

깃허브(GitHub)

GitHub 개요



≡ 깃(Git) 시작하기

≡ Git의 필요성 인식

수정중인 파일을 이전 상태로 되돌리고 싶어요.

파일 리스트

다음은 파이썬 책을 집필하다가 작성한 백업본 파일 목록입니다.

어느 시점까지의 백업본 파일을 년월일 형식으로 저장한 예시입니다.

(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중) – 20230514.docx
(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중) – 20230720.docx
(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중) – 20230815.docx
(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중) – 20231125.docx
(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중) – 20231225.docx
(교재) 02.파이썬 모듈을 활용한 데이터 분석(집필중).docx

버전 관리



초안



2안



...



최종



진짜 최종

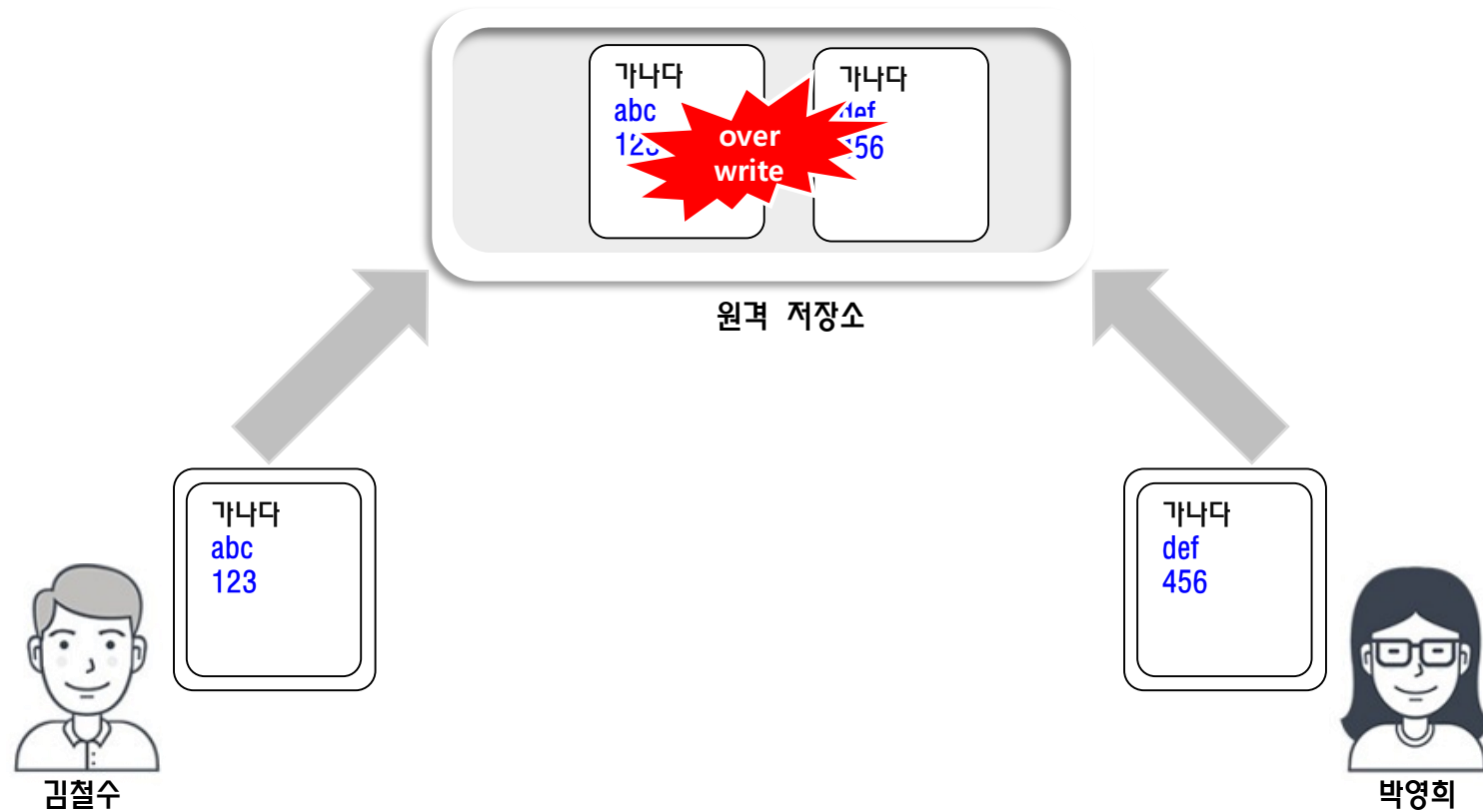


...

≡ 깃(Git) 시작하기

≡ Git의 필요성 인식

팀 프로젝트에서 현재 작업 중인 동일 이름의 파일에 대한 덮어 쓰기(overwrite) 실수를 범할 수 있습니다.



≡ 깃(Git) 시작하기

≡ 버전 관리 시스템

문서들을 작업한 다음 저장을 하게 되면 version이라는 개념이 생깁니다.

Version

문서 저장시 메모와 함께 스냅샷(사진)을 찍어서 저장해 둔 것을 의미합니다.
문서를 수정한 다음 저장할 때 마다 생성이 되는 어떠한 것입니다.

Version 관리 시스템

문서 수정후 저장을 하게 되면 version으로 저장을 수행합니다.
이미 저장된 version을 이전 내용으로 되돌릴 수 있습니다.

포함 내용

- 누가(who)
- 언제(when)
- 무엇을(what)
- 어떻게(how)

≡ 깃(Git) 시작하기

≡ Git으로 할 수 있는 일

Git이 제공하는 핵심 기능은 '버전 관리', '백업', '협업' 등이 있습니다.

버전 관리

문서를 수정할 때 마다, 누가, 언제, 무엇을 수정하였는 지를 매년 기록하는 일을 무척 성가신 일입니다. 버전 관리 시스템을 사용하면 이러한 고충을 덜어낼 수 있습니다.

개인적으로 보관하고 있는 소스 코드 및 기타 자료들에 대한 유실 문제를 고려한다면 반드시 '백업'을 수행해야 합니다. Github가 제공하는 '원격 저장소'를 사용하여 내 문서들을 백업할 수 있습니다.

백업 (backup)

협업 (collaboration)

여러 명의 개발자가 팀을 이루어 프로젝트를 진행하는 경우, '김철수'라는 개발자가 특정 파일 A를 원격 저장소에 업로드하면, 해당 문서를 다른 개발자 '박영희'가 다운로드 받아서 작업을 진행할 수 있습니다. 이와 같이 Git을 사용하게 되면 변경 이력을 이용하여 상호 협업을 수행할 수 있습니다.

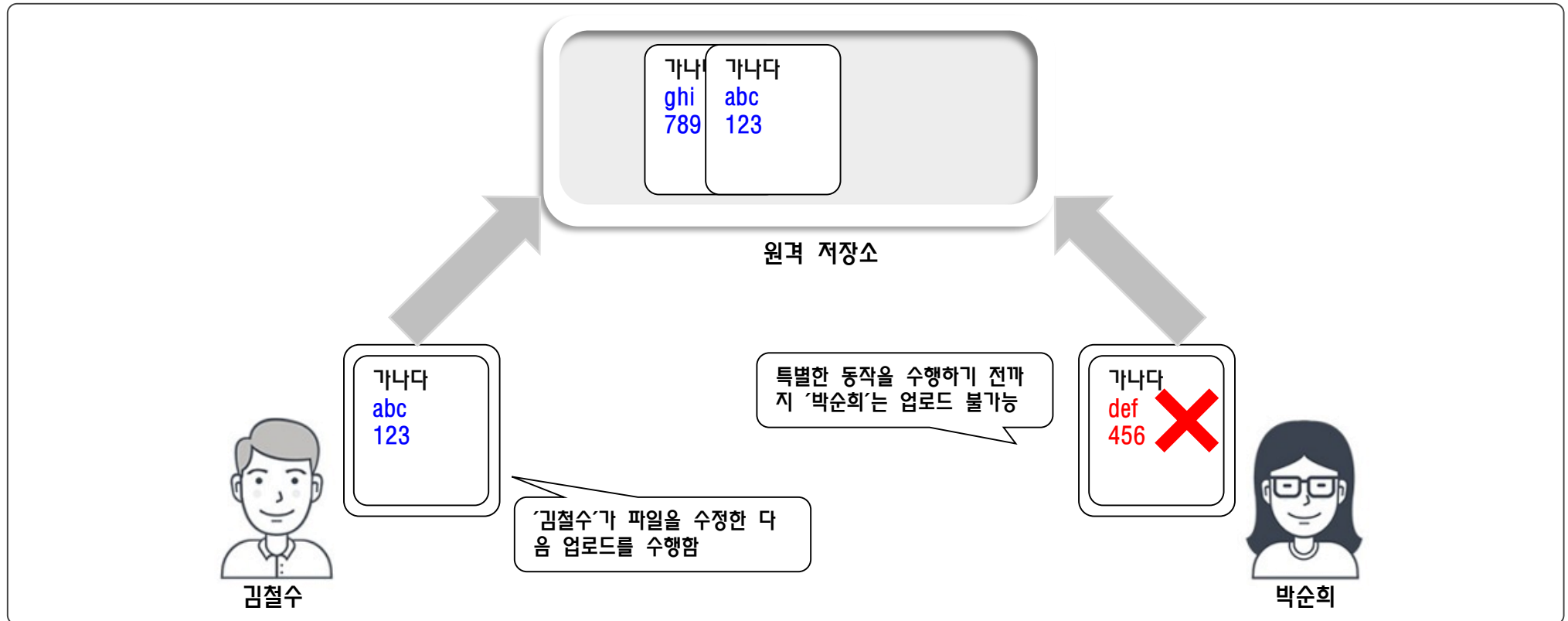
≡ 깃(Git) 시작하기

≡ Git을 이용하여 버전 관리하기

소프트웨어 개발 프로젝트를 위한 소스 코드 관리 서비스

소스 코드 열람 및 버그 관리, SNS 기능 포함

버전 관리를 통한 소스 코드에 대한 이력 관리

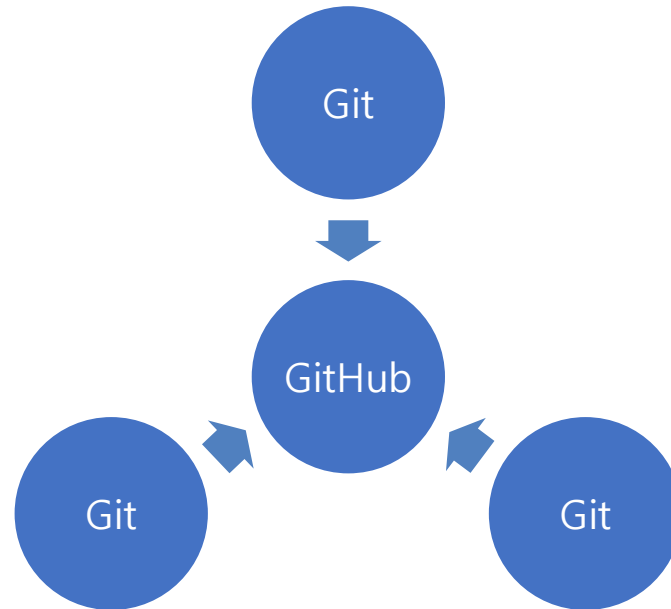


≡ 깃(Git) 시작하기

≡ Git과 GitHub의 차이

Git과 GitHub의 차이점에 대하여 간단히 살펴 봅니다.

항목	설명
Git	로컬 컴퓨터에서 관리하는 버전 관리 시스템(VCS : Version Control System) 소스 코드 변경에 따른 버전을 관리해주는 시스템
Github	클라우드 방식으로 관리하는 버전 관리 시스템(VCS) 오픈 소스는 일정 부분 무료로 저장 기능을 제공함

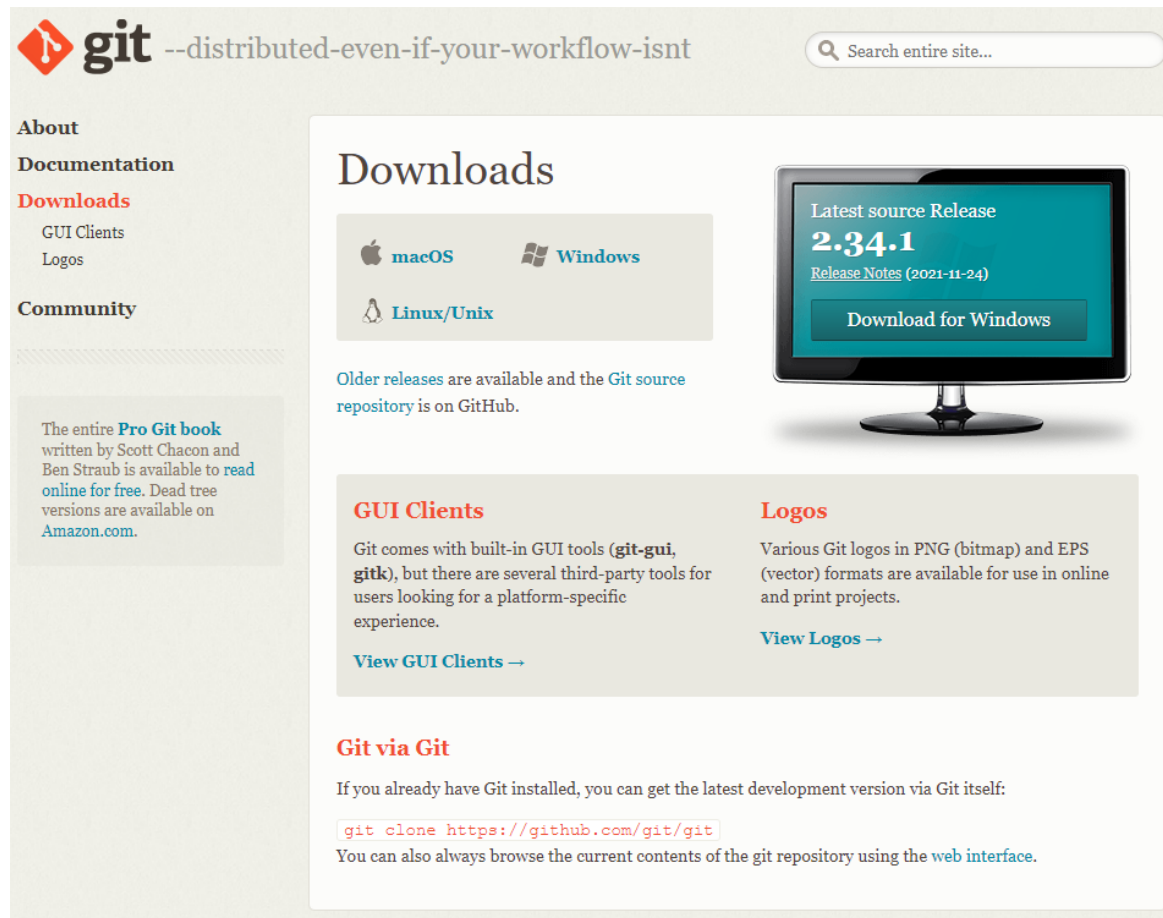


≡ 깃(Git) 시작하기

≡ git 다운로드 및 설치

참조 사이트) <https://git-scm.com/downloads>

사이트에 접속하고 Windows 링크를 클릭하면 설치 파일을 자동으로 다운로드 합니다.



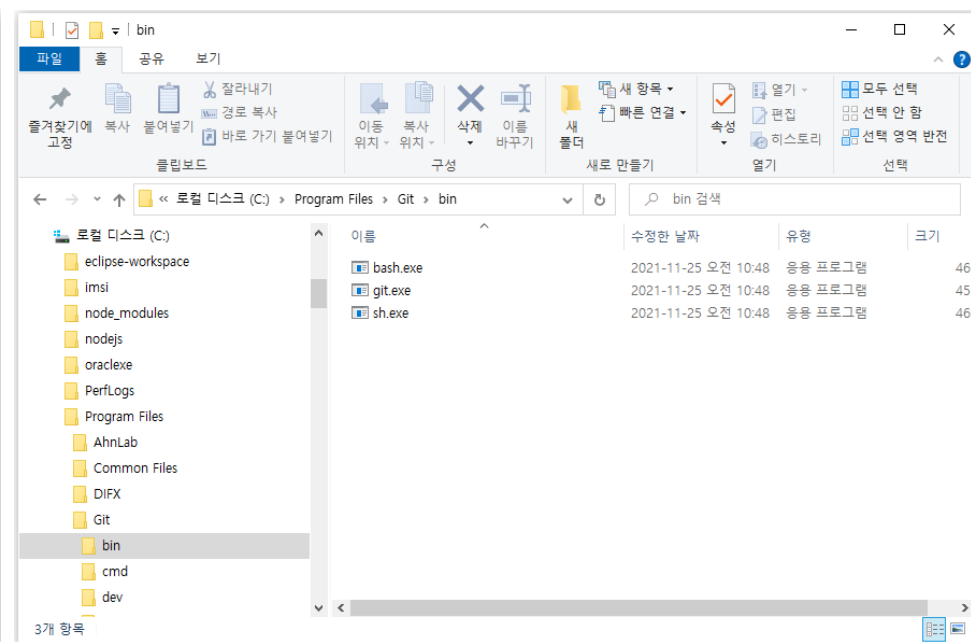
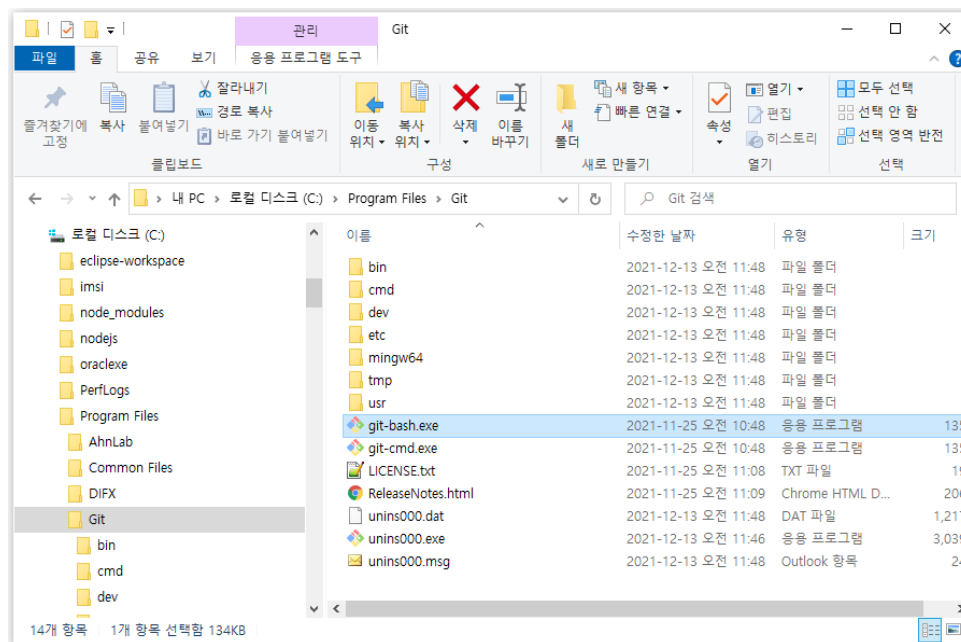
≡ 깃(Git) 시작하기

≡ git 다운로드 및 설치

참조 사이트) <https://git-scm.com/downloads>

설치가 되는 기본 경로는 "C:\Program Files\Git"이며, 설치 완료된 폴더 화면은 다음과 같습니다.

파일	설명
git bash(git-bash.exe)	깃의 터미널(cmd 창) 역할을 해주는 툴입니다.
git.exe	git 아래 bin 폴더에 존재하며, cmd 기반에서 사용하기 위한 명령어입니다.

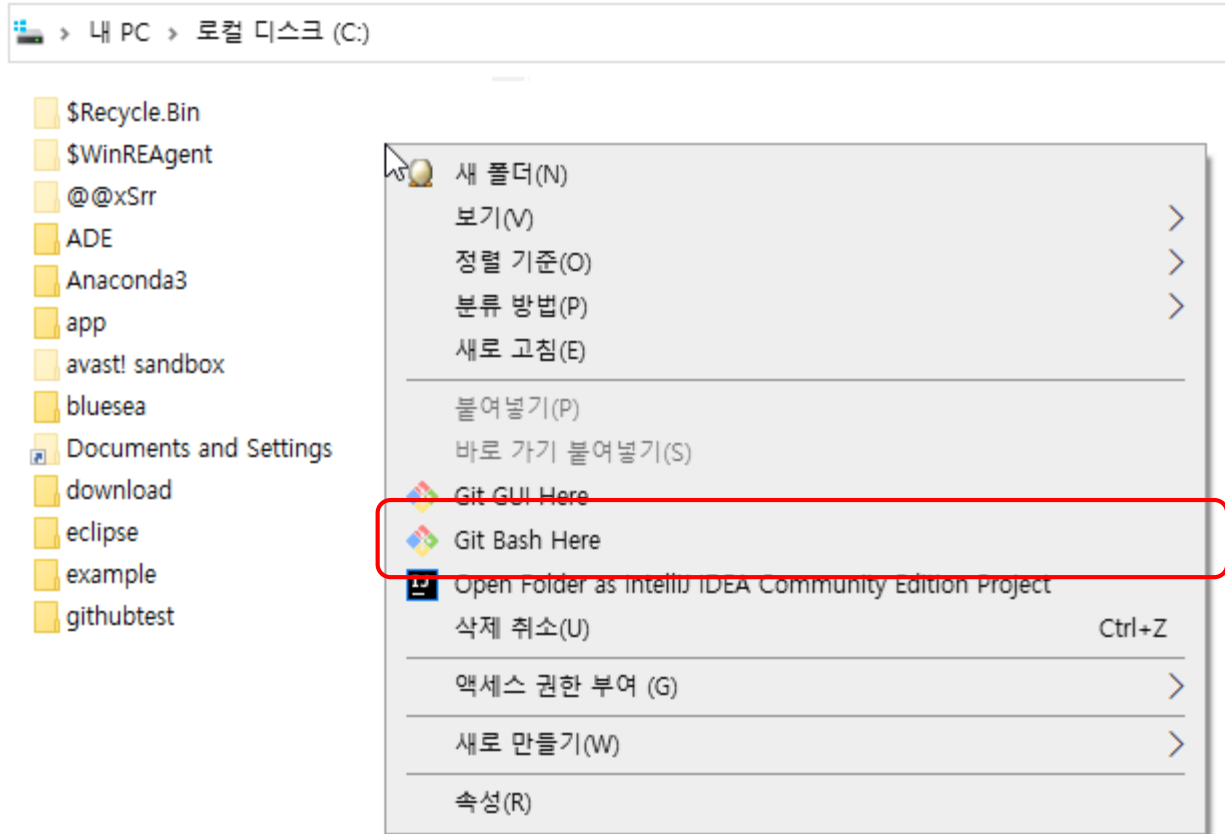


≡ 깃(Git) 시작하기

≡ git bash 실행하기

그림과 같이 C: 드라이브에서 git bash를 실행해 보겠습니다.

편의상 C: 드라이브에서 진행을 하였으며, 시작 위치는 임의의 위치여도 상관이 없습니다.

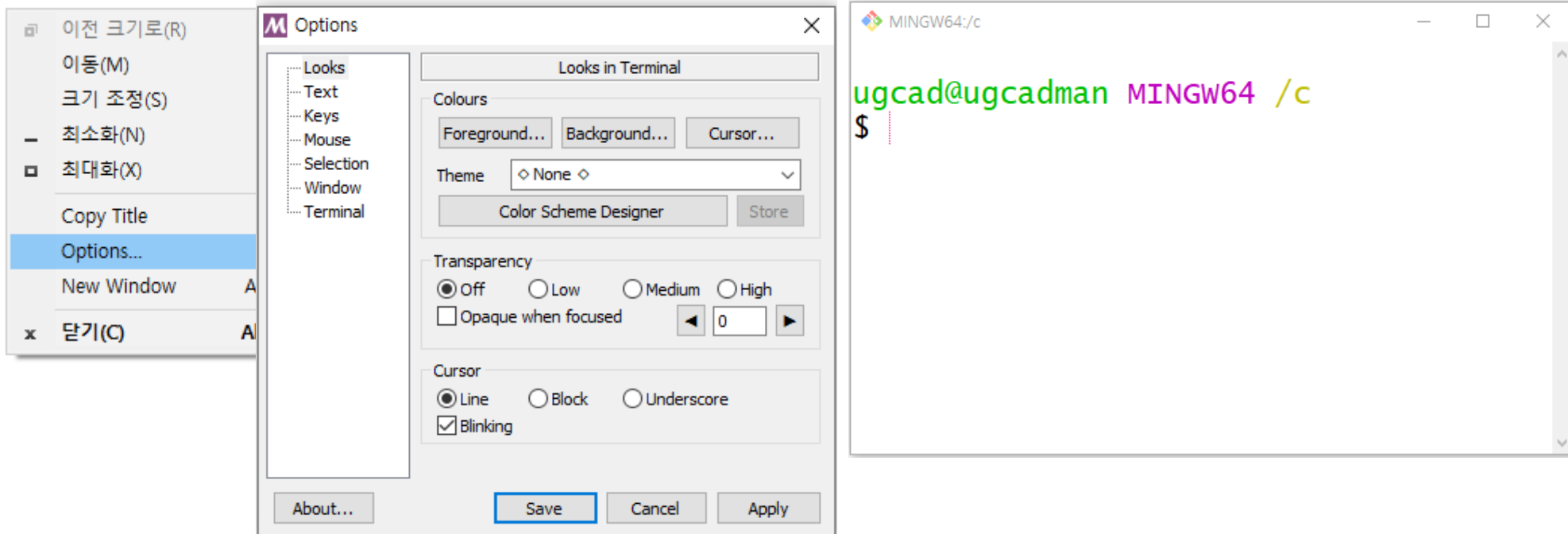


≡ 깃(Git) 시작하기

≡ git bash 실행하기

다음과 같이 git 명령어를 위한 터미널 창이 로딩됩니다.

Options 메뉴에서 배경 색상과 글자 색상을 변경합니다.
Ctrl 키와 마우스 wheel을 사용하면 터미널의 글자 크기를 조절할 수 있습니다.



≡ 깃(Git) 시작하기

≡ Git 환경 설정하기

Git에 사용자 이름과 메일 주소를 등록하기

git 설정하기

```
git config --global user.name "oraman"  
git config --global user.email "oraman@naver.com"
```

.gitconfig 파일

Git 버전 관리 시스템의 설정 파일로, 사용자의 Git 환경과 관련된 설정을 포함합니다.
사용자의 홈 디렉토리(~/.gitconfig)에 위치하며, 개인적인 Git 설정 정보를 저장합니다.
사용자 이름, 이메일 주소, 사용할 에디터 등과 같은 기본 설정 정보를 포함합니다.

.gitconfig 파일

```
[filter "lfs"]  
    process = git-lfs filter-process  
    required = true  
    clean = git-lfs clean -- %f  
    smudge = git-lfs smudge -- %f  
  
[user]  
    name = oraman  
    email = oraman@naver.com
```

깃허브(GitHub)

깃(Git)으로 버전 관리하기



≡ 깃(Git)으로 버전 관리하기

≡ git 관련 명령어

다음은 git에서 많이 사용되는 각 명령어들을 정리한 표입니다.

명령어	설명
git init	깃을 초기화 합니다.
git status	깃의 상태 정보를 조회합니다.
git add filename	filename 정보를 스테이징합니다.
git commit -m '커밋_메시지'	해당 파일을 커밋합니다.
git commit -am 'staging and commit'	스테이징과 동시에 커밋합니다.
git log	깃에 대한 로그 정보를 확인합니다.
git log --oneline --branches	모든 브랜치들에 대하여 로그 정보를 한번에 확인합니다. --oneline 옵션은 커밋 정보를 간략히 표현해 줍니다. --branches 옵션은 모든 브랜치 정보를 한 꺼번에 보여 줍니다.
git diff	깃의 변경 사항에 대하여 확인합니다.(diffenence)
git branch	브랜치 정보를 확인합니다.
git branch abcd	신규 브랜치 abcd를 생성합니다.
git checkout abcd	체크 아웃을 이용하여 abcd 브랜치로 이동합니다.
git merge somebranch	현재 브랜치에 somebranch 브랜치를 병합합니다.

≡ 깃(Git)으로 버전 관리하기

≡ git 관련 명령어

다음은 git에서 많이 사용되는 각 명령어들을 정리한 표입니다.

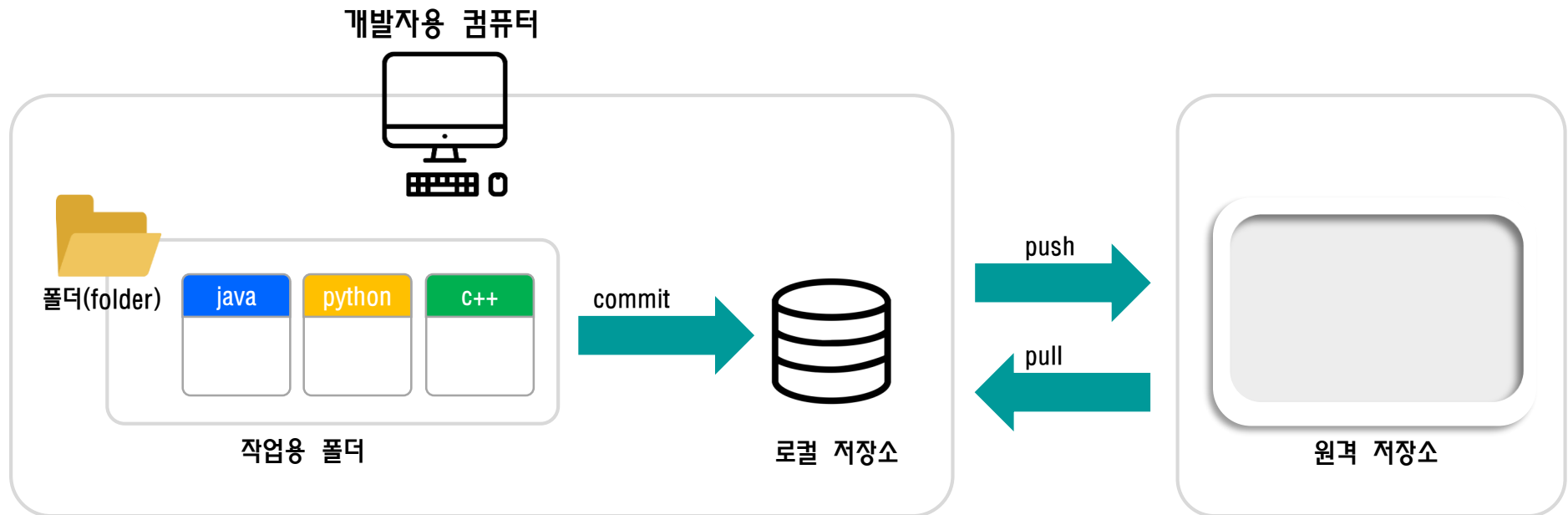
명령어	설명
git remote add origin url_address	url_address이라는 깃허브를 origin이라는 이름으로 원격지 등록을 합니다.
git push origin master	원격지의 master 브랜치에 데이터를 푸시합니다.
git clone remote_url somename	원격지 remote_url의 데이터를 로컬 somename 깃에 복제합니다.
git pull	깃허브의 데이터를 풀링합니다.

≡ 깃(Git)으로 버전 관리하기

≡ 저장소(repository)

저장소는 파일이나 폴더 등이 저장되는 공간을 의미합니다.

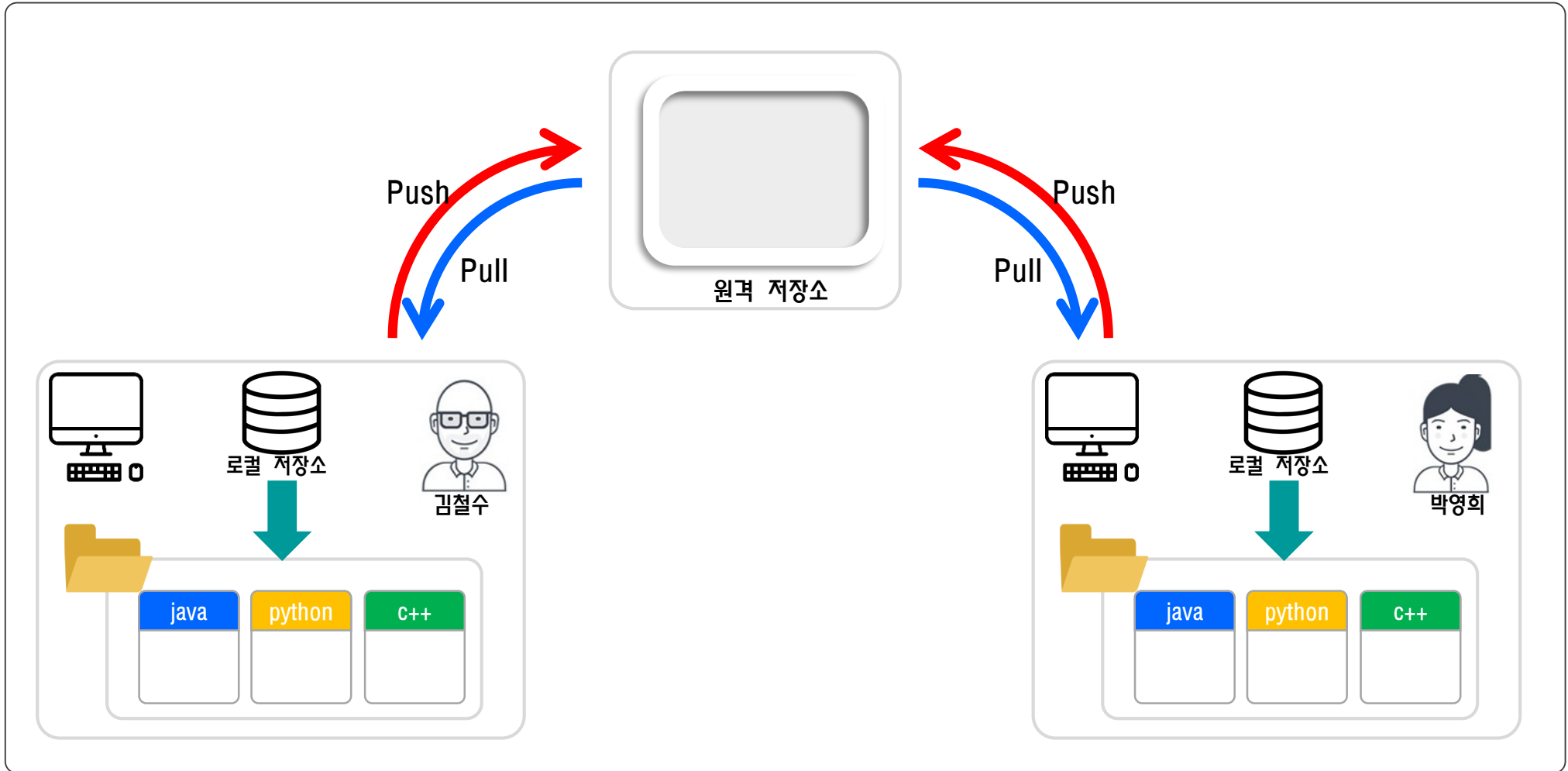
저장소	설명
원격 저장소 (Remote Repository)	파일이 서버나 네트워크상에 존재하는 전용 서버에서 관리되며 여러 사람이 함께 공유하기 위한 저장소입니다.
로컬 저장소 (Local Repository)	자신의 컴퓨터에 파일이 저장되는 개인 전용 저장소입니다.



≡ 깃(Git)으로 버전 관리하기

≡ 저장소(repository)

저장소는 크게 원격 저장소와 로컬 저장소로 구분이 됩니다.

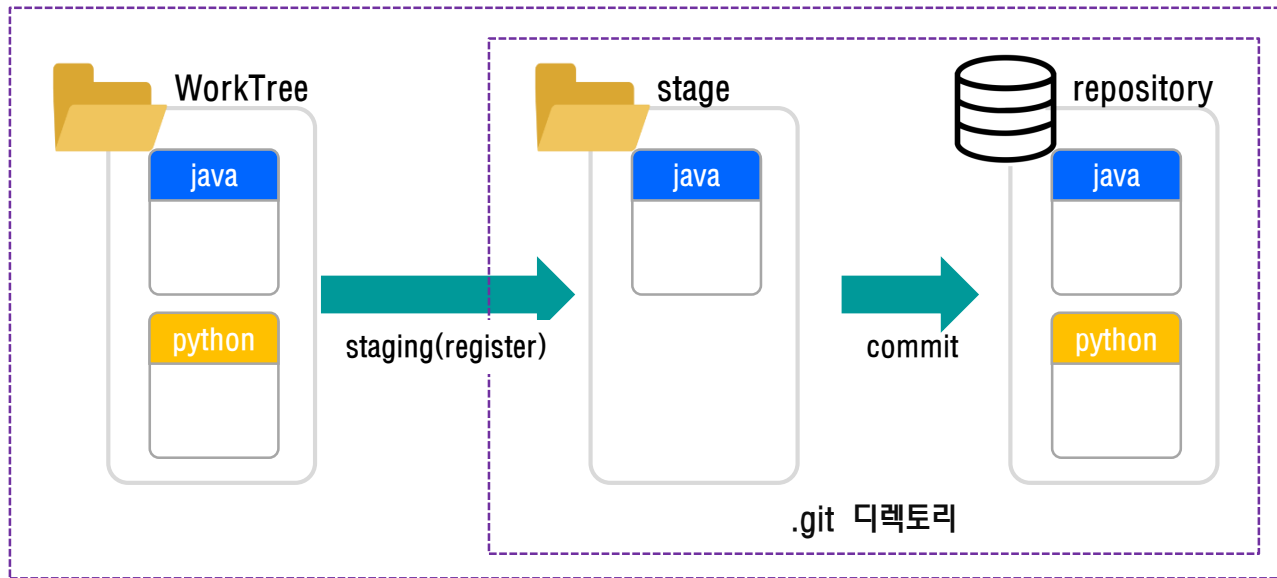


≡ 깃(Git)으로 버전 관리하기

≡ 작업 트리(Work tree)와 stage(스테이지)

작업 트리와 stage 공간에 대하여 살펴 보도록 합니다.

용어	설명	위치
작업 트리(Work tree)	개발자가 수정 작업을 하고 있는 폴더를 의미합니다.	somefolder
스테이지(stage)	커밋을 실행하기 전의 저장소와 작업 트리 사이에 존재하는 공간으로써 커밋이 필요한 목록들을 저장하는 가상의 공간입니다.	somefolder/.git/index
스테이징(stage)	stage에 파일이나 폴더를 등록 시키는 작업을 의미합니다.	-
저장소(repository)	스테이지 영역의 파일들을 version으로 만들어서 저장하는 저장소 역할을 합니다.	somefolder/.git/HEAD



≡ 깃(Git)으로 버전 관리하기

≡ 커밋(commit)

파일을 추가하거나 변경 내용을 저장소에 저장시키는 작업을 의미합니다.

수정된 파일이나 폴더에 추가/변경된 내용을 저장소에 기록하려면 '커밋'이란 동작을 수행해야 합니다.

커밋은 해당 시점의 repo에 대한 스냅샷을 기억해 두기 위한 일종의 체크 포인트 기능을 합니다.

특정 시점으로 이동하려면 이 체크 포인트를 사용하면 됩니다.



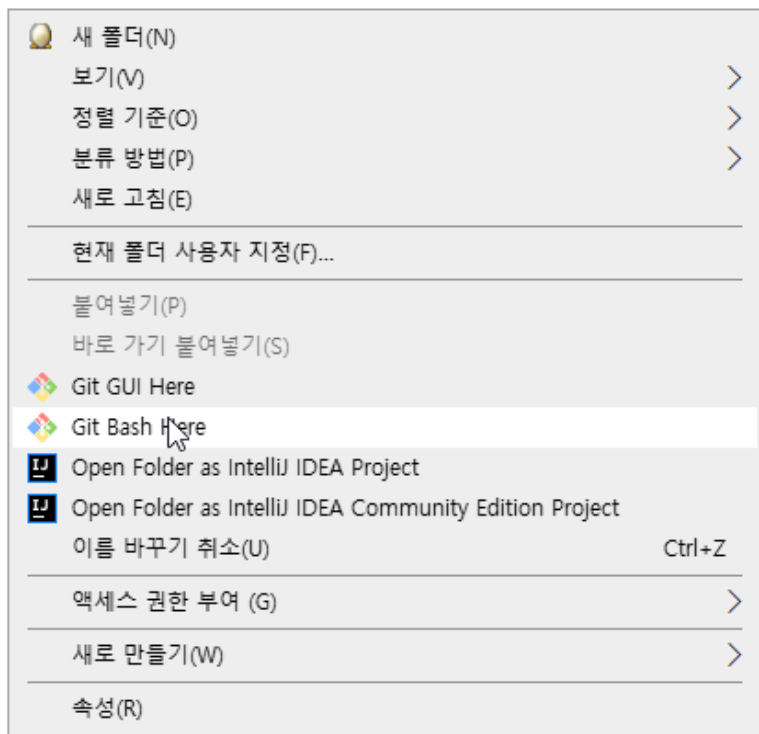
≡ 깃(Git)으로 버전 관리하기

≡ 저장소 생성하기

bluerain이라는 폴더를 생성하고 저장소로 지정해 보도록 하겠습니다.

저장소 지정하기

우선 d 드라이브에 bluerain라는 폴더를 생성하도록 합니다.
그림과 같이 bluerain 폴더로 이동하여 마우스 우측 클릭을 한다음 git bash 창을 엽니다.



≡ 깃(Git)으로 버전 관리하기

≡ 저장소 생성하기

bluerain이라는 폴더를 생성하고 저장소로 지정해 보도록 하겠습니다.

저장소 지정하기

다음 명령어를 git bash 창에서 진행하도록 합니다.

```
$ pwd # 현재 경로를 보여 줍니다.  
/d/bluerain
```

```
$ git init  
Initialized empty Git repository in D:/bluerain/.git/
```

```
$ ls -al # 숨김 파일까지 모두 보여 줍니다.  
total 84  
drwxr-xr-x 1 ugcad 197609 0 Apr 14 21:53 ./  
drwxr-xr-x 1 ugcad 197609 0 Apr 8 20:22 ../  
drwxr-xr-x 1 ugcad 197609 0 Apr 14 21:55 .git/
```



깃(Git)으로 버전 관리하기



저장소 상태 확인하기

우선 저장소에 대한 git의 상태를 확인해 보도록 하겠습니다.

저장소의 상태 확인

```
$ git status  
On branch master
```

```
No commits yet
```

```
nothing to commit (create/copy files and use "git add" to track)
```

≡ 깃(Git)으로 버전 관리하기

≡ 저장소 상태 확인하기

파일을 생성하고, 해당 저장소에 대한 상태를 확인해 봅니다.

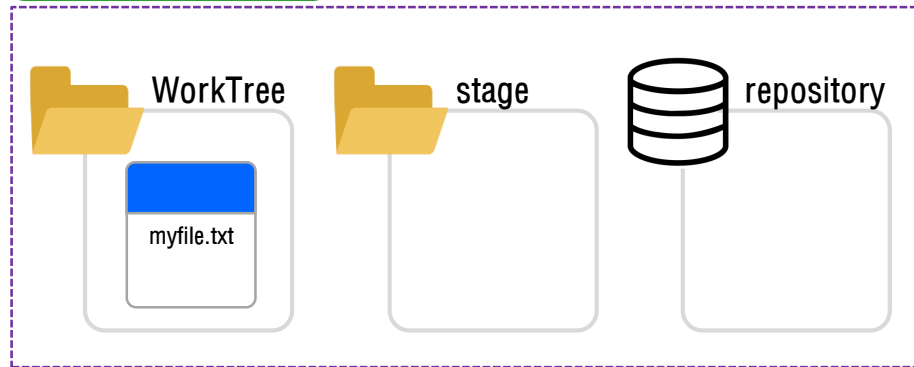
myfile.txt

동해물과 백두산이 마르고 닳도록
i am a boy

저장소의 상태 확인

```
$ git status
# On branch master
#
# No commits yet
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   myfile.txt
#
# nothing added to commit but untracked files present (use "git add" to track)
```

현재 상태



≡ 깃(Git)으로 버전 관리하기

≡ 스테이징하기

파일을 stage에 추가하고, 상태를 재확인합니다.

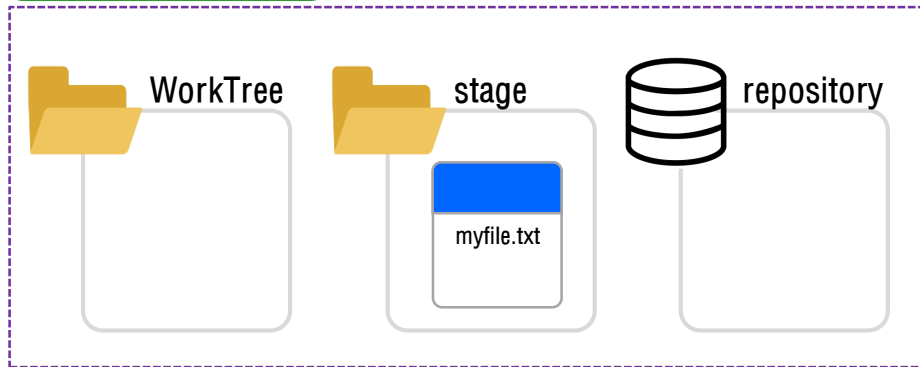
stage에 추가하기

```
git add myfile.txt
```

저장소의 상태 확인

```
git status
# On branch master
#
# No commits yet
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:   myfile.txt
```

현재 상태



경고 메시지

add 명령어 사용시 다음과 같은 경고 메시지는 플랫폼(OS)마다 줄바꿈을 바라보는 문자열이 다르게 인식되므로 Git이 바라볼 땐 둘 중 어느 쪽을 선택할지 몰라 경고 메시지를 띄워주는 내용입니다.

warning: LF will be replaced by CRLF in myfile.txt.

The file will have its original line endings in your working directory

다음 문장으로 경고 메시지를 없앨 수 있습니다.

```
git config --global core.autocrlf true(for windows)
```

```
git config --global core.autocrlf input(for linux)
```


≡ 깃(Git)으로 버전 관리하기

≡ 커밋하기

commit 명령어를 실행해 커밋을 진행합니다.

커밋하기

```
git commit -m "myfile committed"
# [master (root-commit) c55dd06] myfile committed
# 1 file changed, 2 insertions(+)
# create mode 100644 myfile.txt
```

저장소의 상태 확인

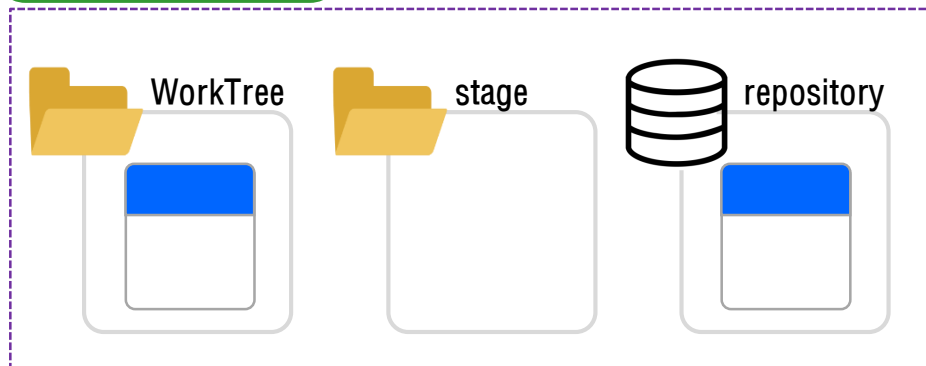
```
git status
# On branch master
# nothing to commit, working tree clean
```

변경 이력 확인

```
git log
# commit c55dd06b89deee213b0ed3af44f304ebd1d34317 (HEAD -> master)
# Author: oraman <oraman@naver.com>
# Date: Tue Dec 14 15:36:38 2023 +0900

# myfile committed
```

현재 상태





깃(Git)으로 버전 관리하기

스테이징과 커밋을 한꺼번에 처리하기

commit 명령어를 실행해 커밋을 진행합니다.

myfile.txt 파일의 내용을 변경하고 다음과 같이 코드를 실습해 봅니다.

myfile.txt

동해물과 백두산이 마르고 닳도록
i am a boy
you are a girl

커밋하기

```
$ git commit -am "staging and commit together"
```

```
$ git log  
commit f5ddb0d47f0b4ba303ba4b5e17682d4dda70c0c2 (HEAD -> master)  
Author: oraman <oraman@naver.com>  
Date: Fri Apr 15 11:23:04 2022 +0900
```

```
staging and commit together
```

```
commit 38f621f18c8386ab38122e61e49e7babe4e5bc1a  
Author: oraman <oraman@naver.com>  
Date: Fri Apr 15 11:17:02 2022 +0900
```

```
myfile committed
```

≡ 깃(Git)으로 버전 관리하기

≡ 커밋 내용 확인해보기

커밋에 대한 기록을 상세하게 살펴 보도록 하겠습니다.

커밋에 대한 기록을 세부적으로 살펴 보려면 'git log' 명령어를 사용하면 됩니다.
commit 옆에 있는 'f5ddb0d47f0b4ba303ba4b5e17682d4dda70c0c2'는 커밋들을 구분하기 위한 식별자입니다.
이것을 commit hash 또는 git hash라고 합니다.

명령어 확인

```
$ git log
commit f5ddb0d47f0b4ba303ba4b5e17682d4dda70c0c2 (HEAD -> master) # 커밋 해시 및 최신 버전
Author: oraman <oraman@naver.com> # 버전 생성자
Date: Fri Apr 15 11:23:04 2022 +0900 # 버전 생성 일자

    staging and commit together # 커밋 메시지

commit 38f621f18c8386ab38122e61e49e7babe4e5bc1a # 이전 커밋 내용
Author: oraman <oraman@naver.com>
Date: Fri Apr 15 11:17:02 2022 +0900

    myfile committed # 커밋 메시지
```

≡ 깃(Git)으로 버전 관리하기

≡ 변경 사항 확인하기

git diff 명령어를 사용하면 커밋에 대한 변경 사항들을 비교해 볼 수 있습니다.

변경 사항을 확인하기 위하여 myfile.txt을 다음과 같이 수정합니다.
현재 상태를 확인한 다음, 차이점을 확인해 보도록 하겠습니다.

myfile.txt

동해물과 백두산이 마르고 닳도록
i am a boy
i am hungry

```
$ git status
On branch master
Changes not staged for commit: # 커밋을 위하여 아직 staging 상태가 아니군요.
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   myfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git diff
diff --git a/myfile.txt b/myfile.txt
index 5d03bea..49a3ba4 100644
--- a/myfile.txt
+++ b/myfile.txt
@@ -1,3 +1,3 @@
     동해물과 백두산이 마르고 닳도록
     i am a boy
-    you are a girl # - 기호는 제거된 내용
+    i am hungry # + 기호는 추가된 내용
```

≡ 깃(Git)으로 버전 관리하기

≡ 단계별 파일 상태 확인하기

버전을 만들어 가는 단계마다 파일의 상태를 다르게 표현합니다.

파일의 상태를 이해하면, 현재 어느 단계에 있는지, 어떠한 일을 진행해야 할 지 등을 알 수 있습니다.

파일의 상태는 크게 'tracked'와 'untracked' 상태로 나누어 집니다.

그럼 다음과 같이 파일을 수정 및 생성한 다음 상태에 대하여 살펴 보겠습니다.

파일	설명	상태	설명
myfile.txt	내용을 수정합니다.	tracked	추적이 필요한 상태로, 이전에 한번이라도 커밋을 수행한 파일이 해당됩니다.
yourfile.txt	신규 파일을 생성합니다.	untracked	이전에 git에서 한번도 추적 관리를 받지 않은 파일로 추적 대상이 아닙니다.

```
$ git status
```

```
On branch master
```

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: myfile.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

yourfile.txt

no changes added to commit (use "git add" and/or "git commit -a")

깃허브(GitHub)

리눅스 명령어



리눅스 명령어

리눅스 명령어에 대하여 살펴 보도록 하겠습니다.

Git은 리눅스 개발자가 최초 개발을 하였다고 설명하였습니다.

따라서, 기본적인 리눅스 명령어를 잘 모르면 다소 어려울 수 있습니다.

깃을 본격적으로 사용하기 전에 미리 숙지를 해야할 몇 가지 명령어에 대하여 살펴 보도록 하겠습니다.

현재 디렉토리 확인

리눅스 명령어에 대하여 살펴 보도록 하겠습니다.

드라이브에서 'git' 명령어를 실행합니다. 하단에 보면 달러(\$) 표시가 보이는 데 이것을 프롬프트(prompt)라고 합니다. 프롬프트는 사용자로 하여금 명령어가 입력되기를 기다리는 곳입니다. 명령어 cd는 디렉토리를 이동시키는 명령어입니다. cd 입력 후 엔터키를 누르면 톨드 표시(물결 ~)가 보이는 데 사용자 홈 디렉토리를 의미합니다.

```
ugcad@ugcadman MINGW64 /c
```

```
$ cd # change directory
```

```
ugcad@ugcadman MINGW64 ~
```

```
$
```

우선 현재 디렉토리가 어디인지 확인하기 위하여 pwd 명령어를 입력해 봅니다. 이 명령어는 print working directory라는 단어의 줄인 말로 현재 작업중인 디렉토리 경로를 알려 주는 역할을 합니다. 샵(sharp) 기호는 주석입니다.

```
ugcad@ugcadman MINGW64 ~
```

```
$ pwd # print working directory
```

```
c:/Users/computer_name
```


현재 디렉토리 확인

리눅스 명령어에 대하여 살펴 보도록 하겠습니다.

출력 결과를 보면 'c/Users/computer_name'인데, 현재 사용자 홈 디렉토리에 위치하고 있음을 알려 주고 있습니다.

현재 디렉토리에 들어 있는 파일이나 디렉토리 목록을 확인하려면 'ls -al' 명령어를 사용합니다. ls 명령어에 사용되는 옵션 스위치는 매우 많은데, a와 l 옵션을 사용하여 출력해 보도록 하겠습니다.

```
ugcad@ugcadman MINGW64 ~  
$ ls -al # 파일 목록 상세 보기  
total 31836  
drwxr-xr-x 1 ugcad 197609 0 Apr 22 22:14 ./  
drwxr-xr-x 1 ugcad 197609 0 Jun 27 2023 ../  
drwxr-xr-x 1 ugcad 197609 0 Dec 24 10:53 .android/  
drwxr-xr-x 1 ugcad 197609 0 Aug 7 2023 .astropy/  
... 이하 중략
```

디렉토리 만들기

신규 디렉토리를 생성해 보도록 합니다.

디렉토리를 신규로 만들어 보는 실습을 해봅니다. 최상위 디렉토리인 `c`로 이동하고, 하위 디렉토리로 `mbc`, `kbs`, `sbs`라는 디렉토리를 생성해 보겠습니다. 우선 디렉토리를 이동하기 위하여 `cd(change directory)`라는 명령어를 사용합니다. 우선 리눅스에서 사용되는 디렉토리 기호를 살펴 보겠습니다.

기호	설명
/	최상위 디렉토리를 의미하거나, 상하위 디렉토리를 위한 구분자로 사용합니다.
.	현재 디렉토리를 의미합니다.
..	상위 디렉토리를 의미합니다.
~	사용자의 홈 디렉토리를 의미합니다.

우선 최상위 디렉토리인 `c`로 이동을 합니다. `cd` 명령어와 슬래시 및 `c` 문자열을 이용하여 디렉토리를 이동하였습니다. 현재 디렉토리가 어디인지 `pwd` 명령어를 이용하여 확인합니다.

```
$ cd /d
```

```
$ pwd
```

```
/d
```

디렉토리 만들기

신규 디렉토리를 생성해 보도록 합니다.

디렉토리 mbc를 우선 만들어 보겠습니다. 디렉토리 생성과 관련된 명령어는 mkdir입니다. make directory라는 명령어의 줄인 말입니다. 다음과 같이 mbc 디렉토리를 우선 생성해 봅니다. 2개 이상의 디렉토리를 한꺼번에 만들려면 다음과 같이 작성하면 됩니다.

```
$ mkdir mbc
$ mkdir kbs sbs
```

디렉토리가 잘 생성이 되었는 지 ls 명령어를 사용하여 확인해 봅니다.

```
$ ls
`$Recycle.Bin`/      `Program Files (x86)`/    Windows/    ojdbc6.jar    temp/
`$WinREAgent`/      ProgramData/             bulkload/    oraclexe/      upload/
`Documents and Settings`@ Python310/                hiberfil.sys pagefile.sys  고니/
DumpStack.log.tmp    Recovery/                 imsi/        sbs/
Intel/               `System Volume Information`/ kbs/         shop/
PerfLogs/            `Tomcat 9.0`/             mbc/         shopping/
`Program Files`/     Users/                    nodejs/       swapfile.sys
```

디렉토리 이동하기

디렉토리를 이동해 보도록 하겠습니다.

이제 mbc 디렉토리로 이동해 봅니다. 디렉토리 이동 명령어는 cd 입니다. 다음과 같이 mbc로 이동한 다음, pwd 명령어로 현재 디렉토리가 어디인지 확인해 봅니다.

```
$ cd mbc
```

```
$ pwd
```

```
/c/mbc
```

디렉토리 이동하기

디렉토리를 이동해 보도록 하겠습니다.

이번에는 kbs 디렉토리로 이동해 봅니다. 현재 디렉토리이므로 mbc에서 kbs로 이동하려면 상위 디렉토리로 이동한 다음 kbs 디렉토리로 이동해야 합니다. 다음과 같이 실습해 봅니다.

```
$ cd ../../kbs
```

```
$ pwd
```

```
/c/kbs
```

디렉토리 이동하기

디렉토리를 이동해 보도록 하겠습니다.

동일한 방식으로 sbs로 이동하려면 다음과 같습니다. 이전 예시와 다르게 ./ 기호는 사용하지 않았습니다. ./ 기호는 현재 디렉토리를 의미하므로 굳이 사용하지 않아도 되는 것입니다.

```
$ cd ../sbs
```

```
$ pwd
```

```
/c/sbs
```

텍스트 문서 만들기

텍스트 문서를 만들어 보겠습니다.

현재 디렉토리는 sbs 입니다. 텍스트 파일을 하나 생성해 보겠습니다. 생성하고자 하는 파일은 sbs01.txt 입니다. 파일의 내용은 다음과 같습니다.

파일의 내용

```
hello world  
여러분 안녕하세요.  
123
```

리눅스의 기본 편집기 이름은 뱀(vim) 입니다. 윈도우의 메모장과 유사한 텍스트 편집기입니다. vim이라는 키워드와 함께 생성할 파일 이름을 작성하면 됩니다. 다음 명령어로 파일을 생성하도록 합니다.

```
$ vim sbs01.txt # make new file sbs01.txt
```

텍스트 문서 만들기

텍스트 문서를 만들어 보겠습니다.

파일이 열리면 기본 모드가 'ex 모드'이며, 텍스트를 입력할 수 없습니다. 입력이 가능하려면 '입력 모드'가 되어야 합니다. '입력 모드'는 문자열 아이(i)를 입력하면 진입할 수 있습니다. sbs01.txt 파일의 내용을 입력하도록 합니다. 작성이 완료되고 나면 파일을 저장하고 vim을 빠져 나가야만 합니다. Esc 키, 콜론, w 및 q 문자를 입력하고 빠져 나가도록 합니다.

'ex 모드'에서 사용 가능한 명령어는 다음과 같은 항목들이 있습니다.

명령어	설명
:wq	편집중이던 문서를 저장하고, 종료합니다.
:q!	편집중이던 문서를 저장하지 않고, 종료합니다.

텍스트 문서 만들기

텍스트 문서를 만들어 보겠습니다.

기본 편집기를 변경하려면, 다음과 같은 명령어를 사용하면 됩니다. 예를 들어서 메모장을 기본 편집기로 지정하고 파일 a.txt를 작성하려면 다음과 같습니다.

```
$ git config --global core.editor 'notepad'
```

```
$ notepad aa.txt
```

텍스트 문서 확인하기

텍스트 문서를 만들어 보겠습니다.

sbs01.txt 파일의 내용을 확인해보는 명령어는 cat 입니다.

```
$ cat sbs01.txt
```

```
hello world
```

```
여러분 안녕하세요.
```

```
123
```

라인 번호와 함께 출력하려면 pipe와 함께 nl을 사용하면 됩니다.

```
$ cat sbs01.txt | nl
```

파일 복사하기

동일하거나 다른 디렉토리에 파일을 복사해 보도록 합니다.

sbs01.txt 파일을 sbs02.txt 라는 이름으로 파일을 복사해 보겠습니다. 복사를 위한 명령어는 cp입니다. 다음과 같이 복사를 진행하고, 파일 목록을 확인해 봅니다.

```
$ ls
```

```
sbs01.txt
```

```
$ cp sbs01.txt sbs02.txt # copy source target
```

```
$ ls .al
```

파일 복사하기

동일하거나 다른 디렉토리에 파일을 복사해 보도록 합니다.

이번에는 파일들을 다른 디렉토리에 복사해 봅니다. sbs01.txt 파일은 mbc에 sbs02.txt 파일은 kbs 디렉토리에 복사해 하되 파일 이름을 다르게 지정하여 복사해 보도록 하겠습니다.

```
$ cp sbs01.txt ../mbc/mbcfile.txt
```

```
$ cp sbs02.txt ../kbs/kbsfile.txt
```

파일 복사하기

동일하거나 다른 디렉토리에 파일을 복사해 보도록 합니다.

mbc 디렉토리 내의 파일 목록은 현재 디렉토리인 sbs에서 확인해 보겠습니다. 다음 명령어를 사용하여 확인해 봅니다.

```
$ ls -al ../mbc/
```

파일 복사하기

동일하거나 다른 디렉토리에 파일을 복사해 보도록 합니다.

kbs 디렉토리 내의 파일 목록은 직접 디렉토리로 이동을 하여 확인해 보겠습니다. 다음 명령어를 사용하여 확인해 봅니다.

```
$ cd ../kbs/
```

```
$ ls -al
```

파일 이동/이름 변경과 디렉토리 삭제하기

mv는 이동/이름 변경을 rm 명령어는 삭제 명령어 입니다.

mv 명령어를 사용하면 파일을 이동시키거나, 이름 변경이 가능합니다. kbsfile.txt 이라는 파일을 newkbs.txt으로 이름 변경해 보겠습니다.

rm 명령어는 특정 디렉토리나 파일을 삭제할 수 있습니다. 우리는 지금 kbs 디렉토리에 있습니다. newkbs.txt 파일이 더 이상 필요가 없다고 가정하고 삭제를 해보겠습니다. 파일이나 폴더를 삭제하기 위한 명령어는 rm 입니다. 그럼 다음과 같이 파일을 삭제하도록 합니다.

```
$ mv kbsfile.txt newkbs.txt
```

```
$ ls -al
```

```
total 13
```

```
drwxr-xr-x 1 kosmo 197121 0 May 26 12:10 ./
```

```
drwxr-xr-x 1 kosmo 197121 0 May 26 11:41 ../
```

```
-rw-r--r-- 1 kosmo 197121 43 May 26 12:03 newkbs.txt
```

```
$ rm newkbs.txt # remove
```

디렉토리 삭제하기

rm 명령어는 특정 디렉토리나 파일을 삭제할 수 있습니다.

이제 모든 디렉토리를 삭제 해보겠습니다. 해당 디렉토리를 삭제하려면 상위 디렉토리로 이동하여야 합니다. 현재 우리는 kbs 디렉토리에 있으므로 상위로 우선 이동을 한 후 삭제 명령어를 사용해야 합니다. 다음과 같이 상위 디렉토리로 이동합니다.

```
$ cd ..  
$ pwd  
/c
```

이제 kbs, mbc 및 sbs 디렉토리를 다음과 같이 삭제해보겠습니다. 디렉토리를 삭제하고자 하는 경우에는 -r 옵션을 추가로 사용하면 됩니다.

```
$ rm kbs  
rm: cannot remove 'kbs': Is a directory  
  
$ rm -r kbs  
  
$ rm -r mbs  
  
$ rm -r sbs
```


깃허브(GitHub)

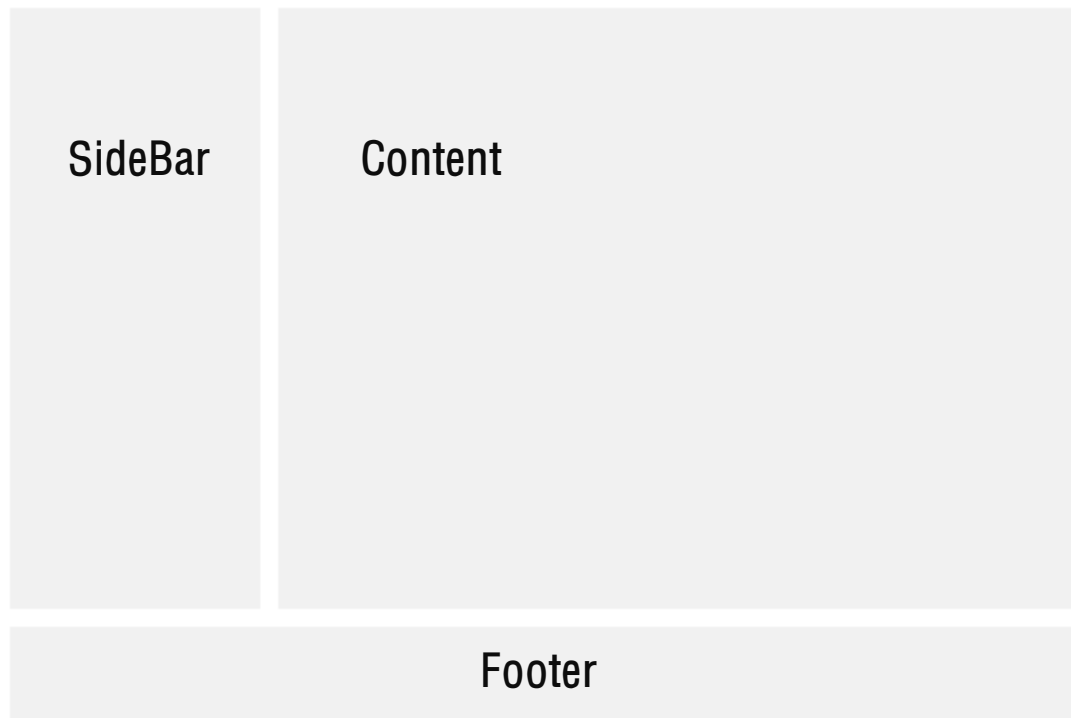
깃(Git)과 브랜치



≡ 깃(Git)과 브랜치

≡ 브랜치란?

브랜치의 개념에 대하여 살펴 봅니다.



≡ 깃(Git)과 브랜치

≡ 브랜치란?

브랜치의 개념에 대하여 살펴 봅니다.

브랜치의 용어 및 특징

브랜치란 어떤 작업을 수행하기 위한 독립적인 개념입니다.

저장소 생성시 기본 브랜치의 이름은 'master'입니다.

여러 브랜치간의 병합(Merge)이 가능합니다.

다른 브랜치의 작업에 영향을 받지 않고 독립적으로 작업 수행이 가능합니다.

브랜치를 변경하는 동작을 체크 아웃(checkout)이라고 합니다.

≡ 브랜치 만들어 보기

≡ 실습을 위한 사전 환경 구성하기

이번에는 직접 브랜치를 만들어 보면서 개념을 살펴 보도록 하겠습니다.

브랜치를 테스트를 위하여 branchtest01 폴더를 생성하도록 합니다.

branchtest01 폴더를 git init 명령어를 사용하여 깃 저장소로 만들고 파일 목록을 확인해 봅니다.

```
$ git init
Initialized empty Git repository in D:/branchtest01/.git/

$ ls -al
total 16
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 .git/
```

≡ 브랜치 만들어 보기

≡ 실습을 위한 사전 환경 구성하기

실습용 파일을 만들고, 커밋을 수행해 보겠습니다.

해당 폴더에 somefile.txt 파일을 생성하고, 다음과 같이 내용을 추가하고 저장합니다.

‘somefile.txt’ 파일을 stage에 올리고, 커밋을 수행합니다.

파일의 내용

hello01

```
$ git add somefile.txt
```

```
$ git commit -m 'commit01'
```

```
[master (root-commit) 6b707f9] commit01
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 somefile.txt
```

≡ 브랜치 만들어 보기

≡ 실습을 위한 사전 환경 구성하기

커밋 로그를 확인해 봅니다.

커밋이 완료가 되었다면 다음과 같이 커밋 내역을 확인해 봅니다. 커밋 해시 정보와 커밋시 사용한 메모 내용을 확인할 수 있습니다.

```
$ git log
commit 6b707f9ef0b3ed58f37ab8a5e46d1dd734f9b9bc (HEAD -> master)
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:02:14 2022 +0900

commit01
```

≡ 브랜치 만들어 보기

≡ 실습을 위한 사전 환경 구성하기

다음과 같은 순서대로 'somefile.txt' 파일에 대하여 두번 더 커밋을 진행해 보겠습니다.

파일의 내용

```
hello01  
hello02
```

```
$ git commit -am 'commit02'  
[master 602ee60] commit02  
1 file changed, 1 insertion(+)
```

somefile.txt 파일에, 내용 'hello03'을 추가합니다.

파일의 내용

```
hello01  
hello02  
hello03
```

```
$ git commit -am 'commit03'  
[master ebb0b92] commit03  
1 file changed, 1 insertion(+)
```

≡ 브랜치 만들어 보기

≡ 실습을 위한 사전 환경 구성하기

커밋 로그를 확인해 봅시다.

커밋이 완료된 다음 커밋 내역을 조회해 보겠습니다. 예시에서는 master 브랜치 항목이 가장 최신 커밋인 'commit03'을 가리키고 있습니다. HEAD는 현재 작업 중인 브랜치를 가리키는 포인터 정보입니다. (HEAD -> master)의 의미는 작업 중인 포인터 HEAD가, master라는 브랜치를 참조하고 있음을 알려 주고 있습니다.

```
$ git log
commit ebb0b92517b499145c501b6ccd3a0a4bd96b25bd (HEAD -> master) # 현재 저장소가 master를 참조하고 있습니다.
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:04:06 2022 +0900
    commit03

commit bbbdce53425e5c7137fb28e02b3cbc82370f20c3
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:03:31 2022 +0900
    commit02

commit 6b707f9ef0b3ed58f37ab8a5e46d1dd734f9b9bc
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:02:14 2022 +0900
    commit01
```


≡ 브랜치 만들어 보기

≡ 신규 브랜치 만들기

현재 구성된 브랜치 정보 확인 및 신규 브랜치를 생성해 봅니다.

신규로 생성하고자 하는 브랜치는 부산(pusan) 브랜치 입니다.
브랜치를 생성하거나 조회하는 명령어는 git branch를 사용하면 됩니다.

```
$ git branch  
* master
```

위의 결과를 보면 ‘* master’이라고 표시가 되는 데, *이 붙어 있는 브랜치가 현재 활성화된 브랜치(active branch)입니다.
‘git init’ 명령어를 사용하게 되면, 최초 저장소를 만들때 기본적으로 master이라는 브랜치가 자동으로 만들어 집니다.

새로운 브랜치를 생성하려면 다음과 같이 작성하면 됩니다. 실행을 하더라도 화면에는 변화된 내용을 확인할 수 없습니다.
다음 명령어를 사용하여 부산(pusan) 브랜치를 만들어 보겠습니다. 그러면, pusan 브랜치는 master 브랜치의 상태를 그대로 복사해 옵니다.
즉, commit01~ commit03 까지의 상태가 그대로 복사됩니다.

```
$ git branch pusan
```

≡ 브랜치 만들어 보기

≡ 신규 브랜치 만들기

존재하는 브랜치 이름들과 활성 상태에 대하여 확인합니다.

브랜치를 확인하려면 다음과 같이 작성하면 됩니다.

어려 개의 브랜치가 존재하는 경우 * 표시가 되어 있는 브랜치가 활성화되어 있는 브랜치입니다.

참고로, 내 컴퓨터 '.git\refs\heads' 폴더 아래에 branch 항목들이 생성되는 것을 확인할 수 있습니다.

```
$ git branch
```

```
* master
```

```
pusan
```

📁 > 내 PC > Yoon (D:) > branchtest01 > .git > refs > heads

이름	수정한 날짜	유형	크기
📄 master	2022-12-02 오후 9:35	파일	1KB
📄 pusan	2022-12-02 오후 9:38	파일	1KB

≡ 브랜치 만들어 보기

≡ 신규 브랜치 만들기

여러 개의 브랜치가 존재하는 경우 git log의 결과는 이전과 다르게 표현됩니다.

다음 명령어 'git log'를 보면 (HEAD -> master, pusan)에서 기호 '->' 이후에 2개의 브랜치가 존재함을 표현하고 있습니다. 문자 '->' 다음에 존재하는 master 브랜치가 현재 활성화 된 브랜치입니다. 그리고, 모든 브랜치가 'commit03' 커밋 상태에서 만들어졌고, 현재 브랜치는 master인 것을 확인할 수 있습니다.

```
$ git log
commit ebb0b92517b499145c501b6ccd3a0a4bd96b25bd (HEAD -> master, pusan) # 2개의 브랜치 중에서 master이 활성화되어 있습니다.
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:04:06 2022 +0900
    commit03

commit bbbdce53425e5c7137fb28e02b3cbc82370f20c3
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:03:31 2022 +0900
    commit02

commit 6b707f9ef0b3ed58f37ab8a5e46d1dd734f9b9bc
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:02:14 2022 +0900
    commit01
```

≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

서로 다른 브랜치간의 이동에 대하여 살펴 봅니다.

현재 시점에서 master와 pusan 브랜치의 마지막 커밋 정보는 동일합니다.

이제 'master' 브랜치에서 다음과 같이 somefile.txt 파일을 수정하고, 커밋을 수행해 보도록 하겠습니다.

파일의 내용

```
hello01  
hello02  
hello03  
hello04 # 신규 추가됨
```

```
$ git commit -am 'master commit 04'  
[master ab84d4a] master commit 04  
1 file changed, 1 insertion(+)
```

≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

서로 다른 브랜치간의 이동에 대하여 살펴 봅니다.

커밋에 대한 정보를 간략히 보고자 할때는 --oneline 옵션을 부여하면 됩니다.

```
$ git log --oneline
ab84d4a (HEAD -> master) master commit 04 # master의 마지막 커밋 시점이며, 현재 활성화 입니다.
ebb0b92 (pusan) commit03 # pusan의 마지막 커밋 시점입니다.
bbbdce5 commit02
6b707f9 commit01
```

≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

체크 아웃을 이용한 브랜치 이동을 해보겠습니다.

체크 아웃(checkout)이란, 내가 사용할 브랜치를 지정하는 것을 의미하는 데, 체크 아웃을 하게 되면 어떠한 결과가 나오는지 테스트를 해보도록 하겠습니다. 브랜치를 이동하려면 checkout 명령어와 함께 이동할 브랜치 이름을 작성하면 됩니다. 다음 예시는 'pusan' 브랜치로 체크 아웃'을 한다고 표현합니다.

참고로 '-b' 옵션은 브랜치 생성과 체크 아웃을 동시에 진행하기 위한 옵션입니다.

```
$ git checkout -b <branch>
```

```
# 대구 브랜치는 존재하지 않습니다. 다음과 같이 의도적으로 작성해 보세요.
```

```
$ git checkout daegu
```

```
error: pathspec 'daegu' did not match any file(s) known to git
```

```
# 브랜치 목록은 다음 명령어를 이용하여 확인이 가능합니다.
```

```
$ git branch
```

```
* master
```

```
pusan
```

```
$ git checkout pusan # from master to pusan
```

```
Switched to branch 'pusan'
```

≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

체크 아웃을 하게 되면 HEAD 파일이 갱신됩니다.

```
$ git branch # 활성화된 브랜치를 확인합니다.
```

```
master
```

```
* pusan
```

```
$ cat .git/HEAD # current HEAD file info
```

```
ref: refs/heads/pusan
```

```
$ git checkout master
```

```
$ cat .git/HEAD
```

```
ref: refs/heads/master
```

```
$ git checkout pusan
```

≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

체크 아웃을 수행하여 이동한 브랜치의 로그 정보를 확인해 봅니다.

다시 로그를 확인해보면, 최신 커밋 정보를 보면 'HEAD -> pusan' 즉, pusan 브랜치가 활성화 되었음을 알 수 있습니다. 그리고, master 브랜치에서 수행되었던 모든 커밋 내용들이 복사가 되어 있음을 확인할 수 있습니다.

```
$ git log --oneline  
ebb0b92 (HEAD -> pusan) commit03  
bbbdce5 commit02  
6b707f9 commit01
```


≡ 브랜치 만들어 보기

≡ 브랜치 간 이동해보기

텍스트 파일 내용을 확인해 봅니다.

somefile.txt 파일의 내용을 확인해 보도록 합니다. 최신 커밋이 'commit03'이기 때문에 3개의 행이 들어가 있을 겁니다.

이유는 master 브랜치 분기된 이후에 master 브랜치에 추가 되었던 커밋은 pusan 브랜치에 영향을 주지 않았기 때문입니다. 따라서, master 브랜치에서 추가하였던 'hello04'라는 문자열은 보이질 않습니다.

```
$ cat somefile.txt
```

```
hello01
```

```
hello02
```

```
hello03
```

☰ 브랜치 정보 확인하기

☰ 새로운 브랜치에서 커밋하기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

각각의 브랜치에서 커밋 과정이 이루어 질 때, 해당 브랜치들간의 관계를 확인하는 방법과 차이점을 확인하는 방법에 대하여 살펴 보고자 합니다.

이전 과정을 잘 수행하였다면 현재 브랜치는 pusan가 됩니다. 다음 명령어를 사용하여 현재 활성화된 브랜치 이름을 확인해 봅니다.

```
$ git branch  
master  
* pusan
```

☰ 브랜치 정보 확인하기

☰ 새로운 브랜치에서 커밋하기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

somefile.txt 파일에 'pusan commit 04'라는 내용을 추가하고 저장합니다.

파일의 내용

파일의 내용

hello01

hello02

hello03

pusan commit 04

새로운 파일 pusan.txt 파일을 생성하고, 다음 내용을 추가하고 저장하도록 합니다.

파일의 내용

pusan commit 04 in file pusan.txt

☰ 브랜치 정보 확인하기

☰ 새로운 브랜치에서 커밋하기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

dot(.) 기호를 사용하면 현재 저장소에서 수정된 파일을 한꺼번에 staging할 수 있습니다.

```
$ git add . # staging in all files
```

```
$ git commit -m 'pusan commit 04'
```

```
[pusan 45ce2cb] pusan commit 04
```

```
2 files changed, 2 insertions(+)
```

```
create mode 100644 pusan.txt
```

☰ 브랜치 정보 확인하기

☰ 새로운 브랜치에서 커밋하기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

커밋 정보를 확인해 보겠습니다. 현재 pusan 브랜치에 체크 아웃을 한 상태이고, pusan 브랜치의 최신 커밋의 내용은 'pusan commit 04'입니다.

```
$ git log --oneline
45ce2cb (HEAD -> pusan) pusan commit 04
ebb0b92 commit03
bbbdce5 commit02
6b707f9 commit01
```

--brahches 옵션을 사용하면, 여러 브랜치의 커밋 정보들을 같이 확인할 수 있습니다.

```
$ git log --oneline --branches
45ce2cb (HEAD -> pusan) pusan commit 04 # pusan의 마지막 커밋 시점
ab84d4a (master) master commit 04 # master의 마지막 커밋 시점
ebb0b92 commit03
bbbdce5 commit02
6b707f9 commit01
```

☰ 브랜치 정보 확인하기

☰ 브랜치들간의 차이점 알아 보기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

두 개의 브랜치 사이에 .. 기호를 넣는 방식으로 차이점을 확인할 수 있습니다. 사용 형식은 'git log A..B'는 A 브랜치에 없고, B 브랜치에 있는 내용만 확인하고자 할 때 사용합니다. 다음 예시는 master 브랜치에는 존재하지 않고, 오직 pusan 브랜치에만 존재하는 커밋을 확인할 수 있습니다.

```
$ git log master..pusan
commit 45ce2cb246088b946a278596f5b5ed4546176757 (HEAD -> pusan)
Author: oraman <oraman@naver.com>
Date: Mon Nov 28 15:09:38 2022 +0900

    pusan commit 04
```

☰ 브랜치 정보 확인하기

☰ 브랜치들간의 차이점 알아 보기

브랜치들 사이의 커밋 정보의 차이점을 확인해보도록 하겠습니다.

물론 반대로 확인하려면 'git log pusan..master' 명령어를 이용하면 됩니다.

pusan 브랜치에는 존재하지 않고, 오직 master 브랜치에만 존재하는 커밋을 확인할 수 있습니다.

```
$ git log pusan..master
commit ab84d4ab9d319da5ce6418df114b941d8dfb363f (master)
Author: oraman <orman@naver.com>
Date: Mon Nov 28 15:06:23 2022 +0900

master commit 04
```

≡ 브랜치 간의 병합

≡ 테스트 시나리오

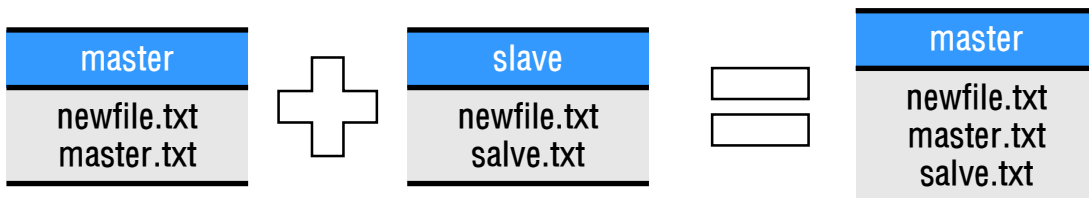
브랜치로 병합시키기 위하여 다음과 같은 순서대로 테스트를 진행해 보도록 하겠습니다.

여러 개로 나누어서 만들어진 브랜치는 어느 시점에 작업을 마무리하고 기준이 되는 브랜치와 병합을 해야 합니다. 여러 개의 상황으로 나누어 브랜치 병합 테스트를 수행해 봅니다. 브랜치 간의 충돌에 대해서도 살펴 보도록 하겠습니다

구현 되는 순서

마스터(master) 브랜치	슬레이브(slave) 브랜치
newfile.txt 파일 생성 후 커밋	x
x	slave 브랜치 신규 생성됨
master.txt 파일 생성 후 커밋	x
slave 브랜치로 checkout →	
x	slave.txt 파일 생성 후 커밋
← master 브랜치로 checkout	
merge 기능 사용하기	x

최종 결과물



≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

브랜치들의 모든 파일을 하나로 모으는 작업을 수행합니다.

브랜치 병합 테스트를 위하여 branchtest02 폴더를 생성하도록 합니다.

branchtest02 폴더를 git init 명령어를 사용하여 깃 저장소로 만들고 파일 목록을 확인해 봅니다.

```
$ git init
Initialized empty Git repository in D:/branchtest02/.git/

$ ls -al
total 16
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 28 13:16 .git/
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

신규 파일을 하나 생성하고, 커밋 작업을 수행합니다.

newfile.txt 파일을 다음과 같이 생성한 다음, 저장하도록 합니다.

‘newfile.txt’ 파일을 stage에 올리고, 커밋을 수행합니다.

파일의 내용

some content 01

```
$ cat newfile.txt
```

```
some content 01
```

```
$ git add newfile.txt
```

```
$ git commit -m 'merge 01'
```

```
[master (root-commit) d3d0912] merge 01
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 newfile.txt
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

병합에 사용될 신규 브랜치를 하나 생성합니다.

‘slave’라는 브랜치를 만듭니다.

이전에 수행한 master 작업 이력이 복사됩니다.

```
$ git branch slave
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

새 파일을 하나 생성하고, 커밋을 진행합니다.

현재 브랜치인 master에 master.txt 라는 파일을 생성하고, 다음과 같이 작업합니다.

‘master merge 02’라는 메시지와 함께 커밋을 수행합니다.

파일의 내용

master 02

```
$ cat master.txt
```

master 02

```
$ git add master.txt
```

```
$ git commit -m 'master merge 02'
```

```
[master 7f65786] master merge 02
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 master.txt
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

새로운 브랜치로 체크 아웃을 진행합니다.

slave 브랜치로 checkout을 수행합니다.

```
$ git checkout slave  
Switched to branch 'slave'
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

이동한 브랜치에 새 파일을 하나 생성하고, 커밋을 진행합니다.

slave 브랜치에 slave.txt 라는 파일을 생성하고, 다음과 같이 작성합니다.

‘slave merge 02’라는 메시지와 함께 커밋을 수행합니다.

파일의 내용

```
slave 02
```

```
$ cat slave.txt
```

```
slave 02
```

```
$ git add slave.txt
```

```
$ git commit -m 'slave merge 02'
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

브랜치들의 로그를 확인해 봅니다.

git log 명령어로 현재의 상태를 확인해 봅니다. 2개의 브랜치가 'merge 01'라는 동일한 커밋 정보를 가지고 있습니다.

slave 브랜치는 'slave merge 02'라는 커밋 정보를, master 브랜치에는 'master merge 02'라는 커밋 정보를 가지고 있습니다.

```
$ git log --oneline --branches
7de9ab5 (HEAD -> slave) slave merge 02
7f65786 (master) master merge 02
d3d0912 merge 01
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

브랜치 병합을 위하여 maste으로 이동합니다.

브랜치 병합은 master 브랜치에서 진행해야 하기 때문에 master 브랜치로 우선 이동합니다.

```
$ git checkout master  
Switched to branch 'master'
```


≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

브랜치 병합 명령어인 merge 컨맨드를 사용합니다.

다음 메시지는 자동으로 병합이 되면서 나타나는 메시지입니다. 이 메시지는 필요하다면 직접 수정할 수 있습니다.

기본 값으로 :wq를 이용하여 종료하도록 합니다.

```
$ git merge slave
Merge branch 'slave'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

수정 후 화면을 빠져 나오면 다음과 같은 결과 화면을 볼 수 있습니다.

```
Merge made by the 'ort' strategy.
slave.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 slave.txt
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

병합이 잘 되었는지 파일 목록을 확인해 봅니다.

slave 브랜치에 존재하던 slave.txt 파일이 master 브랜치에 합쳐진 것을 확인할 수 있습니다.

```
$ ls -al
total 19
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:19 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:16 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:20 .git/
-rw-r--r-- 1 kosmo 197121 10 Nov 28 15:19 master.txt
-rw-r--r-- 1 kosmo 197121 16 Nov 28 15:16 newfile.txt
-rw-r--r-- 1 kosmo 197121 15 Nov 28 15:19 slave.txt
```

≡ 브랜치 간의 병합

≡ 브랜치간 서로 다른 파일들을 한 브랜치로 병합시키기

편집기 창 띄우지 않기

merge 작업을 진행할 때 편집기 창을 로딩하지 않으려면 다음과 같은 옵션을 같이 사용하면 됩니다.

추가 사항

```
$ git merge slave --no-edit
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

동일 문서 내의 다른 라인 위치를 변경하는 실습에 대하여 살펴 봅니다.

서로 다른 branch가 동일한 이름의 파일 happydata.txt를 가지고 있습니다.

동일한 문서에 대하여 다른 위치(라인 번호)의 내용을 수정하고 병합 하고자 하는 경우 어떻게 데이터가 반영이 되는지 살펴 보도록 하겠습니다.

happydata.txt 파일

라인 번호	master 브랜치	subject 브랜치
1	#master area	#master area
2	master01	master01
3	master02	
4		
5	#subject area	#subject area
6	subject01	subject01
7		subject02

구현 되는 순서

master 브랜치	subject 브랜치
happydata.txt 파일 생성 후 커밋	x
x	subject 브랜치 신규 생성됨
happydata.txt 파일 수정 후 커밋	x
subject 브랜치로 checkout →	
x	happydata.txt 파일 수정 후 커밋 master 브랜치와는 다른 라인
← master 브랜치로 checkout	
merge 기능 사용하기	x

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

동일 문서 내의 다른 라인 위치를 변경하는 실습에 대하여 살펴 봅니다.

‘branchtest03’이라는 깃 저장소를 만듭니다.

깃에 대한 초기화 명령어를 이용하여 다음과 같이 시작합니다.

```
$ git init
Initialized empty Git repository in D:/branchtest03/.git/

ls -al
total 16
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:31 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:31 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:31 .git/
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

실습을 위한 신규 파일을 생성하고, 커밋을 수행합니다.

master 브랜치에서 happydata.txt 파일을 생성한 후, 다음과 같은 내용을 추가한 다음, 저장합니다.

happydata.txt 파일을 stage에 올리고, 커밋을 수행합니다.

파일의 내용

```
#master area  
master01
```

```
#subject area  
subject01
```

```
$ git add happydata.txt
```

```
$ git commit -m 'master happydata commit 01'
```

```
[master (root-commit) 6d0190e] master happydata commit 01
```

```
1 file changed, 6 insertions(+)
```

```
create mode 100644 happydata.txt
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

신규 브랜치를 생성합니다.

‘subject’라는 브랜치를 만듭니다.

```
$ git branch subject
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

텍스트 파일을 수정하고, 다시 커밋합니다.

이제 happydata.txt 파일은 현재 양쪽 branch에 모두 존재합니다.
master 브랜치에서 happydata.txt 파일을 다음과 같이 수정하도록 합니다.

파일의 내용

```
#master area  
master01  
master02  
  
#subject area  
subject01
```

수정된 내용을 커밋합니다.

```
$ git commit -am 'master happydata commit 02'  
[master 4fe8dcb] master happydata commit 02  
1 file changed, 1 insertion(+), 1 deletion(-)
```


브랜치 간의 병합

동일 문서 내의 다른 위치를 변경하기

신규 브랜치로 이동하여, 텍스트 파일을 수정하고, 커밋하도록 하겠습니다.

subject 브랜치로 이동하여 happydata.txt 파일을 다음과 같이 수정합니다.

```
$ git checkout subject  
Switched to branch 'subject'
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

신규 브랜치로 이동하여, 텍스트 파일을 수정하고, 커밋하도록 하겠습니다.

happydata.txt 파일을 열고, 다음과 같이 수정합니다.

파일의 내용

```
#master area  
master01
```

```
#subject area  
subject01  
subject02
```

```
$ git commit -am 'subject happydata commit 02'  
[subject bf35853] subject happydata commit 02  
1 file changed, 1 insertion(+)
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

master 브랜치와 동일한 이름의 파일이지만, 변경된 위치가 서로 다를 수 있습니다.

happydata.txt 파일의 내용을 확인합니다.

```
$ cat happydata.txt
```

```
#master area
```

```
master01
```

```
#subject area
```

```
subject01
```

```
subject02
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

subject 브랜치와 동일한 이름의 파일이지만, 변경된 위치가 서로 다를 수 있습니다.

master 브랜치로 체크 아웃을 합니다. 현재 상태는 happydata.txt 파일이 양쪽에서 모두 수정이 되었지만, 수정된 위치가 서로 다릅니다.

```
$ git checkout master
Switched to branch 'master'

$ cat happydata.txt
#master area
master01
master02

#subject area
subject01
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

git merge 명령어를 사용하여 subject 브랜치의 내용을 master 브랜치로 끌어 들입니다.

다음과 같은 메시지를 확인할 수 있습니다. 필요하다면 내용을 수정할 수 있습니다. 기본 값으로 설정하고, ESC 키를 누른 다음 :wq를 입력하여 저장한 다음, 빠져 나갑니다.

Merge branch 'subject'

Please enter a commit message to explain why this merge is necessary,

especially if it merges an updated upstream into a topic branch.

#

Lines starting with '#' will be ignored, and an empty message aborts

the commit.

\$ git merge subject

≡ 브랜치 간의 병합

≡ 동일 문서 내의 다른 위치를 변경하기

최종 파일의 내용을 확인합니다.

cat 명령어를 이용하여 happydata.txt 파일의 내용을 확인해 봅니다.

양쪽에서 작업했던 내용이 자연스럽게 하나의 파일로 합쳐진 것을 볼 수 있습니다.

```
$ cat happydata.txt
```

```
#master area
```

```
master01
```

```
master02
```

```
#subject area
```

```
subject01
```

```
subject02
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

동일 문서임과 동시에 동일 라인 위치를 변경하는 실습에 대하여 살펴 봅니다.

‘branchtest04’라는 이름의 git 저장소를 생성합니다. 이번 실습은 서로 다른 branch에 동일한 이름의 파일 equalline.txt를 가지고 있고, 동일한 문서내에서 동일 위치를 수정하고 병합시 어떻게 되는지 살펴 보도록 하겠습니다.

equalline.txt 파일

라인 번호	master 브랜치	subject 브랜치
1	#master area	#master area
2	master01	master01
3	master02	subject02
4		
5	#subject area	#subject area
6	subject01	subject01

구현 되는 순서

master 브랜치	subject 브랜치
equalline.txt 파일 생성 후 커밋	x
x	subject 브랜치 신규 생성됨
equalline.txt 파일 수정 후 커밋	x
subject 브랜치로 checkout →	
x	equalline.txt 파일 수정 후 커밋 master 브랜치와 동일한 라인
← master 브랜치로 checkout	
merge 기능 사용하기 버전 충돌 메시지 확인 및 수정 다시 커밋	x

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

동일 문서임과 동시에 동일 라인 위치를 변경하는 실습에 대하여 살펴 봅니다.

‘branchtest04’라는 이름의 깃 저장소를 생성합니다. 이번 실습은 서로 다른 branch에 동일한 이름의 파일 equalline.txt를 가지고 있고, 동일한 문서내에서 동일 위치를 수정하고 병합시 어떻게 되는지 살펴 보도록 하겠습니다.

```
$ git init
Initialized empty Git repository in D:/branchtest04/.git/

$ ls -al
total 16
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:43 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:42 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 28 15:43 .git/
```


≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

실습을 위한 신규 파일을 생성하고, 커밋을 수행합니다.

master 브랜치에서 equalline.txt 파일을 생성한 후, 다음과 같은 내용을 추가한 다음, 저장합니다.
equalline.txt 파일을 stage에 올리고, 커밋을 수행합니다.

파일의 내용

```
#master area  
master01
```

```
#subject area  
subject01
```

```
$ git branch subject  
  
$ git add equalline.txt  
  
$ git commit -m 'master equalline commit 01'  
[master (root-commit) 5b98f9a] master equalline commit 01  
1 file changed, 6 insertions(+)  
create mode 100644 equalline.txt
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

신규 브랜치를 생성하고, 텍스트 파일을 수정한 다음 커밋을 진행합니다.

‘subject’라는 브랜치를 만듭니다. master 브랜치에서 equalline.txt 파일의 내용을 다음과 같이 수정합니다.

파일의 내용

```
#master area  
master01  
master02  
  
#subject area  
subject01
```

```
$ git commit -am 'master equalline commit 02'  
[master (root-commit) 5b98f9a] master equalline commit 02  
1 file changed, 6 insertions(+)  
create mode 100644 equalline.txt
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

신규 브랜치를 생성하고, 텍스트 파일을 수정한 다음 커밋을 진행합니다.

subject 브랜치로 이동하여 equalline.txt 파일을 다음과 같이 수정합니다.

equalline.txt 파일을 열고, 다음과 같이 수정합니다.

즉, 동일한 내용에 서로 다른 내용을 작성하는 경우에 해당합니다.

```
$ git checkout subject  
Switched to branch 'subject'
```

파일의 내용

```
#master area  
master01  
subject02
```

```
#subject area  
subject01
```

```
$ git commit -am 'subject equalline commit 02'  
[subject 2d2d82b] subject equalline commit 02  
1 file changed, 1 insertion(+), 1 deletion(-)
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

다음과 같이 파일을 수정하고, 커밋을 진행합니다.

master 브랜치로 체크 아웃을 합니다. 현재 상태는 equalline.txt 파일이 양쪽에서 모두 수정이 되었고, 동일한 라인 위치에서 수정이 되었습니다.

```
$ git checkout master
Switched to branch 'master'

$ cat equalline.txt
#master area
master01
master02

#subject area
subject01
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

git merge 명령어를 사용하면 다음과 같은 충돌 메시지를 확인할 수 있습니다.

git merge 명령어를 사용하여 subject 브랜치의 내용을 master 브랜치로 끌어 들입니다. 다음과 같은 메시지를 확인할 수 있습니다. 두 개의 브랜치를 병합하려고 시도하는 동안 병합 충돌(Merge conflict)이 발생하였음을 알려 주고 있습니다. 왜냐하면 동일한 파일을 동일 라인에 서로 다른 내용을 들어 있어서 충돌하고 있기 때문입니다. 충돌이 발생하였으므로, 이 부분이 해결되어야만 병합할 수 있습니다.

```
$ git merge subject
Auto-merging equalline.txt
CONFLICT (content): Merge conflict in equalline.txt
Automatic merge failed; fix conflicts and then commit the result.
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

파일에 어떠한 충돌 상황이 있는 지 확인해 봅니다.

equalline.txt 파일을 우선 열어 보겠습니다.

‘<<<<<< HEAD’와 ‘=====’ 사이의 내용은 master 브랜치에서 작업했던 내용입니다.

‘=====’와 ‘>>>>>> subject’ 사이의 내용은 subject 브랜치에서 작업했던 내용입니다.

파일의 내용

```
#master area
master01
<<<<<< HEAD
master02
=====
subject02
>>>>>> subject

#subject area
subject01
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

개발자가 원하는 데로 적절히 수정을 가합니다.

내용을 확인한 다음, 개발자가 구현하고자 하는 대로 파일을 수정하도록 합니다.

파일의 내용

```
#master area  
master01  
master02  
subject02  
#subject area  
subject01
```

≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

최종 확인후 커밋을 수행합니다.

수정된 equalline.txt를 스테이징 후 커밋을 하도록 합니다.

```
$ git commit -am 'master subject equalline commit finished'
```

```
[master ad9053d] master subject equalline commit finished
```


≡ 브랜치 간의 병합

≡ 동일 문서 내의 동일 위치를 변경하기

다음 명령어로 최종 결과를 확인하도록 합니다.

지금까지 만든 브랜치와 커밋의 관계를 한눈에 확인 가능합니다.

```
$ git log --oneline --branches --graph
$ git log --oneline --branches --graph
*   ad9053d (HEAD -> master) master subject equalline commit finished
|\
| * 2d2d82b (subject) subject equalline commit 02
* | c23bd10 master equalline commit 02
|/
* 5b98f9a master equalline commit 01
```

깃허브(GitHub)

깃허브로 협업하기



깃허브로 협업하기

깃허브 계정 등록

다음 사이트에 회원 가입하고, 공유 저장소를 생성합니다.

참조 사이트) <https://github.com/>

Sign in Sign up

Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ someone@naver.com

Create a password
✓

Enter a username
✓ gomdori1970

Would you like to receive product updates and announcements via email?

Type "y" for yes or "n" for no

✓ y

Verify your account

확인

이 퍼즐을 풀어서 귀하가 인간이라는
것을 알 수 있게 해주십시오

확인

You're almost done!

We sent a launch code to someone@naver.com

→ Enter code

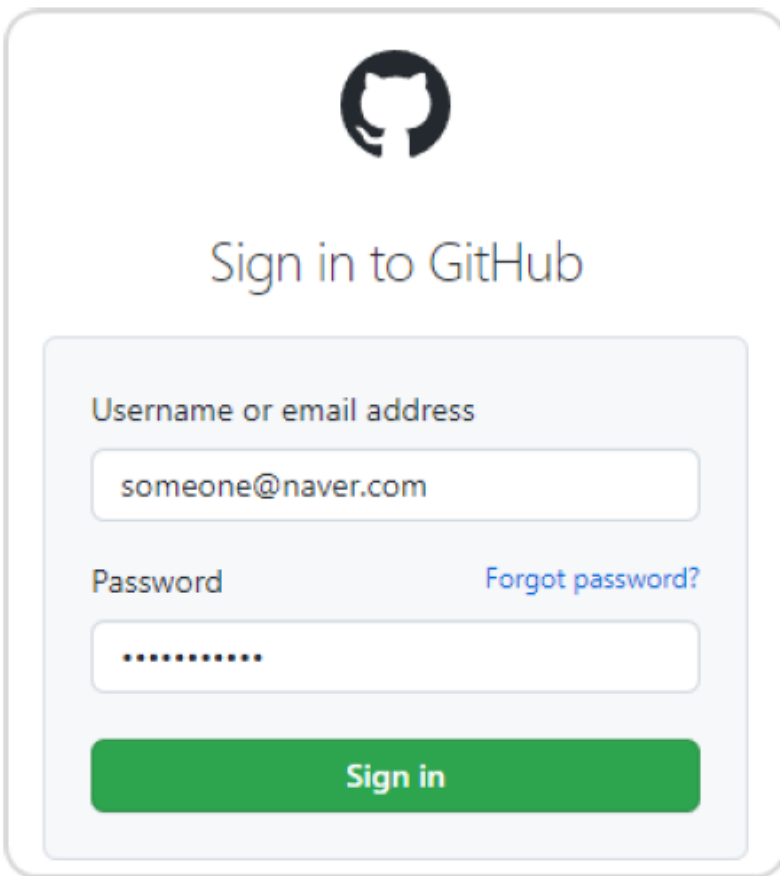
Didn't get your email? [Resend the code](#) or [update your email address](#).

☰ 깃허브로 협업하기

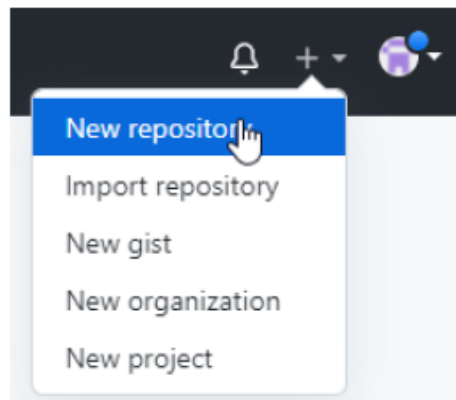
☰ 원격 저장소 만들기

원격 저장소를 생성하기 위하여 깃허브에 다음과 같이 로그인합니다.

로그인이 되지 않았다면 다음과 같이 우측 상단의 [Sign in] 항목을 클릭한 다음 로그인을 우선 진행합니다. 우측 상단의 [New repository] 메뉴를 클릭하여 새로운 저장소(repository)를 생성하도록 합니다.



The image shows the GitHub sign-in page. At the top is the GitHub logo. Below it is the text "Sign in to GitHub". There are two input fields: "Username or email address" with the placeholder text "someone@naver.com" and "Password" with masked dots. To the right of the password field is a link "Forgot password?". At the bottom is a green "Sign in" button.



깃허브로 협업하기

원격 저장소 만들기

원격 저장소는 다음과 같이 생성하면 됩니다.

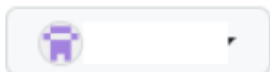
Owner는 사용자의 영문 이름으로 자동르로 채워집니다. Repository name은 필수 입력 사항으로 저장소의 이름이 됩니다. Description 항목은 옵션 사항으로 저장소에 대한 세부 설명을 입력하면 됩니다. Public/Private은 저장소에 대한 공개 여부를 지정하는 라디오 버튼입니다. 그림과 같이 원격 저장소(remote repository)를 생성하도록 합니다.

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 *

/ githubexam ✓

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

Description (optional)

This is my first github test.~~wow!!



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.

Owner는 사용자의 영문 이름으로 자동르로 채워집니다.

Repository name은 필수 입력 사항으로 저장소의 이름이 됩니다.

Description 항목은 옵션 사항으로 저장소에 대한 세부 설명을 입력하면 됩니다.

Public/Private은 저장소에 대한 공개 여부를 지정하는 라디오 버튼입니다.

깃허브로 협업하기

원격 저장소 만들기

원격 저장소는 다음과 같이 생성하면 됩니다.

모든 사항을 입력한 다음 하단의 [Create repository] 버튼을 클릭하여 저장소를 생성하도록 합니다.

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

항목	설명
Add a README file	프로젝트를 소개하는 설명문을 작성한 파일을 추가하는 옵션입니다.
Add .gitignore	깃허브에서 추적할 수 없는 파일의 리스트를 입력할 수 있는 파일을 생성합니다.

깃허브로 협업하기

원격 저장소 만들기

원격 저장소는 다음과 같이 생성하면 됩니다.

생성한 저장소 정보를 확인하도록 합니다. Git Repo의 주소는 'https://github.com/(아이디)/(repo 이름).git'의 형식으로 만들어 집니다.

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#)

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# repository03" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/seoljinuk/repository03.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/seoljinuk/repository03.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

≡ 깃허브로 협업하기

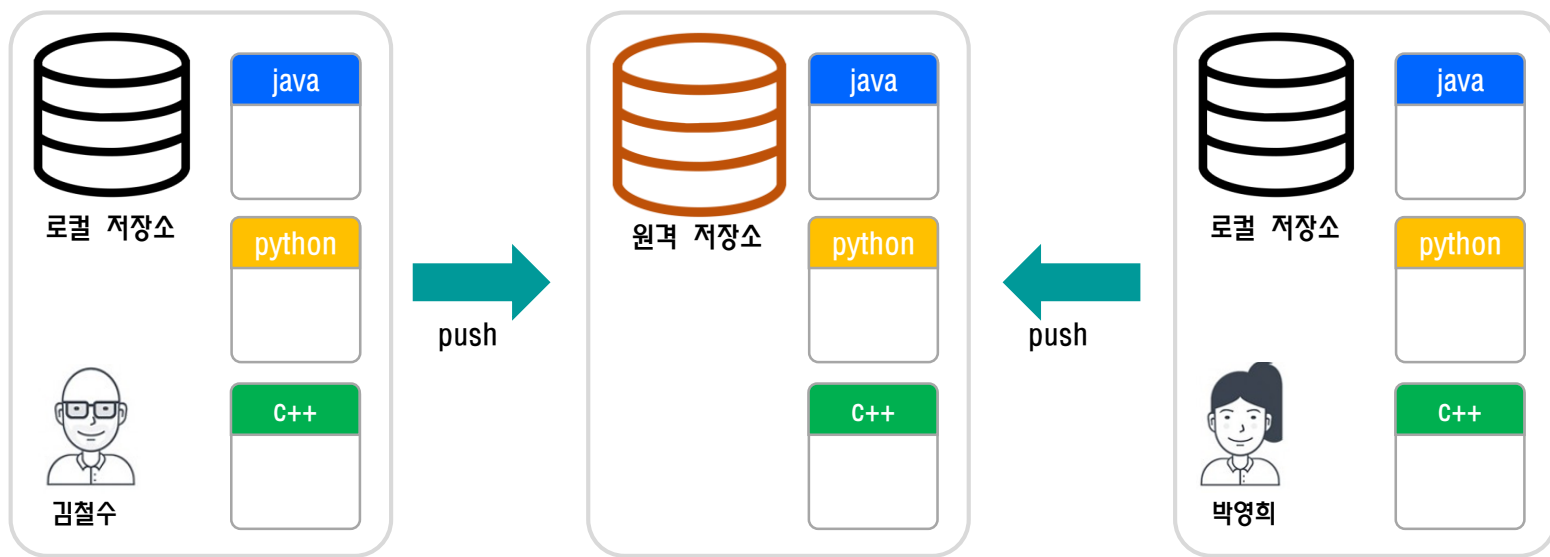
≡ 원격 저장소에 Push하기

로컬 저장소의 데이터를 원격 저장소로 업로드하는 동작을 의미합니다.

내 PC의 로컬 저장소에서 변경된 이력을 원격 저장소에 공유하려면, 로컬 저장소의 변경 이력을 원격 저장소에 업로드해야 합니다.

웹 상의 원격 저장소로 변경된 파일을 업로드하는 것을 Git에서는 푸시(Push)라고 합니다.

push 명령어를 실행하면, 원격 저장소에 내 변경 이력이 업로드되어, 원격 저장소와 로컬 저장소가 동일한 상태가 됩니다.



≡ 깃허브로 협업하기

≡ 원격 저장소에 Push하기

이전에 실습하였던 branchtest01 폴더에서 원격 저장소에 push하는 과정을 실습하도록 하겠습니다.

일반적으로 원격 저장소의 이름은 매우 길어서 매번 입력하기가 수월하지 않습니다. 이러한 긴 이름의 원격 저장소의 주소를 특정한 이름 형태의 별칭으로 등록해 두면 매번 입력할 필요가 없으므로 편리하게 사용할 수 있습니다. 콘솔 모드에서 push 혹은 pull 명령어 사용시 원격 저장소 이름을 명시하지 않으면, 기본 값으로 'origin'이라는 이름을 사용합니다. 통상적으로 'origin'은 사용자가 업로드하고자 하는 원격 저장소의 관례적인 이름으로 많이 사용됩니다. 그럼, 'origin'이라는 이름으로 원격 저장소를 등록시키고 push를 진행하도록 합니다.

remote 명령어 사용 형식

원격 저장소를 등록하려면, remote 명령어를 사용합니다. 사용 명령어 형식은 다음과 같습니다.

<name>은 등록할 원격 저장소의 이름을, <url>은 원격 저장소의 URL을 지정하면 됩니다.

```
git remote add <name> <url>
```

다음 명령어를 사용하여, 원격 저장소의 URL을 origin이라는 이름으로 등록하도록 합니다.

확장자 '.git'은 명시하지 않도록 합니다.

참고 사항으로 이미 등록된 원격 저장소를 제거하려면 remove 명령어를 사용하면 됩니다.

'origin'은 이미 등록되었던 원격 저장소의 이름입니다.

```
$ git remote add origin https://github.com/username/githubexam
$ git remote remove origin # 이걸 실습하지 않습니다.
```

≡ 깃허브로 협업하기

≡ 원격 저장소에 Push하기

원격 저장소에 데이터를 업로드하려면 push 명령어를 사용합니다.

push 명령어 사용 형식

<repository>는 push하고자 하는 경로의 주소, 즉 등록된 원격(remote)의 git 이름 정보를 의미합니다.

저장소의 이름인 'origin'을 입력하면 됩니다. <refspec>에는 push하고자 하는 브랜치 이름을 작성하면 됩니다.

```
git push <repository> <refspec> ...
```

다음 명령어를 실행해 원격 저장소 origin에 커밋을 push합니다.

```
$ git push origin master
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (12/12), 863 bytes | 287.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/seoljinuk/githubexam
* [new branch] master -> master
```

☰ 깃허브로 협업하기

☰ 원격 저장소에 Push하기

GitHub에 접속하여 다음과 같이 파일이 업로드 되었는 지 확인합니다.

The screenshot shows a GitHub repository interface. At the top, there are buttons for 'master' (with a dropdown arrow), '1 branch', and '0 tags'. Below this, a commit is shown for 'oraman master commit04' with a file 'somefile.txt' listed. A light blue box contains the text: 'Help people interested in this repository understand your project by adding a README file.' On the right side, there are buttons for 'Go to file', 'Add file', and a green 'Code' button with a dropdown arrow. The 'Code' dropdown menu is open, showing options for 'Local' and 'Codespaces' (with a 'New' button). Under 'Local', there is a 'Clone' section with tabs for 'HTTPS', 'SSH', and 'GitHub CLI'. The 'HTTPS' tab is selected, showing the URL 'https://github.com/seoljinuk/githubexam.git' with a copy icon. Below the URL, it says 'Use Git or checkout with SVN using the web URL.' There is also an option to 'Open with GitHub Desktop' and a 'Download ZIP' option at the bottom, which is being pointed to by a purple hand cursor.

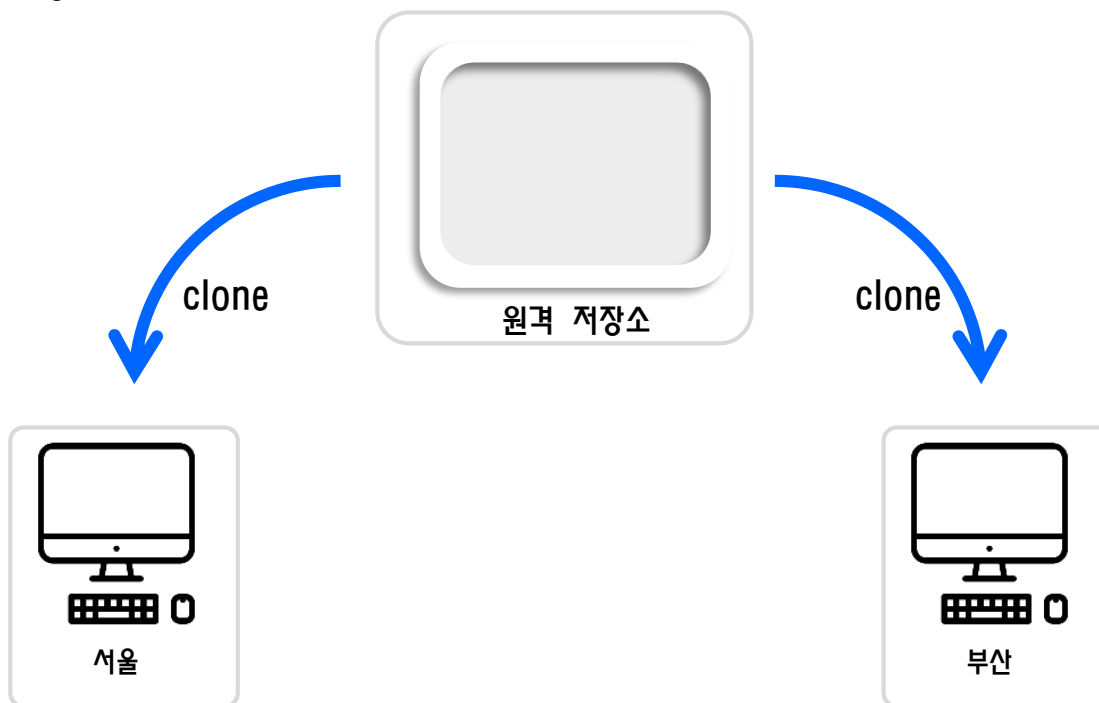
☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 원격 저장소 복제하기

서울과 부산에 있는 로컬 저장소에 복제 작업을 수행합니다.

깃허브를 사용하여 협업을 진행하는 방법을 살펴 보겠습니다. 편의상 2명의 개발자가 서로 다른 지역인 서울과 부산에서 작업을 한다고 가정하겠습니다. 하나의 깃 계정을 사용하여 둘 이상의 컴퓨터에서 원격 저장소를 사용하여 공유해 버전을 관리해 보도록 하겠습니다.

서울과 부산에 있는 개발자는 코딩을 하기 위하여 원격 저장소의 내용을 로컬 저장소로 가져 와야 합니다. 원격 저장소의 내용을 그대로 로컬 저장소로 가져 오는 것을 '복제(clone)'라고 합니다. '저장소를 복제한다'라고 표현합니다.



☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 테스트 시나리오

다음과 같은 순서대로 테스트를 진행해 보도록 하겠습니다.

구현 되는 순서

순번	서울(Seoul)	원격(Remote)	부산(Pusan)
1	원격으로부터 복제함(Clone)	← 데이터 복제 →	원격으로부터 복제함(Clone)
2	somefile.txt 생성 및 커밋 후 Push하기 →		
3	원격지 파일 확인(in browser)		
4	→ somefile.txt Pulling 하기		
5	← somefile.txt 수정 및 커밋 후 Push하기		
6	원격지 파일 확인(in browser)		
7	somefile.txt 수정 및 커밋하기 원격지 파일 Pulling 하기 ← 버전 충돌 메시지 확인 somefile.txt 수정 및 커밋하기 Push하기 →		
8	원격지 파일 확인(in browser)		

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 원격 저장소 복제하기

서울과 부산에 있는 로컬 저장소에 복제 작업을 수행합니다.

이전에 만들어 두었던 원격 저장소 'githubexam'를 활용하도록 하겠습니다. 이 저장소를 '서울'과 '부산' 로컬 저장소에 각각 복제하도록 해보겠습니다.

서울('seoul') 저장소와 부산('pusan') 저장소는 미리 생성해 둘 필요는 없습니다. clone 명령어는 다음과 같습니다.

형식에서 [dirname]은 옵션 사항인데, 명시하지 않으면 repository 이름과 동일한 디렉토리가 생성됩니다.

이 예시에서는 githubexam라는 디렉토리가 생성이 됩니다.

clone 명령어 사용 형식

```
git clone https://github.com/사용자이름/githubexam.git [dirname]
```

D 드라이브에서 마우스 우측 클릭을 하여 git bash를 실행하도록 합니다.

다음 명령어를 사용하게 되면 seoul 디렉토리가 생성이되면서 로컬 저장소가 됩니다.

seoul과 동일한 방식으로 pusan도 복제를 하도록 합니다.

```
$ cd /d # 최상위 d 드라이브로 이동
$ git clone https://github.com/사용자이름/githubexam.git seoul
Cloning into 'seoul'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 원격 저장소 복제하기

터미널에서 `ls -al` 명령어를 사용하여 디렉토리들을 확인해 봅니다.

각각의 디렉토리로 이동을 하여 현재 파일 목록을 확인하고, `git log` 명령어를 각각 수행해 봅니다.

```
$ cd seoul
$ ls -al
total 17
drwxr-xr-x 1 kosmo 197121 0 Nov 29 16:58 ./
drwxr-xr-x 1 kosmo 197121 0 Nov 29 16:58 ../
drwxr-xr-x 1 kosmo 197121 0 Nov 29 16:58 .git/
-rw-r--r-- 1 kosmo 197121 32 Nov 29 16:58 somefile.txt
```

```
$ git log
```

이전 `branchtest01`에서 작업했던 커밋의 내역이 그대로 보이는 지 확인하도록 합니다.

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 서울 컴퓨터에서 작업하고 올리기

서울 컴퓨터에서 파일을 수정한 다음 push 작업을 수행해 봅니다.

서울 로컬 저장소에서 파일 작업을 진행합니다. 이것을 원격 저장소에 push 작업을 수행해 보겠습니다.



☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 서울 컴퓨터에서 작업하고 올리기

서울 컴퓨터에서 somefile.txt 파일 수정 후, 커밋 및 push 작업을 수행해 봅니다.

해당 파일을 스테이징하고 커밋을 진행하겠습니다.

파일의 내용

... 기존 코드 변경 없음

hello05

```
$ git commit -am 'somefile first commit in seoul'
[master 6a86f8a] somefile first commit in seoul
1 file changed, 1 insertion(+)

$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 269 bytes | 269.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/seoljinuk/githubexam.git
ab84d4a..6a86f8a master -> master
```


☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 서울 컴퓨터에서 작업하고 올리기


깃허브에서 바뀐 커밋 메시지와 파일의 내용을 확인합니다.

깃허브의 원격 저장소에서 커밋이 잘 이루어 졌는 지 커밋 문구를 확인해 봅니다.





🔗 master ▾ [githubexam](#) / somefile.txt Go to file ...

 **oraman** somefile first commit in seoul

Latest commit 5d51841 1 minute ago [🕒 History](#)

 0 contributors

5 lines (5 sloc) | 40 Bytes

Raw Blame    

```
1 hello01
2 hello02
3 hello03
4 hello04
5 hello05
```

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 부산 컴퓨터에서 내려 받아서 작업하기

부산 컴퓨터는 원격 저장소에서 최신 커밋 정보를 읽어 와야 합니다.

서울 저장소에서 커밋 후 푸시 작업이 이루어 졌습니다. 부산 저장소는 서울 저장소의 수정 내역이 현재 반영이 되어 있지 않습니다. 따라서, 부산 저장소에서는 원격 저장소의 새로운 내용을 가져 와야 합니다.



☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 부산 컴퓨터에서 내려 받아서 작업하기

부산 컴퓨터는 원격 저장소에서 최신 커밋 정보를 읽어 와야 합니다.

git pull 명령어를 사용하여 원격 저장소의 새로운 커밋 정보를 읽어 와야 합니다.

우선 현재 파일의 상황을 확인해 보면 다음과 같습니다.

파일의 내용

```
hello01  
hello02  
hello03  
hello04
```

git pull 명령어를 사용하여 다음과 정보를 읽어 옵니다.

```
$ git pull  
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 248 bytes | 1024 bytes/s, done.  
From https://github.com/seoljinuk/githubexam  
805aa0e..5d51841 master -> origin/master  
Updating 805aa0e..5d51841  
Fast-forward  
somefile.txt | 1 +  
1 file changed, 1 insertion(+)
```

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 부산 컴퓨터에서 내려 받아서 작업하기

부산 컴퓨터는 원격 저장소에서 최신 커밋 정보를 읽어 와야 합니다.

최신 정보를 읽어 오기 위하여 pull 작업을 진행하였으므로, somefile.txt 파일은 최신 파일로 업데이트 되어야 합니다.

다음과 같이 cat 명령어를 사용하여 somefile.txt 파일의 내용을 확인합니다.

```
$ cat somefile.txt  
hello01  
hello02  
hello03  
hello04  
hello05
```

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 부산 컴퓨터에서 업로드하기

부산 컴퓨터에서 파일을 수정한 다음 push 작업을 수행해 봅니다.



☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ 부산 컴퓨터에서 업로드하기

부산 컴퓨터에서 파일을 수정한 다음 push 작업을 수행해 봅니다.

다음과 같이 파일을 수정하고, 데이터를 푸시해 보겠습니다.

파일의 내용

hello01

hello02

hello world

```
$ git commit -am 'somefile first commit in pusan'
```

```
[master 1c2566e] somefile first commit in pusan
```

```
1 file changed, 1 insertion(+), 3 deletions(-)
```

```
$ git push
```

```
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.
```

```
Writing objects: 100% (3/3), 269 bytes | 269.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/seoljinuk/githubexam.git
```


```
6a86f8a..1c2566e master -> master
```


☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기


☰ 부산 컴퓨터에서 업로드하기

부산 컴퓨터에서 파일을 수정한 다음 push 작업을 수행해 봅니다.





깃허브의 원격 저장소에서 데이터가 잘 반영이 되었는 지 확인합니다.

 master ▾ [githubexam / somefile.txt](#) Go to file ...

 **oraman** somefile first commit in pusan Latest commit 1c2566e 1 minute ago 🕒 History

 0 contributors

3 lines (3 sloc) | 28 Bytes

Raw Blame    

```
1 hello01
2 hello02
3 hello world
```


☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ without do push, directly pull action

서울 컴퓨터에서 파일을 수정 후 push 없이 바로 pull 작업을 하면 어떻게 되는 지 살펴 보도록 하겠습니다.

부산에서 최신 파일을 push 하였습니다. 이번에는 서울 컴퓨터에서 파일을 수정한 후, push 없이 바로 pull 작업을 하게 되면 어떻게 되는 지 한번 살펴 보겠습니다. 서울에서 우선 파일을 다음과 같이, 수정하고 커밋까지 진행하도록 합니다. 그런 다음, pull 명령어를 사용하면 다음과 같이 충돌(CONFLICT)이 발생합니다.

파일의 내용

```
hello01  
hello02  
welcome my home
```

```
$ git commit -am 'somefile second commit in seoul'
```

```
$ git pull
```

```
...
```

```
From https://github.com/seoljinuk/gitbubexam
```

```
cd3dddb..cba0cd4 master -> origin/master
```

```
Auto-merging somefile.txt
```

```
CONFLICT (content): Merge conflict in somefile.txt
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

not push

pull

원격 저장소

서울

부산

☰ 여러 컴퓨터에서 원격 저장소 함께 사용하기

☰ without do push, directly pull action

서울 컴퓨터에서 파일을 수정 후 push 없이 바로 pull 작업을 하면 어떻게 되는 지 살펴 보도록 하겠습니다.

서울과 부산 두 지역 및 원격 저장소의 파일 내용은 다음과 같았습니다.

서울.somefile.txt

```
hello01
hello02
welcome my home
```

부산.somefile.txt

```
hello01
hello02
hello world
```

원격 저장소

부산과 동일합니다.

somefile.txt 파일의 내용을 살펴 보면 파일 수정이 필요해 보입니다
의도하고자 하는 데로 파일을 수정합니다.

```
$ cat somefile.txt
hello01
hello02
<<<<<<< HEAD
welcome my home
=====
hello world
>>>>>>> 1c2566e244da882b54a812cce6c943fa4d20f35b
```

≡ 여러 컴퓨터에서 원격 저장소 함께 사용하기

≡ without do push, directly pull action

서울 컴퓨터에서 파일을 수정 후 push 없이 바로 pull 작업을 하면 어떻게 되는 지 살펴 보도록 하겠습니다.

<<<<<< 영역과 >>>>>> 영역 사이의 값들이 차이가 나는 영역입니다.

최종 파일은 다음과 같습니다.

```
$ cat somefile.txt  
hello01  
hello02  
welcome my home  
hello world
```

파일이 수정되었으므로, 커밋을 하도록 합니다. 최종 파일을 다시 원격지로 push 하도록 합니다.

```
$ git commit -am 'somefile third commit in seoul'  
  
$ git push
```

깃허브(GitHub)

IntelliJ와 깃허브 연동



☰ IDE와의 연동

☰ 통합 개발 환경과의 연동

통합 개발 환경에서 GitHub를 사용하는 방법에 대하여 살펴 봅니다.

통합 개발 환경

통합 개발 환경(Integrated Development Environment, IDE)이란 공통된 개발자 툴을 하나의 그래픽 사용자 인터페이스(Graphical User Interface, GUI)로 결합하는 애플리케이션을 구축하기 위한 소프트웨어입니다. 예를 들면 자바를 사용할 때에는 이클립스(Eclipse) 또는 인텔리제이(IntelliJ), 파이썬을 사용할 때 파이참(PyCharm) 등을 사용하는 데 이러한 항목들을 IDE라고 합니다.

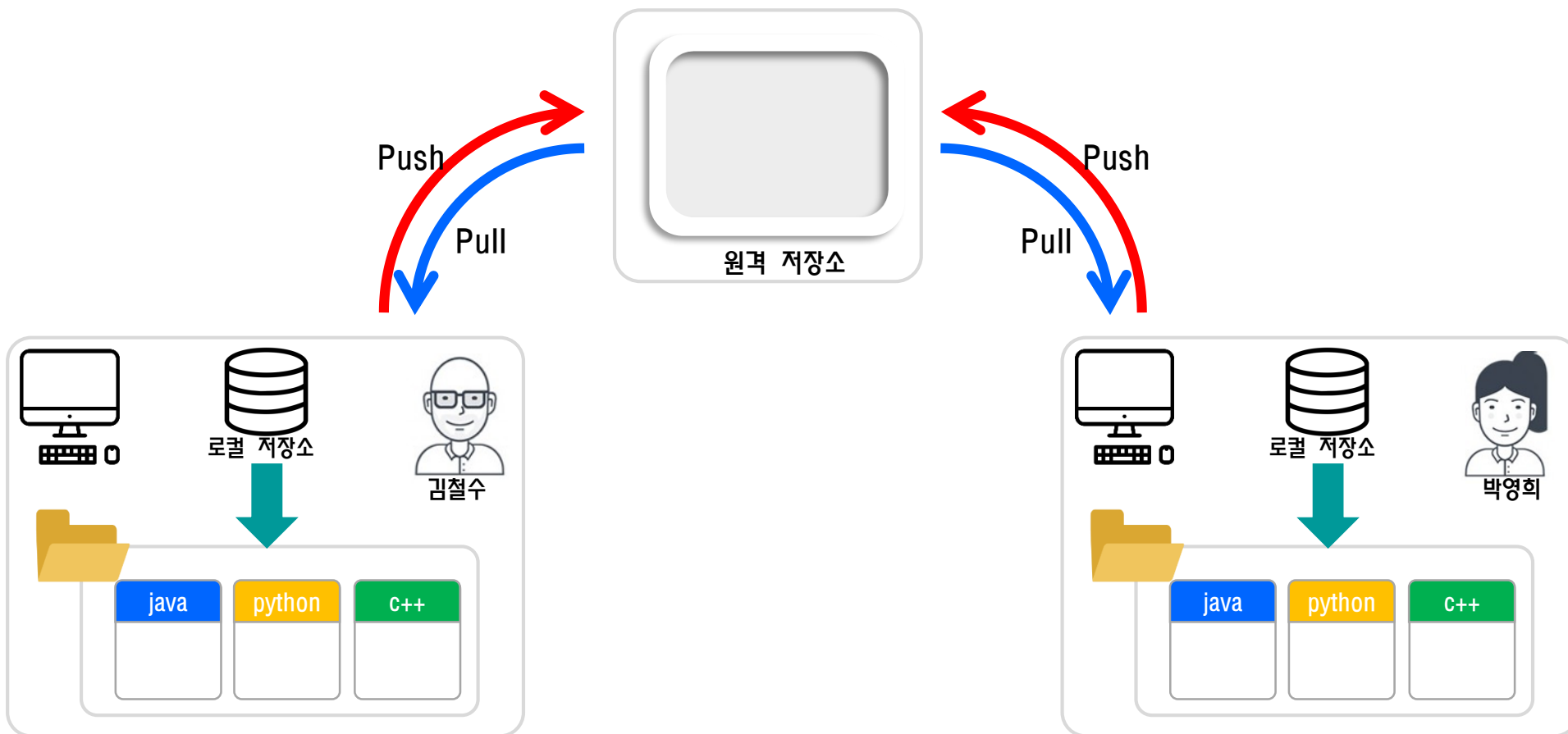


인텔리 제이에서 GitHub 연동하기

IntelliJ와의 연동

IntelliJ를 사용하여 GitHub와 관련된 실습을 수행해보도록 하겠습니다.

원격 저장소에 대한 협업 작업을 진행해보도록 하겠습니다. 개발 인원은 2명이고 각각 '김철수'는 개발 팀장, '박영희'는 개발 팀원이라고 가정하겠습니다. 한 개의 저장소에 대하여 2사람은 각각 로컬 저장소에서 작업한 내용을 원격 저장소로 공유가 가능해야 합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 다음과 같은 순서대로 구현합니다.

인텔리 제이에서 작업할 순서는 다음과 같습니다.

구현 되는 순서

s00	hee
프로젝트 생성하기	x
원격지에 Push하기	x
파일 수정하고 다시 Push하기	x
x	프로젝트 복제하기
x	Welcome 파일 생성 후 Push하기
Welcome 파일 Pull하기	x
Welcome 파일 수정 후 다시 Push하기	x
x	Welcome 파일 Push하기(문제 발생함)
x	Welcome 파일 Pull하기(merge 수행하기)



인텔리 제이에서 GitHub 연동하기

사전 준비 및 공동 작업자 추가하기

공동 작업을 주도하는 '김철수'는 원격 저장소를 다음과 같이 생성하도록 합니다.

인텔리제이를 위한 원격 저장소 레포지터리입니다.


Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** **SSH** 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


...or create a new repository on the command line

```
echo "# collabo_intellij" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/seoljinuk/collabo_intellij.git
git push -u origin main
```



...or push an existing repository from the command line

```
git remote add origin https://github.com/seoljinuk/collabo_intellij.git
git branch -M main
git push -u origin main
```



...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

☰ 인텔리 제이에서 GitHub 연동하기

☰ 신규 프로젝트 생성하기

팀장 '김철수'는 D:\githubtest\s00 프로젝트를 생성하고, 다음과 같이 파일들을 작성하도록 합니다.

ch01.Hello 클래스와 World 클래스를 작성합니다.

```
package ch01;

public class Hello {
    public static void main(String[] args) {
        System.out.println("hello01");
    }
}
```

```
package ch01;

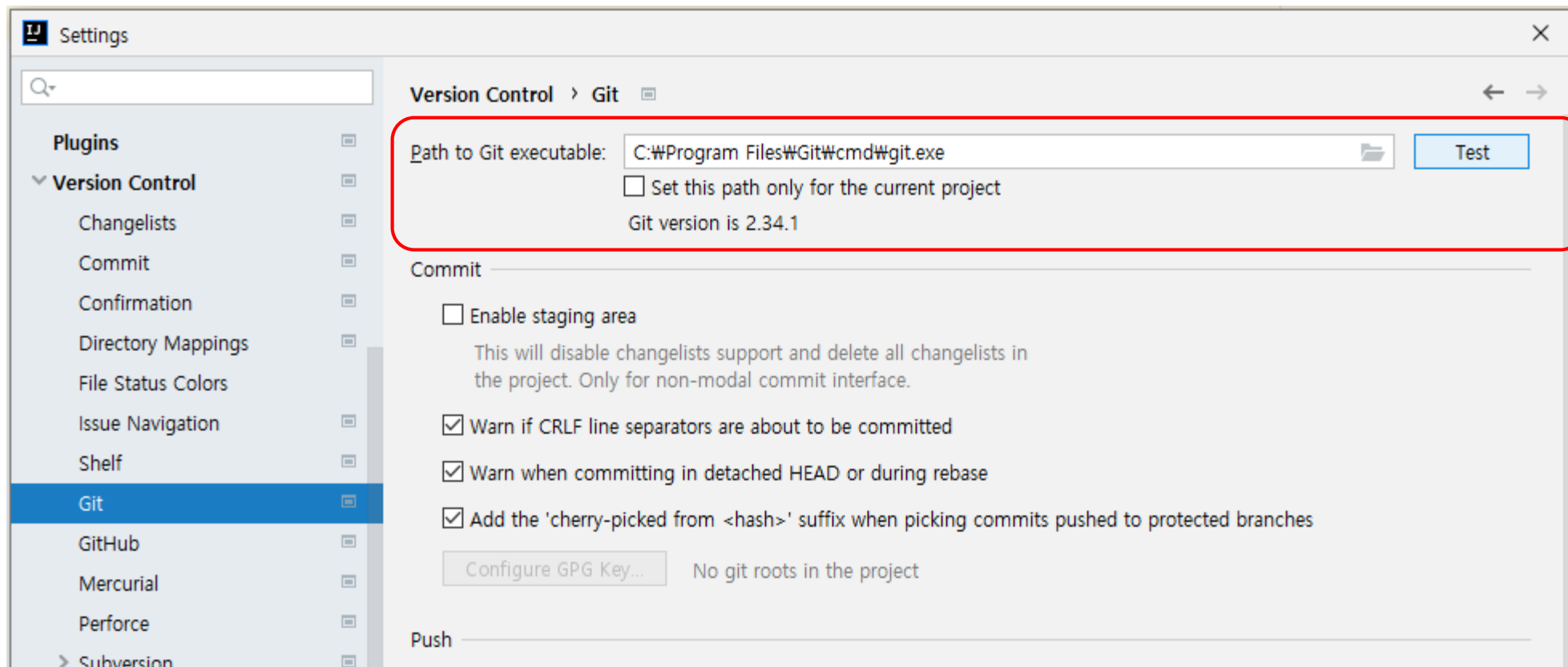
public class World {
    public static void main(String[] args) {
        System.out.println("world01");
    }
}
```

인텔리 제이에서 GitHub 연동하기

Git 실행 파일 연동하기

GitHub에 파일을 Push/Pull 하기 위해서는 git 실행 파일을 IntelliJ와 연동을 해야 합니다.

File-Settings 메뉴를 클릭하여 Version Control 트리 메뉴의 Git으로 이동합니다. [Path to Git executable] 입력 상자에 git.exe 파일의 위치를 지정해 주도록 합니다. [Test] 버튼을 클릭하면 해당 git의 버전을 확인할 수 있습니다.



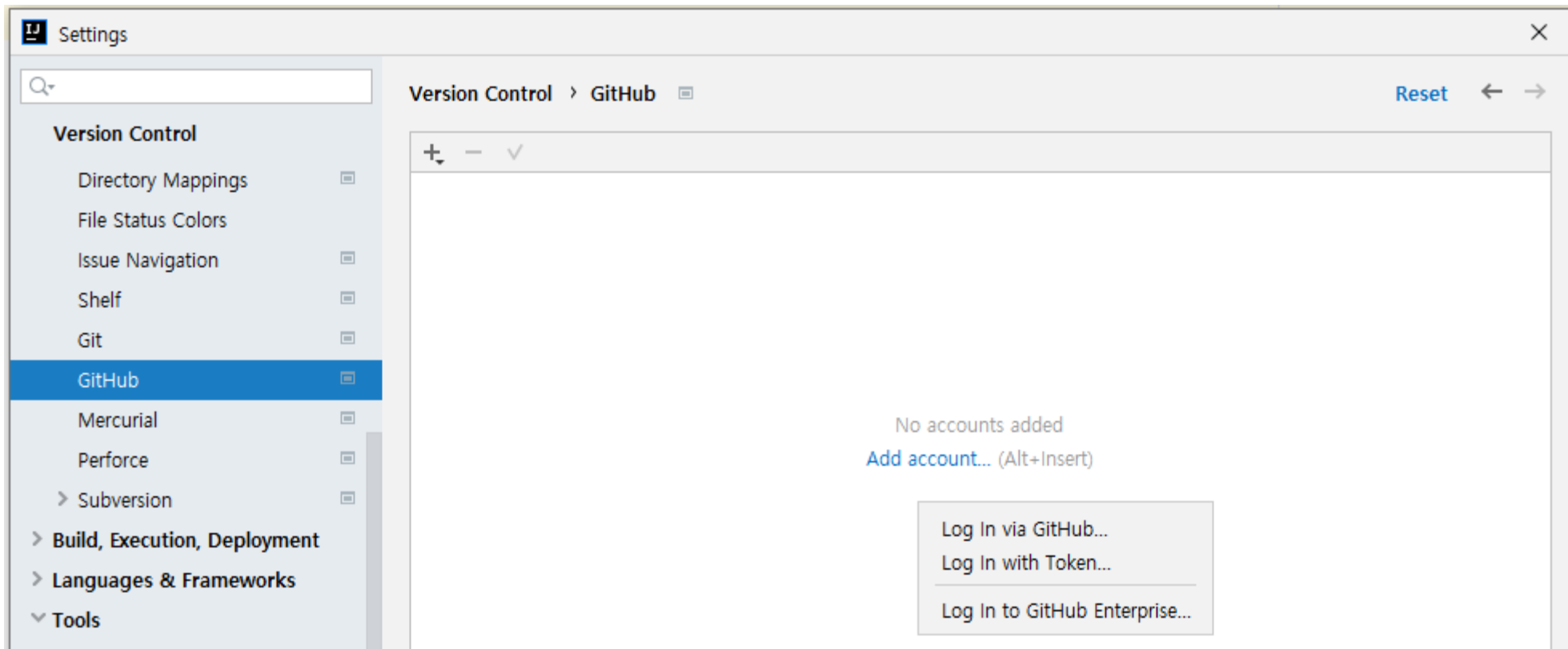
☰ 인텔리 제이에서 GitHub 연동하기

☰ Git 실행 파일 연동하기

Version Control 트리 메뉴의 GitHub 에서 [Add account] 메뉴를 이용하여 GitHub를 등록합니다.

[Add account] 메뉴를 클릭하거나, 단축 키 [Alt + Insert] 키를 누릅니다.

[Log in via GitHub...] 항목을 클릭하면 사용자 인증(Authentication)을 받기 위하여 새로운 웹 페이지가 로딩됩니다.

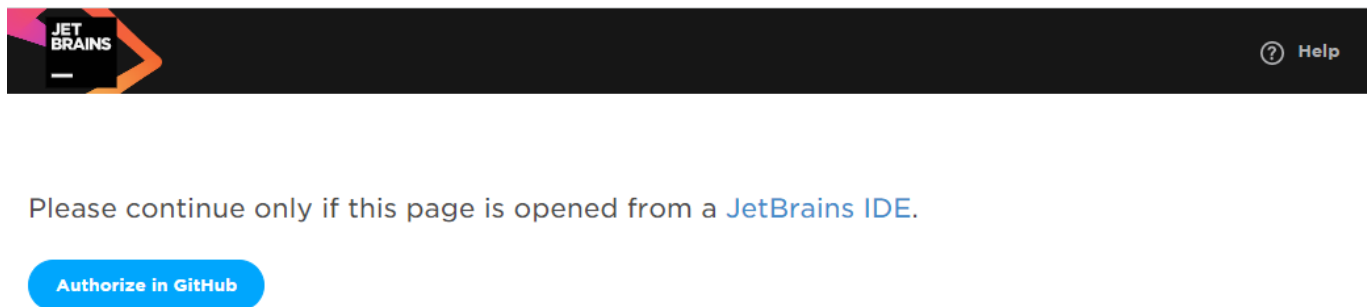


☰ 인텔리 제이에서 GitHub 연동하기

☰ Git 실행 파일 연동하기

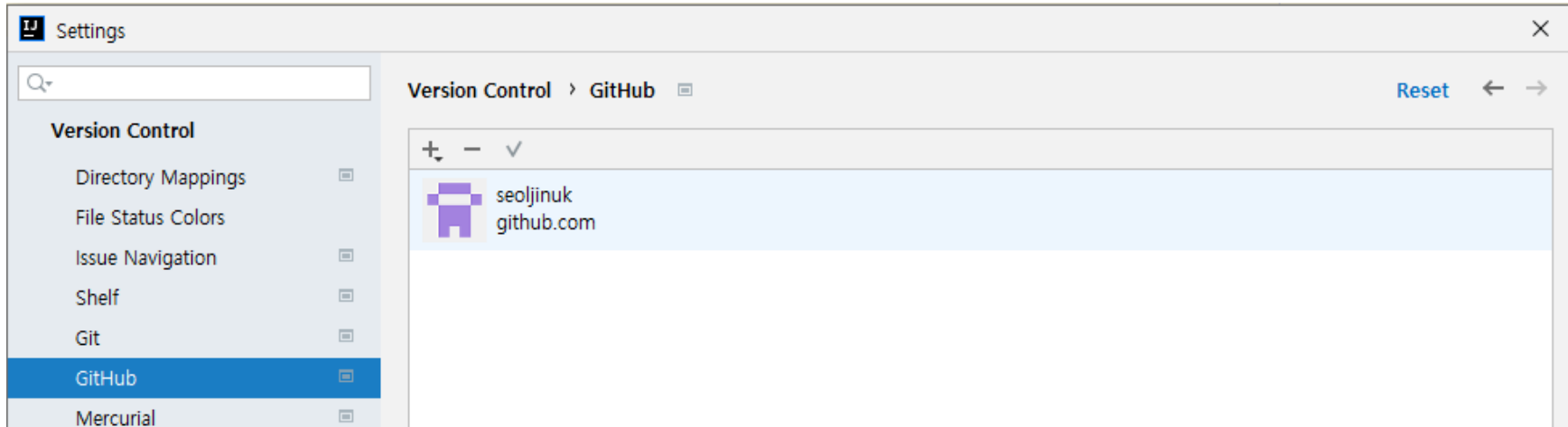
정상적으로 실행이 되면 다음과 같은 메시지를 확인할 수 있습니다.

웹 페이지의 등근 사각형 버튼 [Authorize in GitHub]을 클릭합니다.



정상적으로 실행이 되면 다음과 같은 메시지를 확인할 수 있으며, IntelliJ에 아이콘이 하나 생성되었음을 확인할 수 있습니다.

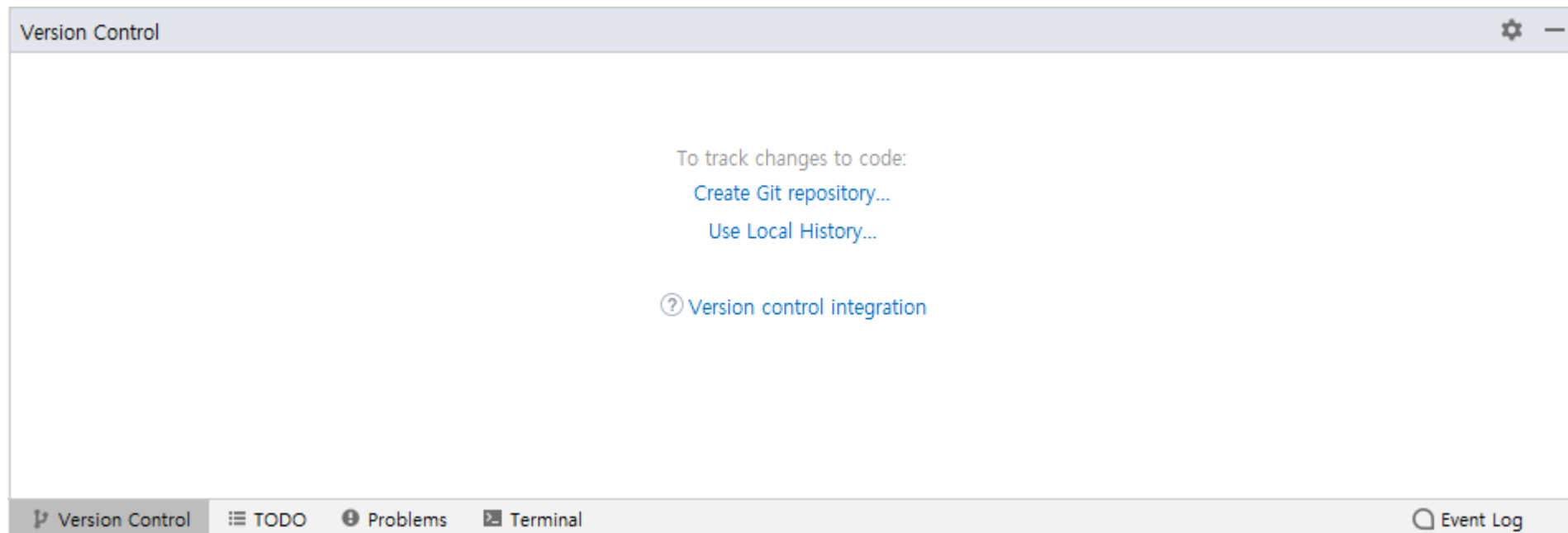
You have been successfully authorized in GitHub. You can close the page.



☰ 인텔리 제이에서 GitHub 연동하기

☰ Git Hub와 연동시키기

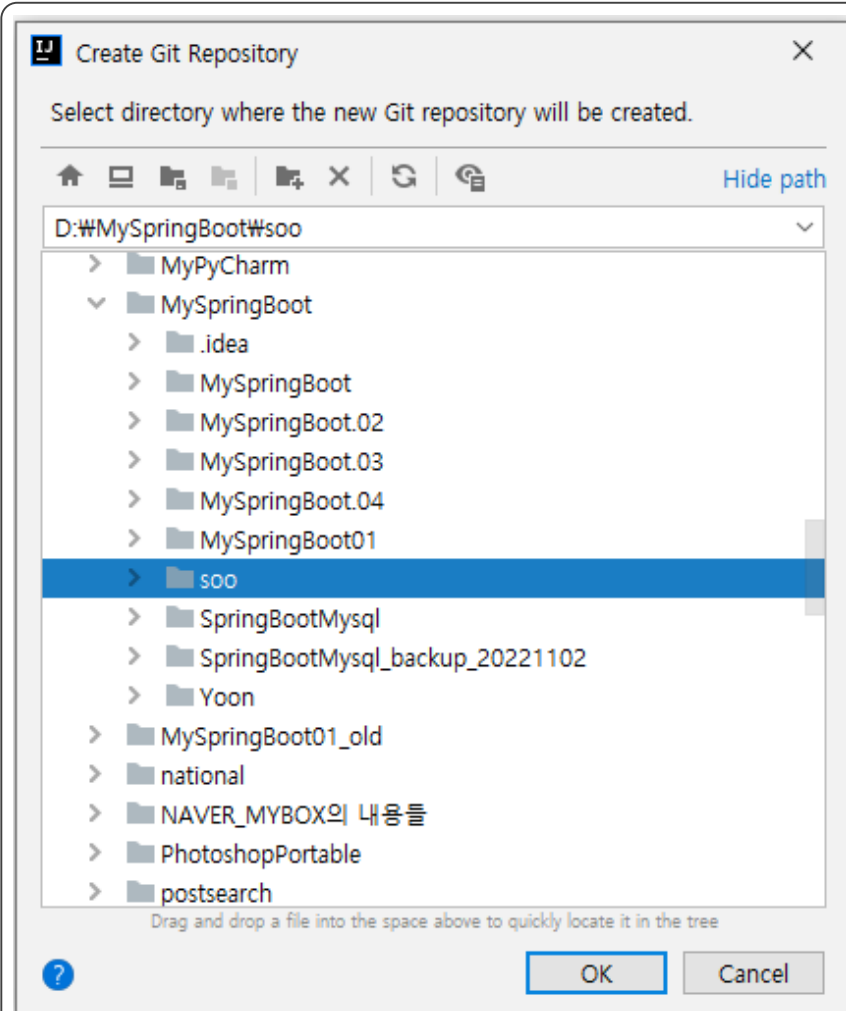
단축키 Alt+9를 눌러서 Version Control 메뉴를 활성화한 다음 [Create Git repository] 링크를 클릭합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ Git Hub와 연동시키기

그림과 같이 Repository 선택 화면이 나오는 데, 해당 프로젝트 soo 폴더를 선택하고 Ok 버튼을 클릭합니다.

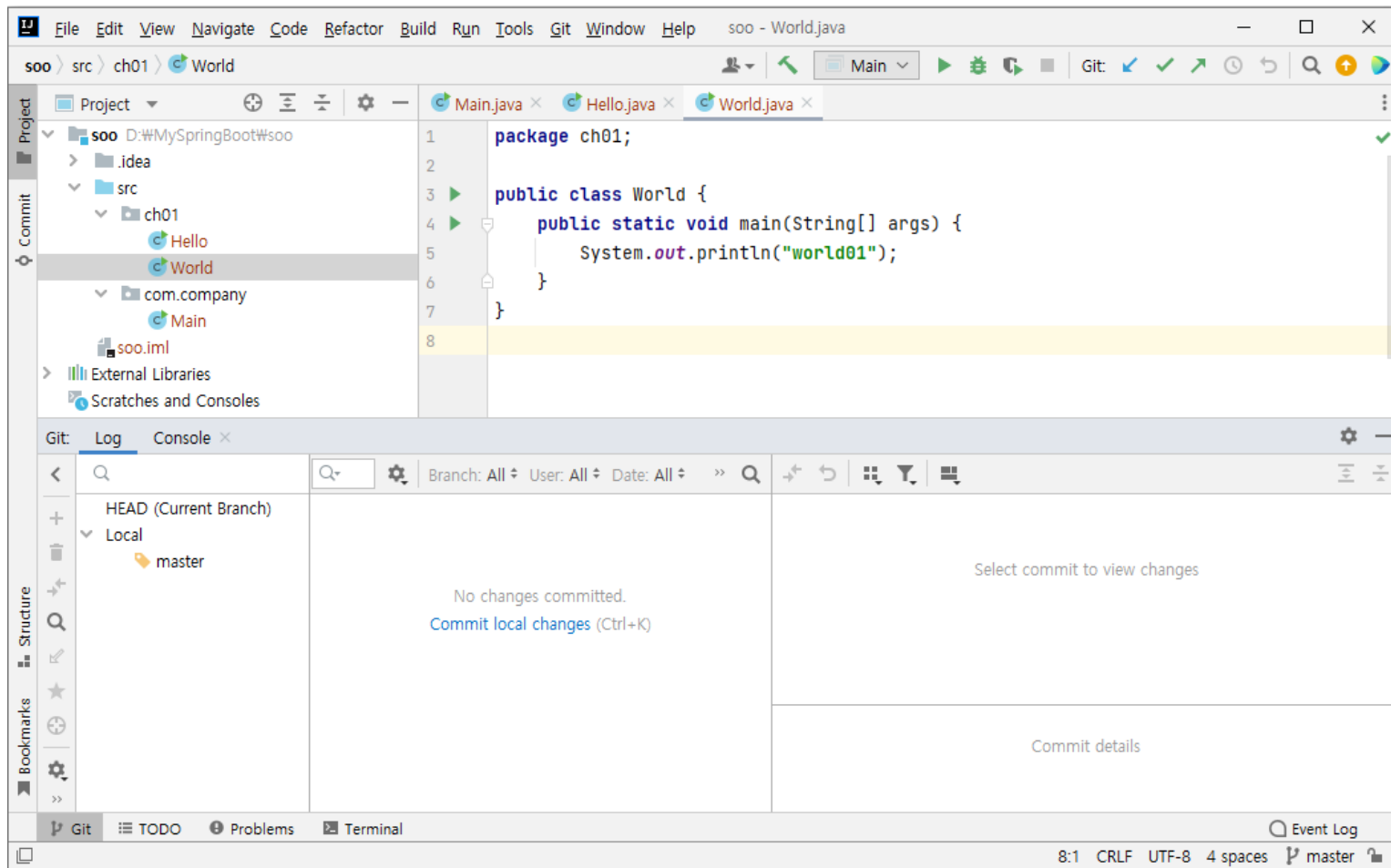


☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

모든 파일을 저장소에 푸시해 보도록 하겠습니다.

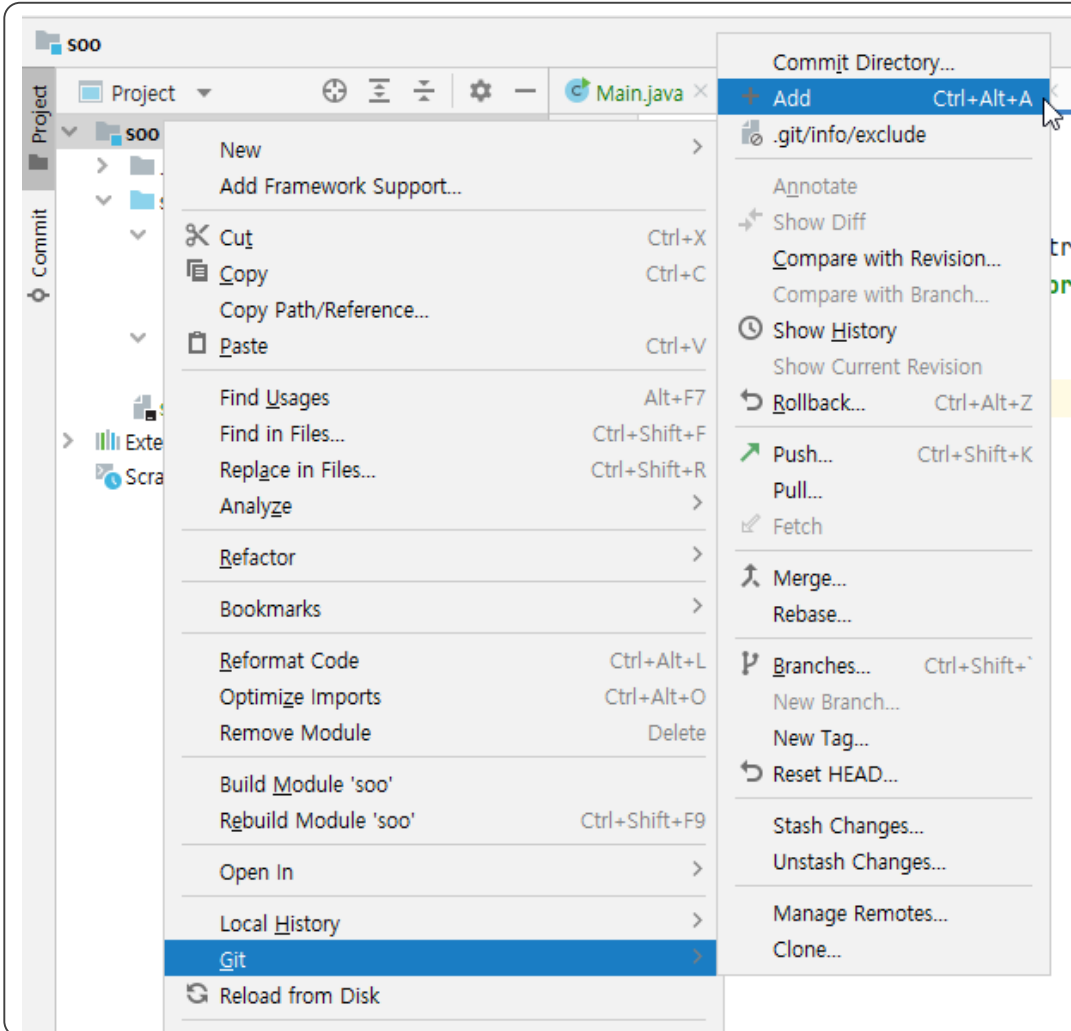
해당 프로젝트에 대하여 깃 저장소를 생성하였습니다. 프로젝트 내의 파일들을 커밋하고 푸시 하는 작업을 진행해 보겠습니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

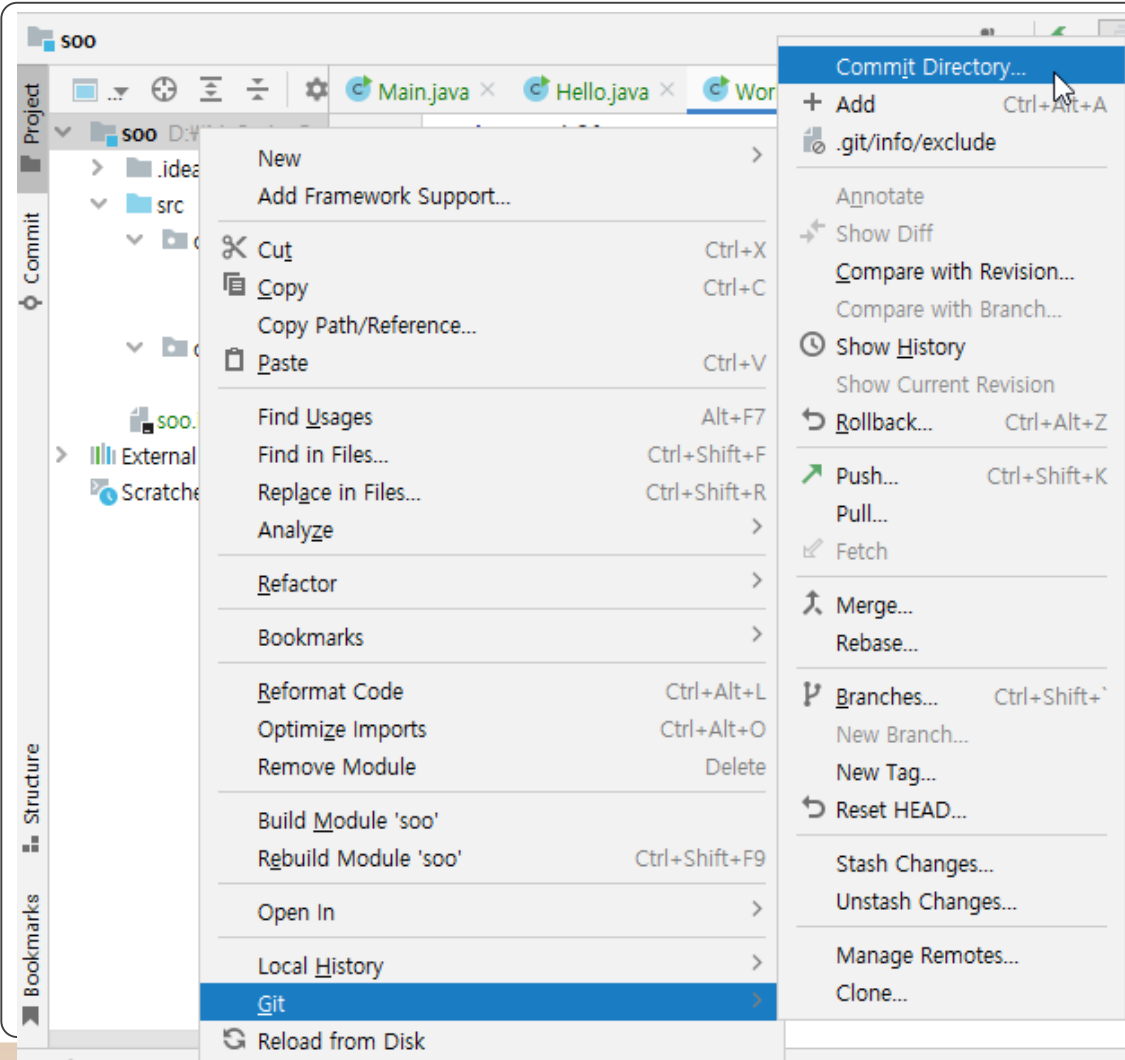
프로젝트를 마우스 우측 클릭한 다음 [Git]-[Add] 메뉴를 클릭하여 모든 파일을 저장소에 추가합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

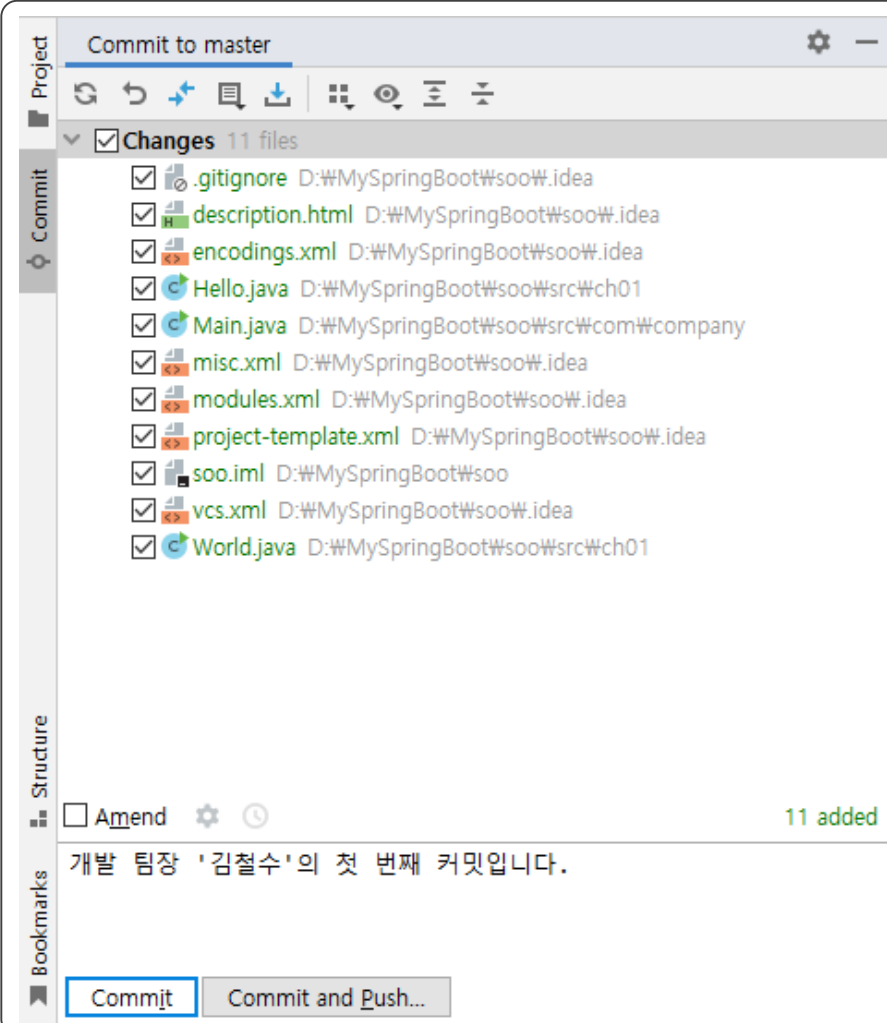
다시 마우스를 우측 클릭하여 [Git]-[Commit Directory] 메뉴를 클릭합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

커밋 메시지 "개발 팀장 '김철수'의 첫 번째 커밋입니다."를 입력 후, [Commit and Push] 버튼을 클릭합니다.

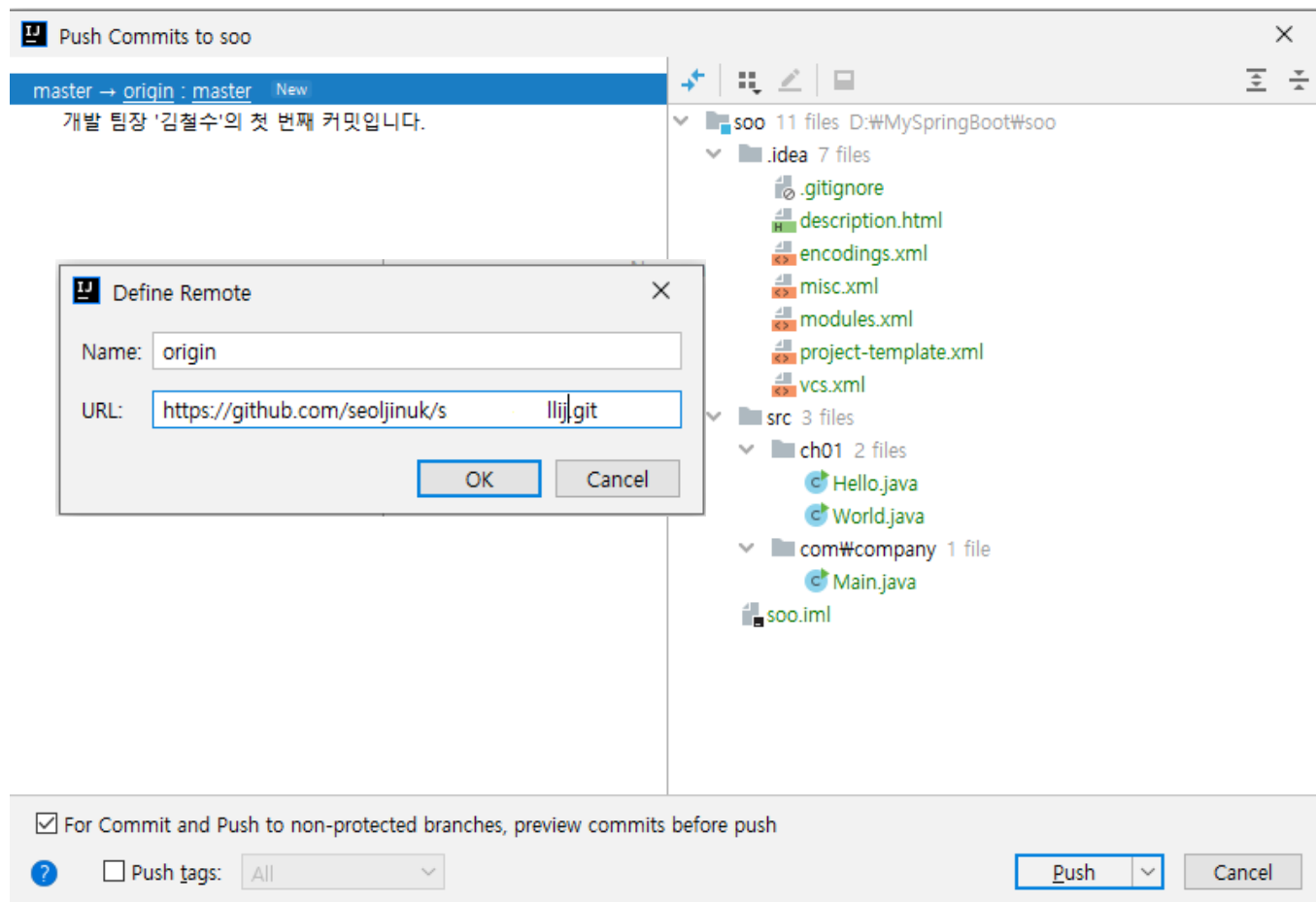


☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

Push Commits to soo라는 새로운 팝업창이 로딩됩니다.

좌측 상단의 [Define reomote] 링크를 클릭하여 원격 저장소의 주소 이름을 입력하고, [Push] 버튼을 클릭합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

원격 저장소에 데이터가 업로드 되었는 지 확인합니다.

master ▾

1 branch

0 tags

Go to file

Add file ▾

<> Code ▾

oraman 개발 팀장 '김철수'의 첫 번째 커밋입니다.

b24e4e0 8 minutes ago ⌚ 1 commit

📁 .idea	개발 팀장 '김철수'의 첫 번째 커밋입니다.	8 minutes ago
📁 src	개발 팀장 '김철수'의 첫 번째 커밋입니다.	8 minutes ago
📄 soo.iml	개발 팀장 '김철수'의 첫 번째 커밋입니다.	8 minutes ago

Help people interested in this repository understand your project by adding a README.

Add a README

☰ 인텔리 제이에서 GitHub 연동하기

☰ 커밋 후 Git Hub에 Push 하기

다음은 하위 패키지 내의 Hello.java 파일의 코드 내용을 확인한 화면입니다.


master ▾ collabo_intellij / src / ch01 / Hello.java / <> Jump to ▾ Go to file ...

 oraman 개발 팀장 '김철수'의 첫 번째 커밋입니다.

Latest commit b24e4e0 11 minutes ago [History](#)

🔍 0 contributors

7 lines (6 sloc) | 128 Bytes

Raw Blame    

```
1 package ch01;
2
3 public class Hello {
4     public static void main(String[] args) {
5         System.out.println("hello01");
6     }
7 }
```

☰ 인텔리 제이에서 GitHub 연동하기

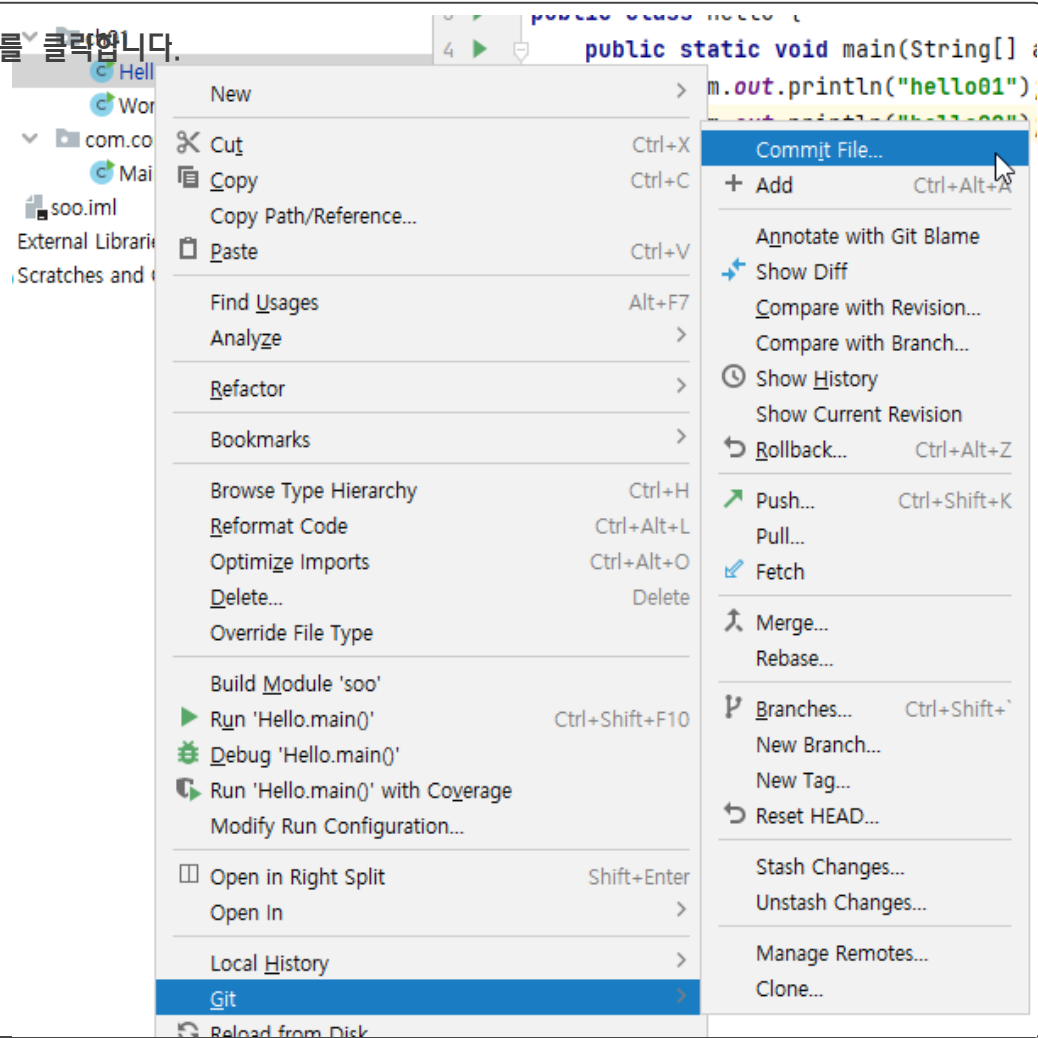
☰ 하나의 파일 수정 후 다시 Push 하기

팀장 '김철수'가 Hello 클래스 파일에 내용을 수정하도록 하겠습니다.

수정이 완료된 다음 해당 파일을 마우스 우측 클릭한 다음 [Git]-[Commit File] 메뉴를 클릭합니다.

```
package ch01;

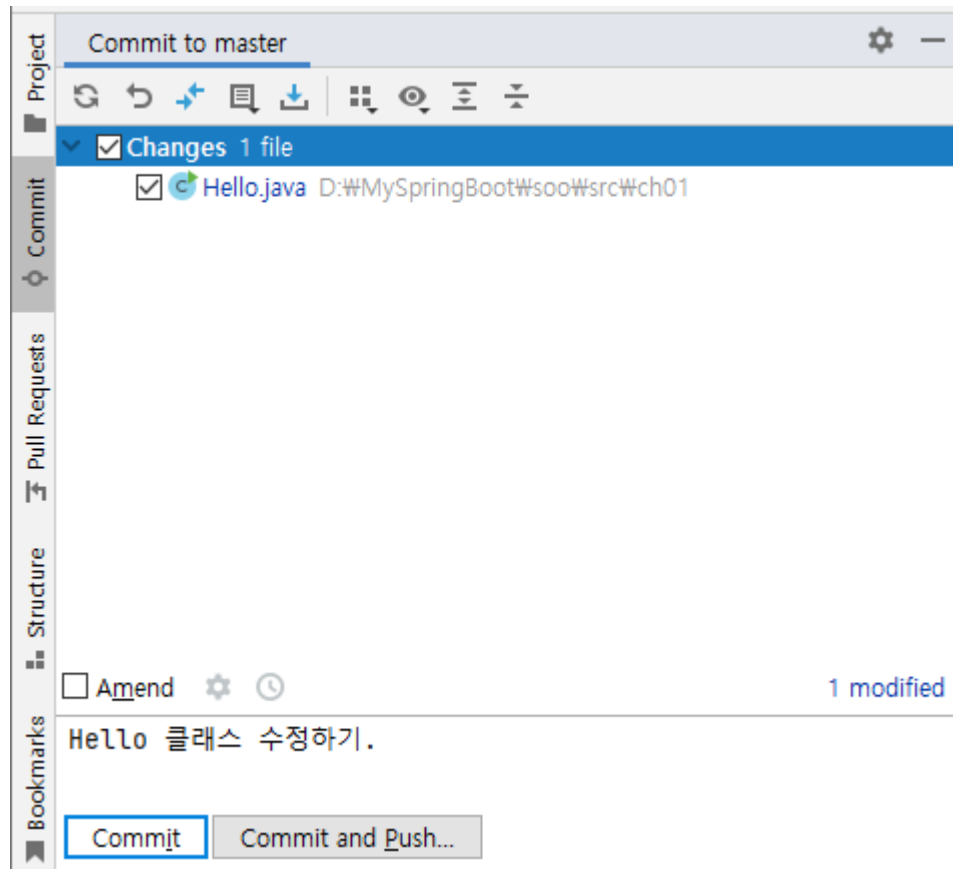
public class Hello {
    public static void main(String[] args) {
        System.out.println("hello01");
        System.out.println("hello02");
    }
}
```



☰ 인텔리 제이에서 GitHub 연동하기

☰ 하나의 파일 수정 후 다시 Push 하기

커밋 메시지 입력란에 'Hello 클래스 수정하기.'라는 문구를 입력하고, [Commit and Push] 버튼을 클릭합니다.




☰ 인텔리 제이에서 GitHub 연동하기

☰ 하나의 파일 수정 후 다시 Push 하기

원격 저장소에 존재하는 Hello.java 파일의 커밋 메시지가 수정되었는 지 확인합니다.

커밋 메시지 'Hello 클래스 수정하기.'를 클릭해 보도록 합니다.

 master ▾ collabo_intellij / src / ch01 /

[Go to file](#) [Add file ▾](#) [...](#)

oraman Hello 클래스 수정하기.		3ac6260 1 minute ago  History
..		
 Hello.java	Hello 클래스 수정하기.	1 minute ago
 World.java	개발 팀장 '김철수'의 첫 번째 커밋입니다.	18 minutes ago

☰ 인텔리 제이에서 GitHub 연동하기

☰ 하나의 파일 수정 후 다시 Push 하기

다음 그림과 같이 해당 파일에 커밋 문구를 클릭하면 변경 내역을 확인할 수 있습니다.

녹색 라인에 '+' 기호가 들어 있는 내용이 이번에 신규로 수정된 내용임을 알려 주고 있습니다.

Hello 클래스 수정하기.

 master

oraman committed 2 minutes ago1 parent b24e4e0commit 3ac626073710a7c312fe7eb2ba47a1708ff8fb95

Browse files

Showing 1 changed file with 1 addition and 0 deletions.

Split

Unified

▼ ↕ 1 ■■■■ src/ch01/Hello.java

↑

@@ -3,5 +3,6 @@

33

public class Hello {

44

public static void main(String[] args) {

55

System.out.println("hello01");

66

System.out.println("hello02");

67

}

78

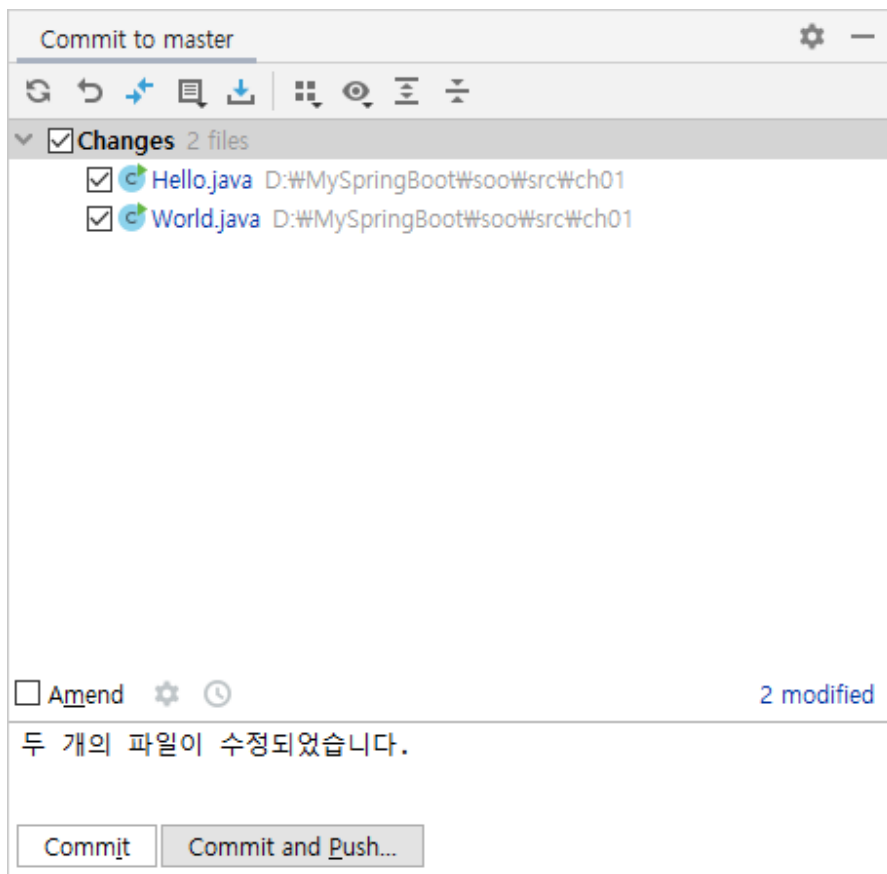
}

☰ 인텔리 제이에서 GitHub 연동하기

☰ 여러 개의 파일을 수정 후 다시 Push 하기

파일 2개(Hello, World) 클래스 파일을 각각 수정하고, 동시에 Push를 수행해 봅니다.

프로젝트를 마우스 우측 클릭한 다음 [Git]-[Commit Directory] 메뉴를 클릭합니다. 커밋 메시지 '두 개의 파일이 수정되었습니다.'를 입력하고, [Commit and Push] 버튼을 클릭합니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ 여러 개의 파일을 수정 후 다시 Push 하기

다음 그림은 Hello.java 파일에 대한 변경 내역을 확인할 수 있습니다.

소스 코드의 내용과 커밋 메시지를 확인해 보시길 바랍니다.

 master ▾ collabo_intellij / src / ch01 / Hello.java / <> Jump to ▾ Go to file ...

 oraman 두 개의 파일이 수정되었습니다. Latest commit 9224a63 31 seconds ago History

 0 contributors

9 lines (8 sloc) | 206 Bytes Raw Blame ✎ ▾ 📄 🗑️

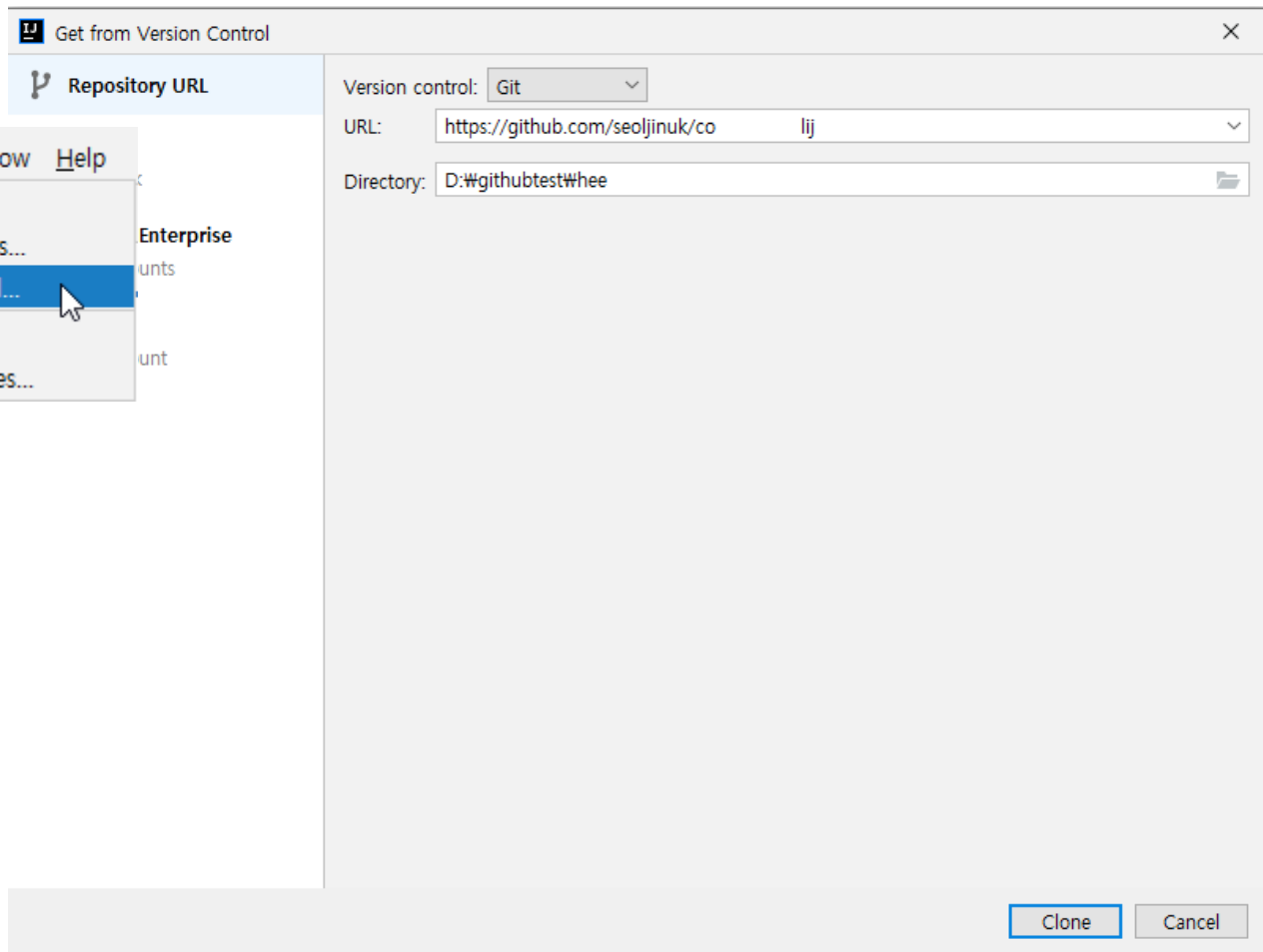
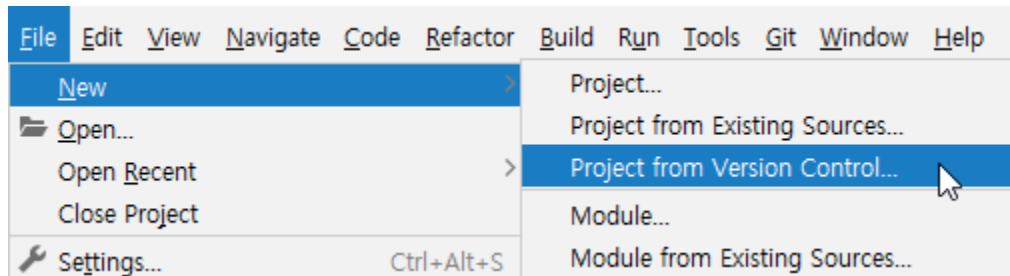
```
1 package ch01;
2
3 public class Hello {
4     public static void main(String[] args) {
5         System.out.println("hello01");
6         System.out.println("hello02");
7         System.out.println("hello03");
8     }
9 }
```

☰ 인텔리 제이에서 GitHub 연동하기

☰ 원격 저장소 복제하기

‘박영희’는 ‘김철수’가 만들어 놓은 프로젝트를 로컬 저장소로 가져 와야 합니다.

원격 저장소의 내용을 그대로 로컬 저장소로 가져 오는 것을 ‘복제(clone)’라고 합니다. IntelliJ 메뉴에서 [Get from VCS] 메뉴를 클릭하고, URL과 Directory 항목을 다음과 같이 작성한 다음, [Clone] 버튼을 클릭합니다.

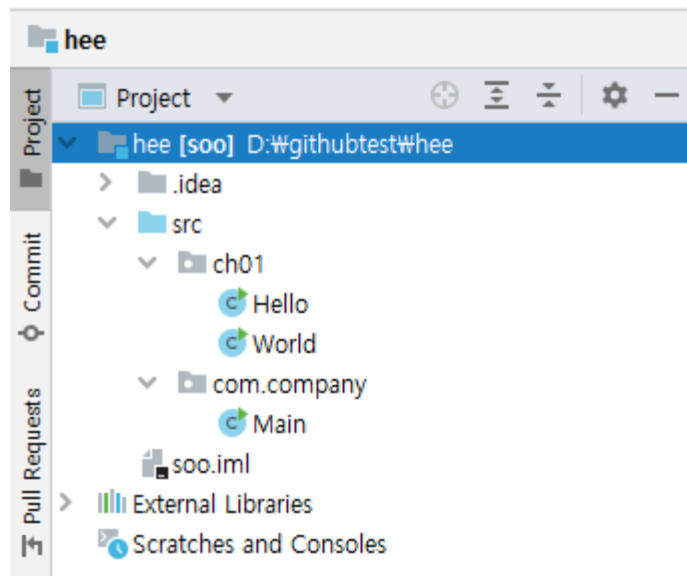


☰ 인텔리 제이에서 GitHub 연동하기

☰ 원격 저장소 복제하기

복제가 완료 되었습니다.

IntelliJ에서 모든 파일이 다운로드가 되었음을 확인합니다. 현재까지의 상태는 '김철수'와 '박영희'의 모든 소스가 동일한 상태입니다.
다음은 복제된 '박영희'의 IntelliJ 화면입니다.

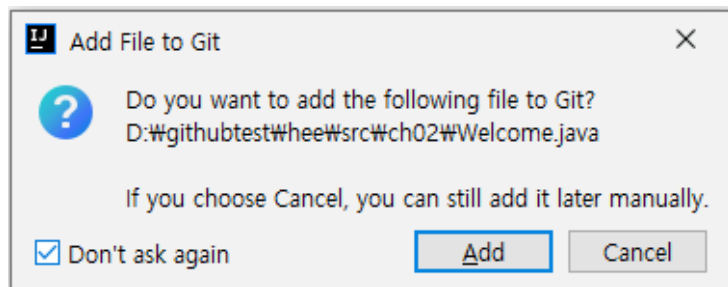


☰ 인텔리 제이에서 GitHub 연동하기

☰ 신규 파일 추가하기

복제된 상태에서 다음과 같이 새로운 파일을 푸시해 보도록 하겠습니다.

‘박영희’ 개발자는 Welcome이라는 클래스를 신규 작성하고, 이를 Push 해보도록 하겠습니다. 메시지는 ‘클론 후에 신규 파일이 추가되었습니다.’라고 입력하도록 합니다. 새로운 파일을 작성하게 되면 그림과 같이 Git에 추가할 지를 확인하는 대화 상자가 로딩됩니다. 더 이상 물어보지 않도록 체크 박스 "Don't ask again"을 체크하고, [Add] 버튼을 클릭합니다.



```
package ch02;



public class Welcome {
    public static void main(String[] args) {
        System.out.println("welcome01");
    }
}
```

☰ 인텔리 제이에서 GitHub 연동하기


☰ 신규 파일 추가하기




수정된 파일을 이용하여 다음과 같이 Push해보도록 하겠습니다.

프로젝트를 마우스 우측 클릭한 다음 [Git]-[Commit File] 메뉴를 클릭합니다. 커밋 메시지 '클론 후에 신규 파일이 추가되었습니다.'를 입력하고, [Commit and Push] 버튼을 클릭합니다.

 master [collabo_intellij / src /](#) 

t Add file ▾ ...

 **oraman** 클론 후에 신규 파일이 추가되었습니다. f9875a2 · now History

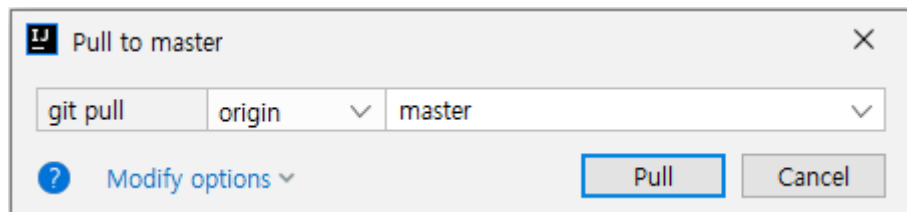
Name	Last commit message	Last commit date
 ..		
 ch01	두 개의 파일이 수정되었습니다.	8 minutes ago
 ch02	클론 후에 신규 파일이 추가되었습니다.	now

인텔리 제이에서 GitHub 연동하기

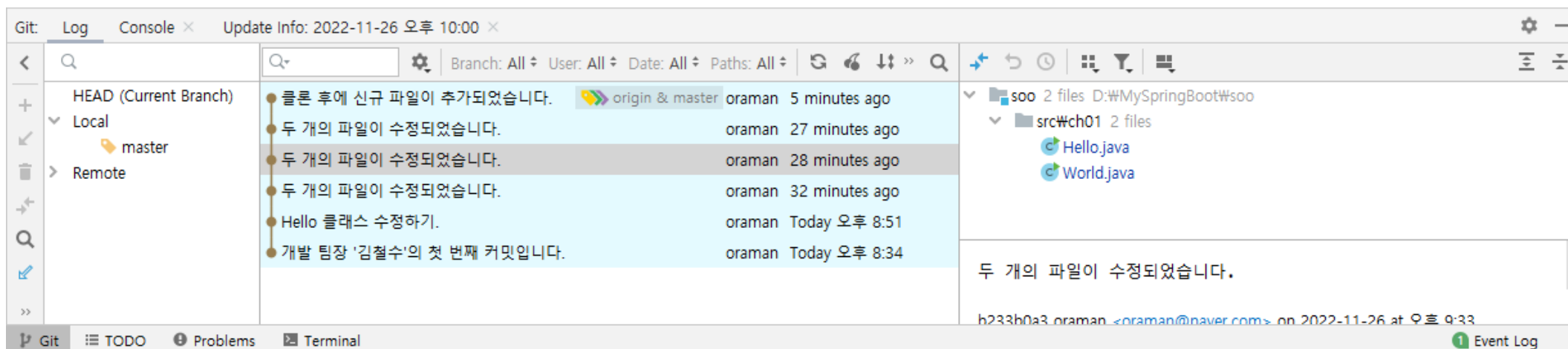
Pull 기능 사용해보기

'김철수' 개발자가 '박영희'가 Push한 파일을 Pull해 보도록 하겠습니다.

'김철수' 개발자는 '박영희'가 조금 전에 Push 했던 Welcome 파일을 보유하고 있지 않습니다. 이 파일을 Pull 기능을 사용하여 데이터를 가져와 보도록 하겠습니다. 프로젝트를 마우스 우측 클릭한 다음 [Git]-[Pull] 메뉴를 클릭합니다.



[Pull] 버튼을 클릭하여, 파일이 다운로드 되었는 지 확인하도록 합니다. 그리고, Git의 Log 항목을 살펴 보면 새로운 커밋 메시지가 보이며, 원격 저장소에서 가져왔음(origin & master)을 알려 주고 있습니다.

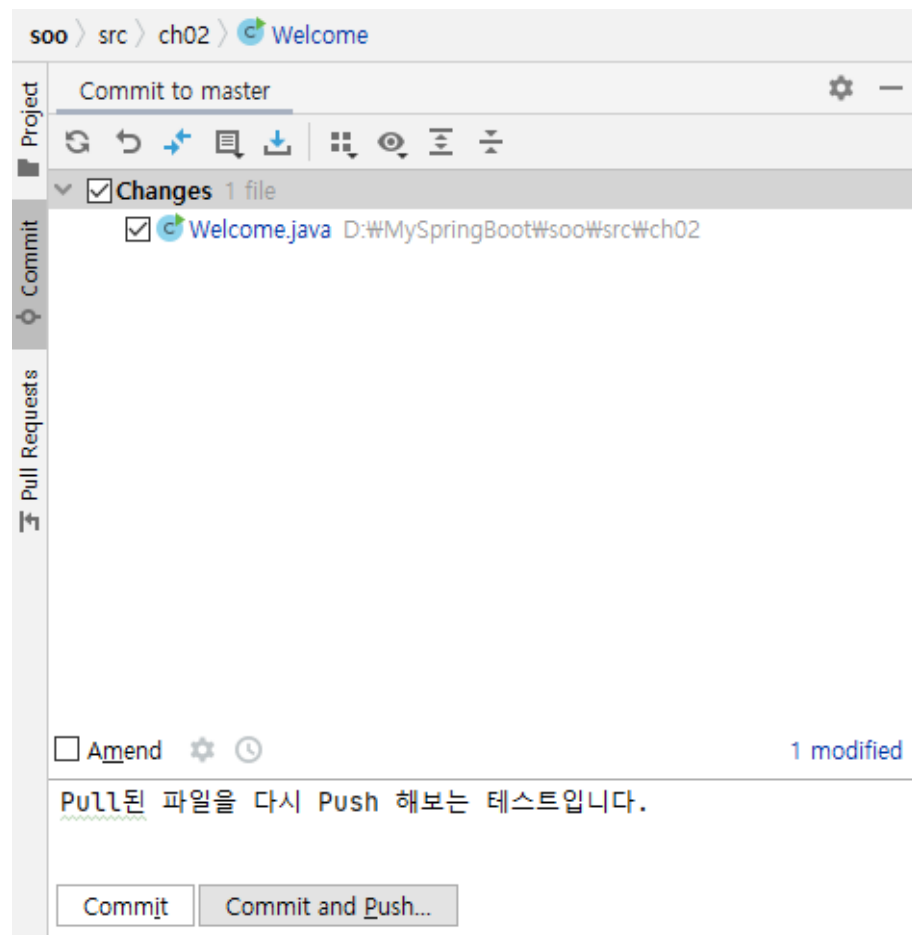


☰ 인텔리 제이에서 GitHub 연동하기

☰ 다시 Push 해보기

‘김철수’ 개발자가 ‘박영희’가 Push한 파일을 Pull해 보도록 하겠습니다.

‘김철수’가 Welcome 클래스를 수정하고, 다시 Push를 해보도록 합니다. 커밋 메시지 ‘Pull된 파일을 다시 Push 해보는 테스트입니다.’라고 입력해 보도록 합니다.




인텔리 제이에서 GitHub 연동하기

다시 Push 해보기

원격 저장소에서 변경된 내역을 다시 확인해 봅니다.

Pull된 파일을 다시 Push 해보는 테스트입니다.

[Browse files](#)

 master

oraman committed 1 minute ago



1 parent 2ffa92b

commit 8c05ba9ed8ae4fc2794acb7e5310054e9e6adf4a

Showing 1 changed file with 1 addition and 1 deletion.

Split

Unified

▼ 2  src/ch02/Welcome.java 

↑
.....

@@ -3,6 +3,6 @@

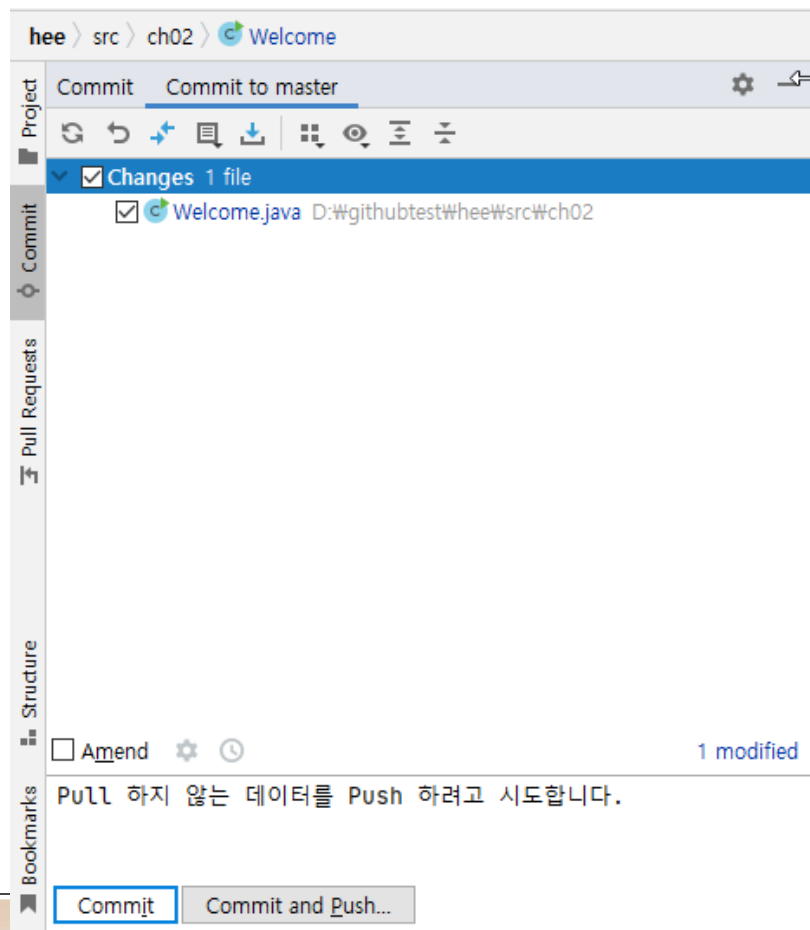
3	3	public class Welcome {
4	4	public static void main(String[] args) {
5	5	System.out.println("welcome01");
6	6	System.out.println("welcome02");
6	7	}
7	-	
8	8	}

인텔리 제이에서 GitHub 연동하기

Welcome 클래스 수정후 Push 해보기

다른 사람이 Push한 데이터를 Pull하지 않고, 현재 개발자가 Push를 시도하는 경우에 문제가 발생합니다.

개발자 '박영희'는 현재 '김철수'가 Push 했던 최신 파일 Welcome 클래스를 Pull 하지 않는 상태입니다. 이러한 내용을 모른다고 가정하고 파일 Welcome을 수정한 다음, 이를 Push 해보도록 하겠습니다. 커밋 메시지 'Pull 하지 않는 데이터를 Push 하려고 시도합니다.' 입니다.

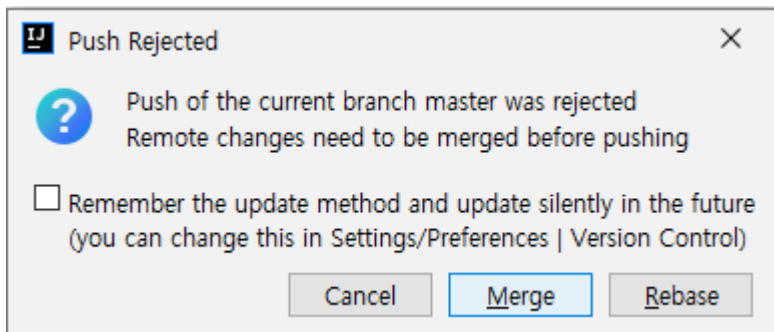


☰ 인텔리 제이에서 GitHub 연동하기

☰ Welcome 클래스 수정후 Push 해보기

다른 사람이 Push한 데이터를 Pull하지 않고, 현재 개발자가 Push를 시도하는 경우에 문제가 발생합니다.

예상되는 결과는 다음과 같이 오류가 발생(Push Rejected)해야 합니다. 데이터를 합쳐야 하므로, [Cancel] 버튼을 클릭하여 취소하도록 합니다. [Merge] 버튼은 병합을, [Rebase] 버튼은 새로운 평가/산정 기준을 설정하는 옵션입니다.

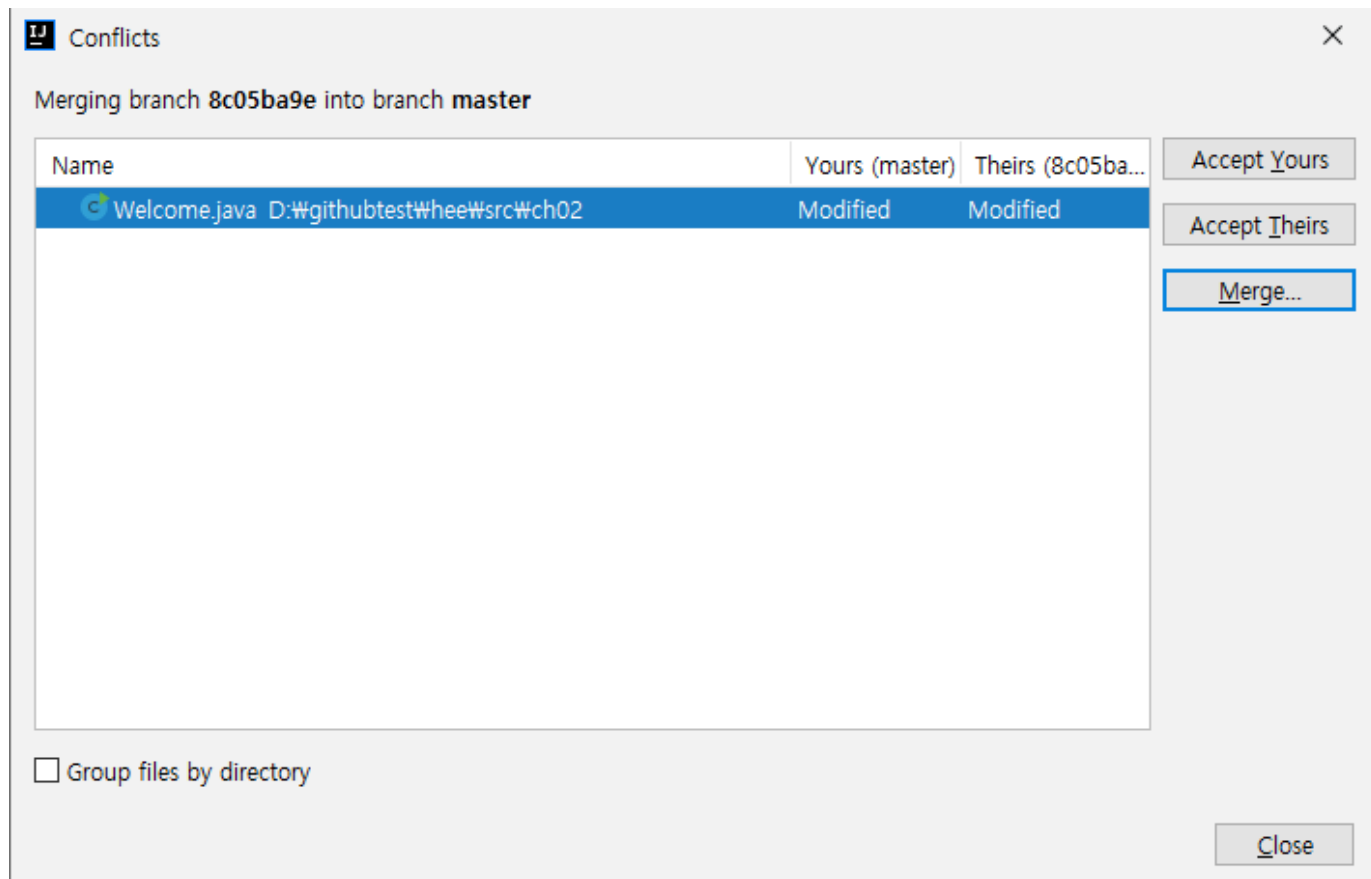


☰ 인텔리 제이에서 GitHub 연동하기

☰ Welcome 클래스 수정후 Push 해보기

다른 사람이 Push한 데이터를 Pull하지 않고, 현재 개발자가 Push를 시도하는 경우에 문제가 발생합니다.

우선 '박영희'는 Welcome 클래스에 대하여 마우스 우측 클릭을 하여 Pull 하도록 합니다. 충돌이 발생하였음을 알려 주는 메시지 창이 로딩됩니다. [Merge] 버튼을 눌러서 데이터를 병합하도록 합니다.

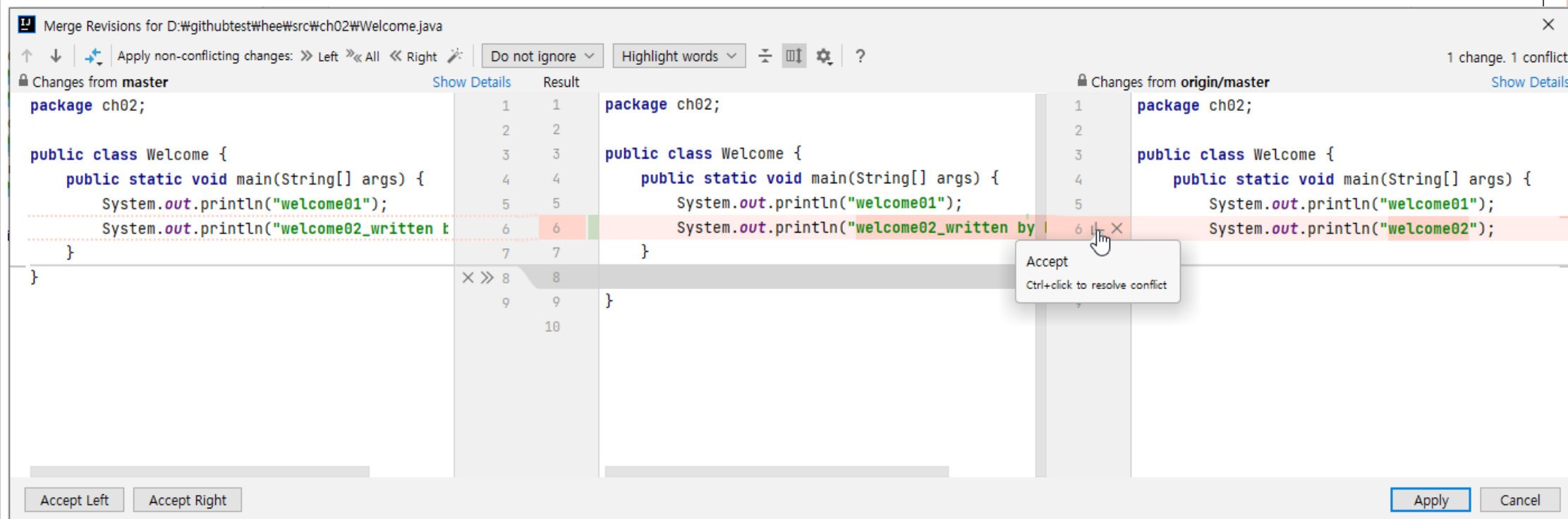


☰ 인텔리 제이에서 GitHub 연동하기

☰ Welcome 클래스 수정후 Push 해보기

다른 사람이 Push한 데이터를 Pull하지 않고, 현재 개발자가 Push를 시도하는 경우에 문제가 발생합니다.

그림과 같이 병합을 수행한 다음, [Apply] 버튼을 눌러 적용시킵니다.



☰ 인텔리 제이에서 GitHub 연동하기

☰ Welcome 클래스 수정후 Push 해보기

다음은 최종적으로 수정된 내역입니다.

```
package ch02;

public class Welcome {
    public static void main(String[] args) {
        System.out.println("welcome01");
        System.out.println("welcome02_written by hee");
        System.out.println("welcome02");
    }
}
```

깃허브(GitHub)

Eclipse와 깃허브 연동

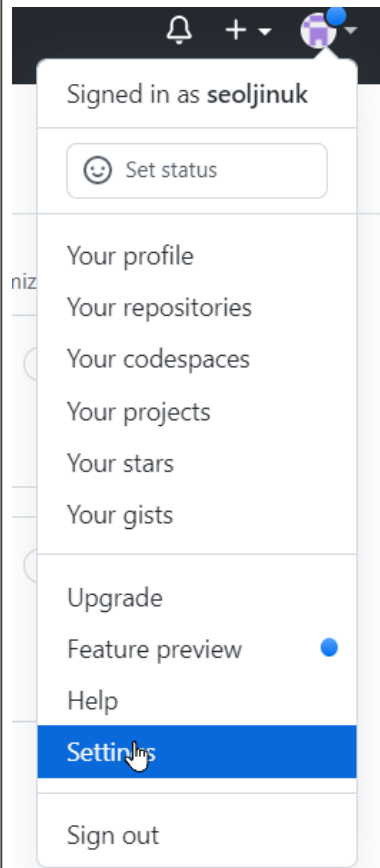


☰ Eclipse에서 GitHub 연동하기

☰ Personal Access Token 발급

이클립스에서 GitHub를 사용하기 위한 첫 번째 절차는 접근 인증키 토큰을 발급받는 것입니다.

GitHub에 로그인한 다음 우측 상단의 버튼을 클릭하고 하위 메뉴에 있는 Settings 항목을 클릭합니다.



☰ Eclipse에서 GitHub 연동하기

☰ Personal Access Token 발급

이동한 페이지의 좌측 맨 하단에 개발자 설정(Developer Settings) 항목을 클릭합니다.



seoljinuk

Your personal account

Public profile

Account

Appearance

Accessibility

Notifications

Access

Billing and plans

Archives

Security log

Sponsorship log

<> Developer settings

☰ Eclipse에서 GitHub 연동하기

☰ Personal Access Token 발급

Personal access tokens 항목에서 [Tokens(classic)] 항목을 클릭합니다.

이 기능은 GitHub API를 이용하여 토큰을 신규로 발급해주는 기능입니다.

[Generate new token] 항목을 누르고, [Generate new token(classic)] 항목을 클릭합니다.

☰ GitHub Apps

👤 OAuth Apps

🔑 Personal access tokens ^

Fine-grained tokens Beta

Tokens (classic)

Personal access tokens (classic)

Generate new token ▼

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Generate new token ▼

- Generate new token Beta
Fine-grained, repo-scoped
- Generate new token (classic)
For general use

☰ Eclipse에서 GitHub 연동하기

☰ Personal Access Token 발급

Note 영역에 적절할 메시지를 작성하고, 하단의 [Generate Token] 버튼을 클릭합니다.

New personal access token (classic)

Select scopes 영역에 원하시는 항목을 선택하고, 화면 맨 하단의 버튼 [Generate token] 버튼을 클릭합니다.

over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

access token for eclipse

What's this token for?

Expiration *

30 days

The token will expire on Tue, Dec 27 2022

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- | | |
|--|--------------------------------------|
| <input type="checkbox"/> repo | Full control of private repositories |
| <input type="checkbox"/> repo:status | Access commit status |
| <input type="checkbox"/> repo_deployment | Access deployment status |
| <input type="checkbox"/> public_repo | Access public repositories |
| <input type="checkbox"/> repo:invite | Access repository invitations |
| <input type="checkbox"/> security_events | Read and write security events |
| <hr/> | |
| <input type="checkbox"/> workflow | Update GitHub Action workflows |

Generate token

Cancel

☰ Eclipse에서 GitHub 연동하기

☰ Personal Access Token 발급

하단의 그림과 같이 Access token이 발급 되었습니다.

이 토큰 정보는 차후 이클립스에서 인증(Authentication)을 위한 계정 정보에 사용될 예정입니다.

GitHub Apps

OAuth Apps

Personal access tokens ^{Beta}

Fine-grained tokens

Tokens (classic)

Personal access tokens (classic)

Generate new token ▼ Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_pTTHM9Xx8hACko3ZGvCcjnoD0nyk0B3IUyYk [Copy](#) Delete

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

☰ Eclipse에서 GitHub 연동하기

☰ 원격 저장소 만들기

GitHub에서 원격 저장소의 이름을 'collabo_eclipse'으로 지정하여 다음과 같이 생성하도록 합니다.

이클립스를 위한 원격 저장소 레포지터리입니다.

Quick setup — if you've done this kind of thing before



Set up in Desktop

or

HTTPS

SSH

https://github.com/seoljinuk/collabo_eclipse.git



Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# collabo_eclipse" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/seoljinuk/collabo_eclipse.git
git push -u origin main
```

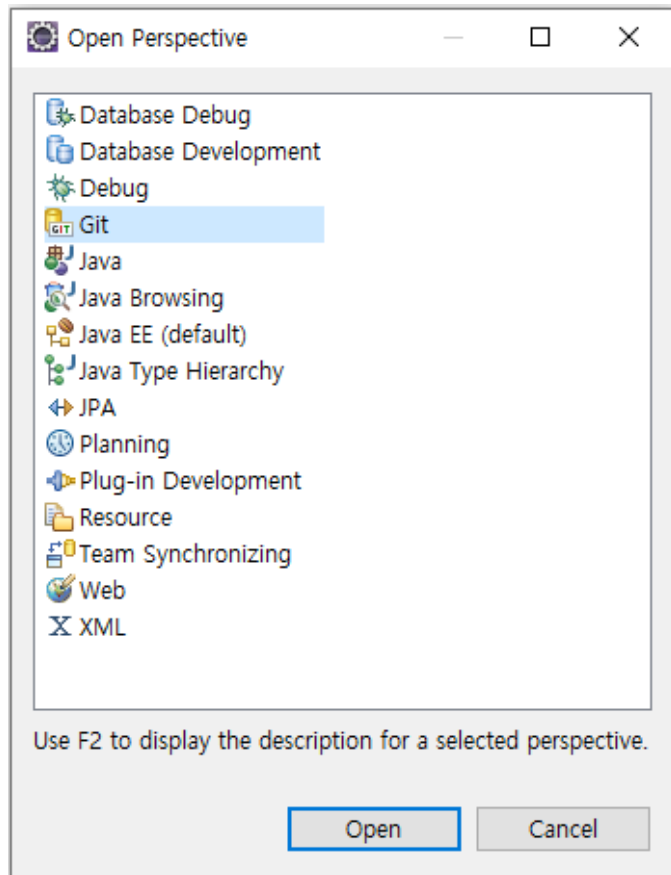


☰ Eclipse에서 GitHub 연동하기

☰ Perspective 변경

이클립스에서 Git을 사용하려면 Perspective를 변경해야 합니다.

이클립스에서 오른쪽 상단의 open Perspective를 클릭한 후 Git을 클릭합니다.



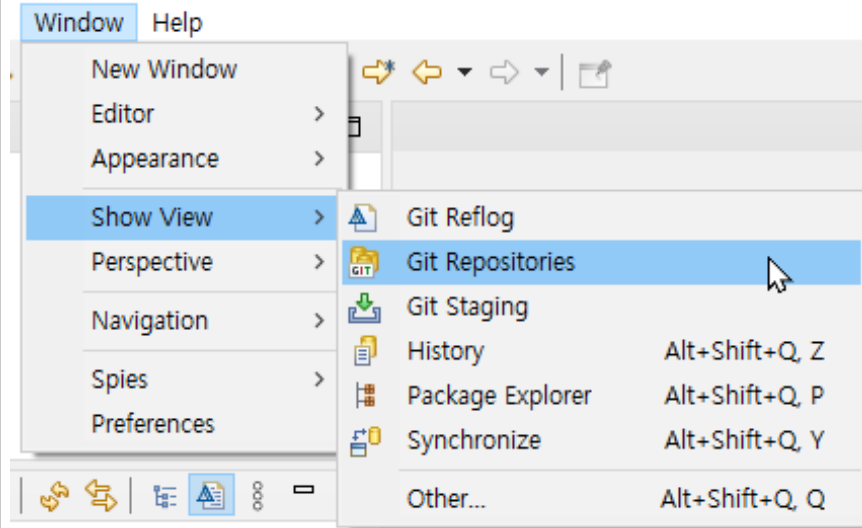
☰ Eclipse에서 GitHub 연동하기

☰ Git 관련 View 보이게 설정하기

이클립스에서 Git과 관련된 View를 on 상태로 설정합니다.

이클립스에서 Git과 관련하여 자주 쓰는 메뉴를 보이도록 설정 하겠습니다.

[Window]-[Show View] 메뉴에서 [Git Repositories]와 [Git Staging] 항목을 그림과 같이 보이도록 지정합니다.

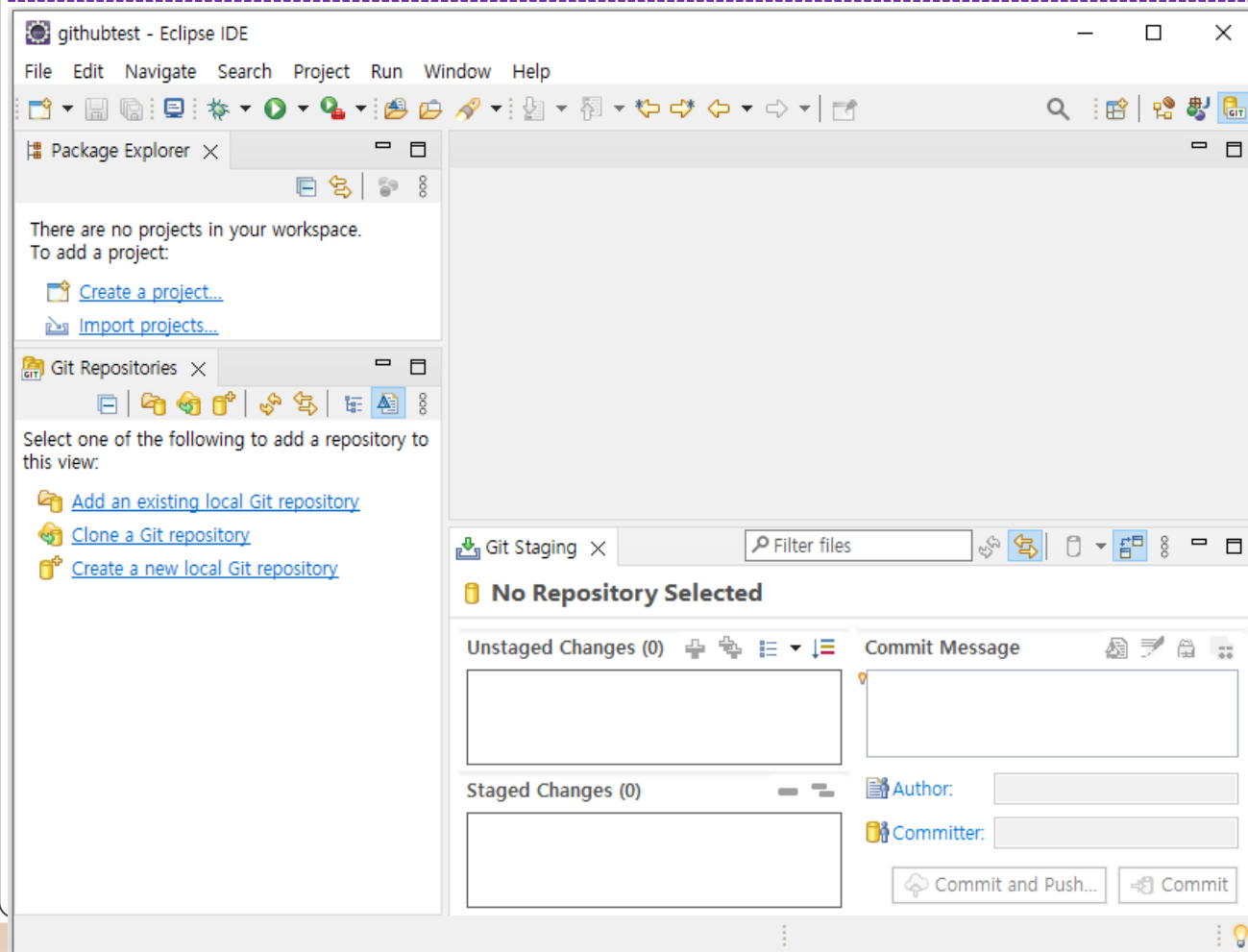


Eclipse에서 GitHub 연동하기

Git 관련 View 보이게 설정하기

이클립스에서 Git과 관련하여 자주 쓰는 메뉴를 보이도록 설정 하겠습니다.

다음 그림에서 왼쪽 하단에 [Git Repositories] 항목이 보이고, [Git Staging] 항목은 우측 하단에 보이고 있습니다.

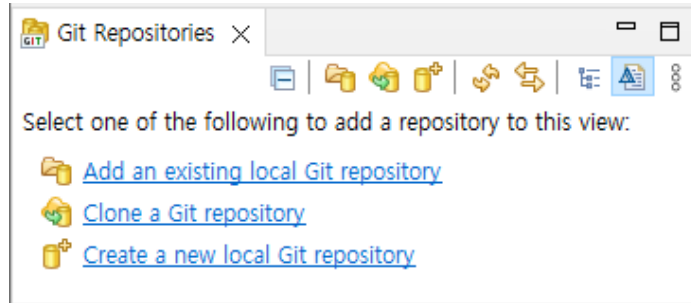


☰ Eclipse에서 GitHub 연동하기

☰ 원격 저장소와 연동하기

원격 저장소와 연동을 하는 방법을 살펴 보겠습니다.

좌측의 Git Repositories 항목에서 [Clone a Git Repository] 메뉴를 클릭하면 Git Repository를 선택하는 팝업 대화 상자가 로딩됩니다.



Eclipse에서 GitHub 연동하기

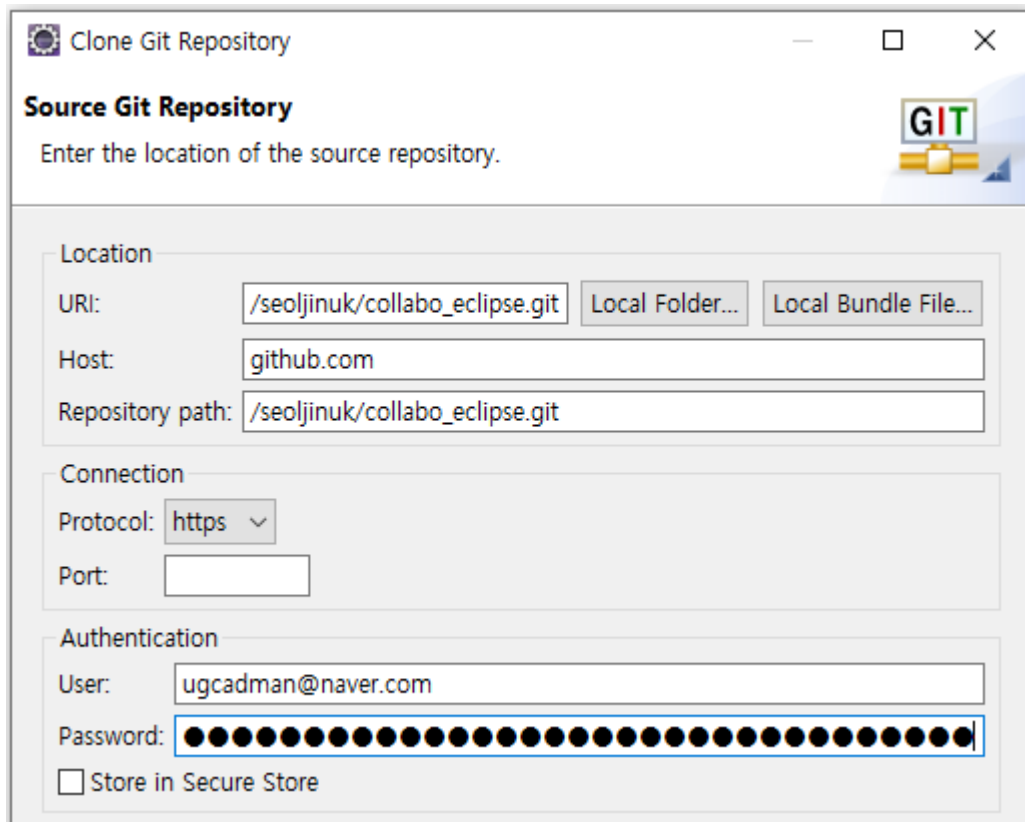
원격 저장소와 연동하기

Git Repository의 위치를 지정하는 화면입니다.

Location 항목에서 URI를 입력합니다. "원격 저장소 만들기" 파트에서 생성하였던 주소를 여기에 입력하면 됩니다.

인증(Authentication) 항목에서 Github의 계정 정보(email, access token key)를 입력하고, Next 버튼을 클릭합니다.

access token key는 이전 "Personal Access Token 발급" 섹션에서 발급을 받은 항목을 기입하면 됩니다.



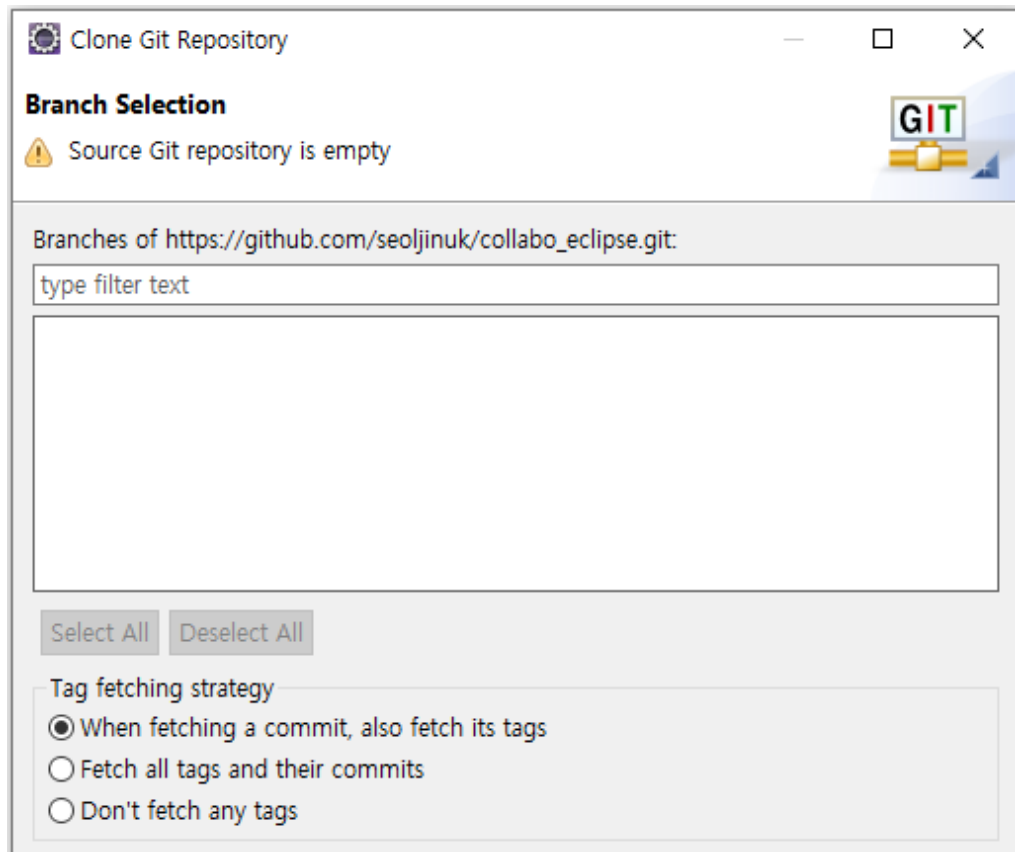
저장소	설명
User	사용자 이메일 입력
Password	access token key 입력

☰ Eclipse에서 GitHub 연동하기

☰ 원격 저장소와 연동하기

브랜치를 선택하는 화면입니다.

GitHub에 데이터를 업로드 한 적이 없으므로, 브랜치가 존재하지 않습니다. 따라서, Branch 선택은 불가능하므로, Next 버튼을 클릭하고 다음으로 이동합니다.

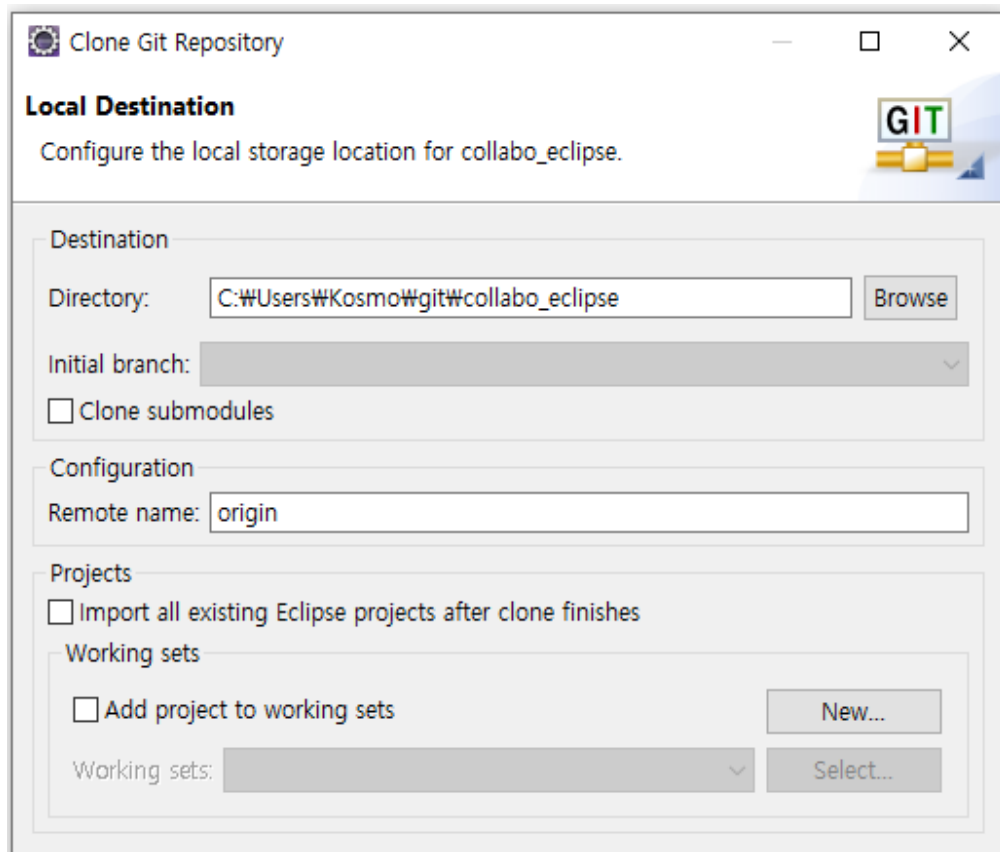


Eclipse에서 GitHub 연동하기

원격 저장소와 연동하기

로컬에 저장될 폴더를 지정하는 부분입니다.

Local Destination 메뉴에서, 작업을 수행할 폴더(Directory)를 선택한 다음, Finish 버튼을 클릭합니다.
Directory는 [Browse] 버튼을 눌러서 사용자가 임의로 지정할 수 있습니다.



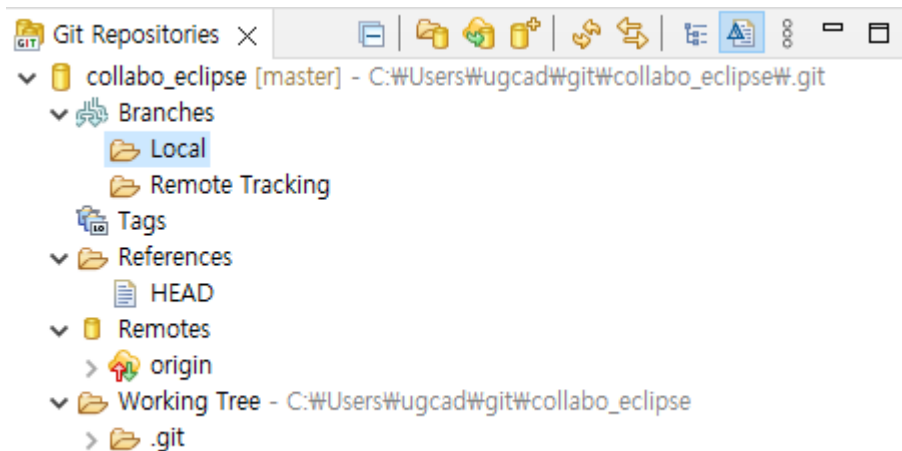
Eclipse에서 GitHub 연동하기

원격 저장소와 연동하기

Git Repositories에 내용을 확인하도록 합니다.

Git Repositories 창에 방금 생성한 내용을 확인할 수 있습니다.

Repository의 이름과 로컬 디렉토리의 git 정보 위치를 확인할 수 있습니다.

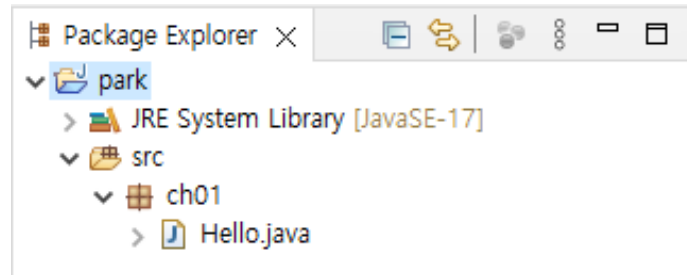


☰ Eclipse에서 GitHub 연동하기

☰ 프로젝트를 저장소에 연동시키기

진행 중인 이클립스 프로젝트와 GitHub 저장소와 연동을 해보도록 하겠습니다.

다음 그림과 같이 프로젝트 및 파일을 생성하도록 합니다.

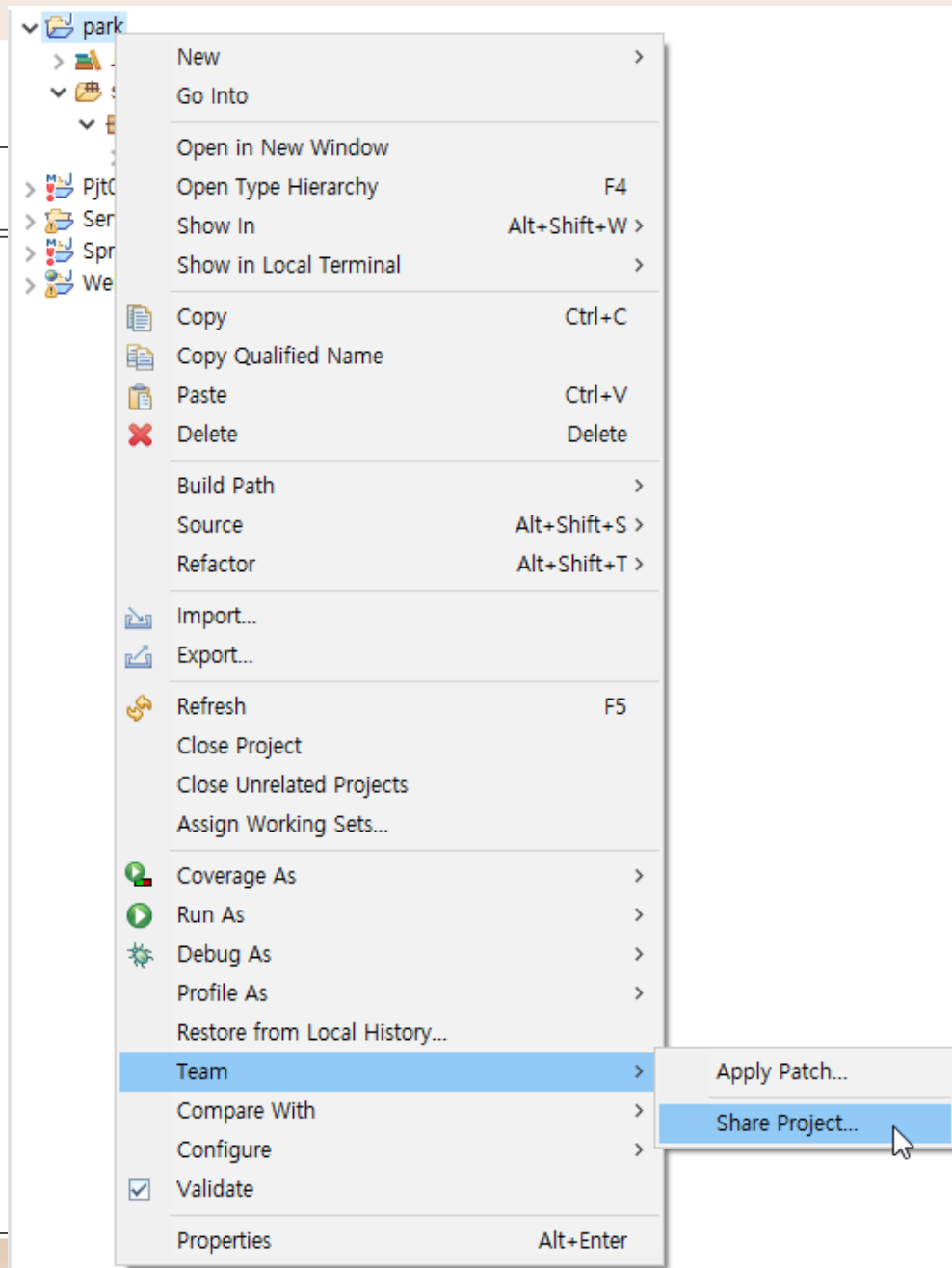


Eclipse에서 GitHub 연동하기

프로젝트를 저장소에 연동시키기

프로젝트를 원격 저장소와 연동하는 방법입니다.

이클립스와 깃허브 저장소를 연동하기 위하여 진행 중인 프로젝트를
마우스 오른쪽 클릭하여 다음과 같이 [Team]-[Share Project] 팝업 메뉴를 선택합니다.

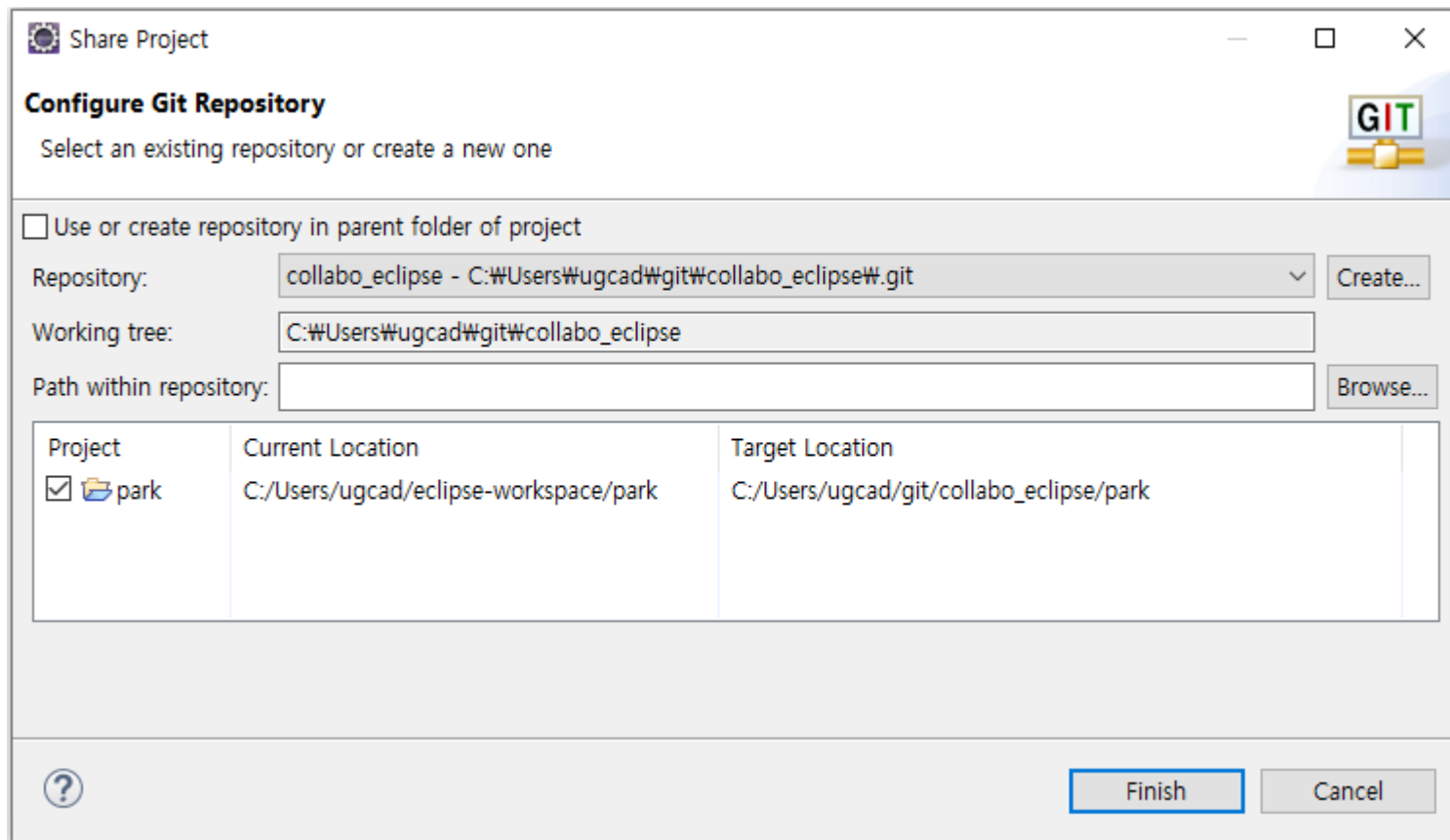


Eclipse에서 GitHub 연동하기

프로젝트를 저장소에 연동시키기

진행 중인 이클립스 프로젝트와 GitHub 저장소와 연동을 해보도록 하겠습니다.

Repository: 항목에 이미 생성해 놓은 로컬 저장소(Repository)를 선택한 후 Finish를 클릭합니다.



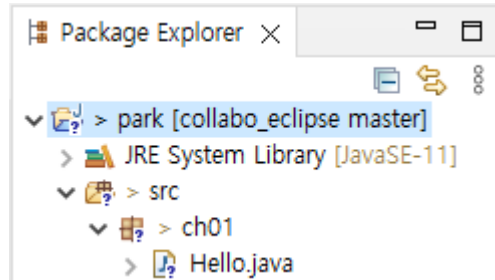
The image shows the 'Share Project' dialog box in Eclipse, specifically the 'Configure Git Repository' tab. The dialog has a title bar 'Share Project' and a close button. Below the title bar, it says 'Configure Git Repository' and 'Select an existing repository or create a new one'. There is a checkbox 'Use or create repository in parent folder of project' which is unchecked. Below this, there are three input fields: 'Repository:' with a dropdown menu showing 'collabo_eclipse - C:\Users\ugcad\git\collabo_eclipse\git', 'Working tree:' with 'C:\Users\ugcad\git\collabo_eclipse', and 'Path within repository:' which is empty. To the right of the 'Repository:' dropdown is a 'Create...' button, and to the right of the 'Path within repository:' field is a 'Browse...' button. Below these fields is a table with three columns: 'Project', 'Current Location', and 'Target Location'. The table has one row with a checked checkbox in the 'Project' column, a folder icon and the name 'park' in the 'Current Location' column, and the path 'C:/Users/ugcad/git/collabo_eclipse/park' in the 'Target Location' column. At the bottom of the dialog, there is a question mark icon on the left, and 'Finish' and 'Cancel' buttons on the right.

Project	Current Location	Target Location
<input checked="" type="checkbox"/> park	C:/Users/ugcad/eclipse-workspace/park	C:/Users/ugcad/git/collabo_eclipse/park

☰ Eclipse에서 GitHub 연동하기

☰ 프로젝트를 저장소에 연동시키기

Package Explorer의 프로젝트를 보면 [저장소이름 master]가 생기는 것을 확인할 수 있습니다.

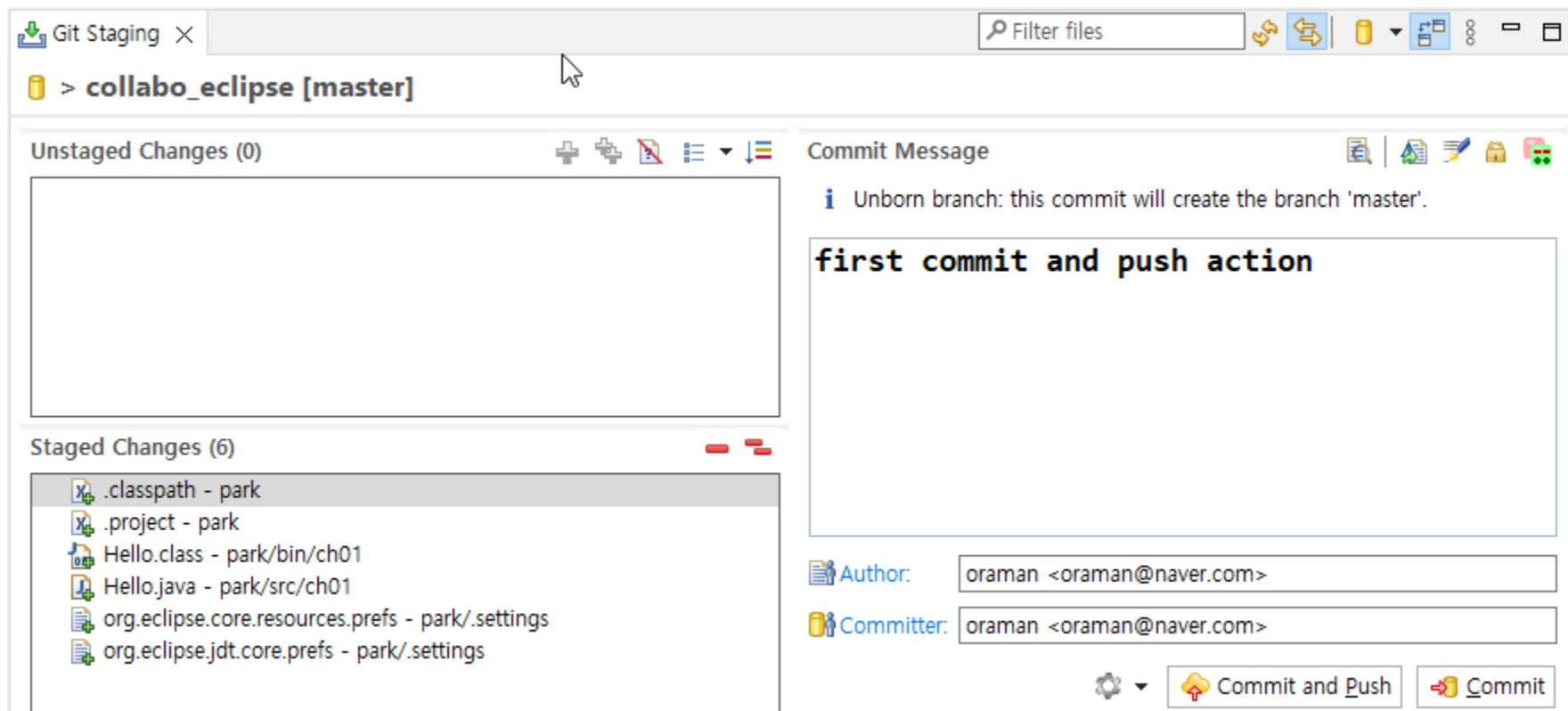


Eclipse에서 GitHub 연동하기

커밋해보기

코딩한 내용을 원격 저장소에 저장해 보도록 하겠습니다.

단축키 Ctrl + Shift + #을 이용하여 Git Staging 창을 엽니다. 최초이므로 모든 파일을, Staged Changes 영역으로 이동하고, Commit Message를 적절히 입력합니다. 우측 하단의 Commit and Push 버튼을 클릭합니다. 파일이 보이지 않으면 해당 프로젝트를 클릭하면 확인할 수 있습니다.



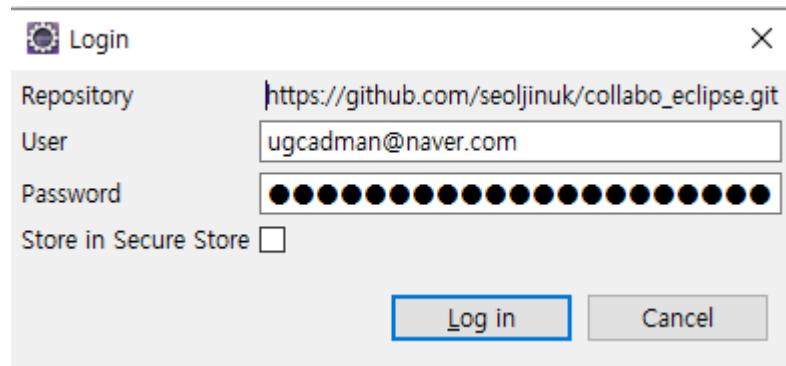
Eclipse에서 GitHub 연동하기

커밋해보기

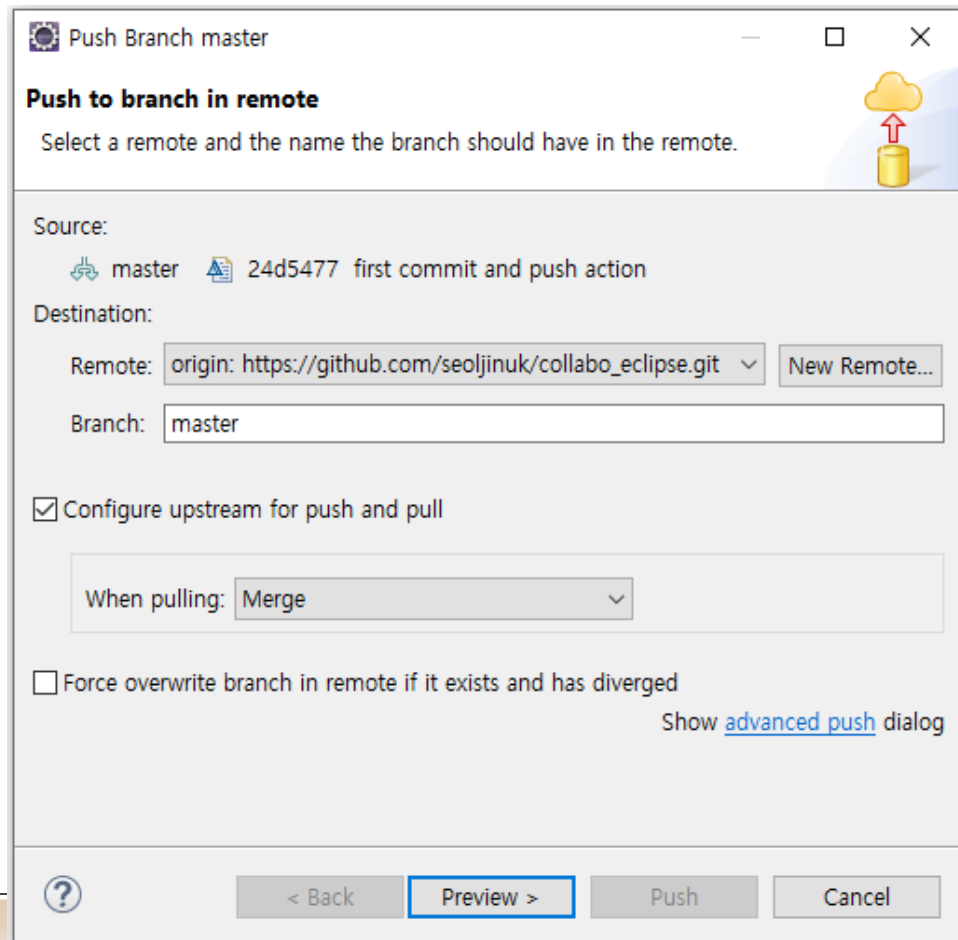
기본 브랜치인 master에 원격으로 push를 진행합니다.

로그인 화면에서 사용자 이메일과 토큰 정보를 입력하고, [Log in] 버튼을 클릭하여 다음으로 진행합니다.

기본 브랜치인 master에 원격으로 push를 진행합니다. [Next] 버튼을 클릭하여 다음으로 진행합니다.



The 'Login' dialog box in Eclipse. It contains fields for 'Repository' (https://github.com/seoljinuk/collabo_eclipse.git), 'User' (ugcadman@naver.com), and 'Password' (masked with dots). There is a 'Store in Secure Store' checkbox and 'Log in' and 'Cancel' buttons.



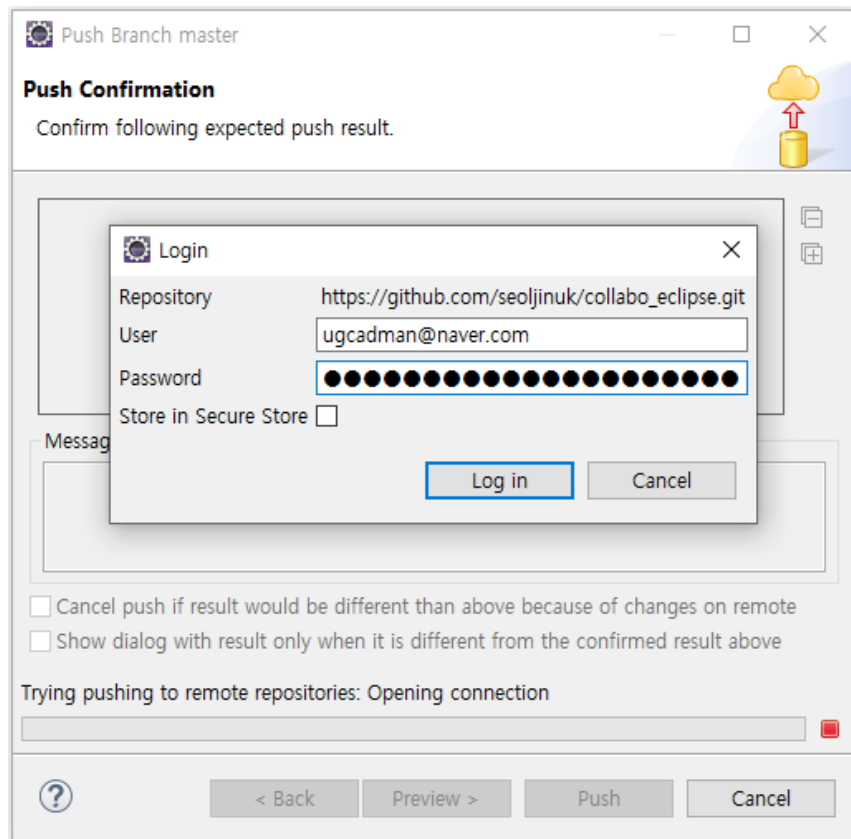
The 'Push Branch master' dialog box in Eclipse. It has a title bar with a gear icon and the text 'Push Branch master'. The main content area is titled 'Push to branch in remote' and includes the instruction 'Select a remote and the name the branch should have in the remote.' Below this, there is a 'Source' section showing 'master' and '24d5477 first commit and push action'. The 'Destination' section has a 'Remote' dropdown set to 'origin: https://github.com/seoljinuk/collabo_eclipse.git' and a 'Branch' dropdown set to 'master'. There is a 'New Remote...' button. A checkbox 'Configure upstream for push and pull' is checked, and a 'When pulling:' dropdown is set to 'Merge'. At the bottom, there is a checkbox 'Force overwrite branch in remote if it exists and has diverged' and a link 'Show advanced push dialog'. The bottom bar contains a help icon, '< Back', 'Preview >', 'Push', and 'Cancel' buttons.

Eclipse에서 GitHub 연동하기

커밋해보기

다음 그림은 Push 정보에 대하여 확인 절차를 밟는 과정입니다.

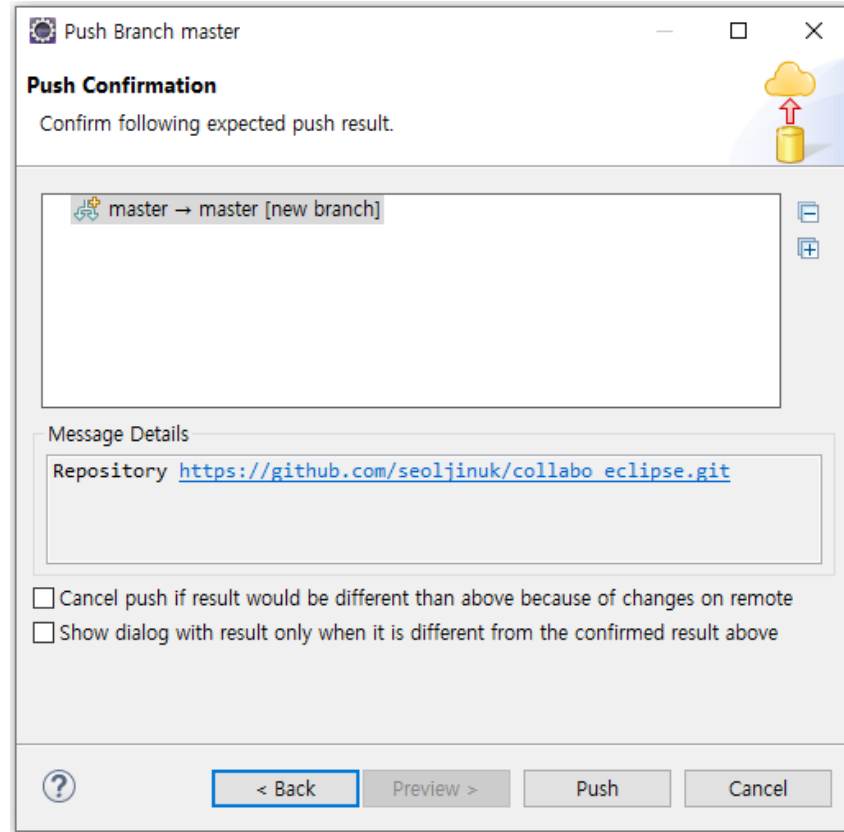
로그인 창에 Github의 계정 정보(email, access token key)를 입력하고, [Log in] 버튼을 클릭합니다.



☰ Eclipse에서 GitHub 연동하기

☰ 커밋해보기

Push 정보에 대한 확인(Confirmation) 작업을 하고, [Push] 버튼을 클릭합니다.

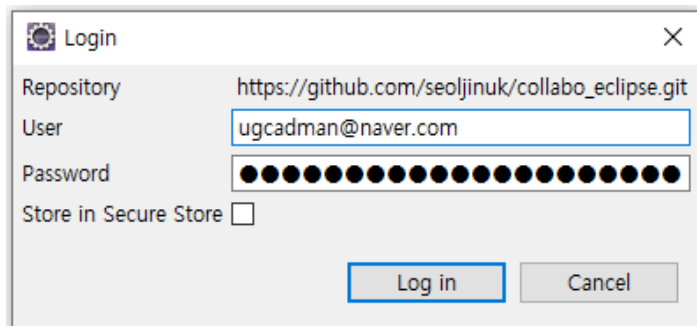


☰ Eclipse에서 GitHub 연동하기

☰ 커밋해보기

로그인 창에 계정 정보를 재입력하고, 최종 확인을 수행합니다.

다시 로그인 창이 로딩됩니다. 최초 시행시에만 2번 로그인해야 하고 이후부터는 한번만 로그인 하면 됩니다.
Github의 계정 정보(email, access token key)를 입력하고, [Log in] 버튼을 클릭합니다.



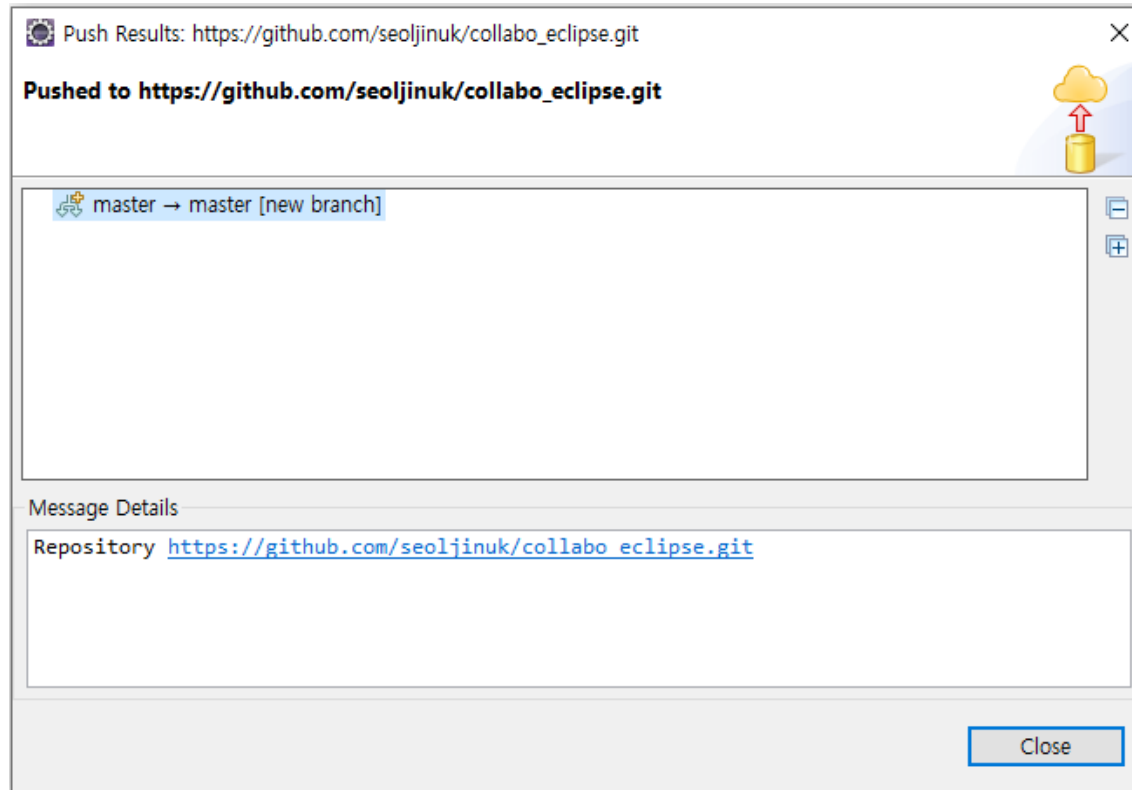
The screenshot shows a 'Login' dialog box with the following fields and controls:

- Repository:** A text field containing the URL `https://github.com/seoljinuk/collabo_eclipse.git`.
- User:** A text field containing the email address `ugcadman@naver.com`.
- Password:** A text field where the password is masked with 20 black dots.
- Store in Secure Store:** A checkbox that is currently unchecked.
- Buttons:** Two buttons at the bottom right, 'Log in' and 'Cancel', with 'Log in' being the primary action.

☰ Eclipse에서 GitHub 연동하기

☰ 커밋해보기

항목들을 확인하고, [Close] 버튼을 눌러서 마무리합니다.




☰ Eclipse에서 GitHub 연동하기

☰ 최종 확인하기

GitHub 홈페이지에서 Push된 프로젝트 소스 코드들을 확인해 봅니다.

프로젝트로 소스 코드가 업로드 되어 있고, 커밋 메시지도 보입니다.

이클립스에서 소스 코드를 수정하고 동일한 방식으로 Commit을 진행하면 해당 프로젝트를 손쉽게 관리할 수 있습니다.

 master ▾ collabo_eclipse / park /

Go to file

Add file ▾

...

oraman first commit and push action24d5477 11 minutes ago🕒 History

..

📁 .settings	first commit and push action	11 minutes ago
📁 src/ch01	first commit and push action	11 minutes ago
📄 .classpath	first commit and push action	11 minutes ago
📄 .gitignore	first commit and push action	11 minutes ago
📄 .project	first commit and push action	11 minutes ago