merge 성공)



merge conflict 가 발생한 파일과 그 파일을 포함하는 폴더에

빨간색 태그가 붙어있는 아이콘이 표시된다.



충돌(merge conflict)이 발생한 파일을 열어보자.

충돌이 발생한 파일의 내용은 아래와 같은 패턴으로 수정되어 있을 것이다.

|  |
| --- |
| <<<<<<< HEAD  내가 수정한 내용  =======  remote repository에서 가져온 수정 내용  >>>>>>> branch 'master' of https://github.com/quack337/springTest.git |

소스 코드에서 동일한 부분을 나도 수정했고 팀 동료도 수정한 경우에

위와 같은 충돌(merge conflict)이 발생한다.

|  |
| --- |
| <<<<<<< HEAD  내가 수정한 내용  ======= |

이 부분은 내가 수정한 내용이고, 아직 push 되지 않았다.

|  |
| --- |
| =======  remote repository에서 가져온 수정 내용  >>>>>>> branch 'master' of https://github.com/quack337/springTest.git |

이 부분은 팀 동료가 수정한 내용이고, 이미 push 되었고, 내 PC로 pull 되었다.

팀 동료가 수정한 내용과 내가 수정한 내용을 하나로 잘 합쳐야 한다.

## merge conflict가 발생한 이유

### 내가 수정한 내용 (아직 내가 push 안 했음)

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  <html>  <body>  <h2>Hello World!</h2>  <a href="/springTest/timeTable.do">timeTable.do</a> <br />  <a href="/springTest/input1/dataInput.do">dataInput.do</a> <br />  <a href="/springTest/input1/dataListInput.do">dataListInput.do</a> <br />  <a href="/springTest/err/index.do">err/index.do</a> <br />  hello world<br />  [${ data.id }] <br />  [${ msg }] <br />  <c:set var="data2" value="${ data }" />  [${ data2 }] <br />  [${ data2.id2 }] <br />  good afternoon <br />  </body>    </html> |

### 팀 동료가 수정한 내용 (이미 팀 동료가 push 했고, 내가 pull 해서 가져왔음)

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  <html>  <body>  <h2>Hello World!</h2>  <a href="/springTest/timeTable.do">timeTable.do</a> <br />  <a href="/springTest/input1/dataInput.do">dataInput.do</a> <br />  <a href="/springTest/input1/dataListInput.do">dataListInput.do</a> <br />  <a href="/springTest/err/index.do">err/index.do</a> <br />  hello world<br />  [${ data.id }] <br />  [${ msg }] <br />  <c:set var="data2" value="${ data }" />  [${ data2 }] <br />  [${ data2.id2 }] <br />  good evening<br />  </body>    </html> |

### 충돌이 표시되는 형태

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  <html>  <body>  <h2>Hello World!</h2>  <a href="/springTest/timeTable.do">timeTable.do</a> <br />  <a href="/springTest/input1/dataInput.do">dataInput.do</a> <br />  <a href="/springTest/input1/dataListInput.do">dataListInput.do</a> <br />  <a href="/springTest/err/index.do">err/index.do</a> <br />  hello world<br />  [${ data.id }] <br />  [${ msg }] <br />  <c:set var="data2" value="${ data }" />  [${ data2 }] <br />  [${ data2.id2 }] <br />  <<<<<<< HEAD  good afternoon <br />  =======  good evening <br />  >>>>>>> branch 'master' of https://github.com/quack337/springTest.git  </body>    </html> |

## 충돌 해결

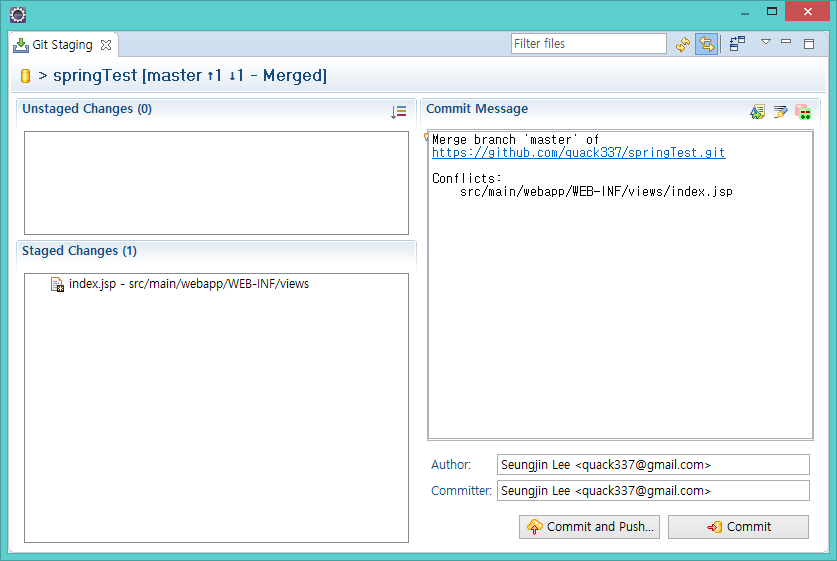
팀 동료가 수정한 내용과 내가 수정한 내용을 하나로 잘 합쳐야 한다.

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  <html>  <body>  <h2>Hello World!</h2>  <a href="/springTest/timeTable.do">timeTable.do</a> <br />  <a href="/springTest/input1/dataInput.do">dataInput.do</a> <br />  <a href="/springTest/input1/dataListInput.do">dataListInput.do</a> <br />  <a href="/springTest/err/index.do">err/index.do</a> <br />  hello world<br />  [${ data.id }] <br />  [${ msg }] <br />  <c:set var="data2" value="${ data }" />  [${ data2 }] <br />  [${ data2.id2 }] <br />  good day <br />  </body>    </html> |

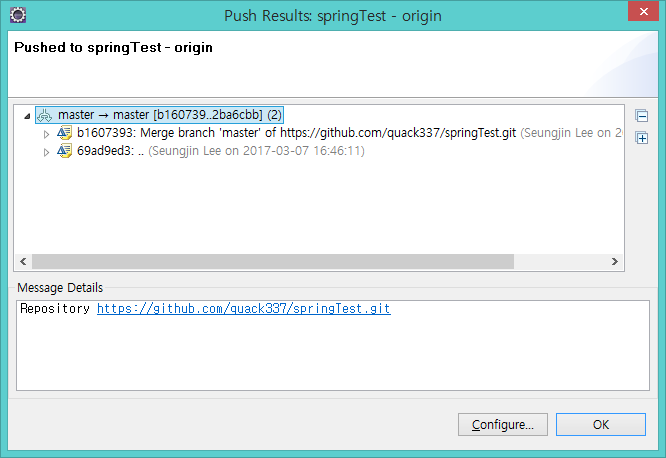
## 다시 commit & push

충돌이 발생한 파일을 잘 수정한 후에, 수정된 내용을 commit 하고 push 해야 한다.

프로젝트 - Team - Commit 메뉴



[Commit and Push] 버튼을 클릭



push 성공

# 바람직한 습관

## 잘 실행되는 것을 확인하고 push 해야 함

내가 구현한 코드를 팀 동료들에게 전달하는 것이 push 이다.

내가 구현한 기능이 이상 없이 잘 실행되는 것을 확인하고 push 해야 한다.

에러 없이 잘 실행되는 코드만 push 해야 한다.

## 가급적 자주 push

내가 push한 코드를 팀 동료들이 pull 할 때, merge conflict가 발생할 수 있다.

아주 오래간만에 push 하는 것이라면,

그래서 여기저기 잔뜩 수정한 다음 push 하는 것이라면,

merge conflict가 발생할 확률이 높다.

merge conflict 해결이 성가시기 때문에,

가급적 자주 push 하는 것이 바람직하다.

너무 큰 기능을 오래 동안 구현하고, 마지막에 한 번 push 하는 것은 바람직하지 않다.

큰 기능 하나를,

3~4일 정도면 구현할 수 있는 작은 기능들로 나누고,

그렇게 나뉜 작은 기능 하나의 구현을 3~4일 동안 완료하고,

잘 실행되는 것을 확인하고,

push 한다.

## 가급적 자주 pull

팀 동료가 push 했다면,

가급적 빨리 내 PC에 pull 해서, 내가 수정한 코드와 merge하는 것이 좋다.

미루다가 나중에 한 번에 몰아서 pull을 하면

merge conflict가 발생할 확률이 높기 때문이다.

# eclipse에서 git 아이콘

## commit 전과 후 아이콘 차이

|  |  |
| --- | --- |
| commit 전수정된 파일 옆에 > 표시.아직 서버에 등록되지 않은 파일에는? 표시. | commit 후commit 성공 후에 아이콘이 위와 같이 변경됨. |

## Assume Unchanged

|  |
| --- |
| .settings/  .classpath  .project |

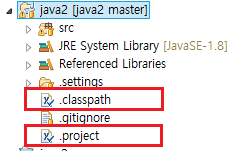
위 파일들을 프로젝트 설정 파일이다.

프로젝트 설정 파일은 서버에 commit, push 하지 않는 것이 좋다.

위 파일들을 선택하고

Team - Advanced - Assume Unchanged 메뉴를 클릭하면

위 그림과 같이 아이콘이 변한다.



이렇게 하면 이 파일들은 서버에 commit, push 되지 않는다.