



Github 사용

 Github 사용 방법을 정리하는 페이지 입니다.

GIT REPOSITORY URL

 [GitHub - addinedu-ros-10th/iot-repo-4: IoT 프로젝트 4조 저장소. 취약계층에 대한 통합지원](#)

모노레포(Monorepo) 기반 개발 형상 관리 지침

이 문서는 독거노인 등 1인 가구 취약계층 통합 돌봄 서비스 개발을 위한 모노레포(Monorepo) 환경에서 프로젝트의 효율적인 형상 관리를 위해 모든 개발자가 준수해야 할 지침을 제공합니다.

1. 모노레포 구조 및 프로젝트 이해

우리가 사용하는 Git 리포지토리는 다음과 같은 개별 프로젝트들을 포함하는 모노레포 구조로 되어 있습니다.

- `apps/user-app` : 사용자 모바일/웹 애플리케이션
- `apps/pyqt-admin-service` : PyQt 기반 관제 서비스
- `services/was-server` : WAS (Web Application Server)
- `services/que-alert-server` : QUE & ALERT 서버
- `iot-device` : IOT 디바이스 관련 코드 및 펌웨어

2. 기본 Git 워크플로

모든 작업은 Git을 통해 이루어지며, 아래 워크플로를 반드시 따르십시오.

1. 리포지토리 복제(Clone)

```
1 Bash
```

```
1 git clone [리포지토리 URL]
```

```
2
```

최초 한 번만 실행합니다. 리포지토리 전체를 로컬에 복제합니다.

2. 브랜치 생성 및 전환

```
1 Bash
```

```
1 git checkout -b feature/your-feature-name
```

```
2
```

새로운 기능 개발이나 버그 수정 시 **main** 브랜치에서 새로운 브랜치를 생성하여 작업합니다. 브랜치명은 명확하고 설명적이어야 합니다.

3. 작업 및 변경 사항 추가

자신이 담당하는 프로젝트 폴더 내에서만 코드를 수정합니다. 작업이 완료되면, **자신의 프로젝트 폴더의 변경 사항만** Staging Area에 추가합니다.

예시 1: IOT Device(Arduino) 펌웨어 업데이트

- **시나리오:** 아두이노 센서 데이터 전송 주기를 변경하는 펌웨어 코드를 수정합니다.
- **커맨드:**

```
1 Bash
```

```
1 # iot-device/arduino/ 폴더 내 코드 수정 (예: `sensor_read.ino`)
```

```
2 # 변경 사항을 스테이징
```

```
3 git add iot-device/arduino/
```

```
4
```

예시 2: PyQt 관제 서비스 기능 추가

- **시나리오:** 관제 서비스에 IOT 디바이스의 배터리 잔량을 표시하는 기능을 추가합니다.
- **커맨드:**

```
1 Bash
```

```
1 # apps/pyqt-admin-service/ 폴더 내 코드 수정 (예: `ui_dashboard.py`)
```

```
2 # 변경 사항을 스테이징
```

```
3 git add apps/pyqt-admin-service/
```

```
4
```

4. 커밋(Commit)

변경 사항을 커밋할 때, 커밋 메시지 규칙을 준수해야 합니다.

```
1 Bash
```

```
1 git commit -m "feat: [프로젝트명] 기능 설명"
2 # 예시:
3 git commit -m "feat: [IoT-Device] 센서 데이터 전송 주기 단축"
4 git commit -m "feat: [PyQt-Admin] 디바이스 배터리 잔량 표시 기능 추가"
5
```

◦ 커밋 메시지 규칙: [타입]: [프로젝트명] [설명]

- **feat**: 새로운 기능 추가
- **fix**: 버그 수정
- **docs**: 문서 수정
- **refactor**: 코드 리팩토링
- **style**: 코드 포맷 변경 (세미콜론, 공백 등)
- **chore**: 빌드 설정, 라이브러리 업데이트 등

5. 푸시(Push)

```
1 Bash
```

```
1 git push origin feature/your-feature-name
2
```

로컬에서 작업한 커밋을 원격 리포지토리로 전송합니다.

6. 풀 리퀘스트(Pull Request) 및 병합(Merge)

푸시 후 GitHub에서 **Pull Request** 를 생성합니다. 코드 리뷰를 통해 변경 사항의 안정성을 확보한 후 **main** 브랜치에 병합(Merge)합니다.

3. 주의사항

- **부분적 푸시/풀은 불가**: Git의 기본 동작은 전체 리포지토리 단위로 이루어집니다. 따라서 특정 폴더의 내용만 푸시하거나 풀하는 것은 불가능합니다. **git pull** 시에는 모든 개발자의 변경 사항이 로컬로 반영되므로, 주기적으로 **git pull** 을 실행하여 최신 상태를 유지하십시오.

예시: 다른 개발자의 변경 사항 가져오기

- **시나리오**: 아두이노 개발자가 PyQt 개발자의 변경 사항을 로컬로 가져와야 할 때
- **커맨드**:

```
1 Bash
```

```
1 # 최신 변경 사항을 모두 가져옴  
2 git pull  
3
```

- **결과:** `iot-device` 폴더 외에 `apps/pyqt-admin-service` 등 다른 폴더의 변경 사항도 함께 업데이트됩니다.
- **파일 충돌:** `git pull` 시 다른 개발자의 변경 사항과 충돌이 발생할 수 있습니다. 충돌 해결 (Conflict Resolution)은 본인의 프로젝트 폴더 내에서만 진행하며, 다른 개발자의 코드에는 영향을 주지 않도록 주의합니다.

예시: PyQt 관제 서비스 충돌 해결

- **시나리오:** 관제 서비스 개발자가 작업 중 WAS 서버 개발자가 같은 공통 유틸리티 파일을 수정하여 충돌이 난 경우
- **해결 방법:**
 - `git pull` 실행 시 충돌이 발생하면 Git이 충돌 파일을 표시합니다.
 - 해당 파일(`apps/pyqt-admin-service/common_util.py`)을 열어 충돌 마커(`<<<<<<<` , `=====` , `>>>>>>>`)를 확인하고, 올바른 코드로 수정하여 충돌을 해결합니다.
 - 수정 후 `git add` 및 `git commit` 을 통해 충돌 해결을 완료합니다.
- **독립적 작업:** 각 개발자는 본인이 맡은 프로젝트 내에서만 작업해야 합니다. 다른 프로젝트의 파일이나 폴더를 임의로 수정, 삭제하는 행위는 엄격히 금지됩니다.