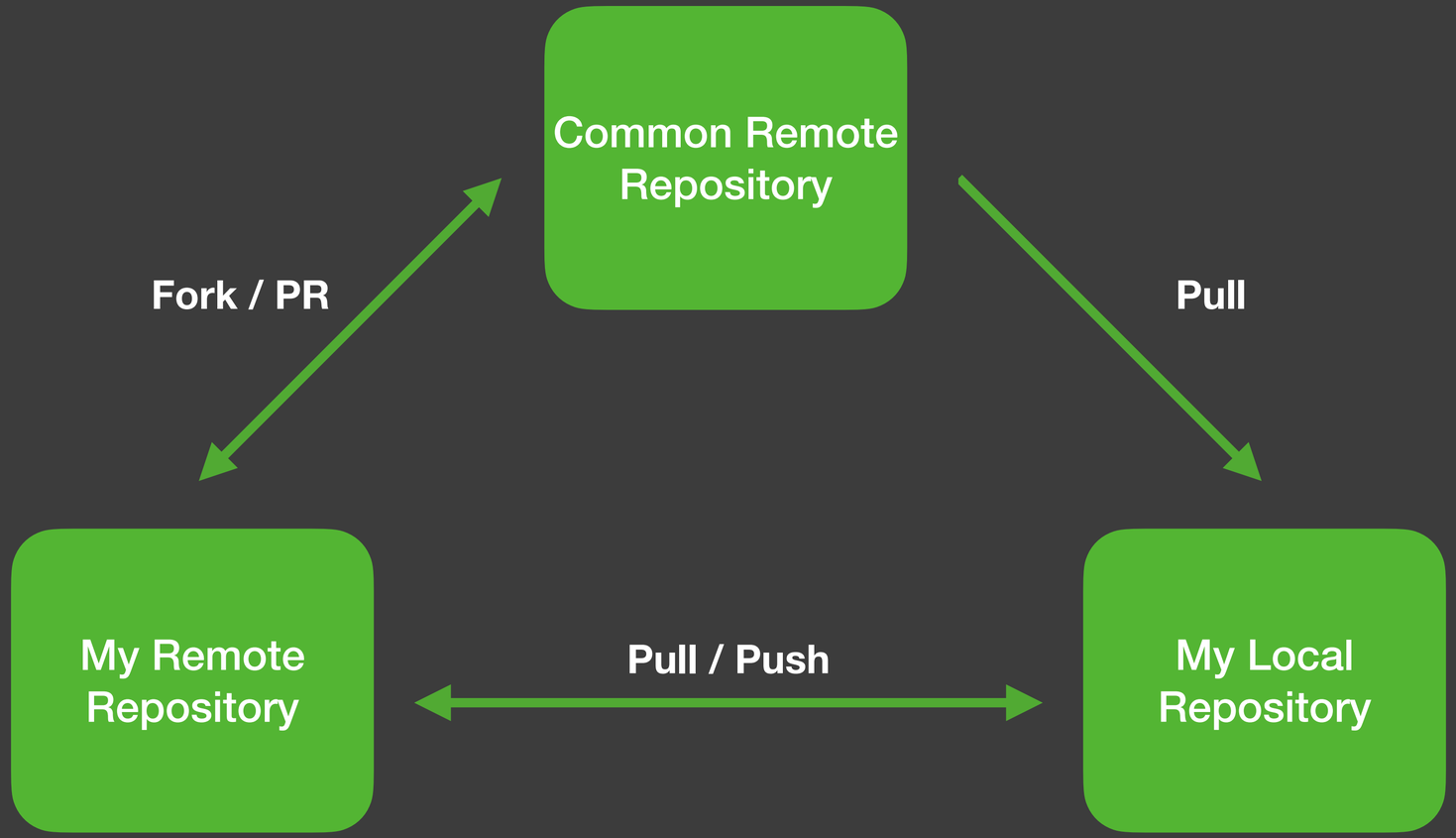
Git and Github

Online Code Review

Abstract

- Example structure
  - Common remote + my remote + my local
  - PR/Branch rule
  - How it works together
- Code Review
- Github

Example Structure



Example Structure

- Common remote repository
  - 프로젝트의 메인 저장소
  - PR은 이 저장소를 기준으로 생성
  - 관리 규칙이 까다로움
    - force push 금지
    - 플러그인 설치 금지 등

Example Structure

- My remote repository
  - 내 작업물이 저장되는 원격 저장소
  - 여러 군데서 작업할 경우, local git들을 싱크할 목적으로 사용
  - PR을 위한 브랜치 생성
  - 테스트 환경이 Common remote repository와 동일

Example Structure

- My local repository
  - 실제로 내가 작업하는 장소
  - 브랜치와 커밋들을 My remote repository에 저장함
  - 작업하는 공간이 여러 곳일 경우, 여러 개의 local repository가 존재할 수 있음
  - Stash/unstash등을 사용해 작성 중인 내용을 저장할 수 있음

PR/Branch

- Feature를 위한 별도 브랜치 사용
  - Main/master 브랜치를 사용하지 않도록 주의
  - Commit hash가 바뀔 경우, conflict이 발생할 수 있음
- PR을 만든 이후에 코드를 변형할 때는 그 브랜치를 계속 사용
  - 해당 브랜치에 push하면, 최신 버전으로 자동 갱신
  - PR이 머지된 후에도 해당 브랜치를 보존 가능

Lab

Lab

- SourceTree
  - GitHub 계정 설정
    - Personal access tokens
    - https://github.com/settings/tokens
  - 원격 저장소 추가
    - origin은 이미 있음
    - common으로 추가
  - Fetch로 정상적으로 저장소를 읽어오는 확인

Abstract

- Example structure
- Code Review
  - What it is
  - Principle
- Github

Code Review

“Code review (sometimes referred to as peer review) is a software quality assurance activity in which one or several people check a program mainly by viewing and reading parts of its source code, and they do so after implementation or as an interruption of implementation. At least one of the persons must not be the code's author. The persons performing the checking, excluding the author, are called ‘reviewers’.”

https://en.wikipedia.org/wiki/Code_review

Code Review

- Better Code quality
  - Internal code quality
  - Maintainability
    - Readability, uniformity, understandability
- Finding defects
  - External aspects, especially correctness
  - Performance problems
  - Security vulnerabilities, injected malware

https://en.wikipedia.org/wiki/Code_review

Code Review

- Learning/knowledge transfer
  - Codebase
  - Solution approaches
  - Expectations regarding quality
  - **Reviewers as well as to the author**

https://en.wikipedia.org/wiki/Code_review

Code Review

- Incase sense of mutual responsibility
  - Collective code ownership and solidarity
- Finding better solutions
  - New and better solutions and ideas
    - Specific code at hand
- Complying to QA guidelines, ISO/IEC standards
  - Code reviews are mandatory in some contexts
  - Air traffic software, safety-critical software

https://en.wikipedia.org/wiki/Code_review

Code Review

- **Design**: Is the code well-designed and appropriate for your system?
- **Functionality**: Does the code behave as the author likely intended? Is the way the code behaves good for its users?
- **Complexity**: Could the code be made simpler? Would another developer be able to easily understand and use this code when they come across it in the future?
- **Tests**: Does the code have correct and well-designed automated tests?
- **Naming**: Did the developer choose clear names for variables, classes, methods, etc.?
- **Comments**: Are the comments clear and useful?
- **Style**: Does the code follow our style guides?
- **Documentation**: Did the developer also update relevant documentation?

https://google.github.io/eng-practices/review/

Code Review

The pages in this section contain best practices for developers going through code review. These guidelines should help you get through reviews faster and with higher-quality results. You don't have to read them all, but they are intended to apply to every Google developer, and many people have found it helpful to read the whole set.

- Writing Good CL Descriptions
- Small CLs
- How to Handle Reviewer Comments

See also How to Do a Code Review, which gives detailed guidance for code reviewers.

https://google.github.io/eng-practices/review/developer/

Code Review

뱅크샐러드의 코드 리뷰에는 기술 조직이 일하는 방식, 문화가 녹아 있습니다.

타협하지 않고 순리를 추구하는 문화

저 문맥 커뮤니케이션 지향하는 문화

커뮤니케이션 비용을 비싼 비용이라고 여기는 문화

Blocker 가 되는 것을 부끄럽게 생각하는 문화

비동기 커뮤니케이션을 지향하는 문화

https://blog.banksalad.com/tech/banksalad-code-review-culture/

Code Review

결과적으로 현재 상태

- 매일 아침 9시 30분(월요일은 오후 1시)에 d-n 라벨을 수정 후 슬랙으로 알람 발송
- MR 단위는 최대한 작게
- 일관된 양식으로 작성된 MR
- 리뷰가 쌓이면 업무 중단 후 리뷰 먼저 처리

현재는 이렇게 일하기 때문에 '쌓이는 리뷰'가 많이 줄었습니다. 또한 MR은 많지만 바쁠 때는 중요한 MR을 먼저 리뷰하기 때문에 효율적으로 일할 수 있게 되었습니다.

https://techblog.woowahan.com/7152/

Code Review

효과적인 코드 리뷰를 위한 팁들

가장 효과적으로 코드 리뷰할 수 있는 방법은 페어 프로그래밍입니다만, 팀의 성향에 따라 GitHub에서 PR(Pull Request)를 이용하는 것도 좋은 방법입니다. 코드 리뷰를 '제대로' 하려면, 우선 코드 리뷰 프로세스를 효율적으로 만들어야 합니다. 한 가지 아이디어는 리뷰어를 희귀한 자원으로 다루는 것입니다. 우리 중 누구도 코드 리뷰를 주 업무로 하고 있진 않다는 것이 그 이유입니다.

그리고 여기 코드 리뷰를 효과적이고 효율적으로 하기 위한 몇 가지 팁이 더 있습니다.

- 변화를 작게 유지하자
- 리뷰는 자주 짧은 세션으로 진행하자
- 리뷰를 위해 최대한 빨리 PR을 보내자
- 의미있는 PR을 만들기에 충분한 정보를 제공하자
- 코드 분석 툴을 활용하고 코드 스타일을 확인하자

https://engineering.linecorp.com/ko/blog/effective-codereview/#team-based-code-coverage

Code Review

우리팀은..?

현재 아이랩팀은 현재 GitHub에서 Master Branch만 사용중이다.

그 후 로컬환경에서 코드를 작성 Merge후 바로 커밋후 저장소에 Push를 진행하고있다.

앞으로 Pull Request 템플릿을 활용해 필요한 작업은 반드시 코드리뷰 후 Master 브랜치에 Merge를 진행할 예정이다.

- develop 브랜치 생성(Develop에 푸쉬 시, 반드시 Reviewer 한명이상 지정)
- 각자 브랜치를 생성하여 작업
- develop 브랜치에 머지 진행전 PR(Pull Request)를 통한 코드리뷰
- develop에 모두 merge된 소스를 배포시 master에 병합하여 어플리케이션 버전업

https://techwell.wooritech.com/blog/2021/04/19/%EC%BD%94%EB%93%9C%EB%A6%AC%EB%B7%B0/#%EB%A6%AC%EB%B7%B0%EB%8A%94-%EB%8B%B9%EC%97%B0%ED%95%98%EB%8B%A4-%EB%84%A4%EC%B9%B4%EB%9D%BC%EC%BF%A0%EB%B0%B0%EC%99%80-%EC%8A%A4%ED%83%80%ED%8A%B8%EC%97%85

Code Review

- 리뷰를 쉽게 만들기
  - 코드를 **최대한** 작게 만들기
  - 테스트 코드를 **충실히** 작성하기
  - 설명을 **열심히** 쓰기
  - 로직에만 집중할 수 있도록 만들기
    - **스타일 지키기**: 코드 스타일 적용하기
    - **변수 이름** 등을 적절히 짓기
    - https://github.com/google/google-java-format

Abstract

- Example structure
- Code Review
- Github
  - Conflict example
  - Merge strategies
    - Create a merge commit
    - Squash and merge
    - Rebase and merge
  - Actions
    - Test
    - Codecov

Conflict Example

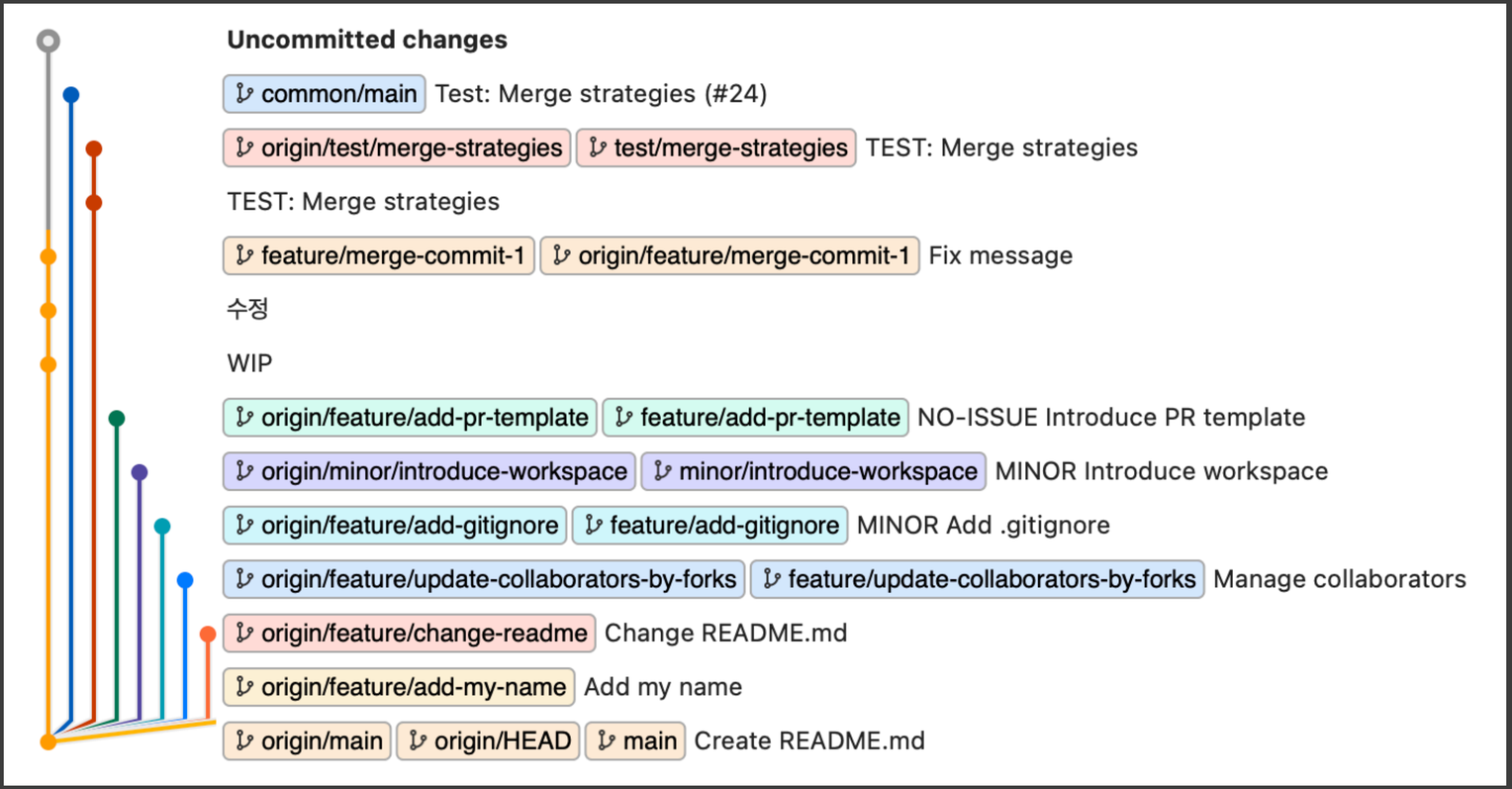
- https://github.com/jongyoul/cnu-lab/pull/18
- https://github.com/jongyoul/cnu-lab/pull/19

Lab

Lab

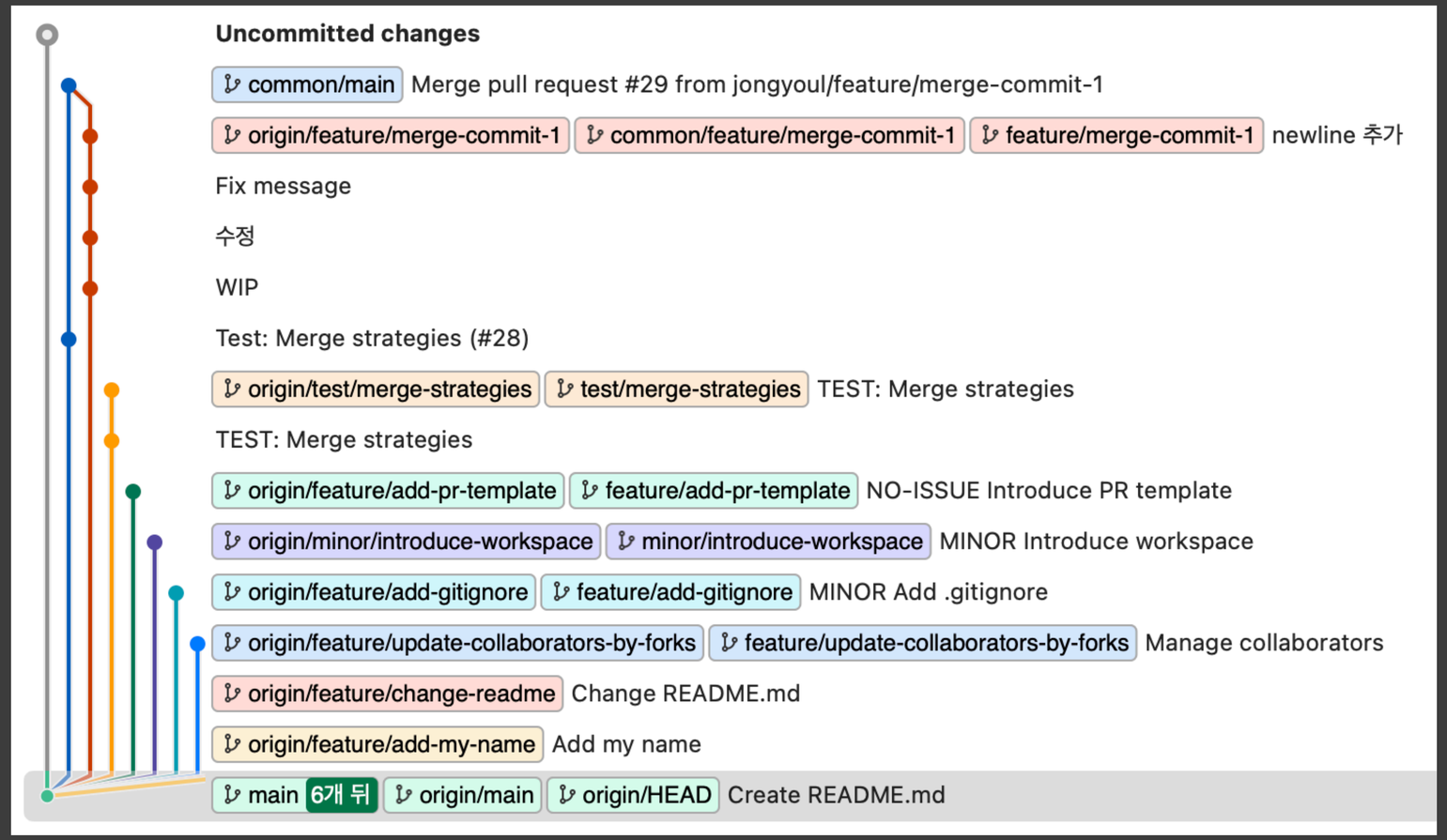
- Source Tree
  - 재배치(Rebase)
- IntelliJ
  - VCS 작업 -> 충돌 해결
- Source Tree
  - 재배치(Rebase) 계속

Create a Merge Commit



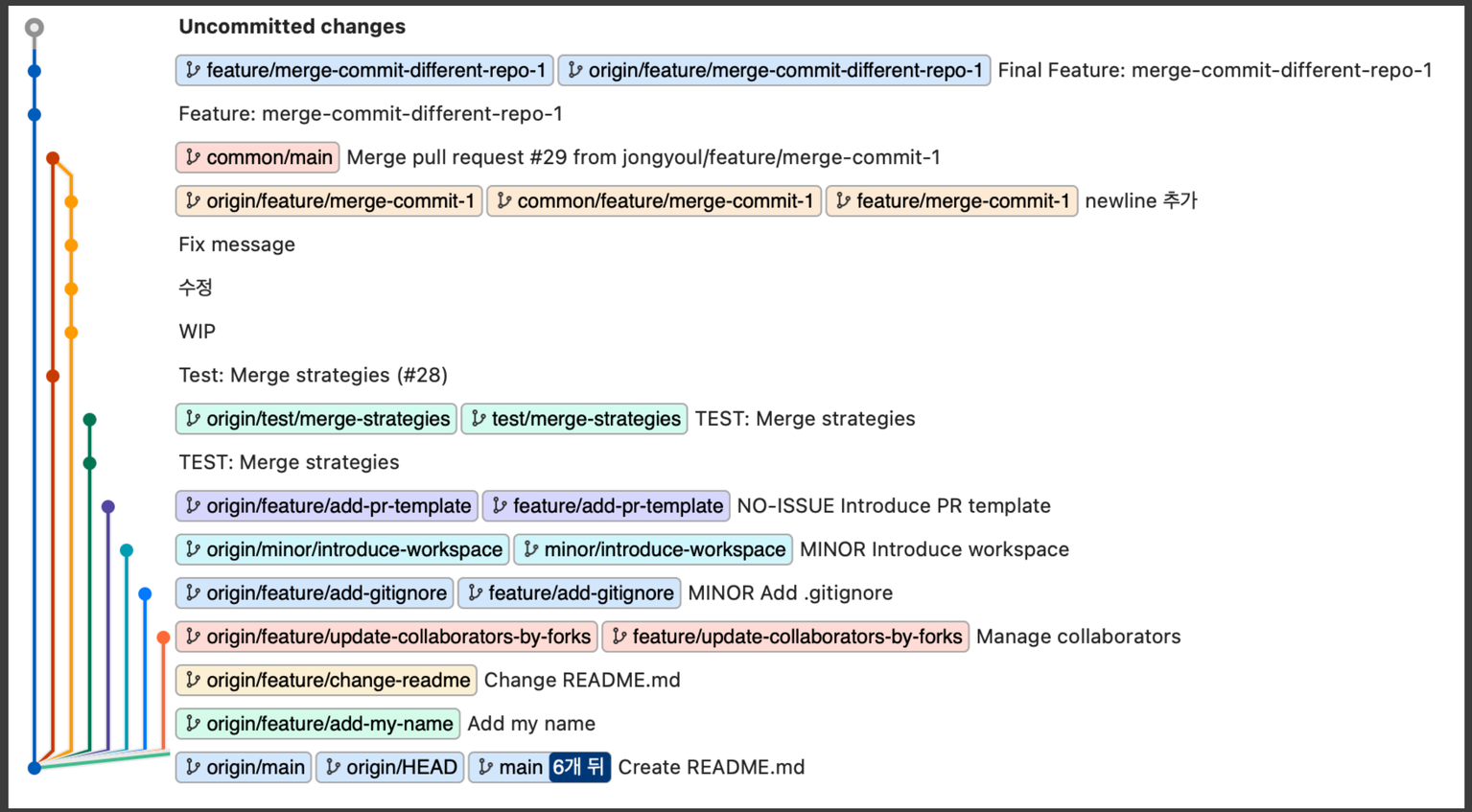
https://github.com/jongyoul/cnu-lab/pull/29

Create a Merge Commit



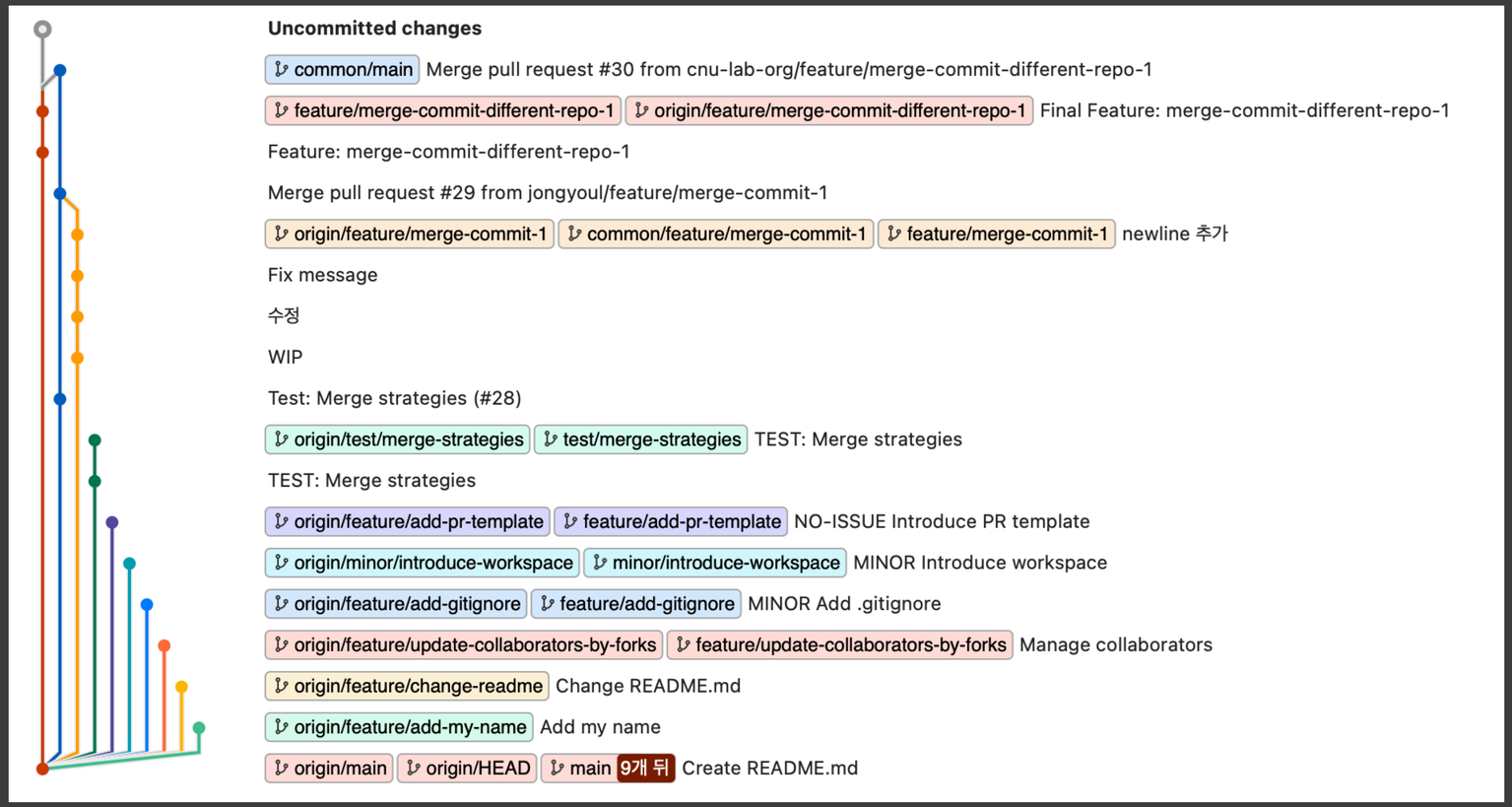
https://github.com/jongyoul/cnu-lab/pull/29

Create a Merge Commit



https://github.com/jongyoul/cnu-lab/pull/30

Create a Merge Commit

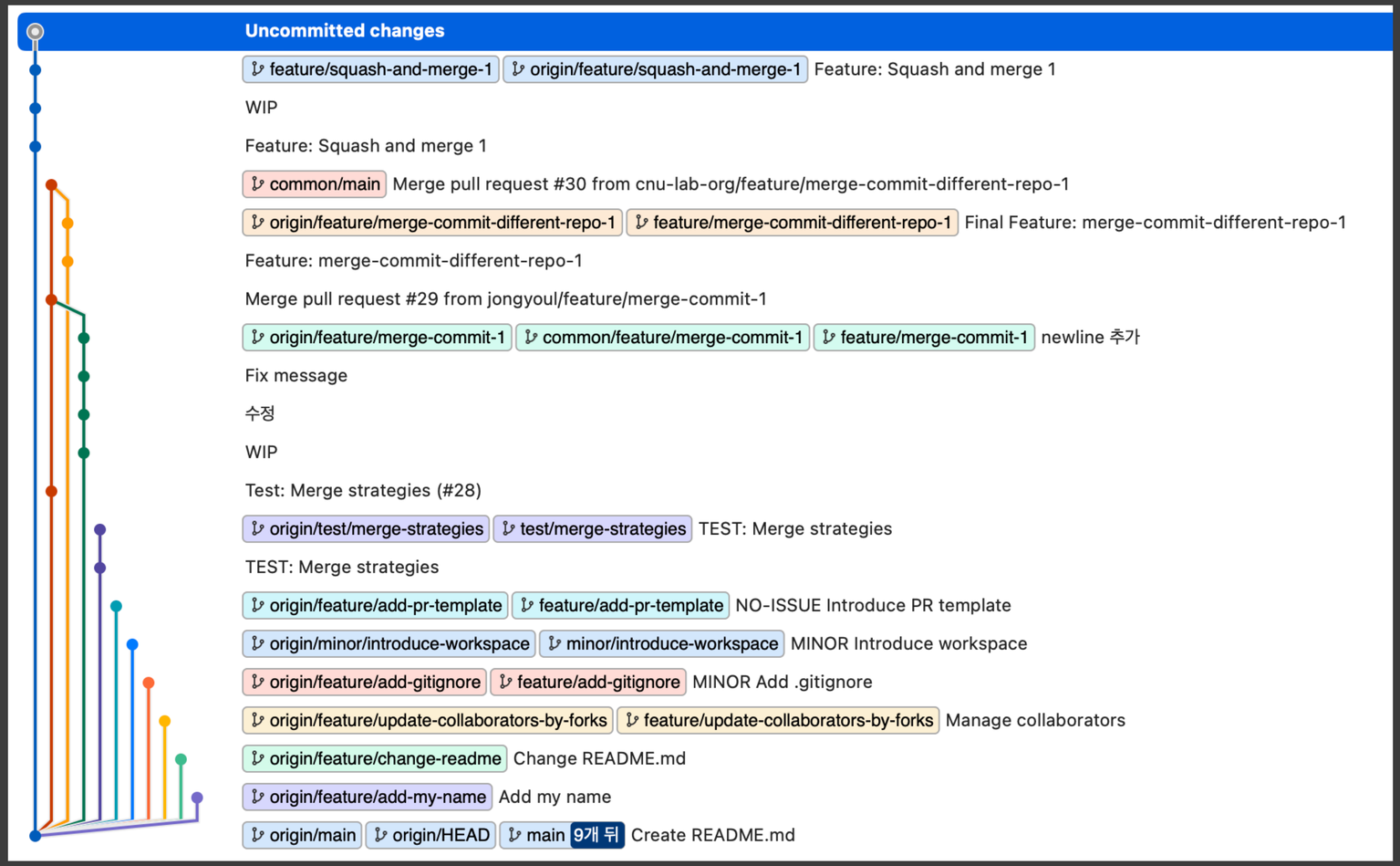


https://github.com/jongyoul/cnu-lab/pull/30

Create Merge Commit

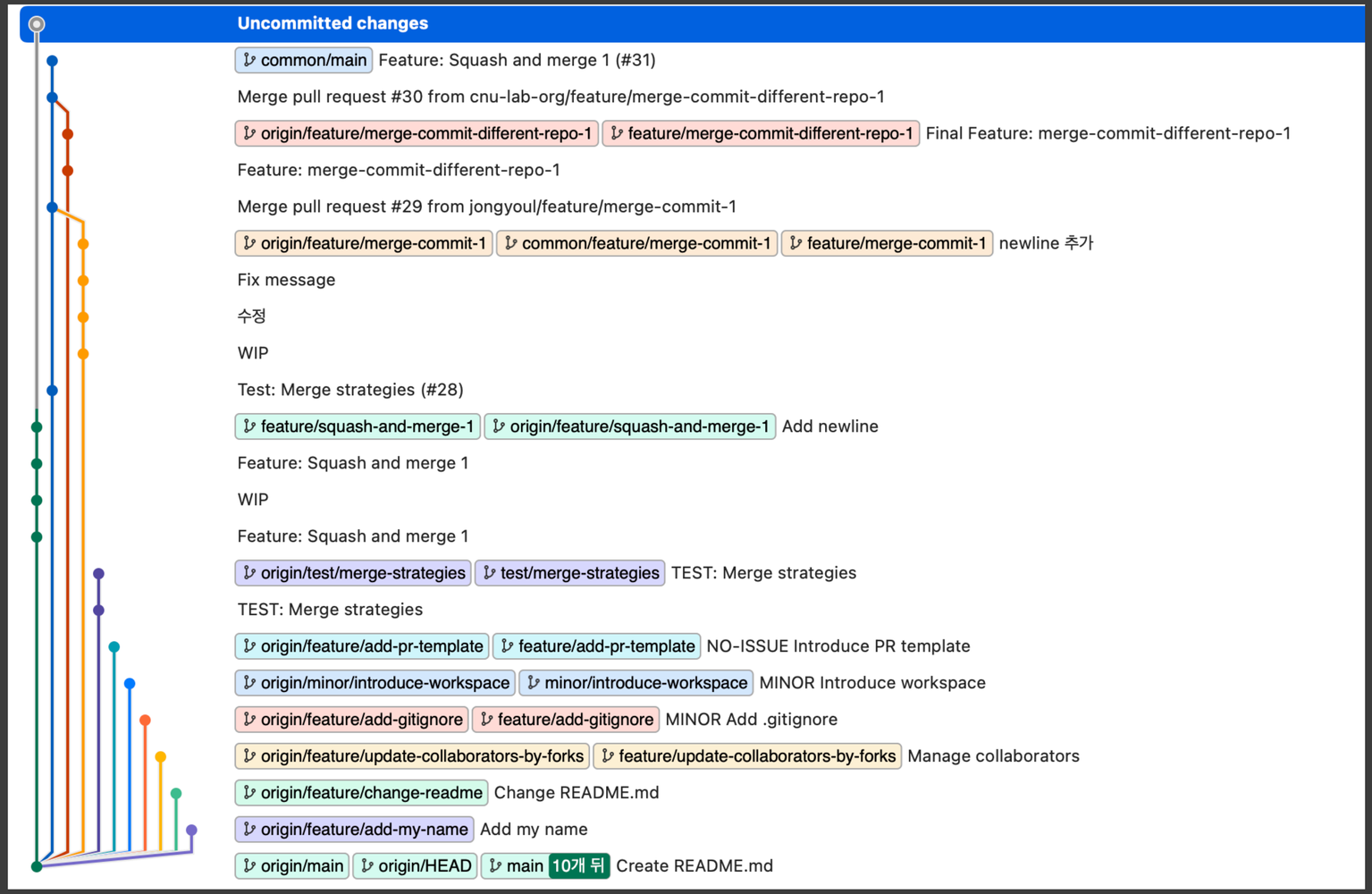
- 장점
  - 모든 커밋이 온전하게 보존됨
- 단점
  - 불필요한 커밋들도 다른 사람들에게 싱크됨
  - 전체적인 repository사이즈가 커짐

Squash and Merge



https://github.com/jongyoul/cnu-lab/pull/31

Squash and Merge

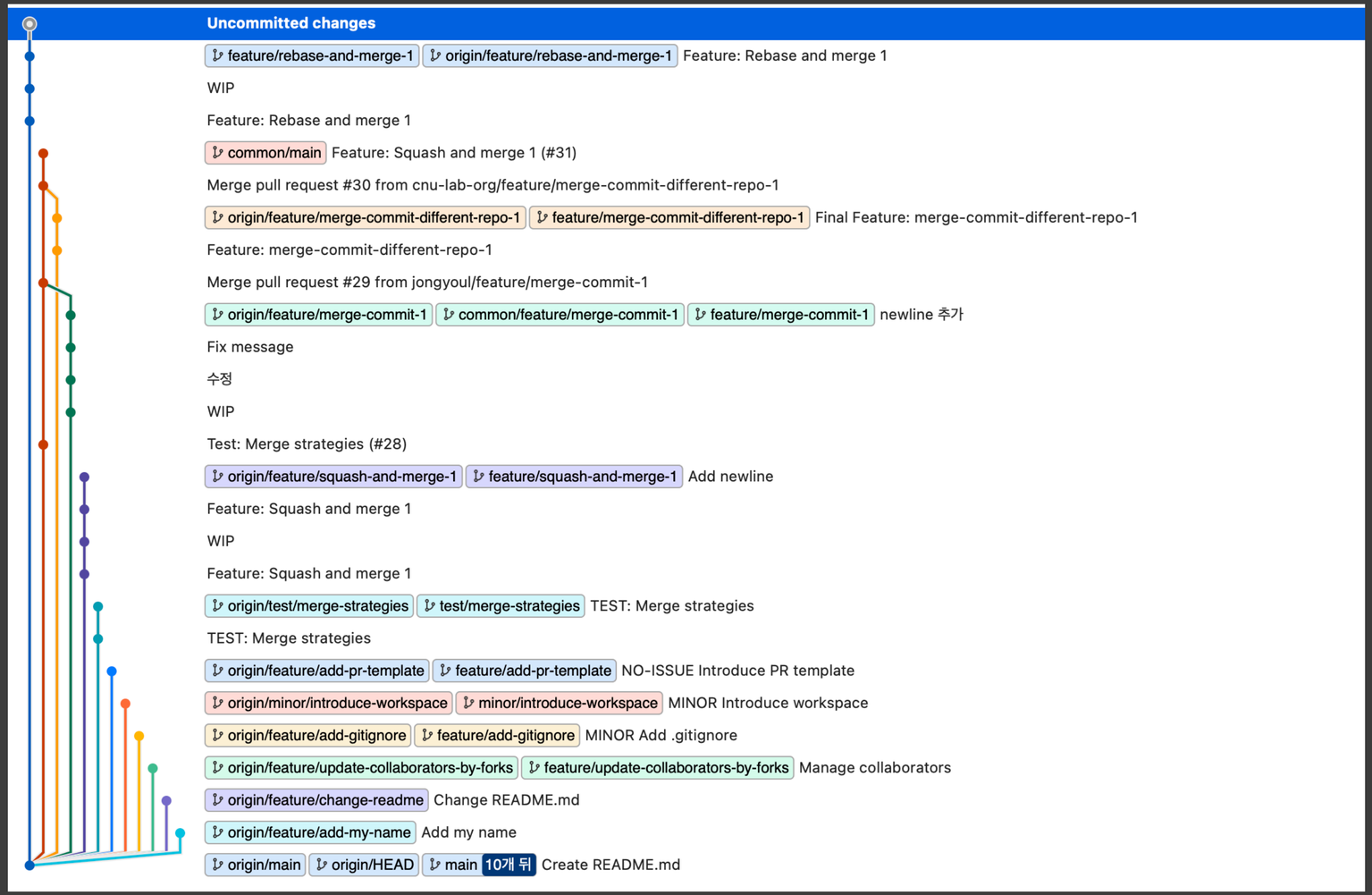


https://github.com/jongyoul/cnu-lab/pull/31

Squash and Merge

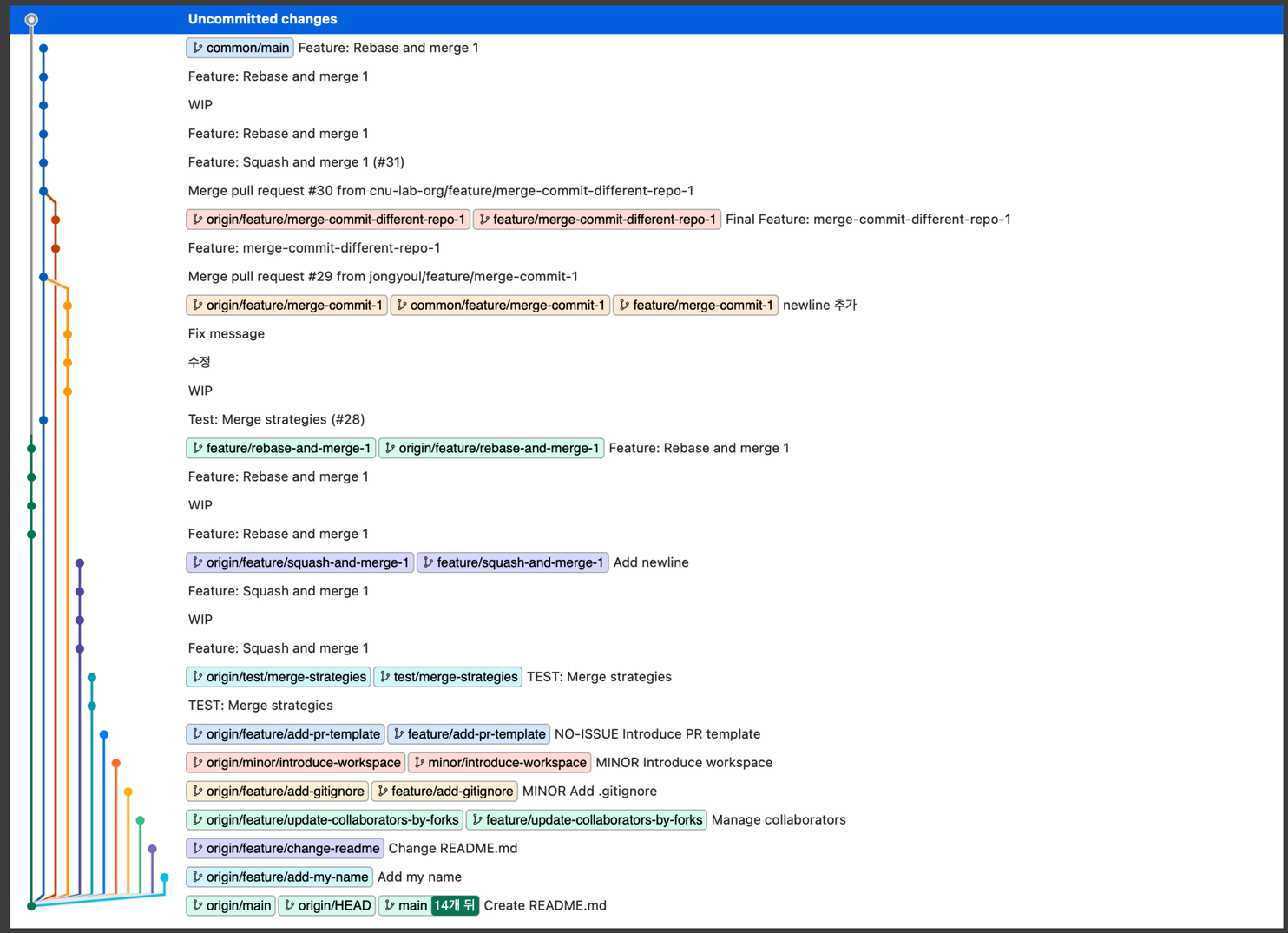
- 장점
  - Git tree의 사이즈가 작고 깔끔하게 유지됨
- 단점
  - 작업 내용 없이 결과만 저장됨
  - 중간 과정은 알아서 저장해야함

Rebase and Merge



https://github.com/jongyoul/cnu-lab/pull/32

Rebase and Merge



https://github.com/jongyoul/cnu-lab/pull/32

Rebase and Merge

- 장점
  - 전체 커밋 히스토리가 유지됨
  - Git tree를 선형으로 유지할 수 있음
- 단점
  - Repository의 사이즈가 커짐

Github Actions

- Github repository가 변경되면, trigger되는 작업들
- .github/workflows/*.yml
- https://github.com/jongyoul/cnu-lab/actions/new
- CI/CD를 지원하기 위한 Github에서 제공하는 방식
  - 테스트, 배포 전부 가능
- 기존에는 webhooks등을 제공했음

Test

```yaml
name: Gradle test
on: [pull_request, push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: '11'
          distribution: 'adopt'
      - name: Validate Gradle wrapper
        uses: gradle/wrapper-validation-action@e6e38bacfdf1a337459f332974bb2327a31aaf4b
      - name: Build with Gradle
        uses: gradle/gradle-build-action@937999e9cc2425eddc7fd62d1053baf041147db7
        with:
          arguments: test
```

https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-gradle

Lab

Lab

- https://github.com/jongyoul/github-action-test 포크
- .github/workflows/test.yml 파일 생성
- yml파일 내용 복사
- Master/main에 커밋
- Actions탭에서 동작 확인

Takeaways

- Example structure
  - Common remote + my remote + my local
  - PR/Branch rule
  - How it works together
- Code Review
  - What it is
  - Principle
- Github
  - Merge strategies
    - Create a merge commit
    - Squash and merge
    - Rebase and merge
  - Actions
    - Test

Q & A