2026.01.05

생성일	@2026년 1월 5일 오전 8:41
태그	Git JavaScript

목차

▼ 1) Git, GitHub

- Private 환경을 위한 셀프 호스팅 서비스 : GitLab, Gitea
- Git 이란?
  - VCS: Version Control System

❓ What is GitHub Repository, Project, Package, Action?

- Repository: 소스 코드 히스토리를 원자 단위로 관리
  - 코드, 커밋 히스토리, 브랜치, 태그 관리
- Project: 여러 리포지토리에 걸친 작업을 조율하는 계층
  - 프론트repo + 백repo + 안드로이드repo + 인프라repo
  - **칸반 보드/스프린트 보드**: 이슈를 상태별로 시각화 (Todo, In Progress, Done)
  - **크로스 리포지토리 이슈 추적**: 여러 repo의 이슈를 하나의 보드에서 관리
  - **마일스톤 및 로드맵**: 릴리스 계획 수립
  - **자동화**: 이슈 상태 자동 변경 (PR 머지 시 Done으로 이동 등)
- Package
  - 빌드된 아티팩트를 저장하고 배포
- GitHub Action
  - Repository에 속하면서도, Repository/Projects/Packages 전체를 연결하는 자동화 레이어
  - 이벤트 기반 자동화 엔진

▼ 작업 흐름

```
1단계: 개발 완료
  └─ 소스 코드 완성 → Git repo push

2단계: 현재 프로젝트 자동화
  └─ GitHub Actions로 CI/CD 설정
    ├─ 테스트 자동화
    ├─ 배포 자동화 (프로덕션)
    └─ 코드 품질 검사

3단계: 코드 재사용 (선택)
  └─ Packages로 공유 라이브러리 배포
    └─ 다른 프로젝트에서 사용 가능하게

4단계: 프로젝트 관리 (선택)
  └─ Projects로 작업 추적
    └─ 이슈/PR 관리 자동화
```

▼ 예제 시나리오 코드
시나리오: 완전히 새로운 Repository에서 시작

```
# 1단계: Repository 생성
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/yourname/my-app.git
git push -u origin main

# ✅ Repository는 즉시 사용 가능!
```

GitHub Actions 추가

```
# 2단계: Actions YAML 생성
mkdir -p .github/workflows
cat > .github/workflows/test.yml << 'EOF'
name: Test
on: [push]
jobs:
```

```
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm test
EOF

git add .github/workflows/
git commit -m "Add GitHub Actions"
git push

# ✅ Actions 탭에서 즉시 실행 확인 가능!
# 추가 설정 필요 없음
```

Packages 추가

```
# 3단계: Package 설정 (사전 작업 필요)

# 3-1. package.json 수정
cat > package.json << 'EOF'
{
  "name": "@yourname/my-app",
  "version": "1.0.0",
  "publishConfig": {
    "registry": "https://npm.pkg.github.com"
  }
}
EOF

# 3-2. 발행 워크플로우 생성
cat > .github/workflows/publish.yml << 'EOF'
name: Publish
on:
  push:
    tags: ['v*']
jobs:
  publish:
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          registry-url: 'https://npm.pkg.github.com'
      - run: npm ci
      - run: npm publish
        env:
          NODE_AUTH_TOKEN: ${{secrets.GITHUB_TOKEN}}
EOF

git add .
git commit -m "Setup package publishing"
git push

# ❌ 아직 Package는 없음!

# 3-3. 태그 생성 (실제 발행)
```

```
git tag v1.0.0
git push origin v1.0.0

# ✅ 이제 Packages 탭에서 확인 가능!
```

Projects 추가

```
# 4단계: Project 설정 (가장 많은 사전 작업)

# 4-1. GitHub 웹에서 Project 생성 (수동)
# → https://github.com/users/yourname/projects/1

# 4-2. Personal Access Token 생성 (수동)
# Settings → Developer settings → PAT
# → ghp_xxxxxxxxxxxx

# 4-3. Repository Secrets에 토큰 추가 (수동)
# Repository → Settings → Secrets → Actions
# Name: PROJECT_TOKEN
# Value: ghp_xxxxxxxxxxxx

# 4-4. 자동화 워크플로우 생성
cat > .github/workflows/project.yml << 'EOF'
name: Project Automation
on:
  issues:
    types: [opened]
jobs:
  add-to-project:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/add-to-project@v0.5.0
        with:
          project-url: https://github.com/users/yourname/projects/1
          github-token: ${{ secrets.PROJECT_TOKEN }}
EOF

git add .github/workflows/project.yml
git commit -m "Add project automation"
git push

# ✅ 이제 이슈 생성 시 자동으로 Project에 추가됨!
```

실전 예제: 처음부터 끝까지

파일 구조

```
my-todo-app/
├── .github/
│   └── workflows/
│       ├── test.yml          # ← push하면 즉시 작동
│       ├── publish.yml       # ← 태그 생성 시 작동
│       └── project.yml       # ← 사전 설정 필요
├── src/
├── package.json              # ← publishConfig 설정 필요
└── README.md
```

1. test.yml - 즉시 작동

```yaml
name: Test
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm ci
      - run: npm test
```

```
git add .github/workflows/test.yml
git push
# ✅ 즉시 Actions 탭에서 실행 확인!
```

2. publish.yml - 조건부 작동

```yaml
name: Publish
on:
  push:
    tags: ['v*']
jobs:
  publish:
    runs-on: ubuntu-latest
    permissions:
      packages: write
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          registry-url: 'https://npm.pkg.github.com'
      - run: npm ci
      - run: npm publish
        env:
          NODE_AUTH_TOKEN: ${{secrets.GITHUB_TOKEN}}
```

```
# YAML push만으로는 아무 일도 안 일어남
git add .github/workflows/publish.yml
git push
# ❌ 워크플로우 등록은 되지만 실행 안 됨

# 태그 생성해야 실행
git tag v1.0.0
git push origin v1.0.0
# ✅ 워크플로우 실행 → Package 생성!
```

3. project.yml - 사전 설정 필요

```yaml
name: Project Automation
on:
  issues:
    types: [opened]
jobs:
  add-to-project:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/add-to-project@v0.5.0
        with:
```

```
      project-url: https://github.com/users/yourname/projects/1
      github-token: ${{ secrets.PROJECT_TOKEN }}
```

```
# 사전 작업 없이 push하면?
git add .github/workflows/project.yml
git push
# ❌ 워크플로우 등록은 됨

# 이슈 생성해도?
# ❌ 에러 발생!
# Error: PROJECT_TOKEN not found
# Error: Project URL not accessible

# 사전 작업 후:
# 1. GitHub 웹에서 Project 생성
# 2. Personal Access Token 생성
# 3. Repository Secrets에 추가
# ✅ 이제 이슈 생성 시 자동 작동!
```

디버깅: 왜 안 되는지 확인하기

Actions 탭에서 확인

```
GitHub → Repository → Actions 탭

워크플로우가 보이나요?
✅ 예 → YAML 파일이 올바르게 등록됨
❌ 아니오 → .github/workflows/ 경로 확인

워크플로우가 실행되었나요?
✅ 예 → on: 조건 확인
❌ 아니오 → 트리거 조건 미충족

실행했지만 실패했나요?
→ 로그 클릭해서 에러 메시지 확인
  - Token 관련: Secrets 설정 확인
  - 권한 관련: permissions 확인
  - API 관련: URL이나 ID 확인
```

요약

항목	YAML만 push	추가로 필요한 것
GitHub Actions 등록	✅ 자동	없음
Actions 실행	✅ 자동 (조건 만족 시)	없음
Package 생성	❌	package.json 설정 + 실제 발행
Project 사용	❌	웹에서 Project 생성 + Token 발급

핵심:

- **GitHub Actions 워크플로우**: YAML push만으로 등록 완료
- **Packages**: 워크플로우가 실제로 npm publish를 실행해야 생성
- **Projects**: 웹에서 수동으로 먼저 만들어야 함

YAML 파일은 "자동화 스크립트"이지 "리소스 생성 도구"가 아닙니다!

▼ 실전 체크리스트

```
# 1단계: 기본 Setup
☐ Git repo 생성 및 push
```

```
# 2단계: Actions (별도 Token 불필요)
☐ test.yml 작성 및 push
☐ deploy.yml 작성 및 push
☐ code-quality.yml 작성 및 push

# 3단계: Packages (별도 Token 불필요)
☐ package.json에 publishConfig 추가
☐ publish-package.yml 작성 (permissions 설정)
☐ git tag 생성하여 발행

# 4단계: Projects (Personal Access Token 필요)
☐ GitHub 웹에서 Project 생성
☐ Personal Access Token 생성
☐ Repository Secrets에 추가
☐ project.yml 작성 및 push
```

▼ 현실적인 구조

```
"Todo 앱" 프로젝트
│
├─ Repository 1: frontend
│  ├─ .github/workflows/
│  │  ├─ test.yml       (frontend 테스트)
│  │  └─ deploy.yml     (Vercel 배포)
│  └─ GitHub Actions → Vercel
│
├─ Repository 2: backend
│  ├─ .github/workflows/
│  │  ├─ test.yml       (backend 테스트)
│  │  └─ deploy.yml     (AWS 배포)
│  └─ GitHub Actions → AWS
│
├─ Repository 3: mobile
│  ├─ .github/workflows/
│  │  ├─ test.yml       (mobile 테스트)
│  │  └─ deploy.yml     (App Store 배포)
│  └─ GitHub Actions → App Store
│
├─ Repository 4: shared-types
│  ├─ .github/workflows/
│  │  └─ publish.yml    (Package 발행)
│  └─ GitHub Packages → npm
│
└─ GitHub Projects (하나의 보드)
   ├─ 📋 Backlog
   │  ├─ [frontend] #12: 새 UI 디자인
   │  ├─ [backend] #34: API 최적화
   │  └─ [mobile] #56: 푸시 알림
   │
   ├─ 🔨 In Progress
   │  ├─ [frontend] #15: 로그인 화면
   │  └─ [backend] #37: 인증 로직
   │
   └─ ✅ Done
      ├─ [frontend] #10: 메인 화면
      └─ [mobile] #52: 스플래시 화면
```

▼ 수동 작업과 Actions 작업 비교

기능	수동 작업 시	GitHub 활용 시
테스트	로컬에서 매번 실행	자동 실행, 실패 시 알림
배포	30분-1시간 수동 작업	5분 자동 배포
코드 리뷰	체크리스트 수동 확인	자동 품질 검사
릴리스 노트	수동 작성 (30분)	자동 생성 (0분)
보안 업데이트	수동 확인	자동 PR 생성
팀 협업	메신저/회의	Project 보드로 시각화

- 협업에선 브랜치 및 커밋의 네이밍룰(Naming role) 필수

▼ 2) Jekyll & GitHub Pages

- Jekyll: Ruby 언어를 사용하여 md 파일을 html 으로 변환시켜주는 블로그 생성 라이브러리?
- GitHub Pages: 개인 도메인으로 정적 웹 사이트 퍼블리싱, 호스팅 가능

▼ 3) JavaScript

- 웹 브라우저에서 HTML, CSS가 못하는 로직을 구현하기 위해 만든 언어
- 변수
  - let
  - var
  - const
- 자료형
  - Primitive Type 원시형
    - string
    - number
    - bigint
    - boolean
    - undefined: 자료형이 정의되지 않음 (정의되지 않음)
    - symbol
    - null: 데이터가 비어있음을 명시적으로 표시 (의도된 표현)
  - Reference Type 참조형
    - Object
    - Array
    - Function
  - 자바스크립트는 '동적 자료형': 프로그래밍에서 변수의 데이터 타입을 프로그램 실행 시점(런타임)에 결정하고, 필요에 따라 자유롭게 변경할 수 있는 방식
    - **실행 시 타입 결정:** 변수에 어떤 종류의 데이터(숫자, 문자열 등)가 들어올지 컴파일 시점이 아닌, 프로그램이 실행될 때 결정됩니다.
    - **자유로운 타입 변경:** 하나의 변수에 숫자, 문자열 등 다른 타입의 데이터를 연속해서 할당해도 오류가 발생하지 않고 타입이 바뀝니다.
    - **유연성:** 간단한 스크립트나 빠른 프로토타입 개발에 유리하며, 코드를 유연하게 작성할 수 있습니다.
    - **단점 (런타임 오류 위험):** 타입이 실행 시점에 결정되므로, 개발자가 의도치 않게 잘못된 타입의 데이터를 사용하면 프로그램 실행 중에 오류가 발생할 수 있습니다.

    → 반대는 정적 자료형
- 연산자
  - 산술 연산자: + - / * %
  - 문자열 연산자: "hello" + "world"
  - 비교 연산자: >= <= < >
  - 전위 증가/감소 연산자, 후위 증가/감소 연산자: ++ --

    - 할당 연산자: = += *= -= /= %=
    - 논리 연산자: && || !
- 객체 (Object)
    - 키(key)-값(value) 묶음의 프로퍼티(property) 구조
    - const로 선언하면 주소값을 바꿀 수 없지만 속성은 수정 가능
    - 함수(메서드)도 속성 값으로 가질 수 있음
- 배열 (Array)
    - 여러 값을 index 순으로 저장하는 자료형
- 조건문
    - if, if-else

```
if (condition1) {
// code;
} else if (condition2) {
// code;
} else {
// code;
}
```

    - 삼항 연산자

```
condition ? true_code : false_code ;
```

    - switch

```
switch (condition) {
   case value1:
      //code
      break;
   case value2:
      //code
      break;
   default:
      //code
}
```

- 반복문
    - for

```
for (initialization; condition; increment) {
// code;
}

for (let v of array) { ... }

for (let v in object) { ... }
```

    - while, do-while

```
while (condition) {
// code;
}

do {
```

```
// code;
} while (condition);
```

- 함수
  - 함수도 '값'이다. 변수에 할당 가능

```
function greet(name) {
 return "Hello, " + name + "!";
}
console.log(greet("John")); // "Hello, John!"
```

```
const greet = (name) => {
 return "Hello, " + name + "!";
};
```

- 예외처리
  -