페어 프로그래밍 가이드

Pair programming

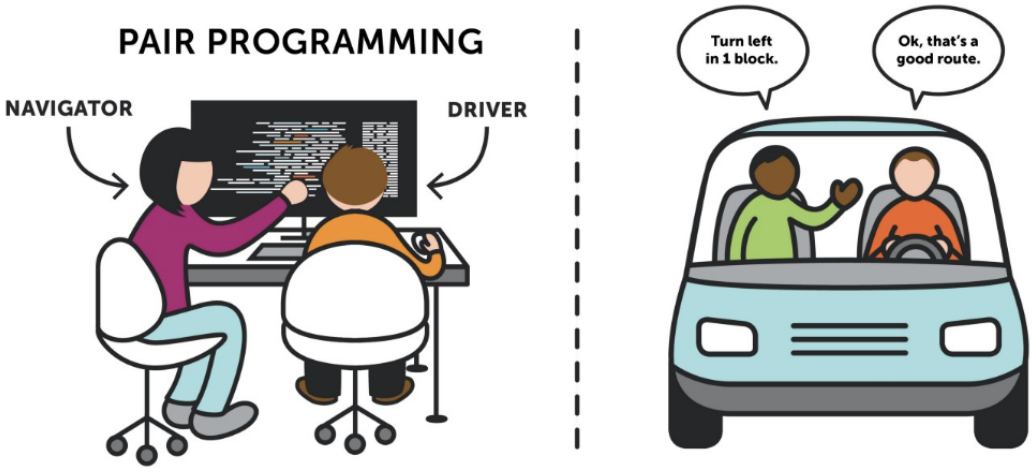
1. Why

> “Pair programming is an important technique for developing higher quality code,
> faster while also reducing risk and spreading knowledge in an organization.”

빠른 개발이 아니라 **서로의 지식 공유 및 함께 고민하는 시간을 갖고,**
소통을 통해 더 좋은 코드를 작성하기 위해 페어 프로그래밍으로 관통PJT를 진행합니다.

---

2. How



2.1 역할

- `Driver` : 코드를 작성하는 사람
- `Navigator` : 구현 하려고 하는 내용을 이야기 하는 사람

2.2 진행

1. 기본적으로 2인 1팀으로 구성합니다.
    a. 홀수 인원인 반의 경우 한 개 팀만 3인 1팀으로 구성합니다.
2. 프로젝트 시작에 앞서 페어와 함께 명세를 분석하는 시간을 가집니다.
    a. 무작정 명세를 따라가기보다는, 함께 프로젝트를 수행할 청사진을 그리는 시간을 갖습니다.
3. 분석이 끝났다면, 각자 역할에 맞춰 시작합니다.
    a. 역할은 방향을 지시하는 `네비게이터(Navigator)`와 코드를 입력하는 `드라이버(Driver)`로 나뉩니다. (3인 1팀일 경우, 2명의 네비게이터와 1명의 드라이버로 구성합니다.)
4. 이때, 네비게이터와 드라이버의 발언 비율은 10:0보다는 `7:3` 정도로 진행하는 것을 권장합니다.
    a. 드라이버는 네비게이터의 지시를 따르되, 의견을 어느 정도 개진하는 것이 소통 측면에서 더 효과적입니다.
5. 역할 교체는 아래의 상황 중 한 가지를 페어끼리 선택하여 진행합니다.
    a. 시간에 따라 교체 (ex. 50분간 진행, 10분간 휴식 후 교체)
    b. 구현할 기능을 분담하고, 맡은 기능을 완성한 이후 교체

c. 서로의 요청에 따라 교체

처음이라면 시간에 따른 교체를 추천합니다

6. `README`는 각자 작성 이후, 마지막에 합쳐서 제출합니다.
   a. 한 파일에 동시에 작성할 경우, 충돌(conflict) 날 수 있습니다.

---

3. git & gitlab 시나리오

📌 아래의 시나리오는 예시입니다.
브랜치를 사용하거나 다른 협업 방식을 택해도 무관합니다.
익숙해진 이후에는, 자유롭게 역할을 분담하여 진행합니다.

☝️ “언제 Commit을 남겨야할까?”

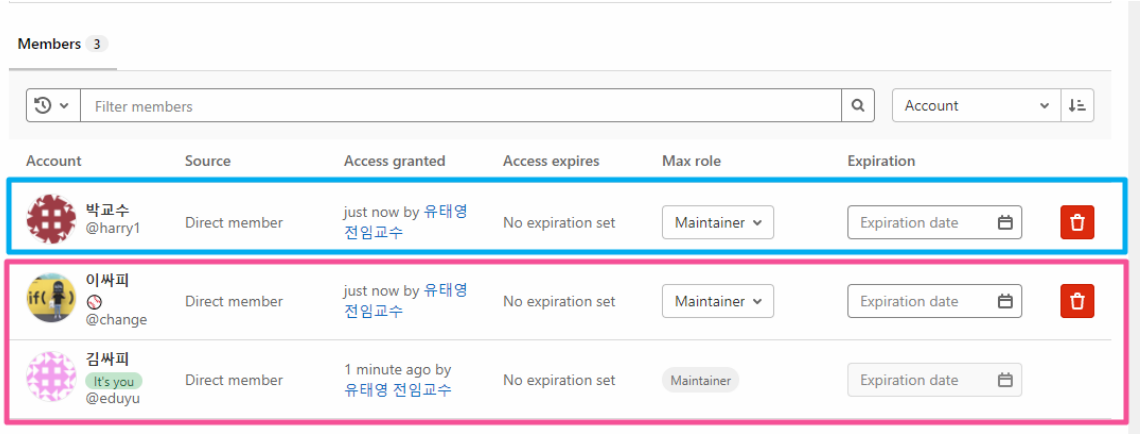
- Commit은 하나의 작은 기능 단위가 완성될 때마다 남깁니다.
- Commit을 하기 전 정상적으로 실행이 되는지 runserver 환경에서 테스트를 거쳐야 합니다.
- 이렇게 작은 단위로 Commit을 해야 할까? 고민하지 말고,
  하나의 기능 단위가 끝날 때마다 Commit을 남기는 습관을 길러봅시다.

- `팀장`의 역할과 `팀원`의 역할로 나누어서 진행합니다.

3.1 프로젝트 설정

팀장

1. 관통PJT gitlab 리포를 `pjt0x` 이름으로 생성
2. 생성 직후 `Members` 탭에서 `Maintainer` 역할로 교수님과 팀원을 등록



예시 이미지

3. 가상환경 설정 및 패키지 설치 & django 프로젝트 생성 & 로컬 저장소 생성

```
# pjt 번호에 맞게 폴더 생성 (ex. pjt06)
$ mkdir pjt번호
$ cd pjt번호

# 가상환경 생성, 활성화 및 패키지 설치
$ python -m venv venv
$ pip install -r requirements.txt

# .gitignore(https://gitignore.io) 및 README 파일 생성
$ touch .gitignore README-김싸피.md README-이싸피.md README.md

# 프로젝트 생성
$ django-admin startproject pjt번호 .

# 원격 저장소 생성 및 push
$ git init
$ git add .
$ git commit -m 'First commit'
$ git remote add origin <REMOTE-URL>
$ git push origin master
```

팀원

1. 원격 저장소 clone

```
$ git clone <REMOTE-URL>
$ cd pjt번호
```

2. 가상환경 설정 및 패키지 설치

```
$ python -m venv venv
$ pip install -r requirements.txt
```

3.2 프로젝트 진행

1. 이제 팀장이 드라이버, 팀원이 네비게이터라고 가정하고 프로젝트를 시작합니다.
   코드 작성은 드라이버인 팀장의 역할이므로, 팀원은 공유된 화면을 통해 팀장에게 지시를 내리며 소통합니다.
2. 역할을 바꿀 차례입니다.
   팀장은 지금까지 작성한 코드를 `add > commit > push` 합니다.

   ```
   $ git add .
   $ git commit -m 'Create 기능 구현'
   $ git push origin master
   ```

3. 팀원은 팀장이 작성한 코드를 `pull` 받고, 역할을 바꿔 개발을 진행합니다.

   ```
   $ git pull origin master
   ```

4. 역할을 바꿀 차례입니다.
   팀원은 지금까지 작성한 코드를 `add > commit > push` 합니다.

   ```
   $ git add .
   $ git commit -m 'Create 기능 구현'
   $ git push origin master
   ```

5. 2~ 4번 과정을 반복합니다.

3.3 프로젝트 종료

1. 각자의 이름으로 작성된 README 파일을 작성합니다.
   a. 기존에 README에 작성하는 내용 **+ 협업 과정에서의 어려움과 느낀 점까지 작성합니다.**

2. Remote repo의 담당자인 팀장이 최종 `README.md`에 취합하고 `add > commit > push`를 진행해 봅시다.
    a. 만약 팀장보다 팀원의 작성이 더 늦다면, 아래 순서는 반대로 진행해도 무관합니다.

```
# 팀원's terminal

# README-B.md 작성 완료
$ git add .
$ git commit -m 'README-B 작성'
$ git push origin master
```

```
# 팀장's terminal

# README-A.md 작성 완료
$ git pull origin master

# README-A.md와 README-B.md의 내용을 README.md 파일에 취합
$ git add .
$ git commit -m 'Finish README'
$ git push origin master
```

3. 프로젝트 종료

---

4. 주의사항

> https://tuple.app/pair-programming-guide/antipatterns

네비게이터

1. 오류들을 너무 빨리 체크하는 것
    - 드라이버가 문법에러와 오타를 수정 할 시간을 주기
    - 너무 작은 에러를 계속 지적하는건 흐름을 끊고, 페어가 타인의 시선을 의식하게 됨
    - 당신의 임무는 틀린 단어를 바로 지적하는게 아니라 큰 그림을 고려하는 것
2. 낮은레벨(low level)의 지시 하기
    - 드라이버에게 제안할 사항이 있으면, 드라이버가 이해할 수 있는 가장 높은 수준의 추상화로 전달할 것

- 코드를 불러주는 것 같다면(심하면 각 키입력까지), 잠깐 멈추고 더 상위 레벨에서 얘기할 수 있는지 볼 것
- 그게 실패하면 아이디어 스케치를 위해 잠시 drive 해달라고 요청할 것

드라이버

1. 너무 빨리 드라이빙 하는 것
    - 에디터에 아주 능숙한 경우, 숙련된 네비게이터 조차 따라가기 힘들만큼 빨라질 수 있음
    - 페어가 따라오고 있다는 확신이 없다면, 당신의 최고 속도로 코드를 조작하지 말 것
    - 하는 일을 입으로 얘기하면서 하면 좋음
2. 체크아웃된 네비게이터를 그대로 두는 것
    - 너무 빨리 하거나, 이해하지 못하는 일을 하면 내비게이터의 주의를 잃기 쉬움
    - 페어의 주의가 흐트러지고 있다면 멈추고 동기화 할 것
    - 나쁜 질문 : "이거 이해하는거 맞죠?"
    - 좋은 질문 : "어떤 부분이 팔로잉하기 어렵나요?"
    - **페어링은 지속적인 쌍방 커뮤니케이션이 이뤄져야 함**
    - 당신이나 당신의 네비게이터가 조용하다면, 멈추고 체크인하게 만들기
3. 휴식하지 않는 것
    - 페어링은 일반적인 프로그래밍 보다 훨씬 더 소모하는 작업
    - 적절한 휴식을 취하는 좋은 방법은 뽀모도로 테크닉을 사용하는 것
    - 시작하기 전에 선호하는 작업/휴식의 길이에 대해 합의해둘 것
4. 경청하지 않고 듣기
    - 듣는 것과 타이핑을 동시에 하는 것은 어려움
    - 네비게이터가 제안을 할 때는, 키보드에서 손을 뗄 것

모두에게

1. 비생산적인 산만함을 허용하는 것
    - 페어링 시작하기 전에 모든 알림을 끌 것(컴퓨터와 폰 모두)

- 세션 동안에는 어떤 알림/문자 메시지도 받으면 안됨
  혹시 놓쳐서 받게 되면, 사과하고 그 다음에는 안 울리게 할 것

2. 역할을 바꾸지 않는 것
   - 드라이빙과 네비게이션은 서로 다른 이유로 에너지가 소모됨
   - 역할을 바꾸면 뇌의 피곤한 부분을 쉬게 하고, 쉬던 부분을 활성화 할 수 있음
   - 드라이버를 교체하는 것은 페어링 세션에 활력을 불어넣을 수 있는 좋은 방법
   - 전환해야할 때 마다 알려주도록 타이머 설정해 두는 것도 좋은 방법
3. 기술이라는 것을 잊어버리는 것
   - 페어프로그래밍은 **배워야 하는 "기술"** 임
   - 처음엔 잘 하기 힘들지만, 꾸준히 연습하면 향상 될 것
   - 어려운 첫 경험 후에 포기하지 말기. 숙련된 개발자라고 좋은 파트너일거라고 단정하지 말기. 연습없이 잘 되기를 기대하지 말기
   - 각 세션후에 페어와 함께 피드백하고 반영하기
     어떻게 하면 더 잘할 수 있었을까 ?