**Git Management**

**& SW Convention Rule**

**Version 0.1 – November 3, 2015**

**Document Version and Control**

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Data | Change Summary | Updated by |
| 0.1.0 | 2015.11.03 | 최초작성 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Distribution**

|  |  |  |
| --- | --- | --- |
| Version | Data | Parties |
| 0.1.0 | 2015.11.03 | 모두의 연구소, #나혼자난다. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

목 차

[1 도입 5](#_Toc425787602)

[1.1 목적 5](#_Toc425787603)

[1.2 용어 및 약어 정의 5](#_Toc425787604)

[1.3 관련 문서 5](#_Toc425787605)

[2 Git/GitLab 설치 및 사용방법 6](#_Toc425787606)

[2.1 Git install 6](#_Toc425787607)

[2.2 GitLab install 6](#_Toc425787608)

[2.3 GitLab Group & Project 생성 및 사용법 8](#_Toc425787609)

[2.4 GitLab Milestone 및 Issue 사용법 13](#_Toc425787610)

[3 SourceTree 설치 및 사용방법 16](#_Toc425787611)

[3.1 SourceTree install 16](#_Toc425787612)

[3.2 SourceTree SSH Key 설정 20](#_Toc425787613)

[3.3 SourceTree Clone 생성 21](#_Toc425787614)

[3.4 SourceTree New Repository 생성 23](#_Toc425787615)

[3.5 commit & push 28](#_Toc425787616)

[3.6 pull & merge 28](#_Toc425787617)

[3.7 충돌 (Conflict) 28](#_Toc425787618)

[4 GIT Repository 31](#_Toc425787619)

[4.1 위치 및 권한 31](#_Toc425787620)

[5 GIT Convention Rule 32](#_Toc425787621)

[6 Software Convention Rule 34](#_Toc425787622)

[6.1 License notice 34](#_Toc425787623)

[6.2 Reference Coding Rule 34](#_Toc425787624)

[6.3 Define 관련 사항 35](#_Toc425787625)

[6.4 Macro 표기법 35](#_Toc425787626)

[6.5 함수 표기법 35](#_Toc425787627)

[6.6 변수 표기법 (헝가리안 표기법) 36](#_Toc425787628)

[6.7 주석표기법 36](#_Toc425787629)

[6.8 SW Coding 시 유의사항 37](#_Toc425787630)

# **도입**

## **목적**

Software Version Control을 위한 문서임. 더 큰 개념이며 복잡한 형상관리시스템 (Software Configuration Management와는 구별할 필요가 있다. 즉, Software Version Control은 일반적인 software source code만을 관리하는 내역을 주로 버전 관리로 정의하면 되고, 형상관리는 추상적인 관리대상을 형상화한다는 의미이다. 형상을 표현하는 것이 문서라고 보면 되고, 그 대상은 시스템일 수도 있고, 프로젝트라고 할 수도 있지만, 제품을 만드는 주체 입장에서는 제품과 소프트웨어의 결합으로 해석하는 것이 적절하다고 보인다.

Version control, revision control, source control, source code management (SCM)은 동일한 정보에 대한 여러 버전을 의미함.

## **용어 및 약어 정의**

|  |  |
| --- | --- |
| 항목 | 내용 |
| Git | * 분산 리비전 컨트롤, 소스 코드를 관리한다. |
| GitLab | * Git의 원격 저장소 기능과 이슈 트래커, 마일스톤 등의 기능을 제공하는 Web 버전의 S/W |
|  |  |
|  |  |
|  |  |
|  |  |

## **관련 문서**

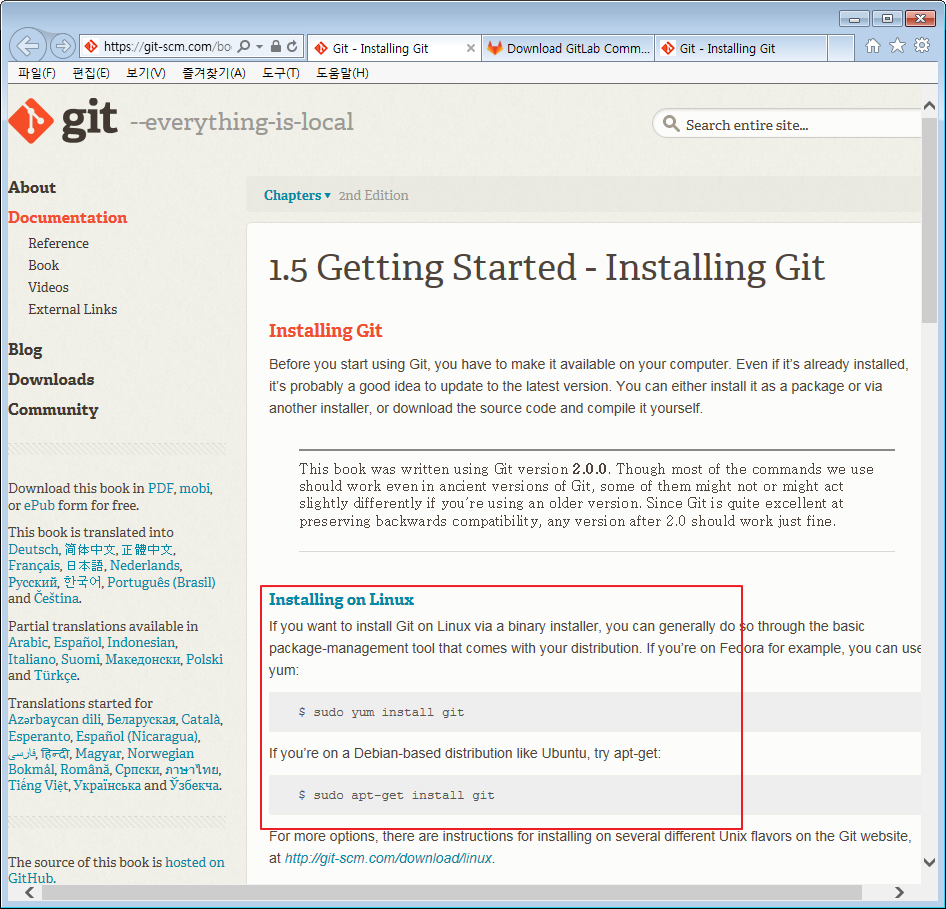
|  |  |  |
| --- | --- | --- |
| 문서명 | 설명 | 비고 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# **Git/GitLab 설치 및 사용방법**

## **Git install**

Git 설치는 간단하다

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git> 접속하여 각 버전에 맞게 설치하면 된다.



## **GitLab install**

GitLab 다운로드 페이지 주소 <https://about.gitlab.com/downloads/> 이고 해당 OS를 선택하면 설치방법이 나오면 Window 버전은 지원을 하지 않는다.

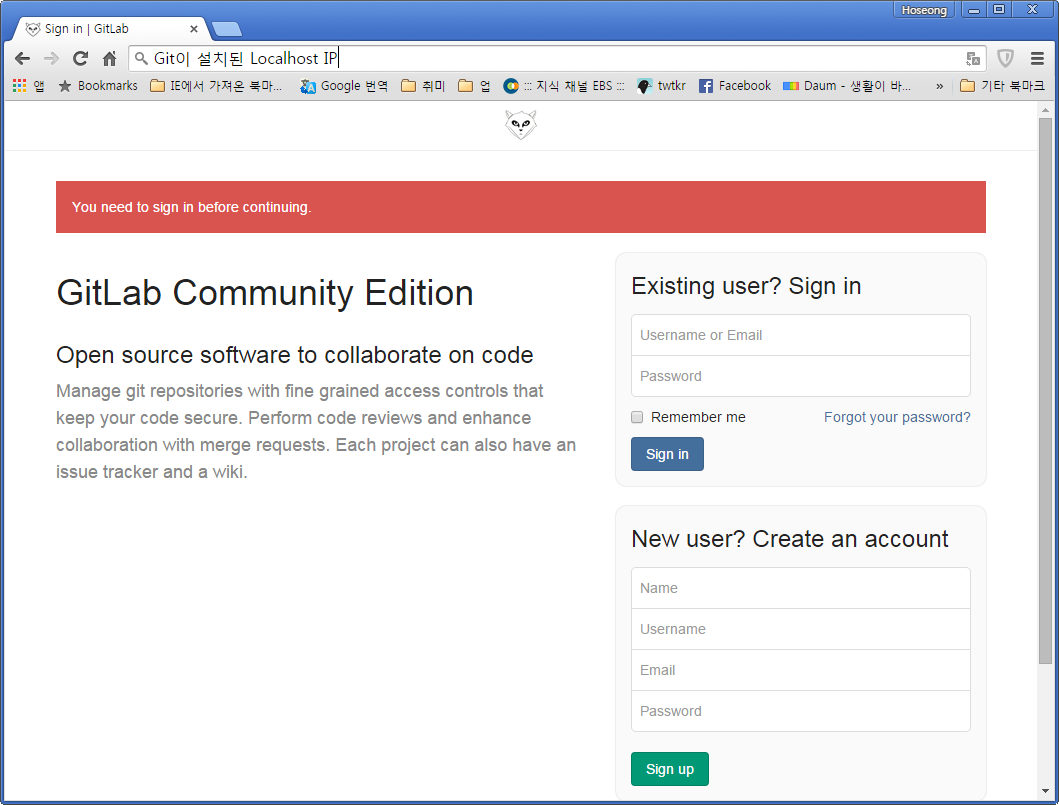
GitLab 설치도 Git만큼 간단하지만 Git 정식 버전은 64bit만 지원하기 때문에 32bit일 경우 설치방법이 달라진다.

설치하려는 Linux OS 가 32bit일 경우 <https://bitnami.com/stack/gitlab/installer> 에 접속하여 64bit 문구가 붙어 있지 않은 파일을 다운받으면 된다. 32bit 버전의 GitLab을 다운 받았으면 다운받은 폴더로 이동하여 터미널에서 아래와 같이 실행시킨다.

|  |
| --- |
| $ chmod 755 bitnami-gitlab-6.1.0.0-linux.run  $ sudo –s //root 권한으로 실행시켜야 설치된다.  $ ./ bitnami-gitlab-6.1.0.0-linux.run |

Bitnami GitLab은 GUI로 설치되기 때문에 Next 버튼만 클릭하면 설치 종료된다.

GitLab 설치 후 GitLab이 설치된 Localhost IP를 웹브라우저에서 실행 시키면 GitLab 화면이 나온다.

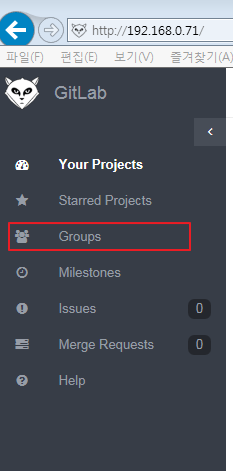


GitLab은 Git 서버 저장소를 운영하는 Tool로써 Web UI 상에서 손쉽게 Project와 저장소를 만들 수 있고 Project의 간단한 Milestones와 Issue Tracker 기능도 할 수 있어 따로 형상관리 Tool을 사용하지 않아도 된다.

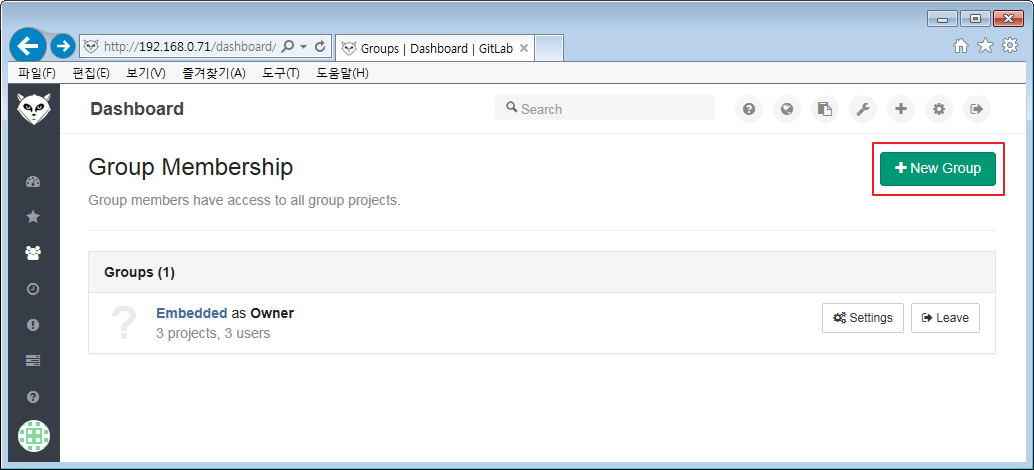
## **GitLab Group & Project 생성 및 사용법**

Group은 User 계정이 Admin 권한 이상, Project는 Master 권한 이상 만들 수 있다.

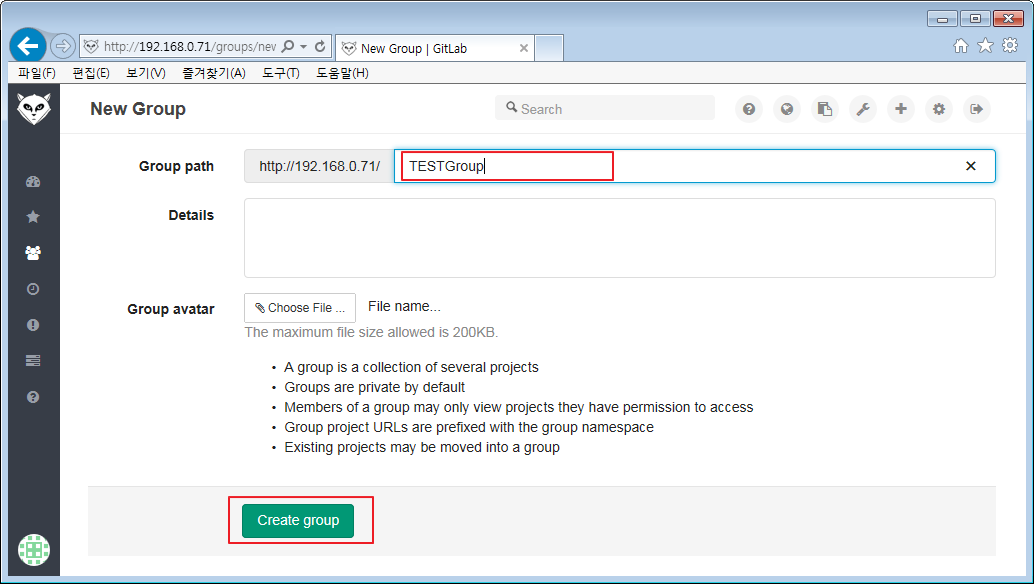
Group 은 Admin 권한의 User만 만들 수 있다.



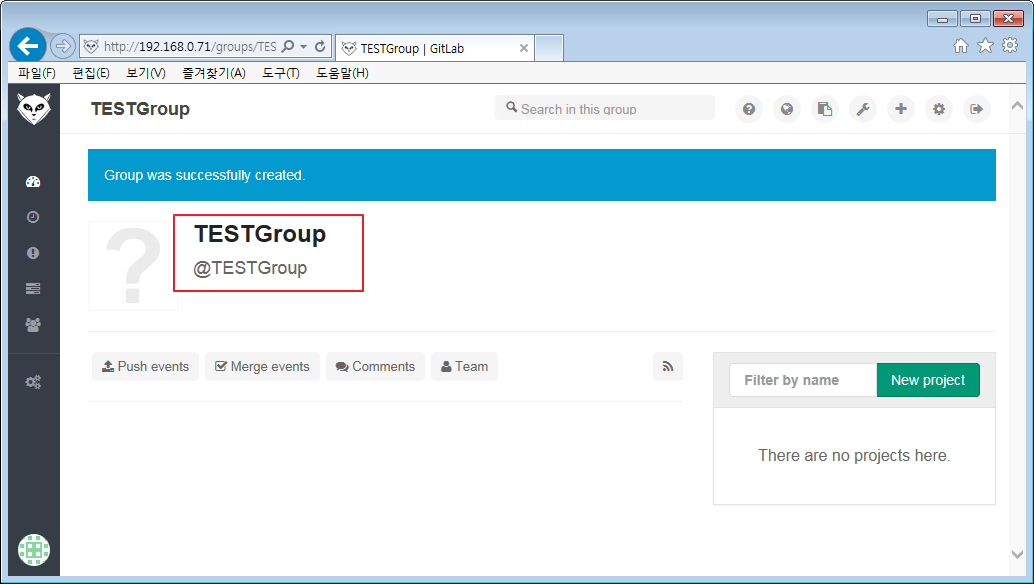
Admin User는 접속 후 GitLab의 왼쪽 메뉴에서 Group를 선택 클릭하면 새로운 Group를 생성할 수 있는 화면이 나온다.



빨간 네모 박스의 +New Group 버튼을 클릭하면 새로운 Group 을 만들 수 있는 페이지로 이동한다.

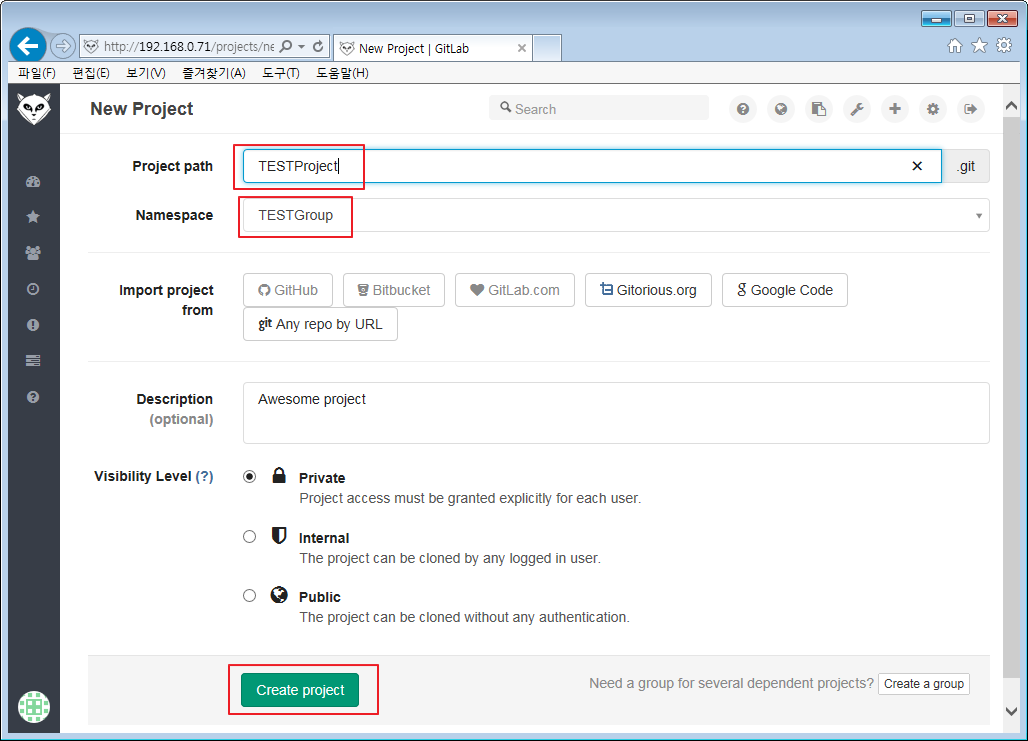


Group path의 에디터 박스에 새로운 그룹명( TESTGropu )을 적은 후 Create group 버튼을 클릭하면 아래와 같이 새로운 그룹이 생성된다.

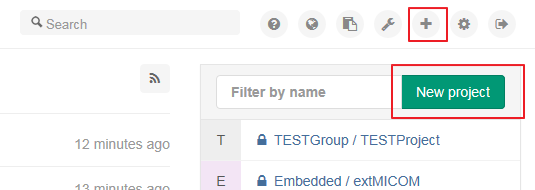


Project 생성은 Master 권한 이상의 User면 가능하다. Project 생성에는 몇 가지 방법이 있다.

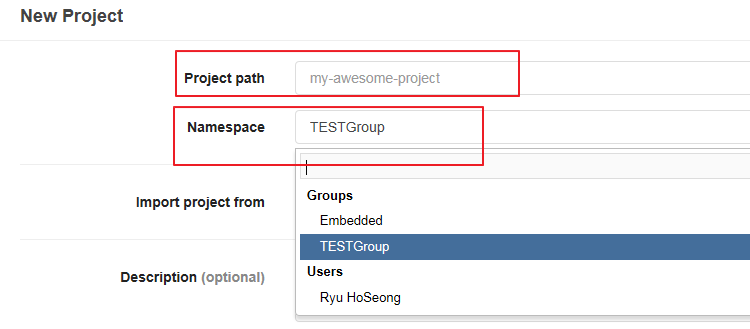
첫 번째 방법으로는 Group 을 생성 후 Group 페이지에 들어가서 New project 버튼을 클릭하는 방법이 있다. 이 경우 Namespace가 해당 Group로 되어 있다. Namespace는 해당 Project의 소유권을 나타낸다.



두 번째 방법으로는 GitLab 메인페이지 화면의 오른쪽 상단의 + 버튼을 클릭하거나 New project 버튼을 클릭하여 New Project 화면으로 이동하여 새로운 Project를 생성하는 방법이다.



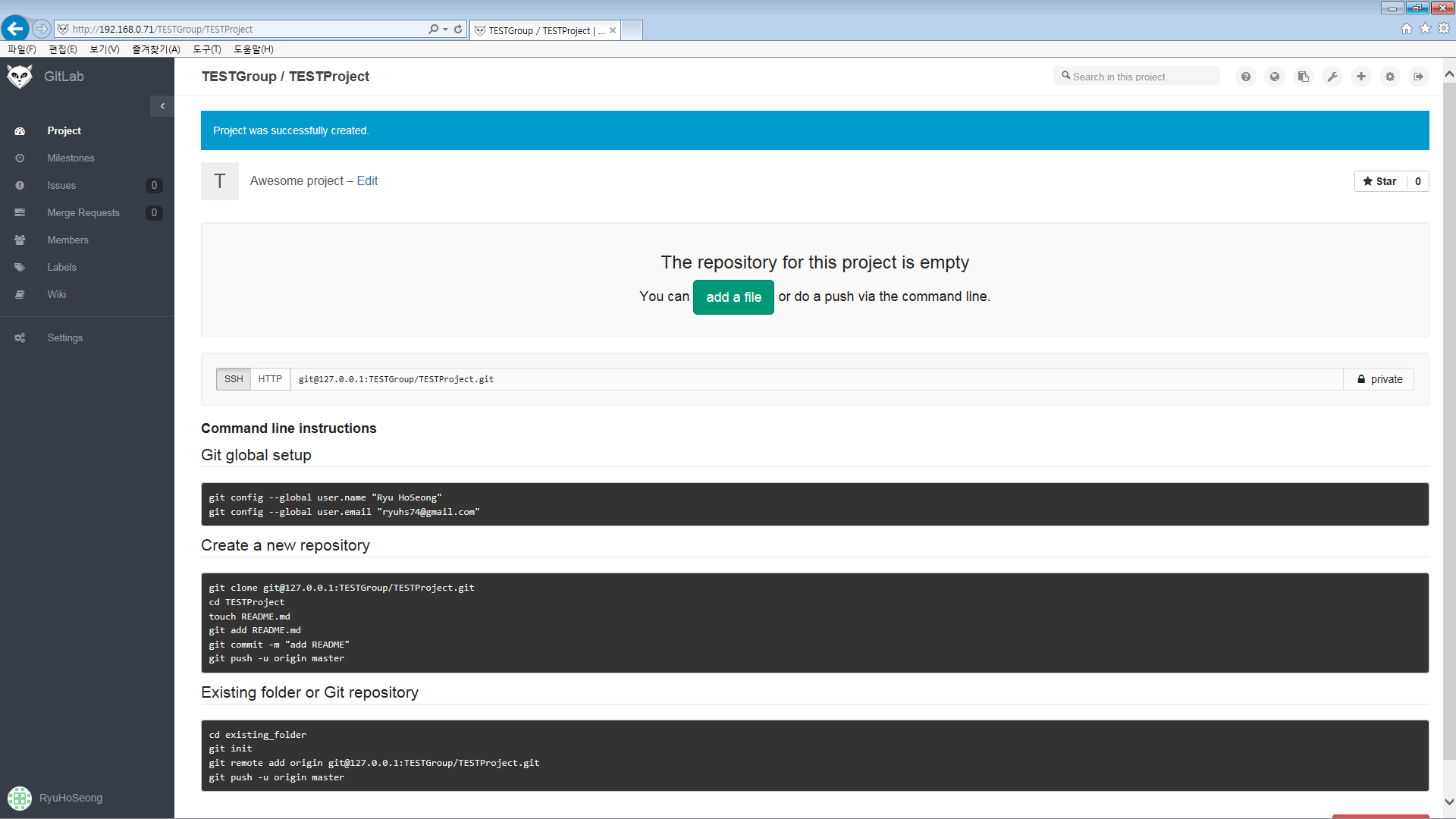
‘+’, New project 버튼을 클릭하면 아래와 같이 New Project 화면이 나온다.



‘Project path’ 에디터 박스에는 Project Name을 적는다. 이 Project Name이 Repository 명이다.

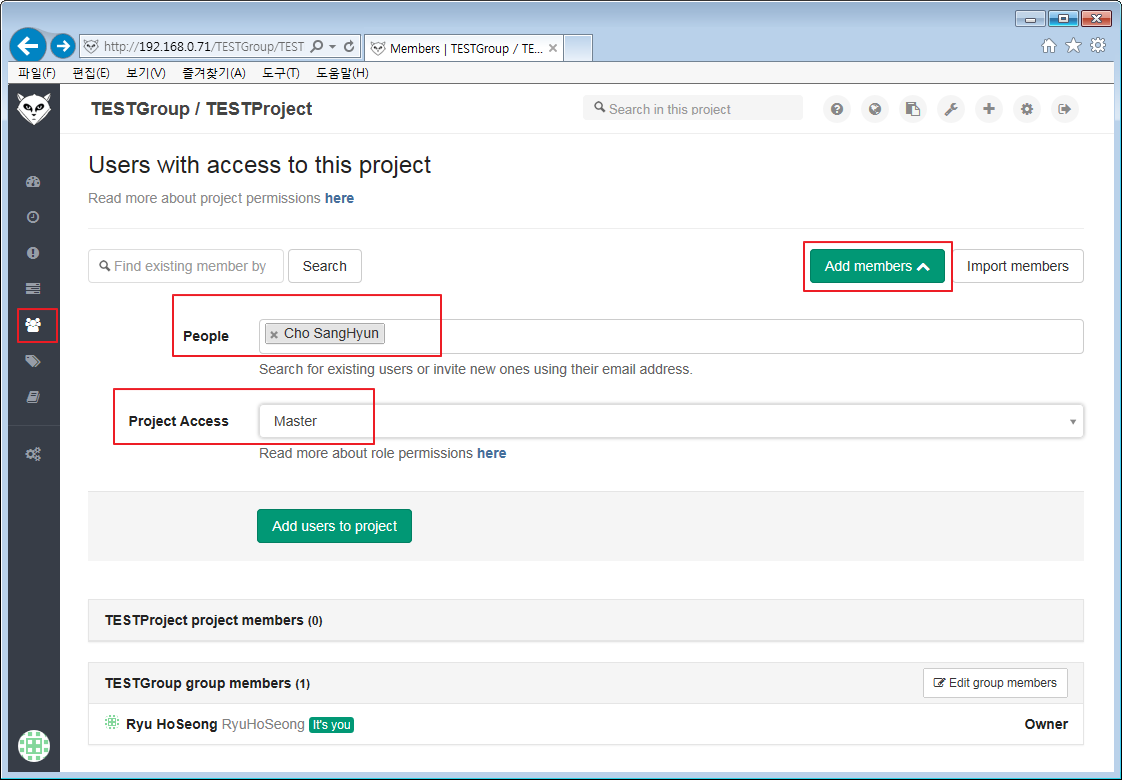
Namespace는 생성된 Project의 소유권을 나타낸다. Namespace는 Group명과 현재 로그인되어 있는 User 명으로 설정할 수 있다.

Project path, Namespace 두 개 모두 설정했다면 밑에 있는 ‘Create project’ 버튼을 눌러 Project 생성을 마무리한다.

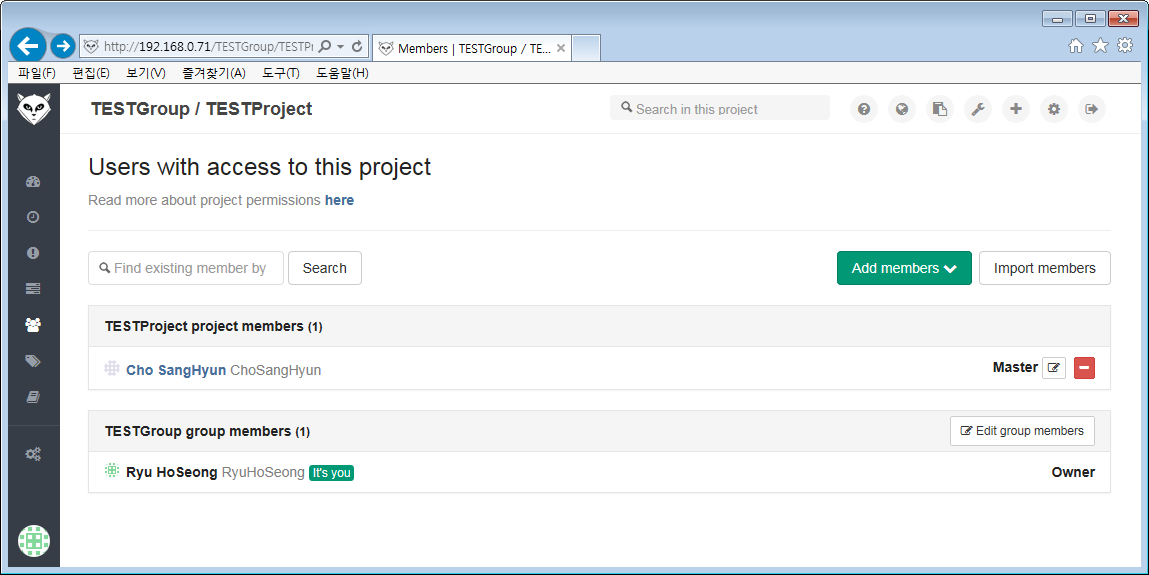


Project가 생성되어 Repository 가 만들어진 모습이다. 아직 Check out 전이기 때문에 자료가 없다.

Project를 생성 한 후 Project에 참가할 Member들을 설정한다. 생성한 Project 페이지에서 좌측 메뉴의 ‘Members’를 클릭하면 Project Member들을 추가하는 페이지로 이동한다. Member 페이지로 이동 후 People의 에디터 컨트롤을 클릭하면 Project에 추가할 수 있는 USER의 이름이 나오고 추가할 USER를 선택하면 된다. Project Access 필드에 추가한 USER의 권한을 줄 수 있는데 Master로 권한을 설정해야 Milestone 설정이 가능하다. 모든 설정이 끝나면 우측 상단에 ‘Add members’라는 버튼을 클릭하면 된다.



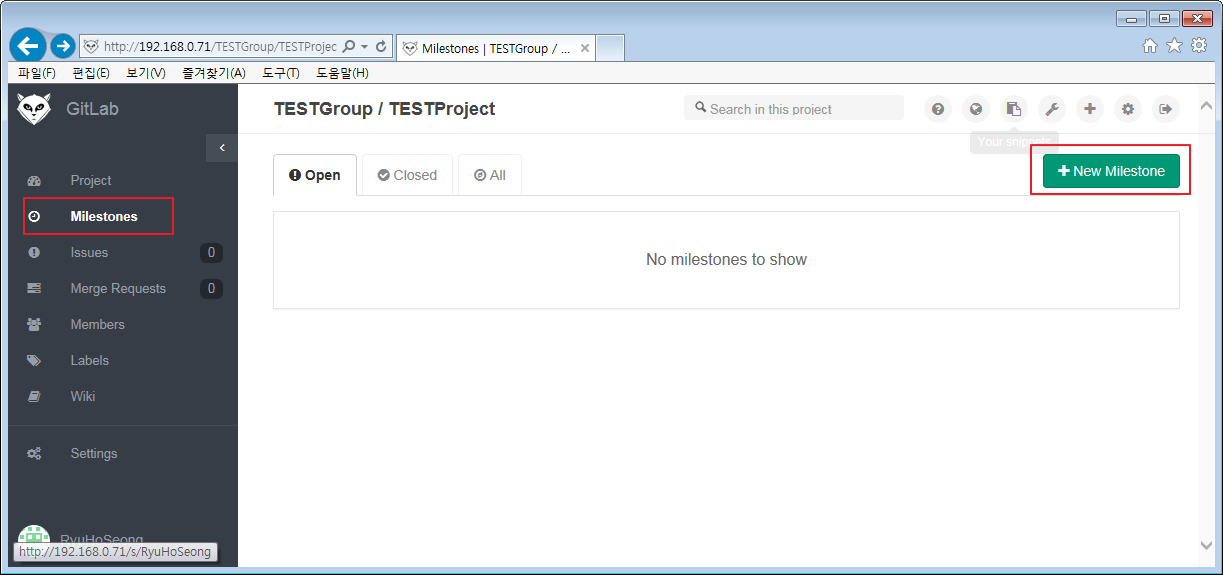
Member 추가 후 화면



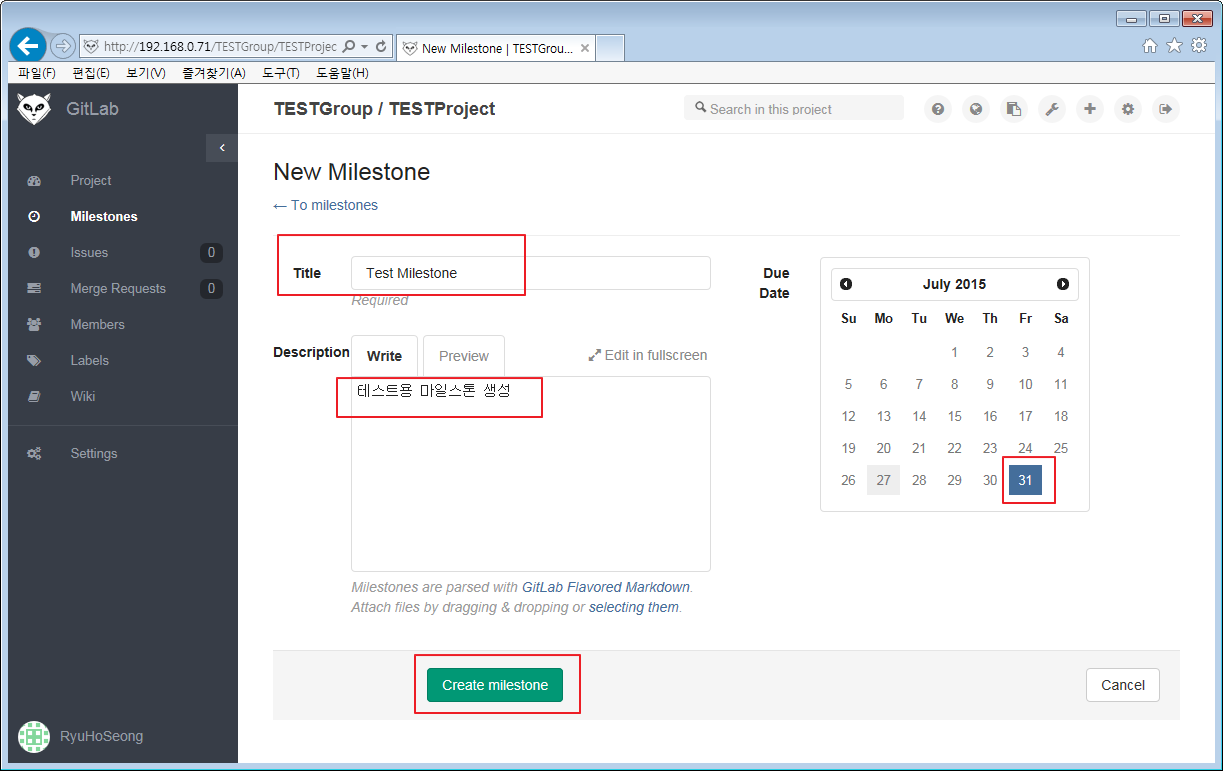
## **GitLab Milestone 및 Issue 사용법**

Milestone과 Issue 등록은 Group Member의 권한이 Master이상일 경우 가능하므로 모든 Group Member의 권한을 Master로 등록해야 한다.

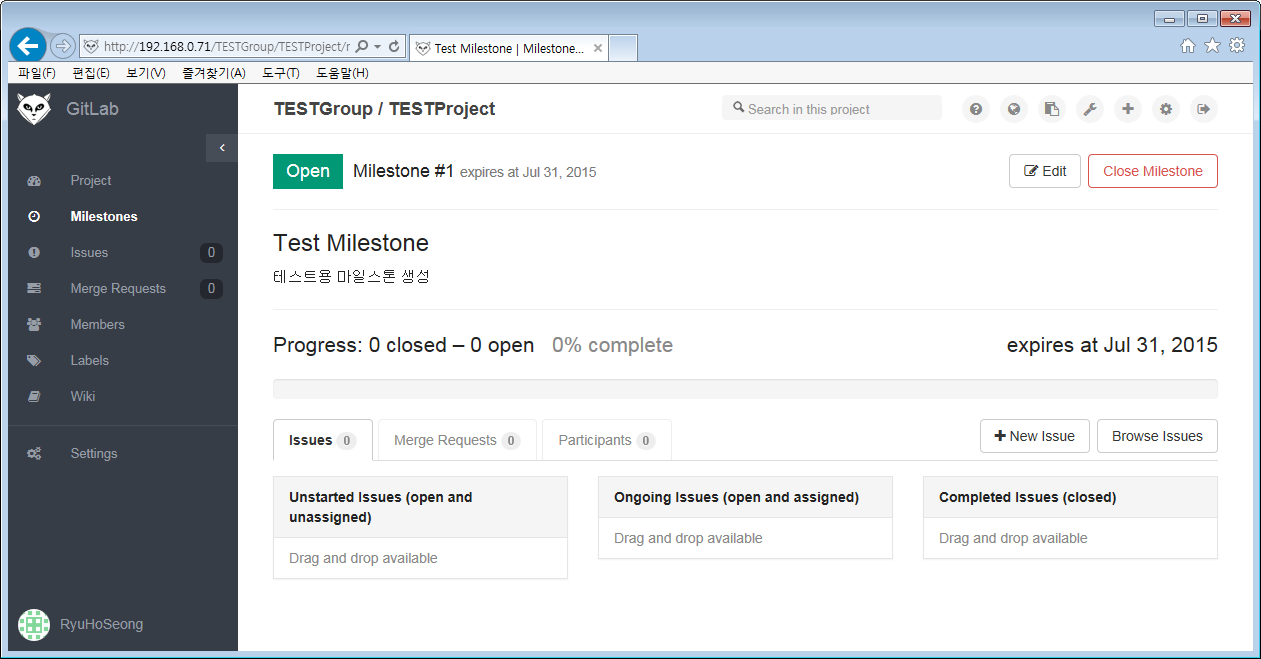
Milestone 등록하기 위해서는 Project 페이지의 좌측 메뉴에서 Milestone 메뉴를 선택하여 Milestone 추가 페이지로 이동한다.



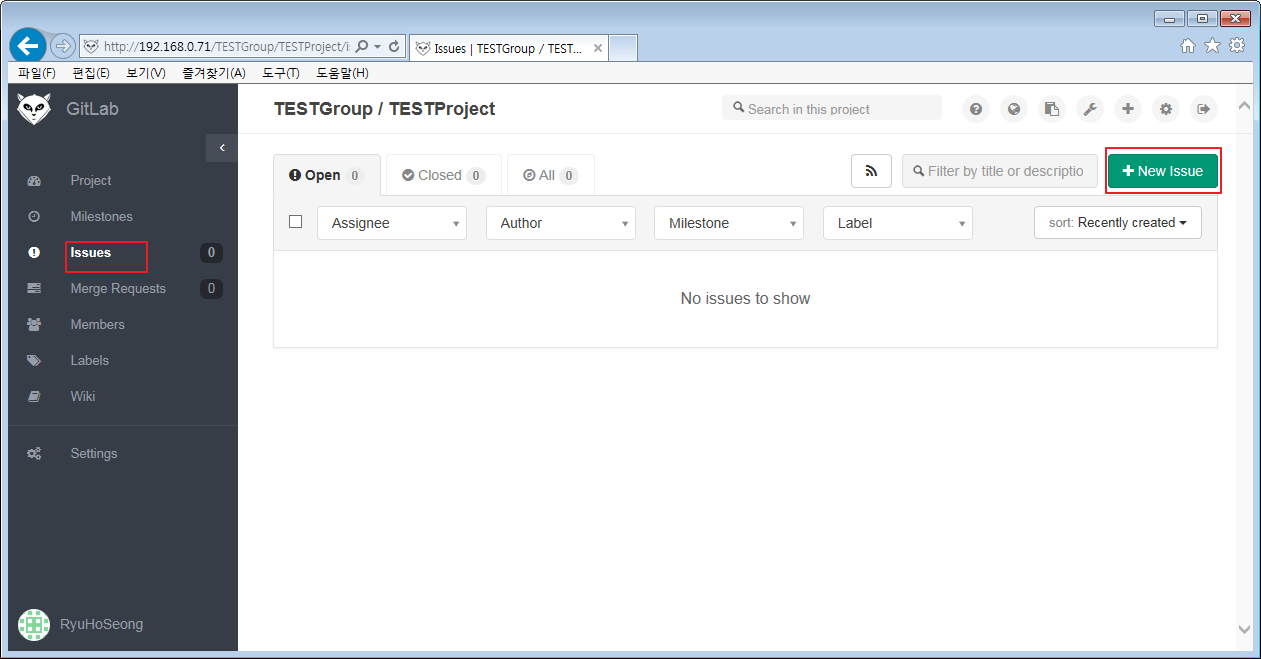
Milestone 추가 페이지에서 ‘New Milestone’ 버튼을 클릭하면 Milestone 추가 페이지가 나온다.



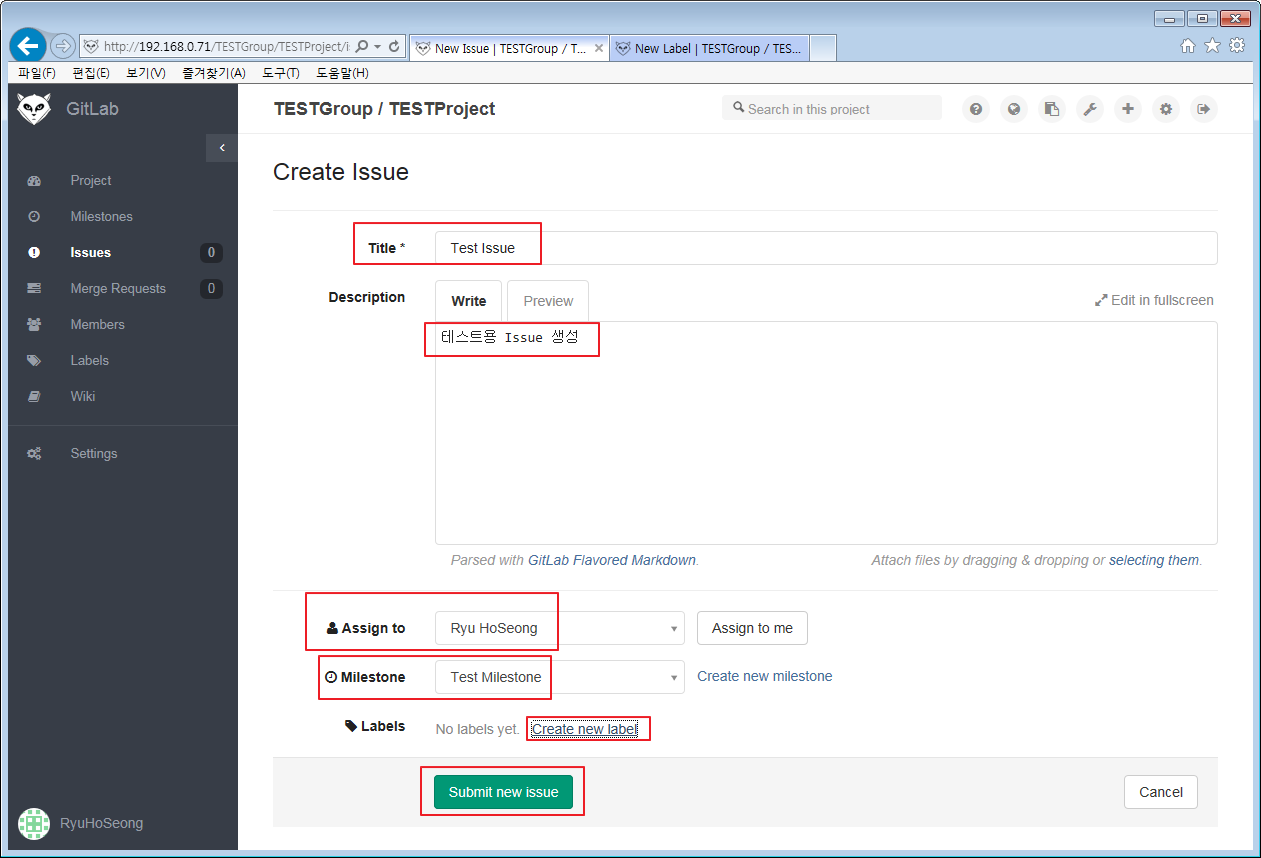
Title과 Description, Close Date를 정한 후 ‘Create milestone’ 버튼을 클릭하면 Milestone이 생성된 페이지를 볼 수 있다.



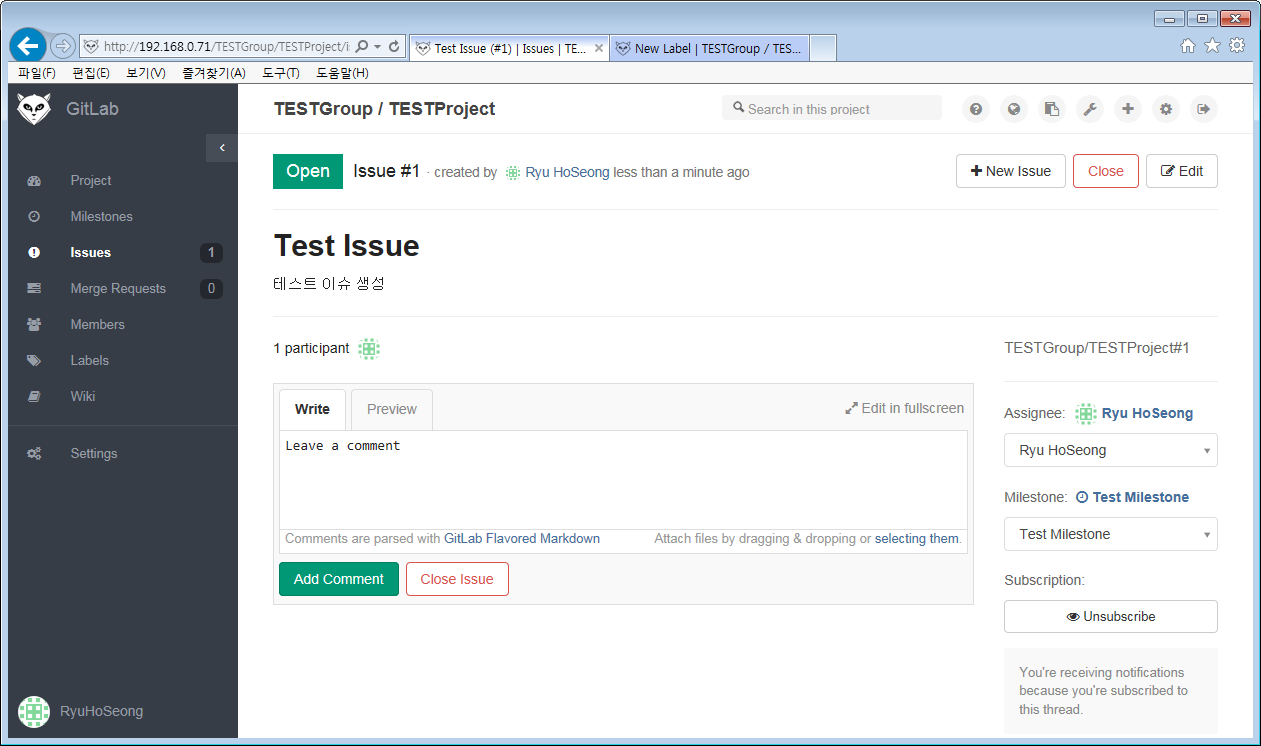
Issue 등록하기 위해서는 Project 페이지의 좌측 메뉴에서 Issue메뉴를 선택하여 Issue추가 페이지로 이동한다.



Issue 추가 페이지에서 ‘New Isse’ 버튼을 클릭하면 Issue 추가 페이지가 나온다.



Title과 Description, Issue에 대한 Assign, 해당 Milestone을 정한 후 ‘Create Issue’ 버튼을 클릭하면 Issue가 생성된 페이지를 볼 수 있다.



GitLab에 관한 추가 사용법은 아래 URL에서 확인 가능하다.

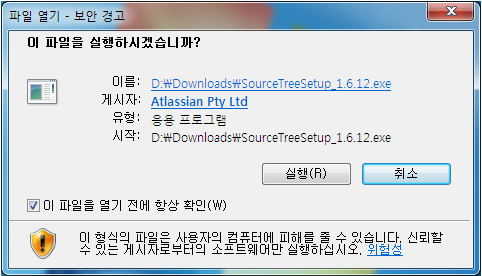
<https://opentutorials.org/course/785/4933>

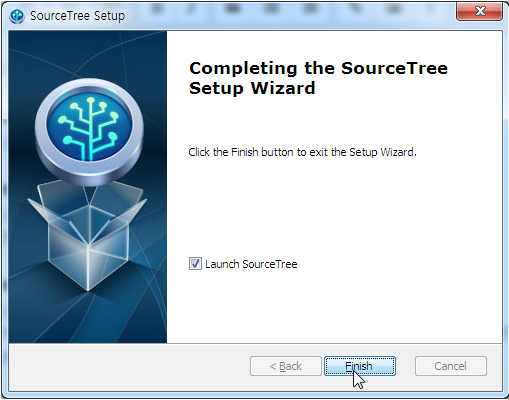
# **SourceTree 설치 및 사용방법**

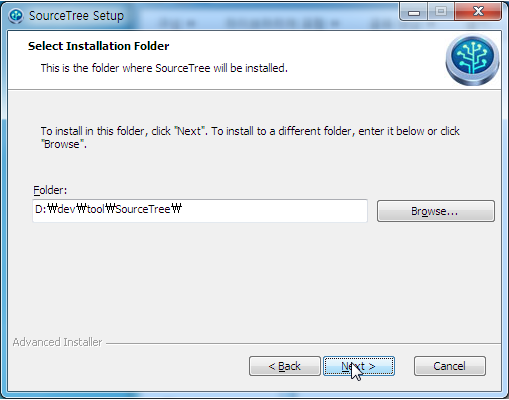
## **SourceTree install**

SourceTree는 윈도7 이상, Mac OS X 10.7 이상만 지원한다.

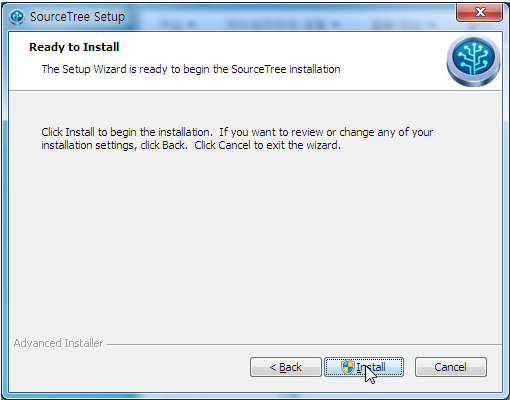
<https://www.sourcetreeapp.com/download/> 다운로드 페이지로 이동하면 자동으로 다운로드를 시작한다. 설치과정은 이미지대로 진행한다.

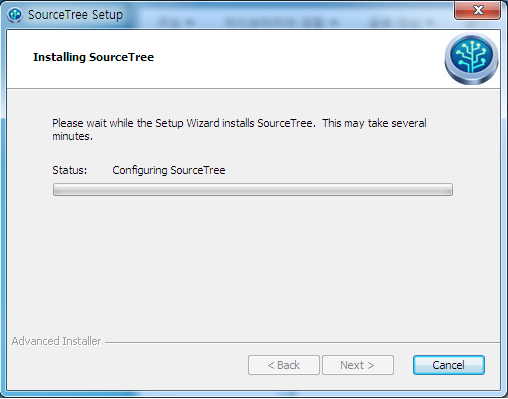


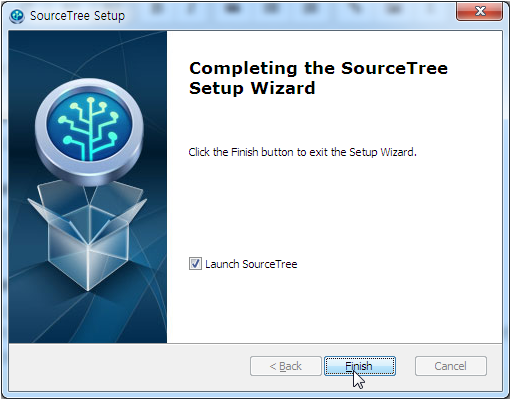


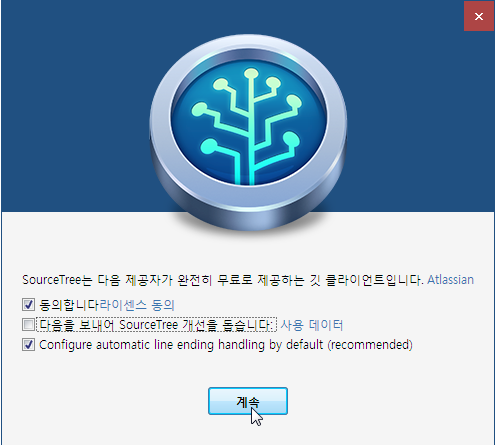


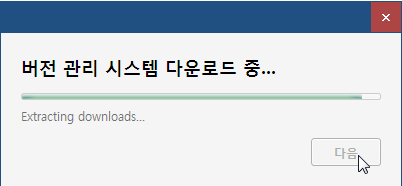
설치 경로를 설정합니다.

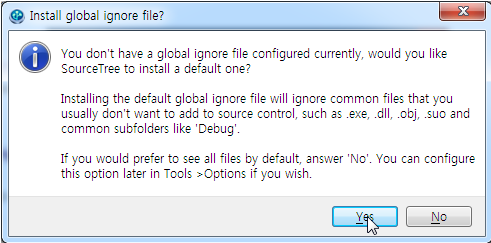




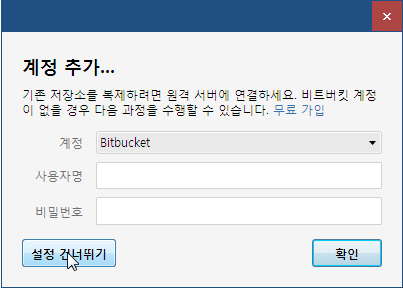








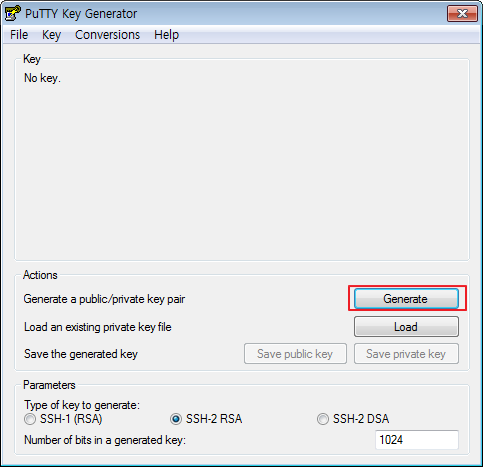
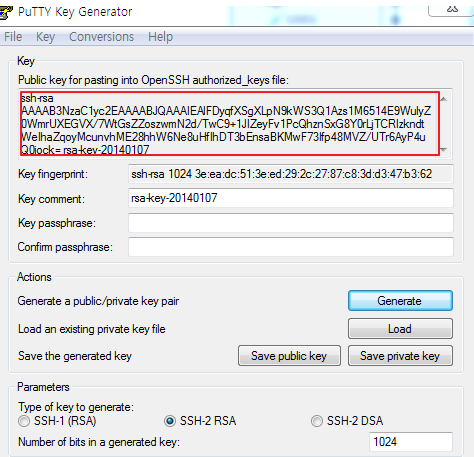
전역 ignore 파일을 사용할 것인지를 묻고 있다. Yes를 선택하면 .exe, .dll, .obj, .suo, .bak 등의 일반적으로 VCS에 올리지 않는 파일을 무시하고 SourceTree에서 보이지 않도록 한다. 도구 > 옵션 > Git 메뉴에서 전역 ignore 파일을 수정할 수 있다.



GitHub 같이 외부 Repository를 사용할 경우 ID/PW를 넣는 곳으로 상관없는 부분이기 때문에 건너뛴다.

## **SourceTree SSH Key 설정**

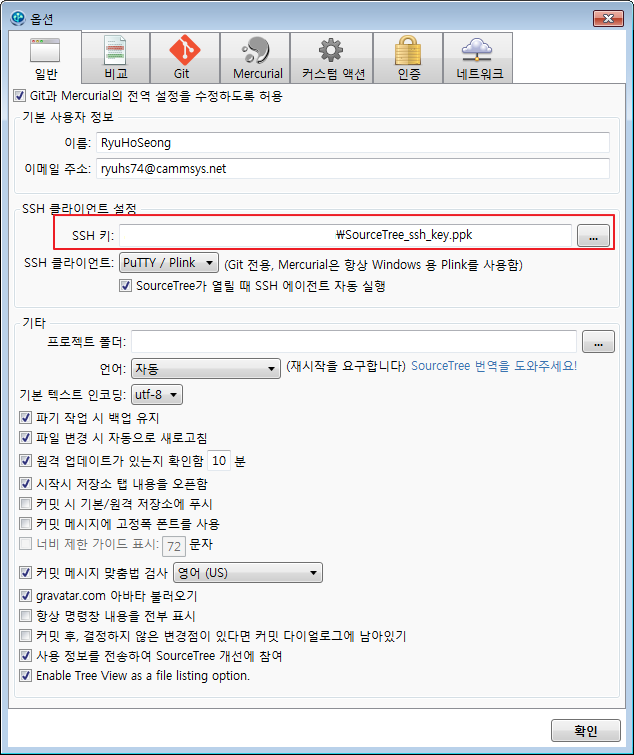
Git 저장소는 SSH Key를 등록하여 인증과정을 간단하게 할 수 있다. SourceTree에서 SSH Key 생성을 하기 위해서 ‘Tools -> Create or Import SSH Key’ 메뉴를 선택하면 ‘PuTTY Key Generator’ 창이 나온다.

‘Generate’ 버튼을 클릭 후 마우스를 오버해서 움직여야 복잡한 Key Code가 생성된다.

SSH Key가 생성되면 GitLab 에 로그인하여 생성된 Public Key를 복사하여 등록해 준다. GitLab에 로그인 후 우측 상단의 기어 모양의 ‘profile settings’ 버튼을 클릭 -> 좌측 메뉴에서 ‘SSH Keys’ 버튼을 클릭( SSH Key 페이지로 이동 후 ) -> 우측 상단의 ‘Add SSH Key’ 버튼 클릭하여 복사한 SSH Key를 설정 후 저장한다. 여러 개의 Client에서 SSH를 생성 했을 경우 여러 개의 SSH를 등록하여 관리 할 수 있다.

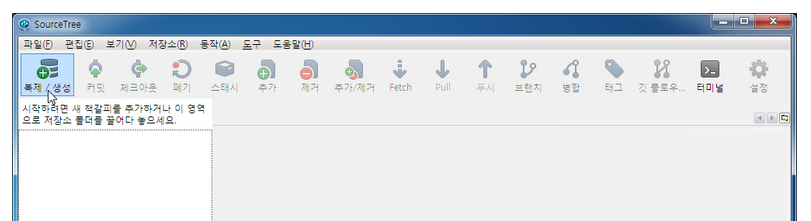
‘Save private key’ 버튼을 클릭하여 생성한 SSH Key를 Local에 ppk파일로 저장한 후 ‘Tools -> Optioin’에서 등록하면 된다.



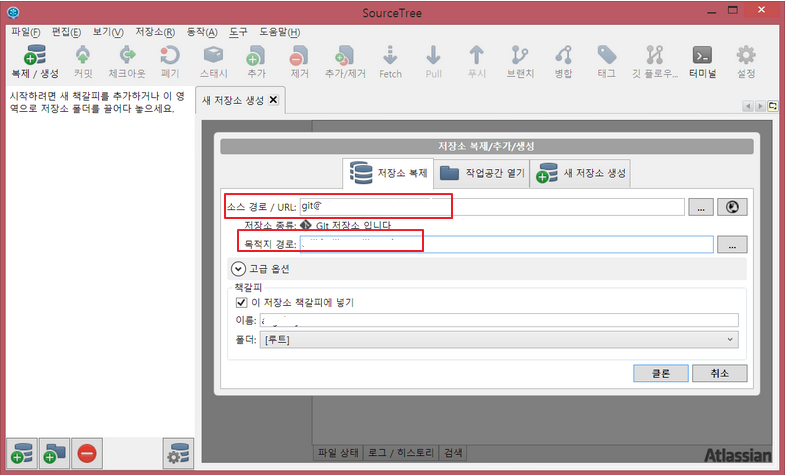
이 후로 SourceTree를 실행하면 로그인 없이 Git 저장소에 접근할 수 있다.

## **SourceTree Clone 생성**

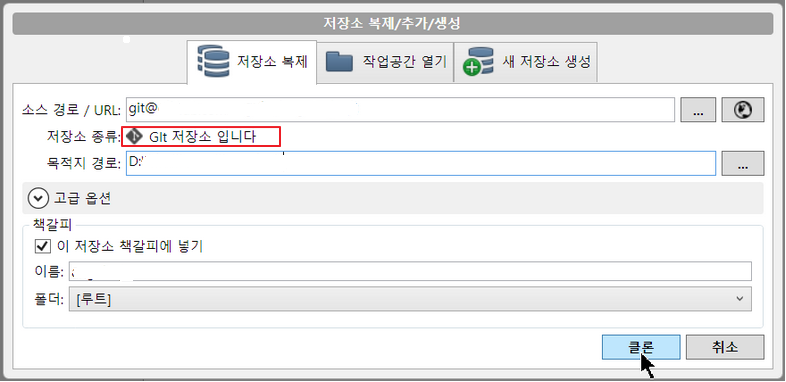
SVM의 Check Out과 같은 기능으로 Git Repository의 소스 코드를 다운로드한다. SSH Key 설정 후 GitLab 에 있는 저장소에 접근하여 해당 프로젝트 소스를 가져온다.



툴바에서 ‘복제/생성’ 메뉴를 선택한다.



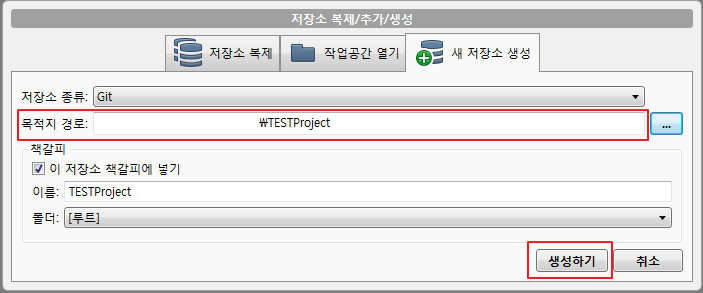
소스 경로에는 Git이 경로를 설정한다. GitLab 에 접속하여 Member로 참여하는 Project의 경로를 복사하여 설정한다. 목적지 경로는 Local 저장소이다.



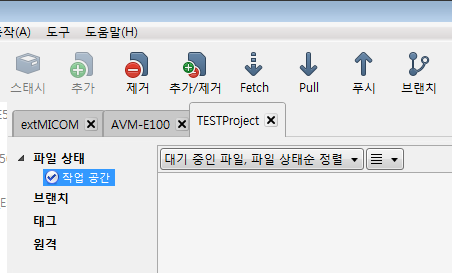
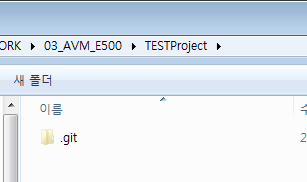
소스 경로의 URL이 올바른 URL이면 ‘Git 저장소 입니다.’ 라는 코멘트를 볼 수 있다. 클론 버튼을 클릭하면 원격저장소에서 로컬저장소로 Project 저장소를 내려받는다.

## **SourceTree New Repository 생성**

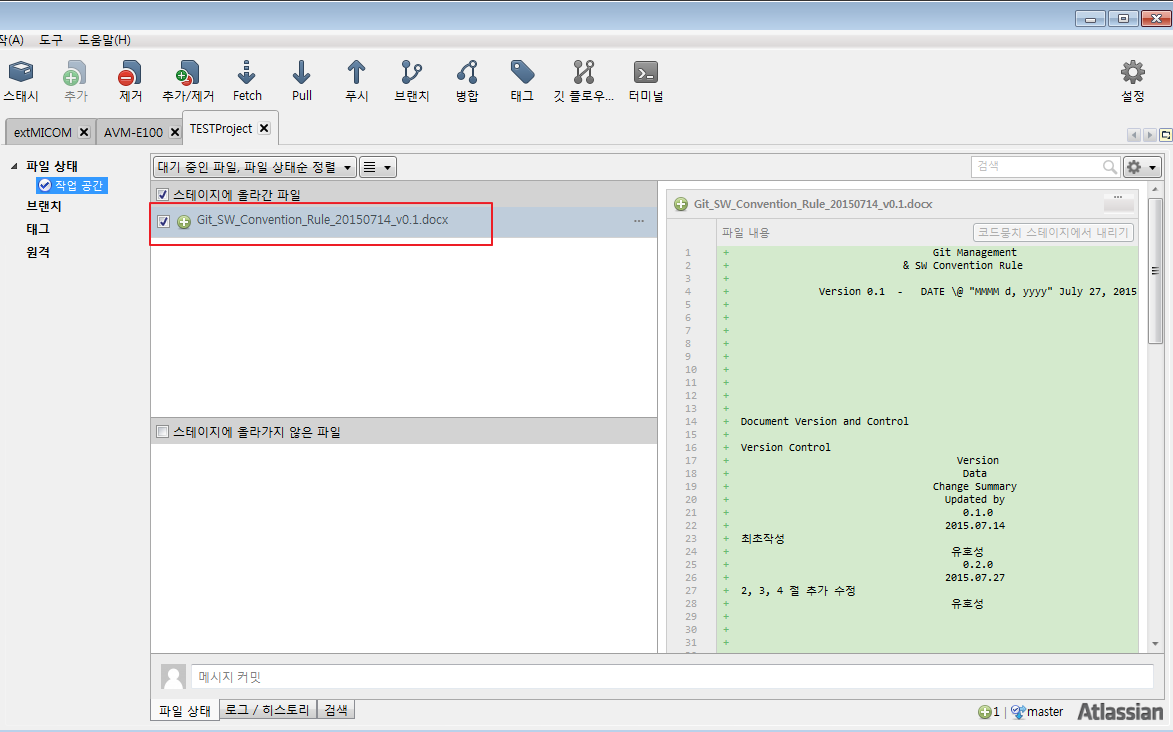
툴바에서 ‘복제/생성’ 메뉴를 선택 후 ‘새 저장소 생성’을 선택한다.

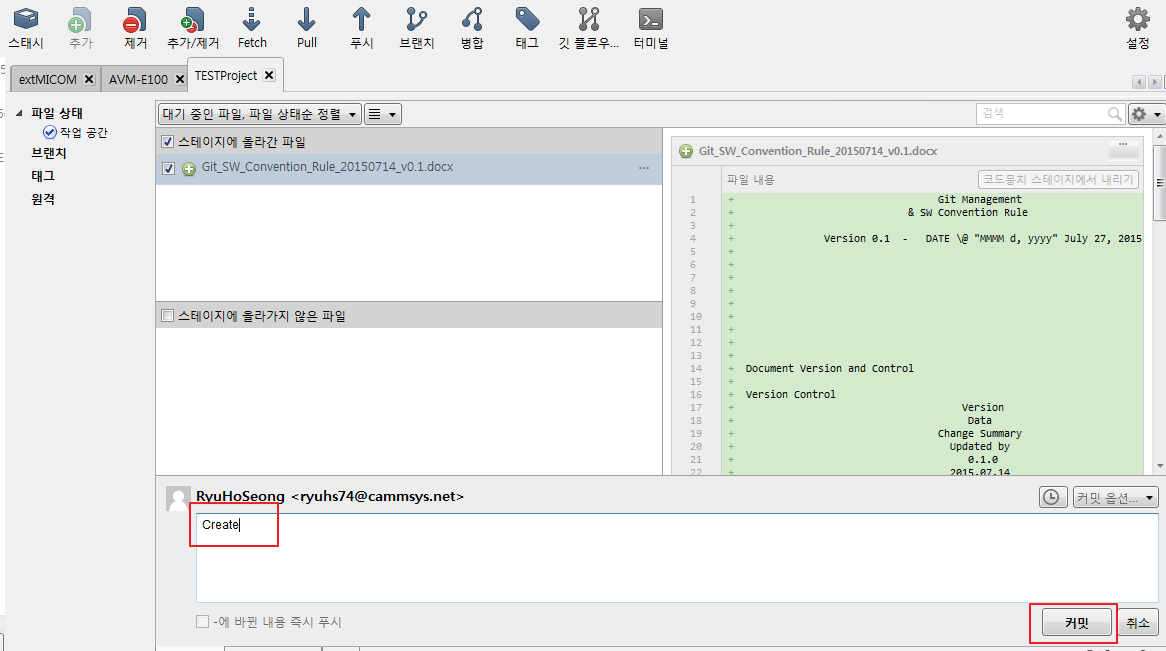


목적지 경로는 Local 폴더의 위치이다. 폴더가 없을 경우 생성한다. 폴더를 선택하면 이름은 자동으로 나온다. ‘생성하기’ 버튼을 누르면 Local 폴더에 GIT 저장소가 생성된 것을 볼 수 있다.

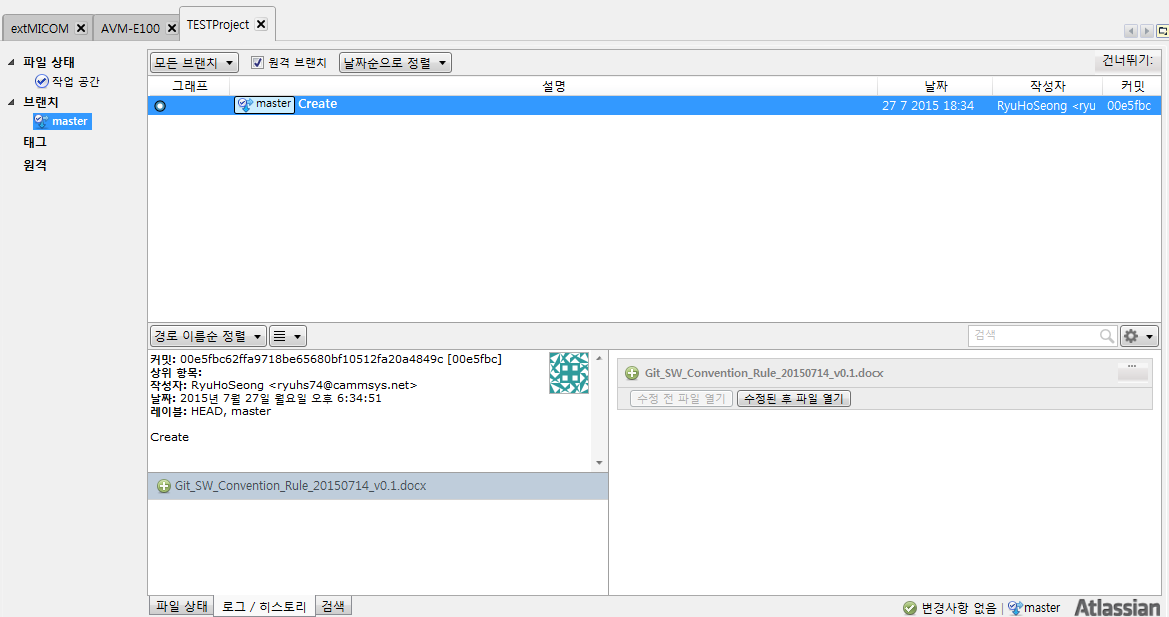
 

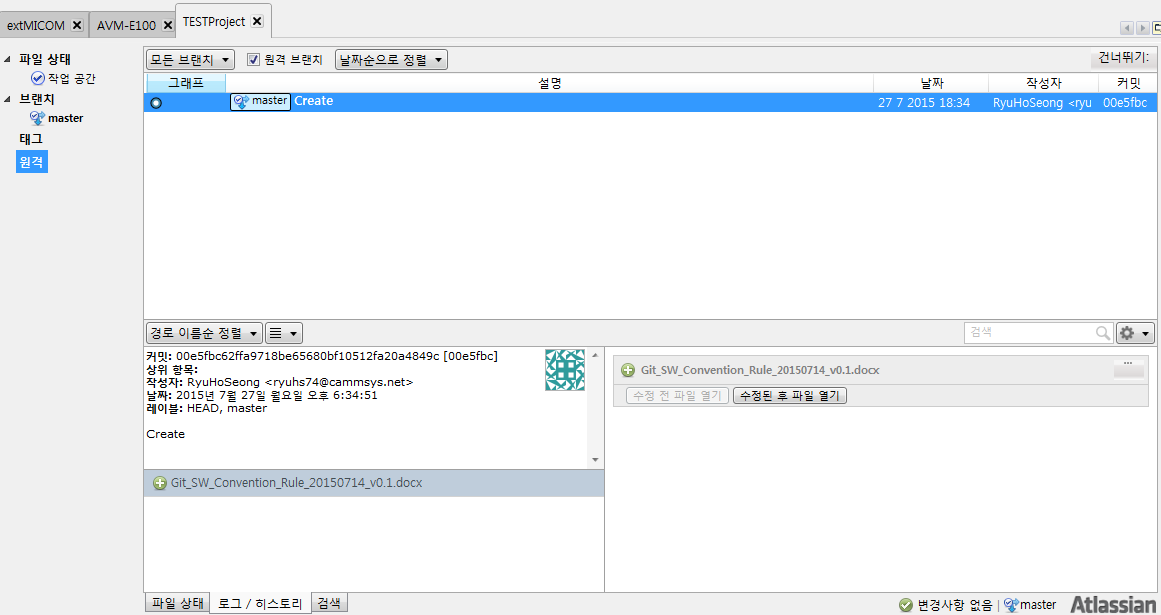
Git 정보가 생성된 폴더에 필요 파일들을 복사해 놓은 후 아래 ‘스테이지에 올라가지 않은 파일’ 들에서 필요한 파일들을 선택한다. 선택된 파일들은 ‘스테이지에 올라간 파일’ 윈도우에 표시된다.



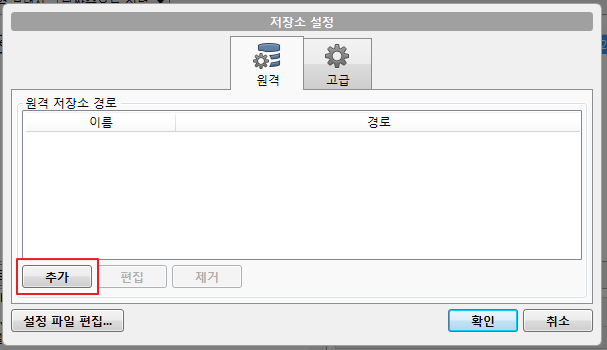


왼쪽 윈도우의 ‘작업공간’을 선택한 후 오른쪽 마우스 클릭->커밋 메뉴를 선택하면 Local작업 공간의 파일을 Local 저장소에 커밋하게 된다. Local 저장소에 커밋 후 master라는 메뉴가 생긴 것을 볼 수 있다.

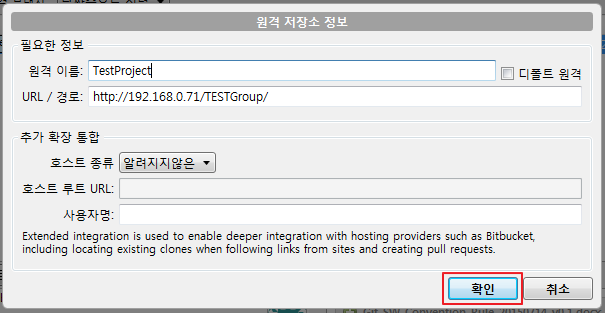




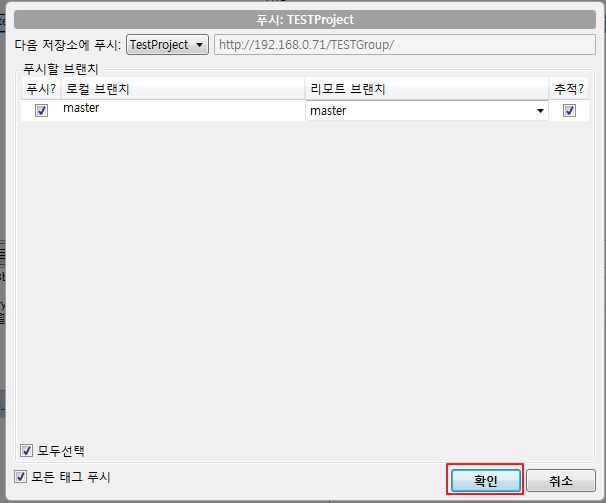
왼쪽 윈도우에서 ‘원격’을 선택 오른쪽 마우스 클릭 -> 새 원격… 을 클릭하면 원격지 저장소를 설정하는 윈도우가 나온다.



추가 버튼을 클릭한다.



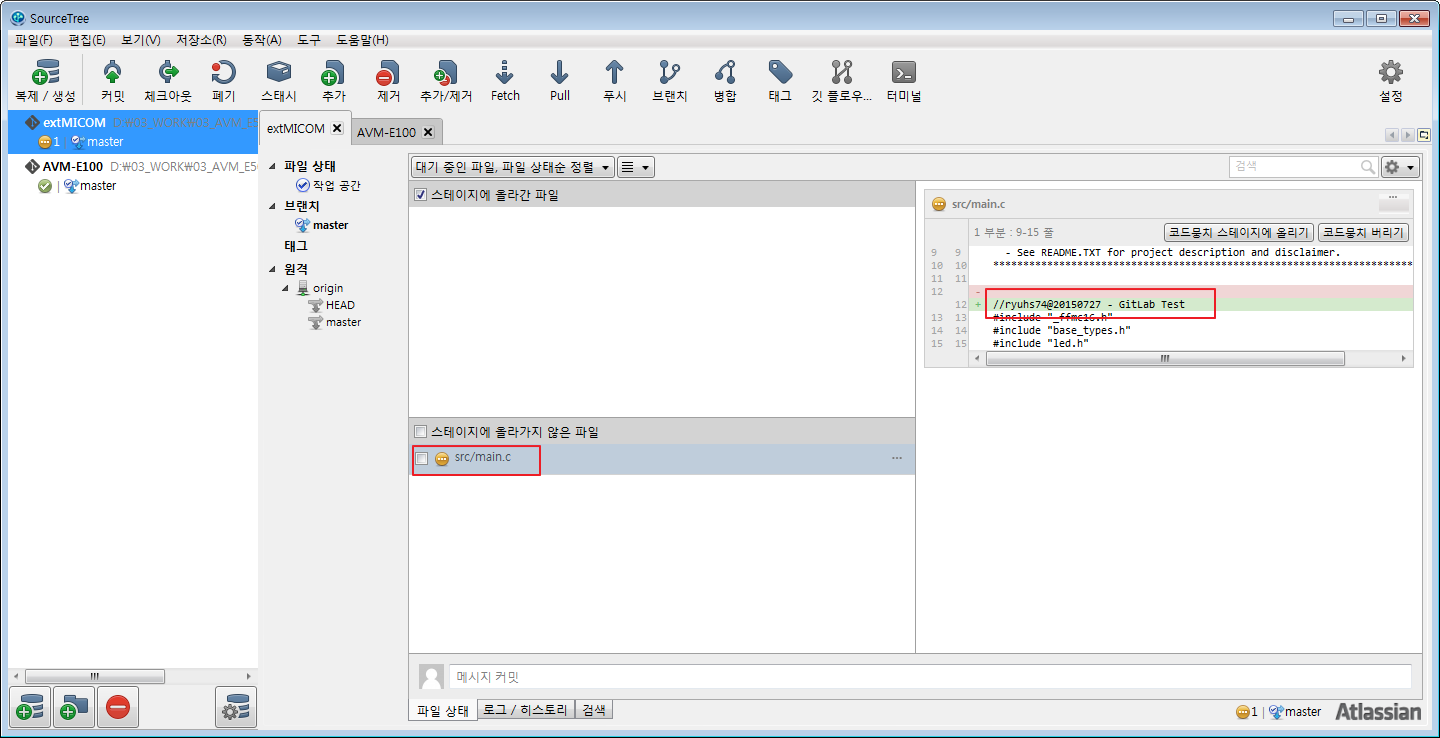
원격 이름은 GitLab의 Project이름이다. 이 프로젝트 이름이 동일해야 한다. URL / 경로는 GitLab의 Project 페이지의 왼쪽 메뉴에서 Project Setting을 선택하면 나오는 페이지에서 ‘Rename repository’ 의 ‘Path’를 복사하여 넣는다.

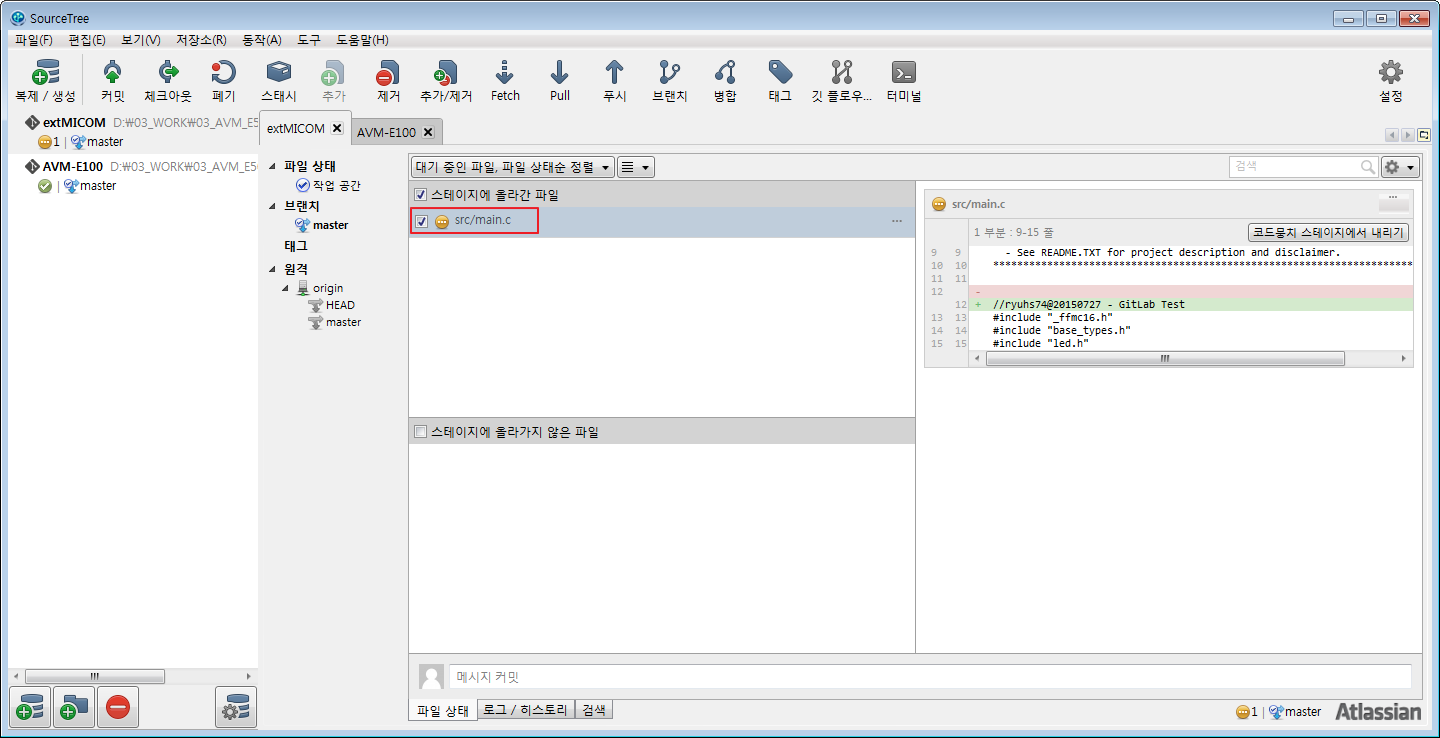


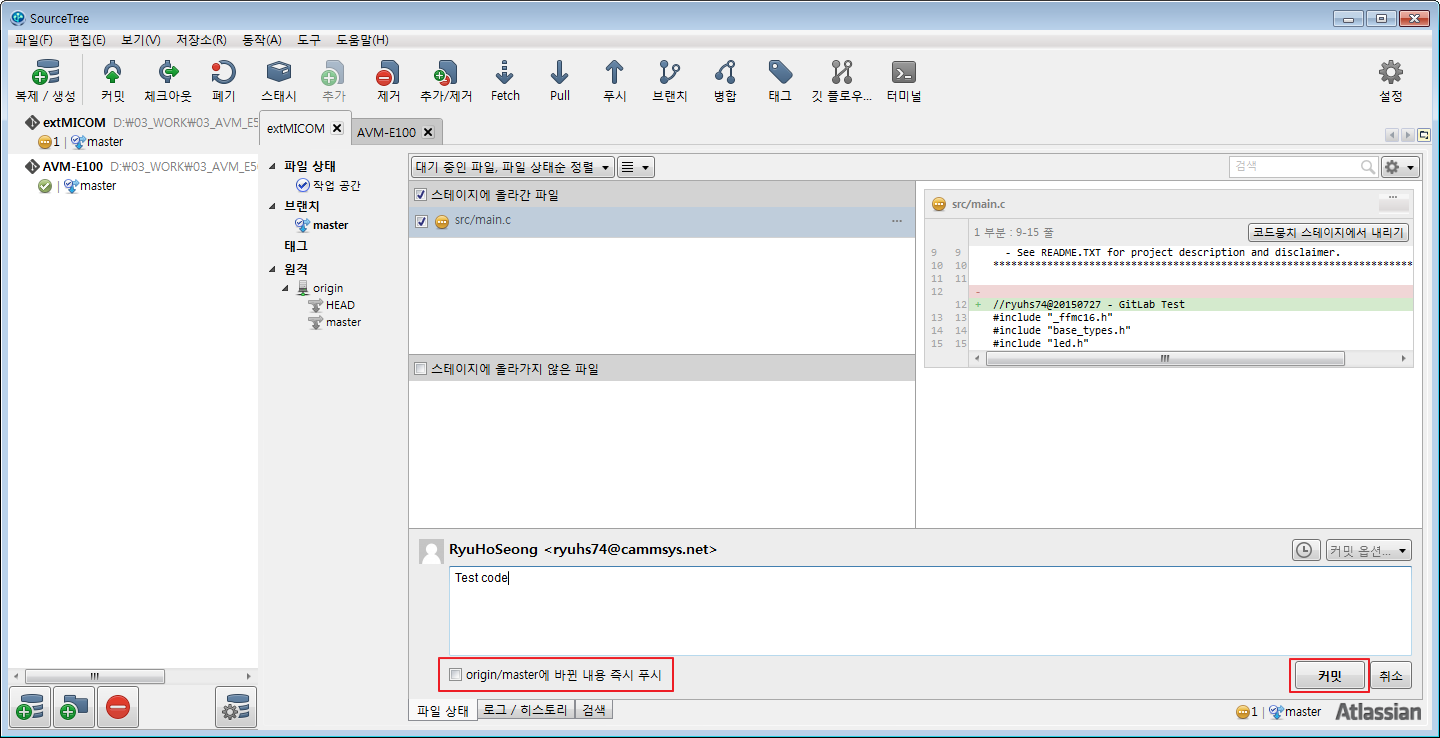
확인을 클릭하여 종료한다.

## **commit & push**

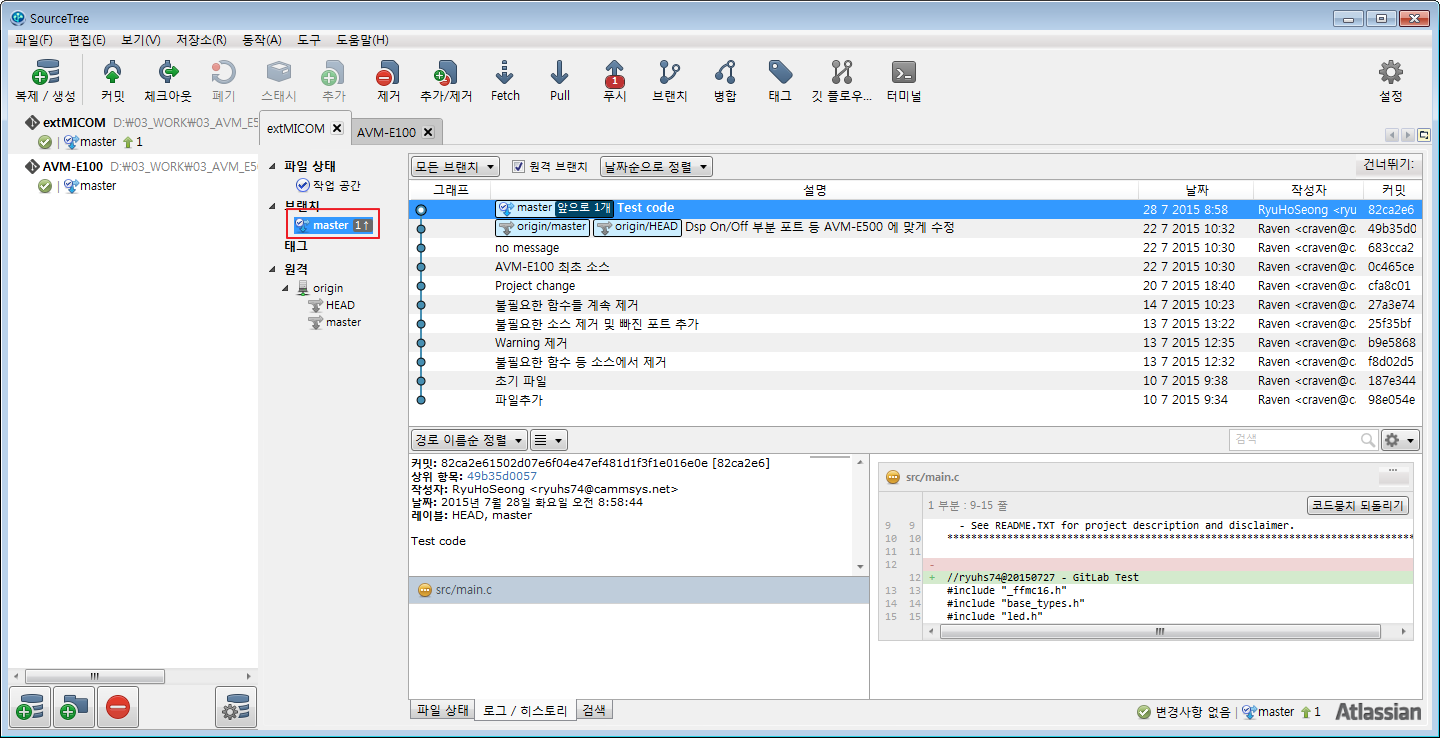
Git은 SVN과 다르게 commit하기 전에 stage라는 한 단계를 더 거친다. commit하기 원하는 파일들을 stage를 거친 후 commit을 진행한다.



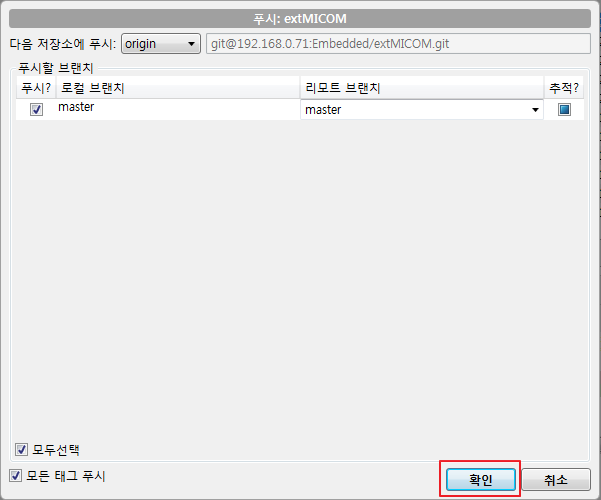




commit 할 때 ‘origin/master에 바뀐 내용 즉시 푸쉬’ 를 선택하면 commit과 동시에 원격지 Repository에 push한다.



master에서 마우스 오른쪽 버튼을 클릭 후 push를 클릭한다.



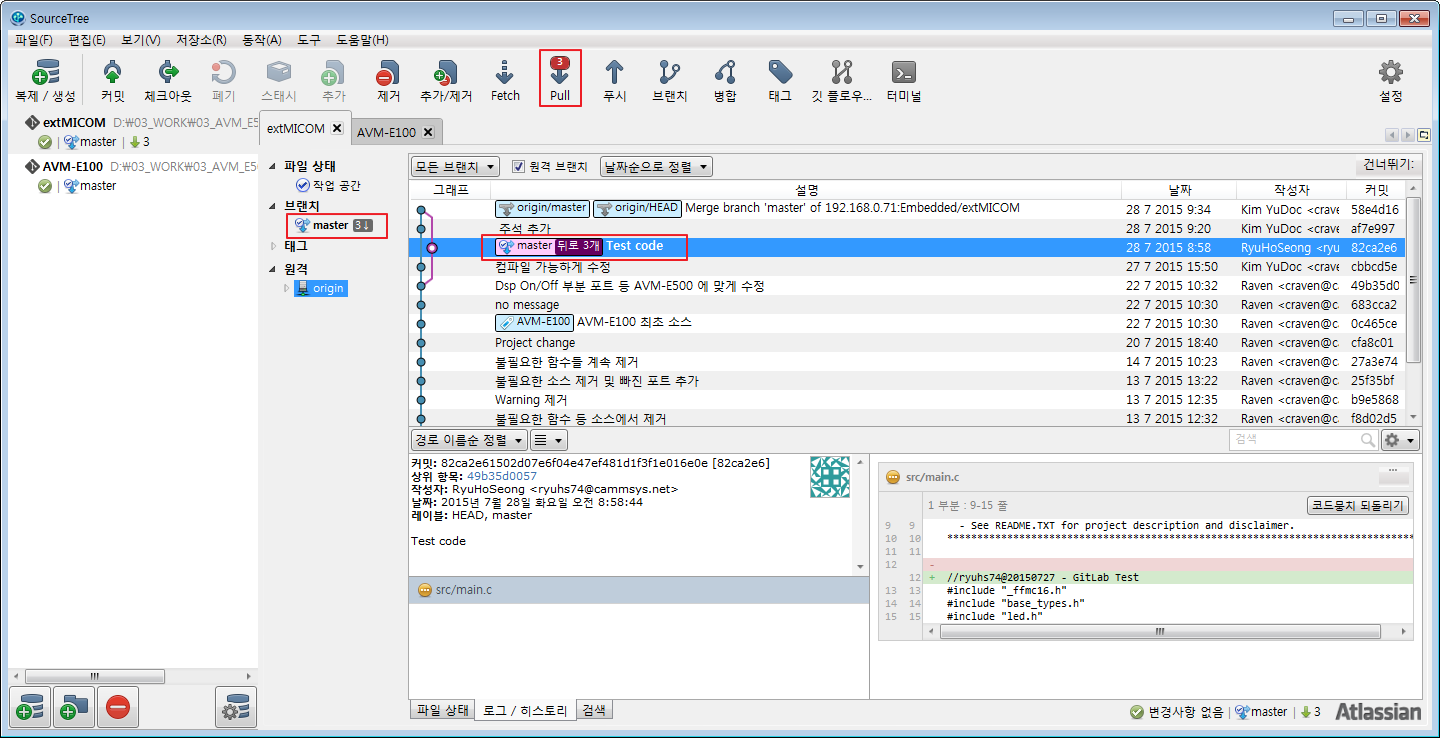
확인을 클릭하면 원격지 Repository로 push한다.

## **pull & merge**

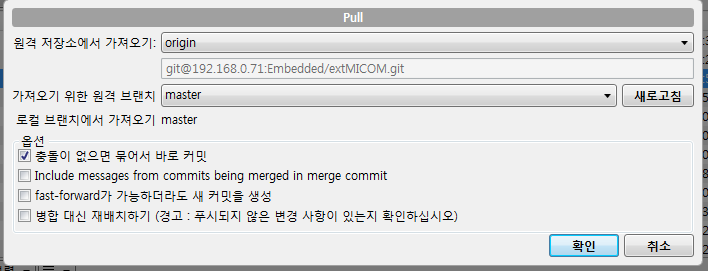
Git에는 Fetch와 Pull이 있다. 이 둘의 공식 설명은 Fetch는 원격서버의 정보를 가져오지만 merge를 하지 않고, Pull은 정보를 가져와서 merge를 한다 라고 되어 있다.

즉 Fetch는 원격서버의 정보를 새로고침 하는 것이고 Pull은 원격서버 정보를 로컬에 반영하는 것이다.

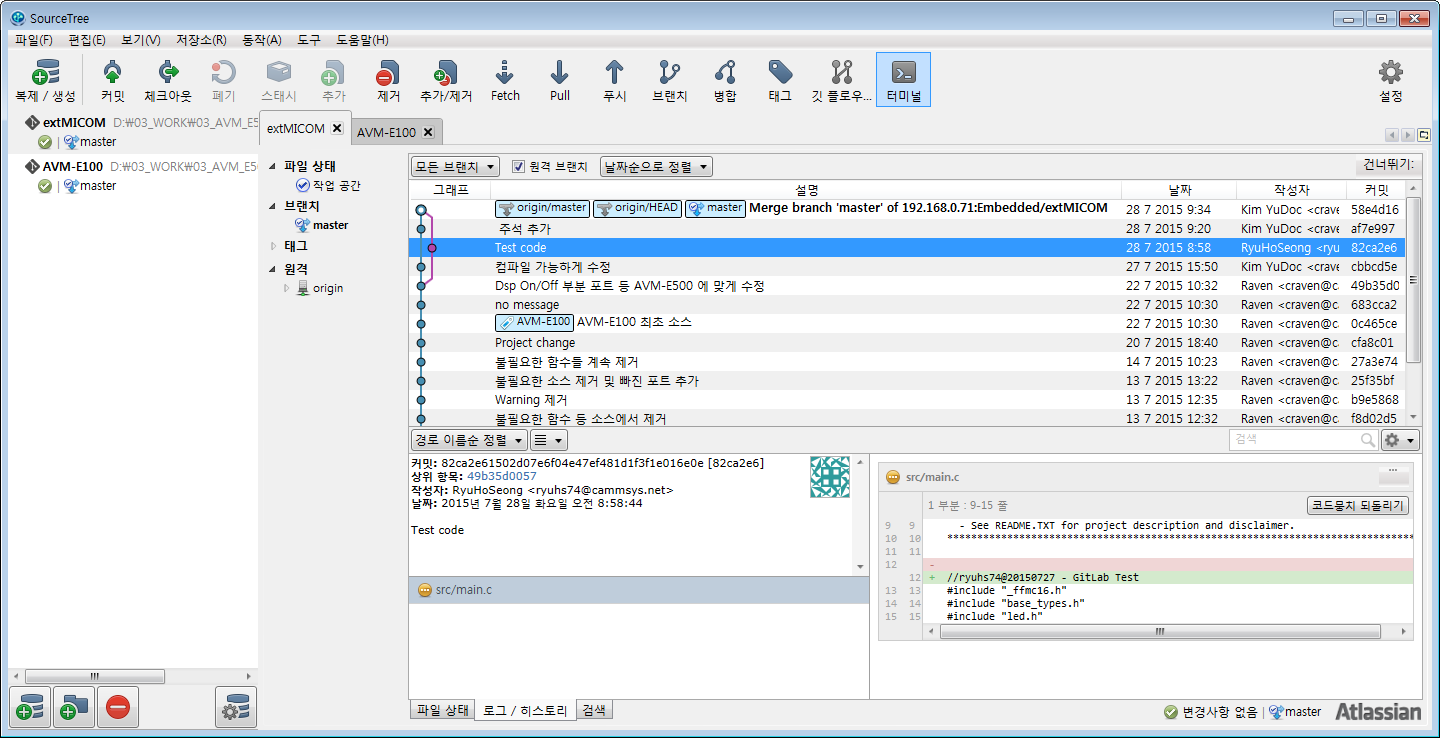
원격지 Repository에 변화가 생기면 Tool bar의 ‘Pull’ 아이콘에 명시된다. Master, Local Repository가 원격지 Repository보다 3개 뒤로 차이점이 있다는 뜻이다.(화살표가 아래로 향해 있다)



‘Pull’ 버튼을 클릭 한 후 Pull 윈도우에서 확인 버튼을 클릭한다.



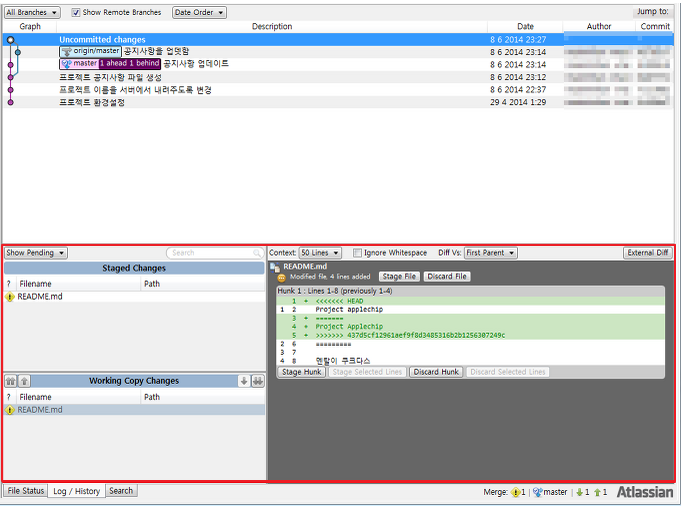
원격지 Repository와 Local Repository가 Merge 된 것을 알 수 있다.

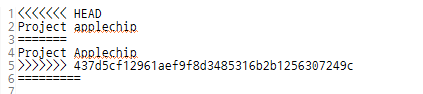


* 주의사항 : Local Repository를 Push하기 전에 항상 Pull 진행해서 Local과 원격지 Repository의 Merge를 먼저 수행해야 한다.

## **충돌 (Conflict)**

* 충돌: 업데이트 과정에서 작업중인 파일이 저장소에서도 수정되어 충돌이 발생하는 경우이다.





Conflict 이 발생하면 ======= 윗부분의 HEAD 까지가 현 로컬에 작성되어 있는 부분이고

밑부분의 commit 해시 값 까지가 원격에서 가져온 코드이다. 여기서 원하는걸 선택하고 버릴걸 지운후 기존에 Commit 하듯이 하면 된다.

SourceTree에 관한 추가 사용법은 아래 URL에서 확인 가능하다.

<https://opentutorials.org/course/1492/8035>

토토이즈를 이용한 GIT 사용방법도 참고

<http://backlogtool.com/git-guide/kr/>

# **GIT Repository**

## **위치 및 권한**

위치는 GitLab에 접속하면 알 수 있기 때문에 따로 기술하지는 않는다.

|  |  |
| --- | --- |
| Group | Repository 권한 |
|  |  |
|  |  |

* 관리자 권한은 연구원 모두에게 부여할 것.
* User Name은 FULL Name을 원칙으로 할 것

# **GIT Convention Rule**

- Project 하위 폴더에 trunk, branches 와 tags 폴더 운영

|  |  |  |
| --- | --- | --- |
| 항목 | | 내용 |
| Project/ | trunk/ |  |
| branches/ |  |
| tags/ |  |
| * Version rule * v[MajorVersion].[MinorVersion].[PatchVersion] * MajorVersion: 전체적인 build 환경에 변화된 사항 적용, SDK version 변경시, 와 같이 큰 틀의 변경이라 판단될 경우 합의하여 major 변경 * MinorVersion: Major version release 이후 각종 버그 해결된 version을 release 할 경우에 합의하여 minor version변경 * PatchVersion: Minor version 에 대해서 patch를 했을 경우,  version up | | |

-    Checkout / Export 차이를 이해할 것 (본 문서 ‘2장 SVN 설치 및 사용법’ 참조.)

-    소스를 수정하거나 Commit하기 전에는 update 명령어를 사용하여, 다른 작업의 수정사항을 반영하여 작업하기. (작업 시간이 오래 걸릴 경우, 수시로 할 필요가 있습니다.)

-    Commit시 수정자, 변경된 내용을 영문으로 표기하기

-  e.g. 2013/11/20, yscho - Add a hello function in hello.c and hello.h

-  e.g. 2013/11/20, yscho - Add New files: aaa.c, bbb.c, ddd.c, header.h

-  변경된 파일이 여러 개 있고, 서로 관련 있는 파일을 한꺼번에 Commit하기 (파일 단위로 올리기 금지, 이유는 쓸모없이 version up이 되고, 장기적으로 파일 관리시 불편을 초래함.)

-  파일에 Lock을 경우, 그 이유와 작업 예상시간을 로그에 저장하고, 작업이 완료되면 반드시 unlock하기 (담당자 부재가 생겼을 경우 불편함을 초래할 수 있음.)

-  Commit한 후, 중요 파일이라 판단이 들면, 반드시 수정 파일이 정상적인지 새로 update 하여 build 하고, 테스트 하기.

# **Software Convention Rule**

## **License notice**

- 최초 파일 및 update 파일 License notice 구문 넣기

e.g.)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* Module Description: i.e. test program to demonstrate IP (HTTP/RTP/UDP) streaming functionality  
\* of Live QAM, IP or Playback data from Disk.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

상기 License notice에서 Revision History: 항목은 삭제했음. 상단에 구성되므로, History가 늘어 날수록, 실제 coding 된 내용이 계속 아래로 밀어나는 단점이 있음.

## **Reference Coding Rule**

* Required 를 반드시 따르도록 하고, advisory는 되도록 준수한다.
* 제시된 규칙은 추후 프로젝트 상황에 따라 유동적으로 적용하도록 한다.
* 자동화 정적 코드 분석 도구를 이용하여 검증한다.
  + 대표적 도구: LDRA, PRQA, Parasoft 등.

## **Define 관련 사항**

* 의미를 명확히 해야함.
* Test 용도는 임의로 define하지 말고, 가급적 기 사용된 것을 이용하고, 따로 정의 해서 사용할 경우, 기 사용된 것과 유사하게 이용할 것.

#define ENABLE\_TEST\_[Feature]

i.e. #define ENABLE\_TEST\_CODES

* H/W define은 prefix -> #define PF\_[HW Name]

i.e. #define PF\_HDMI

필요에 따라 .naming은 \_[Feature] 추가할 수 있다.

i.e. #define PF\_HDMI\_DISABLE\_CEC

* Define 명은 되도록 통용되는 축약어가 아니면 줄이지 말 것.
* Feature/Support/Enable 등 비슷한 유형의 단어를 중복해서 사용하지 말 것.

i.e. #define FEATURE\_OSD\_FONT

## **Macro 표기법**

* Use all capital letters

[CAPITAL LETTER]\_[CAPITAL LETTER]\_[CAPITAL LETTER]

     /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
      \* Local Macro Declarations  
      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
     #define LOCAL\_GET\_CPU\_LOAD() Load\_getcpuload()  
     #define LOCAL\_USE\_CIO  (1)  
     #define SIMPLE\_MEMORY\_MANAGEMENT

## **함수 표기법**

* 함수명을 보고 어느 파일에 위치한 것인지 직관적으로 알 수 있게 할 것
* 이전 정의된 함수명과 유사성을 가질 것.
* 특정 define에 따라 function parameter를 추가 혹은 변경해야 하는 경우가 있는데, 우선 이전 parameter를 공용으로 사용할 수 있는 방법을 모색 후 불가하면 추가할 것. .

[LAYER]\_[FUNCTION NAME]\_[function]

e.g.) APP\_SYSTEM\_checkMemStats();

## **변수 표기법 (헝가리안 표기법)**

 : 작성시 다소 불편하지만, 변수의 가독성 및 명확성을 확보함.

     e.g.)   UInt32 bucketNum;

            LOCAL\_MemorySegmentDescNodeHandle tempNode,currNode;

* 포인터 있는 변수인 경우(asterix 와 p 를 중첩)

      \*p

     e.g.) int \*pnCount

* 습관적이고, 일반화된 경우는 예외 -> 충분히 가독성을 확보했다고 보이면 상관 없음.

     e.g.) XDest, YDest, i, j

* 전역변수인 경우

     g\_[변수명]

* 포인터(\*) 및 참조(&) 표기는 변수 앞에 붙여서 선언

     e.g.) char \*szName, char &szName

* 구조체 맴버는 헝가리안 표기법을 사용하지 않고, 소문자로만 이용하고, 구분시 \_ 을 넣어서 구분

## **주석표기법**

: 항상 영문으로 표기

* 함수, Variable 선언 시작 전 표기법

 e.g.)  
     /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
      \* Global Variable Definitions  
      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* 함수 내 설명을 위한 한 줄 표기법 ‘ **//** ‘
* Debug 용 주석은 ‘ **// Author, YYYY.MM.DD** ‘ - Comment
* Debug 완료되면 주석을 삭제하거나, Comment 내용만 남겨 둘 것.

## **SW Coding 시 유의사항**

* 포인터, 핸들은 NULL 과 비교하고, 숫자는 0과 비교한다.
* 복잡한 구문은 임시 변수를 사용한다.
* Software editor 에서 TAB 은 4 space 간격으로 정의 사용
* 변수 또는 함수 선언시 TAB으로 정렬하지 않는다.
* 한 줄 조건문 및 반복문은 가능하나, 동일 구분에 여러 개를 혼합하지 않고 통일한다.

     e.g.)

          if (bSuccess == TRUE)

          {

               DoSomething();

          }

          else

          {

               DoSomething();

          }

* Compile warning은 안 생기도록 한다.