

Copy of Git 규약 / Coding Convention

앞으로 PNMS 개발진행을 저희 Git 그룹계정으로 진행할 예정이며, 아래와 같은 규칙을 갖고 개발하고자 합니다.

📌 목차

- 1. Git 규약
 - a. Branch
 - b. PR (Pull & Request)
 - c. Issue & MileStone
 - d. Commit Message
- 2. Coding Convention
 - a. Git Repo Rule
 - b. Java Convention
 - c. Frontend Convention
- 3. Test case & TDD 규약 (TDD)

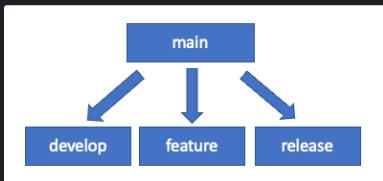
📌 Git 규약

🔗 Git-Flow

- 1. 하나의 기능을 개발한다 → 하나의 브랜치를 생성하여 작업한다.
- 2. 서로 공유하는 브랜치(upstream)는 함부로 변경하지 않는다. (즉, PR을 거쳐 Merge한다.)
- 3. 리뷰어에게 꼭 리뷰를 받는다.

1 Branch

- 기본 branch 구성은 아래 이미지와 같이 진행한다.



main : 배포 완료 된 브랜치

develop : 새로운 기능 개발이 완료 된 브랜치

feature : 새로운 기능을 개발하는 브랜치

release : 배포를 준비 중인 브랜치

- 새로운 기능개발은 feat/(기능명) 브랜치를 생성하여 PR한다.
feature 단어를 줄여, feat으로 명칭하기로 한다.
ex) feat/login
- 하나의 feature 브랜치의 개발이 끝났다면, developer 브랜치로 merge 한다.
! merge가 완료된 후에 feat 브랜치를 삭제해도 무관하다.
(혼자 사용중이라는 가정 하에 적용 가능)
- release 브랜치로 배포 중에 버그 발생한다면, develop 브랜치에서 수정 → release 브랜치에 merge 한다.
- 본인 로컬 환경에서는 어떤 브랜치 이름이어도 무관하나, upstream으로 merge할 때는 반드시 기능이름으로 브랜치를 명확하게 생성하도록 한다.

** How to create Branch? 는 아래의 패널을 참조하세요

📖 » A successful Git branching model ~ 더 자세한 Git Branch 전략 알아보기

2 PR

- PR의 Reviewers는 All(팀 내 모두)로 한다.
- review를 받고 approve를 받아야만 upstream의 repository로 merge할 수 있다.
- reviewer 중 한 명이라도 reject하면 merge할 수 없다.

** How to create PR? 은 아래의 패널을 참조하세요

3 Issue & MileStone

Milestone

- 배포버전 기준으로 마일스톤하기로 한다.
- 마일스톤을 생성할때, [배포버전 - 상세내용] 으로 작성하도록 한다.
단, 추가 설명 필요시 description 사용



💡 배포버전 기준으로 마일스톤 되어 있는 예시

Issue

- upstream에 해당하는 repository에서 Issue를 생성하도록 한다.
- 아래의 라벨들 중 해당되는 것으로 라벨링을 하도록한다.

Label	설명
bug	bug에 해당하는 이슈
not a bug	bug가 아닌 이슈 (bug가 아니라 다른것을 해결하기 위해)
refactoring	리팩토링을 하기 위한 이슈
feature	새로운 기능을 추가하기 위한 이슈
enhancement	기존에 있던 기능을 강화하기 위한 이슈

4 Commit Message

- Commit Message 규약은 아래와 같이 정의한다.

```
1 # 간단하게 작성하는 경우
2 type : description (#issue) (#issue)
3
4 # 자세하게 작성하는 경우
5 # title작성 후, enter해야 본문으로 인식
6 # 한번 더 enter할 경우 title bold로 처리 (생략가능)
7 type : title (#issue) (#issue)
8 body
```

Resolved: 빌드 스크립트, 배포를 위한 파일 추가
fix: build-docker.sh - docker rmi 에 -f 옵션 추가
feat: 설정 변경
Resolved: .gitignore 수정, 일부 파일 제거
Resolved: src, frontend 폴더 source 폴더 하위로 경로 변경
Update README.md
Resolved: 빌드 스크립트, 배포를 위한 파일 추가

💡 실제 IWDF 에서 사용하는 Commit message 예시

- description이나 title은 git graph에서 한 눈에 보일 수 있도록 최대한 명확하게 작성한다.
- issue는 관련된 이슈를 태그하며, 첫번째 태그는, 해결이슈를 두번째 태그는, 참고이슈를 명칭한다.
- type은 아래와 같이 정의한다.


type	설명
FEAT	새로운 기능의 추가
FIX	버그 수정
DOCS	문서 수정
STYLE	스타일 관련 기능

documentation	문서작성 및 문서 업데이트를 위한 이슈
---------------	-----------------------

- 해당되는 마일스톤을 선택하도록한다.
: 선택을 해주어야, 해당 마일스톤의 진행률을 정확히 알 수 있기 때문

** How to create Issue & MileStone ? 은 아래의 패널을 참조하세요

REFACTOR	코드 리팩토링
TEST	테스트코드 추가 및 수정
CHORE	빌드 업무 수정, 패키지 매니저 수정, gitignore 수정 등

 좋은 git 커밋 메시지를 작성하기 위한 7가지 약속 : [HN Cloud Meetup](#) ← 더 자세한 커밋 메시지 규약에 대해 알아보기

? How to create PR (Pull & Request) ? ↗

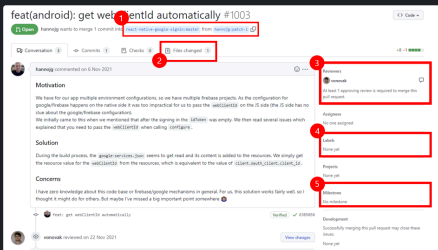
Pull & Request : 본인의 작업을 해당 소스에 반영시키려고 하니 검토를 부탁하는 요청을 뜻한다.

첫 번째 local 개발환경 생성

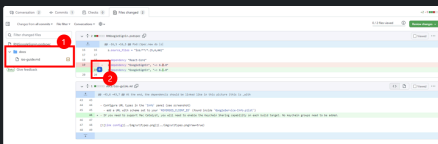
1. 가져올 repository를 본인의 repository로 fork 한다.
2. fork 해온 repository를 clone 하여 개발한다.
이때 clone 해 온 본인의 개발환경의 remote 는 origin(또는 본인이 정한 별칭) 이 된다.
3. 본인의 개발환경에서 기능에 맞는 브랜치 를 생성하여 개발을 진행한다.
- ex) feat/login
- 최대한 기능이 섞이지 않도록 브랜치를 생성하여 진행
4. 이렇게 계속 본인의 local 개발환경에서 계속 개발을 진행하면 된다.

두 번째 PR 요청

- Create Draft PR : 본인이 초안으로 작업한 Commit 내역을 공유하고 추후에 PR
- Create PR : 본인의 작업물이 완벽하게 완료되었을때 PR



PR 생성 시 화면



첫 번째 이미지에서 2)Files changed 를 선택 시 화면

1. 본인의 local 개발 환경에서 작업이 끝난 후, upstream remote 본인의 작업 물을 반영해야 할 때 PR을 생성한다.
 - a. PR 생성 시, 어디에서 어디로 merge 할 것인지 설정한다 (첫 번째 이미지의 1번)
2. PR을 생성하게 되면 첫 번째 이미지와 같은 화면에서 아래와 같은 설정을 진행한다.
 - a. PR에 대한 상세설명 ★
 - b. Reviewer 설정
 - c. Label 설정
 - d. MileStone 설정
3. Reviewer 입장에서 는 해당 PR 에 대한 검토를 진행한다.
 - a. 첫 번째 이미지의 2) Files changed 를 선택하면 두 번째 이미지의 화면이 나온다
 - b. 두 번째 이미지에서 PR에 적용된 변경된 파일들(두 번째 이미지의 1번)을 확인할 수 있으며,
두 번째 이미지의 2번처럼 + 버튼을 눌러 해당 line에 Comment를 달 수 있다.
 - c. reject or approve 와 관련된 Comment는 반드시 하도록 한다.

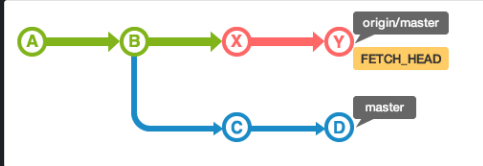
! PR하려고 하는데 Confilct 발생하는 파일이 있는 경우

→ conflict 발생하는 파일을 수정 후 다시 Commit 하면 된다.

! 본인이 merge 하기 전에 다른 협업자가 merge를 같은 브랜치에 이미 한 경우

```
1 git checkout (branch 이름)      # 개발 중인 브랜치로 이동(이미 해당 위치라면 2번부터 시작)
2 git fetch --all                  # fetch를 진행 (즉, 누군가 변경해 놓은 작업을 끌어가져오는 작업)
3
4 git rebase (upstream의 branch)  # rebase 진행 -> 진행 후, conflict 있는 경우 해결
5 git rebase --continue           # rebase 다시 continue
6 git rebase --abort               # rebase 상태 이전으로 다시 돌리기
```

- 2번 항목을 진행하게 되면, 본인의 local 개발환경 브랜치에서의 commit 이력과는 별개로 이름없는 브랜치로 내역을 가져온다.



- 4번 항목을 진행하게 되면 upstream의 브랜치의 변경내역이 본인 branch로 rebase 된다. rebase와 merge는 최종 결과는 같지만 commit history만 다르다.

• Merge

- 시간 순서에 따라 다른 브랜치의 커밋까지 upstream 브랜치로 들어간다.
-



• Rebase

- 본인이 작업하고 있는 브랜치의 commit이 최상단으로 나타나고, 다른 브랜치들도 각각 자신의 브랜치로 묶여 commit history가 남는다.
 - 즉, Rebase의 경우는 브랜치의 변경사항을 순서대로 다른 브랜치에 적용하면서 합치고 Merge의 경우는 두 브랜치의 최종결과만을 가지고 합친다.



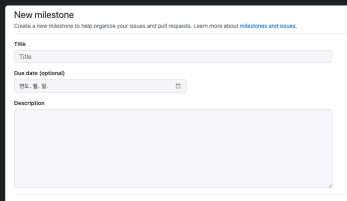
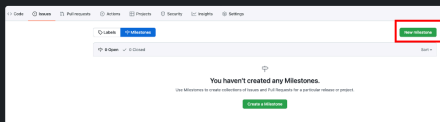
• Squash

- 여러 개의 commit 메시지를 하나로 합친다.

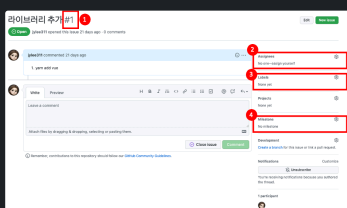
? How to create Issue & MileStone ?

첫 번째 MileStone 생성 (마일스톤이란 [프로젝트 flow가 반드시 거쳐야하는 포인트]를 설정해두는 것)

1. Git Repository에서 아래 이미지와 같이 new MileStone하여 생성한다. 이때, Title은 배포 버전으로 작성한다.



두 번째 Issue 생성



- 본인이 지금 하고 있는 것들을 이슈화하는 것으로, 이슈를 생성하면 해당 이슈 옆에 # (number) 가 부여된다. (위의 이미지의 1번)
- 해당 number를 commit message에 함께 넣어주면 이슈관리가 된다.
ex) feat : 로그인 기능의 OAuth 기능 추가 (#1)

- 해당 이슈와 관련된 참조자를 추가할 수 있다. (위의 이미지의 2번)
- 해당 이슈를 라벨링 할 수 있다. (위의 이미지의 3번)
- ★ 해당 이슈를 관련된 마일스톤에 추가 할 수 있다. (위의 그림의 4번)
이를 해주어야, 마일스톤 관리가 된다.
- 해결된 이슈는 반드시 `close` 처리를 한다.

? branch와 관련된 Git 명령어

- 브랜치 삭제
 - `git branch -D 지을브랜치이름`
 - `git push origin --delete 지을브랜치이름` → 본인의 레파지토리에도 push되어 적용된다.
- `git status` : 지금 브랜치 상황을 보여준다.
- `git diff` : 지금 브랜치에서 변경된 사항들 보여준다.
- `git branch` : 현재 존재하는 브랜치 목록 보여준다.
- `git log` : 현재 브랜치에서 여태까지 커밋한 흔적들을 보여준다.
- `git checkout -b 브랜치 이름` : 해당 브랜치 생성 + 체크아웃을 진행한다.

🔥 Coding Convention

1 Git Repo Rule

1. 소문자로 작성한다.
2. hyphen으로 작성한다
3. 큰기능-작은기능-서브-... 로 작성한다
4. 명사형으로 작성한다
5. MSA 인 경우, `service` 를 suffix로 사용한다
6. 일반 서버 모듈일 경우, `server` 를 suffix로 사용한다

2 Java Convention

1. 패키지명은 `com.mobigen.nl.pnms` 로 정한다.
2. 파일구조는 기능 별로 분리하도록 한다.
3. 패키지 를
 - a. `domain` 은 table명이나, 개념적 개체 명, 명사 일 때 사용한다.
 - b. `dto` 는 외부 송수신용, 시스템 기준으로 수신시 `Request`, 송신시 `Response` 를 사용한다.
 - c. 가능한 패키지명은 상속받은 기능명을 기준으로 정의한다.
 - d. `domain/dto` 패키지도 많아지면 세부 패키지로 분리한다.
 - e. utility성 패키지
 - i. 모든기능에서 쓰는건 공통 패키지 에 둔다.
 - ii. `spring`에서 component로 관리하는건 `component` 로 뺀다.
 - iii. static method성으로 처리하는건 utility로 뺀다.
 - f. `MainApplication` 파일은 root 디렉터리 하위에 단독으로 둔다.
4. 파일 인코딩은 UTF-8
5. 이름
 - a. 식별자에는 영문, 숫자, 언더스코어만 허용
 - b. 한국어 발음대로 표기 금지
 - 나쁜 예 : `moohyungJasan` (무형자산)
 - 좋은 예 : `intangibleAssets` (무형자산)
 - c. 대문자로 표기할 약어 명시
 - d. 패키지 이름은 소문자로 구성
 - 패키지 이름은 소문자를 사용하여 작성한다. 단어별 구문을 위해 언더스코어()나 대문자를 섞지 않는다.

```
1 // 나쁜 예
2 package com.example.apiGateway
3 package com.example.api_gateway
4
5 // 좋은 예
6 package com.example.apigateway
7
```

- e. 클래스, 인터페이스 이름에 대문자 카멜표기법 적용

3 Frontend Convention

1. IWDf의 coding convention을 따른다.
 - a. 파일구조(기능별로 분리)
 - b. css파일이름은 Pair 로 같은 이름으로 짓는다.
2. 주석은 **지양**한다
3. 공통으로 사용하는 Component(사용자 명의 컴포넌트)의 명칭은 아래의 규칙을 따른다

```
1 #1) 모두 소문자이다.
2 #2) 필요 시 하이픈을 포함한다.
3 #3) 큰 기능 → 작은 기능 순으로 명칭 한다.ex) login-input-password.vue
4 #4) 구체적으로 어떤 역할을 하는지 알 수 있어야 한다.
5 #5) 어떤 역할을 하는지에 대한 설명은 /* */ 주석을 이용한다
```

4. props 는 kebab-case를 사용한다.

```
`
```

`my-message`와 같이 케밥케이스를 사용한다

📖 [코딩컨벤션](#) → 더 자세한 코딩컨벤션에 대해 살펴보기

- 클래스 이름은 단어의 첫 글자를 대문자로 시작하는 대문자 카멜표기법(Upper camel case)을 사용한다.

```
1 // 좋은 예
2 public class AccessToken
3
```

f. 클래스 이름에 명사 사용

- 클래스 이름은 명사나 명사절로 짓는다.

g. 인터페이스 이름에 명사, 형용사 사용

- 인터페이스(interface)의 이름은 클래스 이름은 명사/명사절로 혹은 형용사/형용사절로 짓는다.

```
1 // 좋은 예
2 public interface RowMapper {
3
4 public interface AutoClosable {
5
```

h. 테스트 클래스는 'Test'로 끝남

i. 메소드 이름에는 소문자 카멜표기법 적용

j. 상수는 대문자와 언더스코어로 구성

- `static final`로 선언되어 있는 필드일 때를 상수로 간주한다. 상수 이름은 대문자로 작성하며, 복합어는 언더스코어(_)를 사용하여 단어를 구분한다.

k. 변수에 소문자 카멜표기법 적용

- 상수가 아닌 클래스의 멤버변수/지역변수/메서드 파라미터에는 소문자 카멜표기법(Lower camel case)을 사용한다.

l. 1글자 이름 사용 금지

- 메서드 블록 범위 이상의 생명 주기를 가지는 변수에는 1글자로 된 이름을 쓰지 않는다.

```
1 // 나쁜 예
2 HtmlParser p = new HtmlParser();
3
4 // 좋은 예
5 HtmlParser parser = new HtmlParser();
6
```

6. 선언

a. static import에만 와일드 카드 허용

- 클래스를 import할때는 와일드카드(*) 없이 모든 클래스명을 다 쓴다. static import에서는 와일드카드를 허용한다.

```
1 // 나쁜 예
2 import java.util.*;
3
4 // 좋은 예
5 import java.util.List;
6 import java.util.ArrayList;
7
```

- 애너테이션 선언 후 새 줄 사용

```
1 // 좋은 예
2 @RequestMapping("/guests")
3 public void findGuests() {}
4
```

- 한 줄에 한 문장, 하나의 선언문에는 하나의 변수만

```
1 // 나쁜 예
2 int base = 0; int weight = 2;
3
4 // 좋은 예
5 int base = 0;
6 int weight = 2;
7
```

7. 들여쓰기

- a. 탭(tab) 문자를 사용하여 들여쓴다. 탭 대신 스페이스를 사용하지 않는다.
- b. 탭의 크기는 4개의 스페이스
- c. 소스 내 불필요한 공백은 제거



1. Java 단 에서만 진행한다.

