



# git + GitHub

On your mark, git set, go

강사 최인규



# Git과 버전관리

Git 다운로드 받기

<https://git-scm.com/>에서 32 또는 64-bit Git for Windows Setup을 다운로드

다운로드만!!!

# 만약에 Git, GitHub가 없다면?

게임을 켜면 '이어하기'가 없고, '새로하기'만 있다!?

마치 SAVE 기능 없이 게임을 하는 것!

Git과 GitHub는 개발을 이어서 계속할 수 있게 한다.



# CONTENTS

**01**

Git과 버전관리

버전관리의 필요성과 Git 알아보기,  
Git 설치하고 Git의 기본 명령어 활용

로컬 저장소 생성, 파일 추적 및 commit, branch 구성

원격 저장소 연결, push와 pull

**02**

Git & GitHub 기초 (CLI)

**03**

Git & GitHub 심화 (GUI)

**04**

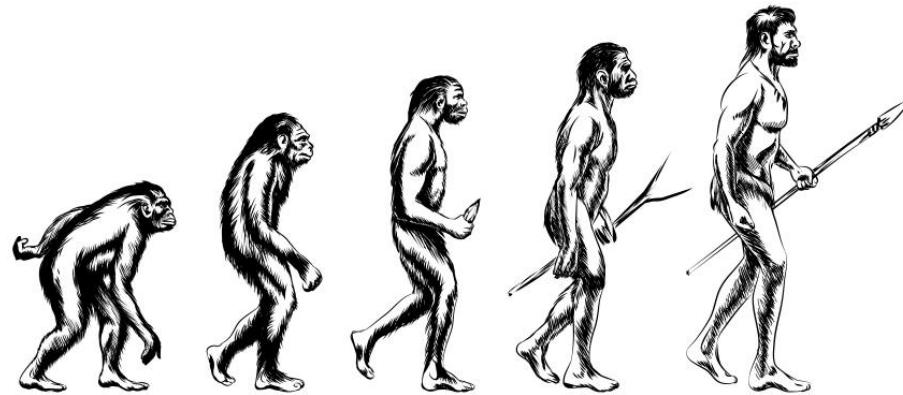
실전을 통해 Git 정복하기

브랜치 및 커밋 관리,  
충돌 시 해결방법,  
pull request 생성 및 검토

MarkDown 문법,  
GitHub 소개글 꾸미기,  
커밋 컨벤션,  
코드 리뷰

# Git과 버전관리

버전이란 무엇일까요?



Windows 1



Windows 3.1



Windows 95



Windows XP



Windows  
Vista



Windows 7



Windows 8

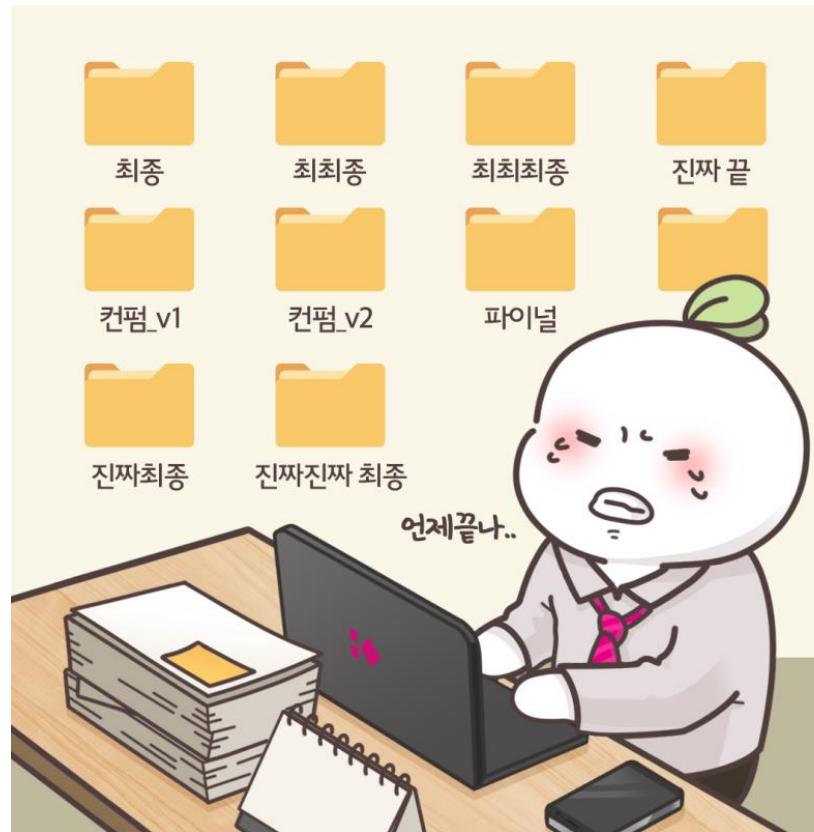


Windows 10



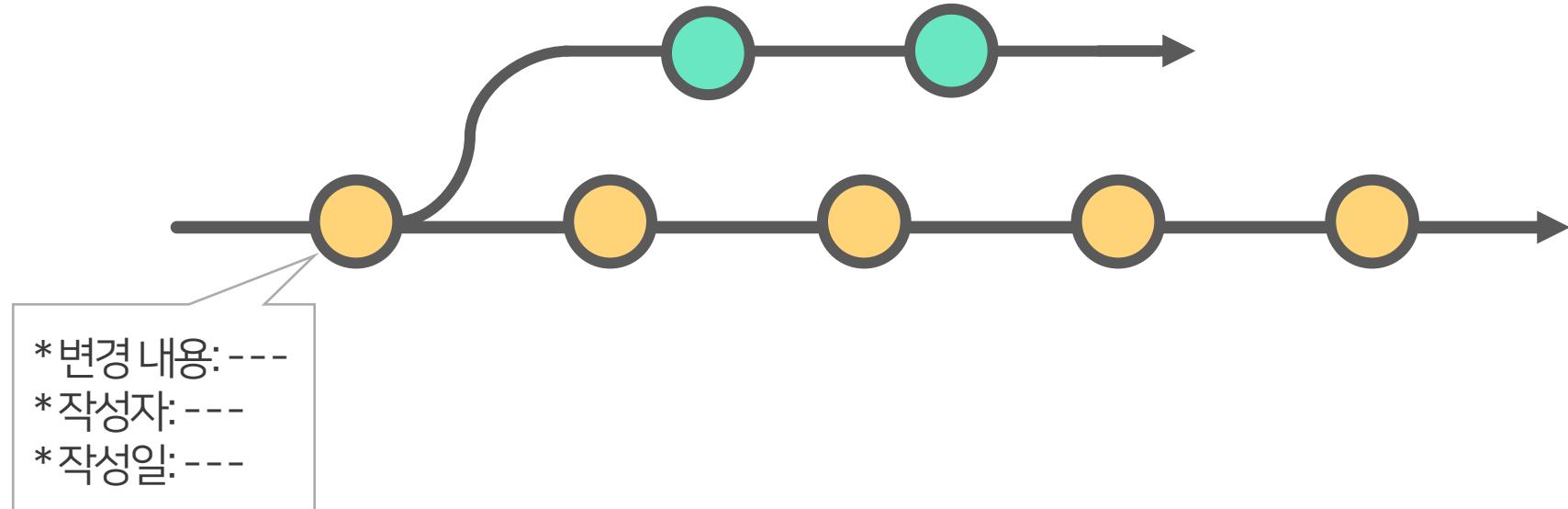
Windows 11

# Git과 버전관리



# Git과 버전관리

버전 관리 시스템을 이용하면, 버전 관리는 물론 변경점 관리, 백업 및 복구, 협업에 큰 이점이 있다.



# Git과 버전관리

## 버전 관리는

"누가, 언제, 어떻게 변경했는지 **변경 내역을 기억하고**,  
필요하다면 **특정 시점의 버전으로 되돌릴 수 있다**는 특징을 바탕으로,  
**협업하는 과정에서 코드를 쉽게 나누고 합치며 개발할 수 있도록 도와주는 것**"이다.

# Git과 버전관리

만약 버전 관리 시스템이 없다면?

1. 각자 개발을 진행한다. (정해진 날짜에 파일을 주고 받아 합치기로 약속)
2. 혹시 합치면서 에러가 발생할 수 있으니, 백업본을 만들어 둔다.
3. 팀원이 내가 작업한 파일을 고친 경우에는 꼭 말해달라고 한다.
4. 코드를 합친 후에는 내 컴퓨터에 반영을 시켜줘야 한다.

# Git과 버전관리

만약 버전 관리 시스템이 있다면?

원할 때 언제든지 코드를 간편하게 합칠 수 있고, 백업도 쉽게 가능해지게 된다. 

# Git과 버전관리

Git은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 분산 버전 관리 시스템이다.

소프트웨어 개발에서 소스 코드 관리에 주로 사용되지만 어떠한 파일 집합의 변경사항을 지속적으로 추적하기 위해 사용될 수 있다.



깃

소프트웨어

깃은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 스냅샷 스트림 기반의 분산 버전 관리 시스템이다. 또는 이러한 명령어를 가리킨다. 위키백과

개발: 리누스 토르발스, Junio C Hamano

프로그래밍 언어: 파이썬, C, C++, 셀 스크립트, 펄, Tcl

개발자: 주니오 하마노(Junio Hamano), 리누스 토르발스 등

ライセンス: GNU 일반 공중 사용 허가서 v2

안정화 버전: 2.43.0 / 2023년 11월 20일

언어: 영어

종류: 버전 관리

# Git과 버전관리



Alice



Bob

# Git과 버전관리



페이지 1 ~ 3을 작성해 'Alice ver.1' 저장



'Alice ver.1'을 다운로드 받은 뒤,  
이어서 페이지 4 ~ 6을 작성해 'Bob ver.1' 저장



페이지 2를 수정한 후, 'Alice ver.2' 저장



'Alice ver.2'을 다운로드 받은 뒤,  
'Bob ver.1'과 차이점(diff)을 반영한 'Bob ver.2' 저장

# Git과 버전관리



- Git은 코딩을 하면서 원하는 시점마다 깃발을 꽂고, 자유롭게 돌아다닐 수 있게 해준다.



- Git은 저장할 공간만 있다면 어디서나 사용이 가능합니다.



- Git은 CLI(Command Line Interface) 방식과 GUI(Graphic User Interface) 방식 모두 활용 가능합니다.



# Git과 버전관리

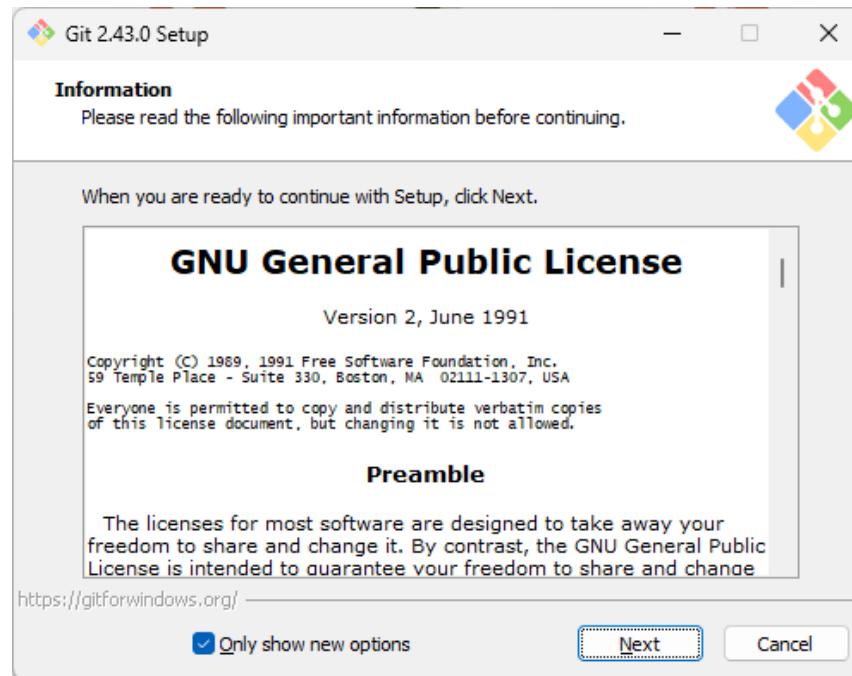
VS code 설치하기

<https://code.visualstudio.com/>에서 다운로드 및 설치

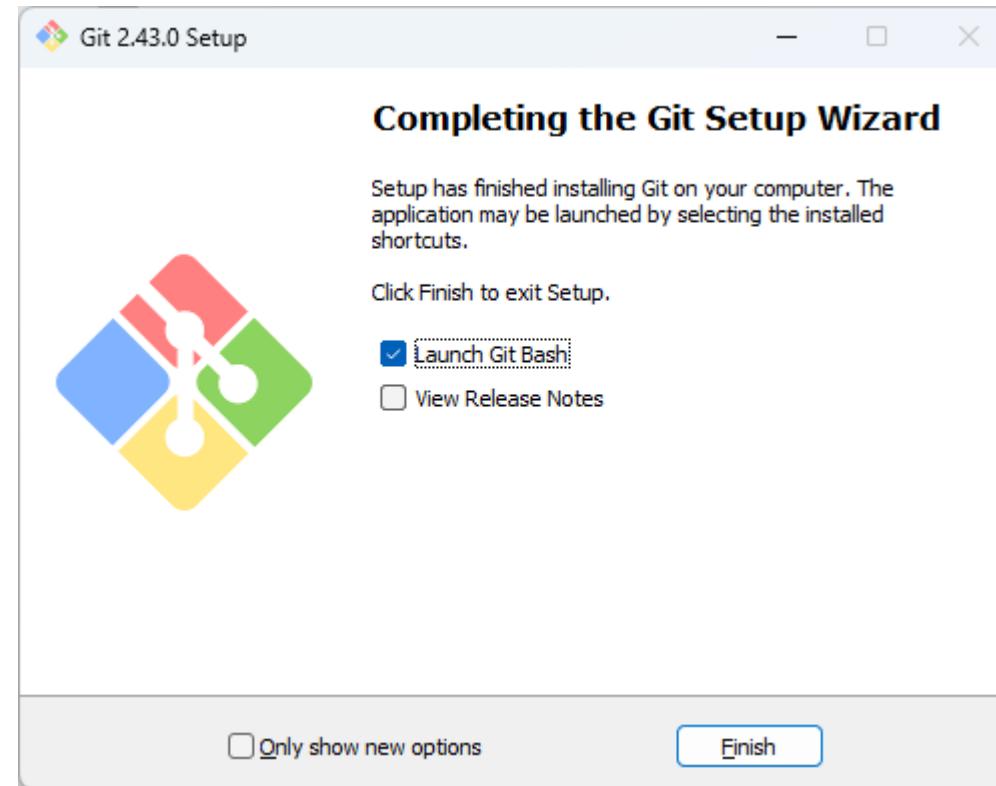


# Git과 버전관리

## Git 설치하기



# Git과 버전관리



# Git과 버전관리

GitHub 가입하기 <https://github.com/signup>

Welcome to GitHub!  
Let's begin the adventure

Enter your email\*

✓ email 주소 입력

Create a password\*

✓ 비밀번호 입력

Enter a username\*

✓ 사용자 이름 입력

Email preferences

Receive occasional product updates and announcements.



# Git과 버전관리

```
Username@DESKTOP MINGW64 ~
$ git config --global user.name "사용자이름"
```

```
Username@DESKTOP MINGW64 ~
$ git config --global user.email "이메일주소"
```

```
Username@DESKTOP MINGW64 ~
$ git config --global --list
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
user.name=사용자이름
user.email=이메일주소
```

# Git과 버전관리

```
Username@DESKTOP MINGW64 ~
$ pwd
/c/Users/Username

Username@DESKTOP MINGW64 ~
$ mkdir my_story

Username@DESKTOP MINGW64 ~
$ cd my_story/

Username@DESKTOP MINGW64 ~/my_story
$ pwd
/c/Users/Username/my_story

Username@DESKTOP MINGW64 ~/my_story
$ start .
```

# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story
$ git init
Initialized empty Git repository in C:/users/Administrator/my_story/.git/

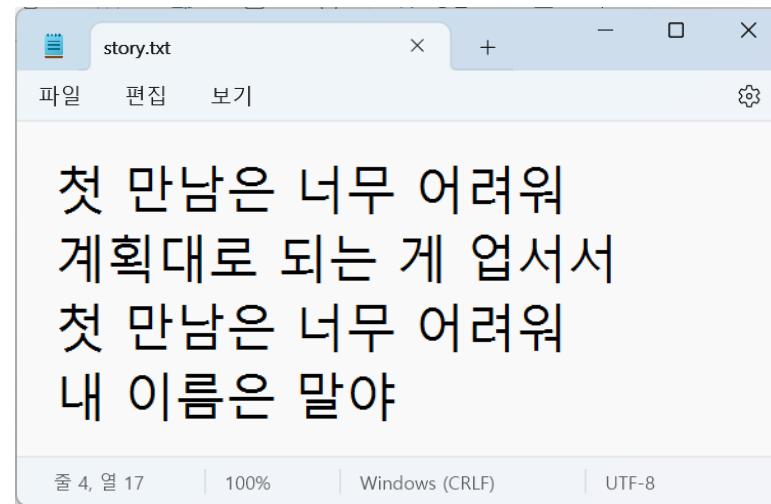
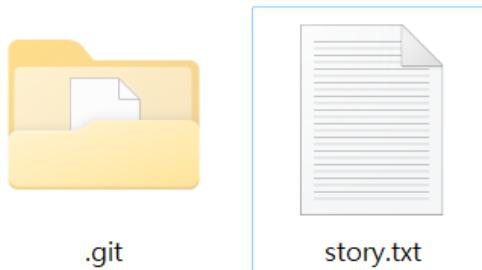
Username@DESKTOP MINGW64 ~/my_story (main)
$ ls -a
./ ../ .git/

Username@DESKTOP MINGW64 ~/my_story (main)
$ touch story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ ls -a
./ ../ .git/ story.txt
```



# Git & GitHub 기초 (CLI)





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ ls -a
./ ../ .git/ story.txt
```

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ cat story.txt
첫 만남은 너무 어려워
계획대로 되는 게 업서서
첫 만남은 너무 어려워
내 이름은 말야
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    story.txt

nothing added to commit but untracked files present (use "git add" to track)
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git add story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git rm --cached story.txt
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git add .

Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch main

No commits yet

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



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git commit -m "first commit"
[main (root-commit) 290b8a8] first commit
 1 file changed, 4 insertions(+)
 create mode 100644 story.txt
```

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch main
nothing to commit, working tree clean
```



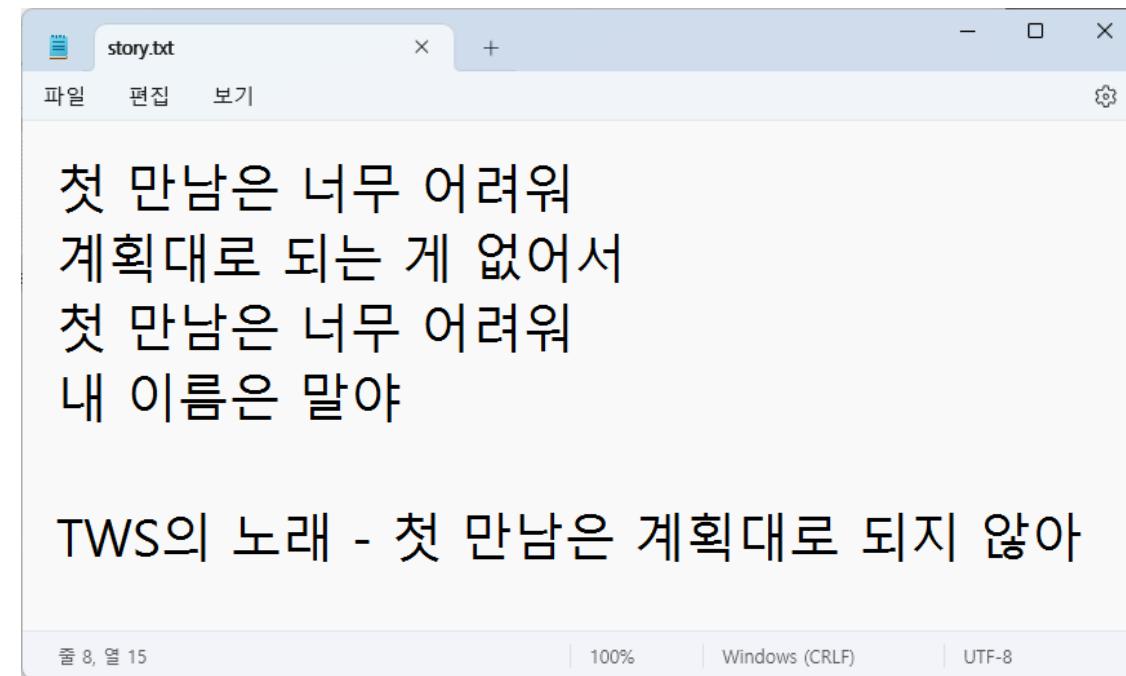
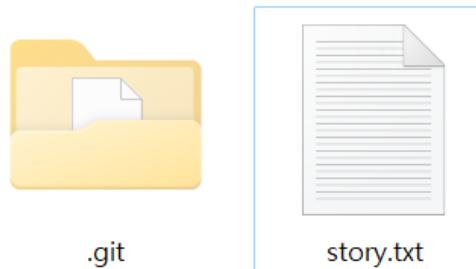
# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log
commit 11657497ead0802191e012d8c6497a2d9d1cd96c (HEAD -> main)
Author: 사용자이름 <이메일주소>
Date:   wed Oct 1 09:00:00 2025 +0900

    first commit
```



# Git & GitHub 기초 (CLI)





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
on branch main
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:   story.txt

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

# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git diff
diff --git a/story.txt b/story.txt
index 72547bc..e9e9e26 100644
--- a/story.txt
+++ b/story.txt
@@ -1,4 +1,6 @@
    첫 만남은 너무 어려워
-계획대로 되는 게 업서서
+계획대로 되는 게 없어서
    첫 만남은 너무 어려워
-내 이름은 말야
\ No newline at end of file
+내 이름은 말야
+
+TWS의 노래 - 첫 만남은 계획대로 되지 않아
\ No newline at end of file
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git commit -m "second commit"
[main bd295f3] second commit
 1 file changed, 5 insertions(+), 1 deletion(-)
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log
commit bd295f362aadb58e9c8de7499eead2c23c17c99d (HEAD -> main)
Author: 사용자이름 <이메일주소>
Date:   wed Oct 1 10:00:00 2025 +0900

    second commit

commit 11657497ead0802191e012d8c6497a2d9d1cd96c
Author: 사용자이름 <이메일주소>
Date:   wed Oct 1 09:00:00 2025 +0900

    first commit
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log --oneline
0da4c89 (HEAD -> main) second commit
5cf58d3 first commit
```

# Git & GitHub 기초 (CLI)

git reset : 이전 commit으로 되돌린다. (다른 사람과 코드 공유가 되어 있지 않은 상태에서 사용)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git reset --옵션 커밋ID
```

종류	설명
git reset --soft 커밋ID	staging 상태로 파일 변경 사항이 남는다.
git reset --mixed 커밋ID	unstaging 상태로 파일 변경 사항이 남는다.
git reset --hard 커밋ID	파일 변경 사항이 전혀 남지 않는다.

# Git & GitHub 기초 (CLI)

작업 중인 디렉토리  
(Working Directory)

Staging Area

Git 저장소  
(Git Repository)

git add 파일명

git restore --staged 파일명

git commit -m "메시지"

git reset --옵션

git init

git status

git log

git log --oneline

git diff





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log --oneline
0681292 (HEAD -> main) E
8b4a3a6 R
df51dbc O
e288f45 T
65fd425 S
bd295f3 second commit
1165749 first commit
```

# Git & GitHub 기초 (CLI)

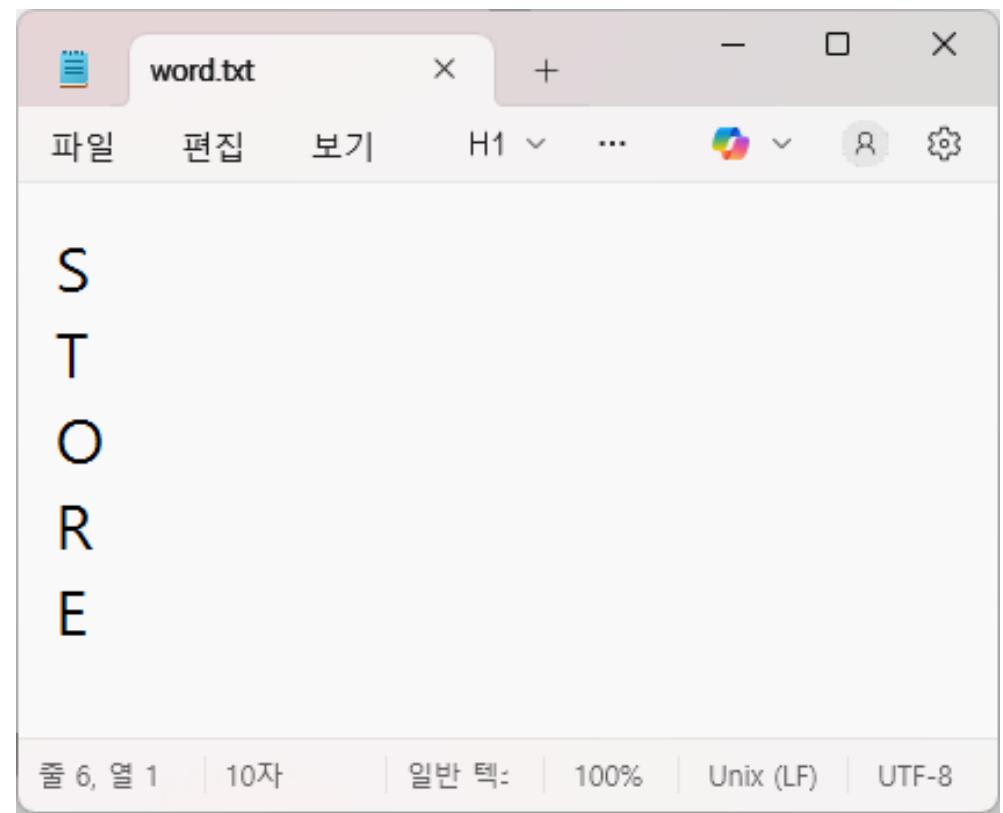
```
Username@DESKTOP MINGW64 ~/my_story (main)
$ echo 'S' >> word.txt
$ git add word.txt
$ git commit -m "S"

$ echo 'T' >> word.txt
$ git commit -am "T"

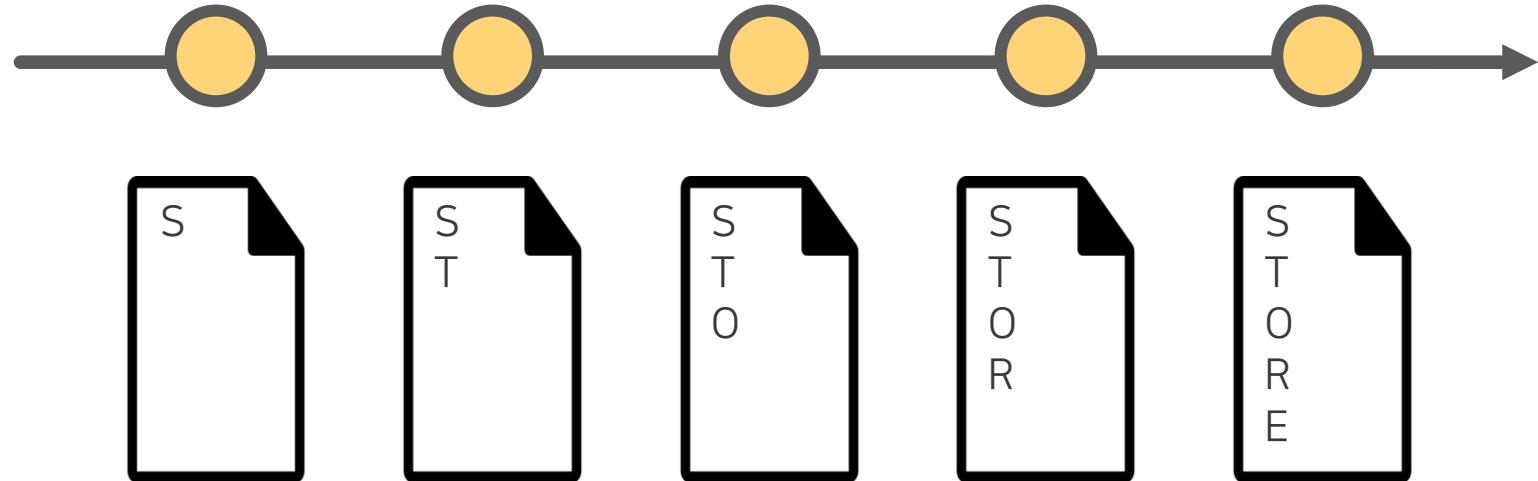
$ echo 'O' >> word.txt
$ git commit -am "O"

$ echo 'R' >> word.txt
$ git commit -am "R"

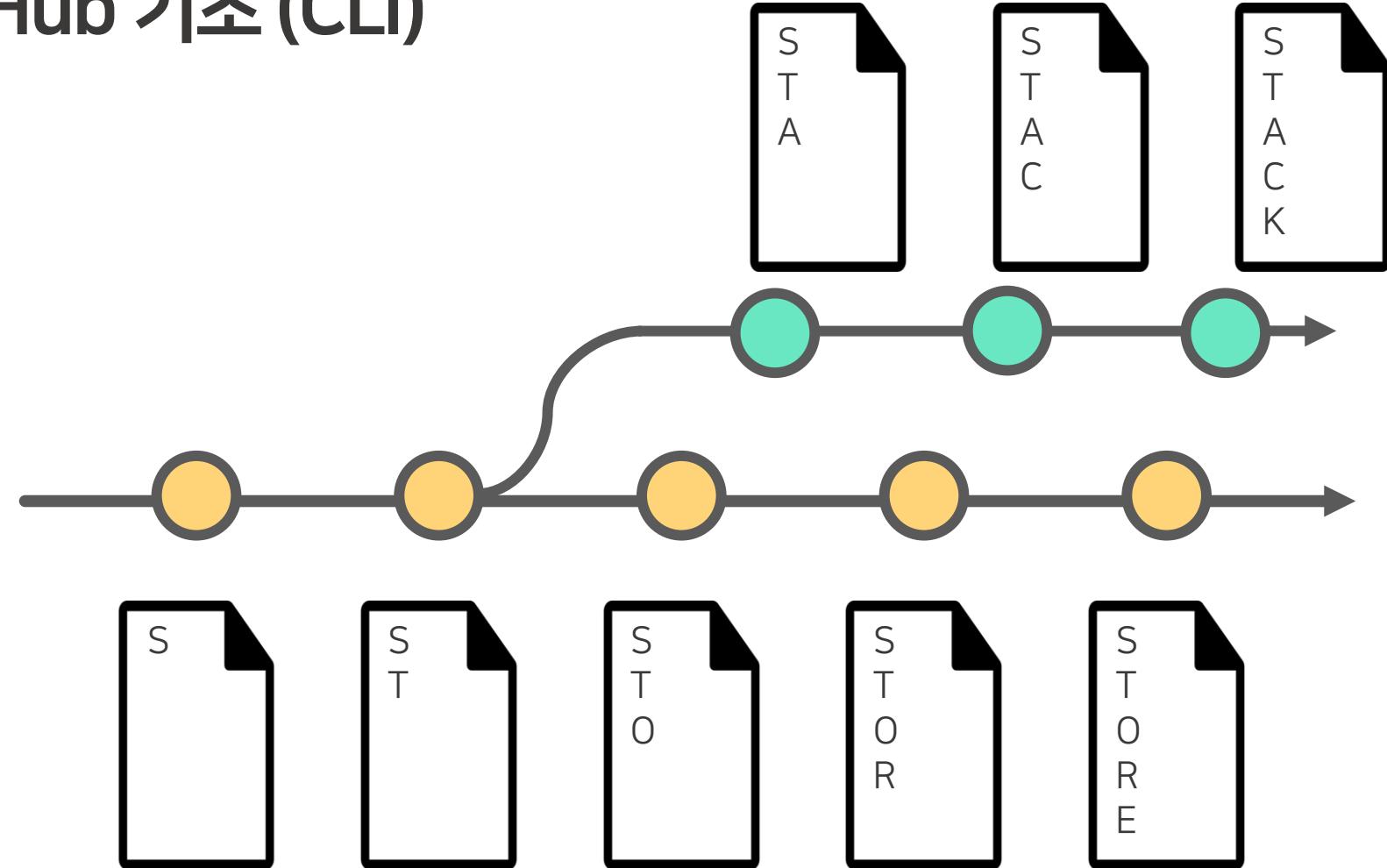
$ echo 'E' >> word.txt
$ git commit -am "E"
```



# Git & GitHub 기초 (CLI)



# Git & GitHub 기초 (CLI)





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git checkout ["T" CommitID]
```

Note: switching to 'e288'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

...

```
Username@DESKTOP MINGW64 ~/my_story ((e288f45...))
$
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git checkout ["T" CommitID]
```

Note: switching to 'e288'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

...

```
Username@DESKTOP MINGW64 ~/my_story ((e288f45...))
$ cat word.txt
S
T
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story ((e288f45...))
$ echo 'A' >> word.txt
$ git commit -am "A"
```

```
Username@DESKTOP MINGW64 ~/my_story ((98762c0...))
$ echo 'C' >> word.txt
$ git commit -am "C"
```

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))
$ echo 'K' >> word.txt
$ git commit -am "K"
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))
$ git log --oneline
15b785b (HEAD) K
243f7e8 C
98762c0 A
e288f45 T
65fd425 S
bd295f3 second commit
1165749 first commit
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))  
$ git switch -c dev
```

```
Username@DESKTOP MINGW64 ~/my_story (dev)  
$
```

# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))  
$ git log --oneline  
15b785b (HEAD) K  
243f7e8 C  
98762c0 A  
e288f45 T  
65fd425 S  
bd295f3 second commit  
1165749 first commit
```

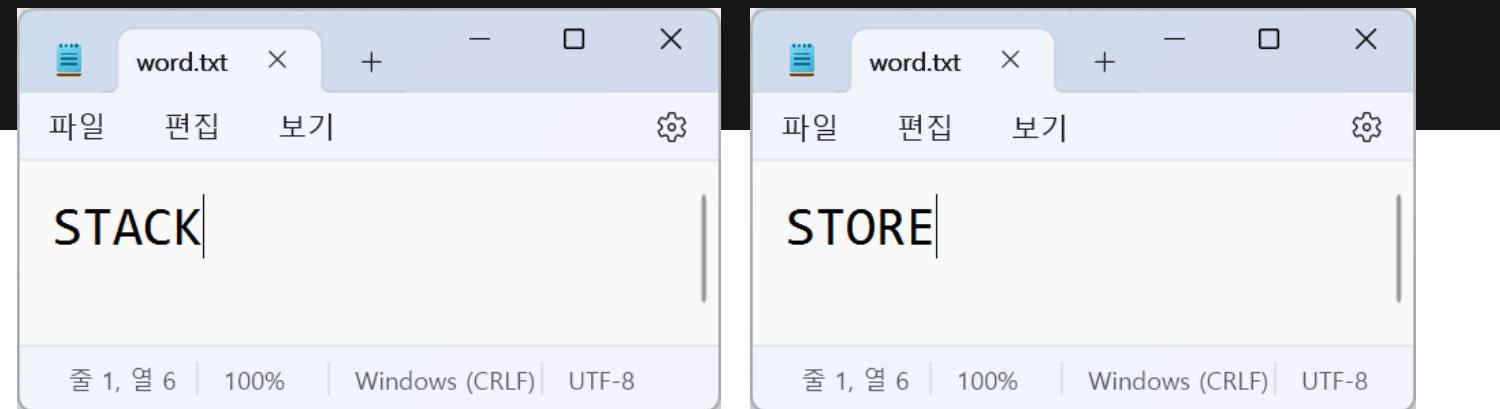


```
Username@DESKTOP MINGW64 ~/my_story (dev)  
$ git log --oneline  
15b785b (HEAD -> dev) K  
243f7e8 C  
98762c0 A  
e288f45 T  
65fd425 S  
bd295f3 second commit  
1165749 first commit
```

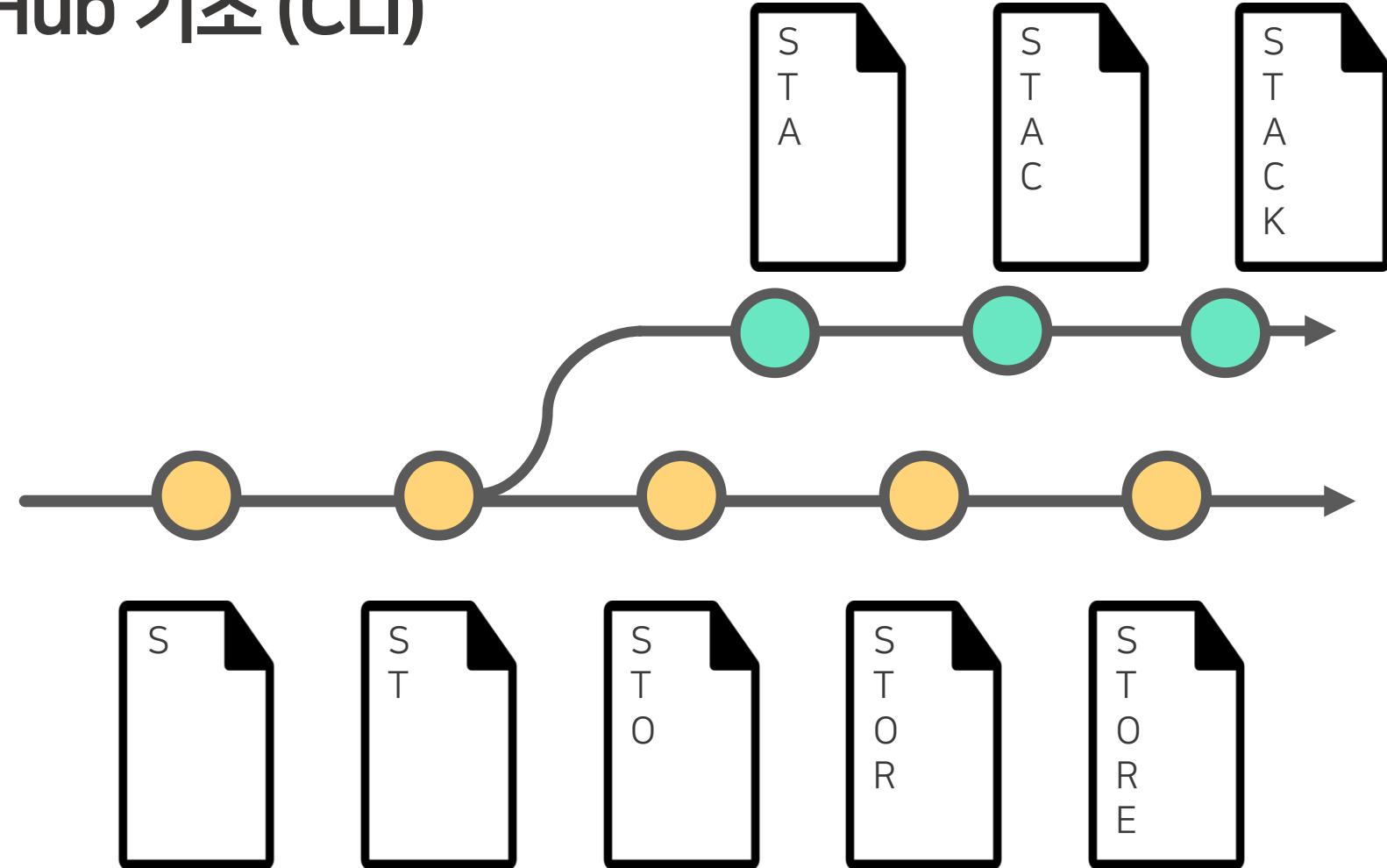
# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/my_story (dev)
$ git switch main
Switched to branch 'main'
```

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git branch -l
  dev
* main
```



# Git & GitHub 기초 (CLI)



# Git & GitHub 기초 (CLI)

GitHub는 원격 Git 저장소(Remote Git Repository)를 제공하는 서비스이다.

내 컴퓨터에 있는 Git 저장소(Local Git Repository)의 기록들을 업로드하는 곳!  
업로드 된 파일들은 언제든 어디서든 버전 관리가 가능해진다.

# Git & GitHub 기초 (CLI)

The screenshot shows the GitHub Home page with the following sections:

- Top Repositories**: Includes a search bar and two repositories: InKyulinst/2023-05\_FE\_MultiCampus and InKyulinst/2023-04\_FE\_MultiCampus.
- Recent activity**: A placeholder message: "When you take actions across GitHub, we'll provide links to that activity here."
- Home**:
  - Updates to your homepage feed**: A message about combining Following and For you feeds.
  - Start writing code**:
    - Create a new repository**: Fields for repository name (InKyulinst / name your new repository...), visibility (Public or Private), and a "Create" button.
    - Introduce yourself with a profile README**: A sample README.md content.
  - Use tools of the trade**:
    - Write code in your web browser**: Information about the github.dev editor.
    - Get AI-based coding suggestions**: Information about GitHub Copilot.
  - Get started on GitHub**: A video thumbnail of a person at a desk with the text "What is GitHub?" and a "Try the GitHub flow" button.
  - Follow this exercise to try the GitHub flow**: Information about GitHub's "Hello World" tutorial.
- Learning Pathways**: A card with the heading "Learn from the best" and a "Start learning" button.
- Latest changes**: A list of recent updates:
  - 19 hours ago: Code scanning is now more adaptable to your codebase with CodeQL threat model settings...
  - Yesterday: Upcoming Deprecation of Organization Insights Activity Overview beta
  - 2 days ago: Code scanning default setup is now available for self-hosted runners on GitHub.com
  - 3 days ago: Log in to multiple GitHub accounts with the CLI
- Explore repositories**: A list of repositories:
  - opencv / cvat**: Annotate better with CVAT, the industry-leading data engine for machine learning. Used and trusted by teams at any scale, for data of any scale.
  - containers / podman**: Podman: A tool for managing OCI containers and pods.
  - Kong / kubernetes-ingress-controller**: Kong for Kubernetes: The official Ingress Controller for Kubernetes.

On the right side, there is a vertical sidebar with icons: +, -, ⌂, ⌂, ⌂, ⌂, and a red notification badge with the number 5.

# Git & GitHub 기초 (CLI)

Settings

InKyulInst (InKyulInst)  
Your personal account

Public profile

Name   
Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Profile picture  [Edit](#)

Public email   
Select a verified email to display [▼](#)  
You have set your email address to private. To toggle email privacy, go to [email settings](#) and uncheck "Keep my email address private."

Bio   
Tell us a little bit about yourself  
You can @mention other users and organizations to link to them.

Pronouns   
Don't specify [▼](#)

URL

Social accounts

Company   
You can @mention your company's GitHub organization to link it.

Location

Display current local time  
Other users will see the time difference from their local time.

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

Go to your personal profile

Public profile

Archives

Security log

Sponsorship log

<> Developer settings

Archives

Security log

Sponsorship log

# Git & GitHub 기초 (CLI)

Settings ➡ Developer settings ➡ Personal access tokens ➡ Generate new token ➡ Generate new token (classic)

The screenshot shows the GitHub developer settings interface. In the top navigation bar, the 'Personal access tokens' option is selected, indicated by a blue border around the icon and the text 'Personal access tokens'. Below this, the 'Personal access tokens (classic)' section is visible. It includes a button to 'Generate new token' (Beta) and another button for 'Generate new token (classic)'. A note at the bottom states: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used to authenticate to the API over Basic Authentication.'

GitHub Apps

OAuth Apps

Personal access tokens

Personal access tokens (classic)

Generate new token

Generate new token (Beta)

Fine-grained, repo-scoped

Generate new token (classic)

For general use

# Git & GitHub 기초 (CLI)

Personal Access Token을 생성한다.

Note : Token for Study (토큰을 식별하기 위한 메모)

Expiration : No expiration (토큰 만료기간)

Select scopes : repo 전체 체크 (해당 토큰 접근 범위 설정)

Generate token

## New personal access token (classic)

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](#).

### Note

Token for Study

What's this token for?

### Expiration \*

No expiration

The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure.  
[Learn more](#)

### Select scopes

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

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

<input type="checkbox"/> workflow	Update GitHub Action workflows
-----------------------------------	--------------------------------

<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry



# Git & GitHub 기초 (CLI)

생성된 Personal Access Token을 생성한다.

Personal access tokens (classic)	Generate new token	Revoke all
Tokens you have generated that can be used to access the <a href="#">GitHub API</a> .		
Make sure to copy your personal access token now. You won't be able to see it again!		
 <code>gpp_0kLkrxvcaevyMKit1StFrsu5VXcaq0215e01</code> 		

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.

# Git & GitHub 기초 (CLI)

GitHub에 개발한 코드를 올릴 때에는 아래와 같은 순서를 따른다.

1. 내 컴퓨터 안에 존재하는 프로젝트 폴더에서 "이제 여기에서 Git을 사용할거야!"라고 명령하기
2. 즐거운 마음으로 코딩하기
3. 내 파일들 중에 GitHub에 올릴 파일들을 선택하기
4. 선택한 파일들을 한 덩어리로 만들어서 설명 적기
5. GitHub에서 프로젝트 저장소를 생성하기
6. 내 컴퓨터 프로젝트 폴더에 "내 GitHub 저장소 주소는 여기야!"라고 알려주기
7. "덩어리들을 GitHub에 올려줘!"라고 명령하기

# Git & GitHub 기초 (CLI)



company

출근

## To Do

- 프로젝트 생성
- 데이터 모델링
- 화면 구성
- 기능 구현



home

귀가

## To Do

- 프로젝트 생성
- 데이터 모델링
- 화면 구성
- 기능 구현

1. 회사에서 로컬 저장소를 생성한다.
2. 원격 저장소를 생성한다.
3. 원격 저장소와 연결한다.
4. 기능 단위로 add과 commit을 진행한다.
5. 원격 저장소에 올린다.

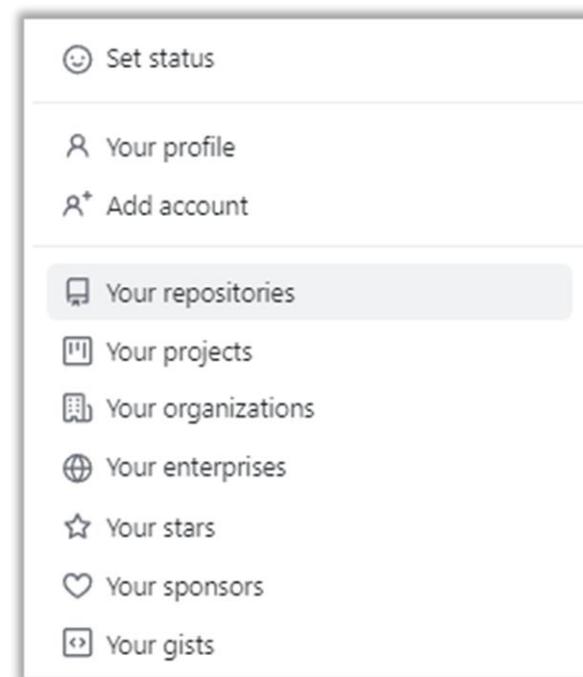
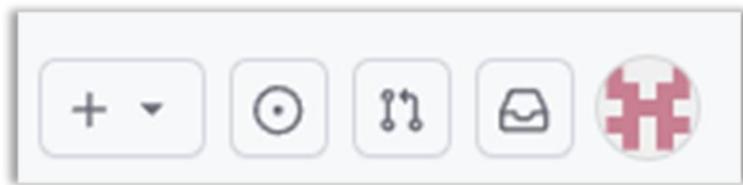
퇴근

취침

6. 원격 저장소에서 받아온다.
7. 이어서 기능 단위로 add과 commit을 진행한다.
8. 원격 저장소에 올린다.



# Git & GitHub 기초 (CLI)



# Git & GitHub 기초 (CLI)

GitHub Repository 생성하기

Repository name : my\_project (저장소 명)

Description (저장소 설명)

public / private (공개 범위)

Add a README file (저장소 설명 파일 추가)

Add .gitignore (Git에서 관리하지 않을 파일 지정)

Choose a license (라이선스 안내)

## Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).  
Required fields are marked with an asterisk (\*).

### 1 General

Owner \*



Repository name \*

my\_project

my\_project is available.

Great repository names are short and memorable. How about [fluffy-octo-fortnight](#)?

#### Description

로그인 프로젝트

8 / 350 characters

### 2 Configuration

#### Choose visibility \*

Choose who can see and commit to this repository



#### Add README

READMEs can be used as longer descriptions. [About READMEs](#)



#### Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)



#### Add license

Licenses explain how others can use your code. [About licenses](#)



**Create repository**

# Git & GitHub 기초 (CLI)

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

Set up in Desktop

or

HTTPS

SSH

[https://github.com/InKyuInst/my\\_project.git](https://github.com/InKyuInst/my_project.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 "# my_project" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/InKyuInst/my_project.git
git push -u origin main
```



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

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



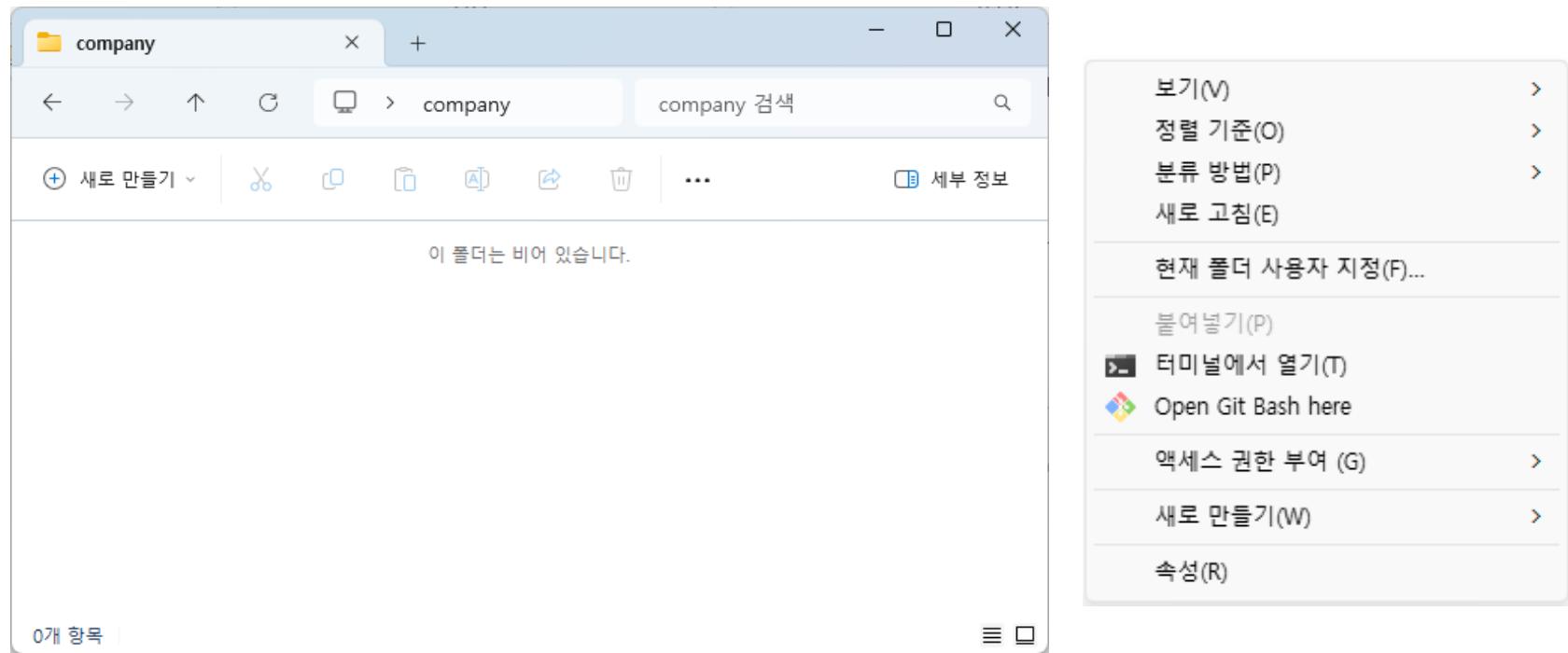
# Git & GitHub 기초 (CLI)

GitHub에 개발한 코드를 올릴 때에는 아래와 같은 순서를 따른다.

1. 내 컴퓨터 안에 존재하는 프로젝트 폴더에서 "이제 여기에서 Git을 사용할거야!"라고 명령하기
2. 즐거운 마음으로 코딩하기
3. 내 파일들 중에 GitHub에 올릴 파일들을 선택하기
4. 선택한 파일들을 한 덩어리로 만들어서 설명 적기
5. GitHub에서 프로젝트 저장소를 생성하기
6. 내 컴퓨터 프로젝트 폴더에 "내 GitHub 저장소 주소는 여기야!"라고 알려주기
7. "덩어리들을 GitHub에 올려줘!"라고 명령하기



# Git & GitHub 기초 (CLI)





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/company
$ git init
Initialized empty Git repository in C:/Users/UserName/Desktop/company/.git/
$ echo "로그인 프로젝트" >> README.md
$ git add .
$ git commit -m "프로젝트 생성"
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git remote add origin https://github.com/사용자이름/my_project.git

$ git remote -v
origin https://github.com/사용자이름/word_play.git (fetch)
origin https://github.com/사용자이름/word_play.git (push)

$ git push -u origin main
```



# Git & GitHub 기초 (CLI)

my\_project Public

Pin Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags Go to file Add file Code

InKyulinst 프로젝트 생성 01b1536 · 16 minutes ago 1 Commit

README.md 프로젝트 생성 16 minutes ago

README

로그인 프로젝트

About

로그인 프로젝트

- Readme
- Activity
- 0 stars
- 0 watching
- 0 forks

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ mkdir model
$ cd model
```

```
Username@DESKTOP MINGW64 ~/Desktop/company/model (main)
$ echo "아이디, 비밀번호" >> data.txt

$ git add .
$ git commit -m "데이터 모델링"
$ git push
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/company/model (main)
$ cd ..
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ mkdir view
$ cd view
```

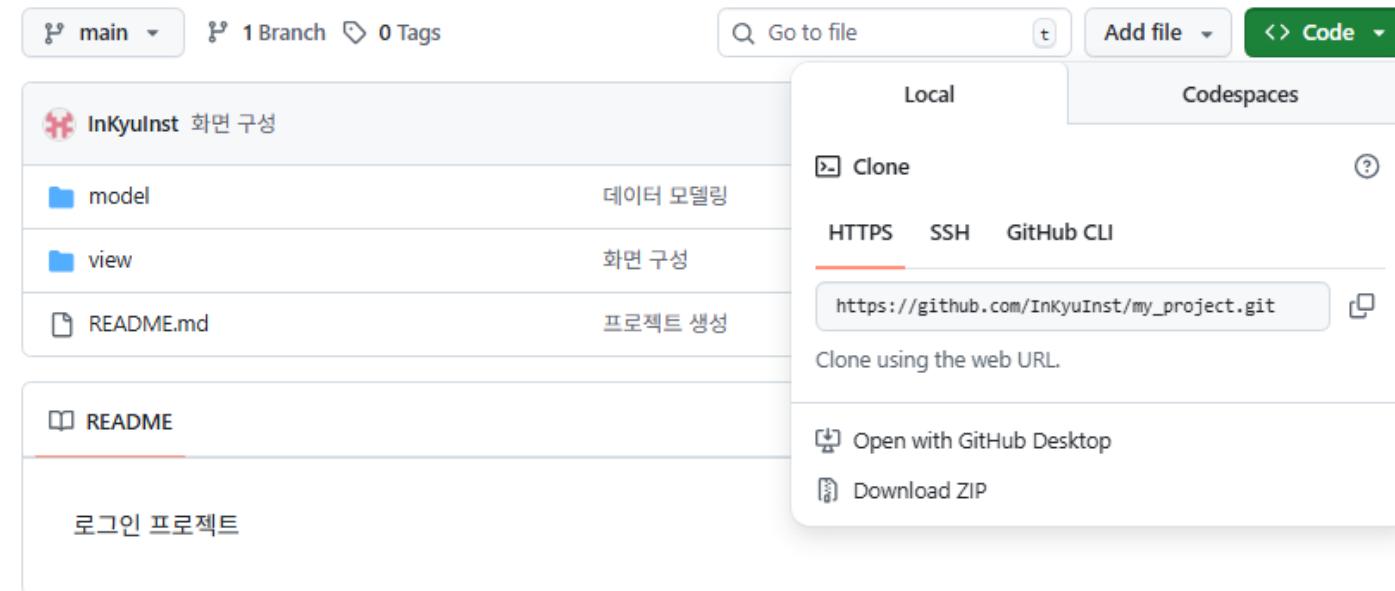
```
Username@DESKTOP MINGW64 ~/Desktop/company/view (main)
$ echo "<html> <h1>로그인 화면</h1> </html>" >> view.html

$ git add .
$ git commit -m "화면 구성"
$ git push
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/home  
$
```





# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/home
$ git clone https://github.com/사용자이름/my_project.git

$ ls
my_project/

$ cd my_project

Username@DESKTOP MINGW64 ~/Desktop/home/my_project (main)
$ ls
model/ README.md view/
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/home/my_project (main)
$ mkdir controller
$ cd controller

Username@DESKTOP MINGW64 ~/Desktop/home/my_project/controller (main)
$ echo "로그인 가능" >> function.txt

$ git add .

$ git commit -m "기능 구현"

$ git push
```



# Git & GitHub 기초 (CLI)

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ 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), 264 bytes | 20.00 KiB/s, done.
From https://github.com/사용자이름/my_project
  e5e3f2f..03d5260 main      -> origin/main
Updating e5e3f2f..03d5260
Fast-forward
  controller/function.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 controller/function.txt
```

# Git & GitHub 기초 (CLI)

작업 중인 디렉토리  
(Working Directory)

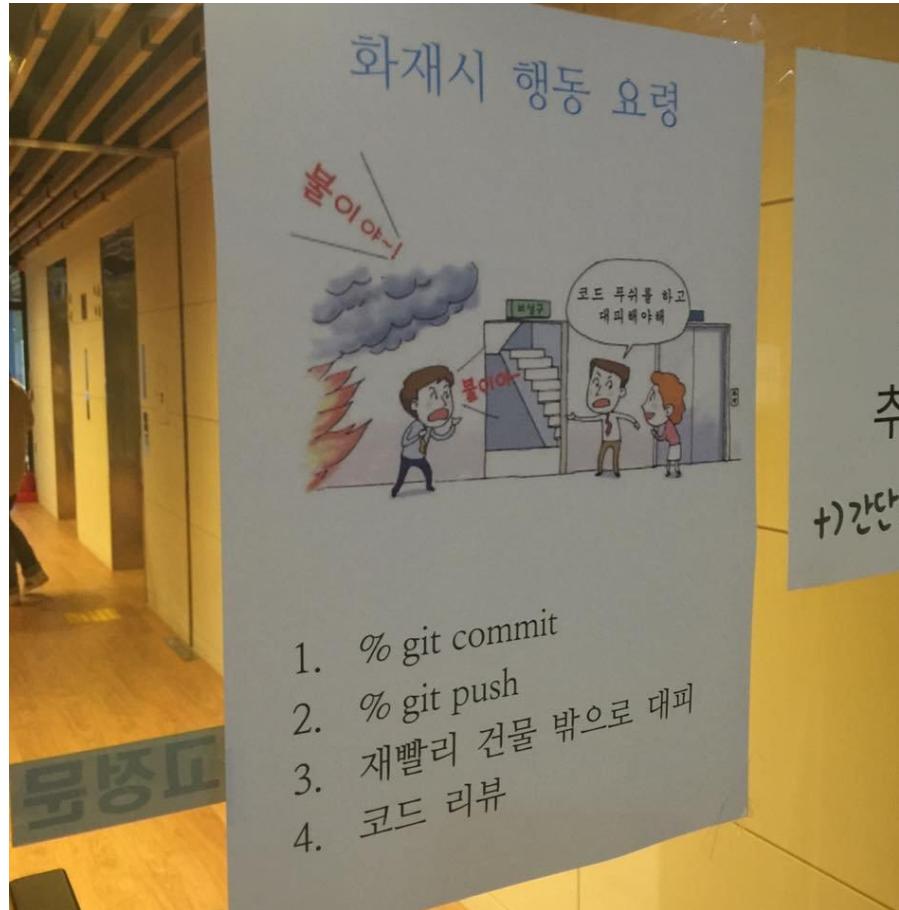
Staging Area

로컬 저장소  
(Local Repository)

원격 저장소  
(Remote Repository)



# Git & GitHub 기초 (CLI)

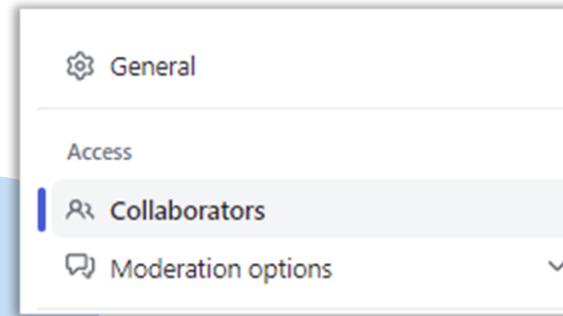


# Git & GitHub 기초 (CLI)



제 원격 저장소를 이용해 작업하시죠!  
Collaborator로 추가해놓을게요.  
이메일 확인해주세요!

Alice



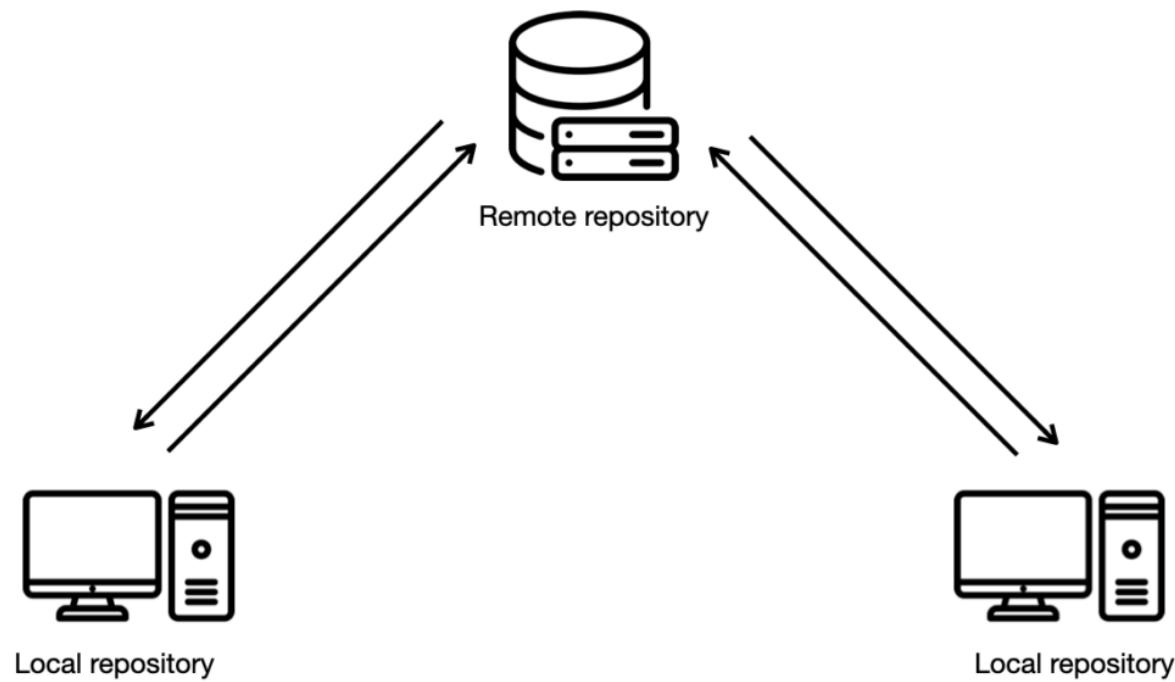
네! 감사합니다. clone해서 작업하겠습니다.



Bob

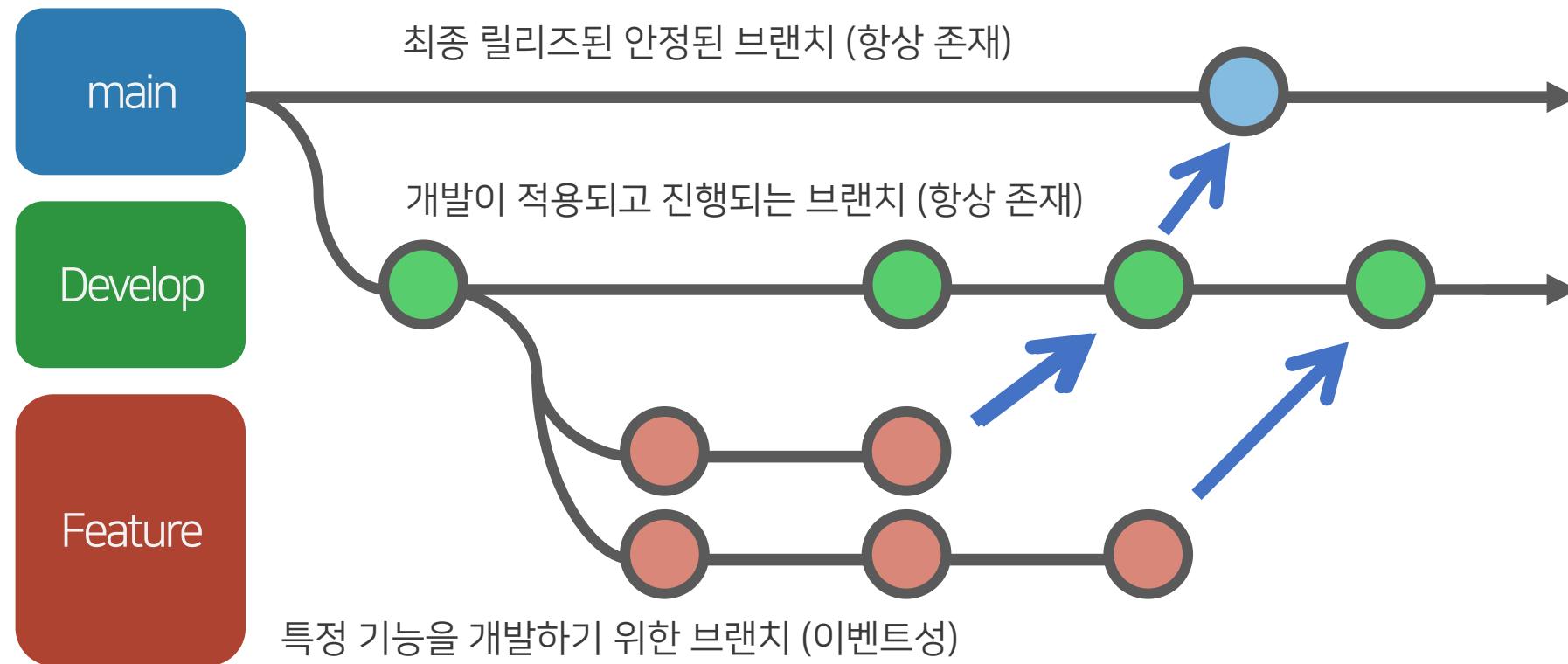


# Git & GitHub 심화 - Branch, 충돌(conflict) 관리



# Git & GitHub 심화 - Branch, 충돌(conflict) 관리

Branch를 활용하면 충돌을 최소화하고 원활한 협업 환경을 조성할 수 있게 됩니다.



# Git & GitHub 심화 - Branch, 충돌(conflict) 관리

To Do

- 프로젝트 생성
- 데이터 모델링
- 화면 구성
- 기능 구현
- 추가 기능 구현

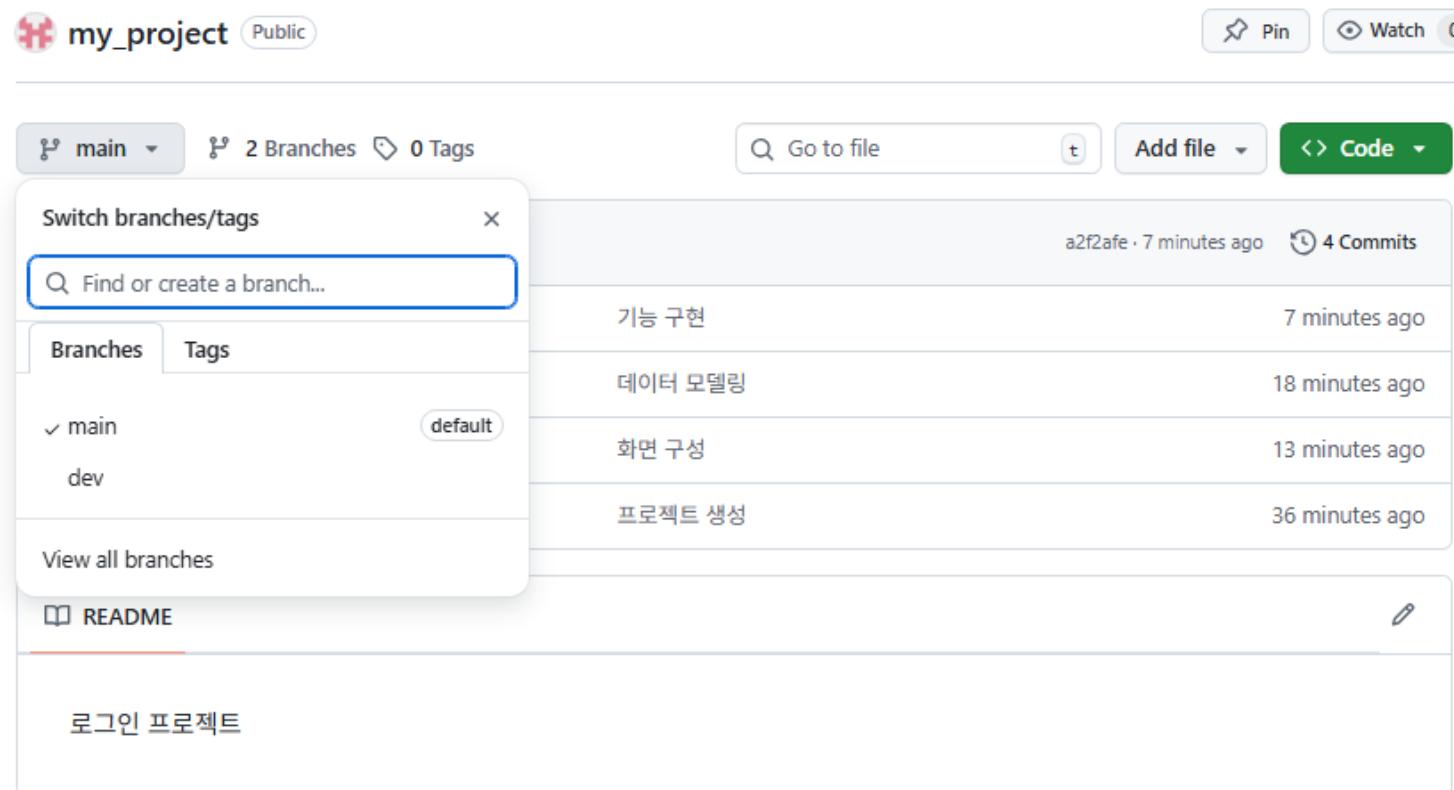
```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git switch -c dev
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ echo "로그아웃 기능" >> controller/function.txt
```

```
$ git commit -a -m "추가 기능 구현"
```

```
$ git push -u origin dev
```

# Git & GitHub 심화 - Branch, 충돌(conflict) 관리



The screenshot shows a GitHub repository named "my\_project" which is public. On the left, a modal dialog titled "Switch branches/tags" is open, showing a search bar and two tabs: "Branches" (selected) and "Tags". It lists two branches: "main" (marked with a checkmark and labeled "default") and "dev". On the right, the main repository page displays a commit history for the "main" branch. The commits are:

Commit Message	Time Ago
a2f2afe · 7 minutes ago	4 Commits
기능 구현	7 minutes ago
데이터 모델링	18 minutes ago
화면 구성	13 minutes ago
프로젝트 생성	36 minutes ago

Below the commit history, there is a "README" file and a link to "로그인 프로젝트".

# Git & GitHub 심화 - Branch, 충돌(conflict) 관리

```
Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ git switch main
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ echo "로그아웃 기능(로그아웃 시 메인페이지로 이동)" >> controller/function.txt

$ git commit -a -m "추가 기능 구현"

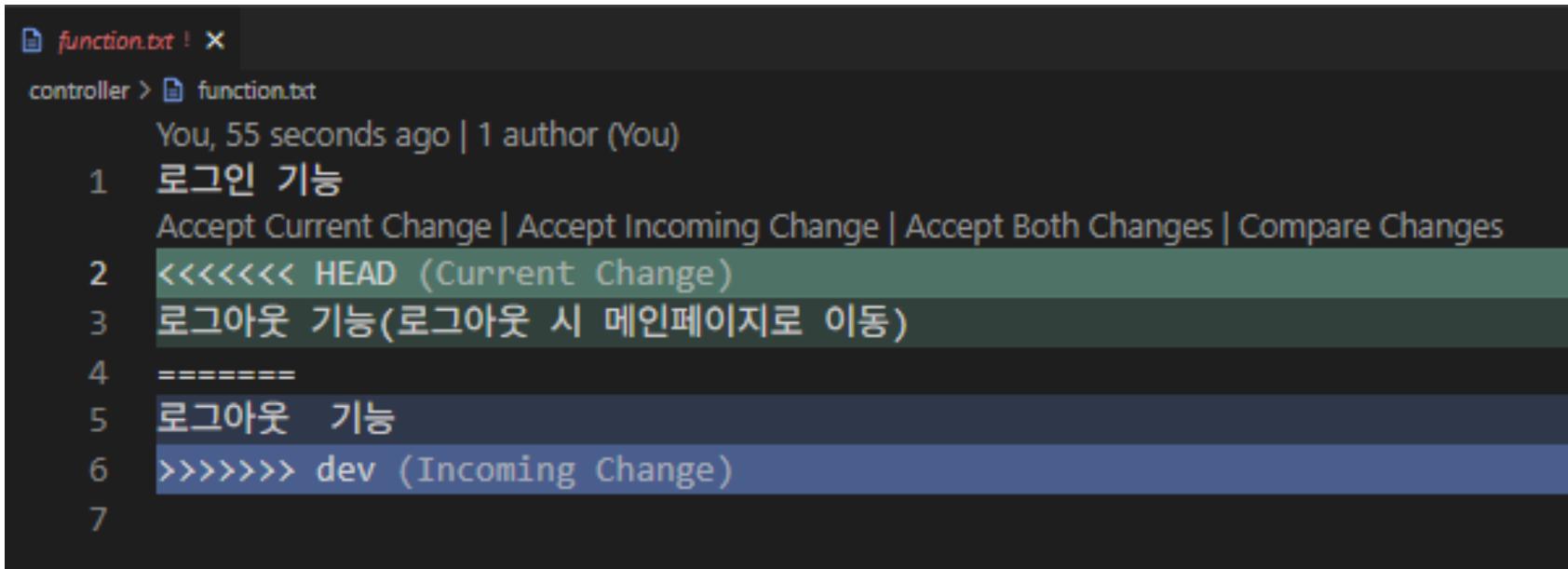
$ git push
```

# Git & GitHub 심화 - Branch, 충돌(conflict) 관리

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git merge dev
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ cat controller/function.txt
로그인 기능
<<<<<< HEAD
로그아웃 기능(로그아웃 시 메인페이지로 이동)
=====
로그아웃 기능
>>>>> dev
```

# Git & GitHub 심화 - Branch, 충돌(conflict) 관리



A screenshot of a terminal window showing a merge conflict in a file named `function.txt`. The file is located in a directory structure: `controller > function.txt`. The commit information shows it was committed by the user 55 seconds ago with one author. The conflict is displayed with line numbers 1 through 7. Lines 1, 3, 5, and 7 are in black, while lines 2 and 6 are highlighted in green and blue respectively, indicating they are from different branches.

```
function.txt ! ✘
controller > function.txt
You, 55 seconds ago | 1 author (You)
1 로그인 기능
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<< HEAD (Current Change)
3 로그아웃 기능(로그아웃 시 메인페이지로 이동)
4 =====
5 로그아웃 기능
6 >>>>> dev (Incoming Change)
7
```

# Git & GitHub 심화 - Branch, 충돌(conflict) 관리

```
Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ git commit -a -m "충돌 해결"
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git push
```



# Git & GitHub 심화 - Pull Request

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git switch dev
```

```
Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ git pull origin main
$ git push
```



# Git & GitHub 심화 - Pull Request

```
Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ echo "획기적인 기능" >> controller/function.txt

$ git commit -a -m "획기적인 기능 구현"

$ git push
```



# Git & GitHub 심화 (GUI)

The screenshot shows the GitHub desktop application's main interface. The top navigation bar includes links for Code, Issues, Pull requests (which is the active tab), Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation bar are filters for Labels (9) and Milestones (0), and a prominent green 'New pull request' button. The main content area displays a message: 'Welcome to pull requests!'. It explains that pull requests help collaborate on code and provides a link to 'create a pull request'.

<> Code    ⚡ Issues    🛡 Pull requests    ⏱ Actions    📈 Projects    📖 Wiki    🛡 Security    🔍 Insights    🌐 Settings

Filters ▾  Labels 9 Milestones 0 New pull request

\_PULL REQUESTS\_ 0 Open ✓ 0 Closed

Welcome to pull requests!

Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#).

# Git & GitHub 심화 (GUI)

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base: main ▾ compare: dev ▾ ✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) Create pull request

-o 1 commit      1 file changed      1 contributor

-o Commits on Oct 23, 2025

**획기적인 기능 구현**  
InKyulInst committed 1 minute ago 79d6e47

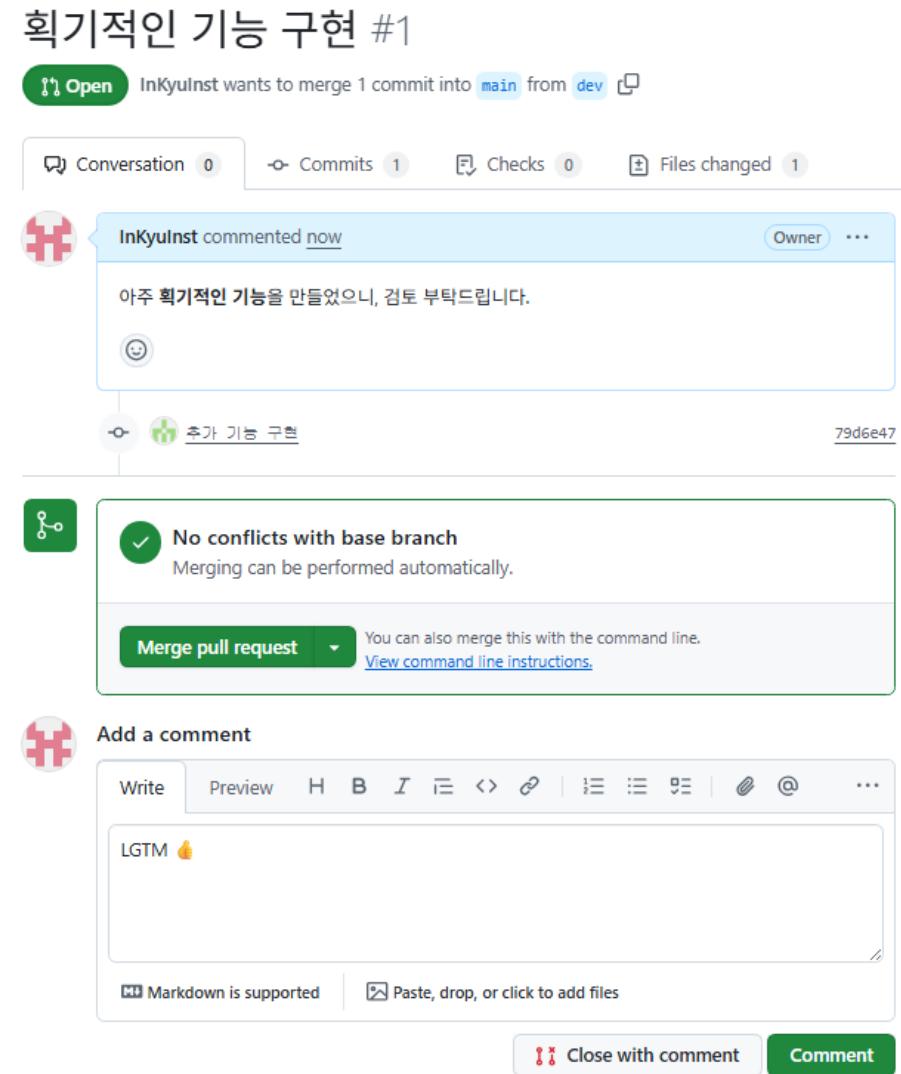
Showing 1 changed file with 1 addition and 0 deletions. Split Unified

...	...	@@ -1,2 +1,3 @@
1	1	로그인 가능
2	2	로그아웃 가능(로그아웃 시 메인페이지로 이동)
3	+ 흑기적인 기능	



# Git & GitHub 심화 (GUI)

# Git & GitHub 심화 (GUI)



# 실전을 통해 Git 정복하기

MarkDown은 텍스트 기반의 마크업 언어로, 문서를 포맷팅하고 스타일을 지정하는 간단한 방법을 제공합니다.  
Github과 같은 버전 관리 플랫폼에서 주로 사용됩니다.



# 실전을 통해 Git 정복하기

MarkDown의 기본 문법 알아보기

# 큰 제목

**큰 제목**

## 작은 제목

**작은 제목**

1. 순서가 있는 목록 : 1.을 누르고 스페이스바를 눌러 생성,
  - 순서가 없는 목록 : -(하이픈)을 쓰고 스페이스바를 눌러 생성.

\*tab키를 이용해 하위 항목을 생성할 수 있고, shift + tab키를 눌러서 상위 항목으로 이동 가능

# 실전을 통해 Git 정복하기

MarkDown의 기본 문법 알아보기

## 코드 블록

인라인 코드 블록 : 인라인 블럭으로 처리하고 싶은 부분을 ` (백틱)으로 감싸줍니다.

코드 블록 : ` (백틱) 을 3번 입력하고 Enter를 눌러 생성

# 실전을 통해 Git 정복하기

MarkDown의 기본 문법 알아보기

링크

[]()를 작성하고 () 안에 링크 주소를 작성하고 []안에 어떤 링크 주소인지 입력

이미지

![]()를 작성하고, () 안에는 이미지 주소를 입력

# 실전을 통해 Git 정복하기

MarkDown의 기본 문법 알아보기

표

|(파이프) 사이에 컬럼을 작성하고 엔터를 입력

마지막 컬럼을 작성하고 |(파이프) 입력

강조

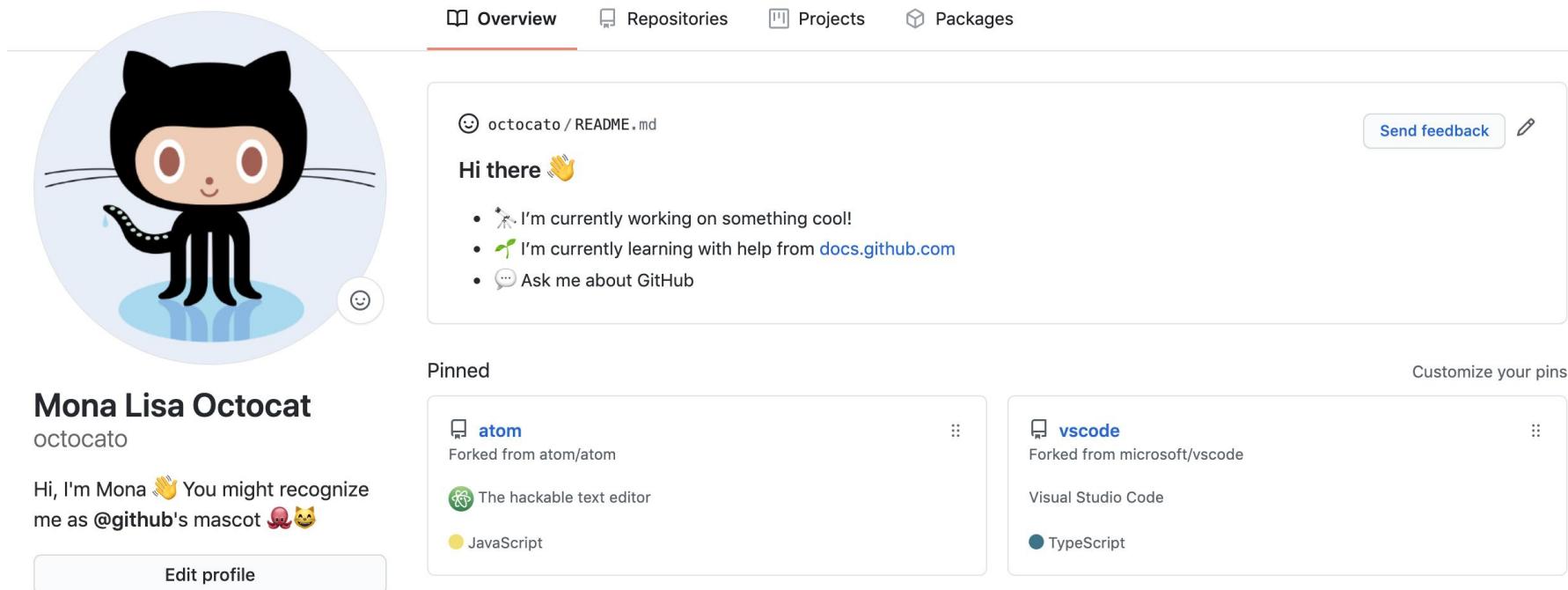
이탤릭체 : \* 혹은 \_로 감싸준다.

볼드체 : \*\* 또는 \_\_로 감싸준다.

취소선: ~~으로 앞뒤를 감싸준다.

# 실전을 통해 Git 정복하기

자신의 UserName과 동일한 저장소를 생성하고, README.md 파일을 꾸미면...



The screenshot shows a GitHub user profile for the account 'octocato'. The profile features a large circular icon of the GitHub Octocat mascot. Below the icon, the name 'Mona Lisa Octocat' and the handle 'octocato' are displayed. A bio message says: 'Hi, I'm Mona 🎉 You might recognize me as @github's mascot 🐙😺'. A 'Edit profile' button is at the bottom. The main content area shows the 'Overview' tab selected, with tabs for 'Repositories', 'Projects', and 'Packages' below it. The 'README' section contains the following text:

```
Hi there 🎉  
• 🚀 I'm currently working on something cool!  
• 🌱 I'm currently learning with help from docs.github.com  
• 💬 Ask me about GitHub
```

There is a 'Send feedback' button with a pencil icon. Below this, under the heading 'Pinned', are two repository cards: 'atom' (Forked from atom/atom) and 'vscode' (Forked from microsoft/vscode). To the right, there is a 'Customize your pins' section.

# 실전을 통해 Git 정복하기

개발 프로젝트의 README.md는 어떻게 구성하면 좋을까?

 New Features

 Bug Fixes

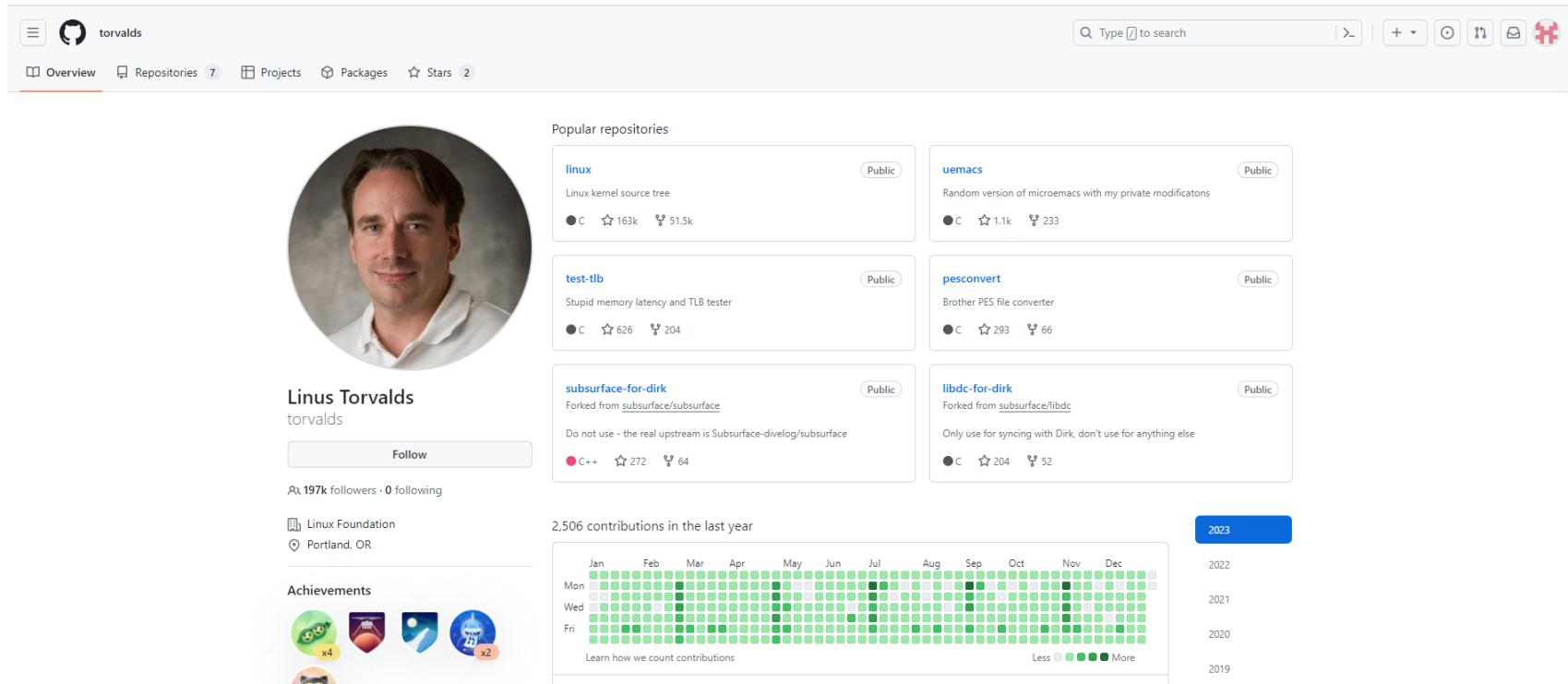
 Documentation

 Dependency Upgrades

 Contributors

# 실전을 통해 Git 정복하기

Github의 잔디는 성실한 개발자의 척도가 되어준다.



The screenshot shows Linus Torvalds' GitHub profile page. At the top, there's a navigation bar with 'Overview' (highlighted in red), 'Repositories' (7), 'Projects', 'Packages', and 'Stars' (2). A search bar is at the top right. Below the header, there's a large circular profile picture of Linus Torvalds. His name 'Linus Torvalds' and GitHub handle 'torvalds' are displayed. A 'Follow' button is present. To the left, it says '197k followers · 0 following'. Below this, it lists 'Linux Foundation' and 'Portland, OR'. Under 'Achievements', there are four small circular icons: a green one with 'x4', a purple one with a shield, a blue one with a gear, and a yellow one with 'x2'. On the right, under 'Popular repositories', there are six boxes: 'linux' (Public, C, 163k stars, 51.5k forks), 'uemacs' (Public, C, 1.1k stars, 233 forks), 'test-tlb' (Public, C, 626 stars, 204 forks), 'pesconvert' (Public, C, 293 stars, 66 forks), 'subsurface-for-dirk' (Public, C++, 272 stars, 64 forks), and 'libdc-for-dirk' (Public, C, 204 stars, 52 forks). At the bottom, there's a heatmap titled '2,506 contributions in the last year' showing activity by month and day. A legend at the bottom right indicates contribution levels: 'Less' (light green), 'More' (dark green), and 'More' (black).

# 실전을 통해 Git 정복하기

협업에 알맞게, 커뮤니케이션에 유용하게 Git을 사용하기 위해서는 Commit Convention을 이용하는 것이 좋다.

Feat	새로운 기능을 추가
Fix	버그 수정
Style	CSS 등 사용자 UI 디자인 변경
HOTFIX	급하게 치명적인 버그를 고쳐야 하는 경우
Refactor	프로덕션 코드 리팩토링
Comment	필요한 주석 추가 및 변경
Docs	문서 수정
Rename	파일 혹은 폴더명을 수정하거나 옮기는 작업만인 경우
Remove	파일을 삭제하는 작업만 수행한 경우

# 실전을 통해 Git 정복하기

Code Review는 코드를 기반으로 피드백을 주고 받는 과정입니다.

기술 부채를 줄이고, 코드의 부작용과 오류를 사전에 찾아낼 수 있도록 해줍니다.

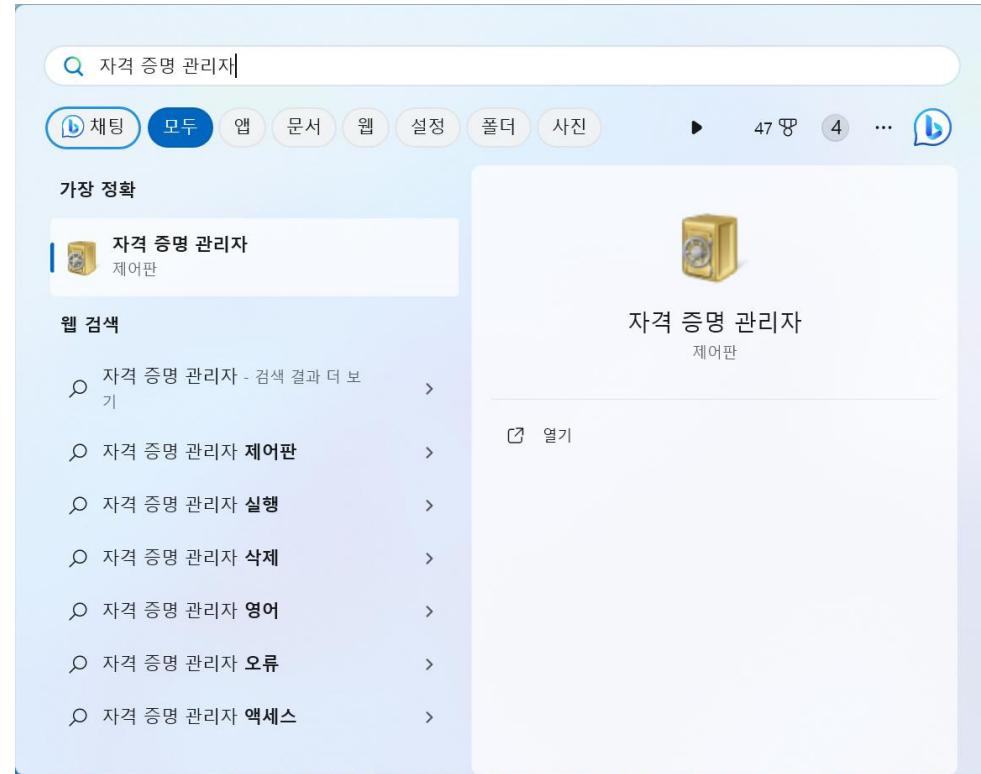
만약 리뷰할 게 없다면? 칭찬!!! 

AFAIK	- "As Far As I Know"
FYI	- "For Your Information"
GOTCHA	- "I've Got You"
IMO (IMHO)	- "In My (Humble) Opinion"
LGTM	- "Looks Good To Me"
SSIA	- "Subject Says It All"
TIA	- "Thanks In Advance"



# THANKS

<https://open.kakao.com/o/sXqQ7wNf>  
강사최인규



# Git & GitHub 기초 (CLI)

자격 증명 관리

웹 사이트, 연결된 응용 프로그램 및 네트워크에 대해 저장된 로그온 정보를 보고 삭제합니다.



웹 자격 증명



## Windows 자격 증명

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

사용자 이름:

InkyuInst

암호: