오픈소스 소프트웨어

저작권은 존재하지만 저작권자가 소스코드를 공개하여 누구나 자유롭게 사용, 복제, 배포, 수정, 활용할 수 있는 SW를 지칭

OSI의 공개 SW정의

1. 자유로운 배포(재배포시 저작권자에 의한 라이선스 의무사항 준수)

2. 소스코드 공개(난독화 금지)

3. 2차 저작물 허용

4. 저작권자의 소스코드 무결성(원 저작자 정보 삭제 금지)

5. 개인이나 단체에 대한 차별 금지

6. 사용 분야에 대한 제한 금지

7. 라이선스의 배포

8. 라이선스의 적용상 동일성 유지(특정 제품 의존 금지)

9. 특정 라이선스 제한 금지(다른 라이선스 포괄 수용)

10. 라이선스의 기술적 중립성

공개sw는 비용의 관점이 아닌 소스코드의 공유 관점에서 이해해야 함

비용 관점

Freeware(OSS) – 개인 무료 : 아무런 대가 없이 사용 가능

Shareware – 기간 제한 : 마케팅 목적으로 무료

ADware – 광고 이용 : 광고의 대가로 무료

Commercial Software – 유료

소스 공유 관점

공개SW – OSS(Open Sourece Software)

비공개SW – Freeware, Freeware, Adware, Commercial Software

유료와 무료 외 구분 지표

(공개 소프트웨어 – 상용 소프트웨어 = )

소스코드

커뮤니티

카피레프트

(상용 소프트웨어 – 공개 소프트웨어 = )

보증

마케팅

제품 가격

카피레프트

저작권 기반의 사용 제한이 아닌 저작권 기반 정보 공유

공개 소프트웨어 장점

개발 효율성 – 공개 SW 활용을 통해 개발기간 단축 및 비용절간 실현

최신기술 도입 – 소스코드 원천기술에 대한 접근으로 선진SW기술 습득 용이

기업 저변확보 – 소스코드 개방을 통해 SW저변 확산 용이

벤더 종속성 탈피 – 국내 시장에서의 독과점 SW 대체재 역할

공개SW 핵심 요소

코드(code)

SW의 핵심, 커뮤니티를 통해 창출이 가능하며 무엇인지 의미

커뮤니티(community)

깃허브(Github), 소스포지(source Forge) 등 공개 SW프로젝트 진행 단체가 누구인지 의미

브랜드(brand)

아파치(Apache), 리눅스(Linux) 등 트레이드 마크 등을 포함하여 프로젝트가 어떻게 반영되는지 의미

버전 관리

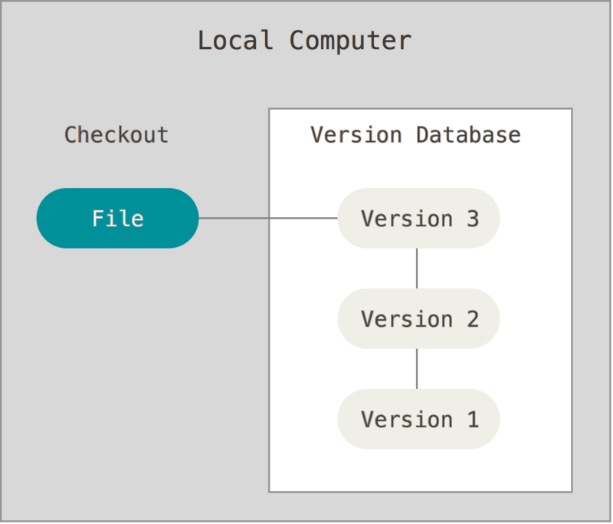
변화(추가, 수정, 삭제)를 기록하고 원하는 시점으로 이동할 수 있도록 도와주는 것

특정 시점으로의 상태로 파일을 되돌리거나 팀 개발에서의 파일관리를 쉽게 도와준다.

버전 관리 시스템(Version Control System)

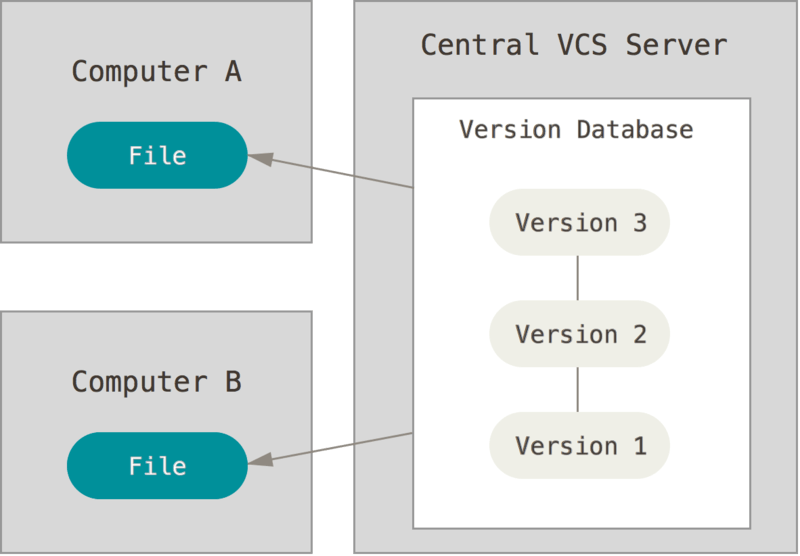
로컬 버전 관리

파일 복사대신 간단한 데이터 베이스를 이용해서 파일변경 정보 관리



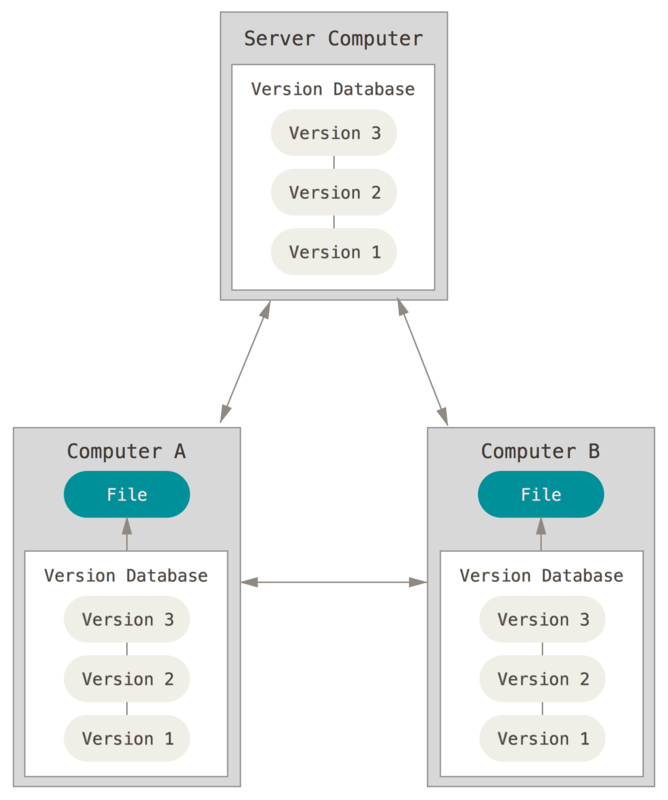
중앙집중식 버전 관리

팀개발에 적용되며 중앙 서버에서 파일을 관리하지만 서버 다운 시 협업이 중단된다.



분산 버전 관리

Git과 같은 분산 버전 관리 시스템으로 저장소를 전부 복제해서 서버로 복원이 가능하다



공개 SW 라이선스

라이선스(License) 정의

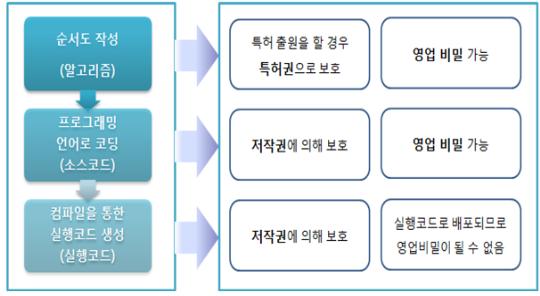
저작권자가 자신의 권리 중 일부분(사용, 복제, 재배포 등)을 일정내용의 조건으로 이용자가 사용할 수 있도록 권한을 부여한 이용 허가권

공개 SW 라이선스 정의

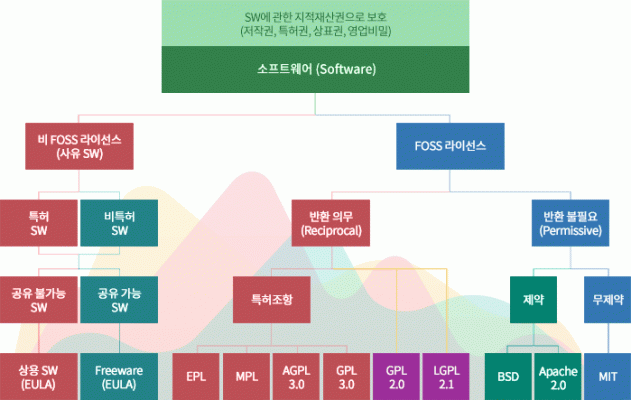
공개 SW 개발자와 이용자 간의 사용 방법 및 조건의 범위를 명시한 계약

-> 둘 다 위반시 법적 책임을 지게 된다.

법적 보호



라이선스 구분



Git

Linux 커널 오픈소스 관리 -> BitKeeper 라는 상용 분산버전관리 사용

2005년 BitKeeper 무료 사용 제고

🡪 Git 탄생 2005년

Git목표

- 빠른 속도

- 단순한 구조

- 비선형적인 개발

- 완벽한 분산(수천 개의 동시 다발적인 브랜치)

- linux커널 같은 대형 프로젝트에도 유용할 것(속도나 데이터 크기 면에서)

Git 기초

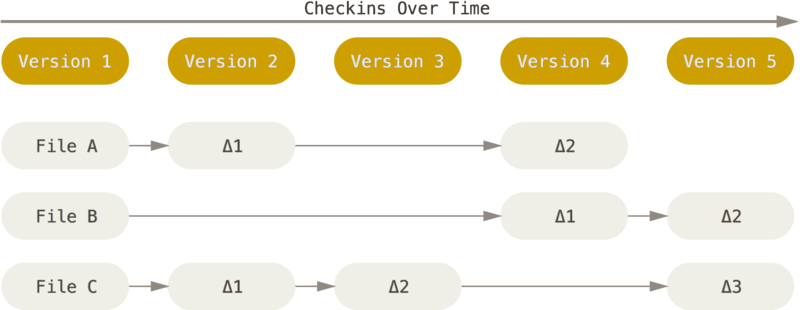
변화된 부분을 찾아서 저장하는 것을 스냅샷(snapshot)이라고 함

SubVersion – 차이점만 저장 delta

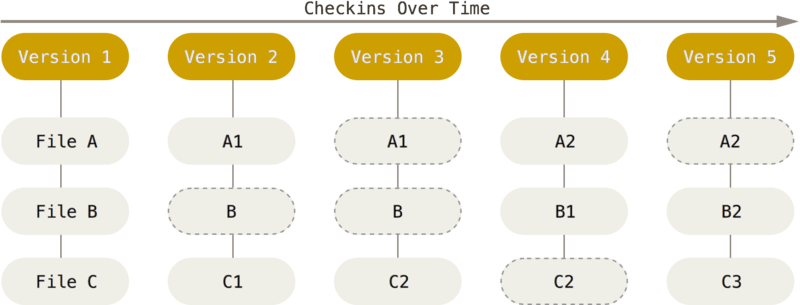
Git – 전체를 저장 Snapshot

변경 시 파일을 저장, 변경이 없다면 링크만 저장

각 파일에 대한 변화 저장



시간순으로 프로젝트의 스냅샷 저장



Git의 무결성

Git은 체크섬으로 데이터를 관리

Git은 SHA-1 해시를 사용하여 체크섬 생성

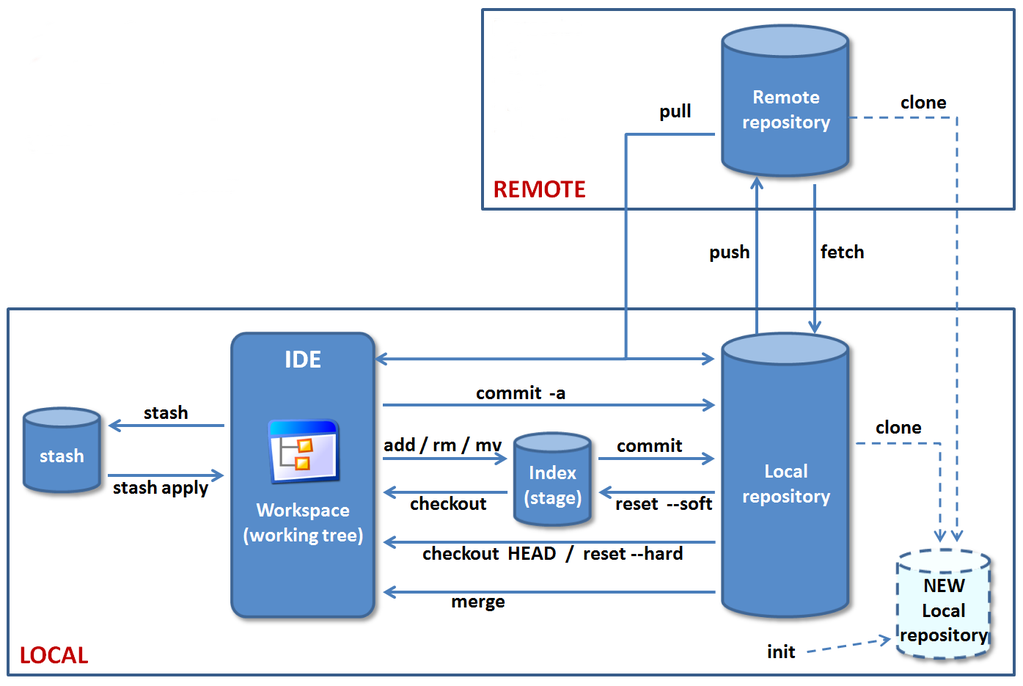
체크섬은 40자 길이의 16진수 문자열

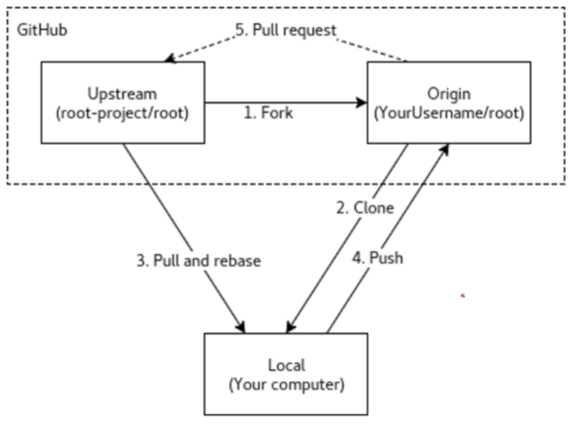
SHA-1 값의 중복

- 해시 값이 중복될 충돌 확률은 p = (n(n-1)/2) \* (1/2^160)

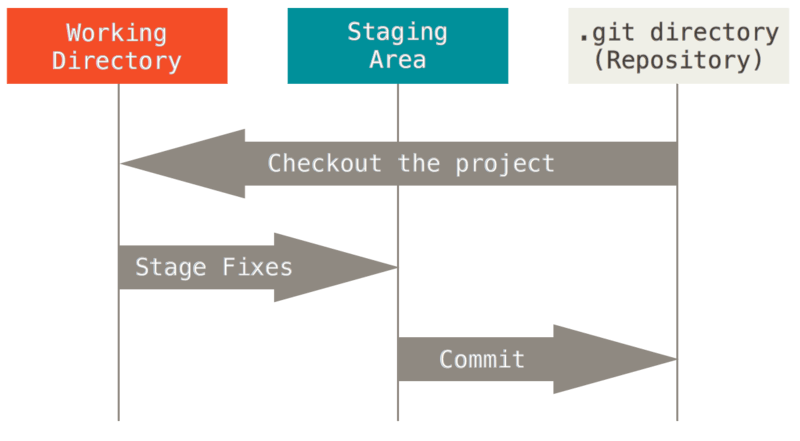
-> 대부분 겹칠 일이 없음

git workflow





Git의 세가지 단계



로컬 저장소 만들기

- $ git init

환경 설정

- $ git config --global user.name “계정 이름”

- $ git config --global user.email “이메일 주소”

- $ git config –list : 설정 확인

스테이지에 올리기

- $ git add “파일명”

커밋하기

- $ git commit -m “커밋 설명”

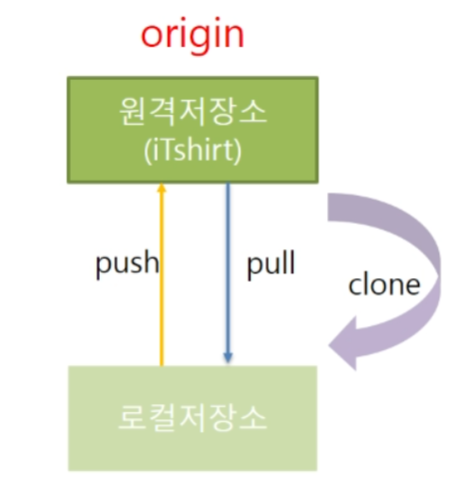
커밋 이력 확인

- $ git log

특정 커밋으로 돌아가기

- $ git checkout “해시값7자리~”

원격 저장소



원격저장소 등록

- $ git remote add origin “저장소 주소”

원격 저장소에 커밋 올리기

- $ git push origin master

master – 로컬 저장소

origin/master – 원격 저장소 버전

로컬 저장소에 내려 받기

클론 – 원격 저장소의 코드와 버전을 애 컴퓨터로 전체 내려 받기 최신 버전과 이전 버전, 원격 저장소 주소 등이 저장됨

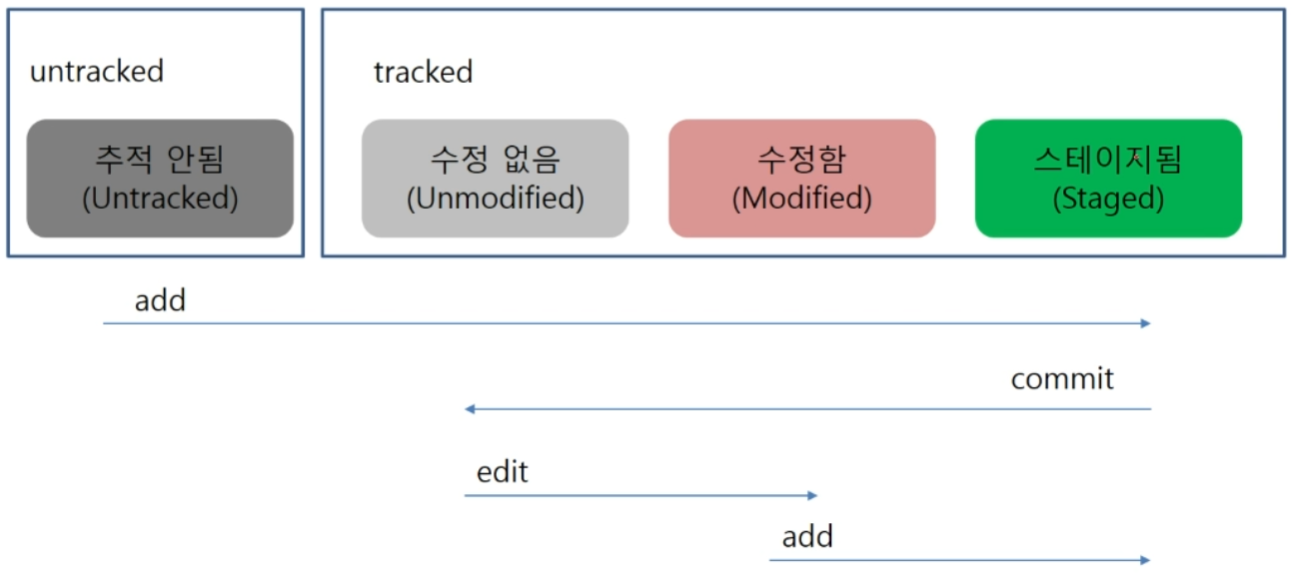
- & git clone “저장소 주소” .

원격 저장소의 새로운 커밋을 받아오기

- $ git pull origin master

패치 fetch 기능 : 이력을 가지고 옴 (파일 병합 X)

Git으로 관리하는 파일의 4가지 상태

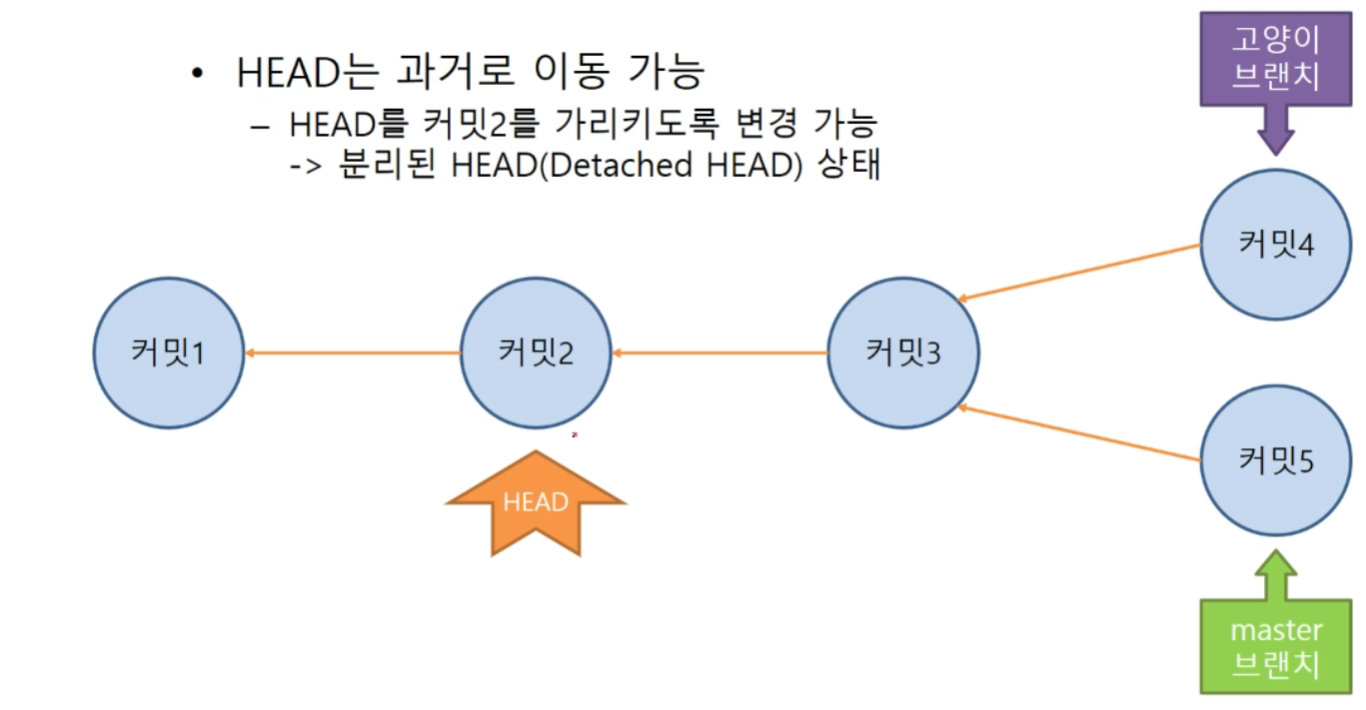


브랜치 branch

- 줄기를 나누어 작업을 할 수 있는 기능

- master는 git에서 제공하는 기본적인 브랜치 (브랜치는 포인터)

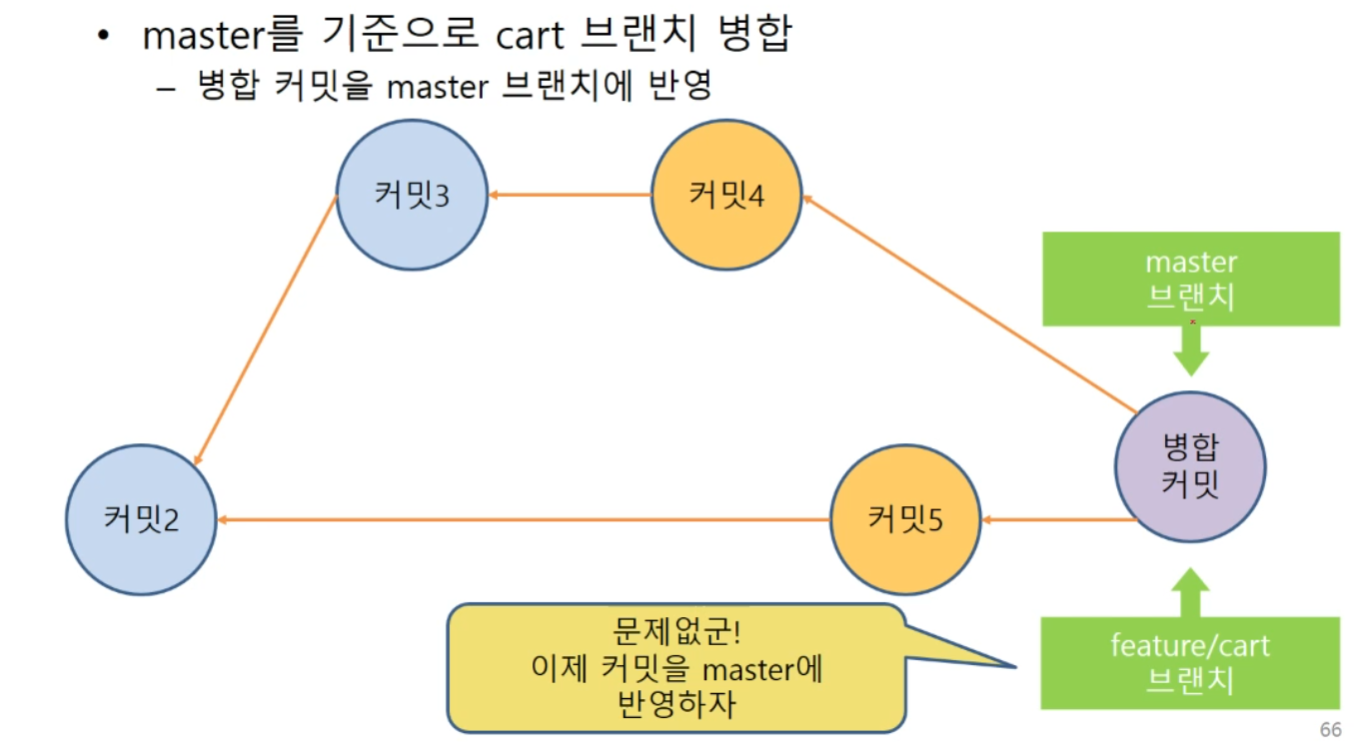
- HEAD라는 특수한 포인터로 커밋 이력 이동 가능(분리된 HEAD(Detached HEAD)상태)



브랜치 이동은 checkout 사용

병합 merge

- 두 버전의 합집합을 구하는 것 -> 겹치면 충돌(Conflict)



포크 fork

- 브랜치를 포함한 원본 저장소의 모든 커밋 이력을 새로운 원격저장소에 통째로 복사함

브랜치와 포크의 차이점

브랜치

- 하나의 원본 저장소에서 분기를 나눔 (평행 세계)

- 하나의 원본 저장소에서 코드 커밋 이력을 편하게 볼 수 있음

- 다수의 사용자가 다수의 브랜치를 만들면 관리가 힘듦

포크

- 여러 원격저장소를 만들어 분기를 나눔 (평행 우주)

- 원본 저장소에 영향을 미치지 않으므로 원격저장소에서 마음껏 코드를 수정할 수 있음

- 원본저장소의 이력을 보려면 따로 주소를 추가해야 함

포크한 원격 저장소에서 pull request를 보내서 요청하면 원본저장소에 변경 내용을 저장 할 수 있다.

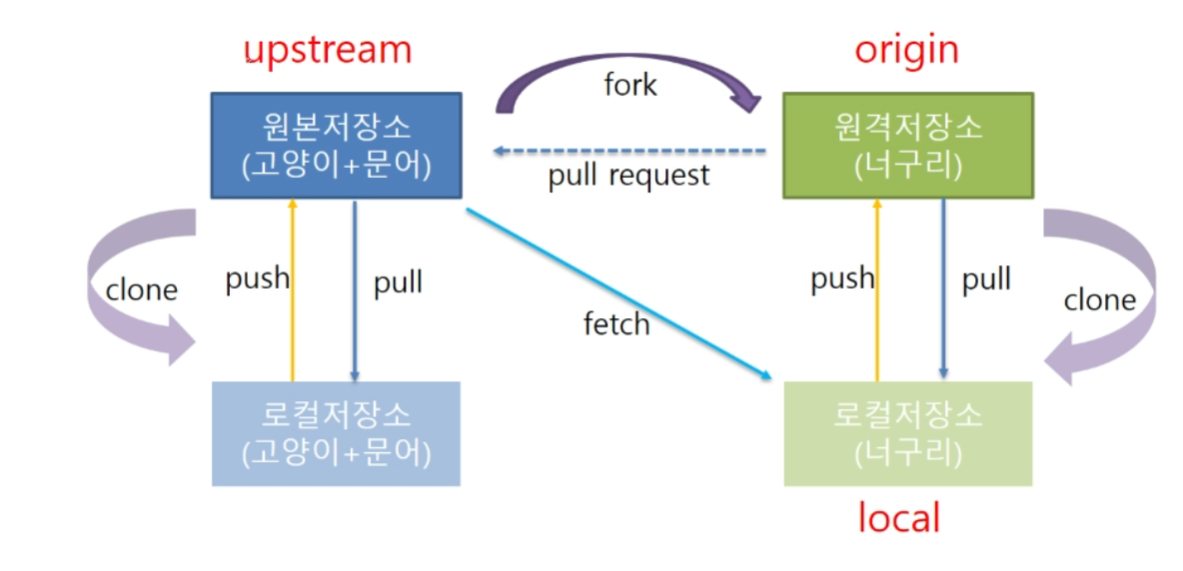
충돌 발생시 내 브랜치에서 병합해서 병합 커밋을 만들고 다시 풀 리퀘스트 시도

태그 tag

- 현재 코드 상태를 버전으로 만들어 기록할 때 활용

- 커밋을 가리키는 포인터

정리



내 원격 저장소에서 다른 원본 저장소 히스토리 보기 : 리모트 추가(Add remote)

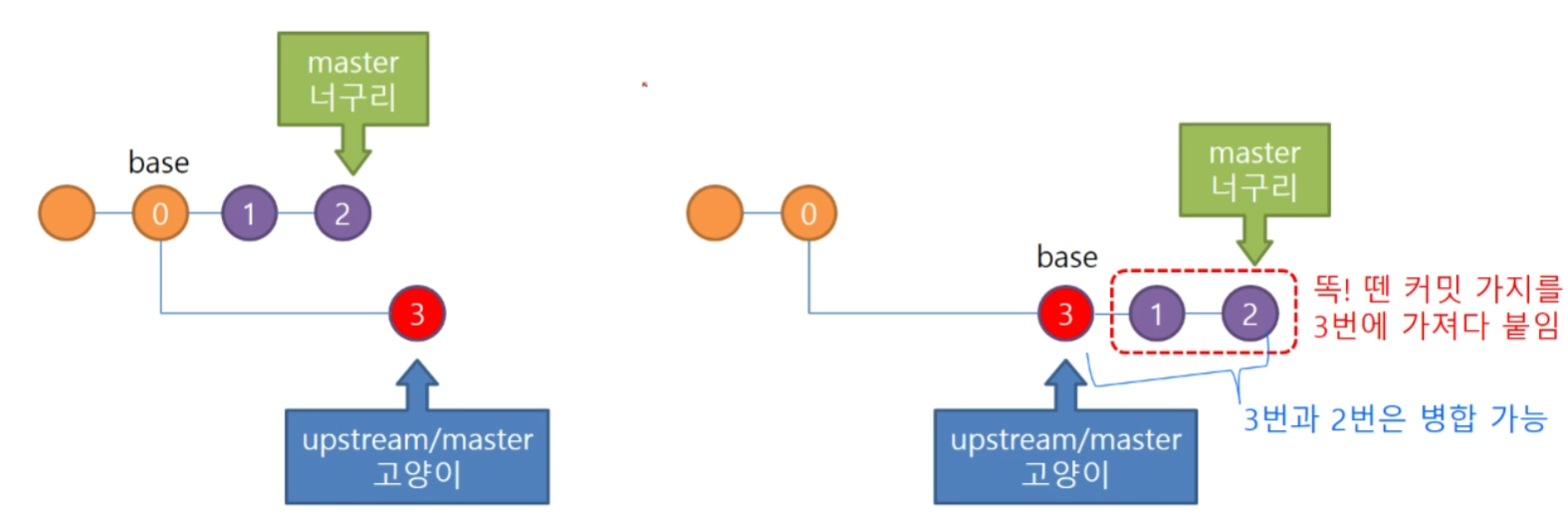
- 원격 저장소와 원본 저장소를 동시 추적

리베이스Rebase

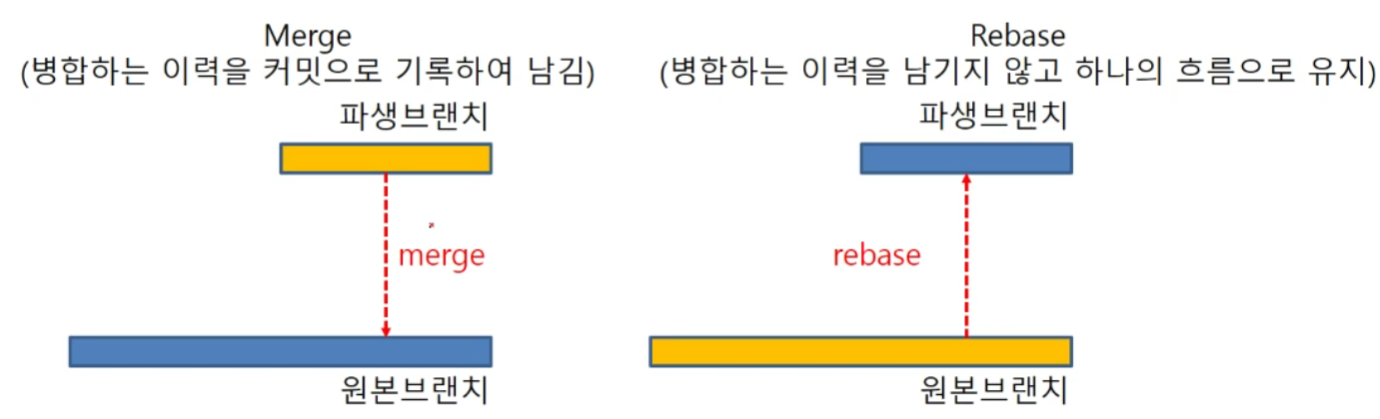
- 브랜치 병합 전략 중 하나로 커밋의 히스토리 정리가능

- 주의점 : 원격 저장소로 푸시한 커밋은 rebase하지 않을 것 -> 중복 발생

- fast-forward merge : 새로운 상태를 만들어줄 필요 없이 뒷 상태로 바꾸는 빨리 감기 병합



리베이스와 머지의 차이점



merge는 가지고 오는 것

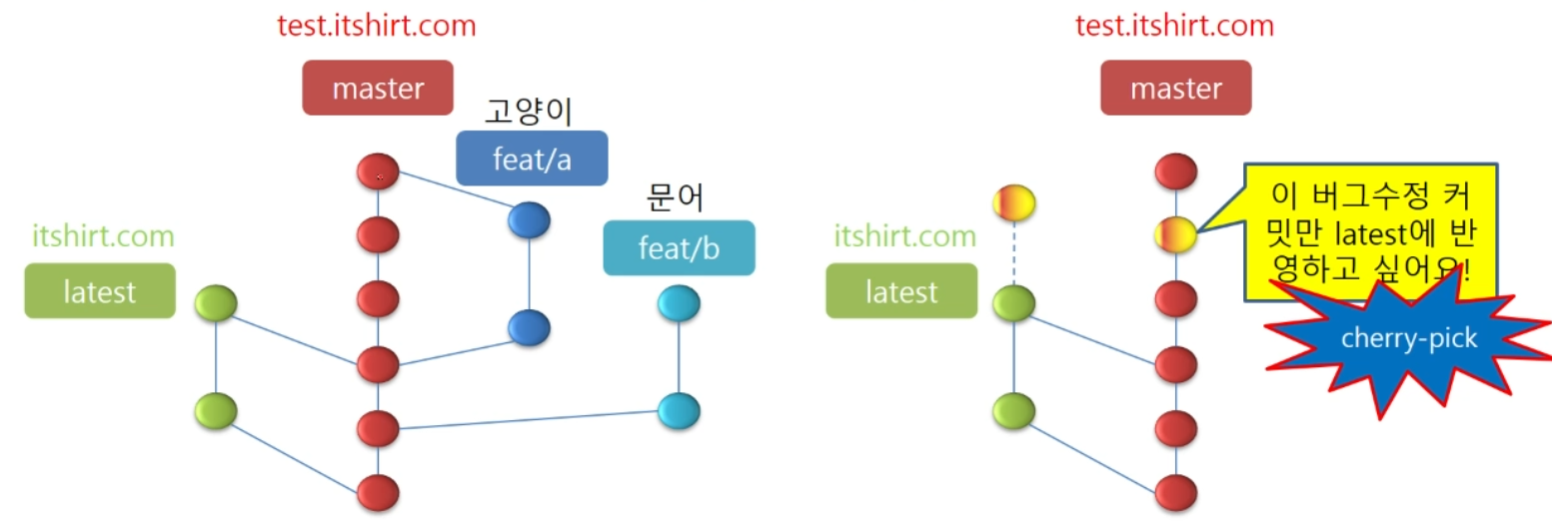
rebase는 보내는 것

마지막 커밋 수정 amend

- 마지막으로 수정한 커밋 메시지를 수정한다

특정 커밋의 변경 사항만 현재 브랜치에 가져오기 cherry-pick

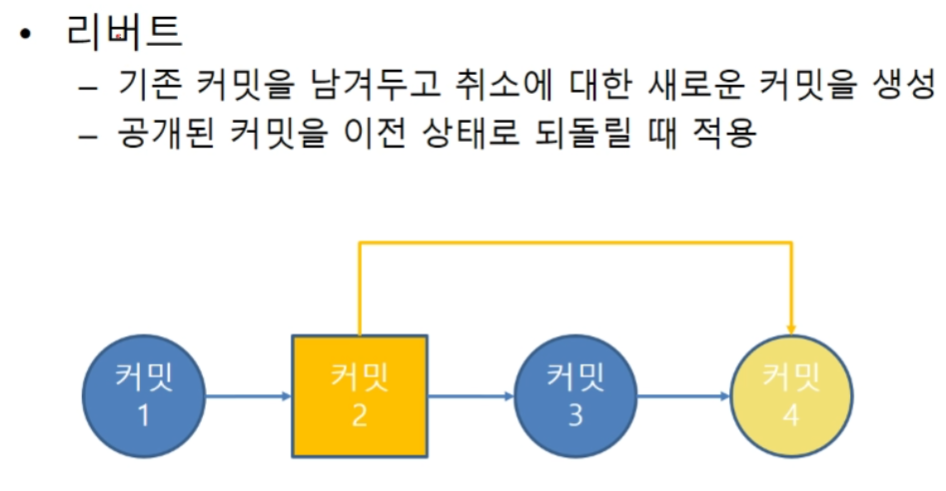
- 다른 브랜치의 변경 사항을 현재 커밋에 붙인다



리버트 Revert

- 변경을 되돌리는 Revert 커밋을 생성

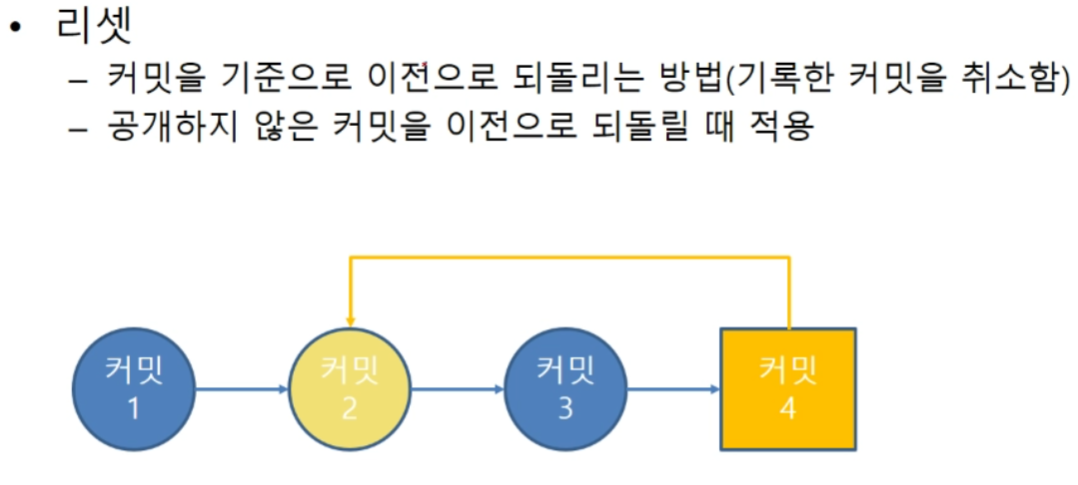
- 기존 커밋을 남겨둠

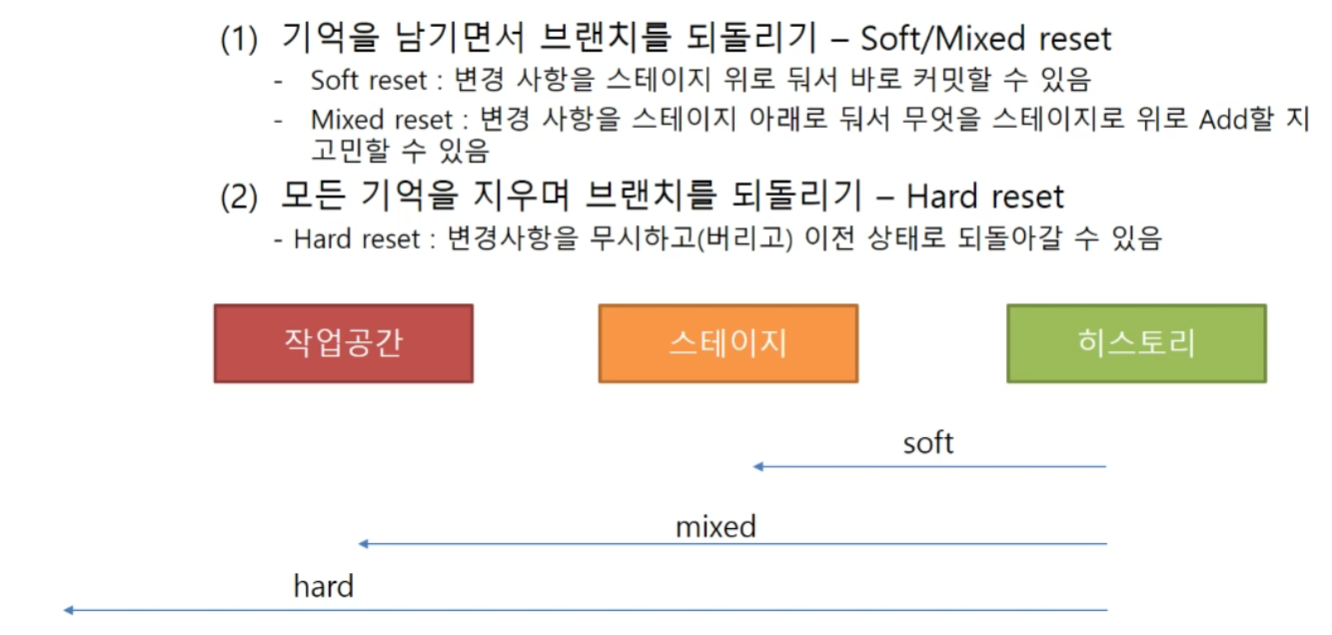


리셋 Reset

- 옛날로 돌아감(기존 커밋 남지 않음)

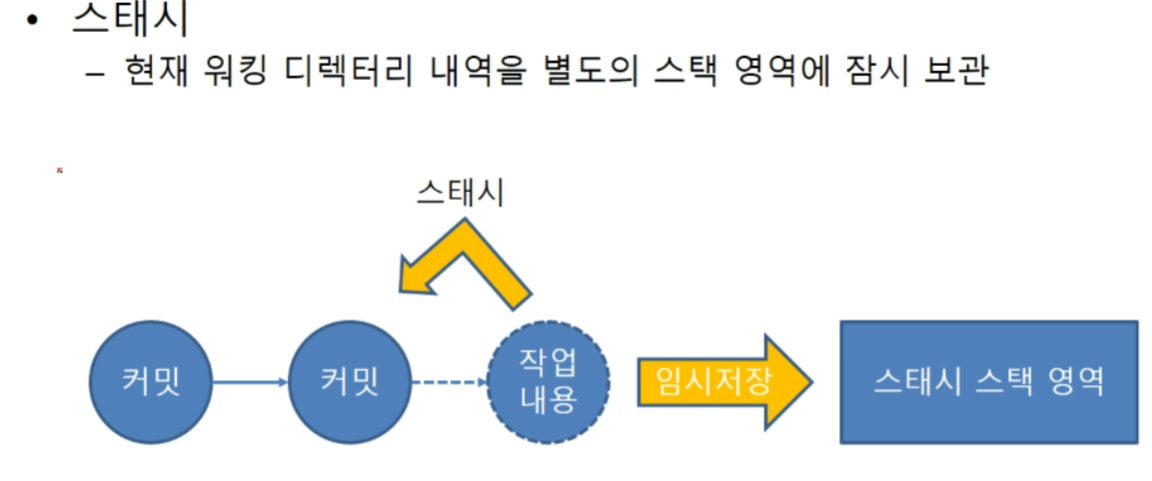
- 커밋만 되돌리려면 Soft 사용





stash

커밋하지 않은 변경사항 임시 저장(새로 만든 파일은 들어가지 않음)



용어 정리

GitHub : 깃 허브, Git으로 관리하는 프로젝트를 올려 둘 수 있는 사이트

GUI : 그래픽 유저 인터페이스, 마우스로 클릭해서 사용하는 방식

CLI : 커맨드 라인 인터페이스, 키보드로 명령어를 하나씩 입력하는 방식

Git Bash : CLI방식으로 Git을 사용할 수 있는 환경

commit : 커밋, 버전 관리를 통해 생성된 파일 혹은 그 행위를 의미

log : 지금가지 만든 커밋을 모두 확인하는 명령어

checkout : 체크아웃, 타임머신처럼 원하는 지점으로 파일을 되돌릴 수 있음

HEAD : 현재 체크아웃 된 커밋을 가리킴(작업중인 가장 최근 커밋)

local repository : 로컬 저장소, Git으로 버전 관리하는 내 컴퓨터 안의 폴더를 의미

remote repository : 원격 저장소, GitHub 에서 협업하는 공간(폴더)를 의미

push : 푸시, 로컬저장소의 커밋(버전 관리한 파일)을 원격 저장소에 올리는 것

pull : 풀, 원격저장소의 파일을 로컬저장소에 내려 받는 것(자동 병합)

fetch : 원격저장소의 파일을 로컬저장소에 내려 받는 것(병합을 수동으로 해야함)

branch : 분기하는 것

merge : 머지, 분기된 버전 간에 통합하는 것

fork : 남의 GitHub 저장소를 내 GitHub 저장소로 통째로 복사하기

clone : 원격 저장소로 내 컴퓨터로 복사하기

Upstream : 다른 사람의 GitHub 저장소를 fork할 때 시도한 다른 사람의 저장소를 Upstream으로 부름(오리지널 레포)

Origin : 다른 사람의 GitHub 저장소를 fork해 만든 내 GitHub 저장소를 Origin으로 부름(리모트 레포)

pull request : 협력자에게 브랜치 병합을 요청하는 메시지를 보내는 것(자신의 버전을 가져가 달라는 요청 메시지)

reset : 이전 이력으로 되돌리는 방법

rebase : 이력을 조작하여 단순화 시키는 방법(통합하는 방법)

stash : 임시로 변경 사항을 저장하는 방법

tag : 특정 커밋에 포스트 잇 붙이기