

# Git & GitHub

Git & GitHub 개념 및 협업

김태민

저번에 우리는

## 반응형 웹 디자인

다양한 기기(스마트폰, 태블릿, 데스크톱)와 화면 크기에 맞춰  
웹 페이지의 레이아웃, 콘텐츠, 기능을 최적화하는 설계 접근법

### 핵심 구성 요소:

- 유연한 레이아웃: 상대 단위(% , vw, vh, rem, em) 사용
- 미디어 쿼리: 화면 크기나 기기 특성에 따라 스타일 조정
- 반응형 이미지: 기기 해상도에 맞는 이미지 제공

모든 사용자에게 일관되고 최적화된 경  
험 제공

## 미디어 쿼리

CSS의 @media 규칙을 사용해 화면 크기, 기기 특성, 해상도 등 특정 조건에 따라 스타일을 적용하는 기술

반응형 웹 디자인의 핵심으로, 다양한 기기(모바일, 태블릿, 데스크톱)와 환경에 맞는 스타일링 제공

### 목적:

- 사용자 경험(UX) 최적화: 기기에 맞는 레이아웃 제공
- 접근성 강화: 다양한 화면 크기와 환경 지원
- 성능 개선: 조건에 따라 최적화된 리소스 로드

## 미디어 쿼리 - 화면크기

767px

```
@media (max-width: 767px) {  
  .box {  
    width: 33.33%;  
  }  
}
```

**max-width:**

화면 너비가 특정 값 이하일 때

1024px

```
@media (min-width: 768px) and (max-width: 1023px){  
  .box {  
    width: 33.33%;  
  }  
}
```

범위 지정

```
@media (min-width: 1024px) {  
  .box {  
    width: 33.33%;  
  }  
}
```

**min-width:**

화면 너비가 특정 값 이상일 때

## 상대 단위

CSS에서 요소의 크기나 간격을 상대적인 기준(부모, 뷰포트, 폰트 크기 등)에 따라 설정하는 단위

**역할:** 반응형 웹 디자인에서 유연하고 동적인 레이아웃 구현, 다양한 기기와 화면 크기에 적응

**주요 상대 단위:** %, vw, vh, em, rem

**장점:**

- 고정 단위(px)와 달리, 화면 크기나 폰트 크기 변화에 따라 자동 조정
- 접근성과 유지보수성 향상

## 모바일 퍼스트:

- 기본 스타일을 모바일(작은 화면)에 맞춰 작성
- 미디어 쿼리(min-width)를 사용해 더 큰 화면에 스타일을 추가

## 데스크톱 퍼스트:

- 기본 스타일을 데스크톱(큰 화면)에 맞춰 작성
- 미디어 쿼리(max-width)를 사용해 작은 화면에 스타일을 조정

실습 환경



OS : Window / Mac

브라우저 : Chrome

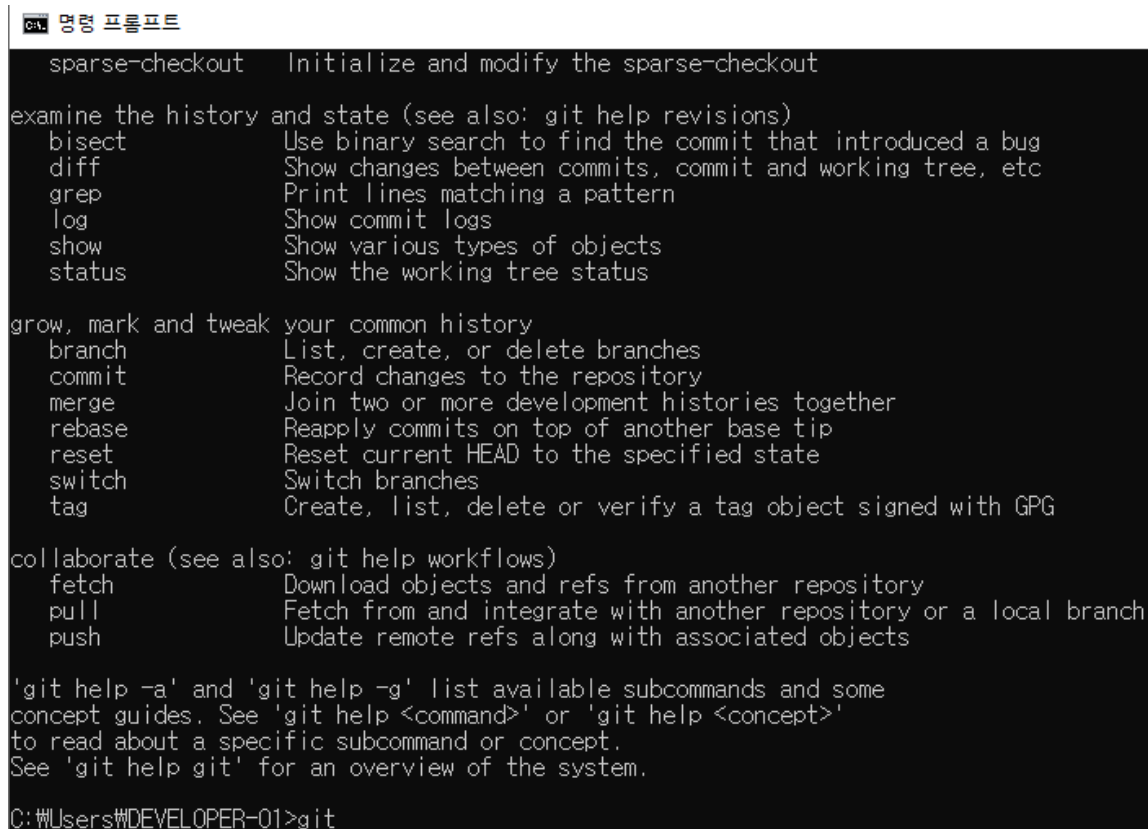
에디터 : VS Code <https://code.visualstudio.com/>

VS Code 익스텐션 : ESLint, Prettier, HTML CSS Support, HTML to CSS autocompletion, Auto Rename Tag, Auto Close Tag, htmltagwrap

**Git : git bash, github 가입**

내 컴퓨터에 git이 이미 설치되어 있는지 확인

- Windows : CMD(명령 프롬프트) 열고 git 입력
- Mac : Terminal 열고 git 입력



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The output of the 'git' command is displayed, listing various subcommands and their descriptions. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\Users\DEVELOPER-01>git

sparse-checkout    Initialize and modify the sparse-checkout

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

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

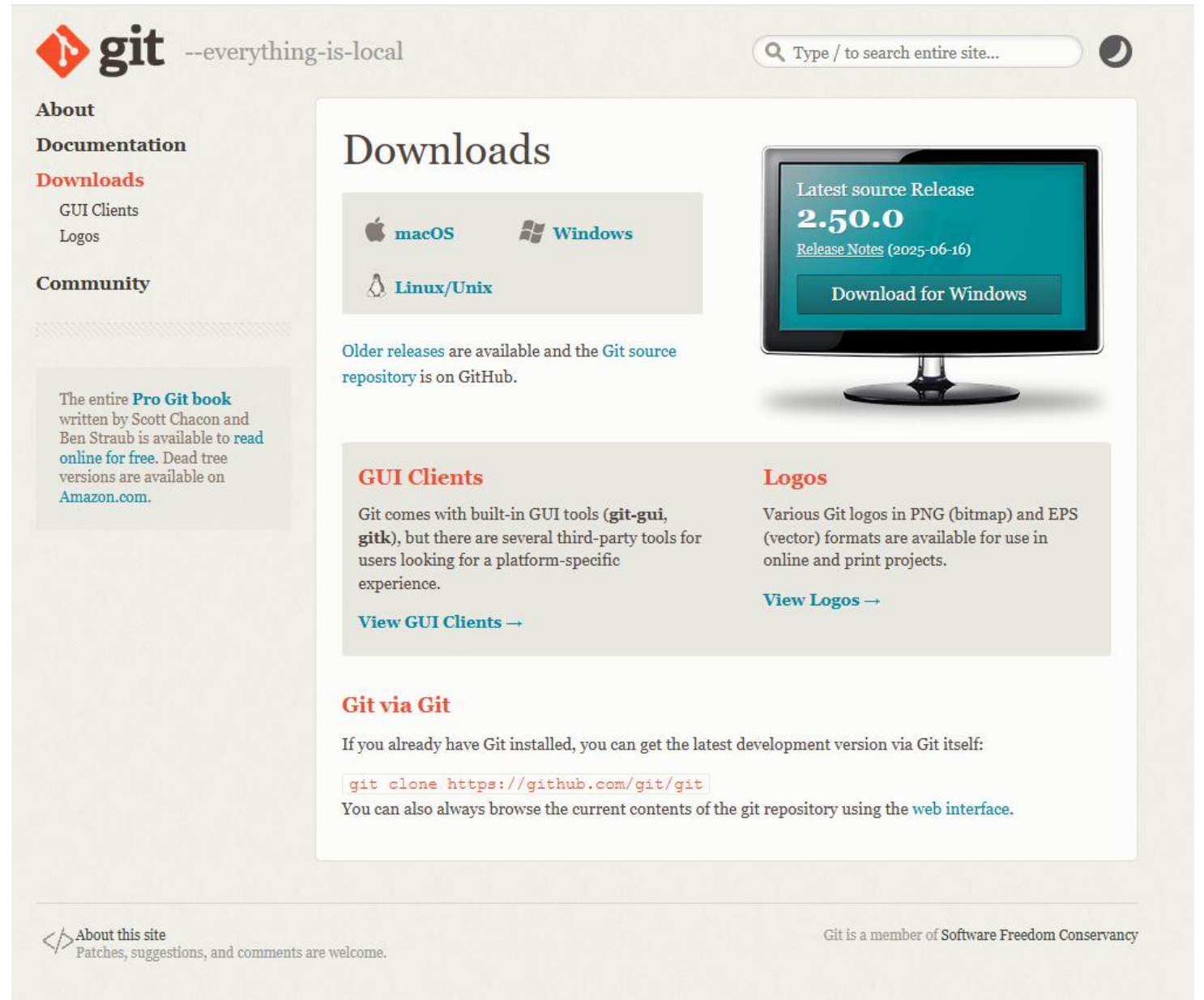
collaborate (see also: git help workflows)
  fetch           Download objects and refs from another repository
  pull            Fetch from and integrate with another repository or a local branch
  push           Update remote refs along with associated objects

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

C:\Users\DEVELOPER-01>
```

## Git, Git Bash 설치 (CLI 환경)

- <https://git-scm.com/>



The screenshot shows the Git website homepage. At the top, the Git logo is followed by the tagline "--everything-is-local". A search bar on the right contains the text "Type / to search entire site...". The left sidebar has a navigation menu with links for "About", "Documentation", "Downloads", "GUI Clients", "Logos", and "Community". The "Downloads" section is highlighted in red. Below the navigation menu, there is a box mentioning the "Pro Git book" by Scott Chacon and Ben Straub, available for free on Amazon.com. The main content area features a "Downloads" section with icons for macOS, Windows, and Linux/Unix. To the right of these icons is a monitor displaying the "Latest source Release 2.50.0" and a "Download for Windows" button. Below the download section, there is a note about older releases being available on GitHub. Further down, there are sections for "GUI Clients" and "Logos". The "GUI Clients" section mentions built-in tools like git-gui and gitk, and lists third-party tools. The "Logos" section mentions various Git logos in PNG and EPS formats. At the bottom, there is a "Git via Git" section with instructions on how to get the latest development version via Git itself, including a code snippet: `git clone https://github.com/git/git`. The footer contains a link to "About this site" and a statement that Git is a member of Software Freedom Conservancy.

**git** --everything-is-local

Type / to search entire site...

**About**  
**Documentation**  
**Downloads**  
GUI Clients  
Logos  
**Community**

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

## Downloads

macOS Windows Linux/Unix

Latest source Release  
**2.50.0**  
[Release Notes \(2025-06-16\)](#)  
Download for Windows

Older releases are available and the [Git source repository](#) is on GitHub.

### GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

### Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

### Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

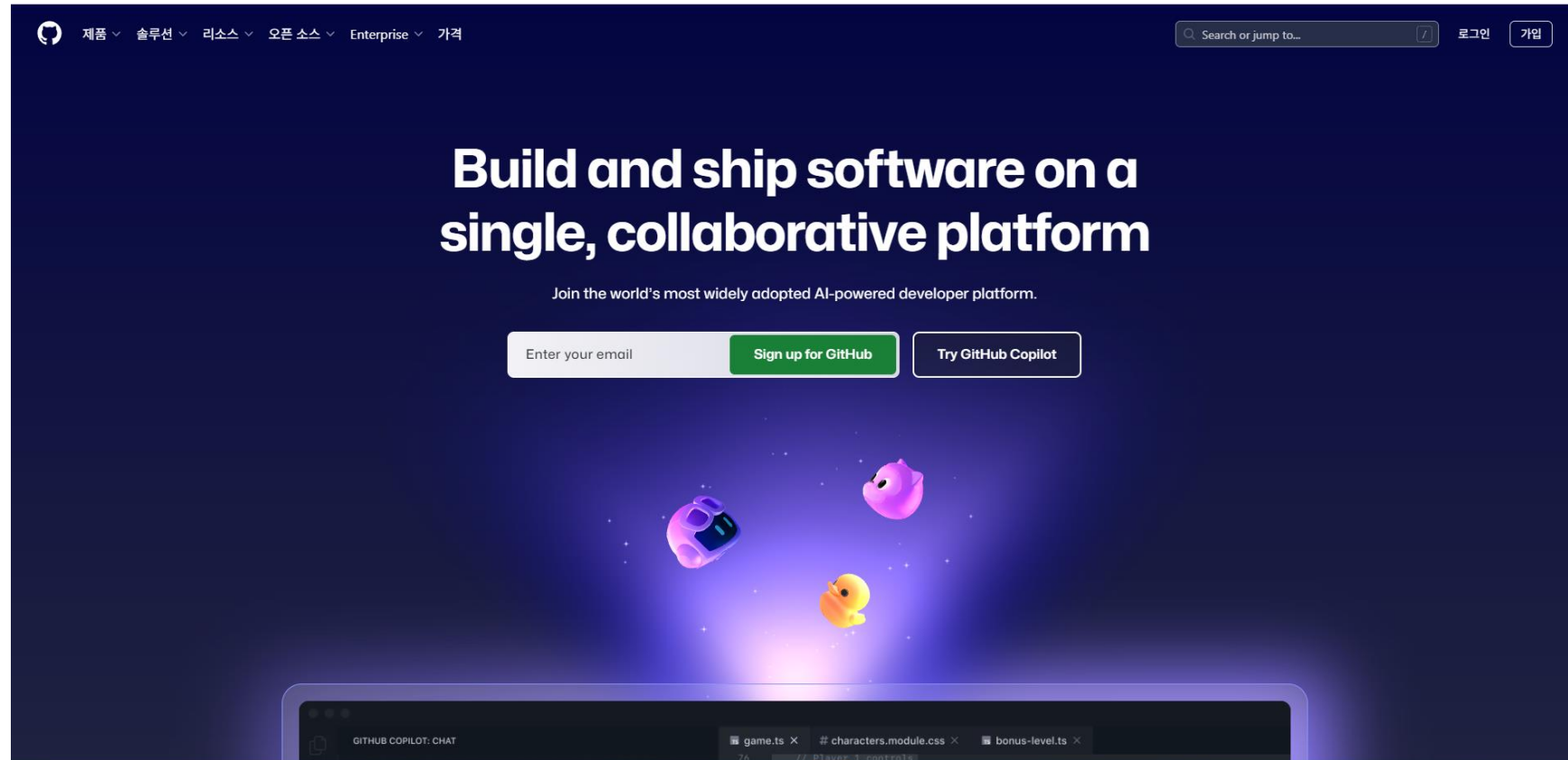
You can also always browse the current contents of the git repository using the [web interface](#).

About this site  
Patches, suggestions, and comments are welcome.

Git is a member of Software Freedom Conservancy

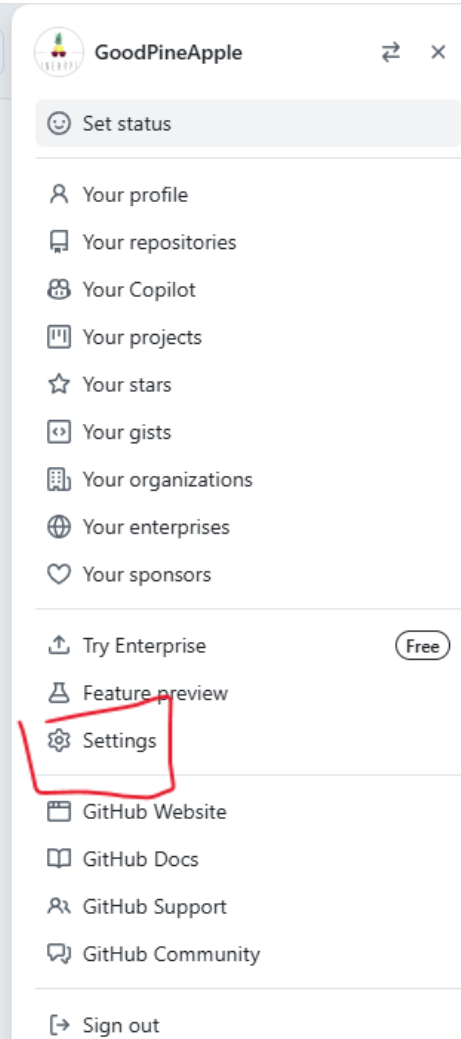
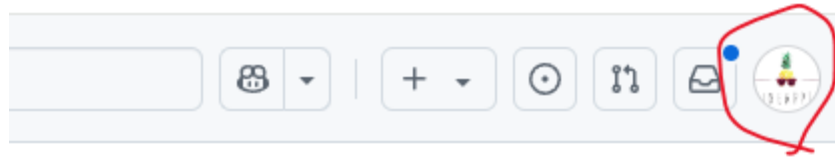
## Git Hub

- <https://github.com/>



## Git Hub

- <https://github.com/>
- 우측상단 프로필 – Settings



# 실습환경

## Git Hub

- <https://github.com/>
- 우측상단 프로필 – Settings
- 가장 하단 Developer Settings



GoodPineApple (GoodPineApple)

Your personal account [Switch settings context](#)

Public profile

- Account
- Appearance
- Accessibility
- Notifications

Access

- Billing and licensing
- Emails
- Password and authentication
- Sessions
- SSH and GPG keys
- Organizations
- Enterprises
- Moderation

Code, planning, and automation

- Repositories
- Codespaces
- Packages
- Copilot
- Pages
- Saved replies

Security

- Code security

Integrations

- Applications
- Scheduled reminders

Archives

- Security log
- Sponsorship log

<> Developer settings

## Public profile

Name

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

Public email

Select a verified email to display

You can manage verified email addresses in your [email settings](#)

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

- Link to social profile 1
- Link to social profile 2
- Link to social profile 3
- Link to social profile 4

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.

ORCID iD

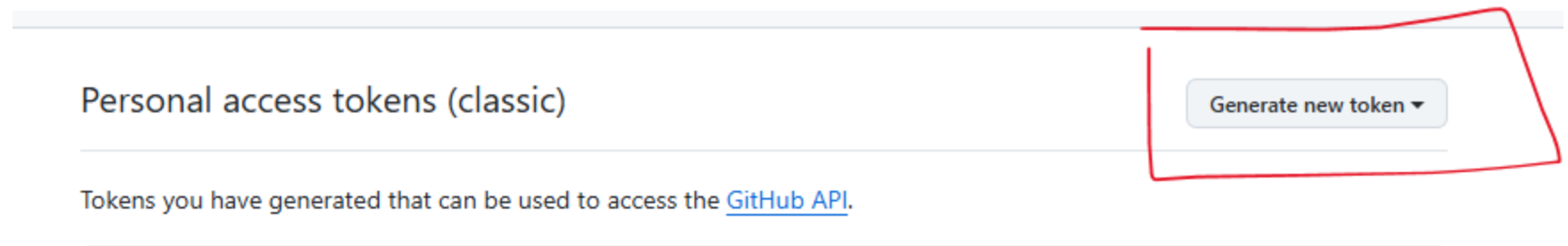
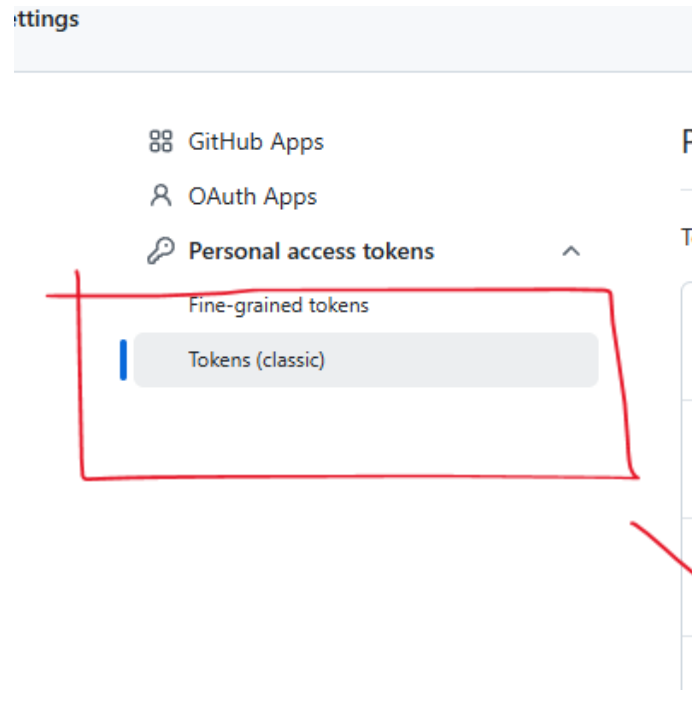
ORCID provides a persistent identifier - an ORCID iD - that distinguishes you from other researchers. Learn more at [ORCID.org](#).

Connect your ORCID iD

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.

## Git Hub

- <https://github.com/>
- 우측상단 프로필 – Settings
- 가장 하단 Developer Settings
- Personal Access Tokens (classic)



## Git Hub

- <https://github.com/>
- 우측상단 프로필 – Settings
- 가장 하단 Developer Settings
- Personal Access Tokens (classic)
- Repo 클릭후 생성

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

### New personal access token (classic)

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

**Note**

What's this token for?

**Expiration**

30 days (Jul 25, 2025)

The token will expire on the selected date

**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 type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<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
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org pro...
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

**Generate token** Cancel



# Git과 버전관리

## 버전관리 없는 개발

“어제 수정이 최신인가요?”

“이 파일의 원본이 뭐죠?”

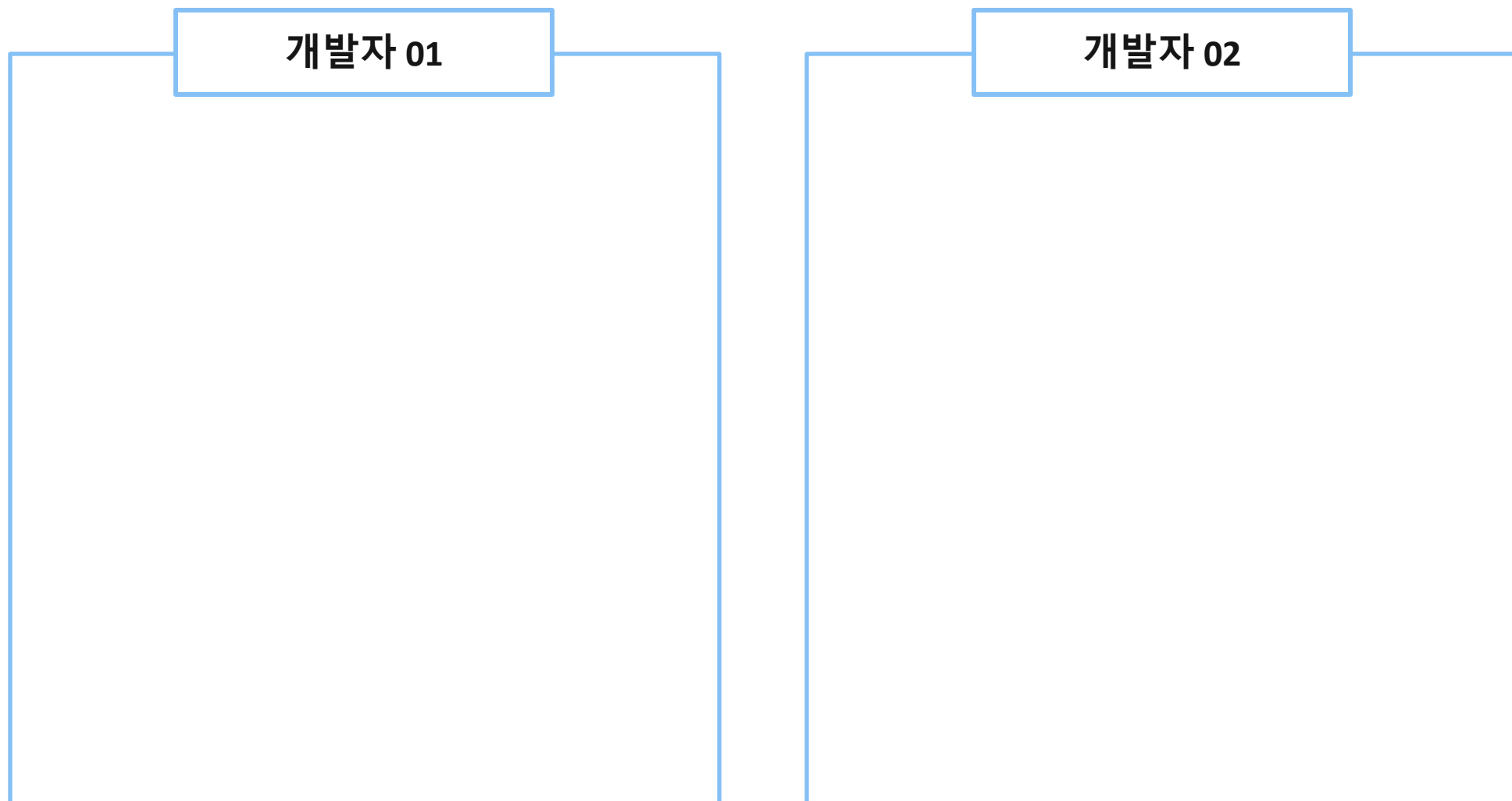
- 팀 작업 시 혼란
- 데이터 손실 위험

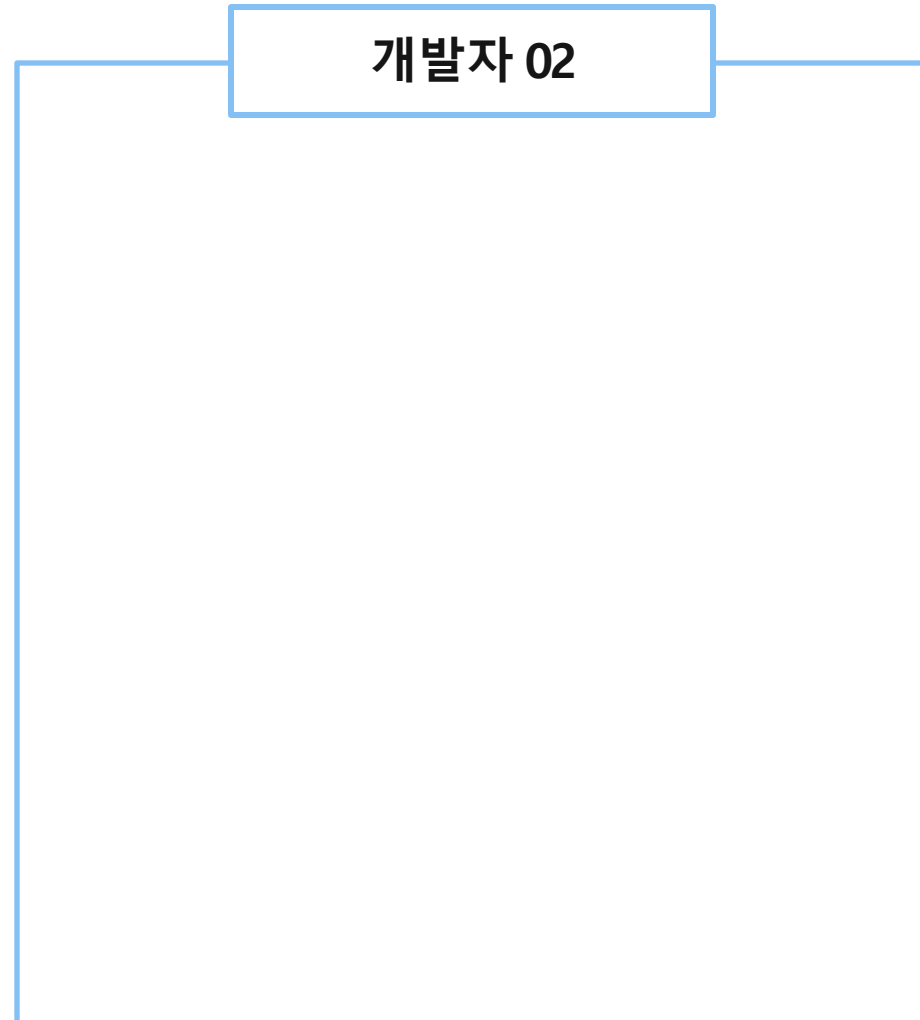
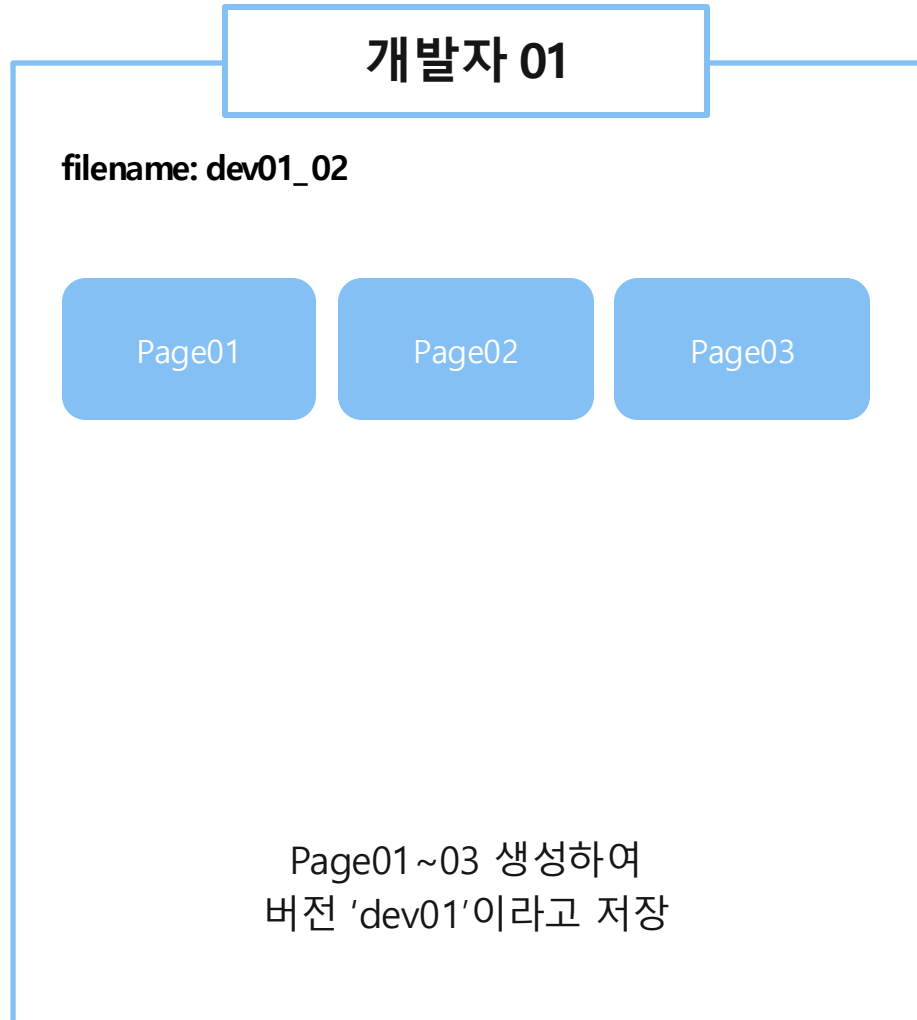


## 버전관리 Version Control System

파일 변경 이력을 기록하고 관리하는 방법

1. 협업 : 여러 사람이 동시에 작업 가능
2. 복구 : 실수해도 이전 버전으로 돌아감
3. 추적 : 누가 언제 무엇을 바꿨는지 확인 가능





## 개발자 01

filename: dev01\_02

Page01

Page02

Page03

## 개발자 02

filename: dev02

Page01

Page02

Page03

Page04

Page05

Page06

'dev01'을 다운받아서  
Page04~06 생성, 'dev02'로 저장

## 개발자 01

filename: dev01\_02

Page01

Page02

Page03

Page02 수정하여  
버전 'dev01\_02' 라고 저장

## 개발자 02

filename: dev02

Page01

Page02

Page03

Page04

Page05

Page06

## 개발자 01

filename: dev01\_02

Page01

Page02

Page03

## 개발자 02

filename: dev02

Page01

Page02

Page03

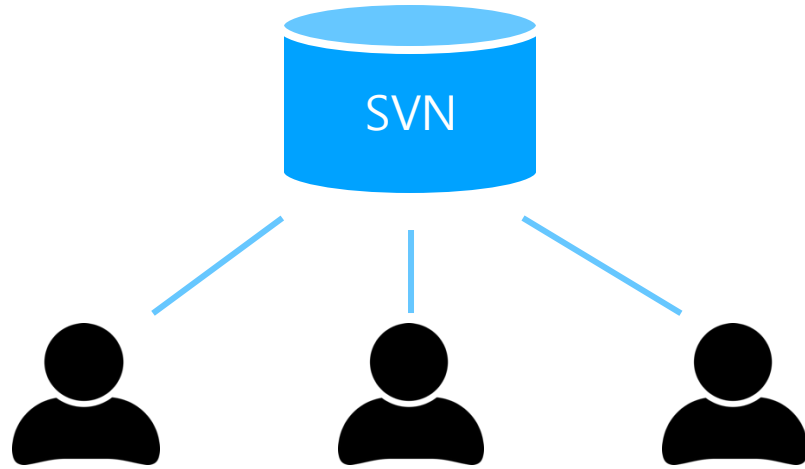
Page04

Page05

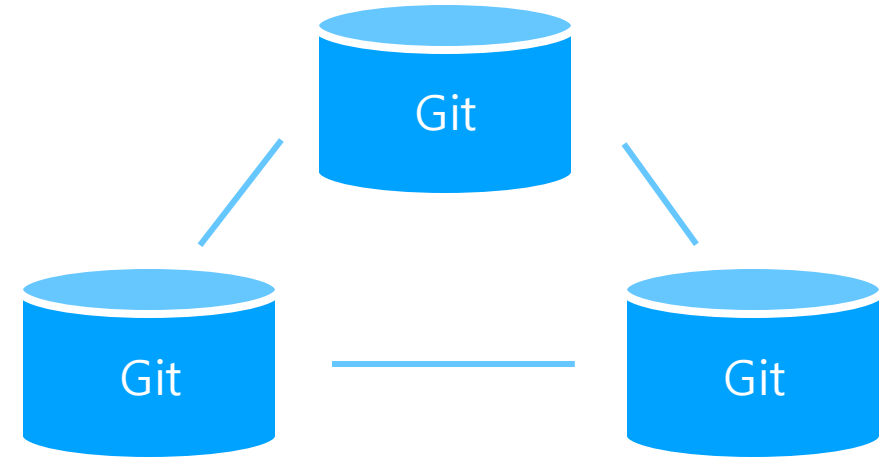
Page06

'dev01\_02'와 'dev02'를 비교(diff)  
Page02 수정사항을 반영





## Git의 탄생과 특징



### 탄생 :

2005년, 리누스 토르발스(리눅스 창시자)가 개발

### 기존 문제:

SVN(Subversion)은 서버가 고장 나면 작업 불가

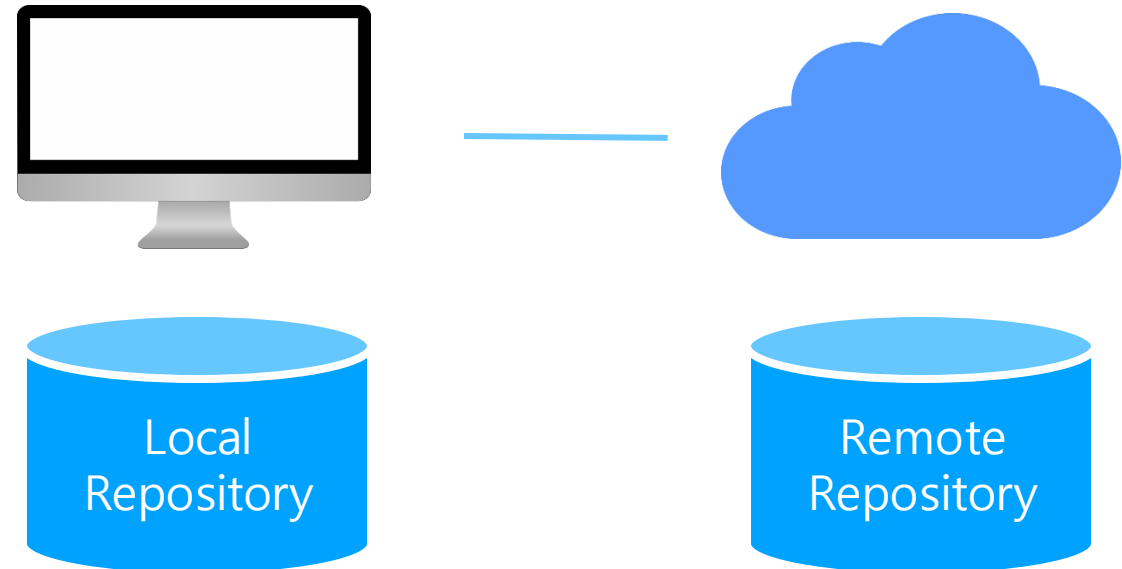
### Git의 특징:

- 분산형: 내 컴퓨터에 모든 이력 저장
- 빠르고 유연: 오프라인에서도 사용 가능

로컬/원격저장소

## 로컬저장소와 원격저장소

- ✓ Git의 두 가지 저장소
- ✓ 로컬은 내 컴퓨터에서 관리
- ✓ 원격은 온라인에 공유



## 로컬저장소 Local Repository

내 컴퓨터에 코드와 이력 저장. Git은 저장할 공간만 있으면 어디서나 사용 가능(pc, usb, cloud 등)

### 새 저장소 생성

```
git init
```

### 수정파일 준비 (추적대상 지정)

```
git add . or ${file name}
```

### 파일상태 확인

```
git status
```

### 변경사항 저장

```
git commit -m "${commit message}"
```

수정 → 준비 → 저장

01

**Working Directory**

파일 수정 단계

02

**Staging Area**

저장할 파일 선택

03

**Repository**

변경 이력 확정

Add

Commit

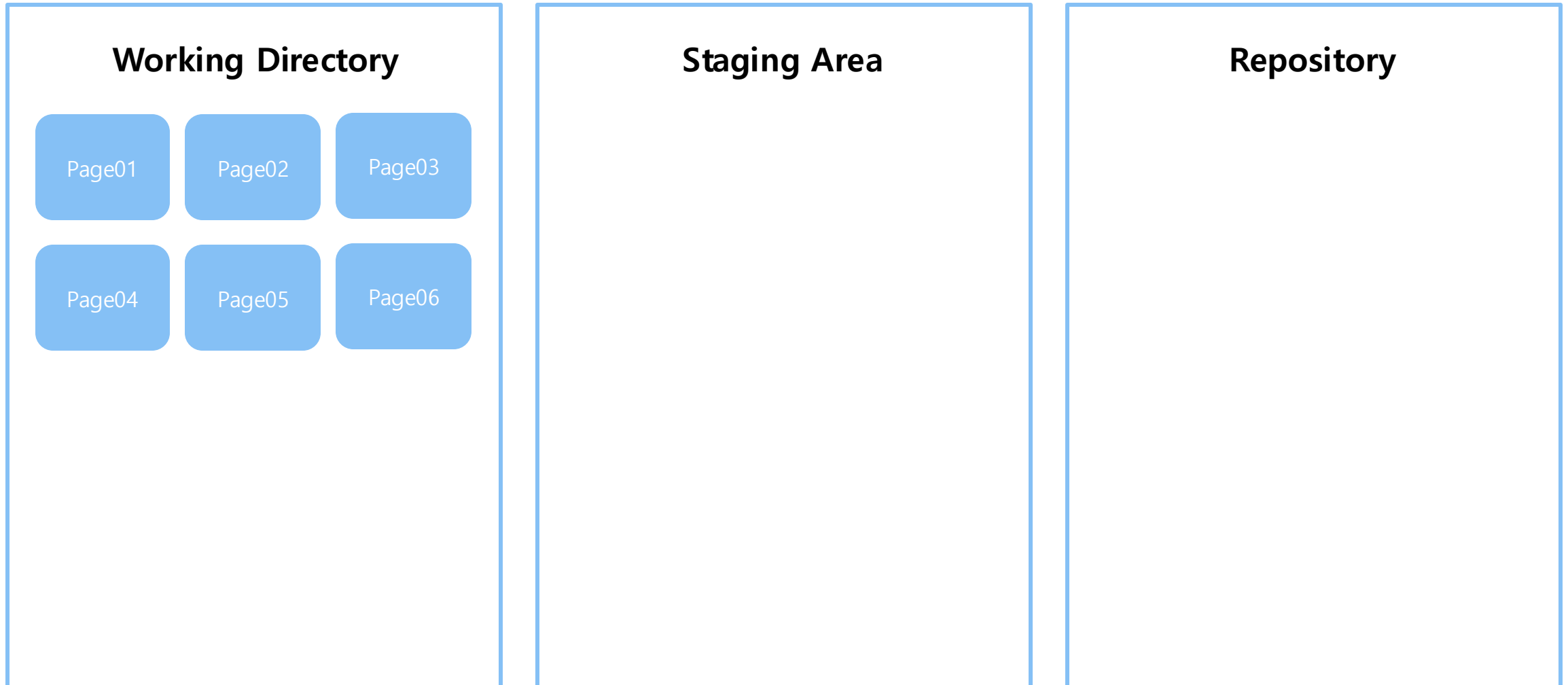
**git init**

**Working Directory**

**Staging Area**

**Repository**

## git init



**git add Page01 Page02**

## Working Directory

Page01

Page02

Page03

Page04

Page05

Page06

## Staging Area

Page01

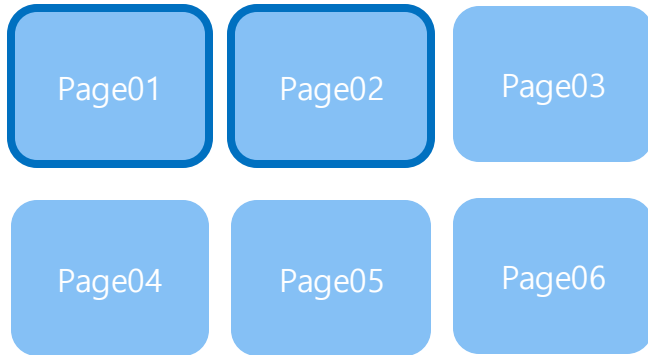
Page02

## Repository



**git commit -m "first commit"**

## Working Directory

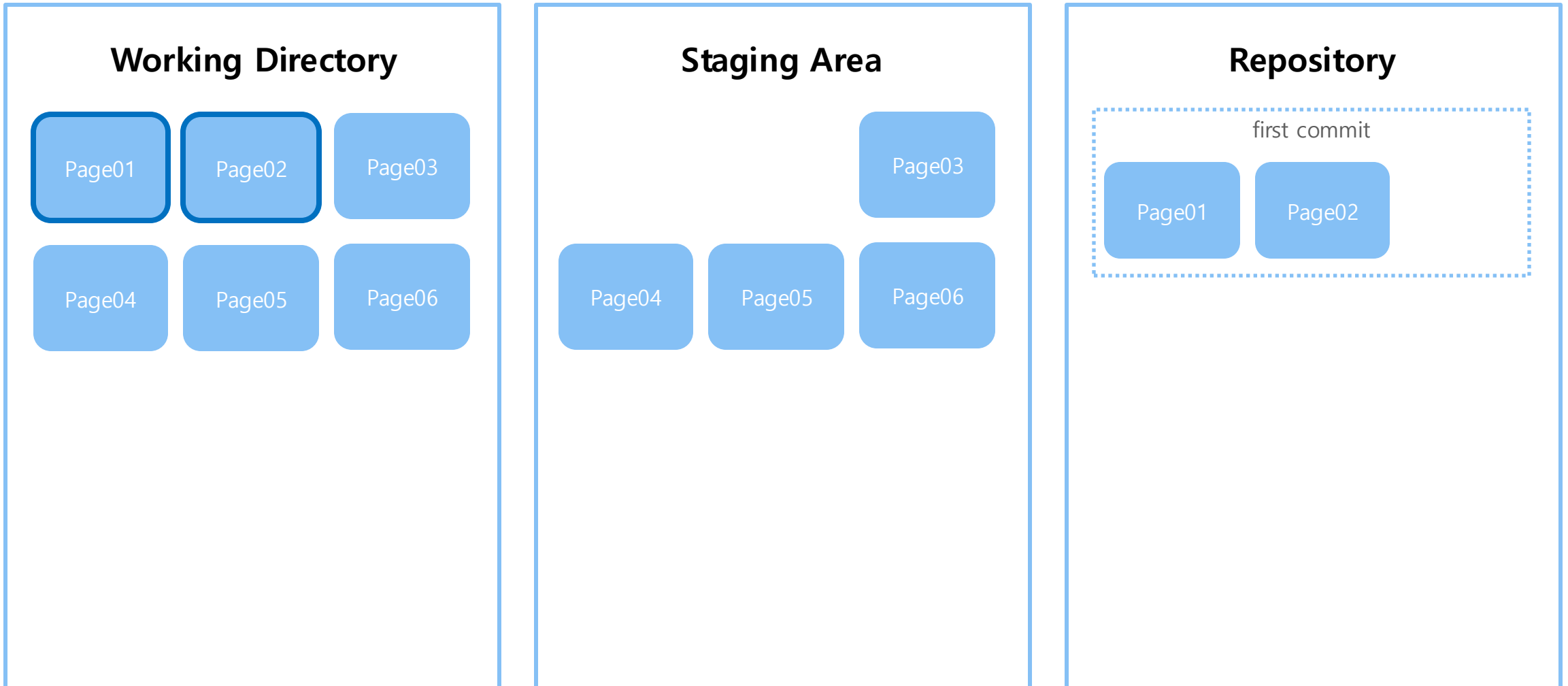


## Staging Area

## Repository

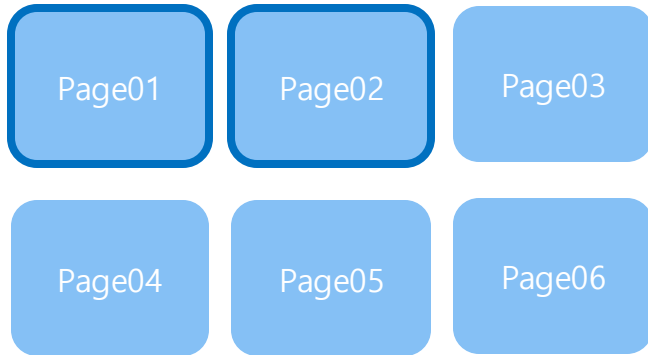


**git add .**



**git commit -m "second commit"**

## Working Directory



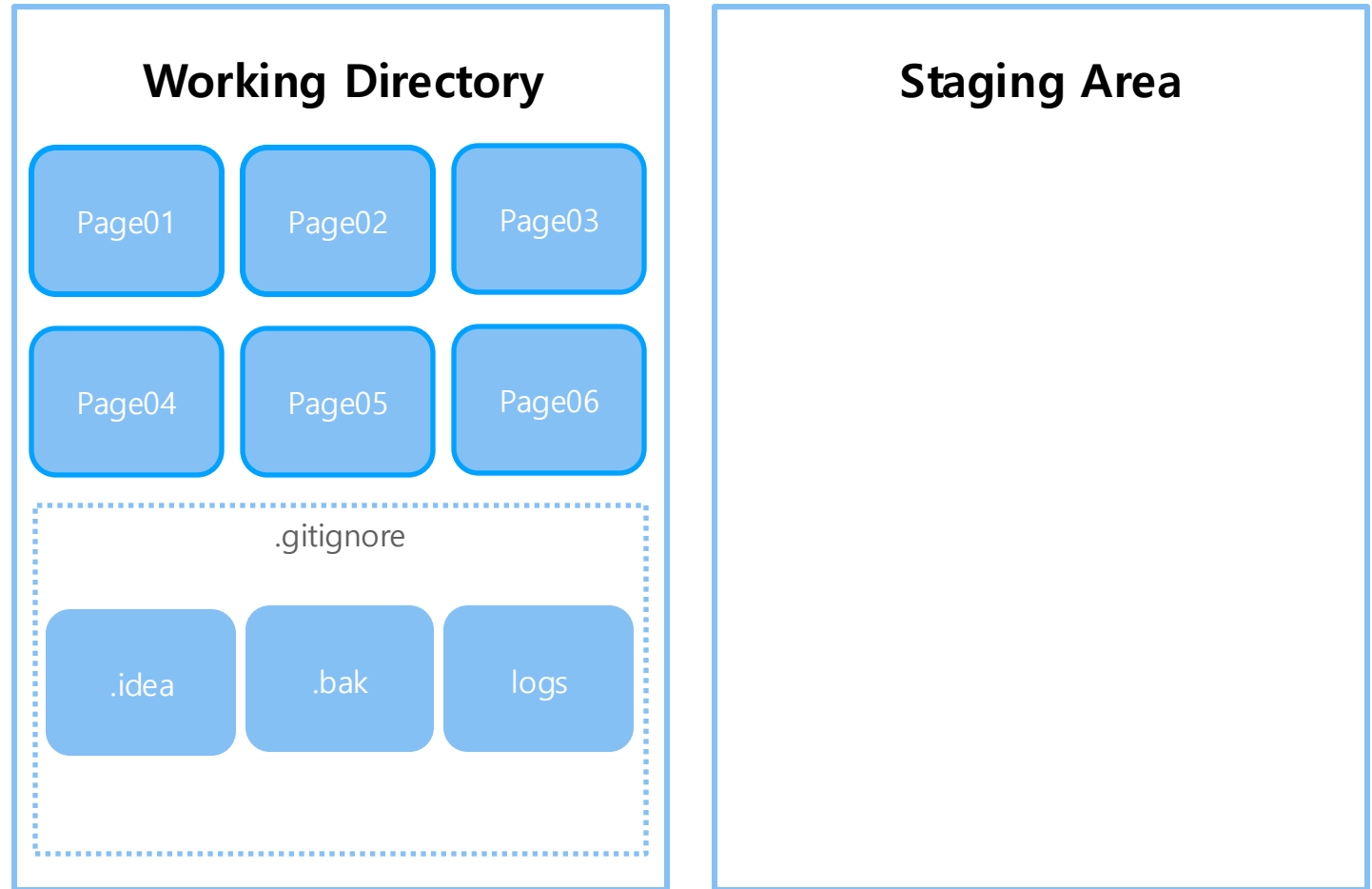
## Staging Area

## Repository



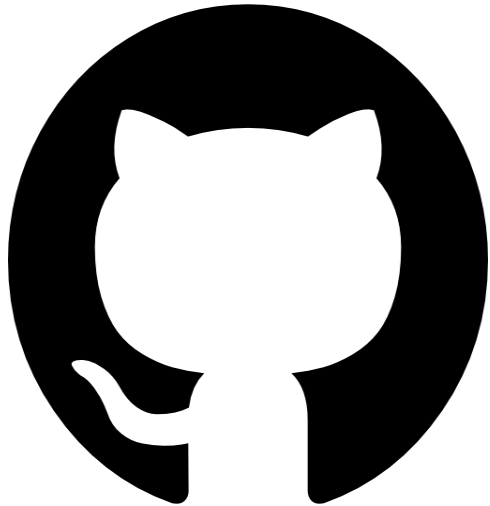
## 추적 제외 .gitignore

- ✓ Git이 추적하지 않는 파일 목록 설정
- ✓ 백업파일이나 로그파일, 컴파일파일 혹은 중요 API 키값과 같은 설정 값
- ✓ 원격저장소에 공유하지 않기 위한 목적



## 원격저장소 Remote Repository

팀과 공유하며 어디서나 온라인으로 어디서나 접근 가능한 저장소



Github



GitLab



BitBucket

## 원격저장소 Remote Repository

### 원격저장소 연결 (Local Repository가 있을 때)

```
git remote add ${remote repo name : origin} ${remote repository url}
```

### 원격저장소 확인

```
git remote -v
```

### 원격저장소 복제 (Local Repository가 없을 때)

```
git clone ${remote repository url}
```

### 원격 저장소 파일 올리기

```
git push ${remote repo name : origin} ${branch name : main / dev ...}
```

### 원격 저장소 파일 내려받기 git fetch + git merge

```
git pull ${remote repo name : origin} ${branch name : main / dev ...}
```

**A : git remote add origin {repositoryUrl}**

## Local Repository A

first commit

second commit

## Remote Repository

## Local Repository B

**A : git push origin main**

## Local Repository A

first commit

second commit

## Remote Repository

first commit

second commit

## Local Repository B



**B : git clone {repositoryUrl}**

## Local Repository A

first commit

second commit

## Remote Repository

first commit

second commit

## Local Repository B

first commit

second commit

## B: git push origin main

### Local Repository A

first commit

second commit

### Remote Repository

first commit

second commit

new commit

### Local Repository B

first commit

second commit

new commit

**A : git pull origin main**

## Local Repository A

first commit

second commit

new commit

## Remote Repository

first commit

second commit

new commit

## Local Repository B

first commit

second commit

new commit

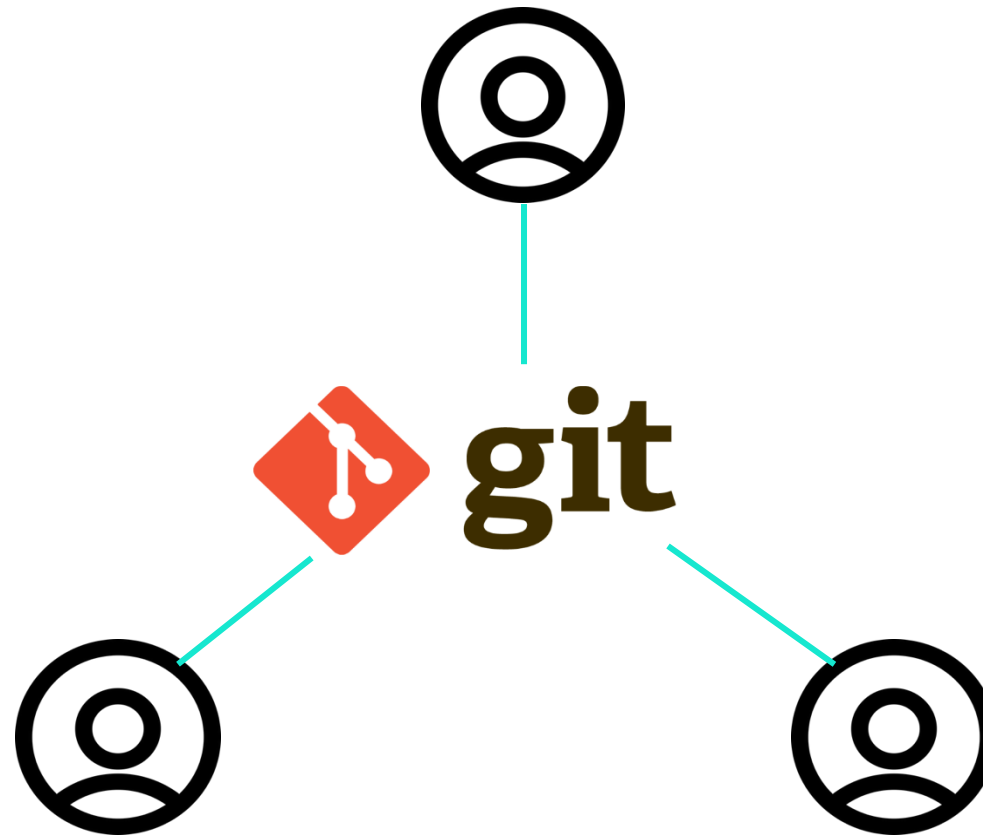
# Git 워크플로우

## Git Workflow

Git을 사용해 코드를 체계적으로 관리하는 규칙

워크플로우를 통해  
작업 분리와 체계적 병합

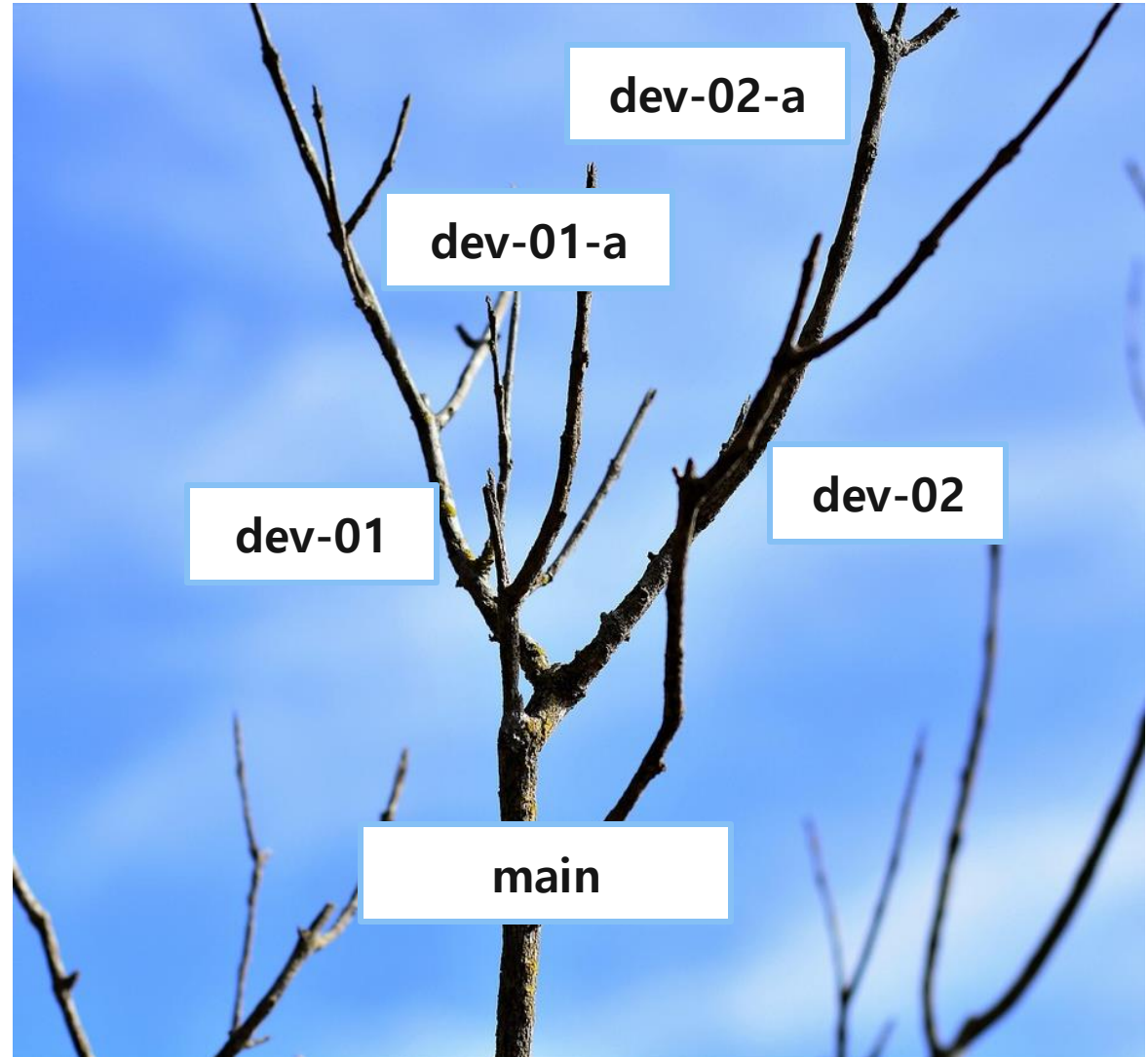
- ✓ 변경 사항 추적
- ✓ 협업 효율성 증대
- ✓ 충돌 최소화



## 브랜치 branch

작업 분리 공간.  
기능별로 코드 분리 가능.

- ✓ 병렬 개발
- ✓ 실험 및 기능 추가



## 브랜치 생성 및 전환

기능(feature/\*), 릴리스(release/\*), 핫픽스(hotfix/\*) 등 목적에 맞는 이름으로 브랜치 생성

### 브랜치 생성

```
git branch ${branch name}
```

```
git branch feature/login-new
```

### 브랜치 확인 (브랜치 목록 및 활성 브랜치 확인)

```
git branch
```

```
git branch
```

### 브랜치 전환

```
git checkout ${branch name}
```

```
git checkout feature/login-new
```

### 브랜치 생성 및 전환

```
git checkout -b ${branch name}
```

```
git checkout -b release/1.0.0
```

## 브랜치 병합

분기됐던 브랜치를 하나의 브랜치로 병합. 충돌 발생 시 수동으로 코드 수정 후 병합 완료

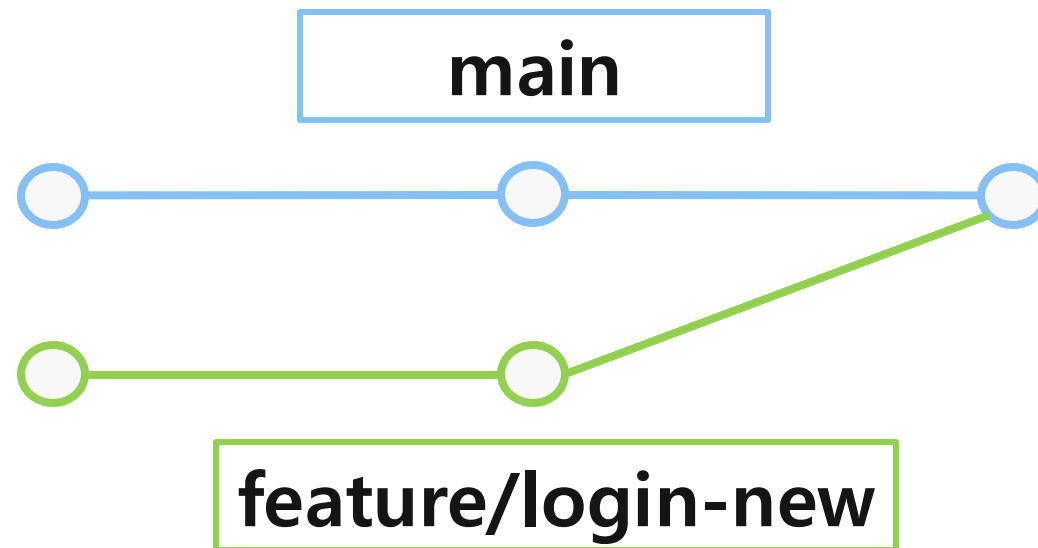
## 브랜치 병합

```
git merge ${branch name}
```

```
// git branch 목록 확인  
git branch
```

```
// main branch로 활성화  
git checkout main
```

```
// feature/login-new 브랜치를 main 브랜치에 병합  
git merge feature/login-new
```





## 이름 규칙

기능이나 목적을 명확히

main(배포), develop(개발 통합),  
feature/gnb(기능), release/v1.0.1(배포 준비), hotfix/bug-135(긴급 수정) 등

## 단기간 유지

장기 브랜치는 병합 복잡도 증가

feature/gnb 브랜치와 feature/login 브랜치가 merge 없이 장기간 나눠져 있을 경우,  
두 브랜치를 병합할 시점에 다양한 지점에서 충돌의 가능성이 높음

## 정기 동기화

메인 브랜치와 자주 병합

가급적 브랜치의 범주를 작게 설정하여 main 브랜치에 병합하는 주기를 짧게 유지

01



## Centralized Workflow

메인 브랜치 중심  
소규모 팀에 적합

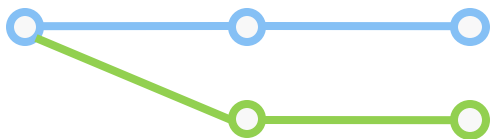
특징: **단일 브랜치**(main / master) 중심

장점: 간단함, 학습 쉬움

단점: 충돌 빈번, 검토 어려움

main(배포)

02



## Feature Branch Workflow

기능별 브랜치 활용  
코드 검토 강화

특징: **기능별 브랜치**(feature/\*) 사용

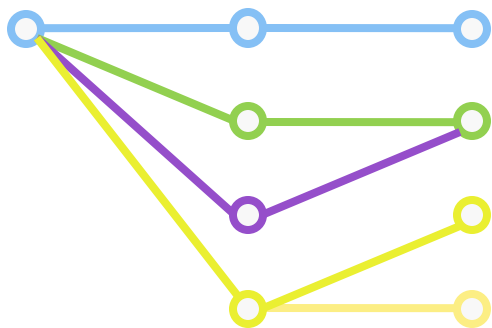
장점: 코드 검토 용이, 협업 강화

단점: 브랜치 관리 필요

main(배포)

└ feature/(기능)

03



## Gitflow Workflow

릴리스/핫픽스 브랜치 포함  
대규모 프로젝트에 유용

특징: 복잡한 프로젝트 관리에 적합.

장점: 작업 분리와 **통합 체계화**.

단점: 초보자에게는 복잡, 팀 규칙 필요.

main(배포)

- └ release/(배포 준비)

- └ hotfix/(긴급 수정)

- └ develop(개발 통합)

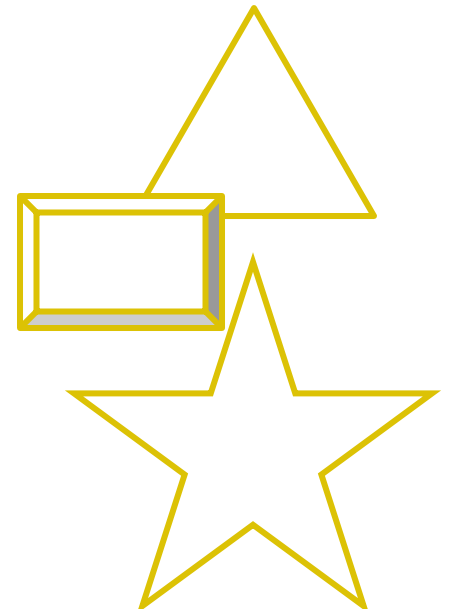
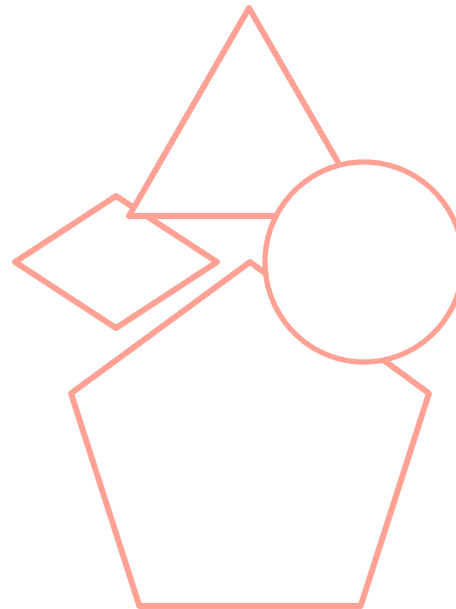
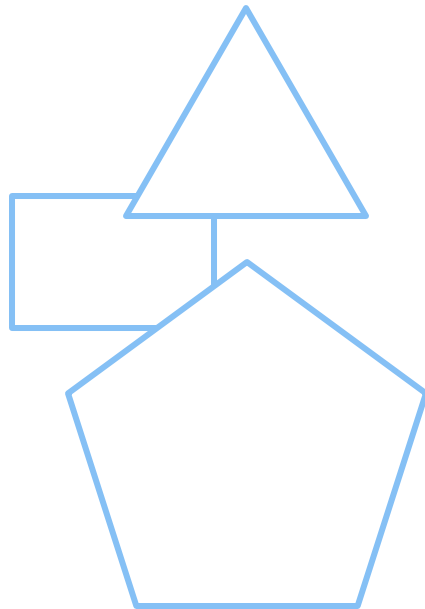
- └ feature/(기능)

브랜치 이름	역할	예시 사용 사례
main	배포용 최종 버전	제품 출시 후 안정 버전 관리
release/*	배포 준비	버전 1.0 배포 전 테스트
hotfix/*	긴급 수정	버그 긴급 패치
develop	개발 중 통합 버전	기능 병합 후 테스트
feature/*	새 기능 개발	로그인 기능 추가

기존 코드

## Conflict 충돌

동일 파일, 동일 라인 수정시  
branch merge 충돌 발생



## 충돌 해결

충돌 마커(Conflict Marker) : Git이 충돌을 표시하는 방식

```
<<<<<< HEAD
현재 브랜치의 코드
=====
병합하려는 브랜치의 코드
>>>>>> feature
```

```
function add(a, b) {
<<<<<< HEAD
    return a + b + 1; // 개발자 A의 수
정 =====
    return a + b - 1; // 개발자 B의 수
정
>>>>>> feature
}
```

### 충돌 해결

1. 충돌 파일 확인 : **git status** 실행
2. 수동 수정 : 코드 편집기로 충돌 파일을 열어 **마커를 제거**하고 원하는 코드로 수정
3. 병합 완료 : `git add ${파일명}` 으로 스테이징 후 **git commit** 으로 커밋



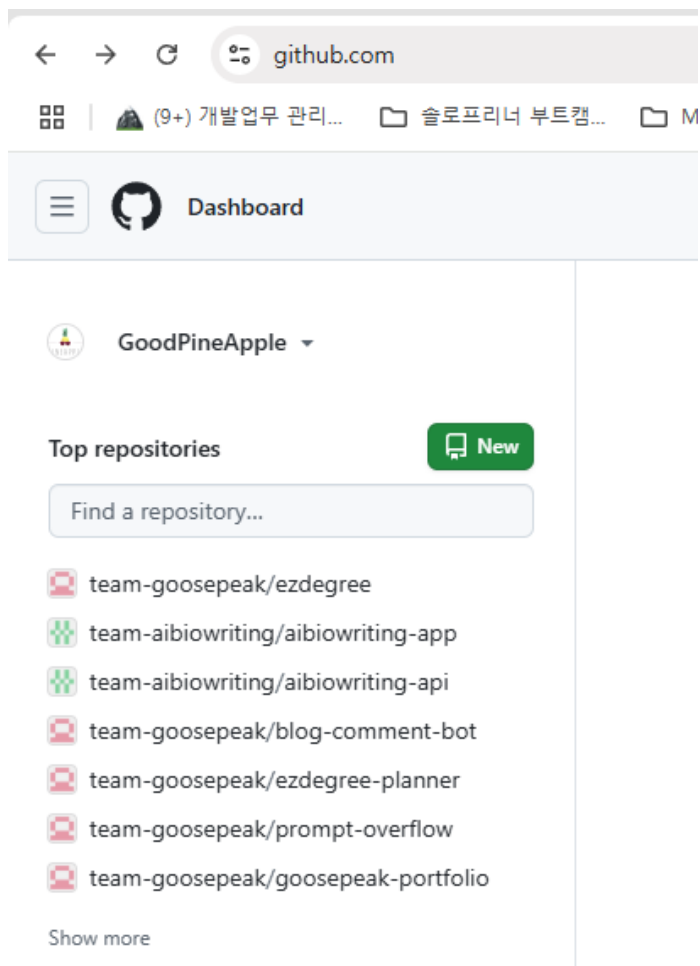
# Oz-lecture

## Github Repositry 생성

좌측메뉴 [New]

Repository Name:  
oz-lecture

공개여부 : Public  
Gitignore: None  
License: None



### 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 \* GoodPineApple / Repository name \* oz-lecture  
✔ oz-lecture is available.

Great repository names are short and memorable. Need inspiration? How about [fuzzy-happiness](#) ?

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.](#)

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

Create repository

## Local Repository 생성

### Vs code 터미널 열기

Window : ctrl + `

Mac : command + J

### Git 명령어

git init

git add .

git commit -m "init project"

```
DEVELOPER-01@DESKTOP-L08QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture
● $ git init
Initialized empty Git repository in E:/98.developer-private/3.lecture/projects/oz-lecture/.git/

DEVELOPER-01@DESKTOP-L08QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture (main)
● $ git add .


DEVELOPER-01@DESKTOP-L08QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture (main)
● $ git commit -m "init project"
[main (root-commit) b35ecd4] init project
14 files changed, 526 insertions(+)
create mode 100644 day01/first.html
create mode 100644 day01/second.html
create mode 100644 day02/form.html
create mode 100644 day02/multimedia.html
create mode 100644 day02/semantic.html
create mode 100644 day03/my-website.html
create mode 100644 day03/styles.css
create mode 100644 day04/my-view.html
create mode 100644 day04/styles.css
create mode 100644 day05/meida/index.html
create mode 100644 day05/meida/styles-desktop-first.css
create mode 100644 day05/meida/styles-mobile-first.css
create mode 100644 day05/meida/styles.css
create mode 100644 day07/bootstrap/index.html
```

## Repository 연결

### Github Repository

Git 주소 복사

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

 Set up in Desktop

or

HTTPS

SSH



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

### Local Repository

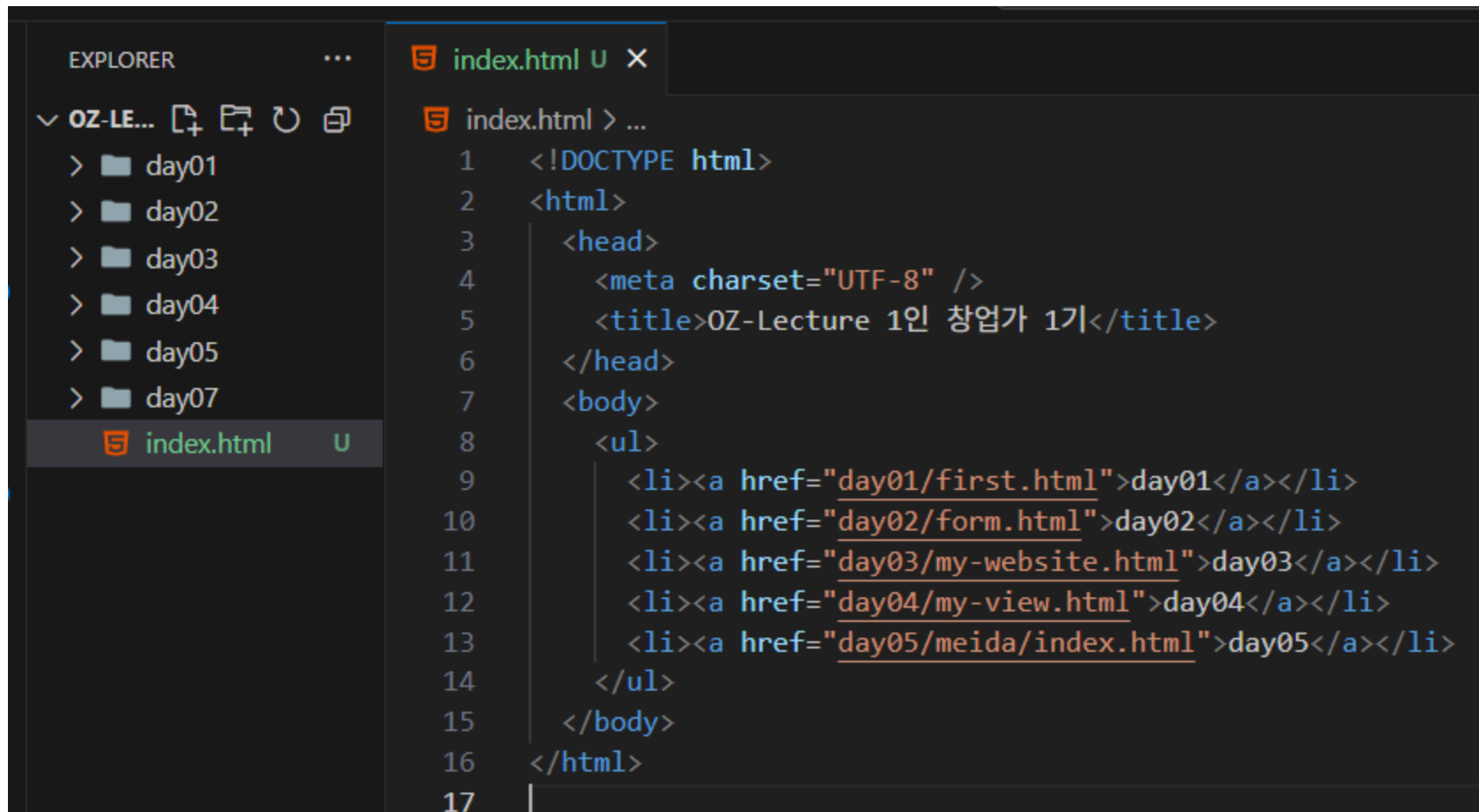
git remote add origin [git hub url]

git remote -v

```
DEVELOPER-01@DESKTOP-L08QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture (main)
● $ git remote add origin https://github.com/GoodPineApple/oz-lecture.git

DEVELOPER-01@DESKTOP-L08QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture (main)
● $ git remote -v
origin  https://github.com/GoodPineApple/oz-lecture.git (fetch)
origin  https://github.com/GoodPineApple/oz-lecture.git (push)
```

## index.html 작성



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8" />
5     <title>OZ-Lecture 1인 창업가 1기</title>
6   </head>
7   <body>
8     <ul>
9       <li><a href="day01/first.html">day01</a></li>
10      <li><a href="day02/form.html">day02</a></li>
11      <li><a href="day03/my-website.html">day03</a></li>
12      <li><a href="day04/my-view.html">day04</a></li>
13      <li><a href="day05/meida/index.html">day05</a></li>
14    </ul>
15  </body>
16 </html>
17
```


## 코드 push

## Local


git push origin main

## GitHub 변경 확인

```
DEVELOPER-01@DESKTOP-LO8QNMQ MINGW64 /e/98.developer-private/3.lecture/projects/oz-lecture (main)
● $ git push origin main
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 16 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (24/24), 6.77 KiB | 2.26 MiB/s, done.
Total 24 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/GoodPineApple/oz-lecture.git
 * [new branch]      main -> main
```

 oz-lecture Public Pin Watch

main 1 Branch 0 Tags  Add file Code

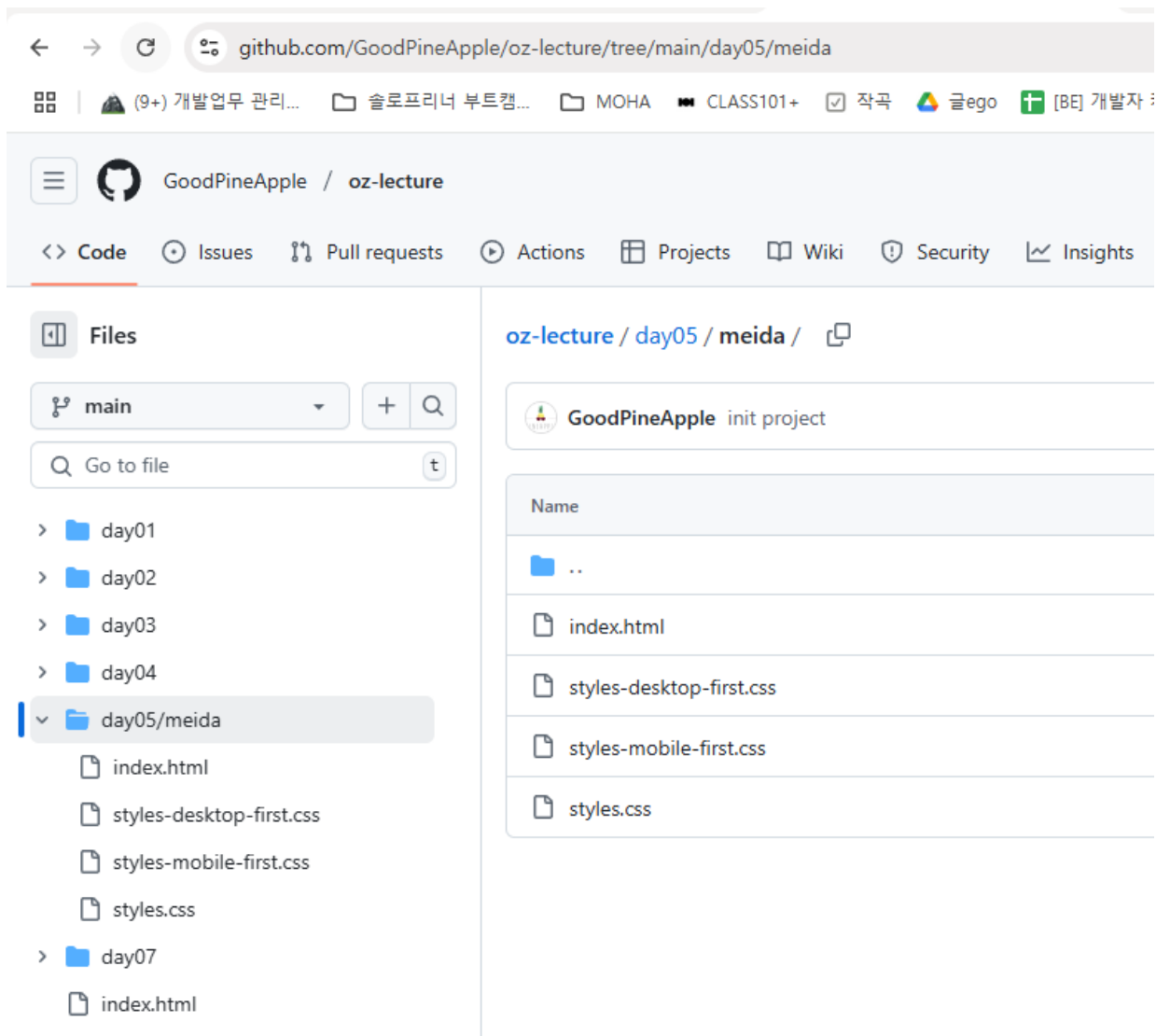
 GoodPineApple	init project	b35ecd4 · 35 minutes ago	1 Commit
day01	init project	35 minutes ago	
day02	init project	35 minutes ago	
day03	init project	35 minutes ago	
day04	init project	35 minutes ago	
day05/meida	init project	35 minutes ago	
day07/bootstrap	init project	35 minutes ago	

## 과제 제출

## Github 링크

+

## 실행 캡처화면



오늘 우리는



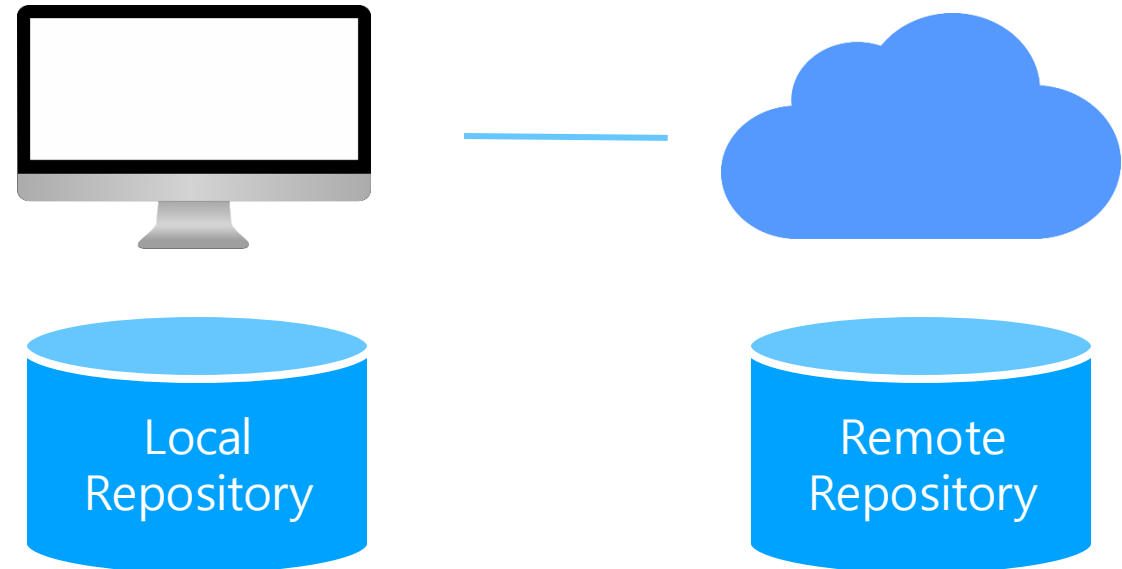
## 버전관리 Version Control System

파일 변경 이력을 기록하고 관리하는 방법

1. 협업 : 여러 사람이 동시에 작업 가능
2. 복구 : 실수해도 이전 버전으로 돌아감
3. 추적 : 누가 언제 무엇을 바꿨는지 확인 가능

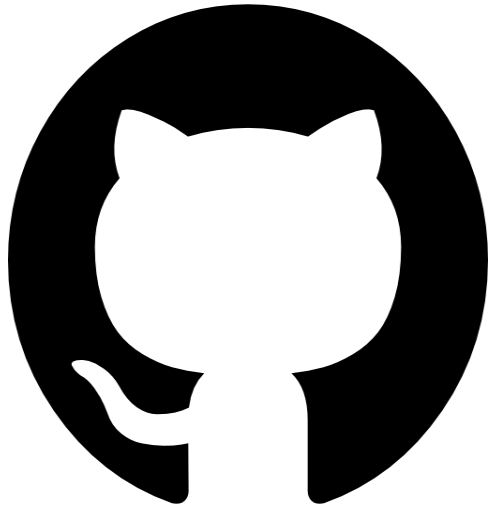
## 로컬저장소와 원격저장소

- ✓ Git의 두 가지 저장소
- ✓ 로컬은 내 컴퓨터에서 관리
- ✓ 원격은 온라인에 공유



## 원격저장소 Remote Repository

팀과 공유하며 어디서나 온라인으로 어디서나 접근 가능한 저장소



Github



GitLab



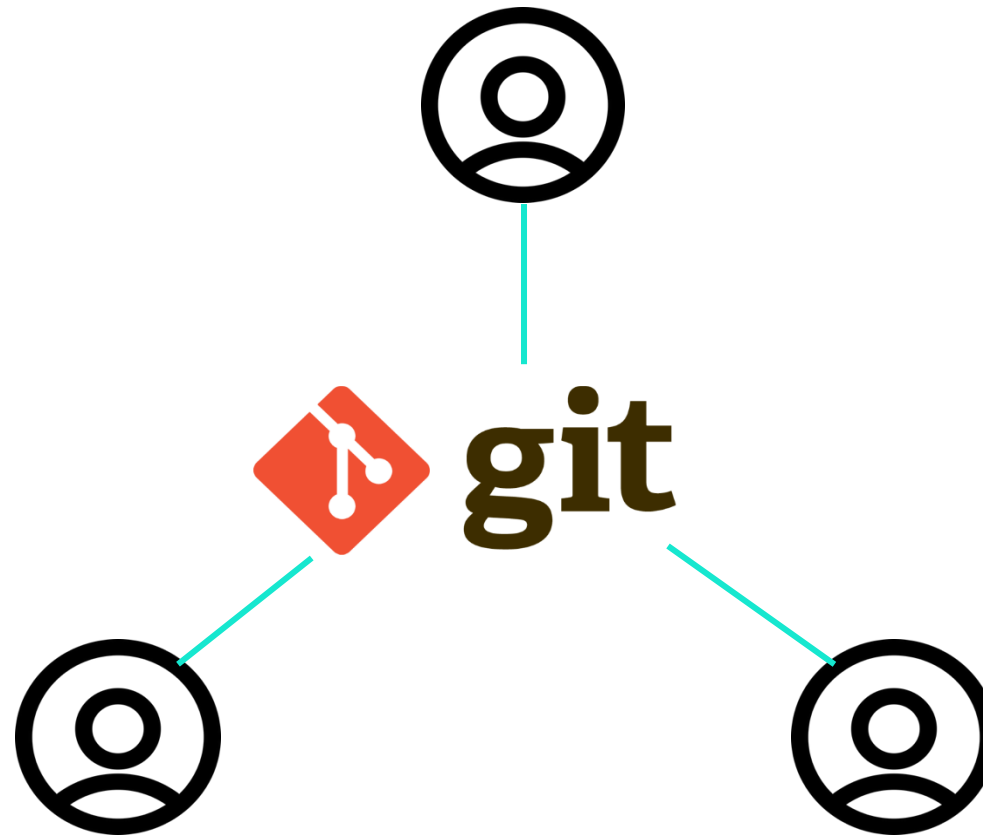
BitBucket

## Git Workflow

Git을 사용해 코드를 체계적으로 관리하는 규칙

워크플로우를 통해  
작업 분리와 체계적 병합

- ✓ 변경 사항 추적
- ✓ 협업 효율성 증대
- ✓ 충돌 최소화



### 충돌 해결

1. 충돌 파일 확인 : **git status** 실행
2. 수동 수정 : 코드 편집기로 충돌 파일을 열어 **마커를 제거**하고 원하는 코드로 수정
3. 병합 완료 : `git add ${파일명}` 으로 스테이징 후 **git commit** 으로 커밋

감사합니다