



Git



Github-flow 전략 : (F_F01_00_BE_최강)

🍪 브랜치 이름

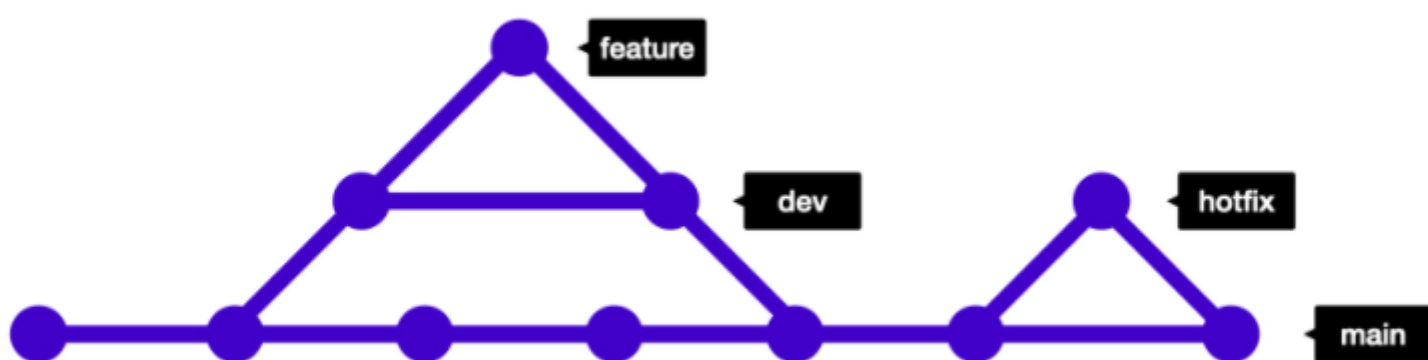
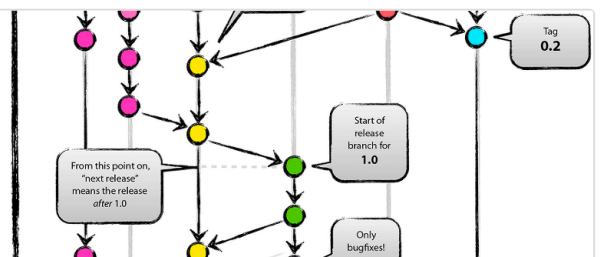
규칙

- 브랜치종류_요구사항id_포지션_이름
 - 브랜치 요구사항 id가 없으면 요구사항에 추가할것.
- **main** : 최종적으로 머지되는 곳, 배포 및 최종본 / 출시 버전 브랜치
- **dev** (from main): 기능을 완성해서 머지하는 곳
- **front** (from develop) : 프론트 기능을 개발하는 브랜치
 - F01_00_FE_이동명
- **back** (from develop) : 백 기능을 개발하는 브랜치
 - F01_00_BE_최강
- **nft** (from develop) : NFT 기능을 개발하는 브랜치
 - F01_00_NFT_김규민
- 만들고자하는 기능을 dev 에서 브랜치함
- 기능 완성 후 MR
- MR에 댓글로 피드백
- 같은 파트 다른사람이 Merge

[Git] git-flow 소개, 설치 및 사용법

하나의 소스코드로 여러명의 개발자들이 협업을 하게 되면서 소스코드의 버전 관리 시스템의 중요성이 매우 높아졌다. 과거에는 SVN, CVS 같은 소프트웨어들도 많이 사용되었지만 최근에는 거의 git으로 통일되어 가는 듯 하다. (그럼에도 아직까지 파일 서버에서 소스코드를 업로드하는 원시적인 방법을 사용하는 프로젝트도 많다.) SVN과 CVS에 비해 git이 갖

🔗 <https://hbase.tistory.com/60>



🍪 Pull request 작성하기

PR 형식

- 코드 컨벤션을 잘 지키기

컨벤션 오류로 인한 불필요한 코멘트는 시간 낭비이기 때문에 지양하는 것이 좋다.

- 리뷰 가이드 잘 작성하기

모든 코드 변경사항에는 의도가 필요하다. 의도치 않게 변경된 부분이 있다면 되돌려 놓아야하고, 줄바꿈과 같이 아주 단순한 변경이라도 그 부분을 리뷰어가 볼 필요가 없다면 "Just line change"와 같은 코멘트를 달아 명시하여 리뷰시간을 줄여줄 수 있을 것이다. 또는 사용된 라이브러리 업데이트가 포함되었다면 해당 라이브러리의 릴리즈 노트 링크나 스크린샷을 첨부하는 것도 좋은 방법이다.

- 작업 중, 리뷰 가능 여부를 잘 명시하기

아직 코드를 작성 중 일때에는 [Wip](Work in Progress)를 타이틀 앞에 추가하고, 만약 작업이 끝났으면 이를 제거하고 review-needed태그를 설정할 수 있다. 한 번 작업을 마쳤다고 끝난 것이 아니기 때문에 리뷰를 반영하는 중에도 이를 반복하여 명시해야한다.

- PR 본문

- 아래는 Github Pull Request의 템플릿으로 지정 후 해당 본문은 삭제하면 된다!

```
### PR 타입(하나 이상의 PR 타입을 선택해주세요)
- [ ] 기능 추가
- [ ] 기능 삭제
- [ ] 버그 수정
- [ ] 의존성, 환경 변수, 빌드 관련 코드 업데이트

### 변경 사항
ex) 로그인 시, 구글 소셜 로그인 기능을 추가했습니다.<< 로직은 간단하게
JWT를 활용했습니다~ <<
```

ISSUE 형식

- 아래 형식을 복사해 Github Issue 템플릿으로 지정 후 해당 본문은 삭제하면 된다!

🍪 Commit 작성하기

프론트 템플릿 따로 백 템플릿 따로

이슈별로 [FE], [BE],[NFT] 나누기

커밋 메시지 예시

타입 : 최대한 한글로 작성하기 #이슈번호

ex) feat : 자유게시판 API 개발 #17


```
PR에 "로그인 기능 구현" 어떤 기능일지 명세

커밋 예시
feat: 새로고침 시 로그인 유지 기능 개발 #22
```

Rules

커밋 메시지는 태그, 본문, 이슈번호 로 나뉘어진다!

- 제목은 명령어, 개조식으로 작성
- 커밋 메시지 태그는 소문자로 작성하기

- 내용은 한글로 작성하기
- 내용은 어떻게 변경했는지 보다 무엇을 변경했는지, 왜 변경했는지 에 맞추어 작성
- 끝에 이슈 번호 달아주기 (있을 경우만)
- 제목 끝에  는 금지

커밋 메시지 (1)

| Aa 제목 태그 이름 | ≡ 설명 |
|-------------|----------------------------------|
| feat | 새로운 기능을 추가할 경우 |
| fix | 버그를 고친 경우 |
| style | 코드 포맷 변경, 세미 콜론 누락, 코드 수정이 없는 경우 |
| refactor | 프로덕션 코드 리팩토링 |
| comment | 필요한 주석 추가 및 변경 |
| docs | 문서를 수정한 경우 |
| test | 테스트 추가, 테스트 리팩토링 (프로덕션 코드 변경 X) |
| rename | 파일 혹은 폴더명을 수정하거나 옮기는 작업만인 경우 |
| remove | 파일을 삭제하는 작업만 수행한 경우 |
| setup | 세팅 코드 추가 |

```
## :hammer: 기능 설명

기능 설명

## 📄 완료 조건

- [ ] 완료 조건 1
- [ ] 완료 조건 2
```

```
## :hammer: 기능 설명

기능 설명

## 📄 완료 조건

- [ ] 완료 조건 1
- [ ] 완료 조건 2
```

```
## ⚠ 버그 설명

버그 설명

## 📄 완료 조건

- [ ] 완료 조건 1
- [ ] 완료 조건 2
```

자동으로 PR 템플릿 지정하는 방법

ISSUE 템플릿 등록하는 방법

```
# [프론트 or 백] : 타입_제목, 기능(페이지도 가능) | 내용들?
##### 제목은 최대 50 글자까지만 입력 ##### -> |

# 본문은 위에 작성
##### 본문은 한 줄에 최대 72 글자까지만 입력 ##### -> |

# 꼬릿말은 아래에 작성: ex) #이슈 번호

# --- COMMIT END ---
# <타입> 리스트
# feat : 추가(새로운 기능)
# fix : 버그 (버그 수정)
# refactor : 리팩토링
# style : 스타일 (코드 형식, 세미콜론 추가: 비즈니스 로직에 변경 없음)
```

```
# docs      : 문서 (문서 추가, 수정, 삭제)
# test      : 테스트 (테스트 코드 추가, 수정, 삭제: 비즈니스 로직에 변경 없음)
# chore     : 기타 변경사항 (빌드 스크립트 수정 등)
# setup    : 셋팅
# -----
#   타입은 대문자로
#   제목은 타입_제목
#   제목 끝에 마침표(.) 금지
#   제목과 본문을 한 줄 띄워 분리하기
#   본문은 "어떻게" 보다 "무엇을", "왜"를 설명한다.
# -----
```