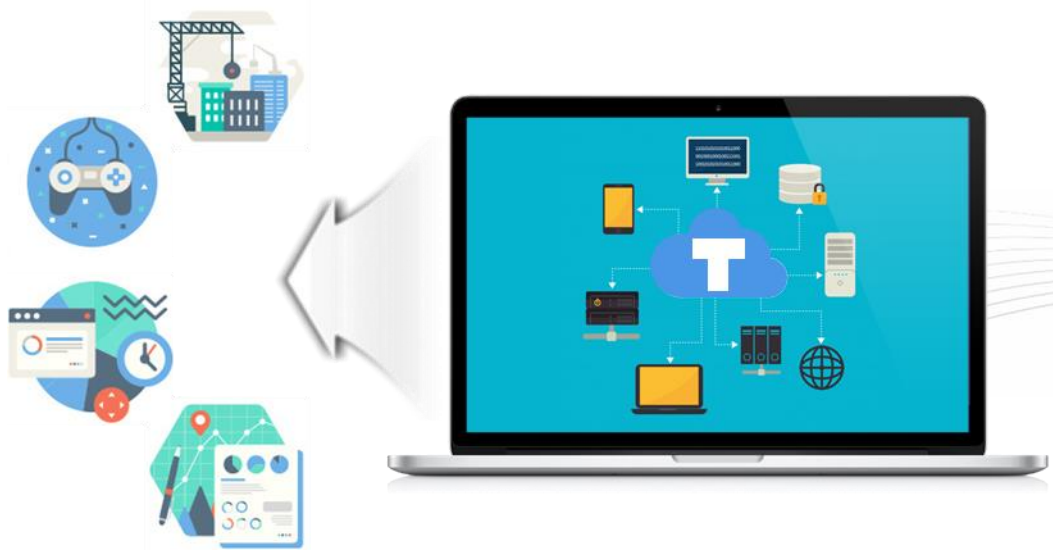
AIoT 데이터 시각화 및 대시보드 개발



Chapter 00.
Github

Ⅰ Git

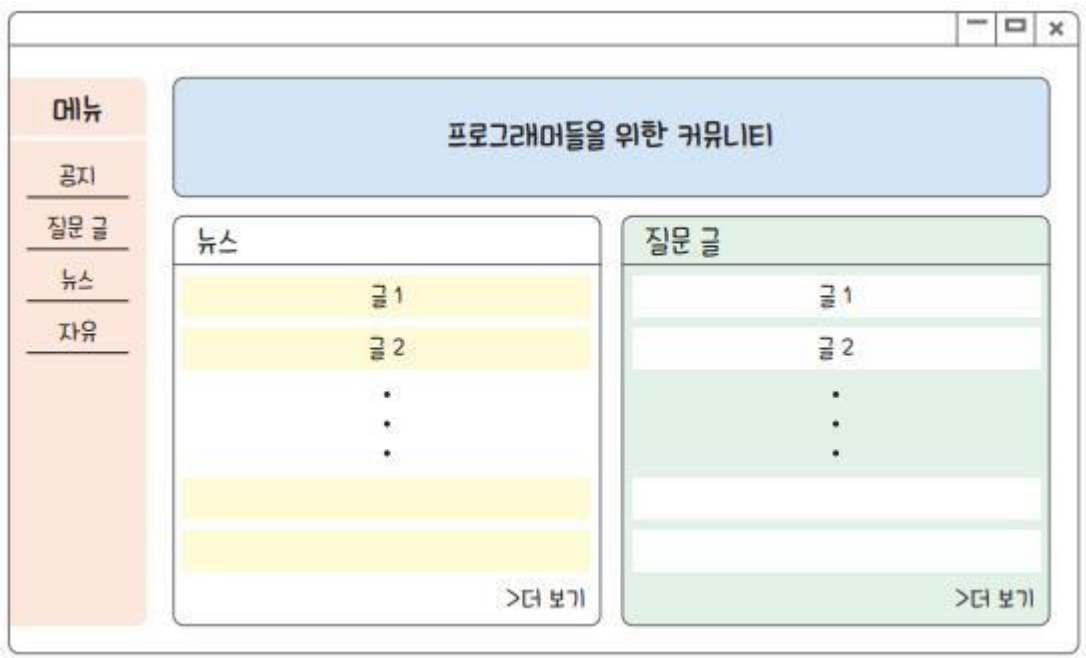
Ⅰ Github 계정 생성

Ⅱ Pycharm과 Git 연동

Ⅲ Pycharm에서 Git 사용방법

■ 깃이 없는 세상

- 깃은 버전을 관리하기 위한 도구
- 버전 관리란 무엇이며, 왜 해야 하는 걸까?
- 깃 없이 다음과 같은 웹 사이트를 만든다고 가정해 보자



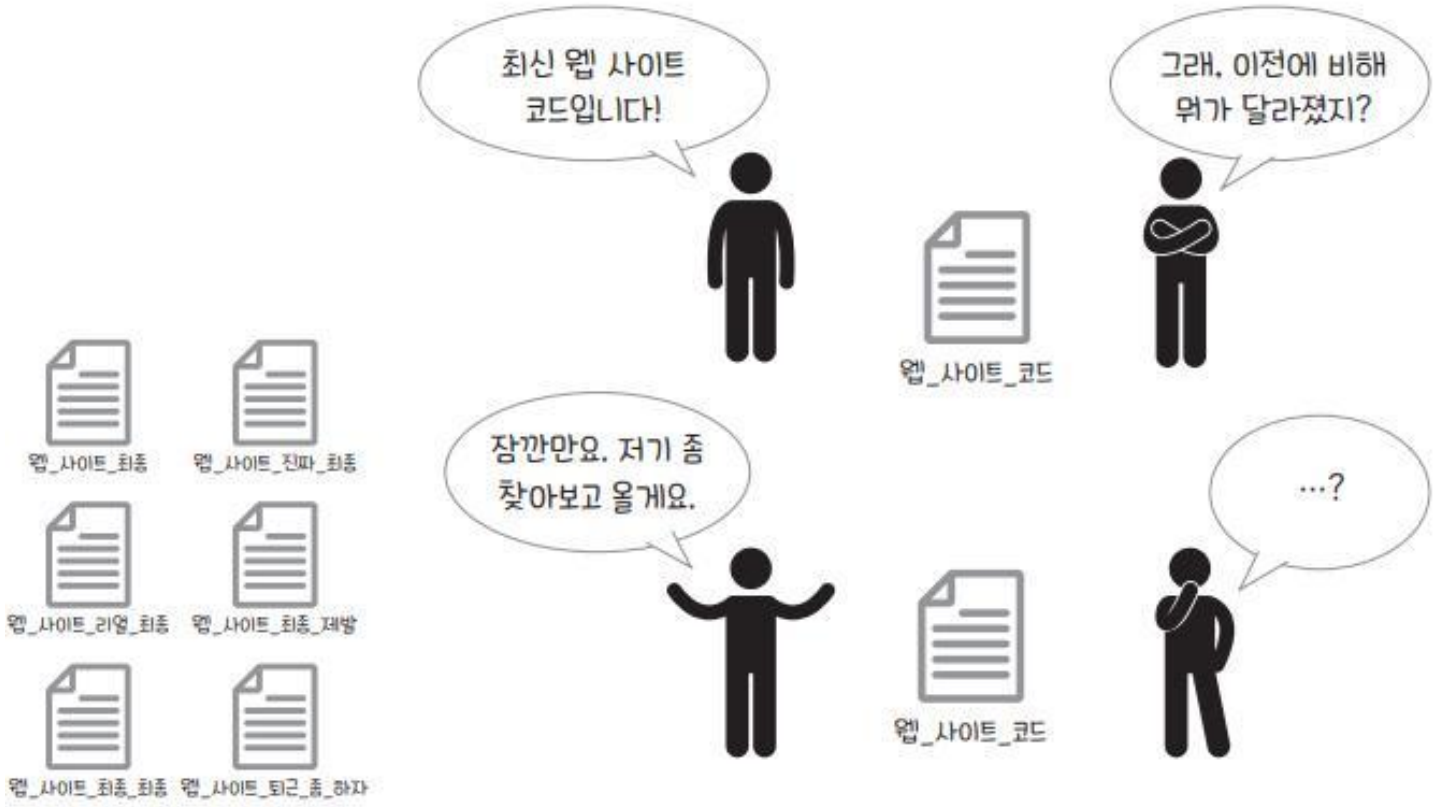
만들 웹 사이트

■ 변경 내역을 확인하기 어렵다

- 대개 파일을 단순히 저장하면 이전에 저장된 내용에서 현재 내용으로 덮어씀
- 즉, 저장된 파일은 항상 최신 상태만 갖게 됨
- 이런 방식으로는 현재 저장된 내용이 이전에 비해 무엇이 어떻게 달라졌는지 알기 어려움
- 다시 말해, 변경 내역을 추적하기가 어려움
- 매번 다른 이름으로 따로 파일을 저장하여 관리하는 방법도 있지만, 이는 프로그램 개발 과정에서만큼은 권장할 만한 방법이 아님
- 매번 파일을 다른 이름으로 새롭게 저장하여 변경 내역을 관리하는 것은 저장 공간을 낭비하는 일일 뿐 아니라 쉽게 실수할 수도 있기 때문임

■ 단순 저장 방식의 문제점

■ 다른 이름으로 파일을 저장하는 방식의 문제점



■ 버전을 되돌리기 어렵다

- 파일을 단순히 덮어쓰거나 다른 이름으로 저장하는 방식으로는 과거 특정 시점으로 파일을 되돌리기도 쉽지 않음
- 가령 우리가 개발할 웹 사이트에서 다음과 같이 옆 메뉴의 디자인을 변경했다고 하자

(수정 전)만들 웹 사이트

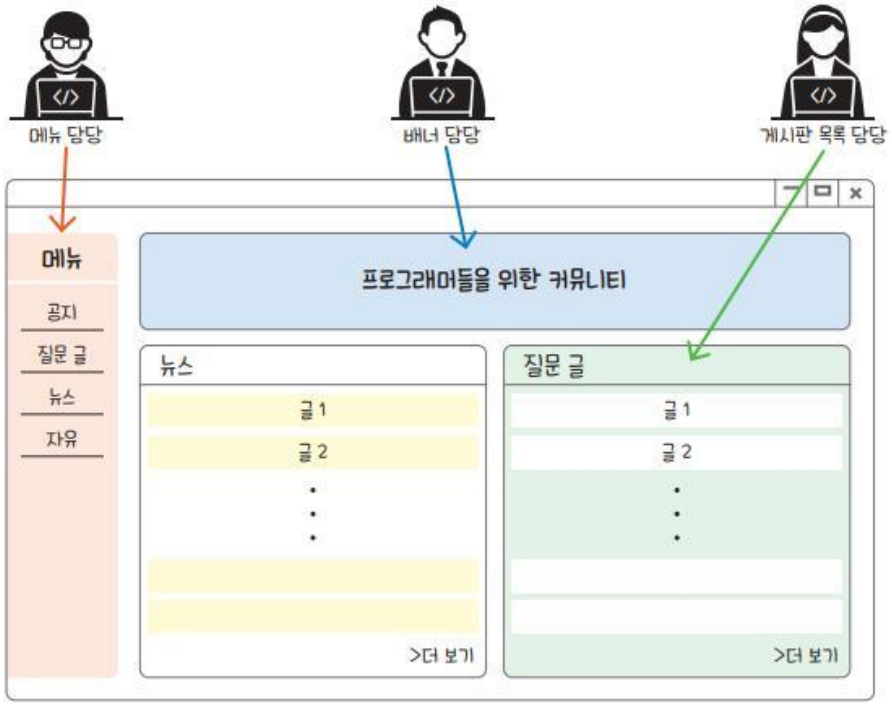
(수정 후)만들 웹 사이트

■ 버전을 되돌리기 어렵다

- 디자인을 변경하는 과정에서 소스 코드 파일 곳곳을 수정하거나 삭제했을 것
- 이때 수정한 코드에 문제가 생겼거나 사용자 반응이 영 좋지 않은 등 여러 문제로
  이전의 모습으로 되돌려야 하는 상황이 생길 수 있음
- 만일 파일을 단순히 덮어썼거나 다른 이름으로 저장하는 방식으로 변경 내역을 관리했다면
  파일의 어느 부분이 삭제됐고, 어느 부분을 어떻게 되돌려야 할지 파악하기 어려울 것
- 다시 말해, 이전으로 되돌리기가 어려워짐

■ 협력하기 어렵다

- 대규모 소프트웨어는 대부분 여러 개발자가 협업하여 개발
- 예를 들면 누군가는 메뉴를 만들고 누군가는 결제 기능을 만들고 누군가는 로그인을 만드는 식으로 각자 개발할 업무를 맡고, 추후 각자 만든 내용들을 합치는 것



개발 협업

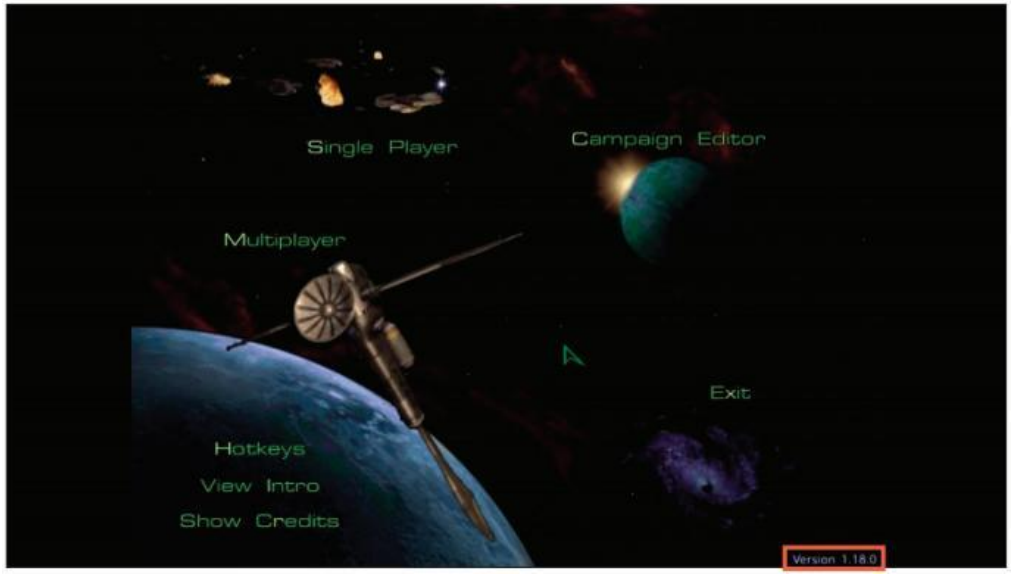
■ 협력하기 어렵다

- 만일 모두가 작업한 파일을 덮어쓰는 방식으로 저장했거나 다른 이름으로 파일을 저장하는 방식으로 파일을 관리했다면
  서로의 작업 내역을 합칠 때 매우 어려워짐
- 웹 사이트를 이루는 파일이 여러 개이고 코드 양이 방대하다면
  누가 어떤 파일에서 어떻게 코드를 수정했는지 파악하기 힘들기 때문임
- 코드를 합치는 과정에서 서로가 작업한 내용을 일일이 비교해야 한다면 시간이 많이 걸릴뿐더러 실수도 매우 빈번하게 발생할 것

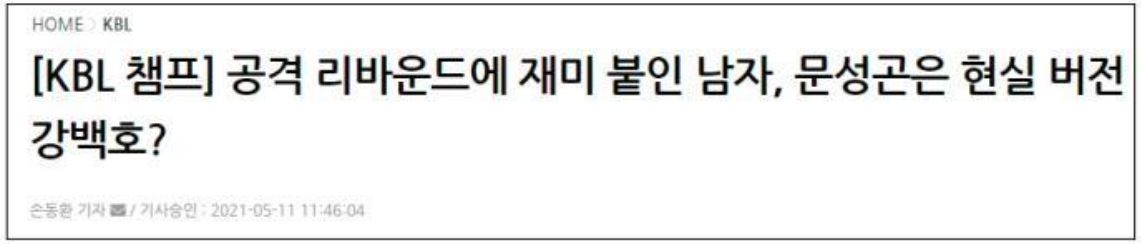
[각자 개발한 소스 코드를 합치는 어려움]

■ 버전과 버전 관리 이해하기

- ‘버전’이라는 말은 그림 1-8의 우측 하단처럼 게임이나 앱부터 그림 1-9와 같은 관용적 표현에 이르기까지 우리 생활 주변에서 쉽게 접할 수 있는 말
- 경험상 알 수 있듯이, 새로운 버전은 주로 기능이 새로 추가되거나 크고 작은 버그가 수정되는 등 기존과는 다른 유의미한 변화가 생겼을 때 만들어짐
- 즉, 버전이란 ‘유의미한 변화가 결과물로 나온 것, 유의미한 변화가 결과물로 저장된 것’을 의미



[그림 1-8 | 게임에서 사용하는 ‘버전’ 용어]


HOME > KBL

[KBL 챔프] 공격 리바운드에 재미 붙인 남자, 문성곤은 현실 버전 강백호?

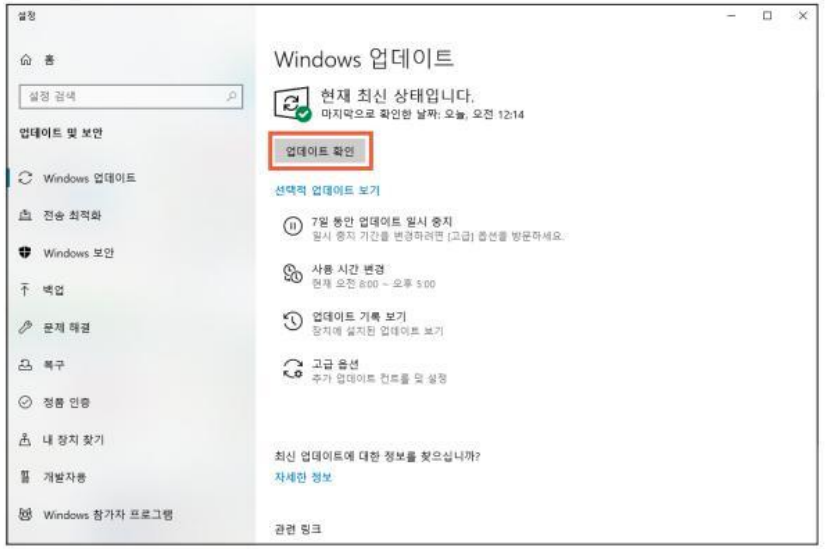
손동환 기자 / 기사승인 : 2021-05-11 11:46:04

[그림 1-9 | 관용적으로 사용하는 ‘버전’ 용어]

■ 버전과 버전 관리 이해하기

버전 vs 패치 vs 업데이트

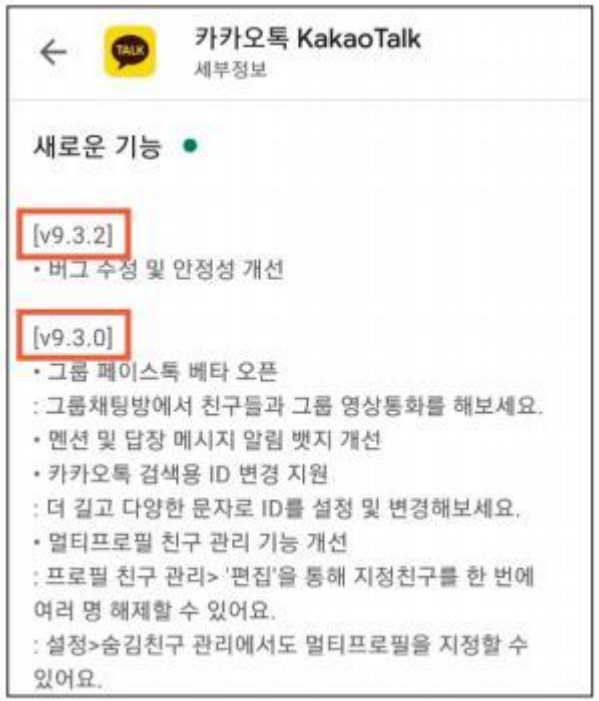
- 패치와 업데이트라는 말도 버전과 비슷한 의미로 자주 쓰임
- 패치는 시급한 오류 해결을 동반하거나 비교적 규모가 작은 버전이라는 의미가 강함
- 업데이트는 패치의 의미와 더불어, 그림 1-10의 Windows 업데이트처럼 주기적으로 추가되는 버전, 새롭게 추가되는 기능을 담은 버전이라는 의미도 포괄



[그림 1-10 | Windows 업데이트]

■ 버전과 버전 관리 이해하기

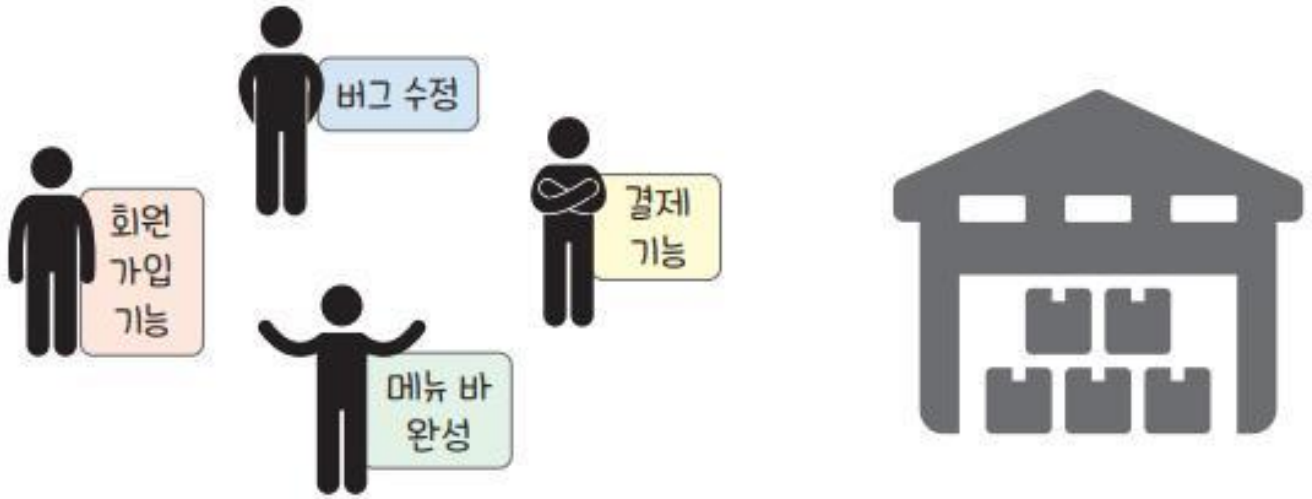
- 버전, 패치, 업데이트는 보통 '용어의 사전적 정의'나 '말'로써 엄격히 구분 짓지는 않으니 이 말들의 정의를 암기하지 않아도 괜찮음
- 버전, 패치, 업데이트, 즉 소프트웨어 변경의 종류는 용어의 사전적 정의보다 버전을 작성하는 규칙을 통해 구분
- 그림 1-11의 [v9.3.2],[v9.3.0]과 같이 점을 기준으로 적힌 숫자 세 개가 버전 규칙에 따른 버전 표기의 예
- 이러한 숫자 표기로 소프트웨어 변경의 종류를 나타낸다고 보면 됨



[그림 1-11 | 소프트웨어 버전 규칙의 예]

■ 버전과 버전 관리 이해하기

- 우리가 코드로 무언가를 만드는 일은 유의미한 변화(버전)들을 쌓아 올리는 것과 같음
- 벽돌이 모이고 모여 거대한 건물이 완성되듯, 버전이 모이고 모여 거대한 소프트웨어 결과물이 만들어지는 것



[그림 1-12 | 여러 버전이 쌓여 만들어지는 소프트웨어]

■ 버전과 버전 관리 이해하기

- 버전 관리는 앞에서 언급한 문제를 발생시키지 않으면서 유의미한 변화를 쌓아 올리며 소프트웨어를 만들어 나가는 과정이라 볼 수 있음
- 다시 말해, 개발에서 버전 관리는 다음과 같이 정리할 수 있음

> • 누가, 어떻게 변경했는지 변경 내역들을 기억하며
> • 필요하다면 특정 시점의 버전으로 되돌리며
> • 여러 명이 협업하는 과정에서 코드를 쉽게 나누고 합치며
> • 개발하는 것

■ 깃(Git)

- 여러분은 깃을 이용해 버전을 만들고 되돌리며, 다른 개발자들과 협업할 수도 있음
- 깃은 리눅스의 아버지 리누스 토르발스(Linus Torvalds)가 전 세계 수많은 개발자와 함께 오픈 소스 프로젝트(리눅스 커널)를 진행하다가 버전 관리에 어려움을 느껴 만든 도구
- 깃 또한 오픈 소스 프로젝트로, 모든 소스 코드가 공개되어 있음

  → https://github.com/git/git
- 깃은 그림 1-14처럼 명령어로 이용하는 소프트웨어이기 때문에 깃을 제대로 활용하려면 깃 명령어와 옵션을 숙지

■ 깃(Git)

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test
$ git init
Initialized empty Git repository in C:/test/.git/

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ echo "This is a test file for git" >> test.txt

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ ls
test.txt

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ git add .
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ git commit -m "first commit"
[master (root-commit) 3db7636] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ git log
commit 3db763638c7967b423c946f5175e09ec2e59365c (HEAD -> master)
Author: Kang Minchul <tegongkang@gmail.com>
Date:   Sun Aug 8 22:50:44 2021 +0900

    first commit

minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$
```

[그림 1-14 | 명령어로 동작하는 깃]

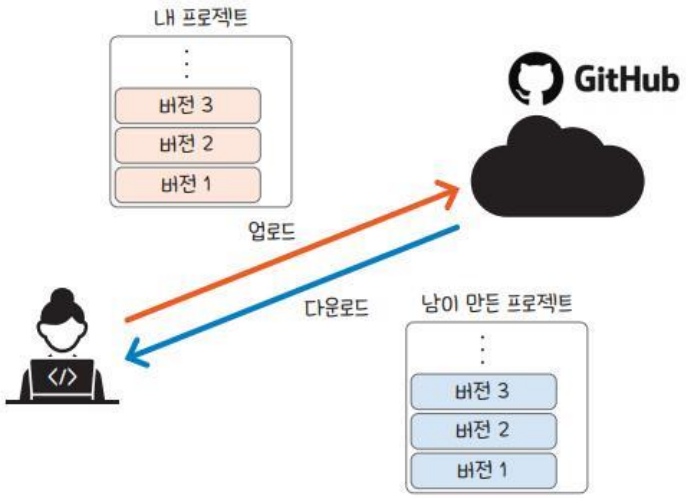
■ 깃허브(Github)

- 깃허브는 원격 저장소 호스팅 서비스
- '원격 저장소'라는 말이 조금 생소하겠지만,
  지금은 '깃으로 버전을 관리하는 프로젝트들이 모여 있는 웹 사이트' 정도로 생각해도 무방함

[그림 1-17 | 깃허브]

■ 깃허브(Github)

- 깃으로 버전 관리한 프로젝트를 깃허브에 업로드할 수 있고,
  깃허브에 업로드한 여러분의 프로젝트에 새로운 버전을 추가할 수도 있음
- 또 반대로 깃허브에 업로드된 전 세계 개발자들의 프로젝트를 여러분의 컴퓨터로 다운로드할 수도 있음
- 텐서플로(tensorflow), 쿠버네티스(kubernetes), 리액트(react) 등
  이름만 들어도 알 만한 유명한 프로젝트들이 이미 깃허브에 업로드되어 있음
- 여러분은 깃허브에 업로드된 프로젝트에 코드를 기여하고, 다른 개발자들과 협업할 수도 있음



[그림 1-18 | 깃으로 버전을 관리한 프로젝트의 업로드 및 다운로드가 가능한 깃허브]

■ 깃 설치하고 설정하기

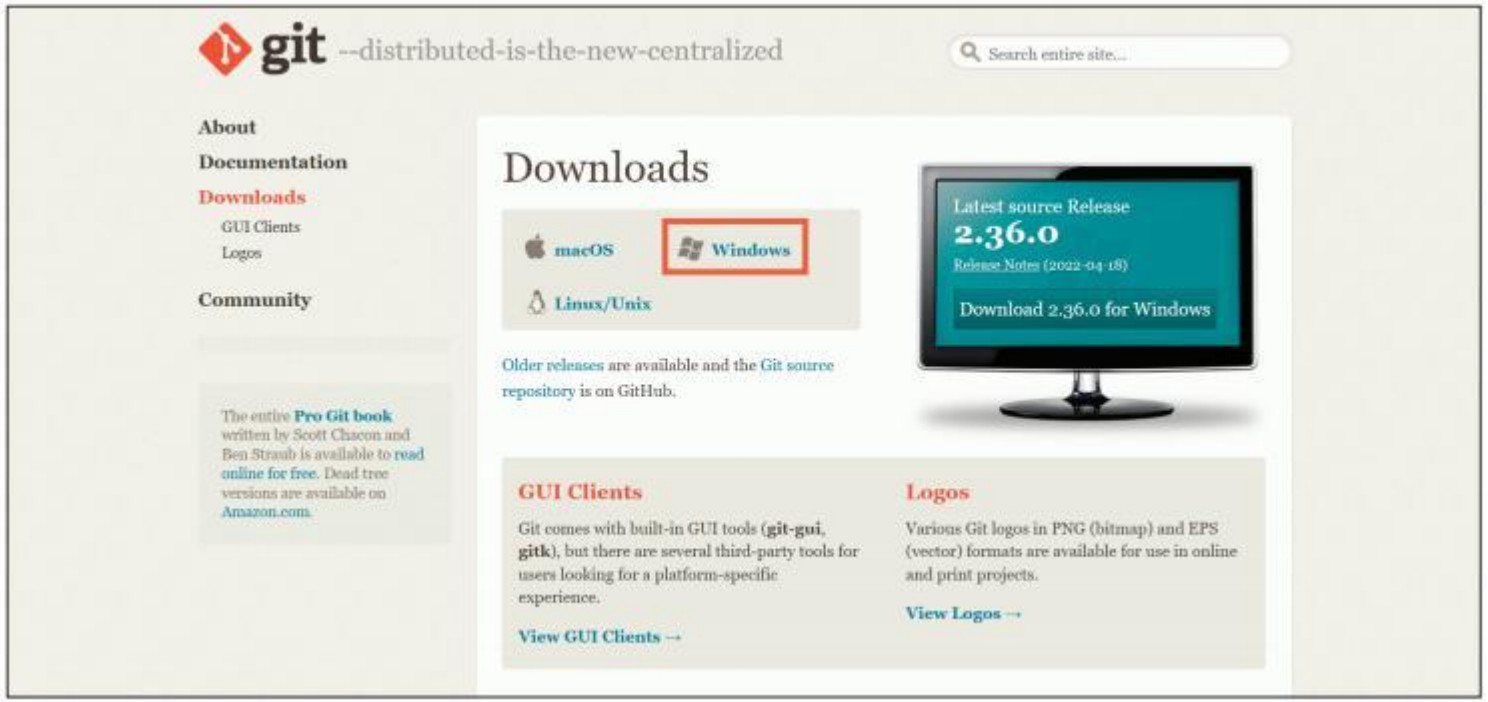
- 1 깃 홈페이지에서 설치 파일을 받을 수 있음
- 홈페이지에 접속한 후 Downloads를 클릭

  → https://git-scm.com

[그림 1-19 | 깃 홈페이지 화면]

■ 깃 설치하고 설정하기

- 2 윈도 운영 체제에 설치할 예정이니 Windows를 선택
- 그러면 설치 파일이 자동으로 내려받아짐



[그림 1-20 | 운영 체제 선택하기]

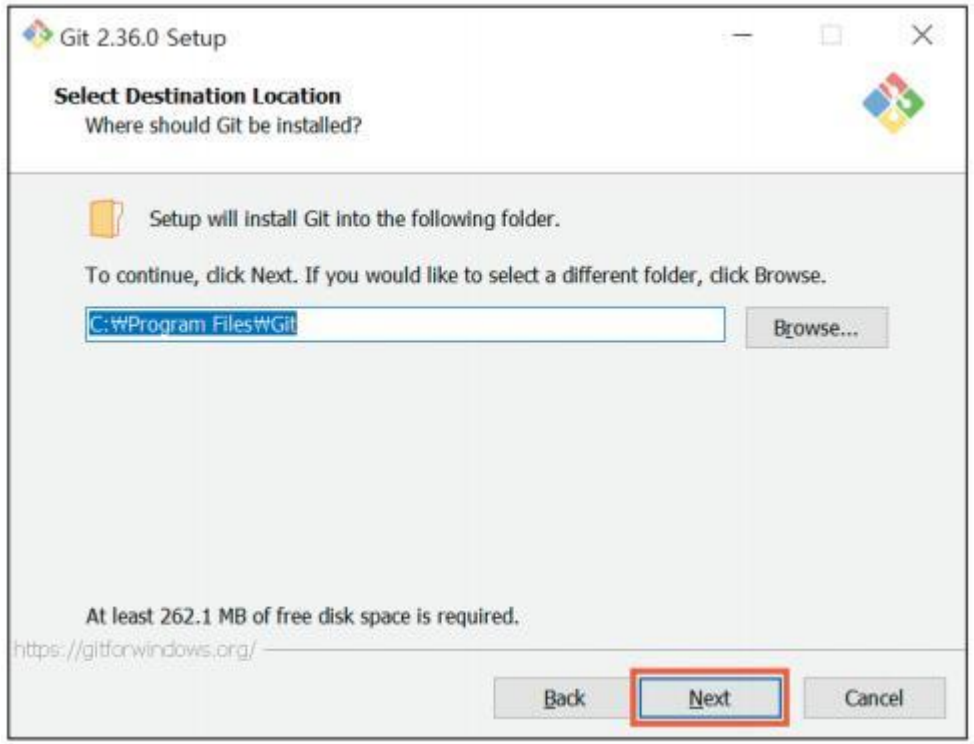
■ 깃 설치하고 설정하기

- 3 내려받은 파일을 실행
- 초기 설치 화면이 나오면 Next를 누름

[그림 1-21 | 초기 설치 화면]

■ 깃 설치하고 설정하기

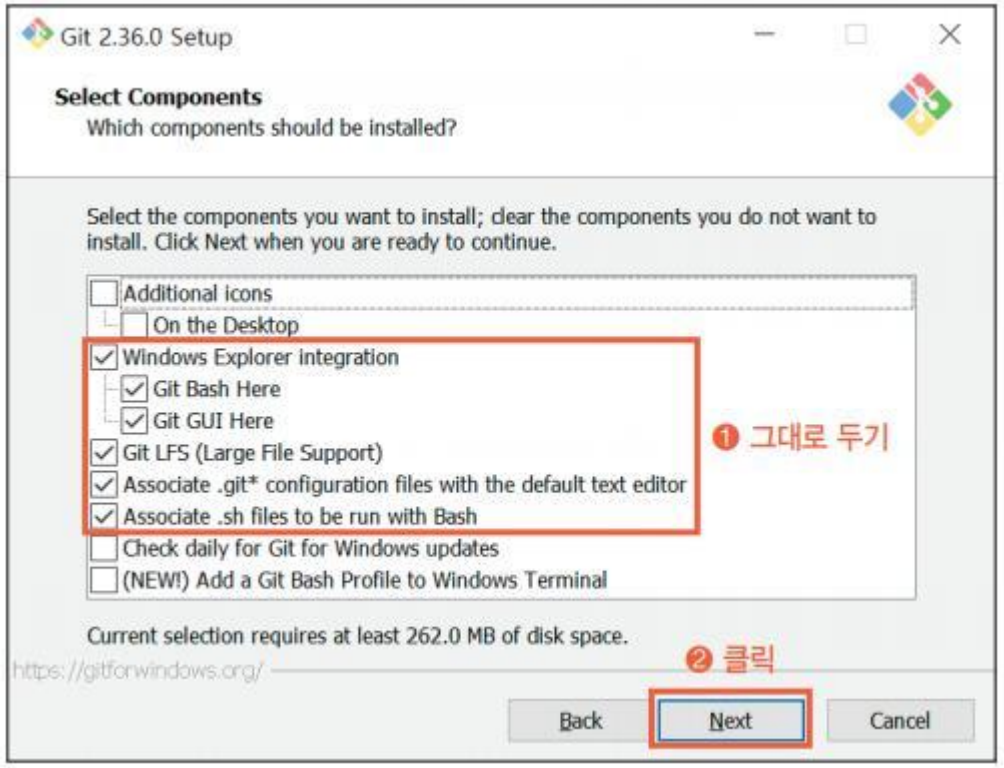
- 4 깃을 설치할 경로를 지정
- 별도의 공간에 깃을 설치하고 싶다면 설치하려는 경로를 입력하면 됨
- 이 책에서는 기본으로 설정된 경로에 설치하므로 그대로 Next를 누름



[그림 1-22 | 설치 경로 선택하기]

■ 깃 설치하고 설정하기

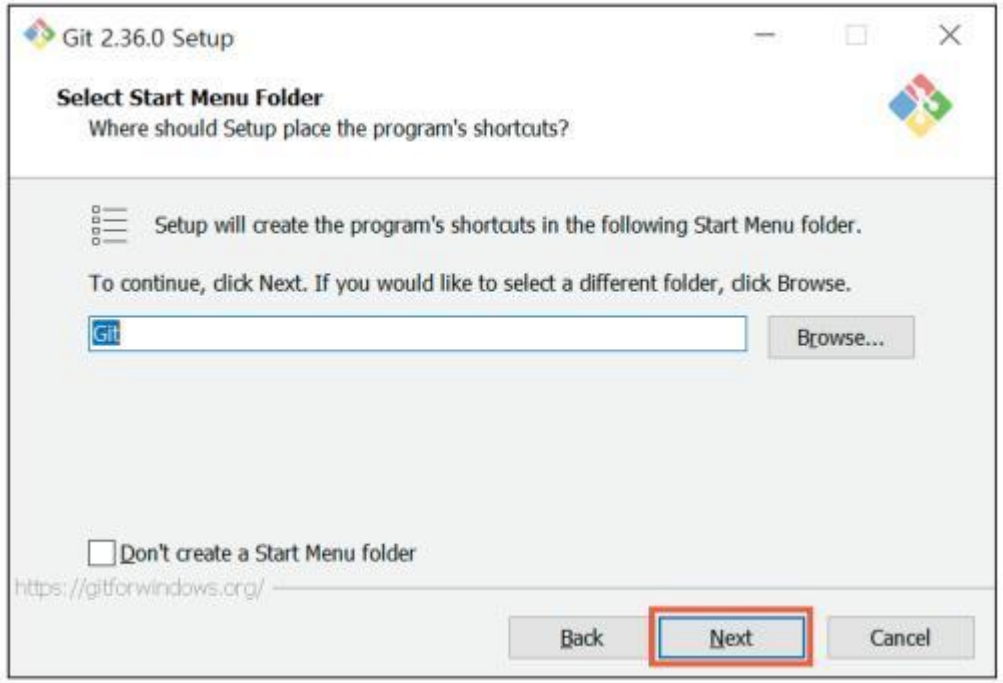
- 5 깃과 함께 설치할 것을 선택하는 창이 나옴
- 기본으로 체크되어 있는 항목을 그대로 두고 Next를 누름



[그림 1-23 | 함께 설치할 프로그램 선택하기]

■ 깃 설치하고 설정하기

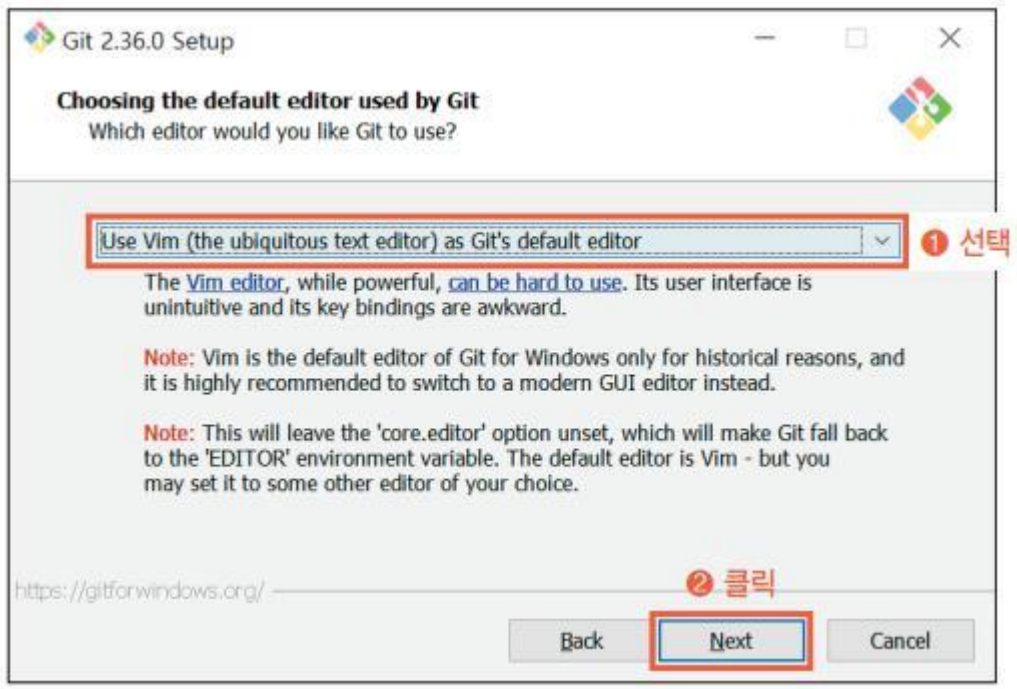
- 6 시작 메뉴에 깃 폴더를 생성하는 항목
- 시작 메뉴에 깃 폴더를 만들고 싶지 않다면 Don’t create a Start Menu folder에 체크한 다음 Next를 누르면 됨
- 이 책에서는 체크하지 않은 상태로 진행



[그림 1-24 | 시작 메뉴에 만들 깃 폴더 설정하기]

■ 깃 설치하고 설정하기

- 7 다음으로 깃에서 사용할 기본 문서 편집기(에디터)를 선택하는 창이 나옴
- 이 책에서는 Vim을 사용할 예정이므로 Use Vim (the ubiquitous text editor) as Git's default editor를 선택하고 Next를 누름



[그림 1-25 | 기본 에디터 선택하기]

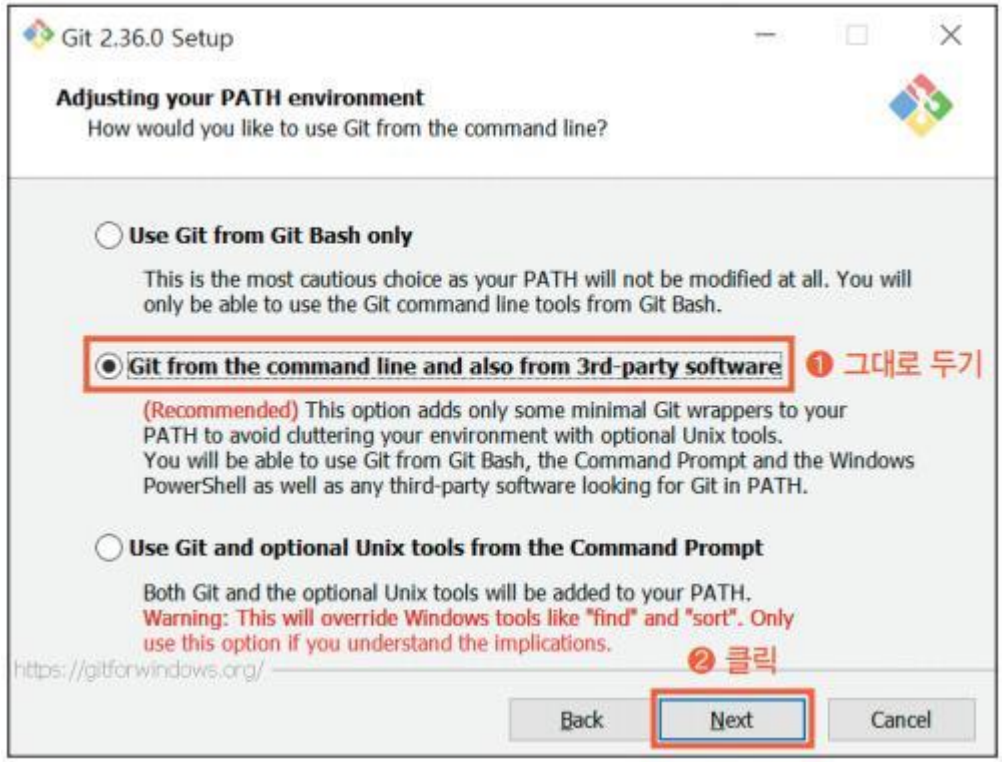
■ 깃 설치하고 설정하기

- 8 initial branch의 이름을 정하는 항목이 나옴
- 기본으로 선택된 Let Git decide를 그대로 두고 Next를 누름

[그림 1-26 | initial branch 이름 결정하기]

■ 깃 설치하고 설정하기

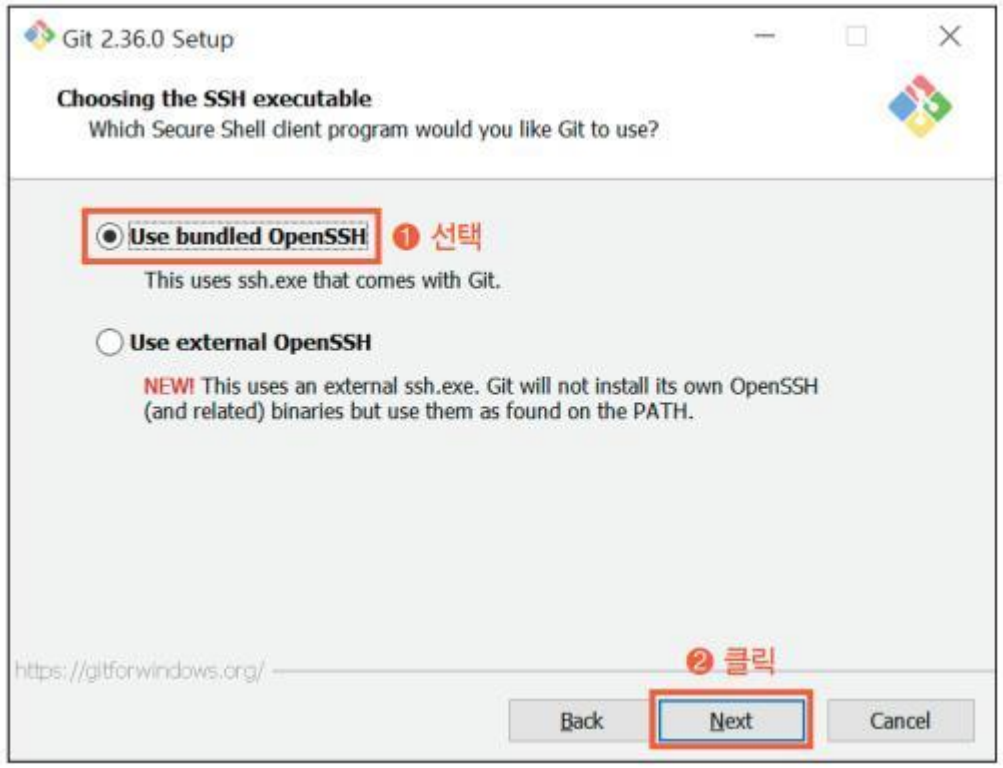
- 9 환경 변수 설정에 대한 선택 항목이 나옴
- 여기에서도 기본으로 선택된 Git from the command line and also from 3rd-party software를 그대로 두고 Next를 누름



[그림 1-27 | 환경 변수 설정하기]

■ 깃 설치하고 설정하기

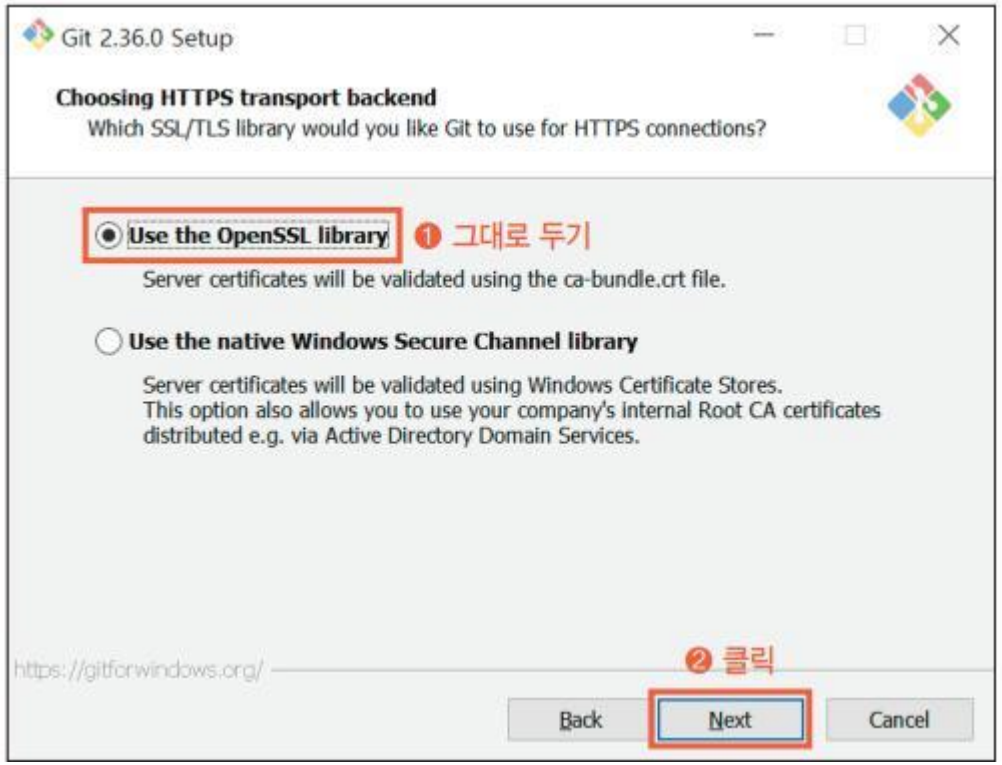
- 10 SSH 클라이언트 프로그램을 선택하는 항목이 나옴
- 여기에서는 Use bundled OpenSSH를 선택하고 Next를 누름



[그림 1-28 | SSH 클라이언트 선택하기]

■ 깃 설치하고 설정하기

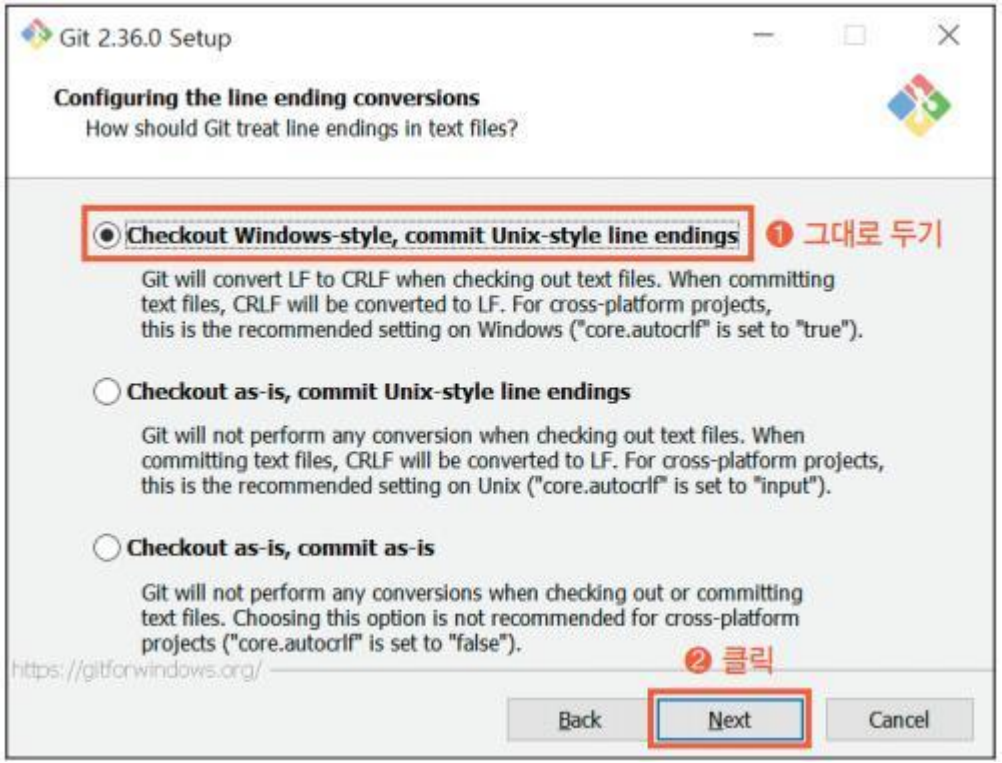
- 11 이번에는 SSL/TLS 라이브러리를 선택
- 마찬가지로 기본으로 선택된 Use the OpenSSL library를 그대로 두고 Next를 누름



[그림 1-29 | SSL/TLS 라이브러리 선택하기]

■ 깃 설치하고 설정하기

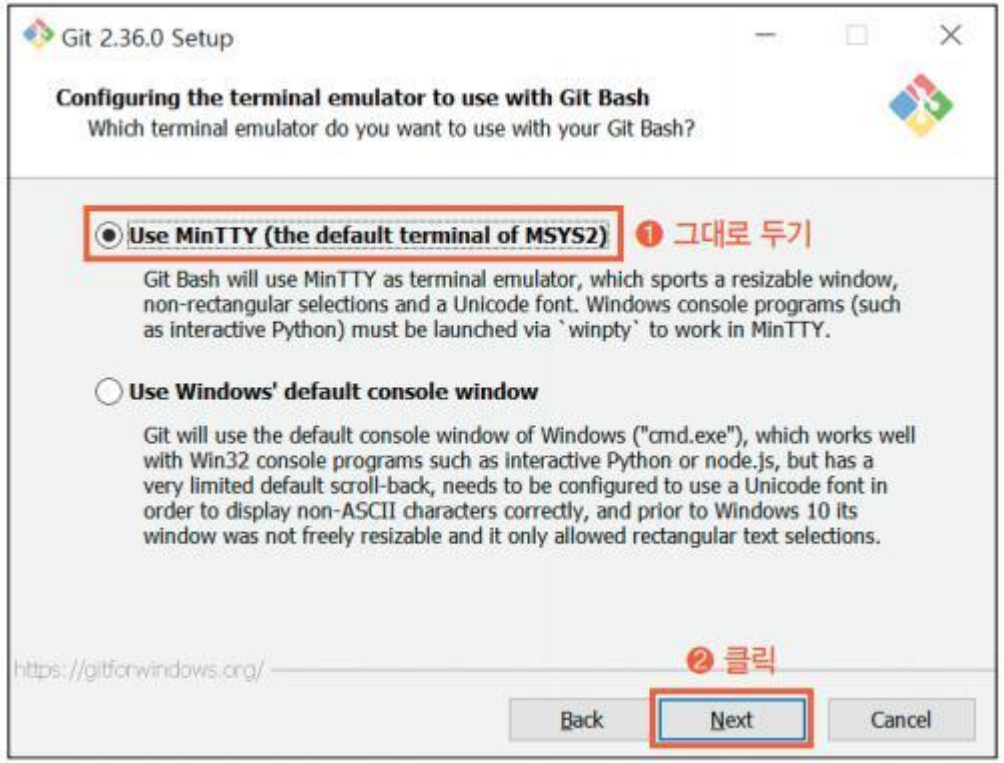
- 12 개행(줄 바꿈)에 관련한 설정
- 기본으로 선택된 첫 번째 항목 Checkout Windows-style, commit Unix-style line endings를 그대로 두고 Next를 누름



[그림 1-30 | 개행 설정하기]

■ 깃 설치하고 설정하기

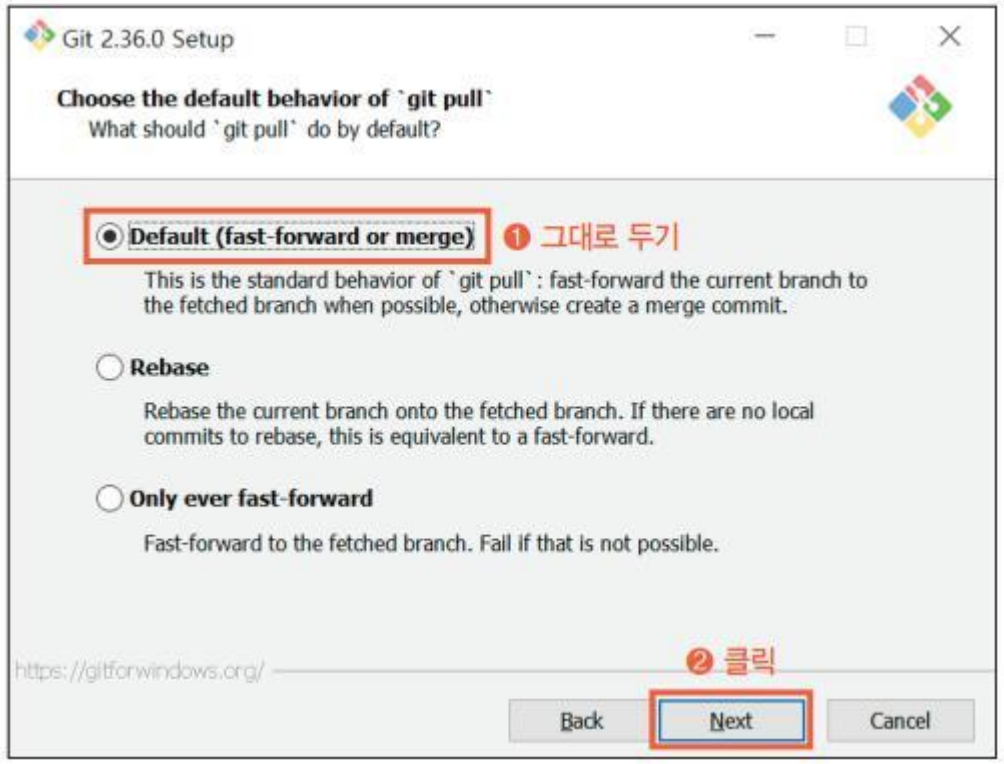
- 13 다음은 터미널을 선택하는 항목
- 기본으로 선택된 Use MinTTY (the default terminal of MSYS2)를 그대로 두고 Next를 누름



[그림 1-31 | 터미널 설정하기]

■ 깃 설치하고 설정하기

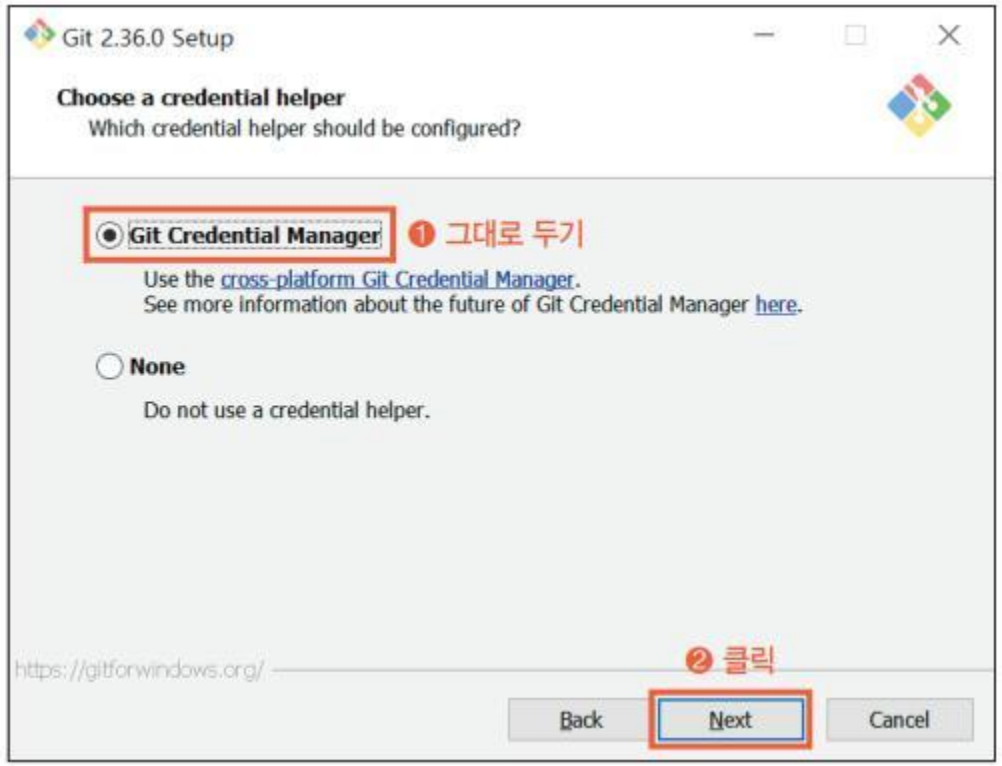
- 14 git pull 명령에 대한 기본 동작을 설정
- 기본으로 선택된 Default (fast-forward or merge)를 그대로 두고 Next를 누름



[그림 1-32 | git pull 명령에 대한 동작 설정하기]

■ 깃 설치하고 설정하기

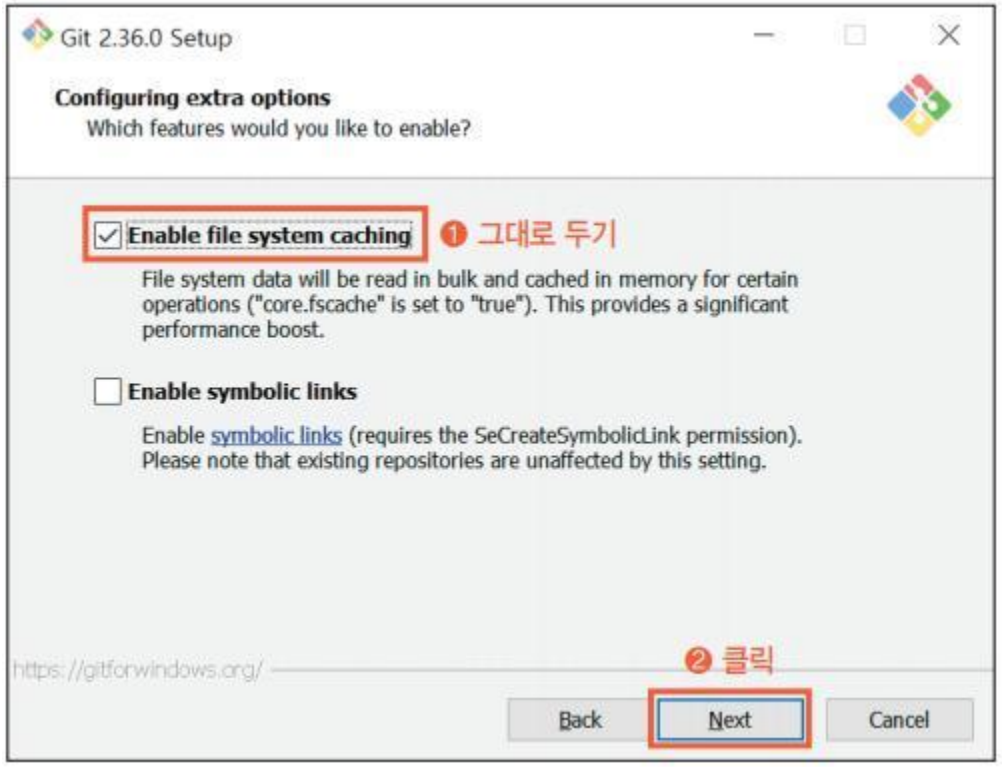
- 15 인증과 관련한 설정 화면이 나옴
- Git Credential Manager를 그대로 두고 Next를 누름



[그림 1-33 | 깃 인증 설정하기]

■ 깃 설치하고 설정하기

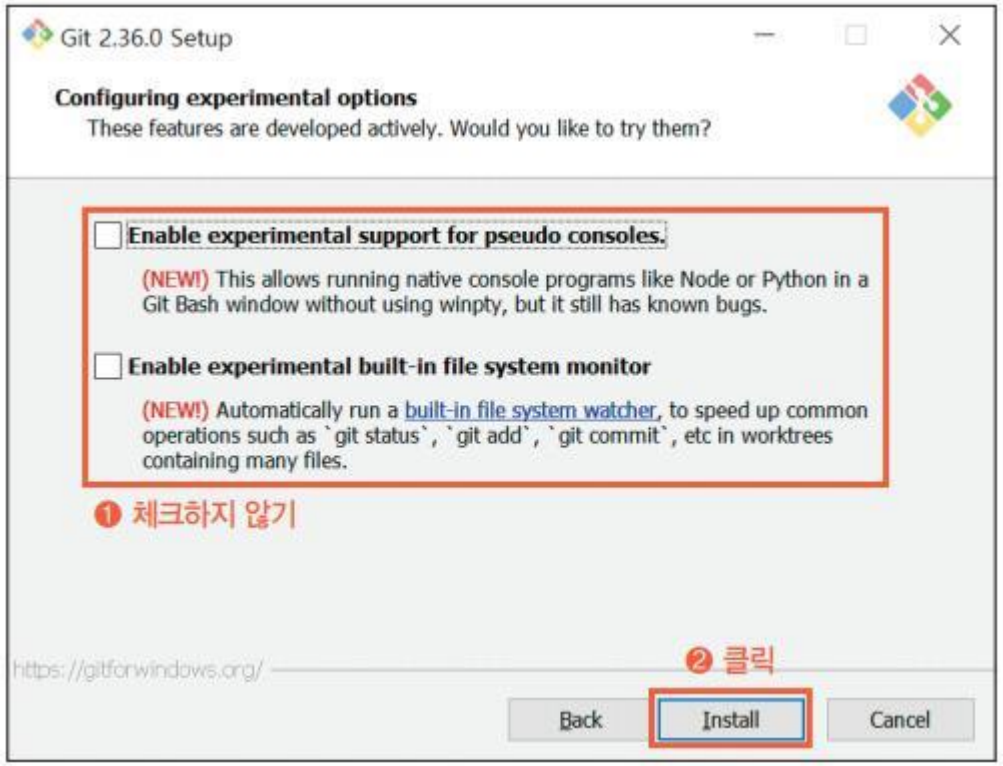
- 16 이 항목은 기타 옵션을 선택하는 항목
- 기본으로 선택된 Enable file system caching을 그대로 두고 Next를 누름



[그림 1-34 | 기타 옵션 설정하기]

■ 깃 설치하고 설정하기

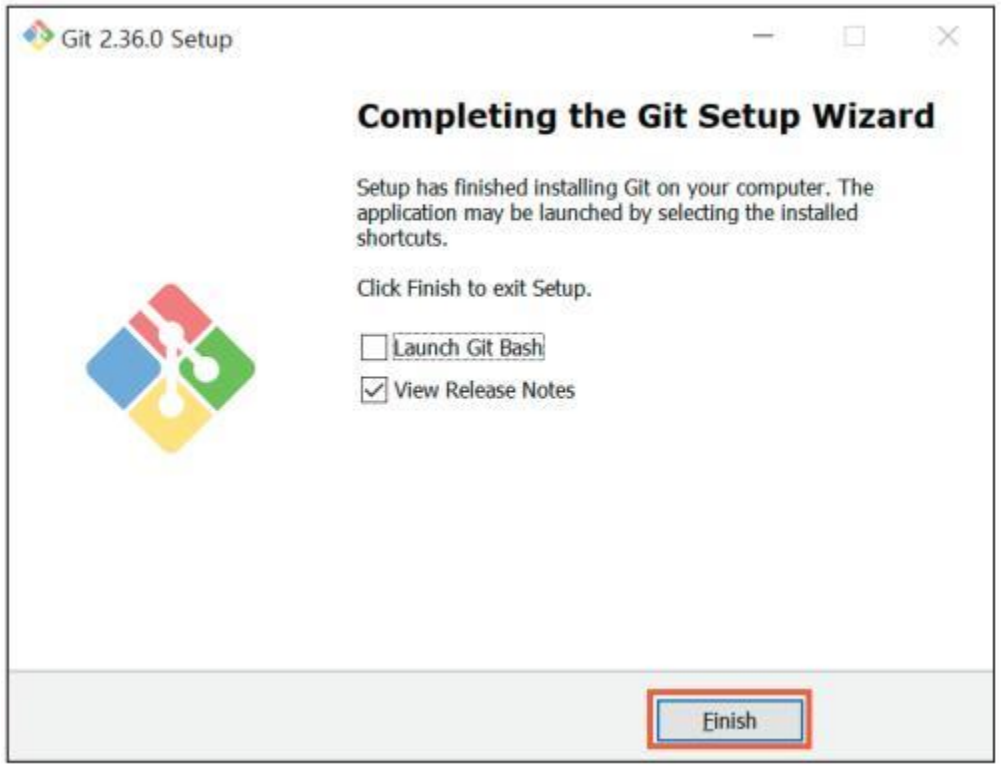
- 17이 항목은 아직 완전하지 않은 실험적 기능에 대한 설정으로, 체크하지 않아도 무방함
- 이 책에서는 체크하지 않고 Install을 눌러 설치를 진행



[그림 1-35 | 실험적 기능 활성화 여부 체크하기]

■ 깃 설치하고 설정하기

▪ 18 설치가 끝나면 Finish를 누름



[그림 1-36 | 깃 설치 완료 화면]

■ 깃 설치하고 설정하기

- 19 잘 설치됐는지 확인해 보자
- 바탕화면 등 편한 공간에 폴더를 만들고 그 안에서 마우스 오른쪽 버튼을 클릭하면 Git Bash Here 항목이 생겼을 것
- Git Bash Here를 클릭

[그림 1-37 | Git Bash Here 항목 확인하기]

■ 깃 설치하고 설정하기

- 20 다음과 같이 명령어를 입력할 수 있는 공간, 즉 깃 배시(git bash)가 나옴
- 여기에 깃 명령어를 직접 입력할 수 있음

[그림 1-38 | 깃 배시]

■ 깃 설치하고 설정하기

- 참고로 다음 그림에서 박스 친 부분이 현재 내가 명령어를 입력하고 있는 작업 공간, 다시 말해 현재 작업 공간을 의미
- 여기서는 C 드라이브의 test 폴더
- 내가 지금 어디에서 명령어를 사용하고 있는지는 매우 중요하므로 이 '현재 작업 공간'에 각별히 유의해야 함

[그림 1-39 | 현재 작업 공간 확인하기]

■ 깃 설치하고 설정하기

- 21 git 명령어를 입력해 잘 설치됐는지 확인
- 다음 그림처럼 git과 관련한 명령어 목록이 잘 뜨는 걸 확인했다면 현재 깃이 잘 설치된 것

```
minchul@DESKTOP-9KULGUE MINGW64 /c/test (master)
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

[그림 1-40 | 깃 설치 확인하기]

■ 깃 설정하기

- 1 깃을 잘 설치했다면 이제 본인 컴퓨터에 사용자 이름과 이메일을 등록하는 간단한 초기 설정을 해보자
- 앞으로 깃을 이용해 만드는 모든 버전에는 '만든 사람', '지은이'와 같은 개념으로 지금부터 설정할 이름과 이메일이 함께 명시될 것
- 다음과 같이 명령을 입력해 보자(이름은 가급적 영어를 사용할 것을 권장)

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --global user.name "Kang Minchul"
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --global user.email tegongkang@gmail.com
```

■ 깃 설정하기

- 2 설정한 이름과 이메일은 다음과 같은 명령으로 확인할 수 있음

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config user.name
Kang Minchul

minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config user.email
tegongkang@gmail.com
```

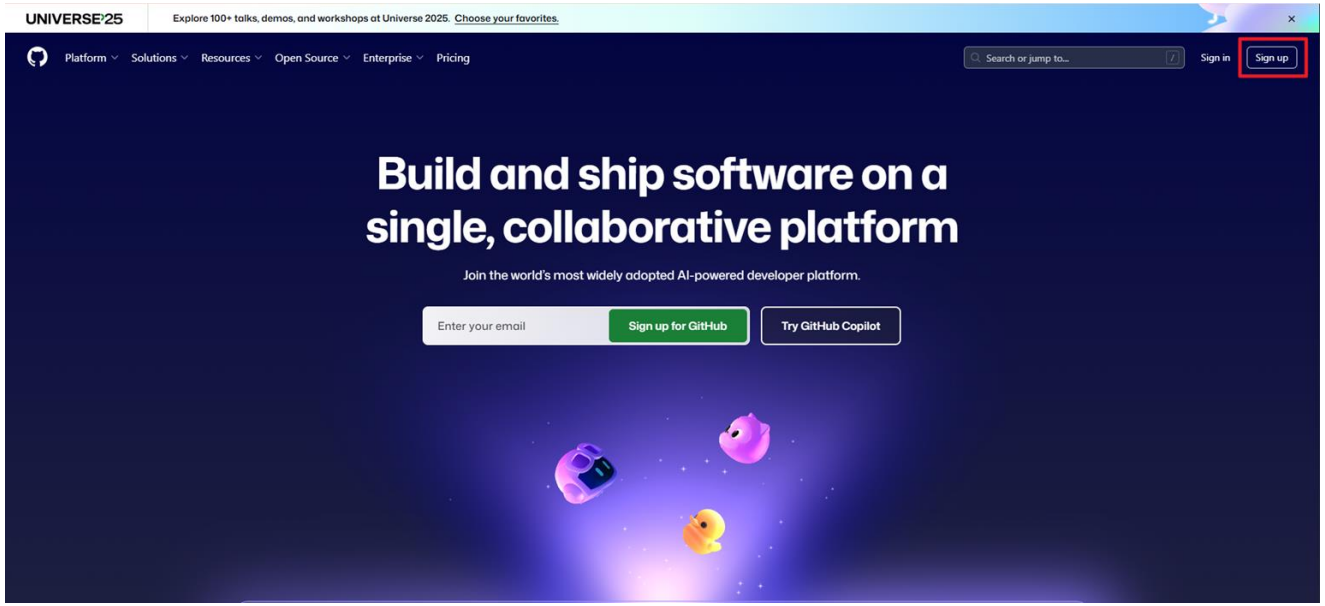
■ 깃 설정하기

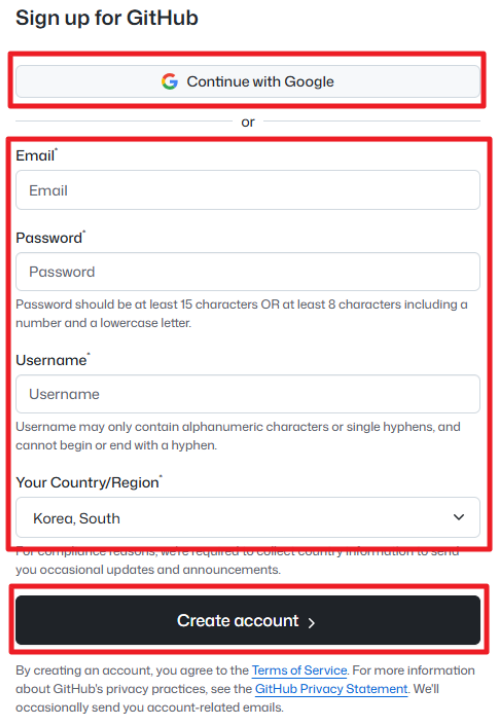
- 3 이처럼 git config 명령으로 깃과 관련한 내용을 설정하거나 설정한 값을 확인할 수 있음
- 다음 명령은 이름과 이메일뿐 아니라 다른 설정 값도 보여주는 명령
- 자세히 살펴보면 우리가 초기 설치 과정에서 선택한 항목들도 설정된 값으로 출력되는 걸 확인할 수 있음(지금 시점에서 설정 값의 의미 하나하나를 모두 알 필요는 없음)

```
minchul@DESKTOP-9KULGUE MINGW64 ~/c/test
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Kang Minchul
user.email=tegongkang@gmail.com
```

■ Git 계정 생성

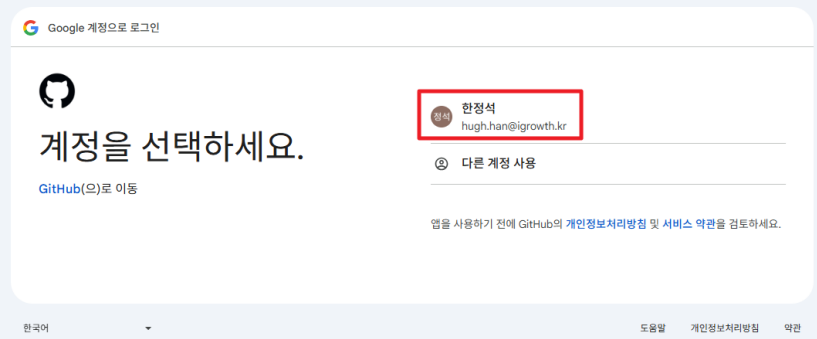
- URL : https://github.com/

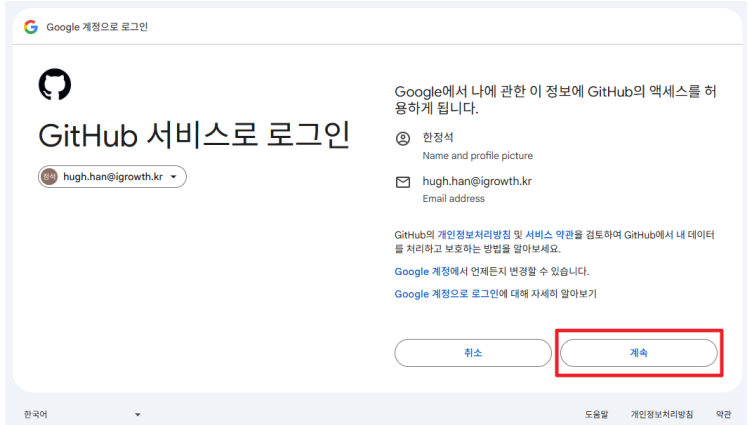


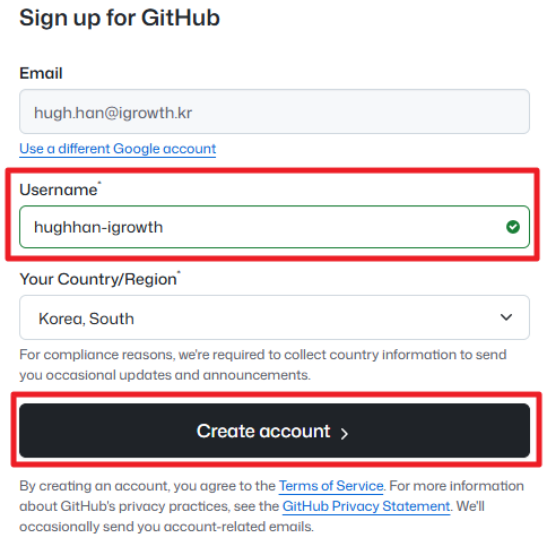


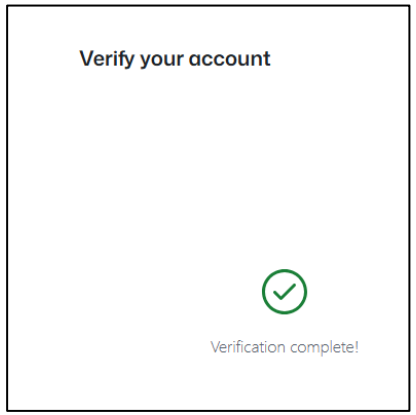
■ Git 계정 생성

- URL : https://github.com/



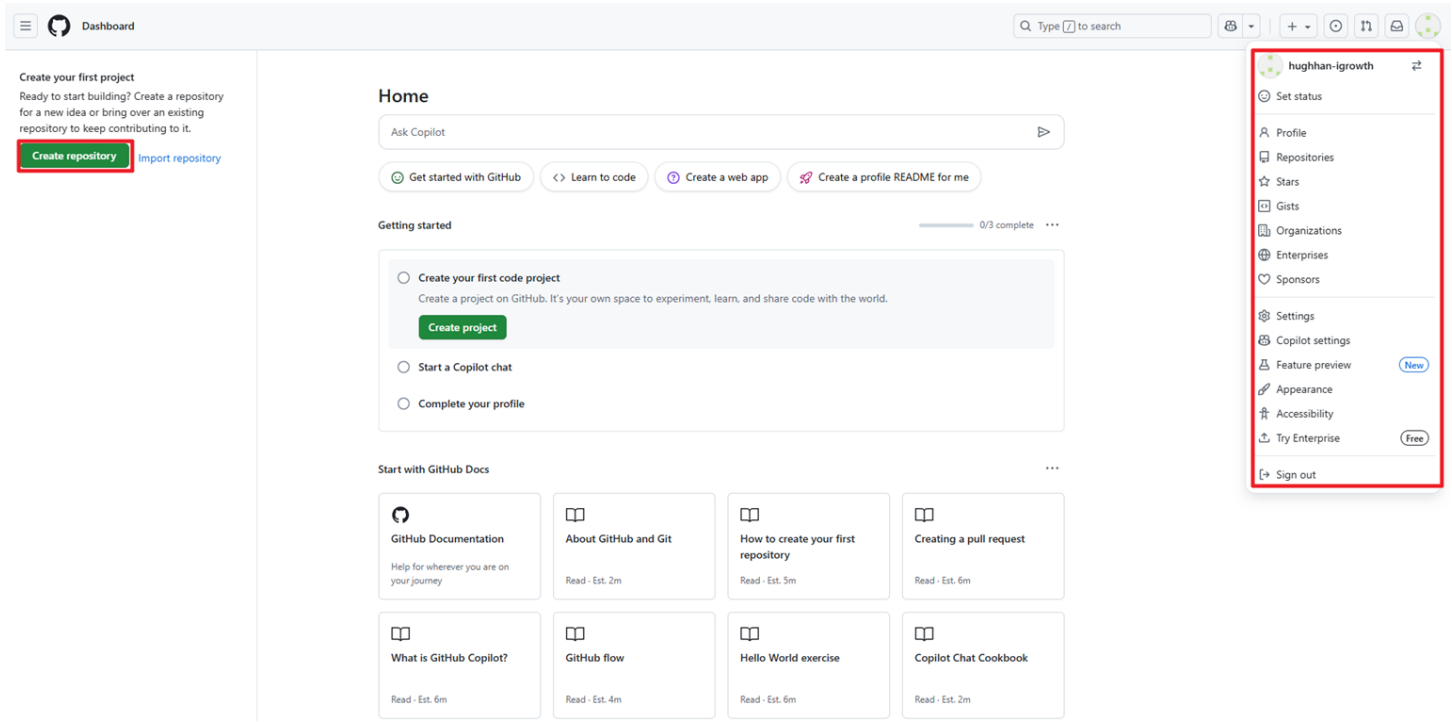






■ Git 계정 생성

- URL : https://github.com/



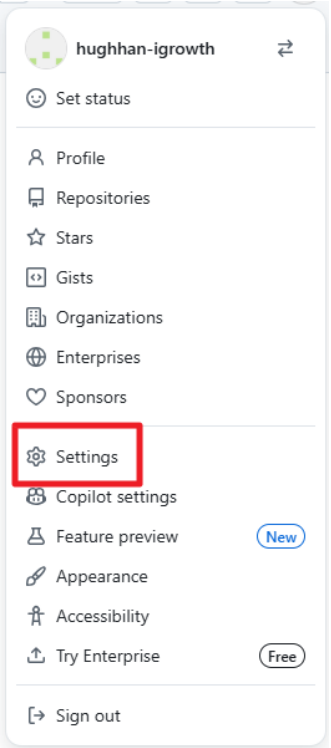
I Github Token 생성

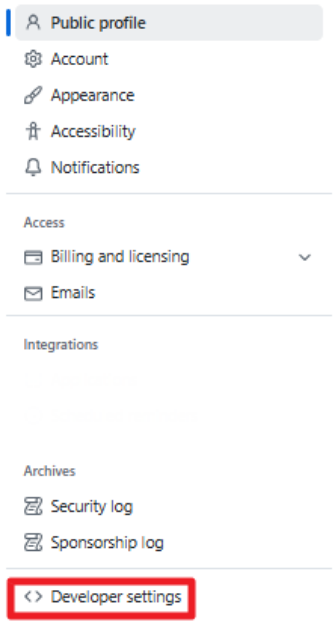
■ Token 이란?

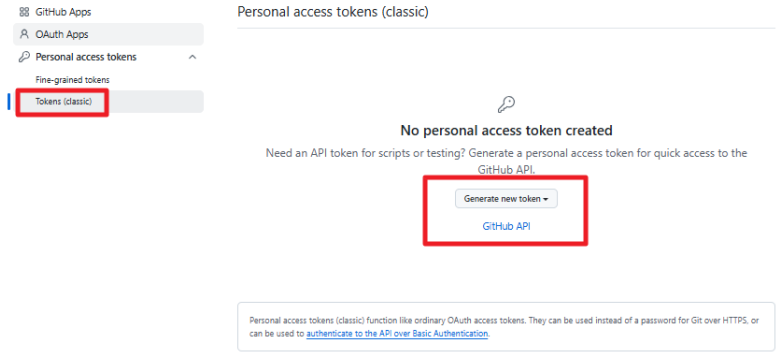
- 외부(개발환경) Tool에서 원격지(Remote)에 있는 GitRepository에 접근 하기위한 암호키

■ Token 생성 절차

- Profile > Setting > Developer settings > Grnerate new token(classic)

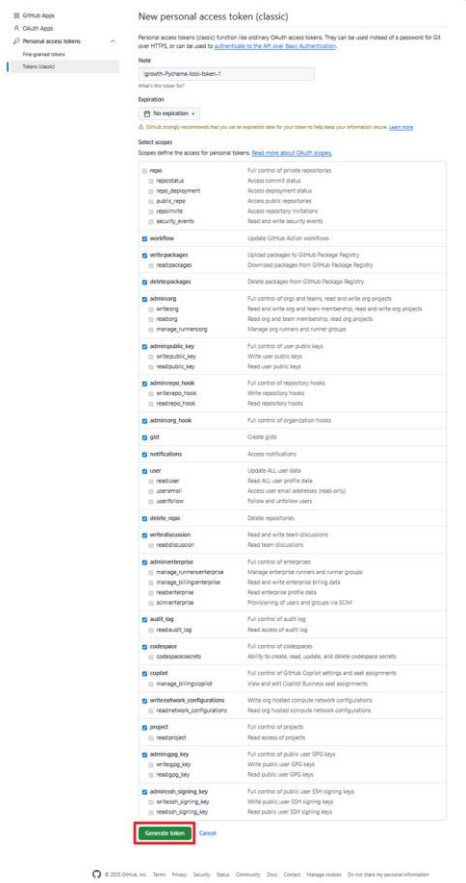


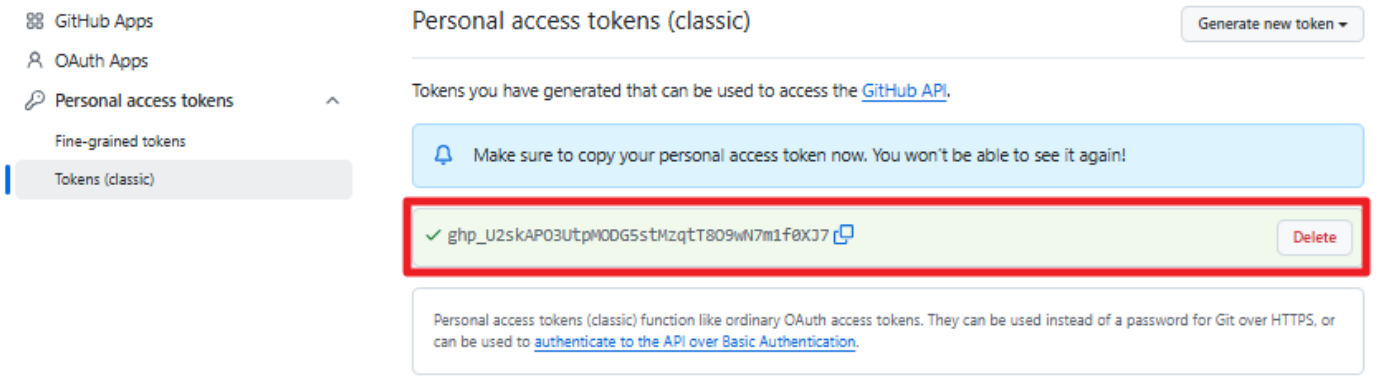




■ Token 생성 절차

- … > 설정 및 “Create” > 생성키 확인 및 보관

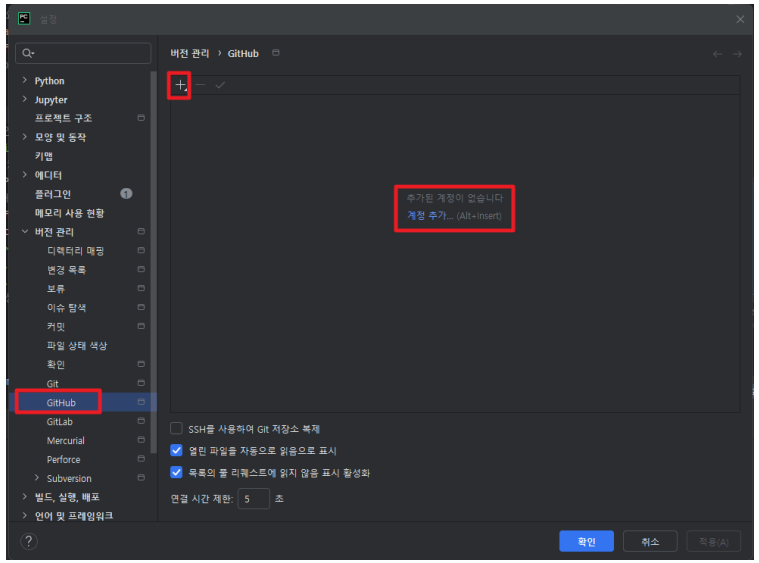


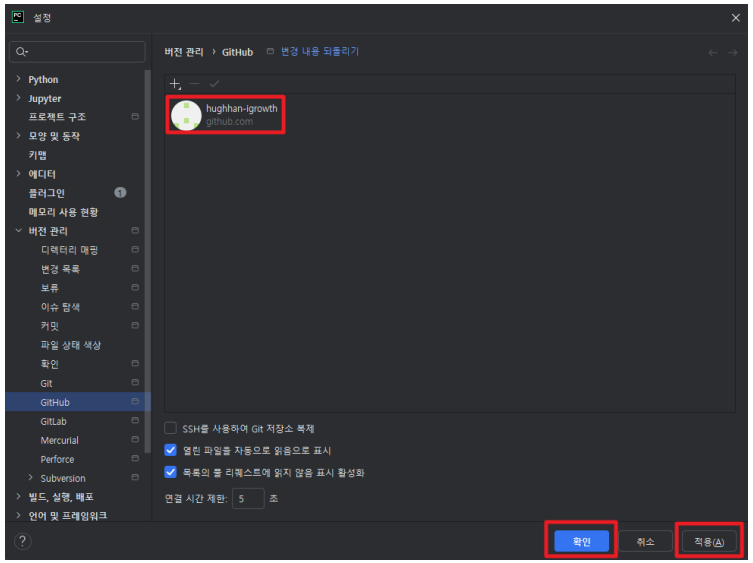


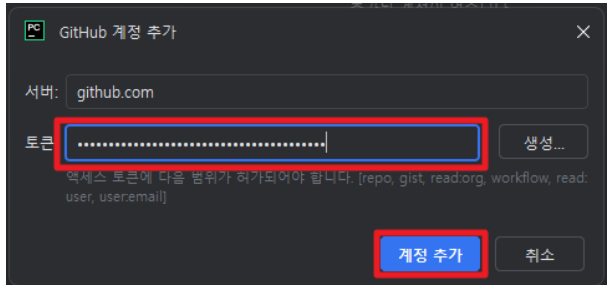
■ Github 연동 방법

- 절차

→ 설정 > 버전 관리 > GitHub > 계정 추가(+) > 토큰으로 로그인



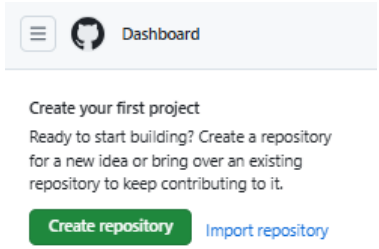


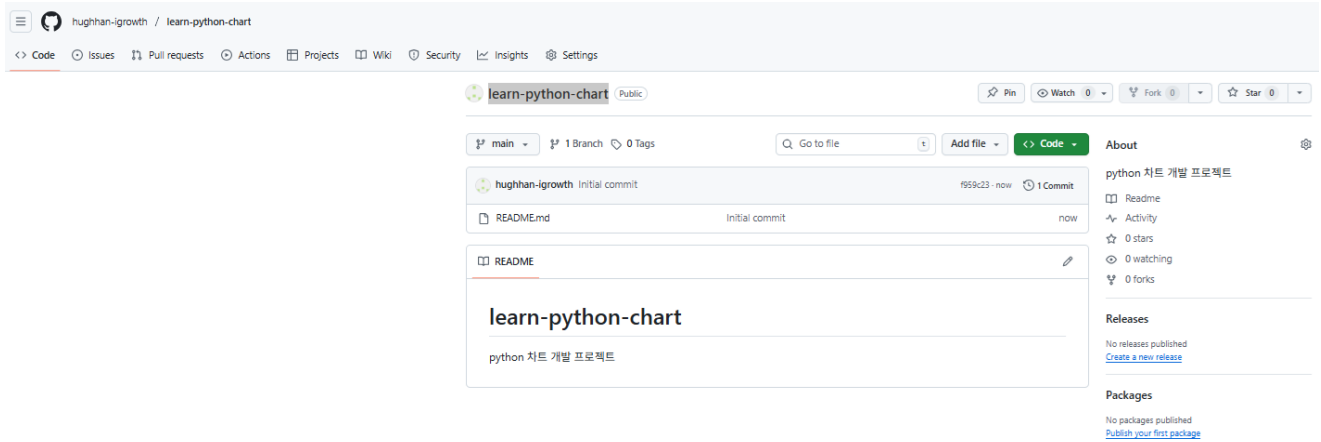


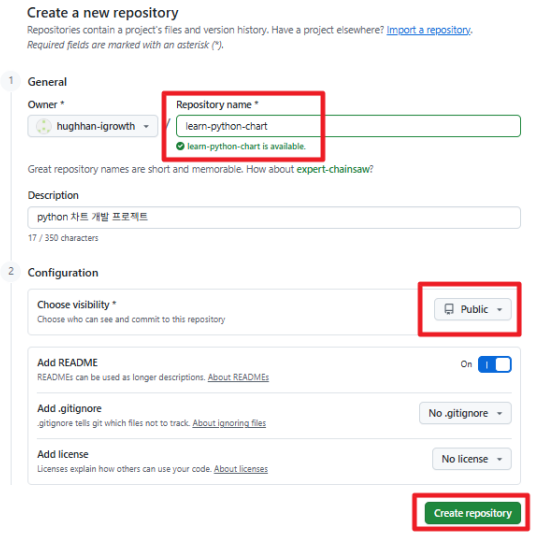
■ 신규 프로젝트 생성

- Github Repo 생성

→ “Create repository” > Repository 설정 및 생성



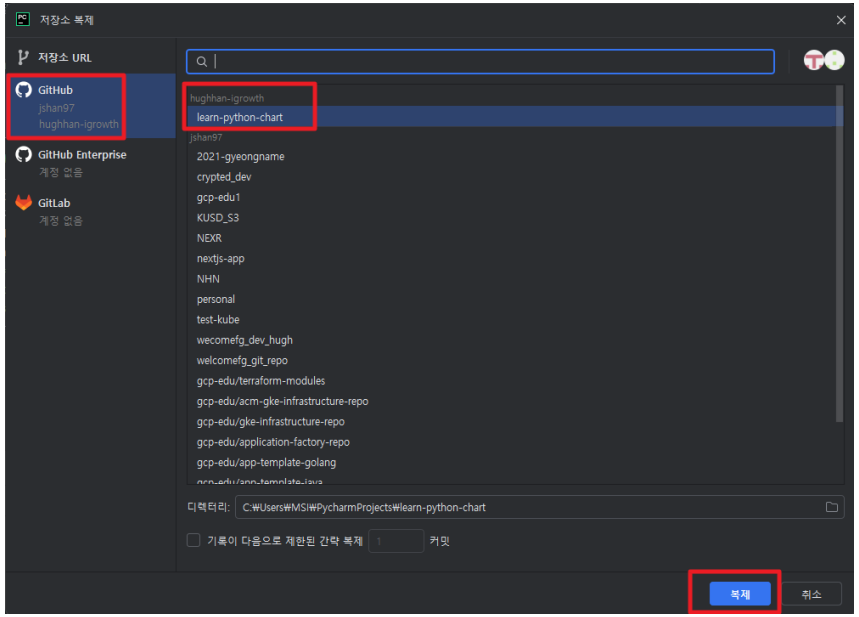


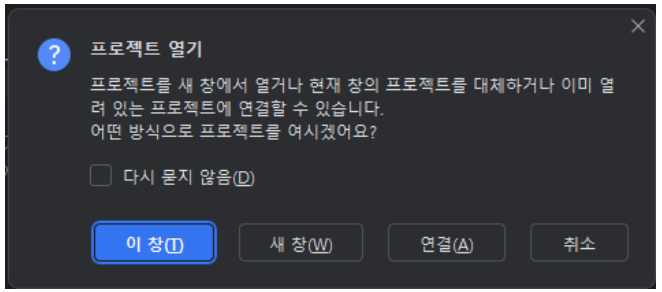


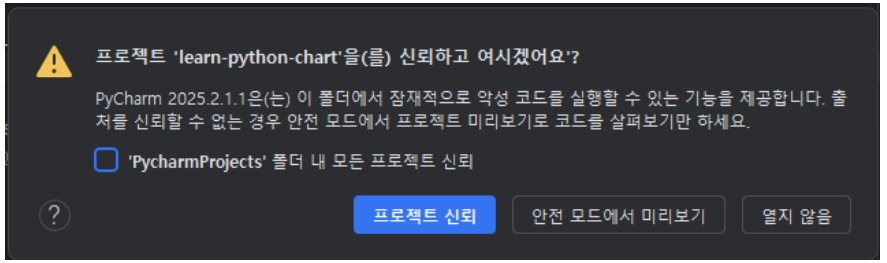
■ Pycharm에서 생성된 github repo 불러오기

- pycharm

  → 메뉴 > “버전 관리에 있는 프로젝트”



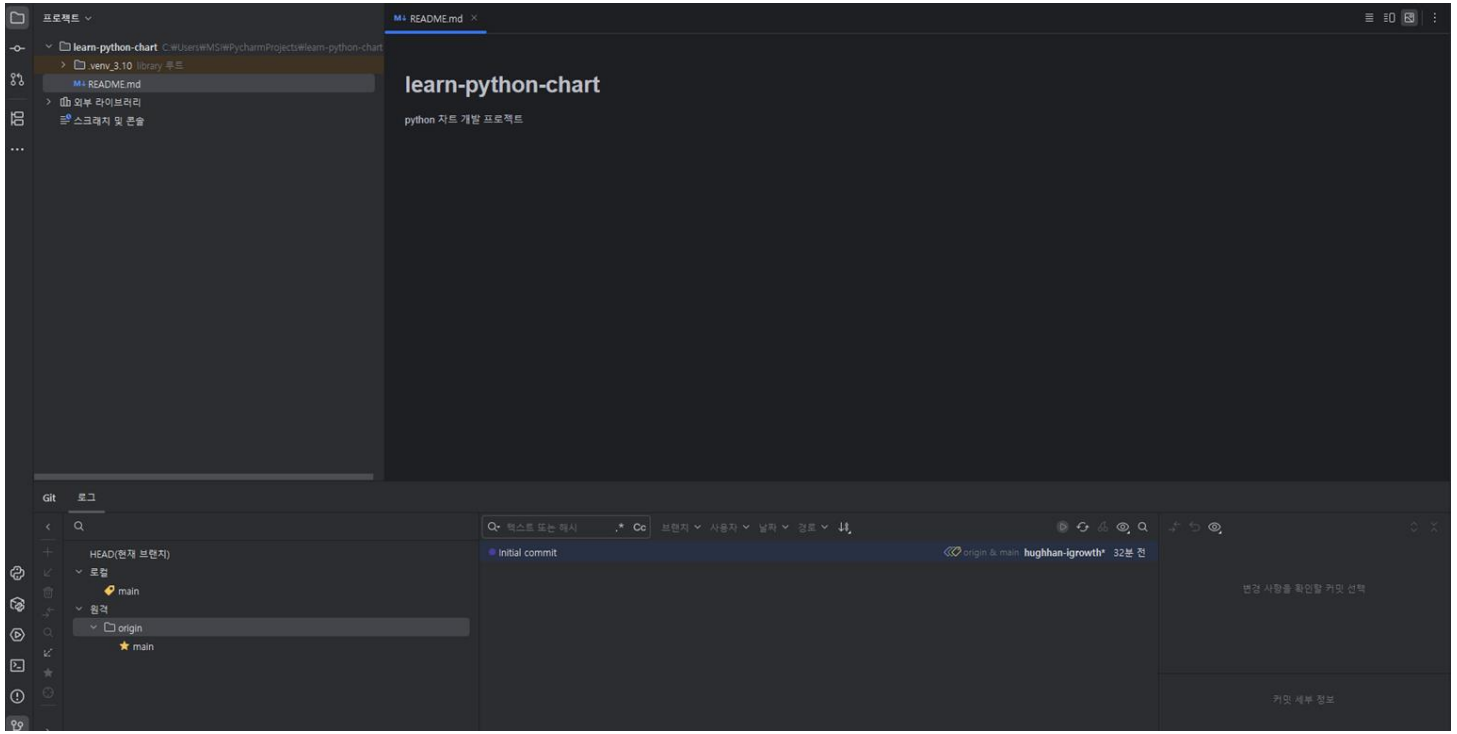




■ Pycharm에서 생성된 github repo 불러오기

- pycharm

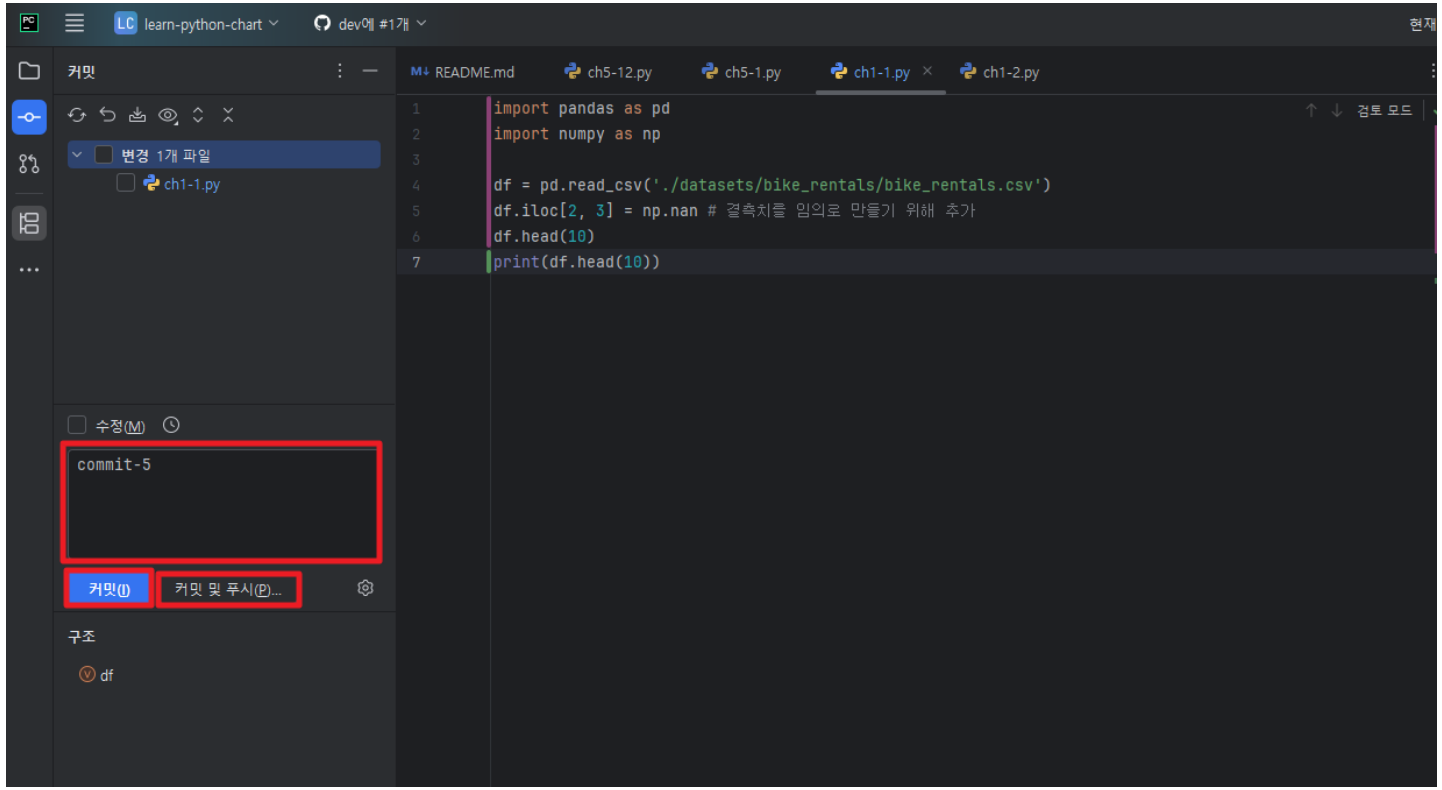
  → 메뉴 > “버전 관리에 있는 프로젝트”



■ 변경분 commit

- pycharm

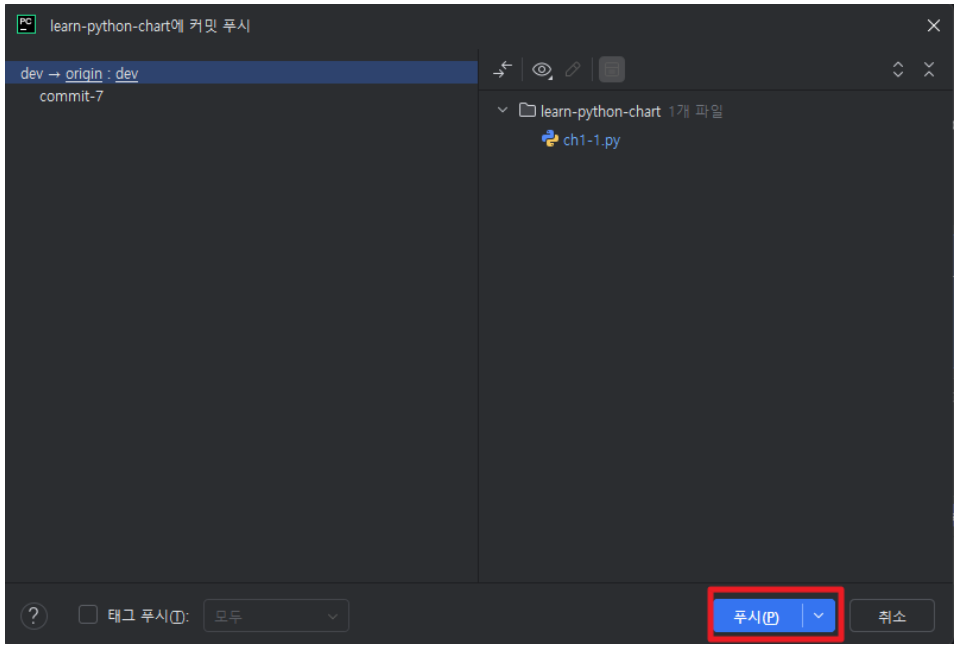
  → 메뉴 > “버전 관리에 있는 프로젝트”



■ 변경분 push

- pycharm

  → 메뉴 > Git > Push



■ 새로운 branch 생성

- pycharm

  → 메뉴 > Git > branch

■ Dev branch -> main branch 로 PR 생성 및 반영

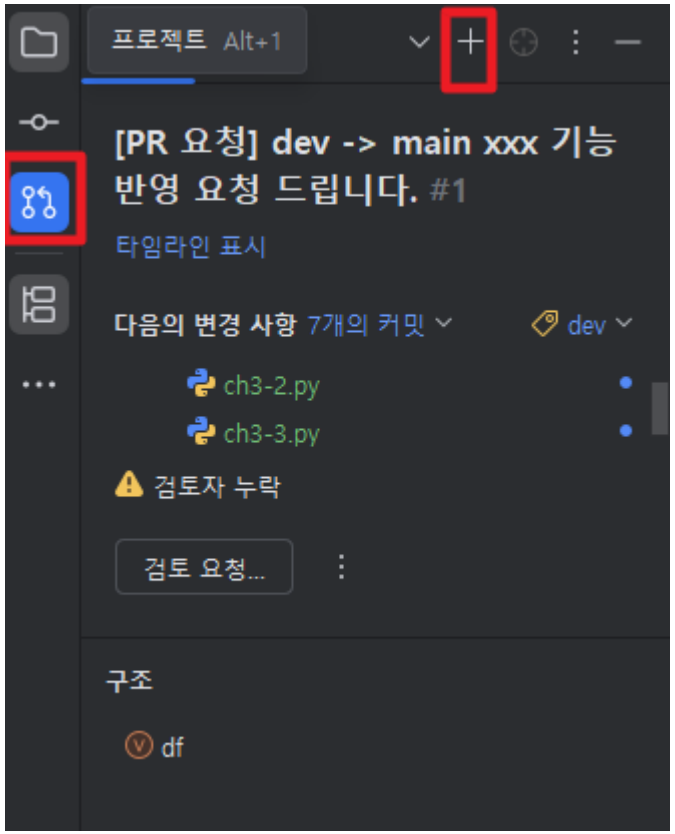
- Github 버전

  → 메뉴 > 풀리퀘스트

■ Dev branch -> main branch 로 PR 생성 및 반영

- pycharm 버전

  → 메뉴 > 풀리퀘스트



■ CLI에서 Git command를 사용한 작업

- pycharm 버전

  → 메뉴 > “버전 관리에 있는 프로젝트”

■ 새로운 맴버 코드 변경 가능하도록 구성

- pycharm

  → 메뉴 > “버전 관리에 있는 프로젝트”