Github

Git/GitHub 기본

1. 버전 관리 시스템
2. Git/GitHub 소개 및 환경 구축
3. GUI 도구와 CLI 환경 비교
4. 기본 명령어 실습

강사 양석환

버전관리 시스템

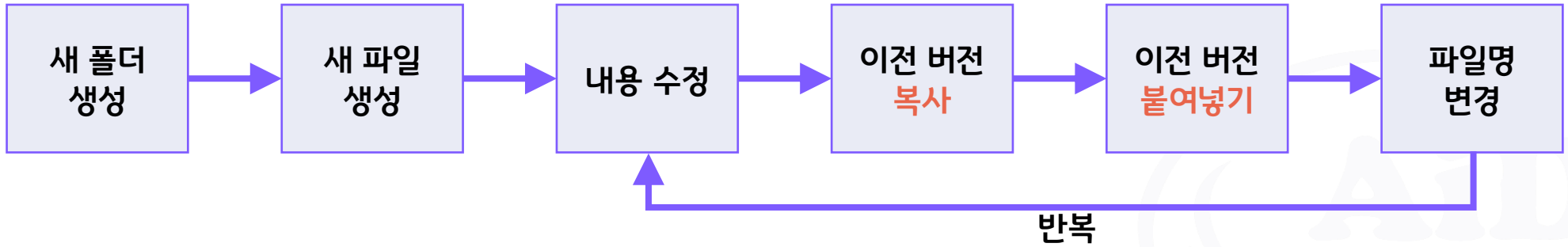
버전 관리 시스템이란 무엇인가?

- 버전 관리 시스템(Version Control System, VCS)
  - 문서나 설계도, 소스 코드 등의 변경점을 관리해 주는 소프트웨어(시스템)

- 버전 관리의 중요성
  - 변경점 관리: 어떤 내용을 누가 작성해서 어느 시점에 들어갔는지 확인해 줌
  - 버전 관리: 특정 시점에 Tag를 달아 버전을 표시, Branch 등으로 동시에 여러 버전을 개발할 수 있게 함
  - 백업&복구: 무엇인가 잘못되었을 때 다시 특정 시점으로 돌아가게 해 주고, 사고로 내용이 소실되었을 때에도 복구할 수 있게 함
  - 협업: 같이 일하는 사람들에게 수정사항을 쉽게 공유할 수 있음

버전 관리 시스템 없는 문서 관리

- **일반적인 프로세스**

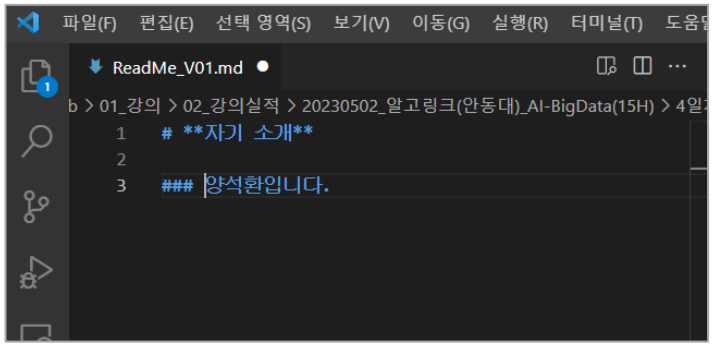


복사 & 붙여넣기 (Copy & Paste) 버전 관리

버전 관리 시스템 없는 문서 관리

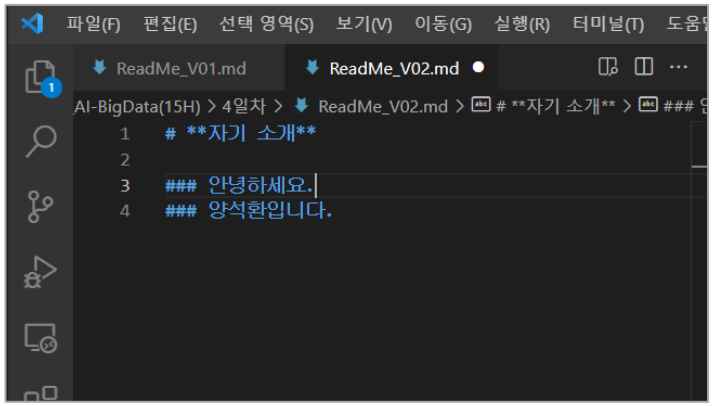
Github

- **일반적인 프로세스**
  - **버전 01 자기소개 작성**
    - (ReadMe_V01.md)



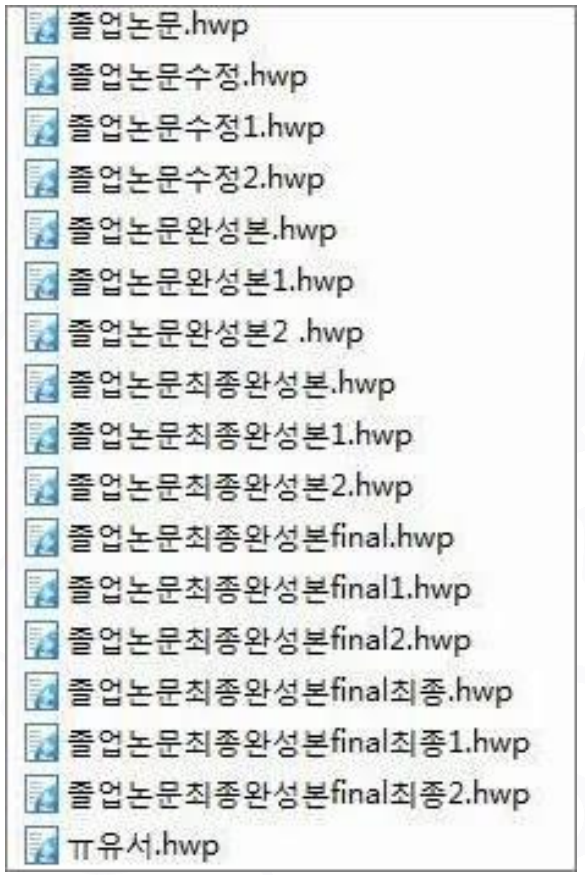
Ctrl+C & Ctrl+V

  - **버전 02 자기소개 작성**
    - (ReadMe_V02.md)



반복

결과

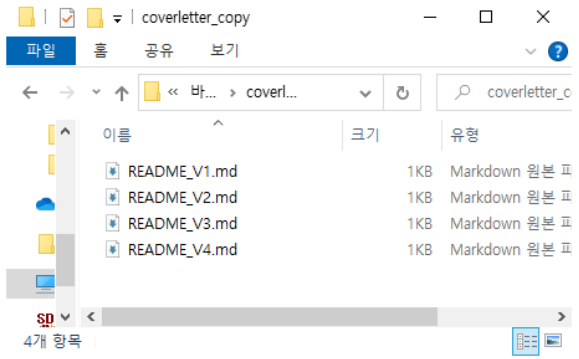


버전 관리 시스템 없이 문서를 작성한 경우

버전 관리 시스템 없는 문서 관리

Github

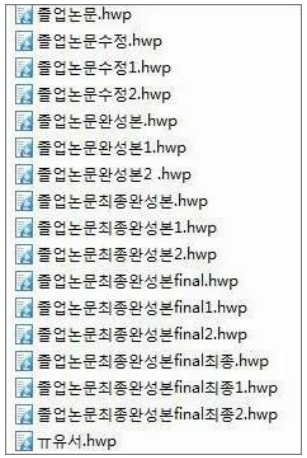
- 문제점
  - 각 버전간 차이를 알 수 없다.



1달 후 → 버전2 내용이 뭐였지??

  - 버전이 증가할 수록 새롭게 파일이 생성된다.

    → 관리가 어렵다. HDD 용량을 쓸데 없이 차지한다.



수많은 파일들...
그러나 내용은 거의 다 중복!!

엄청난 비효율 -> 개선하려면 뭔가 시스템이 필요함
-> 버전 관리 시스템

버전 관리 시스템의 종류

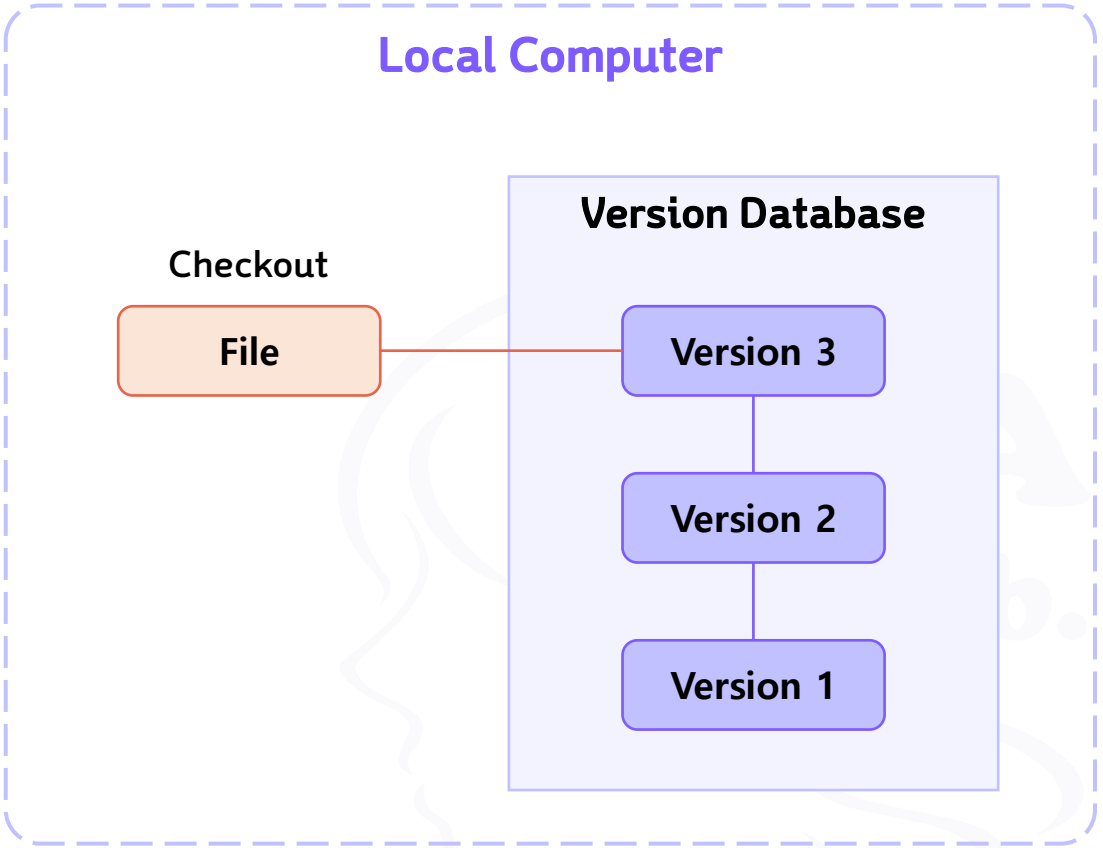
- 버전 관리 시스템의 사용 방식
  - 버전 관리 프로그램만 따로 사용해 관리하는 경우
  - 통합 개발 환경(Visual Studio, IntelliJ IDEA 등)과 텍스트 에디터(Visual Studio Code 등) 등에서 제공하는 소프트웨어와 통합된 버전 관리 기능을 사용하는 경우
  - 클라우드 컴퓨팅(Google Docs, MS OneDrive 등)에서 제공하는 버전 관리 기능을 사용하는 경우

- 버전 관리 시스템의 분류
  - 관리 형태에 따라 크게 로컬 VCS, 중앙집중식 VCS, 분산 VCS 등으로 분류함

버전 관리 시스템의 종류

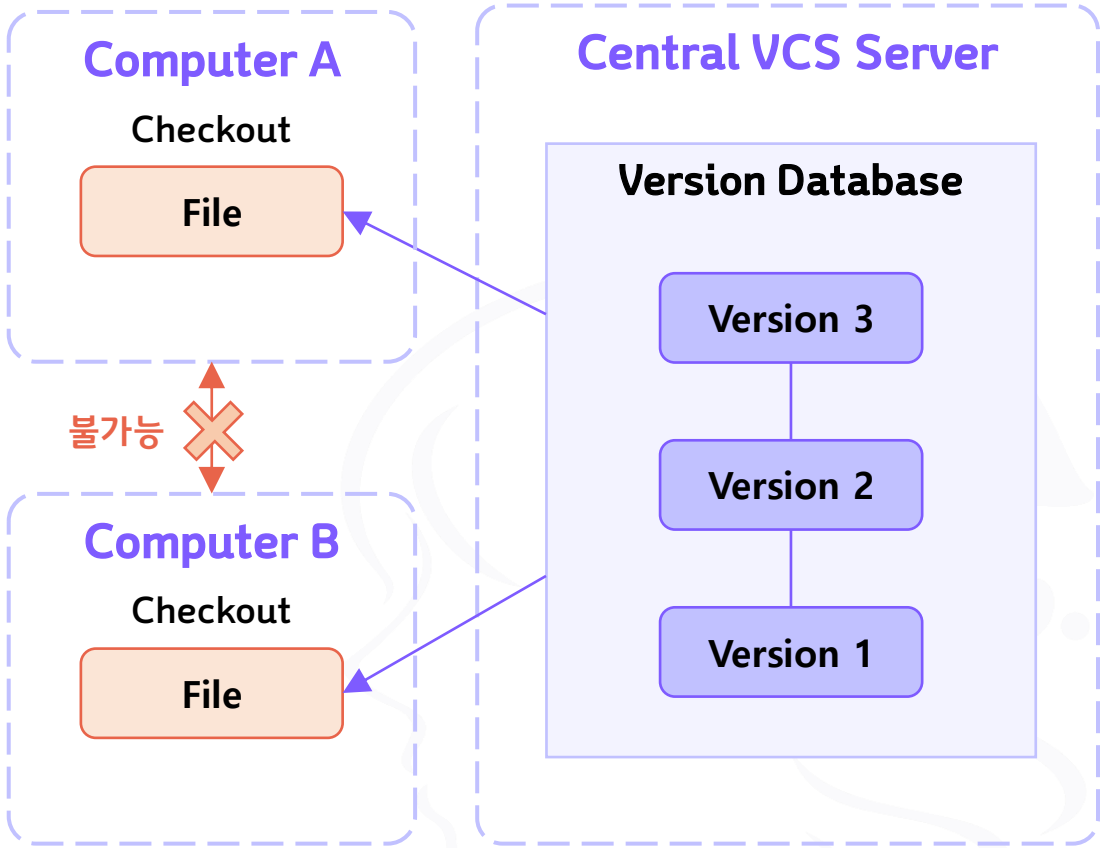
- 로컬 VCS
  - 처리방식
    - 서버 없이 로컬 컴퓨터 내에서 버전을 관리함
  - 장점
    - 간단한 데이터베이스만으로도 구현 가능

      → 단순하고 개인적인 프로젝트에 적합
  - 단점
    - 협업 시 사용하기는 어려움
    - 컴퓨터가 고장나는 등 내부 정보가 소실되는 경우 복구할 방법이 없음



버전 관리 시스템의 종류

- 중앙집중식 VCS (Central VCS, CVCS)
  - 처리방식
    - 서버에 최종본 한 벌이 저장되어 있음
    - 수정을 원하는 파일만 로컬 시스템에 다운로드 한 후, 수정한 파일을 다시 서버에 업로드하는 방식
  - 대표적인 CVCS
    - CVS
    - Subversion(SVN)
    - Perforce 등

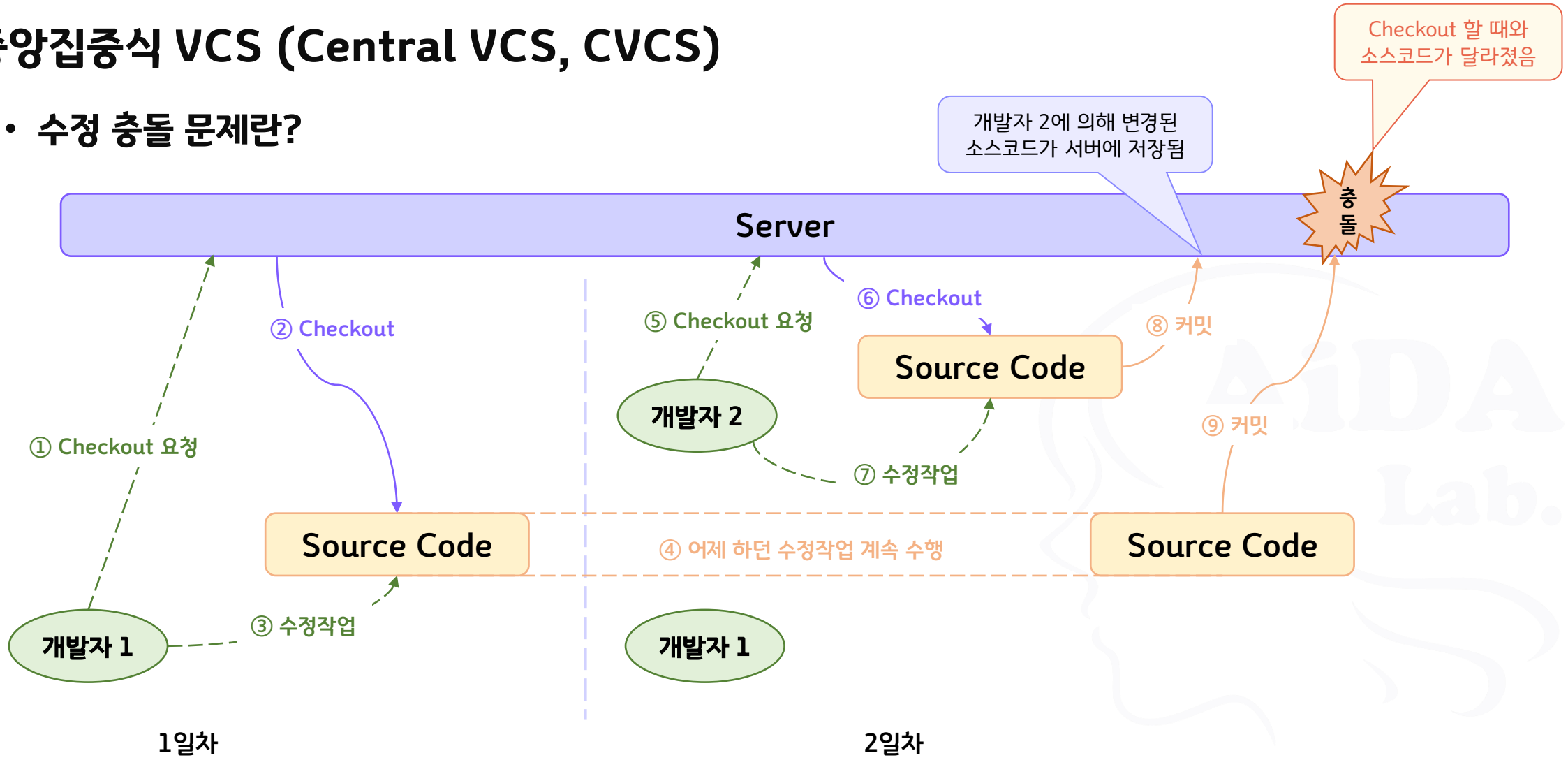


버전 관리 시스템의 종류

- 중앙집중식 VCS (Central VCS, CVCS)
  - 장점
    - 간단한 방법으로 협업이 가능
    - 누가 어떤 일을 하고 있는지 관리자가 파악하기 쉬움
    - 관리가 쉬움 (전체 클라이언트의 로컬 데이터베이스를 관리하는 것보다 CVCS 하나를 관리하기가 훨씬 쉬움)
  - 단점
    - 중앙 서버가 다운될 경우 복원될 때까지 업무가 마비됨 (업무 백업도 불가능 함)
    - 서버의 정보가 소실될 경우 모든 히스토리가 함께 소실됨 (단 개인이 로컬에 저장해 둔 스냅샷은 유지됨)
    - 협업의 규모가 커지면 수정 충돌 문제 등이 발생할 수 있음

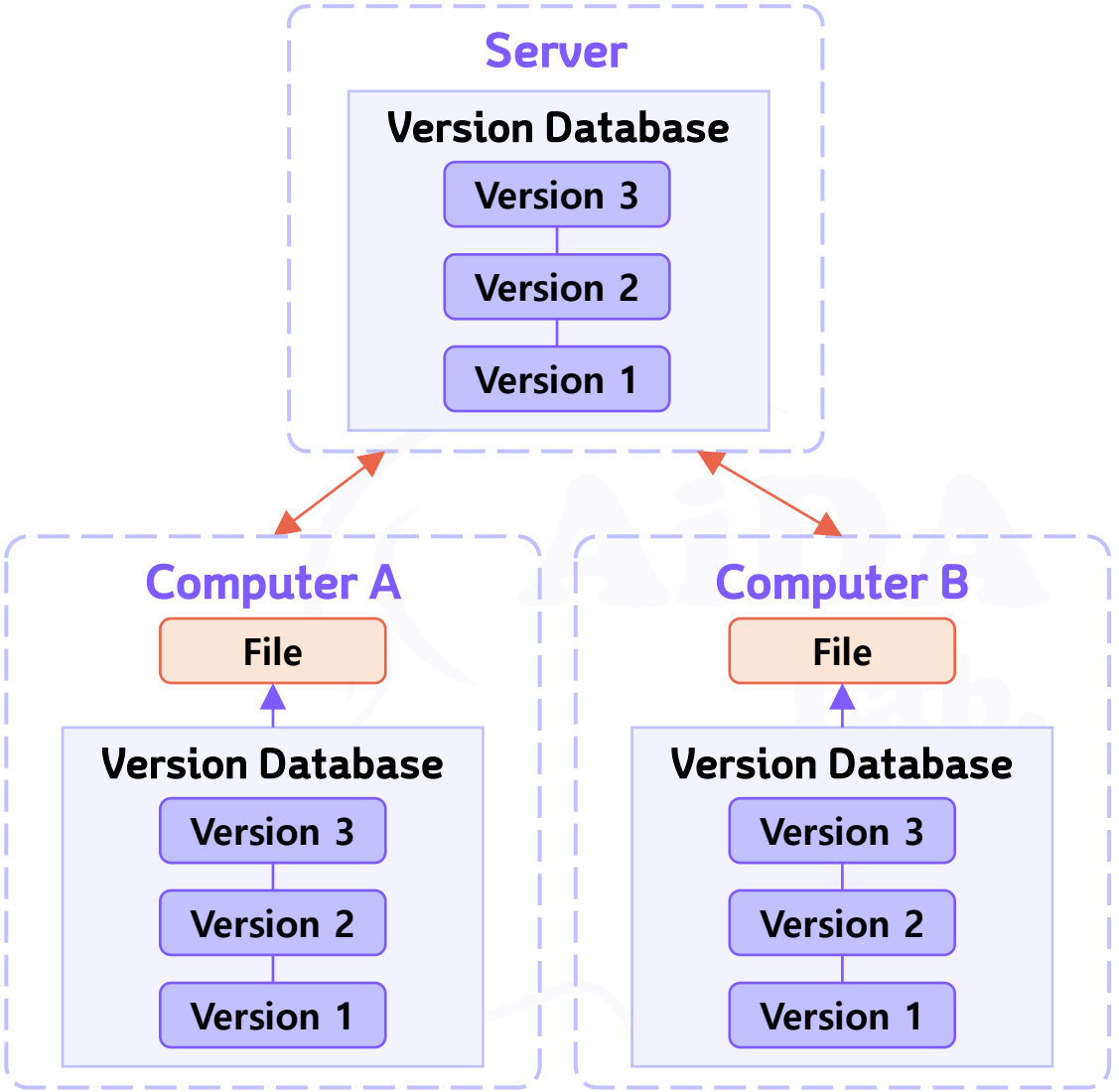
버전 관리 시스템의 종류

- **중앙집중식 VCS (Central VCS, CVCS)**
  - **수정 충돌 문제란?**



버전 관리 시스템의 종류

- 분산 VCS(Distributed VCS, DVCS)
  - 처리방식
    - 파일을 저장하는 서버가 있는 것은 CVCS와 동일
    - 프로젝트 전체를 로컬에 다운 받은 뒤(복제) 수정
  - 대표적인 DVCS
    - git
    - mecurial
    - Bazaar
    - Darcs 등



버전 관리 시스템의 종류

- 분산 VCS(Distributed VCS, DVCS)
  - 장점
    - 저장소의 프로젝트 내용을 히스토리를 포함하여 모두 복제(Clone)해서 사용하므로(분산 저장)
      서버에 문제가 발생했을 경우 복제물을 이용하여 복원 가능(작업도 언제든지 가능)
    - 수정 작업은 복제된 로컬 시스템에서 수행하므로 충돌의 우려없이 작업 가능
      - 최종본을 서버에 업로드 할 때만 신경 써서 통합(Merge)해 주면 됨
    - 원격 저장소를 이용하므로 동시에 다양한 그룹과 다양한 방법으로 협업 가능
    - 중앙집중식 시스템으로는 할 수 없는 워크플로를 다양하게 사용할 수 있음
  - 단점
    - 다양한 기능으로 인하여 중앙집중식 VCS(대표적으로 SVN)보다 복잡하고 배우기가 다소 어려움

Git/GitHub 소개 및 환경 구축

- Git
  - Linux의 창시자인 리누스 토르발스(Linus Benedict Torvalds)가 오픈소스로 개발한 분산형 버전 관리 시스템(Version Control System, VCS)
  - 매우 빠른 속도와 분산형 저장소의 지원이 특징
  - 오픈 소스 개발의 특성상 많은 사람이 자유롭게 소스코드를 수정할 수 있으므로 잘못된 수정으로 인하여 프로젝트에 큰 문제가 발생하기 쉬우며 Git은 이런 환경적인 특성에 잘 대응할 수 있도록 개발됨
  - 공식 사이트 URL: https://git-scm.com

scm: Source Control Management tool

  - 기본 저장소: git.kernel.org
    - GitHub에 git/git 이라는 미러 저장소를 운영 중

GitHub

- **GitHub란?**
  - **2008년 공개된 대표적인 무료 Git 저장소**
  - **2018년 MS에 인수됨**
    - **원래 공개 프로젝트만 무료였고 비공개 프로젝트는 유료 결제를 해야 했으나 MS에 인수된 후**
    - **2019년 1월부터 비공개 저장소도 무료화, 2020년 4월부터 비공개 저장소의 공동 작업자 수 제한도 해제됨**
  - **유사 서비스**
    - **Bitbucket: Jira, Confluence로 유명한 Atlassian에서 개발, 운영함에 따라 높은 신뢰도를 가짐**
    - **GitLab: Bitbucket보다 강력한 기능 제공, 적은 제한 등의 강점을 가지나 서버 운영이 다소 불안정함**
    - **Codeberg: 독일의 비영리 재단에서 운영**
    - **기타 등등**

SourceForge
한때 많이 사용되었던 오픈소스 프로그램을 제공하는 사이트.

GitHub와 경쟁했으나 수익창출을 위한 과도한 광고 유치와
애드웨어, 해킹, 바이러스 등 갖은 악재로 인해 사실상 망함

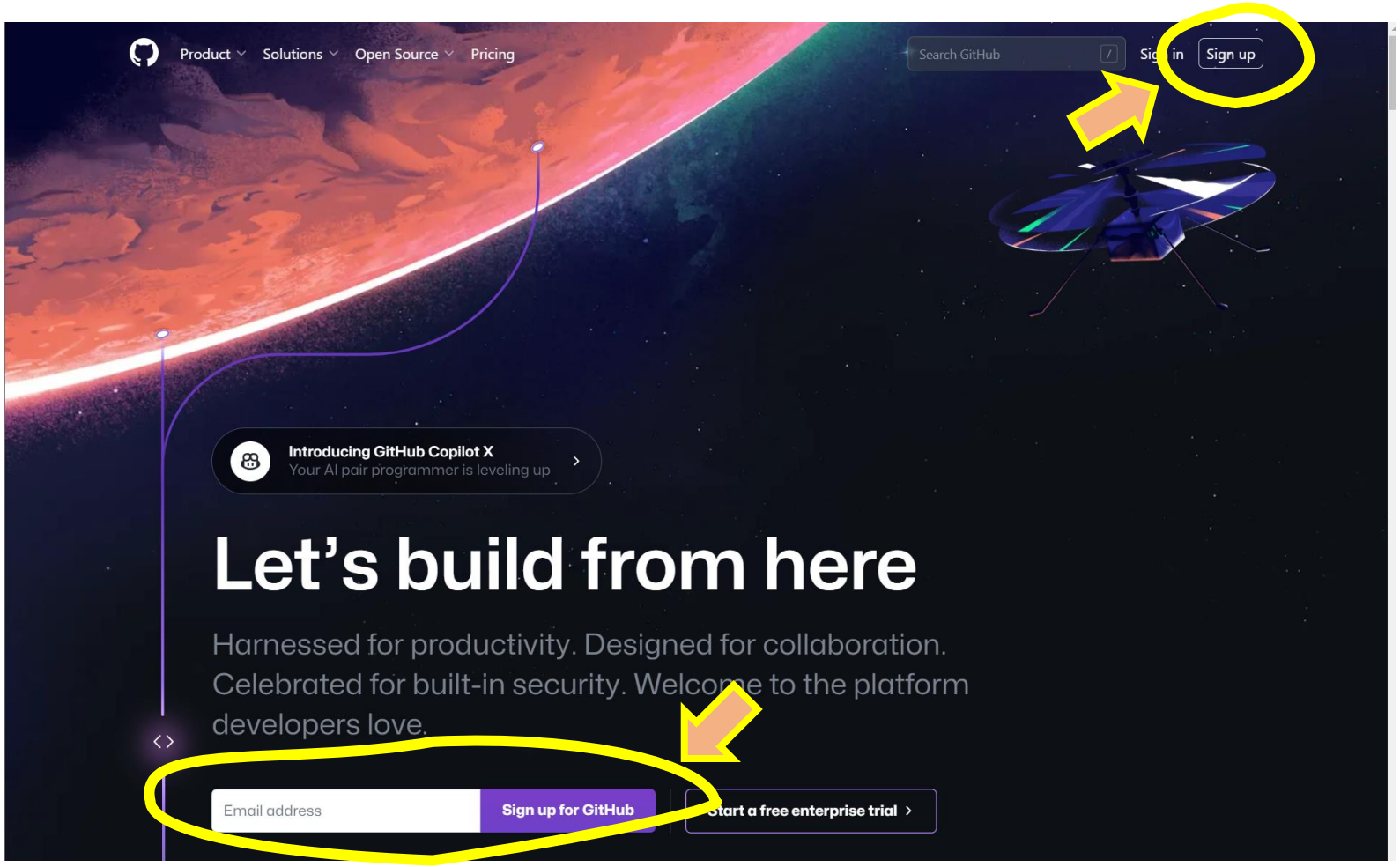
- **GitHub의 서비스**
  - **GitHub Repository(저장소)**
    - 프로그램 소스코드를 저장하게 하고 개발자가 사용하는 GitHub의 대표적인 서비스
  - **GitHub Actions**
    - 지속적 통합 (continuous integration, CI) 그리고 지속적 배포 (Continuous delivery, CD) 를 사용해서 빌드, 테스트, 배포를 가능하게 하는 데브옵스 (DevOps) 파이프라인 자동화 툴
    - GitHub Actions를 사용하여 저장소에서 바로 소프트웨어 개발 워크플로 자동화, 사용자 지정 및 실행이 가능함
  - **GitHub Packages**
    - 소프트웨어 패키지 호스팅 서비스
    - 도커 컨테이너 혹은 프로그램 패키지 (NPM, NuGet, Gem, Maven/Gradle) 를 저장할 수 있는 이미지 레지스트리 개념의 서비스

- **GitHub의 서비스**
  - **GitHub Advanced Security (GHAS)**
    - 코드 패키지에 있는 취약점(프로젝트에서 사용되는 종속성에 대한 취약성)을 찾아주는 Dependabot, 그리고 소스코드에 있는 문제점들을 찾는 것 등 다양한 보안 기능을 제공하는 서비스
  - **GitHub Projects**
    - Atlassian JIRA 같이 프로젝트를 관리하게 도와줄 수 있는 서비스
    - 테이블(스프레드시트) 또는 보드의 형태로 제공됨
  - **GitHub Discussions**
    - 오픈 소스 또는 내부 프로젝트를 중심으로 커뮤니티를 위한 공동 커뮤니케이션 포럼
    - Stackoverflow 나 Quora 같이 질문을 물어보고 대답하는 형식을 제공하는 서비스

- **GitHub의 서비스**
  - **GitHub Pages**
    - 저장소에 간단한 마크다운(Markdown) 형식 파일을 저장하고 설정하면 간단한 웹사이트를 만들어 주는 서비스 (일종의 웹 호스팅 서비스)
    - Jekyll과 같은 웹 사이트 빌드 도구를 사용할 수 있음
  - **GitHub CoPilot**
    - AI, 머신러닝 기술을 사용해서 개발자가 작성하는 코드의 내용을 파악하고 나머지 코드를 대신 적어주는 서비스
  - **GitHub Codespace**
    - 저장소 안에서 별도의 다운로드나 설치 없이 웹 브라우저에서 런칭하여 코드를 개발할 수 있게 해주는 내장 IDE
    - 원하는 방법으로 원하는 곳에서 작동하는 안전하고 구성 가능한 전용 개발 환경에서 개발을 시작할 수 있음

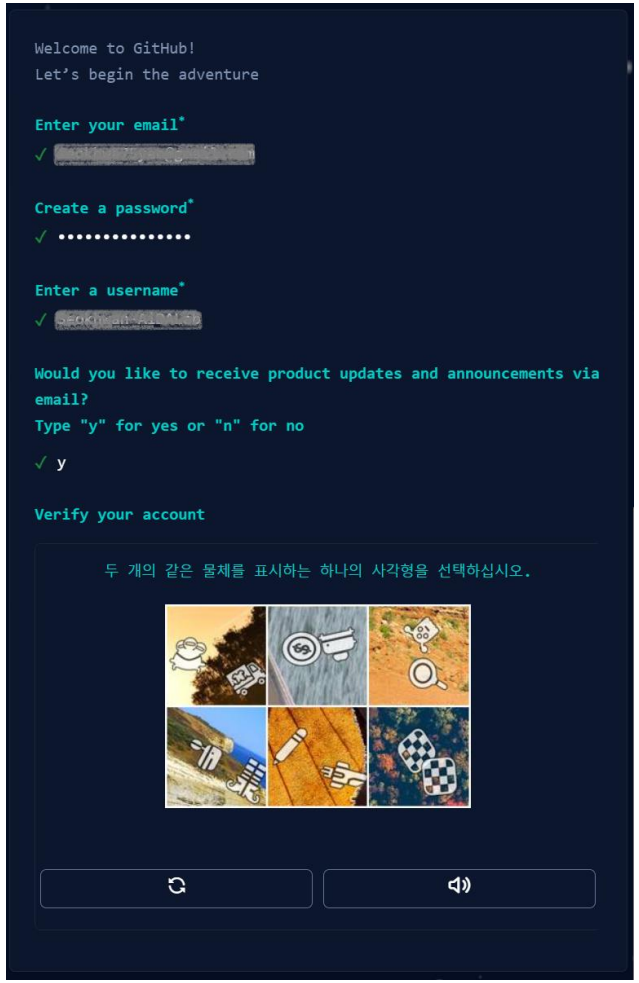
Git 환경 구축하기

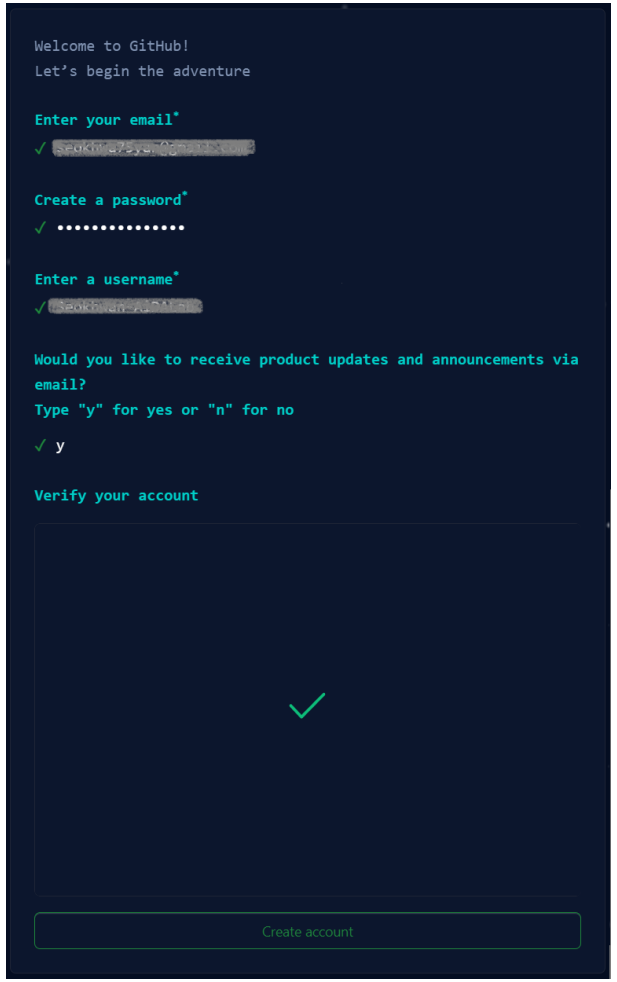
- **Step 1: GitHub 가입하기**
  - https://github.com
  - Email을 이용하여 가입

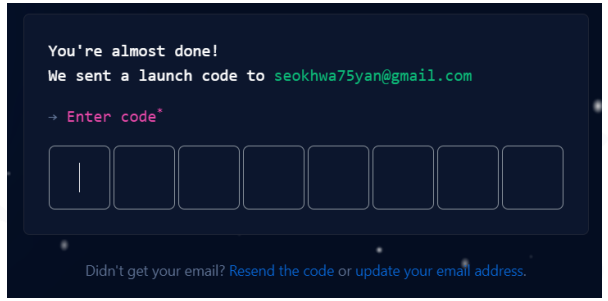


Git 환경 구축하기

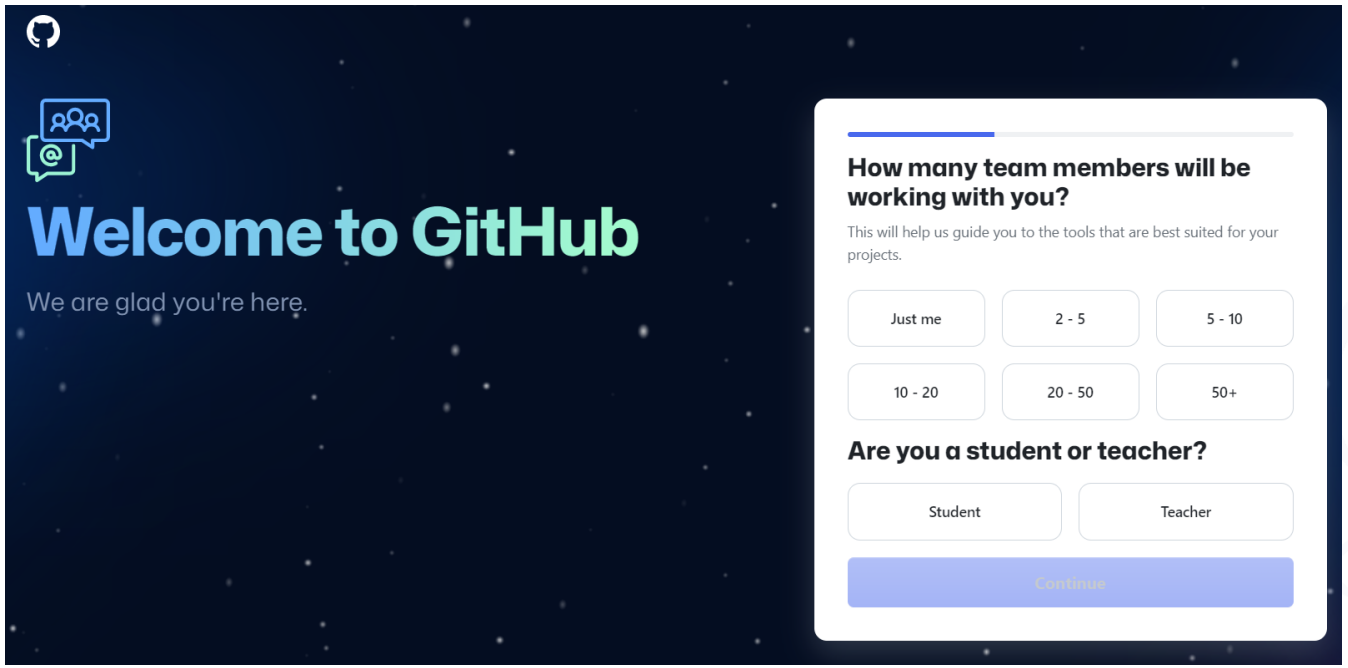
- **Step 1: GitHub 가입하기**

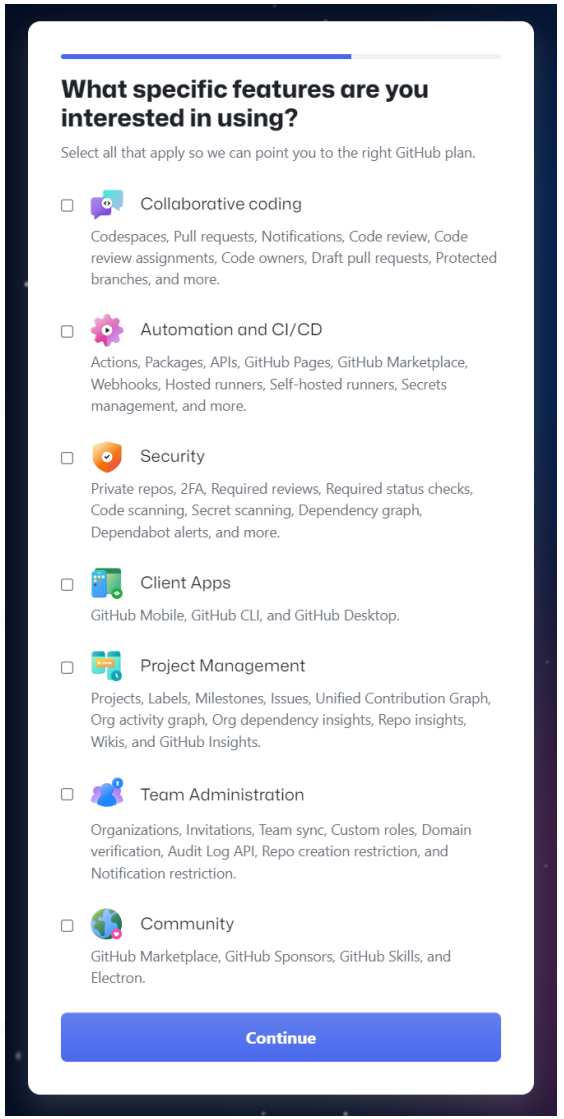


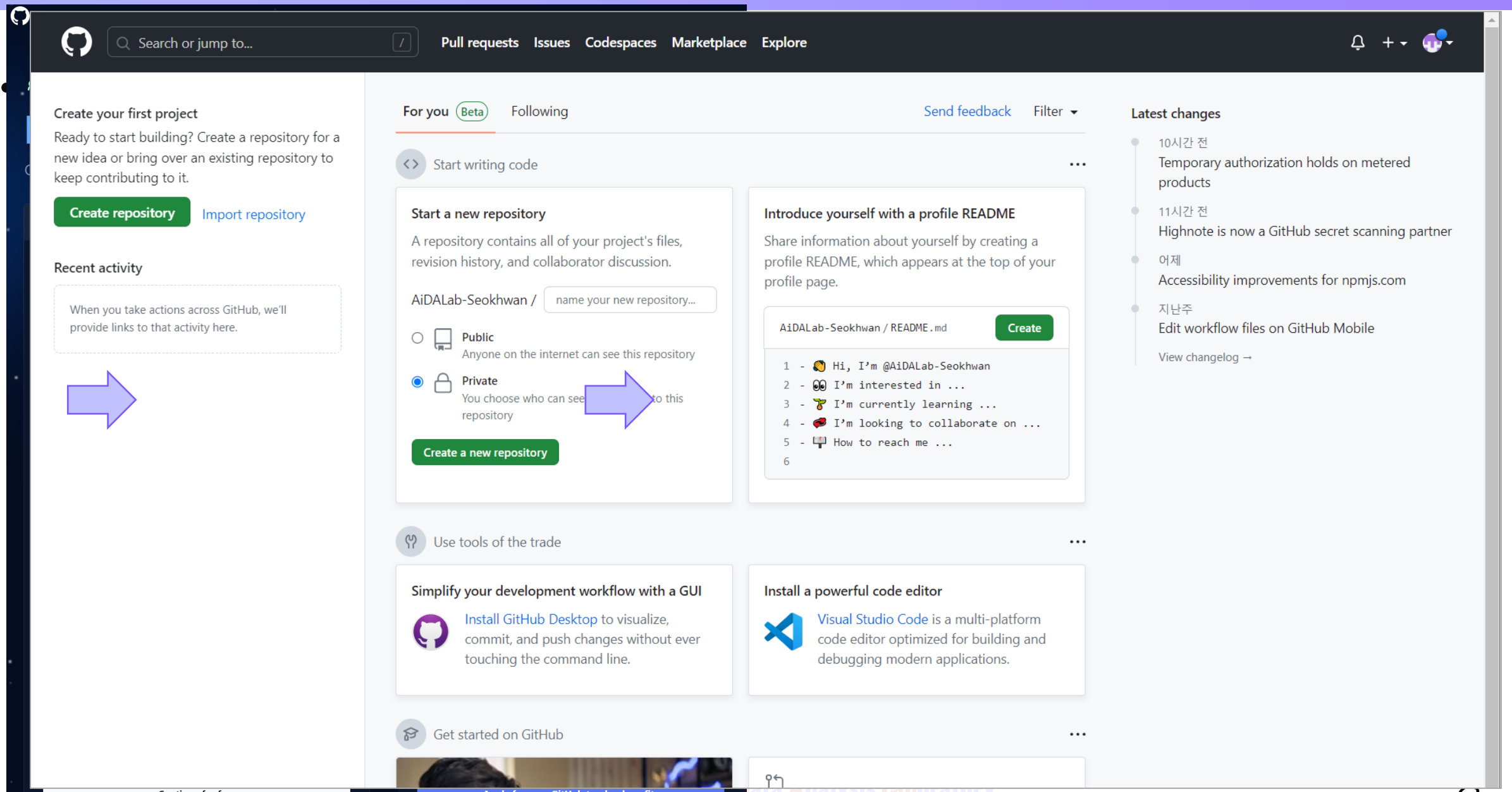




- **Step 1: GitHub 가입하기**

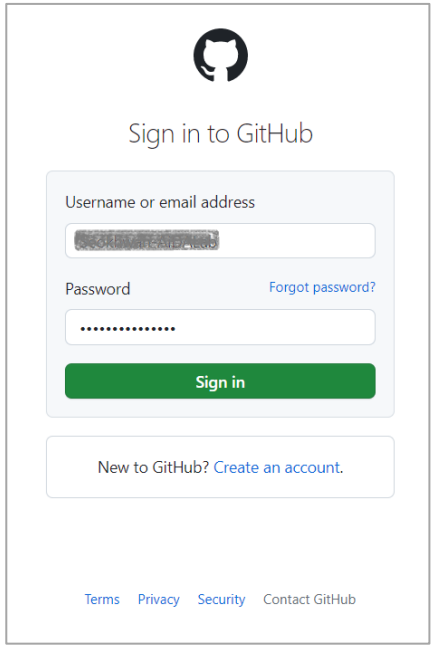




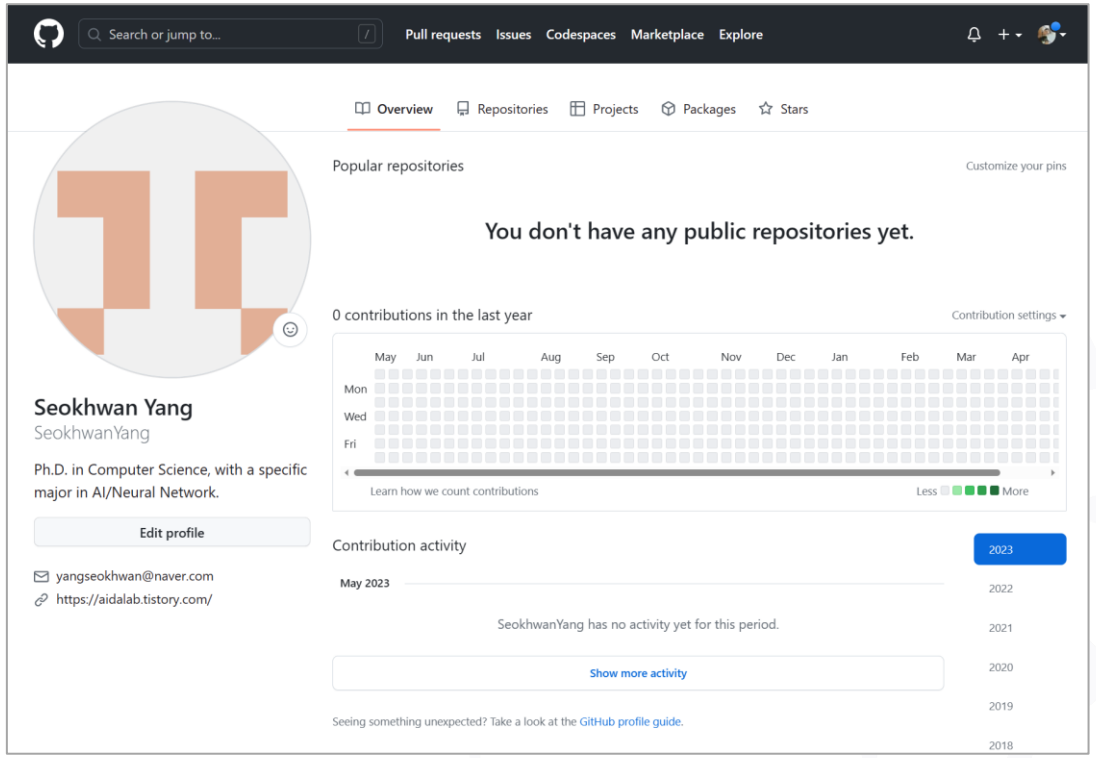

Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
Create your first project
Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.
Create repository
Import repository
Recent activity
When you take actions across GitHub, we'll provide links to that activity here.
For you
Beta
Following
Send feedback
Filter
Start writing code
Start a new repository
A repository contains all of your project's files, revision history, and collaborator discussion.
AiDALab-Seokhwan /
name your new repository...
Public
Anyone on the internet can see this repository
Private
You choose who can see to this repository
Create a new repository
Introduce yourself with a profile README
Share information about yourself by creating a profile README, which appears at the top of your profile page.
AiDALab-Seokhwan / README.md
Create
1 - Hi, I'm @AiDALab-Seokhwan
2 - I'm interested in ...
3 - I'm currently learning ...
4 - I'm looking to collaborate on ...
5 - How to reach me ...
6
Use tools of the trade
Simplify your development workflow with a GUI
Install GitHub Desktop to visualize, commit, and push changes without ever touching the command line.
Install a powerful code editor
Visual Studio Code is a multi-platform code editor optimized for building and debugging modern applications.
Get started on GitHub
Latest changes
10시간 전
Temporary authorization holds on metered products
11시간 전
Highnote is now a GitHub secret scanning partner
어제
Accessibility improvements for npmjs.com
지난주
Edit workflow files on GitHub Mobile
View changelog →
Continue for free
Apply for your GitHub teacher benefits

Git 환경 구축하기

- **Step 1: GitHub 가입하기**
  - **일반적으로 로그인(Sign in) 할 때**



Sign In



Git 환경 구축하기

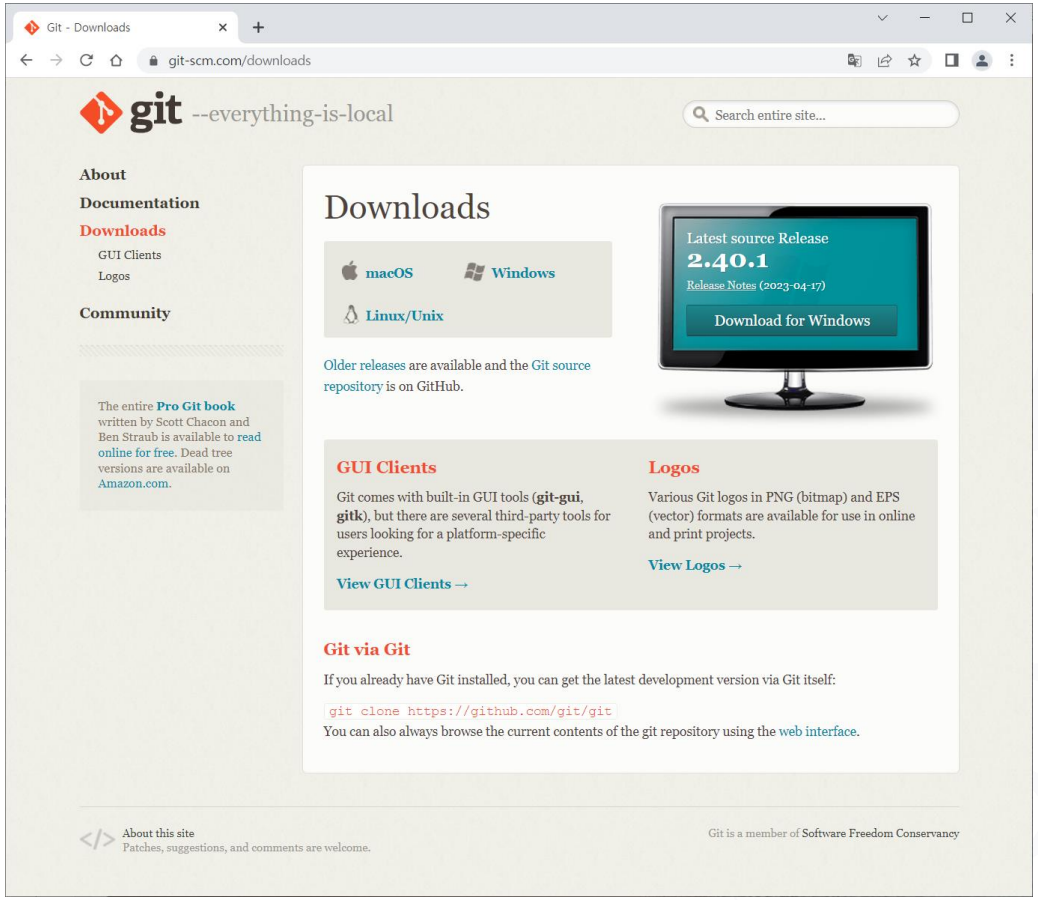
- **Step 2: Git 설치하기**
  - **GitHub의 사용 방법**
    - **웹 브라우저에서 직접 사용하기**
      - **가입 시에 입력한 UserName을 이용하여 GitHub 개인 페이지에 접속**
        - https://github.com/{UserName}
      - **개인페이지의 메뉴 및 인터페이스를 이용하여 사용**
    - **로컬 시스템에 Git을 설치하여 사용하기**
      - **Git을 설치한 후 직접 명령어(Command)를 입력하여 사용**
      - **Git을 설치한 후 GUI 도구(SourceTree 등)를 설치, 활용하여 사용**

여기에서는 이 방법을 기준으로 함

Git 환경 구축하기

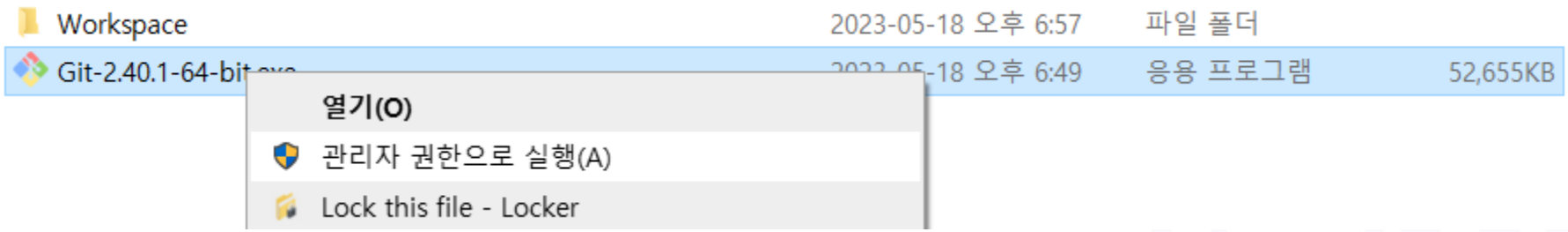
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**
    - Git 홈페이지에서 설치 파일 다운로드하기
      - https://git-scm.com/downloads

자신의 PC에 맞는 것을 다운로드하여 설치함
여기에서는 Windows를 기준으로 함



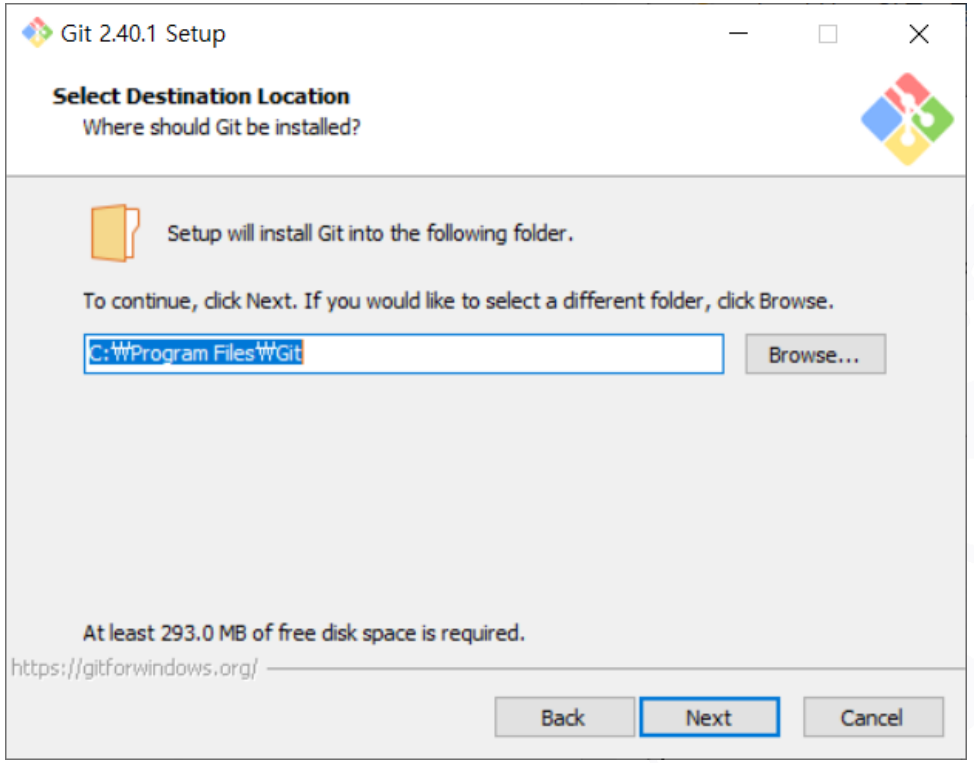
Git 환경 구축하기

- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**



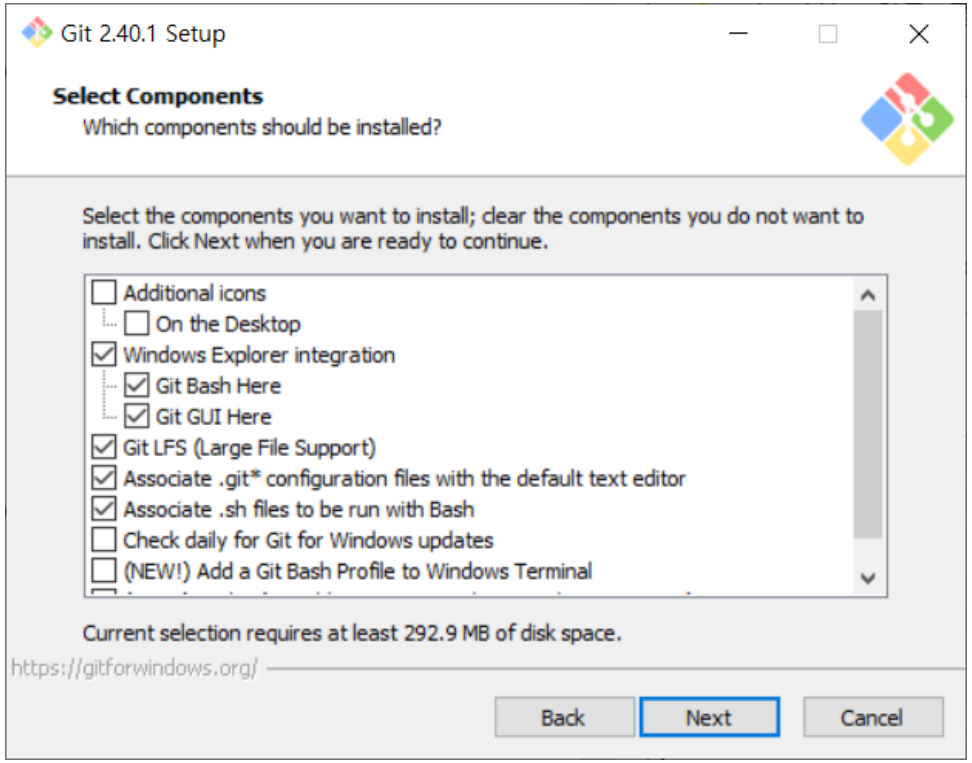
Git 환경 구축하기

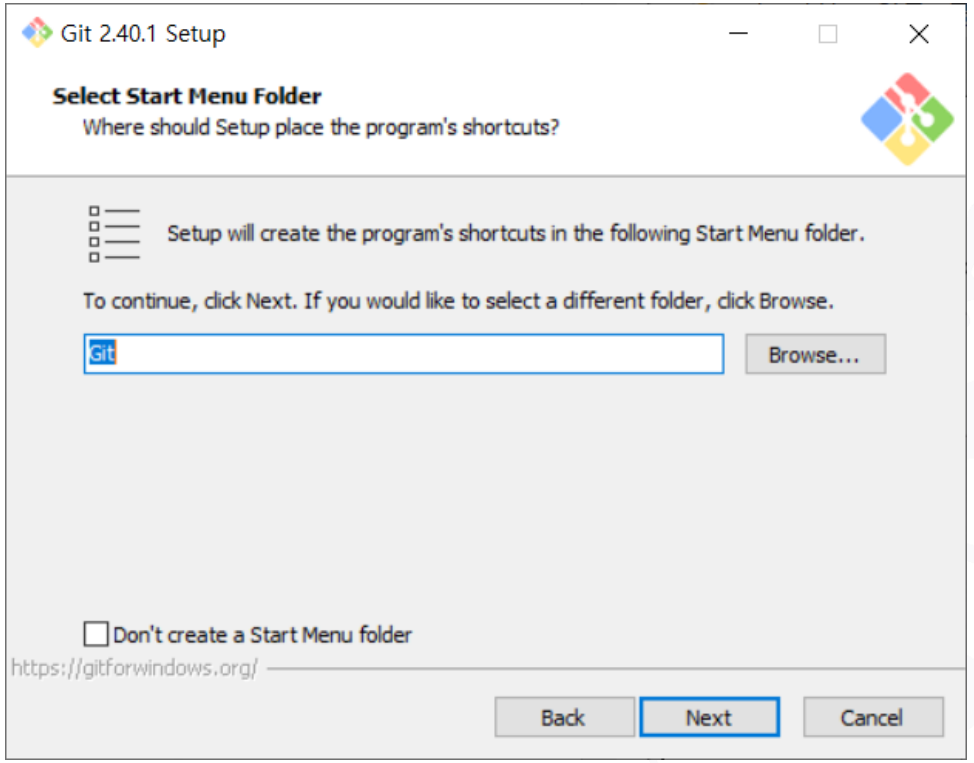
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**



Git 환경 구축하기

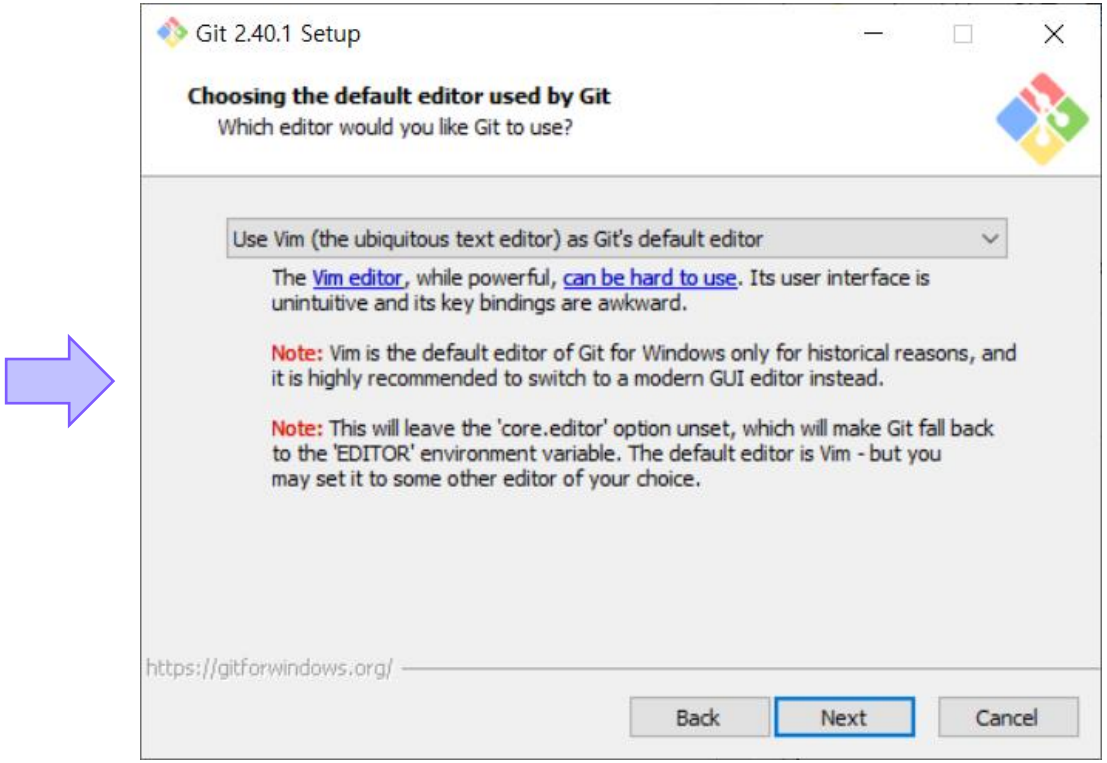
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**

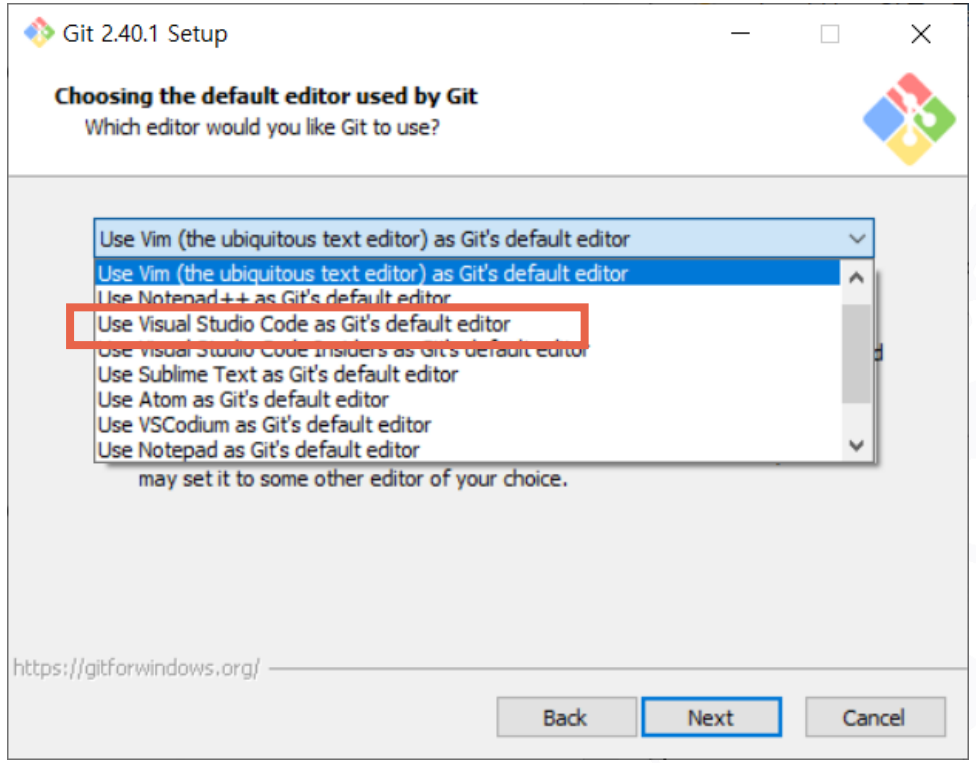




Git 환경 구축하기

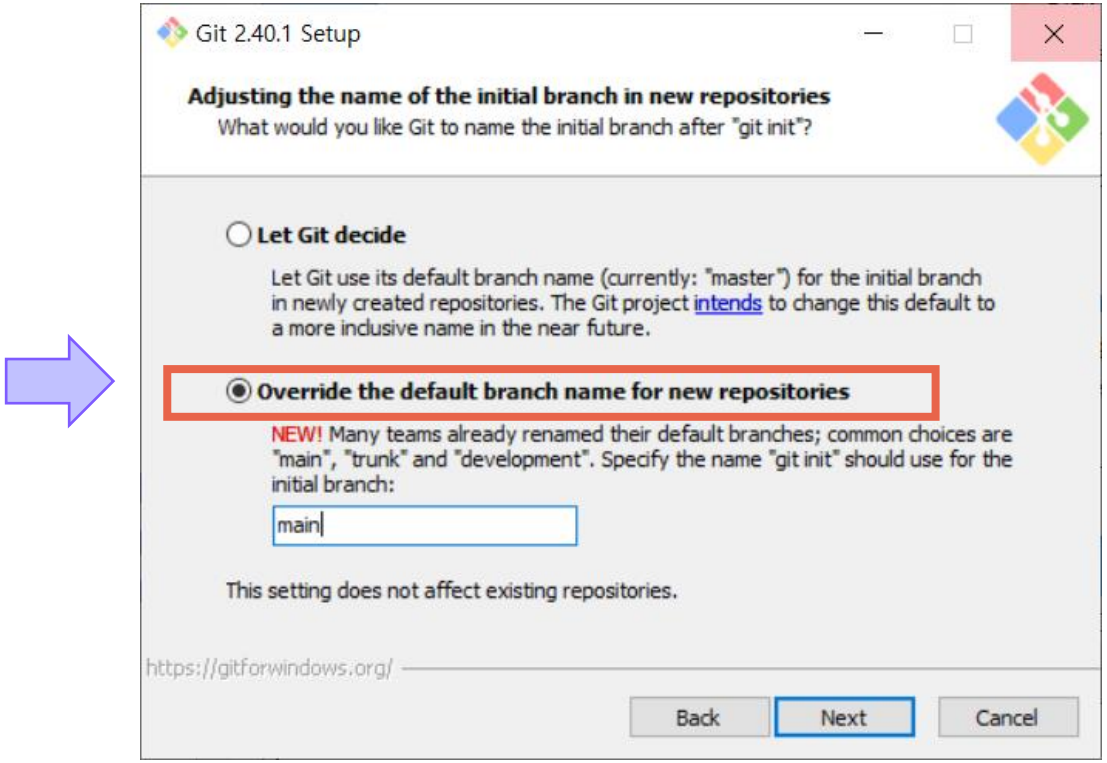
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**

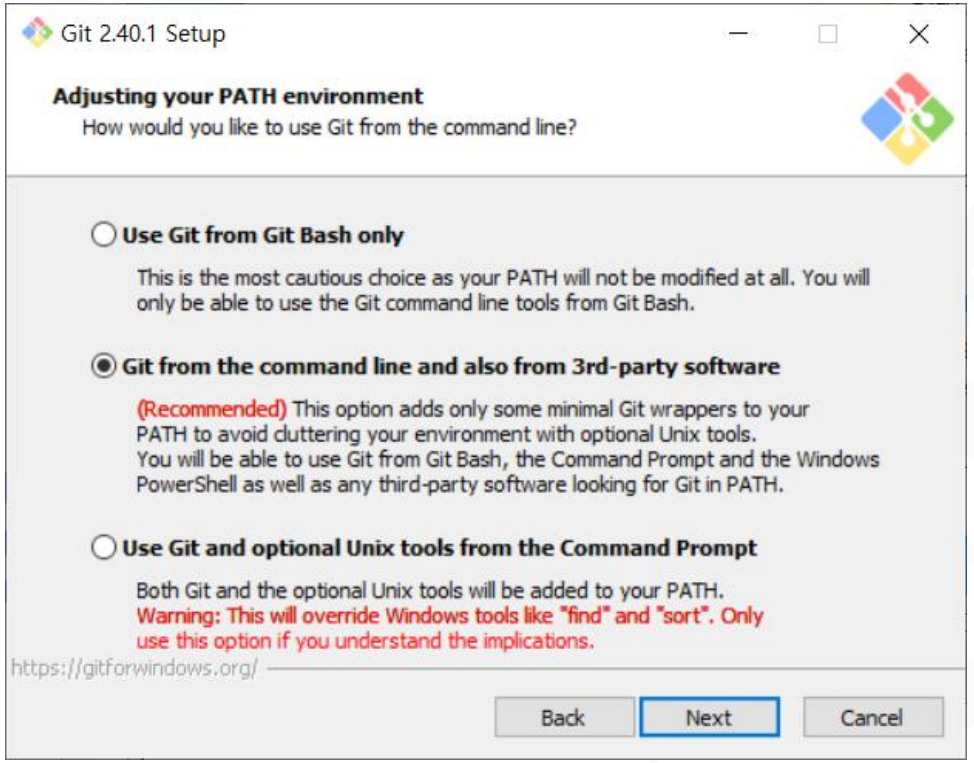




Git 환경 구축하기

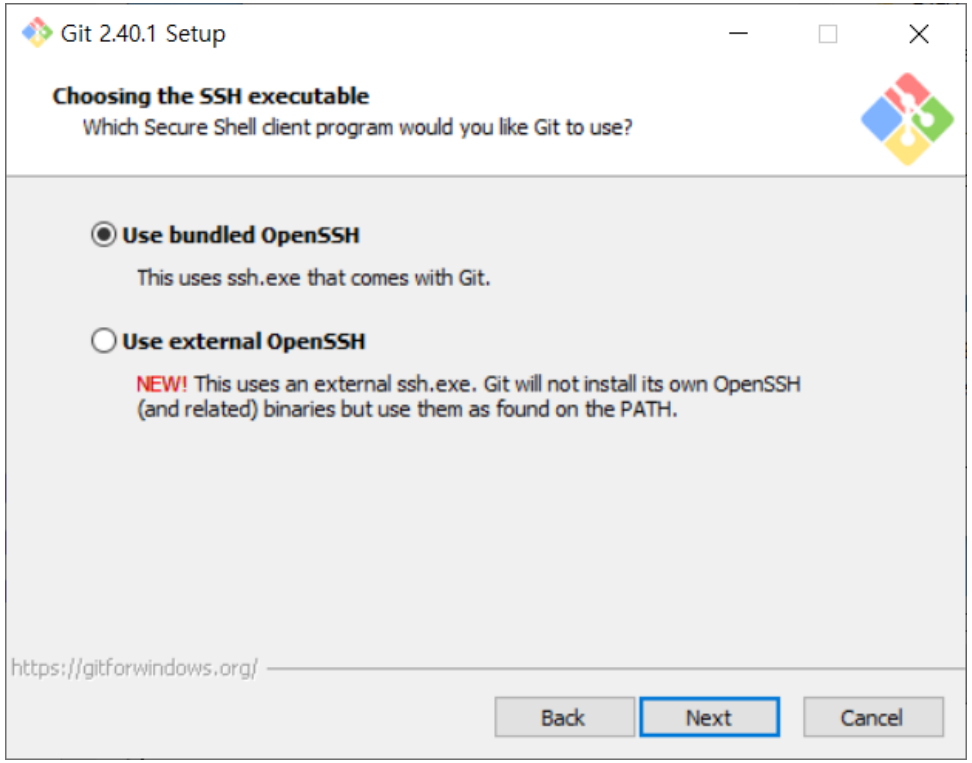
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**

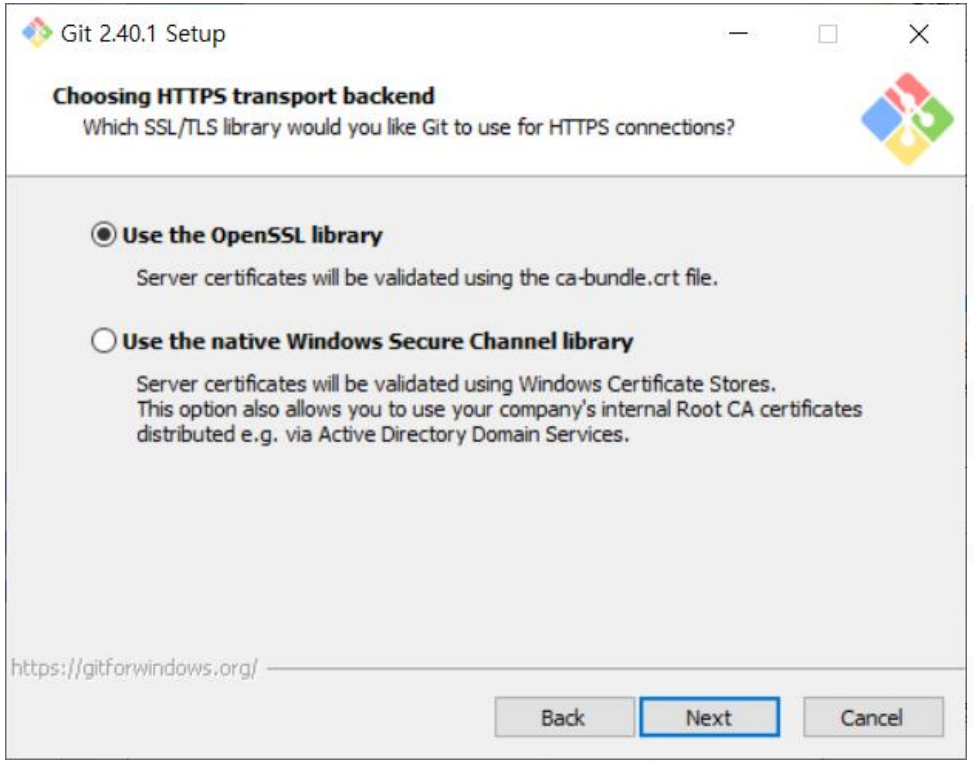




Git 환경 구축하기

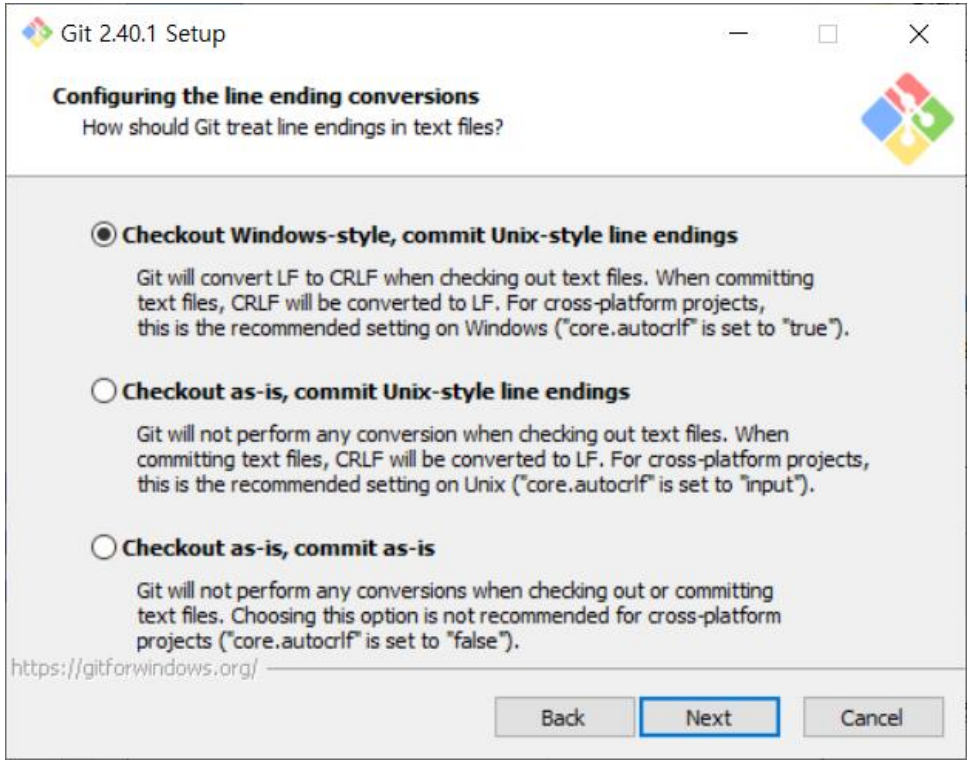
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**

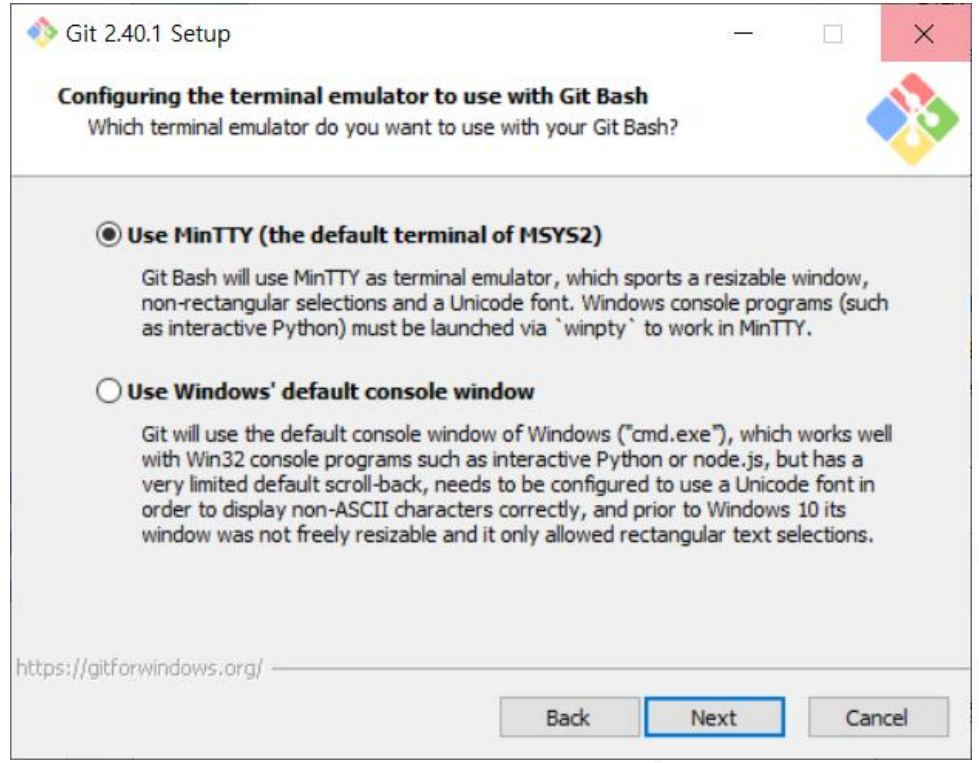




Git 환경 구축하기

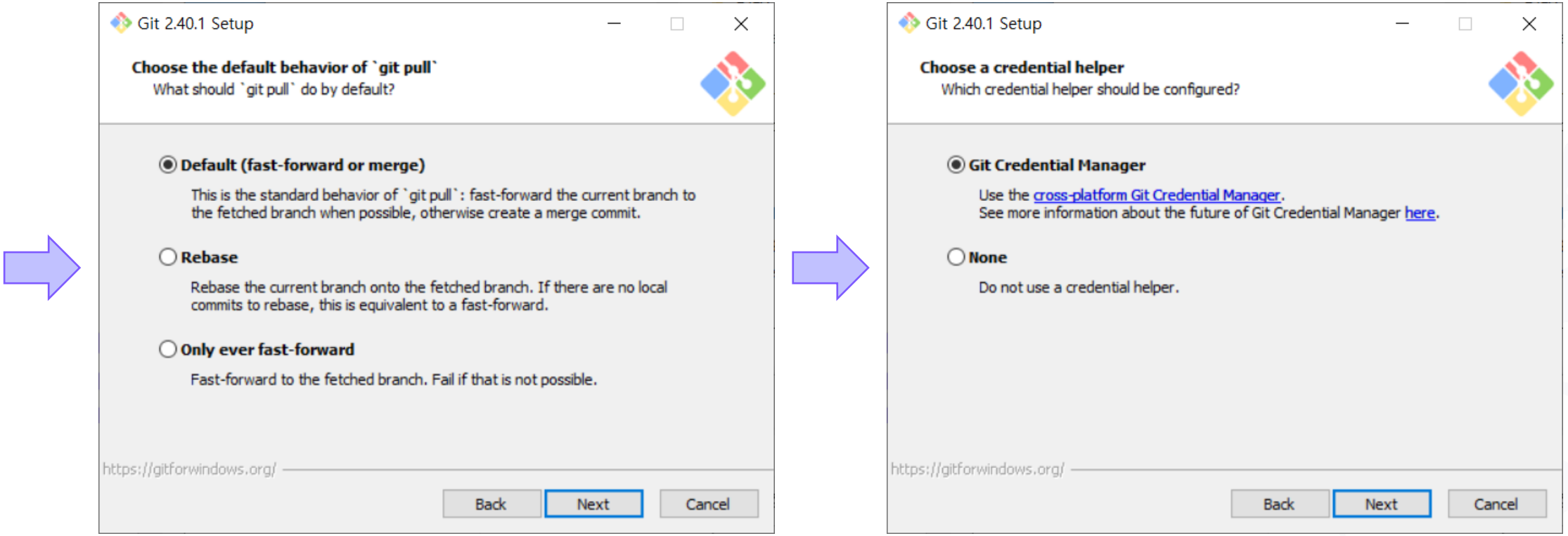
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**





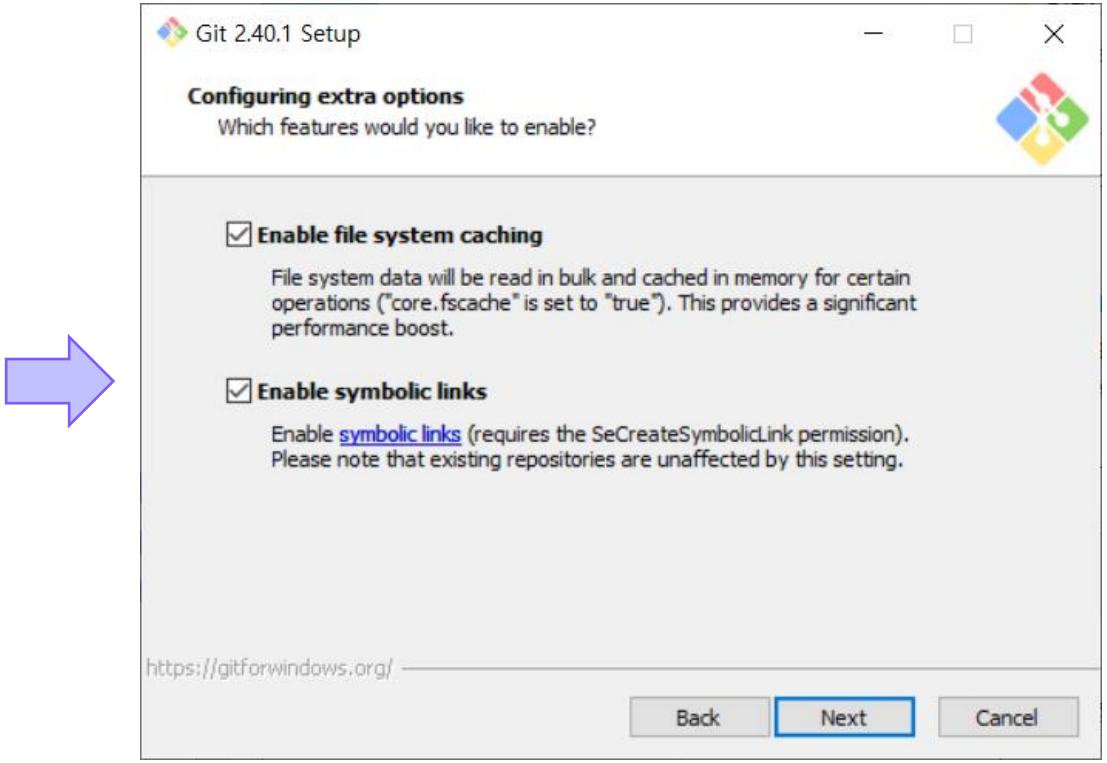
Git 환경 구축하기

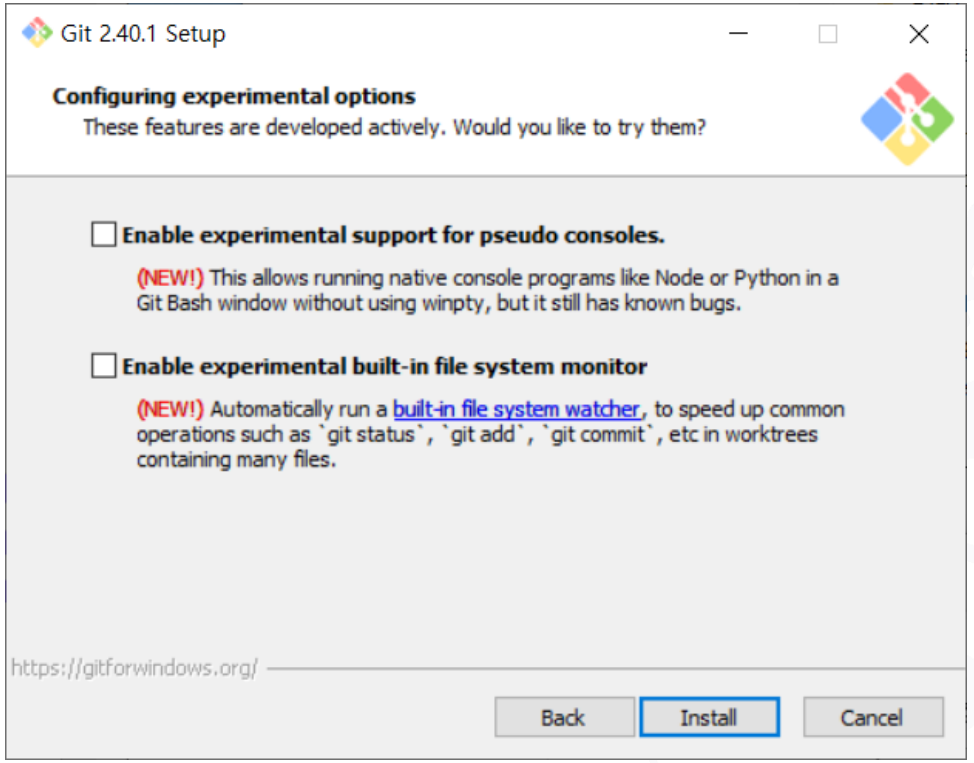
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**



Git 환경 구축하기

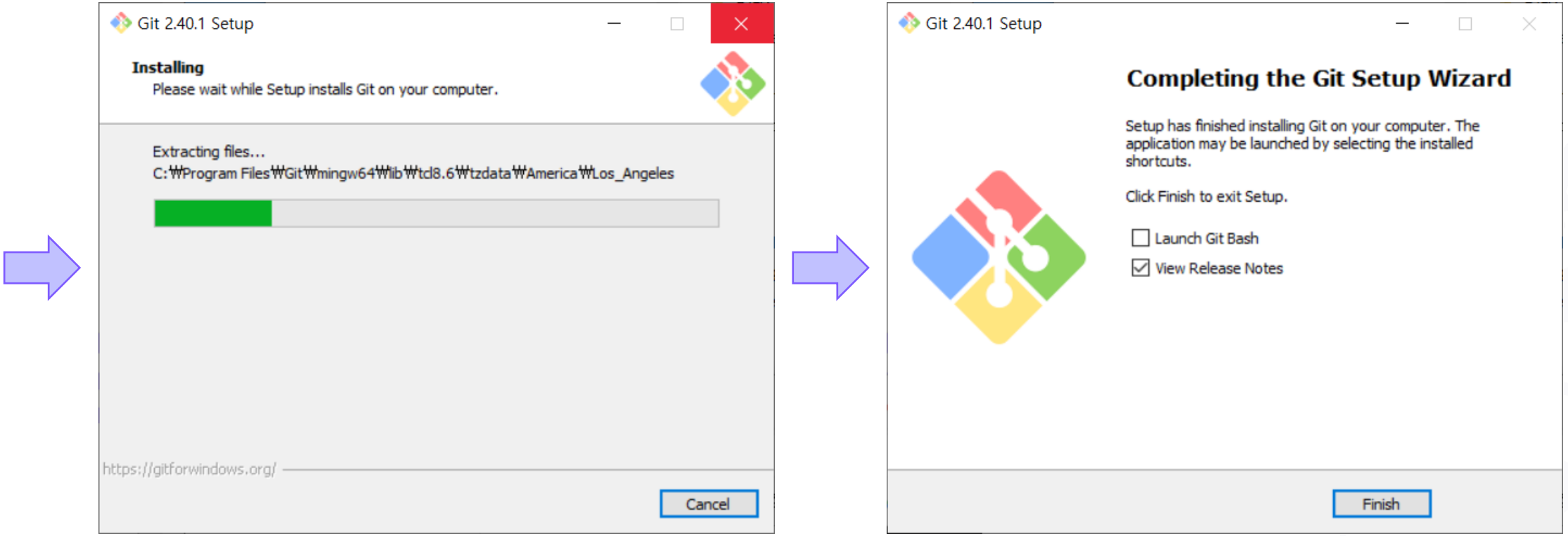
- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**





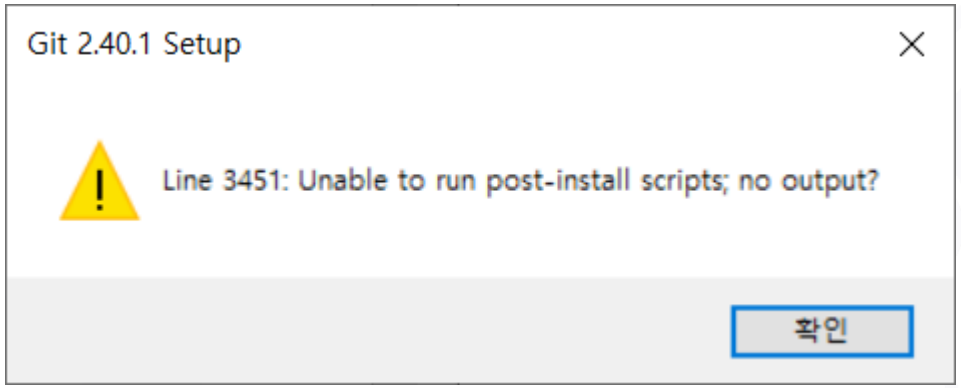
Git 환경 구축하기

- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**



Git 환경 구축하기

- **Step 2: Git 설치하기**
  - **Windows에 Git 설치하기**
    - 가끔 아래와 같은 오류가 발생하기도 함(Windows 버전에 한함) → 신경 쓰지 않아도 됨



Git 환경 구축하기

- **Step 2: Git 설치하기**
  - **Linux에 Git 설치하기**

```
~$ sudo apt update                                  최신 패키지 정보로 갱신
~$ sudo apt upgrade                                 최신 패키지로 업데이트
~$ sudo apt install git                             git 설치
~$ git --version                                    잘 설치되었는지 확인 + 버전 확인
~$ git config --global init.defaultBranch main      깃의 기본 브랜치 이름을 main으로 설정
```

Git 환경 구축하기

- **Step 2: Git 설치하기**
  - **Mac에 Git 설치하기**

```
~$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```
Homebrew 설치 (이미 설치된 경우는 건너뜀)

```
~$ brew --version
```
Homebrew가 정상적으로 설치 되었는지 확인

```
~$ brew install git
```
git 설치

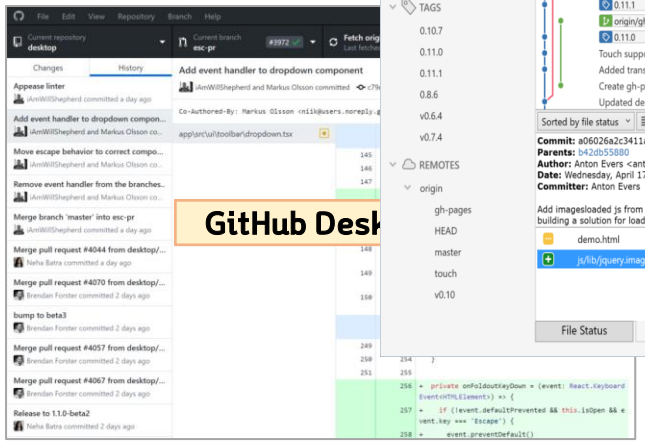
```
~$ git --version
```
잘 설치되었는지 확인 + 버전 확인

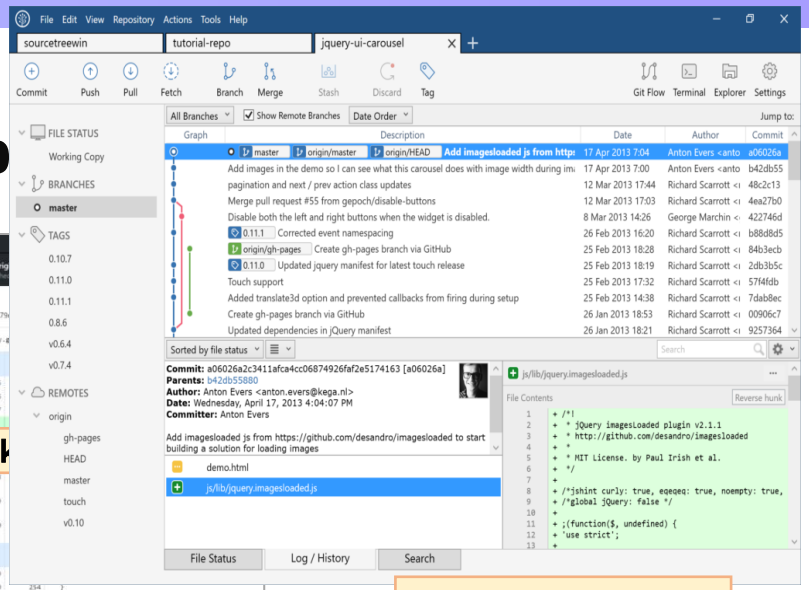
```
~$ git config --global init.defaultBranch main
```
깃의 기본 브랜치 이름을 main으로 설정

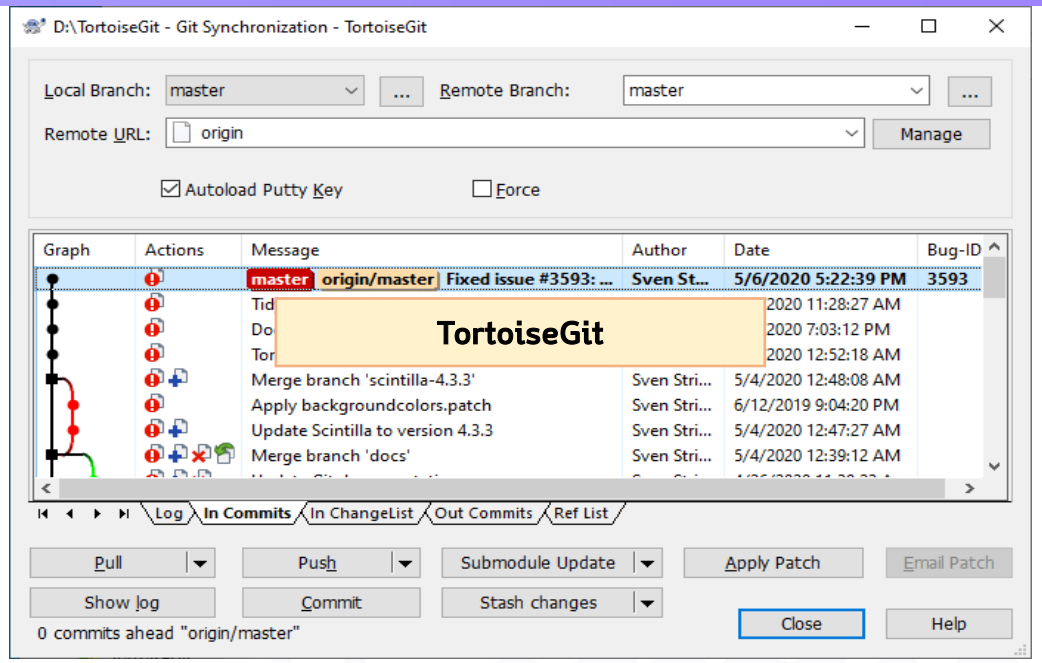
GUI 도구와 CLI 환경 비교

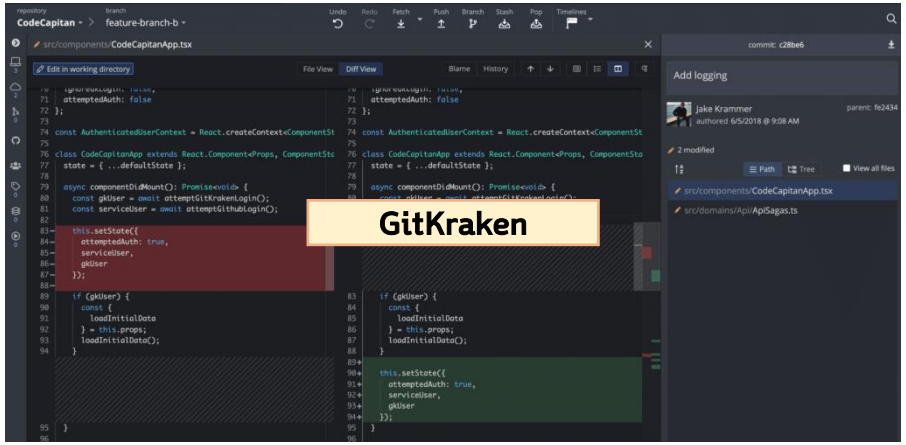
Git을 사용하기 위한 환경

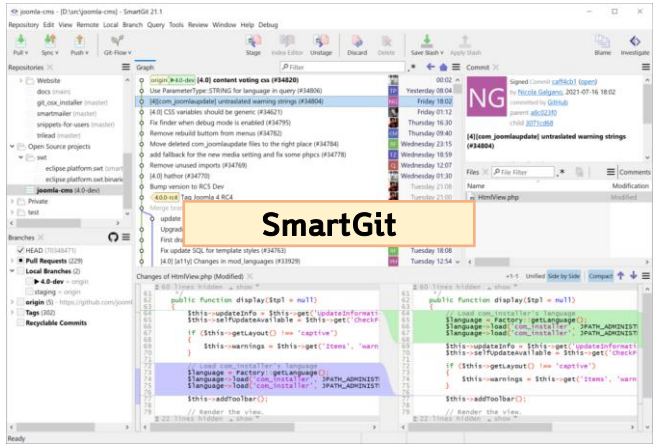
- GUI(Grap

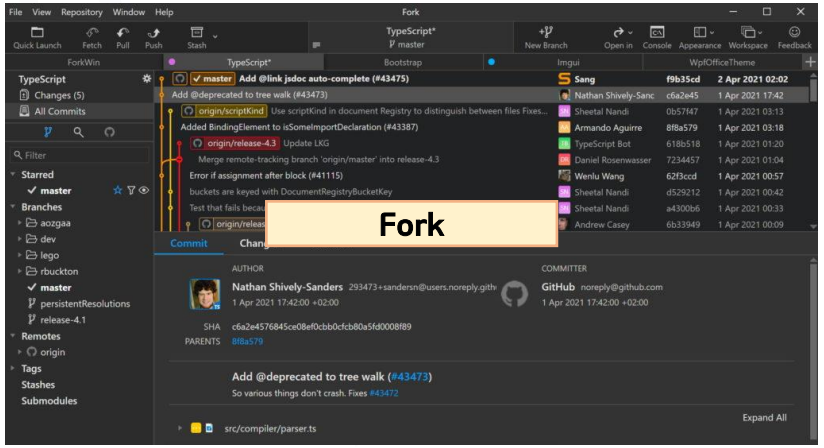












Git을 사용하기 위한 환경

- **CLI(Command Line Interface)**

```
➜  clone git:(master) ✗ git add .
➜  clone git:(master) ✗ git commit -m "theme Bootswatch Journal, fixed Account Drop-down"
[master dcb79b8] theme Bootswatch Journal, fixed Account Drop-down
 6 files changed, 5 insertions(+), 35 deletions(-)
 delete mode 100644 app/assets/javascripts/.tern-port
 delete mode 100644 app/assets/javascripts/static_pages.js
➜  clone git:(master) git push
To bitbucket.org:dapft_mazzini/thewterclone.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'git@bitbucket.org:dapft_mazzini/thewterclone.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
➜  clone git:(master) git push --help
➜  clone git:(master) git pull --rebase
First, rewinding head to replay your work on top of it...
Applying: theme Bootswatch Journal, fixed Account Drop-down
Using index info to reconstruct a base tree...
M       app/assets/javascripts/.tern-port
M       app/assets/javascripts/application.js
A       app/assets/javascripts/static_pages.js
M       app/assets/stylesheets/custom.css.scss
M       app/views/layouts/_header.html.erb
M       app/views/static_pages/_non_logged_in_home.html.erb
Falling back to patching base and 3-way merge...
Auto-merging app/views/static_pages/_non_logged_in_home.html.erb
CONFLICT (content): Merge conflict in app/views/static_pages/_non_logged_in_home.html.erb
Auto-merging app/assets/stylesheets/custom.css.scss
CONFLICT (content): Merge conflict in app/assets/stylesheets/custom.css.scss
Auto-merging app/assets/javascripts/application.js
CONFLICT (content): Merge conflict in app/assets/javascripts/application.js
CONFLICT (modify/delete): app/assets/javascripts/.tern-port deleted in theme Bootswatch Journal, fixed Account
Drop-down and modified in HEAD. Version HEAD of app/assets/javascripts/.tern-port left in tree.
error: Failed to merge in the changes.
Patch failed at 0001 theme Bootswatch Journal, fixed Account Drop-down
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

➜  clone git:(a1104a3) ✗ git rebase --abort
```

GUI 환경과 CLI 환경 비교

Github

- **GUI 환경의 장점**
  - **직관적인 사용자 인터페이스**
    - 그래픽으로 구성된 아이콘, 버튼, 창 등을 통해 사용자와 컴퓨터가 상호 작용
    - 시각적 요소는 사용자가 작업을 더 쉽게 수행할 수 있도록 도와 줌
  - **비기술자에게 친숙함**
    - 대부분의 사람들은 이미 그래픽 인터페이스를 사용하여 스마트폰, 태블릿, 컴퓨터 등과 상호 작용해왔기 때문에 GUI 환경은 대다수의 사용자에게 익숙하고 친숙함

GUI 환경과 CLI 환경 비교

- **GUI 환경의 장점**
  - **시각적 피드백**
    - GUI 환경에서는 사용자가 작업을 수행할 때 시각적인 피드백을 받을 수 있음
      - (예) 버튼을 클릭하면 버튼이 눌렸음을 시각적으로 확인할 수 있음
  - **멀티태스킹 및 멀티윈도우 지원**
    - 여러 작업을 동시에 수행하거나 여러 창을 동시에 열어 작업할 수 있도록 지원
  - **더 많은 기능과 옵션**
    - 많은 GUI 애플리케이션은 다양한 기능과 옵션을 제공함
    - 사용자는 이러한 기능과 옵션을 시각적으로 탐색하고 사용할 수 있음

GUI 환경과 CLI 환경 비교

Github

- **GUI 환경의 단점**
  - **리소스 사용량**
    - GUI는 그래픽 처리와 애니메이션 효과 등이 필요하기 때문에 시스템 리소스를 상대적으로 많이 사용함
  - **학습 곡선**
    - GUI 환경은 사용자가 익숙해 지기까지 그래픽 요소의 사용을 위한 학습이 필요함
    - 특히 복잡한 애플리케이션의 경우 사용법을 익히는 데 시간이 걸릴 수 있음
  - **작업의 번거로움**
    - 일부 작업은 명령어를 통해 CLI 환경에서 더 빠르고 간단하게 수행할 수 있음
    - 그러나 GUI 환경에서는 반드시 일련의 단계를 거쳐야 함

GUI 환경과 CLI 환경 비교

Github

- CLI 환경의 장점
  - 빠른 작업
    - 명령 줄을 통해 직접 명령어를 입력하여 작업을 수행할 수 있기 때문에 작업을 더 빠르게 수행할 수 있음
    - GUI 환경에서는 마우스를 사용하여 메뉴를 탐색해야 하는 경우가 많아 작업에 더 많은 시간이 소요될 수 있음
  - 자동화 및 스크립팅
    - CLI 환경은 명령어를 스크립트로 작성하고 실행함으로써 반복적인 작업을 자동화할 수 있음
  - 리소스 사용량
    - CLI 환경은 일반적으로 GUI에 비해 적은 시스템 리소스를 사용하므로 시스템의 성능과 안정성에 이점을 제공

GUI 환경과 CLI 환경 비교

- CLI 환경의 장점
  - 원격 작업
    - 원격 서버 또는 네트워크 장치에 대한 원격 작업을 수행하는 데 효과적임
    - 텍스트 기반의 인터페이스로 원격으로 접속하고 명령을 실행할 수 있음

GUI 환경과 CLI 환경 비교

- **CLI 환경의 단점**
  - **기술적 지식 요구**
    - CLI는 명령어를 사용하여 작업을 수행하므로 일부 사용자에게는 기술적인 지식이 필요할 수 있음
    - 일부 사용자는 명령어를 배우고 익히는 데 시간이 걸릴 수 있음
  - **시각적 피드백 부족**
    - CLI 환경에서는 작업의 시각적 피드백이 제한적임
    - 작업이 성공적으로 수행되었는지 확인하기 위해 추가적인 명령어나 로그를 확인해야 할 수 있음
  - **작업 명령의 복잡성**
    - 복잡한 작업은 여러 단계의 명령어를 조합하여 수행해야 하며 이는 명령어를 순서대로 정확히 입력해야 함을 의미

GUI 환경과 CLI 환경 비교

- CLI 환경의 단점
  - 작업 명령의 복잡성
    - 일부 복잡한 작업은 여러 단계의 명령어를 조합하여 수행해야 함
    - 이는 명령어를 정확히 입력하고 순서를 지켜야 함을 의미함

Git GUI 환경

- **Git GUI 도구의 장점**
  - **시각적 표현과 편의성**
    - 변경 내역, 브랜치, 커밋 로그 등 Git의 다양한 요소들을 시각적으로 표현해 주므로 이를 통해 코드의 상태를 쉽게 이해하고 변경 사항을 확인할 수 있음
    - 복잡한 Git 명령어를 직접 입력하지 않고도 그래픽 요소를 클릭하거나 드래그 앤 드롭하여 작업을 수행할 수 있어 사용자 편의성이 향상됨
  - **작업의 편리성**
    - Git의 기능을 직관적으로 제공하여 작업을 훨씬 간편하게 수행할 수 있음
    - 파일의 변경 내역을 선택적으로 스테이징하거나 커밋할 수 있으며, 브랜치를 생성하고 전환하는 등의 작업도 시각적으로 쉽게 수행할 수 있음

Git GUI 환경

- **Git GUI 도구의 장점**
  - **히스토리 관리 및 시각화**
    - 커밋 히스토리를 시각적으로 표현해 주므로 이를 통해 개별 커밋과 브랜치 간의 관계를 쉽게 파악할 수 있음
    - 필요한 경우 특정 커밋으로 이동하거나 브랜치를 병합하는 등의 작업을 쉽게 수행할 수 있음
    - 이는 프로젝트의 변경 이력을 관리하고 이해하는 데 큰 도움이 됨
  - **Git 기능의 완전성**
    - Git GUI 도구는 Git의 대부분 기능을 지원함
    - 여러 브랜치와 리모트 저장소 간의 병합, 리베이스, 충돌 해결 등과 같은 고급 Git 작업을 그래픽 인터페이스를 통해 수행할 수 있음

Git GUI 환경

- **Git GUI 도구의 장점**
  - **시각적 피드백과 실시간 업데이트**
    - 작업 중인 파일이나 스테이징된 변경 내역을 시각적으로 확인할 수 있음
    - 작업을 수행하는 동안 실시간으로 변경 사항을 업데이트해 주므로 다른 개발자와의 협업 시에도 변경 사항을 실시간으로 반영하여 효율적인 작업을 도모할 수 있음

  - **종합적으로 Git GUI 환경은**
    - Git을 보다 직관적이고 편리하게 사용할 수 있도록 도와줌
    - 시각적 표현, 작업 편리성, 히스토리 관리 및 시각화, Git 기능의 완전성, 실시간 업데이트 등의 장점을 통해 개발자들은 Git을 보다 쉽고 효율적으로 활용할 수 있음

Git GUI 환경

- **Git GUI 도구의 단점**
  - **학습 곡선**
    - CLI와는 다르게 그래픽 요소와의 작업 흐름에 익숙해지기까지 학습 곡선이 존재할 수 있으며 적응 시간이 필요함
  - **기능 제한**
    - Git GUI 도구는 Git의 대부분 기능을 지원하지만, 모든 고급 Git 작업을 제공하지는 않을 수 있음
    - 복잡한 작업이나 특정한 Git 명령어를 사용해야 하는 경우 CLI를 사용하는 것이 더 적합할 수 있음
  - **성능과 리소스 사용량**
    - CLI에 비해 시스템 리소스를 상대적으로 많이 사용할 수 있음
    - 또한 그래픽 처리와 인터페이스의 다양한 기능으로 인해 성능 저하가 발생할 수 있음

Git GUI 환경

- **Git GUI 도구의 단점**
  - **일관성 및 호환성**
    - Git은 CLI를 기반으로 설계되었으므로 Git CLI를 사용하여 작업하는 경우와 Git GUI 도구를 사용하여 작업하는 경우의 일관성과 호환성에 차이가 있을 수 있음
    - 다른 도구나 환경에서 작업한 후에 Git GUI 도구로 전환하면 일부 동작이 다를 수 있으므로 주의가 필요함
  - **제한된 유연성**
    - Git GUI 도구는 그래픽 요소와 기능이 제한되어 있을 수 있음
    - 사용자 정의나 확장성 측면에서는 자신만의 스크립트를 작성하거나 원하는 대로 작업을 구성할 수 있는 CLI가 더 유연한 선택일 수 있음

Git GUI 환경

- **Git GUI 도구의 단점**
  - **종합적으로, Git GUI 환경은**
    - 사용자에게 편의성과 시각적 피드백을 제공하지만,
    - 학습 곡선, 기능 제한, 성능 및 리소스 사용량, 일관성 및 호환성, 제한된 유연성 등의 단점을 가지고 있음
    - 개발자는 자신의 작업 스타일과 요구 사항에 맞는 Git 도구를 선택하는 것이 중요함

Git CLI 환경

- **Git CLI 도구의 장점**
  - **강력하고 완전한 기능**
    - Git의 강력한 기능들을 직접 명령어를 통해 사용할 수 있으며, 브랜치 관리, 커밋, 병합, 리베이스, 충돌 해결 등 Git의 모든 작업을 다룰 수 있음
  - **자동화 및 스크립팅**
    - 명령어를 스크립트로 작성하고 실행함으로써 반복적인 작업을 자동화할 수 있으며, 이를 통해 Git 작업의 일관성을 유지하고 시간을 절약할 수 있음
  - **성능과 리소스 사용량**
    - Git GUI에 비해 시스템 리소스를 적게 사용함
    - 그래픽 처리나 인터페이스의 추가 오버헤드가 없으므로 일관된 성능을 제공함
    - 대용량 프로젝트에서도 효율적으로 작업할 수 있음

Git CLI 환경

- **Git CLI 도구의 장점**
  - **유연성과 확장성**
    - 각 명령어와 옵션을 조합하여 원하는 작업 흐름을 만들 수 있음
    - Git의 기능을 확장하고 사용자 정의 스크립트를 작성하는 등의 방법으로 작업을 유연하게 구성할 수 있음
  - **멀티플랫폼 지원**
    - 다양한 운영 체제(Windows, macOS, Linux 등 모든 주요 플랫폼)에서 Git CLI를 사용하여 동일한 작업을 수행할 수 있음
  - **원격 작업과 협업**
    - Git CLI를 사용하여 원격 저장소와의 작업을 수행하거나 다른 개발자와의 협업을 진행할 수 있음
    - 원격 저장소의 클론, 풀, 푸시, 충돌 해결 등을 명령어로 수행할 수 있으며, 이는 Git의 브랜치 모델을 이해하고 협업하는 데 도움이 됨

Git CLI 환경

- **Git CLI 도구의 장점**
  - **종합적으로, Git CLI 환경은**
    - 강력하고 완전한 기능, 자동화 및 스크립팅 가능성, 성능과 리소스 사용량의 효율성, 유연성과 확장성, 멀티플랫폼 지원, 원격 작업과 협업 기능 등을 제공하며
    - 개발자들은 Git CLI를 사용하여 복잡한 Git 작업을 자유롭게 수행하고,
    - 원하는 방식으로 작업 흐름을 제어할 수 있음

Git CLI 환경

- **Git CLI 도구의 단점**
  - **학습 곡선**
    - 명령어를 사용하여 작업을 수행하므로 사용자에게 기술적인 지식이 필요할 수 있음
    - 처음 사용하는 사용자들에게는 명령어의 문법과 옵션을 배우고 익히는 데 시간이 걸릴 수 있음
  - **복잡성**
    - 일부 Git 작업은 여러 단계의 명령어를 조합하여 수행해야 함
    - 이러한 작업은 CLI 환경에서는 명령어의 정확한 순서와 옵션을 알아야 하므로 실수가 발생할 수 있음
  - **시각적 피드백 부족**
    - CLI 환경에서는 작업의 시각적 피드백이 제한적임
    - 예시: 파일의 변경 내역이나 브랜치의 상태를 시각적으로 확인하려면 명령어의 결과를 분석해야 함

Git CLI 환경

- **Git CLI 도구의 단점**
  - **복잡한 충돌 해결**
    - CLI 환경에서 충돌이 발생한 경우, 충돌 해결 작업이 복잡할 수 있음
    - 충돌이 있는 파일을 개별적으로 확인하고 수정하는 작업은 일부 사용자에게 어려울 수 있음
  - **사용자 오류 가능성**
    - 명령어를 사용하여 작업을 수행하는 CLI 환경에서는 사용자가 실수로 잘못된 명령어나 옵션을 입력할 수 있음
    - 이는 잘못된 동작이나 예기치 않은 결과를 초래할 수 있음
  - **시스템 종속성**
    - Git CLI는 운영 체제에 종속적임
    - 특정 운영 체제에서는 CLI 명령어의 동작이 다를 수 있으며, 사용자들은 각 운영 체제에 맞는 명령어를 사용해야 함

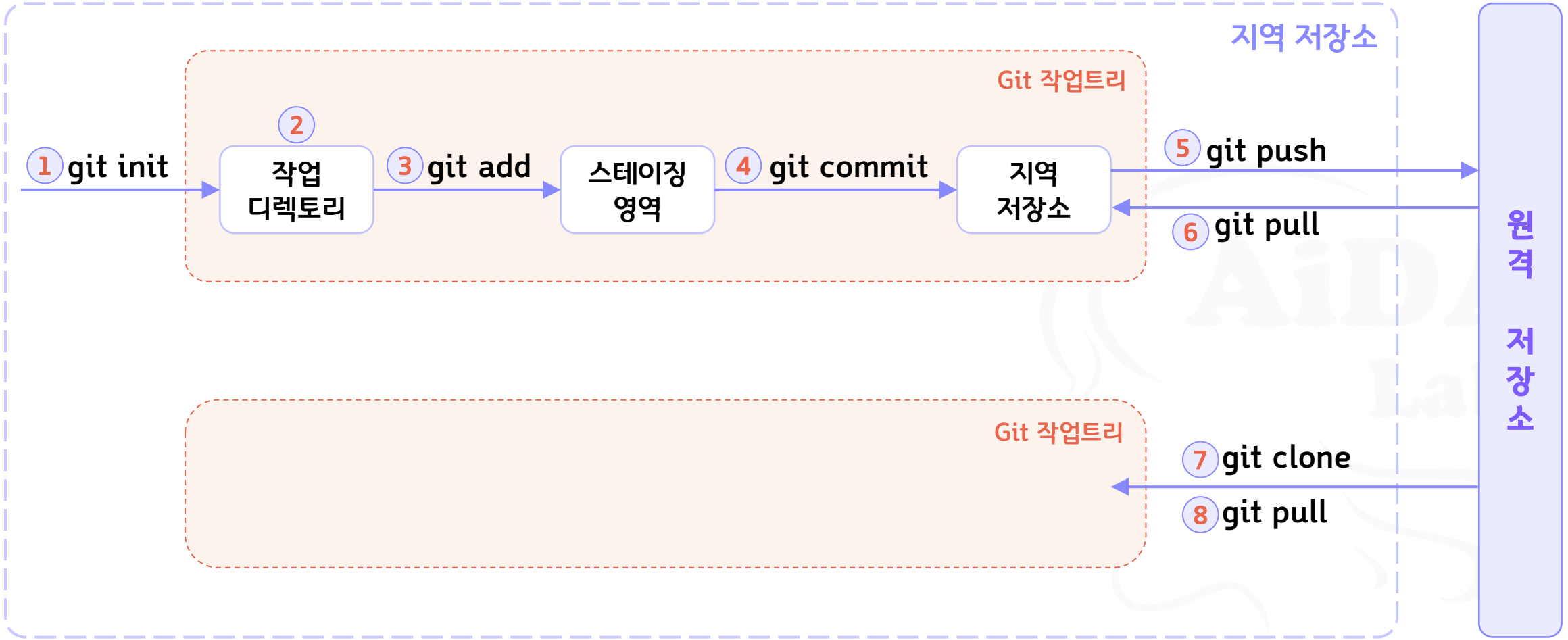
Git CLI 환경

- **Git CLI 도구의 단점**
  - **종합적으로, Git CLI 환경은**
    - 강력하고 유연한 기능을 제공하지만,
    - 학습 곡선, 복잡성, 시각적 피드백 부족, 복잡한 충돌 해결, 사용자 오류 가능성, 시스템 종속성 등의 단점을 가짐
    - 개발자는 자신의 작업 스타일과 요구 사항에 맞는 Git 도구를 선택하는 것이 중요함

기본 명령어 실습

Git/GitHub 소스 관리 기본 흐름

- **Git/GitHub 소스 관리 기본 흐름**



Git/GitHub 소스 관리 기본 흐름

- **Git/GitHub 소스 관리 기본 흐름**
  - **지역 저장소 → GitHub(원격 저장소)**
    1. 지역 저장소에 새 프로젝트 생성
    2. git init 명령어로 해당 프로젝트를 Git 지역 저장소로 지정
    3. 파일 수정
    4. git add 명령어로 수정한 파일을 스테이징 영역으로 이동
    5. git commit 명령어로 지역 저장소에 저장
    6. git push 명령어로 지역 저장소에서 발생한 변경 내역을 원격 저장소에 반영

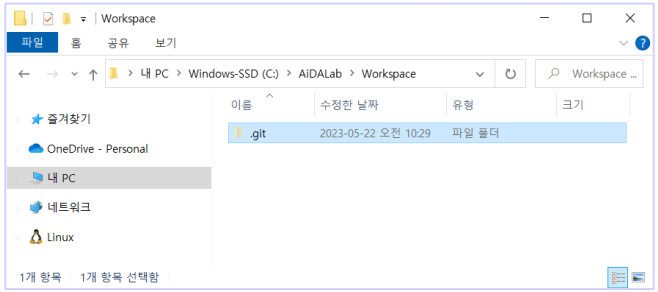
Git/GitHub 소스 관리 기본 흐름

- **Git/GitHub 소스 관리 기본 흐름**
  - **GitHub(원격 저장소) → 지역 저장소**
    - GitHub에 올려진 프로젝트 전체를 git clone 명령어로 다운로드
    - GitHub에 올려진 프로젝트에서 변경 사항만을 git pull 명령어로 다운로드
      (다른 사람이 수정한 내용을 나의 지역 저장소에 통합)

git init

- 기능: Git 초기화(=지역 저장소 생성)
- 명령어 일람
  - 초기화 수행 시
    - git init
  - 초기화 취소 시
    - rm -rf .git

그냥 파일 탐색기에서 삭제해도 됨




```
Windows PowerShell
PS C:\AiDALab\Workspace> ls -h
PS C:\AiDALab\Workspace> git init
Initialized empty Git repository in C:/AiDALab/Workspace/.git/
PS C:\AiDALab\Workspace> ls -h


    디렉터리: C:\AiDALab\Workspace


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d--h--       2023-05-22  오전 10:29                .git


PS C:\AiDALab\Workspace> cd .git
PS C:\AiDALab\Workspace\.git> ls


    디렉터리: C:\AiDALab\Workspace\.git


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       2023-05-22  오전 10:29                hooks
d-----       2023-05-22  오전 10:29                info
d-----       2023-05-22  오전 10:29                objects
d-----       2023-05-22  오전 10:29                refs
-a----       2023-05-22  오전 10:29            112 config
-a----       2023-05-22  오전 10:29             73 description
-a----       2023-05-22  오전 10:29             21 HEAD


PS C:\AiDALab\Workspace\.git>
```

git config

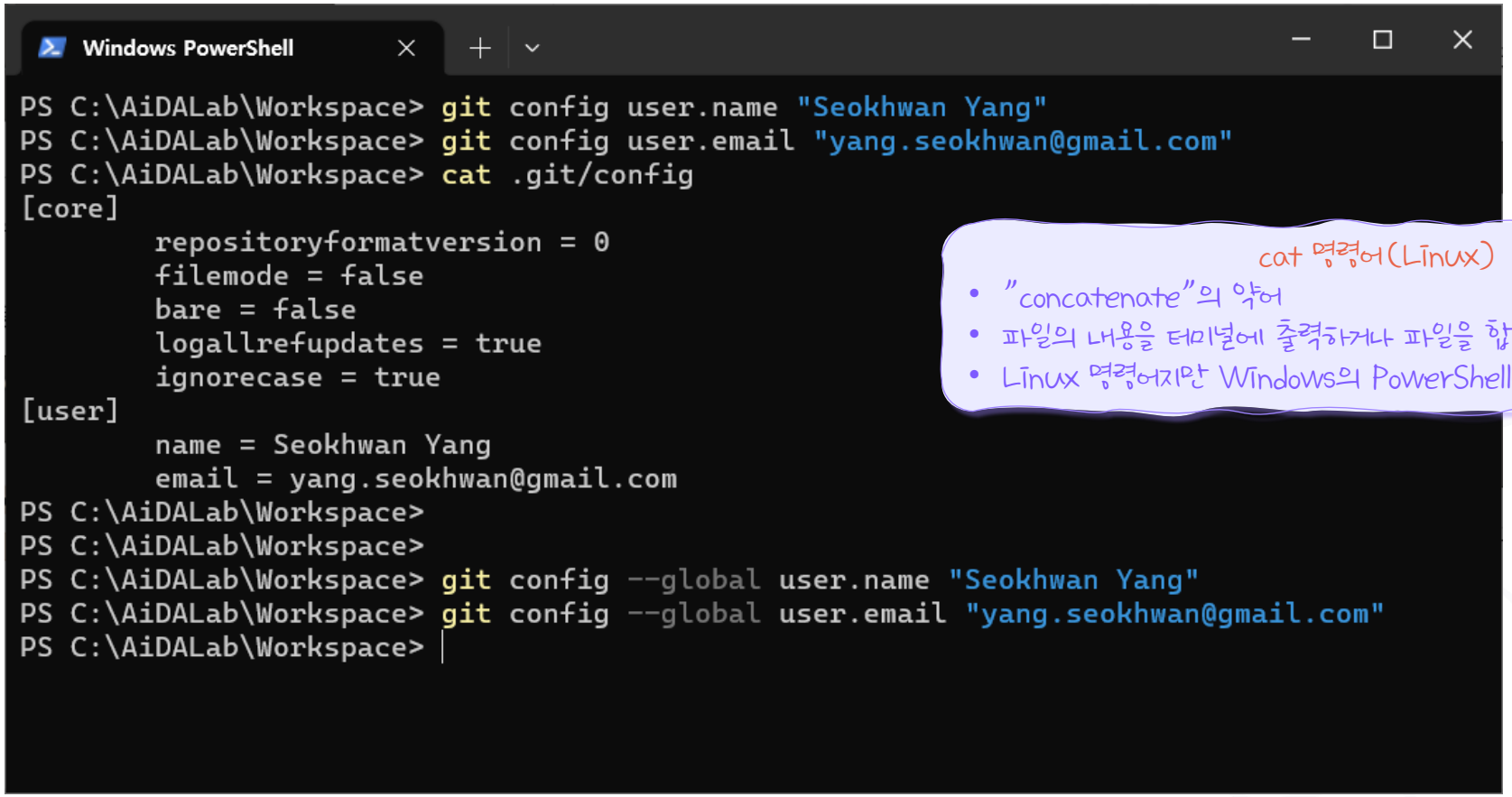
- 기능: 사용자 정보 등록
- 명령어 일람
  - 현재의 깃 지역 저장소에만 해당하는 사용자 정보를 등록할 경우
    - git config user.name “사용자 이름”
    - git config user.email “이메일 주소”

사용자 이름과 이메일 주소는
각자의 GitHub 계정의 Username, Email과 동일해야 함

  - 모든 프로젝트에 적용될 사용자 정보를 등록할 경우
    - git config --global user.name “사용자 이름”
    - git config --global user.email “이메일 주소”

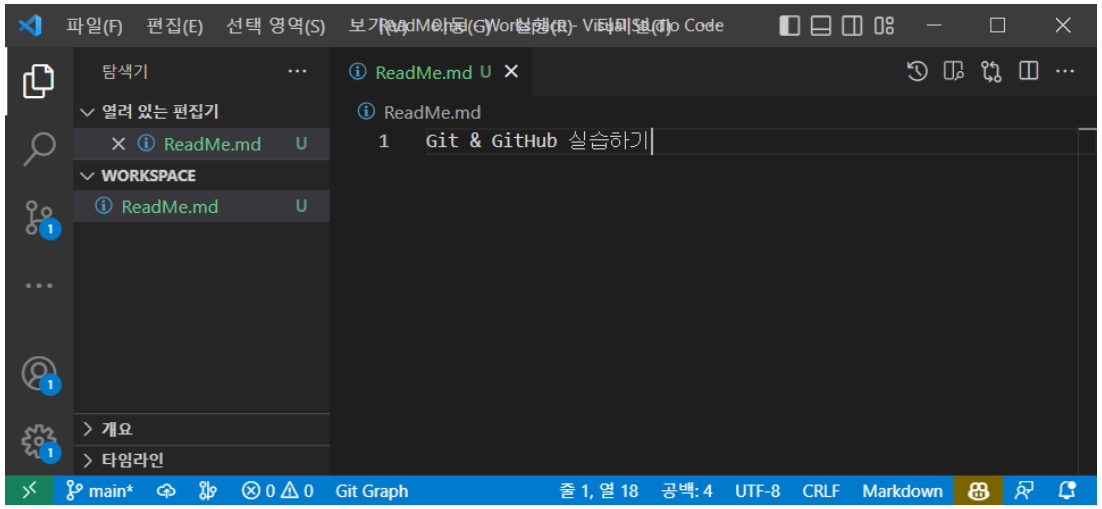
git config

- 기능: 사용자 정보 등록

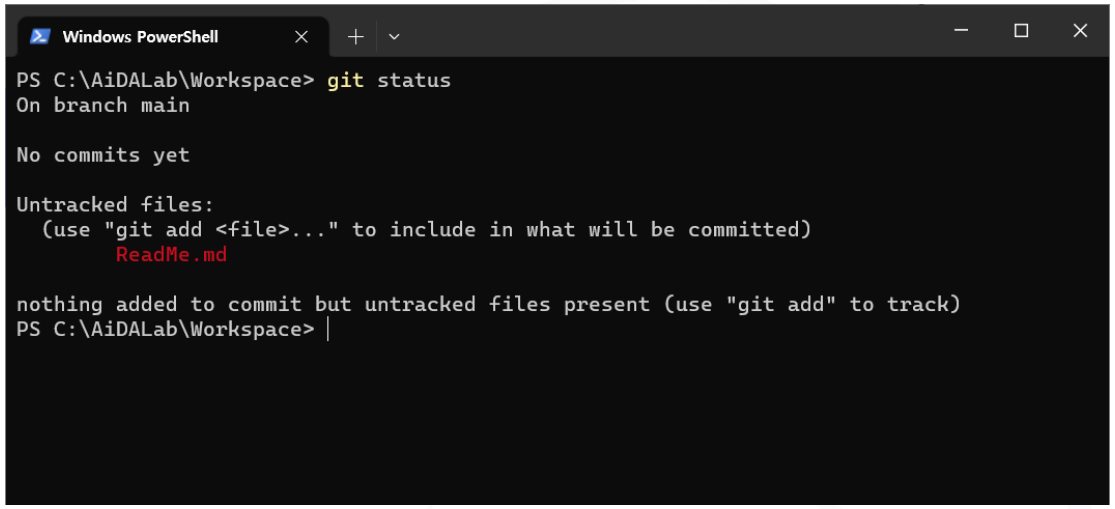


git status

- 기능: 현재 프로젝트의 파일 상태 확인하기
- 명령어 일람
  - git status



파일을 하나 만들고...

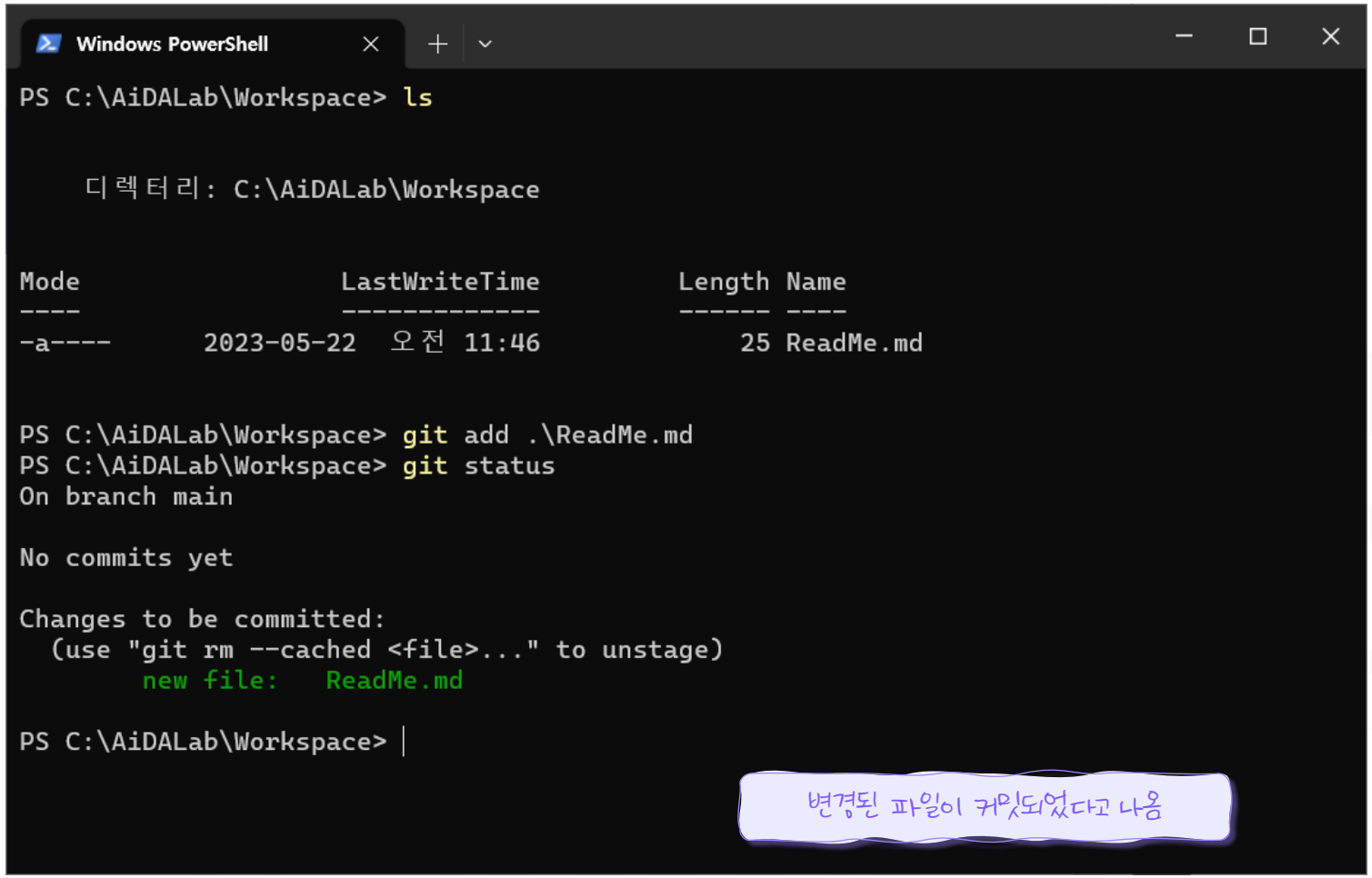


git status로 확인

아직 커밋에 등록된 파일이 없다고 나옴

git add

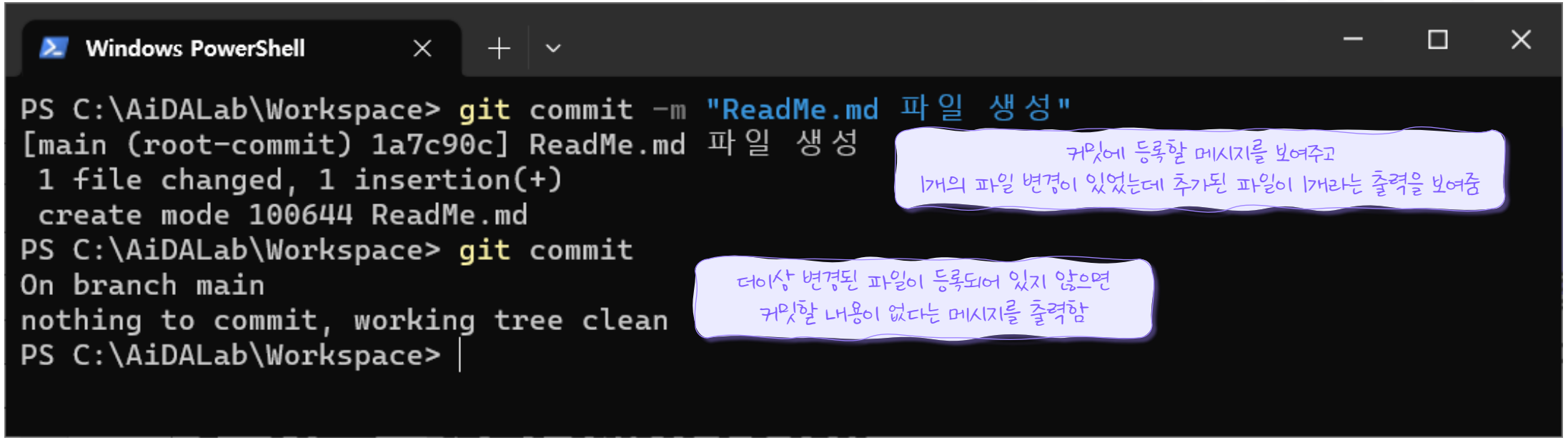
- 기능: 커밋에 포함될 파일 등록
- 명령어 일람
  - git add {파일 명}



git commit

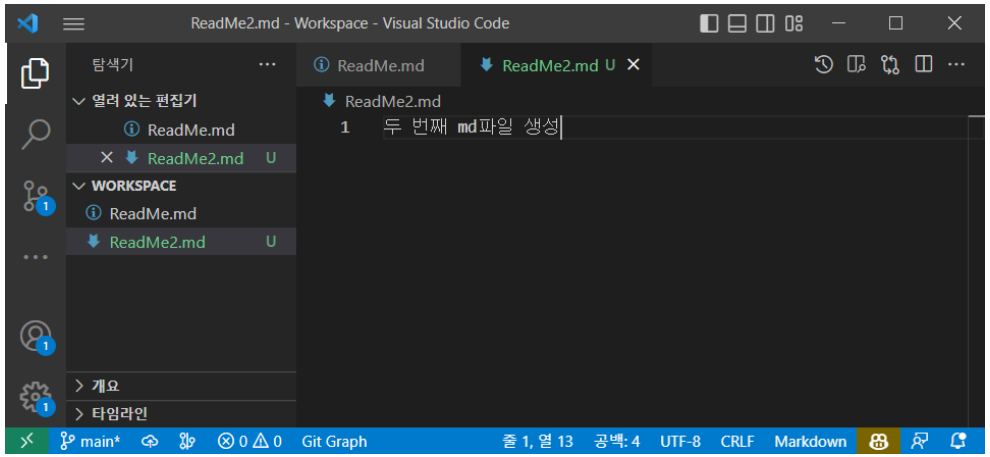
- 기능: 커밋 생성/수정
- 명령어 일람
  - 기본 사용법(커밋 생성)
    - git commit -m “메시지”

변경된 파일들을 묶어서 관련된 설명을 추가한 후 커밋을 수행함

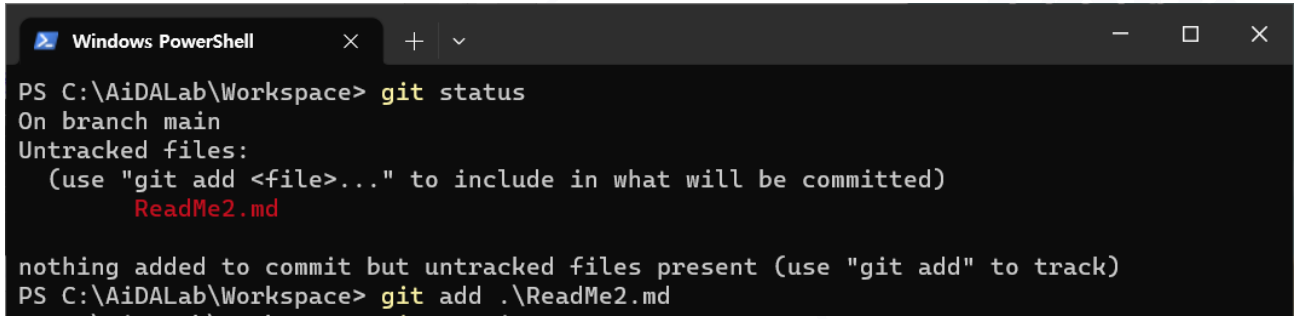


git commit

- **커밋 메시지를 상세하게 작성해야 할 경우(에디터를 통해 작성 가능)**
  - git commit

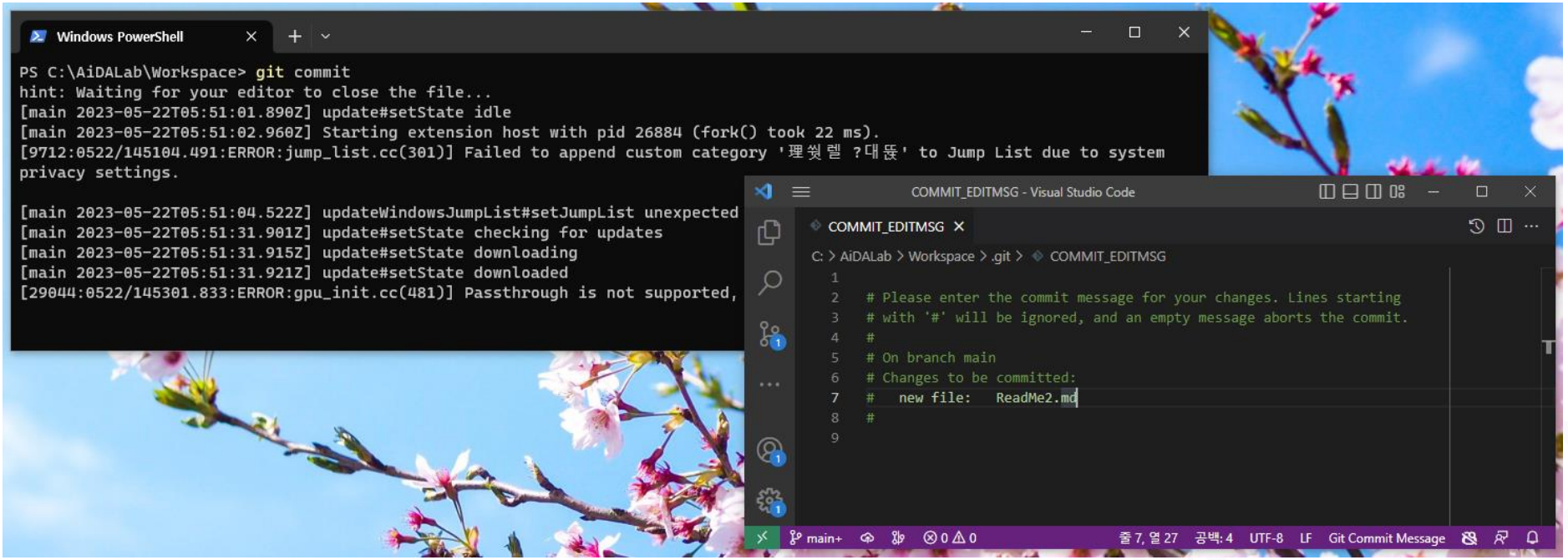


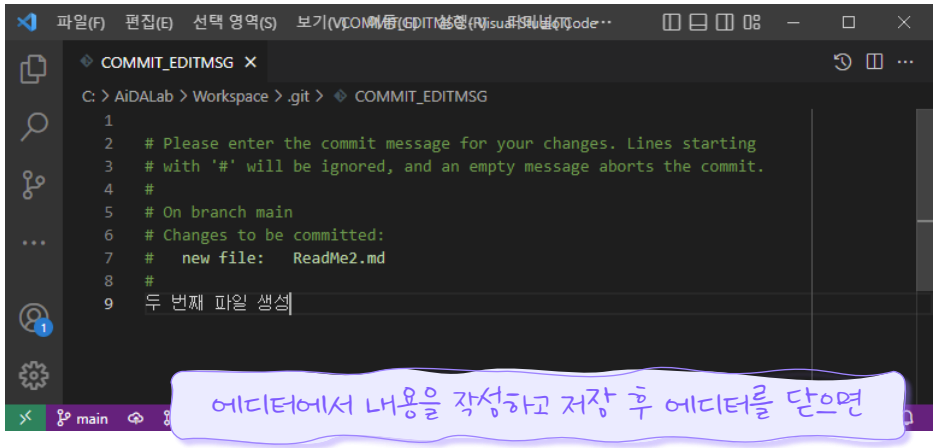
파일을 하나 더 만들고...



커밋 목록에 파일을 추가함

git commit



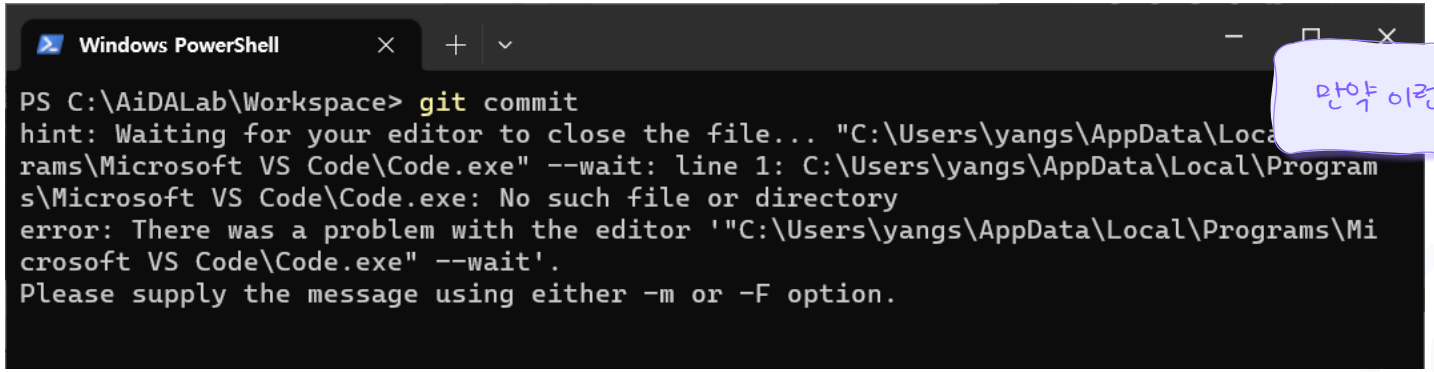


에디터에서 내용을 작성하고 저장 후 에디터를 닫으면

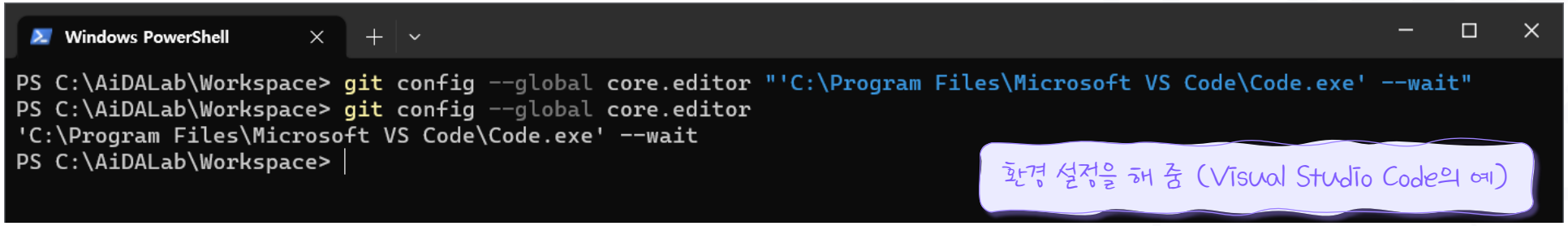
이와 같이 커밋이 완료됨

git commit

- **커밋 메시지를 상세하게 작성해야 할 경우(에디터를 통해 작성 가능)**
  - git commit



만약 이런 식으로 에디터가 실행되지 않는다면



환경 설정을 해 줌 (Visual Studio Code의 예)

git commit

- **에디터 별 환경 설정 방법**
  - VIM을 전용 에디터로 사용하는 경우
    - git config --global core.editor "vim"
  - SubLime을 전용 에디터로 사용하는 경우
    - git config --global core.editor "subl --wait"
  - Atom을 전용 에디터로 사용하는 경우
    - git config --global core.editor "atom --wait"
  - Visual Studio Code를 전용 에디터로 사용하는 경우
    - git config --global core.editor "code --wait"

만약 경로가 설정되어 있지 않아서 실행되지 않는다면 아래와 같이 경로를 직접 입력할 수 있음

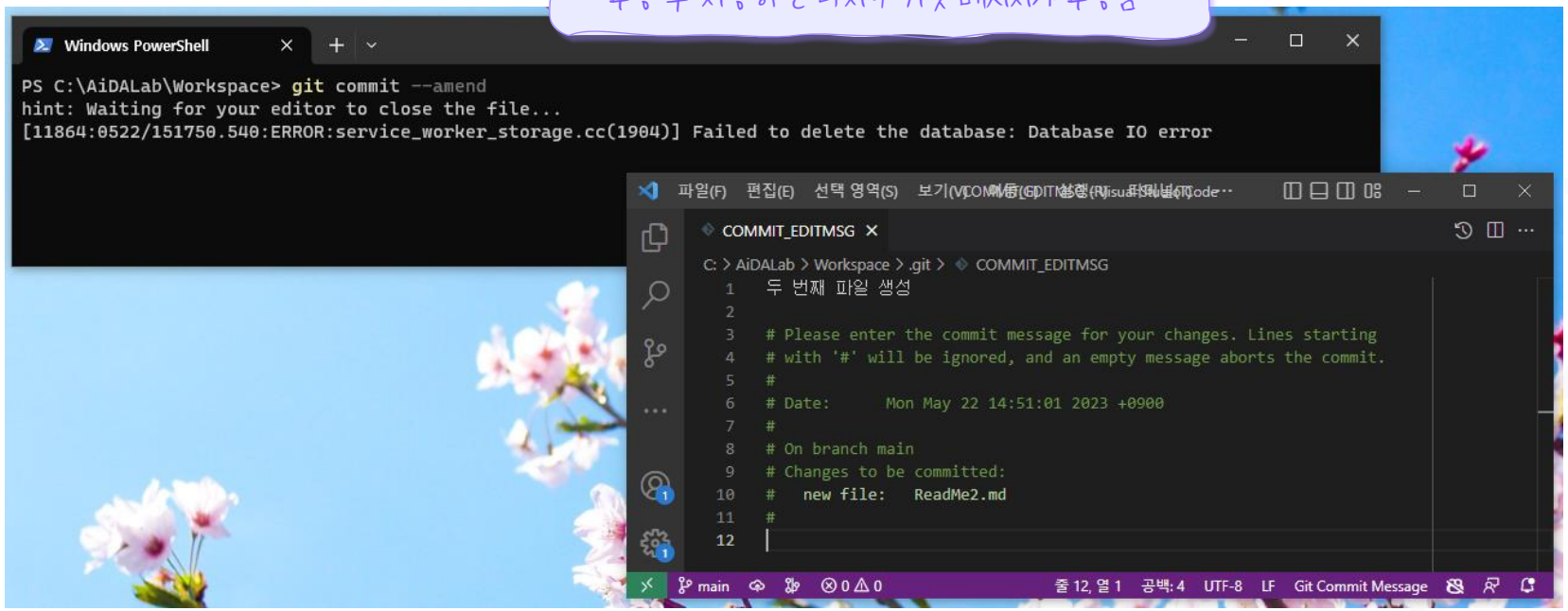
git config --global core.editor "'C:\Program Files\Microsoft VS Code\Code.exe' --wait"

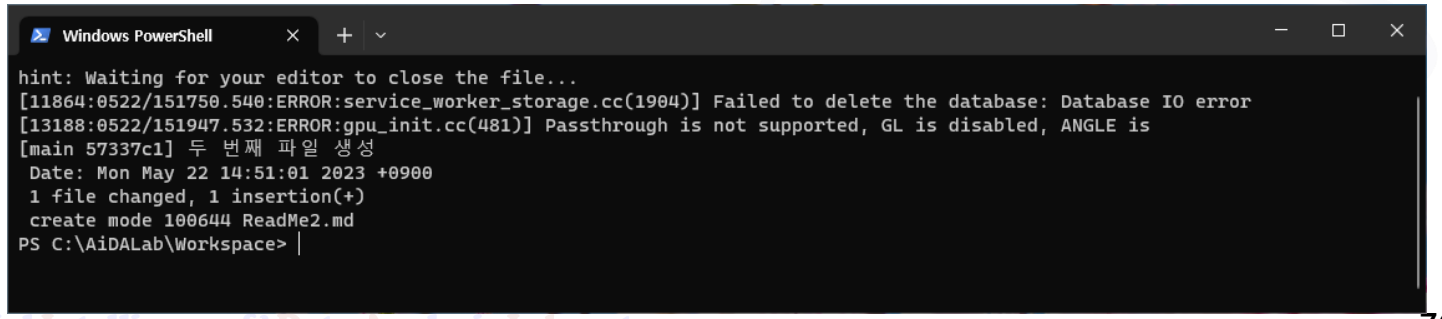
경로에 공백이 있는 경우, 따옴표를 한 번 더 사용해서 묶어 줌

git commit

- 기존 커밋을 수정해야 할 경우
  - git commit --amend

- 마지막 커밋 에디터 화면을 보여줌
- 수정 후 저장하면 마지막 커밋 메시지가 수정됨

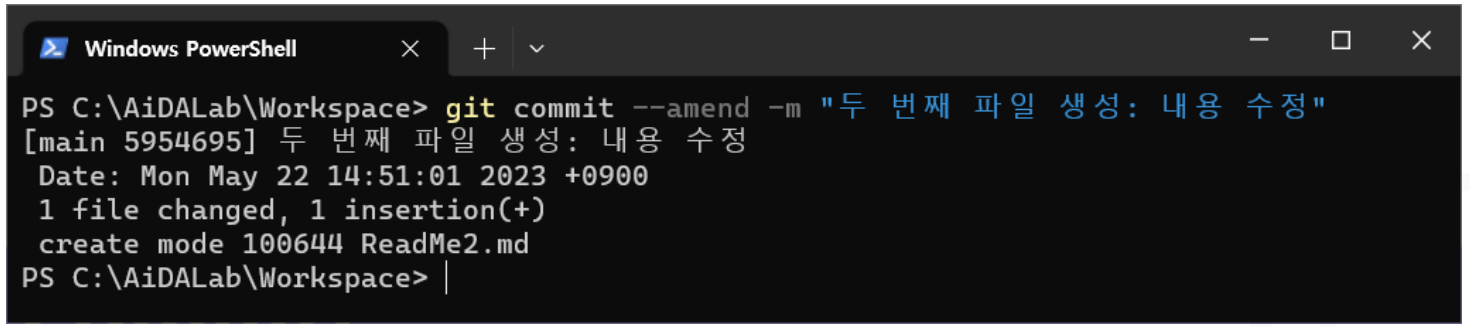




git commit

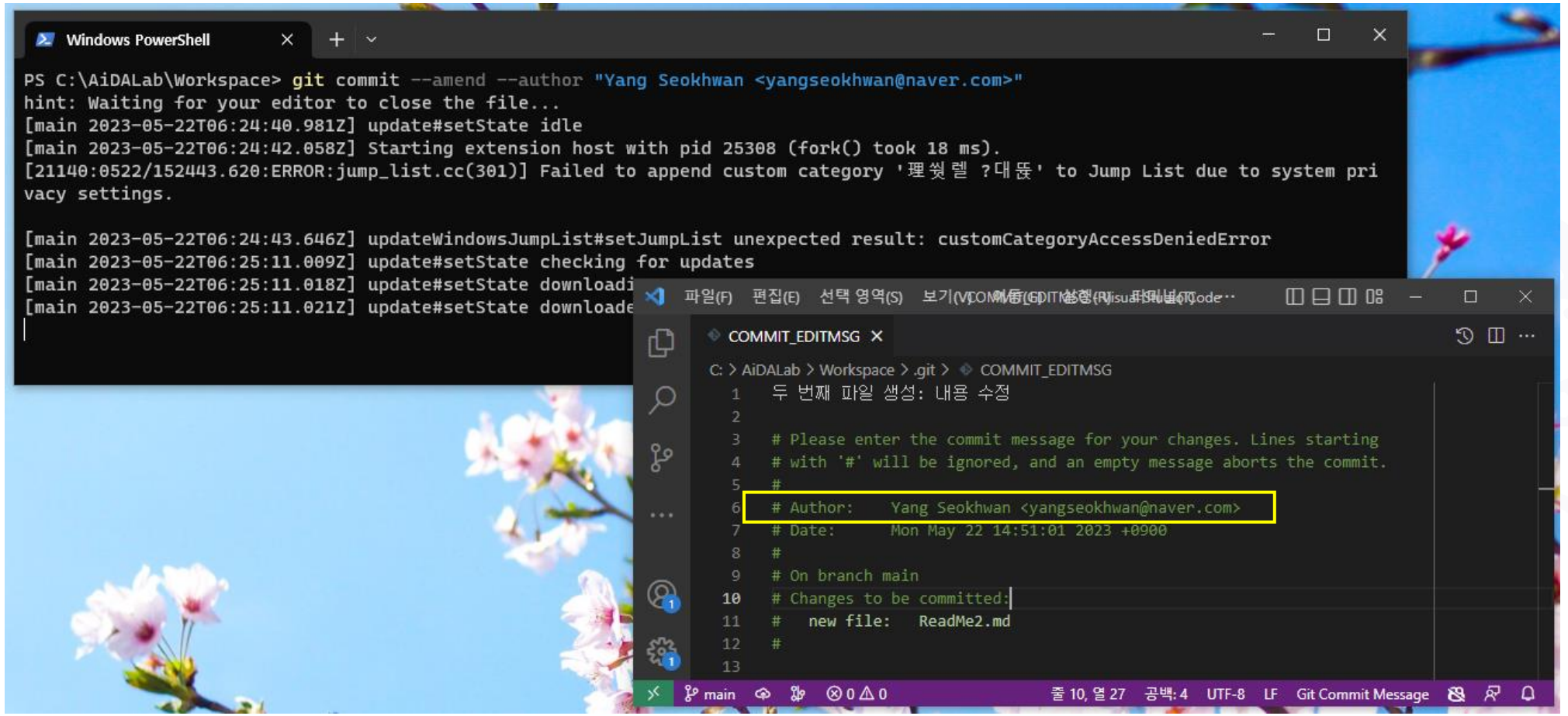
- **기존 커밋을 수정해야 할 경우**
  - git commit --amend -m “수정 메시지”

git commit --amend 명령과 동일하나 터미널 환경에서 수정함



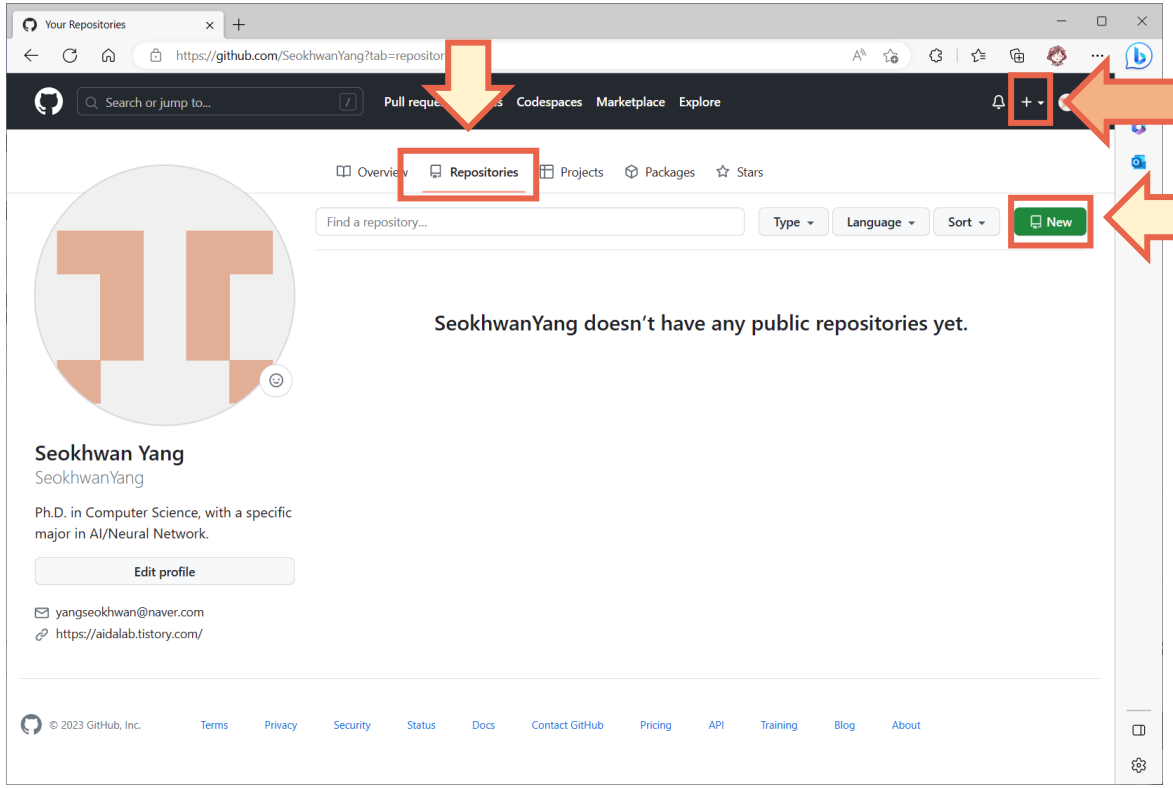
git commit

- 기존 커밋의 사용자를 수정해야 할 경우
  - git commit --amend --author “username <email>”

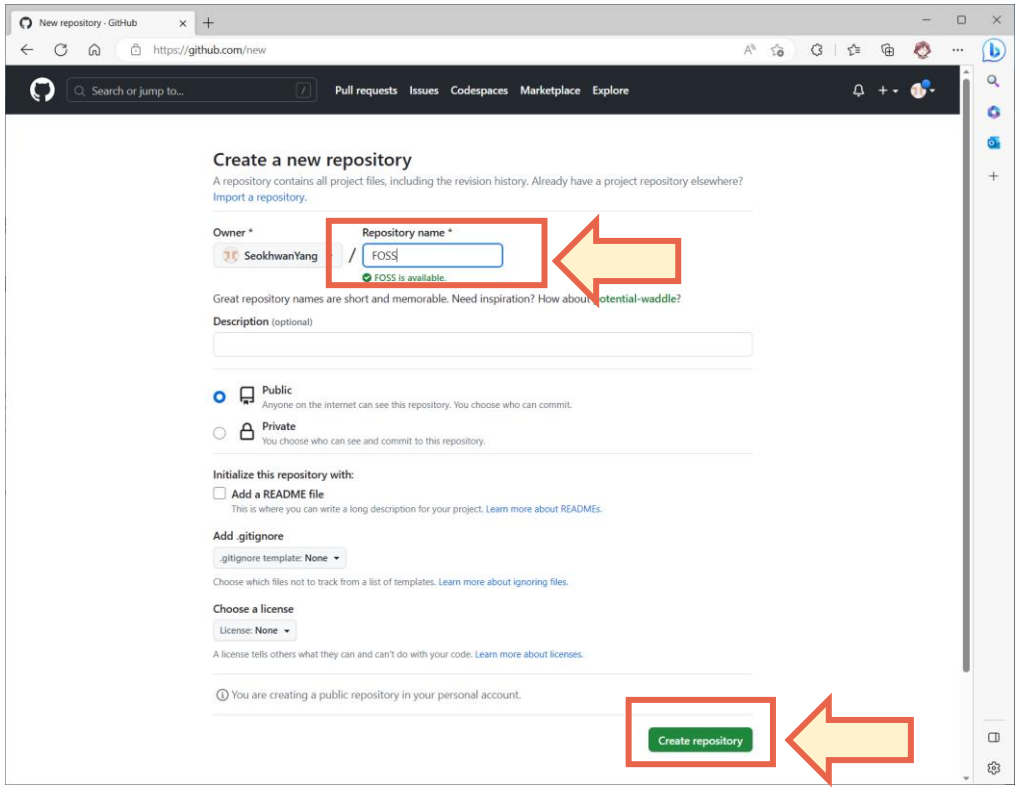


Git 원격 저장소 설정

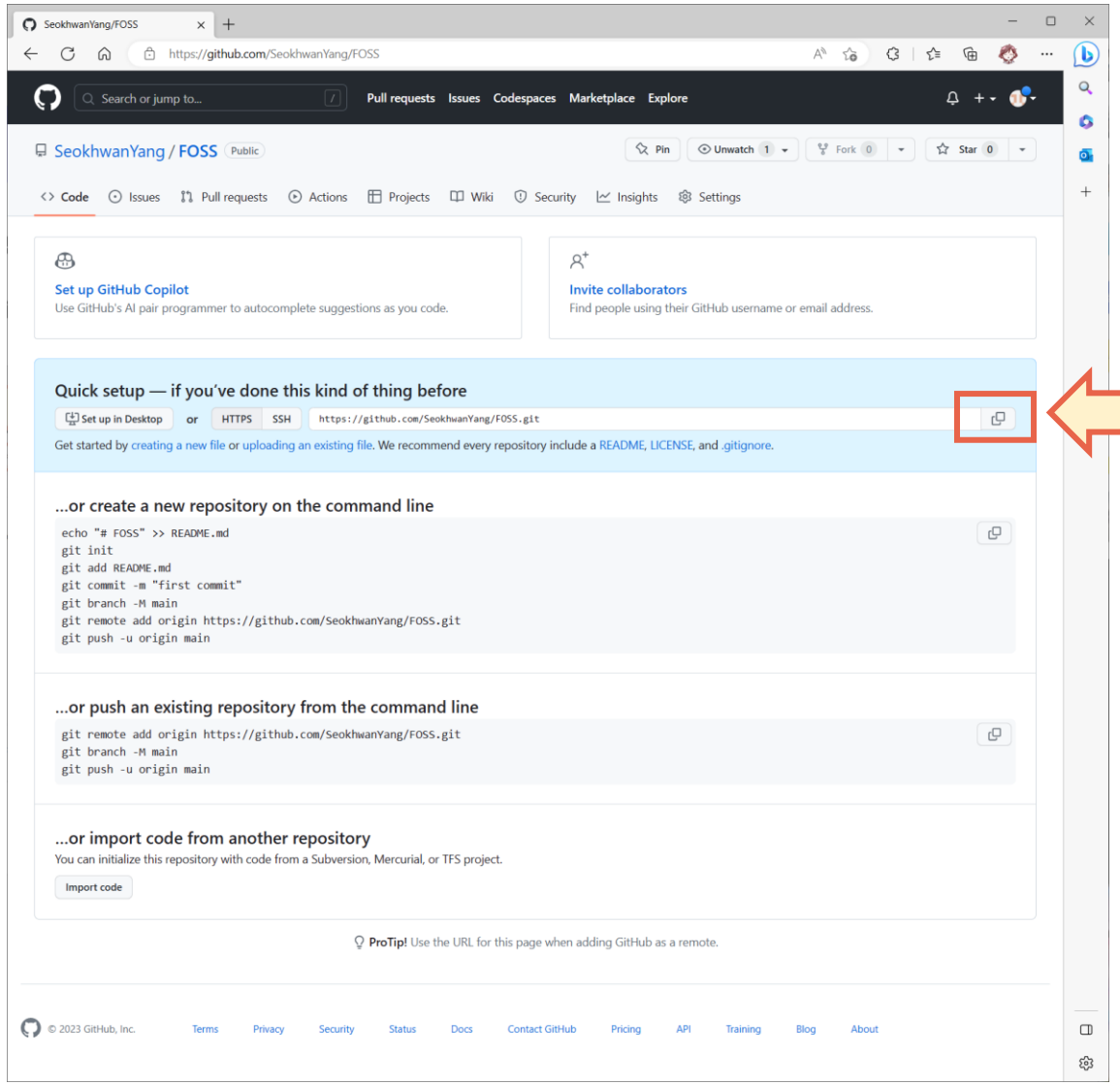
- 기능: Git 원격 저장소 설정
  - 먼저 GitHub 페이지에 가서 새로운 저장소(Create a new repository) 생성 작업을 수행함



또는



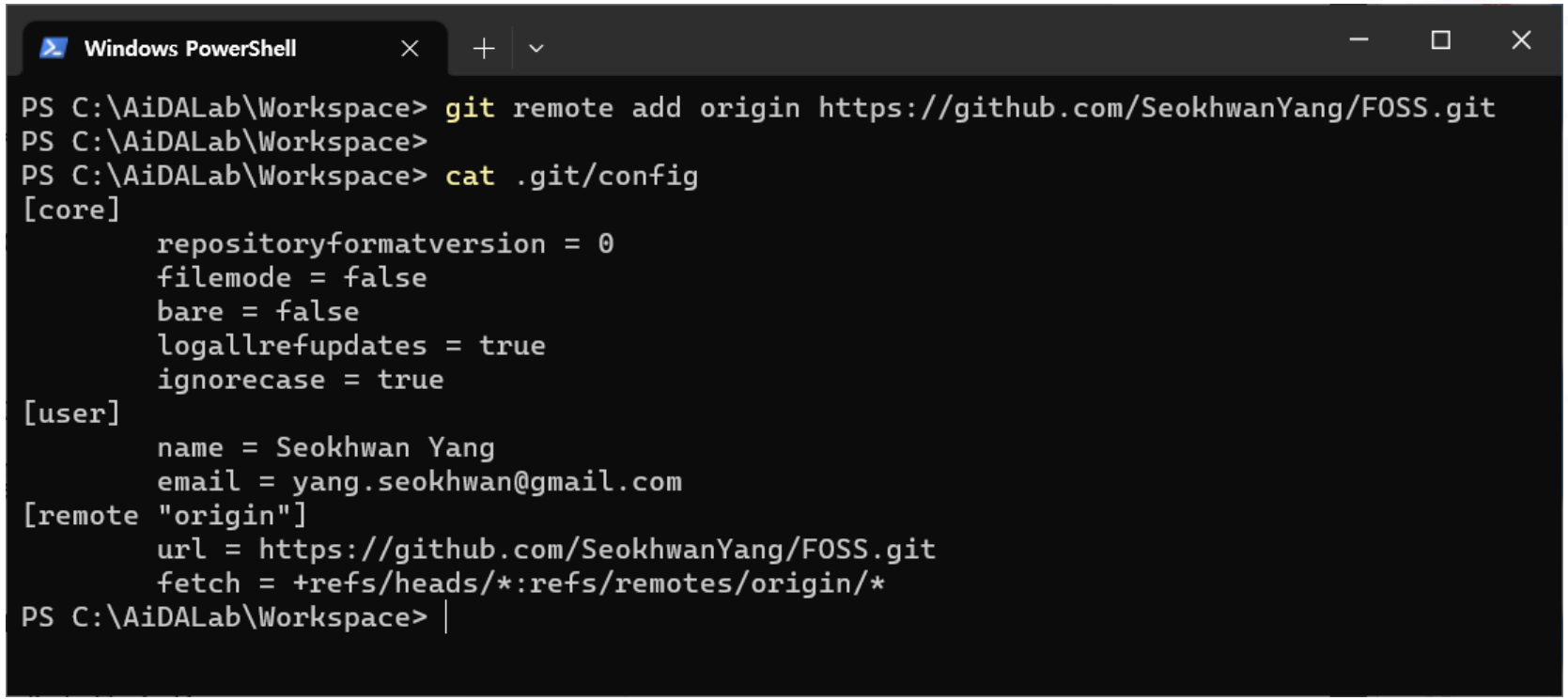
Git 원격 저장소 설정



원격 저장소 주소 복사

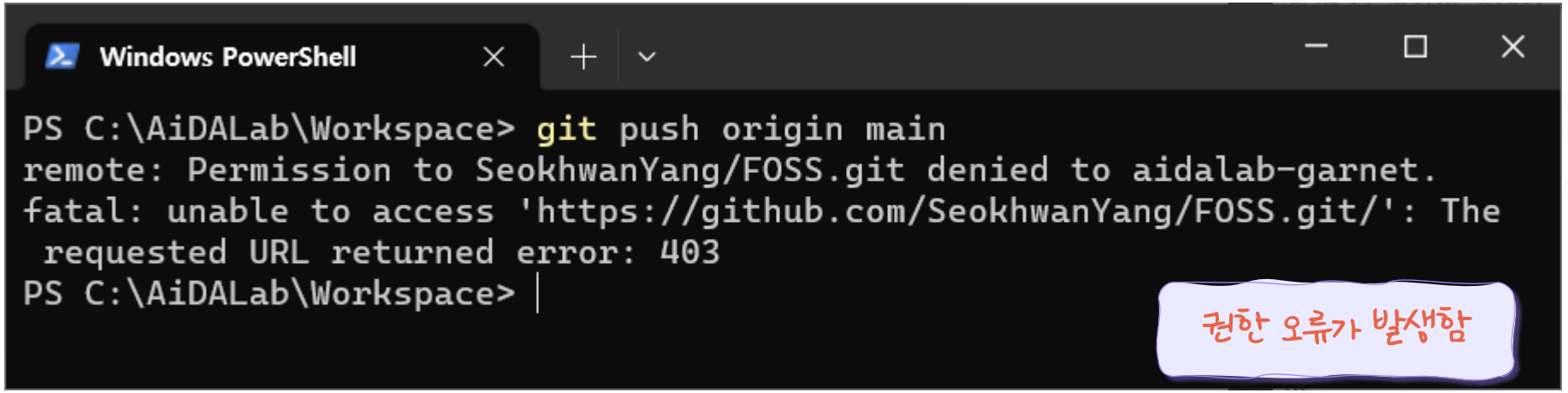
git remote add

- 기능: 원격 저장소 주소를 Git 지역 저장소에 등록
- 명령어 일람
  - git remote add origin {복사한 원격 저장소 주소}



git push

- 기능: 지역 저장소에 있는 커밋을 원격 저장소에 등록
- 명령어 일람
  - git push {원격 저장소(식별자)} {브랜치}

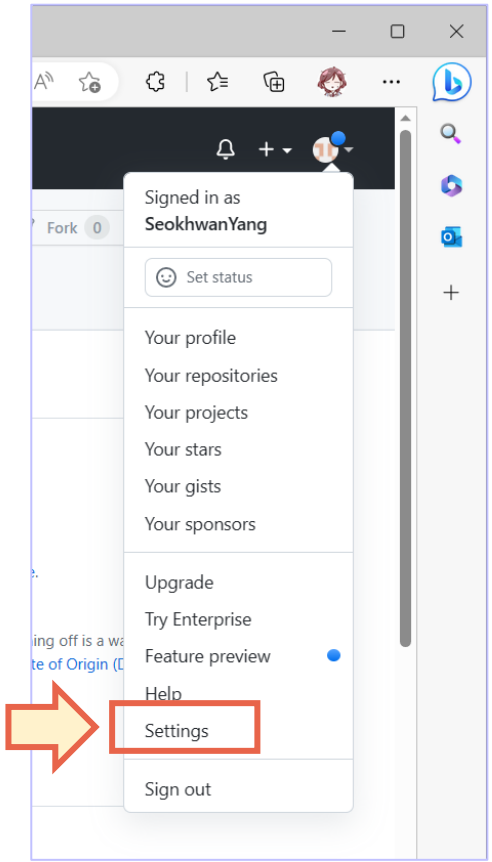


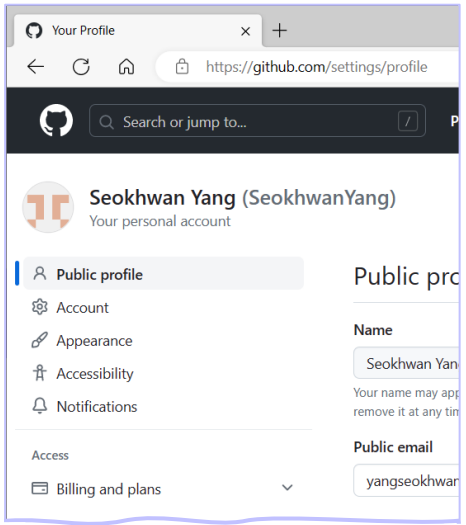
- 2021년 08월 13일부터 기존의 패스워드를 통한 인증 방식이 중단됨
- GitHub에서 제공하는 개인용 Access Token을 발급받아서 해당 Token을 패스워드로 사용해야 함

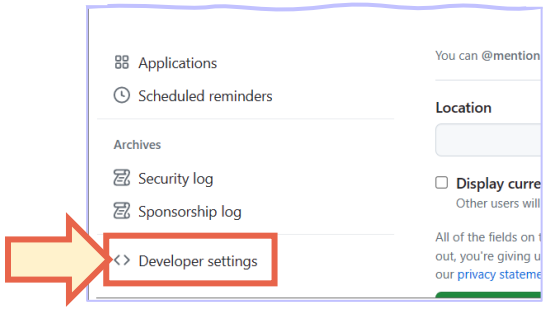
- [Setting] → [Developer settings] → [Personal access tokens]에서 발급

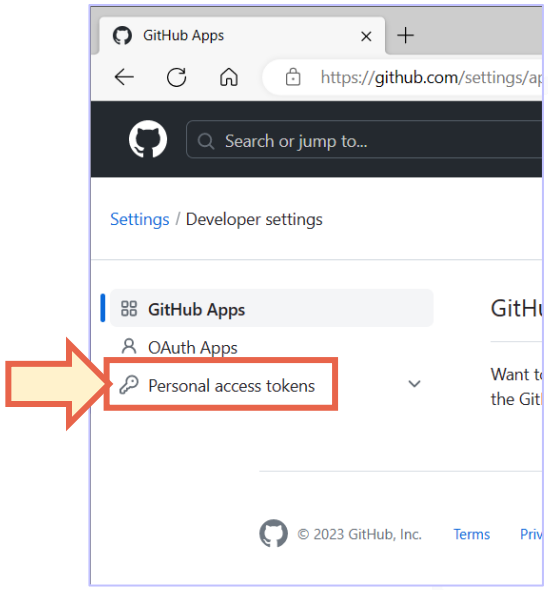
Personal Access Token 발급 받기

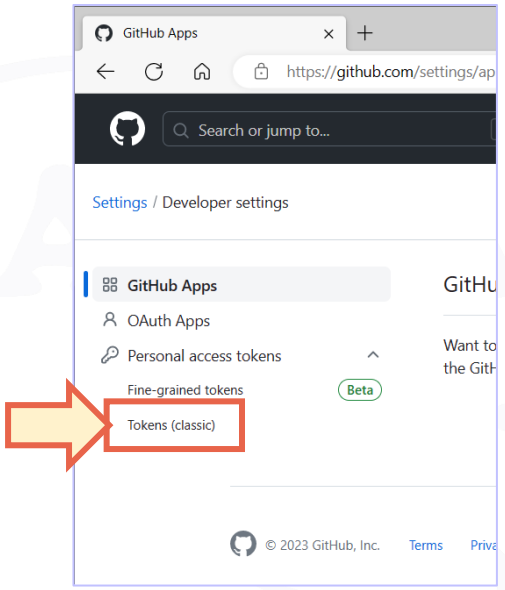
- [Setting] → [Developer][settings] → [Personal access tokens]



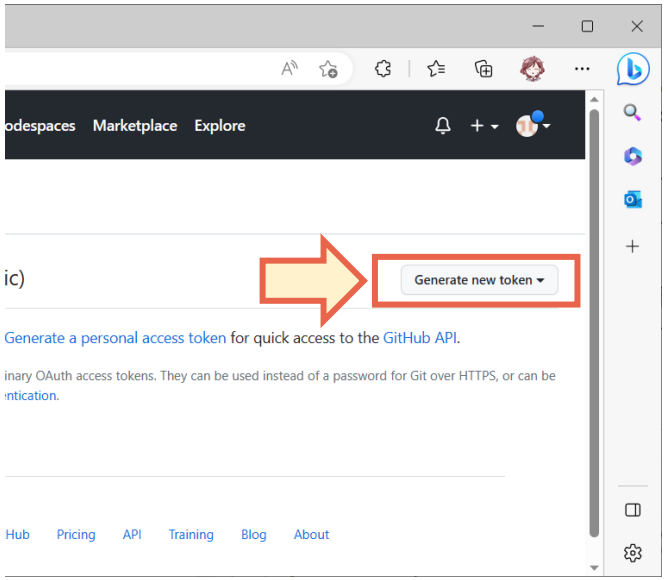


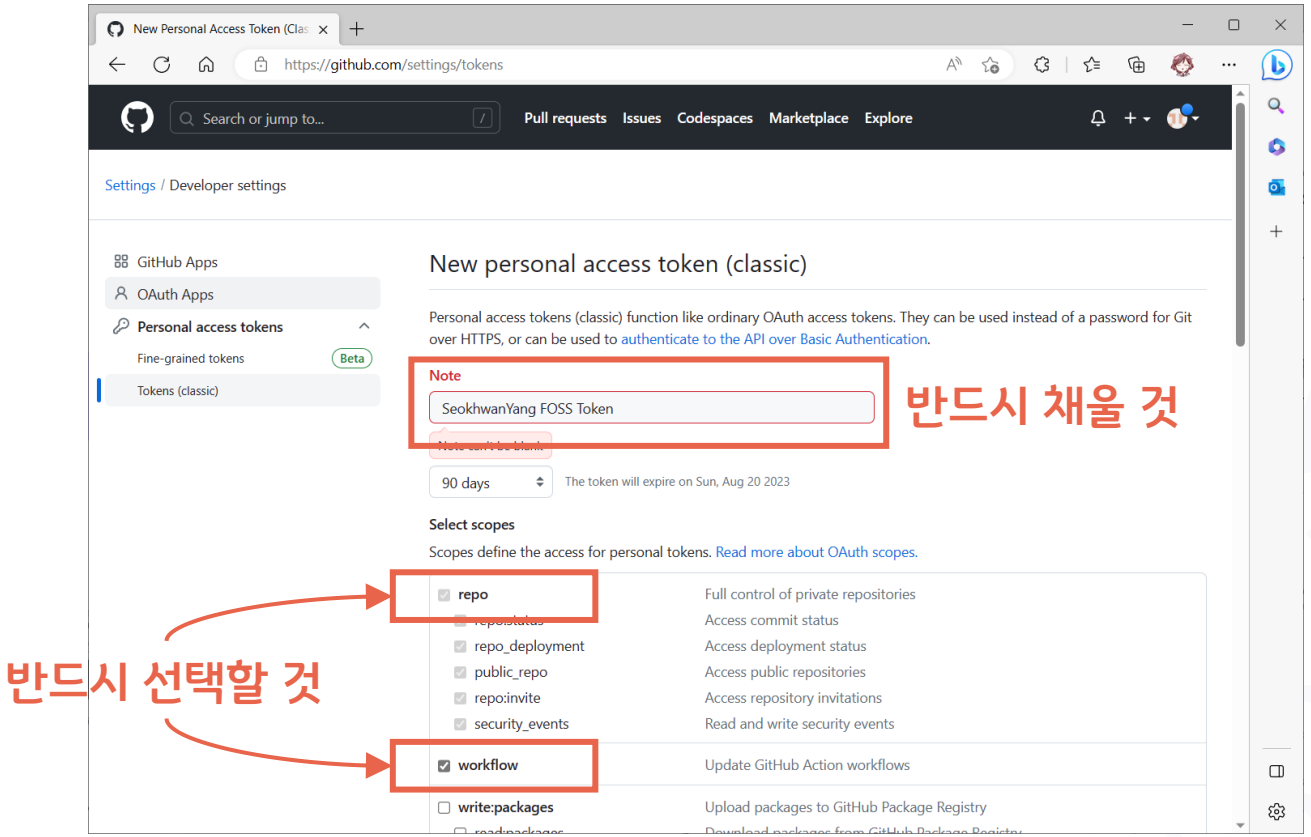




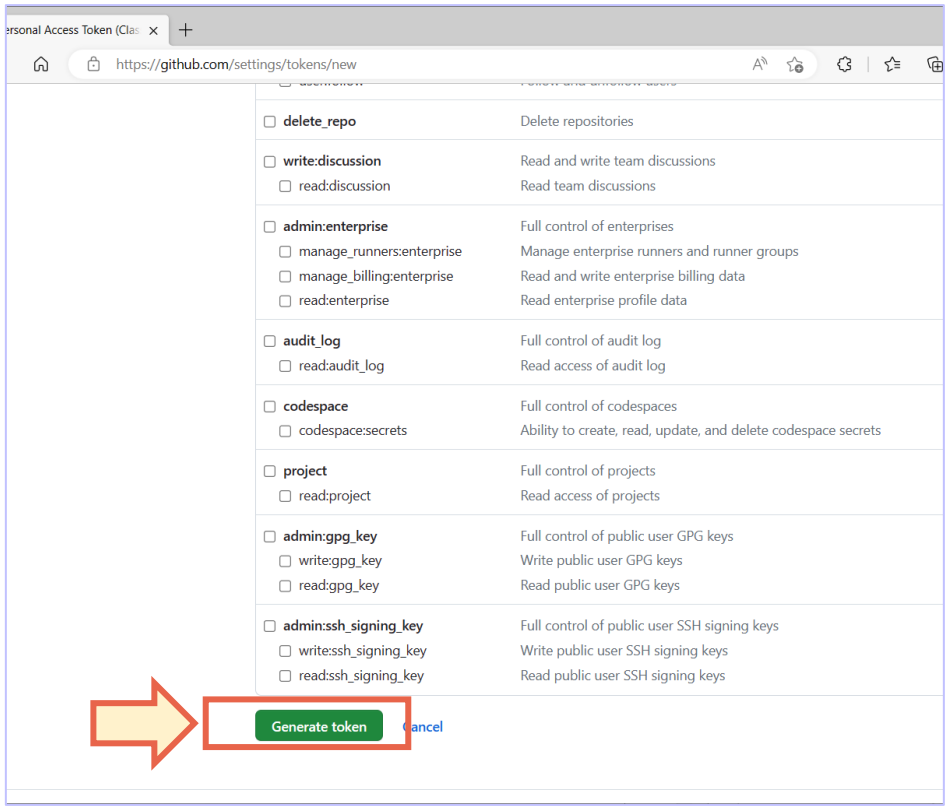


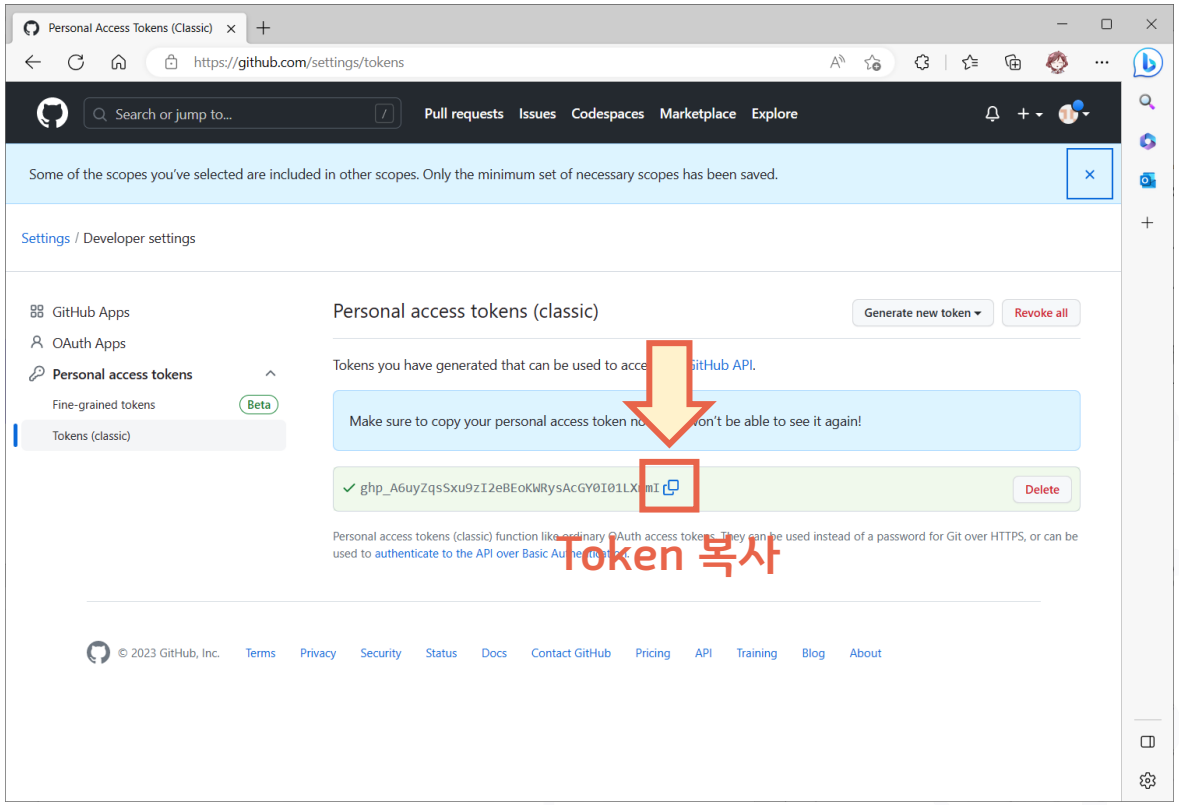
Personal Access Token 발급 받기





Personal Access Token 발급 받기

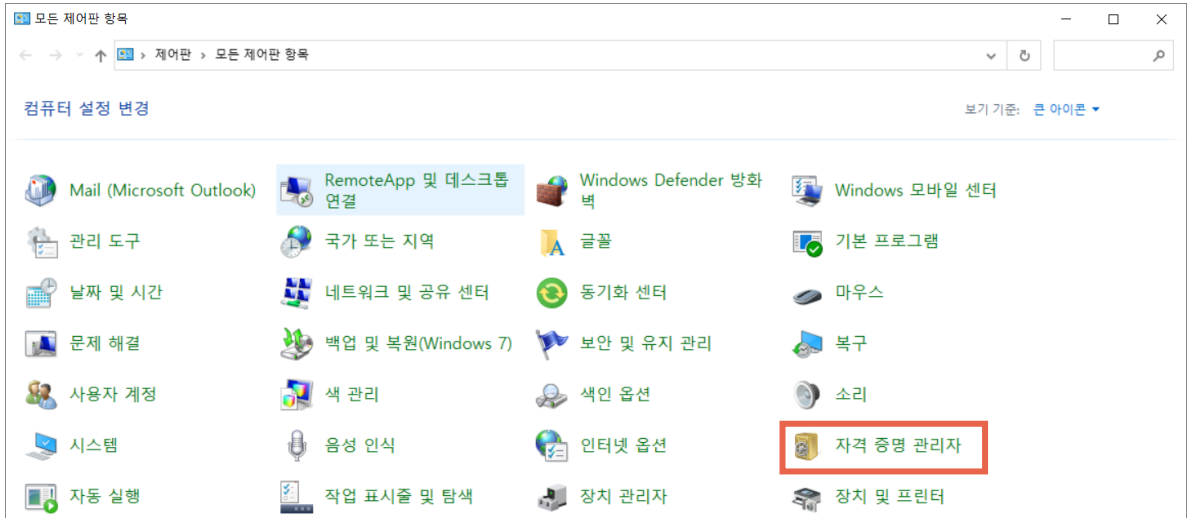


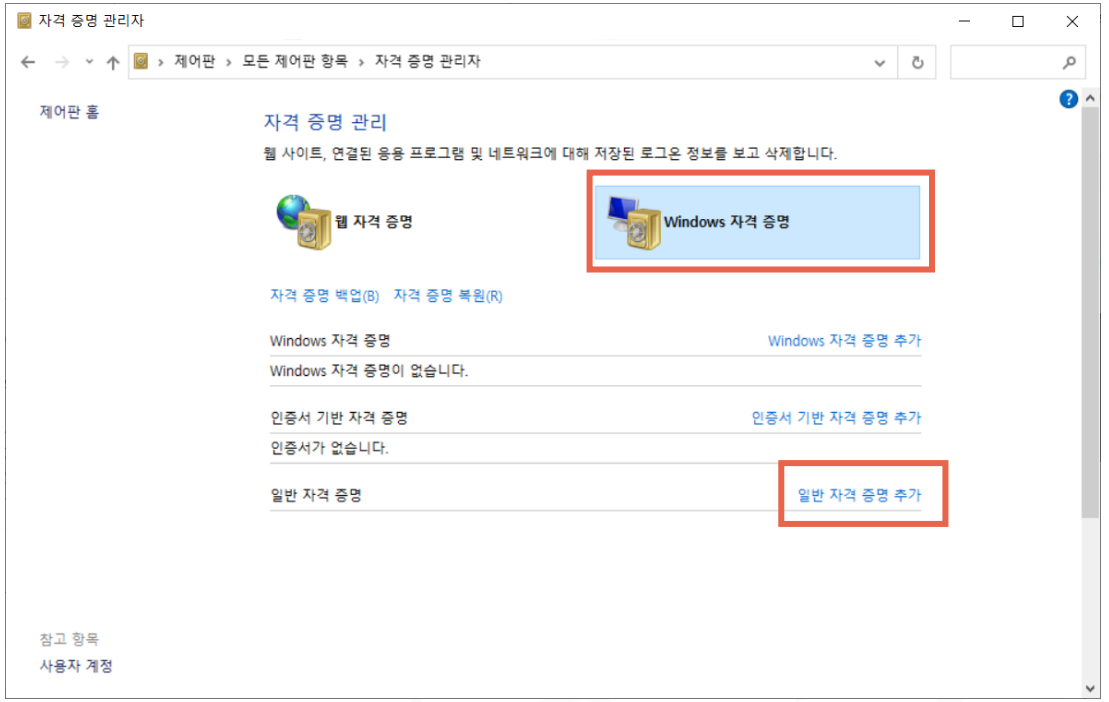


자격 증명 저장하기

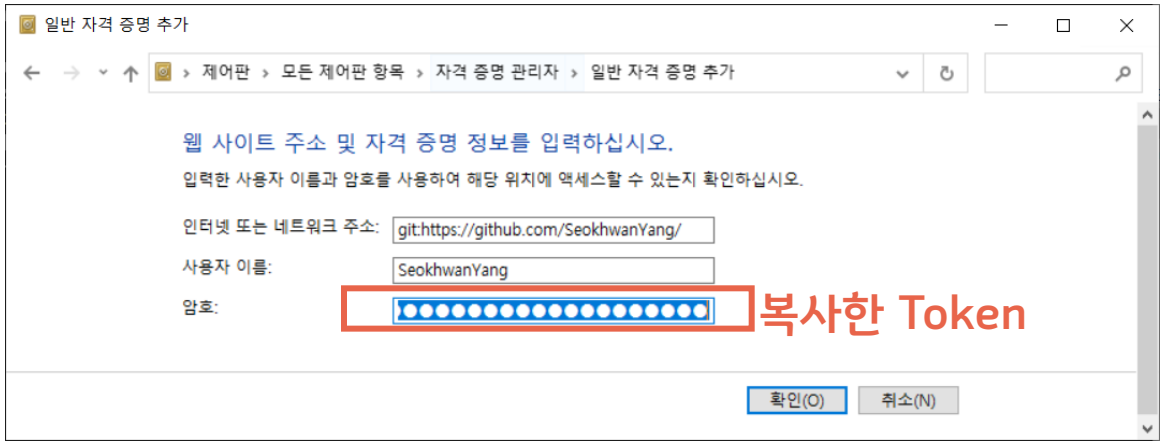
Github

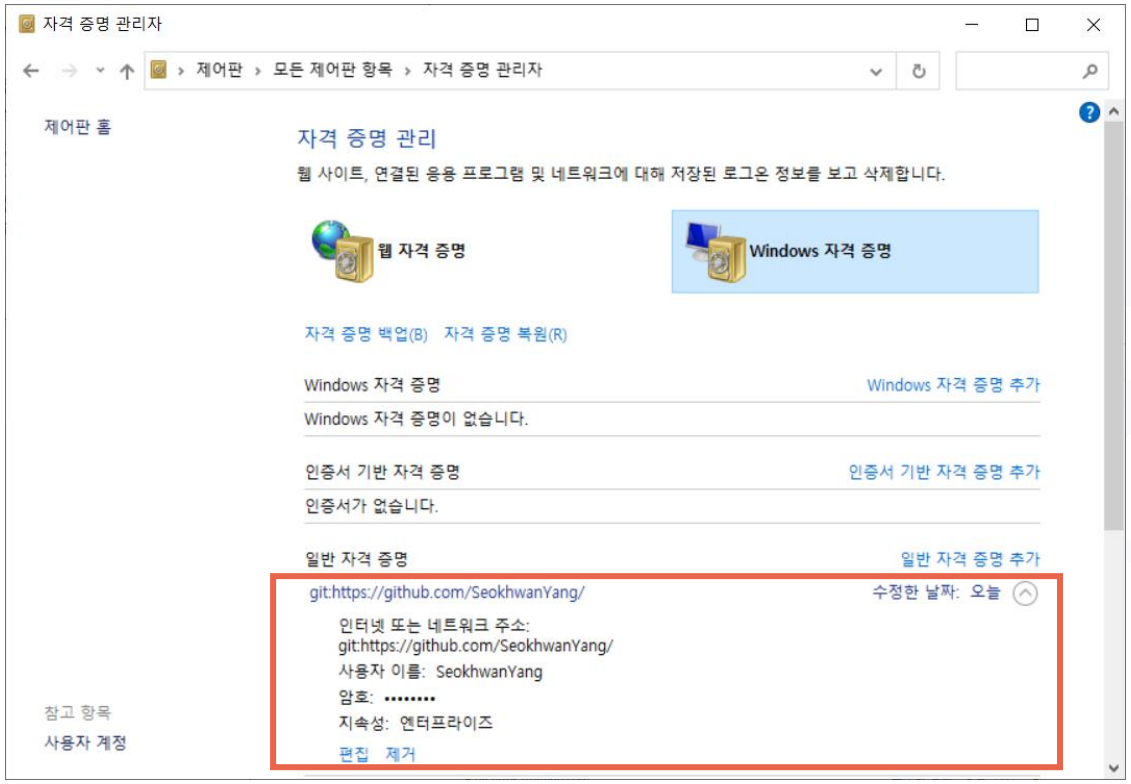
- **Windows의 경우**
  - **제어판 > 자격 증명 관리자 > Windows 자격 증명 > 일반 자격 증명 추가**





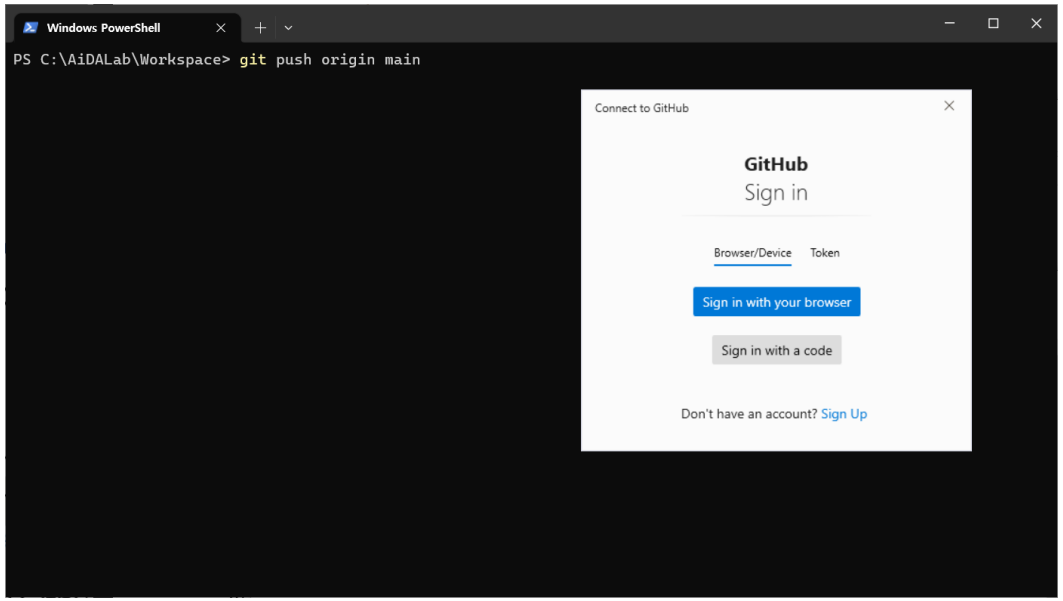
자격 증명 저장하기

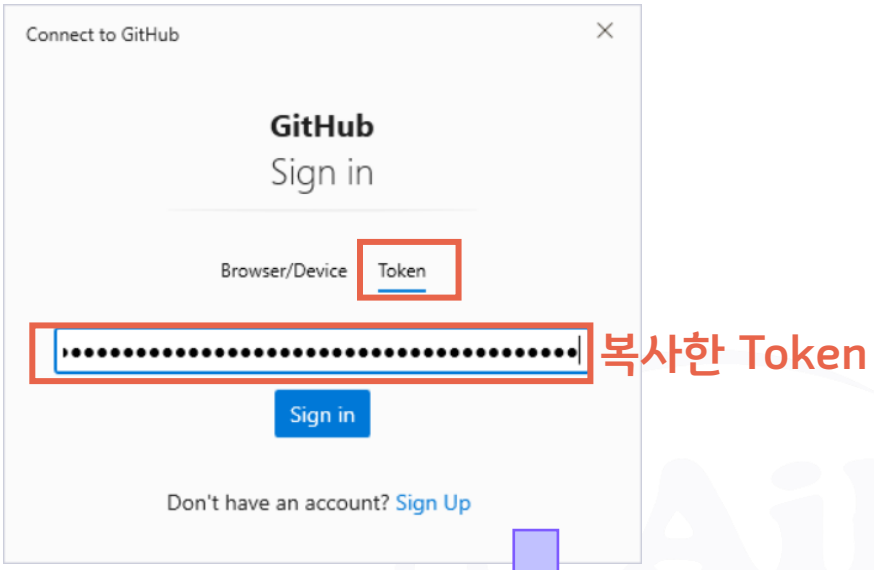

일반 자격 증명 추가
제어판 › 모든 제어판 항목 › 자격 증명 관리자 › 일반 자격 증명 추가
웹 사이트 주소 및 자격 증명 정보를 입력하십시오.
입력한 사용자 이름과 암호를 사용하여 해당 위치에 액세스할 수 있는지 확인하십시오.
인터넷 또는 네트워크 주소:
git:https://github.com/SeokhwanYang/
사용자 이름:
SeokhwanYang
암호:
복사한 Token
확인(O)
취소(N)

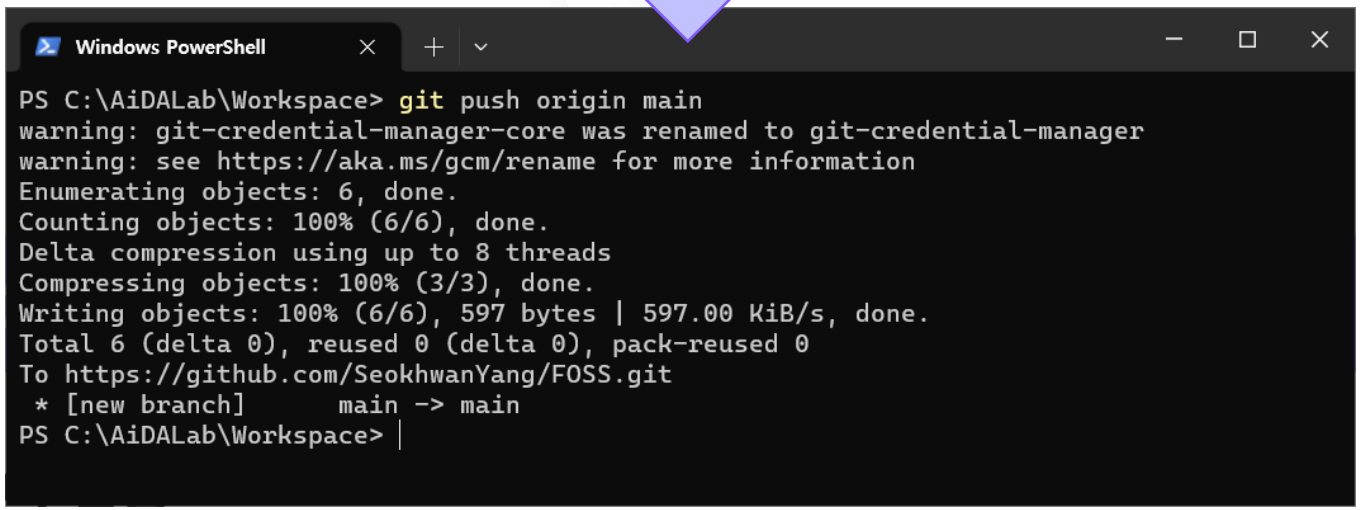

자격 증명 관리자
제어판 › 모든 제어판 항목 › 자격 증명 관리자
제어판 홈
자격 증명 관리
웹 사이트, 연결된 응용 프로그램 및 네트워크에 대해 저장된 로그온 정보를 보고 삭제합니다.
웹 자격 증명
Windows 자격 증명
자격 증명 백업(B)
자격 증명 복원(R)
Windows 자격 증명
Windows 자격 증명 추가
Windows 자격 증명이 없습니다.
인증서 기반 자격 증명
인증서 기반 자격 증명 추가
인증서가 없습니다.
일반 자격 증명
일반 자격 증명 추가
git:https://github.com/SeokhwanYang/
수정한 날짜: 오늘
인터넷 또는 네트워크 주소:
git:https://github.com/SeokhwanYang/
사용자 이름: SeokhwanYang
암호: ••••••••
지속성: 엔터프라이즈
편집
제거
참고 항목
사용자 계정

git push

- 명령어 일람
  - git push origin main

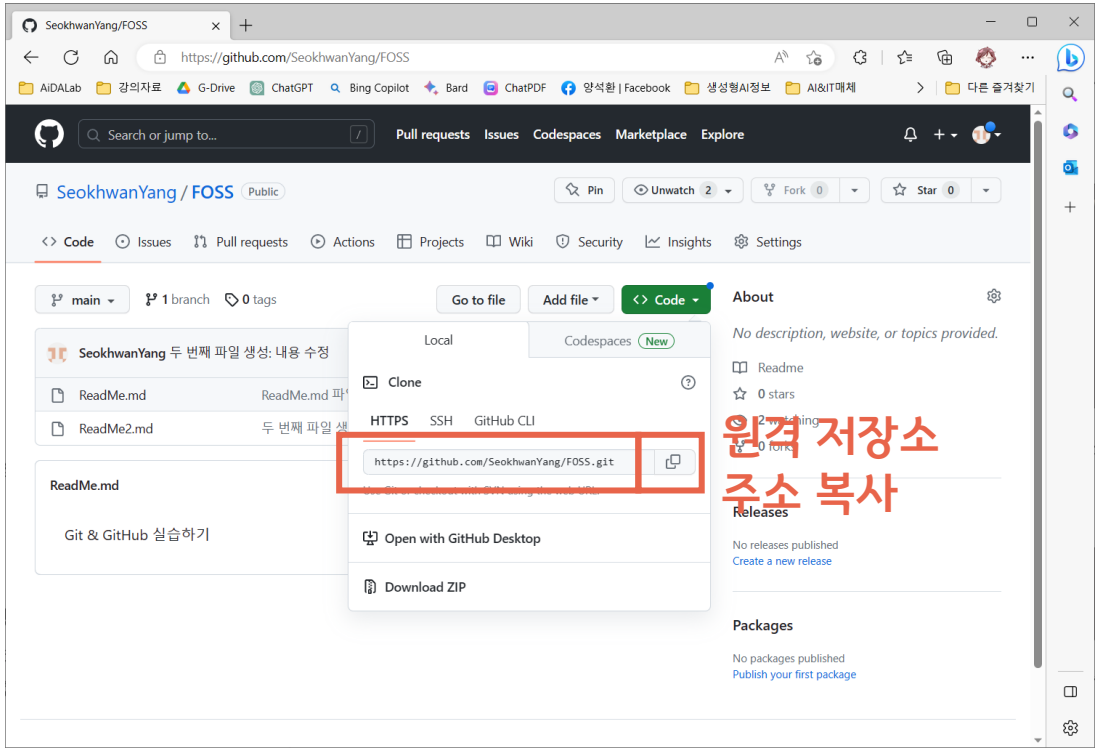


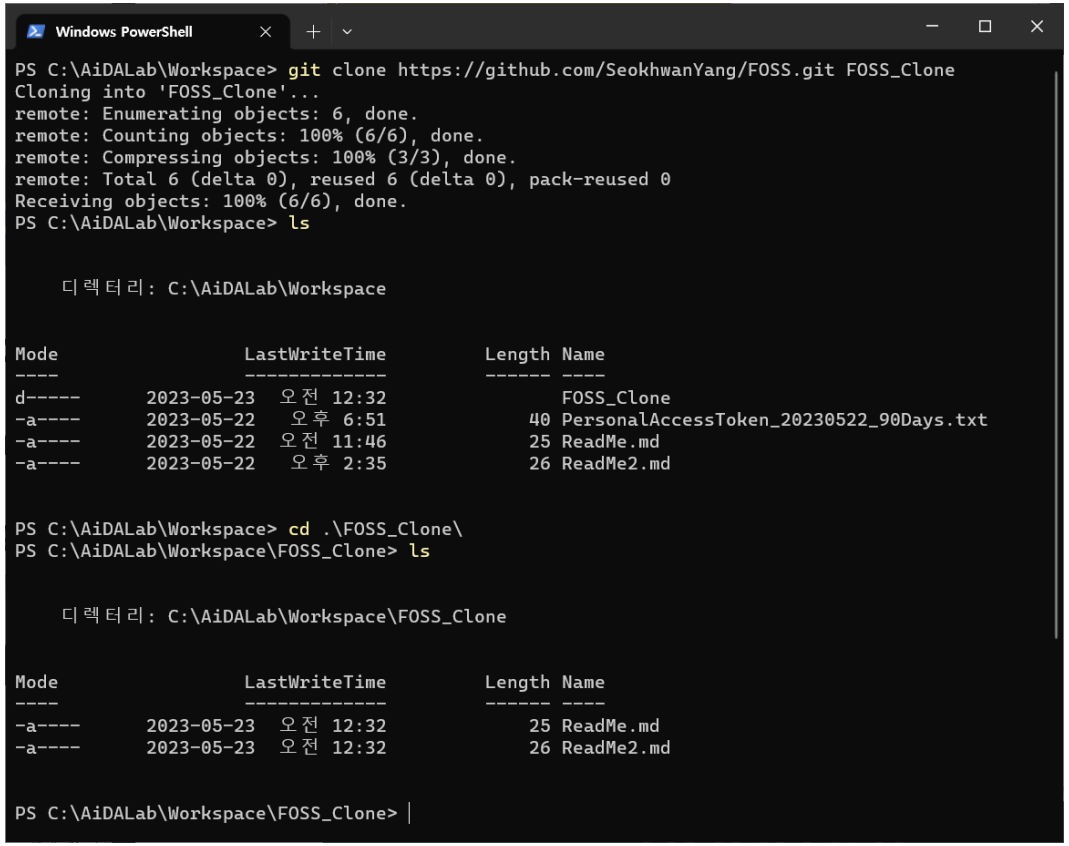




git clone

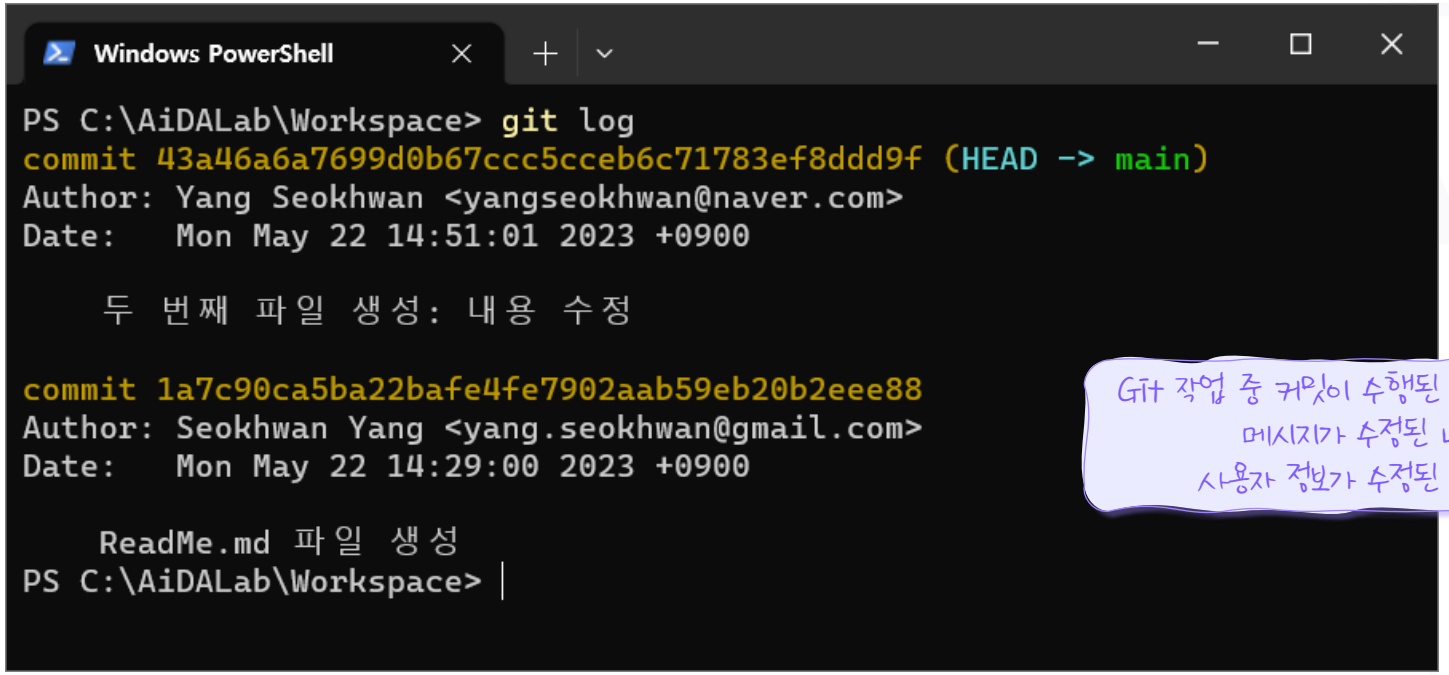
- 기능: 원격 저장소 복제
- 명령어 일람
  - git clone “원격 저장소 주소” “새로운 저장소 이름”





git log

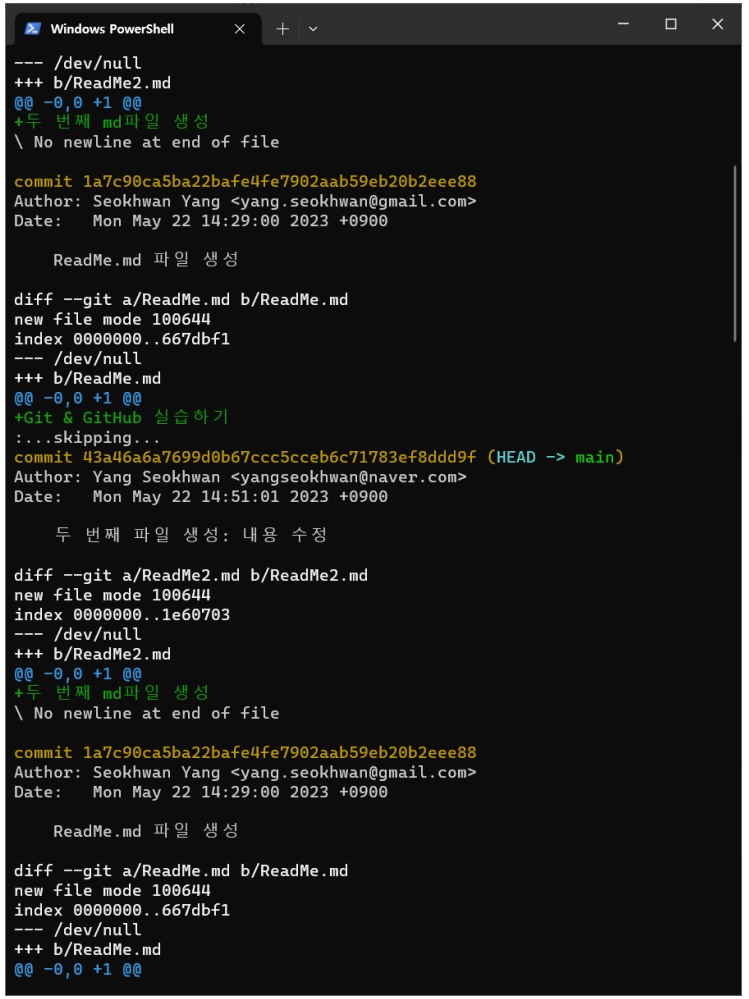
- 기능: git 로그 조회
- 명령어 일람
  - 기본 사용법
    - git log

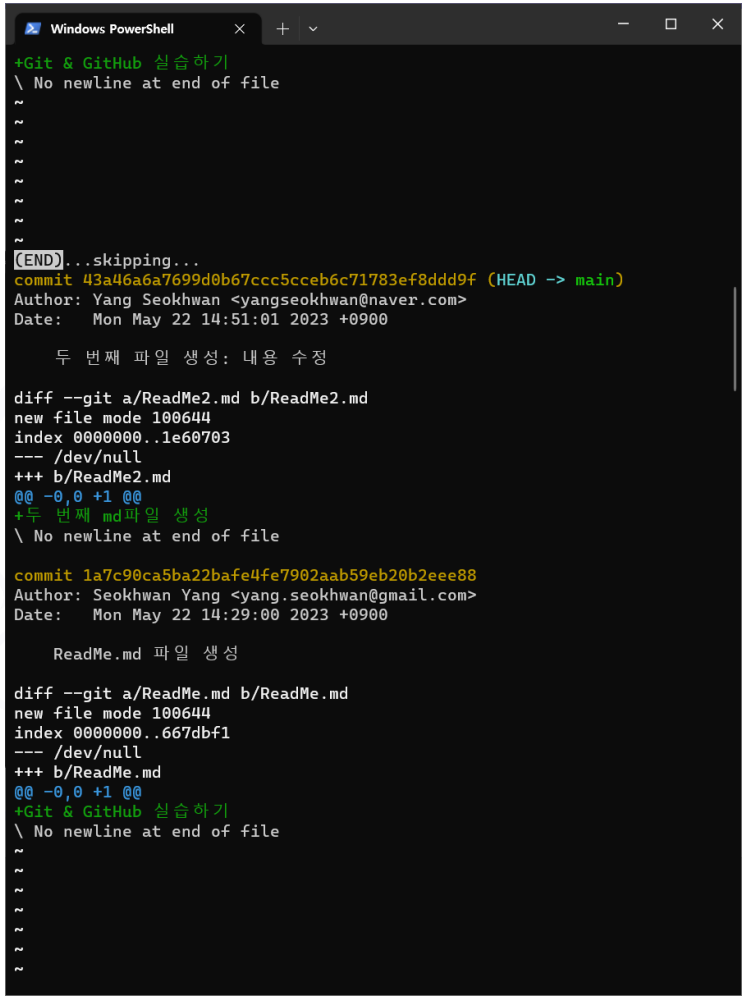


Git 작업 중 커밋이 수행된 로그(기록)을 보여줌
메시지가 수정된 내용이나,
사용자 정보가 수정된 내용이 표시됨

git log

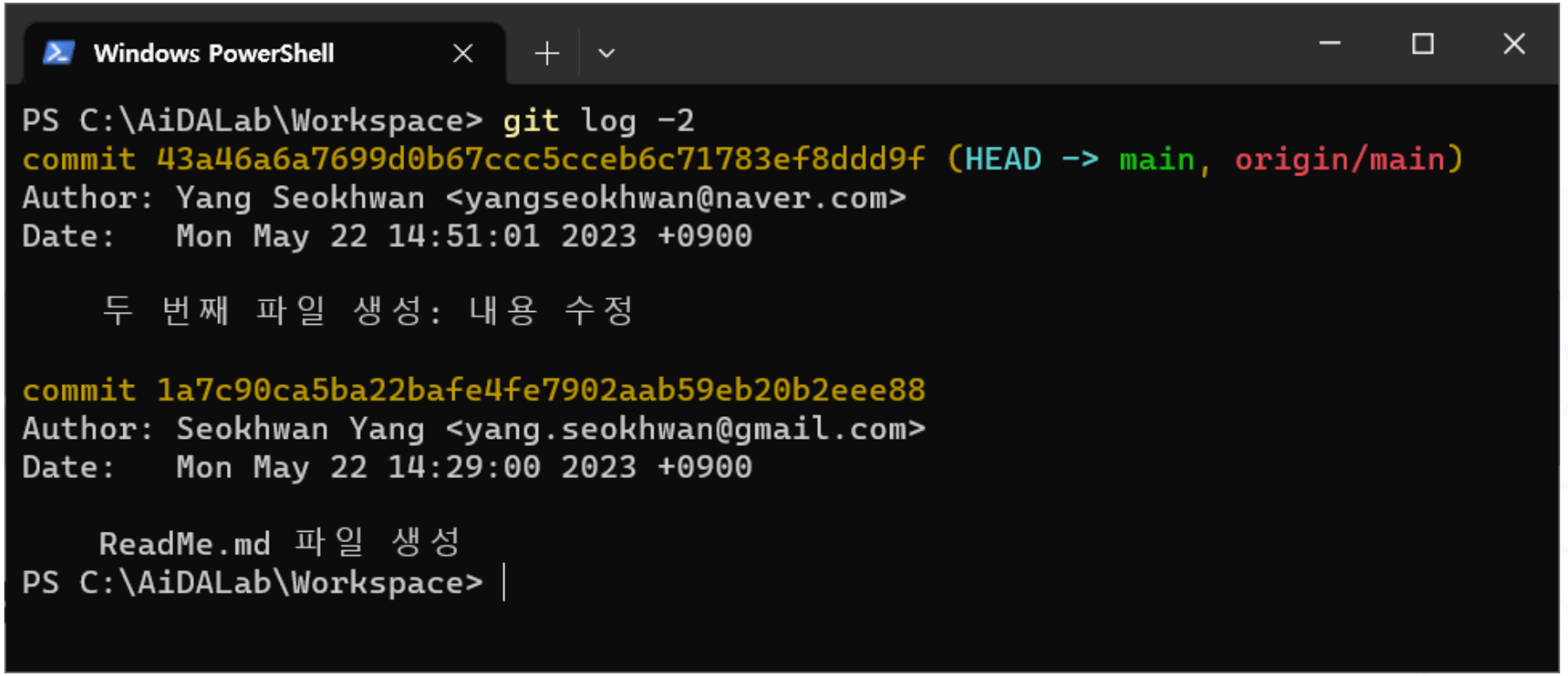
- 커밋 로그를 파일 단위에서 변경 내용을 확인하려면
  - git log -p
  - git log --patch





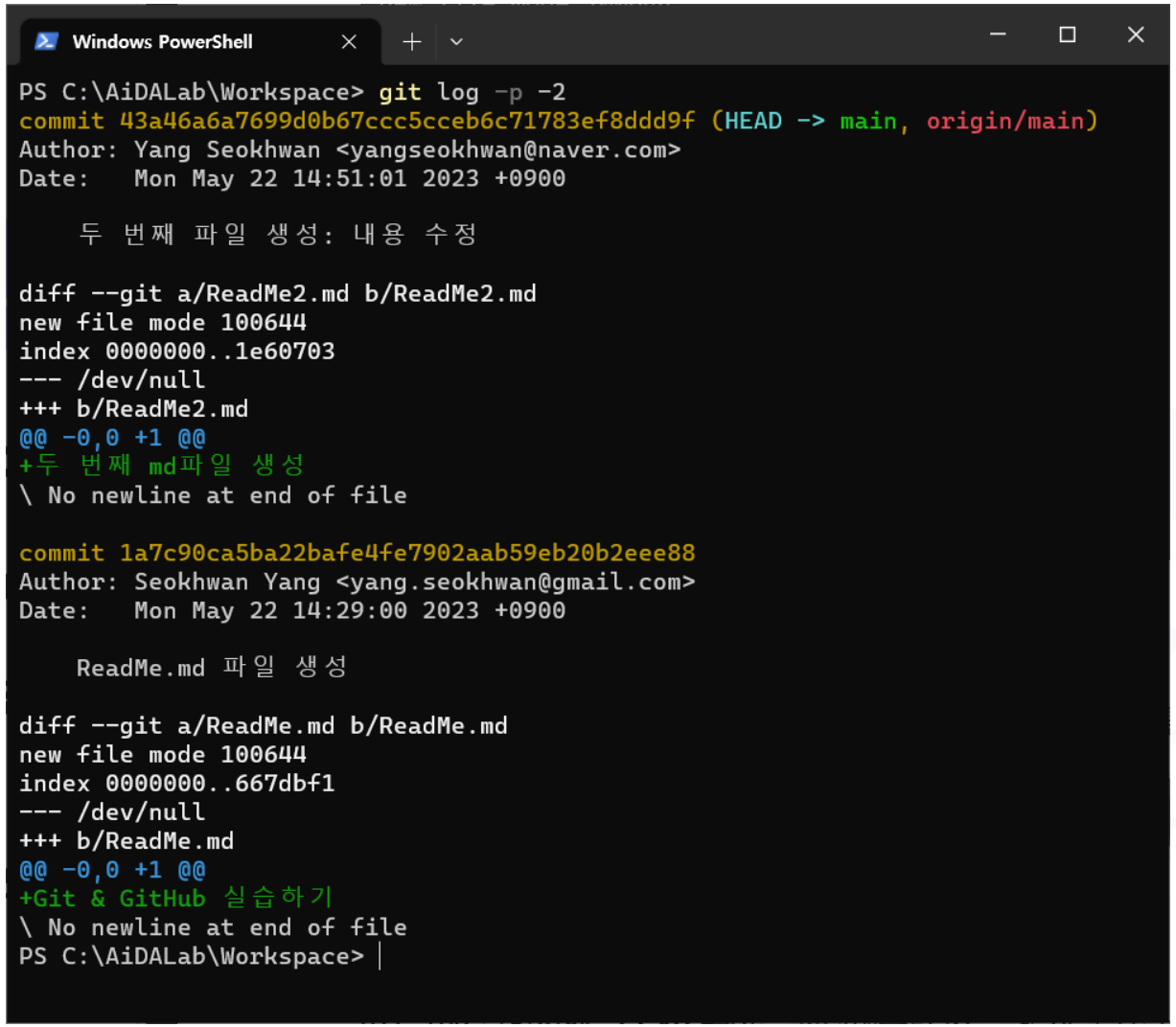
git log

- **최근 몇 개의 커밋 로그를 보려면**
  - git log -{숫자}



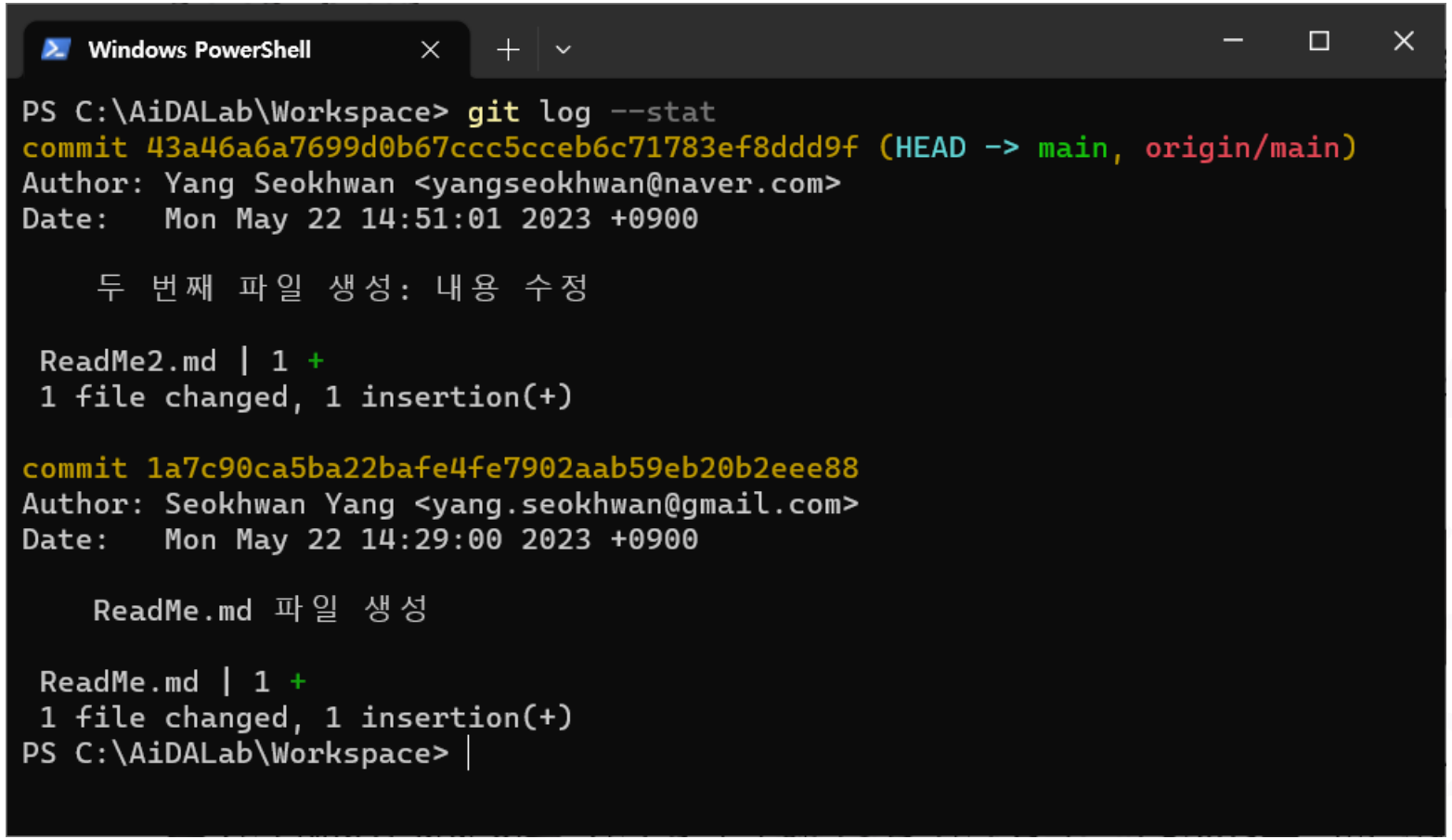
git log

- 기능: git 로그 조회
- 명령어 일람
  - 조건을 결합해서 사용할 수도 있음
    - git log -p -{숫자}



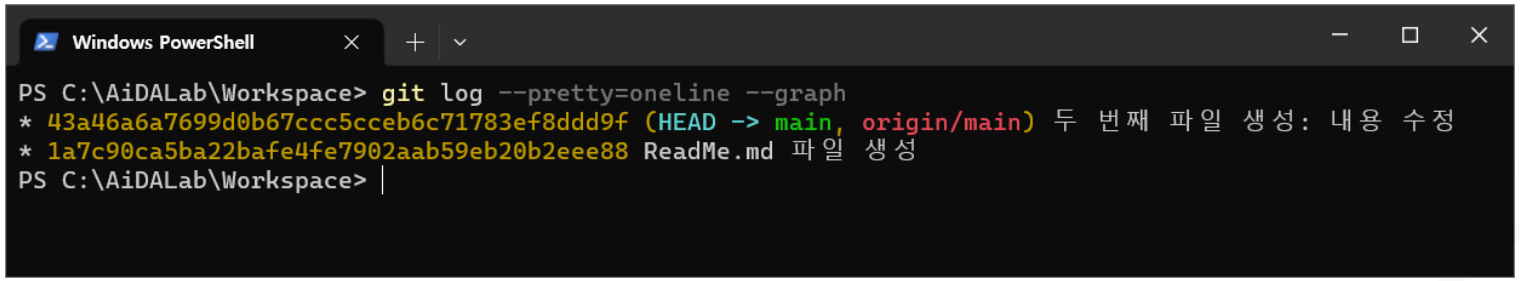
git log

- 각 커밋의 통계 정보를 보려면
  - git log --stat



git log

- **커밋 로그를 보여주는 형식 지정하기**
  - git log --pretty={option}




```
chapter2-basic % git log --pretty=oneline --graph
*   e920f725d586719868ba9811a8f166b9ac30dc08 (HEAD -> main) Merge branch 'test/
local-branch'
|\
| * d2220ec684a1c0960ec85315206dd6ca0e2cead0 (test/local-branch) Change header
* | 2384c658671e4c9ddb1cd439b6fa483efdef2d38 (test/fast-forward) Change title
|/
* d585604d6d30d7a2213fb540afc18b16297ffb29 (origin/test/remote-branch, origin/
test/local-branch, origin/main, test/remote-branch) Add hotline to main page
* 35601018a76e916952fc52258e14262d2b1c039e Change the title of main page
* 8231a202db9a7192c2128d20ca350129b11693ef Add initial files and .gitignore
```

git log

- 기능: git 로그 조회
- 명령어 일람
  - --pretty 옵션에서 사용할 수 있는 출력 형식

해시(Hash)
- 임의의 길이를 가진 데이터를 고정된 길이의 값으로 변환하는 함수. 해시 함수라고도 부름
- 입력 데이터에 대해 일관된 결과를 생성하므로 데이터의 무결성 검사, 암호학에서의 비밀번호 저장, 디지털 서명, 메시지 무결성 검사 등에 활용됨
- 또한 데이터베이스 인덱싱, 데이터 검색, 데이터의 고유성 확인, 압축 알고리즘 등에도 사용됨

형식	설명
%H	커밋 해시
%h	짧은 커밋 해시
%T	트리 해시
%t	짧은 트리 해시
%P	부모 해시
%p	짧은 부모 해시
%s	커밋 요약

형식	설명
%an	저자 이름
%ae	저자 이메일
%ar	저자 상대적 시각
%cn	커미터 이름
%ce	커미터 이메일
%cr	커미터 상대적 시각

Git 기본 명령 요약

명령어	기능	명령 형식
git init	지역 저장소 생성	git init
git config user.name git config user.email	프로젝트 별 지역 사용자 등록	git config user.name “사용자 이름” git config user.email “이메일 주소”
git config --global user.name git config --global user.email	지역 환경의 전체 프로젝트를 위한 사용자 등록	git config --global user.name “사용자 이름” git config --global user.email “이메일 주소”
git remote add	원격 저장소의 주소를 지역 저장소에 등록	git remote add “원격 저장소 주소”
git add	커밋에 포함될 파일 등록	git add “파일명”
git status	현재 프로젝트의 파일 상태 확인	git status
git commit	새로운 커밋 생성	git commit
	기존 커밋 수정	git commit --amend
	기존 커밋 저자 수정	git commit --amend --author “사용자 이름 <이메일 주소>”
git log	커밋 내역 확인	git log
	커밋 내역을 가시적/그래프 표현으로 확인	git log --pretty=oneline --graph
git push	원격 저장소에 커밋 반영	git push “원격 저장소 식별자” “브랜치”
git clone	원격 저장소 복제	git clone “원격 저장소 주소”