

다소 불친절하고

간단한

Eclipse – Git 사용 메뉴얼

<For KOSMO java 52th>

목차

1. Git 이해하기

1-1. Git이란?

1-2. Git의 저장구조

2. Eclipse로 Git 사용하기

2-0. EGit 환경구축(EGit Setting)

2-1. 프로젝트 공유(Project Share)

2-1-1. 원격저장소에 프로젝트 올리기(Share)

2-1-2. 원격저장소의 프로젝트 내려받기(Clone)

2-2. 프로젝트 협업(Collaboration)

2-2-1. Commit and Push

2-2-2. Fetch and Pull

2-2-3. branch and Merge

2-2-4. Merge Tool (Confilct)

3. 참고자료

Git 이해하기

1-1. Git이란?

① Git은 프로그램의 소스 코드 관리를 위한 VCS(버전 관리 시스템)입니다.
VCS를 사용하는 이유는 크게 2가지 입니다.

- 팀 단위의 프로젝트에서 소스코드 공유
- 버전을 부여해서 소스코드의 이력을 관리

VCS의 종류는 3가지가 있습니다.

- 로컬 버전 관리 시스템
- 중앙집중식 버전 관리 시스템
- 분산 버전 관리 시스템

Git은 그 중 분산 버전 관리로 이는 서버에 있는 저장소 자체를 사용자의 컴퓨터로 통째로 받아오는데 이때 소스코드에 변경이력까지 모든 정보를 가져와 로컬컴퓨터 또한 완전한 저장소가 된다는 뜻입니다.

이렇게 됨으로써 이후 개발작업에서는 서버와는 별개로 자신의 로컬에서 진행하게 되고, 로컬이니 빠른 속도로 변경 할 수 있습니다.

Git은 Branch를 마음껏 할 수 있습니다. Backlog나 Bug단위로 Branch를 만들어 개발하고 release하는 Branch(master)는 clean하게 유지 할 수 있습니다.

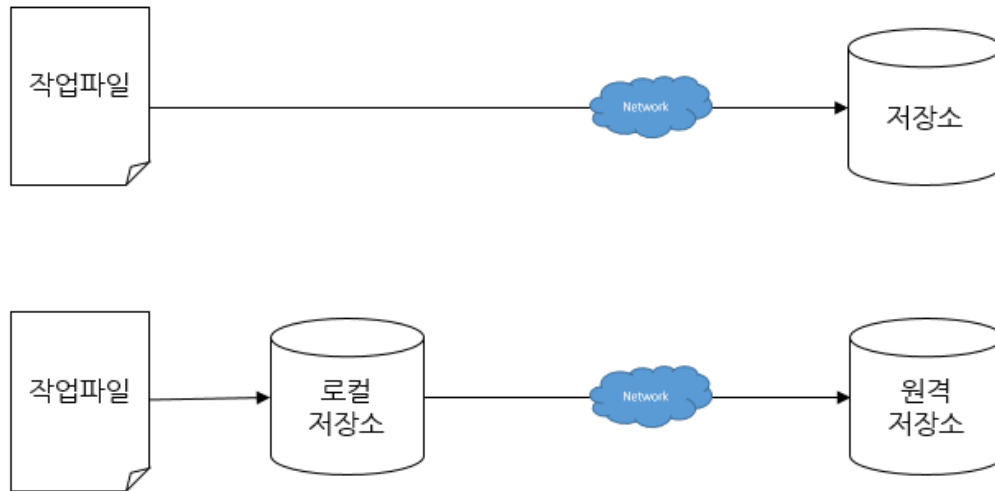
1-2. Git의 저장구조

① Git이 버전 관리를 하는데 있어 저장구조는 3가지 특징을 갖습니다.

- 로컬저장소의 존재
- 스테이지 영역
- 스냅샷

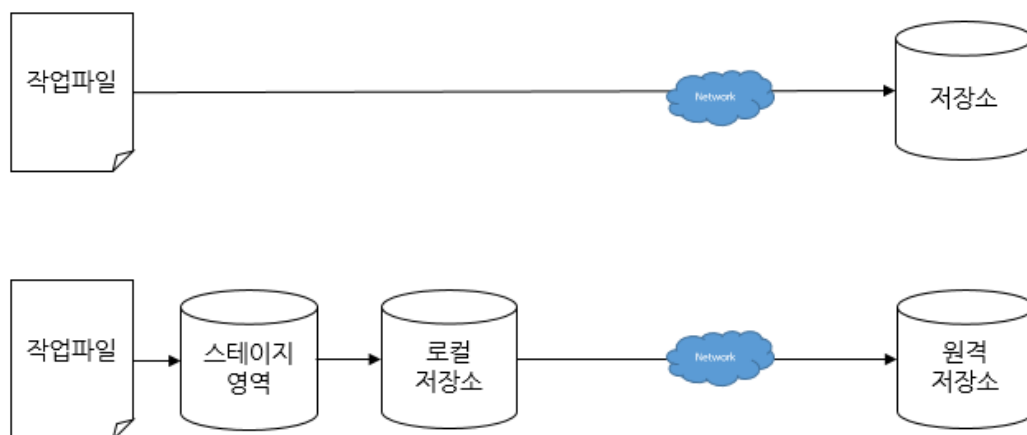
첫 번째로 로컬저장소의 존재입니다.

Git은 로컬저장소(Local repository)와 원격저장소(Remote repository) 두가지를 가지고 있습니다. .



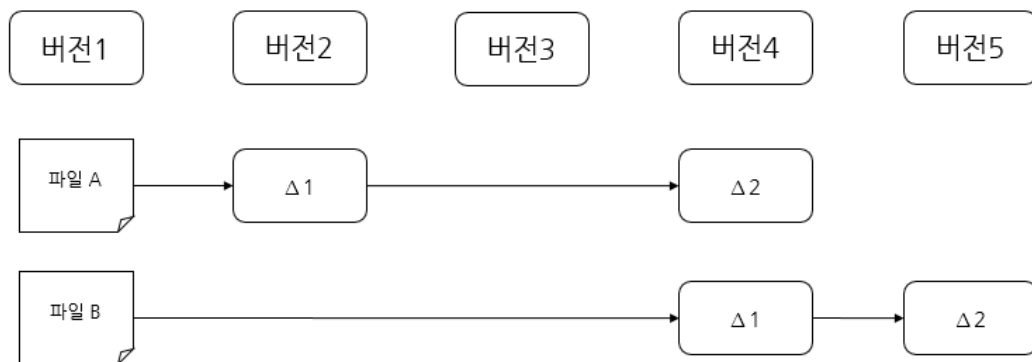
Git을 이용할 때에는 작업파일을 로컬저장소에 먼저 commit한 후, 원격저장소에 push합니다. 이렇게 로컬저장소가 따로 있으면 버전 관리에 용이합니다.

두 번째로 스테이지 영역입니다.



Git은 로컬저장소에 commit하기 전에 commit할 파일들만을 스테이지 영역에 추가합니다. '커밋될 예정인 파일'을 미리 스테이지 영역에 언제든지 add 시키고 스테이지 영역에 있는 파일을 한꺼번에 commit합니다.

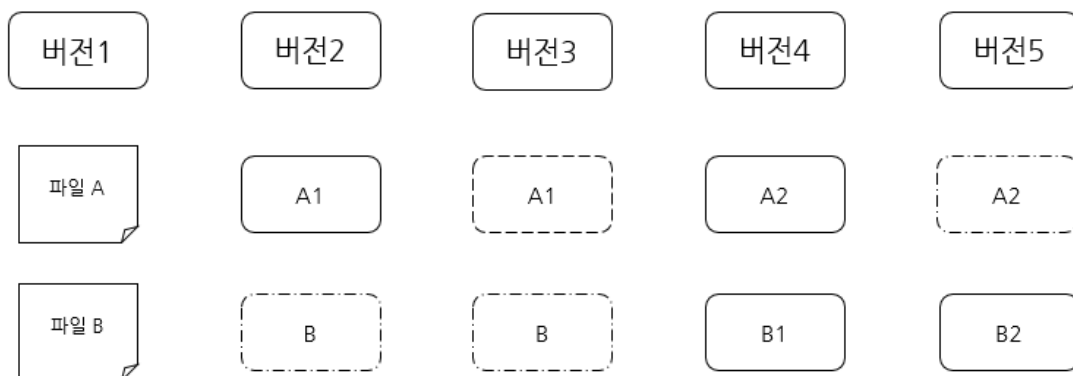
세 번째로 스냅샷입니다.



중앙집중식 버전 관리 시스템인 SVN은 가장 처음에 생성된 파일을 저장하고 그 이후에는 델타라고 부르는 '파일의 변화(차이점)'를 저장합니다.

따라서 버전5를 내려받기 위해서는 버전1의 파일, 그리고 그 이후의 델타들이 전부 필요합니다.

때문에 SVN은 커밋횟수가 많아질수록 내려받는 속도가 느려지는 단점이 있습니다. 그래서 commit은 파일단위가 아닌 작업단위로 해야합니다.



반면 Git은 버전별로 스냅샷으로 저장하는데, 델타를 저장하는 것이 아니라 변경된 파일 자체를 저장합니다. 따라서 버전5를 내려받기 위해서는 버전4의 A2만을 가져옵니다.

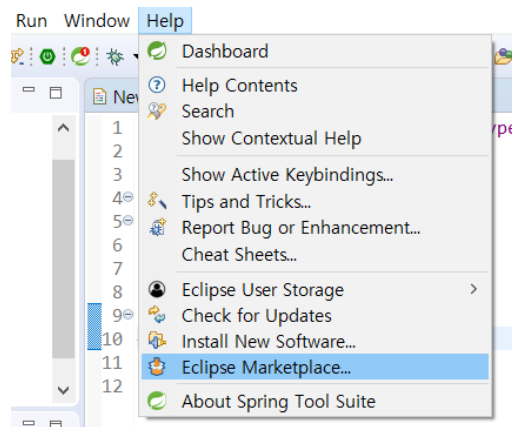
SVN에 비해 가장 최신 파일 한 개만 내려받을 수 있어 빠르지만, 저장용량이 커진다는 단점이 있습니다.

Eclipse로 Git 사용하기

2-0. EGit 환경구축(EGit Setting)

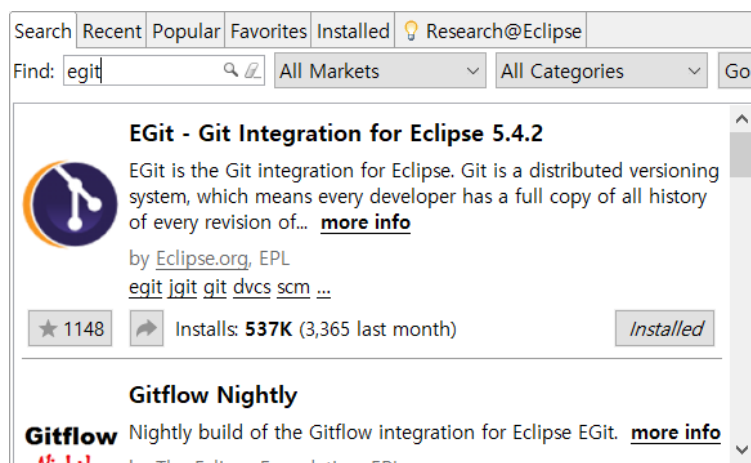
이클립스에서 Git을 사용하기 위해서는 EGit 툴을 설치해야 합니다. (2014버전인 이클립스 Luna이후 버전이나 STS를 이용하고 있다면 EGit이 기본적으로 제공되므로 설치과정은 생략해도 무방합니다.)

이클립스를 실행한 뒤 [Help] -> [Eclipse Marketplace]를 클릭합니다.



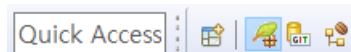
마켓 플레이스의 'Find'항목에 Egit을 검색하고 설치합니다.

(본 매뉴얼에서는 STS환경에 기본적으로 EGit이 installed 되어있는데 혹시라도 설치되어있지 않다면 install 해줍니다.)



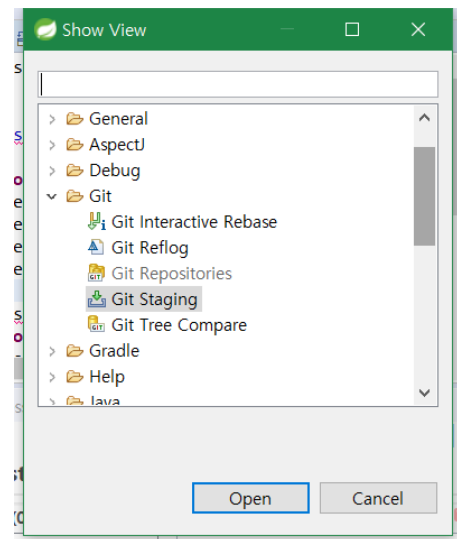
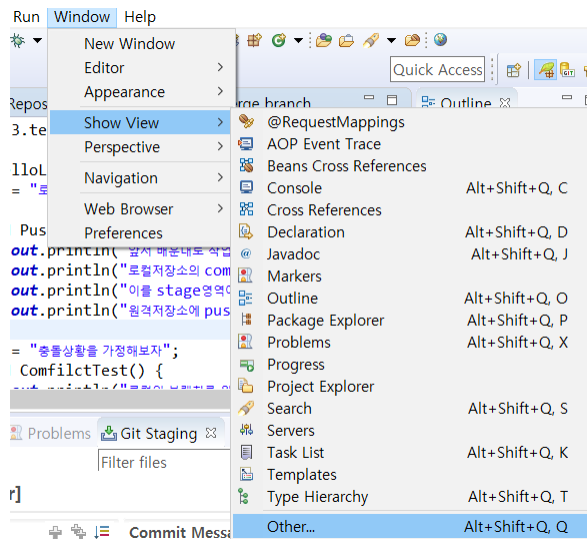
재실행 후에 이클립스의 우상단에 조그마한 영역으로 Perspective항목이 보입니다. Git 퍼스펙티브를 활성화하기 위해선 좌측의 [Open Perspective]-[Git]을 선택한 후 [Ok]를 눌러 활성화시킵니다.

이전 퍼스펙티브로 돌아가고 싶다면 우측의 [Java EE] 혹은 [Spring]를 누르면 기존 환경으로 돌아갑니다.



기존 환경에서 Git의 해당 윈도우를 직접 열어서 사용도 가능합니다.

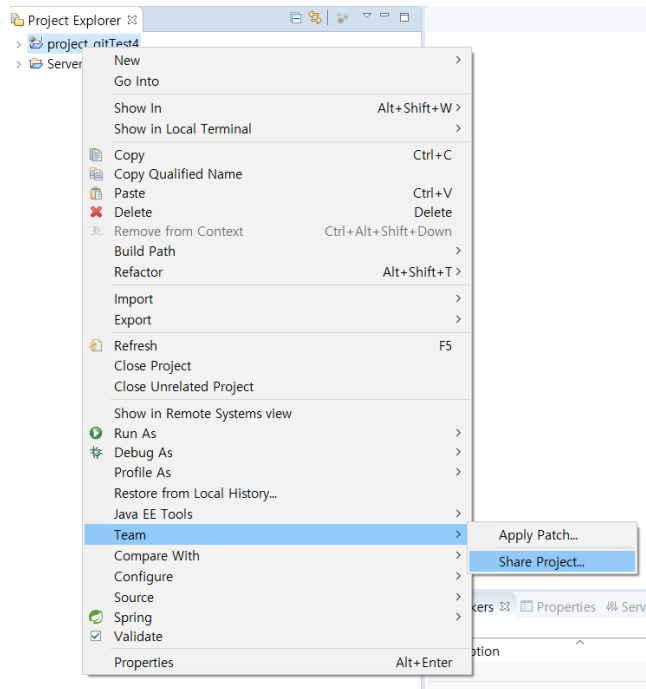
[Window] -> [Show View] -> [Git]을 선택 후 원하는 창을 [Open]하면 됩니다.



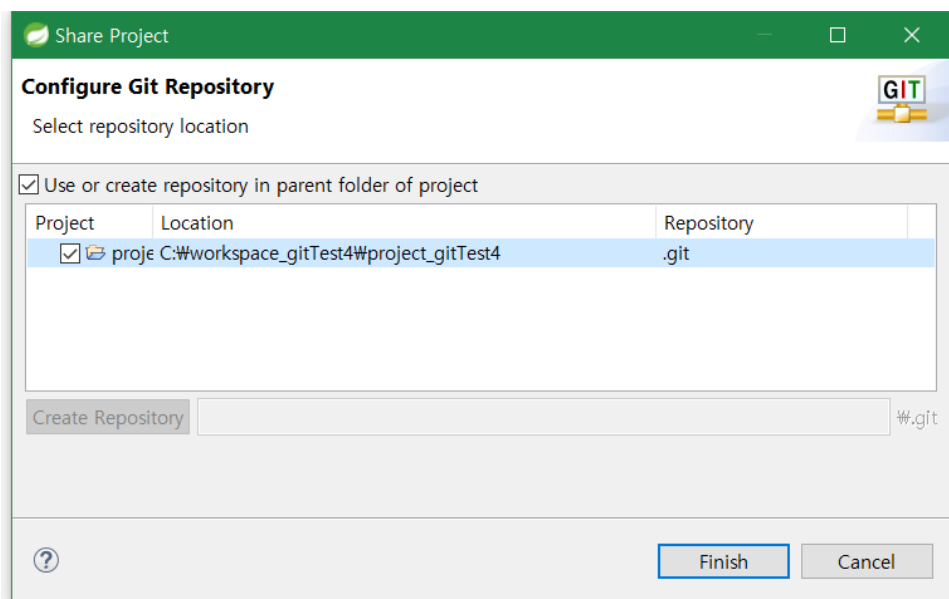
2-1. 프로젝트 공유(Project Share)

2-1-1. 원격저장소에 프로젝트 올리기(Share)

공유할 프로젝트에 마우스 오른쪽 버튼을 클릭하고 [Team] -> [Share Project]를 클릭합니다.

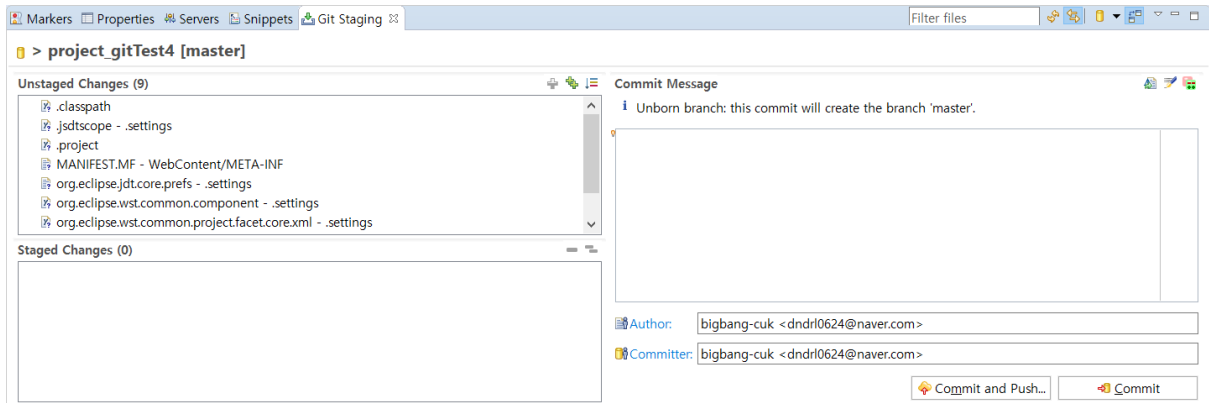


'Configure Git Repository'창이 열리면 [Use or create repository in parent folder of project]를 체크하고 Git의 로컬저장소의 경로를 원하는 경로로 설정하고 [Finish]합니다.



이제 로컬저장소를 만들었고 이 저장소에 프로젝트를 commit합니다.

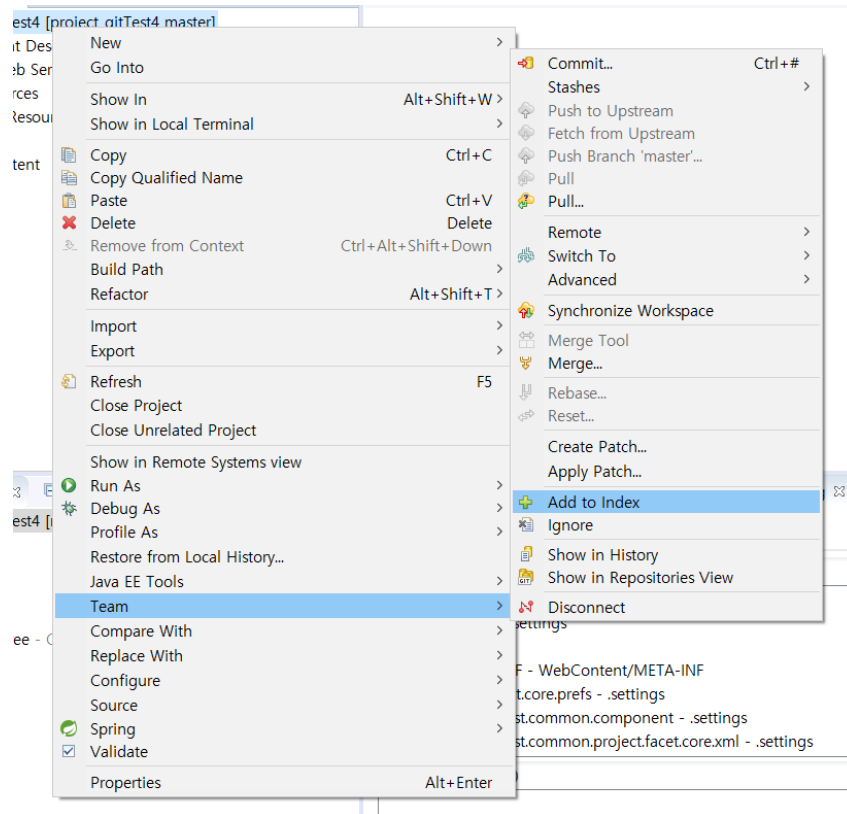
Git Staging 윈도우를 보면 '커밋될 예정인 파일'들의 목록이 Unstaged영역에 표시됩니다.



우측 두번째 아이콘인[Add all files...to the index]를 클릭하거나



프로젝트 오른쪽 마우스 -> [Team] -> [Add to Index]를 누르면 '커밋될 예정인 파일'들이 스테이지 영역에 들어오면서 '커밋 대상 파일'로 바뀝니다.



Commit Message를 입력하고 [Commit]을 클릭하면 로컬저장소로 commit이 완료됩니다.

Commit Message

Unborn branch: this commit will create the branch 'master'.

commit to local repository

<용어>

Commit : 내 프로젝트(작업파일)을 '로컬저장소'에 commit

Push : '로컬저장소'의 데이터를 '원격저장소'에 commit

<버튼기능>

Commit and Push : 로컬에 commit하고 바로 원격에도 commit(push)해줌

Commit : 로컬에 commit만함 (원격에 저장하려면 따로 push해줘야함)

*Commit과 Push를 따로하는것을 권장(안전함)

Author:

bigbang-cuk-branch <dndrl1301@gmail.com>

Committer:

bigbang-cuk-branch <dndrl1301@gmail.com>

Commit and Push...

Commit




이때 [Commit and Push]를 누르면 동시에 원격저장소로 push까지 이루어지는데 로컬에서의 버전관리와 로컬과 원격저장소간의 충돌방지를 위해서 [Commit]을 하고 push는 따로 하는 것을 권장합니다.

2-1-2. 원격저장소의 프로젝트 내려받기(Clone)

이번엔 원격의 프로젝트를 작업자의 로컬에 Clone해 보겠습니다.

[Git Repository] -> [Clone a Git repository]를 클릭합니다.

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](#)

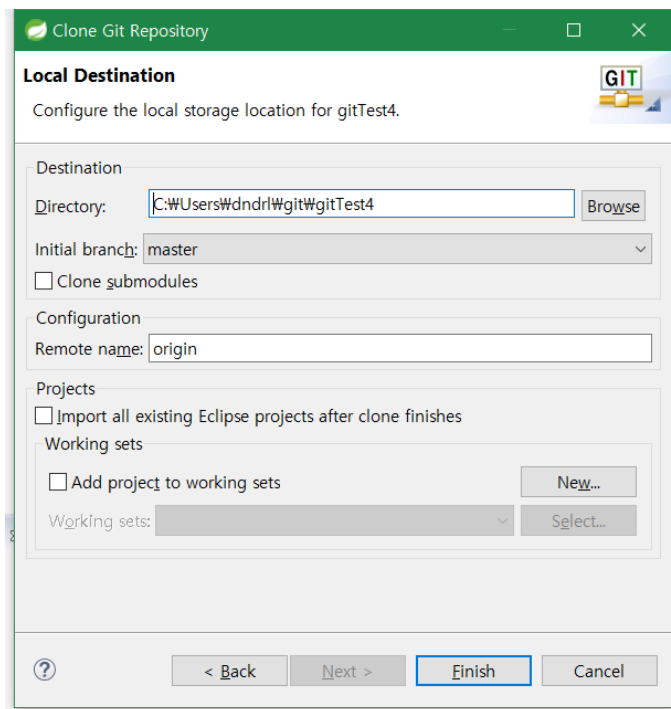
URI에 원격저장소의 주소를 적고 아래 User, Password에 본인 계정정보를 입력한 후 [Next]를 클릭합니다.

The screenshot shows the 'Clone Git Repository' dialog box with the 'Source Git Repository' tab selected. The dialog has a green title bar and a Git logo in the top right corner. The main area is divided into three sections: 'Location', 'Connection', and 'Authentication'. In the 'Location' section, the 'URI:' field contains 'https://github.com/bigbang-cuk-branch/gitTest4', and the 'Repository path:' field contains '/bigbang-cuk-branch/gitTest4'. In the 'Connection' section, the 'Protocol:' dropdown is set to 'https'. In the 'Authentication' section, the 'User:' field contains 'bigbang-cuk-branch' and the 'Password:' field is masked with black dots. There is an unchecked checkbox for 'Store in Secure Store'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

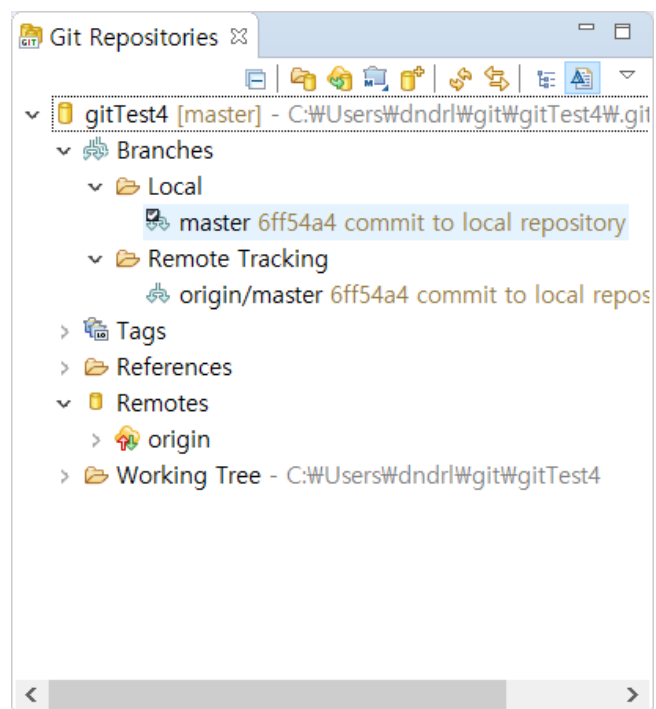
Branch Selection에서 로컬에 내려받을 브랜치를 선택합니다.(각 작업자가 필요한 브랜치만 받아도 됩니다.)

The screenshot shows the 'Clone Git Repository' dialog box with the 'Branch Selection' tab selected. The dialog has a green title bar and a Git logo in the top right corner. The main area is divided into two sections: 'Branches of https://github.com/bigbang-cuk-branch/gitTest4:' and a list of branches. The list contains one entry, 'master', which is checked with a checkbox. Below the list are buttons for 'Select All' and 'Deselect All'. At the bottom, there are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

브랜치를 정했다면 이제 로컬저장소가 될 경로를 지정해주고 [Finish]하면 원격저장소에서 로컬저장소로 프로젝트를 내려받게 됩니다.

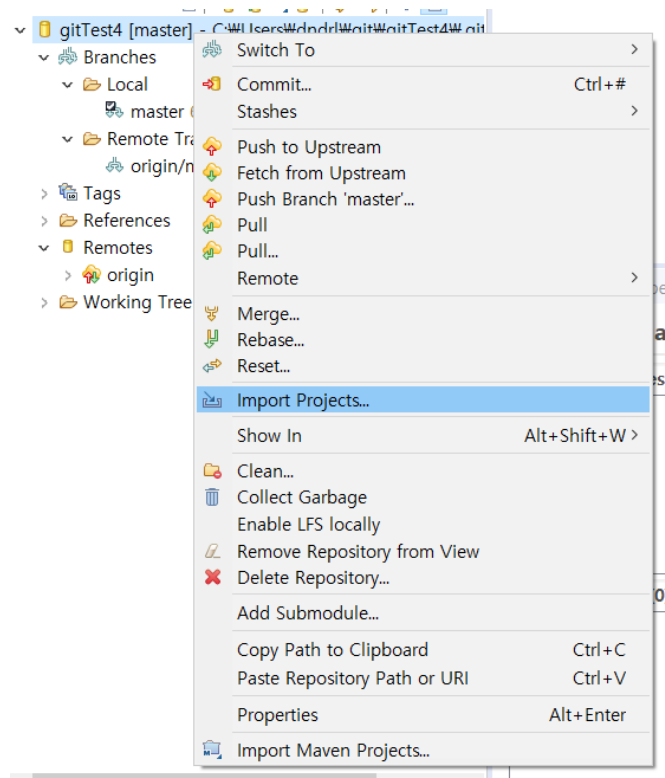


성공적으로 완료되면 Git Repository 윈도우에 내려받은 프로젝트가 브랜치명과 경로를 포함하여 표시됩니다.

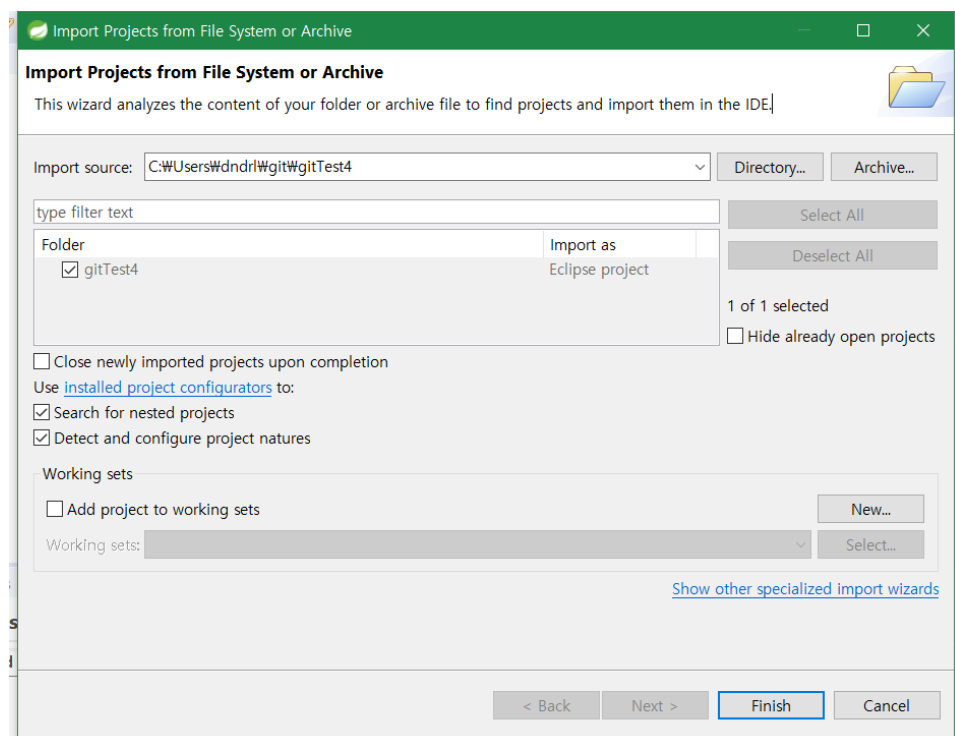


그러면 이제 로컬저장소로 가져왔으니 실제로 소스작업을 하기 위해 본인의 workspace로 해당 프로젝트를 Import시켜줍니다.

프로젝트 마우스 우클릭 -> [Import Project]를 선택합니다.



가져올 소스를 선택하고 [Finish]해줍니다.

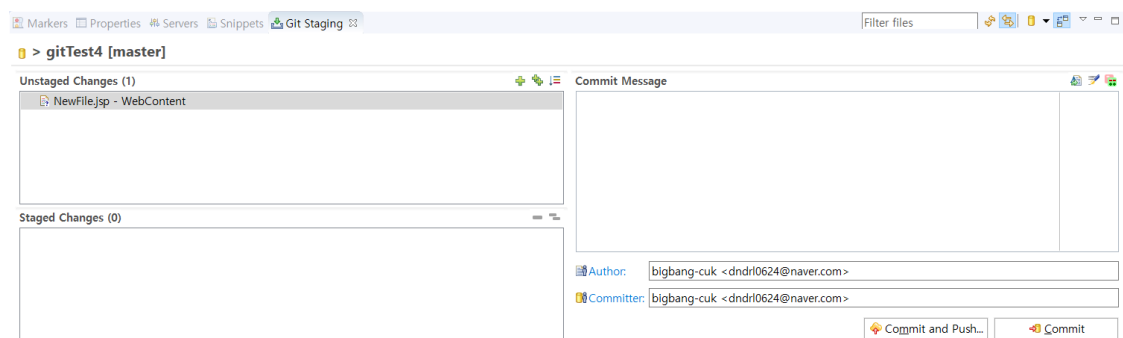


2-2. 프로젝트 협업(Collaboration)

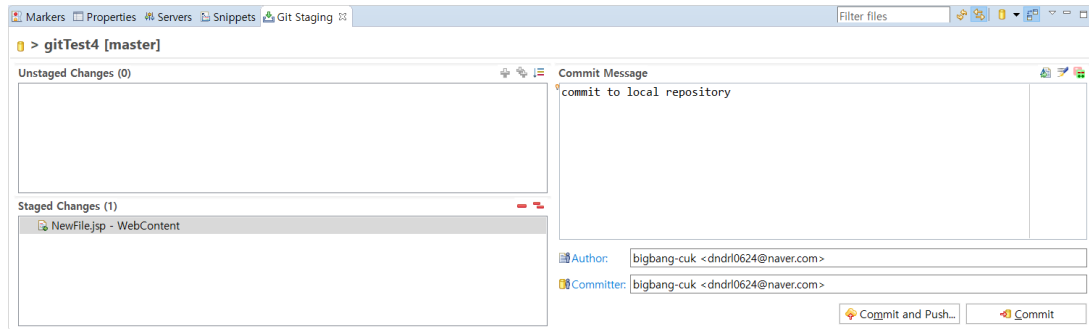
2-2-1. Commit and Push

이번엔 작업공간에서 작업한 소스를 로컬저장소에 commit해보고 원격저장소에 push하는 방법을 알아보겠습니다.

위와 같이 새로운 파일이 생성되거나 기존소스의 변경이 있으면 해당하는 파일들은 모두 '커밋될 예정인 파일'로 인식되어 Unstaged영역에 표시됩니다.



Project Explorer윈도우에서 프로젝트(혹은 해당파일)를 마우스 우클릭 [Team] -> [Add to Index]를 클릭하거나 Unstaged영역 우측의 [Add ~ to index]아이콘을 클릭하면 스테이지영역에 들어가며 '커밋 대상 파일'이 됩니다.



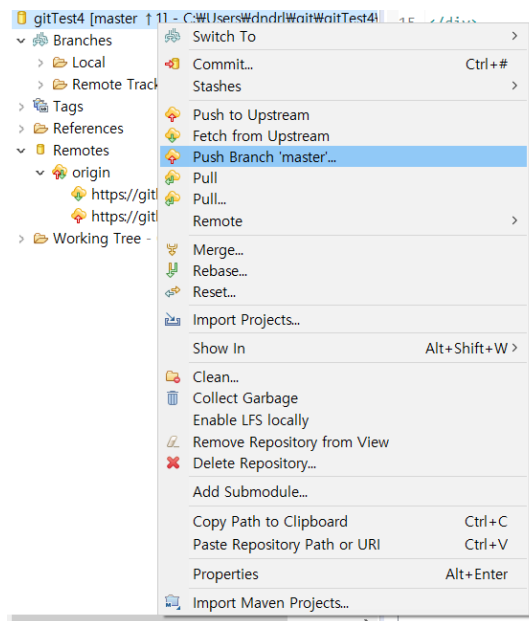
Commit Message 입력 후 [Commit]을 누릅니다. 성공적으로 commit이 되었다면 해당 브랜치명 옆에 새로운 표시가 생깁니다.

gitTest4 [master ↑ 1] -

'↑'은 Remote Tracking과 비교하여 '업로드 할 것이 있다'를 의미하고 그 뒤의 숫자는 대상 파일의 개수를 의미합니다.

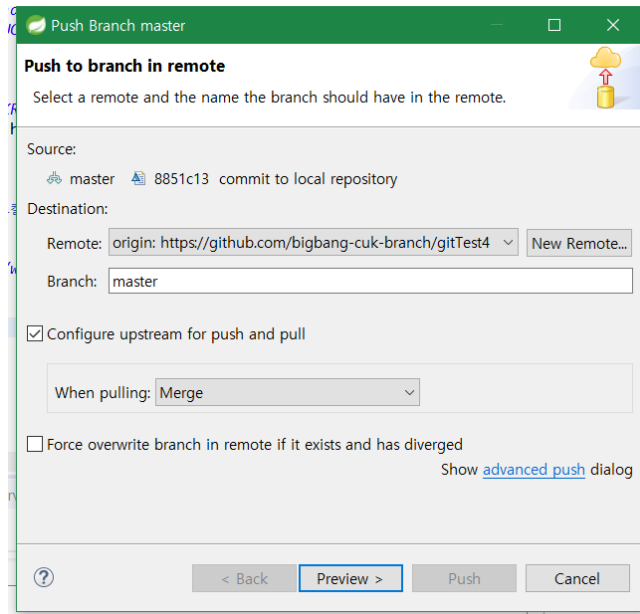
즉 로컬저장소에 commit을 하게 되면 동시에 원격저장소로 push할 것이 생긴다고 생각하면 됩니다.

Git Repository 윈도우에서 로컬저장소 마우스 우클릭 -> [Push Branch 'XXX']을 클릭합니다. 여기서 XXX은 본인이 현재 작업하고 있는(Check out상태에 있는) 브랜치 이름입니다.

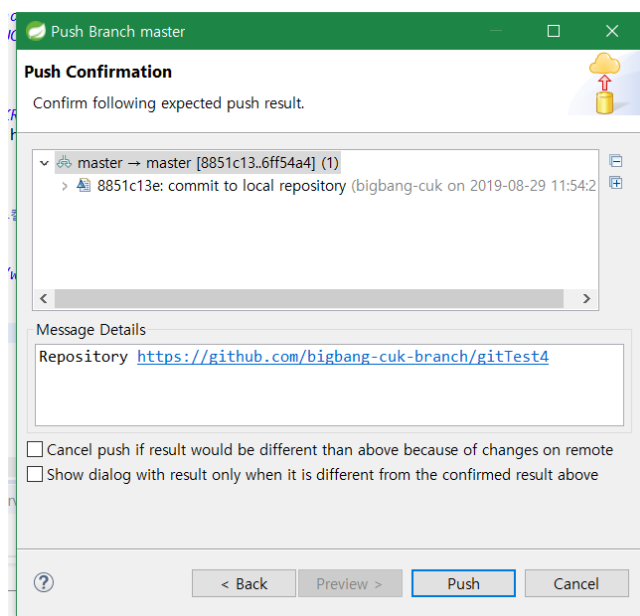


Push to branch in remote창이 뜨면 commit할 브랜치 정보와 commit message, 원격저장소 경로, 그리고 원격저장소에 merge될 브랜치 이름이 나옵니다.

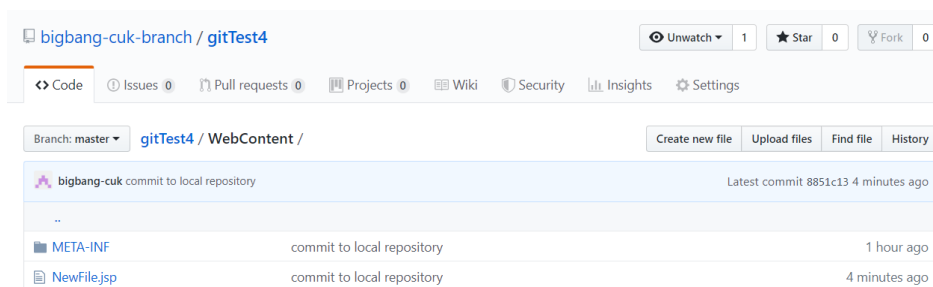
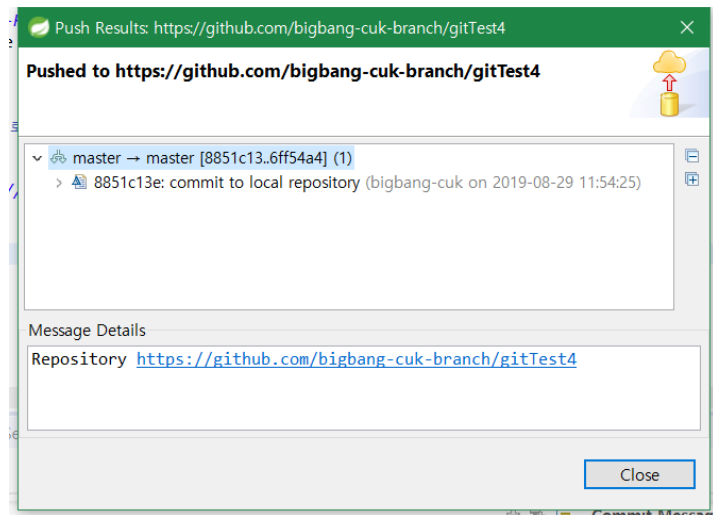
(경우는 적지만 로컬에서 A브랜치를 원격으로 push할 때 B브랜치로 merge하고 싶다면 Branch칸을 B로 바꾸면 됩니다.)



확인 후 [Preview]를 클릭하면 계정정보를 입력하고 계정정보 입력 후 Push Confirm에서 [Push]를 누르면 비로소 원격저장소로 push를 하고 merge가 일어나게 됩니다.



push후에 아래사진과 같이 아이콘에 특별한 표시가 없다면 성공적으로 merge된겁니다. github에서도 입력한 commit message로 push되었음을 확인할 수 있습니다.

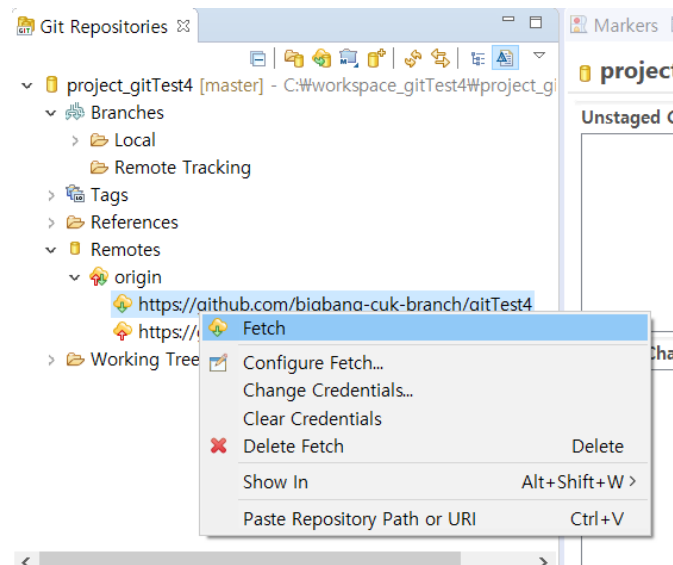


2-2-2. Fetch and Pull

로컬에서 작업한 소스를 원격저장소로 push하기 위해서는 사전에 동기화가 되어있어야 충돌이 일어나지 않기 때문에 작업 전, 혹은 push전에 동기화 처리를 해야 합니다. 또한 작업 전에 최신화된 소스로 작업을 하기 위해서도 미리 원격에서 소스를 받아올 필요가 있습니다.

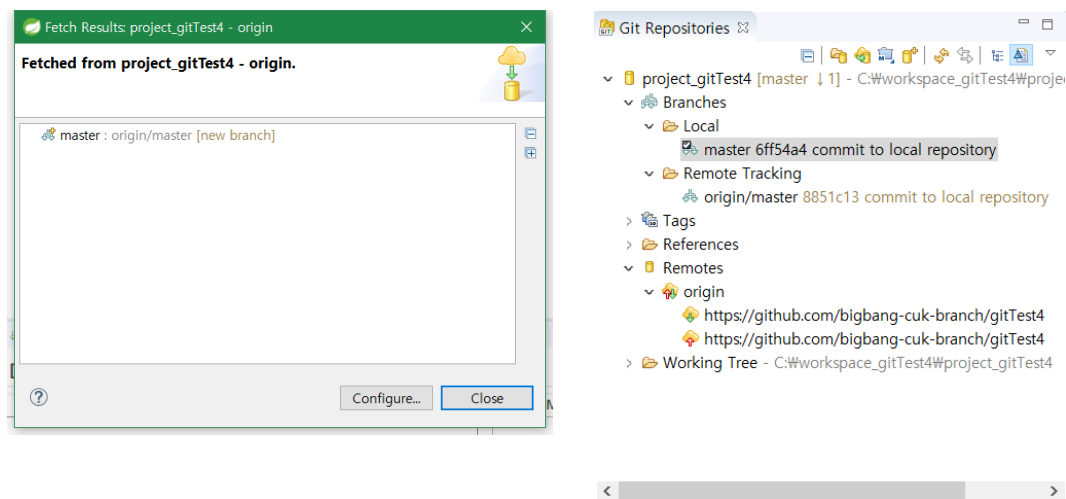
그래서 로컬저장소와 원격저장소를 동기화하는 방법을 알아보겠습니다.

Git Repository윈도우에서 로컬저장소WRemotesWorigin를 열면 로컬저장소의 해당 remote경로가 fetch와 push로 나뉘어져 있음을 볼 수 있습니다.

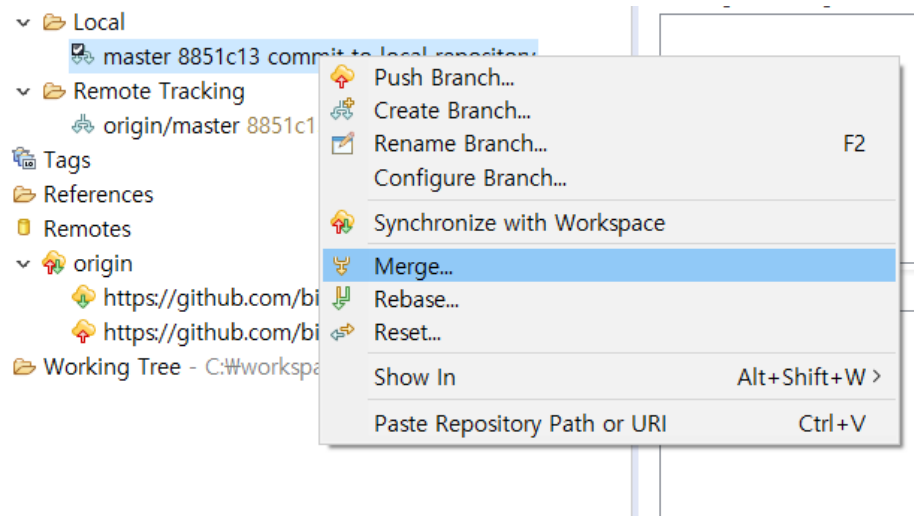


초록화살표 아이콘을 마우스 우클릭 -> [Fetch]를 클릭합니다.

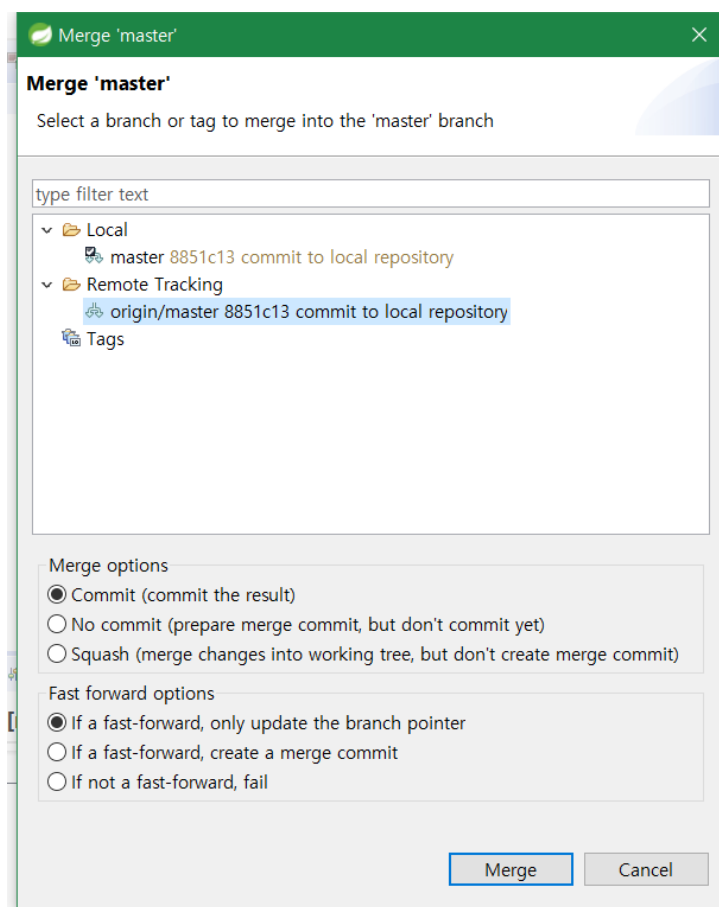
그러면 로컬에 바로 commit 시키지 않고 소스를 원격저장소로부터 가져오기만 합니다. Branches\Local 안에는 실제 로컬저장소의 브랜치가 포함되고 Branches\Remote Tracking에는 원격으로부터 fetch한 브랜치가 포함됩니다.




RemoteTracking에 있는 브랜치와 Local에 있는 브랜치를 merge시켜야합니다. 우선 동기화시킬 (로컬)브랜치를 Check out 상태로 둔 후에 해당 브랜치를 마우스 우클릭 -> [Merge]를 클릭합니다.



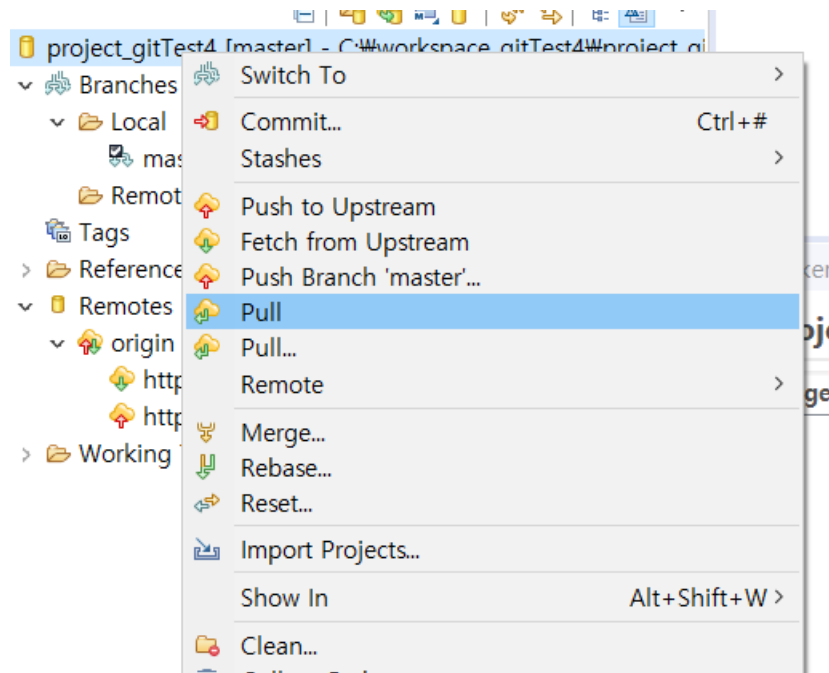
Merge창을 보면 Local에 Check out인 브랜치의 아이콘에 'V'표시가 되어있고 Remote에서 merge할 브랜치를 선택한 후 [Merge]를 클릭합니다.



성공적으로 Merge가 이루어지면 옆쪽의 '↓'표시와 숫자가 사라지면서 정상처리 된 것을 확인할 수 있습니다.

 project_gitTest4 [master]

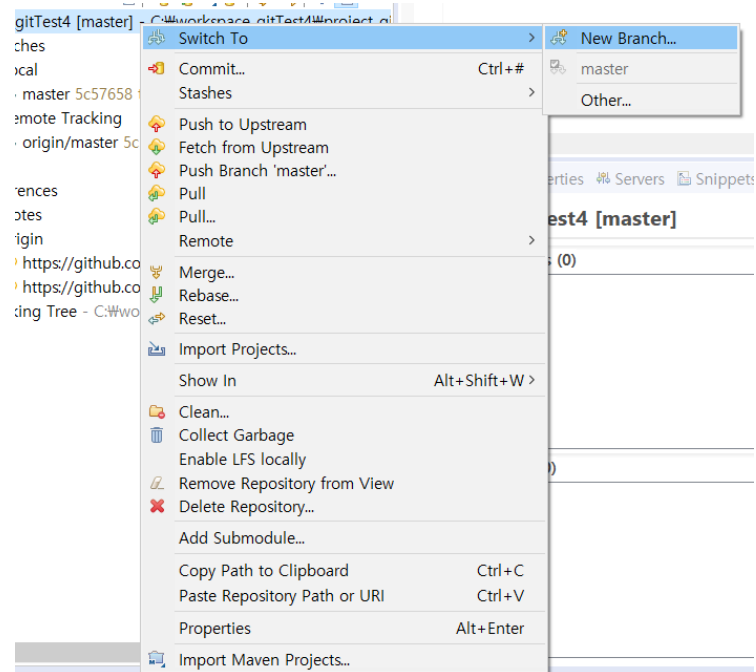
일련의 과정이 번거로워 한번에 처리하고 싶다면 프로젝트 마우스 우클릭 -> [Pull]을 클릭하면 fetch와 commit을 동시에 수행합니다.



2-2-3. branch and Merge

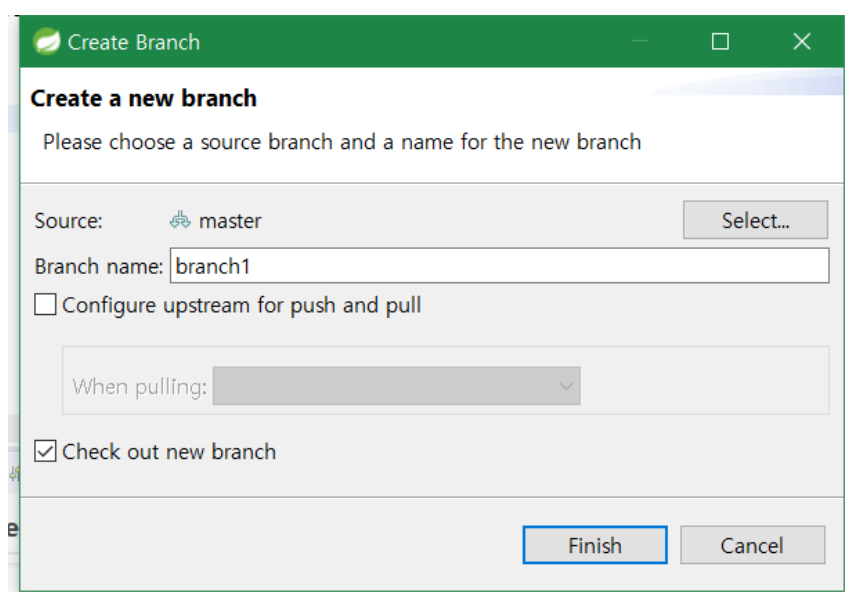
이번엔 로컬저장소에서 branch를 새로 만들고 두 브랜치를 merge하는 방법을 알아보겠습니다.

로컬저장소를 마우스 우클릭 -> [Switch To] -> [New Branch]를 클릭합니다.
(혹은 프로젝트 마우스 우클릭 -> [Team] -> [Switch To] -> [New Branch])

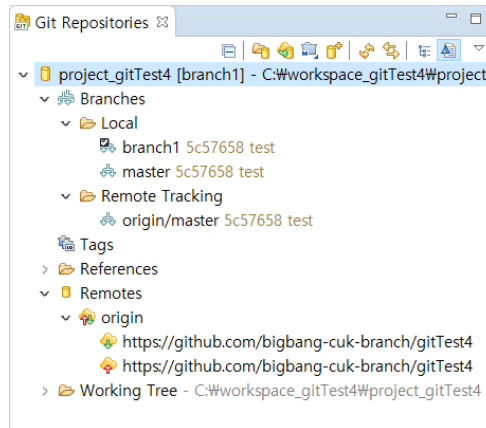


브랜치를 새로 만들기 전에 생성할 브랜치의 줄기 브랜치가 Check out 상태인지 확인해야 합니다. 작업중인 브랜치를 변경하려면 [Switch To]메뉴에서 Check out상태로 전환할 브랜치를 선택하면 됩니다.

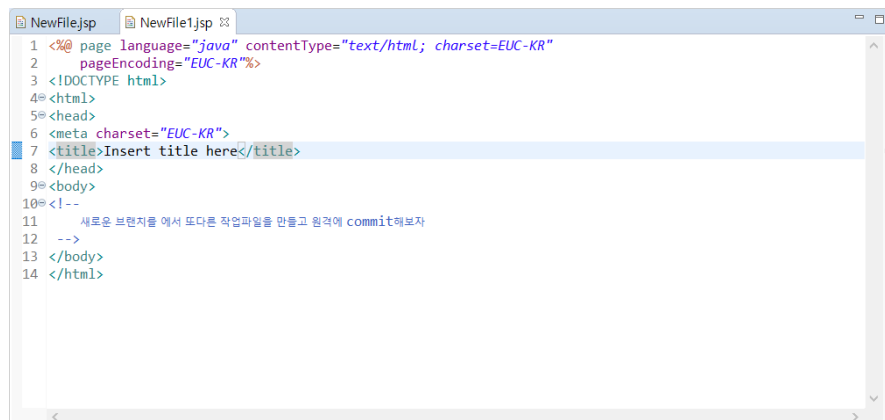
Create a new branch창이 뜨면 Source에 줄기브랜치(Check out상태의 브랜치)가 선택되어있음을 알 수 있고 Branch name에 생성할 브랜치의 이름을 입력한 후 [Finish]를 클릭합니다.



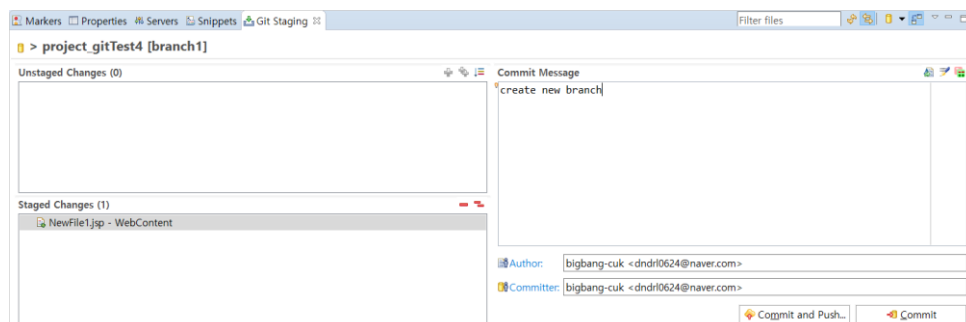
생성한 후 Git Repository 윈도우를 보면 Local에 새로운 브랜치가 생성된 것을 확인할 수 있습니다. 그리고 Remote Tracking에는 해당 브랜치가 존재하지 않으며 이는 로컬 저장소에만 생성이 되었고 원격 저장소에는 아직 push되지 않았음을 의미합니다.



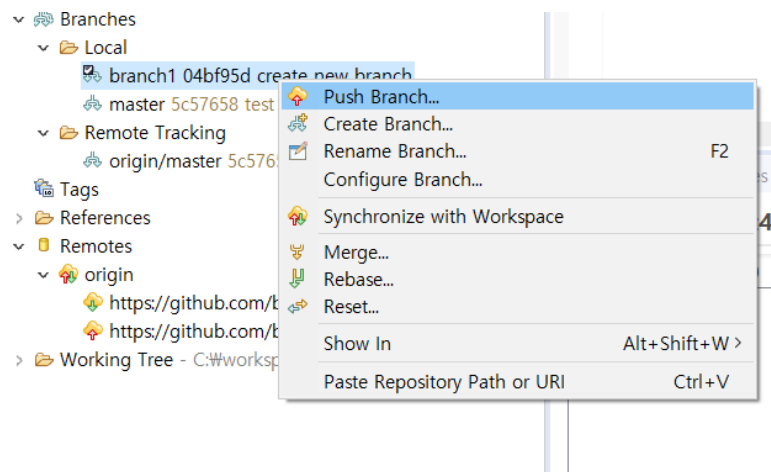
새로운 브랜치를 원격 저장소에 push를 해보기 전에 작업을 했다고 가정하고 새로운 파일을 만들어보겠습니다.



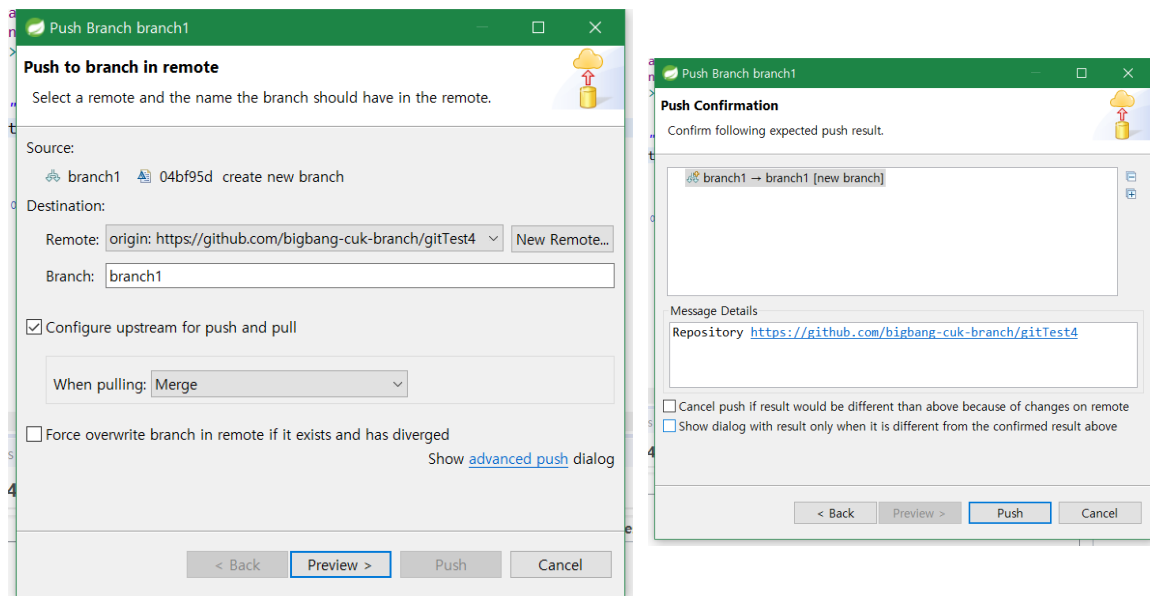
작업한 파일을 스테이지 영역에 올리고 로컬 저장소에 commit합니다.



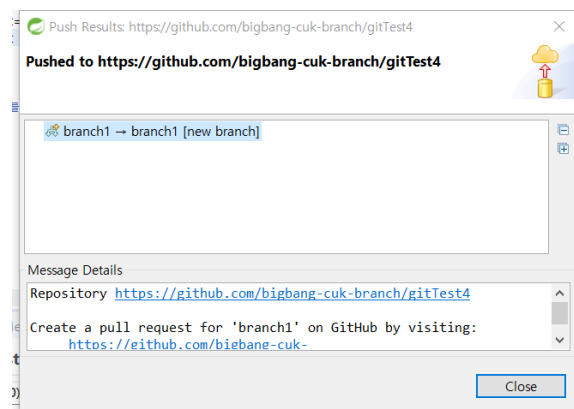
Local에서 push할 브랜치 마우스 우클릭 -> [Push Branch]를 클릭합니다.



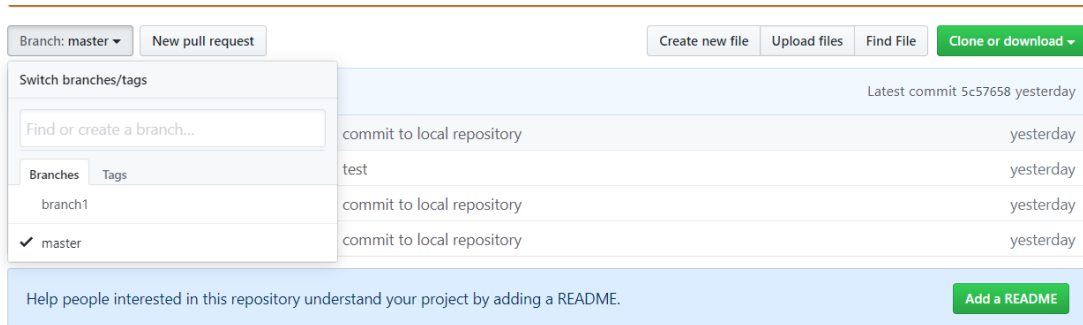
원격저장소에 저장될 브랜치이름을 입력하고 [Preview] -> [Finish] 클릭합니다.



아래사진처럼 표시되면 정상적으로 push된 것입니다.



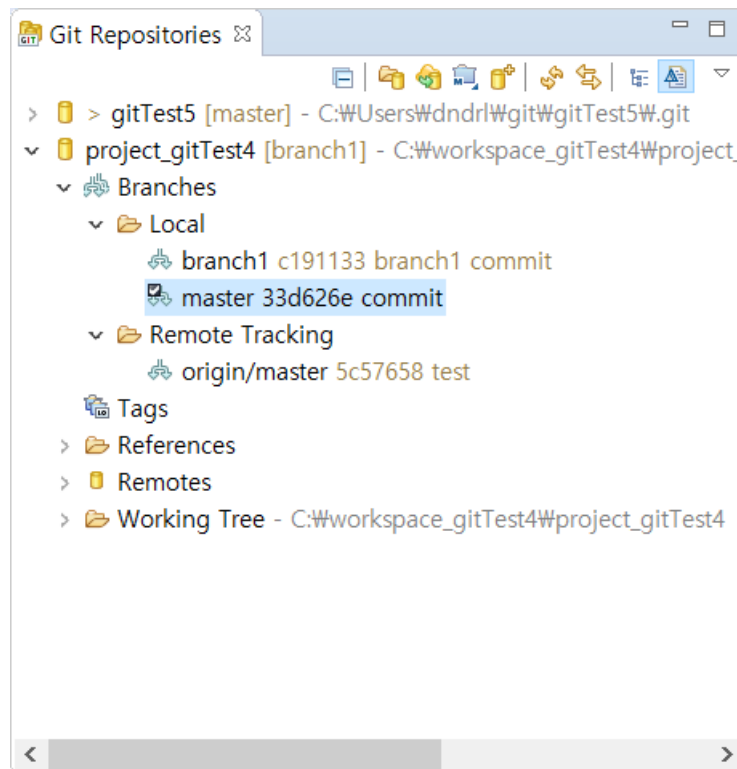
Github로 들어가면 원격저장소에 브랜치가 생성된 것을 볼 수 있습니다.



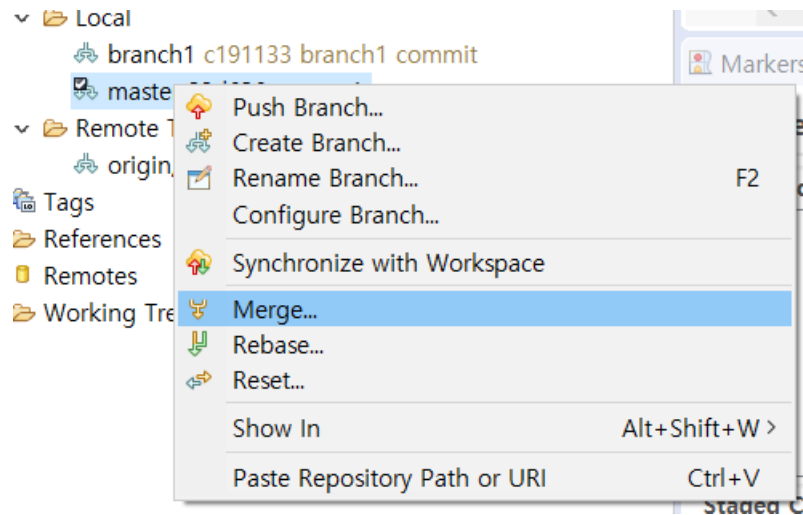
이젠 로컬저장소에서 두 브랜치를 Merge하는 방법을 알아보겠습니다.

merge하기 전에 A브랜치에 B브랜치를 병합시키는 경우라면 A브랜치에 Check out상태여야 합니다. 본 메뉴얼에서는 master브랜치에 branch1을 병합 하겠다고 가정하고 진행합니다.

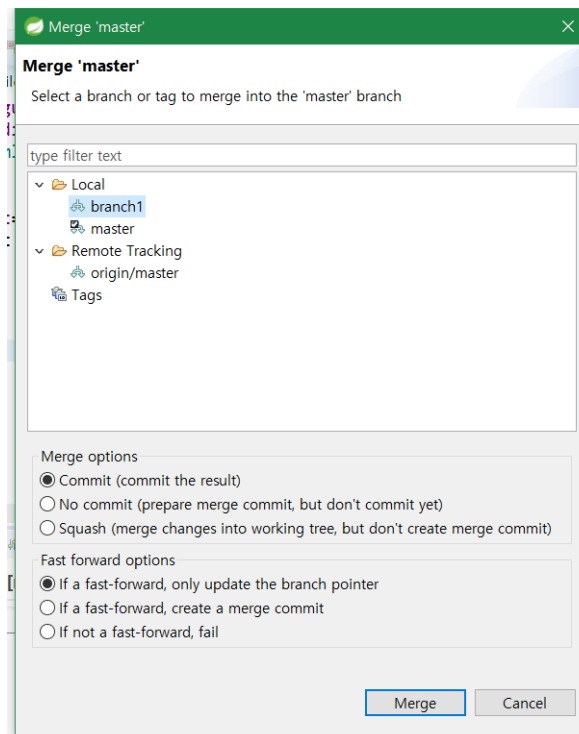
우선 master브랜치로 Check out합니다.



브랜치를 마우스 우클릭 [Merge]를 클릭합니다.



Merge창에서 병합할 브랜치(branch1)을 선택하고 [Merge]를 클릭합니다.



Git Repository윈도우에서 Check out인 브랜치가 아닌 다른 브랜치를 우클릭하여 Merge를 누르게 되면 Merge창이열리며 브랜치를 선택하는 단계가 생략되고 Check out브랜치와 Merge를 누른 브랜치가 바로 병합됩니다.

2-2-4. Merge Tool (Conflict)

Git을 사용하면서 제일 어려운 부분이 바로 브랜치끼리 merge하는 것입니다. 사용순서를 틀리거나 동기화하지 않거나 팀끼리 규칙을 맞추지 않으면 병합하는 과정에서 소스가 서로 상이하여 충돌(conflict)이 일어나게 됩니다.

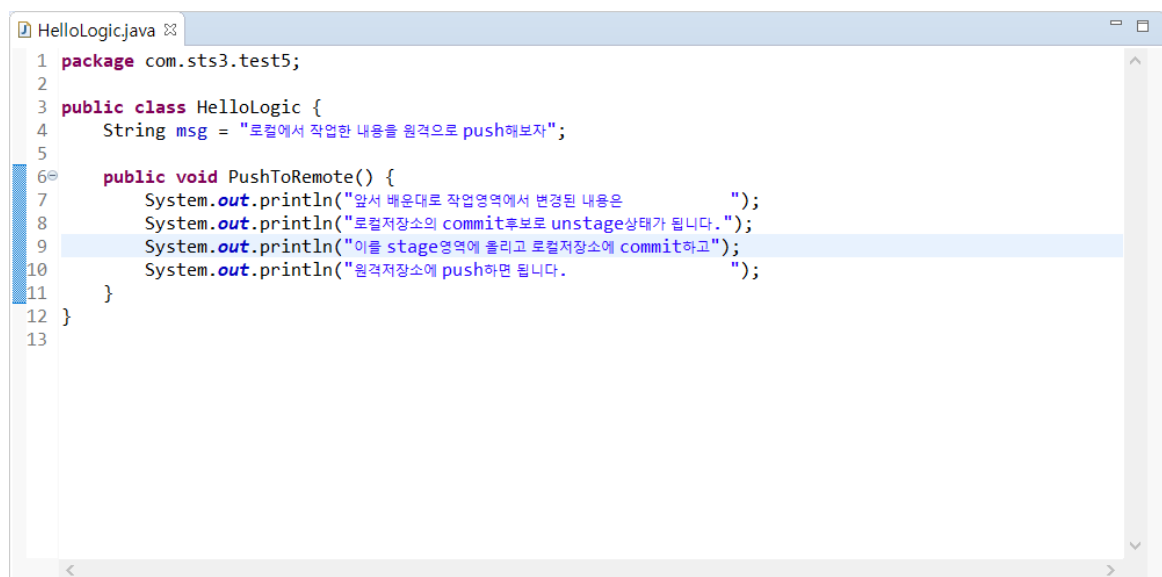
Egit에서는 Merge Tool이라는 기능을 제공합니다. 병합과정에서 충돌이 일어난 소스를 알려주고 해당소스를 수정하는데 도움을 줍니다. 그래서 이번에는 일부로 충돌상황을 만들어보고 어떻게 Merge Tool로 정상적으로 병합을 시키는지에 대해 알아보겠습니다.

시나리오를 먼저 설명해보면

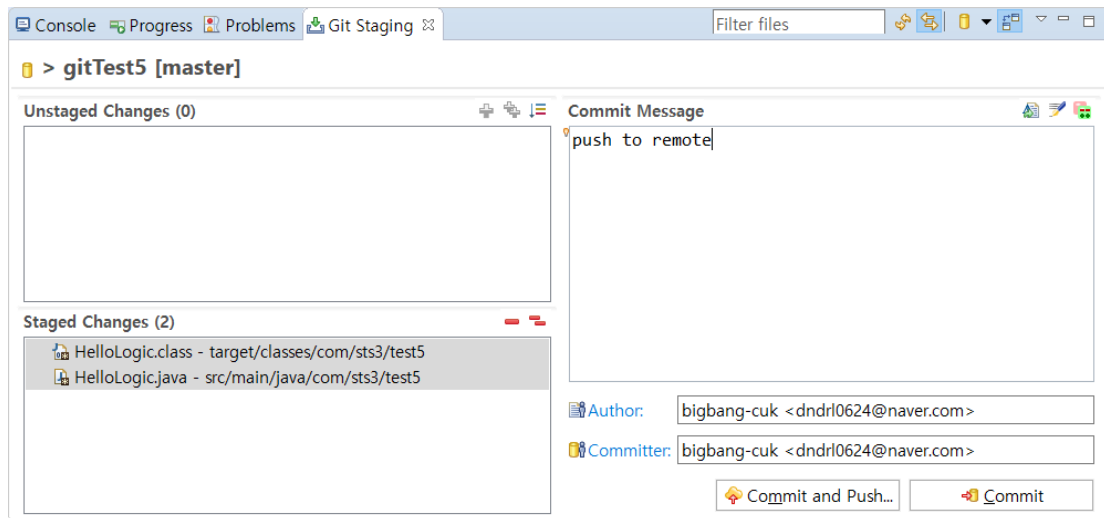
- "master 브랜치가 로컬과 원격이 동기화 되어있지 않고"
- "두 저장소에는 같은 이름의 파일이 존재하며"
- "두 파일은 서로 다른 코드가 작성되어있다"

로 가정해 보았습니다. 물론 로컬과 원격간의 merge가 아니라 로컬 내에서의 다른 브랜치간 병합도 아래에서 보여주는 Merge Tool사용법은 동일합니다.

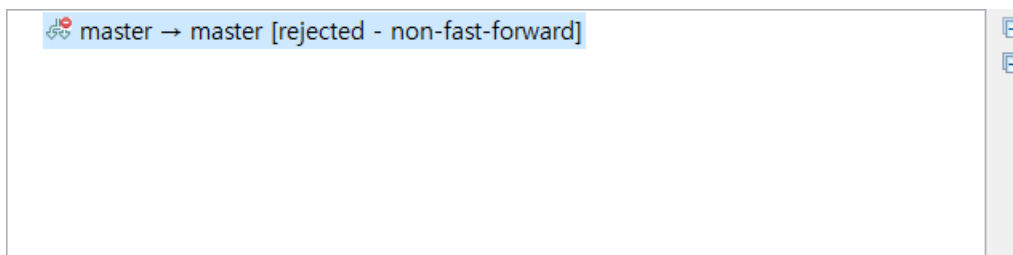
우선 HelloLogic.java를 만들어 원격에 push해보겠습니다.



```
1 package com.sts3.test5;
2
3 public class HelloLogic {
4     String msg = "로컬에서 작업한 내용을 원격으로 push해보자";
5
6     public void PushToRemote() {
7         System.out.println("앞서 배운대로 작업영역에서 변경된 내용은 ");
8         System.out.println("로컬저장소의 commit후보로 unstage상태가 됩니다.");
9         System.out.println("이를 stage영역에 올리고 로컬저장소에 commit하고");
10        System.out.println("원격저장소에 push하면 됩니다.");
11    }
12 }
13
```



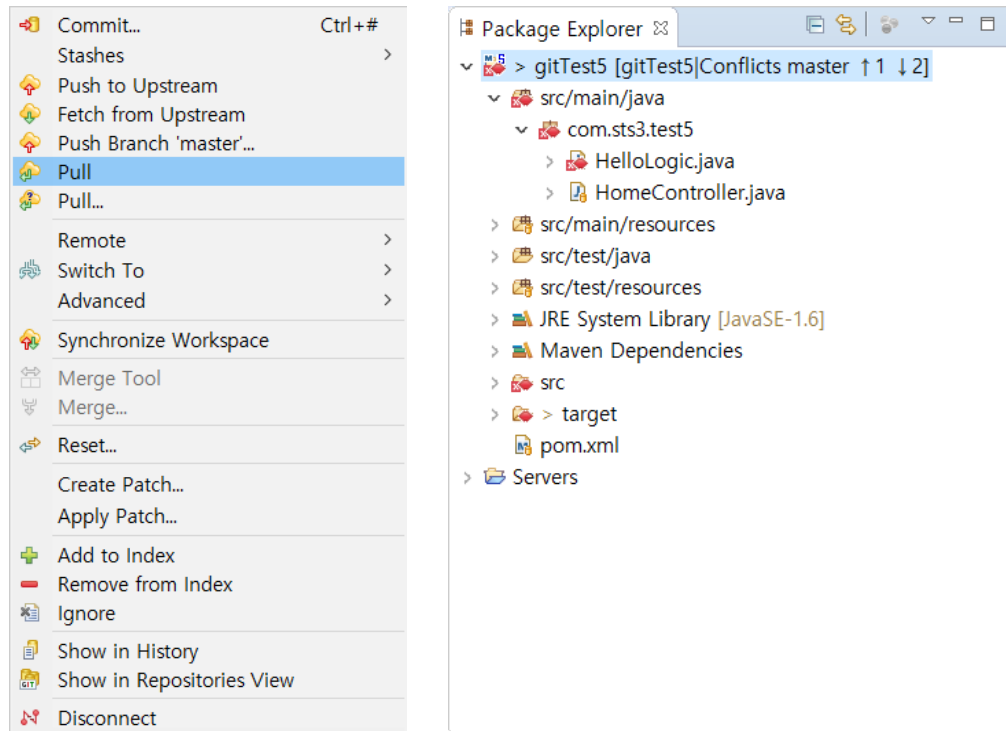
하지만 (시나리오대로) 실패했습니다.



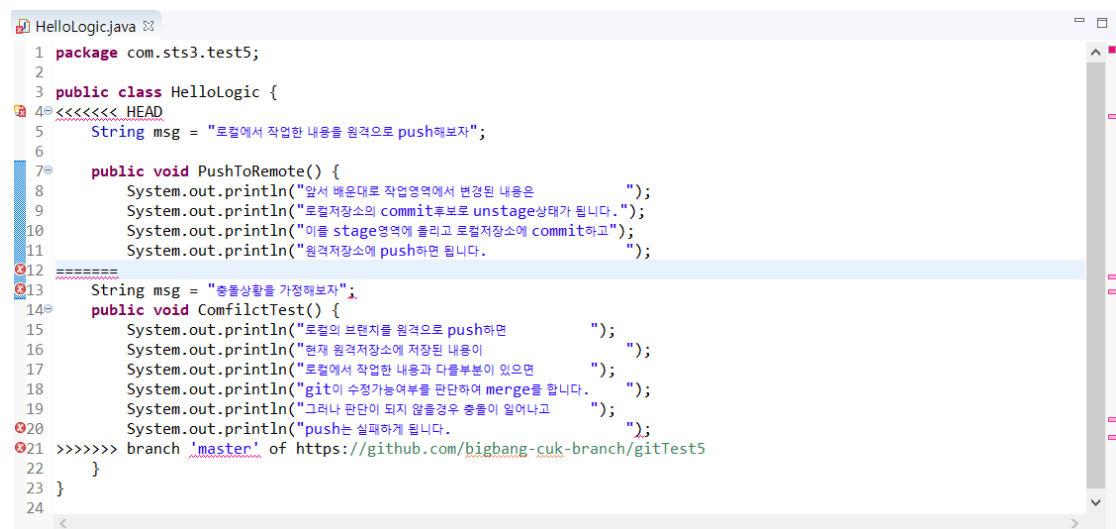
github에서 보니 해당 브랜치의 동일파일에서 소스가 달라 commit과정에서 (계획대로)충돌이 일어난 것입니다.



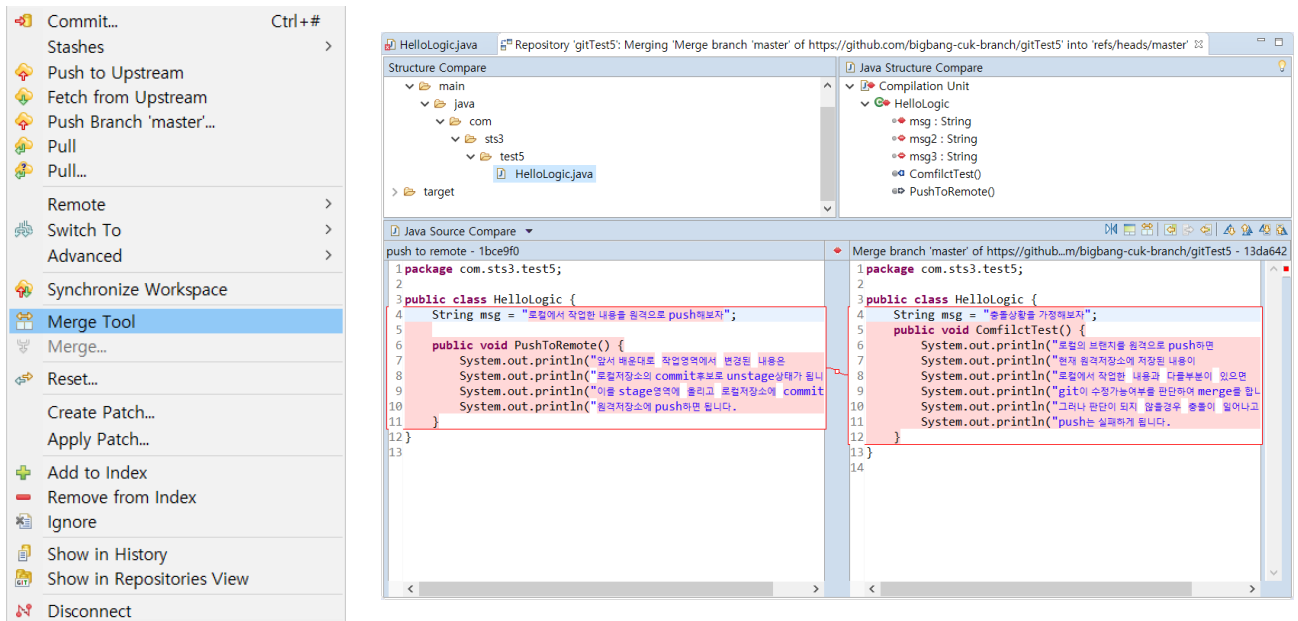
이럴 때는 우선 원격에서 소스를 가져와 로컬과 동기화하여 다시 원격에 push해야 합니다. 그래서 pull을 통해 소스를 가져와 동기화를 시키면 역시나 충돌이 일어나고 충돌이 일어난 파일의 아이콘에 붉은 점이 생겨납니다.



로컬에서 병합된 코드를 보면 충돌이 일어난 부분에 tag로 표시가 되어 있습니다. HEAD 바로 아래가 로컬의 master브랜치의 코드이고 단락구분 아래가 원격의 master브랜치의 코드입니다.






프로젝트(혹은 해당파일)을 마우스 우클릭 -> [Team] -> [Merge Tool]을 클릭하면 아래와 같은 윈도우가 뜹니다. 좌상단에는 충돌이 일어난 파일의 경로를 보여주고 좌하단에는 병합을 '하는' 브랜치의 코드, 우하단에는 병합이 '되는' 브랜치의 코드가 표시되며 충돌이 일어난 부분은 붉은 바탕과 연결고리로 표시해줍니다.

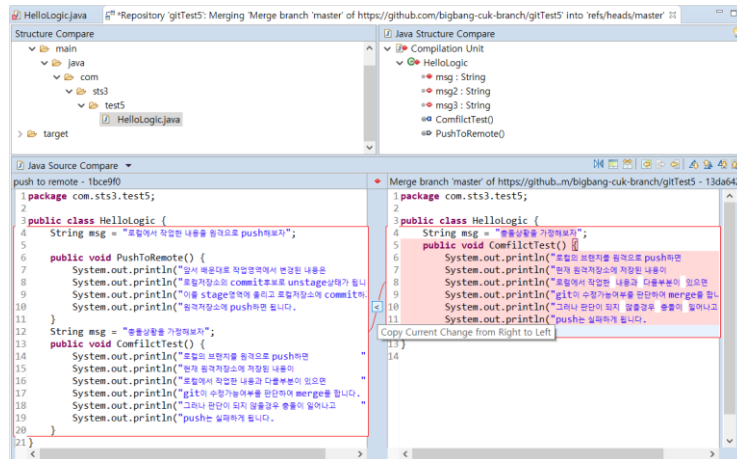


코드의 우상단을 보면 다양한 아이콘을 볼 수 있습니다.

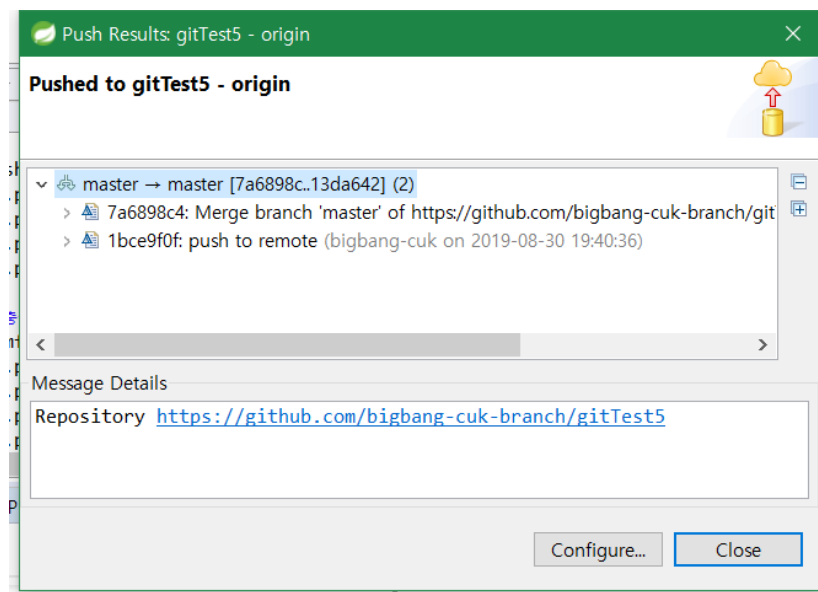
- 좌/우 view 전환
- 두 변경의 공통 원본 표시
- Three way compare : 공통 원본과 함께 두 변경 비교
- Conflict가 발생하지 않는 오른쪽의 내용을 모두 왼쪽에 반영
- 왼쪽 변경을 오른쪽에 반영
- 오른쪽 변경을 왼쪽에 반영
- 다음 차이점 위치
- 이전 차이점 위치
- 다음 변경 위치
- 이전 변경 위치

아래 그림은  을 클릭하여 오른쪽 변경을 왼쪽에 반영한 모습입니다.

 은 모든 변경사항을 반영하고, 부분만 반영하고 싶다면 두 연결고리 사이에  를 클릭하면 해당소스만 반영합니다.



왼쪽에서 소스를 전부 수정한 후 저장을 하면 해당소스는 다시 로컬의 '커밋 될 예정인 파일'이 되므로 unstaged영역에 포함됩니다. 해당파일을 스테이지 영역에 올리고 commit합니다. 그러면 현재까지 상황이 원격에서 가져온 소스를 로컬에 merge하는데 충돌부분을 수정하여 로컬에 commit까지 했습니다. 이제 로컬의 브랜치를 원격으로 다시 push만 하면 됩니다.



아까와는 다르게 충돌 없이 원격저장소에 성공적으로 push된 것을 확인할 수 있습니다.

참고자료

- ECLIPSE USER GUIDE

https://wiki.eclipse.org/EGit/User_Guide#Merging

- SYS4U OPEN WIKI

<http://wiki.sys4u.co.kr/display/SOWIKI/Git>

- TISTORY 코딩팩토리

<https://coding-factory.tistory.com/category/%EA%B8%B0%ED%83%80/Git>

- TISTORY 소소한 저장소

<https://gasaesososo.tistory.com/category/Git>

- TISTORY 개발 이야기

<https://2clipse-story.tistory.com/4?category=241530>