

Lecture 1.

오리엔테이션 & 오픈소스SW 개요

김영빈 교수

01 강의 소개

강의 목표

1. 오픈소스 SW 이해 및 적용

- 오픈소스 라이선스, 커뮤니티 문화, 협업 규칙을 이해하고 실제 프로젝트에 적용할 수 있다.

2. 협업 툴 활용 능력 배양

- GitHub, Notion, Obsidian, Cursor 등 다양한 협업 및 코드 지원 도구를 활용할 수 있다.

3. AI를 활용한 개발 효율 극대화

- AI 코딩 도구(GitHub Copilot, ChatGPT, Perplexity 등)를 활용하여 코드 자동 완성, 문서화, 코드 리뷰를 최적화하는 방법을 익힌다.

4. 실무 기반 프로젝트 수행 능력 배양

- 실습 및 팀 프로젝트를 통해 오픈소스 프로젝트 기여 및 AI 기반 개발 협업 능력을 갖춘다.



협업 툴 활용



팀 프로젝트



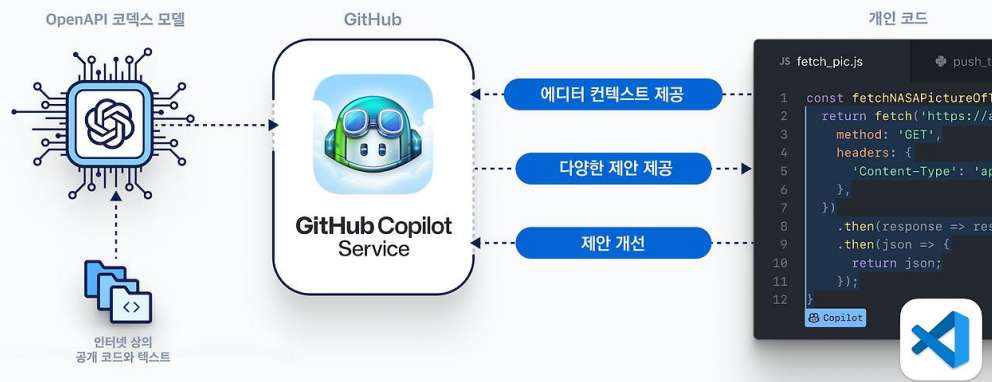
실무 개발 경험

강의 개요 및 학습내용

- 1 AI 코딩 도구 활용 : GitHub Copilot과 ChatGPT를 활용한 코드 자동 완성, AI 기반 코드 추천 활용 방법과 최적화 전략
- 2 VS Code 환경 최적화 : 최신 개발 환경 구축 및 GitHub 연동
- 3 GitHub을 활용한 협업 프로세스 : AI 도구를 활용하여 GitHub Issues, PR, 코드 리뷰 최적화
- 4 팀 프로젝트 기반 AI 협업 : Notion, GitHub를 연계한 협업 프로젝트 진행

GitHub Copilot : AI 기반 코드 자동완성 및 제안 도구

- AI 코딩 도구/VS Code 환경 최적화/팀 프로젝트 기반 AI 협업 지원



◀ GitHub Copilot 흐름도

코드 자동완성 예시 ▶

```
13 export {IAPContextProvider} from './IAPContext';
14 export {JoinContextProvider} from './JoinContext';
15 export {MemoContextProvider} from './MemoContext';
16 export {NetworkContextProvider} from './NetworkContext';
17 export {RegisterInfoContextProvider} from './RegisterInfoContext';
18 export {SearchListContextProvider} from './SearchListContext';
19 export {ServerLogContextProvider} from './ServerLogContext';
20
21 export default ContextProvider;
22
```

출처 : <https://velog.io/@minwoo129/Copilot-3%EA%B0%9C%EC%9B%94-%EC%82%AC%EC%9A%A9%EA%B8%B0>

강의 개요 및 학습내용

강의 진행 방식

20%

이론 (20%)

AI 코딩 도구의 원리 및 오픈소스 활용법 학습

- AI 기반 코드 자동완성 도구의 원리 이해
- GitHub Copilot, ChatGPT 등 AI 도구의 작동 방식 및 활용방법

50%

실습 (50%)

VS Code & Copilot 실습, GitHub 협업 툴 활용

- VS Code에서 GitHub Copilot 및 AI 도구 설정 및 활용
- GitHub Issues와 Pull Request(PR) 작성 및 협업 실습
- AI 도구를 활용한 코드 최적화 및 Refactoring

30%

프로젝트 (30%)

팀 프로젝트에서 AI 기반 코딩 및 협업 진행

- 실습을 바탕으로 프로젝트 진행
- GitHub 및 Notion을 활용한 협업 진행

실습 및 프로젝트 진행 방식

실습

GitHub Copilot 실습

- AI 기반 코드 자동완성 기능 활용
- 주석을 통한 코드 생성 지시 방법 학습
- 함수, 클래스 등 반복 패턴 코드 작성 자동화

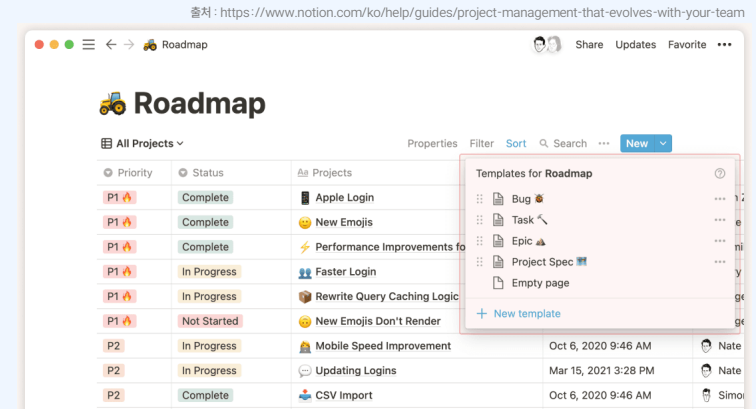


◀ Git Flow Example

VS Code + GitHub 연동 실습

- GitHub Issues 및 PR을 활용한 프로젝트 협업
- Copilot 및 GitHub Actions를 활용한 자동화 실습
- 코드 리뷰 자동화 및 품질 관리
- Branch 관리 및 협업 Workflow 체험

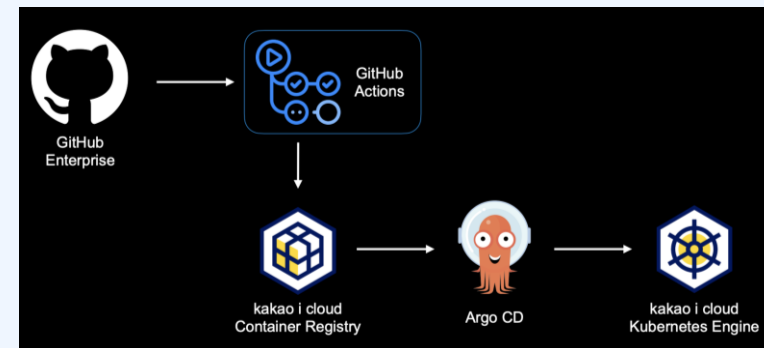
협업



▲ 노션을 활용한 프로젝트 관리 예시

+

출처 : <https://tech.kakao.com/posts/516>



▲ 카카오엔터프라이즈 DevOps 표준안



- Issues
- Actions
- PR

→ AI 도구를 활용한 오픈소스 프로젝트 개발 및 협업 경험을 통한 실무 역량 강화

강의 일정 안내

1 ~ 4주차 기초 환경 설정 및 협업

- AI 도구 개념 이해
- GitHub 기초 및 연동
- VS Code, Copilot 환경 설정

5 ~ 10주차 AI 코딩 실습 및 팀 프로젝트

- AI 코딩 실습 심화
- 팀 프로젝트 기획
- MVP 구현 및 배포

11 ~ 16주차 프로젝트 고도화 및 발표

- 기능 추가 및 코드 리뷰
- CI/CD 자동화 배포
- 최종 발표 및 평가

강의 일정 안내

각 주차별 학습 목표

1 ~ 4주차 기초 환경 설정 및 협업

1주차

오리엔테이션 및 오픈소스 개요

- 강의 목표 및 진행 방식 소개
- VS Code & GitHub Copilot 개념 학습
- AI 도구를 활용한 현대적 개발 방식 개념 정리

2주차

GitHub 설정 및 AI 도구 연동 실습

- GitHub 개인 계정 설정 및 Git 기초 명령어 학습
- VS Code에 GitHub 연동 및 Copilot 활성화
- Copilot을 활용한 첫 코드 자동완성 실습

3~4주차

프로젝트 관리 도구 및 AI 기반 협업

- GitHub Issues, PR, 코드 리뷰 방식 실습
- Notion을 활용한 팀 프로젝트 일정 관리
- AI 도구를 활용한 코드 리뷰 자동화 실습

강의 일정 안내

각 주차별 학습 목표

5 ~ 10주차 AI 코딩 실습 및 팀 프로젝트

5~6주차

팀 프로젝트 기획 및 AI 코딩 적용 실습

- 팀 프로젝트 주제 선정 및 Notion을 활용한 프로젝트 계획 수립
- Copilot을 활용한 초기 코드 작성 및 구조 설계
- AI 도구를 활용하여 코드 최적화 및 리팩토링 수행

7~10주차

오픈소스 프로젝트 기여 및 AI 코딩 활용

- 오픈소스 프로젝트 탐색 및 기여 가능성 분석
- GitHub Pull Request(PR) 제출 실습
- AI 도구를 활용하여 오픈소스 프로젝트 코드 분석 및 개선

강의 일정 안내

각 주차별 학습 목표

11 ~ 16주차

프로젝트 고도화 및 발표

11~14주차

팀 프로젝트 고도화 및 유지보수 실습

- AI 기반 코드 최적화 및 프로젝트 성능 개선
- 자동화된 CI/CD 구축 및 테스트 적용
- 코드 리뷰 및 피드백 반영하여 프로젝트 완성도 향상

15~16주차

최종 발표 및 평가

- 최종 프로젝트 발표 및 시연
 - AI 도구 활용 사례 및 프로젝트 개발 과정 공유
 - 팀별 프로젝트 평가 및 피드백 제공
-

평가항목

평가항목 및 비율

평가 항목	비율
팀 평가	70% (결과물 40%, 발표 10%, 문서화 20%)
개별 평가	30% (GitHub 기여도 15%, 동료 평가 10%, 코드 리뷰 5%)

평가기준

■ 팀 평가기준 (70%)

평가 요소	비율	설명
프로젝트 결과물	40%	기능 완성도, 코드 품질, 목표 충족 여부
발표	10%	팀별 최종 발표 및 데모 시연
문서화	20%	README.md, 프로젝트 문서화 수준

프로젝트 결과물 평가 기준

- 요구사항을 충족했는가?
- AI 도구(Copilot 등)를 효과적으로 활용했는가?
- 팀원 간 역할 분배가 적절했는가?

발표 평가 기준

- 프로젝트 구조와 개발 과정이 명확히 설명되었는가?
- 팀원들이 역할을 균형 있게 수행했는가?

문서화 평가 기준

- README.md가 프로젝트 개요, 실행 방법, 주요 기능을 포함하는가?
- 코드에 주석 및 가이드 문서가 포함되었는가?

평가기준

■ 개별 평가기준 (30%)

평가 요소	비율	설명
GitHub 기여도	15%	코드 기여(PR), Issue 작성, 프로젝트 기여 내역
동료 평가	10%	팀원 간 협업 기여도 평가 (Notion 활용)
코드 리뷰 참여도	5%	GitHub에서 코드 리뷰 작성 및 피드백 제공

GitHub 기여도 평가 기준

- PR 제출 횟수 및 코드 변경량
- Issue 생성 및 해결 참여

동료 평가 기준

- 팀 협업 기여도(적극적인 참여 vs 소극적인 참여)
- 역할 수행의 균형

코드 리뷰 평가 기준

- 팀원들의 PR에 대한 적극적인 피드백 여부
- 코드 리뷰 시 명확한 개선 사항을 제안했는가?

평가기준

■ GitHub 기여도 평가방식

평가 요소	설명
코드 기여(PR 제출)	PR 개수, 코드 변경량, Merge 여부
이슈 기여	Issue 생성 및 해결 내역
코드 리뷰	팀원들의 PR에 대한 피드백 제공 내역

GitHub 기여도 분석

- GitHub Insights → Contributors 탭 활용
- PR 및 Issue 로그를 통해 팀원별 활동 분석
- GitHub Actions를 활용하여 기여 활동 자동 추적 가능

수강생 준비사항

개발 환경 준비

필수 준비 항목

1. [GitHub 계정](#) 생성
2. [VS Code 설치](#)
3. [Notion 계정](#) 생성
4. Git 설치

- ✓ 3주차부터 본격적인 실습 진행 예정
→ 실습 전까지 GitHub 계정 생성 및 기본 설정 요망
- ✓ 노트북/태블릿 지참 필수

수강생 준비사항

개발 환경 준비

항목	참고	
1. GitHub 계정 생성	Account Creation	https://docs.github.com/ko/get-started/start-your-journey/creating-an-account-on-github
2. VS Code 설치	Download	https://code.visualstudio.com/download
3. Notion 계정 생성	Account Creation	https://www.notion.so/signup?from=marketing&pathname=%2Fko%2Fdesktop
	Download	https://www.notion.com/ko/desktop
4. Git 설치	Download	https://git-scm.com/downloads

02 오픈소스 소프트웨어의 개요

OSS 정의 및 개념

오픈소스 소프트웨어(OSS)

오픈소스 소프트웨어(Open Source Software, OSS)란,
소스 코드가 공개되어 누구나 자유롭게 사용, 수정, 배포할 수 있는 소프트웨어이다.

비교항목	일반 소프트웨어	오픈소스 소프트웨어
소스코드	비공개	공개
사용권한	유료	무료
수정 가능 여부	제조사만 수정 가능	누구나 수정 가능
배포 가능 여부	제한적(라이선스 필요)	자유롭게 배포 가능
개발 방식	폐쇄적, 내부 개발팀	개방적, 커뮤니티 협업
예시	<ul style="list-style-type: none">Windows, PhotoshopGPT-4 (OpenAI)	<ul style="list-style-type: none">Linux, VS Code, GitLlama (Meta)

OSS 정의 및 개념

오픈소스의 중요성

전 세계 개발자들이 협업하여 지속적으로 발전

- Linux Kernel: 1,500개 이상의 기업과 15,000명 이상의 개발자가 참여하여 유지보수

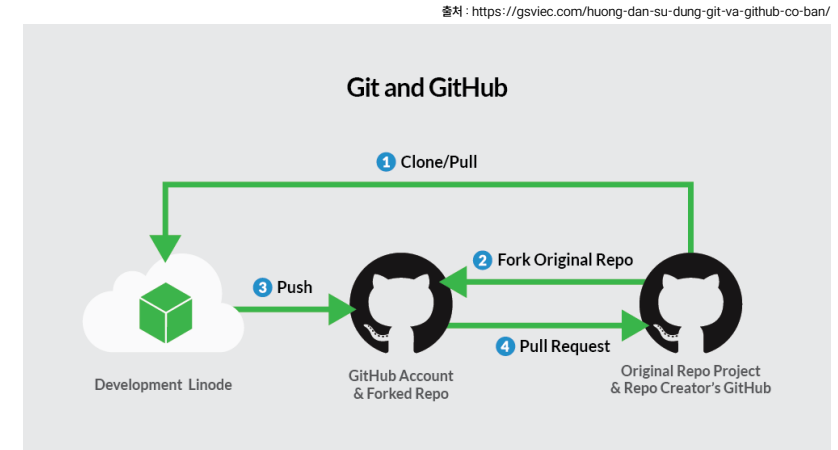
빠른 버그 수정과 보안 패치

- 기업이 단독으로 개발하는 소프트웨어보다 수정 속도가 빠름
- 오픈소스 웹 브라우저 Chromium(Chrome 기반)은 보안 취약점 발견 시 즉시 패치 가능

스타트업, 대기업, 연구기관에서 적극 활용

- 스타트업: 초기 비용 절감 (MySQL, PostgreSQL)
- 대기업: 자체 오픈소스 프로젝트 운영 (Facebook → React, Google → TensorFlow)
- 연구기관: AI, 머신러닝, 클라우드 연구 진행

GitHub 협업 구조



▲ Git과 GitHub를 활용한 협업 프로세스

주요 기업 오픈소스 프로젝트

Google

TensorFlow, Android, Chromium

Facebook

React, PyTorch, Jest

Microsoft

VSCode, TypeScript, .NET Core

Netflix

Chaos Monkey, Zuul, Eureka

오픈소스와 자유 소프트웨어의 차이

자유 소프트웨어(Free Software)

: "소프트웨어의 자유"를 가장 중요한 가치로 삼는 소프트웨어

자유 소프트웨어가 보장해야 하는 4가지 자유

1. 소프트웨어를 실행할 자유
2. 소프트웨어를 연구하고 수정할 자유
3. 복사하여 배포할 자유
4. 개선된 버전을 배포할 자유

오픈소스 소프트웨어(OSS)

: 누구나 자유롭게 사용, 수정, 배포할 수 있는 소프트웨어

- 오픈소스 소프트웨어는 소프트웨어를 개방하여 협업을 극대화하는 방식을 지향

구분	자유 소프트웨어	오픈소스 소프트웨어
중점 가치	자유(Freedom)	실용성(Practicality)
철학	소프트웨어는 반드시 자유로워야 함	소프트웨어는 개방적이며 실용적이어야 함
License	주로 GPL* (강제적 공유)	MIT, Apache, BSD 등

* GPL (GNU General Public License)

- 자유 소프트웨어 라이선스로, 소프트웨어의 실행, 연구, 공유, 수정의 자유를 최종 사용자에게 보장
- 대표적으로 리눅스 커널이 이용하는 사용 허가

Stallman과 Torvalds의 철학

Richard Stallman (자유 소프트웨어 진영)

Stallman의 철학:

"소프트웨어는 반드시 공개되어야 하며, 사용자에게 100% 자유가 보장되어야 한다!"

1983

- GNU 프로젝트 창설
- 자유 소프트웨어 운동의 시작

대표 프로젝트
GNU Project

1985

- 자유 소프트웨어 재단(FSF) 설립
- 자유 소프트웨어 이념을 전파하기 위한 단체

자유 운영체제를 위한 프로젝트

Linus Torvalds (오픈소스 진영)

Torvalds의 철학:

"소프트웨어는 자유로운 것도 좋지만, 더 중요한 건 사람들이 쉽게 사용하고 협업할 수 있도록 하는 것!"

1991

- Linux 커널 개발
- 자신이 개발한 운영체제 커널을 공개

대표 프로젝트

Linux Kernel

1996

- Linux 펭귄 마스코트 도입
- Linux의 상징이 된 Tux 펭귄 채택

전 세계에서 가장 많이 사용되는
오픈소스 커널

Stallman과 Torvalds의 철학

Stallman의 주장

"소프트웨어는 자유로워야 한다"

- ✓ 기업이 소프트웨어를 독점해서는 안 됨
- ✓ GPL 라이선스 사용으로 자유 소프트웨어 철학 유지
- ✓ GPL은 기업에 부담이 됨 (소스코드를 반드시 공개해야 함)

Torvalds의 주장

"실용성이 중요하다"

- ✓ 기업도 오픈소스를 활용할 수 있어야 함
- ✓ Apache, BSD 라이선스 등 다양한 라이선스 허용
- ✓ 기업은 자신들의 기술을 보호하면서 협업할 수 있음

1980년대

- GNU 프로젝트 시작
- Stallman의 자유 소프트웨어 운동

1991년

- Linux 커널 발표
- Torvalds가 학생 시절 개발

1998년

- '오픈소스' 용어 등장
- Netscape 브라우저 소스 공개

2000년대 이후

- 기업의 오픈소스 채택
- Google, Microsoft 등 참여

결과적으로, 기업과 개발자들은 Torvalds가 제시한 **오픈소스 개발 방식을 보다 선호**

오픈소스의 핵심 철학 (4가지 자유)

1. 실행의 자유

오픈소스 소프트웨어는 누구나, 어떤 목적이든 자유롭게 실행할 수 있어야 한다.

주요 내용

- 특정 사용자나 사용 목적에 대한 제한 없음
- 기업, 개인, 정부 기관 모두 동일한 사용 권한 보유
- 사용 목적에 관계없이 자유롭게 소프트웨어 활용 가능

중요성

- 특정 운영체제나 기기에 종속되지 않음
- 개발자가 실행 환경을 제한하지 않음
- 사용자가 원하는 방식으로 소프트웨어 실행 가능

예시) 오픈소스 예시



Linux

누구나 다운로드하고 실행 가능
데스크톱, 서버, 임베디드 장치 등



BLENDER

Blender

개인, 기업, 교육 기관 모두 사용 가능
상업적 영화 제작부터 학생 프로젝트까지

오픈소스의 핵심 철학 (4가지 자유)

2. 코드 수정 및 개선의 자유

소프트웨어 사용자는 **소스 코드에 접근할 수 있어야 하며**,
필요하면 직접 수정하여 자신에게 맞게 개선할 수 있어야 한다.

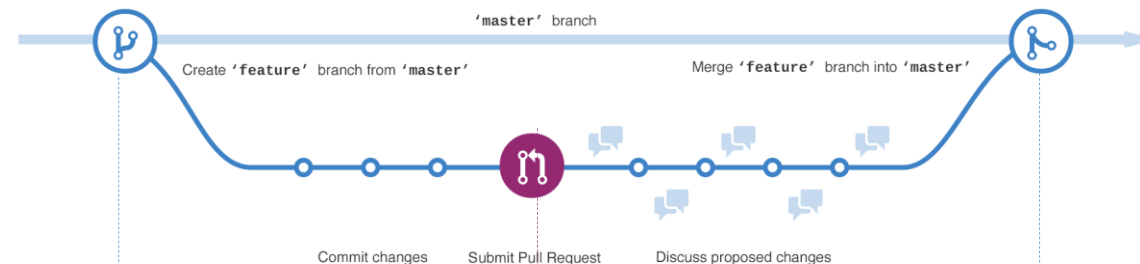
- 폐쇄형 소프트웨어(Windows, Adobe Photoshop)
→ 소스 코드 접근 불가
- 오픈소스 소프트웨어(Linux, VS Code, Firefox)
→ 누구나 코드 변경 가능

중요성

- 원하는 기능 추가 및 불필요한 기능 제거 가능
- 특정 조직이나 환경에 맞춰 소프트웨어 최적화 가능
- 버그 수정 및 보안 취약점 패치를 신속하게 수행 가능

예시) GitHub에서 Pull Request 과정

출처: <https://devwriter.tistory.com/42>



1. Fork 저장소

원본 저장소를 자신의 계정으로 복제



2. 코드 수정

소스 코드를 변경하여 기능 추가 또는 버그 수정



3. Pull Request 제출

변경사항을 원본 저장소에 반영 요청



4. 코드 리뷰 후 병합(Merge)

변경사항이 검토되고 원본 코드에 통합

오픈소스의 핵심 철학 (4가지 자유)

3. 복사 및 배포의 자유


사용자 누구나 **소프트웨어를 복사하여 다른 사람들과 자유롭게 공유할 수 있어야 한다**. 이 과정에서 별도의 허가는 필요하지 않다. (자유로운 소프트웨어 배포)

중요성

- 개발자가 아닌 일반 사용자도 쉽게 소프트웨어를 공유할 수 있음
- 소프트웨어의 접근성을 높이고, 새로운 사용자를 유입 가능
- 교육, 비영리 단체, 개발자 커뮤니티에서 무료로 활용 가능

예시) 오픈소스 프로젝트의 Fork 예시

GitHub Fork

 Fork 39.4k

오픈소스 프로젝트의 **소스 코드를 복사**하여 독립적으로 수정·개발하는 과정

Linux

Linux의 다양한 배포판

🔗 Debian → Ubuntu → Linux Mint

🔗 Red Hat → Fedora, CentOS

Android

Android의 커스텀 버전

🔗 삼성 One UI

🔗 구글 Pixel UI

오픈소스의 핵심 철학 (4가지 자유)

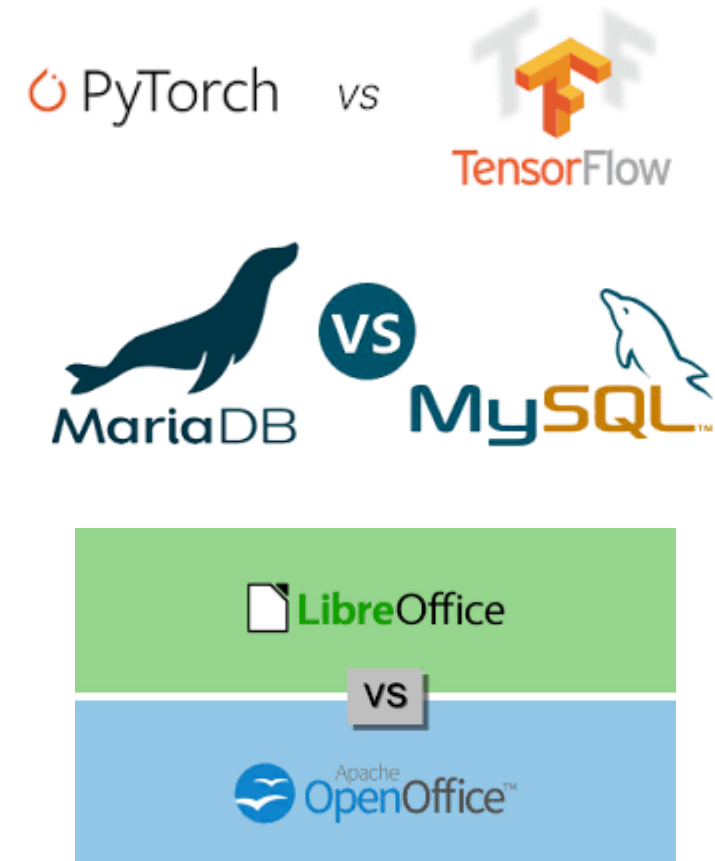
4. 개선된 버전 공유의 자유

사용자가 소프트웨어를 수정한 후, 새로운 버전으로 다시 배포할 자유를 가질 수 있다.

중요성

- 개발자가 기존 소프트웨어의 단점을 보완하여 더 나은 버전을 만들 수 있음
- 다양한 개발자가 기여하면서 소프트웨어의 품질이 향상됨
- 혁신적인 파생 프로젝트가 등장할 수 있는 환경 조성

예시) 성공적인 오픈소스 프로젝트의 발전 사례



오픈소스의 역사와 발전 과정

1. 오픈소스의 기원 : Unix 시대

Unix의 탄생

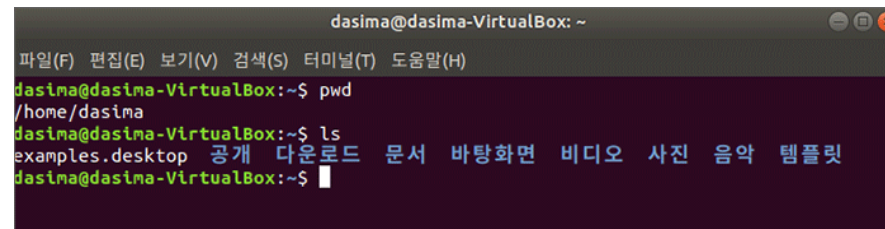
- 1969년, 미국 AT&T 벨 연구소에서 탄생
- Ken Thompson과 Dennis Ritchie가 개발
- 초창기에는 연구 목적으로 개발됨

초기 개발 특징

- 개방적인 협업 방식으로 개발
- 연구기관과 대학에 소스 코드 공유
- 다양한 하드웨어에서 실행 가능하도록 개량

Unix 개발 타임라인

- 1969 : Unix 개발
- 1970s : 대학 및 연구기관에 배포
- 1970s 후반 : AT&T의 상업화
- 1980s : 자유 소프트웨어 운동 촉발



```
dasima@dasima-VirtualBox: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
dasima@dasima-VirtualBox:~$ pwd  
/home/dasima  
dasima@dasima-VirtualBox:~$ ls  
examples.desktop  공개  다운로드  문서  바탕화면  비디오  사진  음악  템플릿  
dasima@dasima-VirtualBox:~$
```

▲ Unix 계열 터미널에서 기본 명령어 실행



Unix에서 오픈소스로의 흐름

1970년대 후반, AT&T가 Unix의 상업적 사용을 제한하면서 자유로운 공유가 어려워졌고,
자유 소프트웨어 운동이 발생하게 되는 계기가 됨

오픈소스의 역사와 발전 과정

2. GNU 프로젝트와 자유 소프트웨어 운동



GNU Project (1983)

- Richard Stallman이 **Unix와 호환되는 자유 소프트웨어 운영체제를 만들고자 시작된 공개 소프트웨어 프로젝트**
- GNU = “GNU's Not Unix”
- 누구나 자유롭게 실행, 복사, 수정, 배포할 수 있고, 누구도 그런 권리를 제한하면 안 된다는 License로 소프트웨어 배포 (완전히 자유로운 운영체제 개발)



한계점 : 커널의 부재

- GNU Hurd라는 커널을 개발했지만 완성되지 못함
- 운영체제의 가장 핵심인 커널(Kernel)이 없어서 실제로 사용 가능한 완전한 시스템을 만들지 못함

◆ 주요 업적

1) FSF(자유 소프트웨어 재단) 설립 (1985)

- GNU 프로젝트를 지원하고 자유 소프트웨어 운동을 확대하기 위해 설립
- 활동 : 소프트웨어 개발, 법적 지원, 라이선스 관리

2) GPL 라이선스(GNU General Public License) (1989)

- GNU 프로젝트에서 만든 소프트웨어를 보호하고, 모든 사용자가 자유롭게 사용, 수정, 배포할 수 있도록 하기 위해 만들어진 라이선스
- GPL 적용 소프트웨어는 변경 후에도 반드시 소스 코드 공개 필수

◆ GNU 프로젝트의 대표 소프트웨어

Bash (\$, Bourne Again Shell)

- Unix/Linux 시스템에서 기본적으로 사용되는 셸(Shell)
- GNU 프로젝트에서 개발한 Bourne Shell(sh)의 향상된 버전
- 스크립트 작성, 명령어 실행, 자동화 작업 등에 필수적인 도구

GCC (GNU Compiler Collection)

- C, C++, Fortran, Ada 등 다양한 언어를 지원하는 컴파일러 모음집
- 오픈소스 컴파일러의 표준으로 자리 잡았으며, 많은 시스템에서 사용됨

▼ Git Bash에서 Linux 기본 명령어를 실행하는 모습

```
( 사용자명 ) MINGW64 ~/Documents/dev/cli-practice
$ ls
README.txt  bin/
( 사용자명 ) MINGW64 ~/Documents/dev/cli-practice
$ mv README.txt bin/
( 사용자명 ) MINGW64 ~/Documents/dev/cli-practice
$ cd bin/
( 사용자명 ) MINGW64 ~/Documents/dev/cli-practice/bin
$ ls
README.txt  introduce.md
```

오픈소스의 역사와 발전 과정

3. Linux와 오픈소스 운동의 시작



Linux의 탄생

- 1991년, 핀란드 헬싱키 대학의 학생 Linus Torvalds가 개발
- MINIX(교육용 Unix 클론)에 영감을 받아 시작
- 완전히 오픈소스로 배포되어 누구나 수정 가능

GNU/Linux의 탄생

- GNU 프로젝트에 부족했던 커널을 Linux가 제공
- GNU 도구 + Linux 커널 = 완전한 자유 운영체제
- GPL 라이선스 채택으로 소스 코드의 자유로운 공유 보장



오픈소스 소프트웨어의 확산

GitHub과 협업 방식의 변화 (2008~)



2008년, GitHub 등장으로 오픈소스 개발 협업 방식 변화
Pull Request(PR), 코드 리뷰, 이슈 트래킹 등 협업 도구 제공



Apache (1995)
오픈소스 웹 서버



오픈소스 웹 브라우저
오픈소스 웹 브라우저



OpenOffice (2000)
무료 오피스 도구



Ubuntu (2004)
친화적 Linux

오픈소스의 역사와 발전 과정

4. 오늘날의 오픈소스

2010년대 이후, 많은 대기업들이 오픈소스 프로젝트를 적극적으로 지원

◆ 오픈소스의 종류

Google

- TensorFlow (2015) - 머신러닝 프레임워크
- Kubernetes (2014) - 컨테이너 오케스트레이션

Microsoft

- VS Code (2015) - 코드 에디터
- TypeScript (2012) - 자바스크립트 확장 언어

Facebook

- React (2013) - UI 라이브러리
- PyTorch - 머신러닝 프레임워크

◆ 오픈소스의 미래

클라우드 및 AI 분야에서의 오픈소스

- 클라우드 기술에서 Kubernetes, Docker와 같은 오픈소스 프로젝트 활용 증가
- AI 분야에서도 PyTorch, Hugging Face 등 오픈소스 도구가 표준으로 자리 잡음

오픈소스의 미래 전망

- 더 많은 기업들이 오픈소스 프로젝트를 공개하고 협업 확장
- 오픈소스가 소프트웨어 개발의 기본적인 방식으로 자리 잡음
- AI 및 클라우드 분야에서 오픈소스의 중요성 계속 증가

주요 오픈소스 프로젝트 소개



: 2005년 Linus Torvalds가 Linux 커널 개발을 위해 제작한
분산 버전 관리 시스템으로, 소스 코드 변경 이력을 추적하고, 협업을 위한 필수 도구로 활용

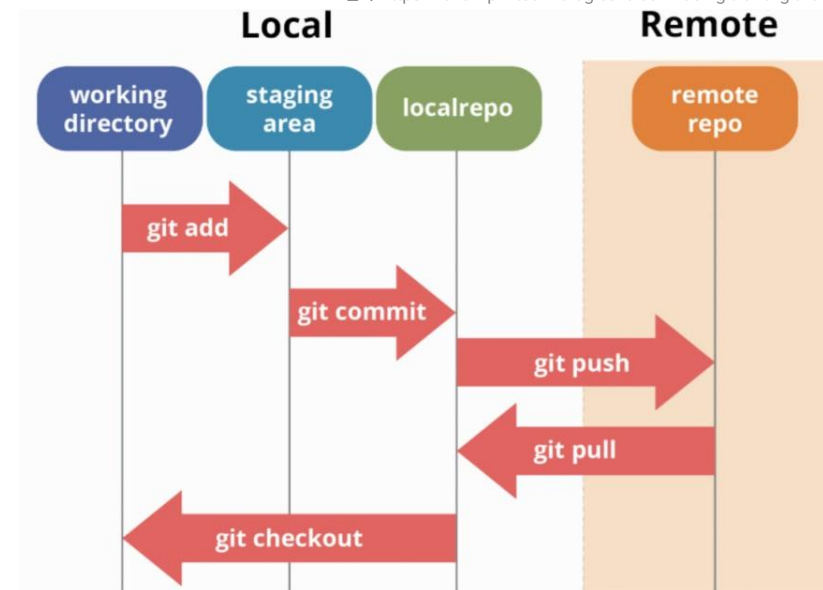
Git의 주요 기능

- 버전 관리: 소스 코드 변경 이력 추적
- 브랜치(Branch): 독립적인 작업 공간
- 병합(Merge): 코드 변경 사항 통합
- 분산 저장소: 로컬에서 코드 관리

Git이 중요한 이유

- 협업 필수 도구 → 대규모 프로젝트 및 팀 개발에 최적화
- 오픈소스 프로젝트의 핵심 기술
- 변경 이력 추적 → 누가, 언제, 무엇을 수정했는지 기록 및 복구 가능

출처: https://champlintechologiesllc.com/05_git-and-gitflow/



▲ Git-Flow

```
$ git init
$ git add .
$ git commit -m "Initial commit"
$ git push origin main
```

▲ Git Command 예시

주요 오픈소스 프로젝트 소개



: 1991년 Linux Torvalds가 개발한 **오픈소스 운영체제 커널**로, GNU 프로젝트와 결합해 GNU/Linux 시스템으로 발전

Linux의 주요 특징

- 완전한 오픈소스: 누구나 코드 확인 및 수정 가능
- 다양한 배포판 제공: Ubuntu, Debian, CentOS 등
- 경량성과 확장성: 다양한 기기 및 환경에서 동작 가능
- 안정성과 보안성: 서버 및 네트워크 환경에서 높은 신뢰성

Linux의 주요 사용 사례

- 서버 운영체제: 전 세계 웹 서버의 약 70%가 Linux 기반
- 모바일 OS: Android는 Linux 커널을 기반으로 개발
- 임베디드 시스템: IoT 기기, 자동차 등에 적용

주요 Linux 배포판



주요 오픈소스 프로젝트 소개



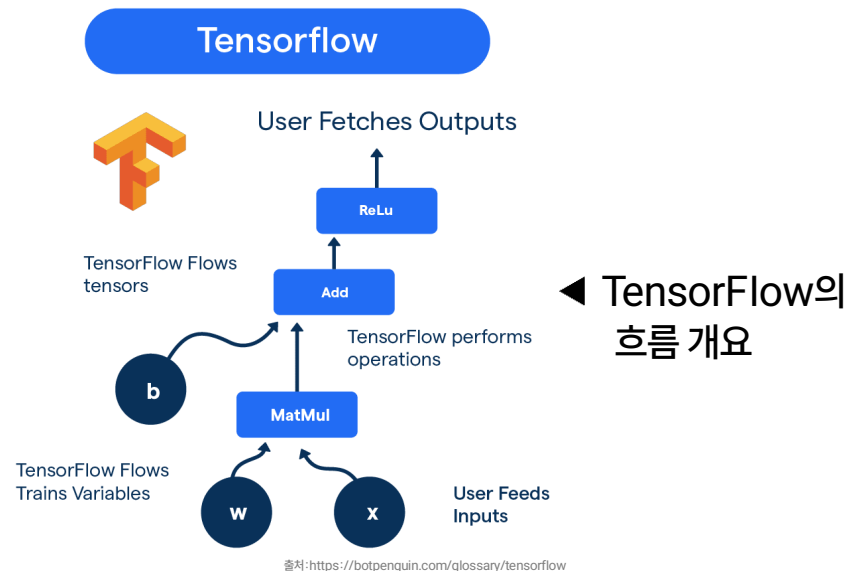
: Google이 2015년에 공개한 오픈소스 **머신러닝 프레임워크**로, 딥러닝 및 AI 연구, 상용 시스템에서 널리 사용

TensorFlow의 주요 기능

- 다양한 플랫폼 지원 : 데스크톱, 모바일, 클라우드 환경
- GPU/TPU 가속 : 고속 연산을 위한 하드웨어 지원
- 딥러닝 모델 학습 : 신경망 모델 구축 및 훈련
- TensorFlow Lite : 모바일 및 임베디드 환경을 위한 경량 ML 지원

TensorFlow 활용 사례

- 이미지 인식 → 자율주행, 얼굴 인식, 의료 영상 분석
- 자연어 처리 → 챗봇, 기계 번역, 감성 분석
- 추천 시스템 → 개인화된 콘텐츠 및 상품 추천



```
import tensorflow as tf
from tensorflow import keras
# 모델 생성
model = keras.Sequential([
    keras.layers.Dense(128,
        activation='relu'),
    keras.layers.Dense(10,
        activation='softmax')
])
```

▲ TensorFlow 코드 예제

주요 오픈소스 프로젝트 소개



Visual Studio Code

: 2015년, Microsoft가 개발한 **오픈소스 코드 편집기**로, Windows, macOS, Linux에서 실행 가능한 크로스 플랫폼 환경 제공

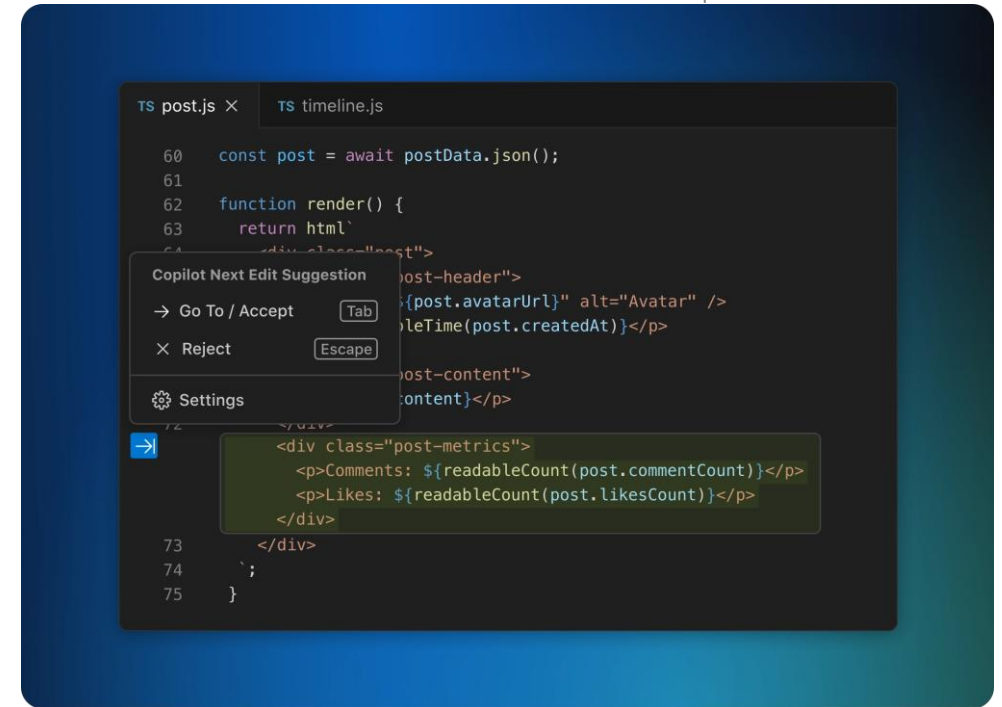
VS Code의 주요 기능

- 다양한 프로그래밍 언어 지원 : Python, JavaScript, C++ 등
- 확장 프로그램 지원 : GitHub Copilot, 테마 변경 등 플러그인 활용
- Git 연동 기능 : 내장 Git 지원으로 버전 관리 용이
- 디버깅 및 터미널 통합 : 코드 실행 및 디버깅 기능 내장

VS Code 특징

- 완전한 오픈소스 : GitHub에서 누구나 기여 가능
- 확장 생태계 : 다양한 플러그인을 통한 기능 확장
- 플랫폼 독립적 : 모든 OS에서 동일한 환경 제공

출처: <https://code.visualstudio.com/>



▲ VS Code의 GitHub Copilot을 활용한 Code suggestions 예시

오픈소스의 장점

1. 비용 절감 효과

오픈소스는 **라이선스 비용 없이** 누구나 고품질 소프트웨어를 활용 가능하게 함

- 무료 소프트웨어 - 구매 비용 없이 사용 가능
- 라이선스 비용 제거 - 상용 소프트웨어와 달리 라이선스 비용 없음
- 스타트업 부담 감소 - 초기 비용 부담 없이 기술 도입 가능

2. 기술 공유 및 발전 촉진

오픈소스는 **코드 공유와 협업**을 통해 기술 발전을 가속화하고 혁신을 촉진

- 학습 도구 - 누구나 소스 코드를 확인하고 배울 수 있음
- 혁신 가속화 - 기업 간 기술 공유로 기술 발전 속도가 가속화
- 코드 품질 향상 - 많은 개발자들이 검토하고 기여하면서 품질 향상

※ 유료 소프트웨어와 오픈소스의 비교

카테고리	상용 소프트웨어	오픈소스 대안	비용 절감
운영체제	Windows (\$200+)	Linux 무료	100%
이미지 편집	Photoshop (\$20+/월)	GIMP 무료	100%
개발 도구	Visual Studio (\$1,199+)	VS Code 무료	100%

오픈소스의 장점

3. 글로벌 협업 기능

오픈소스는 국경과 기업을 초월한 협업을 가능하게 함

- 다양한 시각 - 다양한 국가와 문화권의 개발자 참여
- 24시간 개발 - 시간대가 다른 개발자들이 연속적으로 작업
- 넓은 사용자 기반 - 다양한 환경에서 테스트 가능

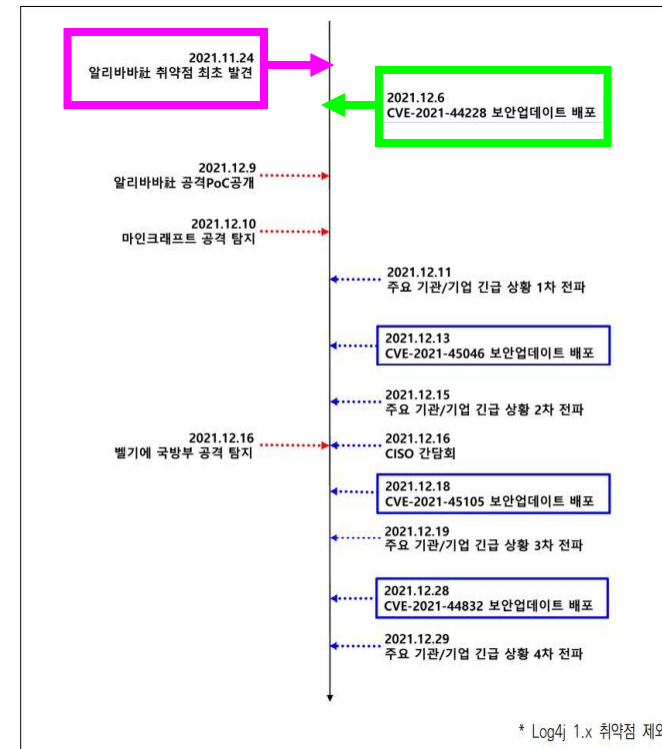
4. 빠른 문제 해결 및 보안 패치

오픈소스는 커뮤니티를 통해 문제를 더 빠르게 발견하고 해결

- 다양한 관점 - 여러 개발자가 문제를 다양한 각도에서 검토
- 신속한 대응 - 보안 취약점 발견 시 빠른 해결 가능
- 지속적 개선 - 실시간으로 이슈를 확인하고 개선

Apache Log4j 취약점(CVE-2021-44228) 해결 사례

: 심각한 보안 취약점 발견 후 72시간 내에 커뮤니티가 협력하여 대응



◀ 좌측 그림 설명 :

취약점 공개

→ 오픈소스 커뮤니티의 빠른 취약점 공유

→ 취약점 공개 후 전 세계 보안 연구원과

개발자들이 즉시 분석 및 대응

→ 긴급 보안 패치 개발 및 배포 → ...

→ 최종적으로 2.17.0 버전을 배포하며 완
전한 패치 완료

출처: <https://www.kisa.or.kr/20205/form?postSeq=1017&page=1>

오픈소스의 단점

1. 커뮤니티 의존성

오픈소스는 개발자 커뮤니티에 의존하기 때문에 프로젝트 안정성이 불확실

* Apache Log4j 보안 취약점이 여전히 존재

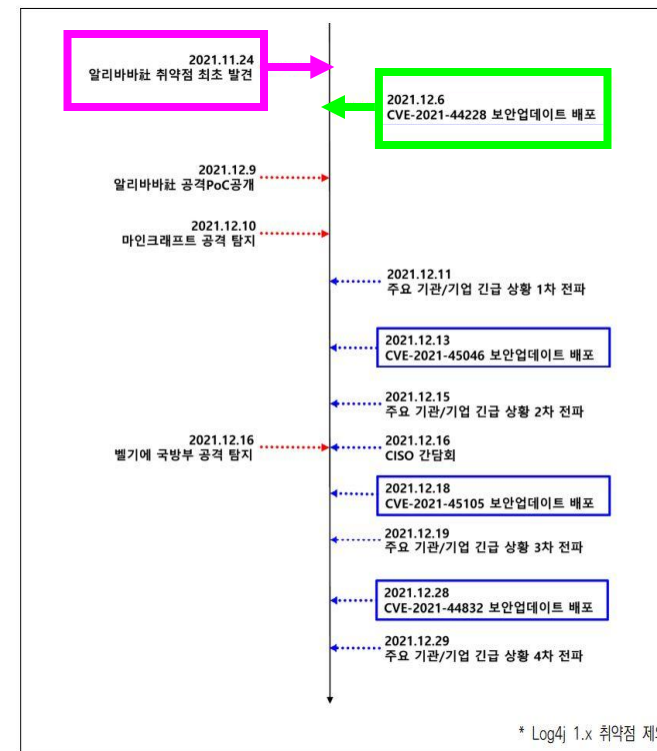
- 오픈소스 커뮤니티의 신속한 대응이 있었지만, **지속적인 보안 유지에는 한계가 있음**
- 수백만 개의 시스템이 사용하는 핵심 인프라임에도 불구하고, **관리 인력과 재정 지원이 부족**
- 즉각적인 패치가 배포되었지만, 많은 시스템이 업데이트되지 않아 여전히 위협이 존재
- 궁극적으로 오픈소스 소프트웨어의 "**의존성 문제**"가 보안 유지에 있어서 근본적인 한계를 가질 수밖에 없음

2. 지속 가능성 문제

오픈소스 프로젝트는 장기적인 지속 가능성이 불확실하여, 기업과 조직이 의존성을 결정할 때 위험 요소가 될 수도 있음

Apache Log4j 취약점(CVE-2021-44228) 해결 사례

: 심각한 보안 취약점 발견 후 72시간 내에 커뮤니티가 협력하여 대응



◀ 좌측 그림 설명 :

취약점 공개

→ 오픈소스 커뮤니티의 빠른 취약점 공유

→ 취약점 공개 후 전 세계 보안 연구원과

개발자들이 즉시 분석 및 대응

→ **긴급 보안 패치 개발 및 배포** → ...

→ 최종적으로 2.17.0 버전을 배포하며 완전한 패치 완료

출처: <https://www.kisa.or.kr/20205/form?postSeq=1017&page=1>

오픈소스의 장단점 정리

구분	장점	단점
비용	무료 사용 가능 라이선스 비용 없이 소프트웨어 사용 가능	유지보수 비용 발생 내부 개발자가 코드를 이해하고 관리하는 비용
	초기 비용 부담 감소 스타트업 및 중소기업에 유리한 비용 구조	교육 및 적응 비용 교육 및 시스템 적응에 시간/비용 필요
협업	글로벌 개발자들이 협력 국경과 조직을 넘어선 다양한 시각과 경험	기여자 부족 위험 기여자가 줄어들면 유지보수 어려움
	투명한 개발 과정 GitHub 등을 통한 공개적인 코드 리뷰 및 개발	의사결정 지연 다수의 의견 조율로 인한 의사결정 지연 가능성
보안	빠른 취약점 패치 전 세계 개발자들이 함께 보안 문제 해결	코드 품질 관리 어려움 다양한 기여자로 인한 일관된 품질 관리 문제
	투명한 코드 검토 다수의 눈이 코드를 검토하여 문제 발견 가능성 높음	취약점 공개 코드가 공개되어 공격자도 취약점 분석 가능
지속 가능성	커뮤니티 주도 발전 사용자와 개발자 커뮤니티에 의한 지속적 발전	개발자 이탈 위험 핵심 개발자가 떠나면 프로젝트가 중단될 위험
	기업 지원 가능성 인기 있는 프로젝트는 기업의 지원을 받기도 함	불확실한 로드맵 명확한 장기 계획이 없는 경우가 많음

따라서, 조직의 상황과 필요에 따라 적절한 오픈소스 소프트웨어를 선택하고, 위험 요소를 관리하는 전략이 필요함

오픈소스 선택 시 고려사항

- 프로젝트의 활성도와 커뮤니티 규모 확인
- 기업 또는 재단의 후원 여부 확인
- 기술 스택과의 호환성 고려
- 문서화 품질과 지원 리소스 평가
- 조직 내 유지보수 능력 확보

오픈소스 라이선스 개요

오픈소스 라이선스 : 오픈소스 소프트웨어는 누구나 자유롭게 사용할 수 있지만, 특정 규칙을 따라야 한다.

라이선스(License)


- 소프트웨어 사용 및 배포 조건을 정하는 문서
- 개발자가 직접 라이선스를 선택 가능
- **저작권법 기반의 법적 계약 문서**

GitHub에서의 License

octo-repo / in main

Edit

Preview

 Choose a license template

1 Enter file contents here

출처: <https://docs.github.com/ko/communities/setting-up-your-project-for-healthy-contributions/adding-a-license-to-a-repository>

▲ GitHub License 선택 화면

MIT License

짧고 간단한 라이선스. 실질적으로 모든 것이 허용됨

Apache License 2.0

특허권 부여가 포함된 허용적 라이선스

GNU GPL v3.0

변경사항은 반드시 같은 라이선스로 공개해야 함

◆ 오픈소스 라이선스의 중요성

- 법적 보호 기능 제공 (저작권 보호)
- 개발자와 기업이 준수해야 할 규칙 명확화
- 상업적 활용 시 라이선스 검토 필요

⚠ 위반 시 발생할 수 있는 법적 문제

- 저작권 침해 소송
- 배상금 지불
- 소프트웨어 사용 금지

오픈소스 라이선스 개요

주요 오픈소스 라이선스 비교

1) 허용적 라이선스 (Permissive)

: 사용이 자유로우며 제한이 적은 라이선스 유형

- 코드 수정 후 비공개 배포 가능
- 소스 코드 공개 의무 없음
- 상업적 이용 가능
- 주로 원저작자 표시만 요구
- 주요 허용적 라이선스 : MIT, Apache 2.0, BSD License

2) 카피레프트 라이선스 (Copyleft)

: 개작한 소프트웨어도 동일한 라이선스로 공개해야 하는 라이선스 유형

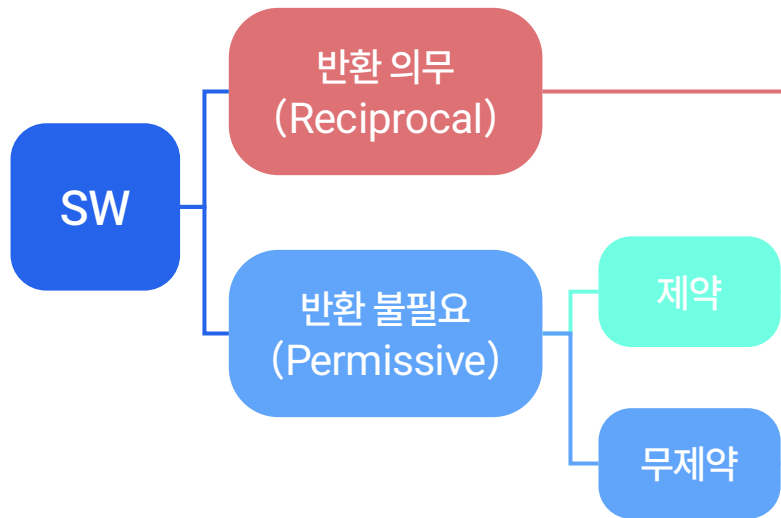
- 코드 수정 후 소스 코드 공개 의무 있음
- 파생 저작물도 같은 라이선스 적용 필수
- 상업적 이용 가능하나 소스 코드 공개 필요
- 자유 소프트웨어 철학 반영
- 주요 카피레프트 라이선스 : GNU GPL/AGPL/LGPL License

▼ 라이선스별 특징 비교

라이선스	상업적 사용	비공개 배포 가능	수정 코드 공개 의무	특허 보호 조항	저작권 표시 필요
MIT	✓	✓	✗	✗	✓
Apache 2.0	✓	✓	✗	✓	✓
BSD	✓	✓	✗	✗	✓
GPL v3	✓	✗	✓	✓	✓
AGPL v3	✓	✗	✓	✓	✓

주요 오픈소스 라이선스

• MIT, GPL, Apache 라이선스 비교



GPL 라이선스 *가장 엄격한 카피레프트 라이선스*

- 소프트웨어를 수정하여 배포할 경우, 변경된 코드도 오픈소스로 공개해야 함
- 상업적 사용 가능하지만, 수정된 코드가 비공개일 수 없음
- GPL 코드를 포함한 소프트웨어는 모두 GPL로 배포해야 함 (전염성)
- 라이선서의 특허권을 자동으로 라이선시에게 부여
- 라이선스 길이 : 약 5,600단어 (매우 길고 복잡함)
- 주요 프로젝트 : Linux 커널(Linus Torvalds), MySQL(Oracle)

MIT 라이선스 *가장 자유로운 허용적 라이선스*

- 누구나 자유롭게 사용, 수정, 배포 가능
- 상업적 사용 가능 (폐쇄 소스 코드로 배포 가능)
- 소프트웨어 사용 시 원래 라이선스 정보 유지 필요
- 원 저작자의 법적 책임 면제 조항 포함
- 라이선스 길이 : 약 171단어 (매우 짧음)
- 주요 프로젝트 : React(Facebook), VSCode(Microsoft)

Apache 2.0 라이선스 *기업 친화적인 허용적 라이선스*

- MIT와 유사하지만 특허 보호 조항이 추가됨
- 소프트웨어를 수정하여 비공개로 배포 가능
- 수정 사항을 명시해야 함
- 원본 라이선스, 저작권, 특허, 상표권 등의 고지사항 유지 필요
- 라이선스 길이 : 약 2,500단어 (중간 길이)
- 주요 프로젝트 : TensorFlow(Google), Kubernetes(Google), Hadoop(Apache)

주요 오픈소스 라이선스

▼ 오픈소스 라이선스 비교

라이선스	사용 제한	변경 후 배포 가능 여부	대표적인 사용 예
MIT License 가장 허용적인 라이선스	거의 없음 • 저작권 고지 유지 • 라이선스 텍스트 포함	✅ 가능 (비공개 배포 가능)	React (Facebook) VSCode (Microsoft) jQuery
GNU GPL v3 강력한 카피레프트 라이선스	수정 후 소스 코드 공개 필수 • 파생물도 GPL로 공개 • 링크된 코드도 영향 받음 • 수정사항 명시	❌ 불가능 (무조건 오픈소스로 배포)	Linux 커널 MySQL WordPress
Apache 2.0 특허 보호가 포함된 허용적 라이선스	특허 보호 조항 포함 • 수정 사항 명시 • 저작권, 특허, 상표 고지 유지 • 라이선스 텍스트 포함	✅ 가능	TensorFlow (Google) Kubernetes (Google) Hadoop
BSD License 허용적 라이선스	최소한의 제한 • 저작권 고지 유지 • 라이선스 조건 포함 • 개발자 이름 사용 제한	✅ 가능	FreeBSD NetBSD PyTorch (Facebook)
GNU LGPL 약한 카피레프트 라이선스	라이브러리 자체 수정 시만 공개 • 라이브러리 자체 수정 시 공개 필요 • 링크하는 애플리케이션은 영향 없음 • 동적 링크 시 비공개 가능	⚠️ 조건부 가능	Qt (GUI 프레임워크) FFmpeg (일부) Mozilla Firefox (일부)

▼ 라이선스 호환성* 개요

라이선스	MIT와 호환	GPL과 호환	Apache와 호환	BSD와 호환
MIT	✓	✓	✓	✓
GPL	✗	✓	✗	✗
Apache 2.0	✓	✗	✓	✓
BSD	✓	✓	✓	✓

* 라이선스 호환성 : 서로 다른 라이선스를 가진 코드를 결합했을 때, 법적으로 충돌 없이 사용할 수 있는지에 대한 여부

기업이 오픈소스 라이선스를 고려하는 방식

기업은 법적 위험 관리와 비즈니스 전략을 위해 오픈소스 라이선스를 신중하게 고려

라이선스 고려 주요 이유

- 법적 위험 관리 - 저작권 침해 등 법적 문제 예방
- 제품 전략과의 일치 - 오픈소스와 자사 제품 통합 시 전략적 고려
- 기업 평판 - 오픈소스 커뮤니티와의 관계 유지
- 지적재산권 보호 - 특허 및 상표권 보호

라이선스 위반 위험 사례

- GPL 코드를 자사 제품에 포함시키고 소스 코드를 공개하지 않는 경우
- 라이선스 의무 사항(저작권 표시 등)을 무시하는 경우
- 라이선스 호환성 문제가 있는 코드를 결합하는 경우
- 실제 사례

Oracle vs Google (Android/Java)

Google이 Android에서 Java API를 사용한 것에 대한 라이선스 및 저작권 분쟁으로 수년간의 법적 다툼 발생

기업이 선호하는 라이선스 유형

기업 유형	선호하는 라이선스	이유
스타트업 & 중소기업	MIT, Apache	•코드를 수정하여 비공개 제품 개발 가능 •소스 코드 공개 의무 없음 •비즈니스 모델 유연성 확보
대기업 (Google, Microsoft)	Apache, BSD	•특허 보호 조항 중요 •오픈소스 생태계 구축 + 기술 주도권 확보 •기업 평판 관리 및 개발자 관계
오픈소스 중심 기업 (Red Hat, Canonical)	GPL, AGPL	•커뮤니티와 협력 중요 •서비스 및 지원 기반 비즈니스 모델 •오픈소스 생태계의 지속 가능성 유지

기업의 오픈소스 정책 사례

Google
Apache 2.0

Microsoft
MIT 라이선스

Facebook
BSD 및 MIT

오픈소스 커뮤니티 문화 및 참여 방법




- 오픈소스 커뮤니티 : 전 세계 개발자들이 협력하여 소프트웨어를 개발하고 개선하는 공간

오픈소스 커뮤니티의 특징

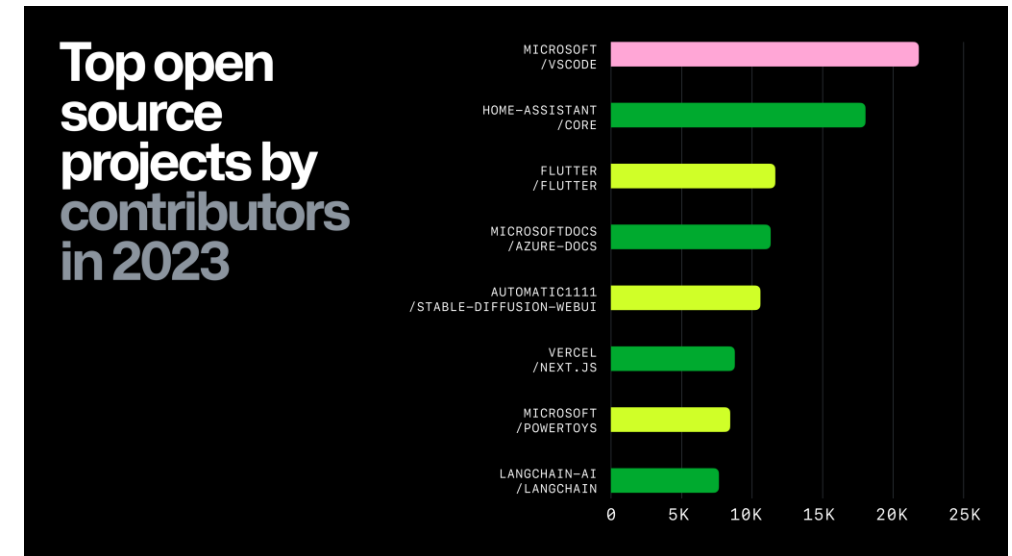
- 다양한 참여자 - 개인, 기업, 연구기관 등 다양한 주체가 참여
- 역할 분담 - 코딩, 문서화, 테스트, 디자인 등 다양한 역할 수행
- 지속적 발전 - 커뮤니티의 기여로 프로젝트가 유지되고 발전
- 자발적 참여 - 관심과 필요에 따라 자발적으로 참여



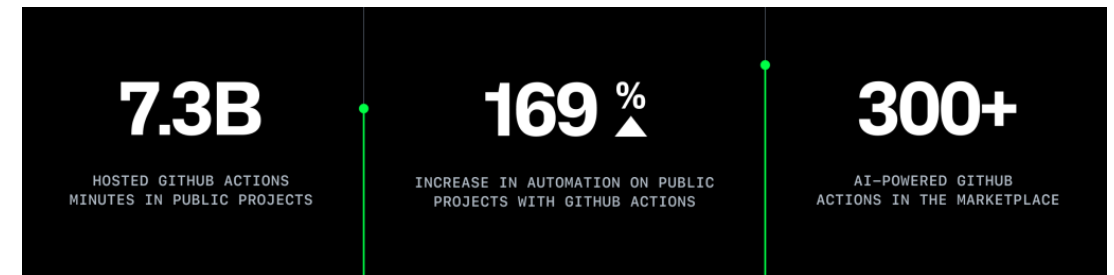
주요 오픈소스 커뮤니티 플랫폼

플랫폼	특징	대표 프로젝트
 GitHub	<ul style="list-style-type: none">가장 인기 있는 오픈소스 협업 플랫폼코드 저장소 및 관리 기능 제공MS에 인수되어 지속적인 기능 개선	VSCode (Microsoft) TensorFlow (Google) React (Facebook)
 GitLab	<ul style="list-style-type: none">GitHub과 유사하지만 자체 호스팅 가능CI/CD 기능이 강력함프라이빗 저장소 무료 제공	GNOME Debian GitLab 자체
 SourceForge	<ul style="list-style-type: none">초기 오픈소스 프로젝트 호스팅 플랫폼오래된 프로젝트가 많음무료 웹 호스팅 및 다운로드 제공	Apache OpenOffice GIMP FileZilla

자료 출처 :
공개SW 포털(한국지능정보사회진흥원, <https://www.oss.kr>) '2023년 공개SW 가이드 리포트'
https://www.oss.kr/oss_guide/show/d60f542e-9e7a-48e8-a3ec-ead75e9664e4



▲ 2023년 오픈소스 현황 분석



▲ 오픈소스에서 GitHub Actions* 사용 및 자동화 증가

* GitHub Actions

: 2018년부터 깃허브에서 제공하고 있는 서비스로, CI/CD와 같이 소프트웨어 개발 워크플로우를 자동화할 수 있는 도구

오픈소스 커뮤니티 문화 및 참여 방법

- 오픈소스 협업 방식 : 오픈소스 프로젝트는 **Git** 기반으로 관리되며, 협업을 통해 지속적으로 발전한다.

협업의 기본 과정

- 1 프로젝트를 Fork하여 개인 저장소로 복사
원본 프로젝트를 자신의 GitHub 계정으로 복사하여 작업할 수 있는 사본 생성
- 2 개선할 기능이나 버그를 찾고 Issue 생성
작업하기 전에 먼저 Issue를 생성하여 어떤 문제를 해결할지 커뮤니티에 공유
- 3 코드를 수정하고 Pull Request(PR) 제출
자신의 저장소에서 변경 사항을 적용한 후, 원본 프로젝트에 Pull Request를 생성
- 4 프로젝트 관리자가 코드 리뷰 후 Merge (병합)
Pull Request에 대한 검토가 이루어진 후, 승인되면 원본 프로젝트에 병합

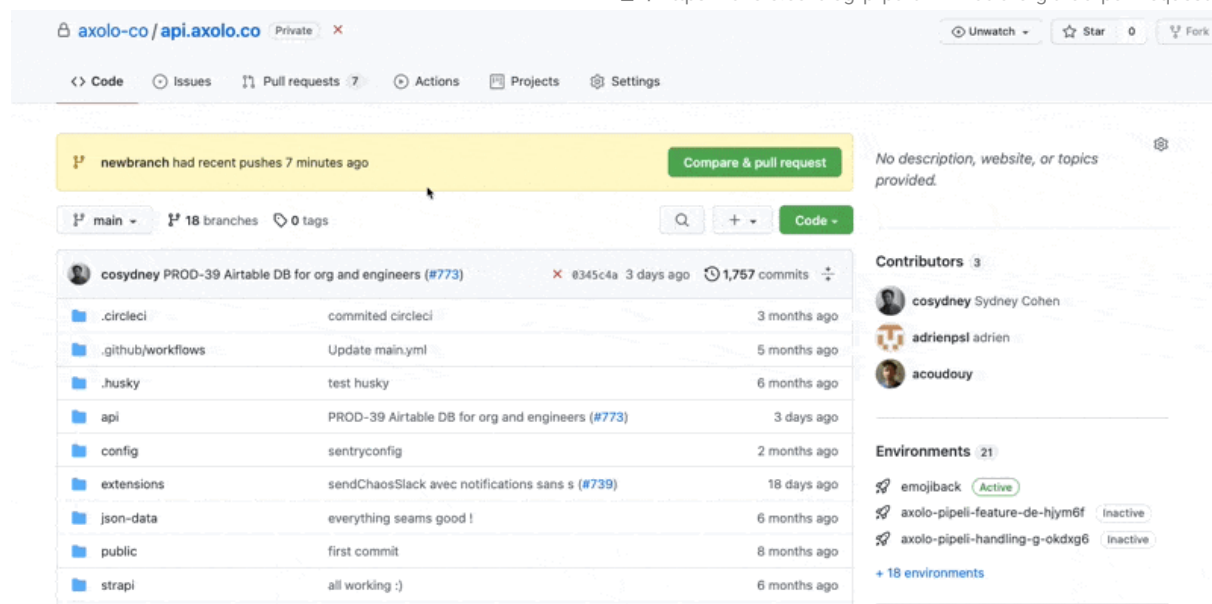


GitHub을 통한 협업 방식

협업 단계	설명
Fork	프로젝트를 개인 저장소로 복사 프로젝트 페이지에서 'Fork' 버튼 클릭
Issue	버그 수정 또는 기능 추가 요청 작업 내용을 미리 공유하고 논의
Pull Request	코드 변경 사항을 반영하도록 요청 변경 내용, 해결한 이슈, 영향도 설명
Code Review	다른 개발자가 코드 검토 후 승인 개선점 제안, 버그 발견, 코드 품질 향상
Merge	변경 사항을 공식 프로젝트에 반영 검토가 완료되면 메인 브랜치에 병합

▼ GitHub pull requests : gif summarizing

출처: <https://axolo.co/blog/p/part-1-what-are-github-pull-requests>



오픈소스 커뮤니티 문화 및 참여 방법

• 오픈소스에 기여하는 방법

다양한 기여 방식

- 코드 기여
버그 수정, 새로운 기능 개발, 성능 개선
- 문서화
README 개선, 튜토리얼 작성, API 문서 작성
- 번역
다양한 언어로 문서 및 인터페이스 번역
- 커뮤니티 지원
질문 답변, 이슈 분류, 토론 참여

첫 기여를 위한 팁

- ✓ "Good First Issue" 태그 활용
초보자 친화적인 이슈를 찾기 위해 GitHub에서 특별히 표시된 이슈 찾기
- ✓ 문서 개선부터 시작하기
코드 작성에 부담스럽다면 문서 개선, 오타 수정, 번역 기여부터 시작
- ✓ 작은 기여부터 점진적으로
처음에는 작은 기여부터 시작하여 점점 프로젝트에 익숙해지는 것이 중요
- ✓ 커뮤니티 가이드라인 읽기
프로젝트의 CONTRIBUTING.md 문서를 읽고 기여 방법 이해하기

○ Fix typo in documentation
project/repo • 2 days ago
good first issue documentation

○ Add unit test for login function
another/project • 5 days ago
good first issue testing

▲ "Good First Issue" 검색 화면 예시

기여 과정 다이어그램



03 오픈소스 프로젝트 진행방식

오픈소스 프로젝트의 개요

오픈소스 프로젝트 : 단순한 코드 저장소가 아닌, **지속적인 유지보수와 협업이 이루어지는 소프트웨어 생태계**

◆ 오픈소스 프로젝트의 특징

- 공개적 접근성 - 누구나 코드에 접근하고 기여할 수 있음
- 지속적 발전 - 문서화와 커뮤니티 피드백을 통해 계속 개선됨
- 투명한 의사결정 - 프로젝트 방향성과 결정 과정이 공개됨
- 경계없는 협업 - 세계 각지의 개발자들이 참여 가능

◆ 오픈소스 프로젝트 참여 방식

- **개인 프로젝트 공개**
개발자가 직접 만든 프로젝트를 오픈소스로 공개하고 커뮤니티를 구축
예: 개인 라이브러리, 도구, 애플리케이션
- **대형 프로젝트 기여**
기존의 대형 오픈소스 프로젝트에 코드, 문서 등으로 기여
예: Linux, React, TensorFlow에 기여

◆ 오픈소스 프로젝트 진행 과정






오픈소스 프로젝트의 개요

※ 오픈소스 프로젝트의 주요 구성 요소

구성 요소	설명
코드 저장소	GitHub, GitLab, SourceForge 등에서 코드 관리 소스 코드, 브랜치 관리, 버전 관리
문서화	README.md, CONTRIBUTING.md, Wiki 등 문서 제공 설치 방법, API 문서, 사용 예제, 기여 가이드라인
이슈 및 피드백 관리	Issue, Discussion, Bug Tracking을 통해 개선 버그 보고, 기능 요청, Q&A, 토론
커뮤니티 기여	코드 수정뿐만 아니라 번역, 디자인, 테스트 등의 기여 Pull Request, 코드 리뷰, 각종 기여 활동
지속적인 유지보수	코드 업데이트, 보안 패치, 기능 개선 지속 정기적인 릴리스, 버그 수정, 성능 개선



주요 오픈소스 커뮤니티 플랫폼

플랫폼	특징	대표 프로젝트
 GitHub	<ul style="list-style-type: none">가장 인기 있는 오픈소스 협업 플랫폼코드 저장소 및 관리 기능 제공MS에 인수되어 지속적인 기능 개선	VSCode (Microsoft) TensorFlow (Google) React (Facebook)
 GitLab	<ul style="list-style-type: none">GitHub과 유사하지만 자체 호스팅 가능CI/CD 기능이 강력함프라이빗 저장소 무료 제공	GNOME Debian GitLab 자체
 SourceForge	<ul style="list-style-type: none">초기 오픈소스 프로젝트 호스팅 플랫폼오래된 프로젝트가 많음무료 웹 호스팅 및 다운로드 제공	Apache OpenOffice GIMP FileZilla

오픈소스 프로젝트의 일반적인 구조

오픈소스 프로젝트 파일 구조

- 오픈소스 프로젝트에서 코드, 문서화, 협업을 위한 가이드라인을 체계적으로 관리하기 위한 디렉터리 및 파일 구성을 의미
- 개발자들이 프로젝트를 쉽게 이해하고 기여할 수 있도록 돕는 것이 목적

코드 (Source Code)

: 프로젝트의 핵심 소스 코드와 자동화 테스트, CI/CD 설정 등 포함

문서화 (Documentation)

: 프로젝트 이해와 사용에 필요한 모든 문서를 포함

협업 가이드라인

: 기여 방법, 코드 스타일, PR 리뷰 과정, 커뮤니티 행동 강령 등을 정의

파일명	역할	추가 고려 사항
README.md	프로젝트 소개 및 설치 방법 처음 방문한 사람이 가장 먼저 보는 문서	커뮤니티 친화적 문서화 필수 예제 코드, 스크린샷 포함 시 더 효과적
CONTRIBUTING.md	기여 방법 가이드 코드 작성부터 PR 제출까지의 과정 설명	PR 제출 방식과 코드 스타일 명시 코드 컨벤션, 커밋 메시지 형식 등 포함

파일명	역할	추가 고려 사항
LICENSE	오픈소스 라이선스 정보 코드 사용 및 배포 조건을 명시	기업 사용 가능 여부 결정 라이선스에 따라 상업적 활용 제한 가능
CODE_OF_CONDUCT.md	커뮤니티 윤리 규정 기여자들이 지켜야 할 행동 원칙	건전한 기여 환경 조성 포용적이고 존중하는 커뮤니티 문화 구축
ISSUE_TEMPLATE.md	버그 리포트 및 기능 요청 양식 이슈 생성 시 기본 템플릿 제공	문제 해결 프로세스 일관성 유지 필요한 정보를 정확히 수집하여 빠른 해결 지원
PULL_REQUEST_TEMPLATE.md	PR 제출 시 작성해야 할 기본 템플릿 변경 내용과 관련 이슈 연결 등 안내	코드 리뷰가 원활하게 진행될 수 있도록 지원 체크리스트로 PR 품질 향상
.github/workflows/	CI/CD 자동화 설정 자동 테스트, 빌드, 배포 등 설정	PR이 열릴 때 테스트 자동 실행 코드 품질을 유지하고 버그 조기 발견

GitHub 프로젝트 구조 분석 – README.md

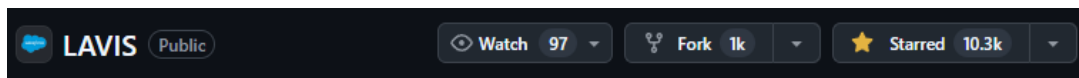
출처: <https://github.com/openai/CLIP>

README.md

: 프로젝트의 첫인상을 결정하는 문서

README의 효과

- GitHub 스타 증가 : 잘 작성된 README가 있는 프로젝트는 더 많은 Star와 Fork를 받을 수 있다.
- 기여자 유입 촉진 : 명확한 설명과 가이드는 새로운 기여자의 유입을 촉진한다.
- 커뮤니티 활성화 : 설치 및 사용이 쉬울수록 사용자 커뮤니티가 활성화된다.



▲ GitHub Repository Status 지표 예시

README.md 작성 체크리스트

기본 요소

- 프로젝트 이름과 로고
- 간결한 프로젝트 설명
- 목차 (큰 프로젝트의 경우)
- 상태 배지 (빌드, 버전 등)

커뮤니티 정보

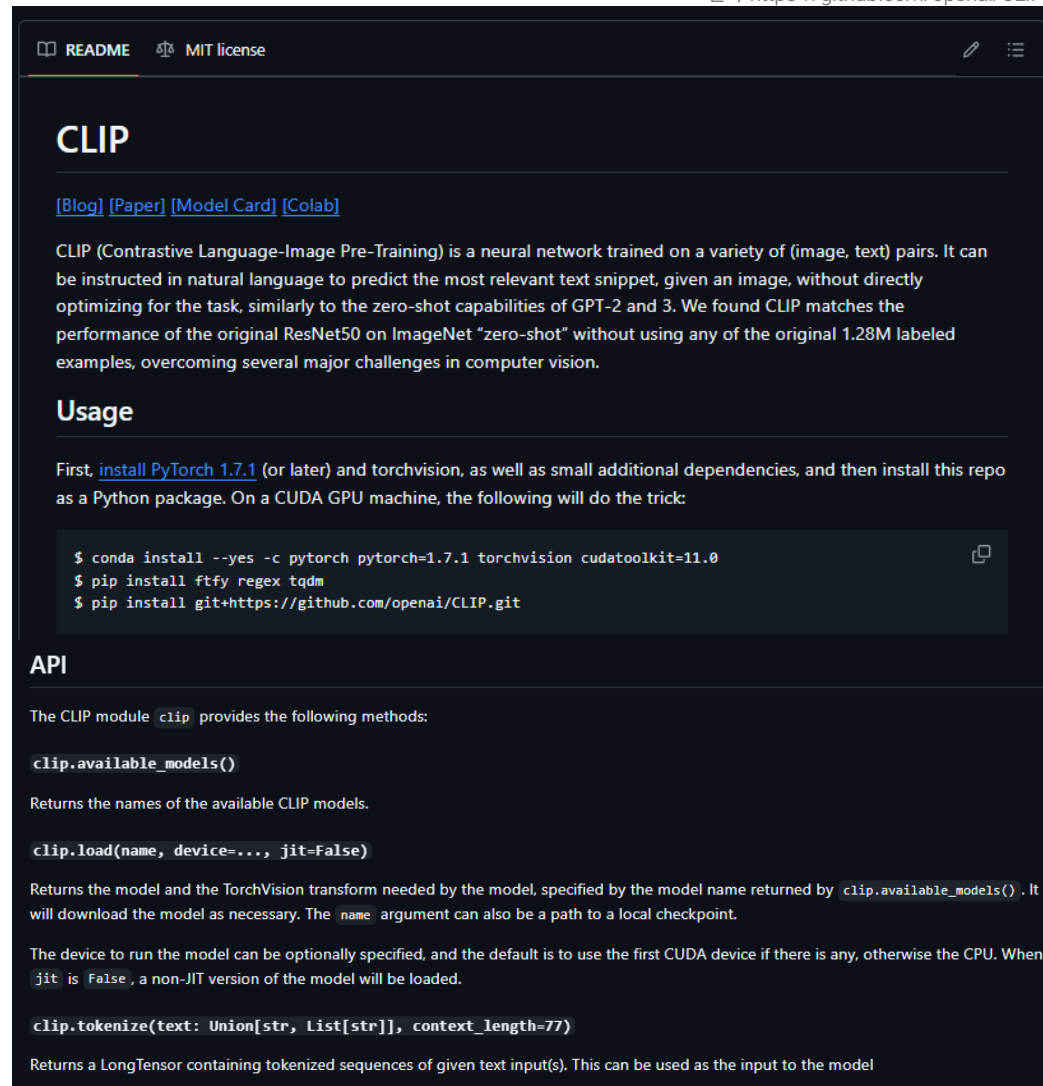
- 기여 방법 안내
- 행동 강령 링크
- 라이선스 정보
- 연락처 또는 커뮤니티 채널

설치 및 사용

- 필수 의존성 및 사전 요구사항
- 단계별 설치 방법
- 기본 사용 예제
- 실행 스크린샷 또는 데모

추가 개선 사항

- 자주 묻는 질문 (FAQ)
- 로드맵 또는 향후 계획
- 문제 해결 가이드
- 프로젝트 관련 출판물/인용



▲ README 예시

GitHub 프로젝트 구조 분석 - Insights 활용



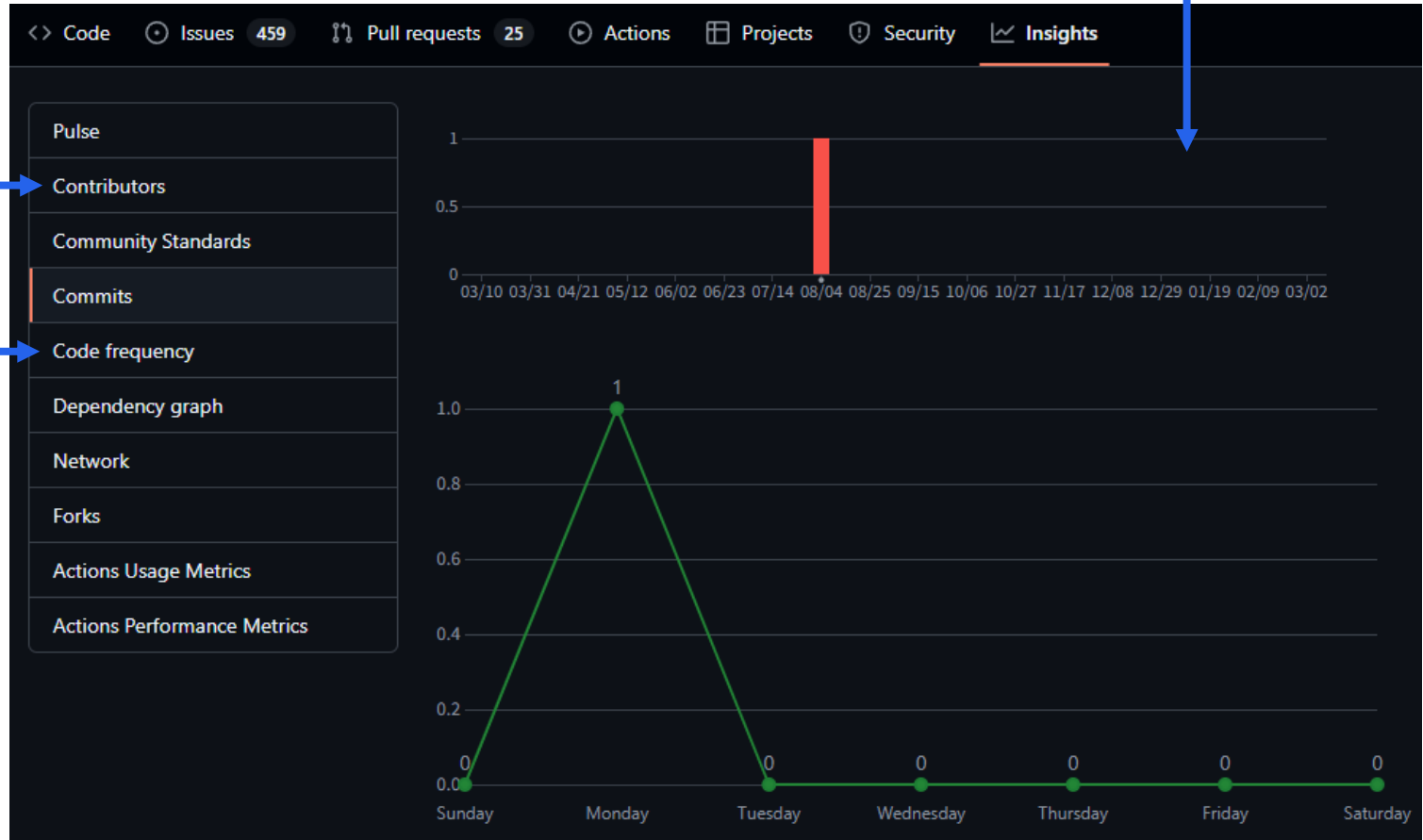
기여자 분석

Code frequency over the history of salesforce/LAVIS



코드 변경 패턴

커밋 빈도



GitHub 프로젝트 구조 분석 – Issue 관리

이슈 템플릿 (Issue Template)

- 프로젝트에서 이슈를 생성할 때 자동으로 적용되는 기본 양식
- `..github/ISSUE_TEMPLATE/` 디렉토리에 저장되며, 버그 리포트, 기능 요청 등 다양한 유형의 템플릿을 정의

장점

- 문제 해결에 필요한 정보를 체계적으로 수집
- 형식이 통일되어 이슈 분류 및 처리가 용이
- 초보 기여자도 쉽게 양질의 이슈를 생성 가능

이슈 템플릿 예시

```
name: 버그 리포트
description: 버그를 보고하는 템플릿
labels: [bug]

---

## 버그 설명
## 재현 방법
## 예상 동작
## 스크린샷
## 환경 정보
```

이슈 라벨(Label) 활용

-  **bug**
버그나 오류를 나타내는 이슈 라벨입니다. 수정이 필요한 문제를 표시합니다.
-  **enhancement**
새로운 기능 요청이나 기존 기능 개선 제안을 의미합니다.
-  **documentation**
문서 개선, 수정, 추가에 관련된 이슈입니다.
-  **good first issue**
초보 기여자에게 적합한 비교적 간단한 이슈를 표시합니다.
-  **help wanted**
커뮤니티의 도움이 필요한 이슈를 나타냅니다.
-  **wontfix**
현재 처리할 계획이 없는 이슈를 나타냅니다.

라벨 활용 전략:

- 우선순위 라벨(high, medium, low)을 사용하여 작업 순서 결정
- 버전별 라벨(v1.0, v2.0)로 로드맵 관리
- 영역별 라벨(frontend, backend, docs)로 담당자 할당

GitHub 프로젝트 구조 분석 – Pull Request(PR) 관리

PR 템플릿 (Pull Request Template)

- PR 생성 시 자동으로 적용되는 기본 양식
- `.github/PULL_REQUEST_TEMPLATE.md` 파일에서 관리
- 변경 사항을 명확히 정리하여 리뷰 절차를 표준화하고 효율적 리뷰 가능

PR 템플릿 예시

- 변경 내용, 관련 이슈, 테스트 방법, 체크리스트 포함 → PR 품질 유지
- Fixes #(이슈번호)를 포함하여 자동 이슈 닫기 가능

효과 : 일관된 PR 구조 제공, 리뷰어가 변경 사항을 쉽게 이해 가능, 체크리스트를 통해 기본적인 사항 누락 방지

```
## 변경 내용
<!-- 변경 사항 요약 -->

## 관련 이슈
Fixes #(이슈번호)

## 테스트 방법
<!-- 변경 사항을 테스트하는 방법 설명 -->

## 체크리스트
- [ ] 테스트 코드 작성
- [ ] 문서 업데이트
```

▲ PR 템플릿 예시

CI/CD*_(60p)를 활용한 자동 테스트

- `.github/workflows/` 디렉토리에 YAML 파일을 추가하여 PR 이벤트 발생 시 자동으로 빌드 & 테스트 실행

자동 테스트의 효과 :

- 코드 품질 유지 및 버그 조기 발견
- 리뷰어의 수동 테스트 부담 감소
- 병합 전 코드 검증으로 안정성 향상

```
name: CI

on: [pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup Node.js
        uses: actions/setup-node@v2
      - run: npm install
      - run: npm test
```

▲ GitHub Actions 워크플로우 예시

대규모 프로젝트에서의 PR 승인 프로세스

1

PR 제출

템플릿에 따라 PR 작성 및 제출

2

자동 테스트

CI/CD 파이프라인 실행

3

코드 리뷰

최소 2명의 리뷰어 승인 필요

4

병합

모든 조건 만족 시 병합 가능

프로젝트의 생애주기

오픈소스 프로젝트는 처음부터 완성된 소프트웨어가 아니라 **지속적인 성장과 개선을 거치는 과정**이다. 프로젝트가 성장하고 유지되기 위해서는 **기여자 친화적인 환경과 지속적인 유지보수 전략**이 필요하다.

◆ 생애주기

아이디어 단계 → 초기 구현
→ 커뮤니티 확장 → 유지보수 → 장기적 성장

◆ 성공적인 프로젝트의 생애주기 요소

- 명확한 비전과 목표
- 활발한 커뮤니티 지원
- 지속적인 개선과 혁신
- 코드 품질과 테스트
- 생태계 확장성



성공적인 오픈소스 프로젝트의 유지 및 성장 전략

오픈소스 프로젝트의 성공과 장기적인 유지를 위해서는 개발만큼 **유지보수와 커뮤니티 관리가 중요**

※ 프로젝트 유지 전략

유지 전략	설명	사례
기여자 친화적인 환경 조성	CONTRIBUTING.md, 코드 리뷰 프로세스 정립 신규 기여자의 진입 장벽을 낮추고, 기여 방법을 명확히 안내	React의 상세한 기여 가이드 코딩 스타일, 테스트 요구사항, PR 프로세스 안내
커뮤니티 피드백 적극 반영	Issue 및 PR 관리 체계 정립 사용자와 기여자의 피드백을 적극적으로 수집하고 반영	VSCode의 Issue 관리 시스템 라벨링, 마일스톤, 우선순위 체계적 관리
자동화된 CI/CD 도입	GitHub Actions, Travis CI 활용 코드 품질 검사, 테스트, 빌드, 배포 자동화	TensorFlow의 자동화 파이프라인 모든 PR에 자동 테스트 및 코드 품질 검사 진행
문서화 및 교육 자료 제공	튜토리얼, API 문서 제공 사용자와 기여자를 위한 상세한 문서 및 가이드 제공	Kubernetes의 포괄적인 문서 공식 문서, 튜토리얼, 블로그 등 다양한 리소스
정기적인 릴리스 및 로드맵 공개	개발 계획을 투명하게 공유 일정한 주기로 릴리스하고 향후 계획 공유	React의 버전 로드맵 실험적 기능, 향후 변경사항 공개

성공적인 오픈소스 프로젝트의 유지 및 성장 전략

오픈소스 프로젝트의 유지보수 - CI/CD 자동화 시스템

CI (Continuous Integration, 지속적 통합)

- 개발자가 코드 변경사항을 자주 병합하며, 자동화된 빌드 및 테스트를 통해 검증하는 개발 방식
- 주로 코드 품질 유지 및 버그 예방을 목적으로 사용
- CI의 핵심 요소
 - 코드 변경 시 자동 테스트 실행
 - 빌드 자동화 (컴파일 및 패키징)
 - 코드 품질 검사 (정적 분석 및 코드 스타일 검사)
 - 테스트 결과 보고 및 알림 (CI 도구를 통한 피드백 제공)

CD (Continuous Deployment, 지속적 배포)

- 모든 코드 변경사항이 **자동으로 테스트 및 검증(Continuous Delivery, 지속적 제공)**된 후, 모든 코드 변경사항이 **자동으로 운영 환경(프로덕션)에 배포됨 (Continuous Deployment, 지속적 배포)**
- 운영 배포(수동 승인) 후 배포(배포 자동화)
- CD의 핵심 요소:
 - 자동화된 배포 프로세스
 - 환경별 설정 관리 (개발, 테스트, 프로덕션)
 - 롤백 메커니즘
 - 배포 상태 모니터링

※ CI/CD 프로세스 흐름

단계	설명	예시
1. 코드 변경	개발자가 새로운 기능을 개발하거나 버그를 수정한 후 코드를 저장소에 푸시	새로운 기능 브랜치를 생성하고 작업 완료 후 GitHub에 푸시
2. Pull Request 생성	개발자가 작업한 내용을 메인 브랜치에 병합하기 위해 PR을 생성	GitHub UI에서 PR 생성 및 리뷰어 지정
3. 자동 CI 검사 실행	PR 생성 시 자동으로 빌드, 린트, 테스트 등이 실행	GitHub Actions, Travis CI 등이 자동으로 테스트 스크립트 실행
4. 코드 리뷰 & 승인	CI 검사 통과 후 유지보수자가 코드를 리뷰하고 승인	GitHub PR 리뷰 기능을 통한 코드 리뷰 및 "Approve" 처리
5. 자동 배포	승인된 PR이 메인 브랜치에 병합되면 자동으로 배포 과정이 시작	패키지 빌드 및 npm, PyPI 등에 자동 배포

※ 오픈소스 프로젝트에서 CI/CD 활용 사례

프로젝트	CI/CD 도구	활용 방식	효과
Linux Kernel	Travis CI	코드 변경 사항 자동 테스트 빌드 컴파일 및 다양한 환경에서 검증	호환성 이슈 조기 발견 다양한 하드웨어에서의 안정성 확보
TensorFlow	GitHub Actions	모델 학습 및 배포 자동화 멀티 플랫폼 테스트 자동화	다양한 환경 호환성 보장

성공적인 오픈소스 프로젝트의 유지 및 성장 전략

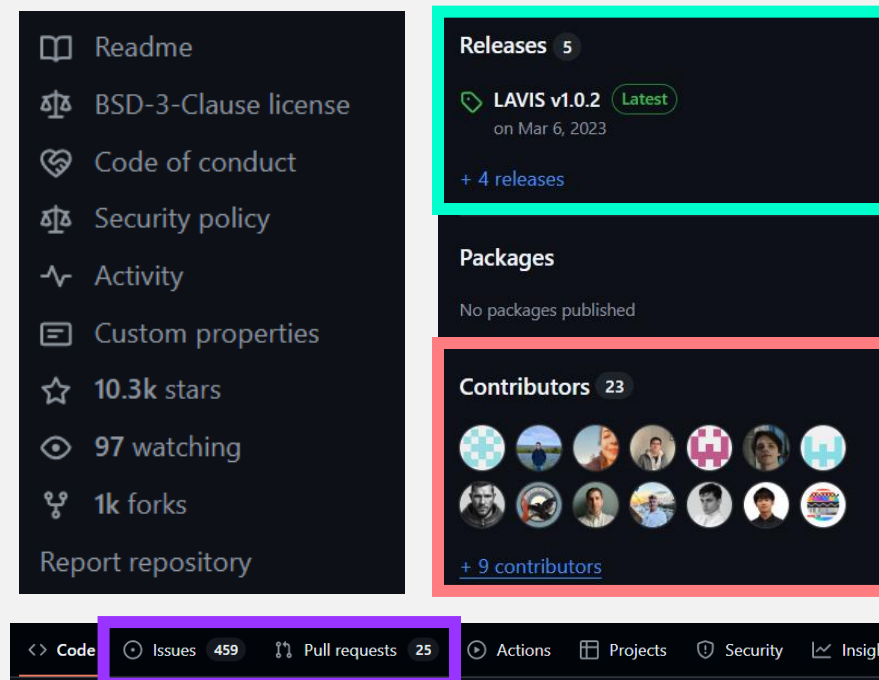
프로젝트 유지보수 상태 비교

오픈소스 프로젝트의 지속 가능성은 프로젝트의 유지보수 상태를 보면 쉽게 알 수 있음

기준	잘 유지되는 프로젝트	관리되지 않는 프로젝트
README 및 문서화	✓ 명확한 가이드 제공 상세한 설치 방법 및 사용 예제	✗ 설명 부족 불완전한 문서 또는 오래된 정보
Issue 및 PR 관리	✓ 최신 이슈 응답 및 해결 체계적인 PR 리뷰 및 병합 프로세스	✗ 오래된 이슈 방치 PR 응답 지연 또는 무시
CI/CD 활용 여부	✓ 자동화된 테스트 및 배포 체계적인 코드 품질 관리	✗ 수동 빌드, 릴리스 불명확 테스트 자동화 없음
커뮤니티 지원	✓ 활발한 토론 및 지원 다양한 기여자와 소통 채널	✗ 기여자 및 관리자가 없음 질문에 대한 응답 부재
릴리스 주기	✓ 정기적인 릴리스 명확한 버전 관리 및 릴리스 노트	✗ 오랫동안 릴리스 없음 버전 관리 미흡

GitHub에서 프로젝트 평가하기

- GitHub Insights의 'Contributors' 탭
- 'Releases' 탭에서 최근 릴리스 날짜
- Issue & PR 상태



<https://github.com/salesforce/LAVIS>

GitHub에서 좋은 프로젝트 찾기

GitHub에서 좋은 오픈소스 프로젝트를 찾는 기준

기준	설명	확인 방법
활동성	최근 커밋 여부 및 기여자 수 확인	GitHub Insights 탭에서 커밋 이력 확인
문서화	README, CONTRIBUTING 파일 여부	루트 디렉토리에서 문서 파일 확인
커뮤니티 참여	Issue 응답 속도, PR 승인율 분석	Issues 및 PR 탭에서 응답 속도 확인
라이선스	기업 또는 개인 사용 가능 여부	LICENSE 파일에서 라이선스 종류 확인

◆ 주의해야 할 점

- Star나 Fork 수만으로 판단하지 않기 → 방치된 프로젝트일 수 있음
- 대형 프로젝트가 항상 최선은 아님 → 진입 장벽이 높을 수 있음
- 트렌드만 쫓지 않기 → 본인의 관심사와 기술 스택에 맞는 프로젝트 찾기

GitHub에서 좋은 프로젝트 찾기

문서화 상태와 기여 가이드 확인

- 문서화 수준이 높을수록 신규 기여자가 쉽게 참여 가능
- README가 불충분하거나 기여 가이드가 없는 프로젝트는 진입 장벽이 높음
- 잘 정리된 문서는 코드 이해와 프로젝트 목적 파악을 용이하게 함

문서 파일	역할
README.md	프로젝트 개요, 설치 방법, 기본 사용법 제공
CONTRIBUTING.md	기여 방법 및 PR 제출 프로세스 설명
ISSUE_TEMPLATE.md	이슈 생성 가이드 (버그 리포트, 개선 요청 등)
PULL_REQUEST_TEMPLATE.md	PR 제출 시 필요한 정보 제공

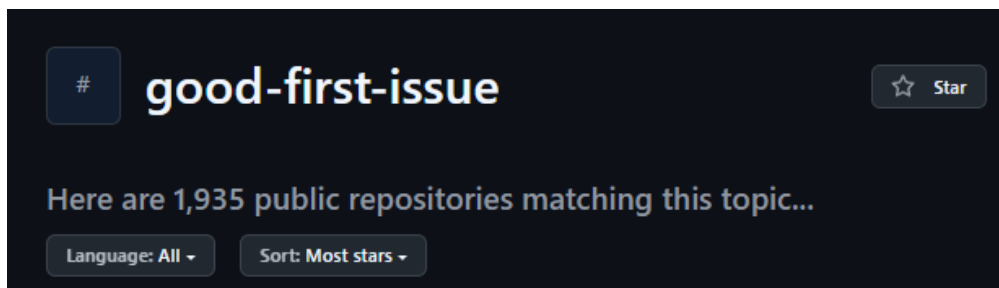
GitHub에서 커뮤니티 참여도 분석

- 개발자가 적극적으로 피드백을 주는 프로젝트일수록 기여 경험이 긍정적
- PR이 오래 열려 있거나, Issue 응답이 느린 프로젝트는 기여 후 피드백을 받기 어려울 수 있음
- 활발한 커뮤니티는 코드 리뷰를 통한 학습 기회 제공

지표	활발한 커뮤니티	비활성 커뮤니티
Issue 응답 시간	24-48시간 이내 응답	1주일 이상 응답 없음
PR 리뷰 속도	1주일 이내 리뷰 완료	한 달 이상 리뷰 없음
코드 리뷰 품질	상세하고 건설적인 피드백	피드백 없이 거절/방치
Issue 라벨링	체계적인 라벨 시스템 활용	라벨 없이 이슈 방치

GitHub에서 좋은 프로젝트 찾기

초보자를 위한 좋은 프로젝트 찾기 – “Good First Issue”



"Good First Issue"

: GitHub에서 초보자를 위한 기여를 유도하기 위해 사용되는 이슈 라벨

- 문서 개선 (오타 수정, 예제 추가)
- 작은 버그 수정 (간단한 엣지 케이스 처리)
- 단위 테스트 추가 (기존 기능에 대한 테스트)
- 로깅 개선 (디버깅 메시지 추가)

◆ "Good First Issue" 태그 검색 방법

1. GitHub 검색창에 다음 쿼리 입력:

```
is:issue is:open label:"good first issue"
```

2. 언어나 프로젝트별 필터 적용:

```
is:issue is:open label:"good first issue" language:javascript
```

오픈소스의 기여 방식

- 오픈소스 프로젝트에 기여하는 방법

※ 기여방식

기여 유형	설명	주요 역할
코드 기여	코드 작성 및 버그 수정	새로운 기능 추가, 성능 개선
문서 작성	프로젝트 문서 및 가이드 정리	README, 튜토리얼, API 문서 개선
번역 기여	다국어 지원을 위한 문서 번역	글로벌 사용자 확장
디자인 기여	UI/UX 개선, 로고 제작	프로젝트의 사용자 경험 향상
이슈 보고	버그 제보 및 개선 요청	문제점 공유 및 해결 기여
테스트 및 리뷰	코드 리뷰 및 테스트 진행	품질 관리 및 코드 검증

오픈소스의 기여 방식

- 초보자를 위한 효과적인 오픈소스 기여 전략

※ 초보자가 첫 기여를 할 때의 단계별 전략

단계	설명
1단계: 프로젝트 탐색	관심 있는 프로젝트를 찾아 GitHub의 'Good First Issue' 태그 검색
2단계: 문서화 확인	README, CONTRIBUTING.md를 읽고 기여 방법 숙지
3단계: 작은 기여 시작	문서 수정, 오타 수정, 코드 스타일 개선 등 작은 PR 제출
4단계: 커뮤니티 참여	Issues에 댓글 달고, 질문하면서 프로젝트 이해도 향상
5단계: 본격적인 코드 기여	기능 추가, 버그 수정 PR 제출 및 코드 리뷰 반영

◆ 초보자가 쉽게 기여할 수 있는 유형

- 문서화 및 튜토리얼 개선
README.md 개선, 코드 예제 추가, 설치 가이드 보완
- 버그 리포트
사용 중 발견한 문제를 Issue로 등록, 재현 방법 정리
- 기본적인 코드 스타일 수정
들여쓰기, 주석 추가, 변수명 개선 등 기초적인 리팩토링

◆ 초보자가 피해야 할 흔한 실수

- 지나치게 큰 PR 제출
한 번에 많은 코드 변경은 리뷰 어려움, 작은 단위로 나누기
- 문서를 읽지 않고 바로 코드 작성
CONTRIBUTING.md 등 프로젝트의 가이드라인 숙지 필수
- 피드백에 방어적 태도
코드 리뷰는 개인 비판이 아닌 코드 품질 향상을 위한 것

오픈소스의 기여 방식

• 효과적인 코드 기여 전략

※ 코드 기여를 할 때 고려해야 할 요소

고려 요소	설명
코드 스타일 준수	프로젝트에서 사용하는 코드 스타일 가이드 확인
기존 코드 분석	기존 코드의 구조를 이해하고, 동일한 패턴을 유지
테스트 코드 작성	새로운 기능 추가 시 테스트 코드 포함
PR 제목 및 설명 명확히 작성	어떤 변경을 했는지 간결하게 설명

◆ 코드 기여 시 좋은 PR 작성법

- 한 번에 너무 많은 변경을 하지 않기
작은 PR이 Merge 확률이 높음 (기능별로 분리)
- PR 제목을 명확하게 작성
좋은 예: "Fix: 로그인 오류 수정", "Feat: 사용자 검색 기능 추가"
- PR 본문에 변경 이유 및 테스트 결과를 포함
왜 이 변경이 필요한지, 어떻게 테스트했는지 설명

◆ 커뮤니티에서 협업할 때의 핵심 전략

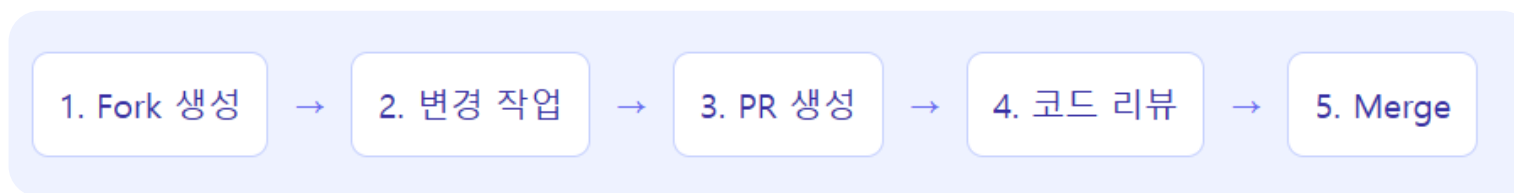
- 질문할 때는 명확하게 질문하기
재현 가능한 예제와 시도한 해결책 포함
- 기여 전에는 반드시 README 및 CONTRIBUTING.md 확인
프로젝트별 규칙과 가이드라인 준수
- 코드 리뷰 피드백 적극 반영
피드백을 개인적으로 받아들이지 않고 성장 기회로 활용

오픈소스 협업 방식 (Fork, Pull Request, Issue 활용법)

Pull Request(PR)란?

- Pull Request(PR)는 오픈소스 프로젝트에서 외부 기여자가 코드 변경 사항을 프로젝트에 병합(Merge)하도록 요청하는 방식
- GitHub, GitLab 등에서 기본적인 코드 협업의 단위로 사용되며, PR이 승인되면 코드가 공식 코드베이스(메인 Branch)로 합쳐진다
- 효과 : 코드 변경 사항 검토 / 팀원 간 협업 강화 / 기록 및 문서화 / 자동화된 테스트 연동

※ PR 흐름도



오픈소스 협업 방식 (Fork, Pull Request, Issue 활용법)

- 효과적으로 PR을 제출하는 방법

※ PR 제출 전 체크리스트

체크 항목	설명
이슈 확인	관련 Issue가 있는지 확인하고 연결
브랜치 전략 준수	main 또는 develop 브랜치에서 직접 작업하지 않기
코드 스타일 준수	프로젝트의 코드 스타일 가이드 확인
테스트 코드 작성	기능 추가 시 테스트 코드 포함
문서화 추가	README 또는 API 문서 업데이트 포함

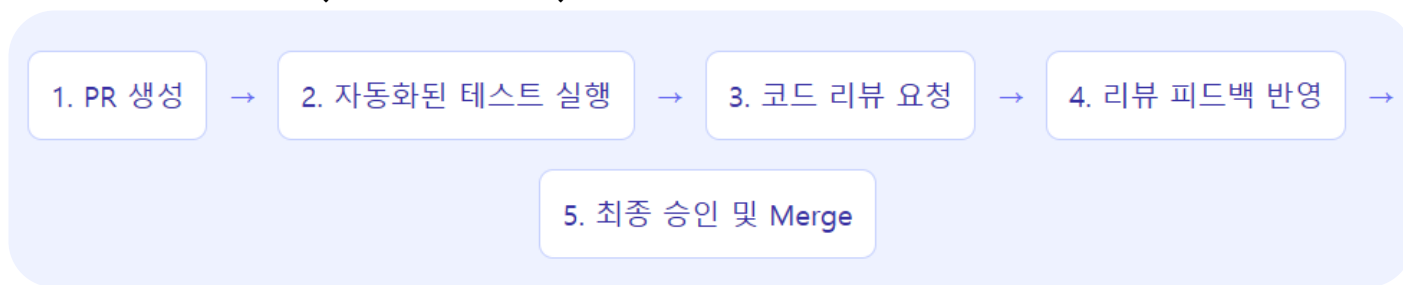
※ PR 작성 시 포함해야 할 내용

항목	설명
PR 제목	변경 사항을 명확히 설명 (예: "Fix: 로그인 오류 수정")
PR 본문	변경한 이유, 주요 코드 변경 사항, 테스트 결과 포함
관련 이슈 연결	해결하려는 이슈 번호(#123) 참조
스크린샷 또는 로그 첨부	UI 변경 또는 주요 기능 수정 시 필요

오픈소스 협업 방식 (Fork, Pull Request, Issue 활용법)

• PR의 승인 과정

※ PR 승인 과정 (GitHub 기준)



◆ 코드리뷰 기준

- 코드 스타일 준수 여부
들여쓰기, 변수명 규칙, 주석 등 코딩 규칙 준수
- 기능 정상 동작 여부
테스트 코드 실행 결과, 에지 케이스 처리
- 보안 취약점 존재 여부
SQL Injection, XSS 등 보안 이슈 확인

◆ GitHub PR 상태 유형

- Draft (초안)
- Changes Requested (수정 요청)
- Review Required (리뷰 필요)
- Approved (승인됨)
- Merged (병합됨)

• PR이 거절되는 이유와 해결 방법

PR이 거절되는 일반적인 이유

- 코드 스타일 위반 - 프로젝트의 코딩 규칙 미준수
- 테스트 부족 또는 실패 - 테스트 코드 누락 또는 기존 테스트 실패
- 문서화 부족 - 변경 사항에 대한 문서 업데이트 누락
- 너무 큰 PR - 한 번에 너무 많은 파일 변경
- 프로젝트 방향성 불일치 - 프로젝트 로드맵과 맞지 않는 기능 추가

효과적인 해결 방법

- PR 크기 축소 - 큰 변경사항은 여러 개의 작은 PR로 분할
- 코드 리뷰 적극 반영 - 받은 피드백을 성실히 반영하고 개선
- 테스트 케이스 추가 - 변경 사항에 대한 충분한 테스트 작성
- 문서화 보완 - API 변경시 관련 문서도 함께 업데이트
- 사전 이슈 논의 - 큰 변경 전 이슈로 먼저 논의하기

성공적인 오픈소스 프로젝트의 조건

- 성공적인 오픈소스 프로젝트의 **핵심 요소**

성공적인 오픈소스 프로젝트는 단순히 코드가 공개되어 있는 것이 아니라, 지속적인 기여와 유지보수를 통해 성장하는 프로젝트 장기적인 유지보수와 커뮤니티 활성화가 필수적이다

요소	설명
1. 명확한 목적과 비전	프로젝트가 해결하려는 문제와 목표가 명확해야 함
2. 지속적인 유지보수	코드 업데이트, 버그 수정이 정기적으로 이루어져야 함
3. 커뮤니티 참여 유도	기여자들이 쉽게 참여할 수 있도록 문서화 및 피드백 제공
4. 자동화된 개발 및 배포 프로세스	CI/CD 도입을 통한 테스트 및 배포 자동화
5. 라이선스 및 법적 고려	프로젝트의 오픈소스 라이선스를 명확하게 정의

◆ 성공적인 프로젝트의 특징

- **활발한 커밋 기록**
최근까지 지속적인 코드 업데이트가 이루어짐
- **다수의 기여자 참여**
다양한 개발자가 코드에 기여하고 있음
- **체계적인 이슈 관리**
이슈가 빠르게 처리되고 태그가 잘 정리됨
- **상세한 문서화**
API 문서, 설치 가이드, 기여 방법이 잘 정리됨
- **활성화된 커뮤니티**
Discord, Slack 등 소통 채널이 운영됨

◆ 오픈소스 프로젝트가 실패하는 경우

- **기여자 부족**
프로젝트에 지속적으로 기여하는 개발자가 없음
- **문서화 부족**
README, 기여 가이드가 없어 신규 기여자가 참여하기 어려움
- **버그 및 유지보수 지연**
이슈가 방치되면서 사용자 신뢰도 하락
- **커뮤니티 소통 부족**
질문에 대한 응답이 느리고 관리자가 소통하지 않음
- **비전 및 로드맵 부재**
프로젝트의 장기적인 방향성이 불명확함

성공적인 오픈소스 프로젝트의 조건

- 오픈소스 프로젝트의 **유지보수** 전략

오픈소스 프로젝트가 지속적으로 성장하기 위해서는 체계적인 유지보수 전략이 필요
단기적인 기여보다 장기적인 프로젝트 건강성을 고려한 접근이 중요하다

전략	설명
기여자 친화적인 환경 조성	CONTRIBUTING.md, 코드 리뷰 프로세스 정립
이슈 및 PR 관리 체계 구축	Issue 라벨링, PR 승인 기준 명확화
자동화된 테스트 및 배포	CI/CD를 활용하여 코드 품질 유지
정기적인 릴리스	기능 추가 및 버그 수정 릴리스 일정 관리
커뮤니티 소통 강화	GitHub Discussions, Discord, Slack 등을 활용

◆ CI/CD 도입을 통한 자동화

자동화된 CI/CD 파이프라인을 구축하면,
코드 품질을 일관되게 유지하고 유지보수
부담을 크게 줄일 수 있다

CI/CD 파이프라인 흐름

1. 코드 변경 사항 Push
2. 자동 린트 및 코드 스타일 검사
3. 단위 및 통합 테스트 실행
4. 테스트 커버리지 검사
5. 문서 자동 생성
6. 테스트 통과 시 자동 배포

◆ 유지보수가 잘 되는 프로젝트의 특징

- PR이 빠르게 리뷰되고 Merge됨
- 이슈가 방치되지 않고 정기적으로 해결됨
- 커뮤니티가 활발하게 운영됨
- 자동화된 CI/CD가 도입됨

성공적인 오픈소스 프로젝트의 조건

- 성공적인 오픈소스 프로젝트를 위한 **커뮤니티** 운영 전략

오픈소스 프로젝트는 기여자와 사용자들이 함께 만들어가는 협업 시스템

프로젝트가 지속되려면 기여자들이 쉽게 접근하고 참여할 수 있는 환경 조성이 필수적이다

전략	설명
새로운 기여자 환영	"Good First Issue" 태그 활용 및 가이드 제공
명확한 문서화 제공	README.md, CONTRIBUTING.md 업데이트
적극적인 코드 리뷰 및 피드백	PR 리뷰 시간을 단축하고, 명확한 피드백 제공
커뮤니티 이벤트 개최	Contributor Meetup, Hackathon, 온라인 워크숍 운영
커뮤니케이션 채널 유지	Discord, Slack, GitHub Discussions 활용

◆ 성공적인 커뮤니티 운영 사례

Kubernetes

- SIG(Special Interest Group) 체계 운영
- 정기적인 Contributor Summit 개최
- 명확한 거버넌스 문서화

React

- GitHub Discussions 적극 활용
- 상세한 RFC 프로세스
- 정기적인 블로그 업데이트

VS Code

- 상세한 기여자 가이드
- 월간 반복 개발 주기
- 투명한 로드맵 공유

기업과 오픈소스: 참여 방식 및 사례 분석

• 기업의 참여 방식

- 기업은 단순히 오픈소스를 사용하기만 하는 것이 아니라, 적극적으로 기여하고 자체적인 오픈소스 프로젝트를 운영한다
- 기업이 기존 오픈소스를 활용하게 되면 '개발 비용이 절감'되며 개방형 개발 방식을 통해 '기술 발전 속도 증가'하고, 오픈소스에 기여하는 기업은 '브랜드 신뢰도 강화'로 이어질 수 있기 때문이다

◆ 내부 도구로 활용

- 개발 비용 절감
- 오픈소스 라이브러리 활용
- 개발 속도 향상
- 내부 개발 표준화

◆ 외부 프로젝트에 기여

- 자사의 요구에 맞게 기능 개선
- 버그 수정 및 성능 향상
- 개발자 커뮤니티 인지도 확보
- 기술적 영향력 강화

◆ 자체 오픈소스 프로젝트 운영

- 기술 리더십 확보
- 외부 개발자 기여 유도
- 생태계 구축 및 확장
- 관련 비즈니스 모델 개발

기업과 오픈소스: 참여 방식 및 사례 분석

- 기업이 오픈소스 프로젝트에 기여하는 형태

기여 방식	설명	대표 기업
코드 기여	오픈소스 프로젝트에 코드 추가 및 수정	Google (TensorFlow), Microsoft (VSCode)
재정적 지원	오픈소스 프로젝트에 자금 제공	Meta (PyTorch), Linux Foundation
오픈소스 컨퍼런스 개최	개발자 컨퍼런스를 통해 생태계 조성	Red Hat, IBM
오픈소스 개발자 고용	오픈소스 유지보수자를 직접 고용	Amazon (Kubernetes), Google

◆ Google의 오픈소스 기여 사례

주요 프로젝트

- Kubernetes: 컨테이너 오케스트레이션 도구
- TensorFlow: 머신러닝 프레임워크
- Android: 모바일 OS
- Chromium: 웹 브라우저 엔진

기여 방식

- 핵심 개발자 팀을 구성하여 프로젝트 리드
- 정기적인 커밋 및 릴리스 관리
- 개발자 교육 및 문서화 지원
- Google Summer of Code 통한 신규 기여자 유입

오픈소스 참여 시 유의할 점 : 보안, 윤리, 라이선스 문제

• 오픈소스 프로젝트에서의 다양한 보안 위험

- 오픈소스는 누구나 접근할 수 있기 때문에 보안 취약점이 발생할 가능성이 높음
- 공급망 공격(Supply Chain Attack)을 방지하려면 의존성 검토가 필수
- PR 검토 시 코드 보안 점검을 포함하는 것이 중요

▼ 오픈소스에서 발생할 수 있는 보안 위험

보안 위험	설명	예방 방법
취약한 코드	보안 리뷰 없이 Merge된 코드로 인해 보안 취약점 발생 가능	코드 리뷰 시 보안 관점 검토, SAST 도구 활용
의도적인 악성 코드 삽입	공격자가 악성 코드가 포함된 PR을 승인받아 배포할 가능성 있음	다중 검토자 시스템, 자동화된 보안 검사
의존성(Dependency) 공격	오픈소스 라이브러리의 취약점을 이용한 공급망 공격	의존성 검사 도구 사용 (Dependabot, Snyk)
민감한 정보 노출	코드 내 API 키, 비밀번호 등이 실수로 공개될 가능성	환경 변수 사용, .gitignore 설정, 비밀 스캐닝 도구

PR 검토 시 보안 체크리스트

- ✓ 입력값 검증 및 유효성 확인
- ✓ SQL 인젝션 및 XSS 방지 확인
- ✓ 민감 정보(API 키, 비밀번호) 포함 여부 확인
- ✓ 사용자 권한 관리 및 인증 로직 검토
- ✓ 오류 처리 및 로깅 적절성 확인

자동화된 보안 검사 도구

- GitHub Dependabot
의존성 패키지의 보안 취약점 자동 감지 및 패치
- Snyk
코드와 의존성의 취약점 스캔 및 모니터링
- SonarQube
정적 코드 분석을 통한 보안 이슈 감지
- GitHub Secret Scanning
코드에 포함된 API 키 및 비밀번호 감지

오픈소스 참여 시 유의할 점 : 보안, 윤리, 라이선스 문제

• 오픈소스에서 윤리적으로 고려해야 할 사항

- 오픈소스 프로젝트에서 저작권 침해와 공정한 기여 기회 보장이 중요
- CODE_OF_CONDUCT.md 파일을 활용하여 커뮤니티 윤리를 정립할 필요 있음
- PR 제출 시 출처를 명확히 하고, 라이선스를 준수해야 법적 분쟁을 방지 가능

▼ 오픈소스에서 윤리적으로 고려해야 할 사항

윤리적 이슈	설명	해결 방법
저작권 침해	원작자의 허가 없이 코드를 복사하여 사용	출처 표시, 라이선스 준수, 허가된 범위 내에서 사용
공정한 기여 기회 보장	특정 그룹만 기여할 수 있도록 제한하는 경우	포용적인 커뮤니티 지침 마련, 다양한 기여자 환영
커뮤니티 내 갈등	코드 스타일, 기능 방향성 등을 두고 발생하는 분쟁	명확한 기여 가이드라인, 중재 프로세스 마련
사용자 보호	악성 코드가 포함되거나 사용자를 속이는 기능이 추가될 가능성	코드 검토 과정 강화, 투명한 기능 설명 제공

윤리적 문제 사례 분석

코드 표절 사례

문제: 특정 개발자가 다른 프로젝트의 코드를 복사하여 자신의 기여물로 제출

해결: 코드 검토 과정에서 표절 발견, 원작자에게 허가 요청 및 출처 표시 요구

■ CODE_OF_CONDUCT.md 파일을 활용하여 커뮤니티 윤리를 정립할 필요가 있음

서약 (Pledge)

모든 참여자가 존중받는 환경을 조성하겠다는 서약

시행 (Enforcement)

규칙 위반 시 대응 방법과 처리 과정

기준 (Standards)

긍정적인 행동과 금지되는 행동에 대한 명확한 기준 제시

범위 (Scope)

행동 강령이 적용되는 범위와 공간 정의

오픈소스 참여 시 유의할 점 : 보안, 윤리, 라이선스 문제

• 오픈소스 라이선스를 준수하는 방법

- 오픈소스 라이선스를 준수하지 않으면 법적 문제가 발생할 수 있음
- LICENSE 파일을 확인하고, 적절한 라이선스를 적용하는 것이 중요
- 기업에서는 오픈소스 사용 규정을 명확히 설정하고, 법적 리스크를 줄이는 노력이 필요

▼ 오픈소스 라이선스 위반 사례 및 예방방법

라이선스 위반 사례	설명	예방 방법
GPL 코드 비공개 배포	GPL 라이선스 코드를 수정하고 비공개로 배포하면 위반	파생 코드도 GPL로 공개, 소스코드 제공 의무 준수
저작권 표시 제거	원작자의 라이선스 및 저작권 정보를 삭제한 경우	저작권 및 라이선스 표시 유지, 출처 명시
라이선스 혼용 문제	서로 다른 라이선스(예: MIT+GPL)를 적절히 관리하지 않은 경우	라이선스 호환성 확인, 호환되지 않는 라이선스는 분리 사용
기업의 오픈소스 사용 규정 위반	상업용 제품에 특정 라이선스를 혼합하여 사용할 경우 발생하는 법적 문제	법무팀 검토, 오픈소스 사용 정책 준수, 라이선스 감사

주요 오픈소스 라이선스 비교

라이선스	소스코드 공개 의무	상업적 사용
MIT	×	○
Apache 2.0	×	○
GPL 3.0	○	○
LGPL 3.0	일부만	○
BSD	×	○

오픈소스 라이선스 준수를 위한 전략

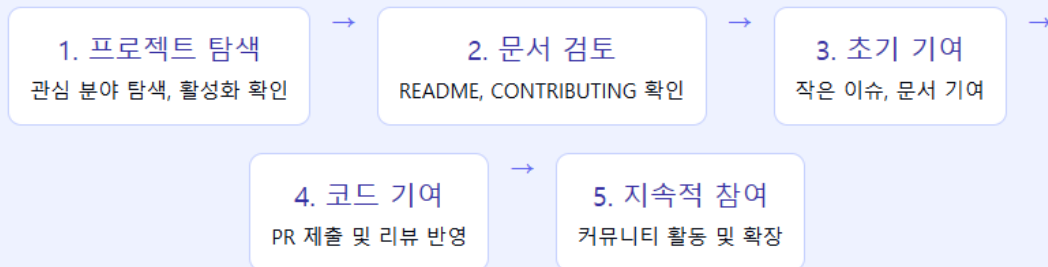
- LICENSE 파일 확인
프로젝트와 호환되는 라이선스인지 확인
- 라이선스 호환성 검토
혼용하는 경우 호환성 문제 없는지 확인
- 오픈소스 사용 정책 수립
기업에서는 명확한 오픈소스 사용 가이드라인 필요
- 자동화 도구 활용
FOSSA, Black Duck 등 라이선스 검토 도구 사용

오픈소스 프로젝트, 어떻게 참여할 것인가?

◆ 요약

주요 주제	핵심 내용
오픈소스의 개념 및 철학	오픈소스는 개방성과 협업을 중심으로 운영됨
오픈소스 프로젝트의 구조	README, LICENSE, ISSUE 관리 등이 기여를 용이하게 함
효과적인 오픈소스 기여 방식	코드뿐만 아니라 문서, 번역, 디자인 등 다양한 기여 가능
PR 승인 과정 및 리뷰 시스템	효율적인 PR 작성과 코드 리뷰 반영이 중요
성공적인 프로젝트 운영 전략	유지보수, 커뮤니티 활성화, 자동화된 CI/CD 도입 필수
기업과 오픈소스의 관계	기업은 오픈소스를 활용하고 기여하며 생태계를 조성
오픈소스 참여 시 유의할 점	보안, 윤리, 라이선스 문제를 고려해야 함

오픈소스 참여 프로세스



◆ 오픈소스 프로젝트 참여를 위한 다음 단계

관심 있는 오픈소스 프로젝트 찾기

- GitHub에서 'Good First Issue' 검색
GitHub 검색창에 is:issue is:open label:"good first issue" 입력
- 사용 중인 오픈소스 탐색
자주 사용하는 라이브러리나 도구의 GitHub 페이지 방문
- 관심 분야 프로젝트 찾기
topic:machine-learning language:python 등으로 검색

기여하기 전 준비사항

- 문서 확인하기 - README.md, CONTRIBUTING.md, CODE_OF_CONDUCT.md 검토
- 개발 환경 설정 - 프로젝트 클론, 빌드, 테스트 실행 방법 숙지
- 커뮤니케이션 채널 확인 - Slack, Discord, 메일링 리스트 등 참여

작은 기여부터 시작하기

- 문서 기여 - 오타 수정, 예제 추가, 설명 개선 등 문서 관련 작업
- 번역 작업 - 문서나 UI의 다국어 지원을 위한 번역 기여
- 단위 테스트 추가 - 기존 기능에 대한 테스트 코드 작성 및 추가