

Git

1. 버전관리 시스템이란

가. 버전관리 시스템 유형

- (1) 로컬버전 관리시스템 : RCS
- (2) 중앙집중식 버전관리 시스템 : CVS, SVN
중앙서버에 문제가 발행하면 버전관리를 할 수 없음
- (3) 분산 버전관리 시스템 : **git**, 머큐리얼(Mecrurial), 바질(Bazzar), 다크스(Darcs)
저장소 전부를 복제하여 로컬 저장소에 관리. 중앙 서버에 문제가 발생하더라도 각 로컬 저장소에서 복구할 수 있다.

나. git 원격저장소

(1) 깃허브(github)

- (2) 깃랩
- (3) 비트버킷

다. git 클라이언트

- (1) cli git : **git**
- (2) eslipse git plugin : **egit**
- (3) gui git : gittable, **SourceTree**, GitHub Desktop ...
git gui(graphical User interface)의 약자로 초보자가 명령이나 작업을 이해하기 쉽도록 프로젝트 히스토리를 시각화하여 도와주는 도구

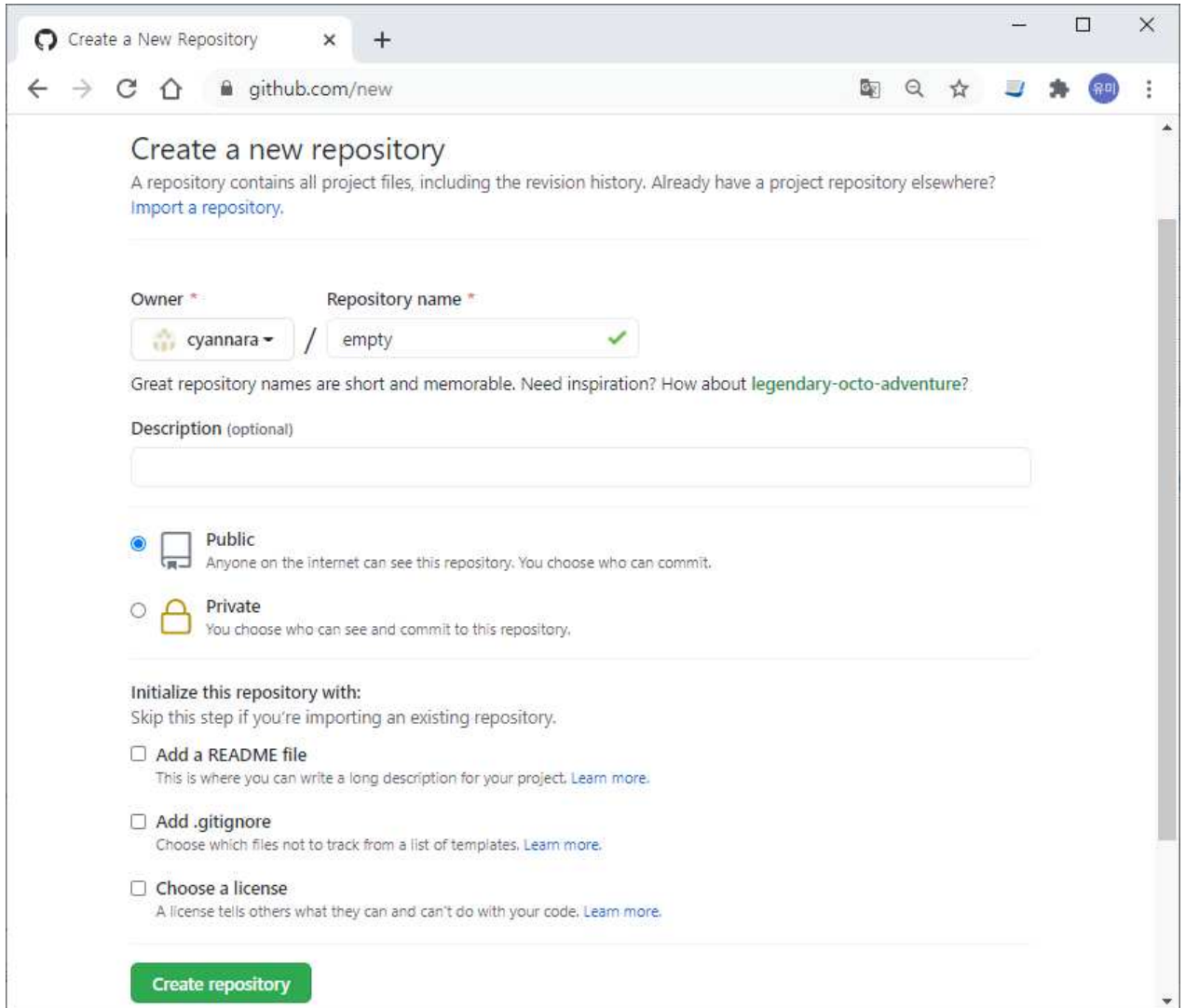
참고사이트 : <https://git-scm.com/book/ko/v2>

라. git 상태

- (1) Modified : 수정한 파일을 아직 로컬 데이터베이스에 Commit 하지 않음
- (2) Staged : 수정 파일을 곧 Commit 할 것
- (3) Committed : 데이터가 로컬 데이터베이스에 안전하게 저장

2. 깃허브(github)

가. repository 생성





Create a New Repository

github.com/new

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)


Owner ^{*} / Repository name ^{*}

 cyannara / empty 

Great repository names are short and memorable. Need inspiration? How about [legendary-octo-adventure?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

[Create repository](#)

repository name : 새로 생성할 원격 저장소 이름. git 프로젝트 폴더명과 동일하게 지정

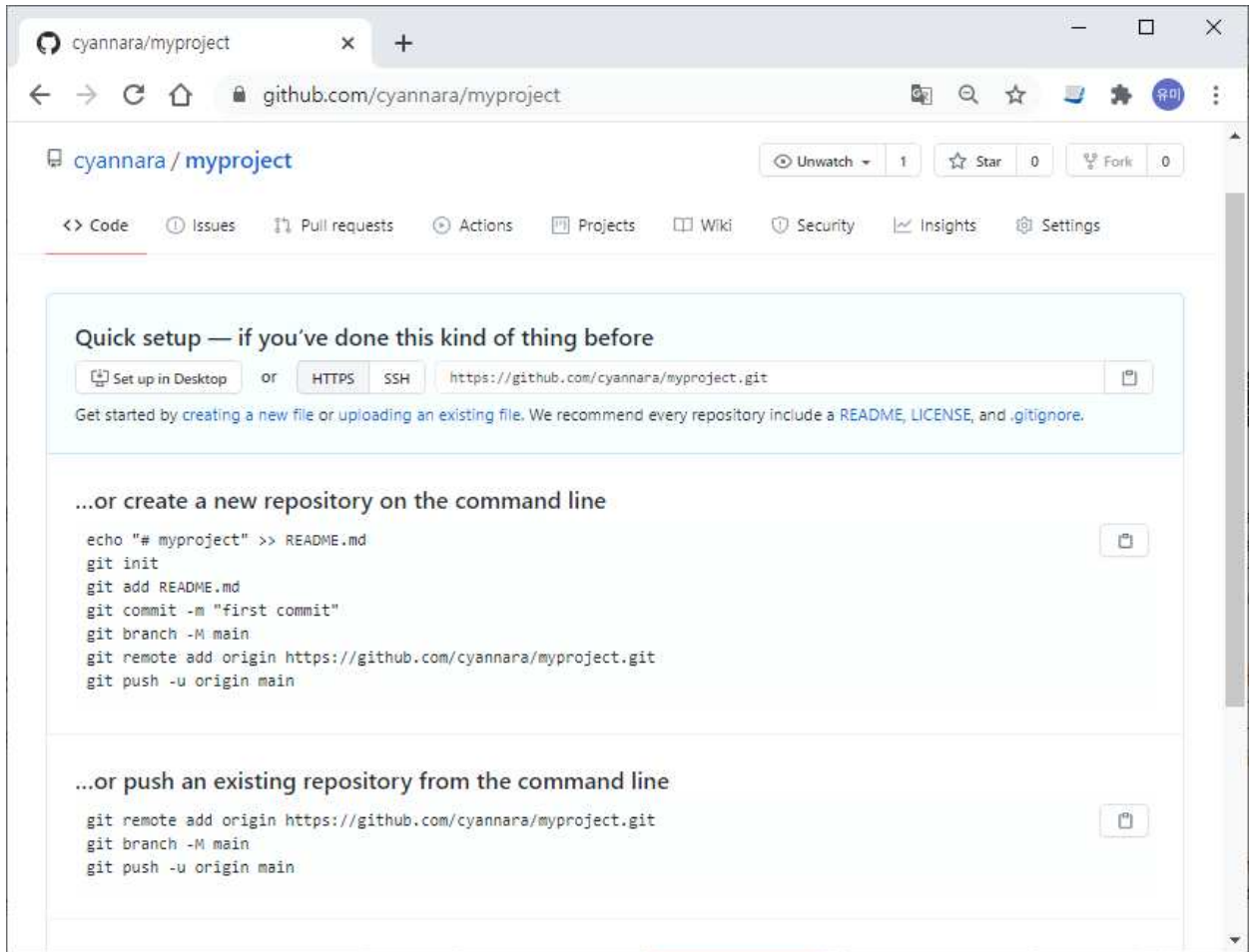
Description : 원격저장소의 역할을 설명(옵션)

public/private : 원격저장소의 공개 여부를 선택

Initialize this repository with : 3개 모두 체크하기를 권장.

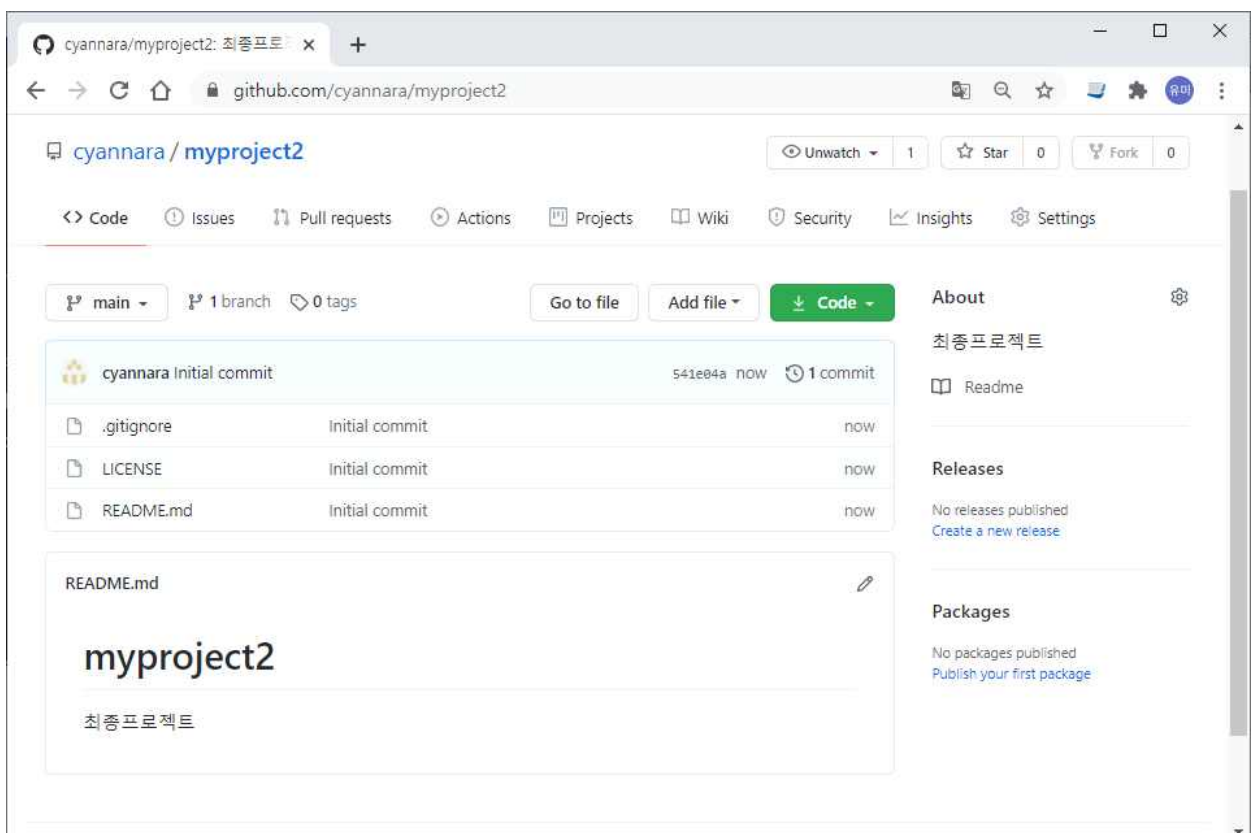
Add a README file을 체크하면 저장소 이름과 Description 항목의 내용을 담은 README.md 파일이 생성된다. 원격저장소에 하나라도 파일이 존재해야 로컬저장소에 복사(clone)할 수 있다.

- (1) Initialize this repository with 를 체크한 안하고 생성한 경우



- (2) Initialize this repository with 를 모두 체크하고 생성한 경우

We recommend every repository include a README, LICENSE, and .gitignore.



3. git

가. git

(1) git 설치

설치파일 : Git-2.31.0-64-bit.exe

(2) git의 상태관리

작업디렉토리 -----> 스테이지 -----> 로컬저장소 -----> 원격저장소(repository)
 git add git commit git push

local에 add, commit하고 서버에는 push한다.

git에는 로컬에 staging area라는 중간 단계를 두고 있어서 여기에 저장하는 것을 commit이라고 한다.
 내가 작업하고 있는 위치는 working directory, 내 컴퓨터의 git이 관리하는 위치는 staging area, 원격서버의 github가 관리하는 위치는 repository이다.

나. Git 명령어

(1) git 저장소 만들기

(가) 기존 디렉토리를 저장소로 만들기

- 디렉토리 git 초기화

```
git init
```

이 명령은 .git이라는 하위 디렉토리를 만든다.

- 원격저장소를 연결(추가)

```
git remote add origin https://github.com/{owner}/{repository}
```

- 추가된 원격저장소 확인

```
git remote -v
```

- 원격저장소 삭제

```
git remote remove {원격저장소별칭}
```

(나) 원격저장소를 복사

다른 프로젝트에 참여하거나(Contribute) Git 저장소를 복사하고 싶을 때 사용.

서버에 있는 모든 데이터와 히스토리를 전부 받아온다. 실제로 서버의 디스크가 망가져도 클라이언트 저장소 중에서 아무거나 하나 가져다가 복구할 수 있다.

```
git clone {원격저장소URL}
```

원격저장소의 이름과 같은 폴더를 생성하고 .git 폴더를 만든다. 그리고 저장소의 데이터를 모두 가져와서 자동으로 가장 최신 버전을 Checkout해 놓는다.

- 원격저장소에 파일이 하나도 없는 경우(empty) clone이 안됨

```
D:\Wgitclone>git clone https://github.com/cyannara/empty.git
Cloning into 'empty'...
warning: You appear to have cloned an empty repository.
```

- 다른 폴더명으로 clone

```
git clone {원격저장소URL} 폴더명
```

다른 폴더명으로 생성. 나머지는 동일함.

(2) config 설정

(가) 설정정보 변경

- 글로벌 설정정보 변경

```
git config --global user.email "이메일"
git config --global user.name "이름"
```

- 로컬저장소 설정정보 변경

```
git config user.email "이메일"
git config user.name "이름"
```

(나) 설정정보 삭제

- 글로벌 설정정보 삭제

```
git config --unset --global user.name
```

- 로컬 설정정보 변경

```
git config --unset user.name
```

(다) config 정보 확인

```
git config --list
```

(3) 스테이지에 추가

git add는 파일을 새로 추적할 때도 사용하고 수정한 파일을 Staged 상태로 만들 때도 사용

- git add 명령어로 직접 Tracked 상태로 만들어 파일을 추적하도록 만든다

```
git add README.md
git add *
```

git add를 하기 전에 어떤 파일을 만들었다면 git status를 실행해 보면 파일중에 Git은 Untracked 파일

을 보여줍니다. git add 를 한 후 git status 명령을 다시 실행하면 README 파일이 Tracked 상태이면서 Staged 상태라는 것을 확인할 수 있다.

- 파일 상태 확인

파일 상태와 현재 작업중인 branch를 알려준다.

파일은 크게 Tracked(관리대상임)와 Untracked(관리대상이 아님)로 나누어진다.

Tracked 파일은 이미 스냅샷에 포함돼 있던 파일이고, Tracked 파일은 또 Unmodified(수정하지 않음)와 Modified(수정함) 그리고 Staged(커밋하면 저장소에 기록되는) 상태 중 하나이다.

```
git status
```

```
D:\Wgitwork>git status
```

```
On branch master
```

```
Your branch is up to date with 'origin/master'.
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
new file: LICENSE
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git restore <file>..." to discard changes in working directory)
```

```
modified: README.md
```

(4) 변경사항 커밋(commit)

스테이지 변경 내용을 확정

```
git commit
```

```
git commit -m "first commit"
```

Unstaged 상태의 파일과 git add 명령으로 추가하지 않은 파일은 커밋하지 않는다.

커밋하기 전에 git status 명령으로 모든 것이 Staged 상태인지 확인할 수 있다.

커밋은 변경사항을 내 컴퓨터에 저장한다는 의미입니다. 위 명령어를 실행하면 이제 작업흐름상에 변경된 파일이 HEAD에 반영될 것입니다. 하지만, 원격 저장소에는 아직 반영이 되지 않는 상태입니다.

- 파일 수정 시 자동으로 스테이지에 추가

```
git commit -am "first commit"
```

또한 git add 명령을 실행했을 지라도 git add 명령을 실행한 후에 또 파일을 수정하면 파일의 상태는 Unstaged 상태로 나옵니다. 그렇기 때문에 파일을 수정한 후에는 git add 명령을 다시 실행해서 최신 버전을 Staged 상태로 만들어야 한다.

-a 옵션을 추가하면 Git은 Tracked 상태의 파일을 자동으로 Staging Area에 넣어준다.

- 커밋이력 확인

```
git log
```

```
D:\Wgitwork>git commit -m "first commit"
[master (root-commit) 82e75af] first commit
1 file changed, 1 insertion(+)
create mode 100644 readme.md

D:\Wgitwork>git log
commit 82e75af958697ec51cd125b6e1f597446fbc3c6e (HEAD -> master)
Author: 홍길동 <cyannara@naver.com>
Date: Thu Mar 25 13:45:15 2021 +0900

first commit
```

(5) push

push는 마지막으로 커밋한 사항을 git repository 에 올린다. push가 안되면 원격 서버에 변경사항이 저장되지 않는다.

```
git push -u {remote저장소} {branch}
git push -u origin master
```

옵션 -u는 원격저장소로부터 업데이트를 받은 후 push를 한다는 의미이므로 습관적으로 -u 사용을 권장한다. 이유는 Clone한 사람이 여러 명 있을 때, 다른 사람이 Push한 후에 Push하려고 하면 Push할 수 없기 때문에 먼저 다른 사람이 작업한 것을 가져와서 머지한 후에 Push할 수 있다.

```
D:\Wgitwork>git push -u origin master
fatal: helper error (-1): at least one authentication mode must be specified
git: 'credential-os' is not a git command. See 'git --help'.
```

```
The most similar command is
credential-store
Password for 'https://cyannara@github.com':
git: 'credential-os' is not a git command. See 'git --help'.
```

```
The most similar command is
credential-store
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 241 bytes | 80.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/cyannara/myproject.git
* [new branch] master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
git: 'credential-os' is not a git command. See 'git --help'.
```

```
The most similar command is
credential-store
Everything up-to-date
Branch 'master' set up to track remote branch 'master' from 'origin'.
```


여러분이 뭔가 변경될 때마다 해야 할 일은 간단히 다음과 같을 것입니다.

pull -> commit -> push

- 자격증명이 다른 경우 에러 발생

```
remote: Permission to cyannara/myproject.git denied to cyanClass.
```

```
fatal: unable to access 'https://github.com/cyannara/myproject.git/': The requested URL returned error: 403
```

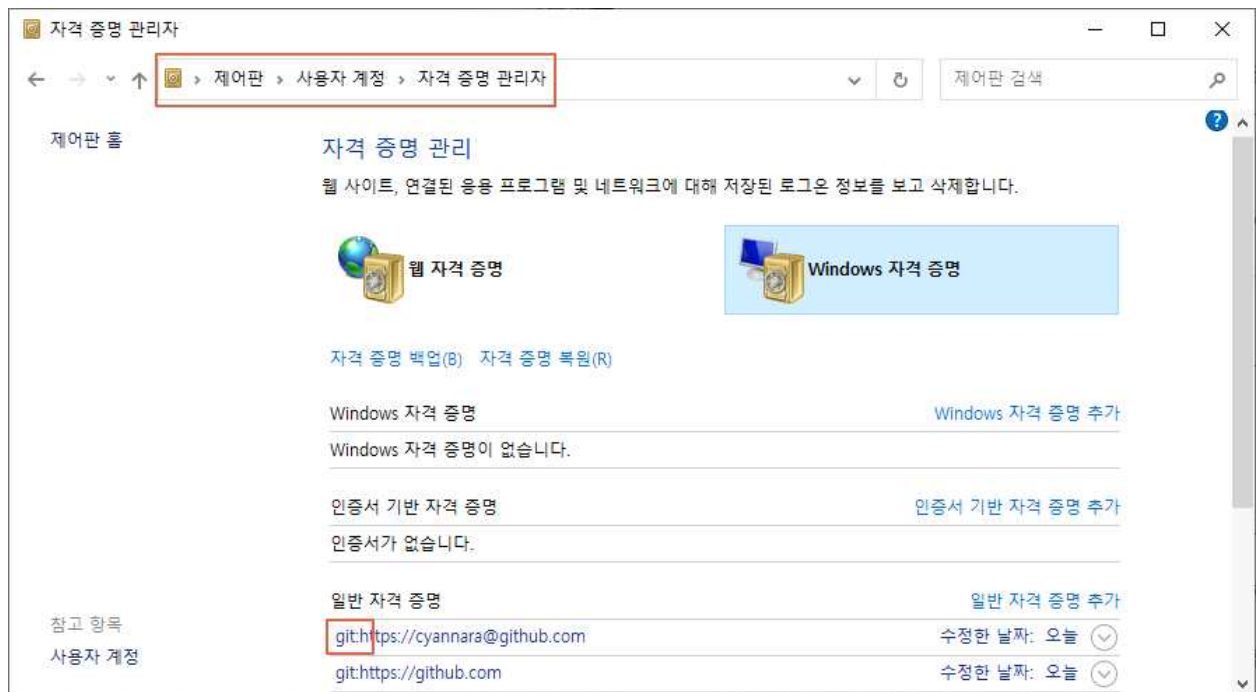
깃허브에 푸쉬할 때 권한이 없다고 에러가 발생하는 경우에는 다른 유저로 인증정보가 등록되어 있을 가능성이 있다. 그런 경우에는 예전 증명서를 삭제하고 다시 시도하거나 자격증명을 추가한다.

- 자격증명 추가

```
git remote set-url origin https://{user.name}@github.com/{owner}/{repository}
```

- 자격증명 삭제

제어판 -> 사용자 계정 -> 자격증명 관리자 에서 일반 자격증명 확인



- 아이디와 비밀번호 저장하기

git 콘솔에서 다음과 같이 한번만 입력해 놓으면 push나 pull 등 서버에 접속하면서 팝업으로 뜨는 아이디 비밀번호창을 띄우지 않도록 아이디 및 비번을 기억하도록 할 수 있다.

```
git config credential.helper store
```

(6) 변경내용 확인

```
git add LICENSE
git diff --staged
```

```
D:\Wgitwork>git add license.md
D:\Wgitwork>git diff --staged
diff --git a/license.md b/license.md
new file mode 100644
index 0000000..5c9bf33
--- /dev/null
+++ b/license.md
@@ -0,0 +1 @@
+라이선스 정보입니다.
\ No newline at end of file
```

(7) branch

브랜치란 독립적으로 작업을 진행하려는 용도로 만들어진 개념. 각 브랜치는 다른 브랜치에 영향을 주지 않아서 동시에 안전하게 동시 개발이 가능하다. 각 작업이 끝나면 브랜치를 병합하여 최종 릴리스 버전을 만들 수 있다.

- branch 생성

```
git branch user1_branch
```

- branch 리스트 확인

```
git branch
```

```
* master
user1_branch
```

- 작업디렉토리를 branch 변경

```
git checkout {branch이름}
```

- branch 삭제

```
git branch -d {branch이름}
```

- 원격저장소 branch 삭제

```
git push --delete origin {branch이름}
```

- branch에 push

```
git push origin {branch이름}
```

- branch를 master 병합

```
git checkout master
git merge {branch이름}
```

(8) 충돌(conflict)

브랜치를 분리해서 개발하면 개발도중에는 소스의 충돌이 발생하지는 않지만 작업을 끝내고 병합할 때 충돌이 발생할 수 있다.

다. git 실습

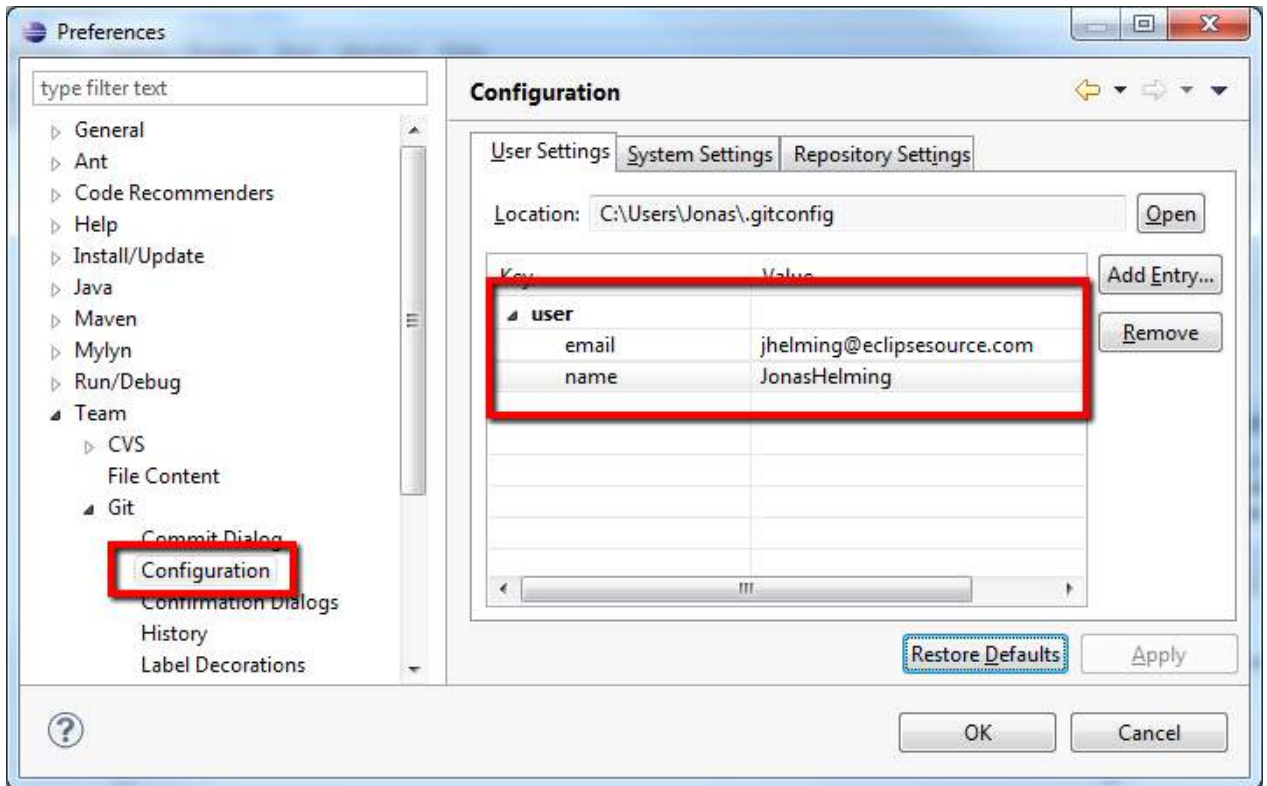
	<pre>//clone D:\W>mkdir gitclone D:\W>cd gitclone</pre>
<pre>//로컬저장소 생성 D:\W>mkdir gitwork D:\W>cd gitwork D:\Wgitwork>git init D:\Wgitwork>dir /a //git 설정정보 변경 D:\Wgitwork>git config --global user.email "이메일" D:\Wgitwork>git config --global user.name "이름" //파일을 생성한 후 스테이지에 추가(add)하고 commit D:\Wgitwork>notepad README.md D:\Wgitwork>git add README.md D:\Wgitwork>git commit -m "first commit" D:\Wgitwork>git log //로컬저장소와 원격저장소를 연결 D:\Wgitwork>git remote add origin https://github.com/{owner}/{repository} D:\Wgitwork>git remote -v //push D:\Wgitwork>git push -u origin master</pre>	
	<pre>D:\Wgitclone>git clone https://github.com/{owner}/{repository}</pre>
<pre>//파일 추가 D:\Wgitwork>notepad .gitignore D:\Wgitwork>git status D:\Wgitwork>git add * D:\Wgitwork>git commit -m "work gitignore 추가" D:\Wgitwork>git status D:\Wgitwork>git push</pre>	
	<pre>//최신소스 반영(받기) D:\Wgitclone\Wmyproject>git pull D:\Wgitclone\Wmyproject>dir //커밋이력 확인 D:\Wgitclone\Wmyproject>git log</pre>
<pre>//브랜치 생성 D:\Wgitwork>git branch user1_branch D:\Wgitwork>git branch D:\Wgitwork>git checkout user1_branch //브랜치에 파일 추가 D:\Wgitwork>notepad .gitignore D:\Wgitwork>git commit -m "branch " .gitignore D:\Wgitwork>git push origin user1_branch D:\Wgitwork>notepad user1.txt D:\Wgitwork>git add user1.txt D:\Wgitwork>git commit -m "사용자파일 추가" user1.txt D:\Wgitwork>git push origin user1_branch</pre>	
	<pre>//브랜치 내용 반영안됨</pre>

	D:\Wgitclone\Wmyproject> git pull D:\Wgitclone\Wmyproject> dir
//브랜치를 마스터에 병합 D:\Wgitwork> git checkout master D:\Wgitwork> git merge user1_branch	
	//브랜치 내용 반영됨 D:\Wgitclone\Wmyproject> git pull D:\Wgitclone\Wmyproject> dir

4. egit

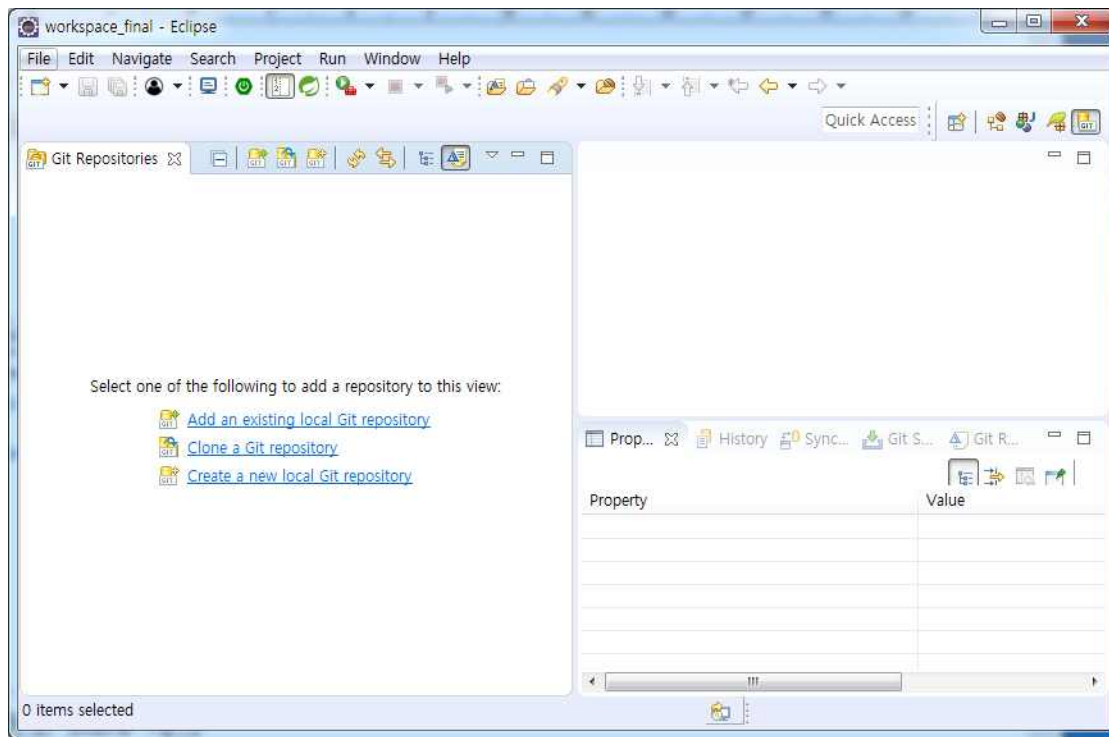
가. Egit 설정

Every commit in EGit will include the user's name and his email-address. These attributes can be set in the Preferences-window Window => Preferences. Navigate to Team => Git => Configuration and hit the New Entry... Button. Enter user.name as Key and your name as Value and confirm. Repeat this procedure with user.email and your email address and click OK in the Preferences window. The username and email should be the same you use for your Git account, ie. your GitHub account.

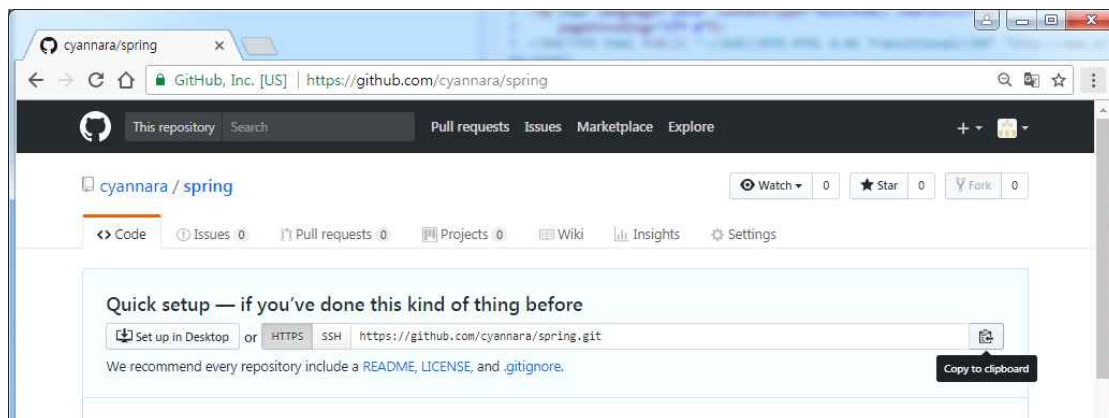


나. github repository clone

Perspective 에서 Git 을 선택합니다.



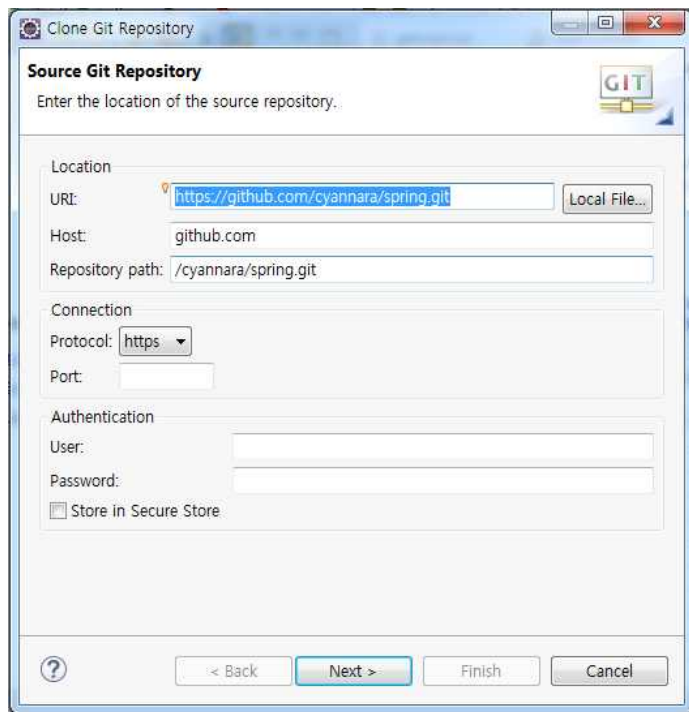
git url 복사



컨텍스트메뉴 -> Paste Repository Path or URI



■ user와 password 입력하고 store 체크



URI: GitHub에서 알려주는 HTTPS clone URL 주소를 입력

URI를 입력하면 자동으로 Host, Repository path가 입력됨

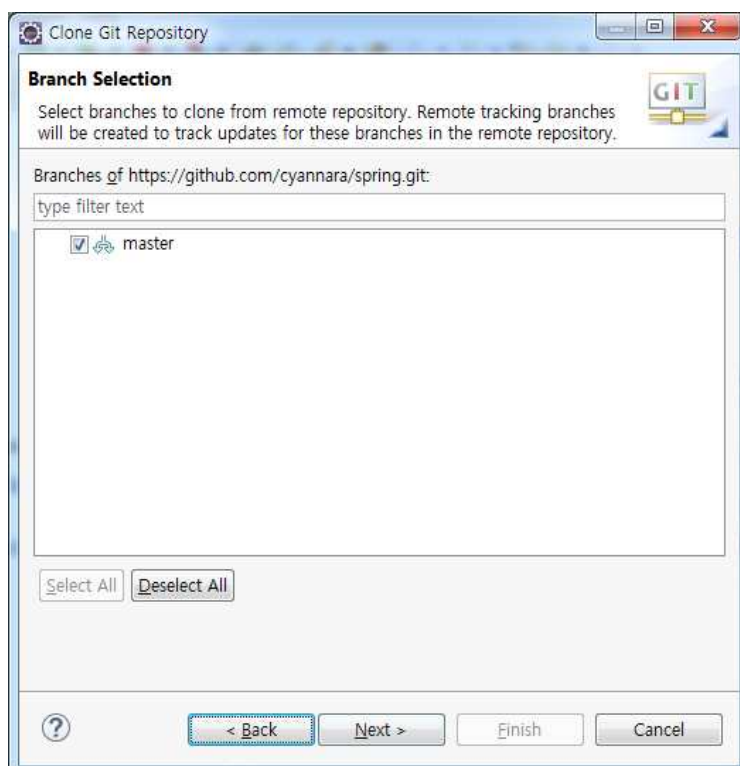
Protocol : 원격저장소를 가져오는 데 필요한 네트워크 프로토콜을 선택한다. file, ftp, git, https, sftp, ssh 중 하나를 선택한다.

Port: 특정 포트에 접속해야만 원격 저장소를 가져올 수 있는 경우 해당 포트 번호를 입력한다. 아니면 비워둔다.

User/Password : pull만 받는 경우는 필요 없으면 push할 경우는 입력

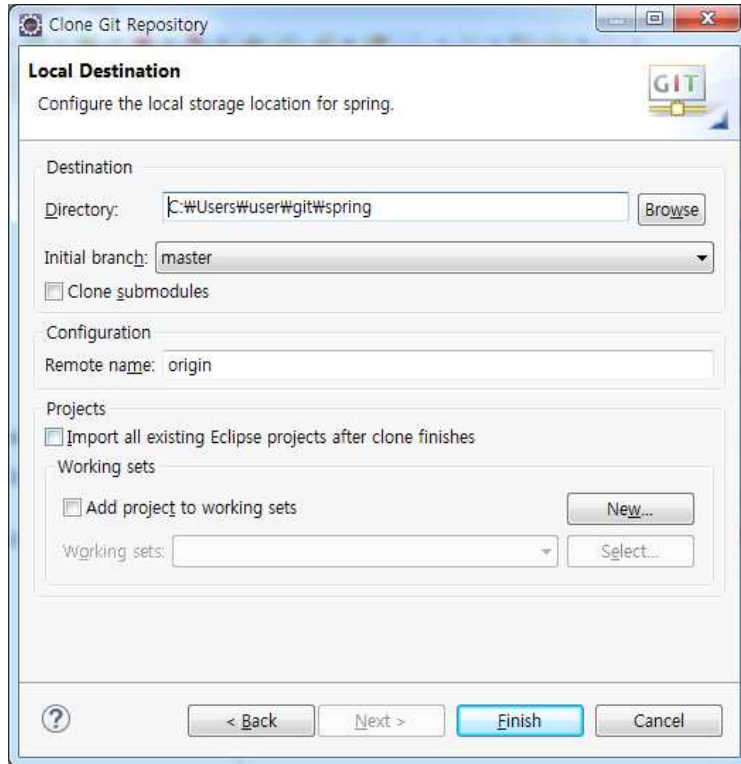
Store in Secure Store: 로컬 보안 저장소에 원격 저장소를 저장해야 할 경우 체크

■ Branch Selection



master 이외의 다른 브랜치 정보도 함께 로컬 저장소에 가져오고자 하는 경우 브랜치를 선택한다.

■ Local Destination : 프로젝트를 저장하는 workspace를 지정하면 된다.

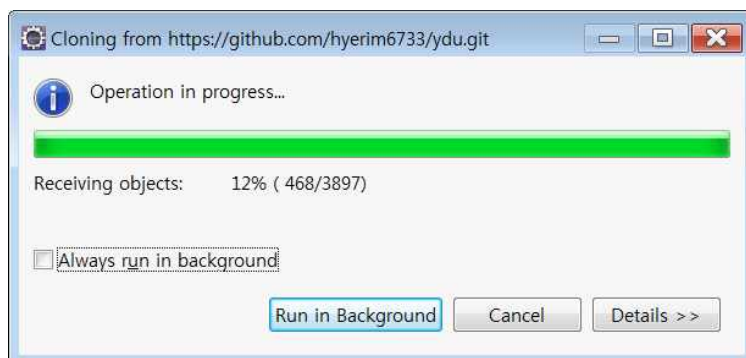


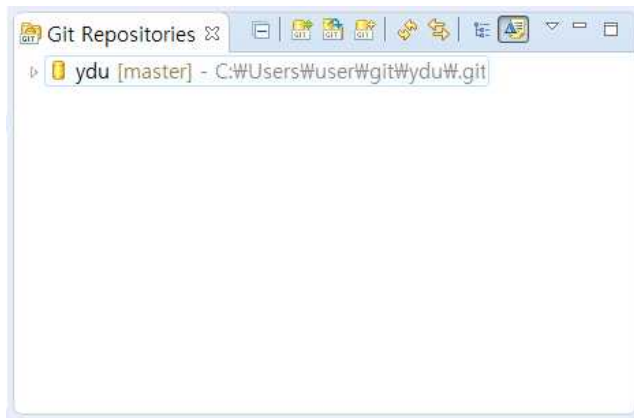
Directory : 원격 저장소의 디렉토리 경로를 설정

Initial branch : 초기 브랜치를 선택. 원격 저장소에 여러 개 브랜치가 있는 경우 원하는 브랜치를 선택

clone submodules: 현재 원격 저장소를 메인 프로젝트의 하위 프로젝트인 서브 모듈로 클론하려면 체크 표시를 해준다. 해당 원격 저장소가 프로젝트에 사용되는 라이브러리인 경우 유용하다.

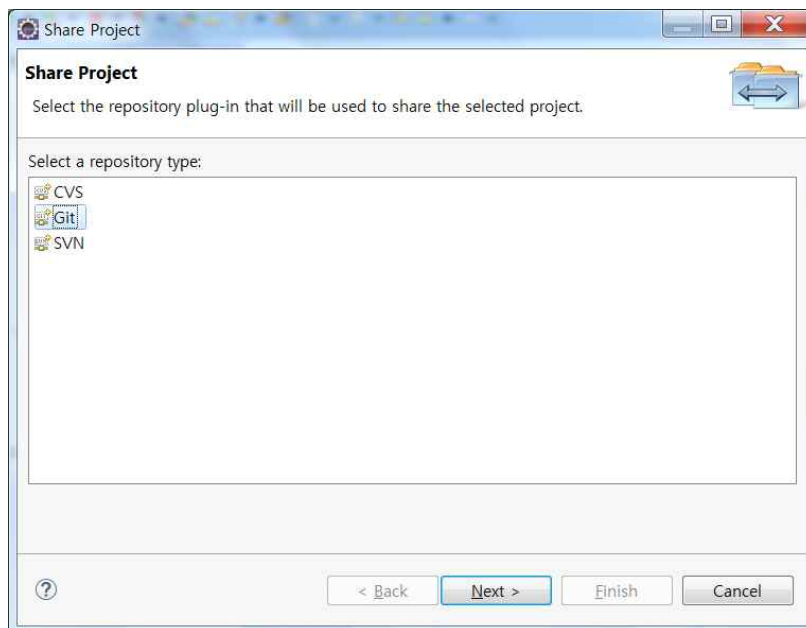
Remote name :



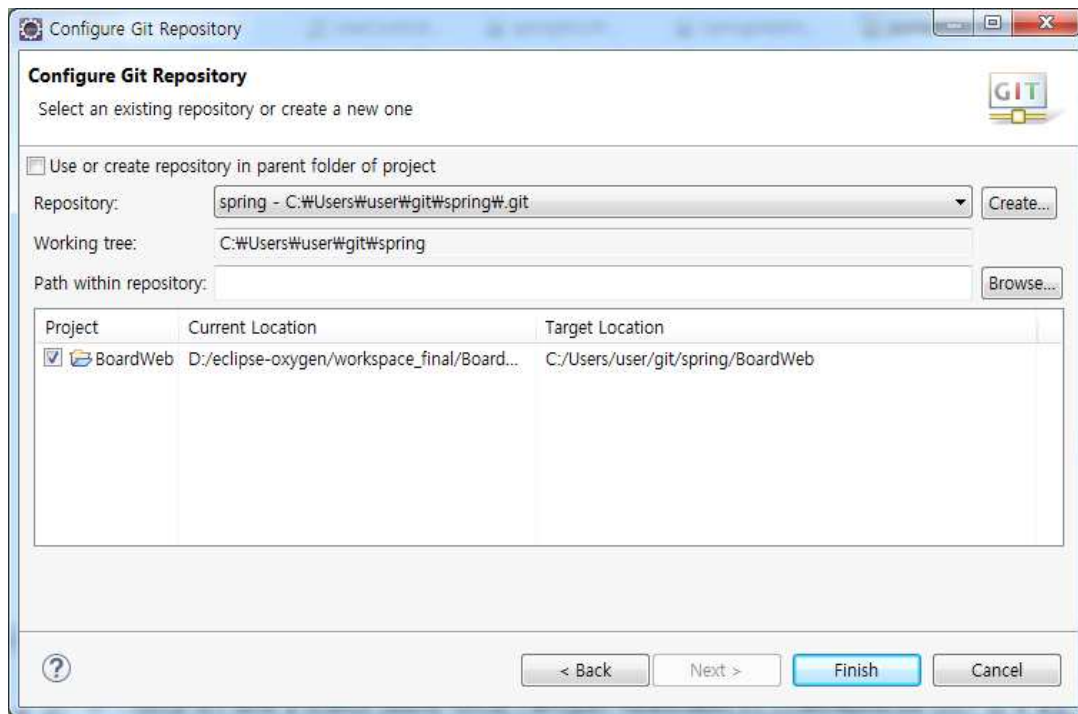


다. 프로젝트 공유하기(올리기)
프로젝트 소스를 업로드

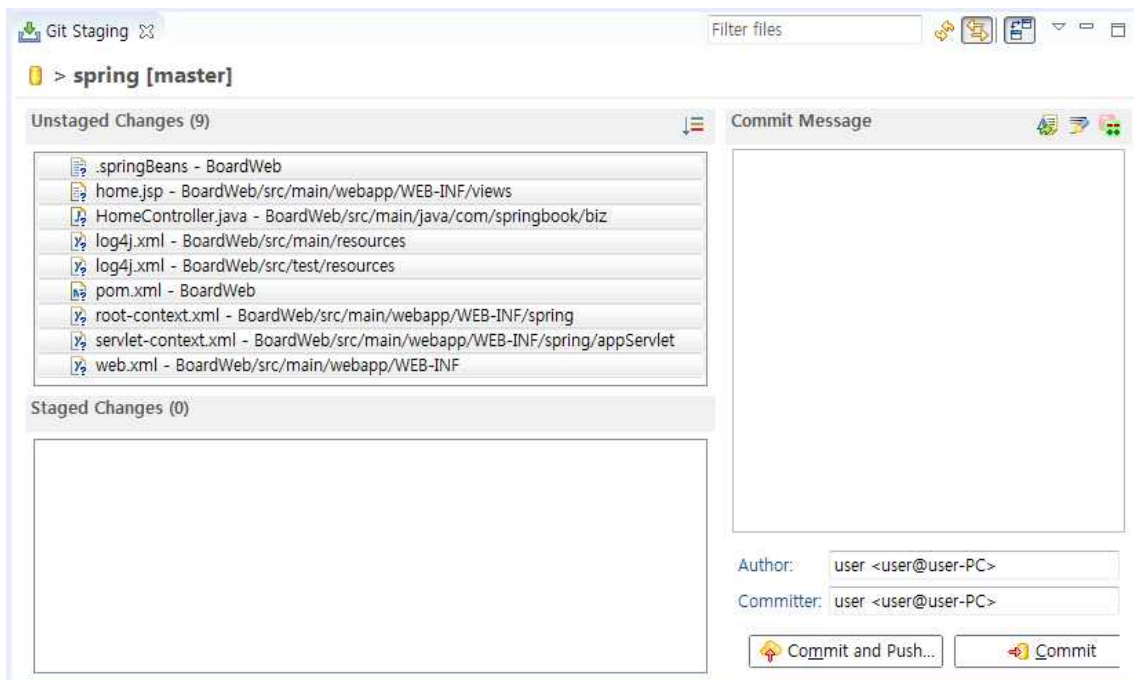
(1) team -> share project



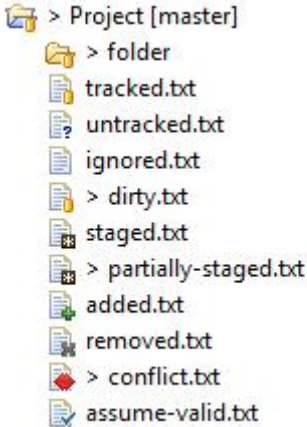
(2) repository 선택



(3) Add to Git Index



Unstaged Changes의 파일목록을 Staged Change 상태로 드래그하여 이동 또는 이클립스에서 파일을 Staged 하기 위해서는 Add to Git Index를 선택해 주면 된다.

	<ul style="list-style-type: none"> ● dirty (folder) - At least one file below the folder is dirty; that means that it has changes in the working tree that are neither in the index nor in the repository. ● tracked - The resource is known to the Git repository. ● untracked - The resource is not known to the Git repository. ● ignored - The resource is ignored by the Git team provider. Here only the preference settings under Team -> Ignored Resources and the "derived" flag are relevant. The .gitignore file is not taken into account. ● dirty - The resource has changes in the working tree that are neither in the index nor in the repository. ● staged - The resource has changes which are added to the index. Not that adding to the index is possible at the moment only on the commit dialog on the context menu of a resource. ● partially-staged - The resource has changes which are added to the index and additionally changes in the working tree that are neither in the index nor in the repository.
---	---

(4) commit

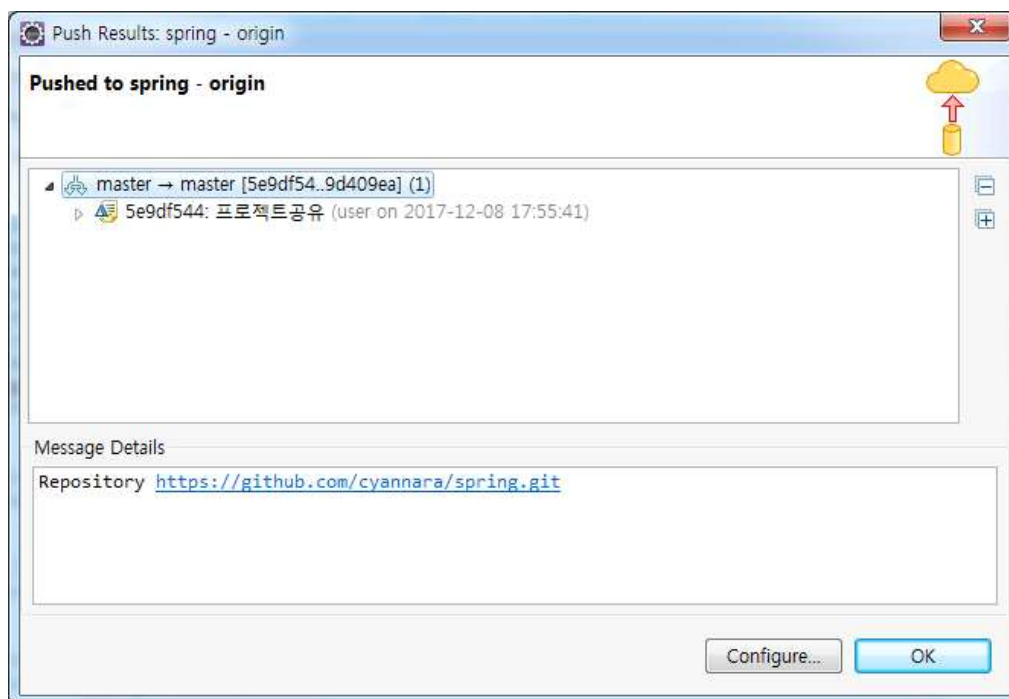
git Staging에서 commit message 입력 후 commit 버튼 클릭

또는 프로젝트 선택 > Team > commit

(5) push

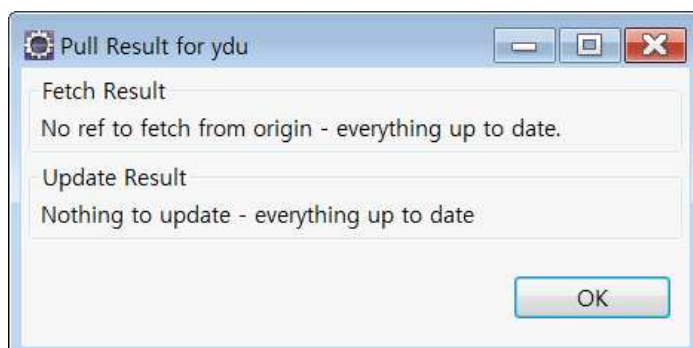
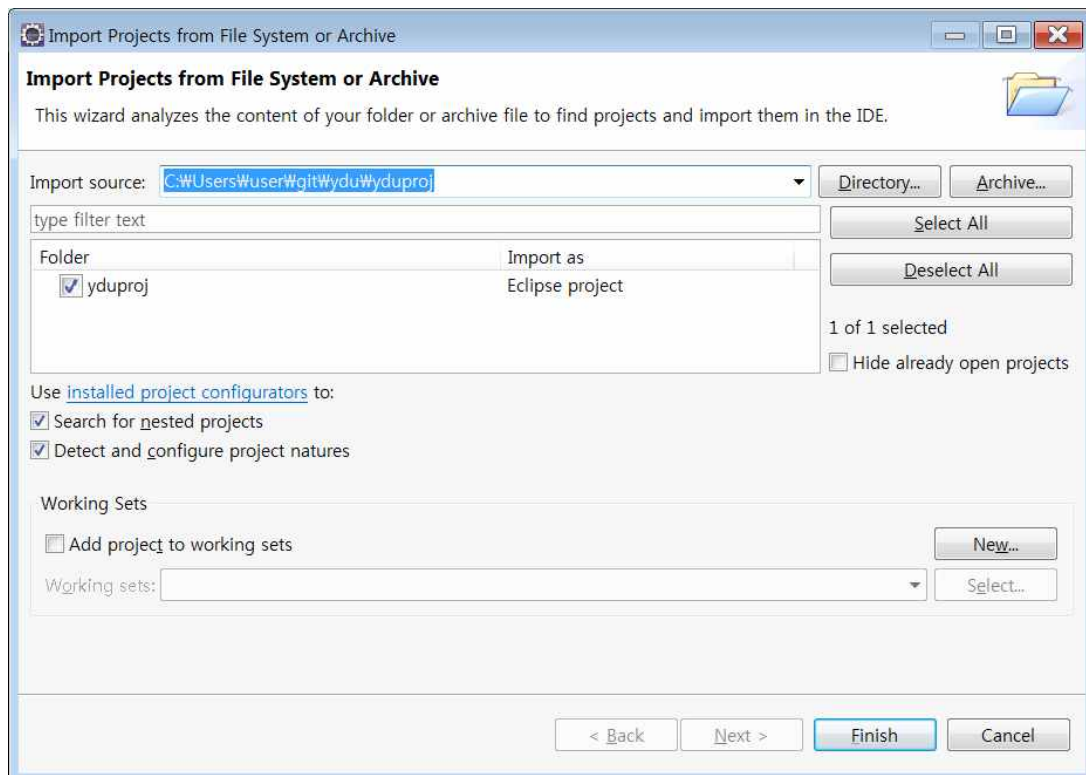
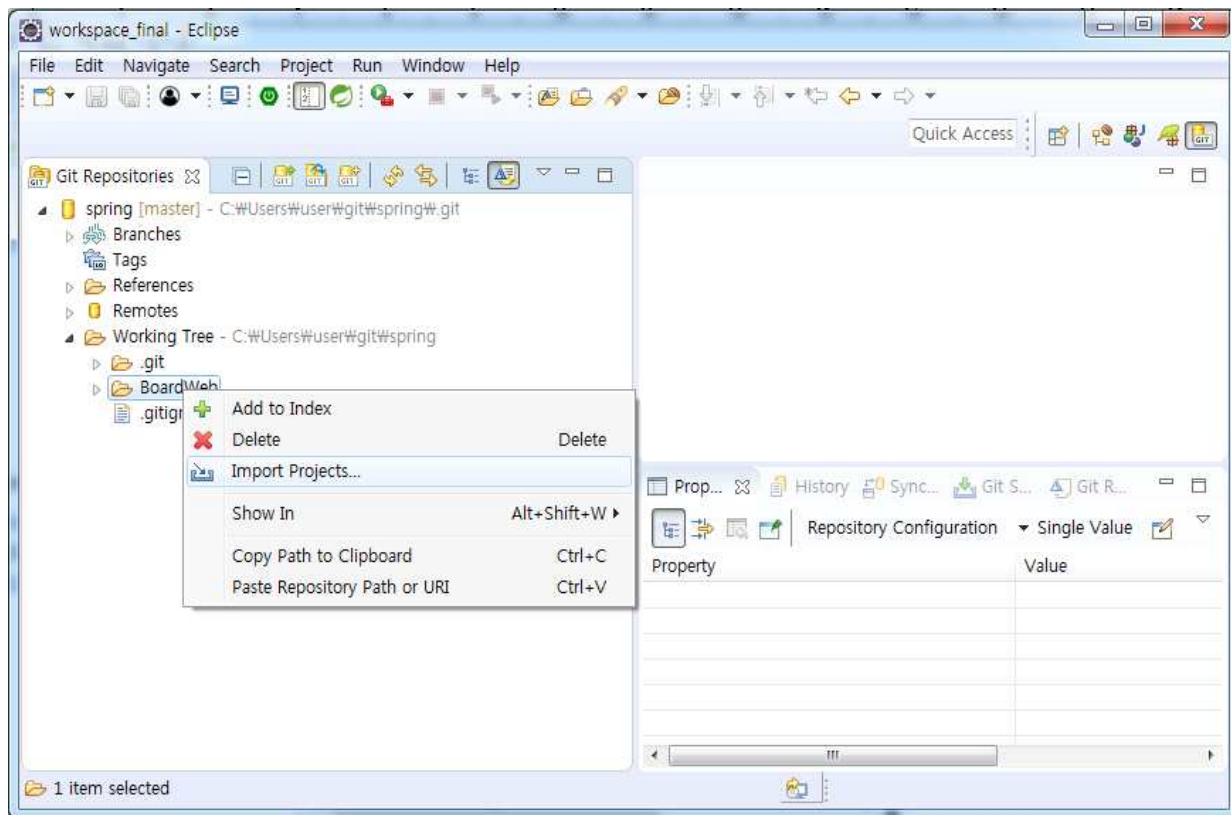
git Staging에서 push

또는 프로젝트 선택 > Team > Push



라. 프로젝트 내려 받기

Working Tree에서 프로젝트를 선택하고 컨텍스트 메뉴에서 Import Projects를 선택



마. conflict

① (평소처럼) 프로젝트 우클릭 > Team > Commit > Commit and Push 를 했을 때, 난데없이

[rejected - non-fast-forward]

가 발생했다면? 여러분의 팀원이 여러분 모르는 사이에 commit 을 해서 원격 저장소(remote repository)로 push 한 상황입니다. 해당 원격 저장소를 새로운 경로로 clone 해 보세요. 여러분이 작업 하고 있는 로컬 저장소에는 보이지 않는 commit 이 보일 것입니다.

② 프로젝트 우클릭 > Team > Pull

→ Fetch Result : 서버의 commit 내역

Update Result :

- Result Conflicting

- Merge input : Local 의 마지막 commit VS 서버의 마지막 commit

③ Conflict 가 발생하는 파일 우클릭 > Team > Merge Tool > Use HEAD

④ 저장 합니다.

⑤ 수정한 파일(Conflict 가 발생했던 파일) 우클릭 > Team > Add to Index

⑥ 프로젝트 우클릭 > Team > Commit > Commit and Push

바. SYNC

프로젝트를 클릭 한 후 team > Synchronize

사. revert

아. reset

(1) Soft:

The current branch's tip will point to this branch/tag/commit. Changes in the index and working directory, however, won't be reset.

(2) Mixed:

Same as a soft reset, only that the current index will be replaced by the selected branch/tag/commit's index. The working directory stays unchanged.

(3) Hard:

All changes will be reverted to the selected branch/tag/commit. Uncommitted changes will be lost, therefore this operation has to be confirmed.

자. history view

차. git에서 ignore 파일 설정하기

1. 무시할 [폴더/파일]을 오른쪽클릭 -> Team -> ignore
2. git 에 업데이트합니다.(.gitignore 파일이 적용)

직접 하기

1. git을 받은 폴더로 이동(숨김파일 .git 이 있는 폴더)
- 2 .git이 있는 폴더내에 .gitignore 파일을 만든다.
3. .gitignore 파일에 아래와 같이 입력한다..

Java + Maven 프로젝트를 Eclipse 로 개발할 때 아래 내용대로 작성해서 git ignore 해주면 된다.

.gitignore 파일

```
파일 : .gitignore

### Eclipse ###
*.pydevproject
.metadata
.gradle
bin/
tmp/
*.tmp
*.bak
*.swp
*~.nib
local.properties
.settings/
.loadpath

# External tool builders
.externalToolBuilders/

# Locally stored "Eclipse launch configurations"
*.launch

# CDT-specific
.cproject

# PDT-specific
.buildpath

# sbteclipse plugin
.target

# TeXlipse plugin
.texlipse

### Maven ###
target/
pom.xml.tag
pom.xml.releaseBackup
pom.xml.versionsBackup
pom.xml.next
release.properties

### Java ###
*.class

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.ear

# virtual machine crash logs, see
#http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
```

5. gittable

The screenshot shows the 'GUI Clients' page on the git-scm.com website. The page has a sidebar on the left with links for 'About', 'Documentation', 'Downloads' (including 'GUI Clients' and 'Logos'), and 'Community'. The main content area is titled 'GUI Clients' and explains that Git includes built-in GUI tools like `git-gui` and `gitk`, but also lists third-party tools for platform-specific experiences. It provides a link to follow instructions for adding more GUI tools. Below this, there are buttons for 'All', 'Windows', 'Mac', 'Linux', 'Android', and 'iOS'. A note states '29 Windows GUIs are shown below ↓'. Two screenshots are displayed: one of SourceTree and one of GitHub Desktop. SourceTree is a Windows application with a sidebar for 'Working Copy', 'BRANCHES', 'TAGS', and 'REMOTES', and a main area for 'Commit', 'Checkouts', and 'Graph'. GitHub Desktop is a macOS application showing a 'Changes' list with 6 changed files and a 'History' list with 48 commits. It also shows a diff view for the selected commit.

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire **Pro Git** book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

GUI Clients

Git comes with built-in GUI tools for committing (`git-gui`) and browsing (`gitk`), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just [follow the instructions](#).

[All](#) [Windows](#) [Mac](#) [Linux](#) [Android](#) [iOS](#)

29 Windows GUIs are shown below ↓

SourceTree
Platforms: Mac, Windows
Price: Free
License: Proprietary

GitHub Desktop
Platforms: Mac, Windows
Price: Free
License: MIT

참고사이트

egit

egit tutorial : <https://eclipsesource.com/blogs/tutorials/egit-tutorial/>

Icon Decorations : http://wiki.eclipse.org/EGit/User_Guide/State

Egit 시나리오별 사용법 : <http://lng1982.tistory.com/177>

Github(깃허브) 와 이클립스 연동하기 : <http://jwgye.tistory.com/38>

git 사용법 : <http://www.itpaper.co.kr/eclipse에서-Github-연동하기/>

팀원초대 : <http://www.itpaper.co.kr/github에-프로젝트-생성하고-팀원-초대하기>