Git 컨벤션(Git Flow)

📌 참고 : https://techblog.woowahan.com/2553/

Git 활용 규칙 (Conflict 방지)

1. **다른사람이 작성한 코드는 임의로 건들지 말자**
2. **git pull을 통해 원격저장소(gitlab)와 로컬저장소를 자주 동기화하자**

   🚨 **Commit, Push 전에 Pull 하는 습관 들이자!**

3. **master, develop 브랜치에 직접 push 금지 ⇒ 대신 MergeRequest(MR)을 날리자**
4. **commit, push를 활성화 하자**

명명 규칙

Feature 브랜치 이름 규칙

- 예시 : “**feature-front/230112-add-readme**”
- 구조
  - “feature-front(back)/”
  - 날짜(YYMMDD) + “-”
  - 동사
    - **현재형**[ex: add, change, modify]
      과거형 금지[ex: added, changed, modified]
  - 내용

    - 띄어쓰기는 “-”

Commit 규칙

- 예시 : “**fix: deliveryList page header**”
- 구조 : Keyword + “: “ 내용
- Keyword 리스트
    - `feat` : 새로운 기능 추가
    - `fix` : 버그 수정
    - `docs` : 문서 수정
    - `style` : 코드 포맷팅, 세미콜론 누락, 코드 변경이 없는 경우
    - `refactor` : 코드 리펙토링
    - `test` : 테스트 코드, 리펙토링 테스트 코드 추가
    - `chore` : 빌드 업무 수정, 패키지 매니저 수정

Merge Request(MR) 규칙

- Title : 어떤 내용인지 정확히 기재
- Description : 최대한 구체적으로 어떤 기능을 추가/수정/삭제했는지 기재
- MR 예시

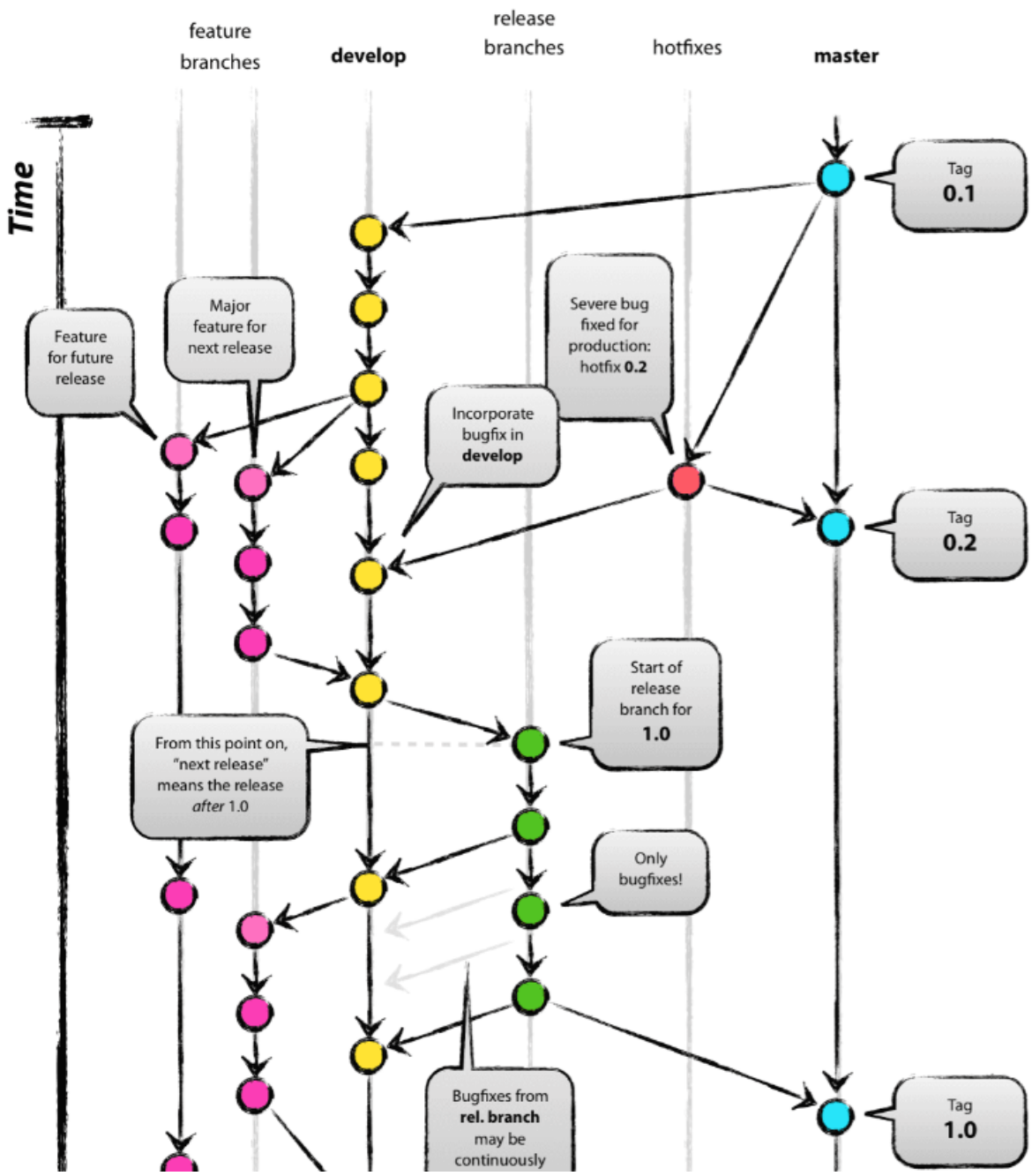
Branch 만드는 법

1. git branch feature-front[또는 back]/230116-add-readme
2. git checkout feature-front[또는 back]/230116-add-readme => feature 브랜치로 바꾸기

3. git add .
4. git commit -m "[커밋메세지]"
5. git push -u origin feature-front[또는 back]/230116-add-readme => 이번 첫 push 이후에는 git push만 해도 됨 (-u origin feature... 이게 default를 origin feature....를 고정시킴)

브랜치 종류

1. **master** : 제품으로 출시될 수 있는 브랜치
    - 배포용 브랜치
2. **develop** : 다음 출시 버전을 개발하는 브랜치
    - 개발 과정에서는 develop 브랜치를 주 브랜치로 활용
    - master 브랜치로 MR 요청
3. **feature** : 기능을 개발하는 브랜치
    - 팀원 각각 담당하는 기능을 개발하기 위해 feature 브랜치를 생성하고, develop 브랜치로 MR 요청
4. **hotfix** : 급한 에러 수정



개발 흐름 설명

1. 처음에는 master와 develop 브랜치가 존재
    a. develop 브랜치는 master에서부터 시작된 브랜치
2. 새로운 기능 추가 작업이 있는 경우 develop 브랜치에서 feature 브랜치를 생성
    a. feature 브랜치는 언제나 develop 브랜치에서부터 시작
    b. 기능 추가 작업이 완료되었다면 feature 브랜치는 develop 브랜치로 merge

3. develop 브랜치에서 이번 버전에 포함되는 모든 기능이 merge되면 master 브랜치로 merge 후 배포
4. **브랜치 구조**
    - master
        - dev
            - dev-front
                - feature
            - dev-back
                - feature
        - hotfix

Git 명령어 모음

```
############################## 브랜치 지우기 ###################################

# 원격에서 삭제된 브랜치 지우기
git fetch --prune origin

# 로컬에서 브랜치 지우기
git branch -d [브랜치명]

############################## 수정 전으로 되돌리기 ############################

# git add 아직 하지 않은 상태에서 해당하는 파일만 수정 전으로 되돌리기
git restore 경로명/[파일명]

# git add 아직 하지 않은 상태에서 변경 사항 전부 수정 전으로 되돌리기
git restore .

# git add 취소하기 (### 단, git commit 작성 이전에만 적용 가능 ###)
git restore --staged [파일명]
```

Git 브랜치명 변경하기

`old-branch` : 잘못 만든 브랜치명

`new-branch` : 새로운 브랜치명

✔️ old-branch로 git push 한 적 없는 경우

1. **old-branch** 가 **아닌 브랜치**로 이동

ex) `git checkout dev-front`

2. 다음 명령어로 **브랜치명 변경**

`git branch -m old-branch new-branch`

3. `git branch -a` 명령어로 new-branch만 존재하는지 확인

✔️ old-branch로 git push 한 적 있는 경우

1. **old-branch** 가 **아닌 브랜치**로 이동

ex) `git checkout dev-front`

2. 다음 명령어로 **브랜치명 변경**

`git branch -m old-branch new-branch`

3. Git에 존재하는 old-branch를 삭제하기 위해 다음 명령어 실행

`git push origin --delete old-branch`

4. 새로운 브랜치명으로 푸시하기

`git push origin -u new-branch`

5. `git branch -a` 명령어로 new-branch만 존재하는지 확인

master, develop에서 dev-front/back으로 역머지

```
### Master까지 merge 한 상태에서

# Master 브랜치 에서
git pull

# Develop 브랜치 에서
git pull
git fetch origin master
git merge master
git push

# Dev-front/back 브랜치 에서
git pull
git fetch origin develop
git merge develop
git push
```