7. GitHub의 핵심 기능

GitHub 계정 생성 및 SSH 키 설정

✅ 1. GitHub 계정 생성

① GitHub 공식 페이지 접속

👉 https://github.com/

② 회원가입 절차

- **Username**: 공개되는 GitHub ID
- **Email**: 로그인/푸시용
- **Password**: 강력한 비밀번호 입력
- **무료 플랜** 선택 후 가입

✅ 2. SSH 키란?

- GitHub와 내 컴퓨터 간 **암호화된 인증**을 위한 키 쌍
- GitHub는 **비밀번호 없이** 안전하게 `git push` 가능하게 해줌

> Git 명령어로 GitHub에 접근할 때 HTTPS 대신 **SSH 방식**을 쓰기 위해 꼭 설정해야 해.

✅ 3. SSH 키 생성 (로컬 PC에서)

▶️ 명령어 입력 (터미널/명령프롬프트/Git Bash)

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

- `ed25519`: 최신 알고리즘 (권장)
- `-C`: 설명용 주석

▶️ 출력 예

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/you/.ssh/id_ed25519):
```

✅ 그냥 엔터 → 기본 위치에 생성됨
(기존 키 덮어쓰지 않게 조심)

✅ 4. SSH 공개키 GitHub에 등록

① 공개키 내용 복사

```
cat ~/.ssh/id_ed25519.pub
```

또는

```
clip < ~/.ssh/id_ed25519.pub  # Windows
pbcopy < ~/.ssh/id_ed25519.pub  # macOS
```

② GitHub 사이트 → SSH 키 등록

- 오른쪽 상단 → [Settings] → [SSH and GPG keys]
- [New SSH key] 클릭
- Title: 내 PC 이름 등
- Key: 방금 복사한 `id_ed25519.pub` 내용 붙여넣기
- 저장

✅ 5. SSH 연결 테스트

```
ssh -T git@github.com
```

처음 시도 시:

```
The authenticity of host 'github.com (IP)' can't be established.
Are you sure you want to continue connecting (yes/no)?
```

→ `yes` 입력

성공 시:

```
Hi your-username! You've successfully authenticated, but GitHub does not provide shell access.
```

✅ 6. Git에서 SSH 주소로 사용하기

```
git clone git@github.com:username/repo.git
```

👉 주소 복사 시 GitHub에서 **"SSH" 탭** 선택 후 주소 복사

✅ 7. Git 설정에 사용자 정보 입력

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

⚠️ 이 email은 GitHub 계정의 이메일과 같아야 커밋이 GitHub 프로필과 연결됨

✅ 마무리 요약

작업	명령어/링크
SSH 키 생성	`ssh-keygen -t ed25519 -C "이메일"`
공개키 확인	`cat ~/.ssh/id_ed25519.pub`
GitHub에 등록	GitHub → Settings → SSH Keys
연결 테스트	`ssh -T git@github.com`
Git 설정	`git config --global user.name/email`
클론 (SSH)	`git clone git@github.com:...`

Repository 만들기 (공개/비공개)

✅ 1. 저장소(Repository)란?

- Git 프로젝트를 저장하는 **단위 공간**
- GitHub에서 **코드**, **커밋 이력**, **브랜치**, **이슈**, **위키**, **PR** 등을 관리
- 로컬 저장소와 연결해서 `git push/pull`으로 동기화함

✅ 2. 저장소 생성 방법

▶️ ① GitHub 접속 후

1. 상단 우측 `+` 클릭 → **New repository** 선택
   👉 또는 https://github.com/new

▶️ ② Repository 정보 입력

항목	설명
Repository name	고유한 이름 지정 (`my-first-repo` 등)
Description (optional)	저장소 설명
Public	**누구나 볼 수 있음** (오픈소스, 포트폴리오에 적합)
Private	**나와 팀원만 볼 수 있음** (회사 코드, 개인 실험 등)

항목	설명
✔ Initialize with a README	기본 README 파일 생성 (선택)
.gitignore	언어에 맞는 무시 파일 생성 (예: `Python`, `Node`)
License	MIT, Apache 등 오픈소스 라이선스 선택 (필요 시)

▶️ ③ [Create repository] 버튼 클릭

✅ 3. 이후 연결 방법

🔹 ① 새 저장소를 로컬에 클론

```
git clone git@github.com:username/repo-name.git
```

→ SSH 방식 (추천)

또는

```
git clone https://github.com/username/repo-name.git
```

→ HTTPS 방식 (매번 로그인 필요)

🔹 ② 기존 프로젝트와 연결

```
cd my-project/
git init
git remote add origin git@github.com:username/repo-name.git
git add .
git commit -m "초기 커밋"
git push -u origin main
```

✅ 4. Public vs Private 차이점

항목	Public	Private
접근 권한	누구나 가능	설정된 사용자만
검색 노출	Google, GitHub 검색에 노출	비공개
오픈소스 기여	적합	부적합
비용	무료	팀 규모 제한 시 유료 가능성
GitHub Copilot 학습 데이터	사용됨	사용되지 않음

✅ 5. 저장소 공개/비공개 전환

1. 저장소 > Settings > General
2. "Change repository visibility" 섹션
3. "Make private" 또는 "Make public" 클릭

⚠️ 비공개 → 공개 시 전체 공개되므로 주의!
⚠️ 공개 → 비공개는 유료 요금제일 경우 제한 없음

✅ 6. 실전 예시: 나만의 첫 GitHub 저장소 만들기

1. `my-first-app`이라는 이름으로 GitHub에서 저장소 생성
2. `README`, `.gitignore(Node.js)`, `MIT License` 선택
3. 로컬에서 프로젝트 폴더 만들고 연결:

```
mkdir my-first-app && cd my-first-app
npm init -y
git init
git remote add origin git@github.com:your-id/my-first-app.git
git add .
git commit -m "init: 프로젝트 초기화"
git push -u origin main
```

✅ 마무리 요약

작업	명령어 또는 위치
저장소 생성	GitHub → + → New repository
로컬에 클론	`git clone <url>`
로컬 프로젝트 연결	`git remote add origin <url>`
최초 푸시	`git push -u origin main`
공개/비공개 전환	Settings → General → Visibility

README, LICENSE, .gitignore 자동 생성

> `README.md`, `LICENSE`, `.gitignore`의 의미와 설정법

✅ 1. `README.md`: 프로젝트 소개서 📘

📌 역할

- 프로젝트의 **설명**, **사용법**, **설치법**, **기여 방법** 등을 담는 **메인 문서**
- 저장소의 첫 화면에 자동으로 보여짐

📦 주요 섹션 예시

```
# MyProject

## 소개
이 프로젝트는 ___을 위한 ___입니다.

## 설치 방법
```

npm install

```
## 사용법
```

npm run start

```
## 기여
이슈를 먼저 확인하고, Pull Request를 보내주세요.

## 라이선스
MIT
```

📍 GitHub 설정 위치

> 저장소 생성 시
> ✅ `✓ Initialize this repository with a README` 체크

✅ 2. `.gitignore`: Git 무시 목록 설정 🛑

📌 역할

- Git이 **추적하지 않아야 할 파일/폴더**를 명시
- 예: 로그 파일, 빌드 산출물, `.env`, `node_modules/`, `.DS_Store` 등

📦 예시 (.gitignore for Node.js)

```
node_modules/
.env
npm-debug.log
dist/
```

📍 GitHub 설정 위치

> 저장소 생성 시
> ✅ `.gitignore template` 드롭다운 → 언어/환경 선택 (`Python`, `Java`, `Node`, `Unity`, 등)

> GitHub는 다양한 언어별 템플릿을 제공함

✅ 3. `LICENSE`: 오픈소스 라이선스 설정 📜

📌 역할

- 프로젝트의 **법적 사용 조건**을 명시
- 오픈소스 배포 시 필수 (MIT, Apache 2.0, GPL 등)

📦 주요 예시

라이선스	설명
MIT	자유도 높음, 저작권 명시만 하면 됨
Apache 2.0	특허 보호 조항 포함
GPL	파생물도 GPL로 공개해야 함 (강제적 공유)
BSD	MIT와 유사, 매우 자유로움
Unlicense	퍼블릭 도메인 선언

📍 GitHub 설정 위치

> 저장소 생성 시
> ✅ `Add a license` → 원하는 라이선스 선택

✅ 4. 자동 생성 후의 디렉토리 구조 예시

```
my-repo/
├── .gitignore
├── LICENSE
├── README.md
```

→ 이 세 파일은 GitHub 웹에서 생성되며, 첫 커밋에 포함됨
→ 이후 로컬로 클론하면 같이 내려받음

✅ 5. 수동으로 추가도 가능

터미널에서 직접 생성

```
echo "# MyProject" > README.md
echo "node_modules/" > .gitignore
touch LICENSE
```

GitHub 웹 UI에서 나중에 추가

- 저장소 진입 → [Add file] → [Create new file]
  → 파일명: `README.md`, `.gitignore`, `LICENSE`

✅ 6. 권장 사용 팁

파일	권장사항
`README.md`	프로젝트 목적, 사용법, 설치법, 개발환경 명시
`.gitignore`	언어에 맞는 템플릿에서 시작 후 필요에 따라 수정
`LICENSE`	오픈소스일 경우 반드시 포함 (MIT 권장)

✅ 마무리 요약표

파일	역할	생성 위치
`README.md`	설명서	저장소 생성 시 체크 or Add file
`.gitignore`	무시 대상	언어 선택 시 자동 템플릿
`LICENSE`	사용 조건 명시	라이선스 선택 시 자동 생성

Issues, Discussions, Projects (칸반보드)

✅ 1. Issues (이슈)

📌 개념

- 버그 리포트, 기능 요청, 질문 등 **작업 단위와 문제 제기를 위한 티켓**
- **To-Do 리스트**처럼 기능별로 관리
- 코드 변경(PR)과 직접 연결 가능

🛠️ 주요 기능

항목	설명
제목 + 본문	마크다운 작성 가능
Labels	`bug`, `enhancement`, `help wanted`, 등
Assignee	담당자 지정
Milestone	릴리즈나 기간 단위 목표 지정
댓글	토론, 수정 논의
자동 닫힘	PR에서 `Fixes #12` 작성 시 자동 Close

✅ 추천 사용법

- 작업 목록으로 관리
- 팀별 책임 배분
- 커밋/PR과 연결해서 추적 가능

✅ 2. Discussions (토론)

📌 개념

- 자유로운 **의견 교환, 질문, 투표** 등을 위한 게시판
- 공식 이슈로 다루기 전의 아이디어 제안
- Stack Overflow + 커뮤니티 게시판 느낌

🛠️ 주요 기능

항목	설명
카테고리	질문, 아이디어, 일반 토론 등
답변 채택	Stack Overflow처럼 '채택' 가능
좋아요/이모지	👍 ❤️ 😄 등으로 피드백
링크 공유	이슈, PR과 연동 가능
모더레이션	관리자 승인/잠금 기능

✅ 추천 사용법

- 이슈가 되기 전 사전 논의
- 외부 기여자 Q&A
- 피드백 받기

✅ 3. Projects (프로젝트 보드/칸반)

📌 개념

- **칸반(Kanban) 방식**의 프로젝트 관리 보드
- 이슈, PR, 작업 등을 **컬럼형 보드**로 시각화

🛠️ 주요 기능 (Projects v2 기준)

항목	설명
보드	Todo / In Progress / Done 컬럼
이슈 연결	Drag & Drop으로 상태 이동 가능
필터	Label, Assignee, 상태 등 필터링
자동화	PR 병합 시 Done으로 이동 등
타임라인	일자 기반 뷰도 가능
커스텀 필드	우선순위, 예상시간 등 직접 필드 추가

✅ 실전 예시

칼럼	설명
📝 Todo	해야 할 기능 (이슈 연결)
⚙️ In Progress	작업 중인 기능 (Assignee 있음)
✅ Done	완료된 기능 (PR 병합됨)

→ 이슈/PR을 끌어다 놓기만 해도 프로젝트 관리가 가능함!

✅ 4. 비교 요약표

항목	Issues	Discussions	Projects
목적	버그/작업 추적	질문/아이디어 토론	전체 작업 흐름 시각화
대상	담당자 중심	전체 사용자	관리자/팀 리더
연결	PR/커밋과 직접 연결	느슨한 연결	이슈/PR 드래그로 연결
권장 상황	명확한 작업 항목	자유 토론 필요할 때	팀 프로젝트 운영
시각화	라벨, 목록	댓글 중심	칸반 보드 UI

✅ 실전 흐름 예시

1. 사용자가 Discussion에서 기능 요청 아이디어 제시
2. 팀원이 좋다고 판단 → Issue로 전환
3. Issue를 Project 보드의 "To Do"에 추가
4. 개발자가 PR 작성 → "In Progress"로 이동
5. PR 머지되면 → "Done"으로 자동 이동
6. Issue는 자동으로 Closed

✅ 활성화 위치

1. 저장소 → `Settings` → `Features` 항목에서
   `✓ Issues`, `✓ Discussions`, `✓ Projects` 전부 체크
2. 프로젝트 보드는 `Projects` 탭
   → `New project` 클릭
   → `Board`, `Table`, `Roadmap` 중 선택 가능

Pull Request 생성 및 리뷰

✅ 1. Pull Request(Pull 요청)란?

> **Pull Request (PR)** 는 브랜치에서 작업한 내용을
> **메인 브랜치(또는 다른 브랜치)**에 병합해달라고 요청하는 GitHub 기능이다.

💡 "변경사항을 병합해주세요!" 라는 공식적 요청이자,
리뷰 → 승인 → 병합(Merge) 과정을 포함하는 협업 절차임.

✅ 2. 기본 흐름 요약

1. 🔧 **새 브랜치에서 작업**
2. ✅ **커밋 및 푸시**
3. 🔀 **PR 생성**
4. 👀 **리뷰 및 토론**
5. ✅ **승인 → Merge → 브랜치 삭제**

✅ 3. PR 생성 방법

▶️ 사전 준비: 브랜치에서 작업 후 푸시

```
git checkout -b feature/login
# 파일 수정...
git add .
git commit -m "feat: 로그인 기능 추가"
```

```
git push origin feature/login
```

▶️ GitHub에서 PR 생성

1. GitHub 접속 → 저장소로 이동
2. "Compare & pull request" 버튼 클릭 (푸시 직후 보임)
   또는
   `Pull requests` 탭 → [New pull request]
3. **Base**: 병합 대상 (보통 `main`)
   Compare: 작업 브랜치 (예: `feature/login`)
4. 제목(`Title`)과 본문(`Description`) 작성
   - `feat: 로그인 기능 추가`
   - 변경 목적, 영향 받는 모듈, 리뷰 포인트 등
5. 옵션 설정:
   - 📄 Reviewer: 리뷰어 지정
   - 📌 Assignee: 책임자
   - 🏷️ Labels: `bug`, `feature`, `WIP` 등
   - 🧩 Projects, Milestone 연동 가능
6. [Create Pull Request] 클릭

✅ 4. PR 리뷰 및 토론

- **리뷰어**는 GitHub UI 상에서
  변경된 코드 비교(diff)를 확인하고 **코멘트** 작성 가능
- `Start a review`로 여러 코멘트 묶어서 일괄 제출 가능
- `Approve`, `Request changes`, `Comment` 세 가지 버튼 중 하나로 응답

✅ 5. 병합(Merge)

- 리뷰가 완료되면 PR 페이지에서 [Merge pull request] 클릭

병합 옵션

옵션	설명
Create a merge commit	기본, 커밋 히스토리 보존
Squash and merge	커밋들을 하나로 합침
Rebase and merge	직선 히스토리 유지 (중급자용)

→ 병합 후 **브랜치 삭제 버튼**도 제공됨 (`Delete branch`)

✅ 6. 커맨드라인 기반 PR 병합도 가능

```
git checkout main
git pull origin main
git merge feature/login
git push origin main
```

⚠️ 단, 리뷰/토론 기능은 GitHub 웹에서만 가능

✅ 7. 자동 닫힘 기능 (Fixes)

커밋 메시지나 PR 본문에 다음 문구를 넣으면:

```
Fixes #123
Closes #45
```

→ 해당 번호의 이슈가 PR 병합과 함께 **자동으로 닫힘**

✅ 8. 팀 협업 시 권장 전략

전략	설명
Feature 브랜치 전략	각 기능마다 브랜치 생성 (`feature/signup`)
PR마다 리뷰 필수	리뷰 없이 merge 방지
코드 컨벤션 체크	ESLint, Prettier + GitHub Actions
제목 컨벤션	`feat:` , `fix:` , `refactor:` 등 (Conventional Commits)

✅ 9. 실전 예시 흐름

```
main
|
├── feature/login
|   └── PR → 리뷰 → Merge
|
├── feature/signup
|   └── PR → 리뷰 → Merge
```

✅ 10. PR 리뷰 알림 및 참여

- GitHub은 리뷰 요청 시 **알림** 발송 (웹/메일/앱)
- PR 코멘트는 **Slack/Discord 연동**으로도 실시간 확인 가능
- ✅ Approve 없이 merge 불가하도록 **Branch protection rules** 설정도 가능

✅ 마무리 요약

단계	설명
새 브랜치 생성	`git checkout -b feature/xxx`
커밋 및 푸시	`git push origin feature/xxx`
PR 생성	GitHub 웹에서 Create Pull Request
리뷰	코드 비교, 코멘트 작성, 승인
병합	Merge pull request → 브랜치 삭제

Fork와 Upstream Pull 구조

✅ 1. 개념 요약

용어	설명
Fork	타인의 저장소를 내 계정으로 복제한 **개인 사본**
Upstream	원본 저장소 (기여하려는 대상)
Origin	내 Fork 저장소
PR (Pull Request)	Origin → Upstream으로 변경사항 요청

✅ 2. Fork 구조 이해

```
[upstream] github.com/org/project.git   ← 원본 저장소
      ↑
   Pull Request 요청
      ↑
[origin]   github.com/you/project.git   ← 내 Fork
      ↑
  작업 및 커밋
      ↑
[local]    내 PC 로컬 저장소
```

✅ 3. Fork 기반 기여 흐름 (Step-by-Step)

🔹 Step 1. Fork 저장소 만들기

1. 원본 저장소 접속 (`github.com/org/project`)
2. 우측 상단 **[Fork]** 버튼 클릭
3. 내 계정에 복제된 저장소 생성됨 (`github.com/me/project`)

🔹 Step 2. 로컬에 클론

```
git clone git@github.com:me/project.git
cd project
```

🔹 Step 3. 원본 저장소(upstream) 등록

```
git remote add upstream https://github.com/original-owner/project.git
```

```
git remote -v
# origin    → 내 포크 저장소
# upstream → 원본 저장소
```

🔹 Step 4. 새 브랜치에서 작업

```
git checkout -b feature/some-feature
# 파일 수정
git add .
git commit -m "feat: add some feature"
git push origin feature/some-feature
```

🔹 Step 5. GitHub에서 Pull Request 생성

- 내 Fork 저장소로 이동 → `feature/some-feature` 브랜치 선택
- "Compare & pull request" 클릭
- Base: `upstream/main`, Compare: `origin/feature/some-feature`
- 제목/본문 작성 → [Create Pull Request]

🔹 Step 6. 리뷰 → 머지 대기

- PR은 **Upstream의 관리자(maintainer)** 가 리뷰하고 머지함
- 내 저장소에서는 자동 반영되지 않음

✅ 4. Upstream 변경 사항 반영 (동기화)

> 시간이 지나면서 원본 저장소(upstream)에 변경이 생기면,
> 내 Fork와 로컬 저장소에 동기화해줘야 해.

```
git checkout main
git fetch upstream
git merge upstream/main
# 또는 git rebase upstream/main
git push origin main
```

✅ 5. 요약 명령어 정리

작업	명령어
Fork 저장소 클론	`git clone`
원본 저장소 등록	`git remote add upstream <url>`
브랜치 생성	`git checkout -b feature/foo`
변경 푸시	`git push origin feature/foo`
PR 생성	GitHub에서 생성 (origin → upstream)
Upstream 동기화	`git fetch upstream` + `git merge` 또는 `rebase`

✅ 추천 규칙 및 팁

항목	내용
`main` 브랜치는 손대지 말고 upstream으로만 갱신	
작업은 항상 **새로운 브랜치**에서	
커밋 메시지는 명확하게 (`feat:`, `fix:`, `docs:` 등)	
PR 보내기 전에 upstream과 **충돌 없는지 확인**	
하나의 PR에는 **하나의 목적만 포함**하기	

✅ 실제 예시 흐름

```
upstream/main
    ↑
    PR (origin/feature/signup)

you/project.git
├── main              ← 원본 동기화용
├── feature/signup    ← 작업 브랜치
```

✅ GitHub UI 상의 Pull Request 경로

- PR 만들 때는 항상 **내 저장소(origin)**에서 → **원본 저장소(upstream)**으로
- 주소 예시:

```
github.com/original-owner/project/pull/new/feature/signup
```

GitHub Actions 간단 소개

✅ 1. GitHub Actions란?

> GitHub 저장소에서 일어나는 **이벤트(push, PR 등)** 에 따라
> 자동으로 **빌드, 테스트, 배포, 알림** 등의 작업을 실행하는
> **CI/CD(지속적 통합/지속적 배포) 플랫폼**

🧠 핵심 개념

- **Workflow**: 자동화 정의 파일 (`.yml`)
- **Event**: 트리거 발생 조건 (예: `push`, `pull_request`)
- **Job**: 병렬 혹은 순차적으로 실행되는 작업 묶음
- **Step**: Job 안의 개별 명령어 단계
- **Runner**: 실제 명령어를 실행하는 서버 (GitHub 제공 또는 Self-hosted)

✅ 2. 어떤 자동화를 할 수 있나?

사용 목적	예시
테스트 자동화	`npm test`, `pytest`, `maven test` 등
코드 검사	ESLint, Prettier, SonarQube
빌드	Java → `.jar`, React → `npm run build`
배포	S3 업로드, Docker 이미지 Push, Firebase 배포
PR 자동 검사	PR마다 테스트 후 통과 여부 표시

사용 목적	예시
정적 사이트 생성	Jekyll, Hugo, Docusaurus 자동 배포
알림	Slack, Discord, Email 연동

✅ 3. 기본 구조 (YAML 예시)

```yaml
# .github/workflows/ci.yml
name: CI 테스트

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout 코드
        uses: actions/checkout@v4

      - name: Node.js 설정
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: 의존성 설치
        run: npm install

      - name: 테스트 실행
        run: npm test
```

✅ 4. 실행 위치

- `.github/workflows/*.yml` 디렉토리에 워크플로우 파일 저장
- 저장소에 커밋되면 자동으로 감지

✅ 5. Actions 마켓플레이스

- GitHub는 **수천 개의 커스텀 액션**을 제공
- 예: `actions/checkout`, `setup-node`, `docker/login-action`, `firebase-actions`, 등
- 마켓 주소: https://github.com/marketplace/actions

✅ 6. 실행 결과 보기

- 저장소 → **Actions** 탭에서 확인
- 각 Job, Step의 로그 확인 가능
- 실패한 Step도 어디서 실패했는지 상세 표시됨

✅ 7. 실행 비용

플랜	무료 사용량	비고
퍼블릭 저장소	무제한 무료	Open source 지원
프라이빗 저장소	월 2,000분 (무료 계정 기준)	초과 시 과금 발생

✅ 8. 실전 예: 배포 자동화

```yaml
on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Build
        run: npm run build
      - name: Deploy to Firebase
        uses: w9jds/firebase-action@v13
        with:
          args: deploy --only hosting
        env:
          FIREBASE_TOKEN: ${{ secrets.FIREBASE_TOKEN }}
```

✅ 정리 요약

요소	설명
`on:`	트리거 이벤트 설정 (`push`, `pull_request`)
`jobs:`	여러 작업 정의 (테스트, 빌드, 배포 등)
`steps:`	실제 명령어 단계
`uses:`	외부 액션 사용 (`actions/checkout` 등)

요소	설명
`run:`	명령어 직접 실행 (`npm install`)
`secrets:`	API 키 등 민감 정보 안전 저장 및 사용

GitHub Actions는 **별도 서버 없이**,
GitHub 안에서 바로 자동화를 시작할 수 있다는 점에서
✅ 배우기 쉽고
✅ 확장성이 높고
✅ 오픈소스에 특히 강력한 CI/CD 솔루션이다.