개발 입문 스터디

4주차

출석체크

Announcement

복습

HEAD

- HEAD는 현재 작업 중인 위치를 가리킨다
- 현재 작업중인 브랜치의 가장 최근 commit
- 다음 commit의 base가 되는 commit
- 새로운 commit이 생기면 HEAD도 변경

```
aee7ca2 (HEAD -> develop, origin/develop, origin/HEAD) hotfix: spotless 적용 (#285)
e462002 hotfix: 학과 쿼리 메서드 수정 (#284)
1f39f54 feat: 2차 모집 기간 마감 시 LandingStatus 조건 추가 (#278)
24a51dd refactor: 발생 가능성 있는 NPE를 제거 (#170)
a67923c fix: CI 워크플로우에서 Gradle 빌드가 수행되는 문제 해결 (#276)
2319f1f refactor: 쿼리 메서드 이름 개선 및 연관 로직 주석 추가 (#274)
```

commit --amend

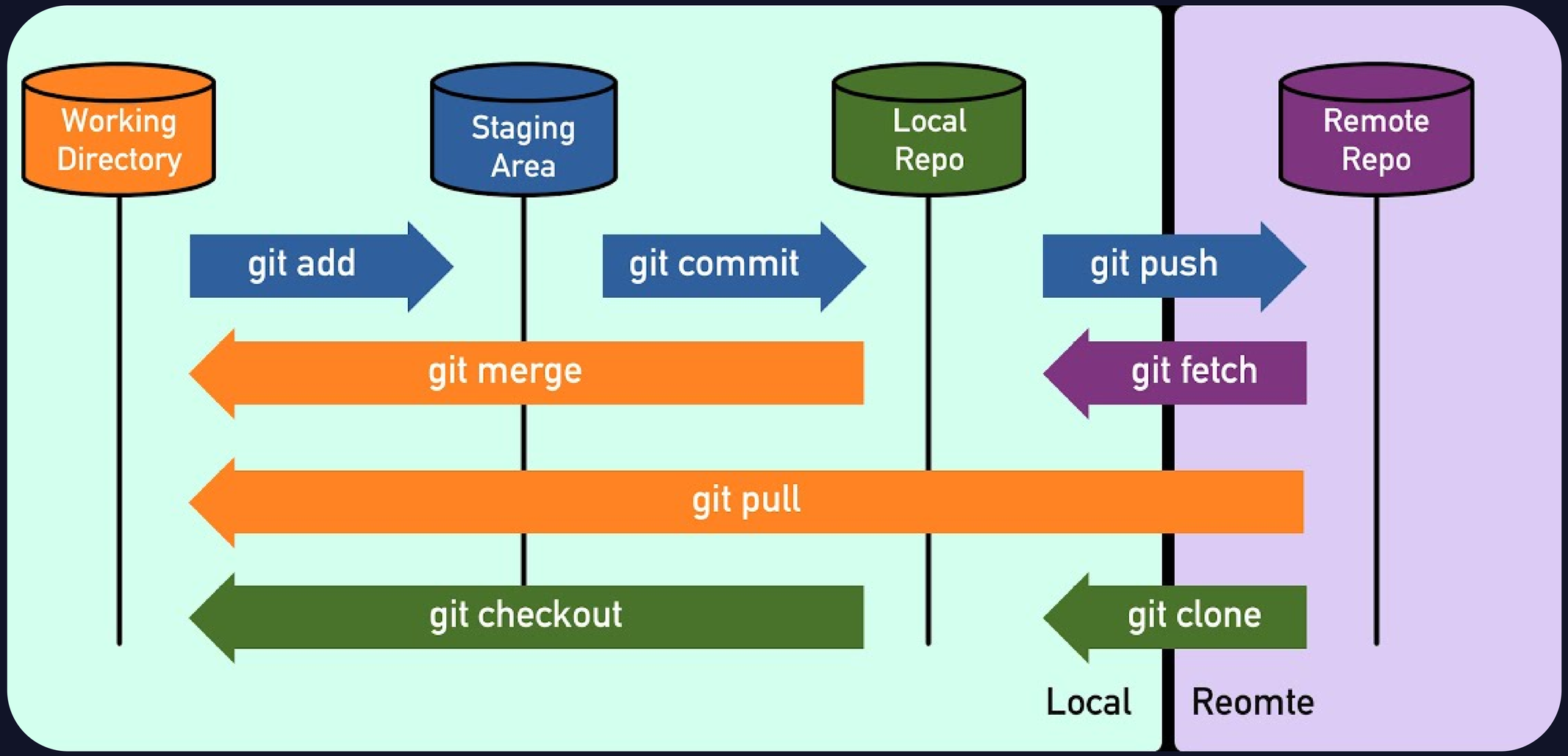
- 마지막 commit의 내용에 변경이 있을 때 사용
- 완전히 새로운 commit으로 대체
  - commit id가 바뀜
- vim으로 진입하여 commit 메시지를 수정하게 됨

🚨 commit --amend 주의사항

다른 사람이 작업 기반으로 삼고 있는 commit은
amend하면 안 됩니다!

reset

- commit을 제거하는 데 사용
- 3가지 옵션 존재
  - soft, mixed(default), hard


Working Directory
Staging Area
Local Repo
Remote Repo
git add
git commit
git push
git merge
git fetch
git pull
git checkout
git clone
Local
Reomte

reset --soft

- 커밋만 취소
- 변경 사항이 Staging Area로 돌아감

reset --mixed

- 커밋을 취소
- 변경 사항을 working directory로 돌아감

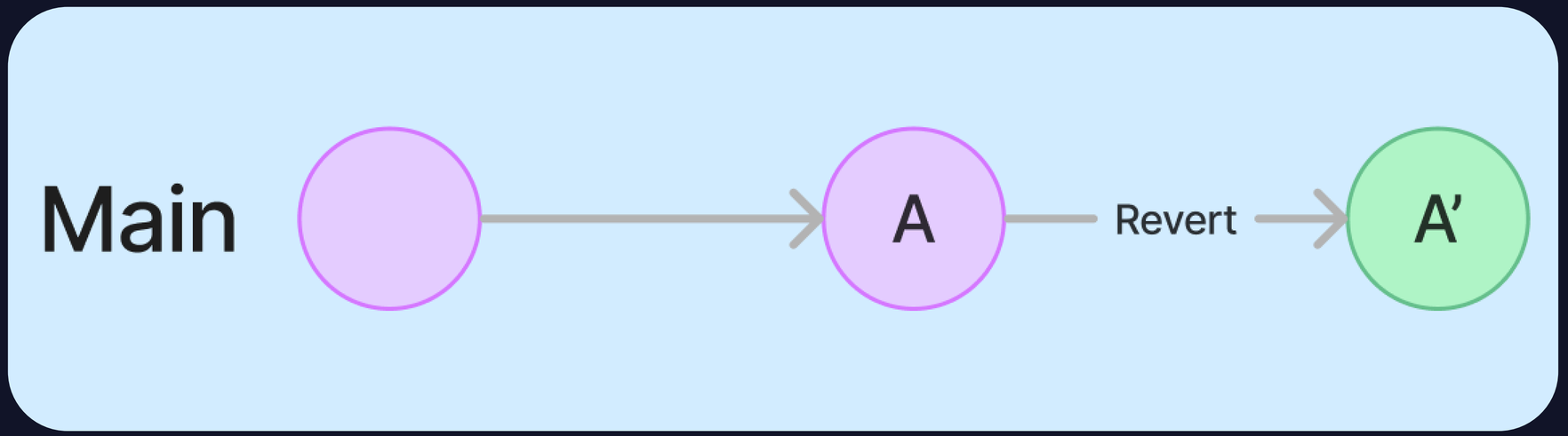
reset --hard

- 커밋을 취소
- 변경 사항을 모두 제거하고 이전 커밋으로 돌아감

revert

- commit을 제거하지 않고 되돌림
- 되돌리기 위한 새로운 commit이 생성됨
- --edit 옵션이 default

  ex) $ git revert a1s2d3f



reset vs revert

- reset은 commit을 삭제하므로 위험
- commit을 공유하는 다른 브랜치에도 영향을 줄 수 있음
- commit을 삭제하기보다 생성하여 되돌리는 revert가 안전

오늘의 토픽

- 브랜치 전략
  - git flow
  - github flow
- 개발자로서의 Attitude

브랜치 관리의 필요성

- 서로의 작업에 영향을 주지 않기 위해 브랜치의 분리 필요
- 각 브랜치가 어떤 작업을 위해 존재하는지 구분 필요
- main 브랜치의 안전한 관리 필요

브랜치 보호 규칙

- 승인 없이 Pull Request를 병합할 수 없도록 제한 가능
- 특정 브랜치에 Push 가능자를 제한 가능

⇒ 브랜치를 안전하게 관리

브랜치 전략

Git Flow vs. GitHub Flow

Git Flow

- 2010년에 Vincent Driessen이 고안한 방법
- 브랜치를 관리하기 위한 전략

Git Flow

Vincent Driessen의 글

브랜치 종류

Main Branches

- main(master)
- develop

Supporting branches

- feature
- release
- hotfix

main

- 프로젝트 생성 시 기본으로 생성되는 브랜치
- 영원히 존재하는 첫 번째 브랜치
- 병합될 때마다 제품의 새로운 버전이 탄생
- main이라는 이름 대신 master를 사용하기도 함

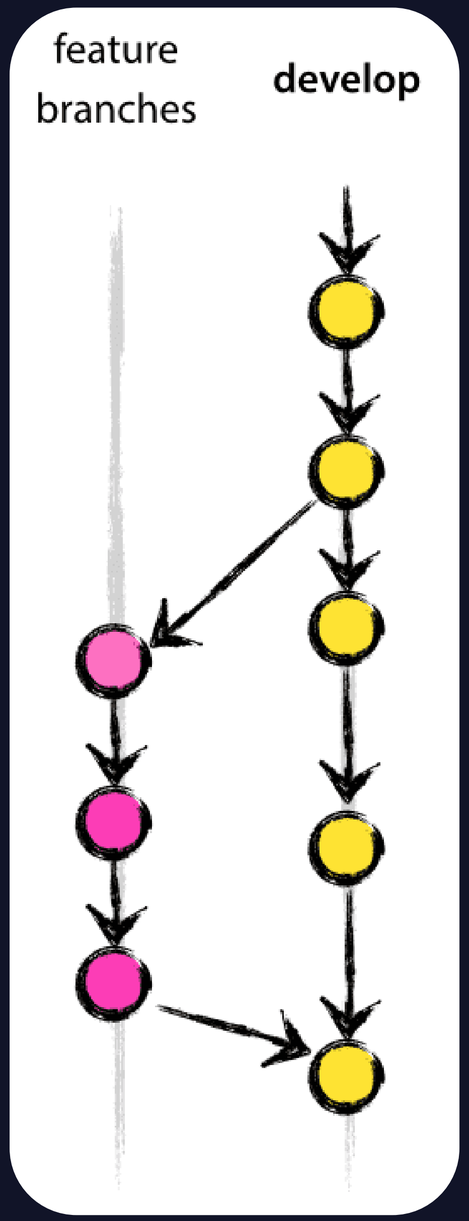
develop

- 영원히 존재하는 두 번째 브랜치
- feature 브랜치의 기반이 됨

feature

- develop 브랜치에서 분기하여 작업
- 기능 개발 완료 후 다시 develop으로 병합
- 대부분의 이름 가능
  - 단, main, master, develop, release, hotfix 제외

feature



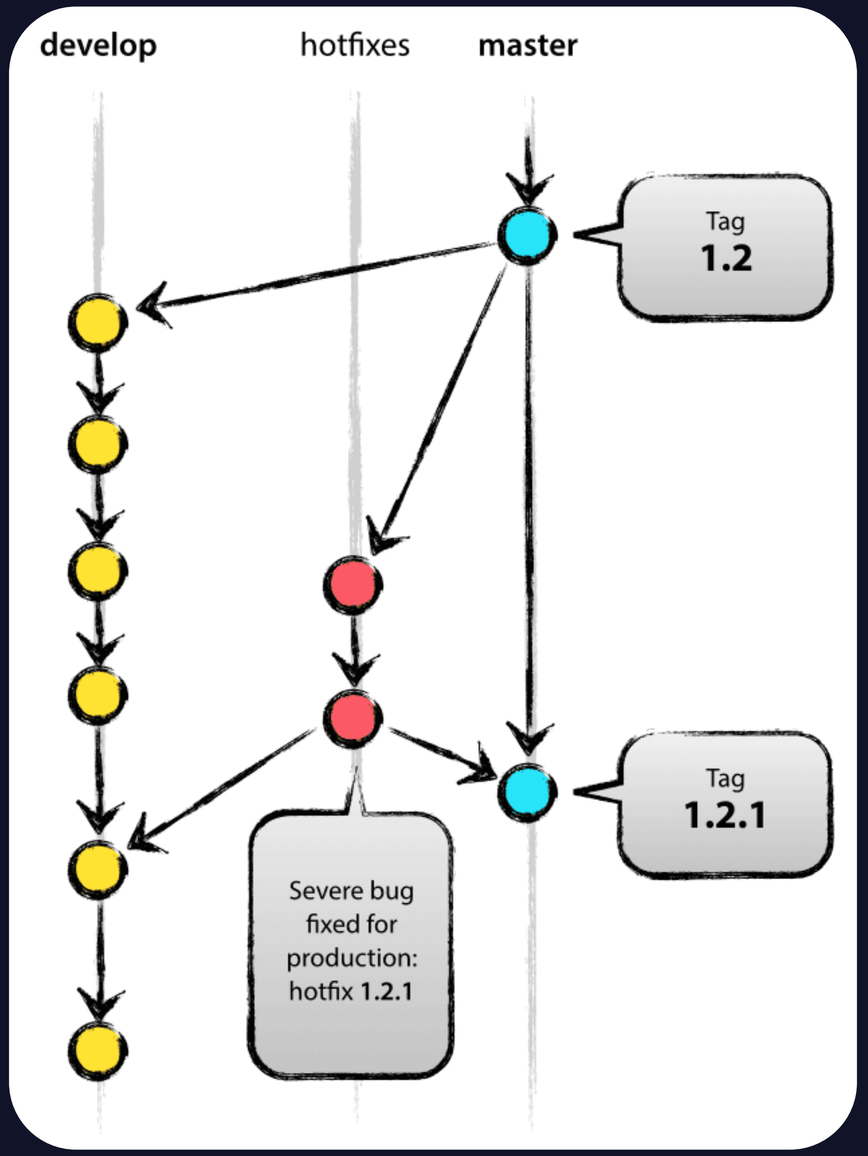
release

- 배포 준비를 위한 브랜치
- 자잘한 버그를 수정하고 QA 작업을 함
- develop 브랜치에서 분기하여 main 브랜치로 병합

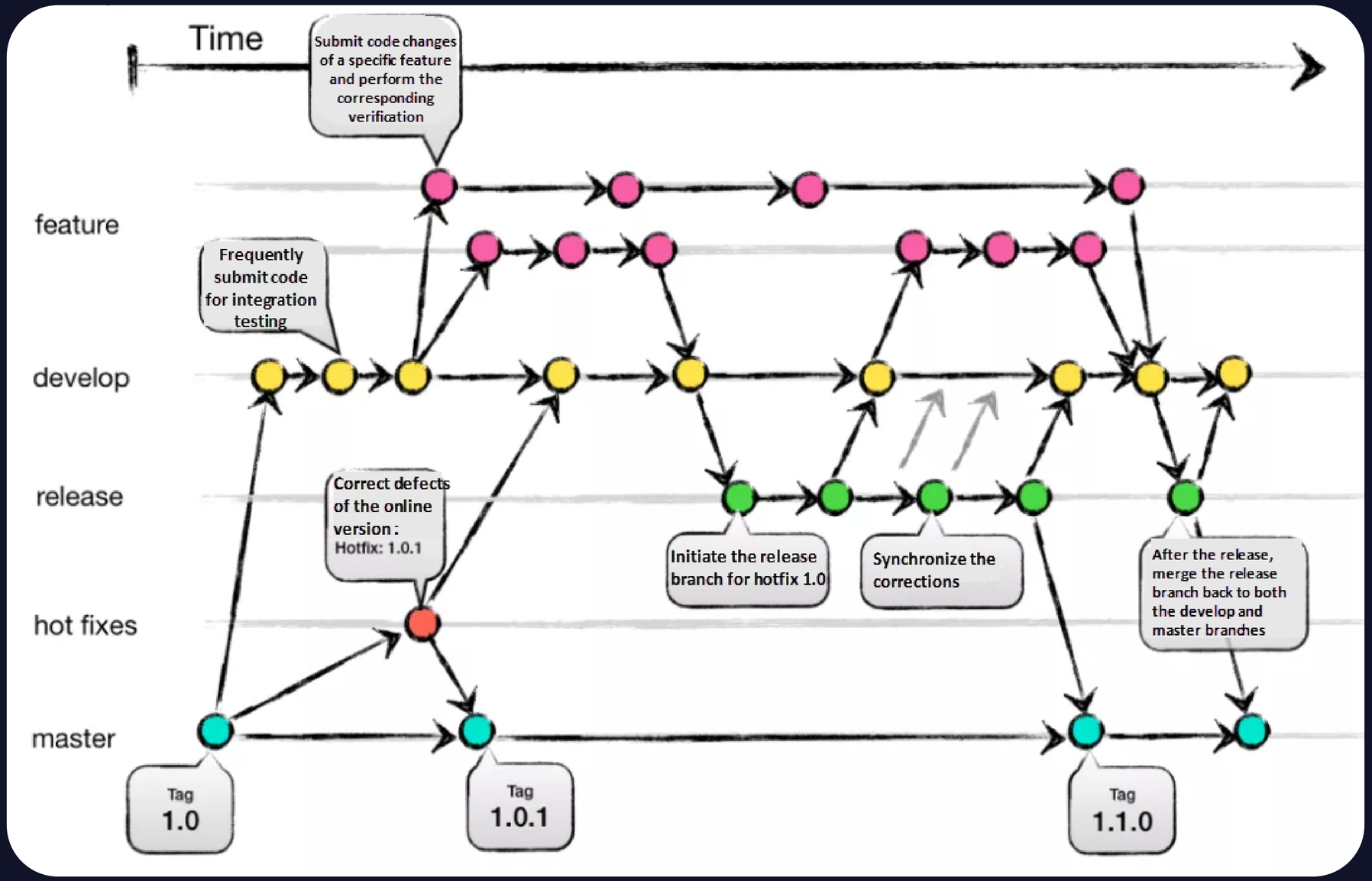
hotfix

- 배포 환경에서 즉각적인 수정이 필요할 경우 사용
- main 브랜치에서 분기
- main, develop 모두에 병합 필요

hotfix


develop
hotfixes
master
Tag
1.2
Tag
1.2.1
Severe bug
fixed for
production:
hotfix 1.2.1

Git Flow


Time
Submit code changes of a specific feature and perform the corresponding verification
feature
Frequently submit code for integration testing
develop
release
Correct defects of the online version : Hotfix: 1.0.1
Initiate the release branch for hotfix 1.0
Synchronize the corrections
After the release, merge the release branch back to both the develop and master branches
hot fixes
master
Tag 1.0
Tag 1.0.1
Tag 1.1.0

Git Flow의 시대는 끝났다?

- 배포가 수시로 이루어지는 현 시대의 웹앱과는 부적합
- Vincent Driessen도 GitHub Flow와 같은 전략을 추천

GitHub Flow

수시로 배포되는 상황이 Git Flow와 안 어울리는 것을
개선하기 위해 고안한 방법

GitHub Flow

Scott Chacon의 글

브랜치 종류

Main과 Feature 브랜치만 사용

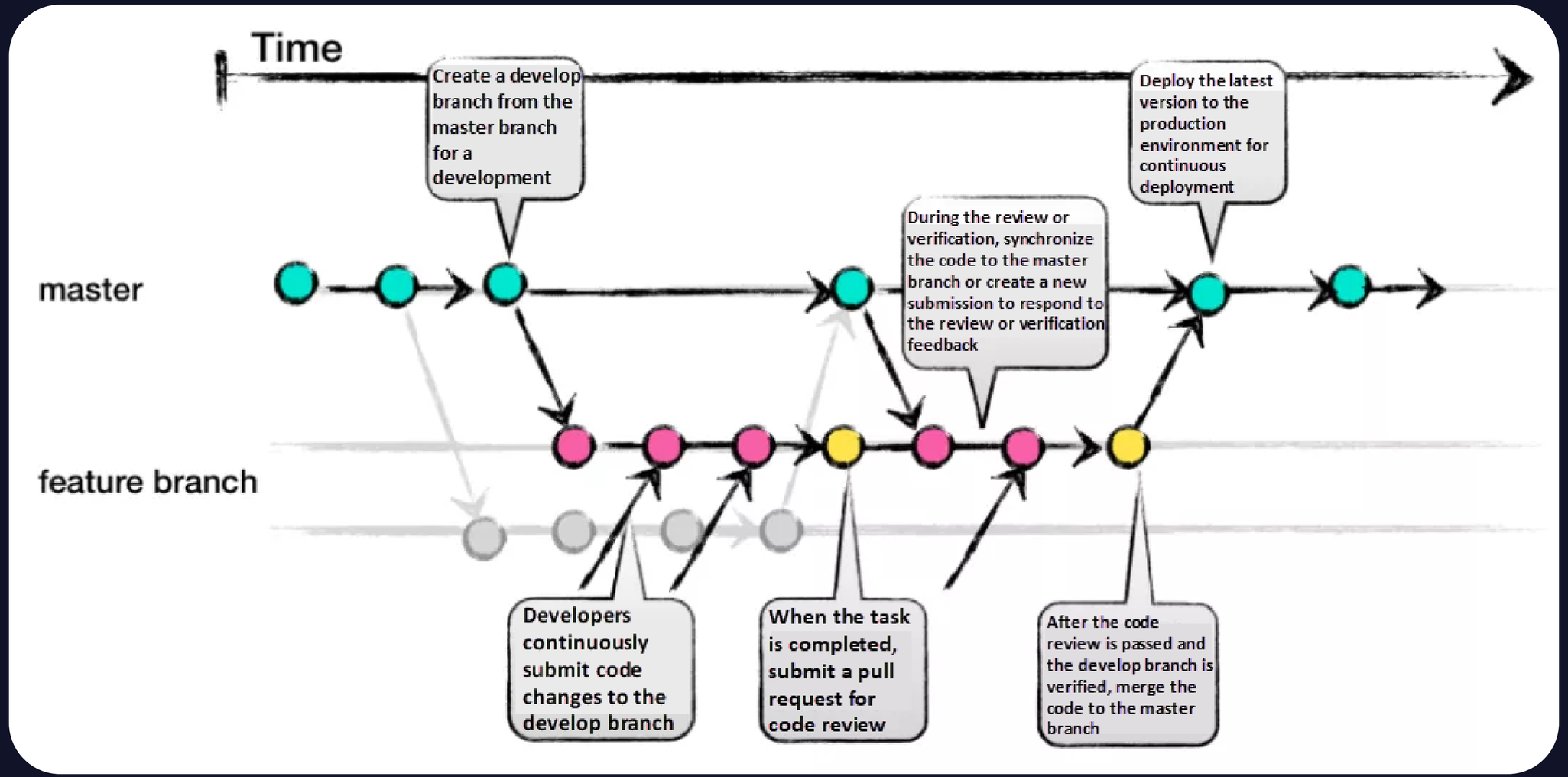
main

- 항상 배포 가능 상태로 유지
- main으로 병합 전에 충분한 test 필요

feature

- Git Flow와 달리 이외 브랜치들을 구분하지 않음
- main에서 분기하여 작업 후 다시 main으로 병합
- 브랜치의 목적을 이름에 잘 담아야 함
- Git Flow에 비해 코드 리뷰가 더 중요함

GitHub Flow


Time
Create a develop branch from the master branch for a development
Deploy the latest version to the production environment for continuous deployment
During the review or verification, synchronize the code to the master branch or create a new submission to respond to the review or verification feedback
master
feature branch
Developers continuously submit code changes to the develop branch
When the task is completed, submit a pull request for code review
After the code review is passed and the develop branch is verified, merge the code to the master branch

그래서 어떤 전략을 사용해야 하는가

Attitude

convention을 만들어 사용하자

긴 시간이 지난 후에도 convention 덕에
쉽게 의도를 파악할 수 있다.

+ 보기 좋은 프로젝트가 되는 건 덤

구글링을 꼼꼼히 하자

알 것 같은 사람을 붙잡고 물어보는 것은 쉽지만,
금방 휘발된다.

직접 찾아보고 진짜 모르겠을 때만
주변 사람들에게 도움을 요청하자!

코드에 대한 주인 의식을 가지자

public repository에 있는
코드는 곧 내 얼굴이다.

돌아가기만 하는 코드보다는
잘 설계된 코드를 짜도록 노력하자.

질문을 잘 하자

결국 누군가에게 질문을 해야하는 순간이 온다.

좋은 질문을 할 수 있도록 노력하자.

Summary

Assignment

고생하셨습니다!