

Git & Github 가이드

버전 관리를 위한 필수 도구 이해와 활용



목차

1. Git과 GitHub 소개

- 형상 관리 시스템이란?
- Git vs GitHub 차이점
- 왜 필요한가? (버전 관리의 중요성)

2. Git 기본 개념

- 로컬 저장소, 원격 저장소
- Working Directory / Staging Area / Commit
- 브랜치와 병합(Merge)

3. Git 기본 명령어 실습

- Git 설치 및 설정
- `git init`, `add`, `commit`, `status`, `log`
- 브랜치 생성 및 병합

4. GitHub 시작하기

- GitHub 계정 만들기
- 원격 저장소 만들기
- GitHub와 로컬 저장소 연동
(`remote`, `push`, `pull`)

5. GitHub 협업 기능 이해

- Pull Request
- 이슈(Issue)와 프로젝트 보드

6. 정리 및 꿀팁

- 자주 쓰는 명령어 요약
- Git 사용 시 주의할 점
- GUI 툴 (GitHub Desktop) 소개

01 Git과 Github 소개

- 형상 관리 시스템이란?
- Git vs GitHub 차이점
- 왜 필요한가? (버전 관리의 중요성)

Git과 Github 소개

형상 관리 시스템(Version Control System, VCS)

- 소스 코드, 문서, 파일 등의 변경 이력을 관리하고, 여러 사람이 협업할 수 있도록 도움을 주는 도구
- 주요 기능:
 - 변경 이력 추적
 - 특정 시점으로 복구
 - 협업 지원
- Git vs GitHub:
 - Git : 분산 형상 관리 시스템. 로컬에서 버전 관리 가능.
 - GitHub : Git 저장소를 호스팅하는 클라우드 플랫폼. 협업 및 코드 공유에 특화.
- 왜 필요한가? (버전 관리의 중요성)
 - 변경 이력 관리로 실수 복구 가능
 - 협업 시 충돌 방지 및 효율적 관리
 - 코드 품질 향상 및 작업 투명성 제공

02 Git 기본 개념

- 로컬 저장소, 원격 저장소
- Working Directory / Staging Area / Commit
- 브랜치와 병합(Merge)

Git 기본 개념

Git 기본 개념

- 로컬 저장소, 원격 저장소
 - 로컬 저장소 : 내 컴퓨터에 저장된 Git 저장소
 - 원격 저장소 : GitHub와 같은 클라우드에 저장된 저장소
- Working Directory / Staging Area / Commit
 - Working Directory : 현재 작업 중인 파일
 - Staging Area : 커밋 준비 단계
 - Commit : 변경 사항을 저장소에 기록
- 브랜치(Branch)와 병합(Merge)
 - 브랜치(Branch) : 독립적인 작업 공간
 - 병합(Merge) : 브랜치의 변경 사항을 통합

03 Git 기본 명령어 실습

- Git 설치 및 설정
- `git init`, `add`, `commit`, `status`,
`log`
- 브랜치 생성 및 병합

Git 설치 (1)

Git 설치

- 링크 : <https://git-scm.com/downloads>
- 시스템 비트에 맞게 다운 (Window)



Download for Windows

Click here to download the latest (2.49.0) 64-bit version of Git for Windows. This is the most recent **maintained build**. It was released **32 days ago**, on 2025-03-17.

Other Git for Windows downloads

Standalone Installer

32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Portable ("thumbdrive edition")

32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

시스템 종류 64비트 운영 체제, x64 기반 프로세서

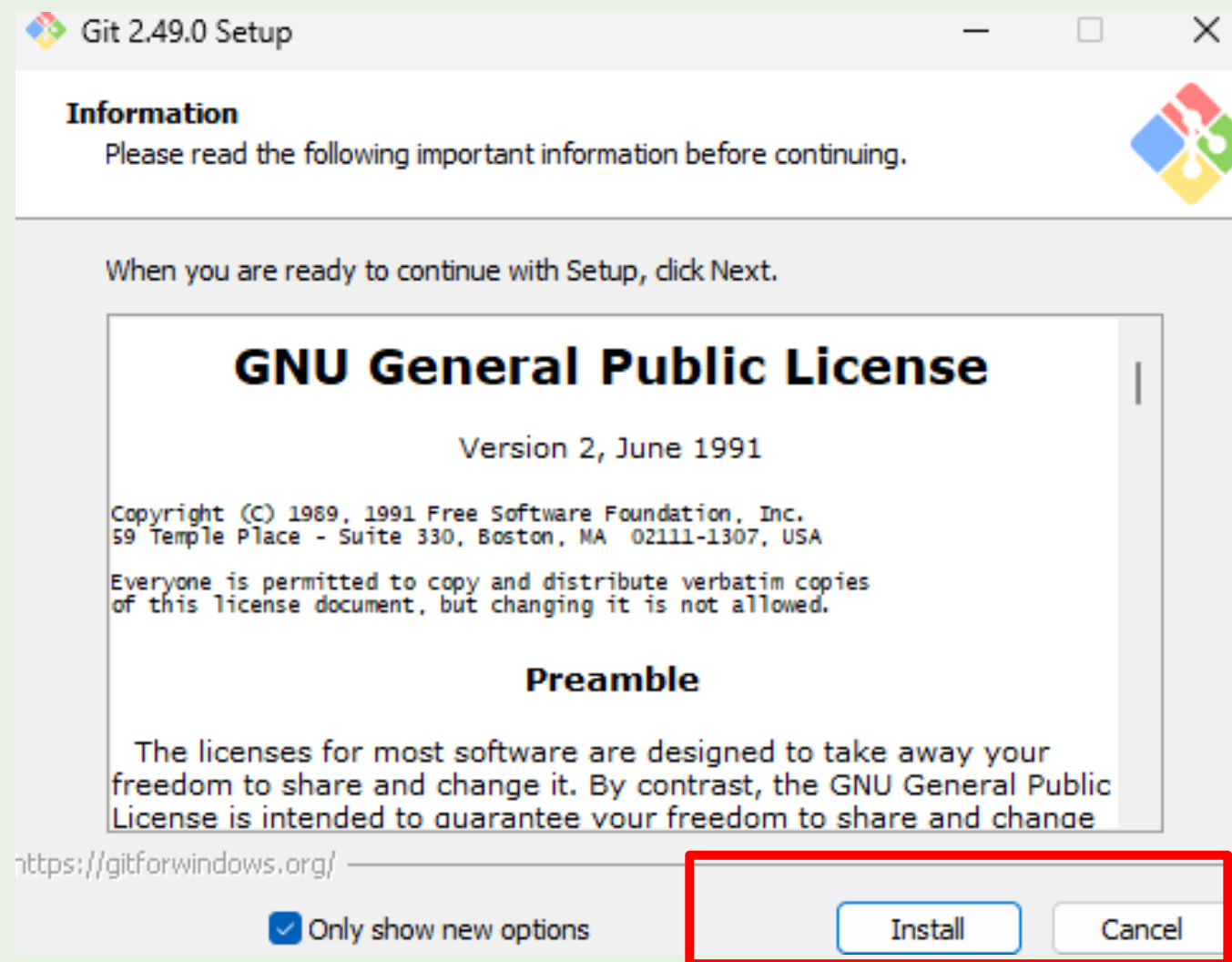
Git 설치 (2)

Git 설치



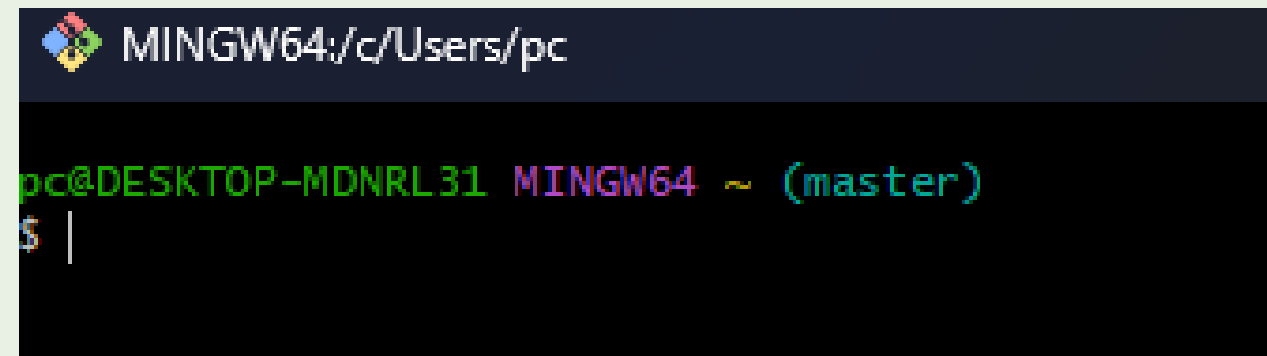
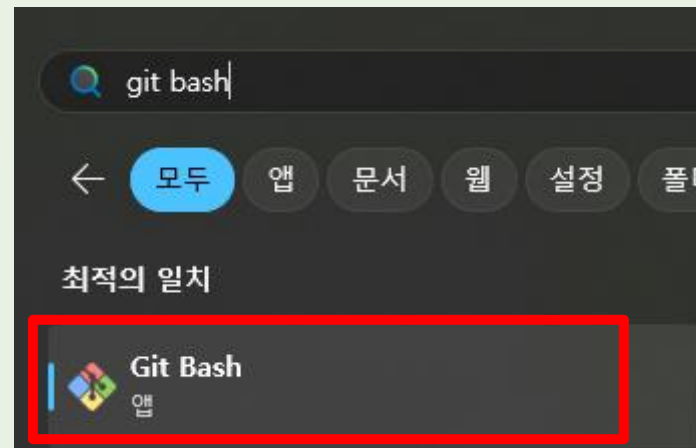
Git 설치 (2)

Git 설치



Git 설정 (1)

Git bash 열기



Git 설정 (2)

사용자 정보 설정

- `git config --global user.name "Your Name"`
- `git config --global user.email "your.email@example.com"`

```
pc@DESKTOP-MDNRL31 MINGW64 ~ (master)
$ git config --global user.name "유승진"

pc@DESKTOP-MDNRL31 MINGW64 ~ (master)
$ git config --global user.email "dbtmdwlstest@gmail.com"
```

Git 저장소 초기화

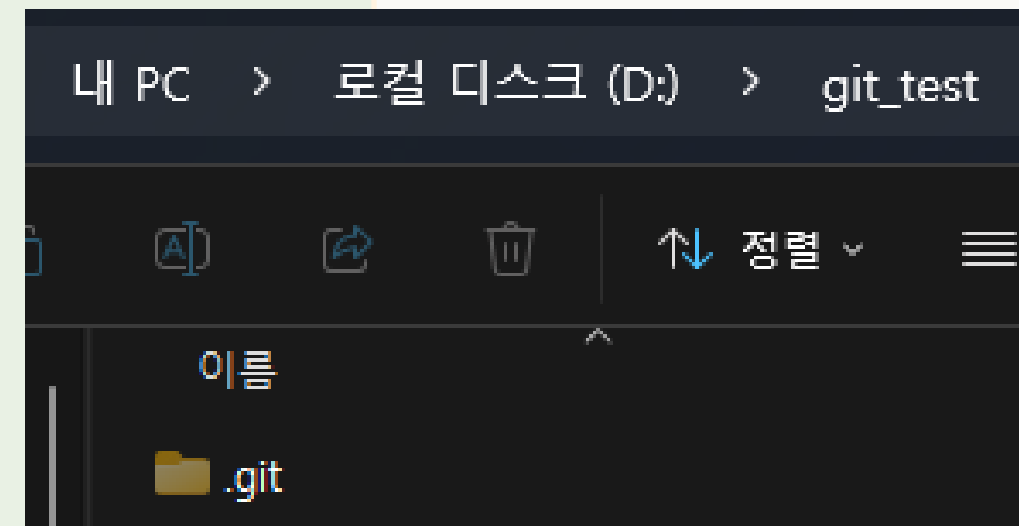
작업 폴더로 이동 혹은 작업 폴더 생성

- `cd` : change directory 폴더 변경
- `mkdir` : make directory 폴더 생성
- `git init` : git 초기화 (초기 설정)

```
pc@DESKTOP-MDNRL31 MINGW64 /d
$ mkdir git_test

pc@DESKTOP-MDNRL31 MINGW64 /d
$ cd git_test

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test
$ git init
Initialized empty Git repository in D:/git_test/.git/
```



Git에 파일 추가

Git에 파일 추가

- `echo "# 내 첫 Git 프로젝트" > README.md`
혹은 텍스트 파일 생성

- 문법

`git add <파일명> # 특정 파일`

`git add . # 모든 파일`

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ echo "# 내 첫 Git 프로젝트" > README.md

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git add README.md
```

커밋 (commit)

커밋 (commit)

- 커밋은 파일 수정, 삭제, 추가, 설정 변경 등 작업한 내용을 Git 저장소에 하나의 이력으로 남기는 것입니다.
- 문법

`git commit -m "커밋 메시지"`

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git commit -m "첫 깃 커밋"
[master (root-commit) 1a4a846] 첫 깃 커밋
1 file changed, 1 insertion(+)
create mode 100644 README.MD
```

상태 확인 (git status)

상태 확인

- git status는 현재 작업 중인 파일의 상태를 확인할 수 있는 명령어

- 문법

git status

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.MD
```

변경된 사항이 있을 경우

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git status
On branch master
nothing to commit, working tree clean
```

변경된 사항이 없는 경우

커밋 로그 확인 (git log)

상태 확인

- git log는 커밋 이력을 확인할 수 있는 명령어
- 프로젝트가 진행되면서 누가, 언제, 무엇을 변경했는지 알 수 있다.
- 문법

git log

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git log
commit 1a4a846e374abc7573e95afa015bfea2c62f7ea3 (HEAD -> master)
Author: 유승진 <dbtmdwlstest@gmail.com>
Date:   Fri Apr 18 12:54:02 2025 +0900
```

첫 커밋

브랜치 (branch)

브랜치 (branch) 란?

- 브랜치는 독립적인 작업 공간
- 기존 코드를 보존한 채, 새로운 기능을 개발하거나 수정할 수 있도록 도와주는 기능
- 비유:
 - 책의 복사본을 만들어서 그 위에 메모하거나 수정하는 것과 비슷하다.
 - 기존 코드(main 브랜치)는 안전하게 두고, 새로운 기능(feature), 버그 수정(hotfix) 등을 별도 브랜치에서 작업.
- 브랜치가 유용한 이유
 - 새로운 기능 : 개발 메인 코드에 영향 없이 개발 가능
 - 버그 수정 : 안정 버전에서 따로 수정 후 반영 가능
 - 실험 : 여러 가지 코드를 시도하고 실패해도 걱정 없음
 - 협업 : 개발자마다 다른 브랜치에서 작업 후 병합 가능

브랜치 확인

브랜치 확인

- 현재 브랜치를 확인하는 명령어
- 문법

git branch

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch
* master
```

초록색 글자가 현재 작업하고 있는 환경 (master)

브랜치 생성

브랜치 생성

- 브랜치 생성하는 명령어

- 문법

`git branch <브랜치이름>`

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch branch_test

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch
  branch_test
* master
```

브랜치 전환 (1)

브랜치 전환

- 브랜치 전환하는 명령어

- 문법

`git checkout <브랜치이름>`

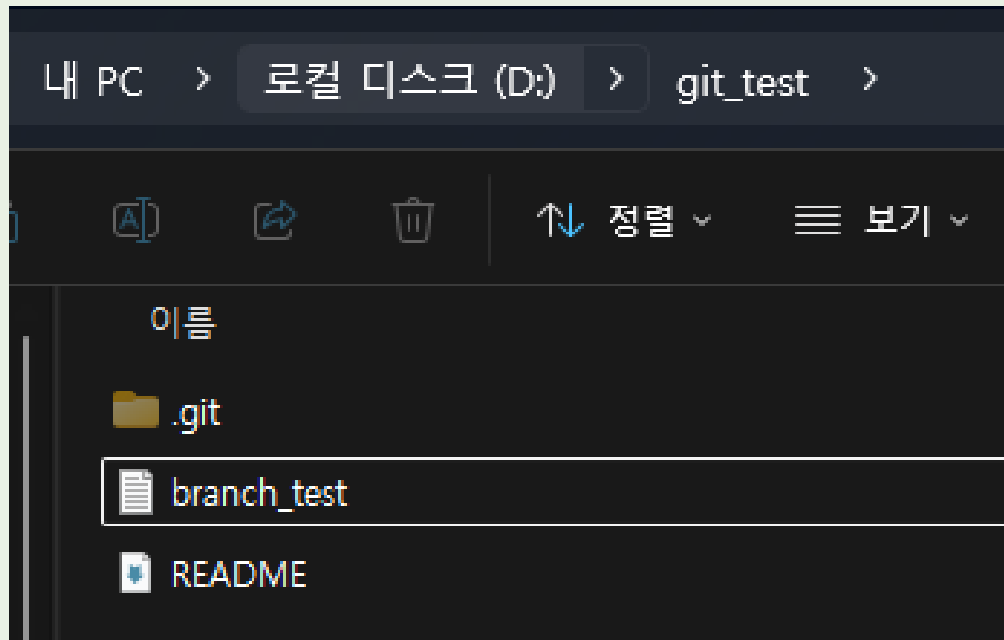
```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git checkout branch_test
Switched to branch 'branch_test'

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git branch
* branch_test
  master
```

브랜치 전환 (2)

브랜치 전환

- 전환한 브랜치 작업 환경에 테스트 파일 생성
- ls : 폴더 안의 파일 리스트



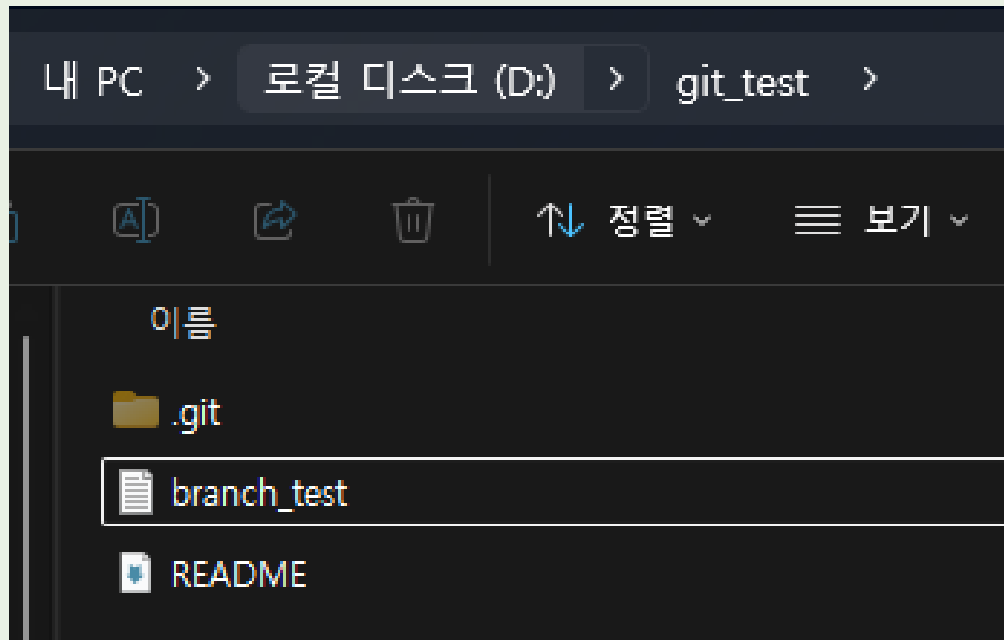
```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git branch
* branch_test
  master

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ ls
README.MD  branch_test.txt
```

브랜치 전환 (3)

브랜치 전환

- 전환한 브랜치 작업 환경에 테스트 파일 생성 후 커밋까지 진행
- ls : 폴더 안의 파일 리스트



```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git branch
* branch_test
  master

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ ls
README.MD  branch_test.txt
```

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git add .

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git status
On branch branch_test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   branch_test.txt
```

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git commit -m "브랜치 테스트"
[branch_test f02150e] 브랜치 테스트
1 file changed, 1 insertion(+)
create mode 100644 branch_test.txt
```

브랜치 전환 (4)

브랜치 전환

- master 브랜치로 변경 후 폴더 내 파일 확인

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (branch_test)
$ git checkout master
Switched to branch 'master'

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ ls
README.MD
```


병합 (merge)

병합 (merge)

- merge는 두 갈래로 나뉜 작업을 다시 하나로 합치는 것
- 현재의 브랜치에 다른 브랜치의 내용을 가져오는 것 * 혼동 주의
- 문법

`git merge <브랜치이름>`

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git merge branch_test
Updating 1a4a846..702150e
Fast-forward
 branch_test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 branch_test.txt

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ ls
README.md  branch_test.txt
```

브랜치 삭제

브랜치 삭제

- 브랜치를 삭제하는 명령어
- 작업을 다하여 사용하지 않는 명령어는 삭제하는 것이 브랜치 관리에 좋다.
- 문법

git branch -d <브랜치이름>

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch
  branch_test
* master

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch -d branch_test
Deleted branch branch_test (was f02150e).

pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git branch
* master
```

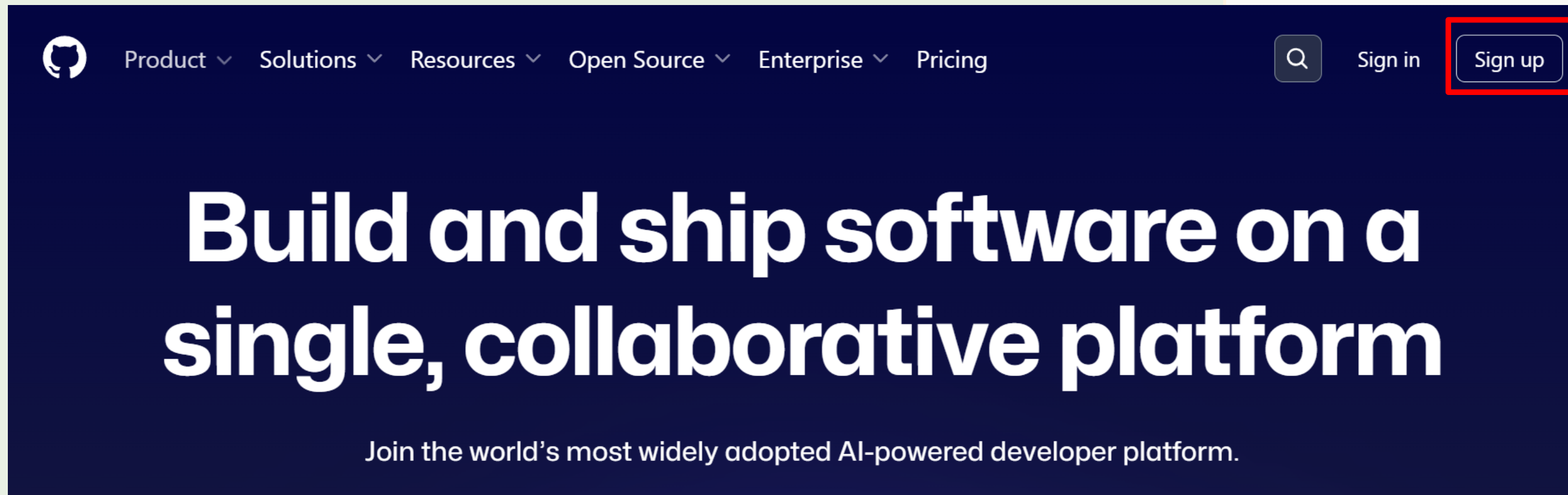
04 GitHub 시작하기

- GitHub 계정 만들기
- 원격 저장소 만들기
- GitHub와 로컬 저장소 연동
(`remote`, `push`, `pull`)

GitHub 계정 만들기

GitHub 계정 만들기

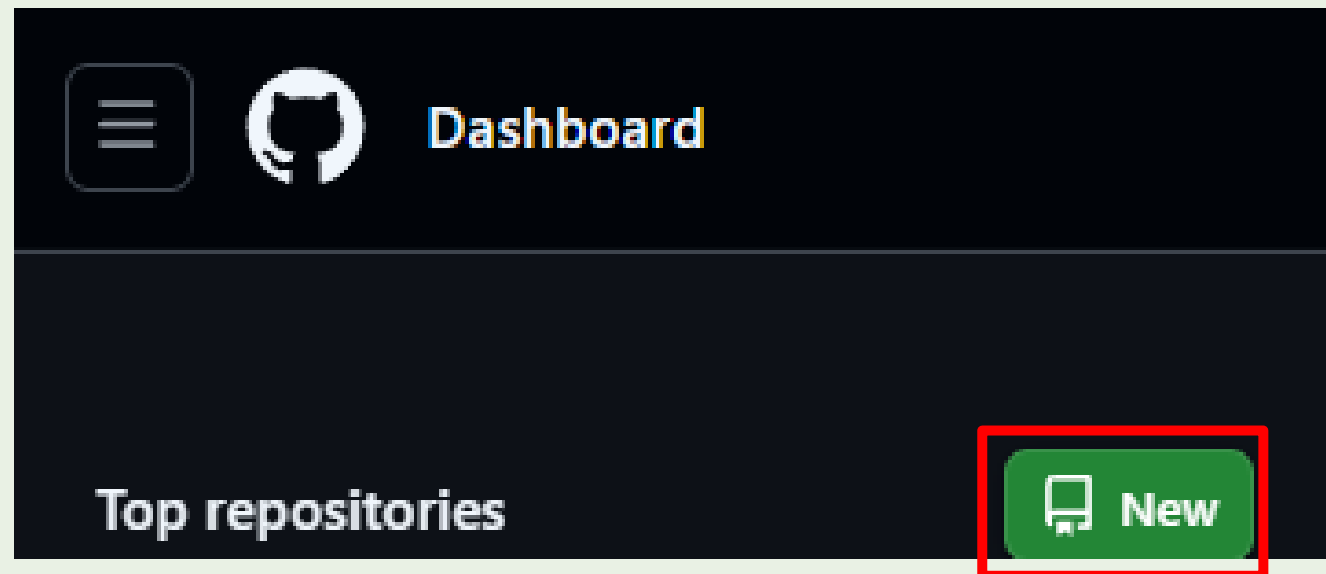
- 링크 : <https://github.com/>



원격 저장소 만들기

원격 저장소 만들기

- Top repositories 옆의 New 버튼 클릭
- 상세 정보 입력 후 Create repository 버튼 클릭



Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * sjgopromer2025 / Repository name * git_test
git_test is available.

Great repository names are short and memorable. Need inspiration? How about [bug-free-barnacle](#) ?

Description (optional)

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

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

Initialize this repository with:

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

Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: None
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

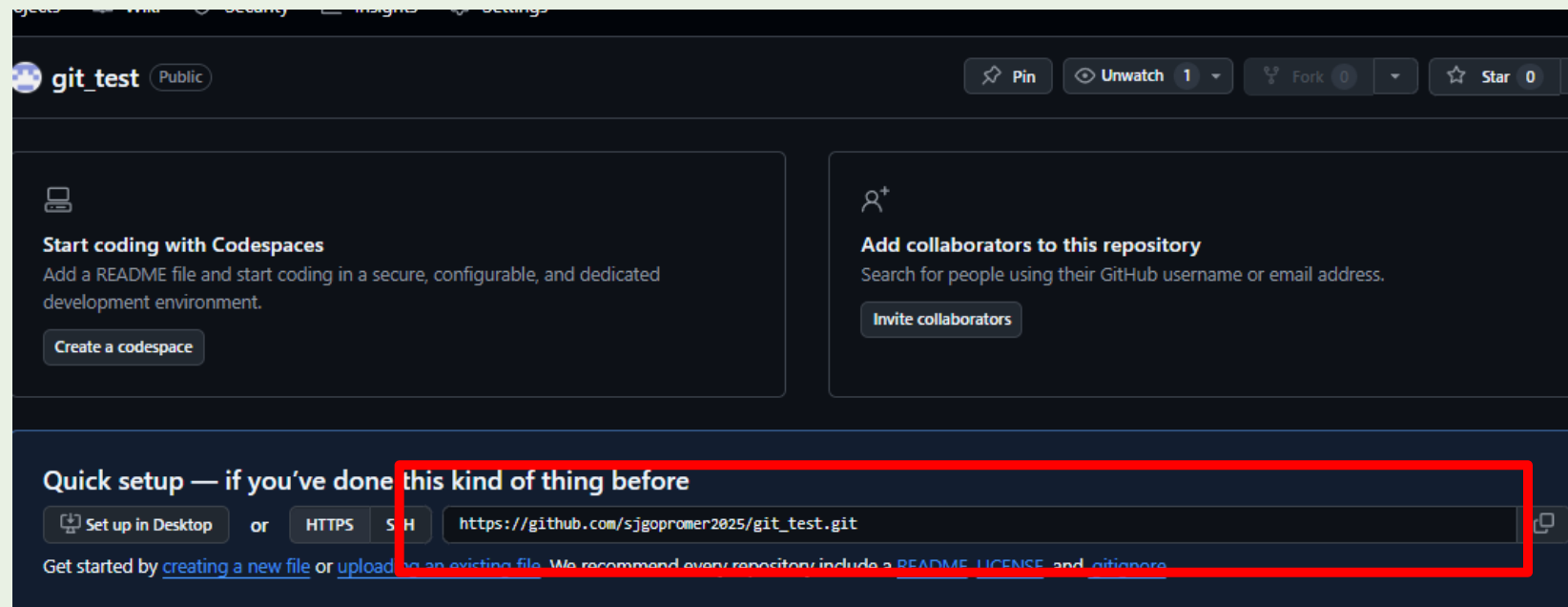
① You are creating a public repository in your personal account.

Create repository

Github와 로컬 저장소 연동 (1)

Github와 로컬 저장소 연동

- 저장소 링크 복사



- 혹은 현재 주소창의 링크를 복사하고 .git을 붙여서 진행

Github와 로컬 저장소 연동 (2)

Github와 로컬 저장소 연동 (remote)

- 로컬 저장소 경로에서 명령어 실행

- 문법

`git remote add origin <저장소 URL>`

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git remote add origin https://github.com/sjgopromer2025/git_test.git
```

로컬 변경 사항 업로드 (push)

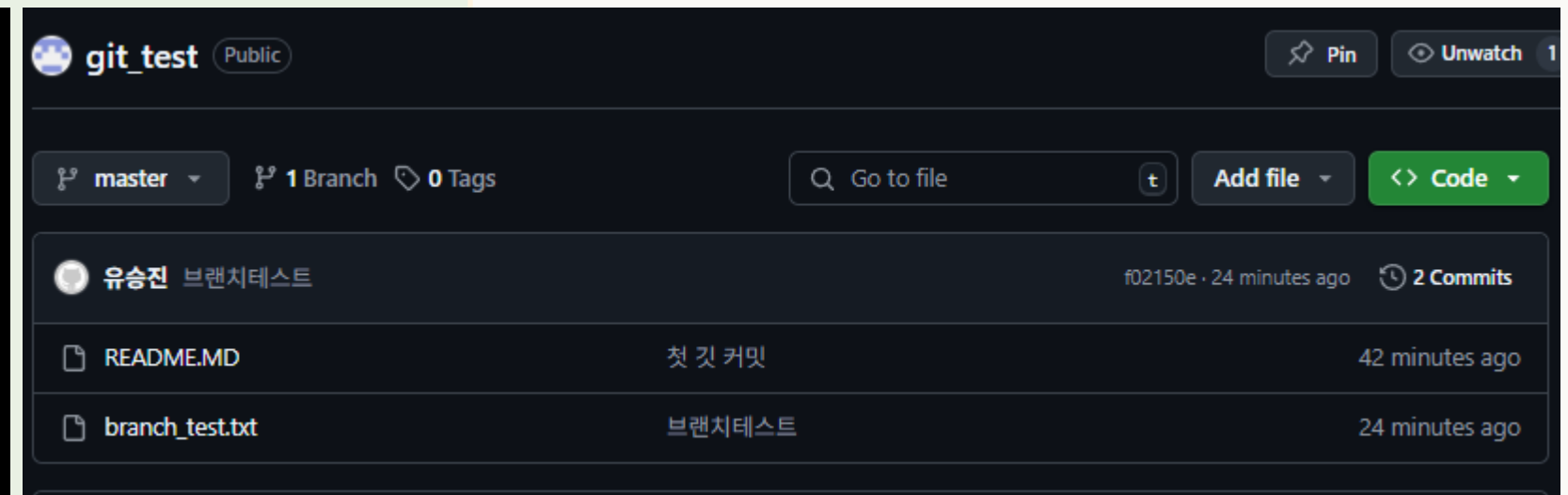
로컬 변경 사항 업로드

- 로컬에서 작업한 내용을 GitHub(원격 저장소)에 업로드(=푸시)하는 명령어

- 문법

```
git push -u origin master
```

```
pc@DESKTOP-MDNRL31 MINGW64 /d/git_test (master)
$ git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 28 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 548 bytes | 548.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/sjgopromer2025/git_test.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```



원격 저장소 변경 사항 가져오기 (pull)

로컬 변경 사항 업로드

- GitHub(원격 저장소)의 master 브랜치에서 최신 변경사항을 내 로컬 저장소로 가져오는 명령어
- 문법
`git pull origin master`

05

GitHub 협업 기능 이해

- Pull Request
- 이슈(Issue)와 프로젝트 보드

Pull Request

Pull Request란?

- Pull Request (PR)는 GitHub에서 협업을 할 때, 다른 사람의 저장소에 내가 한 변경 사항을 반영하도록 요청하는 과정입니다.
- 주로 기능 추가, 버그 수정 등의 작업을 다른 사람과 공유하고, 검토와 병합(Merge)을 요청하는 데 사용됩니다.
- PR의 단계
 1. 브랜치 생성
 2. 작업 후 커밋
 3. 원격 저장소에 푸시
 4. Pull Request 생성
 5. 코드 리뷰 및 승인

이슈(Issue)

이슈(Issue)란?

- GitHub에서 이슈는 버그, 기능 제안, 질문 등 작업할 항목을 기록하고 추적하는 데 사용하는 도구입니다.
- 예시:
 - 버그 리포트: "회원가입 시 오류 발생"
 - 기능 제안: "다크 모드 추가"
 - 질문: "API 요청은 어디에서 처리하나요?"
- 작성 내용:
 - 제목(무엇에 대한 이슈인지)
 - 상세 설명
 - 라벨(label), 담당자(assignee), 마일스톤(milestone) 설정 가능
 - 여러 명이 댓글로 의견을 주고받으며 이슈를 함께 해결

이슈(Issue)

IssuesPull requestsActionsProjectsWikiSecurityInsightsSettings

Create new issue

Add a title *

이슈 테스트

Add a description

WritePreview

H B I | ☰ <> 🔗 | ☰ ☰ ☰ | @ ↗ ↶ ↵

이슈 테스트


📎 Paste, drop, or click to add files

☐ Create more

Cancel

Create ↗

Assignees

 sjgopromer2025

Labels

bug

Projects

No projects

Milestone

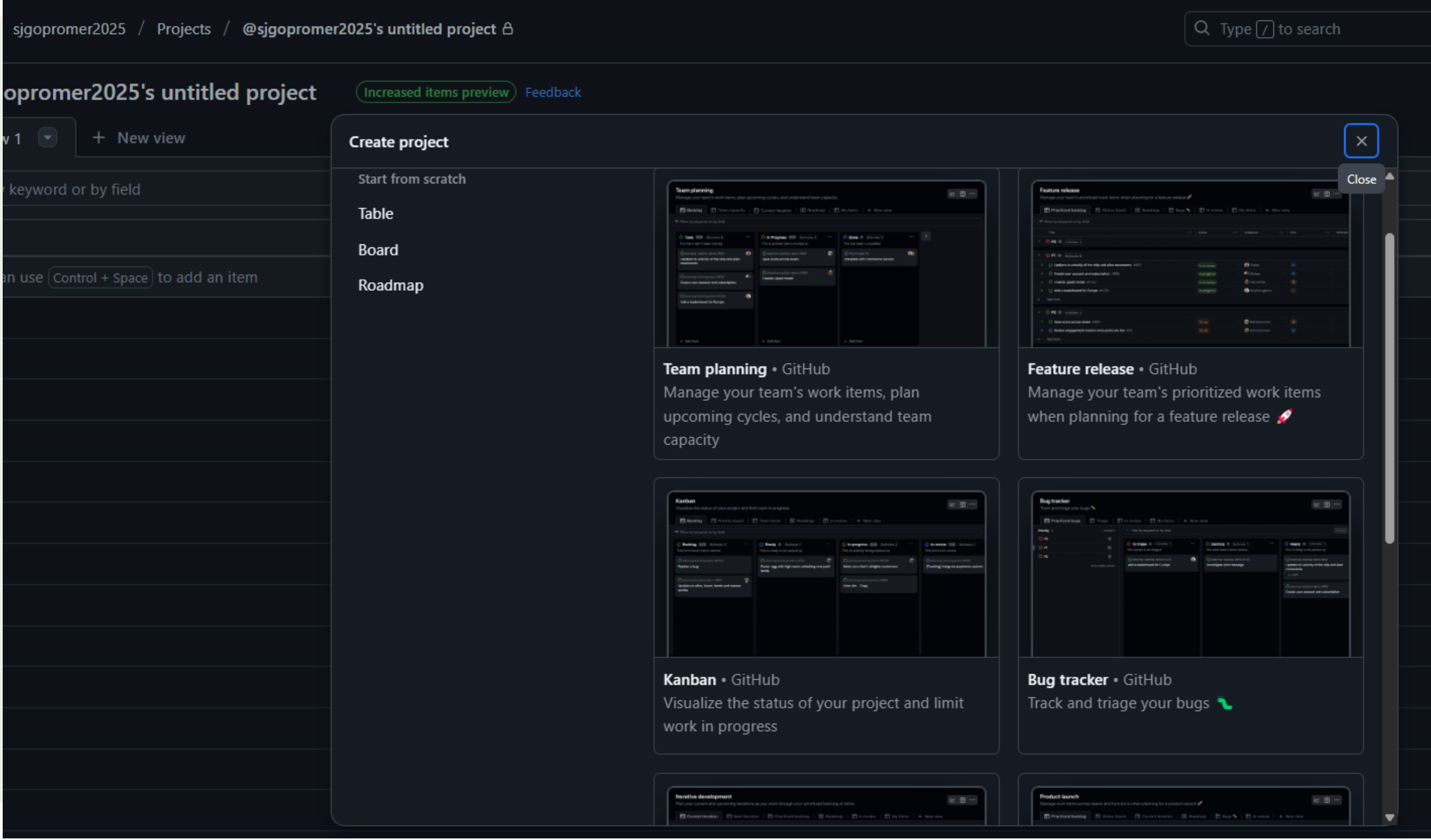
No milestone

프로젝트 보드

프로젝트 보드란?

- 작업들을 카드 형태로 시각화하여 관리할 수 있는 도구입니다.
- 보통 칸반(Kanban) 스타일로 사용되며, 작업 흐름을 명확히 보여줍니다.
- 구성 예시:
 - To Do: 해야 할 일
 - In Progress: 진행 중인 일
 - Done: 완료된 일
- 이슈를 카드처럼 드래그해서 이동하며 팀의 업무 현황을 관리할 수 있습니다..
- 프로젝트 보드는 이슈와 연동할 수 있어, 팀 작업을 한눈에 파악할 수 있게 도와줍니다.

프로젝트 보드



06

정리 및 꿀팁

- 자주 쓰는 명령어 요약
- Git 사용 시 주의할 점
- GUI 툴 (GitHub Desktop) 소개

자주 쓰는 명령어

로컬 변경 사항 업로드

- `git init` 새 Git 저장소 초기화
- `git add .` 모든 변경 사항을 Staging Area에 추가
- `git commit -m` 변경 사항을 커밋하고 메시지 추가
- `git status` 현재 작업 상태 확인 (추적, Staging 여부)
- `git log` 커밋 히스토리 확인
- `git branch` 현재 브랜치와 다른 브랜치 목록 확인
- `git checkout` 다른 브랜치로 전환
- `git merge` 다른 브랜치의 변경 사항을 현재 브랜치에 병합
- `git push` 로컬 변경 사항을 원격 저장소에 업로드
- `git pull` 원격 저장소의 변경 사항을 로컬로 가져오기

Git 사용 시 주의할 점

주의 사항

- 커밋 메시지는 명확하게 작성.
- 병합 전 항상 `git pull`로 최신 상태 유지.
- 민감한 정보는 `.gitignore`로 제외.

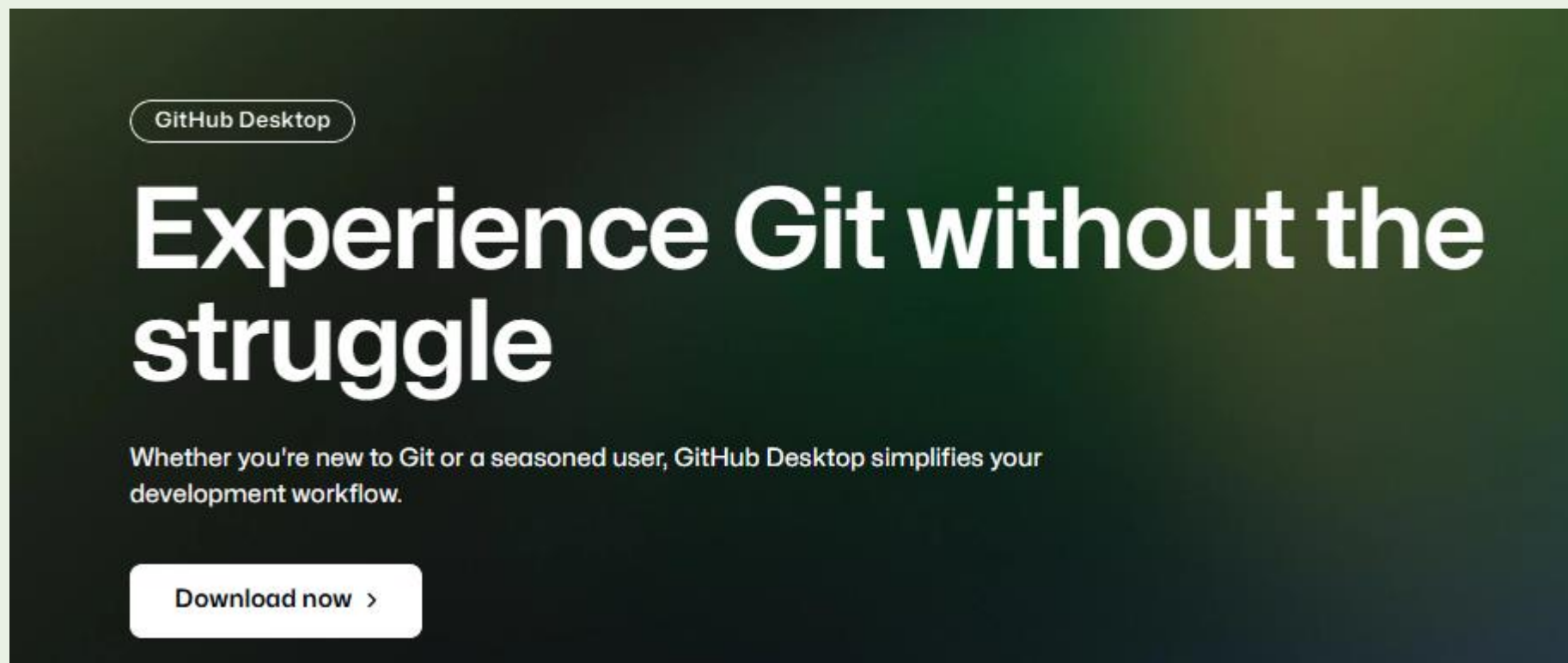
Github Desktop

- GitHub Desktop은 Git을 사용하여 버전 관리 작업을 시각적으로 쉽게 할 수 있도록 돕는 그래픽 사용자 인터페이스(GUI) 툴입니다.
- 주로 Git 명령어를 익히기 어려운 초보자나, Git을 더 직관적으로 사용하고 싶은 사람들에게 유용합니다.
- 주요 기능:
 - 버전 관리
 - 브랜치 관리
 - 변경 사항 비교
 - 원격 저장소와 연동
- 장점:
 - 간편한 사용: 명령어를 몰라도 시각적으로 Git의 기능을 쉽게 사용할 수 있어, GitHub를 처음 사용하는 사용자에게 매우 유용합니다.
 - GitHub 통합: GitHub Desktop은 GitHub와 완벽하게 통합되어 있어, GitHub의 저장소를 쉽게 관리하고 협업할 수 있습니다.

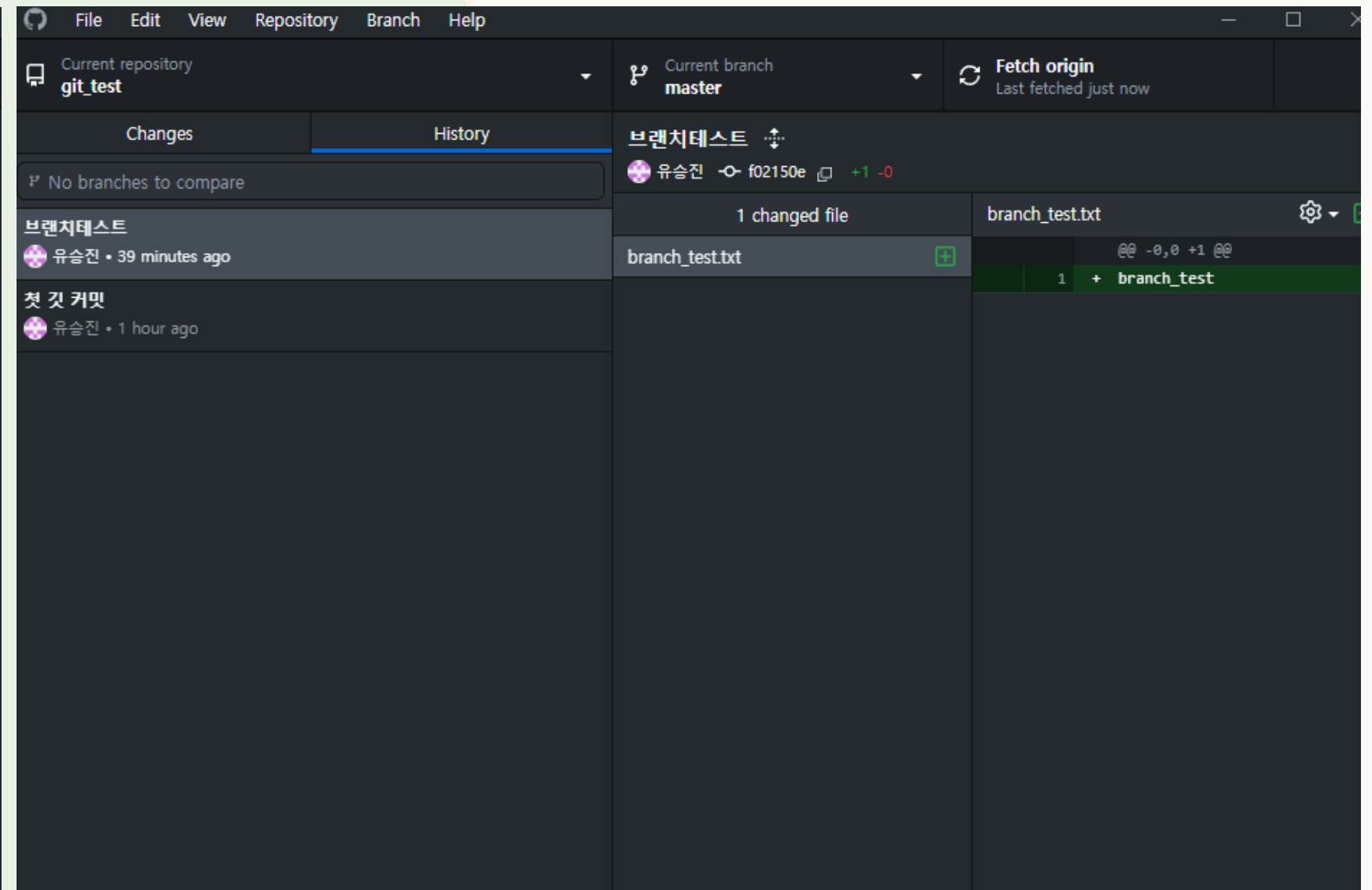
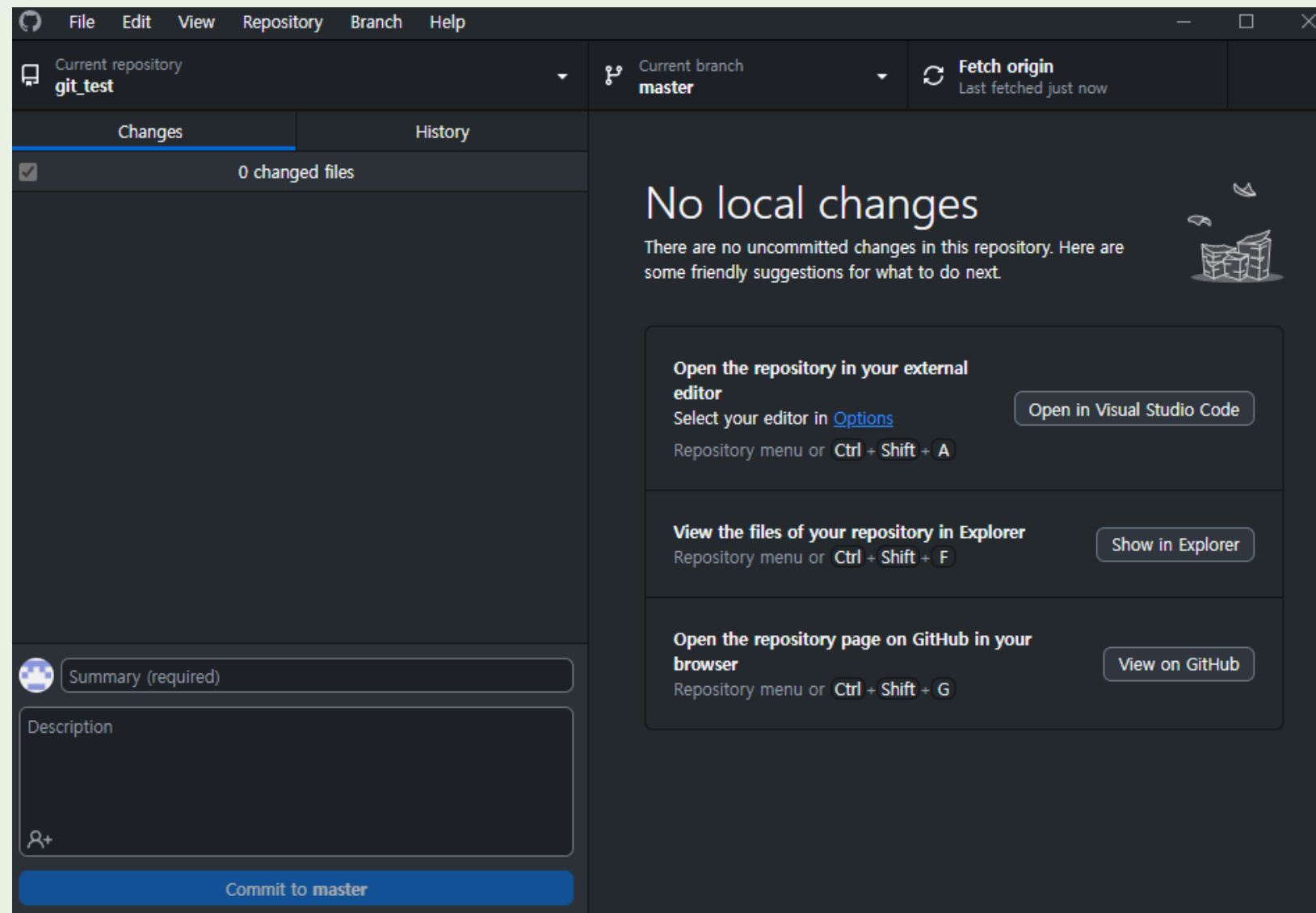
Github Desktop 설치

- 링크 :

https://github.com/apps/desktop?ref_cta=download+desktop&ref_loc=installing+github+desktop&ref_page=docs



Github Desktop



고생하셨습니다!

