

# **GitHub 포트폴리오**

4주차. Github 기본 사용법 I



# 지난 주 복습



Git 저장소의 개념	로컬 및 원격 저장소의 개념과 차이점
	로컬 저장소 생성 및 사용 방법
	명령 : git init, git add, git commit
저장소 데이터 동기화	원격 저장소(github) 생성
	로컬 저장소와 원격 저장소 동기화(연결)
	명령 : git remote add, git push, git pull

# 학습개요



## 학습목표

01. Github에서 새로운 리포지토리를 생성하고 관리할 수 있다.
02. Github 리포지토리의 파일 및 폴더 구조를 이해하고 활용할 수 있다.
03. 기본적인 Github 명령어를 사용하여 리포지토리를 조작할 수 있다.

## 학습내용

- Github 리포지토리 생성
- Github 리포지토리의 파일 및 디렉토리 구조 이해
- Github 리포지토리 관리

# 학습에 앞서...



- Github는 소프트웨어 개발 및 프로젝트 관리에 있어 중요한 도구임
- 이를 잘 활용하면 효율적으로 프로젝트를 관리하고 협업할 수 있음
- 리포지토리 생성 및 관리를 통해 프로젝트의 버전 관리를 체계적으로 할 수 있으며, 파일 및 폴더 구조를 이해함으로써 코드와 자료를 효율적으로 관리할 수 있음
- 따라서 이번 주차 학습을 통해 Github 사용에 대한 기초 지식을 확립하고, 이를 바탕으로 더 복잡한 기능을 학습할 준비를 갖추는 것이 중요함

# Github 리포지토리 생성

UNIT

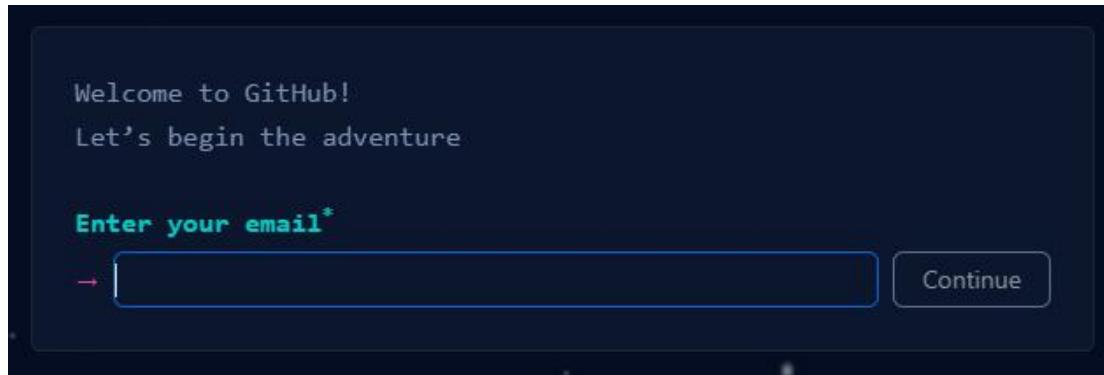


# 리포지토리 생성



## ① 리포지토리 생성 사전 준비

- Github 계정 생성



> 계정 생성 가능한 환경 : PC, 노트북, 모바일 모두 가능

# 리포지토리 생성



## ① 리포지토리 생성 (원격저장소 생성)

- Github 새 저장소 : Github 접속 & 로그인 후 저장소 생성 (Repositories → New)

### 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 \*      Repository name \*



/

◀ 저장소명(필수)

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

Description (optional)

◀ 저장소 설명(옵션)

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 ▶

◀ Git 관리 비대상 파일 목록 파일(옵션, 기본값은 비생성)

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▶

◀ 라이선스 설정 (옵션, 기본값은 라이선스 없음)

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**

## Q Add a README file 옵션지만 생성하는 이유

- 새로 생성하는 저장소에 README 파일을 자동으로 추가

Initialize this repository with:

Add a README file ◀ 저장소 처음 접속 시 설명내용을 담고 있는 파일(옵션)

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

- 장점 : 협업자, 기여자, 새로운 사용자 등 타인의 이해력 향상 도움
- 작성내용
  - 목적, 기능, 사용 방법, 설치방법, 실행방법, 필수 종속성, API참조, 트러블슈팅, FAQ 등
  - 라이선스 유형, 저작권 고지 등
  - 기여방법, 코드 스타일 가이드, 이슈 및 풀 요청 규칙 등
- 작성방법 : 마크다운(Markdown) 형식

# 리포지토리 생성



## ① 리포지토리 생성 이후 나중에 생성하는 방법 #1

- 리포지토리 생성 후 아무 파일도 없는 경우

The screenshot shows a GitHub repository creation interface. At the top left, there's a "Set up GitHub Copilot" section with a "Get started with GitHub Copilot" button. To its right is an "Add collaborators" section with a "Invite collaborators" button. Below these is a "Quick setup" section for experienced users, which includes links for "Set up in Desktop" (with an icon), "HTTPS", "SSH", and a copy link. It also shows the repository URL: [https://github.com/trex99/hello\\_repo2.git](https://github.com/trex99/hello_repo2.git). A note at the bottom of this section encourages adding a [README](#), [LICENSE](#), and [.gitignore](#). A red box highlights the [README](#) link.

# 리포지토리 생성



## ② 리포지토리 생성 이후 나중에 생성하는 방법 #2

- 리포지토리 생성 후 아무 파일을 commit 한적이 있는 경우

The screenshot shows a GitHub repository page for a repository named "hello\_repo". The repository is public. At the top, there are buttons for "Pin" and "Unwatch 1". Below the repository name, it shows "main", "1 Branch", "0 Tags", a search bar "Go to file", and buttons for "Add file" and "Code". A commit by "trex99" titled "Delete README.md" is listed, along with the timestamp "1437513 · 2 minutes ago" and "2 Commits". Below the commit list, there is a section for the "README" file, which is currently empty. A red box highlights the green "Add a README" button at the bottom of this section. The text "Add a README" is centered above the button, with the instruction "Help people interested in this repository understand your project by adding a README." below it.

## ④ 리포지토리 생성 이후 나중에 생성하는 방법 #3

- 리포지토리 생성 후 로컬 저장소와 연동하고 로컬에서 만들기를 시도하는 경우

- 1 로컬 저장소와 관계된 프로젝트 디렉토리로 이동
- 2 “README.md” 파일명으로 파일 만들기
- 3 파일내용은 Markdown 형식으로 작성 후 저장
- 4 git add README.md
- 5 git commit -m “Add README file”
- 6 git push origin main

## Q 기본 관리 항목 : 설명, 언어, 라이선스

- 설명 : 리포지토리 페이지로 이동 → Settings → Repository description 란에 새로운 설명을 입력
- 언어 : 리포지토리 페이지로 이동 → Settings → Repository language → Edit → 언어 선택
- 라이선스 : 리포지토리 페이지로 이동 → Settings → Licenses & info → Edit → 라이선스 선택

## ④ 협력자 초대(Collaborators)

- 1 단계 : 리포지토리 페이지 접속 → Settings → 왼쪽 메뉴 "Collaborators" 선택  
→ "Add people" 버튼 클릭
  - 초대할 사람의 GitHub 유저네임이나 이메일 주소를 입력
  - 해당 사용자에게 부여할 액세스 권한(Write, Admin 등)을 선택
- 2 단계 : "Add collaborator" 클릭 초대 전송
- 3 단계 : 초대받은 사용자는 이메일로 알림을 받고 초대를 수락

## 브랜치 (branch)

- 필요 상황 : 개발과정에 코드를 여러 개로 복사해서 독립적으로 수정하는 경우
- Git 장점 추가 : 새로운 브랜치를 쉽고 빠르게 만들며, 손쉽게 브랜치 이동 가능
- Git 권장 개발 : 여러 개의 브랜치를 만들고 추후 병합(Merge)하는 방법 권장

```
$ git add README test.rb LICENSE  
$ git commit -m 'The initial commit of my project'
```

◀ 기본적으로 master 브랜치에 저장

```
$ git branch testing
```

◀ 신규 testing 브랜치 생성

```
$ git checkout testing
```

◀ 신규 testing 브랜치를 저장 대상으로 설정

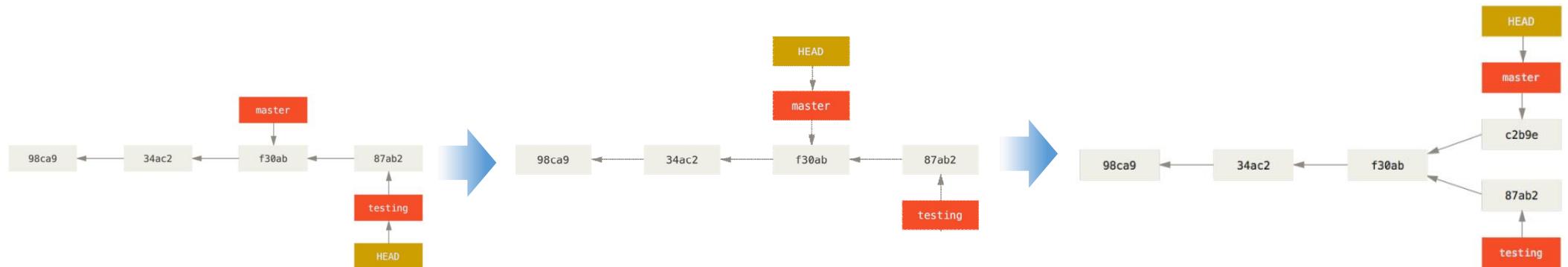
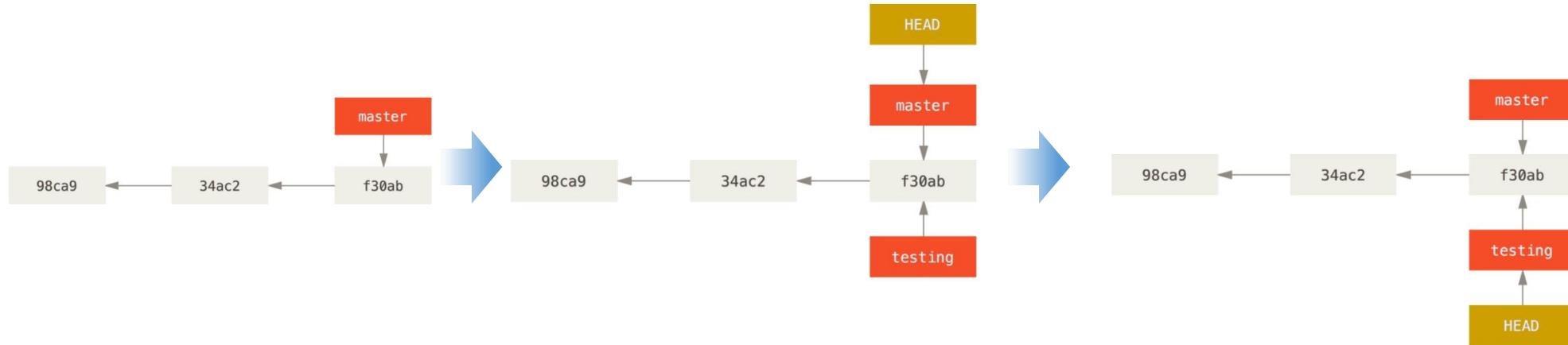
```
$ vim test.rb  
$ git commit -a -m 'made a change'
```

◀ 신규 testing 브랜치에 저장, master 브랜치는 이전 수정내용으로 저장

# 리포지토리 관리



## 브랜치 (branch)



## ① 브랜치 (branch) 응용

1. 운영중인 웹사이트가 있다. 당연 브랜치가 있을 것이다. master라고 하자(master branch).
2. 웹사이트 새로운 이슈 작업이 있다고 가정하자(A 작업발생).
3. 새로운 이슈를 처리할 새 Branch를 하나 생성한다. (A branch made & checkout).
4. 새로 만든 Branch에서 작업을 진행한다(A branch & commit).
5. 이때 중요한 문제가 생겨서 그것을 해결하는 Hotfix를 먼저 만들어야 한다(hotfix 작업발생).

**그러면 아래와 같이 브랜치와 병합을 응용한다.**

1. 새로운 이슈를 처리하기 이전의 운영(Production) 브랜치로 이동한다(master branch checkout).
2. Hotfix 브랜치를 새로 하나 생성한다(B branch made & checkout).
3. 수정한 Hotfix 테스트를 마치고 운영 브랜치로 Merge 한다(B & master branch merged).
4. 다시 작업하던 브랜치로 옮겨가서 하던 일 진행한다(A branch checkout).

# Github 리포지토리 이해

UNIT



## Github 웹 인터페이스 이용

- 로컬 파일 추가 : 리포지토리 이동 → Add files → Upload files → commit
- 신규 파일 생성 : 리포지토리 이동 → Add files → Create new file → commit

## 로컬 파일 Github 추가

- 프로젝트 디렉토리로 이동 → git add . → git commit -m “메시지” → git push
- 대량 파일 업로드 시 권장

# 폴더 이해



## Project 폴더 예시

```
repository/
  └── .github/
      └── workflows/    # GitHub Actions 워크플로우 파일 (CI/CD 등)
  └── .husky/          # Git Hooks 설정 파일 (Husky 사용 시)
  └── docs/            # 프로젝트 문서 파일 (README, 위키 등)
  └── src/             # 프로젝트 소스 코드
      ├── assets/        # 이미지, 폰트 등 정적 자산 파일
      ├── components/   # 재사용 가능한 UI 컴포넌트 파일
      ├── config/         # 설정 파일 (환경 변수 등)
      ├── services/       # 외부 API 통신 관련 파일
      ├── styles/         # CSS, SCSS 등 스타일 파일
      ├── utils/          # 유ти리티 함수 파일
      └── ...              # 기타 프로젝트 소스 코드
  └── tests/           # 단위 테스트, 통합 테스트 파일
      └── unit/
      └── integration/
  └── editorconfig     # 편집기 설정 파일
  └── .gitignore       # Git 무시 파일 목록
  └── LICENSE          # 라이선스 파일
  └── package.json     # NPM 패키지 매니페스트
  └── README.md        # 프로젝트 설명 및 문서
  └── ...              # 기타 구조 파일
```

## README.md

- 역할 : 프로젝트 이해를 위한 파일 (개요, 설치 방법, 사용 방법, 기여 가이드 등 설명)
- 작성방법
  - Markdown 형식으로 작성
  - 프로젝트 제목과 간단한 설명
  - 목차를 만들어 문서 구조화
  - 설치 및 실행방법, 의존성, 예제코드 작성
  - 기여방법, 코드 스타일 가이드, 이슈 및 풀 요청 규칙 작성

# 폴더 이해



## README.md

```
<div align="center">
  
</div>

[![Python](https://img.shields.io/pypi/pyversions/tensorflow.svg)](https://badge.fury.io/py/tensorflow)
[![PyPI](https://badge.fury.io/py/tensorflow.svg)](https://badge.fury.io/py/tensorflow)
[![DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.4724125.svg)](https://doi.org/10.5281/zenodo.4724125)
[![CII Best Practices](https://bestpractices.coreinfrastructure.org/projects/1486/badge)](https://bestpractices.coreinfrastructure.org/projects/1486)
[![OpenSSF Scorecard](https://api.securityscorecards.dev/projects/github.com/tensorflow/tensorflow/badge)](https://api.securityscorecards.dev/projects/github.com/tensorflow/tensorflow/badge)
[![Fuzzing Status](https://oss-fuzz-build-logs.storage.googleapis.com/badges/tensorflow.svg)](https://oss-fuzz-build-logs.storage.googleapis.com/badges/tensorflow.svg)
[![Fuzzing Status](https://oss-fuzz-build-logs.storage.googleapis.com/badges/tensorflow-py.svg)](https://oss-fuzz-build-logs.storage.googleapis.com/badges/tensorflow-py.svg)
[![OSSRank](https://shields.io/endpoint?url=https://ossrank.com/shield/44)](https://ossrank.com/p/44)
[![Contributor Covenant](https://img.shields.io/badge/Contributor%20Covenant-v1.4%20adopted-ff69b4.svg)](https://img.shields.io/badge/Contributor%20Covenant-v1.4%20adopted-ff69b4.svg)
[![TF Official Continuous](https://tensorflow.github.io/build/TF%20Official%20Continuous.svg)](https://tensorflow.github.io/build/TF%20Official%20Continuous.svg)
[![TF Official Nightly](https://tensorflow.github.io/build/TF%20Official%20Nightly.svg)](https://tensorflow.github.io/build/TF%20Official%20Nightly.svg)

**`Documentation`** |
----- |
[![Documentation](https://img.shields.io/badge/api-reference-blue.svg)](https://www.tensorflow.org/api_docs/python/tensorflow)

TensorFlow](https://www.tensorflow.org/) is an end-to-end open source platform
for machine learning. It has a comprehensive, flexible ecosystem of
[tools](https://www.tensorflow.org/resources/tools),
[libraries](https://www.tensorflow.org/resources/libraries-extensions), and
[community](https://www.tensorflow.org/community) resources that lets
researchers push the state-of-the-art in ML and developers easily build and
deploy ML-powered applications.

TensorFlow was originally developed by researchers and engineers working within
```

The screenshot shows the TensorFlow GitHub repository's README page. At the top, there are navigation links for README, Code of conduct, Apache-2.0 license, and Security. Below that is the TensorFlow logo and the word "TensorFlow". Underneath the logo, there are several shields: Python 3.9 | 3.10 | 3.11 | 3.12, pypi package 2.16.1, DOI 10.5281/zenodo.4724125, openssf best practices passing, openssf scorecard 8.2, oss-fuzz build failing, oss-fuzz build failing, OSSRank #9 (Top 1%), Contributor Covenant v1.4 adopted, TF Official Continuous 0 passed, 0 failed, TF Official Nightly 17 passed, 4 failed. There are two tabs: Documentation (which is selected) and api reference. The main content area starts with a paragraph about TensorFlow being an end-to-end open source platform for machine learning. It mentions the comprehensive, flexible ecosystem of tools, libraries, and community resources. It also notes that TensorFlow was originally developed by researchers and engineers working within the Machine Intelligence team at Google Brain. The page also provides information about stable Python and C++ APIs, backward compatible API for other languages, and release announcements and security updates via mailing lists.

## .gitignore

- 역할 : Git에서 추적하지 않아야 할 파일과 디렉토리를 지정
- 작성방법

- 각 항목은 새 줄에 작성
- 디렉토리 경로는 슬래시(/)로 구분
- 패턴 지정 시 와일드카드(\*)를 사용
- 주석은 #으로 시작
- 일반적으로 무시해야 할 파일/디렉토리

\* .DS\_Store(macOS), Thumbs.db(Windows), build, dist, tmp, cache, env, vscode, log

## .gitignore

```
.DS_Store
.ipynb_checkpoints
node_modules
/.bazelrc.user
/.tf_configure.bazelrc
/xla_configure.bazelrc
/bazel/*
/bazel_pip
/tools/python_bin_path.sh
(tensorflow/tools/git/gen
/pip_test
/_python_build
*.pyc
__pycache__
*.swp
.vscode/
cmake_build/
tensorflow/contrib/cmake/_build/
.idea/**
/build/
[Bb]uild/
/build_output/
(tensorflow/core/util/version_info.cc
(tensorflow/python/framework/fast_tensor_util.cpp
/tensorflow/lite/gen/**
```

# Github 리포지토리 관리

UNIT



## git clone : 전체 복제

- 상황과 이유 : 새로운 프로젝트 시작, 기존 프로젝트 기여, 다른 환경에서 작업, 백업
- 복제 방법 #1 (로컬저장소)
  - Github 원격 저장소에서 Repositories 탭으로 이동
  - Code 탭에서 “Code” 클릭 후 리포지토리 URL 복사
  - git clone <복사한 리포지토리 URL>
    - \* `git clone https://github.com/username/repo.git`
- 복제 방법 #2 (백업)
  - Github 원격 저장소에서 Repositories 탭으로 이동
  - Code 탭에서 “Code” 클릭 후 다양한 Download.zip
  - ???

## git pull : 변경 수신

- 상황과 이유 : 다른 사용자의 변경사항을 가져와서 수정 (팀 작업 환경)
- 반영 방법
  - 로컬 리포지토리로 이동 → git remote -v로 확인
  - git pull : 기본 원격 리포지토리 (origin)의 현재 브랜치에서 최신 변경사항 수신
  - git pull <원격이름> <브랜치이름> : 특정한 원격지에서 원하는 브랜치 변경사항 수신

## git push : 변경 저장

- 상황과 이유 : 버그 수정 등 코드 변경 및 새로운 기능 개발, 백업결과 복구(동기화)
- 저장 방법
  - 로컬 리포지토리로 이동
  - git add <파일 또는 디렉토리>
  - git commit -m "커밋 메시지"
  - git remote add origin <원격 저장소 URL> # 최초 push할 경우만
  - git push origin <브랜치 이름>