

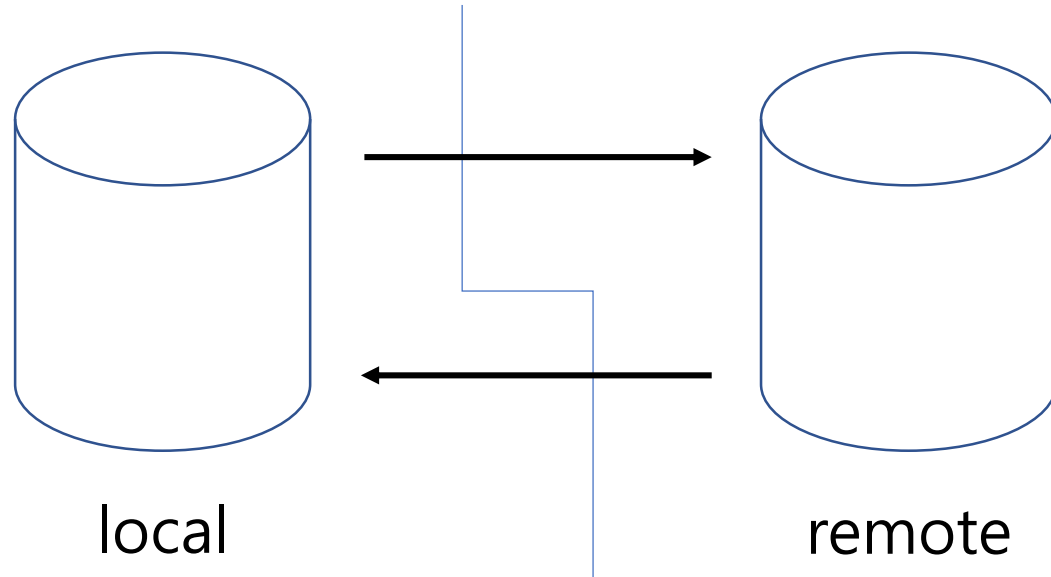
Git

/w SourceTree

깃(Git)이란?

깃(Git)은 프로그램 등의 **소스 코드 관리**를 위한
분산 버전 관리 시스템이다.

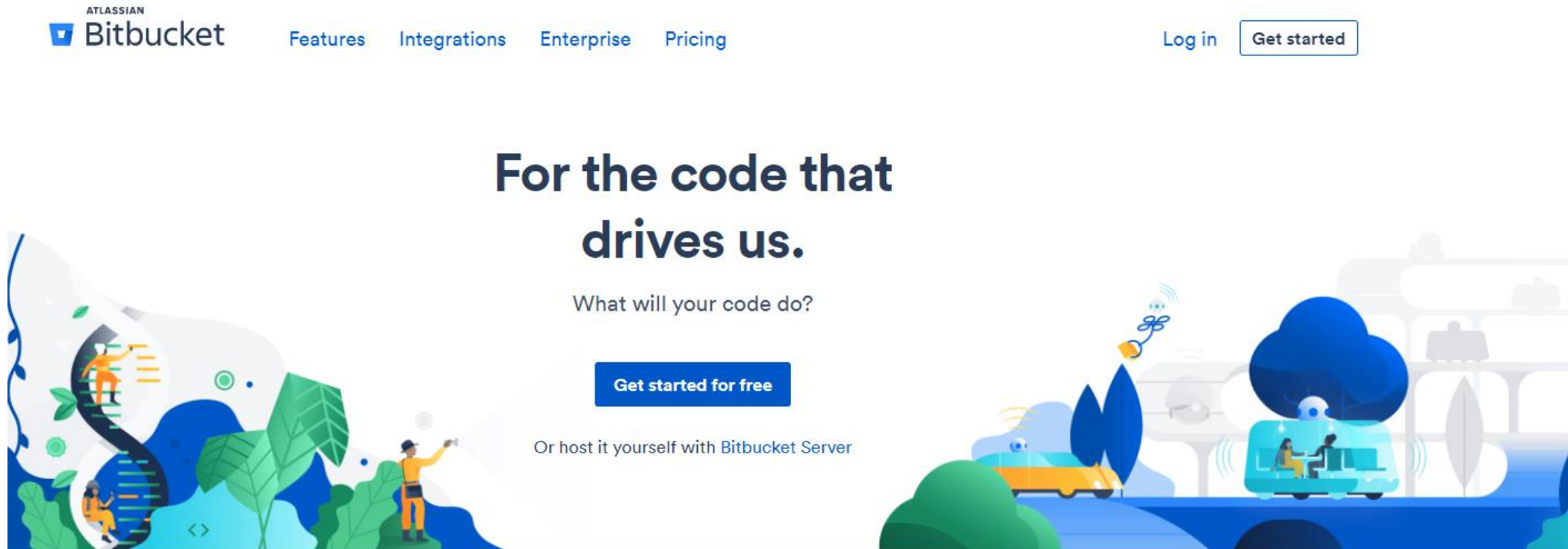
local과 remote



local과 remote로 나누어, local을 변경하고 remote에 올리거나
혹은 remote의 변경상태를 자신의 local에 반영하거나 한다

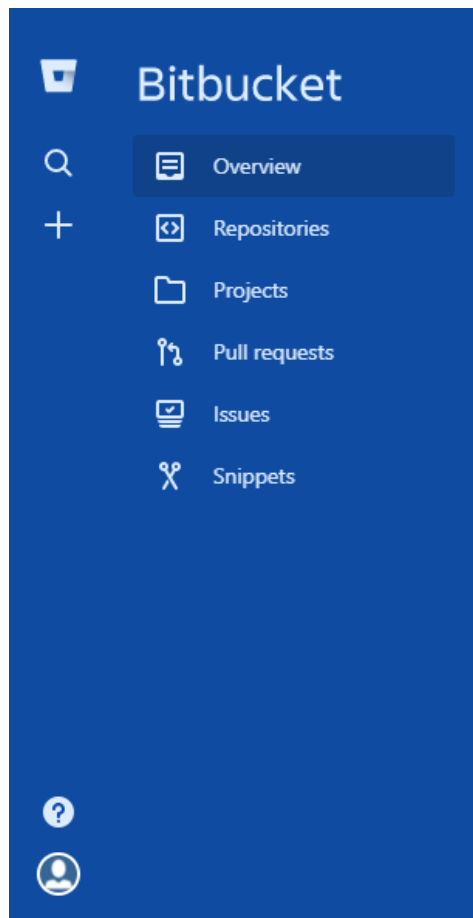
Bitbucket 가입

Bitbucket 가입



<https://bitbucket.org/>로 접속한다

Bitbucket 메인 페이지



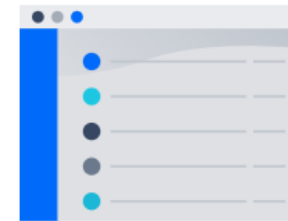
Dashboard



Everything's awesome!

All your pull requests and reviews are done and dusted.

[View all pull requests](#)



Your repository list is empty

Repositories that you own or watch will show up here.

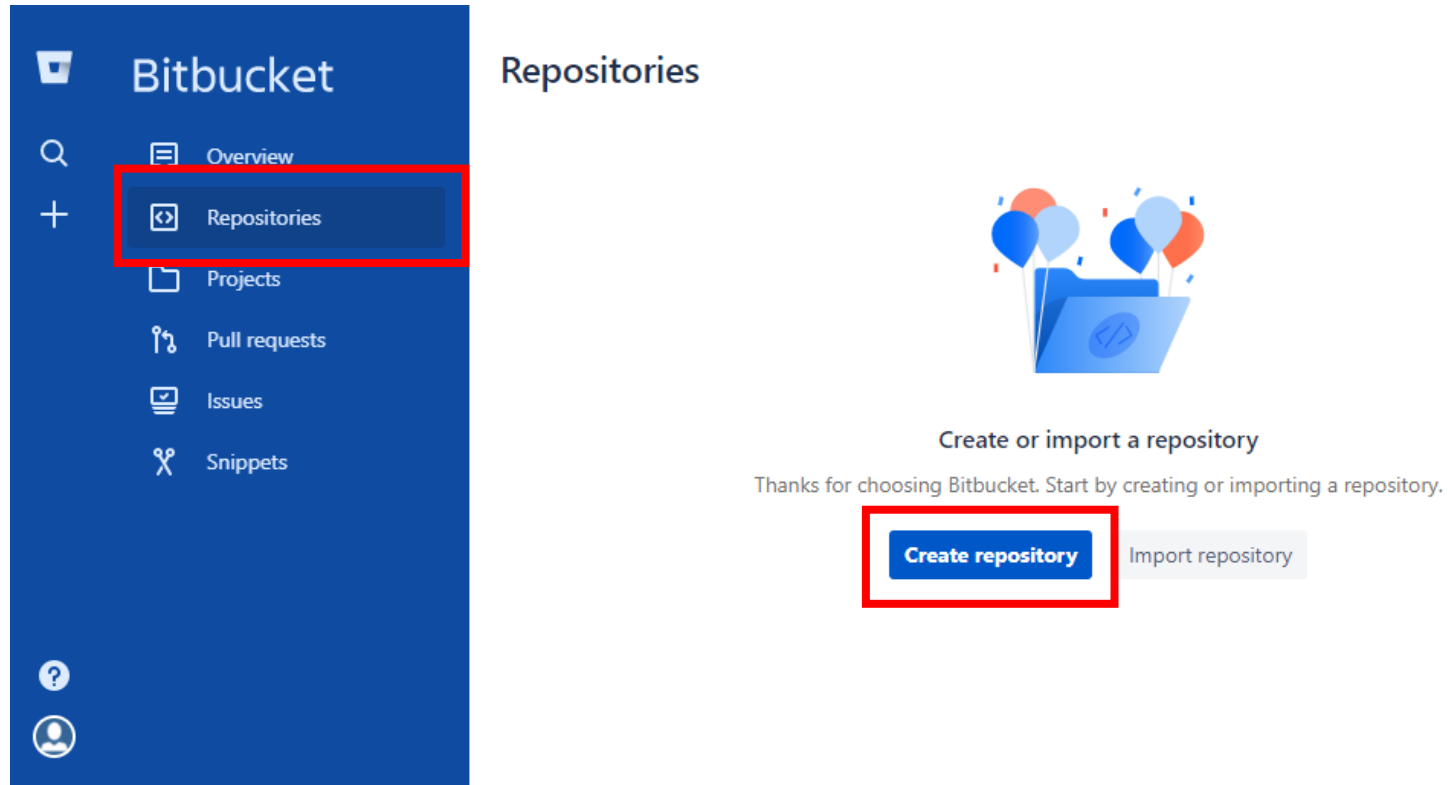
[Create a repository](#)

저장소 생성

저장소(Repository)란?

저장소(Repository)란 파일이나 폴더를 저장해 두는 곳.
Git 저장소는 파일이 변경 이력 별로 구분되어 저장된다.

저장소 생성

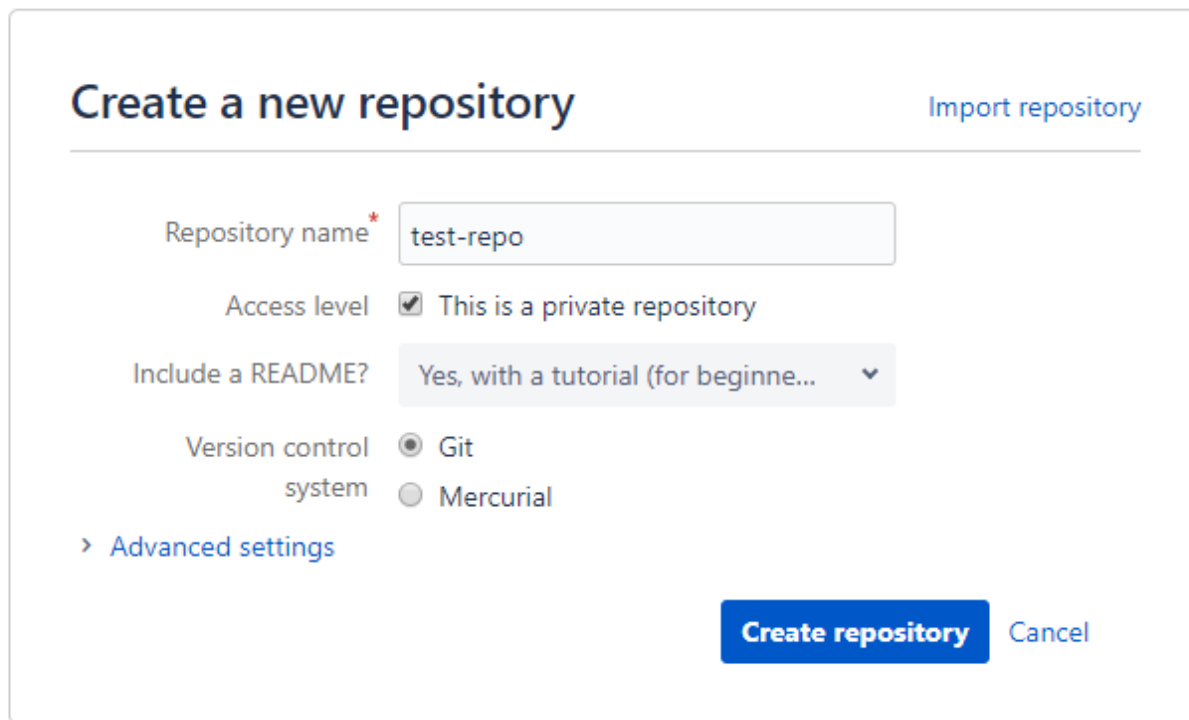


왼쪽 메뉴 중 **Repositories**으로 접속하여
Create Repository를 클릭

저장소 생성

- Repository name (필수)
: 저장소의 이름
- Access Level
: 저장소 공개 여부 (private은 개인용)
- Include a README?
: README 파일을 만들 것인가
- Version control
: 버전 관리를 무엇으로 할 것인가?
: **Git으로 설정**

=> 다 되면 Create repository 클릭



The screenshot shows the 'Create a new repository' dialog box. At the top, there's a title 'Create a new repository' and a link 'Import repository'. Below the title, there are several form fields: 'Repository name' with a text input containing 'test-repo', 'Access level' with a checked checkbox 'This is a private repository', 'Include a README?' with a dropdown menu showing 'Yes, with a tutorial (for beginne...', and 'Version control system' with radio buttons for 'Git' (selected) and 'Mercurial'. There is also a link '> Advanced settings'. At the bottom right, there are two buttons: 'Create repository' (blue) and 'Cancel'.

저장소 생성

The screenshot shows the Bitbucket interface for a newly created repository named 'test-repo'. On the left is a blue sidebar with navigation icons and a menu. The main content area is titled 'Overview' and includes a folder icon, instructions on how to clone the repository, a terminal snippet for the 'git clone' command, and a table of repository statistics. A 'Share' button and a user invitation panel are also visible.

test-repo

Overview

Take the next steps for this new repository and its freshly added files

Sync the repository locally so that you can push any updates you make and pull changes others make. To sync, enter **git clone** and the repository URL at your command line:

```
git clone https://[redacted]@bitbucket.org/[redacted]/test-repo.git
```

[Learn more](#) or [clone in Sourcetree](#) to avoid the command line. [Sourcetree](#) is a free Git and Mercurial client.

HTTPS [https://\[redacted\]@bitbucket.org/](#) [Share](#)

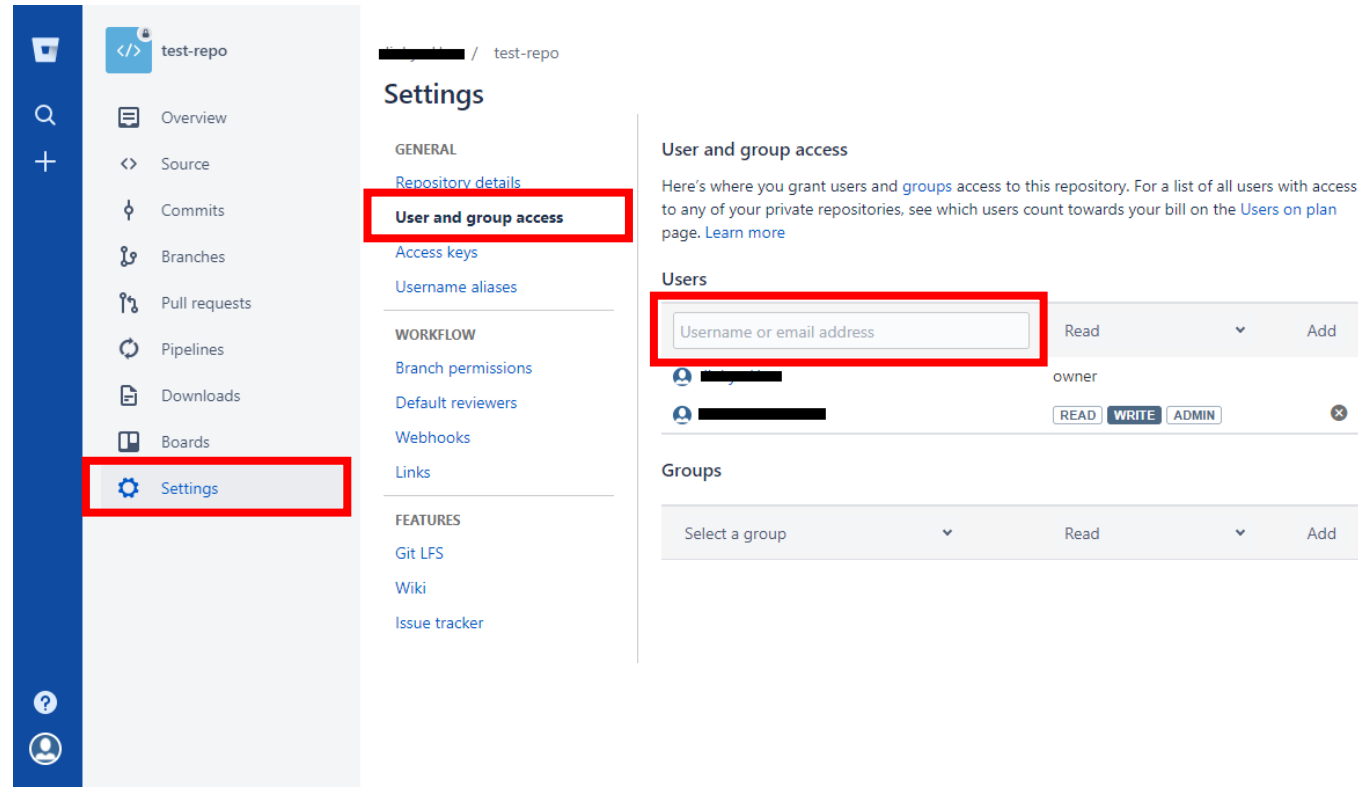
Last updated just now	0	1
Access level Admin	Open PRs	Watcher
	1	0
	Branch	Forks

Invite users to this repo

[Send invitation](#)

Repository가 완성되었다

저장소에 유저 추가



저장소 생성 후에 Settings에 가면 다른 유저를 추가할 수 있다.
추가하면 그 유저에게도 현재의 repository가 보일 것이다.

SourceTree 설치

SourceTree 다운로드

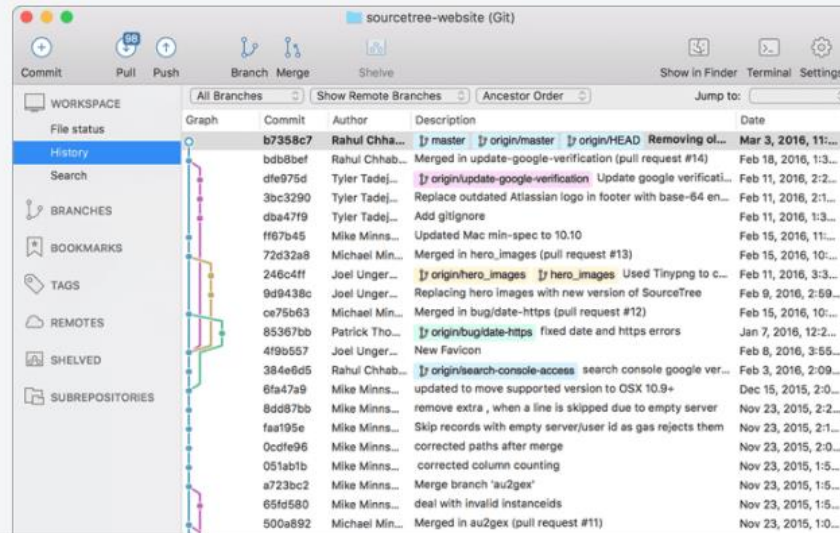
 Sourcetree

[Download free](#)

**Simplicity and
power in a
beautiful Git GUI**

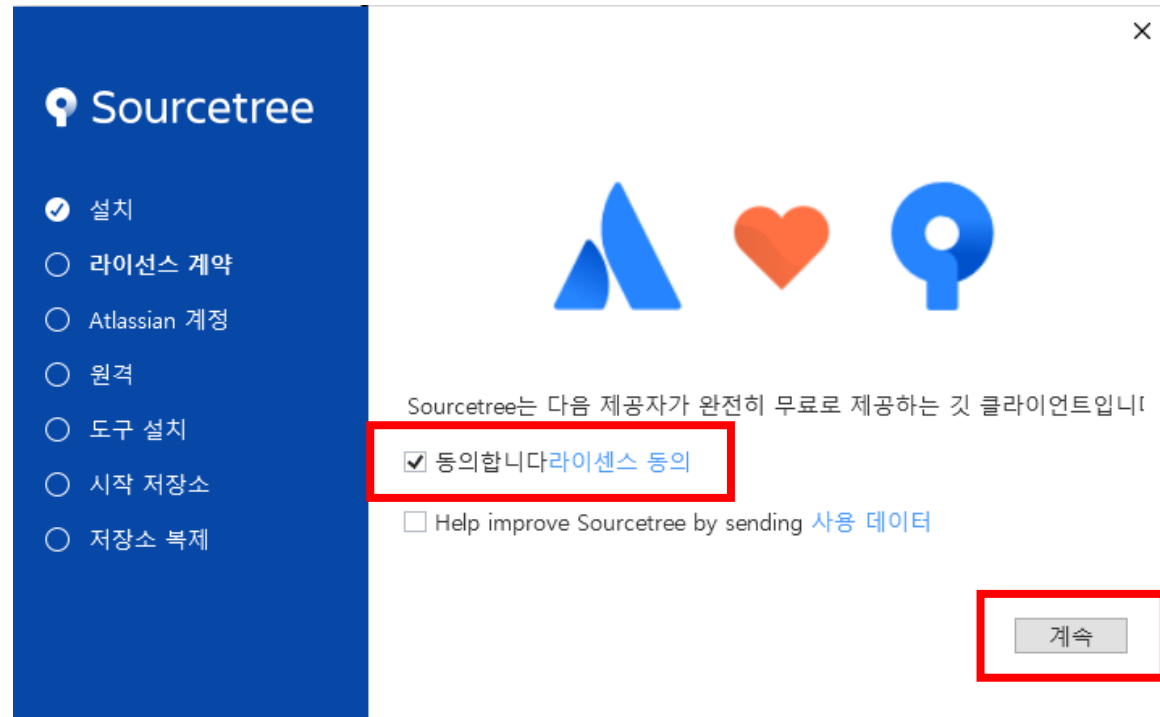
[Download for Windows](#)

Also available for Mac OS X



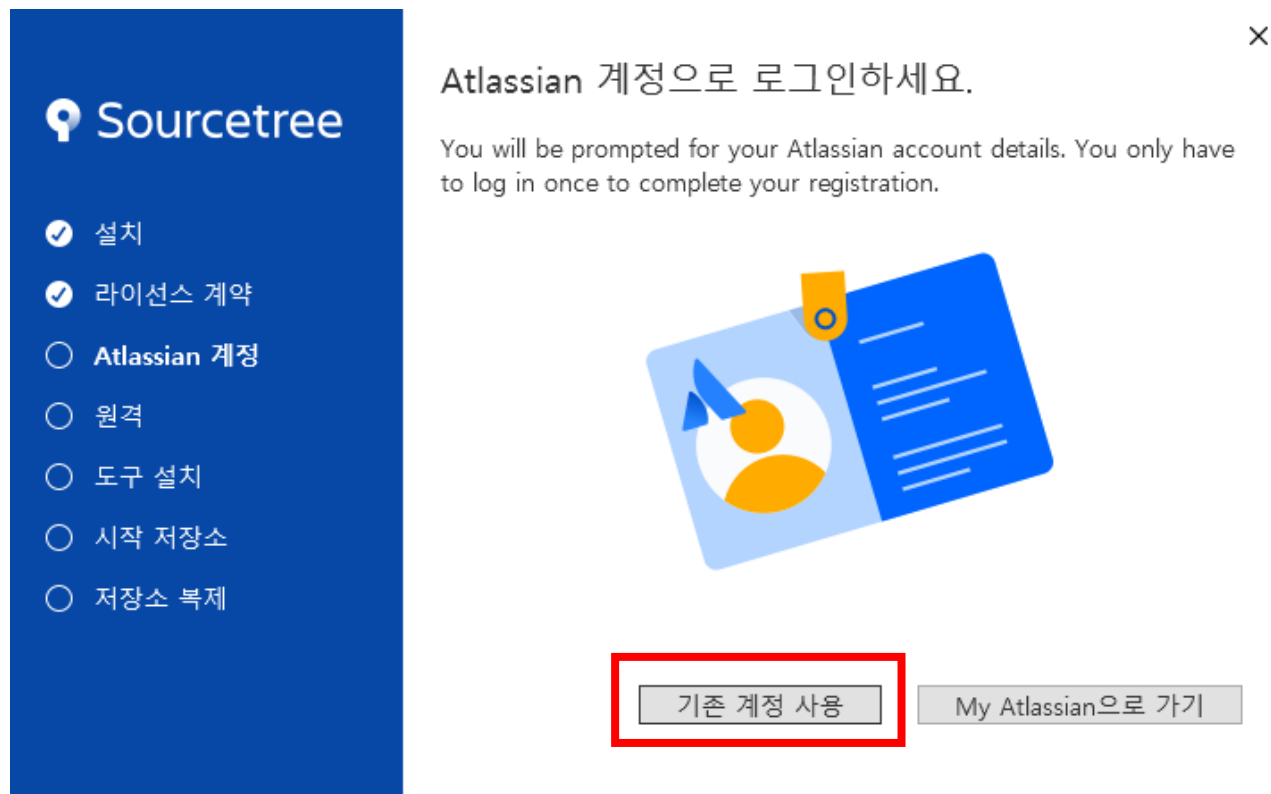
<https://www.sourcetreeapp.com/>

SourceTree 설치



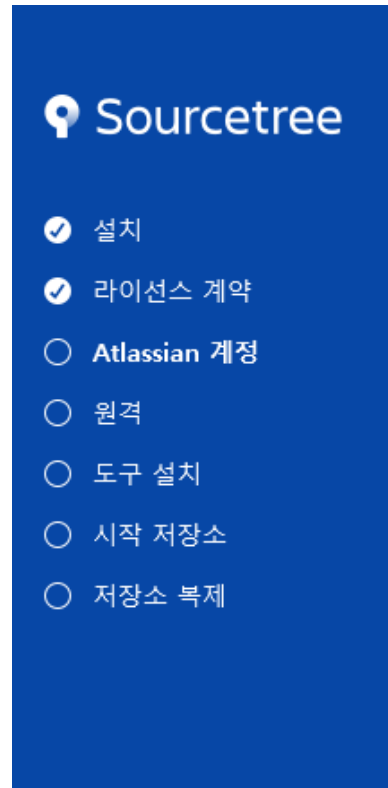
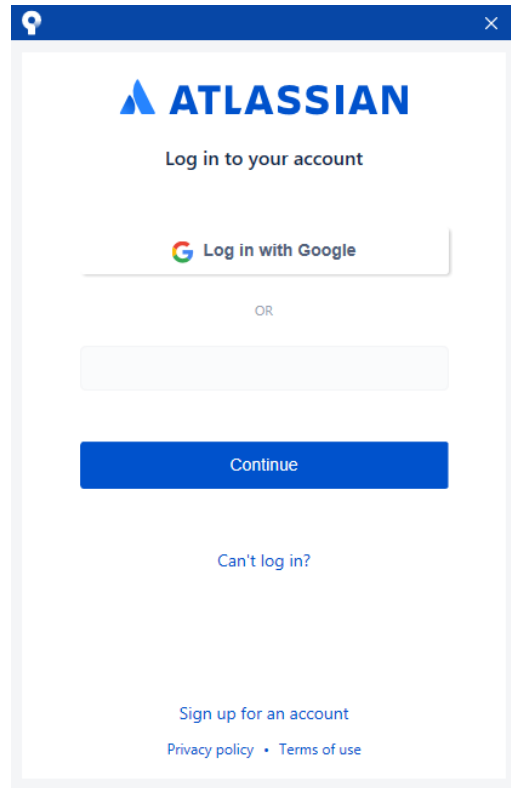
설치를 시작

SourceTree 설치



계정 가입을 했으므로 [기존 계정 사용] 클릭

SourceTree 설치



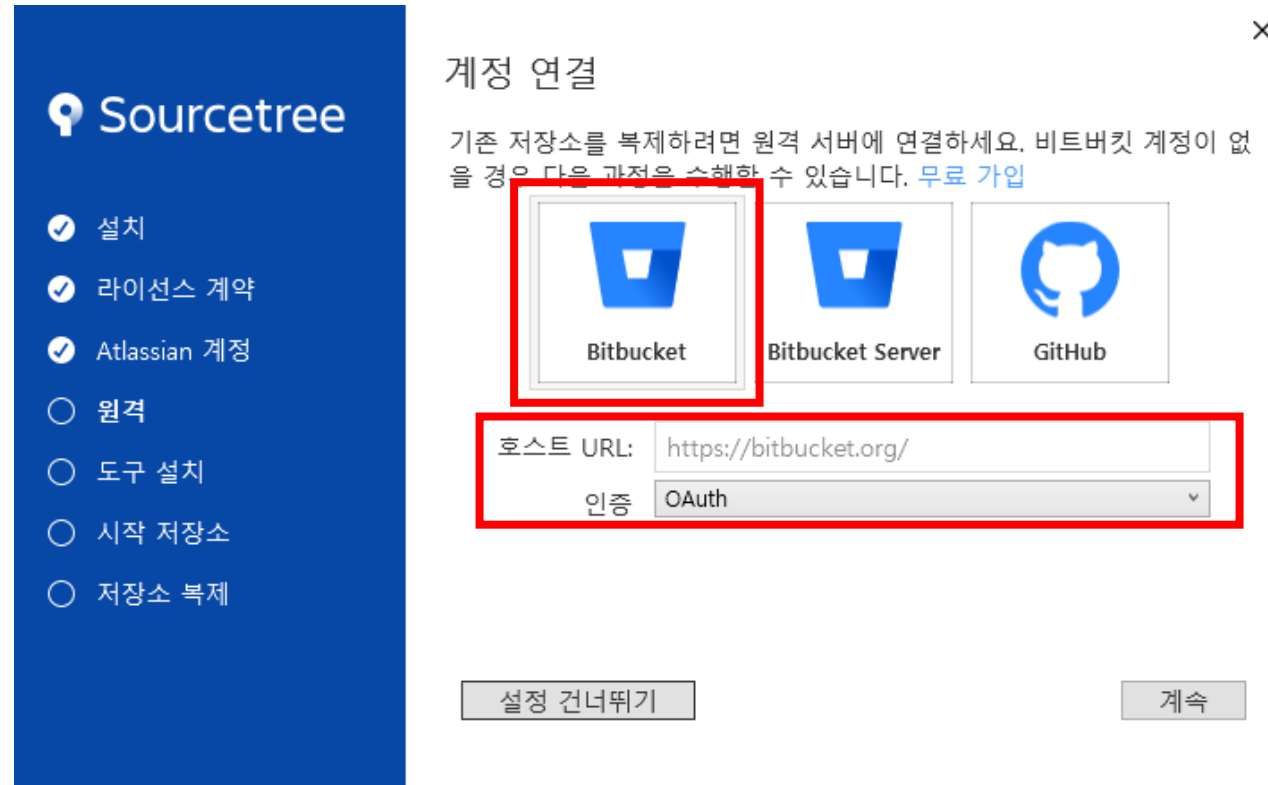
등록 완료!



계속

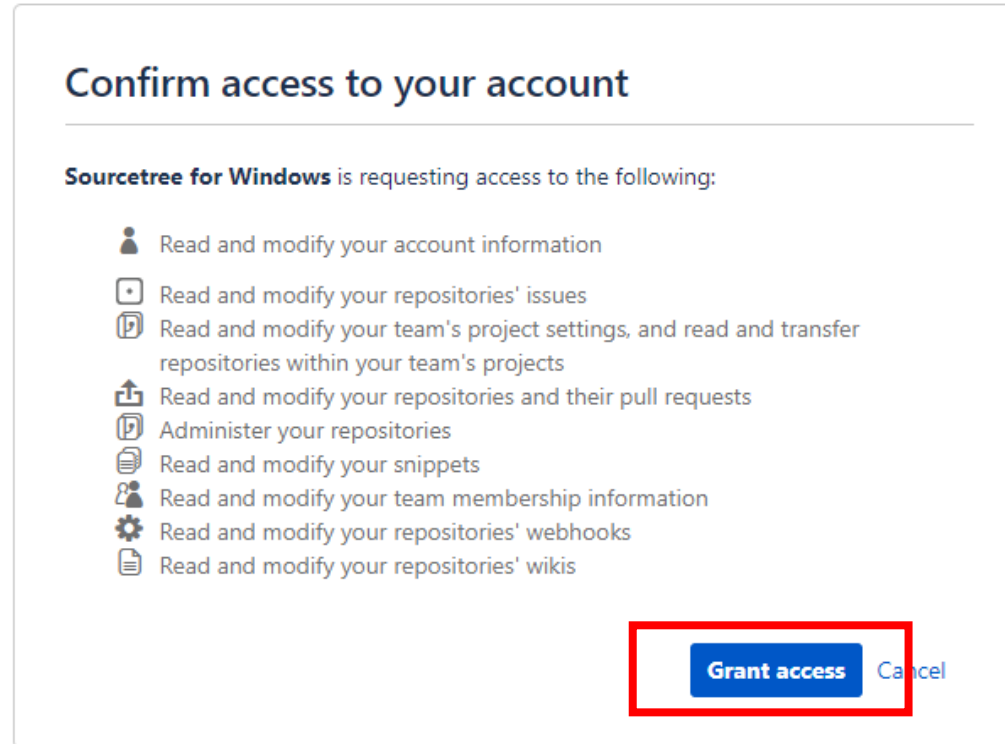
가입했던 계정으로 로그인하면 오른쪽 화면을 볼 수 있다

SourceTree 설치



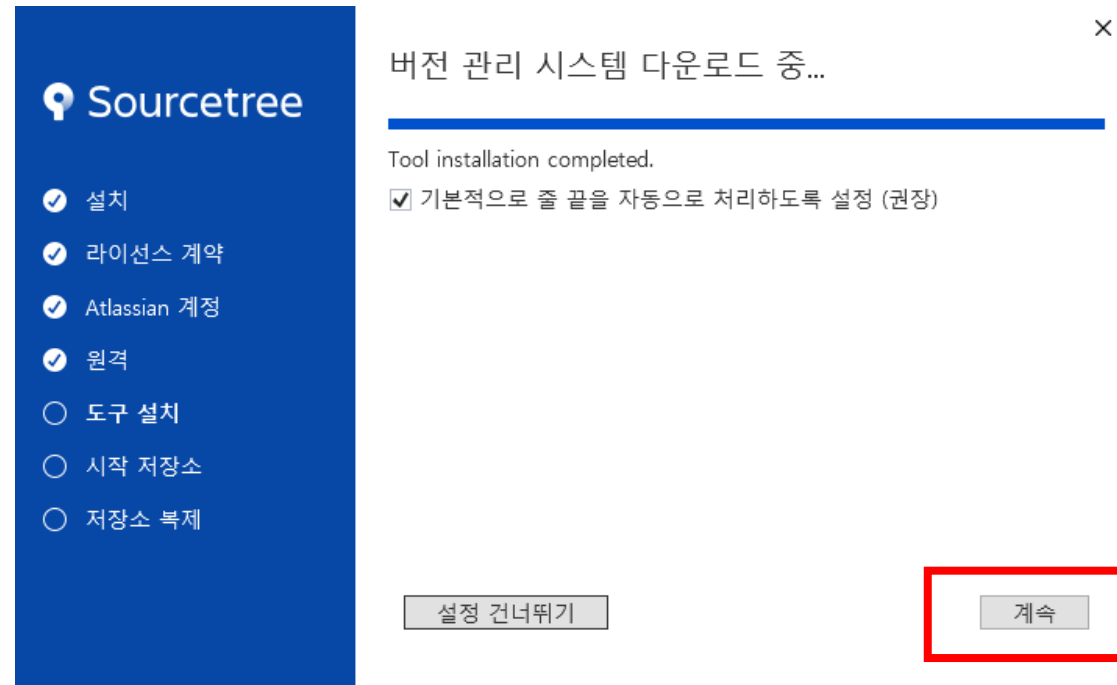
Bitbucket을 선택하고 인증을 OAuth로 선택

SourceTree 설치



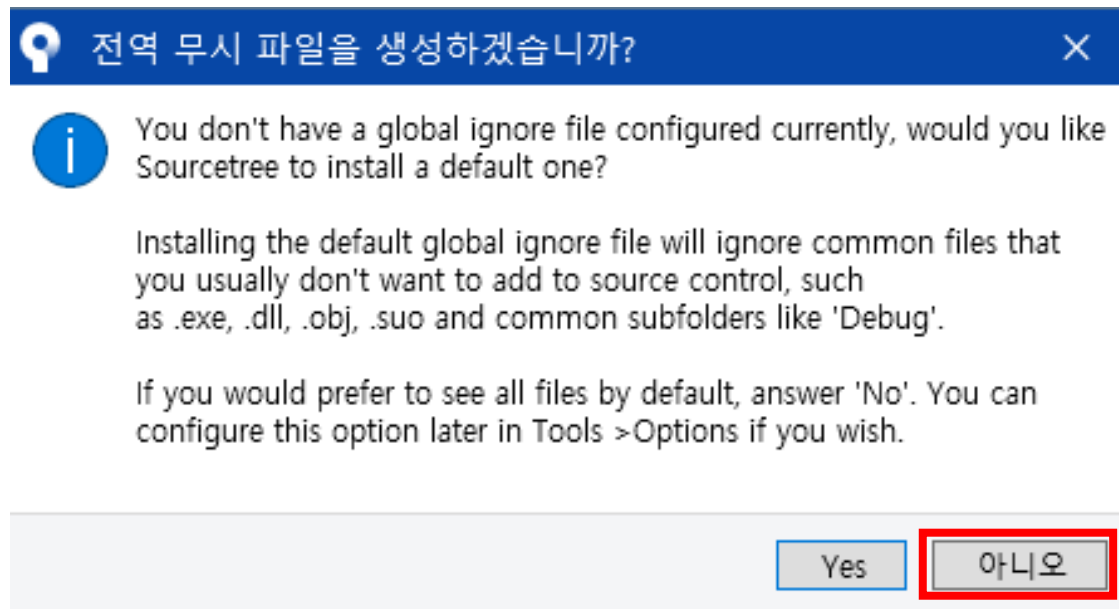
브라우저 상에서 이러한 창이 뜨면 [Grant access]를 클릭

SourceTree 설치



다운로드가 완료되면 [계속]을 누른다

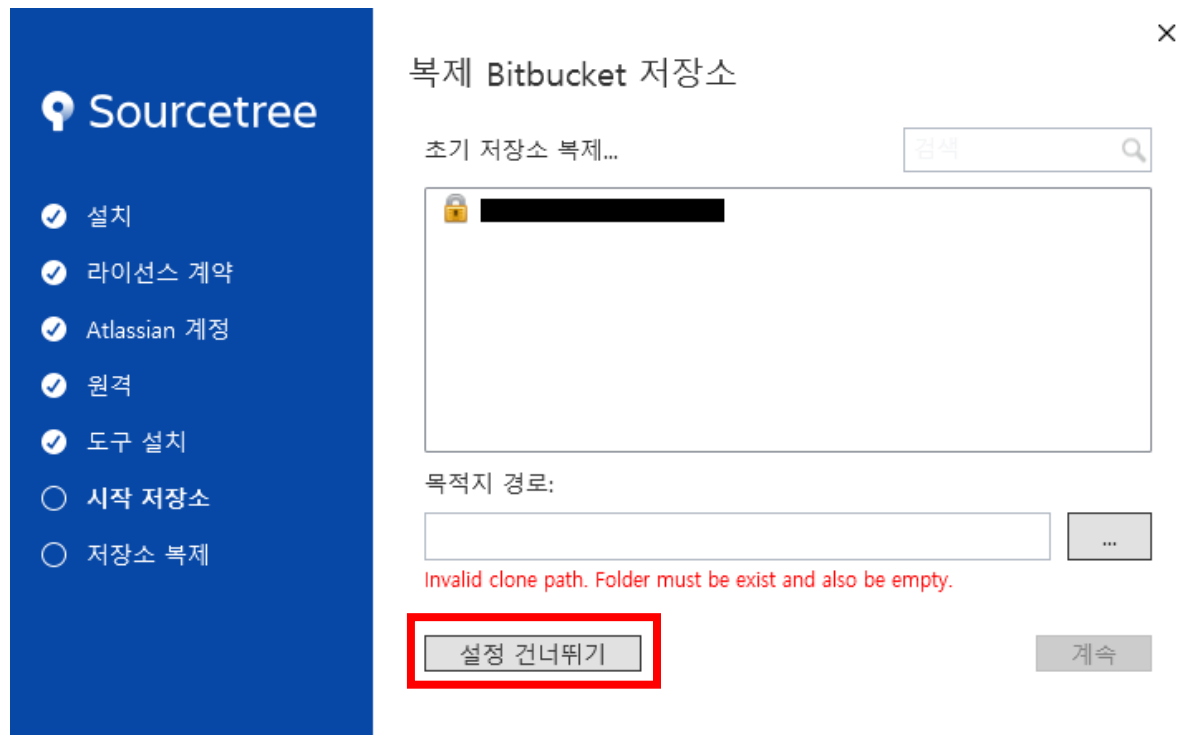
SourceTree 설치



전역 무시 파일을 생성할 것인가에서 No를 선택

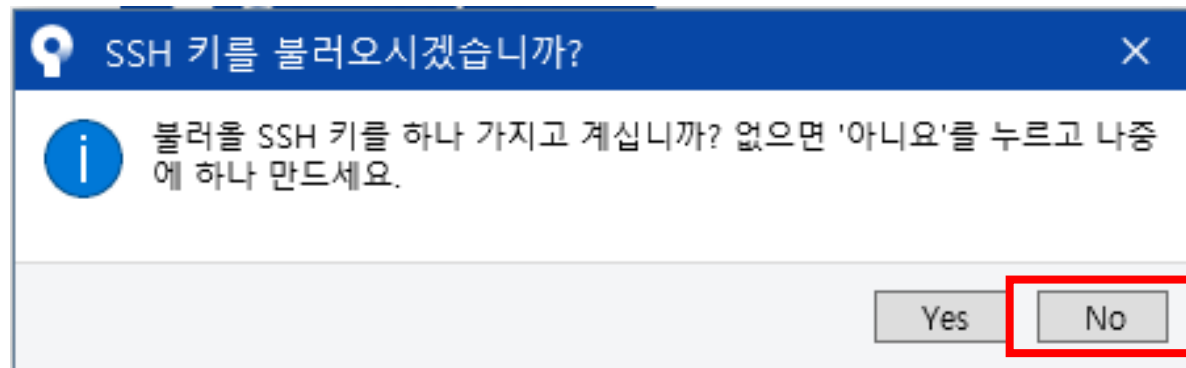
** 모든 Git 저장소에서 무시할 파일이 있다면 Yes를 선택해도 된다

SourceTree 설치



현재 단계에서는 [설정 건너뛰기]를 클릭
** 만약 바로 복제할 저장소가 있다면 저장소를 선택

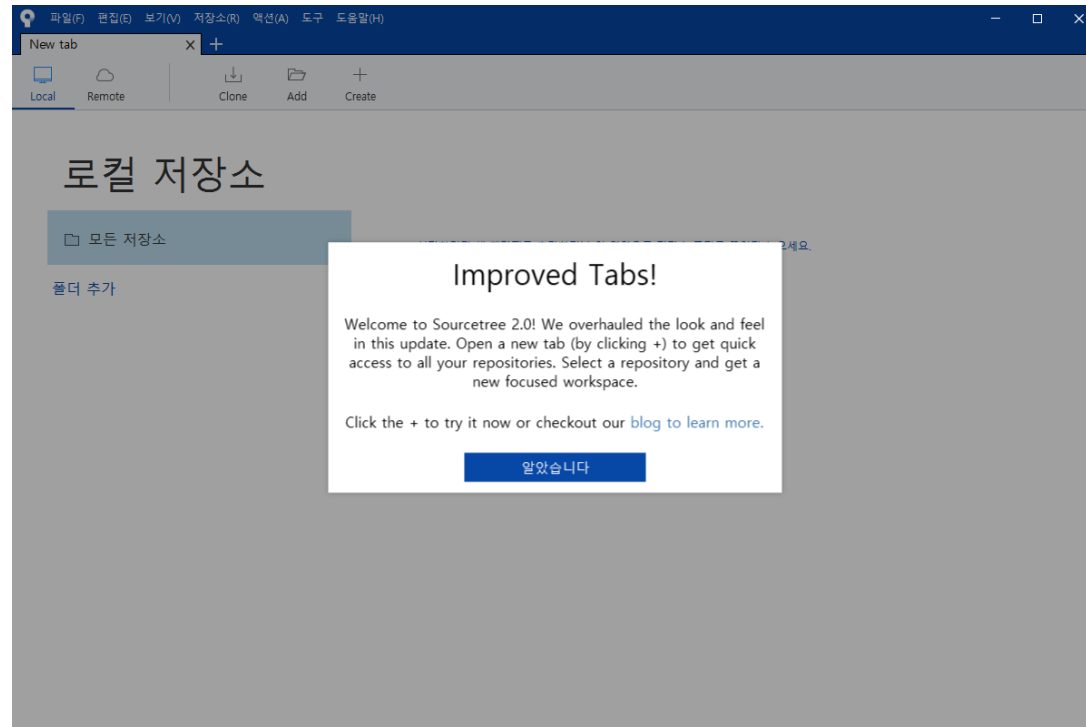
SourceTree 설치



SSH 키가 없을 것이므로 No를 선택

** 키가 이미 있다면 Yes를 선택해도 괜찮다

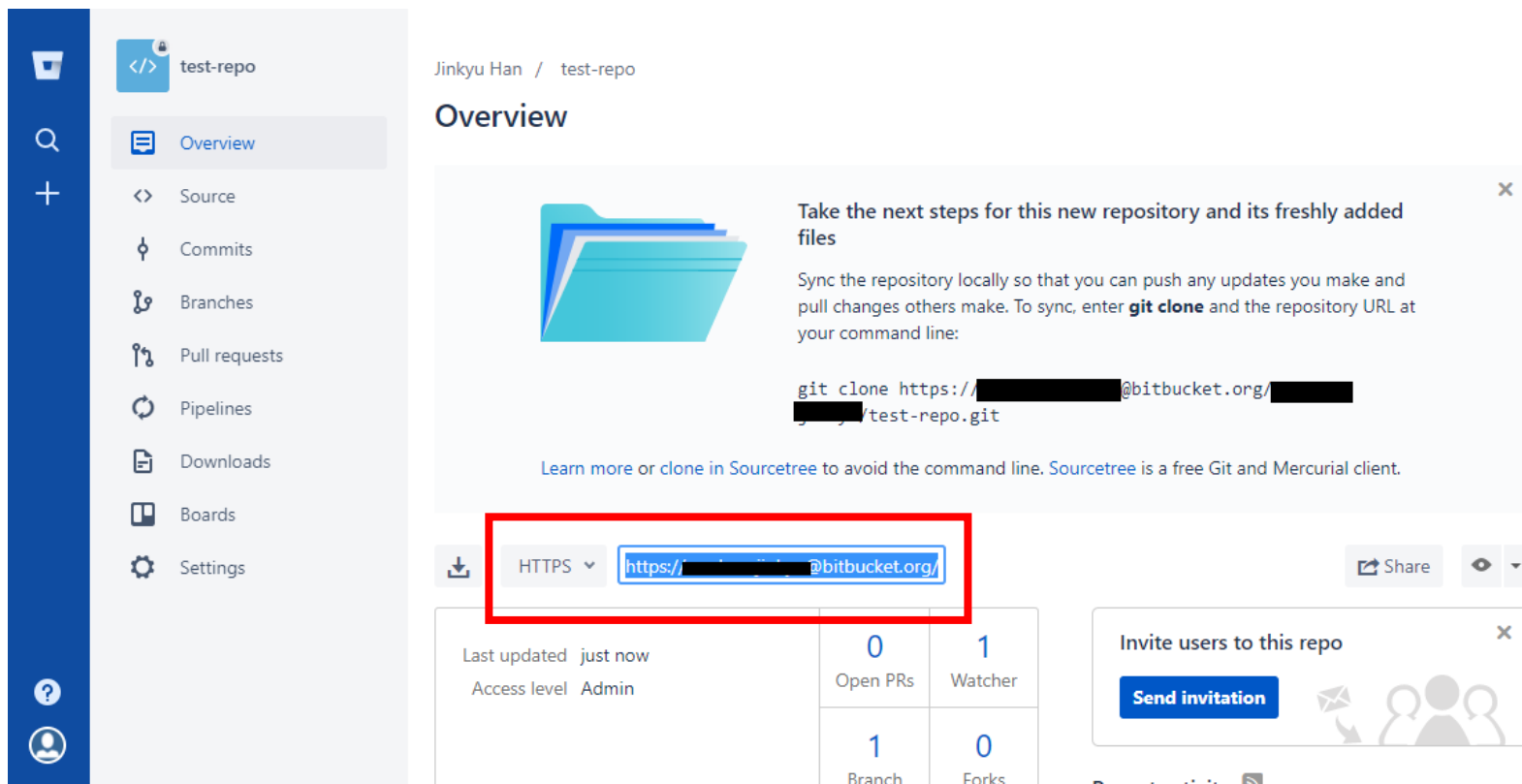
SourceTree 설치



SourceTree를 사용할 수 있게 되었다

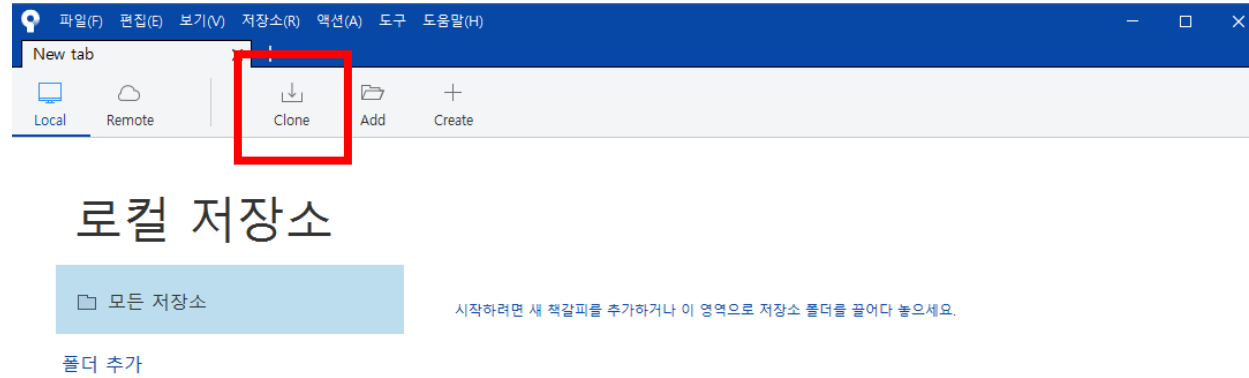
git clone

git clone



Bitbucket 홈페이지에서 다음 부분 주소를 복사
** HTTPS 부분을 SSH로 바꿔서 해도 좋다. 다만 SSH키를 갖고 있을 것.

git clone



메인 화면에서 상단의 [Clone] 메뉴를 클릭

git clone

Clone

Cloning is even easier if you set up a [remote account](#)

Remote URL:

Local Path:

test-repo

Local Folder:

고급 옵션

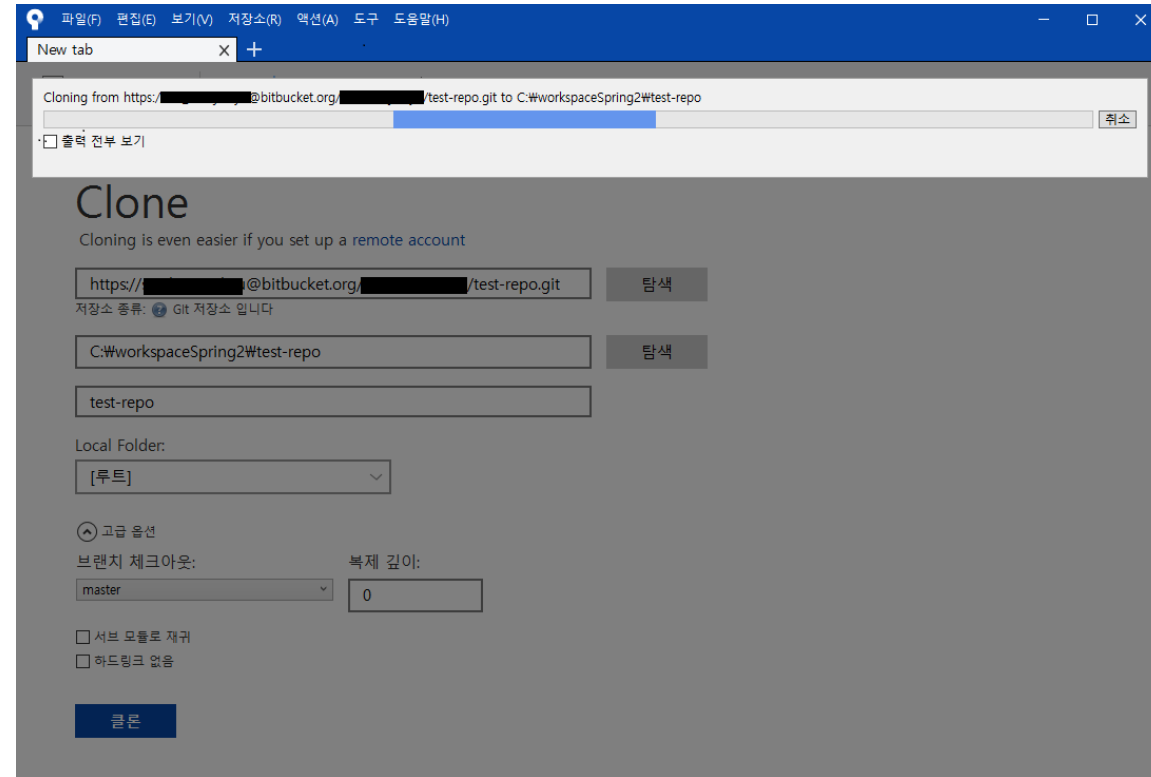
브랜치 체크아웃: 복제 깊이:

☐ 서브 모듈로 재귀

☐ 하드링크 없음

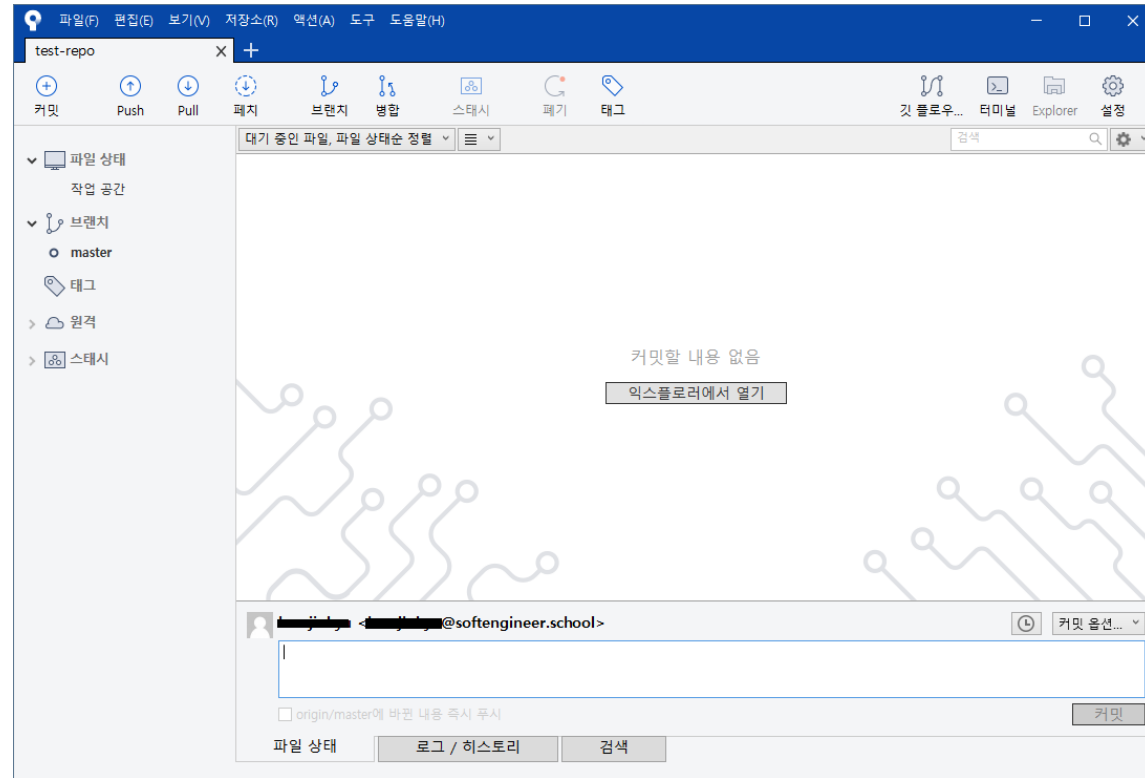
첫번째 사각형에 복사해온 주소를 복사 + 붙여넣기
두번째 사각형엔 **[project-workspace]/[프로젝트명]**

git clone



다음과 같이 복제가 일어난다

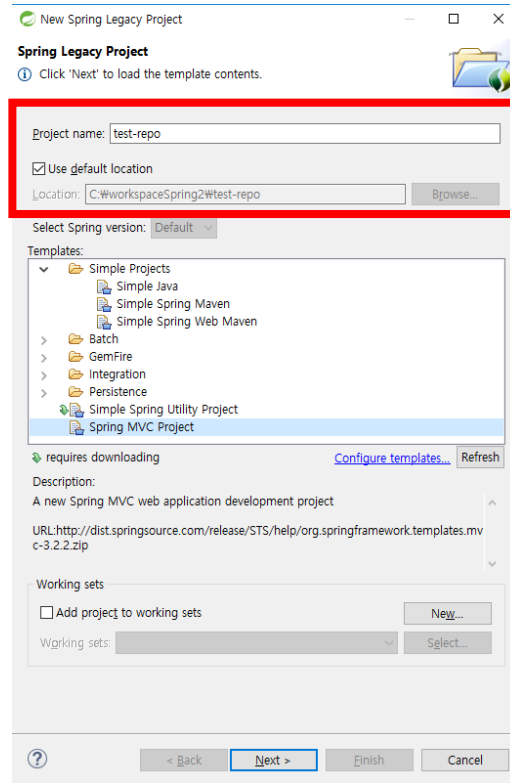
git clone



복제되고 나면 다음과 같이 화면이 나온다

프로젝트 생성

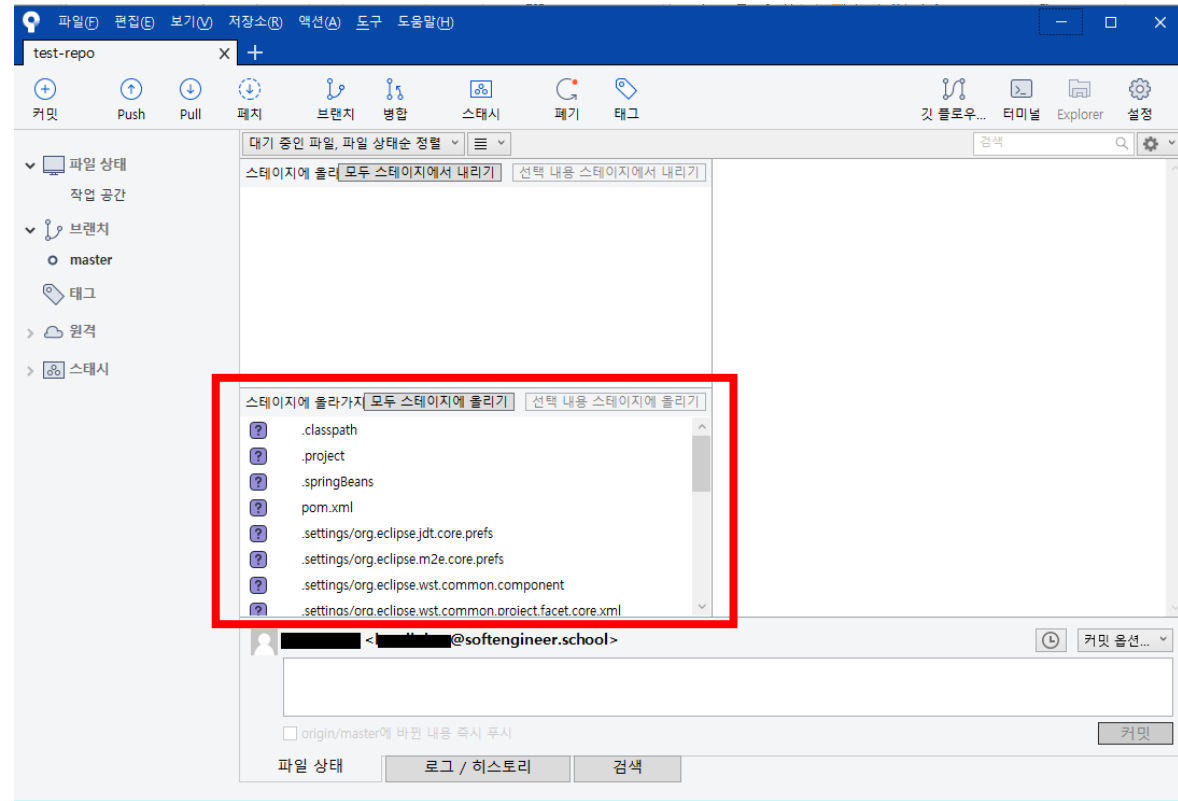
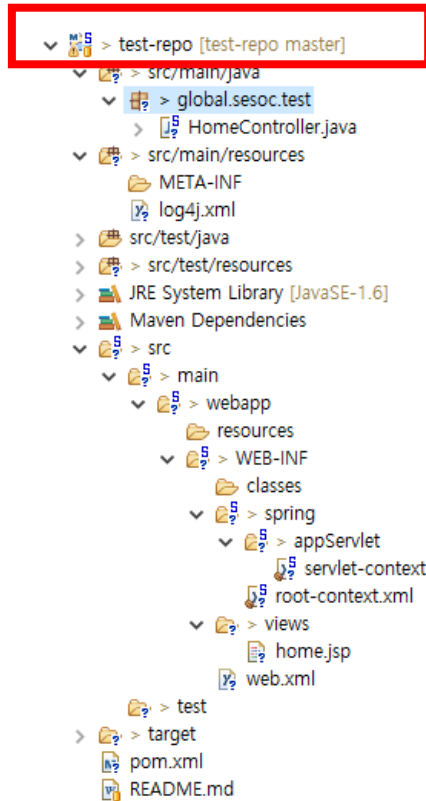
Spring 프로젝트 생성



** 사실 이 방법보다는 프로젝트를 먼저 생성 후에
[git init]을 사용하는 것이 좋다.
할 수 있다면 참고할 것을 권장한다.

Project name을 복제한 폴더명으로 하고
Location을 복제한 폴더로 설정

Spring 프로젝트 생성



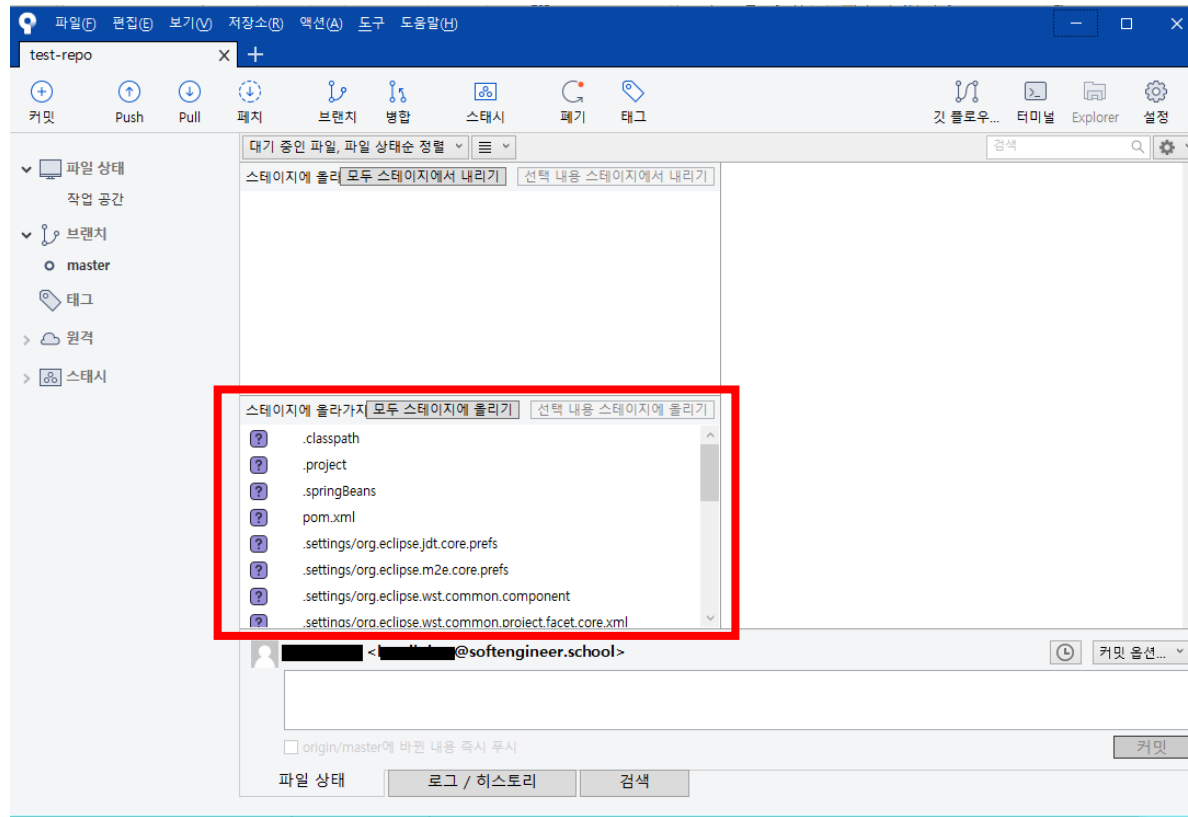
- 잘 생성되었다면 프로젝트 옆에 [xxx master]라는 문구가 표시
- SourceTree 내부에서도 빨간 박스처럼 생성된 파일들이 표시

git commit

커밋(commit)이란?

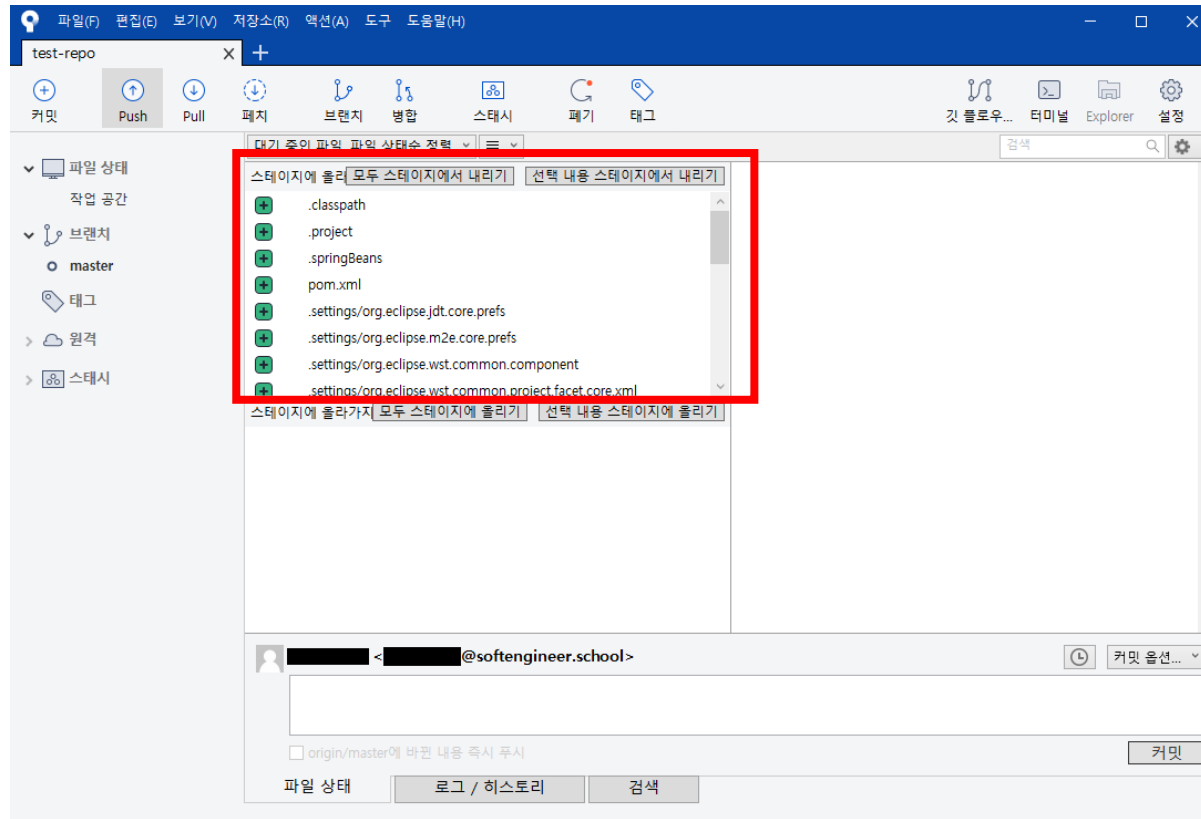
현재 자신의 프로젝트 변경 상태를
local 저장소에 반영하는 것

commit할 파일 선택



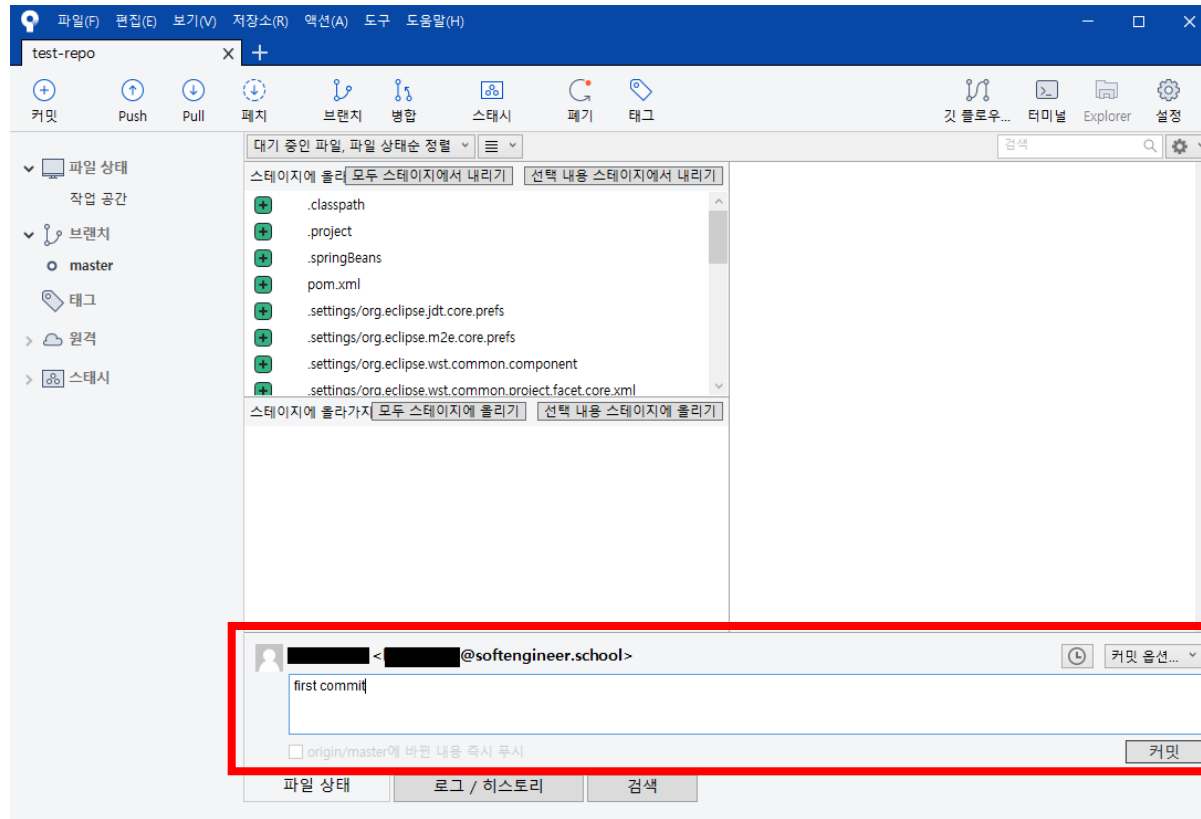
[모두 스테이지에 올리기] 버튼을 클릭한다

git commit



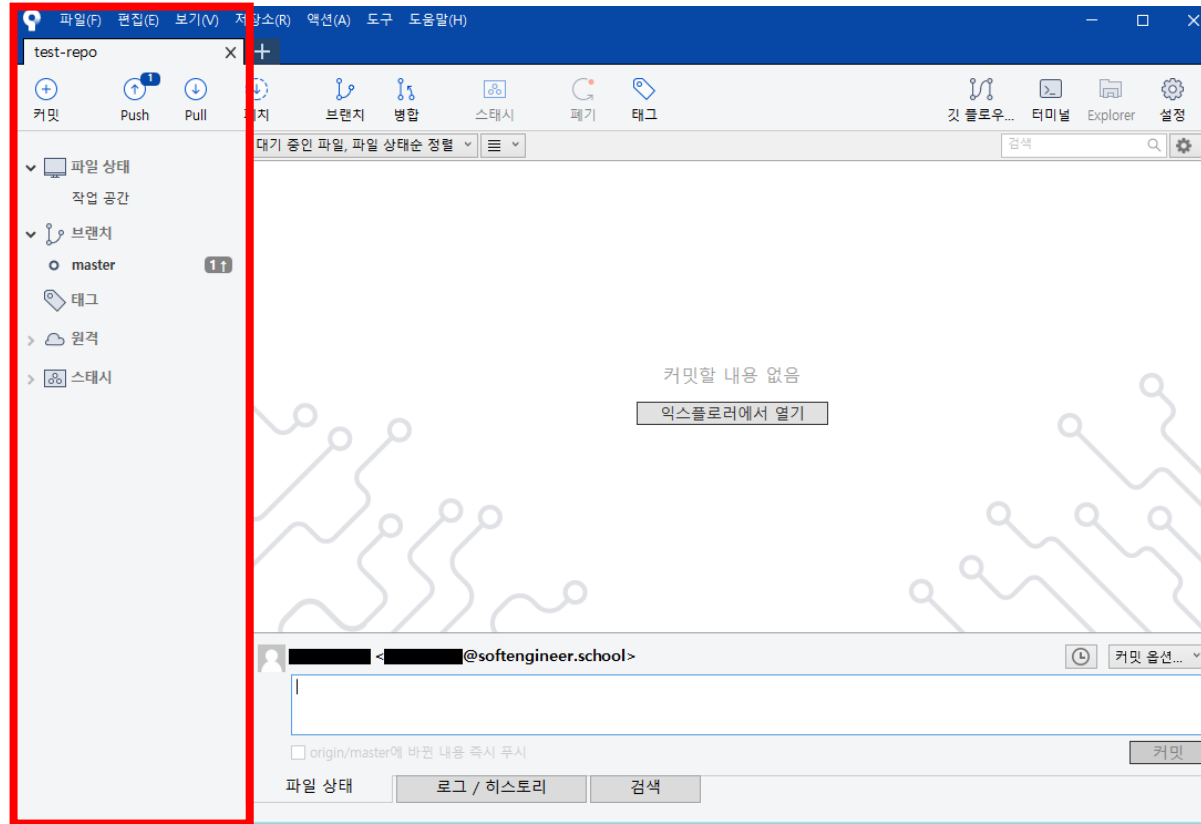
클릭하면 모든 파일이 상단의 창으로 옮겨가게 된다

git commit



화면과 같이 first commit이라 작성하고 [커밋] 버튼 클릭

git commit



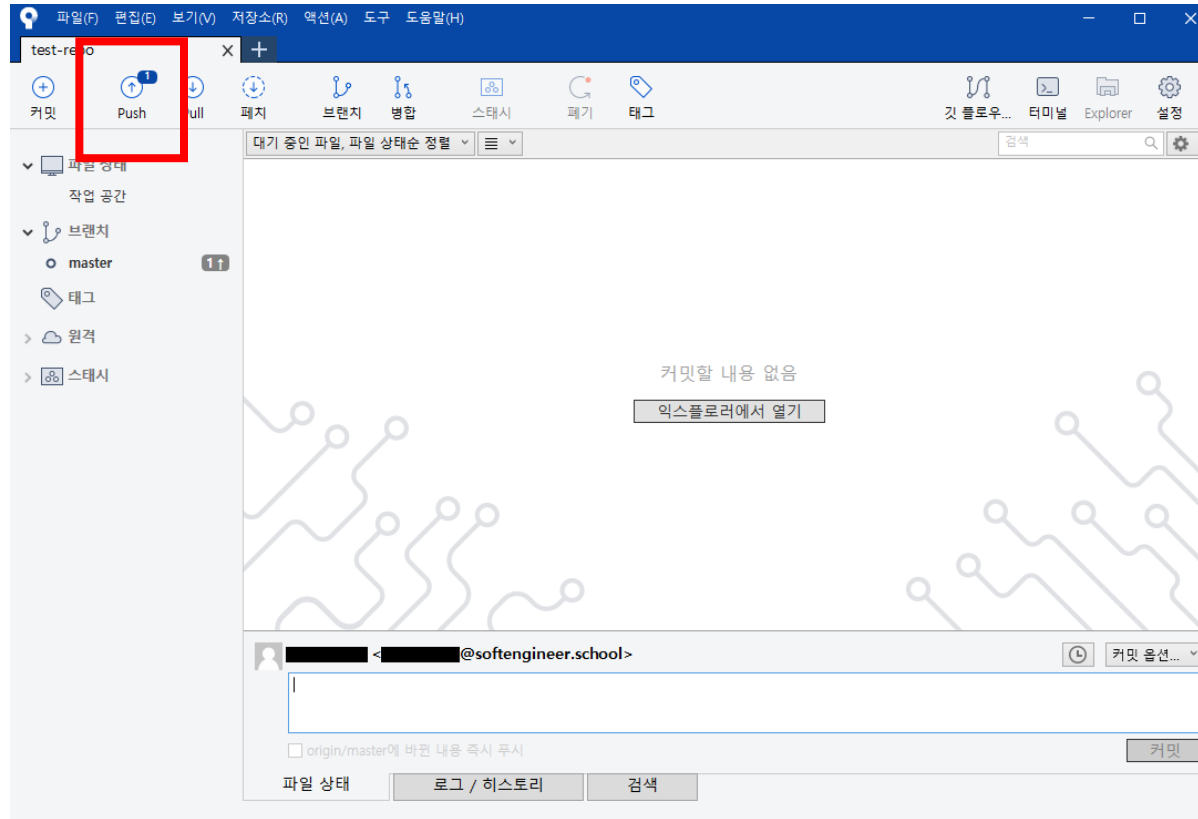
화면과 같이 왼쪽에 master에 1이란 표시와 push에 1이 표시되면 commit이 완료된 것이다

git push

푸쉬(push)란?

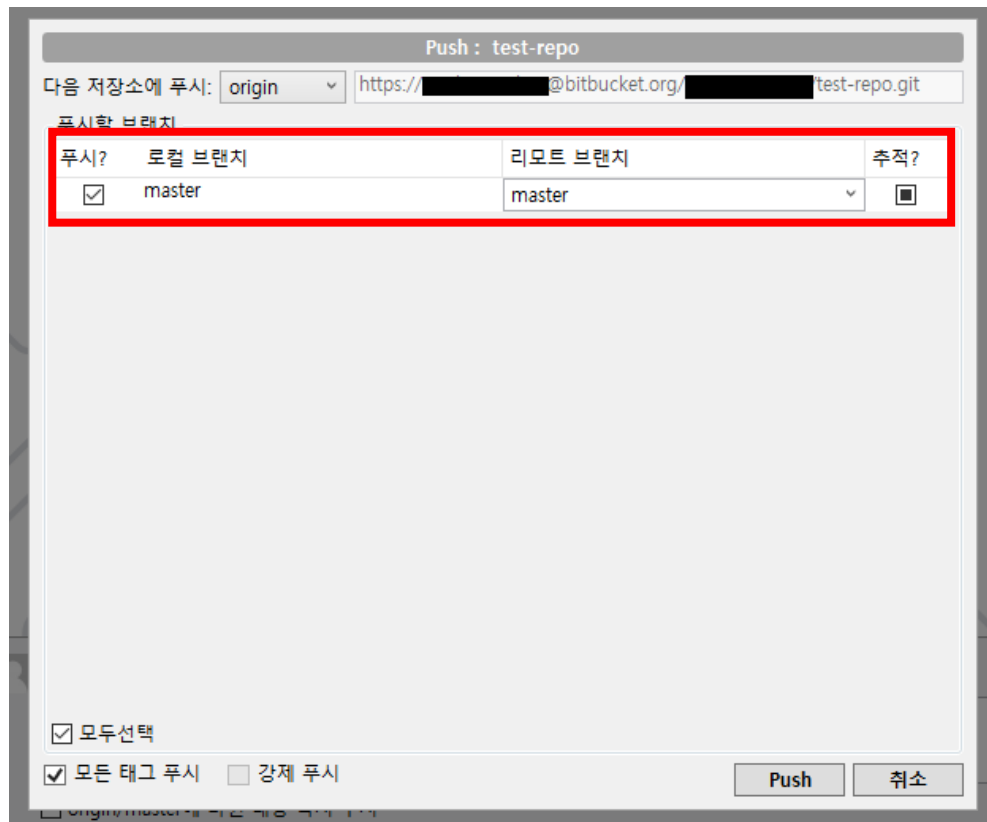
local 저장소에 commit하였던 내용을
remote 저장소에 반영하는 것

git push



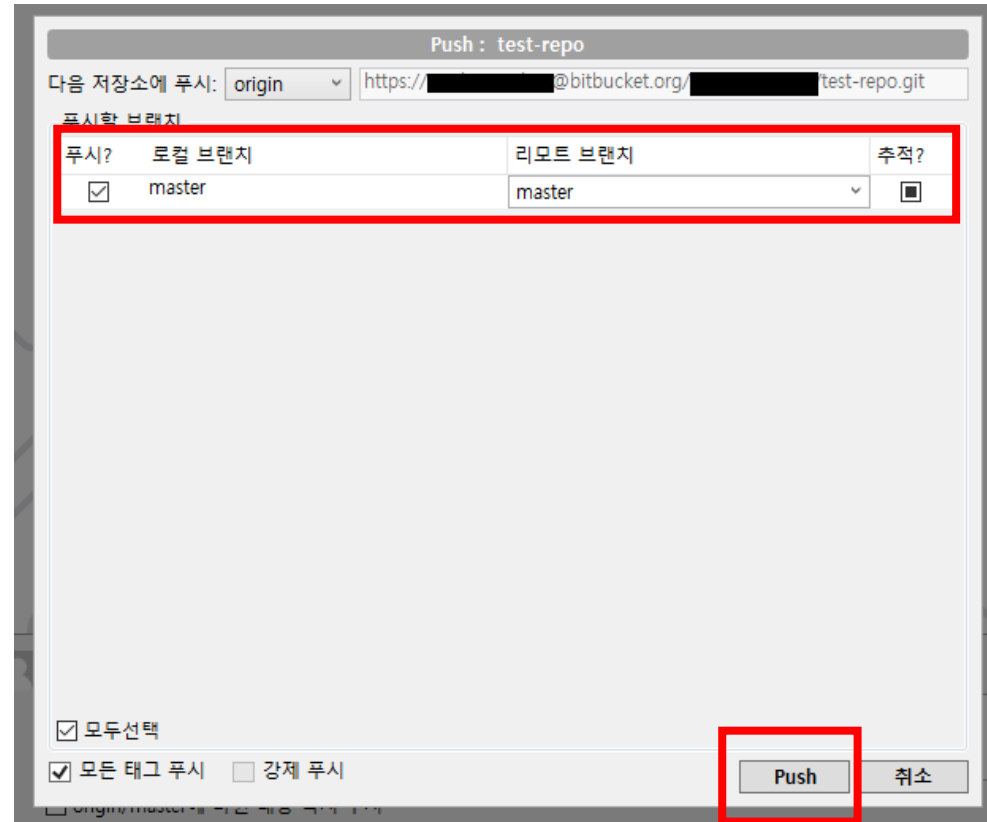
commit을 했다면 commit된 개수가 [Push] 버튼에 표시된다.
화면 상단의 [Push] 버튼을 클릭하여 진행한다.

git push



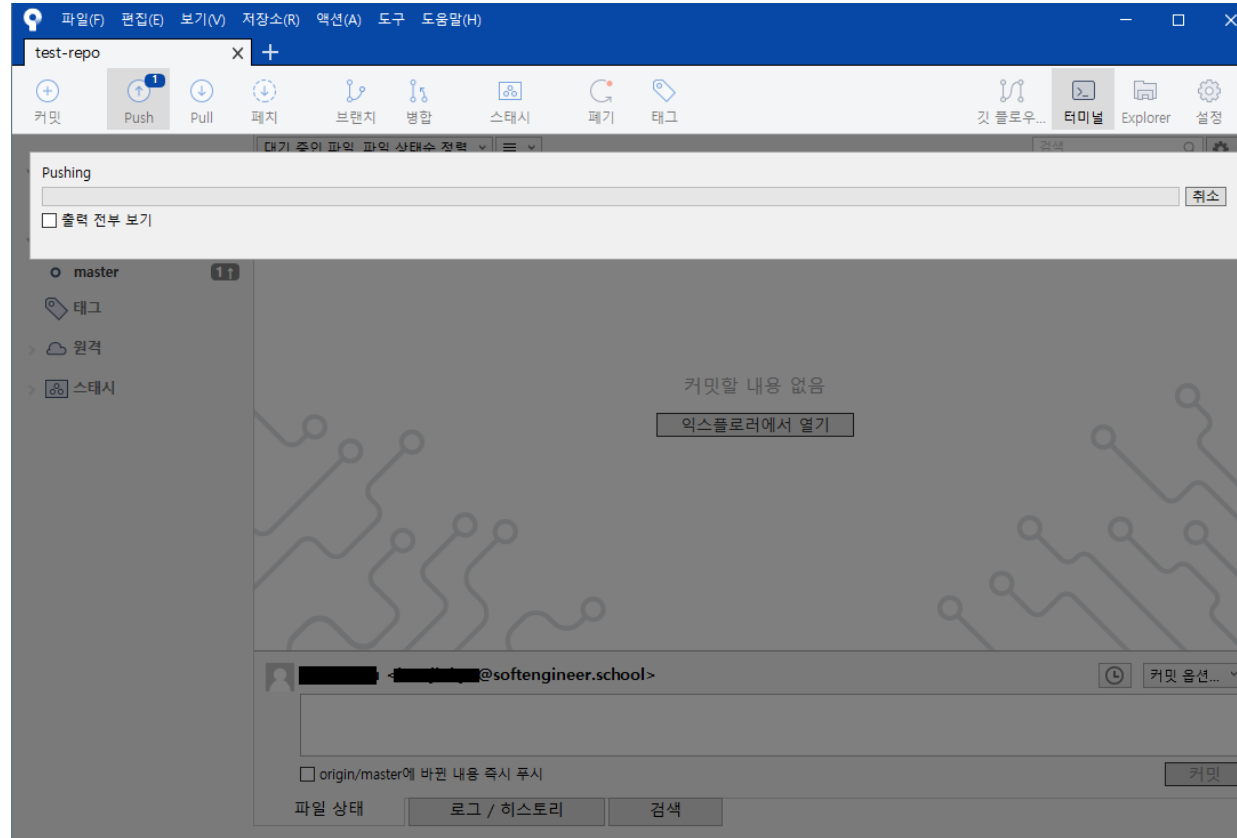
클릭하면 다음과 같이 push할 브랜치를 선택할 수 있다
[로컬 브랜치]와 [리모트 브랜치]를 잘 확인하도록 한다

git push



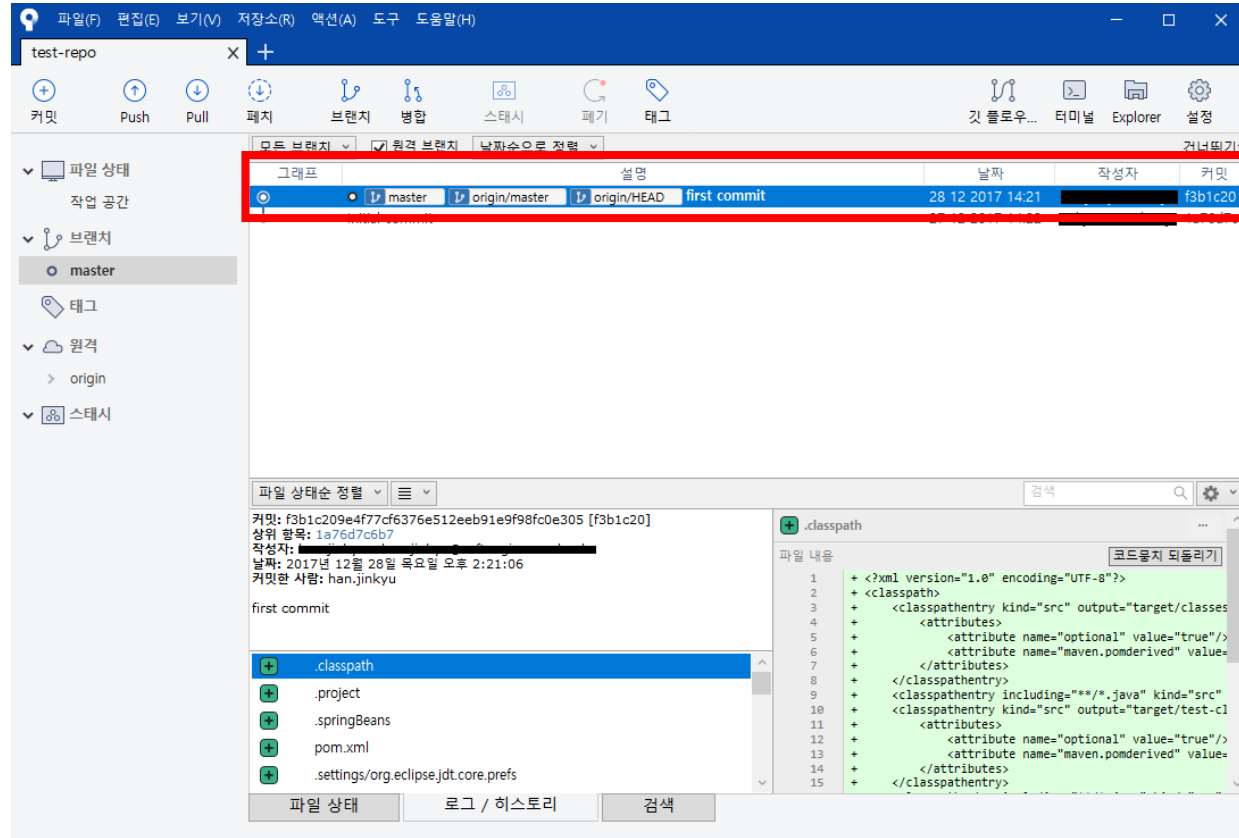
[푸시?] 부분이 체크되었는지 확인하고 [Push] 버튼을 클릭한다
브랜치를 처음 Push 한다면 [푸시?]를 체크해야 한다

git push



누르면 다음과 같이 진행사항이 나온다
[출력 전부 보기]를 체크하면 상세 정보가 나오며 에러도 확인 가능하다

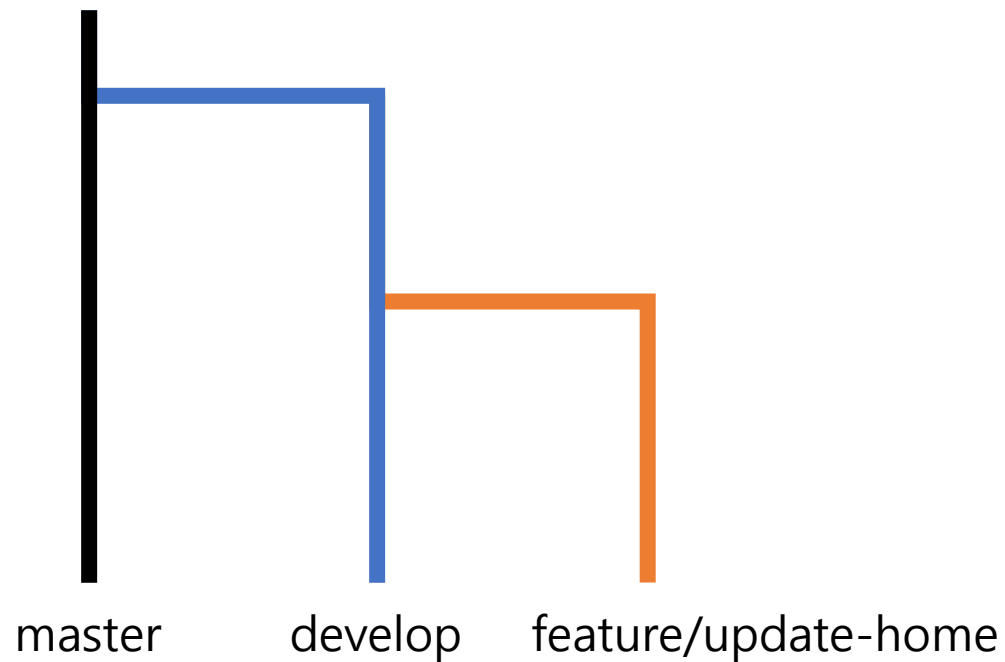
git push



다음과 같이 [master]와 [origin/master]가 확인되면
push과 완료된 것이다

git checkout

브랜치(branch)란?

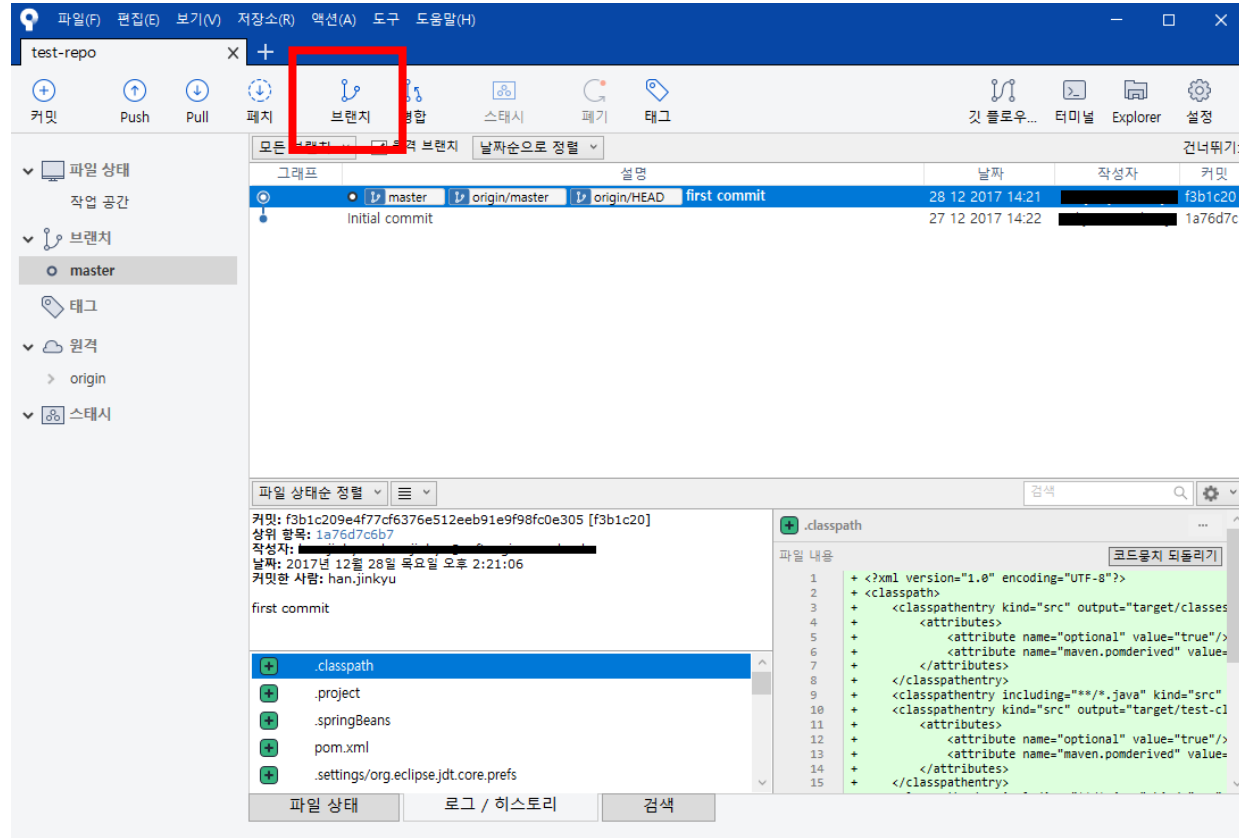


코드를 통째로 복사하여 원래 코드와는 상관없이
독립적으로 개발을 진행 가능하게 해주는 것

기본적인 branch의 종류

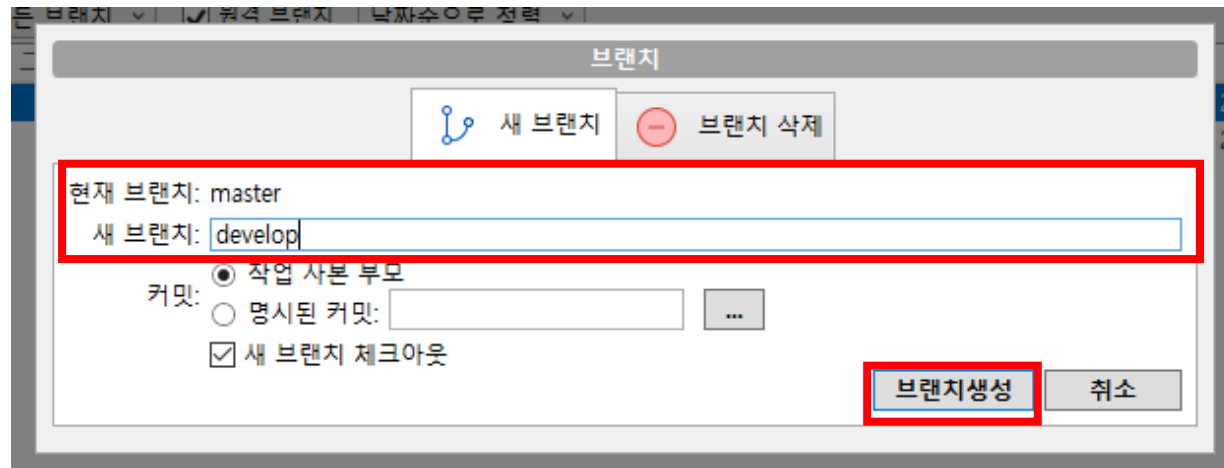
- **master** : 제일 처음에 생성되는 브랜치. 유저에게 배포될 **완성버전만** 올라온다.
- **develop**: 개발할 때 사용되는 브랜치로, **완전하게 개발된 버전만** master로 합친다. 'master'로부터 분기된다.
- **feature**: **새로운 기능 개발 및 버그 수정**이 필요할 때마다 사용하는 브랜치.
- **release**: 버그를 수정하거나 새로운 기능을 포함한 상태로 **모든 기능이 정상적으로 동작하는지** 확인
- **hotfix**: 배포한 버전에 **긴급하게 수정**을 해야 할 필요가 있을 경우, 'master' 브랜치에서 분기하는 브랜치.

git checkout



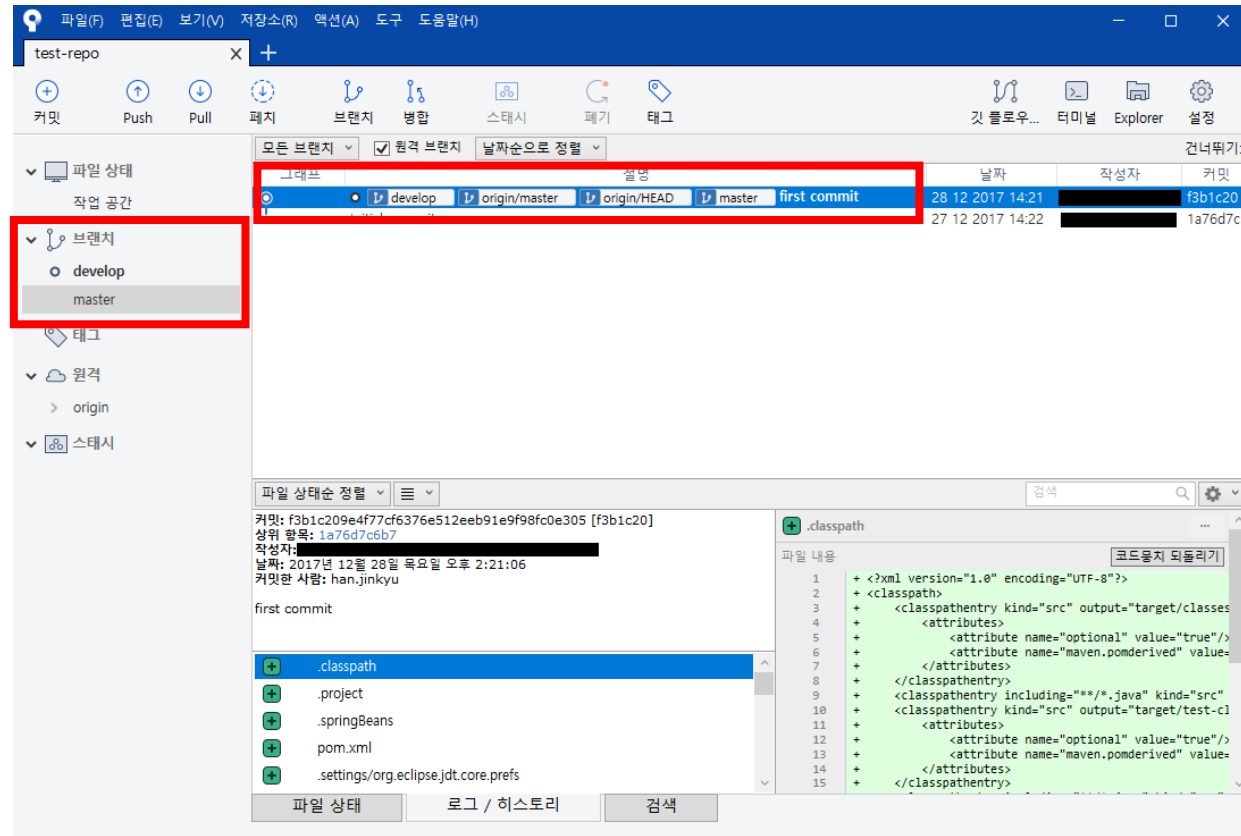
화면 상단의 [브랜치]를 클릭한다

git checkout



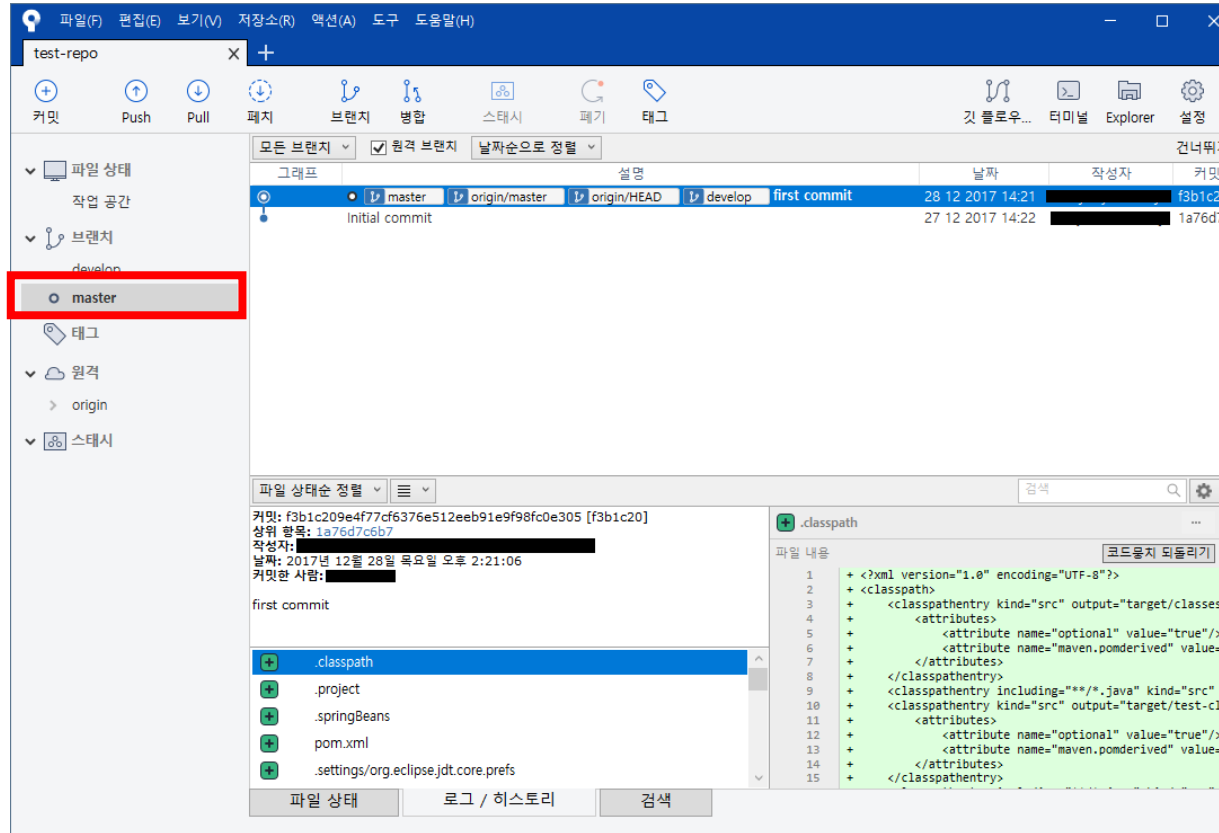
[현재 브랜치]로부터 [새 브랜치]를 만든다
이번엔 [새 브랜치]에 'develop'이라 쓰고 [브랜치생성] 버튼을 클릭한다

git checkout



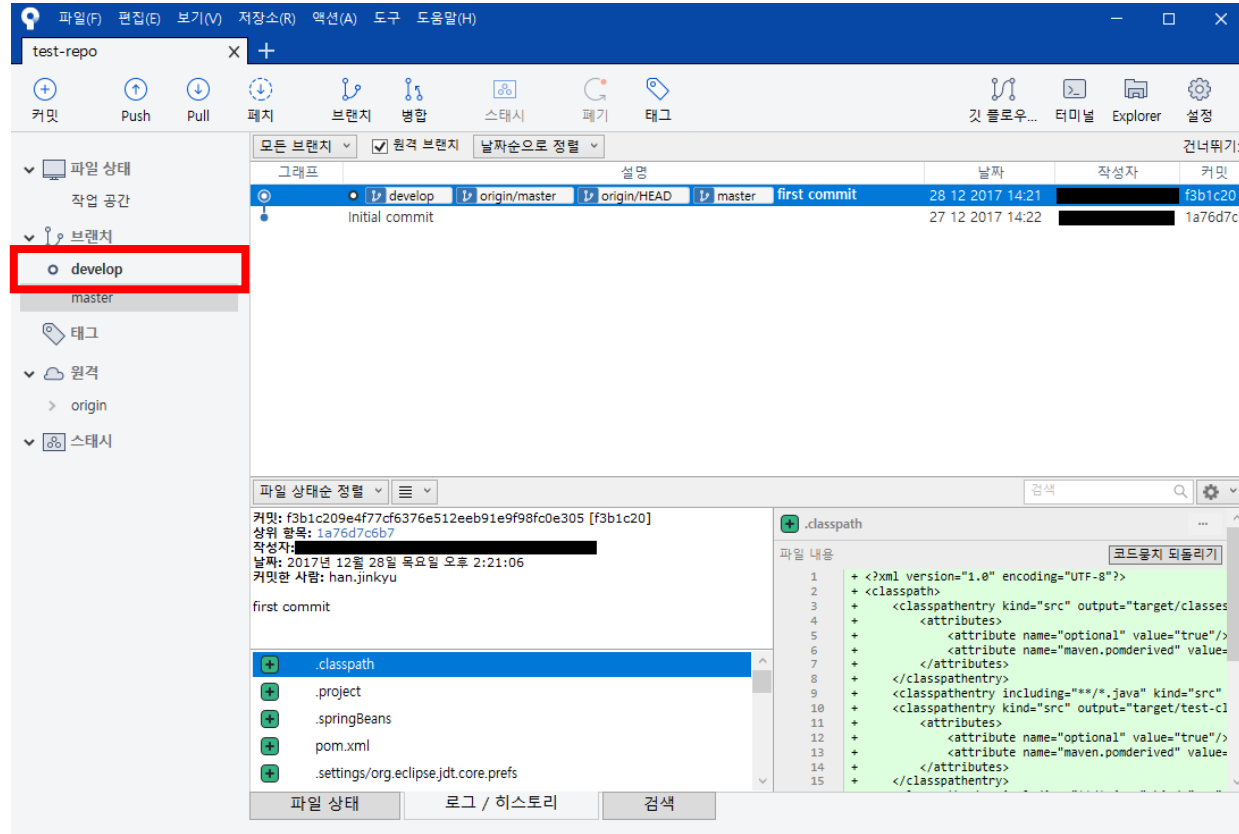
화면과 같이 왼쪽의 메뉴 중 [브랜치]라는 곳에
'develop'이라는 브랜치가 생성된다

git checkout



'master'로 다시 checkout 하려면
'master' 브랜치를 더블 클릭하면 된다

git checkout



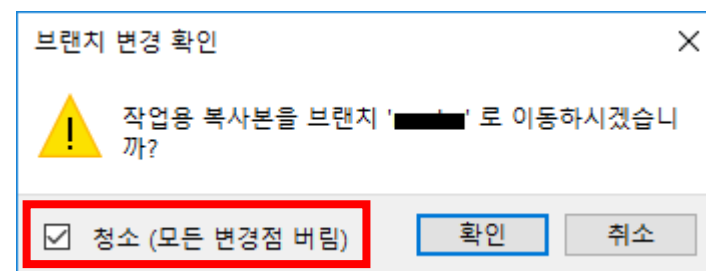
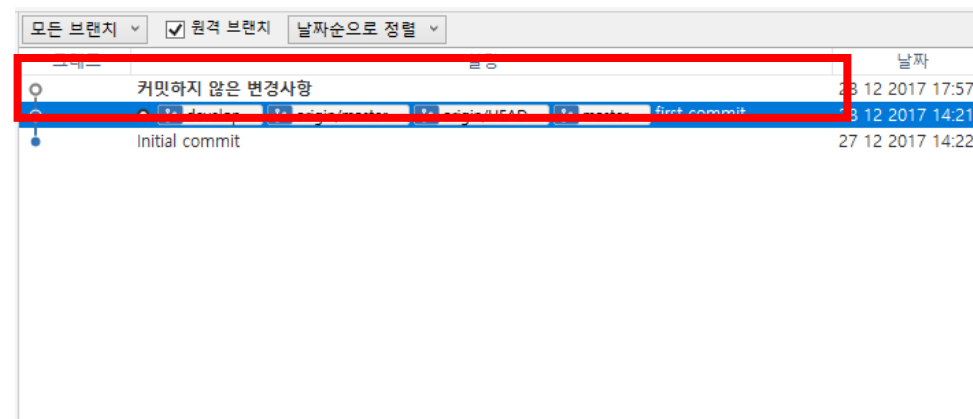
'develop'에서 개발이 진행될 것이므로 다시 돌아간다
그리고 'develop'을 remote에 Push하여 둔다

git checkout

만약 왼쪽 그림과 같이 '**커밋하지 않은 변경 사항**'이라 적혀 있다면 체크아웃이 원활하지 않을 것이다.

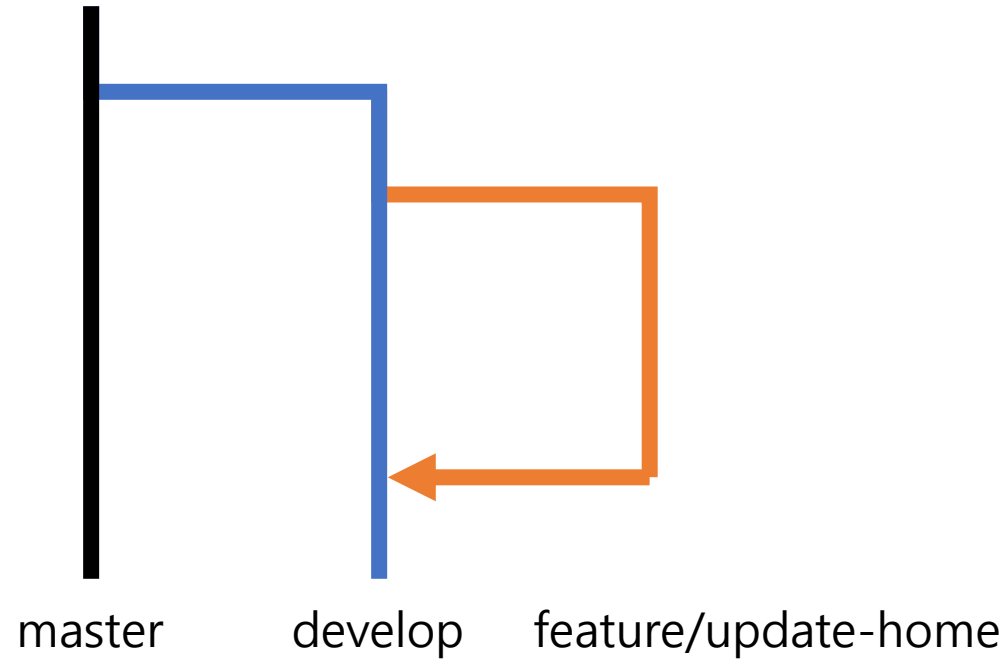
이유는 local 상에 파일 변경이 생겨 있기 때문이다. 이를 해소하려면 3가지 방법이 있다.

1. 변경사항을 **commit**하고 체크아웃 한다.
2. 변경사항을 **모두 버리고** 체크아웃 한다.
3. stash를 사용하여 현재 **변경사항을 잠시 저장**하고 체크아웃 한다.



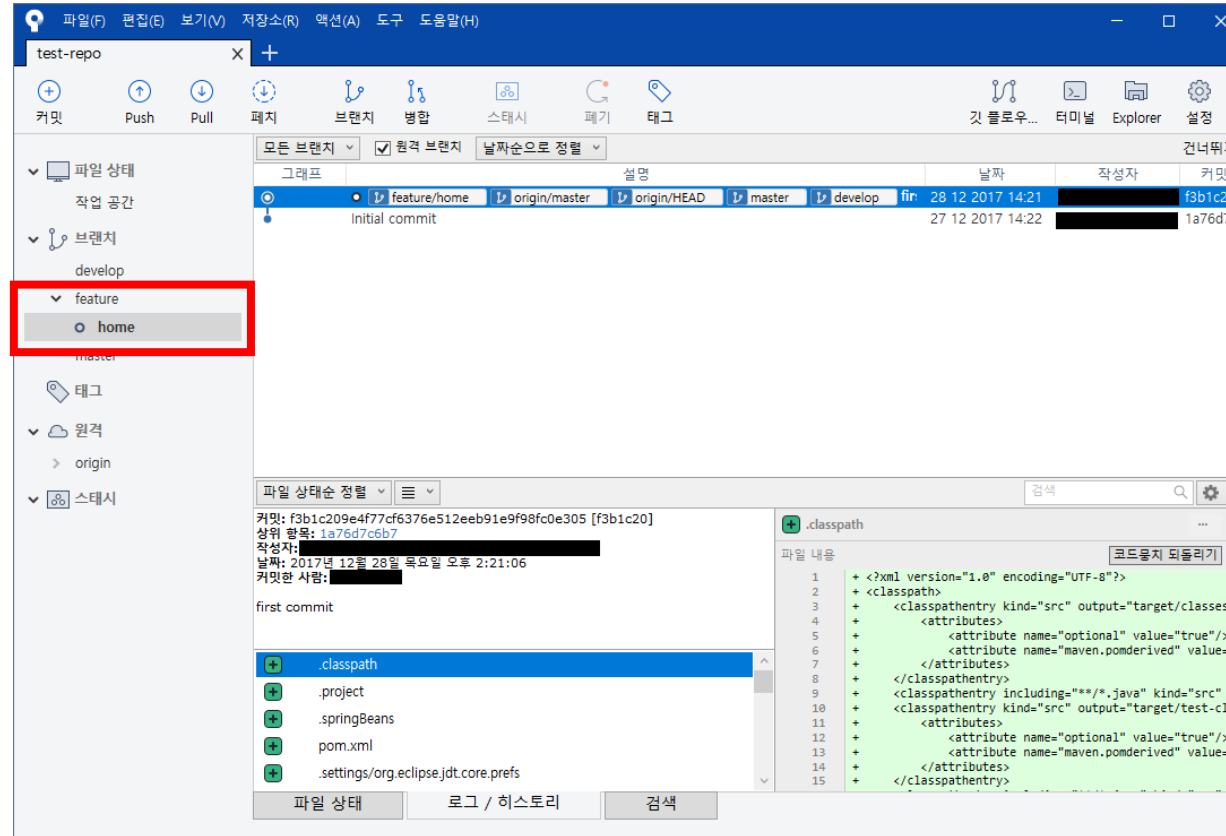
git merge

병합(merge)이란?



하나의 브랜치를 다른 하나의 브랜치로 합치는 것을 뜻한다

git merge



화면 상단의 [브랜치]를 클릭하여
새로운 브랜치 [feature/update-home]을 만든다

git merge

```
1 package global.sesoc.test;
2
3 import java.util.Locale;
4
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestMethod;
11
12 /**
13  * Handles requests for the application home page.
14  */
15 @Controller
16 public class HomeController {
17
18     private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
19
20     /**
21      * Simply selects the home view to render by returning its name.
22      */
23     @RequestMapping(value = "/", method = RequestMethod.GET)
24     public String home(Locale locale, Model model) {
25         logger.info("Welcome home! The client locale is {}.", locale);
26         return "home";
27     }
28 }
29
30
```

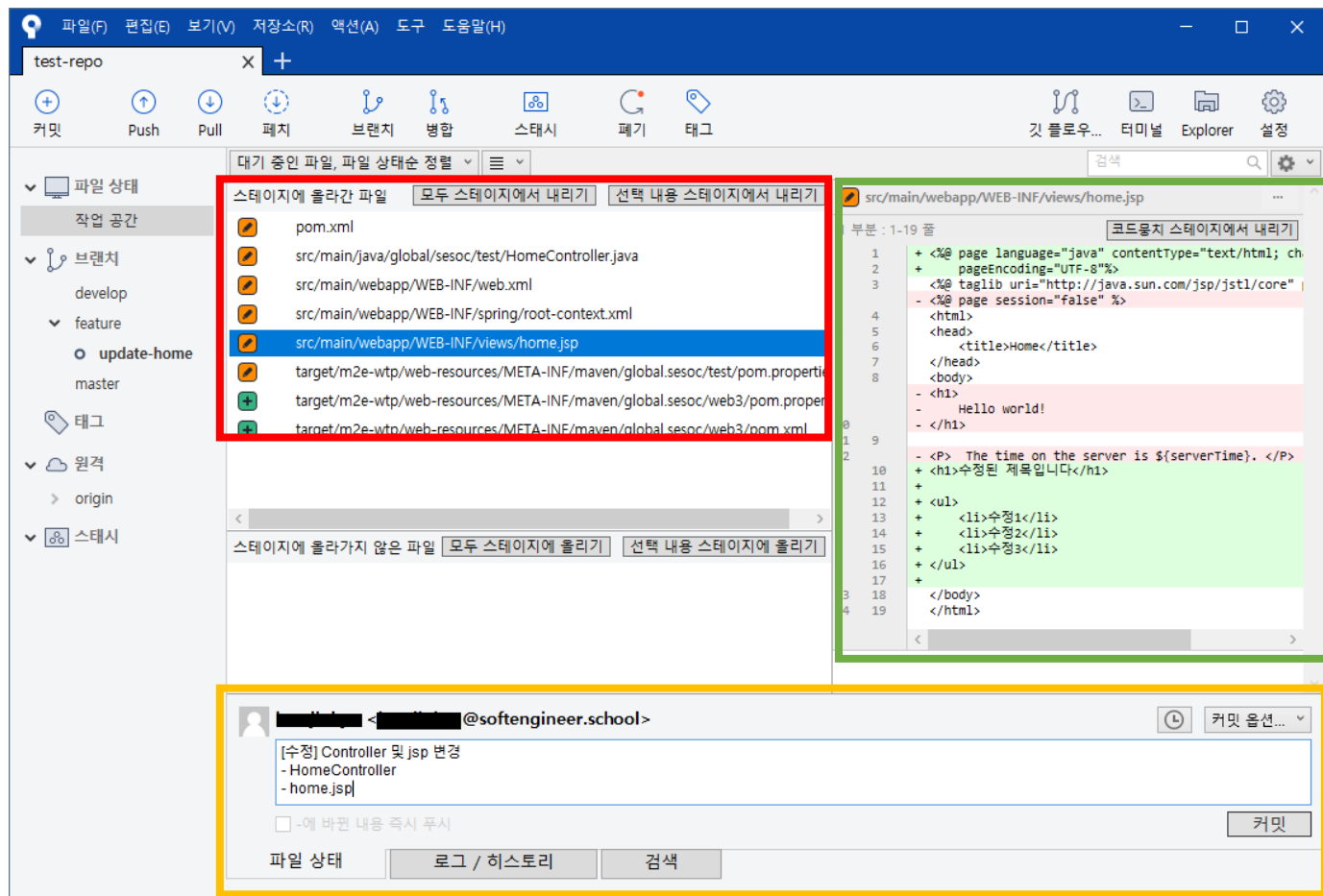
HomeController.java

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
4 <html>
5 <head>
6     <title>Home</title>
7 </head>
8 <body>
9
10 <h1>수정된 제목입니다</h1>
11
12 <ul>
13     <li>수정1</li>
14     <li>수정2</li>
15     <li>수정3</li>
16 </ul>
17
18 </body>
19 </html>
20
```

home.jsp

그리고 간단하게 코드를 수정한다

git merge

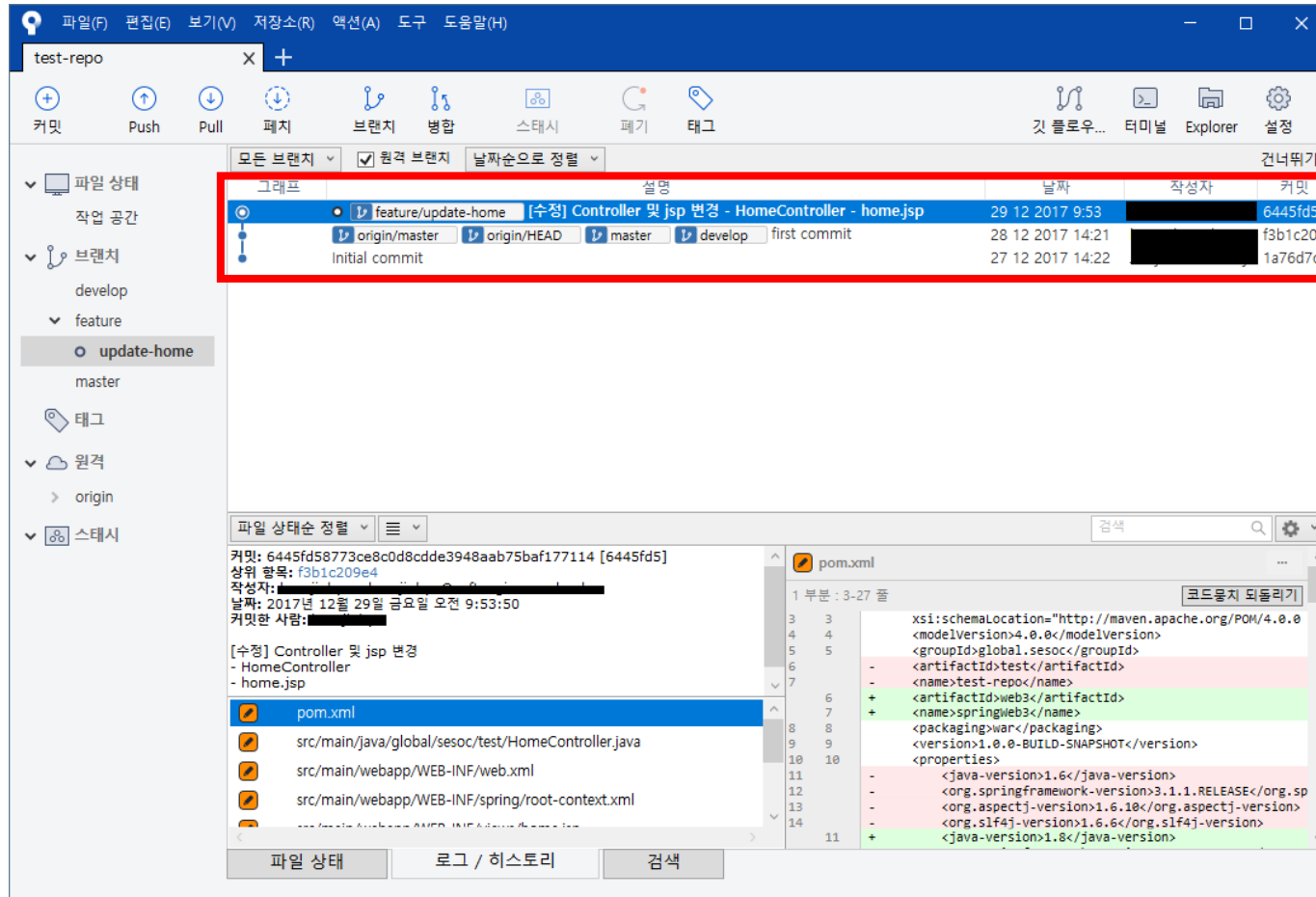


[스테이지에 올라가지 않은 파일]에 변경된 파일들이 확인된다. [모두 스테이지에 올리기]를 클릭하여 전부 커밋하도록 한다. 그러면 빨강 영역처럼 바뀐다.

파일 하나 하나를 클릭해보면 초록 영역에서 어떤 코드가 수정되었는지 확인 가능하다.

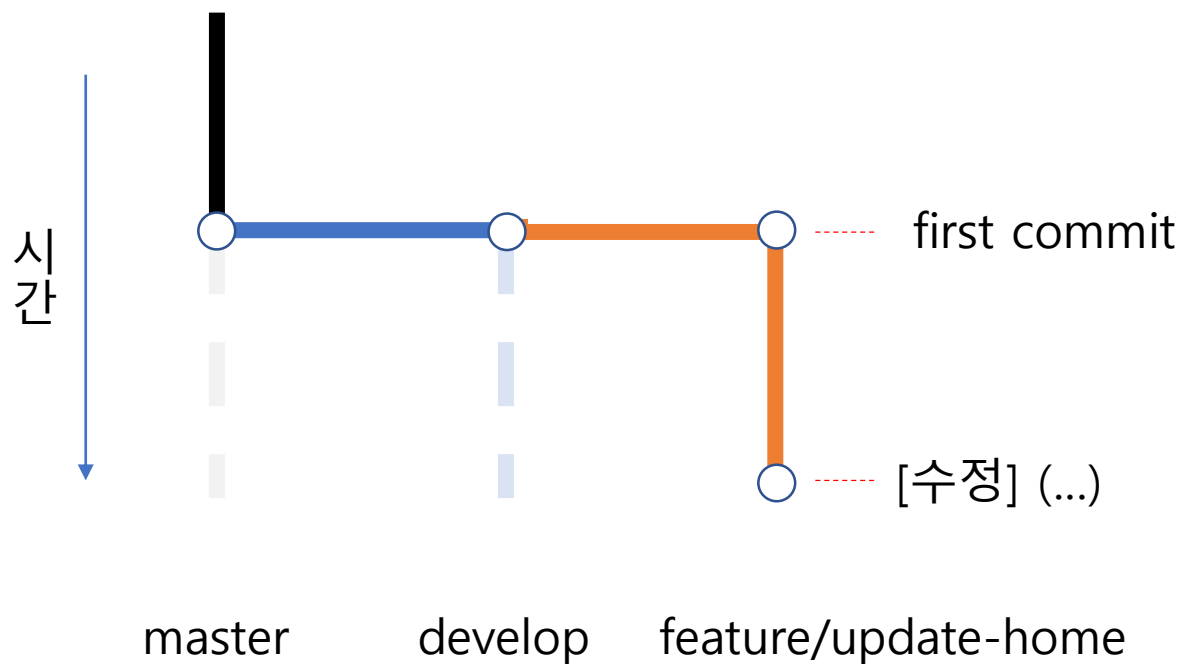
노랑 영역에 commit할 내용에 대한 메시지를 화면과 같이 쓰고 commit을 한다.

git merge



commit이 완료되면 화면과 같이 'feature/update-home'만 수정사항을 적용한 것으로 표현될 것이다.

git merge

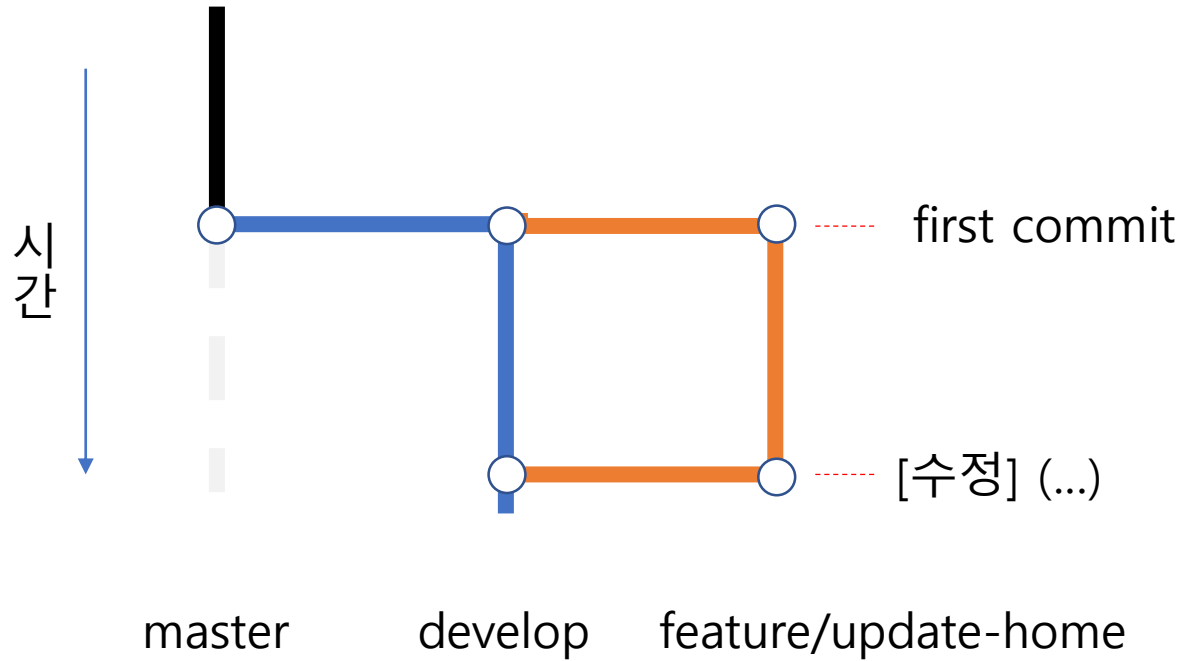


그림으로 표현하면 현재 이런 상태라고 볼 수 있다.

'develop'은 'master'로부터 분기되었고, 'feature/update-home'은 'develop'으로부터 분기되었으므로 [first commit]은 모두 같은 모습을 가질 것이다.

하지만 'feature/update-home'에서 변경사항을 commit하였으므로 현재 상황은 'feature' 브랜치가 한 발 더 나아간 모습이다.

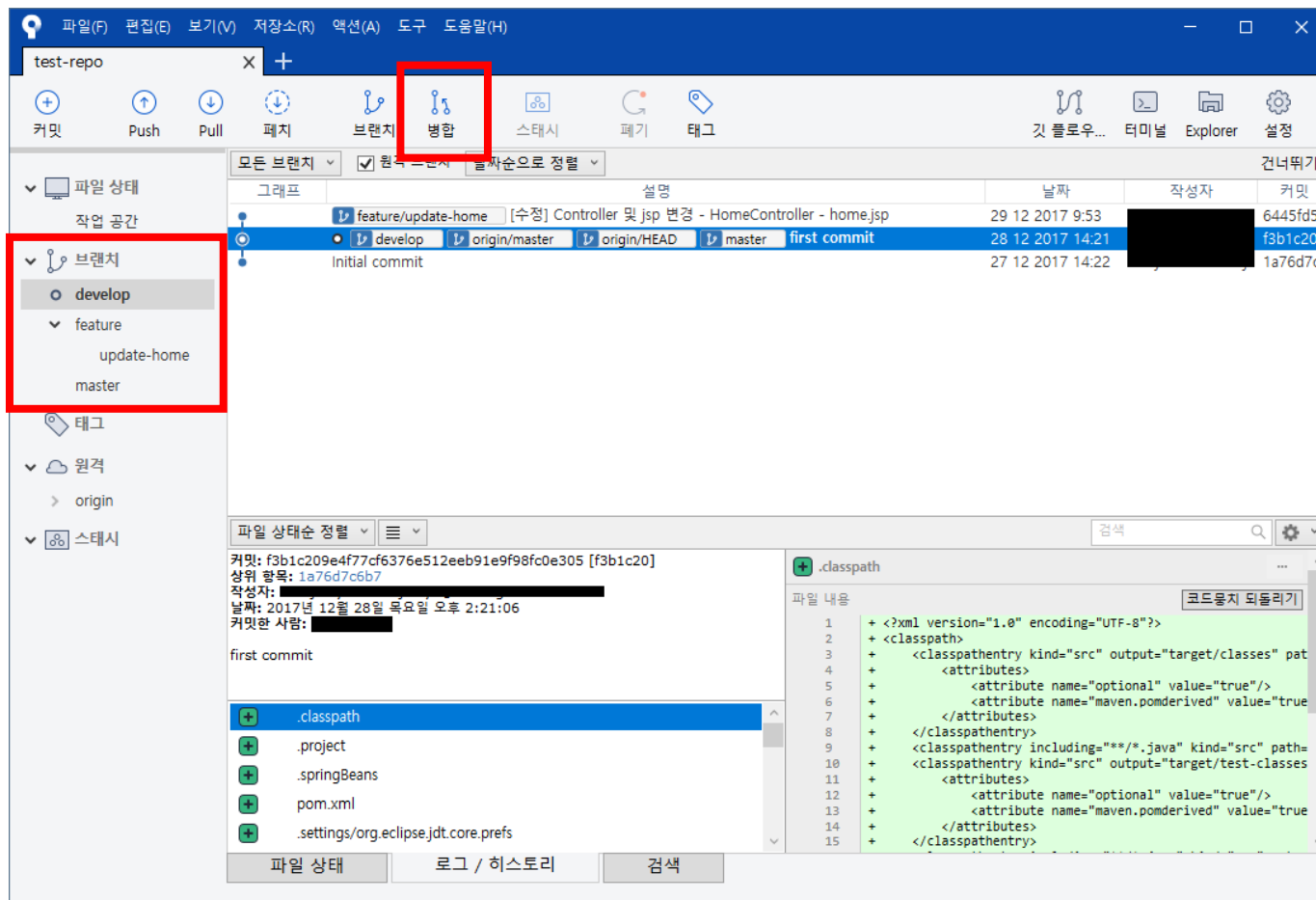
git merge



'feature/update-home'의 변경사항을
이제 'develop' 브랜치에 적용할 것이다.

이것을 브랜치를 병합(merge)한다고
표현한다.

git merge

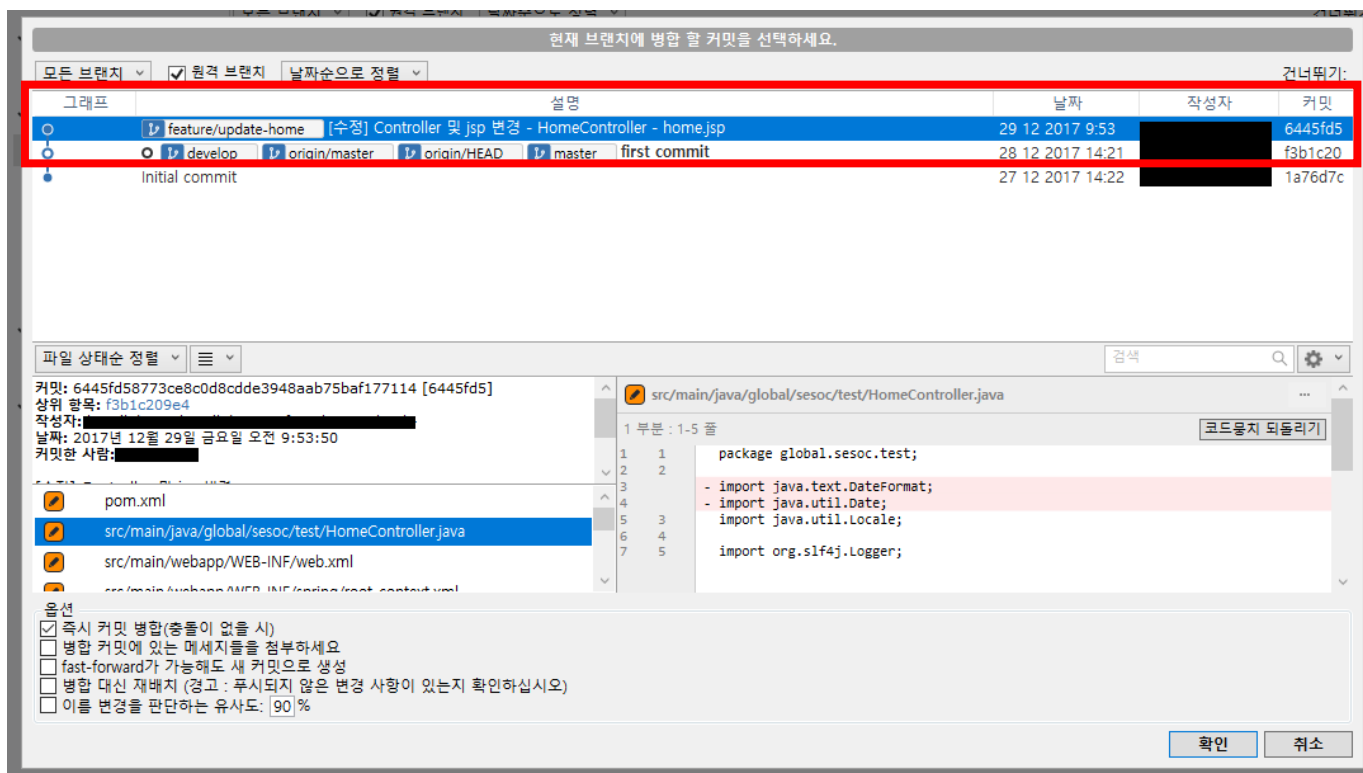


먼저 merge가 될 브랜치로 체크아웃한다.

현재는 'develop'에 'feature/update-home'을 merge할 것이므로 'develop'으로 체크아웃 했다.

그리고 상단 메뉴의 [병합]을 클릭한다.

git merge

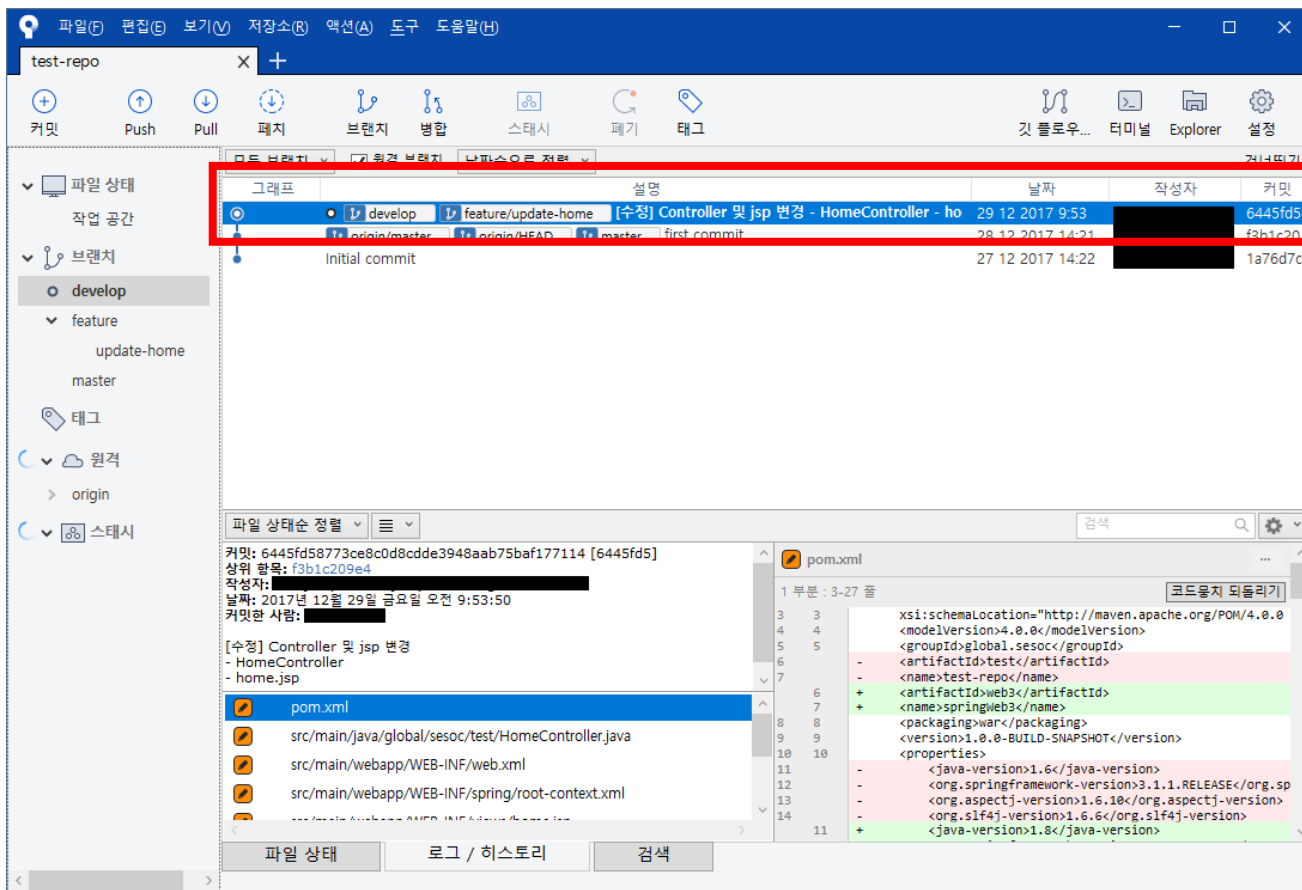


다음과 같은 창이 열리고 이 안에서 현재 브랜치로 merge할 commit 시점을 선택한다.

현재는 'feature/update-home'의 수정 사항을 'develop'에 반영할 것이므로, 'feature/update-home'에서 commit한 "[수정] Controller (...)"을 선택하고 확인을 누른다.

아래 옵션엔 [즉시 커밋 병합]이 있는데, 이는 아무런 문제가 없을 시엔 바로 merge와 commit을 진행하는 것을 의미한다.

git merge

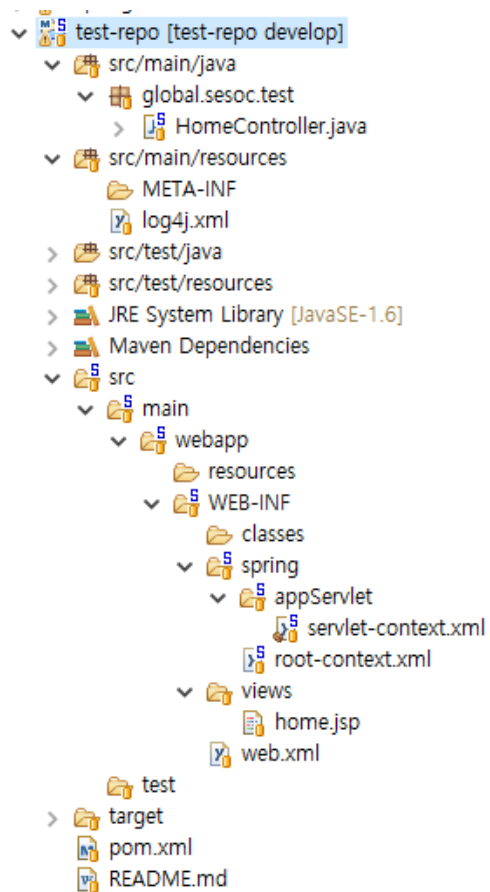


제대로 병합이 되었다면 'develop'과 'feature/update-home'의 위치가 같은 것을 확인할 수 있다.

이는 제대로 'develop'이 'feature/update-home'을 따라 가고 있는 것을 보여준다.

제대로 진행되었다면 이들 모두 push를 진행하여 **remote** 저장소에도 반영하도록 한다.

git merge

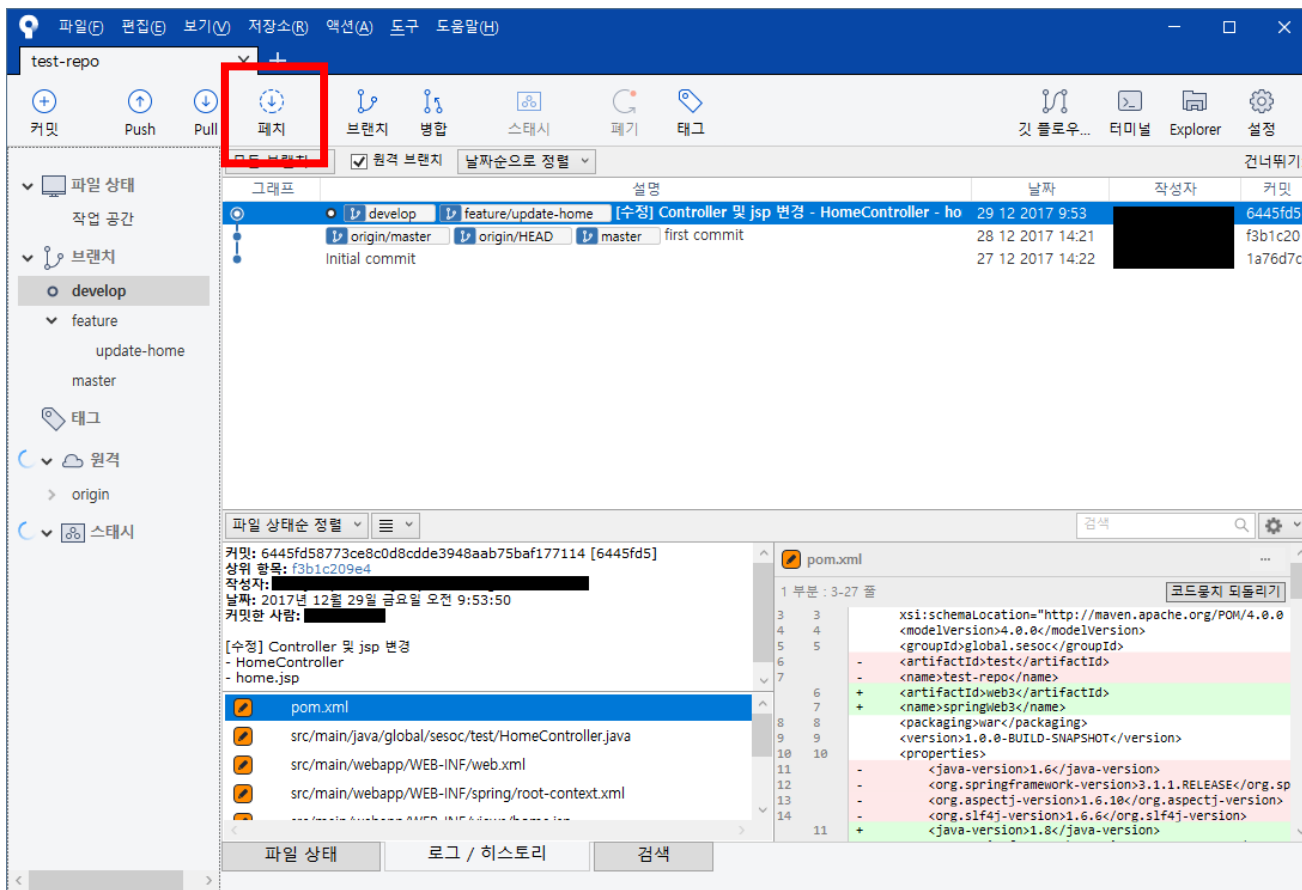


변경된 것은 STS나 Eclipse 상에서도 확인할 수 있으며, 코드들이 변경되었는지 확인해 보자.

만약 merge를 진행하고 나서 필요 없는 브랜치가 있다면 전부 삭제하도록 한다.

git fetch

git fetch

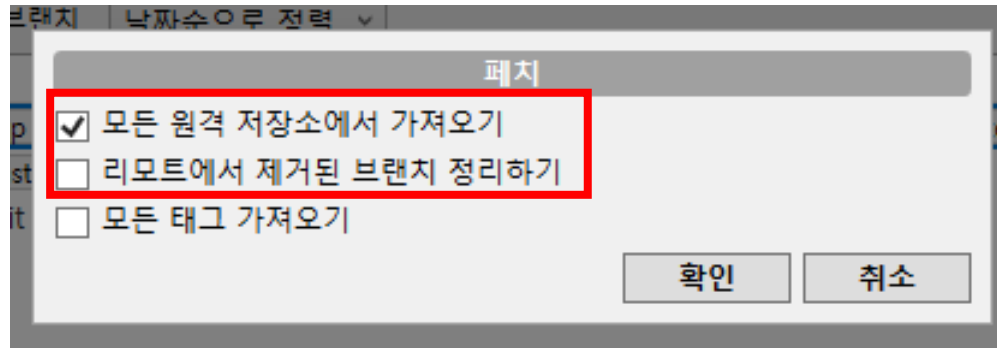


git repository를 다른 프로그래머와 공유하면 나 뿐만 아니라 다른 프로그래머 역시 commit하고 push 할 것이다. 그리고 브랜치를 나누고 병합하고 할 것이다.

하지만 이를 실시간으로 반영해주는 것은 아니기 때문에 remote 저장소로부터 현재 상황을 다운로드 받아야 한다. 이것을 fetch라고 한다.

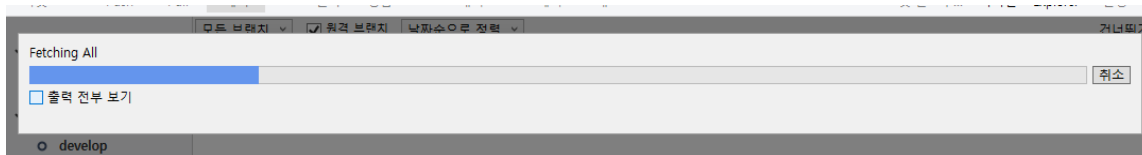
메뉴 상단의 [페치] 버튼을 클릭한다.

git fetch



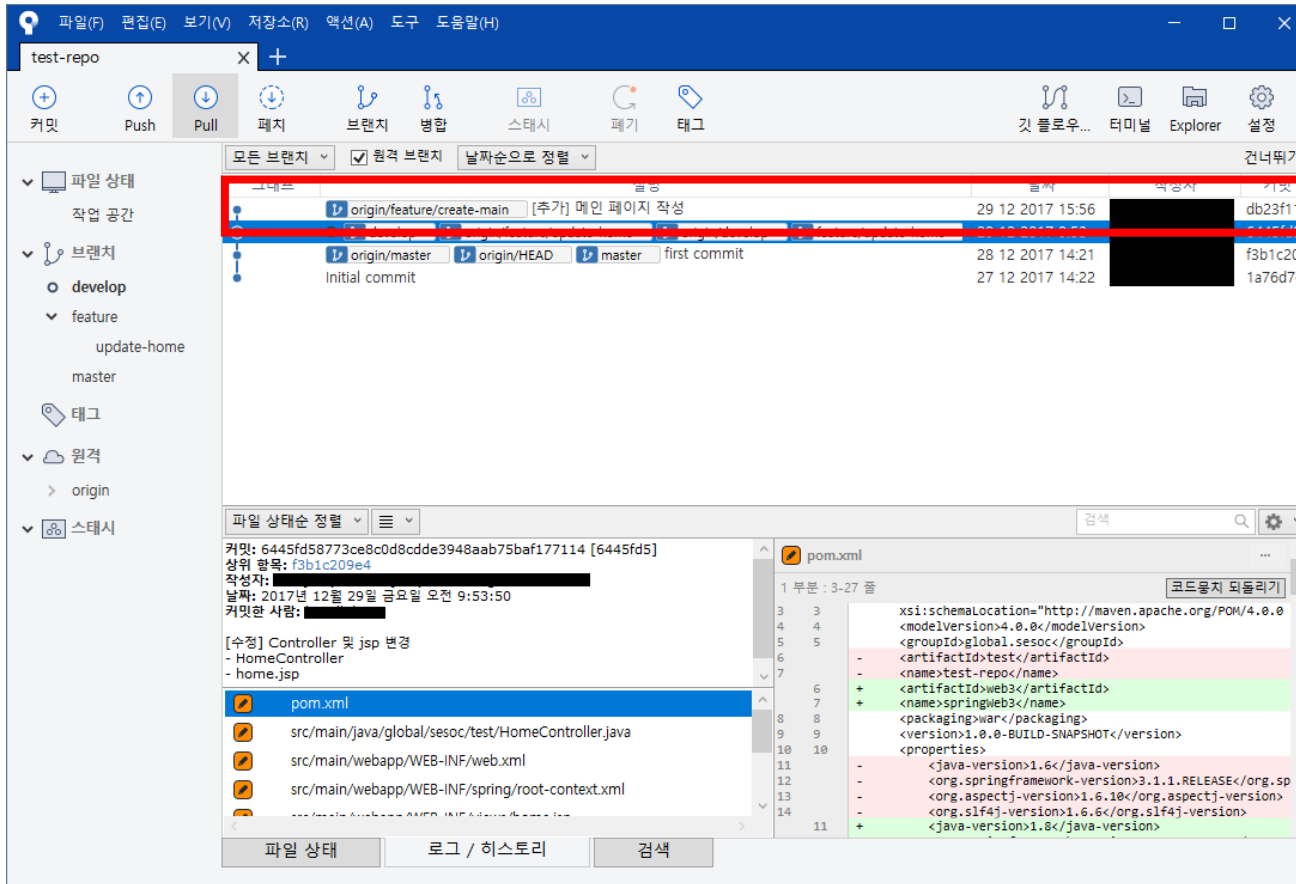
[페치] 버튼을 누르면 왼쪽 상단의 창이 뜨며 어떤 동작을 할 것인지 고를 수 있다.

[모든 원격 저장소에서 가져오기]는 꼭 선택하고 [리모트에서 제거된 브랜치 정리하기]는 필요하다면 선택하도록 한다.



[확인]을 누르면 아래와 같이 진행되면서 remote로부터 브랜치 정보를 가져오게 된다.

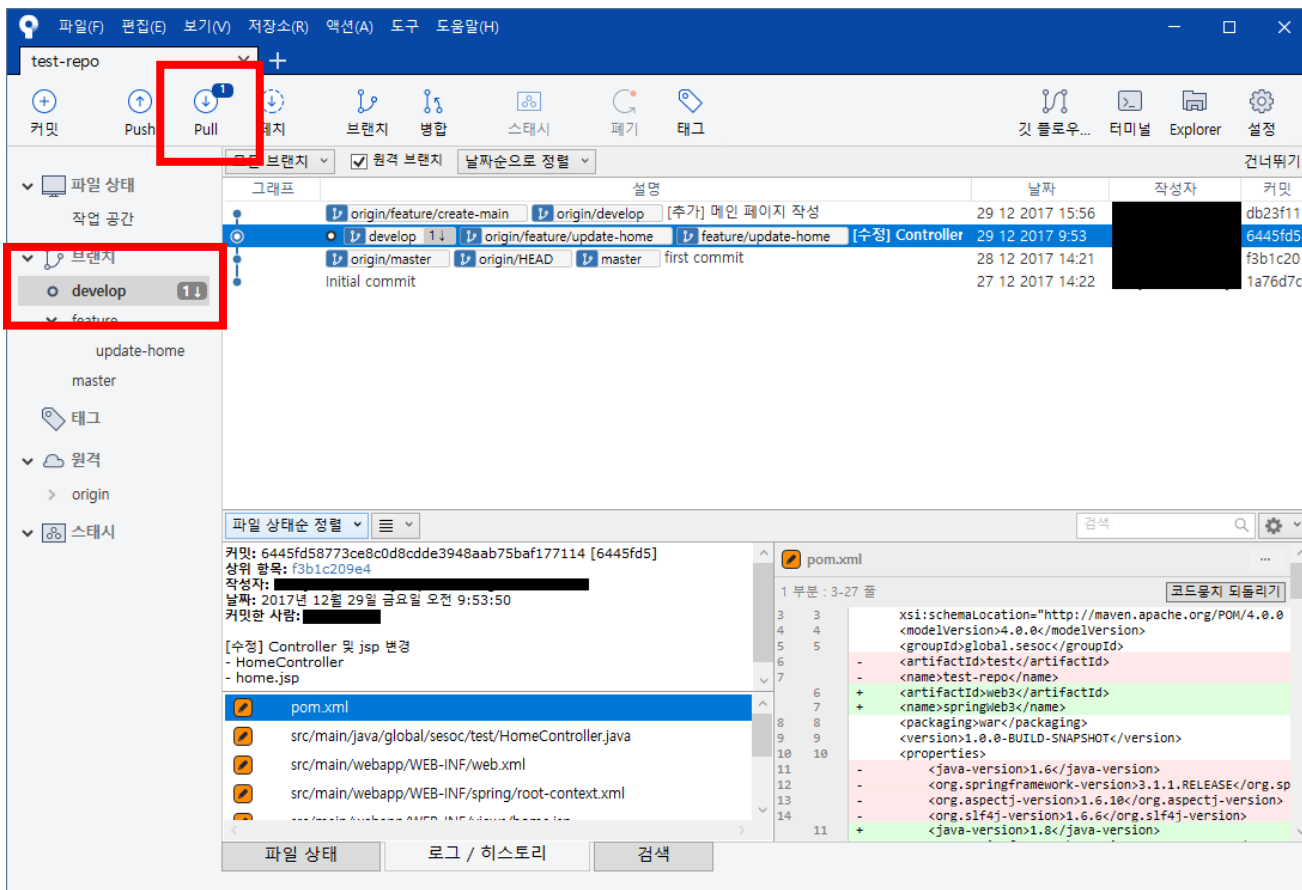
git fetch



만약 다른 사람이 진행한 파일이 존재
한다면 그림과 같이 추가되는 것이 확
인될 것이다.

git pull

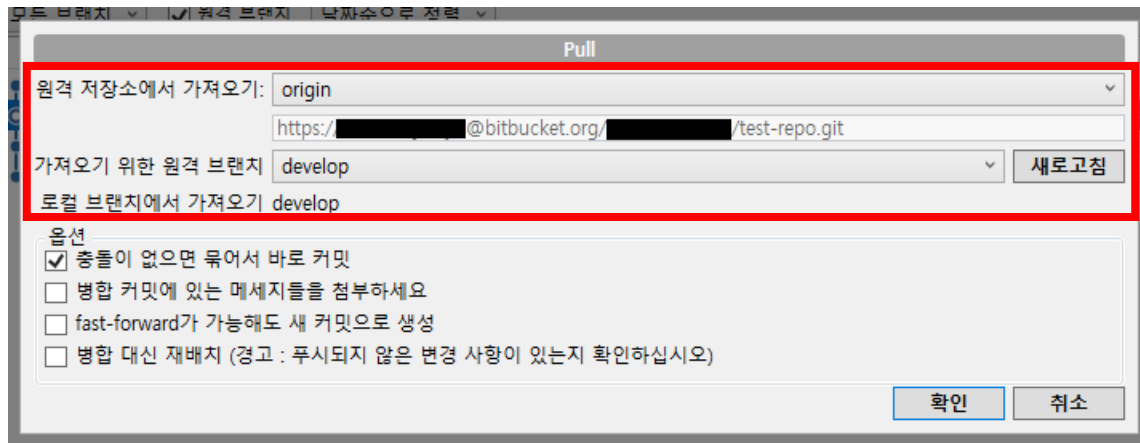
git pull



만약 다른 프로그래머가 push를 진행하였고 그것을 fetch하였다면 다음과 같이 상단 메뉴의 [Pull]에 **몇 개의 commit**이 밀려 있는지 뜨게 된다.

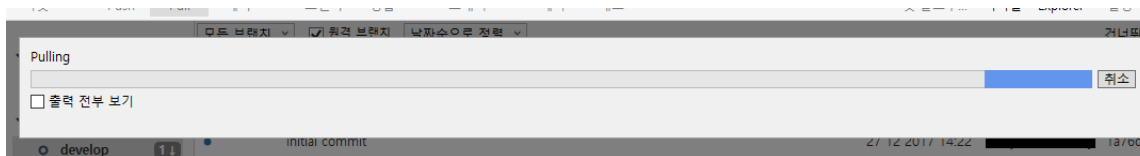
이를 나의 local 저장소에 반영하기 위해서는 pull을 진행해야 된다.

git pull



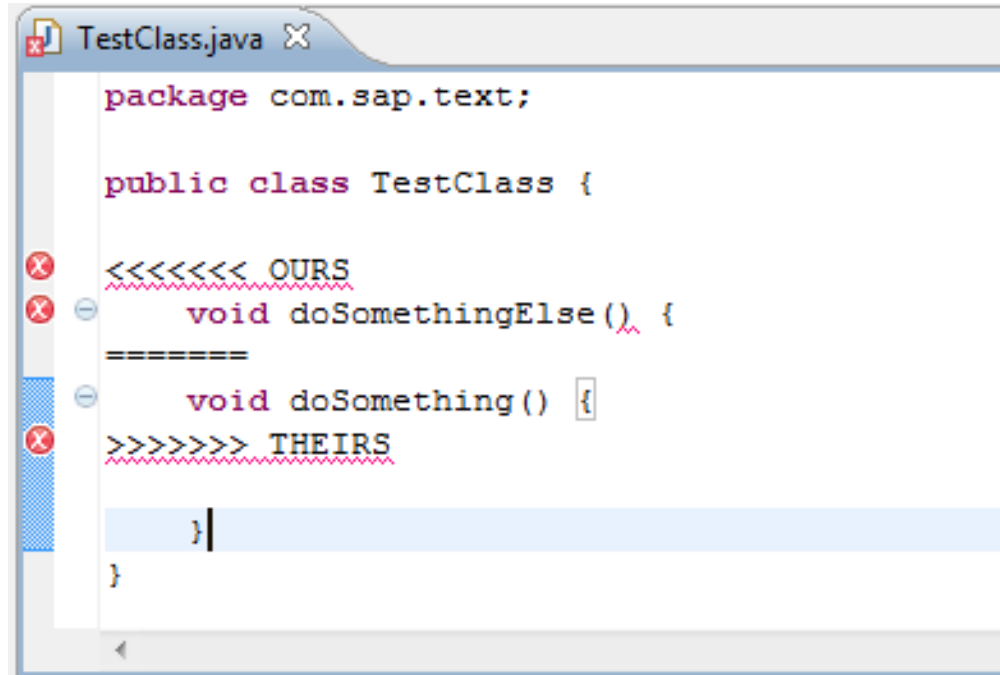
[원격 저장소에서 가져오기]를 확인하고 (대부분은 origin일 것이다), [가져오기 위한 원격 브랜치]가 현재 pull을 진행할 브랜치인지 확인한다.

[확인] 버튼을 누르면 pull이 진행될 것이다.



conflict

충돌(conflict)이란?



```
TestClass.java X
package com.sap.text;

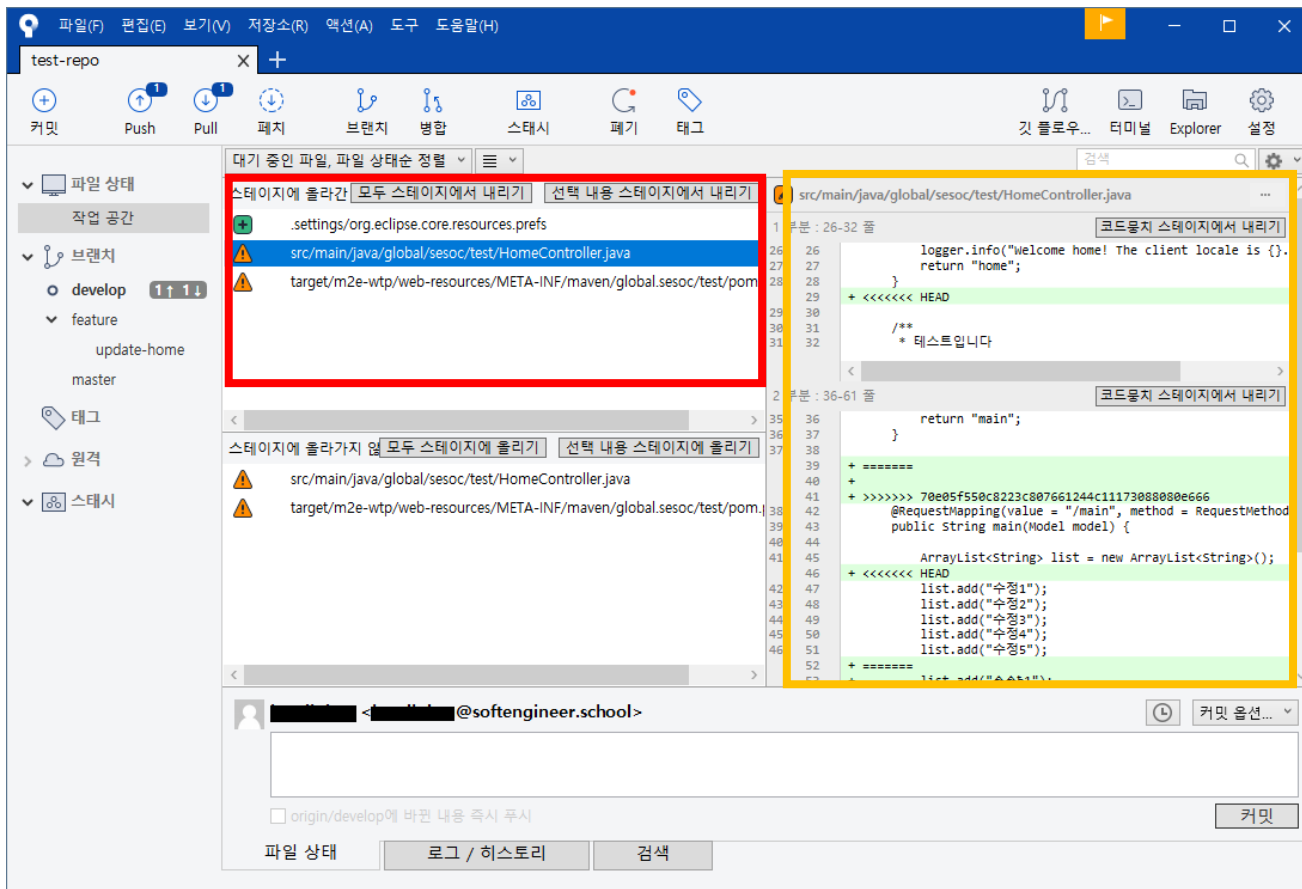
public class TestClass {

<<<<<<< OURS
    void doSomethingElse () {
=====
    void doSomething() {
>>>>>> THEIRS

    }
}
```

서로 다른 이가 같은 파일을 변경하여,
병합 시에 에러가 나는 것을 충돌했다고 표현한다

conflict

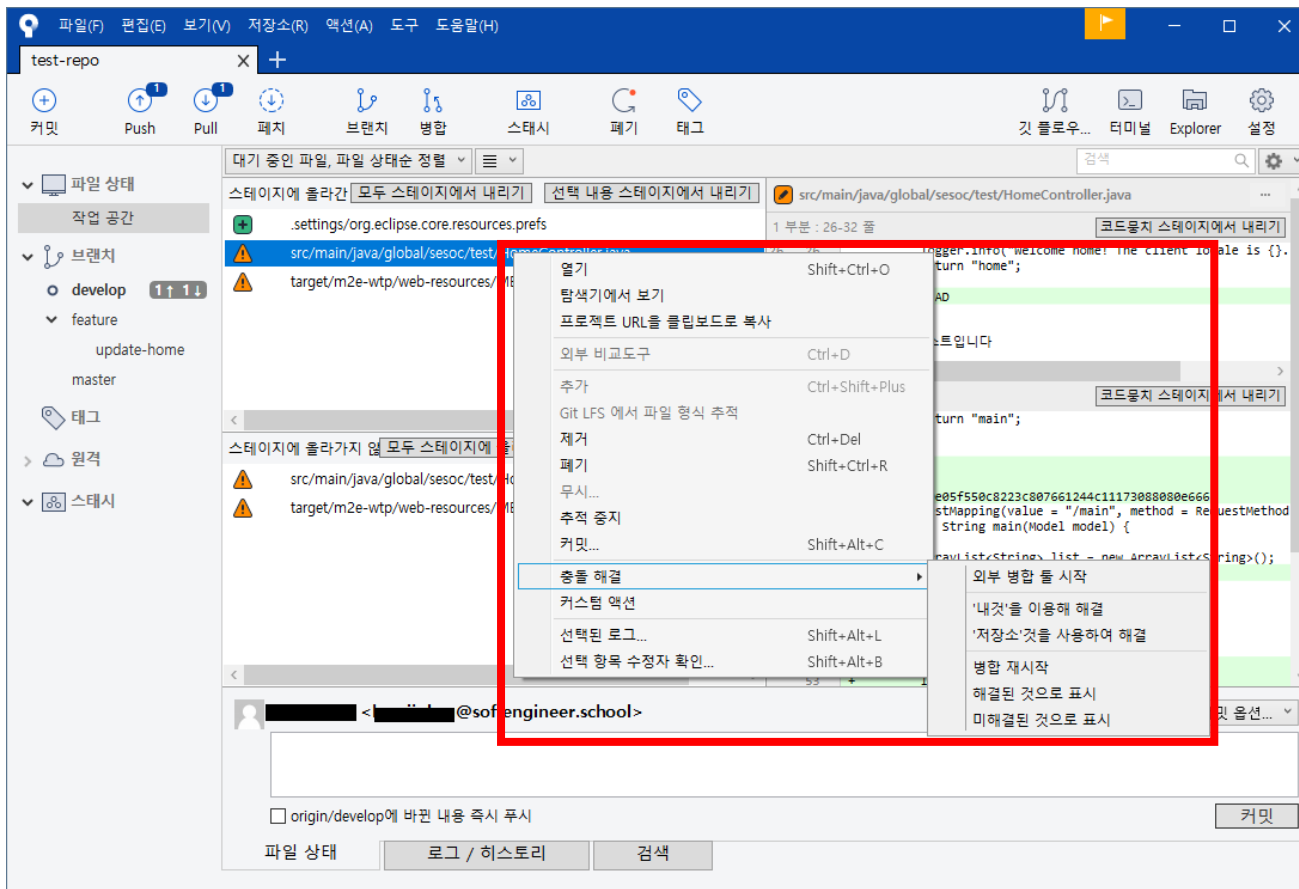


코드를 수정하고 commit을 하고 나니 **다른 프로그래머가 push**를 한 사실이 나타났다.

pull을 받아보니 다음과 같이 같은 파일을 수정한 것을 확인할 수 있었다.

이러한 경우 파일에 **conflict**가 발생한다.

conflict



해결 방법은 다음과 같다.

첫번째, local의 내용을 전부 remote에 반영하거나, 혹은 remote의 내용을 전부 local에 반영하거나 둘 중 하나를 고른다.

왼쪽 그림과 같이 충돌 난 파일에 오른쪽 클릭을 하면 메뉴가 뜨고 그 가운데 [충돌 해결]이란 메뉴가 있다.

메뉴 안에서 다음 중 하나를 고르면 된다.

- 1) '내것'을 이용해 해결
- 2) '저장소'것을 사용하여 해결

conflict

```
25 public String home(Locale locale, Model model) {
26     logger.info("Welcome home! The client locale is {}.", locale);
27     return "home";
28 }
29 <<<<<< HEAD
30
31 /**
32  * 테스트입니다
33  * @return 테스트
34  */
35 public String test() {
36     return "main";
37 }
38
39 =====
40 o o
41 >>>>>> 70e05f550c8223c807661244c11173088080e666
42 @RequestMapping(value = "/main", method = RequestMethod.GET)
43 public String main(Model model) {
44
45     ArrayList<String> list = new ArrayList<String>();
46 <<<<<< HEAD
47     list.add("수정1");
48     list.add("수정2");
49     list.add("수정3");
50     list.add("수정4");
51     list.add("수정5");
52 =====
53     list.add("h1");
54     list.add("h2");
55     list.add("h3");
56     list.add("h4");
57     list.add("h5");
58 >>>>>> 70e05f550c8223c807661244c11173088080e666
59
60     model.addAttribute("list", list);
61
62     return "main";
63 }
64
```

두번째는, 파일을 직접 열어서 수정하고 commi을 한다.

왼쪽 그림과 같이 충돌 난 파일을 들어가보면 다음과 같이 나올 것이다.

<<<<<< [code]

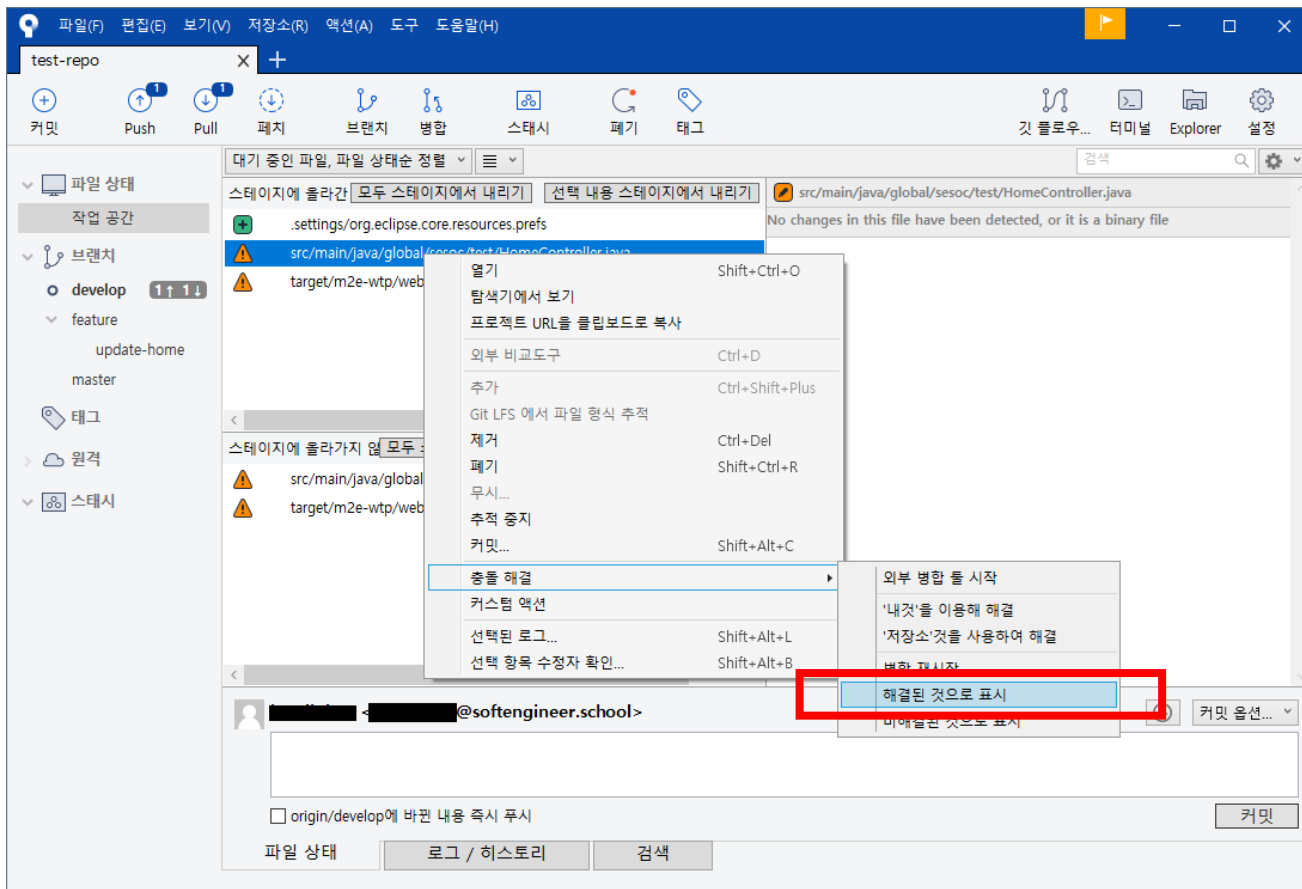
=====

>>>>>> [code]

'===== '을 기준으로 위, 아래로 코드가 나뉘어 있으며 이는 local과 remote의 코드를 나누어 놓은 것이다.

둘 중 적절하다고 판단되는 코드로 수정한다.

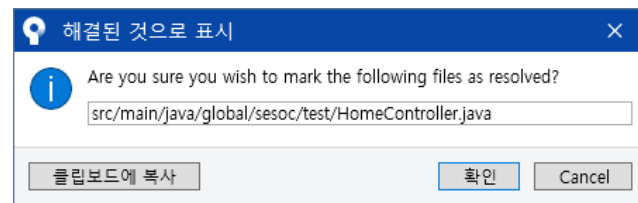
conflict



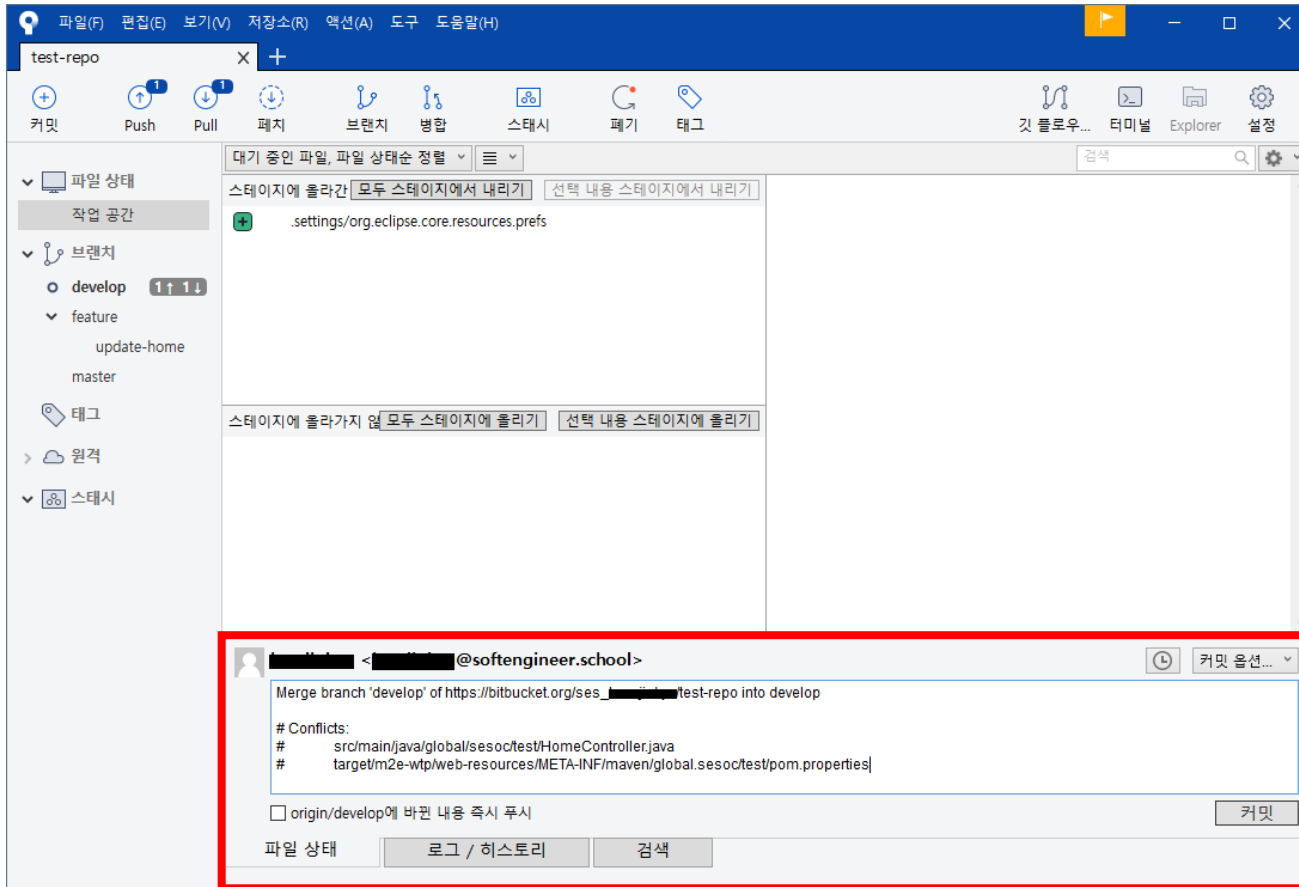
직접 파일을 수정하여 수정이 정상적으로 완료되었다면, SourceTree에서 왼쪽 그림과 같이 오른쪽 클릭을 하고, 충돌 해결 메뉴에서 [해결된 것으로 표시]를 선택한다.

그럼 아래와 같은 창이 나오면서 해결된 것으로 표시할 지를 묻는다.

확인을 누르면 전부 해결된 것이다.



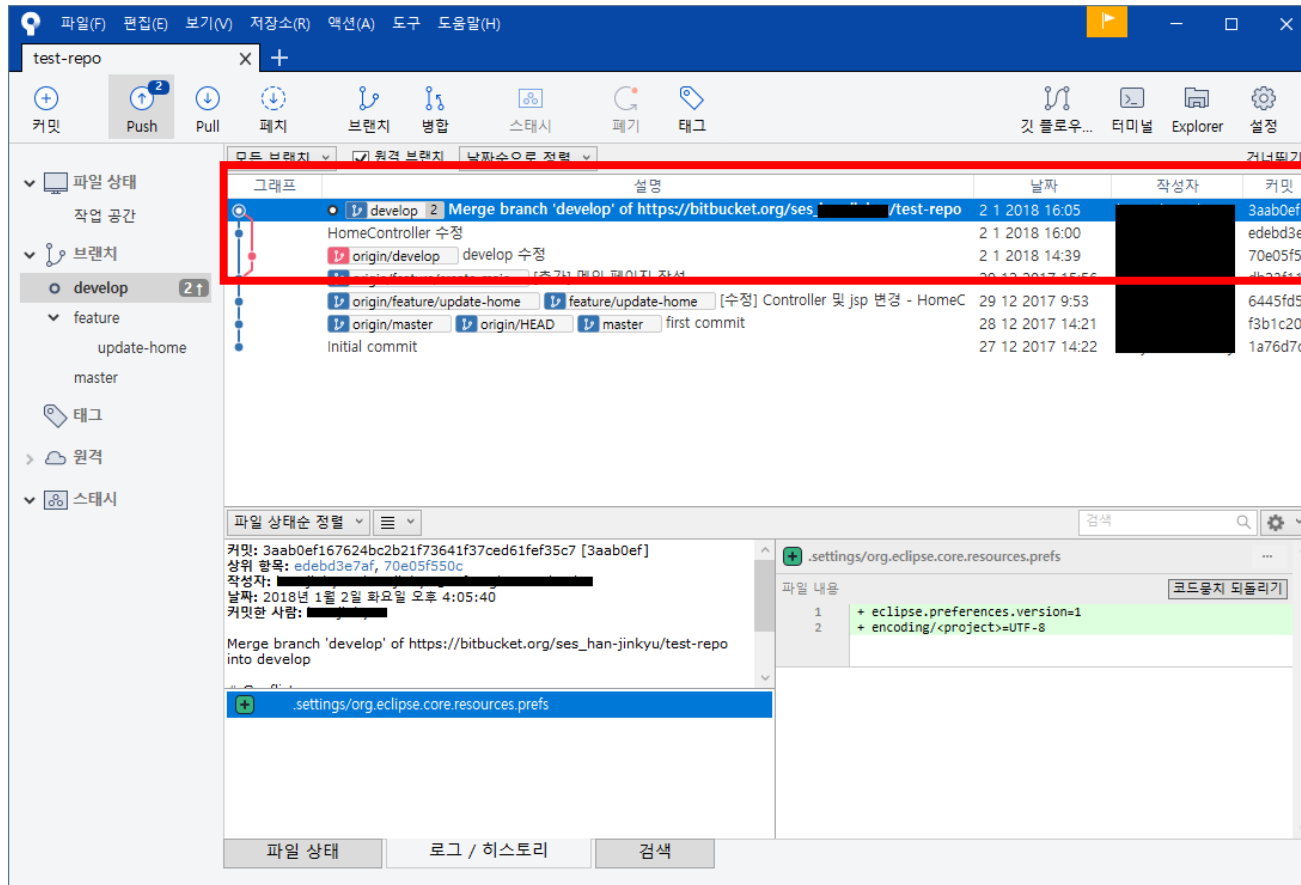
conflict



전부 수정하고 나면 commit 창에 다음과 같이 자동으로 수정된 내용이 들어있을 것이다.

이대로 commit한다.

conflict



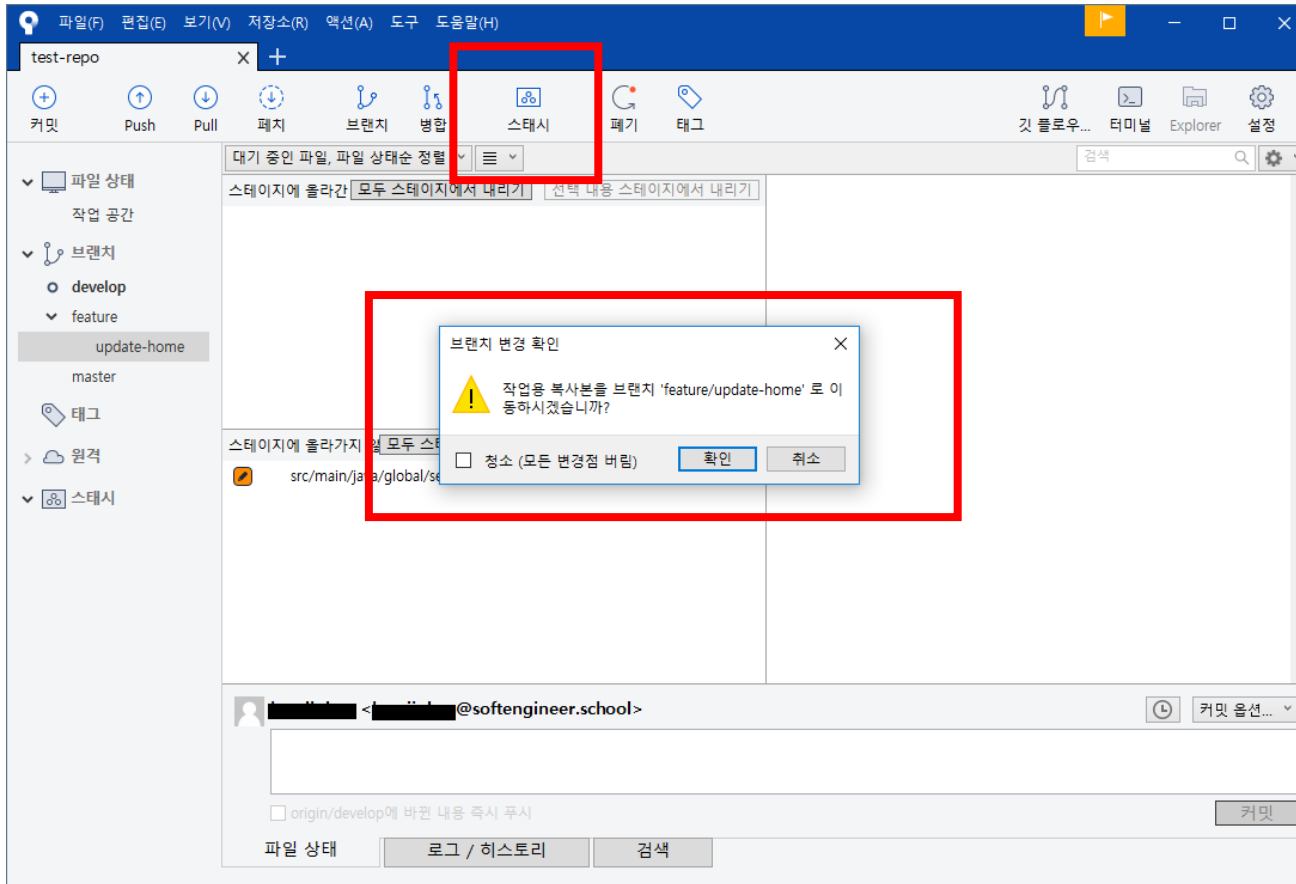
그러면 다음 그림과 같이 해결하고 머지한 것으로 표현되며, push를 진행하여 remote에 반영하도록 한다.

git stash

스태시(stash)이란?

현재 변경사항을 잠시 보관하는 것을 뜻한다.

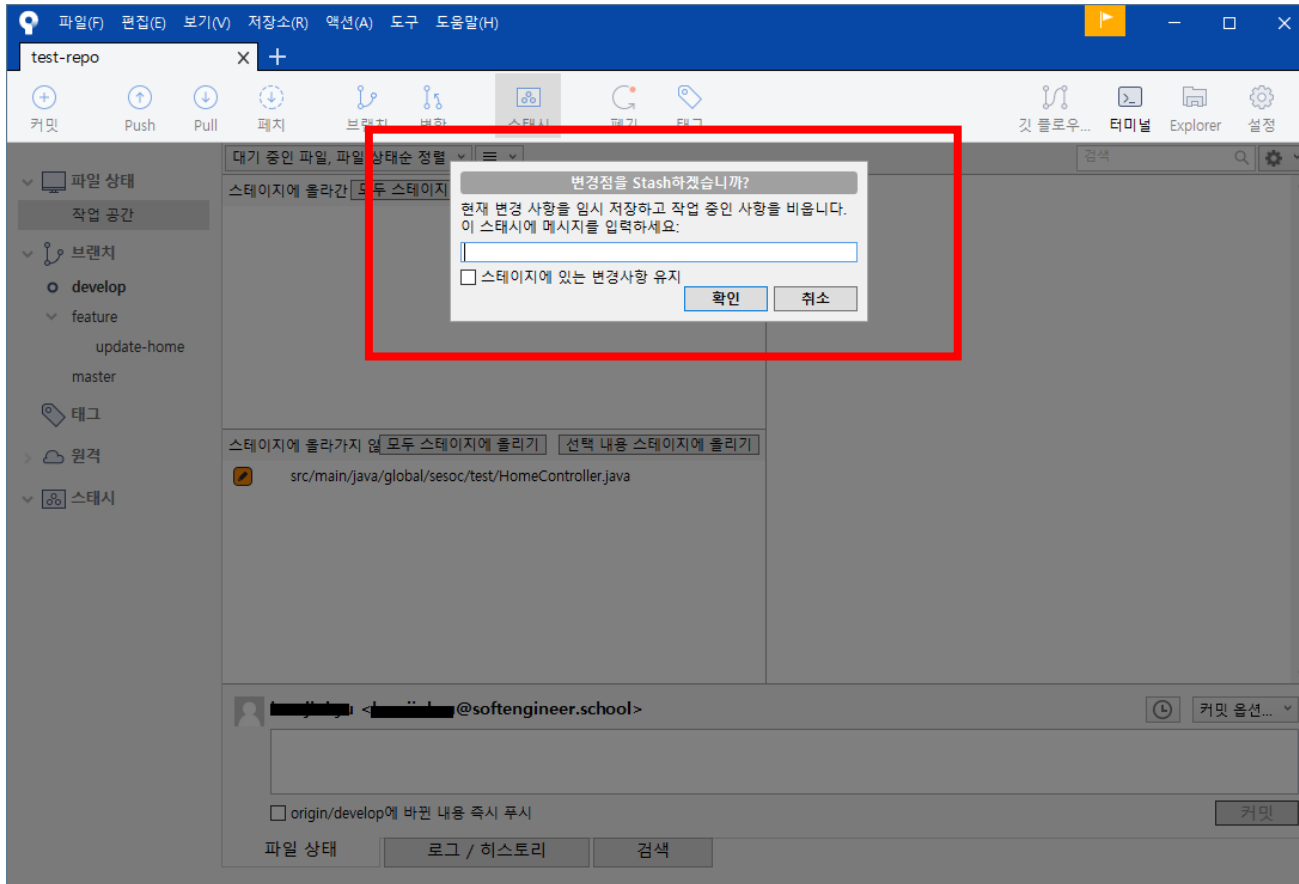
git stash



그러면 다음 그림과 같이 해결하고 머지한 것으로 표현되며, push를 진행하여 remote에 반영하도록 한다.

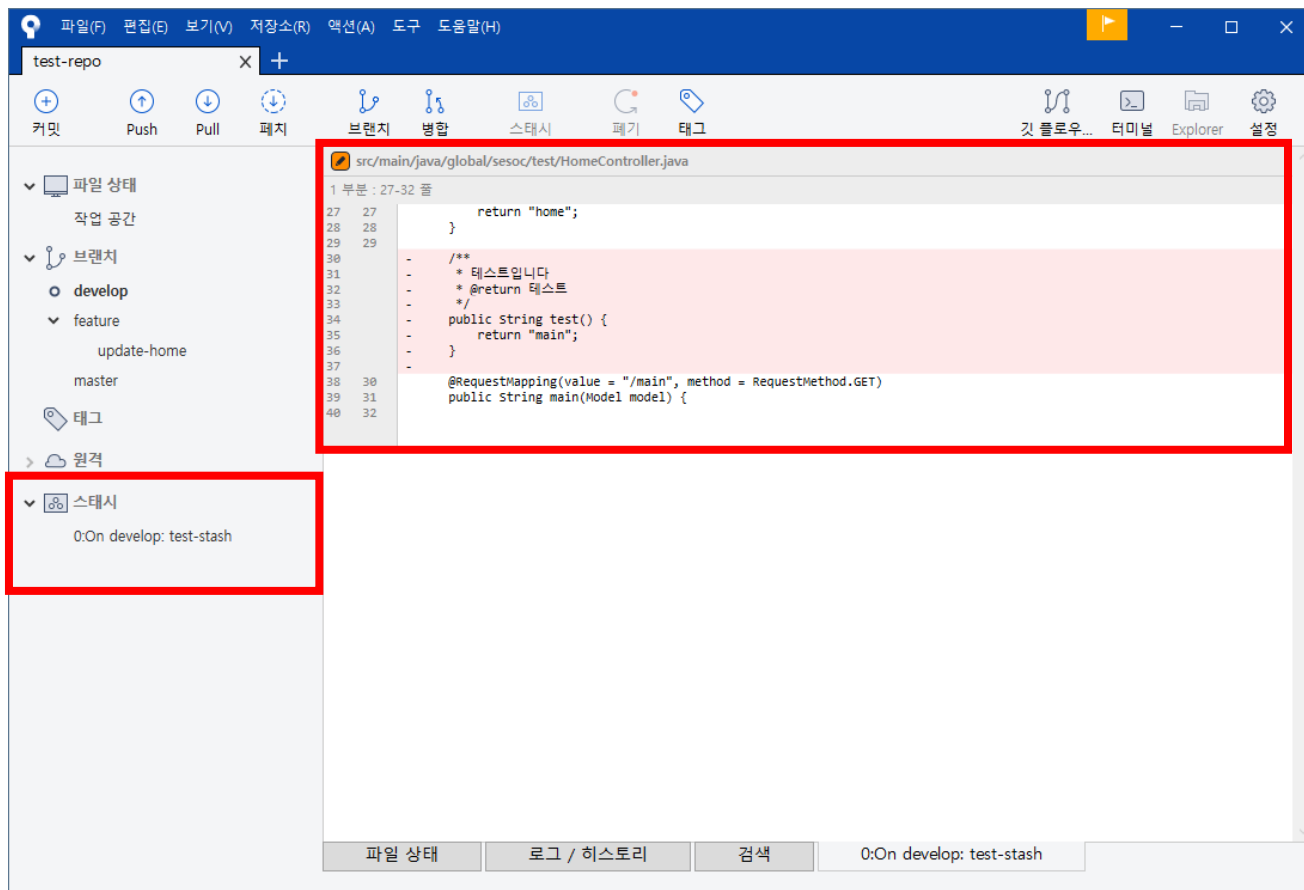
상단 메뉴의 [스테시] 버튼을 누른다.

git stash



[변경점을 Stash하겠습니까?]라는 창이 뜨면 텍스트필드에 저장하고 싶은 이름을 쓰고 확인을 누른다.

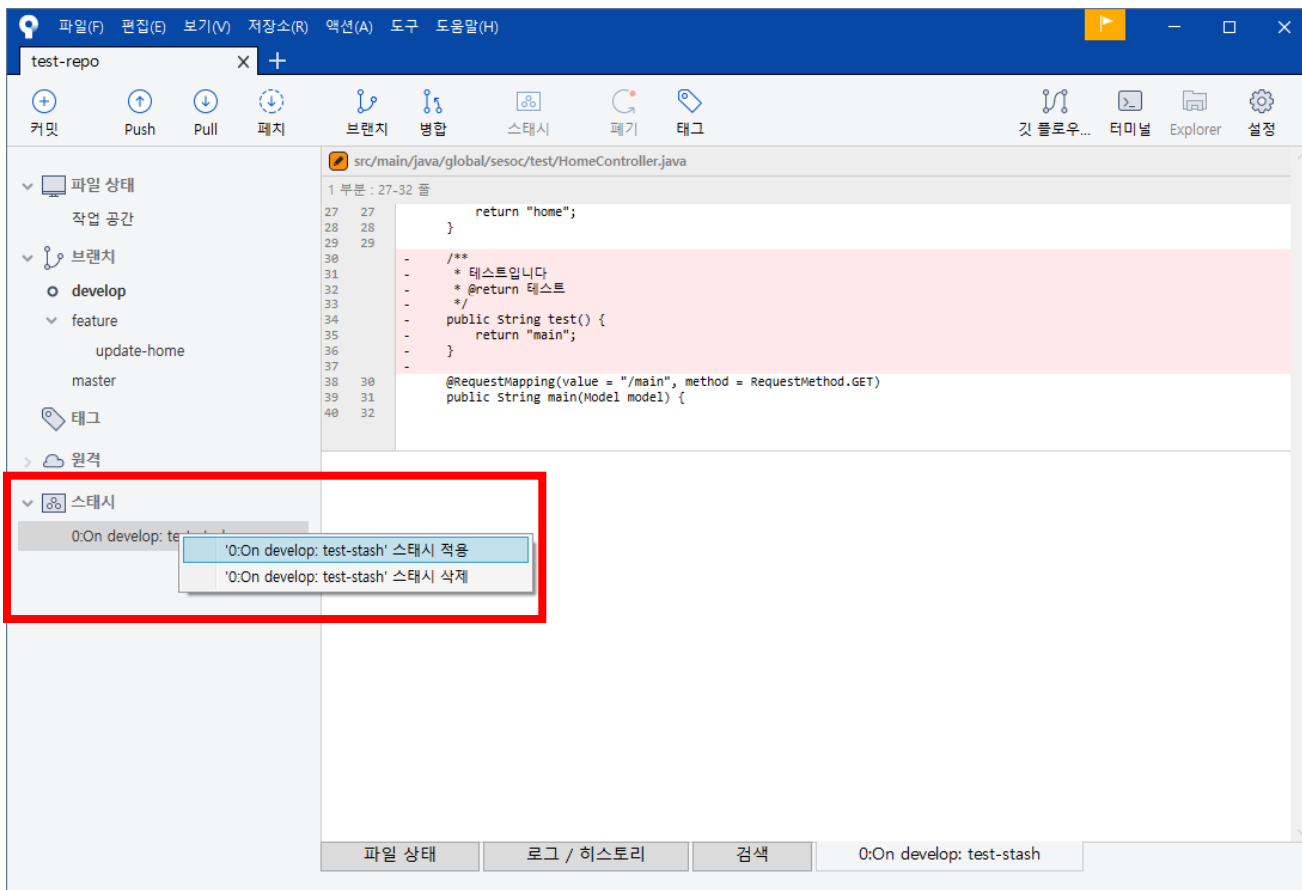
git stash



Stash가 완료되면 화면의 왼쪽과 같이 스테시 항목에 저장한 이름과 함께 무언가가 생긴다.

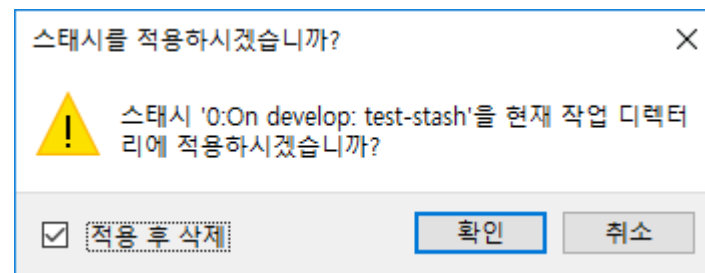
이를 찍어보면 오른쪽과 같이 어떤 변경점이 존재했는가에 대해서 나온다.

git stash



이 Stash를 다시 브랜치로 돌려놓고자
한다면 Stash를 오른쪽 클릭하고 [스태
시 적용] 을 클릭한다.

그러면 Stash되었던 모든 변경된 파일들
이 **현재 체크아웃한 브랜치로 복구된다.**



더 좋은 정보를 얻고 싶다면 한글로 친절하게 설명된
[이곳](#)을 클릭하여 확인해보자.

감사합니다

Git /w SourceTree