18. 실전 예제 및 프로젝트

React 프로젝트 Git 버전 관리 적용

1️⃣ Git 초기화

React 프로젝트 생성 후 (예: `npx create-react-app my-app`)

```
cd my-app
git init
```

→ `.git/` 디렉토리 생성됨 → Git 시작

2️⃣ .gitignore 설정

React 프로젝트에는 기본적으로 `node_modules/`, `build/` 등 **버전 관리에 포함하면 안 되는 파일/디렉토리**가 있음.
`create-react-app` 으로 생성하면 `.gitignore` 기본 포함됨 → 아래 항목이 기본으로 들어가 있음:

```
node_modules/
build/
.env
.DS_Store
npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

→ 필요 시 **환경 설정 파일 등 추가** 가능:

```
.env.local
.env.development.local
.env.production.local
```

3️⃣ 초기 커밋

```
git add .
git commit -m "Initial commit: React project setup"
```

4️⃣ 원격 저장소 연결 (GitHub 예시)

GitHub Repo 생성 후

```
git remote add origin https://github.com/username/repo-name.git
git branch -M main
git push -u origin main
```

5️⃣ 주요 폴더 관리 전략

폴더/파일	Git 관리 여부
src/	포함 (소스 코드)
public/	포함 (정적 리소스)
package.json, package-lock.json	포함 (의존성 관리)
.gitignore	포함 (Git 설정)
node_modules/	제외 (로컬 설치)
build/	제외 (빌드 결과물 → 배포 시만 필요)
.env / .env.local	필요 시 제외 (비밀 정보 포함 가능성)

6️⃣ 커밋 메시지 관리 팁

- 팀 개발 시 → **Conventional Commits** 규칙 적용 추천:

```
feat: add new feature X
fix: fix bug Y
docs: update README
refactor: simplify component logic
style: adjust component CSS
test: add unit test for Z
```

- 자동 릴리즈(semver), changelog 자동 생성 등과도 잘 연동됨

정리

단계	작업
1️⃣	Git init
2️⃣	.gitignore 확인/수정
3️⃣	초기 커밋

단계	작업
4️⃣	원격 저장소 연결 (GitHub 등)
5️⃣	src/, public/, package.json 등만 관리, build/node_modules 제외
6️⃣	커밋 메시지 규칙 적용 추천

결론

- React 프로젝트는 Git 기반으로 소스 코드, 설정 파일만 버전 관리 → 빌드 결과물은 제외
- GitHub Pages 배포 시 → gh-pages 브랜치 별도로 관리
- 팀 협업 시 Conventional Commits 적용 → PR, 리뷰, changelog 관리까지 체계화 가능

협업용 Repository 템플릿 설계

1️⃣ Repository 기본 구조

```
repo-root/
├── .github/                  → GitHub Actions, Issue/PR 템플릿 등
│   ├── workflows/            → CI/CD Workflow 정의
│   ├── ISSUE_TEMPLATE/       → Issue 템플릿
│   ├── PULL_REQUEST_TEMPLATE.md
├── src/                      → 소스 코드
├── public/                   → 정적 리소스 (React 기준 등)
├── tests/                    → 테스트 코드
├── docs/                     → 문서 (README 보완용 or GitHub Pages용)
├── scripts/                  → 빌드/배포 등 유틸 스크립트
├── .gitignore
├── .editorconfig             → 에디터 코드 스타일 통일
├── .prettierrc / .eslintrc   → 린트/포맷 설정
├── package.json / requirements.txt / build.gradle 등 → 의존성/빌드 정의
├── README.md
├── LICENSE
└── CHANGELOG.md              → 변경 이력
```

2️⃣ Repository 구성 요소

📄 README.md

- 프로젝트 개요
- 개발 환경 설정 방법
- 실행 방법
- 배포 방법
- 협업 가이드 (브랜치 전략 등)

LICENSE

- 오픈소스일 경우 → 명확한 라이선스 명시 (MIT, Apache-2.0 등)

CHANGELOG.md

- 릴리즈 별 변경 이력 기록
- Conventional Commits 기반 자동 생성 추천

---

3 GitHub 템플릿

.github/ISSUE_TEMPLATE/

- 버그 리포트 템플릿
- 기능 요청 템플릿
- 기타 커스텀 템플릿

.github/PULL_REQUEST_TEMPLATE.md

- 변경 목적 / 관련 Issue
- 변경 내용 요약
- 테스트 여부
- 리뷰어 체크리스트

---

4 CI/CD 구성 (.github/workflows/)

예시:

- Lint + Test → PR 발생 시 자동 실행
- main → Deploy → 프로덕션 자동 배포
- develop → Dev 환경 배포 자동화 가능

```yaml
name: CI

on:
  pull_request:
    branches: [ main, develop ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install dependencies
        run: npm ci
      - name: Run lint
        run: npm run lint
      - name: Run tests
```

```
17          run: npm test
```

5 코드 품질 관리

린트 & 포맷

- ESLint / Prettier / Stylelint 등 설정
- Husky + lint-staged로 pre-commit hook 구성 추천

```
npm install --save-dev husky lint-staged prettier eslint
npx husky install
npx husky add .husky/pre-commit "npx lint-staged"
```

.editorconfig

- 에디터 간 코드 스타일 차이 최소화

```
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 2
end_of_line = lf
insert_final_newline = true
```

6 브랜치 전략 (권장 예시)

- main → 배포 가능 안정 버전
- develop → 다음 배포 준비용
- feature/xxx → 기능 개발
- fix/xxx → 버그 수정
- release/x.x.x → 릴리즈 준비
- hotfix/x.x.x → 긴급 수정

정리

구성 요소	목적
.github/workflows	CI/CD 자동화
.github/ISSUE_TEMPLATE	일관된 Issue 등록
.github/PULL_REQUEST_TEMPLATE.md	통일된 PR 리뷰 문화
src/, public/, tests/, docs/	코드, 리소스, 테스트, 문서

구성 요소	목적
.gitignore, .editorconfig	불필요한 파일 제외, 코드 스타일 통일
README.md, LICENSE, CHANGELOG.md	문서화, 라이선스, 변경 이력

결론

- 협업용 Repo는 **코드만이 아니라 작업 과정 전체(작성/테스트/리뷰/배포)까지 흐름을 구조화**하는 것이 중요
- GitHub Actions, Issue/PR Template, Lint/Format/Hook 구성 → **팀 협업 품질 급격히 향상**
- 잘 설계된 Repo 템플릿은 신규 팀원 온보딩 속도까지 높여줌

블로그 템플릿 배포 with GitHub Pages

기본 흐름

1️⃣ 블로그 템플릿 선택 → 프로젝트 초기화
2️⃣ 템플릿 수정/커스터마이징
3️⃣ GitHub Repository에 Push
4️⃣ GitHub Pages 설정 → 배포
5️⃣ 커스텀 도메인 연결(Optional)

1️⃣ 블로그 템플릿 선택

Jekyll 기반

- Jekyll Themes → https://jekyllthemes.io
- GitHub Pages 기본 지원 → 별도 빌드 서버 불필요

Hugo 기반

- https://themes.gohugo.io
- GitHub Actions 기반으로 빌드 + GitHub Pages 배포 가능

React 기반

- Gatsby → https://www.gatsbyjs.com/starters
- Next.js → `next export` 사용 시 정적 사이트로 GitHub Pages 배포 가능

2️⃣ 템플릿 적용 예시 (Jekyll)

clone

```
git clone https://github.com/<template-repo> my-blog
cd my-blog
```

수정

- `_config.yml` → 사이트 설정 (제목, 설명, URL 등)
- `_posts/` → Markdown 글 작성
- `_layouts/`, `_includes/` → 디자인 수정 가능

GitHub Repository에 Push

```
git remote add origin https://github.com/username/blog-repo.git
git branch -M main
git push -u origin main
```

---

3 GitHub Pages 설정

Settings → Pages

- Source → `main` 브랜치 → `/ (root)` 또는 `/docs` 디렉토리 선택
- 저장 후 → https://username.github.io/blog-repo 에서 접속 가능

---

4 자동 배포 구성 (Optional)

- Jekyll은 GitHub Pages에서 기본 빌드 지원 → 자동 배포됨
- Gatsby / Hugo / Next.js는 GitHub Actions로 빌드 후 `gh-pages` 브랜치에 배포 추천

Gatsby 예시

```
npm run build
npm run deploy
```

→ `gh-pages` 브랜치로 배포

---

5 커스텀 도메인 연결 (Optional)

- Settings → Pages → Custom Domain → 원하는 도메인 입력
- DNS 설정에서 CNAME 추가 → GitHub Pages 도메인 연결 가능
- GitHub Pages에서 HTTPS 자동 적용 지원

정리

단계	내용
1	템플릿 선택 후 clone or create
2	블로그 내용 작성 / 디자인 수정

단계	내용
3	GitHub Repository Push
4	GitHub Pages 설정 → 자동 배포
5	필요 시 커스텀 도메인 연결

결론

- GitHub Pages + 블로그 템플릿 활용 → **무료 + 빠른 정적 블로그 운영 가능**
- 초보자도 쉽게 배포 가능
- 유지보수 간편, 자동 HTTPS 지원
- 커스텀 도메인까지 연결하면 **브랜드 블로그 수준으로 운영 가능**

GitHub Actions를 활용한 자동 테스트/배포 구성

기본 개념

- **GitHub Actions** = GitHub에서 제공하는 **자동화 플랫폼**
- **워크플로(Workflow)** 를 정의하면 Git 이벤트(예: push, PR 발생 등)가 발생할 때 자동으로 테스트, 빌드, 배포 실행 가능

기본 흐름

1 `.github/workflows/` 폴더에 Workflow 파일(`.yml`) 작성
2 GitHub Events (push, pull_request 등)에 반응
3 GitHub Actions Runner가 Workflow 실행
4 결과 확인 (Actions 탭에서 확인 가능)

구성 예시 ① React 앱 - 테스트 + gh-pages 배포

1 기본 Workflow 구조

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

```

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

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test

      - name: Build project
        run: npm run build

      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./build
```

2️⃣ 동작 설명

✅ `on.push.main` → main 브랜치에 push 시 자동 실행
✅ Checkout → 소스 코드 pull
✅ Node.js 설치 → Node 기반 프로젝트 대응
✅ npm ci → 의존성 설치
✅ npm test → 테스트 실행 (테스트 실패 시 실패 처리됨)
✅ npm run build → 빌드 실행 (React 앱의 build 디렉토리 생성)
✅ peaceiris/actions-gh-pages → GitHub Pages에 자동 배포 (`gh-pages` 브랜치로 push)

구성 예시 ② Node.js + Jest 테스트만 자동 실행

```
name: Run Tests

on:
  pull_request:
    branches:
      - main
      - develop

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
```

```
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test
```

→ PR 발생 시 자동으로 테스트 실행 → 실패 시 Merge 방지 가능

구성 예시 ③ React + Prettier + ESLint + Test + Build + Deploy

```
name: Full CI/CD Pipeline

on:
  push:
    branches: [ main ]

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - uses: actions/setup-node@v4
        with:
          node-version: '18'

      - run: npm ci

      - run: npm run lint
      - run: npm run format:check
      - run: npm test
      - run: npm run build

      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./build
```

주요 구성 요소 설명

구성 요소	용도
`on`	어떤 GitHub Event에 반응할지 정의 (push, pull_request 등)
`jobs`	실행할 Job 정의 (test, build, deploy 등 여러 개 가능)

구성 요소	용도
`runs-on`	어떤 OS 환경에서 실행할지 지정 (보통 ubuntu-latest 사용)
`steps`	단계별 명령 실행 (checkout, node setup, npm ci, npm test 등)
`actions`	미리 구현된 재사용 가능 기능 블록 사용 (`actions/checkout`, `setup-node`, `gh-pages deploy` 등)
`secrets.GITHUB_TOKEN`	GitHub Actions에서 안전하게 사용 가능한 기본 제공 token (repo에 push 가능)

정리

단계	설명
테스트 자동화	PR 또는 push 발생 시 테스트 실행
빌드 자동화	push 시 build 실행하여 결과물 생성
배포 자동화	GitHub Pages (gh-pages 브랜치) 또는 외부 서비스로 자동 배포
코드 품질 체크	Lint/Format 체크 추가 가능
비용	GitHub Actions는 public repo 무료, private repo는 일정 용량 무료 제공

결론

- GitHub Actions를 활용하면 **GitHub 기반 프로젝트에서 매우 쉽게 CI/CD 구성 가능**
- React 앱도 push → 자동 test + build + GitHub Pages 배포 흐름 완전 자동화 가능
- 팀 개발 시 PR 단계에서 자동으로 테스트 실행 → 품질 관리 효과 매우 큼
- 별도 CI 서버 없이 GitHub 자체에서 워크플로 구성 → 설정만 잘하면 매우 강력하게 활용 가능