

Git

Git + Github

Git 왜 씬?

- 파일 변경내역 보존하고 관리
- 코드 수정하다가 개망해서 2일 전으로 돌아가고 싶음
- 같이 일하는 사람들에게 수정사항 쉽게 공유(협업)

-> Version Control System(VCS)

버전 관리 시스템

수정본 무한생성 멈춰!

졸업논문수정.hwp
졸업논문수정1.hwp
졸업논문수정2.hwp
완성본.hwp
본1.hwp
본2.hwp
완성본.hwp
최종완성본1.hwp
졸업논문최종완성본2.hwp
졸업논문최종완성본final.hwp
졸업논문최종완성본final1.hwp
졸업논문최종완성본final2.hwp
졸업논문최종완성본final최종.hwp
졸업논문최종완성본final최종1.hwp
졸업논문최종완성본final최종2.hwp

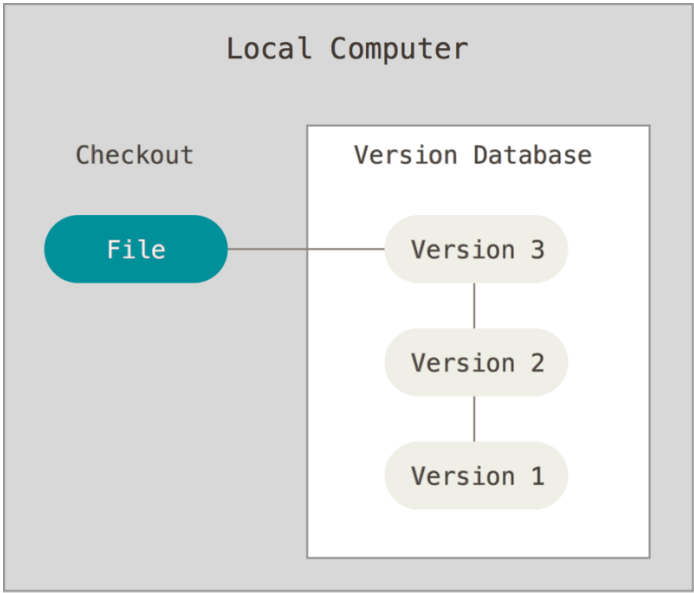
Git 왜 씀?

Version Control System(VCS)

버전 관리 시스템

Local VCS

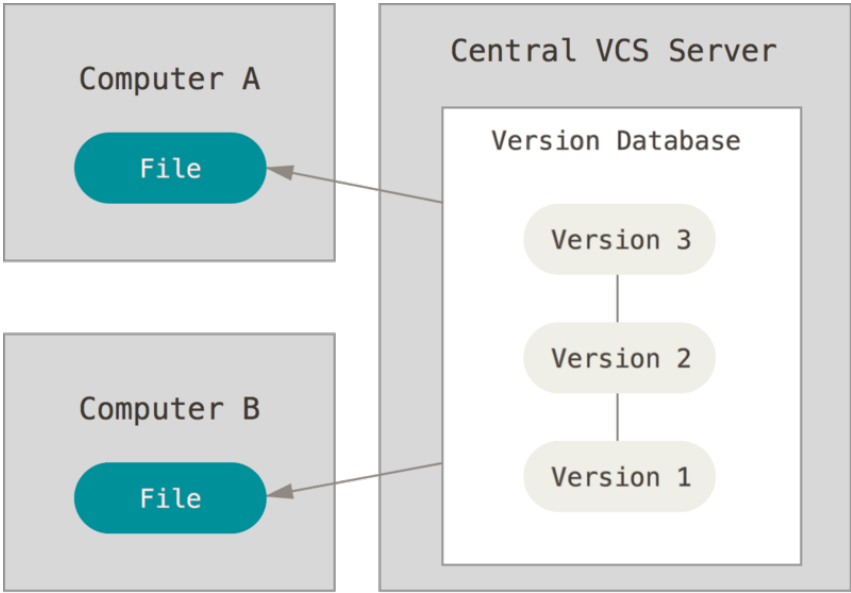
로컬 버전관리



RCS(Revision control system)

CVCS

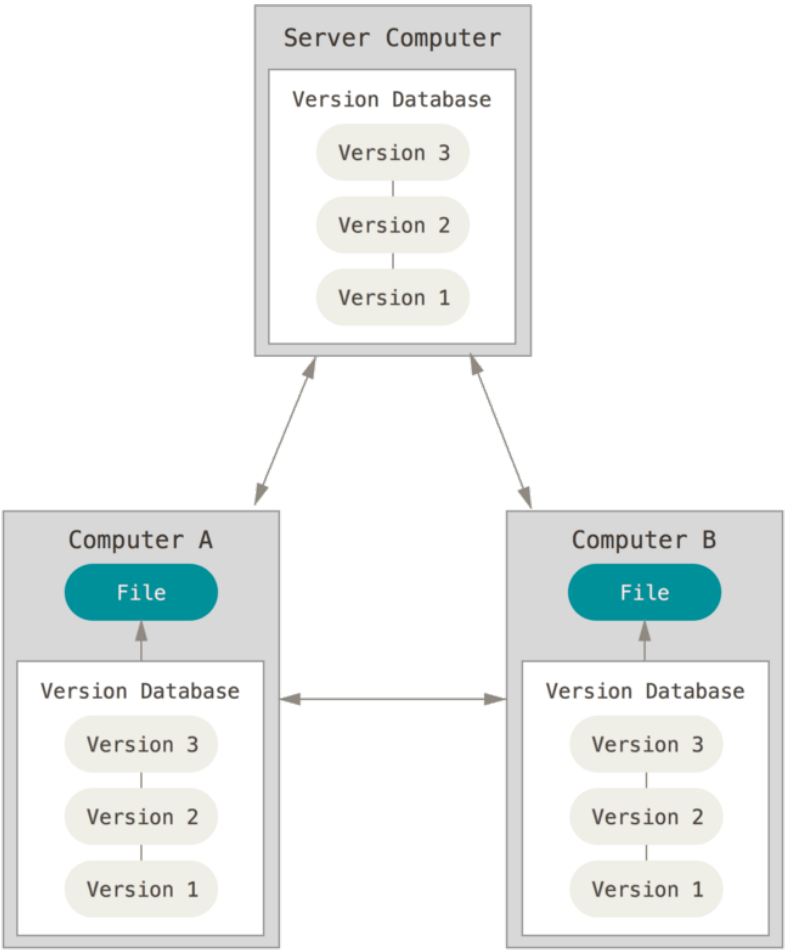
중앙집중식 버전관리



CVS Subversion Perforce

DVCS

분산 버전관리 시스템



Git Mercurial Bazaar Darcs

Git 목차

- git 설치와 설정
- 작업폴더에서 git 이용하기 `git init`
- commit 전에 차이점 확인하기 `git difftool`
- 프로젝트 복사본(branch) 만들기 `git branch`
- branch 합치기 `git merge`
- 파일 복구하기 `git restore`

git 설치와 설정

설치

<https://git-scm.com/book/ko/v2/%EC%8B%9C%EC%9E%91%ED%95%98%EA%B8%B0-Git-%EC%84%A4%EC%B9%98>

`git --version` -> 버전 정보 뜨면 성공!

git 설치와 설정

설정 (git config)

```
git config --global user.name "younyoung"
```

```
git config --global user.email "00younyoung@naver.com"
```

: 누가 깃 쓰고 있는지 보여주기 위한 아이디 등록

```
git config --global init.defaultBranch main
```

: 기본 브랜치 이름을 main으로

```
git config --global core.editor "code --wait"
```

: 기본 에디터를 VSCode로

git 설치와 설정

설정 (git config)

```
git config --global diff.tool vscode
```

```
git config --global difftool.vscode.cmd 'code --wait --diff $LOCAL $REMOTE'
```

```
git config --global difftool.prompt false
```

: difftool 기본 에디터를 VSCode로 (이따 할 거)

```
git config --global -e
```

: 그 외에도 다양한 설정값들 확인 및 수정 가능

<https://git-scm.com/book/ko/v2/%EC%8B%9C%EC%9E%91%ED%95%98%EA%B8%B0-Git-%EC%B5%9C%EC%B4%88-%EC%84%A4%EC%A0%95>

git 설치와 설정

설정 (git config)

우선순위

저장위치: [로컬레포]\.git\config

```
git config --local user.name "younyoung"
```

1

저장위치: C:\Users\[사용자계정]\.gitconfig

```
git config --global user.name "younyoung"
```

2

저장위치: [설치폴더]\etc\gitconfig

```
git config --system user.name "younyoung"
```

3

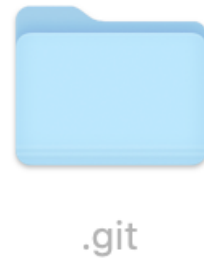
작업폴더에서 git 이용하기

git init, git add, git commit

1. 폴더 만들고 그 폴더에서 터미널 열기

2. repository(저장소) 생성

```
git init
```



3. commit할 파일 골라두기 : staging

```
git add 파일명
```

```
git add 파일명1 파일명2
```

```
git add .
```

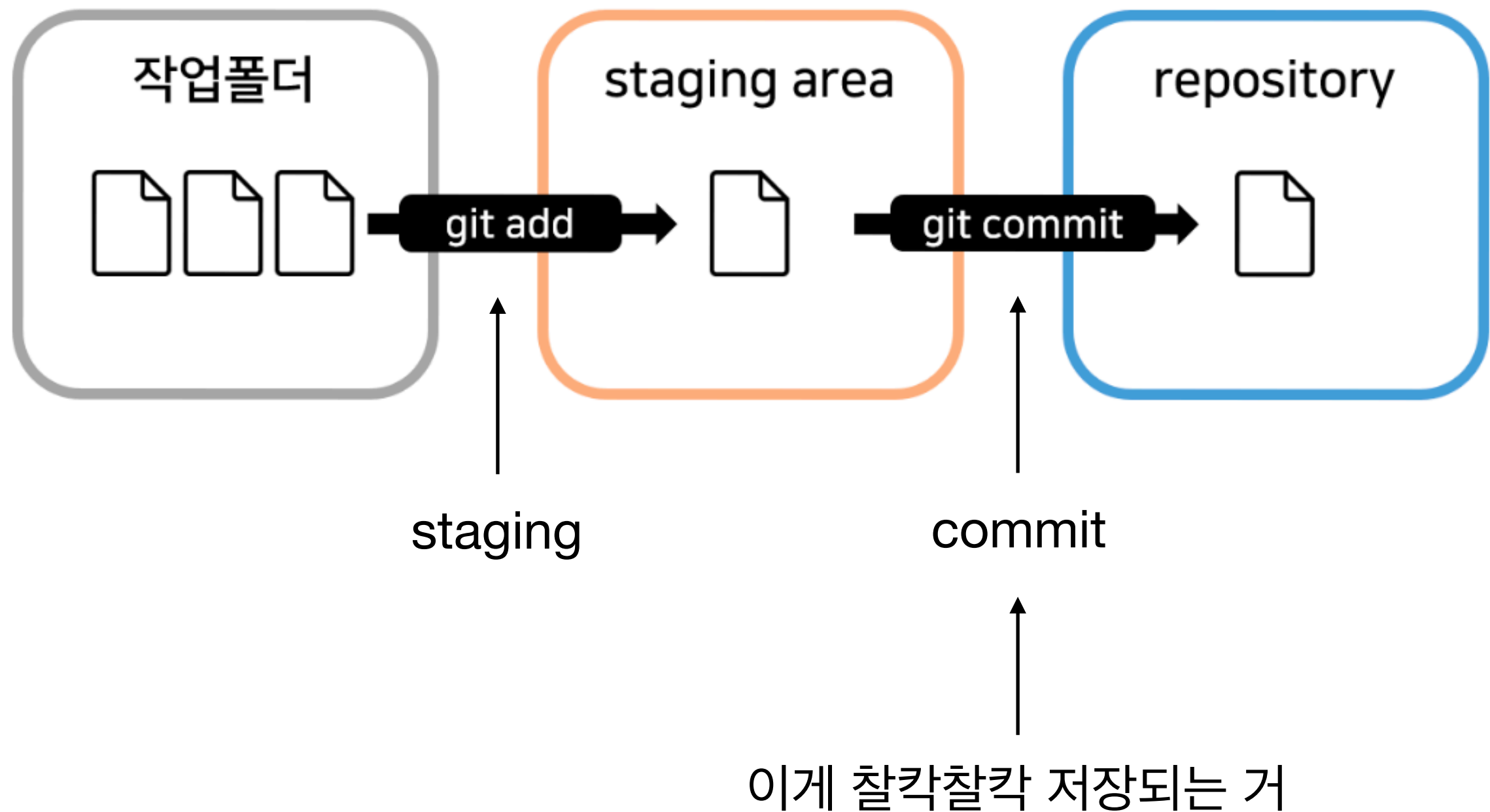
4. 저장소에 commit 하기

```
git commit -m '커밋메시지'
```

commit:
a snapshot of your
files(code)

작업폴더에서 git 이용하기

staging area, repository



작업폴더에서 git 이용하기

git status, git log

```
git status
```

: 상태를 보여줘!!

```
git restore --staged 파일명
```

: 파일의 staging 취소!!

```
git log --oneline
```

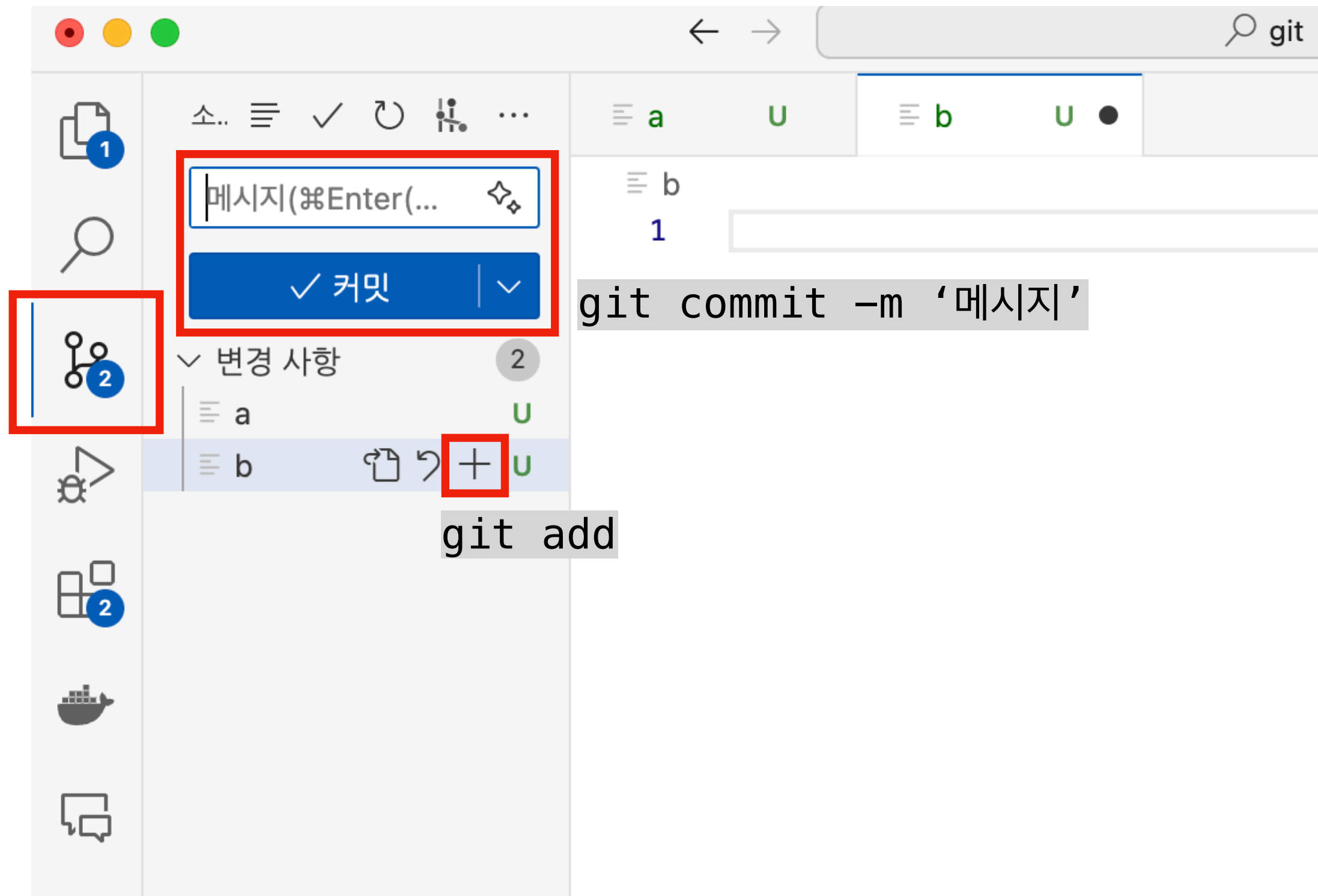
: 커밋 로그 보여줘!!

```
git log --oneline --graph
```

: 커밋 로그 그래프로 예쁘게 보여줘!!

작업폴더에서 git 이용하기

VS Code로 더 쉽게



commit 전에 차이점 확인하기

git difftool

`git diff`

: 보기에 지저분

`git difftool`

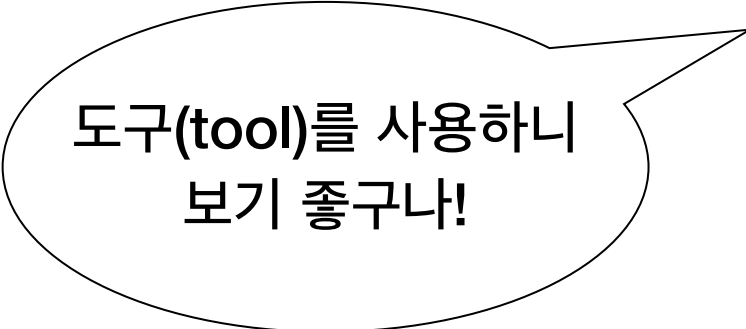
: 현재 파일과 최근 commit의 차이점 확인

`git difftool 커밋id`

: 현재 파일과 특정 commit의 차이점 확인

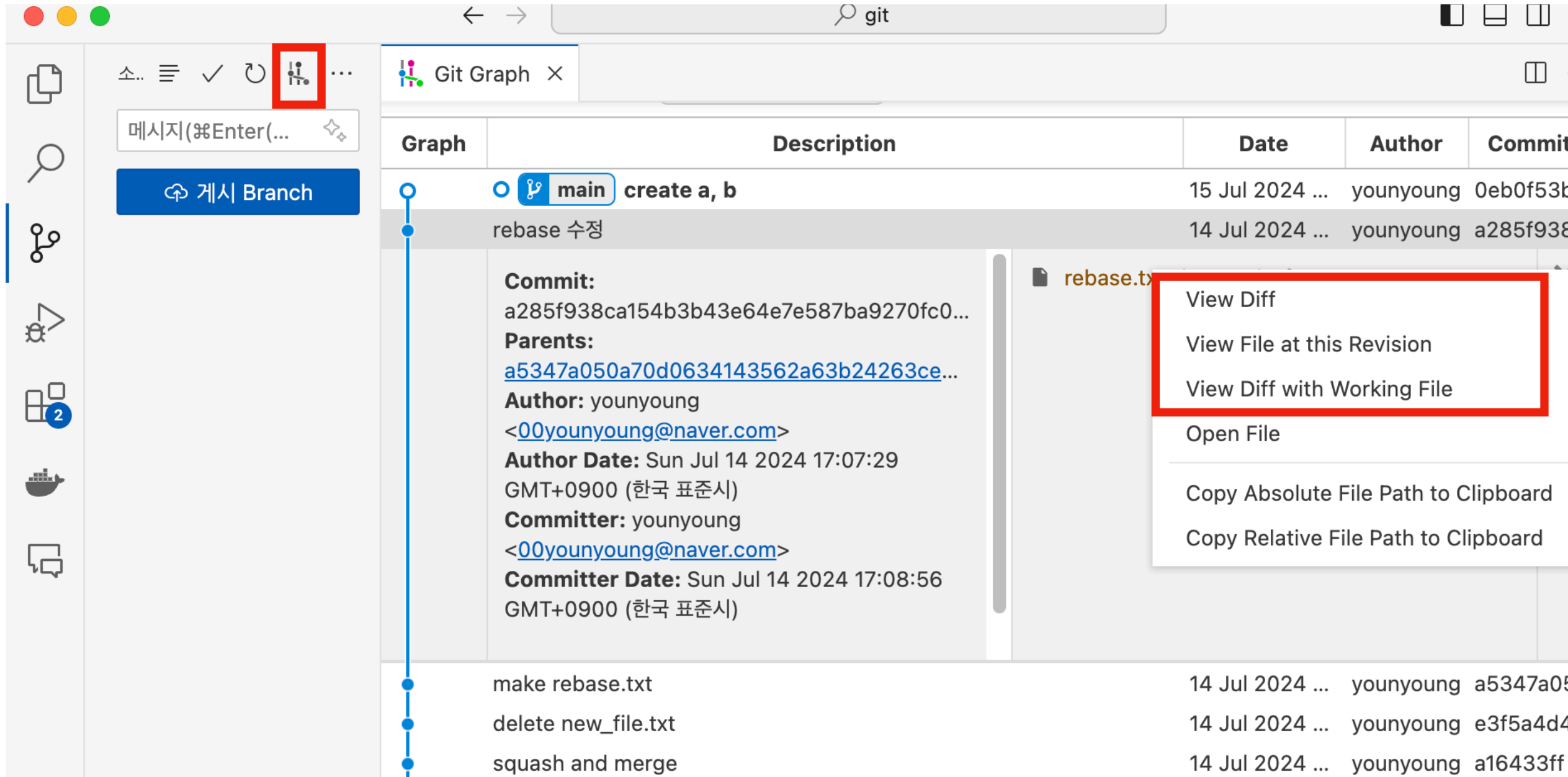
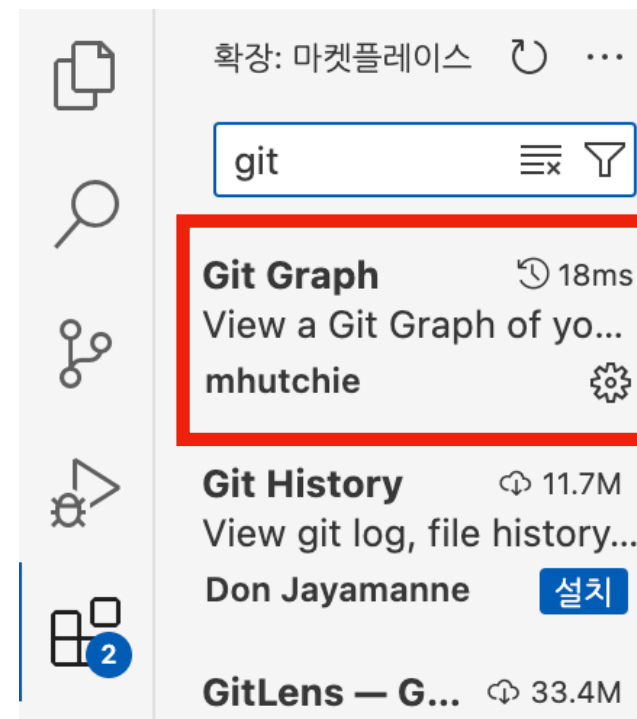
`git difftool 커밋id1 커밋id2`

: 특정 commit 둘의 차이점 확인



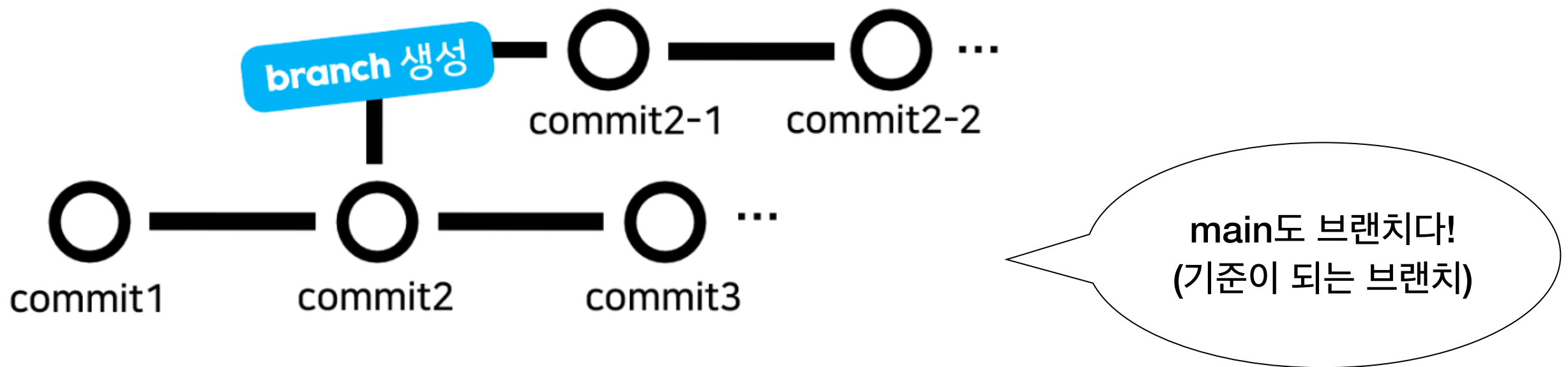
도구(tool)를 사용하니
보기 좋구나!

commit 전에 차이점 확인하기 VS Code로 더 쉽게



프로젝트 복사본(branch) 만들기

git branch



`git branch 브랜치이름`

: 프로젝트 사본(=branch) 생성

`git switch 브랜치이름`

: 브랜치로 이동

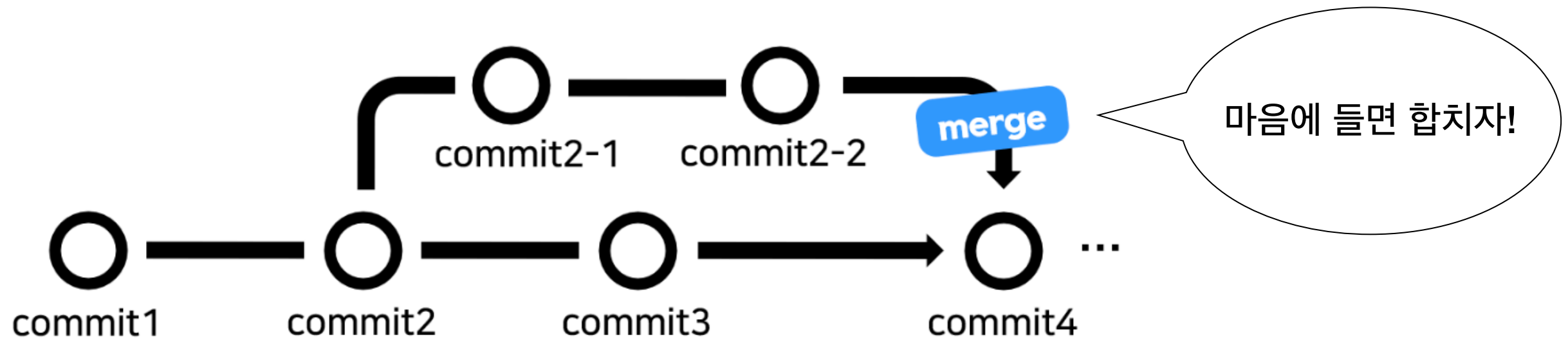
`git switch -c 브랜치이름`

: 브랜치 만들고 이동

(실습) main 브랜치와 새로 만든 브랜치에서 각각 2번 이상 commit 해보기
+ log도 확인해보자

branch 합치기

git merge



1. 기준이 되는 브랜치(main)로 이동

```
git switch main
```

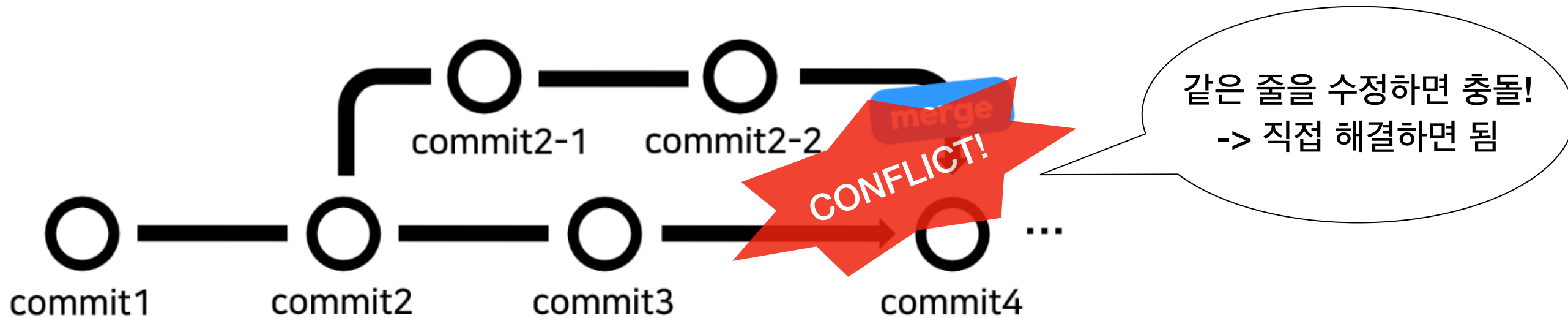
2. 합치기!

```
git merge 브랜치이름
```

그런데...

branch 합치기

conflict 해결하기



1. Conflict 해결 : 코드 보면서 비교해서 수동으로 수정하기

2. 해결한 파일 commit

```
git add 파일명
```

```
git commit -m '커밋메시지'
```

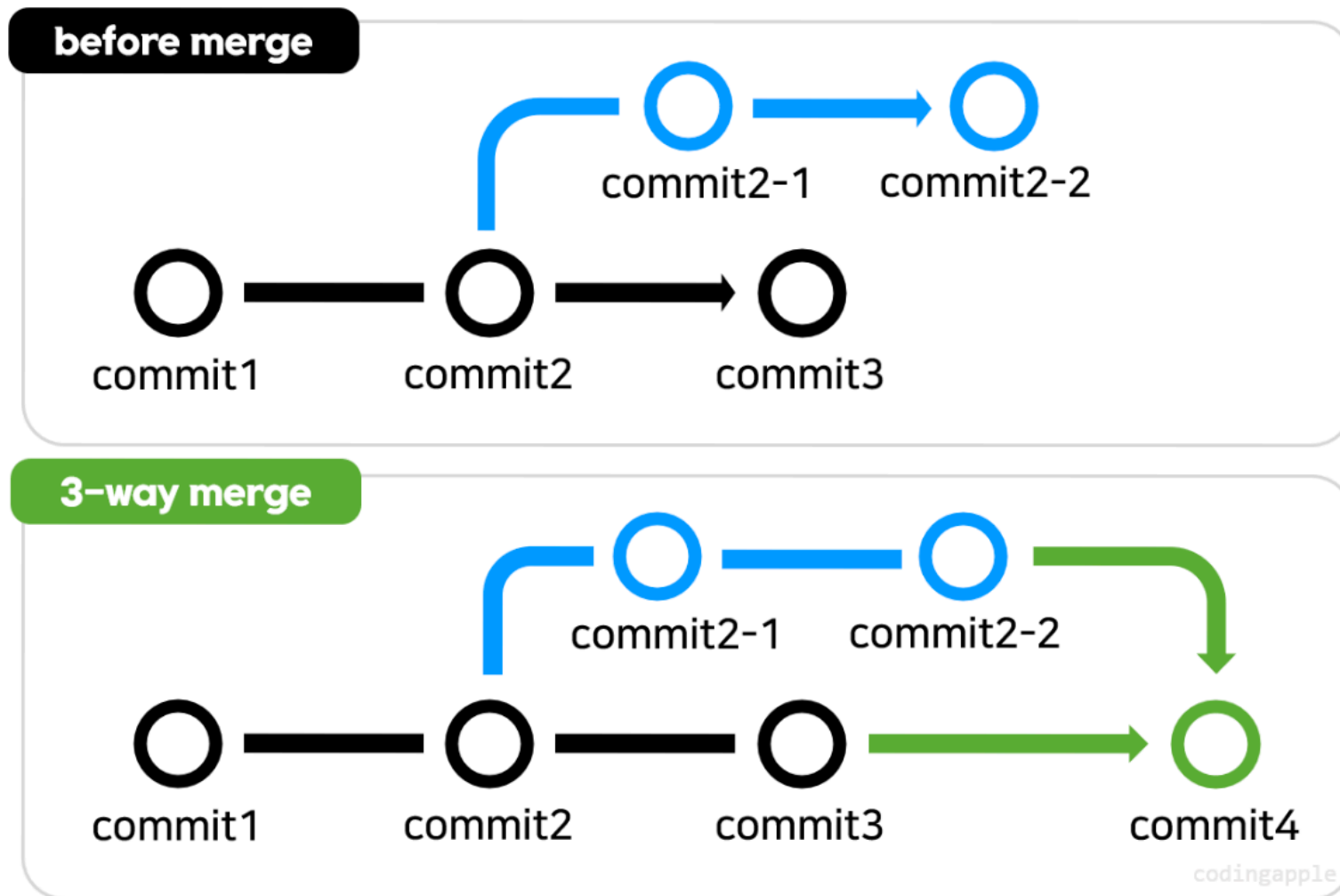
(실습 이어서) merge 해보자

branch 합치기

다양한 merge 방법

3-way merge

: 브랜치에 각각 신규 커밋이 1회 이상 있는 경우, 기본 동작방식

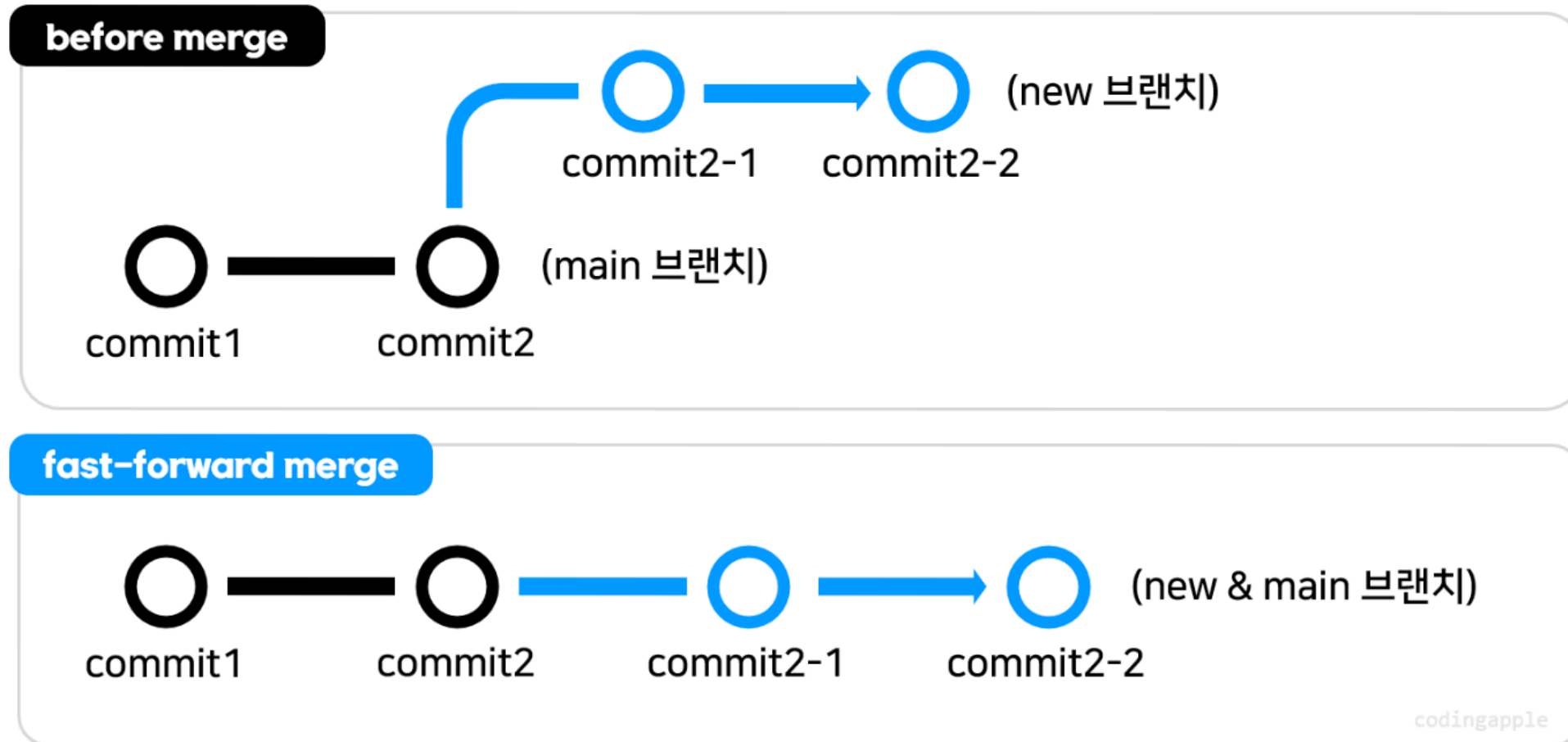


branch 합치기

다양한 merge 방법

fast-forward merge

: 새로운 브랜치에만 커밋이 있는 경우, 기본 동작방식



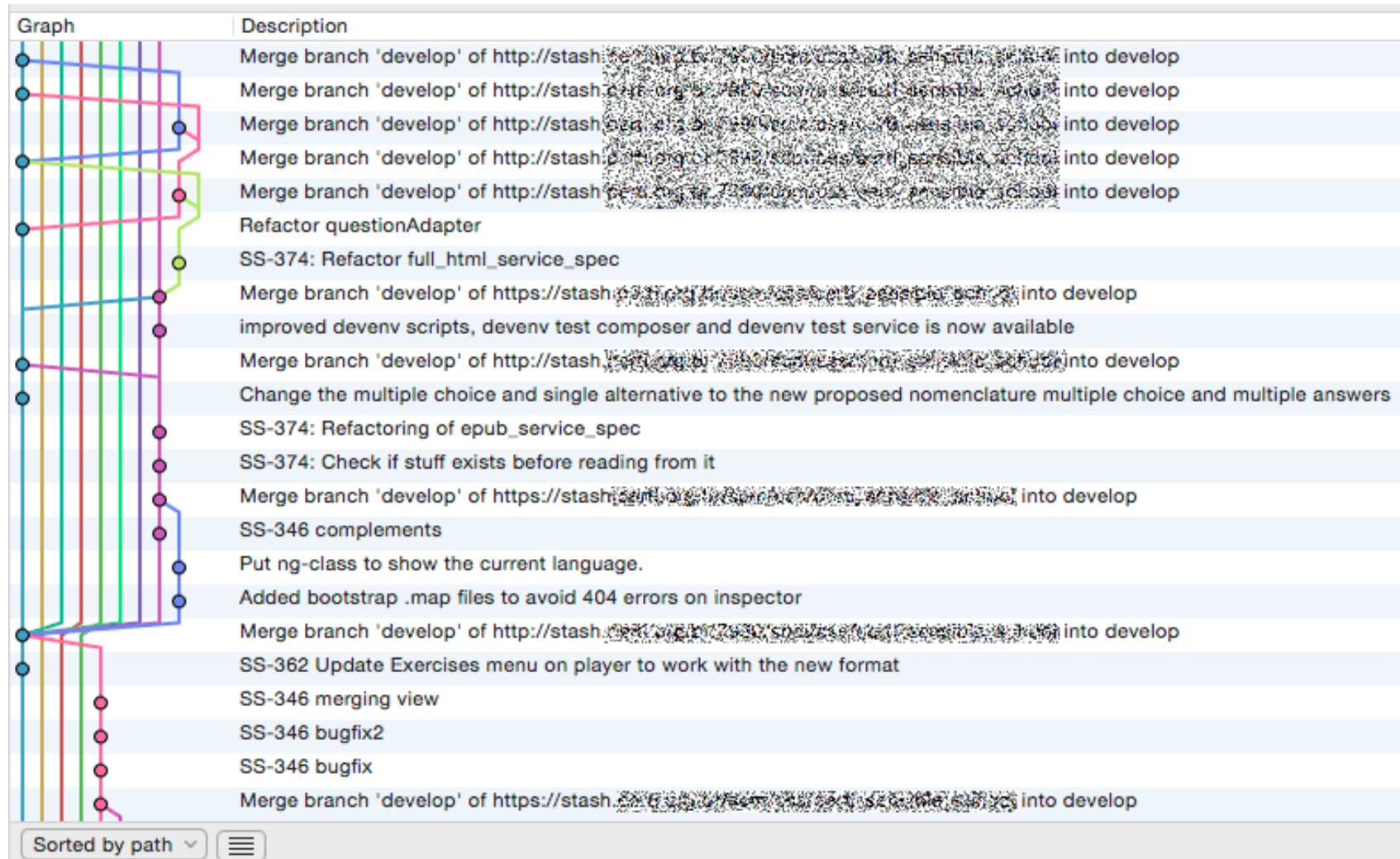
새로운 브랜치가 그냥 기준 브랜치(main 브랜치)가 됨

`git merge --no-ff 브랜치이름` : 강제로 3-way merge

branch 합치기

다양한 merge 방법

그런데... 다 3-way로 그냥 병합하면...



commit 기록 지저분하고 복잡함 -> 버전 관리 어려움

클린 앤 깔끔을 위해서는 어떻게 해야?

branch 합치기

branch 삭제

클린 앤 깔끔

merge해도 브랜치가 자동으로 삭제되지 않으니,

필요 없어진 브랜치는 그때그때 삭제하는 게 클린 앤 깔끔

```
git branch -d 브랜치이름
```

: 병합 완료된 브랜치 삭제

```
git branch -D 브랜치이름
```

: 병합되지 않은 브랜치 삭제

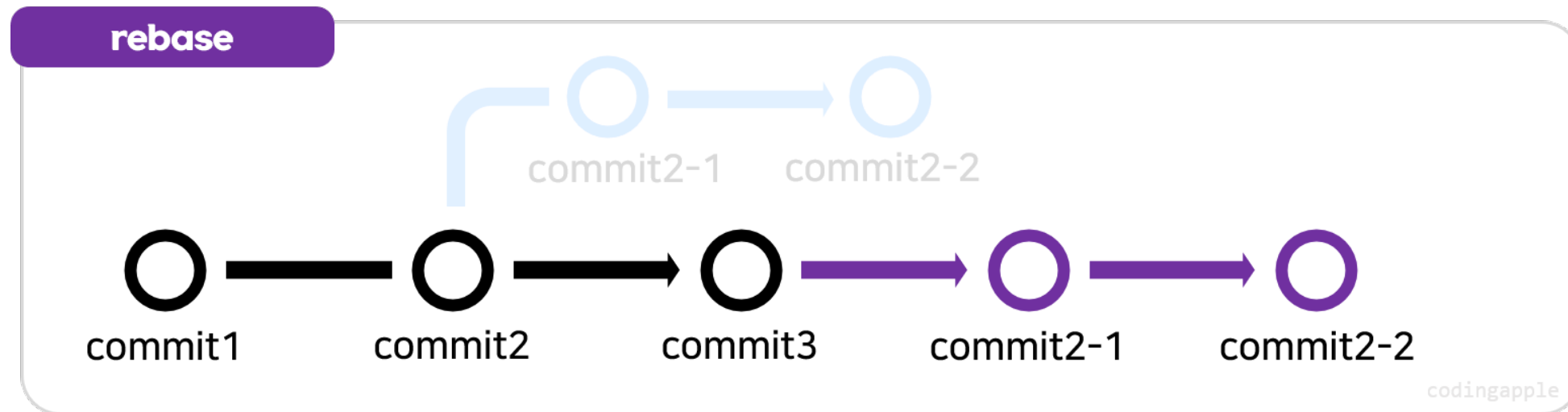
branch 합치기

다양한 merge 방법

클린 앤 깔끔

rebase and merge

: 새로운 브랜치의 시작점을 메인 브랜치 끝으로 옮긴 뒤(rebase)
& fast-forward merge



1. 새로운 브랜치에서 `git rebase main`
2. 메인 브랜치에서 `git merge 브랜치이름`

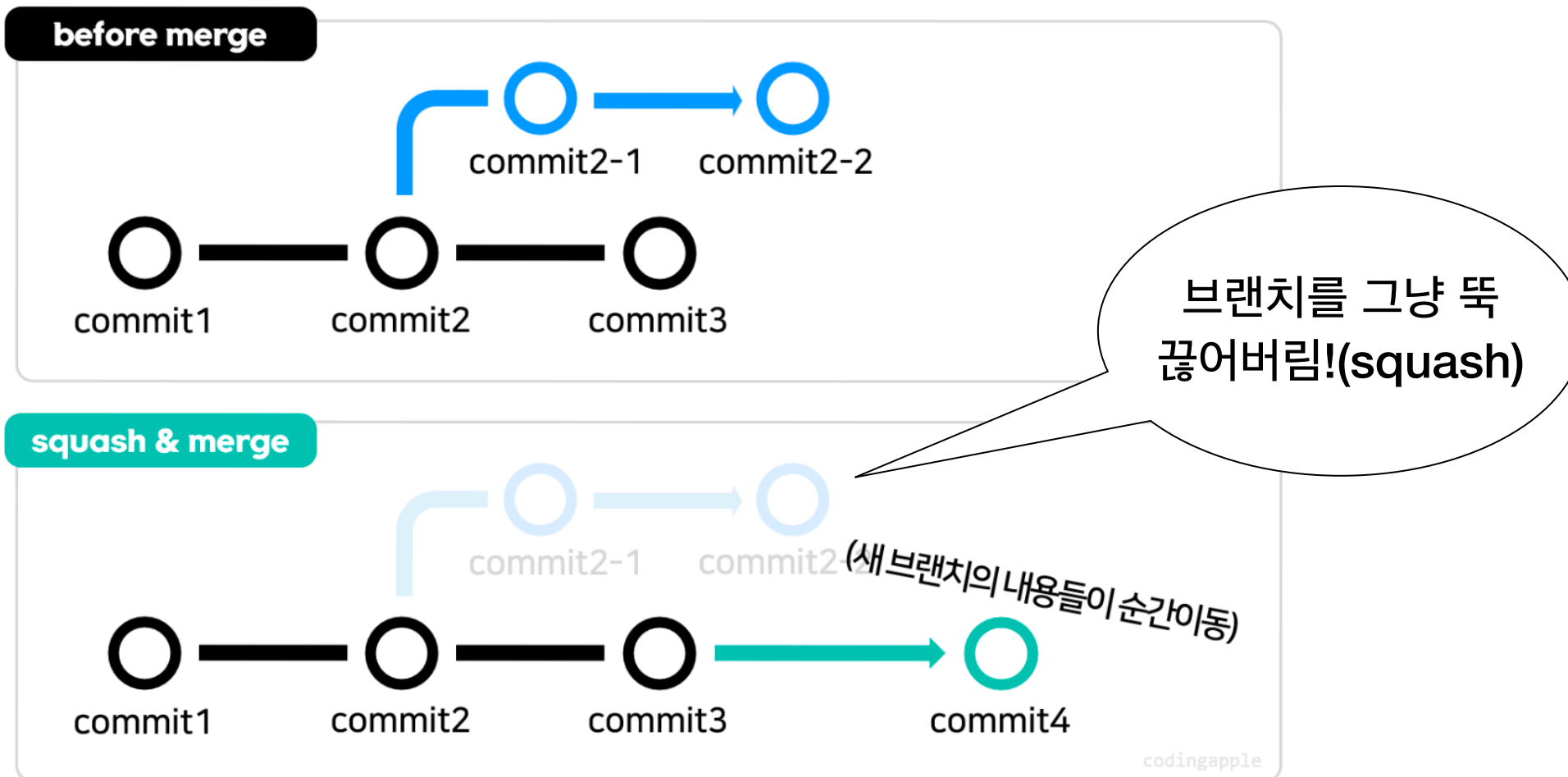
branch 합치기

다양한 merge 방법

클린 앤 깔끔

squash and merge

: 새로운 브랜치에 있던 애들이 새로운 커밋 하나로 순간이동



1. `git merge --squash` 브랜치이름
2. `git commit -m '커밋메시지'`

branch 합치기

다양한 merge 방법

무엇을 선택해야 하나요?

-> 회사/팀에서 정한 가이드라인에 따르세요

파일 복구하기

git restore : 되돌리기

`git restore 파일명`

`git restore .`

: 최근 커밋된 상태로 수정내역 되돌리기

`git restore -source 커밋id 파일명`

: 최근 커밋된 상태로 수정내역 되돌리기

`git restore -staged 파일명`

: 파일의 staging 취소!!



도르마무

파일 복구하기

git revert : 커밋 없애기

1.

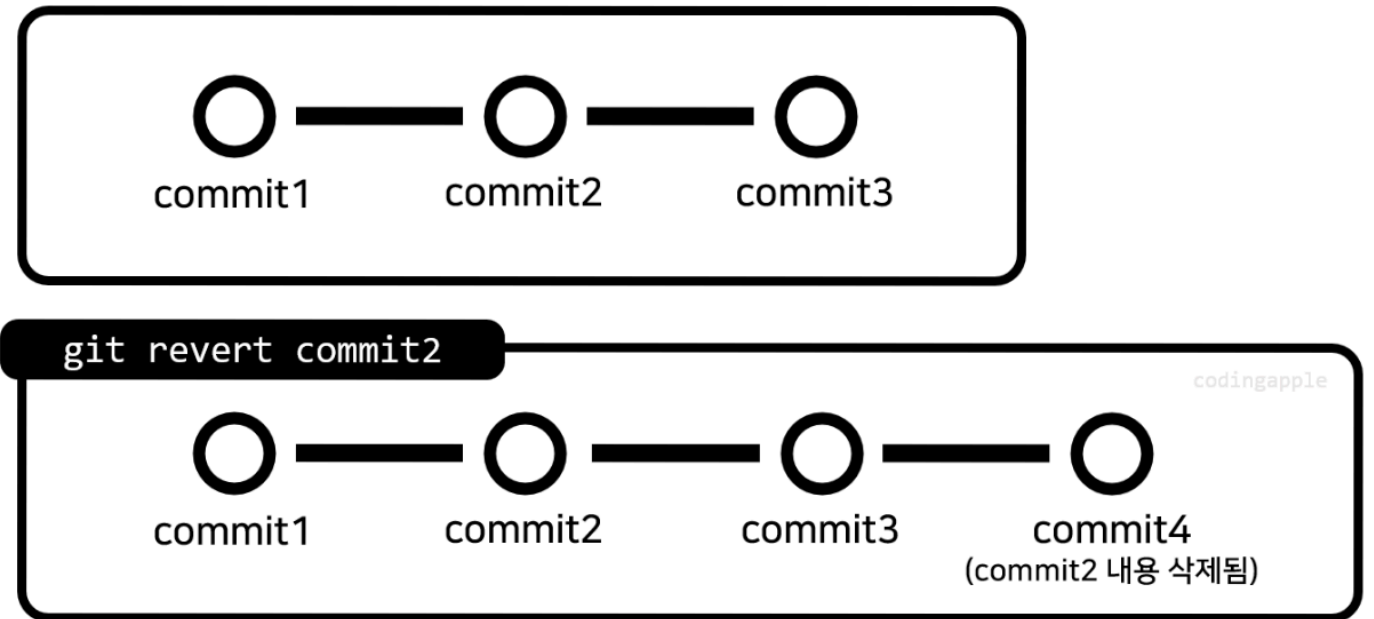
```
git revert 커밋id
```

```
git revert 커밋id1 커밋id2
```

: 커밋id에 해당하는 커밋 삭제
(정확히는 삭제하는 커밋을 함)

```
git revert HEAD
```

: 가장 최근 커밋을 삭제



2. 에디터창이 뜨면 커밋메시지 수정하고 창 닫기

HEAD

What is HEAD?



HEAD is YOU

```
cat .git/HEAD
```

-> ref: refs/heads/main

: 일반적으로 현재 위치하는 branch를 가리킴
(attached HEAD)

-> abc1234

: 특정 commit을 가리킴
(detached HEAD)

파일 복구하기

git reset : 시간을 되돌려

`git reset --hard 커밋id`

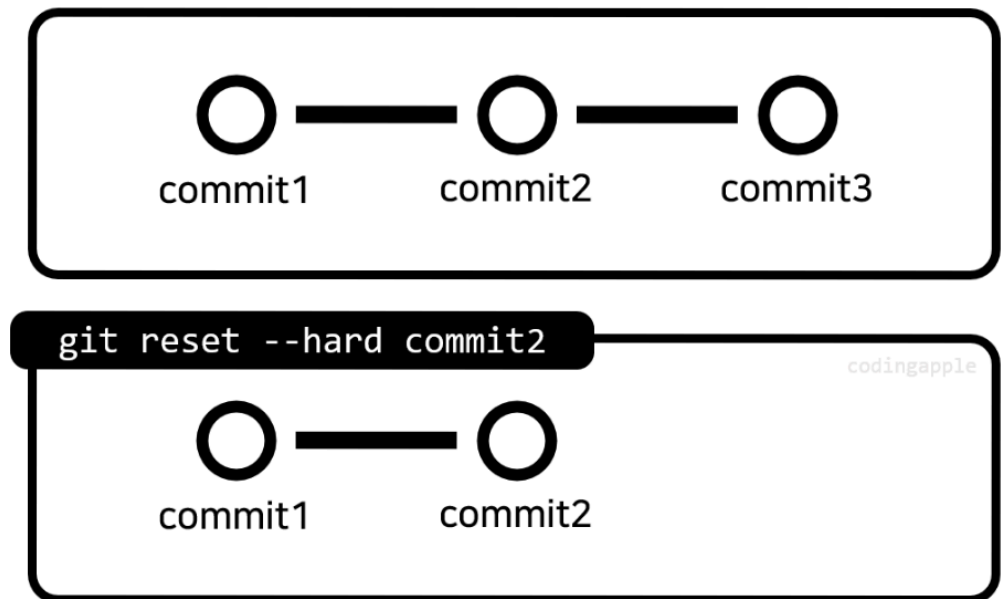
: 커밋id 시점 이후 삭제

`git reset --soft 커밋id`

: 커밋id 시점 이후를 staging area로

`git reset --mixed 커밋id`

: 커밋id 시점 이후를 staging 되지 않은 상태로



짧은 시간여행 정도는 괜찮지만, 되도록 ㄴㄴ

협업 시에는 금지(어둠의 마법)

그 외 다양한 git 어찌구들

git stash, git clean

<https://git-scm.com/book/ko/v2/Git-%EB%8F%84%EA%B5%AC-Stashing%EA%B3%BC-Cleaning>

<https://git-scm.com/docs>

Github

Github 왜 씀?

- 로컬에만 저장했다가 컴퓨터 부서지면 어떡함?
- 협업!!!!

-> Online repository

원격저장소

Github 목차

- 레포 올리기 `git push`
- 레포 내려받기 `git clone`
- 협업하기 `git pull`
- 여러가지 브랜치 전략들

레포 올리기

git push

1. github.com 가서 원격 레포 만들기

2. 로컬 레포의 브랜치를 원격 레포에 올리기

```
git push 원격저장소주소 main
```

- 주소가 너무 길다!!

(방법1) 변수(origin)에 주소 저장 `git remote add origin 원격저장소주소`

(방법2) 주소 기억하라고 명령 `git push -u 원격저장소주소 main`

```
git push -u origin main
```

레포에 올리지 않을 파일들은 .gitignore이라는 파일에 넣으면 됨

레포 내려받기

git clone

```
git clone 원격저장소주소
```

```
git clone origin
```

협업하기

git pull

1. github 레포의 collaborators에 등록된 사람만 push 가능



2. `git push` -> 에러 남

2. `git pull`

`git pull origin 브랜치이름`

: 로컬 레포와 원격 레포 내용을 먼저 같게 해줘야 push 가능

`git pull` = `git fetch` + `git merge`

-> merge conflict 일어날 수 있음

협업하기

Pull request

Pull request: merge 요청 + 코드 리뷰 주고받기

The screenshot shows the GitHub interface for the repository 'younyoungieo / cute_git'. The top navigation bar includes the repository name, a search icon, and buttons for '+', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', and 'Security'. The 'Pull requests' tab is selected and highlighted with a red underline. Below the navigation bar, a message states: 'Label issues and pull requests for new contributors'. It explains that GitHub will help potential first-time contributors discover issues labeled with 'good first issue'. A 'Dismiss' link is available. Below this message, there are filters for 'is:pr is:open', a 'Labels' section showing 9 labels, and a 'Milestones' section showing 0 milestones. A green button labeled 'New pull request' is also present. At the bottom, it shows '0 Open' and '0 Closed' pull requests.

youyoungieo / cute_git

<> Code Issues **Pull requests** Actions Projects Wiki Security

Label issues and pull requests for new contributors [Dismiss](#)

Now, GitHub will help potential first-time contributors [discover issues](#) labeled with **good first issue**

Filters Labels 9 Milestones 0 **New pull request**

0 Open ✓ 0 Closed

여러가지 브랜치 전략들

브랜치 전략이 왜 필요함?

1. 브랜치 관리가 쉬워지고
2. 팀원이 많아도 개발 절차가 **클린 앤 깔끔**해짐

종류

- git flow
- truck-based
- github flow
- gitlab flow
- ...

Reference

git documentation <https://git-scm.com/docs>

코딩애플 git 강좌 <https://codingapple.com/course/git-and-github/>

나무위키 - git <https://namu.wiki/w/Git>

끝