

2019 Fall

# 오픈소스SW 프로그래밍



조거리

(kyurijo@chungbuk.ac.kr)













# GIT 기초

# 버전 관리 (Version Control)

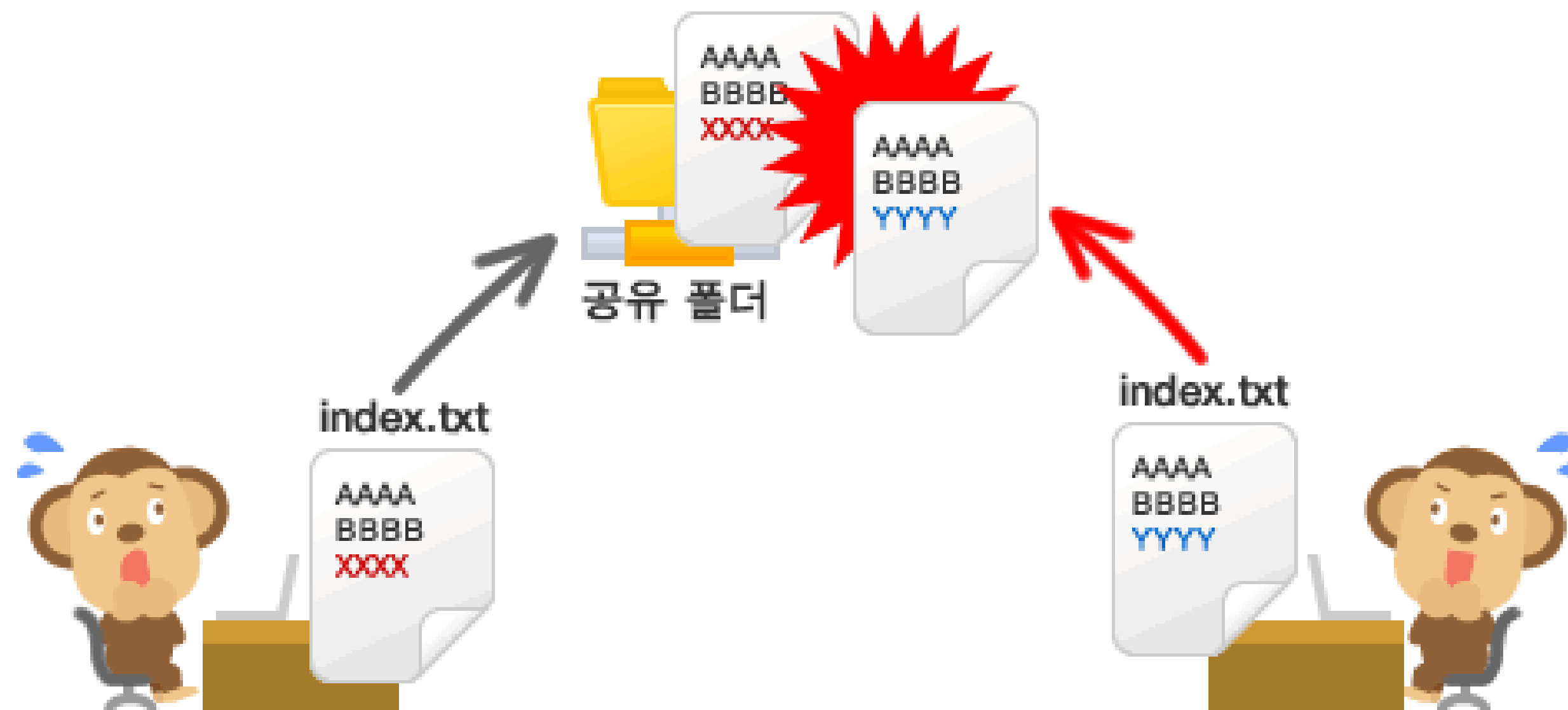
- 어떤 프로젝트에 속한 파일들을 지속적으로 수정하고, 각각의 버전을 관리하고자 할 때
- Issue 1. 저장 방법**

Name	
	120525_문서_업데이트.txt
	120604_문서.txt
	120605_문서_수정판.txt
	120605_문서_수정판2.txt
	120605_문서_최신 복사.txt
	120605_문서_최신.txt
	120605_문서.txt
	1200602_문서.txt
	문서_회의용.txt



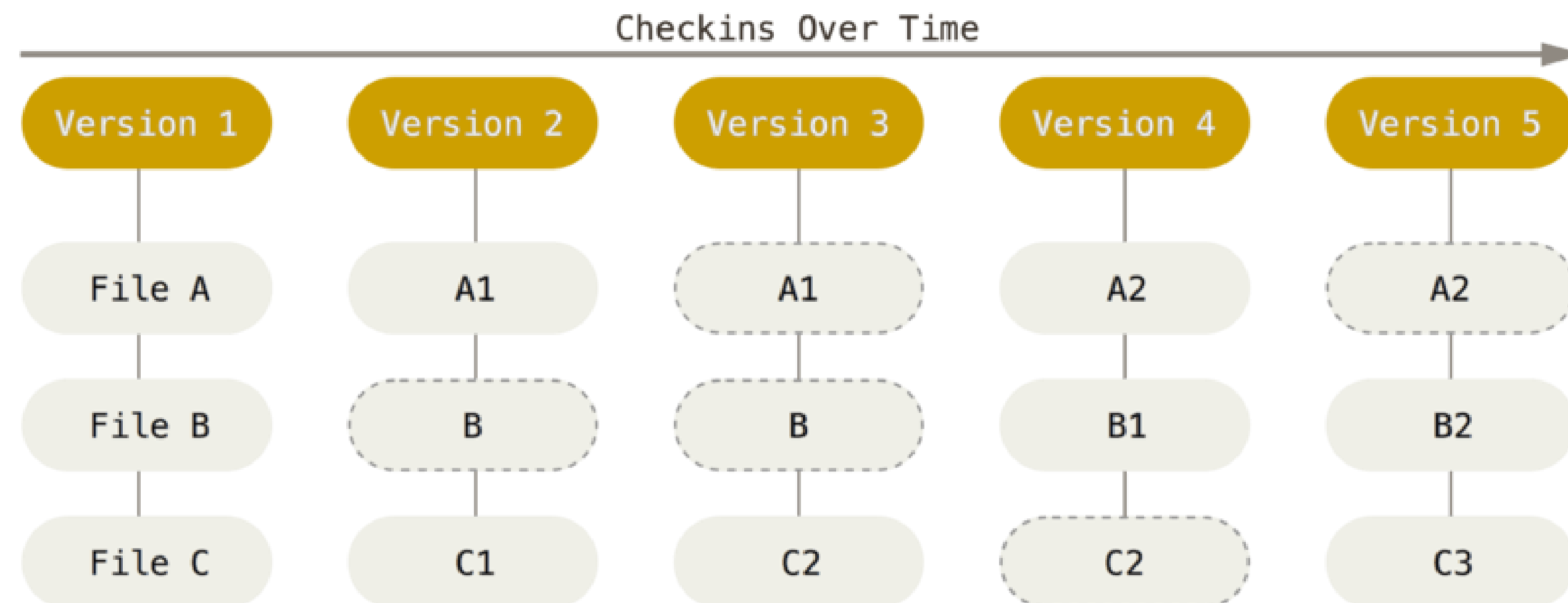
# 버전 관리 (Version Control)

- 어떤 프로젝트에 속한 파일들을 지속적으로 수정하고, 각각의 버전을 관리하고자 할 때
- Issue 2. 공유 방법



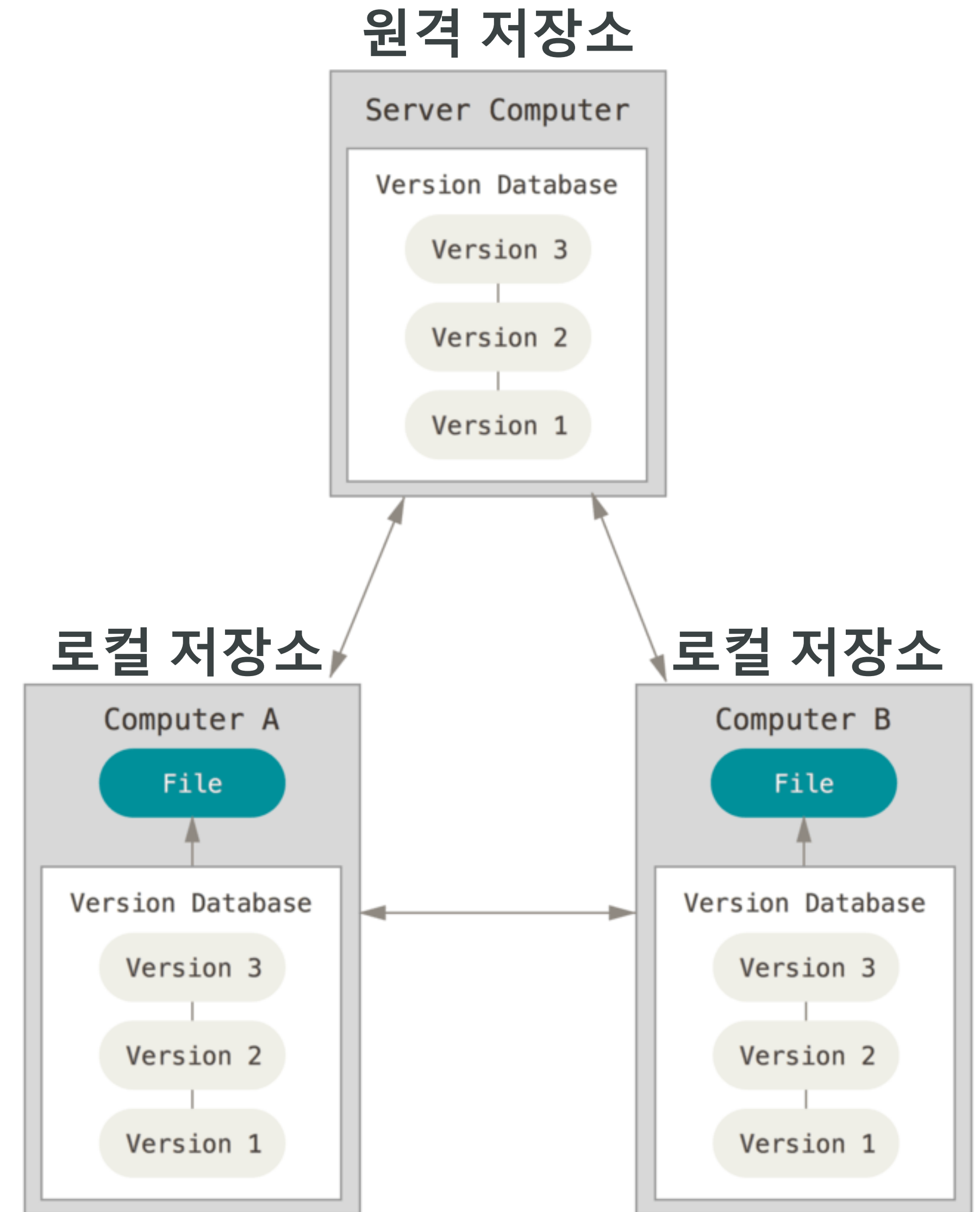
# Git이란

- 분산 버전 관리 시스템(Distributed Version Control System)
- **Issue 1. 파일들의 스냅샷을 저장**
  - 사용자가 커밋(Commit) 할 때마다 파일들의 현 상태를 저장
  - 변경사항이 없는 파일은 이전 상태에 대한 링크만 저장



# Git이란

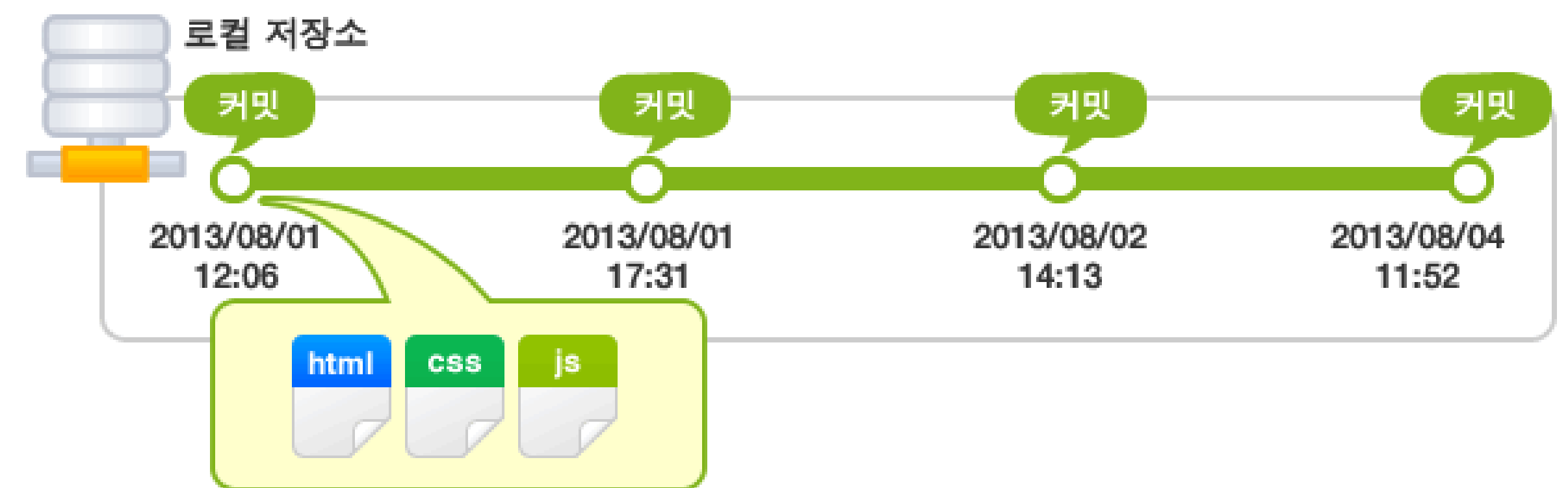
- 분산 버전 관리 시스템
- **Issue 2. 원격-로컬 저장소**
  - **원격 저장소**(Remote repository):  
여러 사람이 공유하는 저장소
  - **로컬 저장소**(Local repository):  
프로젝트에 참여하는 개개인 혹은  
한 사람의 여러 컴퓨터에 존재하는 저장소
  - 원격 → 로컬로 버전 히스토리까지  
전부 복제하여 저장



# Git의 특징

- 프로젝트의 모든 히스토리가 로컬 저장소에 존재
- 원격 저장소 접속할 필요 없이 작업 수행 (e.g. 비행기에서 작업)
- 로컬에서 파일 수정 후 새로운 버전 커밋 → 원격 저장소에 업데이트

- 버전 히스토리는 커밋 단위로 저장





# Git의 특징

- Git이 인식하는 파일의 상태 : modified/staged/committed





# Git의 특징

- 사용자가 원하는 파일만 새로운 버전에 추가시킬 수 있음
- 커밋된 적 있는 파일은 tracked / 없는 파일은 untracked

## 작업 디렉토리 (working directory)

README.txt  
log.txt  
main.py  
sub1.py  
sub2.py

→  
log.txt  
main.py  
sub1.py  
수정

log.txt  
main.py  
sub1.py  
**Modified**

## 준비 영역 (Staging area)

→ **add**  
main.py와  
sub1.py만 추가

main.py  
sub1.py  
**Staged**

## Git 저장소 (Git repository)

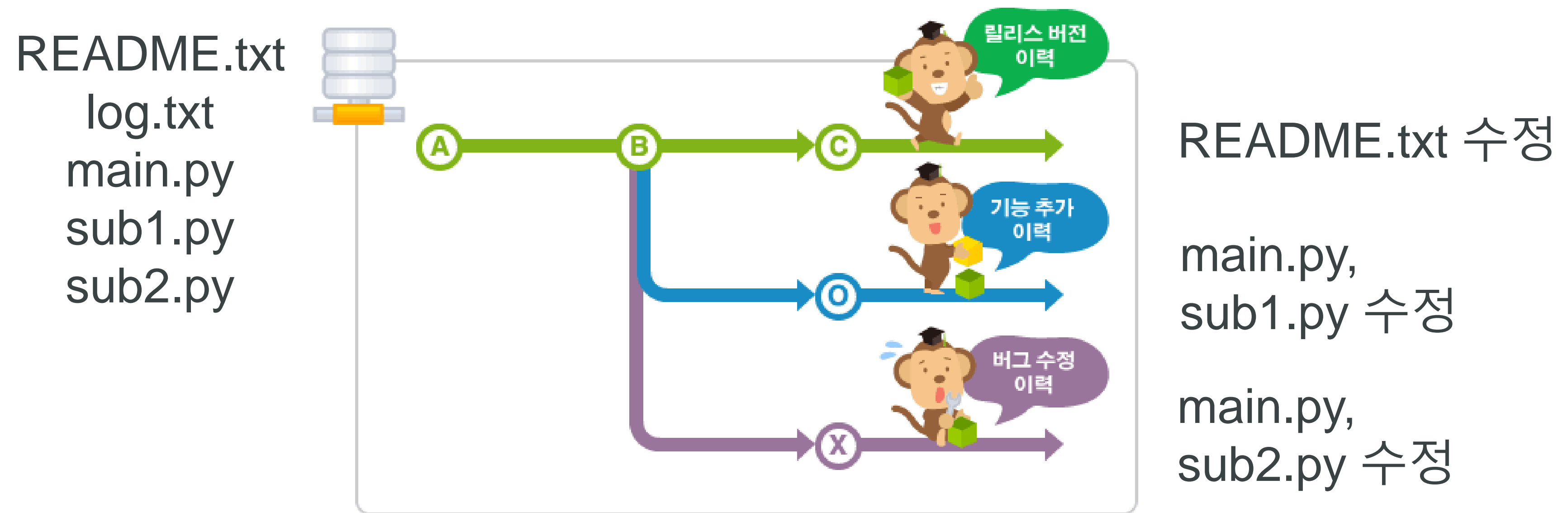
→ **commit**  
커밋

main.py  
sub1.py  
**Committed**

**Tracked :** main.py, sub1.py  
**Untracked :** log.txt, sub2.py

# 커밋과 브랜치

- 커밋(commit) : 사용자가 저장한 프로젝트의 상태
- 연관된 커밋들은 하나의 브랜치(branch)를 이룸



# 커밋 (Commit)

- 이전 커밋과 달라진 점을 기록하는 것이 필요함
  - 짧게 작성 시 한 줄로 작성
  - 길게 작성 시 **첫번째 줄(제목) + 한 줄 공백 + 구체적인 내용**
- 좋은 커밋 메시지 작성하기 (영문 기준)
  1. 제목과 본문을 한 줄 띄워 분리하기
  2. 제목은 영문 기준 50자 이내로
  3. 제목 첫글자를 대문자로
  4. 제목 끝에 **.** 금지
  5. 제목은 **명령조** 로
  6. Github - 제목(이나 본문)에 이슈 번호 붙이기
  7. 본문은 영문 기준 72자마다 줄 바꾸기
  8. 본문은 **어떻게** 보다 **무엇을**, **왜** 에 맞춰 작성하기

# 커밋 (Commit)

- 커밋 로그(log)를 통해 기존 커밋 메시지를 확인할 수 있음
- 각 커밋은 체크섬(checksum, 데이터의 상태를 표현한 숫자)을 가짐

```
$ git log
commit 42e769bdf4894310333942ffc5a15151222a87be
Author: Kevin Flynn <kevin@flynnsarcade.com>
Date:   Fri Jan 01 00:00:00 1982 -0200

Derezz the master control program

MCP turned out to be evil and had become intent on world domination.
This commit throws disc of Tron into MCP (causing its deresolution)
and turns it back into a chess game.
```

```
$ git log --oneline
42e769 Derezz the master control program
```

# Git 커맨드 예제



```
git add main.py sub1.py
git commit -m 'The initial commit of my project'
```

\* 여러 줄의 커밋 설명을 쓰고자 할 때는 `git commit`  
→ 기본 텍스트 에디터가 열림 → 설명 작성 후 빠져나옴

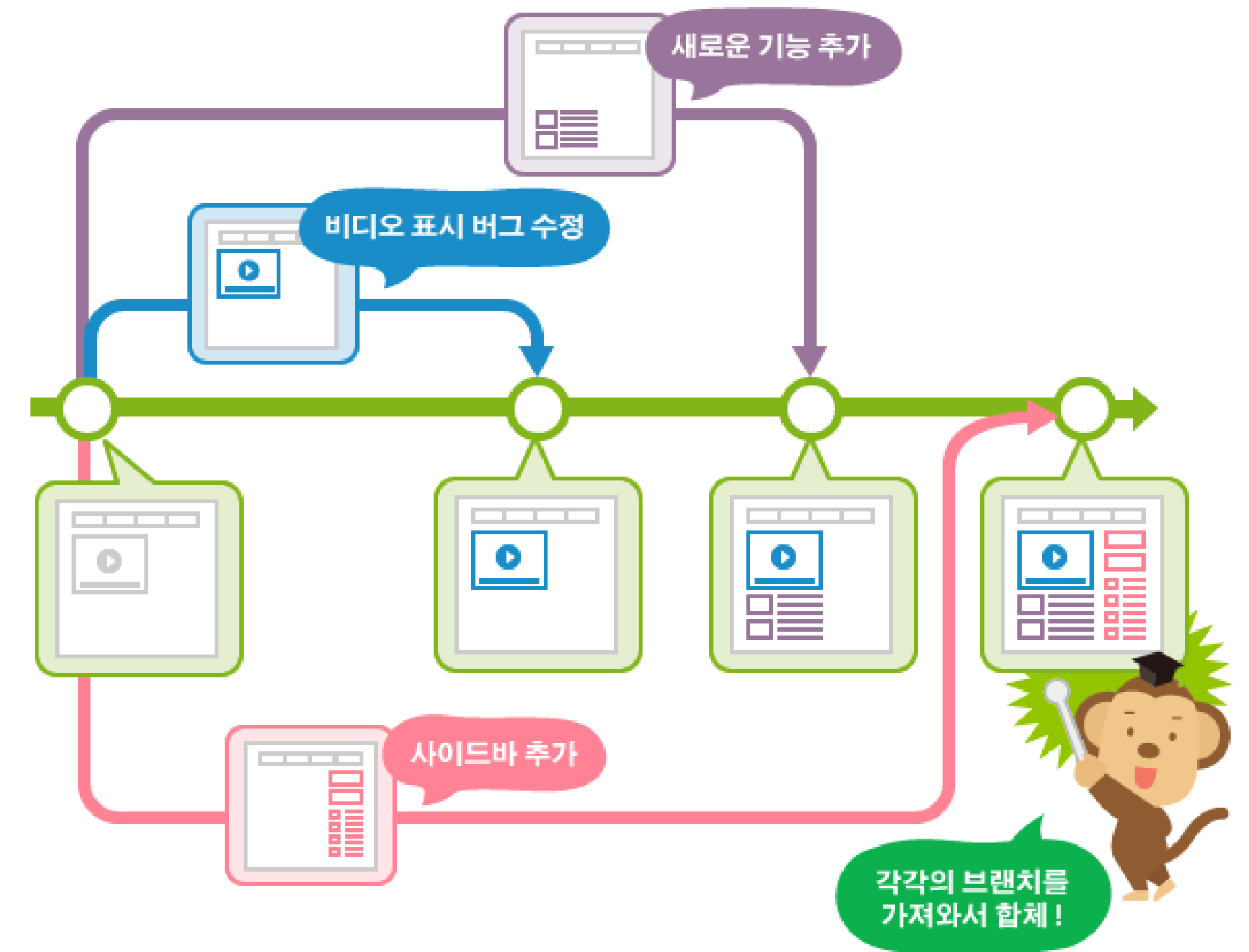
# Git 커맨드 예제

- 다양한 add 옵션 (<https://git-scm.com/docs/git-add>)

<code>git add -A</code>	모든 파일들을 준비 영역에 추가
<code>git add .</code>	모든 수정된, 새로운 파일들을 준비 영역에 추가 (삭제된 파일 제외)
<code>git add -u</code>	모든 수정된, 삭제된 파일들을 준비 영역에 추가 (새로운 파일 제외)

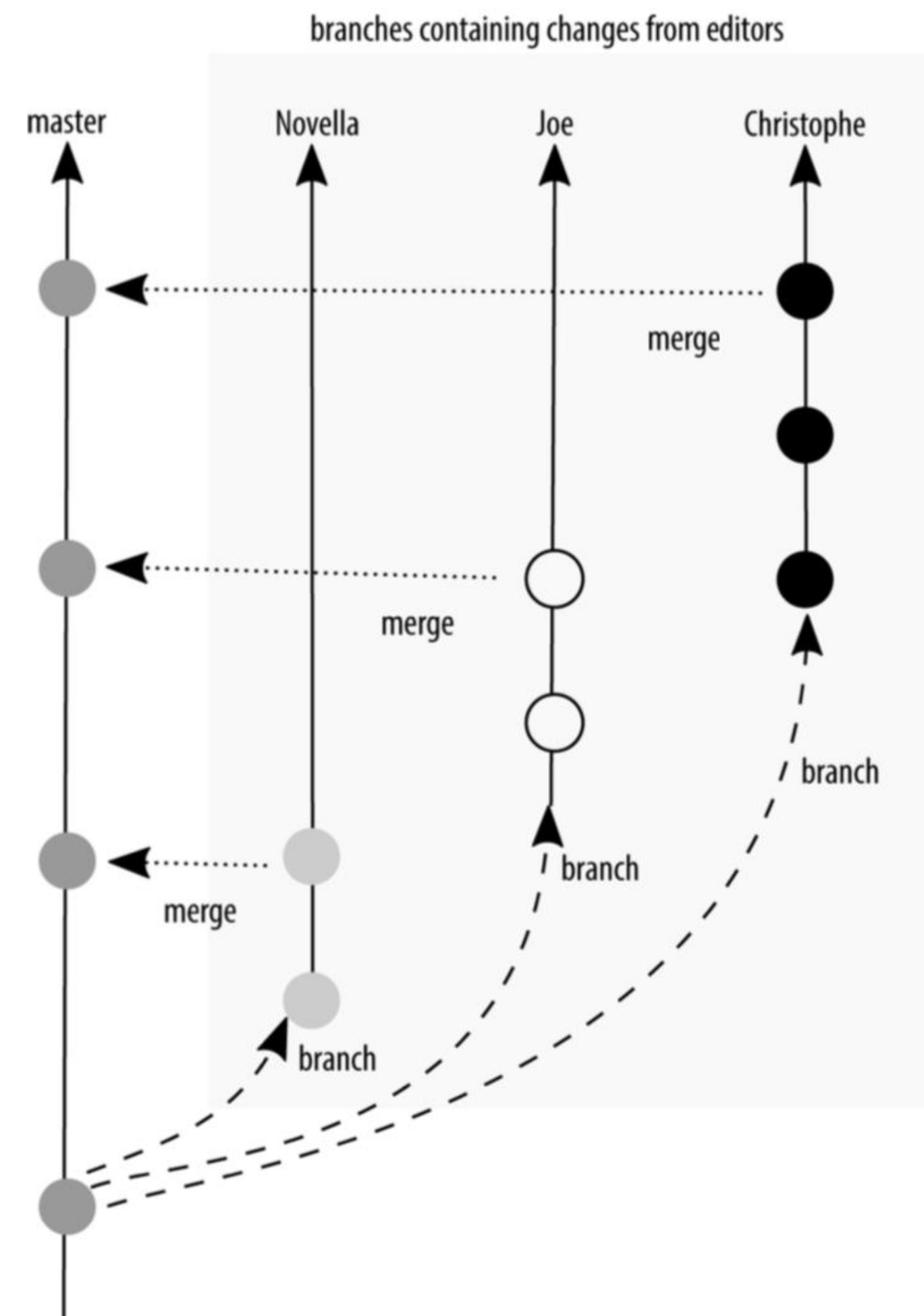
# 브랜치(Branch)

- 여러 작업을 나누어 처리하고자 할 때,  
여러 사람이 따로 작업할 때 유용
- 일반적인 방식:  
메인 브랜치 +  
작업자/기능에 따라 새로운 브랜치 생성 후  
작업 완료 시 메인 브랜치에 병합(merge)

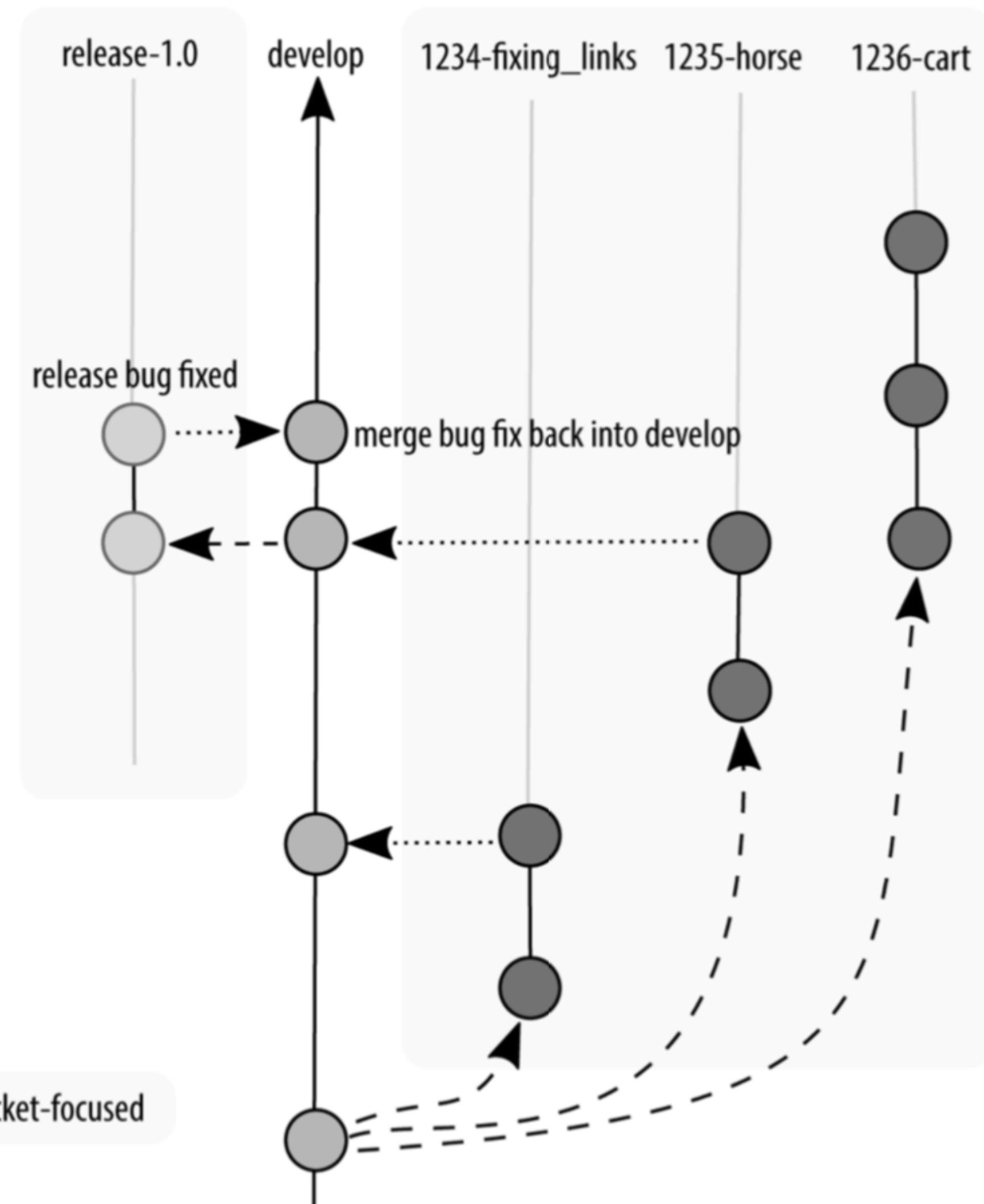




# 브랜치(Branch)

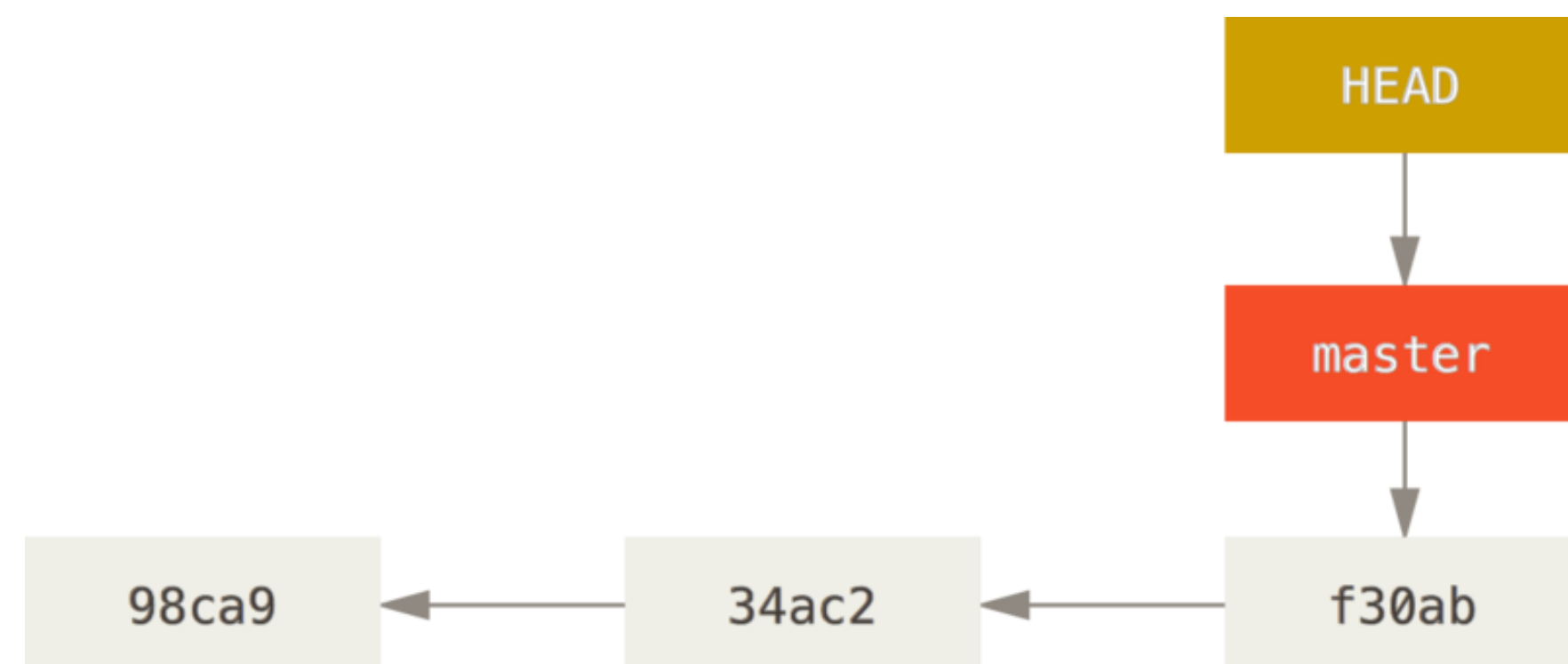


branch - - - ->  
merge .....>  
short-lived branches; usually ticket-focused



# 브랜치(Branch)

- master 브랜치: 처음 저장소를 생성하면 만들어지는 브랜치  
일반적으로 메인 브랜치로 사용
- 체크아웃(checkout): 현재 위치 변경 (브랜치 이동, 커밋 단위로 이동 등)
- HEAD: 현재 위치를 가리키는 포인터  
일반적으로 사용 중인 브랜치의 마지막 커밋을 가리키고 있음

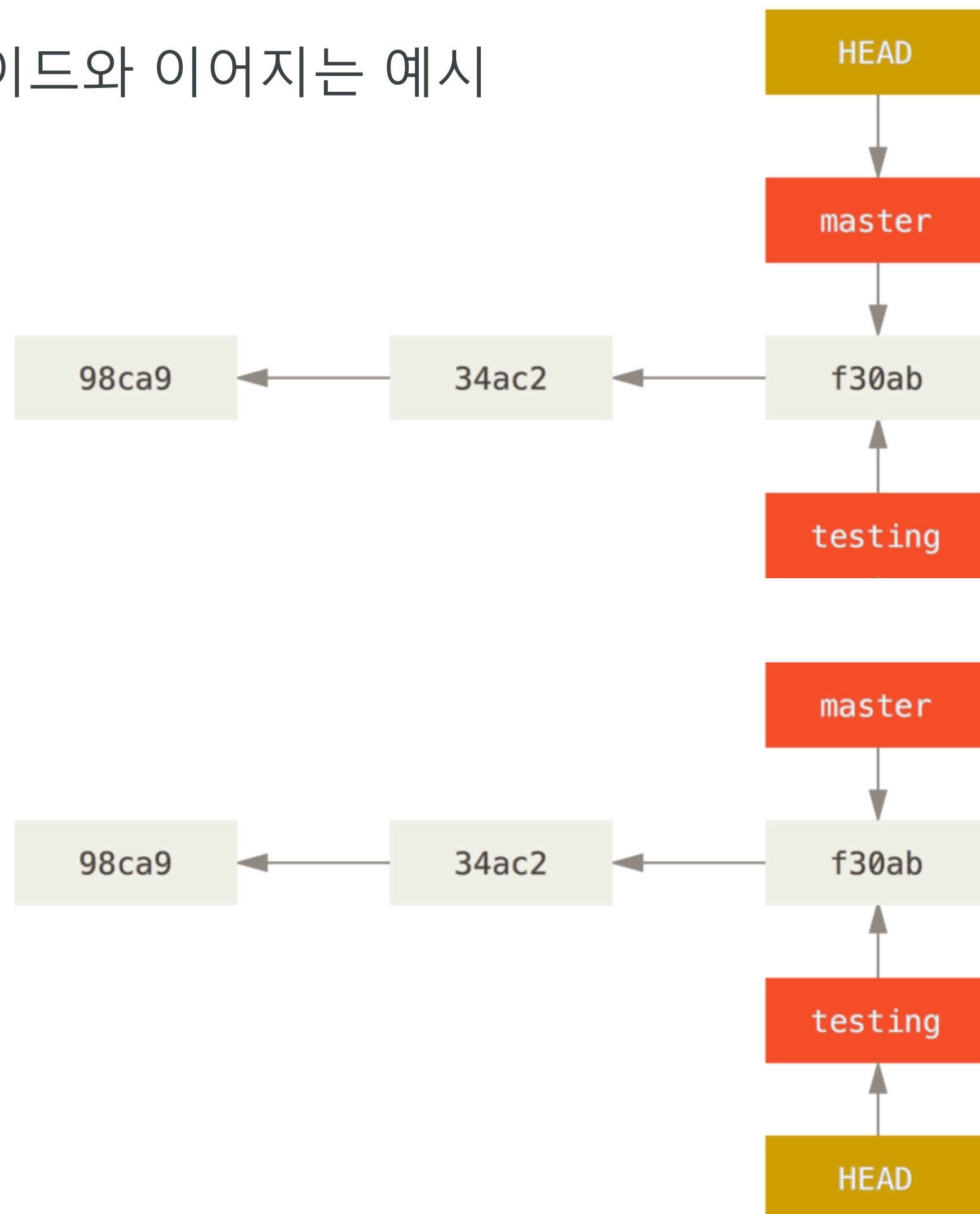


예시)

1. master 브랜치에서 커밋을 세 번 한 상태  
\* 각 커밋은 이전 커밋을 가리키도록 표현되어 있음

# 브랜치 예제

\* 앞 슬라이드와 이어지는 예시



2. 새로운 브랜치(testing)를 만듦

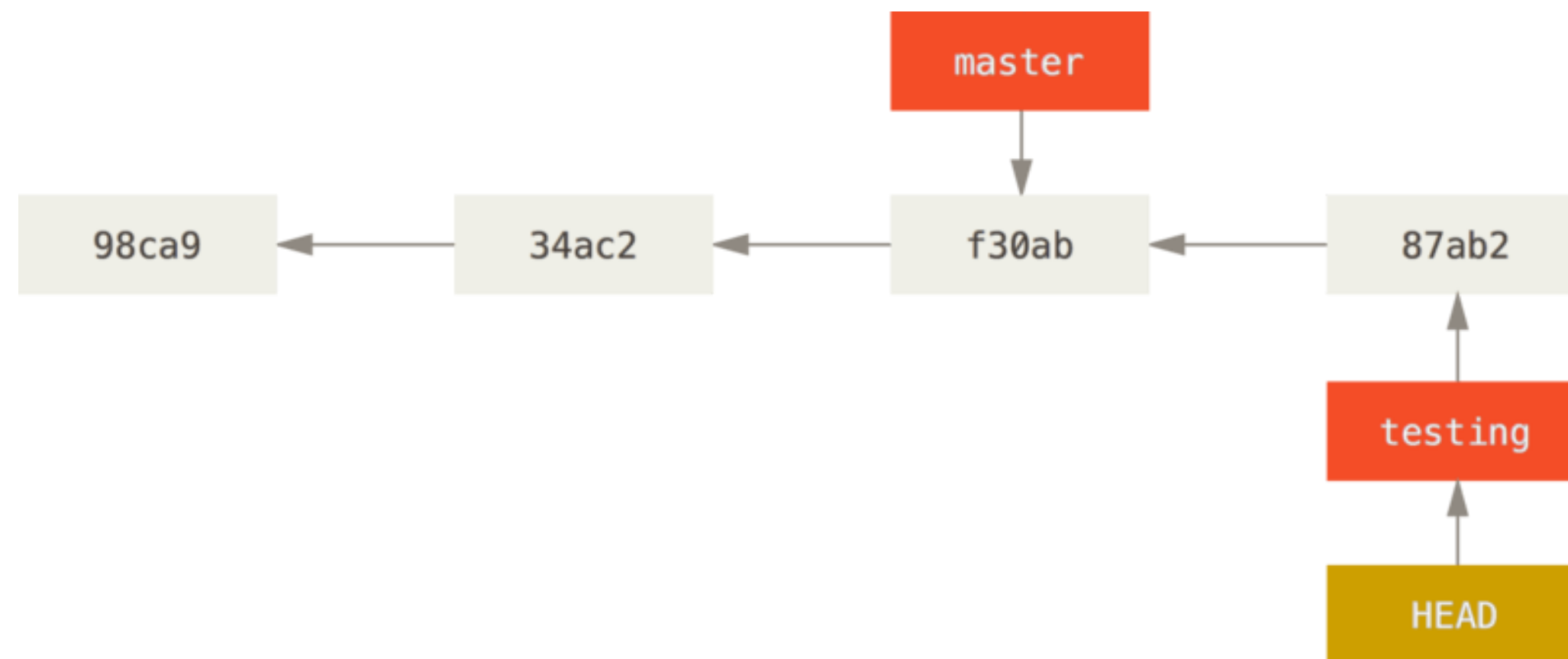
```
git branch testing
```

3. testing 브랜치로 체크아웃(이동)

```
git checkout testing
```

\* 위 두 작업(만들고 이동)을 한 번에 하려면  
`git checkout -b testing`

# 브랜치 예제



## 4. testing 브랜치에서 새로 커밋

```
(modify file1.py)  
git add file1.py  
git commit -m 'Modified file1'
```

## 5. master 브랜치로 다시 체크아웃

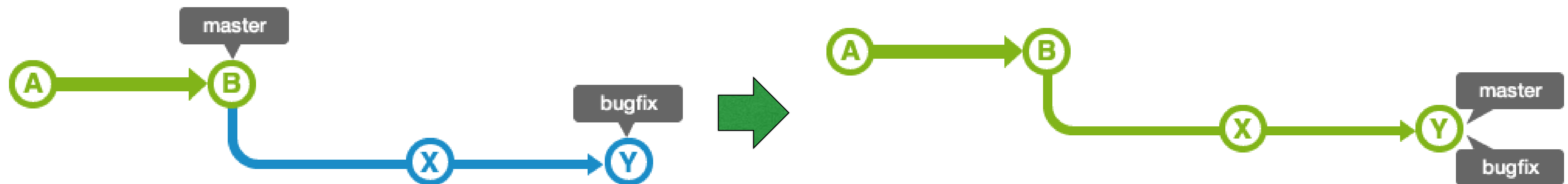
```
git checkout master
```

**주의:** 브랜치를 이동하면 워킹 디렉토리의 파일이 그 브랜치에서 가장 마지막으로 했던 작업 내용으로 변경됨.

# 브랜치 병합

## 1. Fast-forward

브랜치 A, B를 병합할 때, 브랜치 B가 A의 최신 커밋으로부터 만들어진 경우  
브랜치 A가 동일한 커밋을 가리키도록 포인터 이동만 하면 됨



```
git checkout master  
git merge bugfix
```

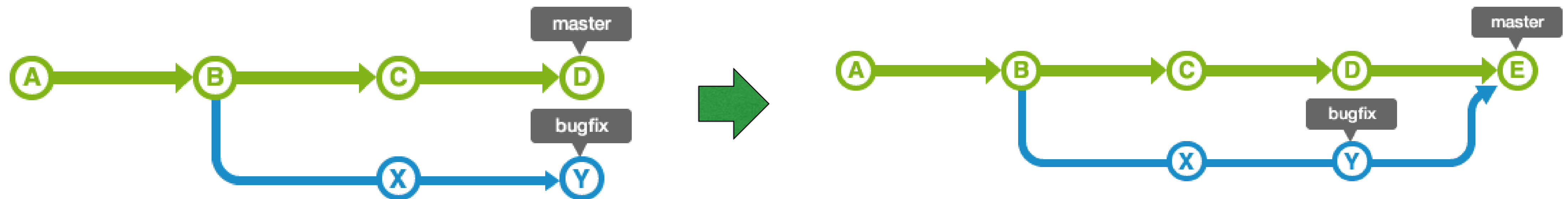
# 브랜치 병합

## 2. 3-way merge

브랜치 A, B를 병합할 때, 두 브랜치가 공통 조상 커밋을 가지는 경우

병합 결과를 새로운 커밋으로 만들고 포인터 이동

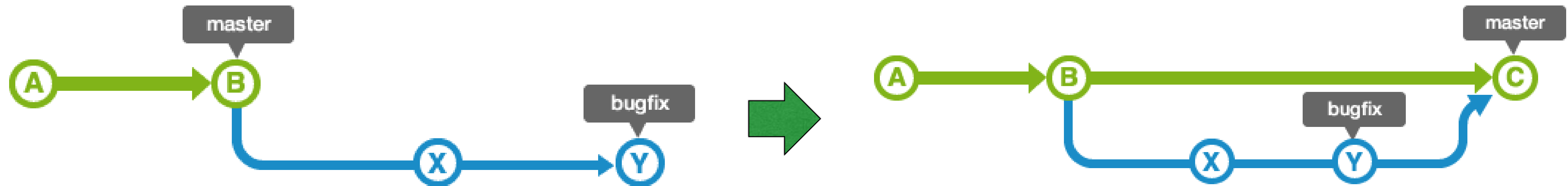
브랜치가 갈라진 뒤 두 브랜치에서 같은 파일을 수정했다면 충돌이 발생할 수 있음



```
git checkout master
git merge bugfix
```

# 브랜치 병합

참고: fast-forward가 가능한 경우에도 옵션을 통해 병합 커밋을 만들 수 있음

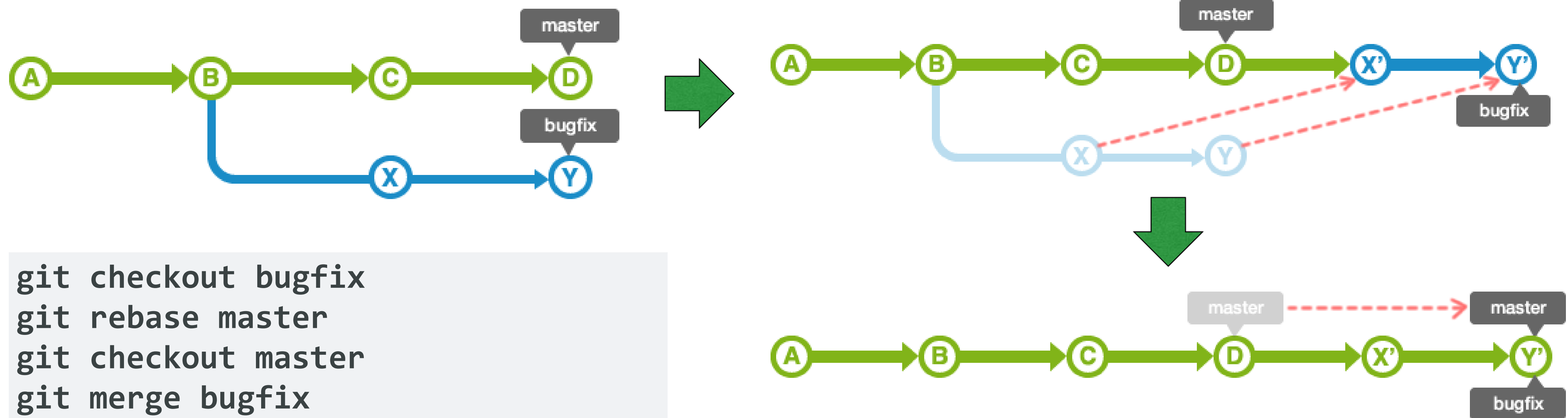


```
git checkout master  
git merge --no-ff bugfix
```



# 브랜치 병합 (rebase)

- Rebase는 두 브랜치의 커밋 히스토리까지 통합
- 아래 커밋 x, y의 내용이 c, d와 다른 경우 충돌이 발생할 수 있음



# 기존 커밋 수정하기 (--amend)

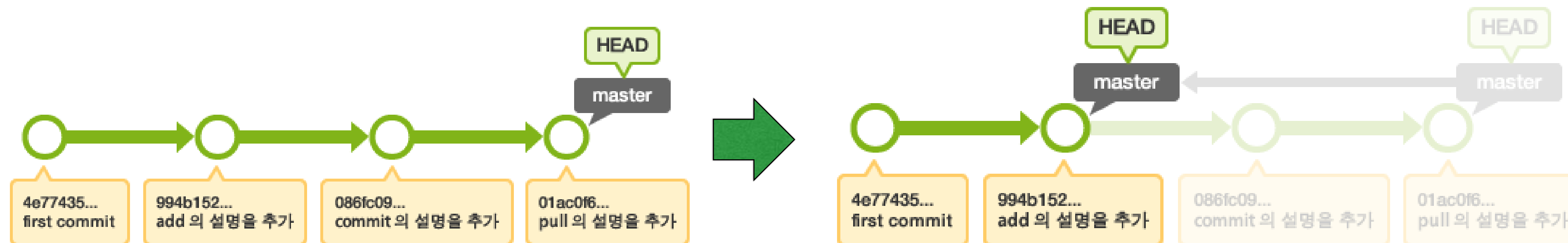
- 기존 커밋에 파일 추가 혹은 커밋한 파일을 추가로 수정하고자 할 때
- 기존 커밋의 설명을 변경하고자 할 때



```
(sample.txt 추가 수정)
git add sample.txt
git commit --amend
(commit 설명도 수정)
```

# 기존 커밋으로 되돌아가기 (reset)

- Reset(hard) 사용 시 HEAD, 준비 영역, 워킹 디렉토리의 상태를 전부 해당 커밋 시의 상태로 리셋

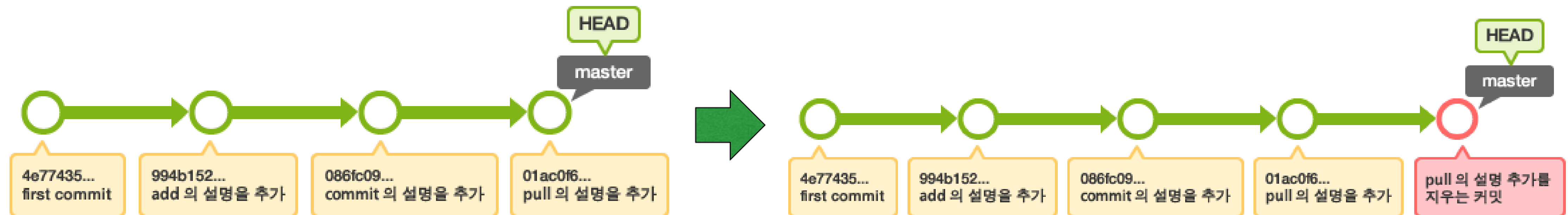


```
git reset --hard HEAD~~
```

옵션	HEAD 위치	준비 영역	워킹 디렉토리
soft	변경	변경 안 함	변경 안 함
mixed	변경	변경	변경 안 함
hard	변경	변경	변경

# 기존 커밋 지우기 (revert)

- Revert를 사용 시 원하는 커밋을 지운 새로운 커밋이 생성됨
- 다른 사람과 같이 작업할 때 사용하기에 reset보다 안전한 방법



```
git revert HEAD
```

이전 커밋을 가리키는 방법:

HEAD^, HEAD~, or HEAD~1 (HEAD의 바로 앞 커밋)

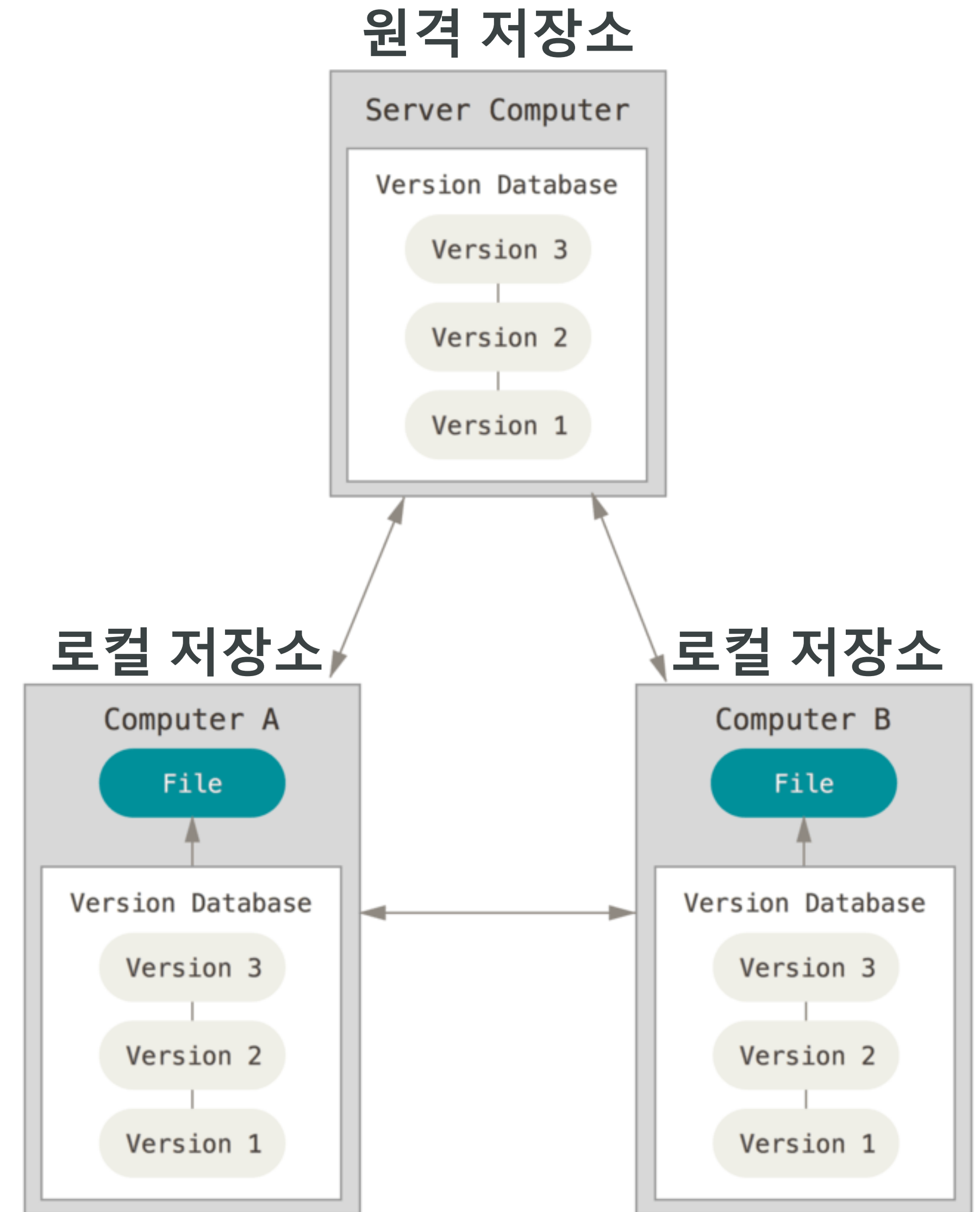
HEAD^^, HEAD~~, or HEAD~2 (HEAD의 두번째 앞 커밋)



# GITHUB 기초

# Git이란

- 분산 버전 관리 시스템
- **Issue 2. 원격-로컬 저장소**
  - **원격 저장소**(Remote repository):  
여러 사람이 공유하는 저장소
  - **로컬 저장소**(Local repository):  
프로젝트에 참여하는 개개인 혹은  
한 사람의 여러 컴퓨터에 존재하는 저장소
  - 원격 → 로컬로 버전 히스토리까지  
전부 복제하여 저장



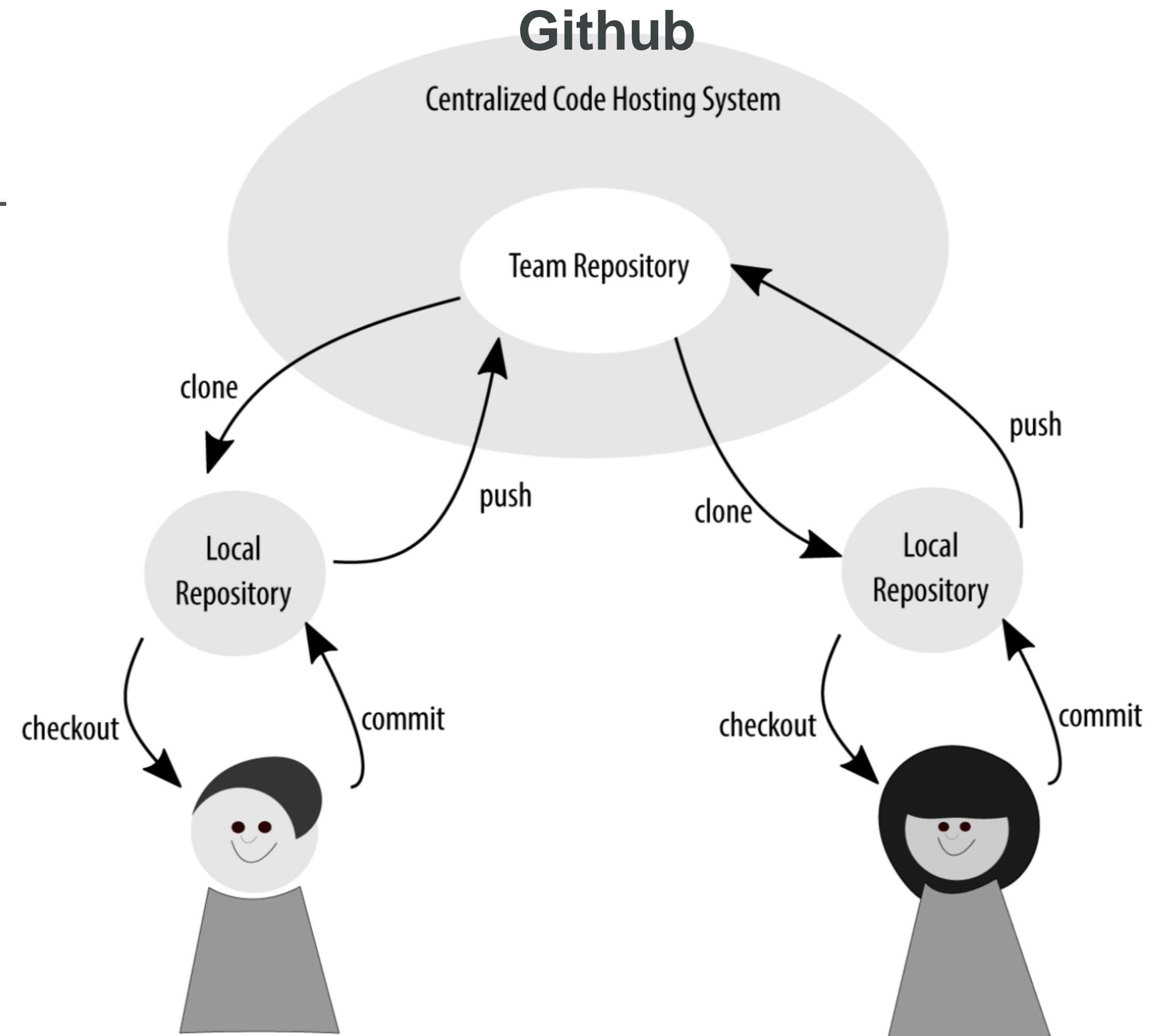
# Github이란

- Git 저장소 호스팅 (서버를 이용할 수 있도록 빌려줌)
- Github 계정에
  - 1) 여러 사람이 공유하는 저장소, 2) 본인 개인 저장소 등을 만들어 사용
- 여러 사람의 공동 프로젝트 등에 유용함
- 1인 프로젝트의 경우 백업 저장소로 사용할 수 있음



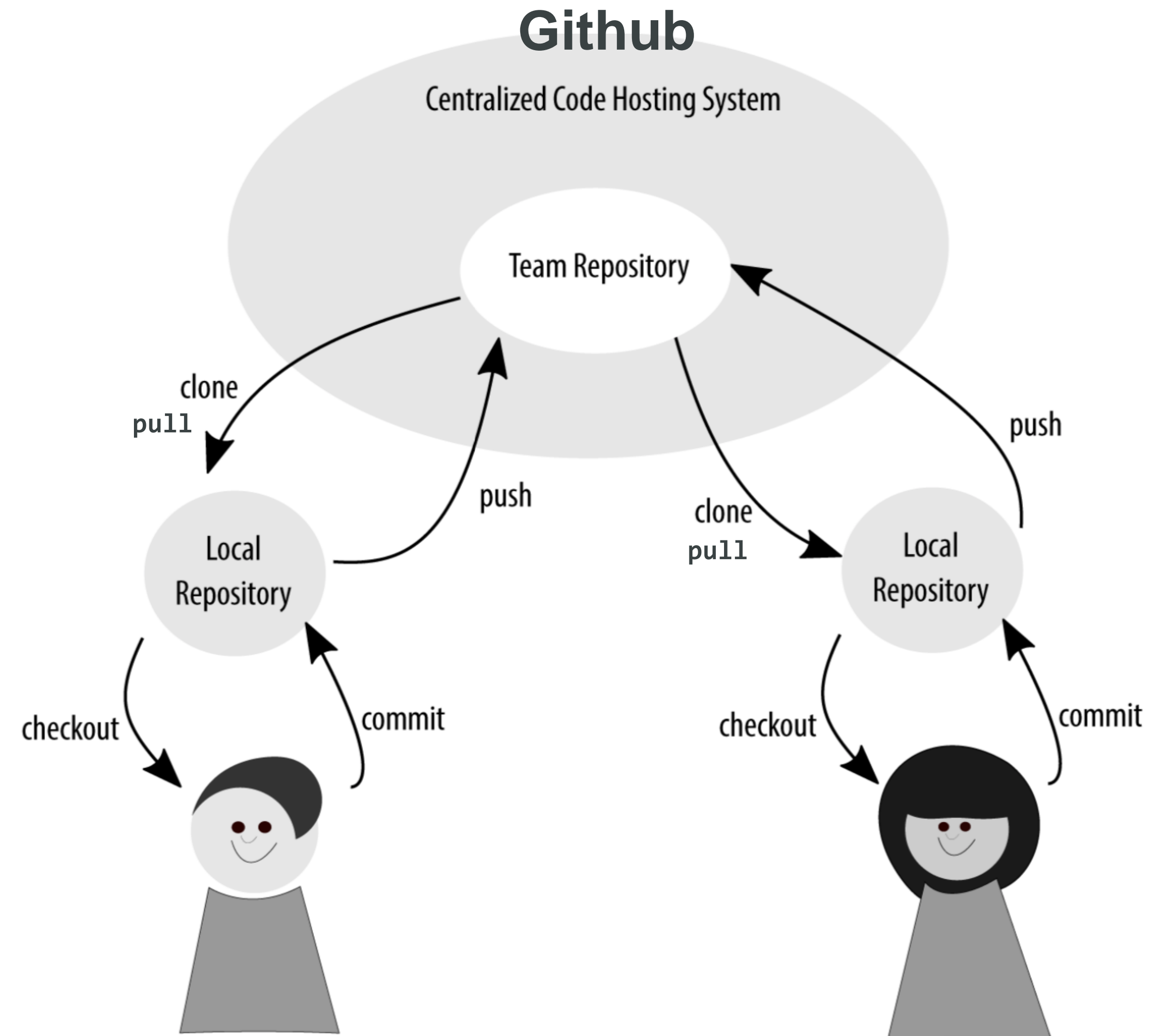
# Github이란

- 예1) 팀이 사용하는 공동 저장소가 있고, 각 팀원이 본인의 로컬 저장소에서 나눠서 작업하는 경우



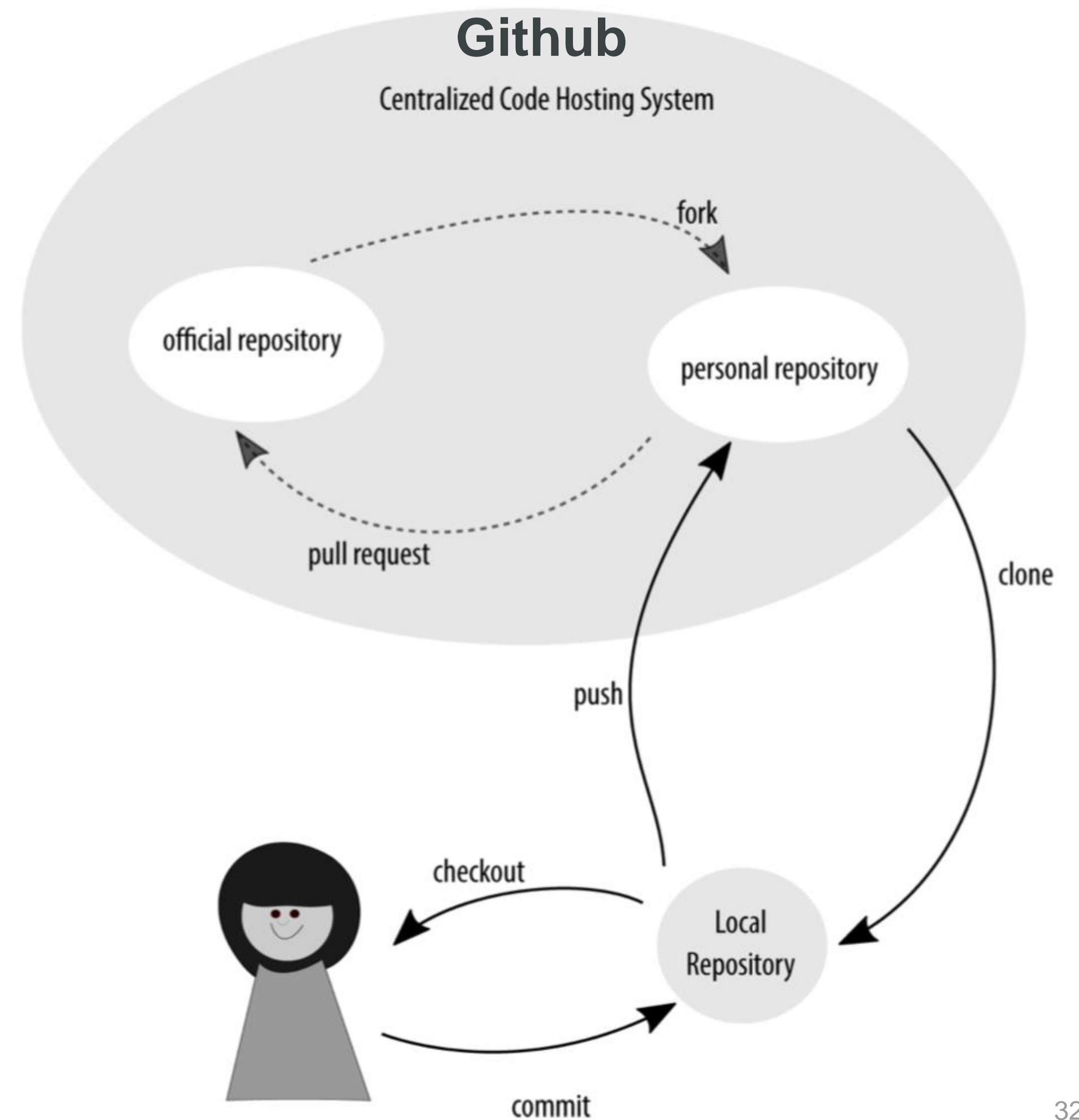
# Github이란

- 원격 저장소 → 로컬 저장소로  
저장소 복사: **clone**
- 로컬 저장소 → 원격 저장소로  
수정 사항 업데이트: **push**
- 원격 저장소의 새로운 업데이트를  
(다른 사람의 수정 내용 등) 로컬  
저장소에 적용: **pull**



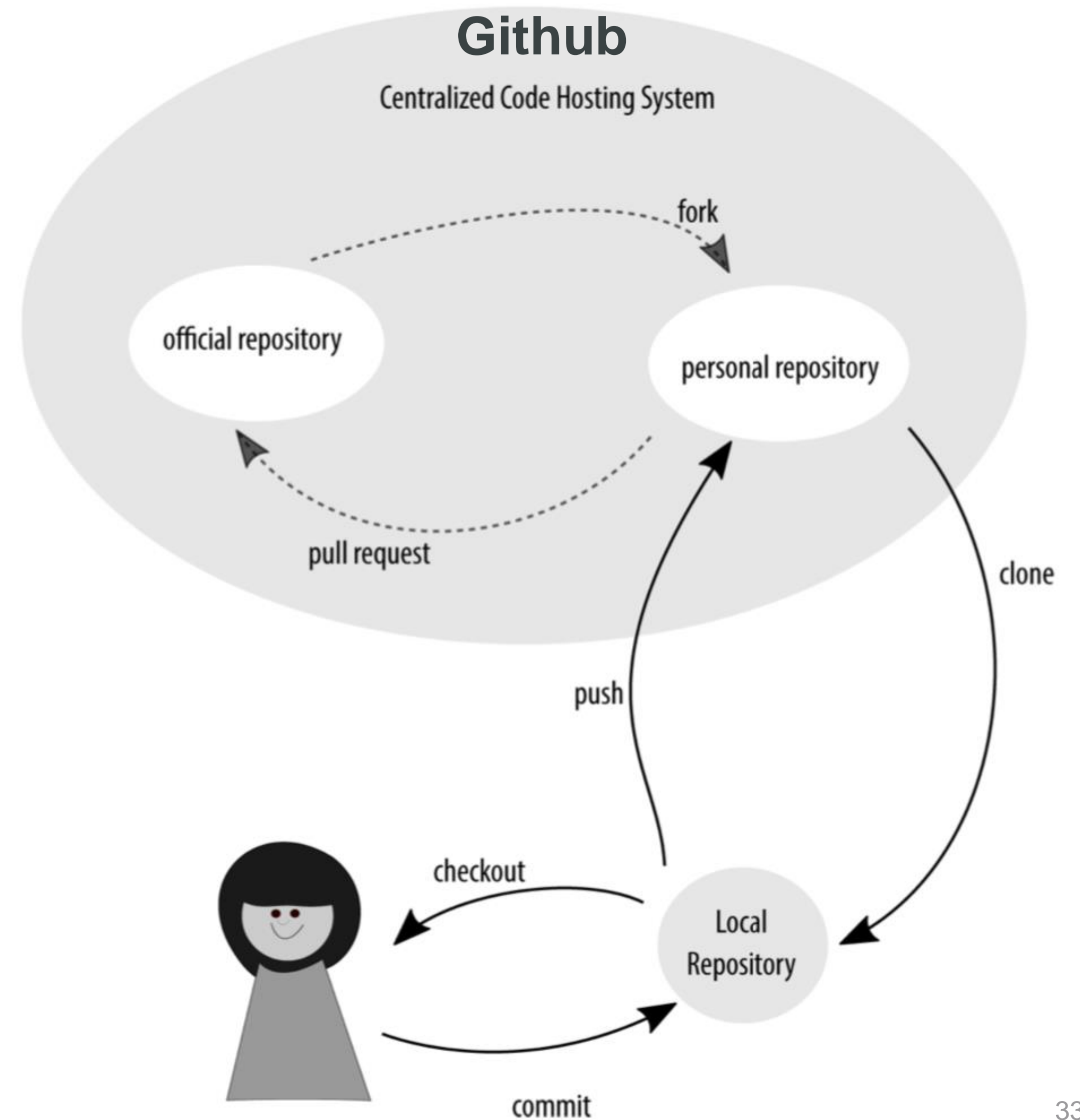
# Github이란

- 예2) 사용자에게 공개되는 프로그램A의 공식 저장소가 있고(open source!), 개발자B가 본인의 Github 개인 저장소에 이를 복사한 뒤, 로컬 저장소로 가져와서 작업하는 경우



# Github이란

- Github의 저장소 간의 복사: **fork**
- Github (개인)저장소에 나의 수정사항을 다른 Github 저장소에 반영하고 싶은 경우: **pull request** (open source에 기여!)
- Pull request 시 다른 저장소의 주인이 허가하면 수정사항이 반영됨



# 로컬 저장소를 만드는 세 가지 방법

- (git 설치 및 기본 설정이 되어있다고 가정)
- 1. 기존 원격 저장소의 내용을 통째로 복사해오기(clone)
- 2. 작업 해놓은 폴더를 새로운 저장소로 설정
- 3. 새로운 폴더를 만들고 저장소로 설정

# 로컬 저장소를 만드는 세 가지 방법

## 1. 기존 원격 저장소의 내용을 통째로 복사해오기(clone)

- Github 저장소에 있는 clone or download 버튼 → URL 복사
- 로컬 작업 폴더에서 `git clone <URL>` 하면 저장소의 모든 파일과 히스토리가 복사됨
- 예) 참고 사이트3의 튜토리얼 저장소

```
git clone https://monkey.backlogtool.com/git/BLG/tutorial.git
```



# 로컬 저장소를 만드는 세 가지 방법

pandas-dev / pandas

Sponsor Used by 154k Watch 1k Star 22.4k Fork 8.9k

Code Issues 3,172 Pull requests 144 Actions Projects 3 Wiki Security Insights

Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more <https://pandas.pydata.org>

data-analysis pandas flexible alignment python

20,815 commits 11 branches 0 packages 108 releases 1,692 contributors BSD-3-Clause

Branch: master New pull request Create new file Upload files Find file Clone or download

jbrockmendel and gfyong DEPR: remove Index.summary (#29807)

.github	CI: GitHub action for checks (linting, docstrings...) (#295)
LICENSES	Add reader for SPSS (.sav) files (#26537)
asv_bench	F-string (#29663)
ci	DEPR: enforce deprecations in core.internals (#29723)

Clone with HTTPS ? Use SSH

Use Git or checkout with SVN using the web URL.

`https://github.com/pandas-dev/pandas.git`

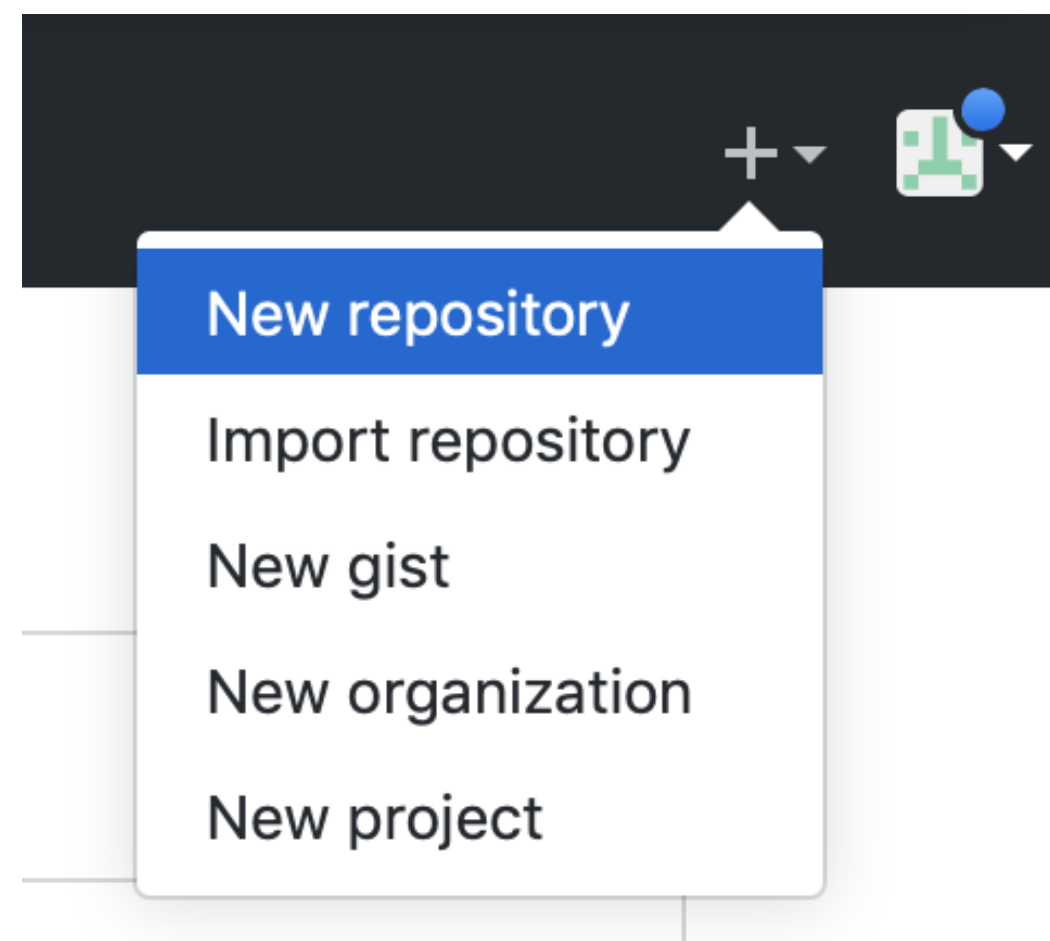
Open in Desktop Download ZIP



# 로컬 저장소를 만드는 세 가지 방법

## 1-2. 원격 저장소를 Github에서 새로 만들고 복사해오기

- Github 계정 만들기
- 본인 페이지 (<http://github.com/아이디>) 상단의 new repository 선택





# 로컬 저장소를 만드는 세 가지 방법

## 1-2. 원격 저장소를 Github에서 새로 만들고 복사해오기

- 저장소 이름과 공개 여부 설정
- README 파일, .gitignore 파일, 라이선스 파일 생성 여부 설정

- .gitignore: git add -A 와 같이 모든 파일을 준비 공간에 추가하는 작업을 실행해도 특정 파일(로그, 임시파일 등)은 추가 되지 않도록 설정하는 파일

☒  **Public**  
Anyone can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

---

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▼ | Add a license: None ▼ ⓘ

# 로컬 저장소를 만드는 세 가지 방법

## 1-2. 원격 저장소를 Github에서 새로 만들고 복사해오기

- 만들어진 저장소 페이지에서 URL 복사
- 로컬 작업 폴더에서 `git clone <URL>`
- README, .gitignore, 라이선스 파일 등이 로컬에 복사됨

# 로컬 저장소를 만드는 세 가지 방법

## 2. 작업 해놓은 폴더를 새로운 저장소로 설정

- 나의 작업 폴더로 이동하여 `git init` 수행
- `git status`를 사용하면 현재 상태를 확인할 수 있음

```
git init
```

```
git status
```

```
Initialized empty Git repository in /Users/emmajane/gitforteam5/gitforteam5-zip/.git/
```

```
On branch master
```

```
Initial commit
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
[ lots of files listed here ... ]
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

# 로컬 저장소를 만드는 세 가지 방법

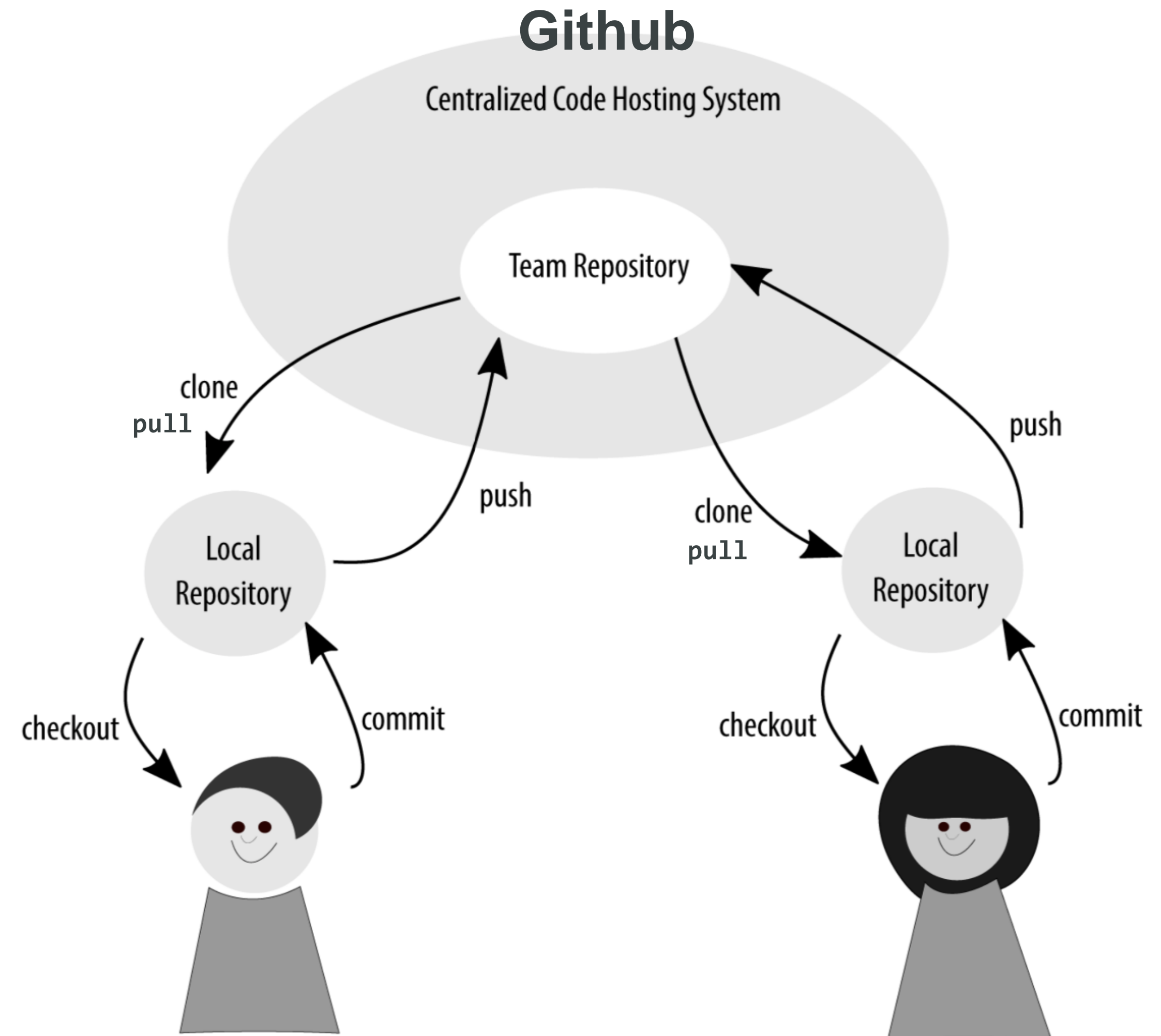
## 3. 새로운 폴더를 만들고 저장소로 설정

- 원하는 위치에 폴더 만들기
- 이동하여 `git init` 수행

```
mkdir <repository name>  
mv <repository name>  
git init
```

# Github이란

- 원격 저장소 → 로컬 저장소로  
저장소 복사: **clone**
- 로컬 저장소 → 원격 저장소로  
수정 사항 업데이트: **push**
- 원격 저장소의 새로운 업데이트를  
(다른 사람의 수정 내용 등) 로컬  
저장소에 적용: **pull**



# 로컬 저장소와 원격 저장소 연결

- `remote add`

연결할 원격 저장소의 별명을 정하고, URL 정보를 입력

- 별명은 디폴트로 `origin`을 많이 사용 (`push/pull` 시 디폴트 값)

```
git push <repository name> <branch name>
```

# 로컬 저장소와 원격 저장소 연결

- push

로컬 저장소의 수정사항을 원격 저장소에 업데이트

원격 저장소의 별명과 업데이트할 브랜치 이름을 명시

```
git push <repository name> <branch name>
```



# 로컬 저장소와 원격 저장소 연결

- `pull (fetch + merge)`

원격 저장소에는 있지만 로컬에는 없는 커밋들 다운로드하고(fetch),  
원격 & 로컬에서 이름이 같은 브랜치를 병합(merge)

원격 저장소의 별명과 업데이트할 브랜치 이름을 명시  
원격 저장소 아이디, 비밀번호 입력

```
git pull <repository name> <branch name>
```

# 로컬 저장소와 원격 저장소 연결

- 전체 브랜치 목록을 확인해보면, 로컬과 원격 저장소의 모든 브랜치 이름을 볼 수 있음
- \*가 붙은 것이 현재 브랜치,  
원격 저장소의 브랜치는 `remotes/별명/브랜치` 로 표현되어 있음

```
git branch --all
```

```
* master  
remotes/origin/master  
remotes/origin/sandbox  
remotes/origin/video-lessons
```

# Git 브랜치 실습

- 다양한 git 명령어 실습 <https://learngitbranching.js.org>
- commit, branch, checkout 등
- 메인 1~8, 원격 1~4

# 참고 사이트

- 이미지 및 코드 예제는 아래 사이트와 책에서 발췌

1. Pro Git 2 :

<https://git-scm.com/book/ko/v2>

2. 팀을 위한 Git, 엠마 제인 호그빈 웨스트비, 한빛미디어

3. 누구나 쉽게 이해할 수 있는 Git 입문

<http://backlog.com/git-tutorial/kr>

**Q&A**

