

svn 능력자를 위한 git 개념 가이드



svn을 능숙하게 다루던 능력자들
처음 git을 만나면 대개 이런 표정이죠.



하지만 곧 이렇게 됩니다.



<http://ggamangi.tistory.com> 블로그의 <민찬> 어린이입니다.

git은 svn과 비슷해 보이지만 사실 상당히 다릅니다.
그래서 막상 덤벼보면 아리송한게 한 두가지가 아닙니다.

git != svn

주위에서 흔히 볼 수 있는 git 가이드들은 무척 친절합니다.
하지만, svn 숙련자들에게는 오히려 혼란스럽습니다.

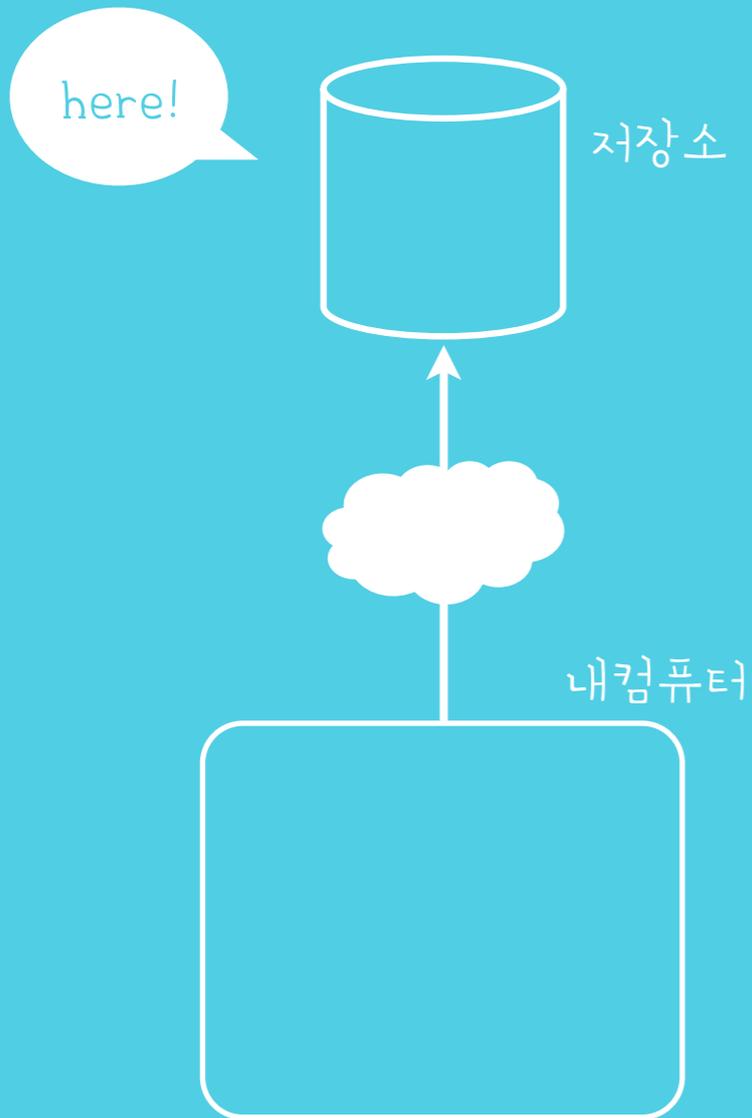


지금부터 svn을 기준으로 git을 살펴보겠습니다.

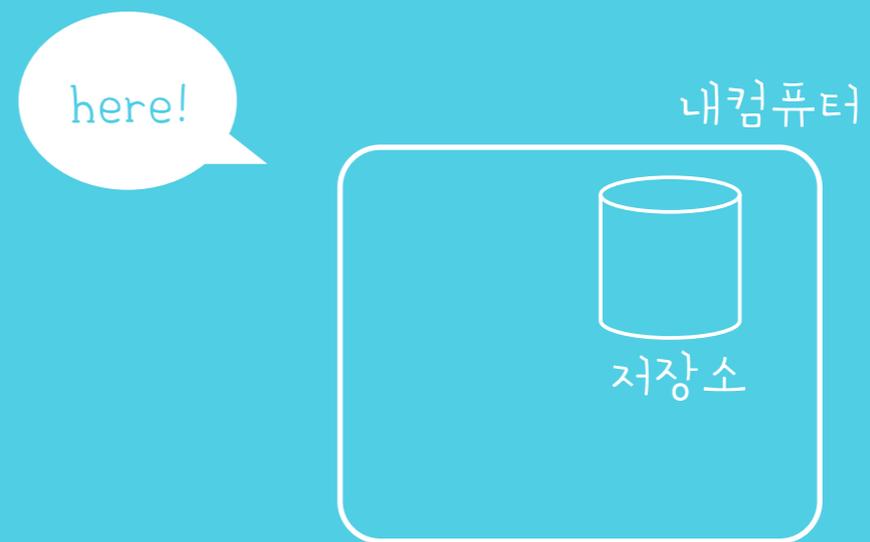


git

svn은 보통 저장소가
서버에 있습니다.

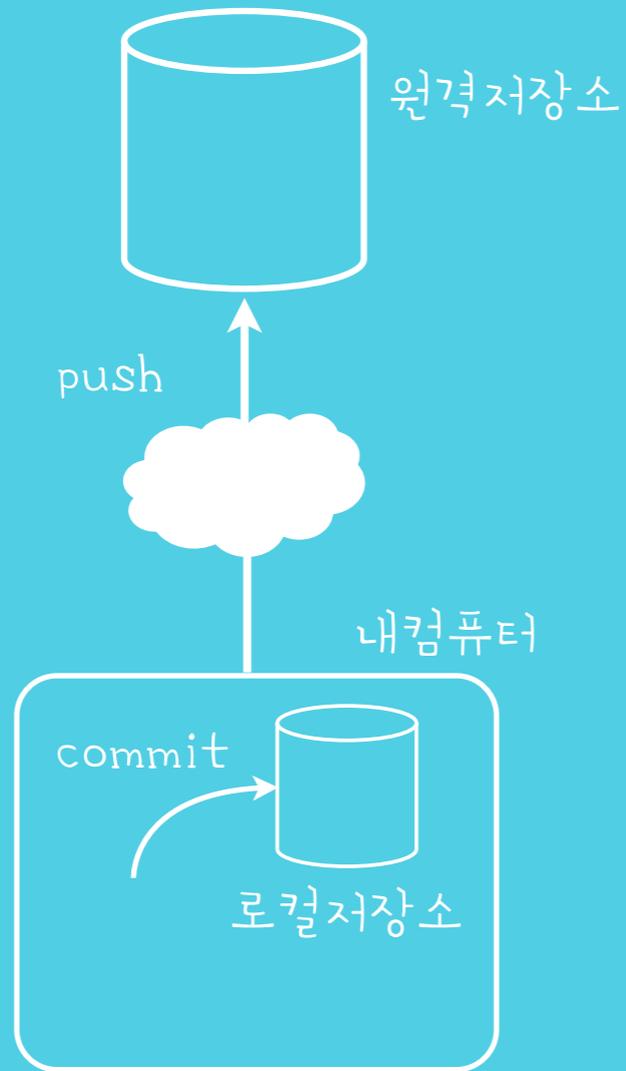


git은 저장소가
내컴퓨터에 있습니다. 응?



그럼 git은 다른 사람들과 작업을 할 수 없나요?

원격저장소를 만들면 됩니다.
remote repository



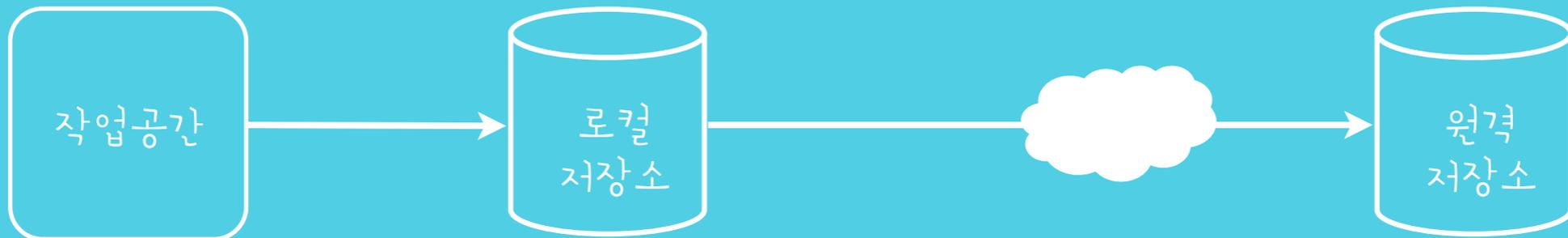
② 다른 사람과 공유할 때 원격 저장소에 푸시합니다.

① 내 컴퓨터의 저장소에 열심히 작업 내용을 커밋하고

svn



git

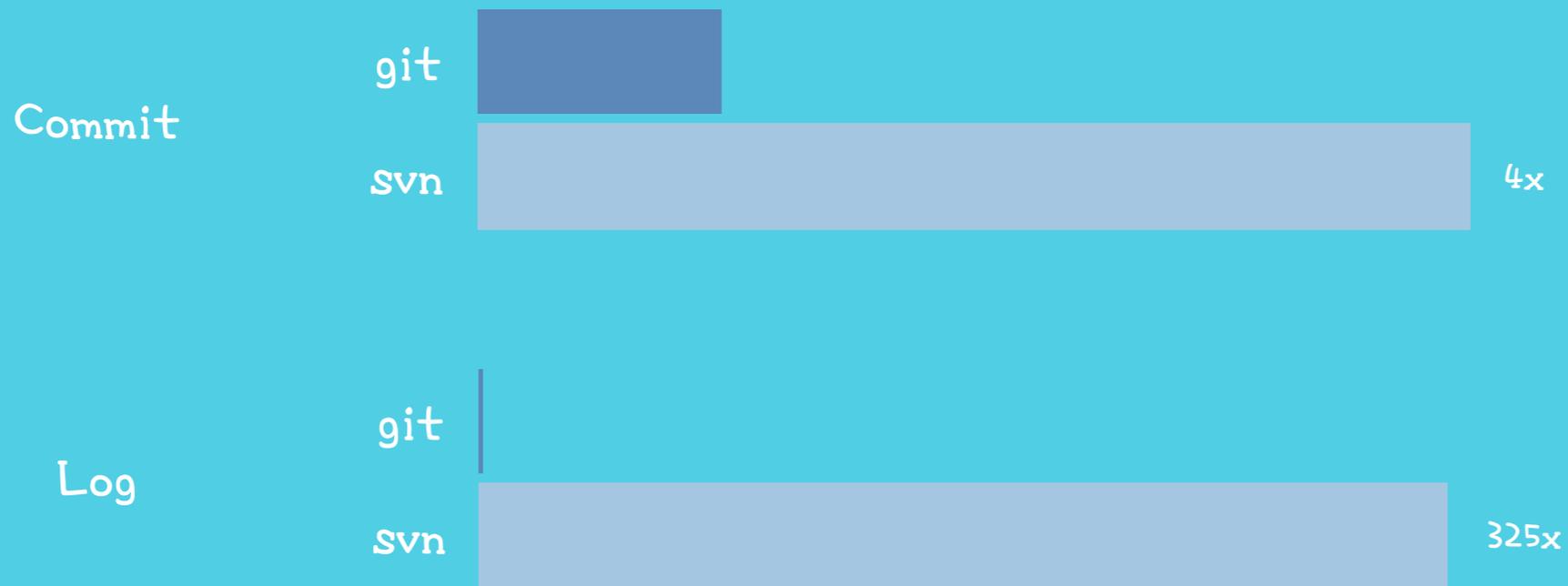


이렇게 저장소가 분산되는 구조를 분산버전관리시스템(DVCS)라고 합니다.

로컬저장소가 따로 있으면 어떤 **장점**이 있을까요?

엄청나게 빠릅니다.

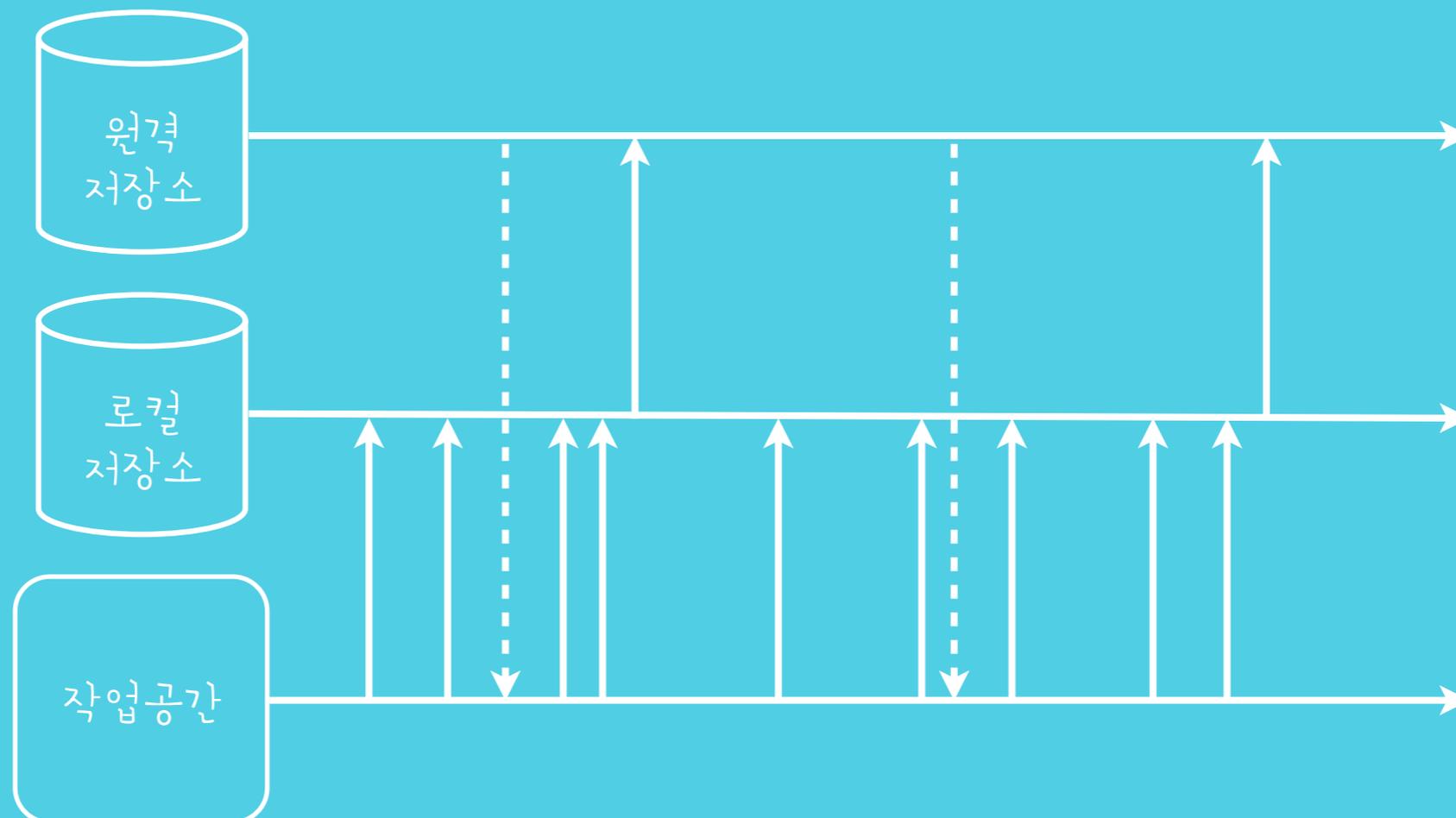
인터넷을 경유할 필요가 없기 때문에 훨씬 빠릅니다.



git 공식홈페이지의 벤치마크 결과를 참고하였습니다.

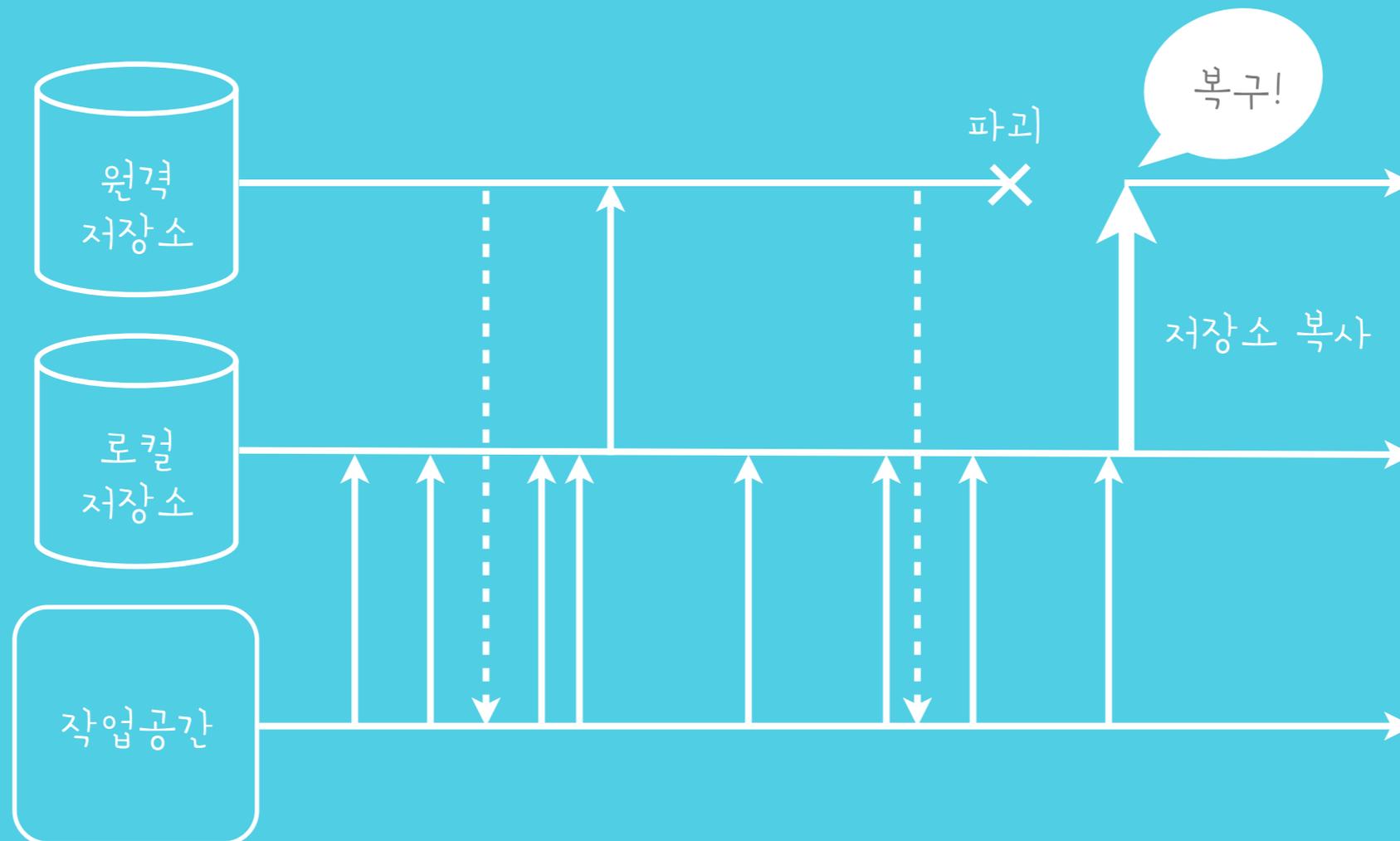
커밋에 부담이 없습니다.

내 로컬 저장소에서 마음대로 실험하고 테스트하세요.



원격저장소가 폭파되어도 로컬저장소로 복원이 가능합니다.

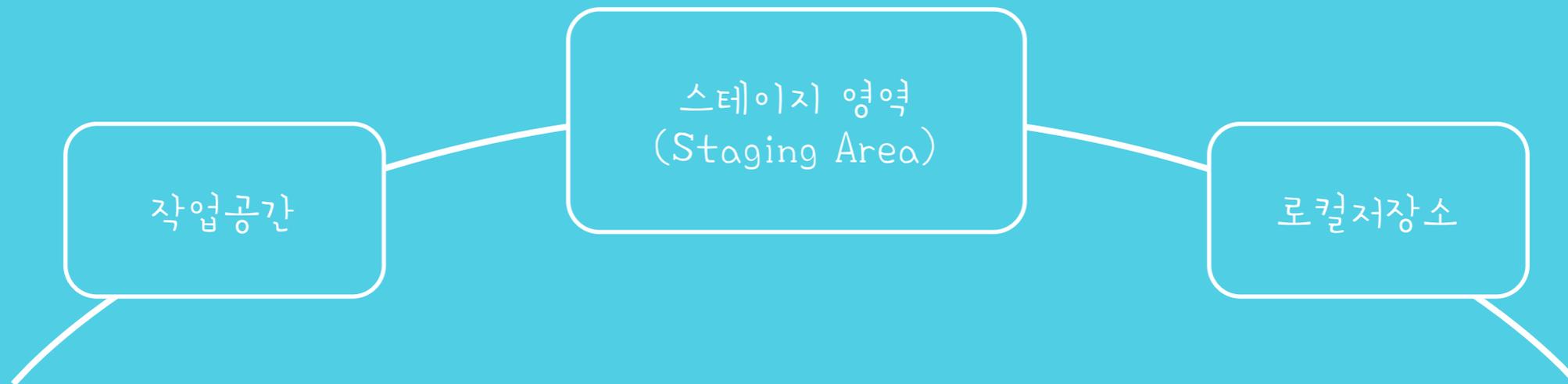
원격저장소와 연결된 모든 로컬저장소는 사본을 보유하고 있습니다.



그런데 git에는 로컬저장소에 커밋 전, 하나의 단계가 더 있습니다.

바로 **스테이지 영역**입니다.

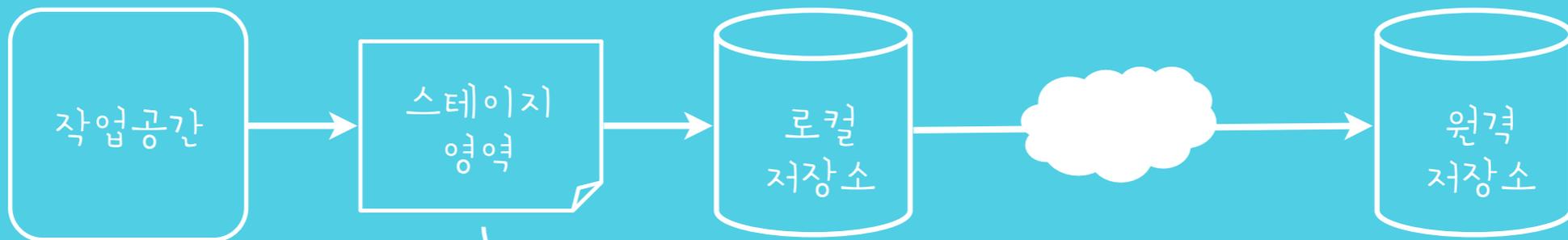
인덱스(index)라고도 부릅니다.



svn



git



커밋 할 파일들을 스테이지 영역에 추가합니다.

스태이지 영역이 어떤 역할을 하는지 알아보을까요?

① 이런 빌드 목표를 가지고 작업을 하고 있습니다.

빌드 목표!

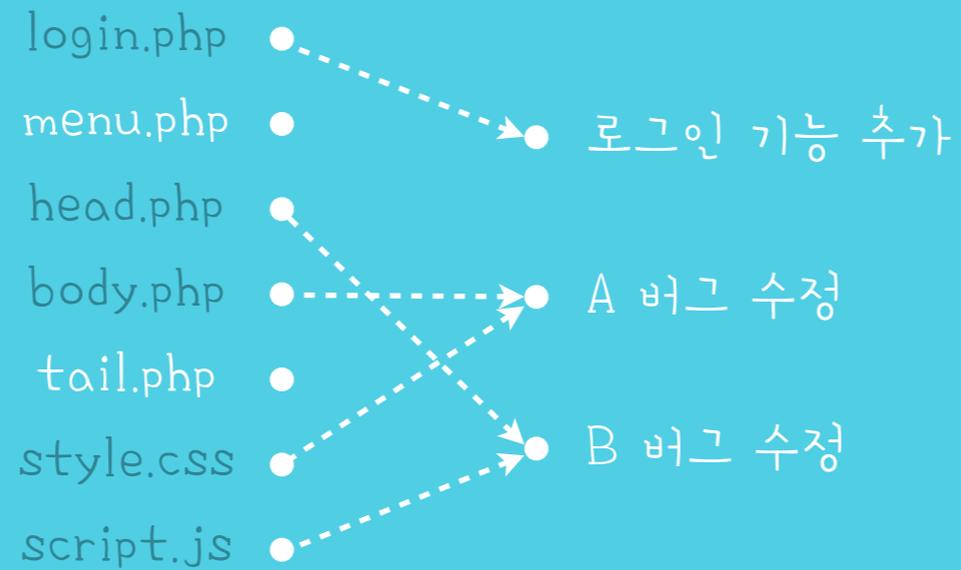
로그인 기능 추가

A 버그 수정

B 버그 수정

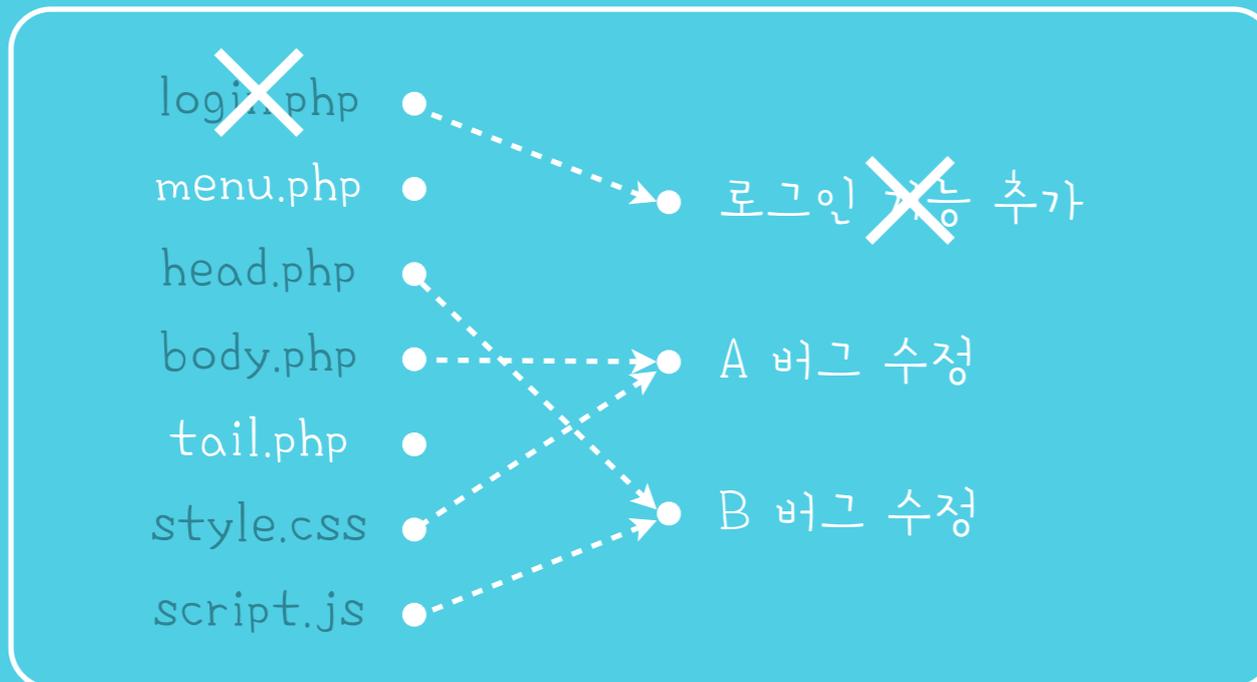
스태이지 영역이 어떤 역할을 하는지 알아보을까요?

② 다음과 같이 파일들이 수정되었습니다.



스태이지 영역이 어떤 역할을 하는지 알아보을까요?

③ 그런데 로그인 기능 추가가 다음 빌드로 미루어졌습니다.
svn 이라면 보통 어떻게 할까요?



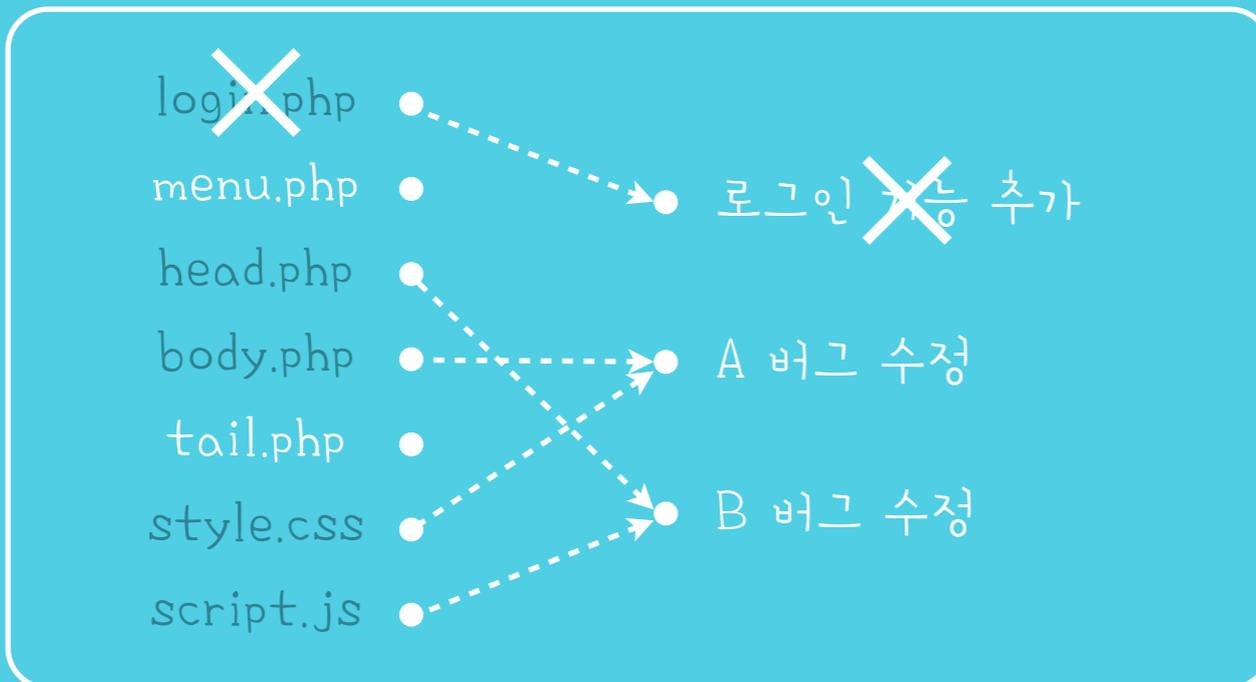
- ① 수정 된 login.php를 어딘가로 백업
- ② svn revert를 이용해 원래 상태로 복원
- ③ 수정 내역 전체를 커밋
- ④ 백업해두었던 login.php를 다시 복구



변경 된 파일들은 무조건 커밋 대상이 되기 때문에 생기는 문제입니다.

스태이지 영역이 어떤 역할을 하는지 알아보을까요?

④ 그럼 git은 어떻게 하나요?



① 커밋 할 파일들만 staging area에 추가

② 로컬저장소로 커밋

귀찮을 수 있습니다.

특별히 파일들을 구분 할 필요가 없을 때에는
-a 옵션으로 스테이지 추가와 커밋을 동시에 할 수 있습니다.

git commit -a

커밋을 마쳤으면, 이제 다른 사람들에게 작업물을 공유합니다.

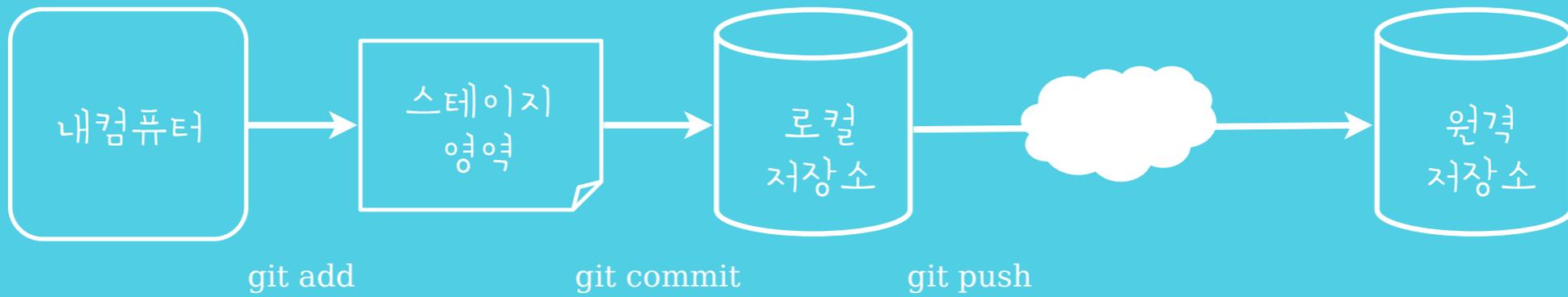
원격저장소에 올리는 명령어는 **push** 입니다.

git push

svn



git



내 작업을 올리는데 오류가 발생했어요.
다른 사람들이 작업 한 내용을 먼저 받아야 한대요.

원격저장소로부터 내려받기: **fetch**
내려받은 데이터를 병합: **merge**

하지만 이 둘을 함께 처리하는 명령은 **pull**

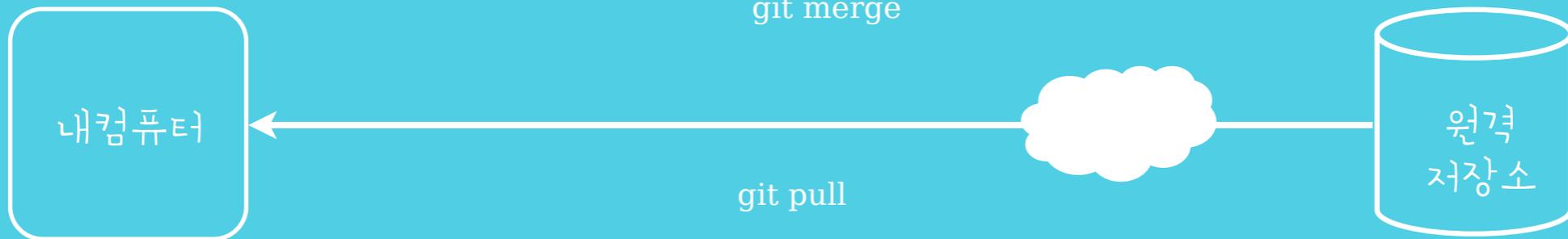
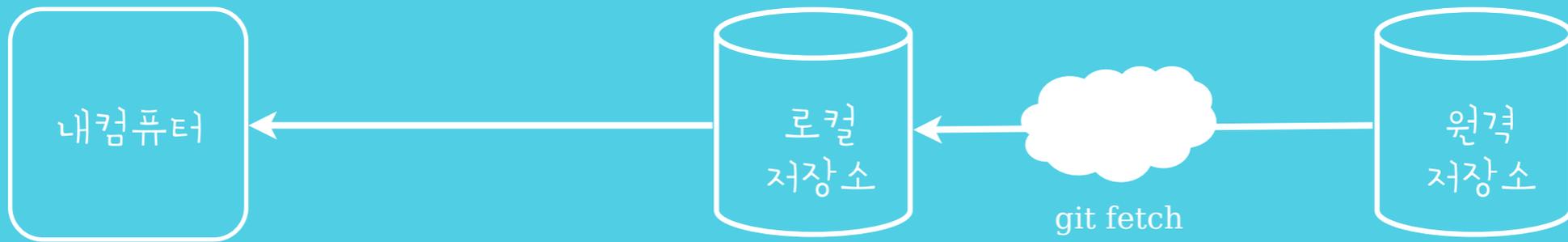


git pull

Svn



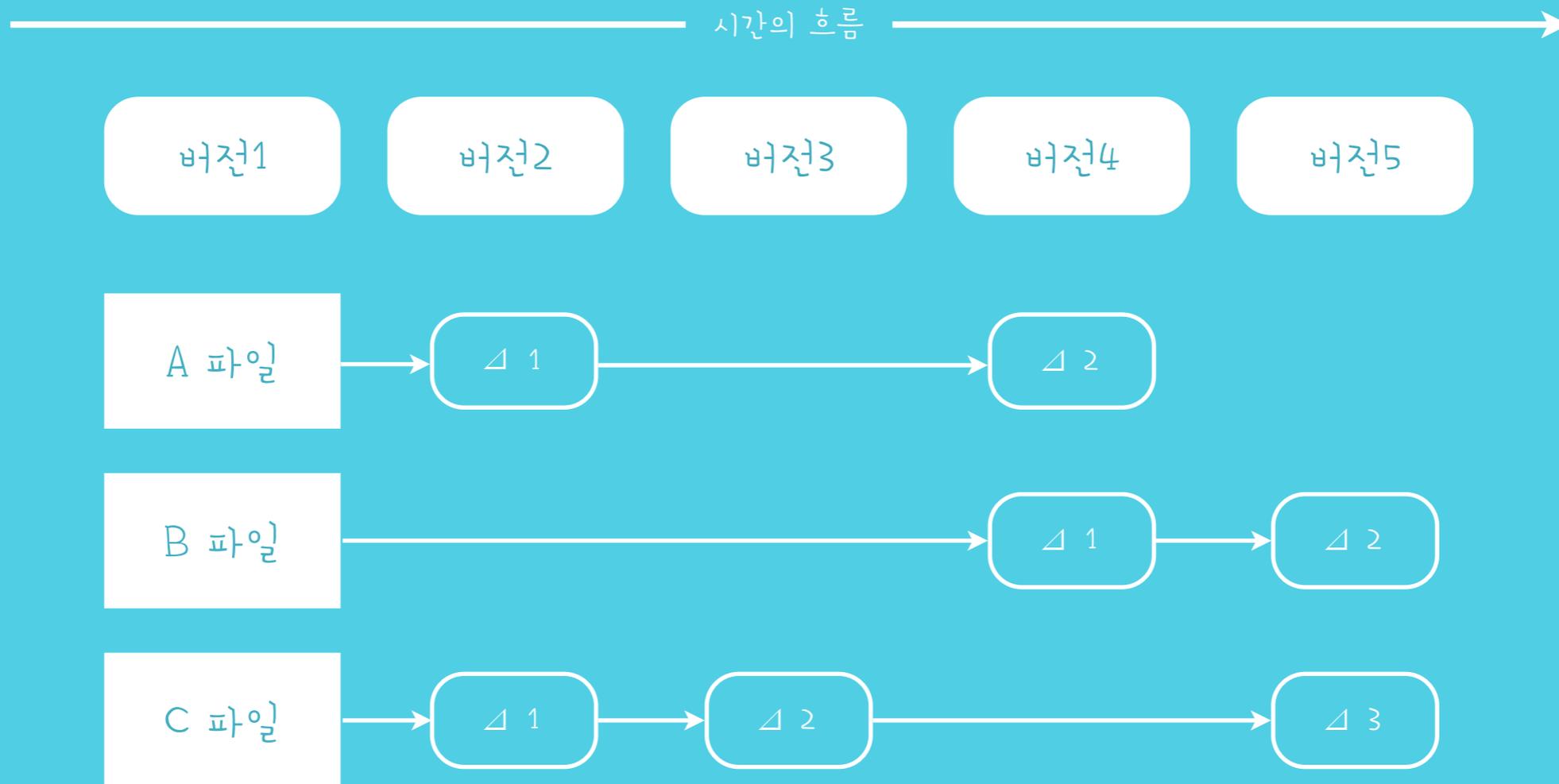
git



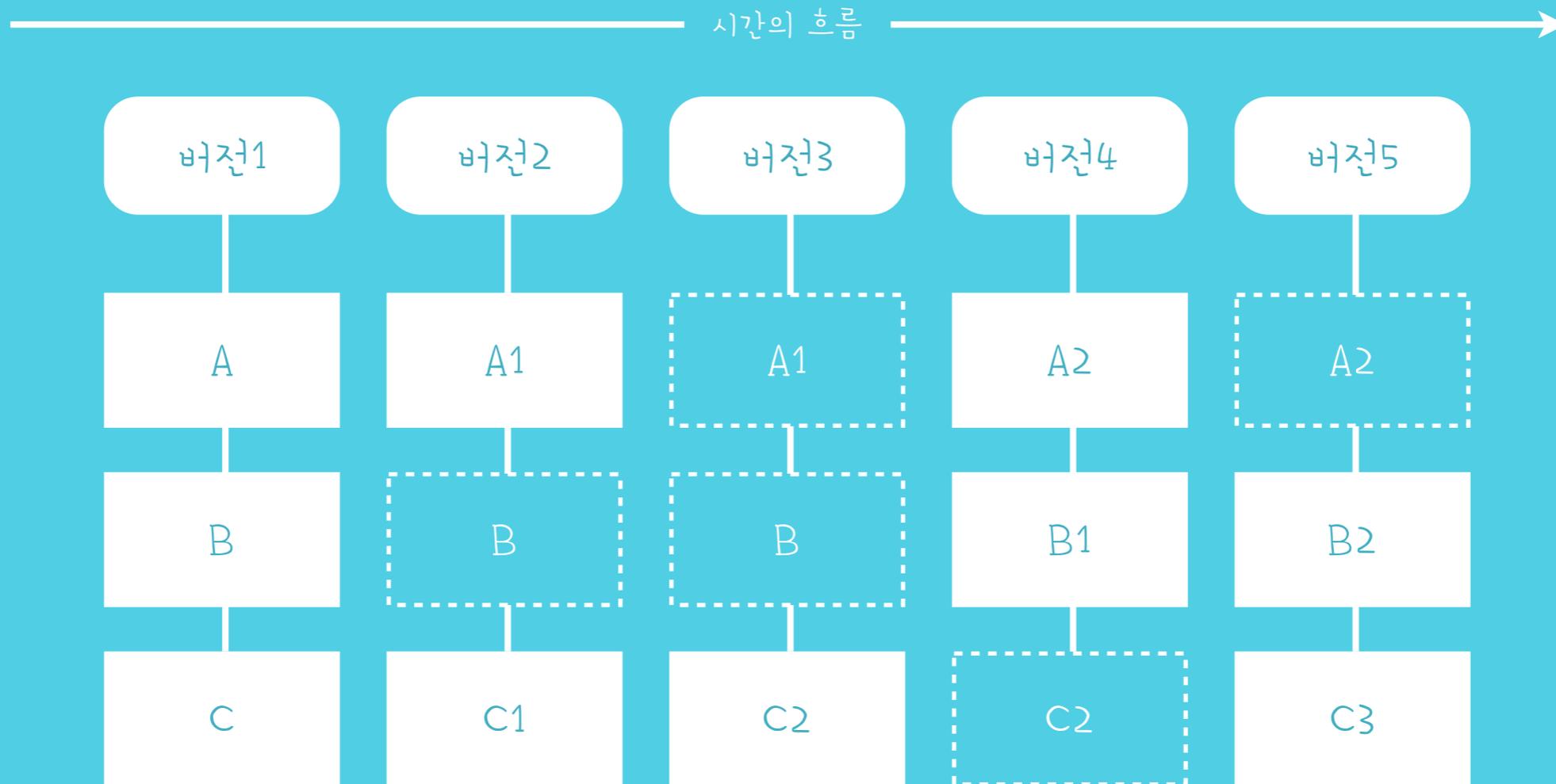
스냅샷 snapshot

git은 각각의 버전을 스냅샷으로 저장합니다.

svn은 파일의 변화(차이점)를 저장합니다.

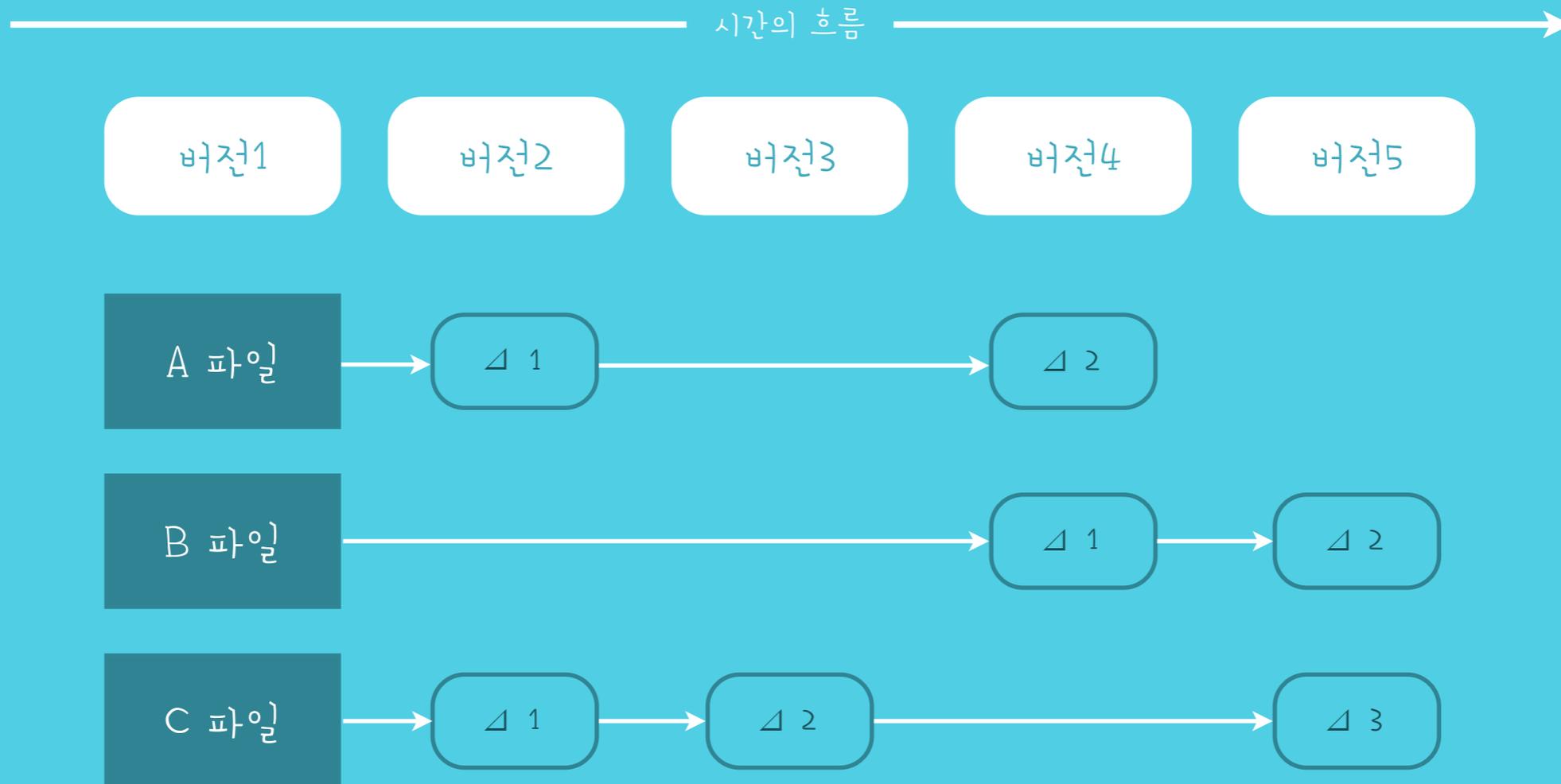


git은 그 순간의 스냅샷으로 저장합니다.



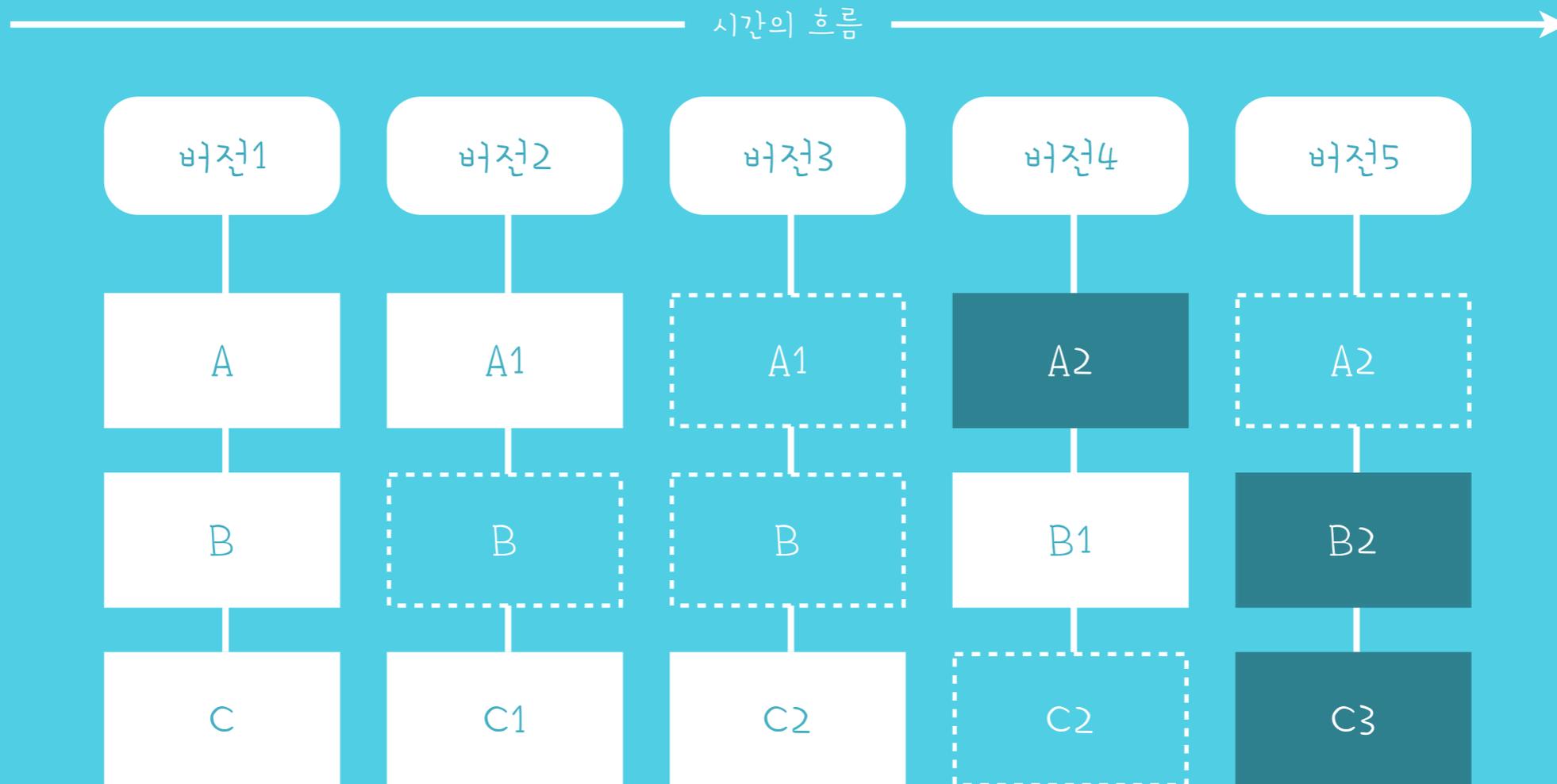
svn에서 버전5의 파일들을 가져오겠습니다.

기초가 되는 파일과 함께 모든 변경 내역을 서버로부터 내려받습니다.



git에서 버전5의 파일들을 가져오겠습니다.

가장 가까운 스냅샷들만으로 특정 버전을 빠르게 만들어낼 수 있습니다.
게다가 네트워크를 거치지 않습니다.

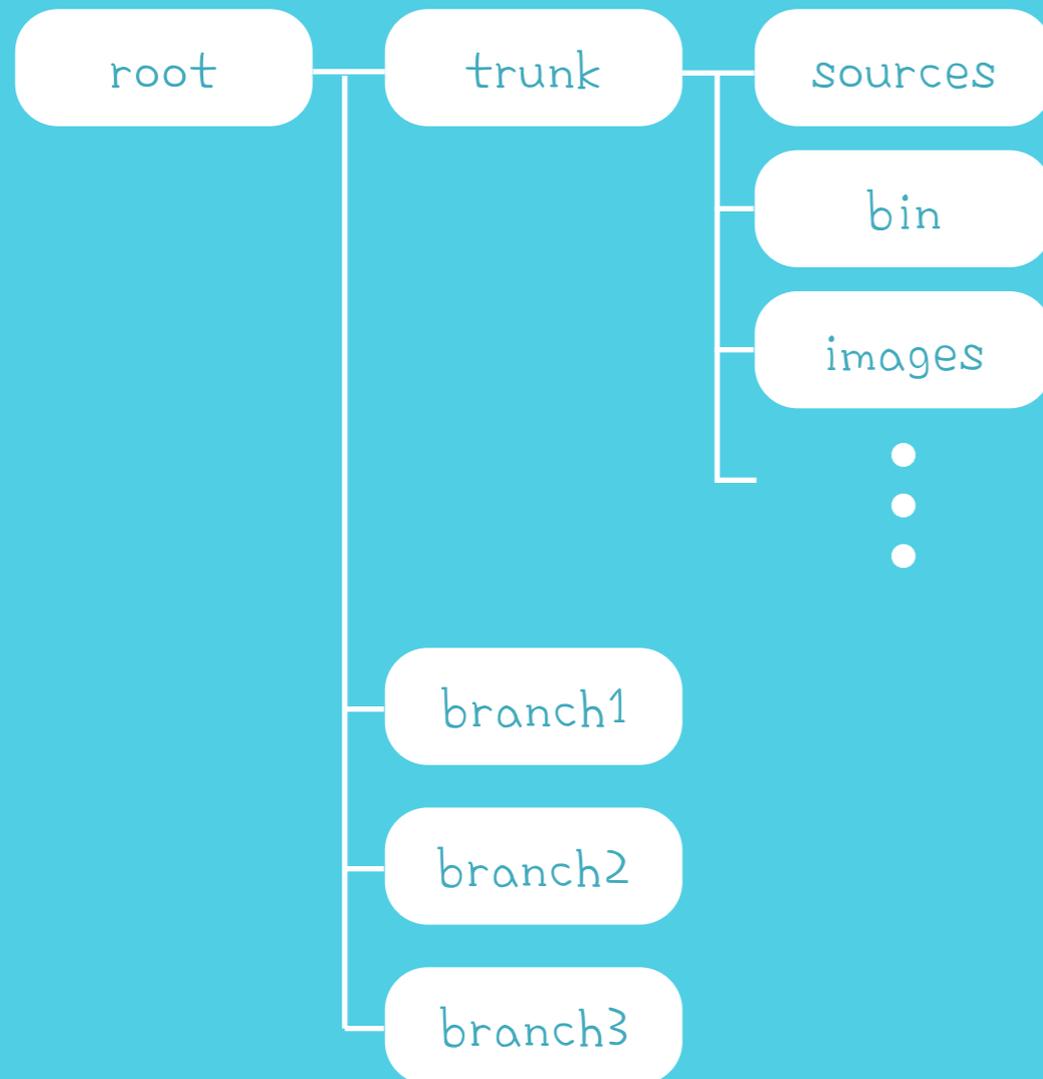


지금까지 스냅샷에 대해서 알아보았습니다.
그런데 왜 이렇게 열심히 설명한 걸까요?

바로 git의 최대 장점인 **브랜치** branch를 소개하기 위해서입니다.

→ 이게 바로 스냅샷 덕분에 구현 될 수 있었답니다.

svn의 브랜치



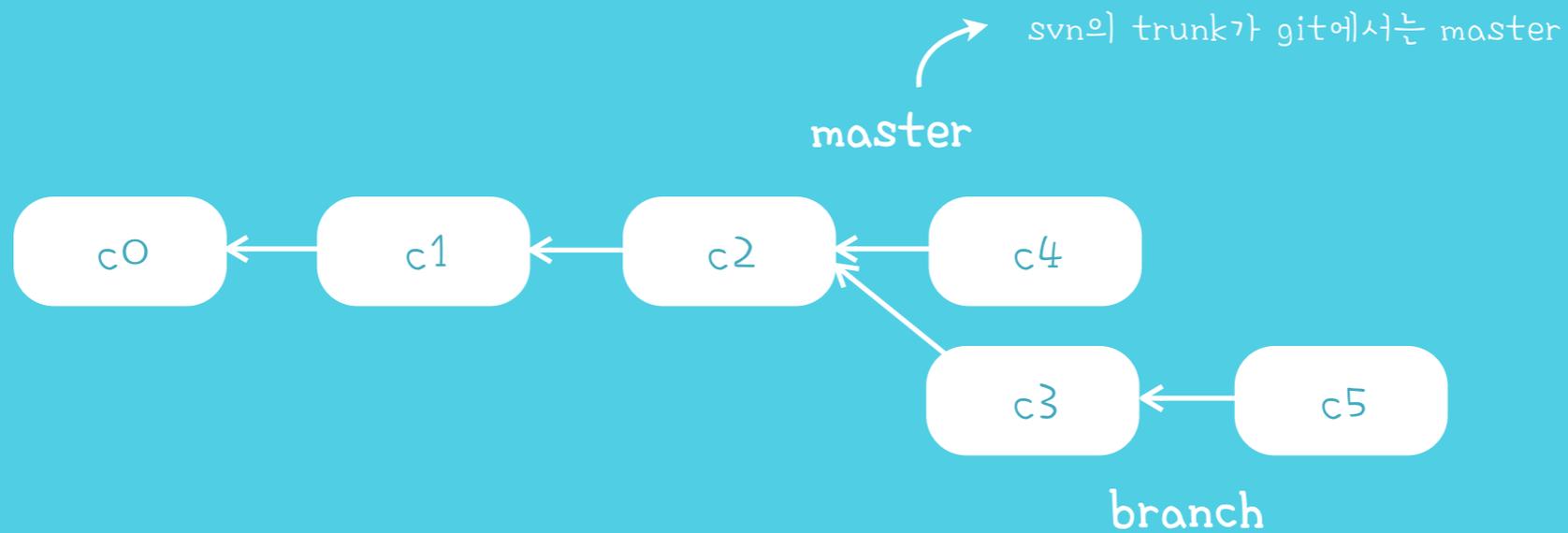
svn은 간단히 말해 디렉토리 구조입니다.

서버는 단지 변경사항만 저장하며
최소한의 자료구조를 유지하지만,

작업자가 작업을 위해 브랜치를 내려받으면
그 변경 내역들을 순차 적용하여
실제 파일들을 만들어냅니다.

브랜치를 만들면,
전체 파일을 네트워크를 통해
통째로 내려받기 때문에
느리고 부담스럽습니다.

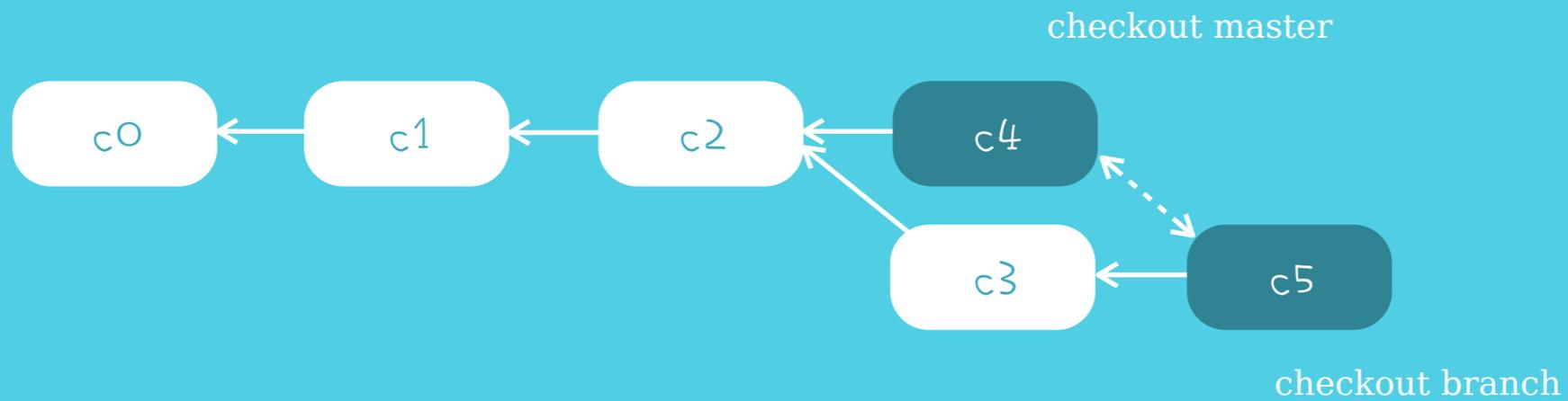
git의 브랜치



git은 서버에 그런 논리적 디렉토리 구조를 만들지 않습니다.
다만, 연속된 스냅샷이 순차적으로 이어지다가 가지를 치면서 브랜치가 만들어집니다.

git의 브랜치

작업자는 작업중에 branch와 master를 자유롭게 이동 할 수 있습니다.

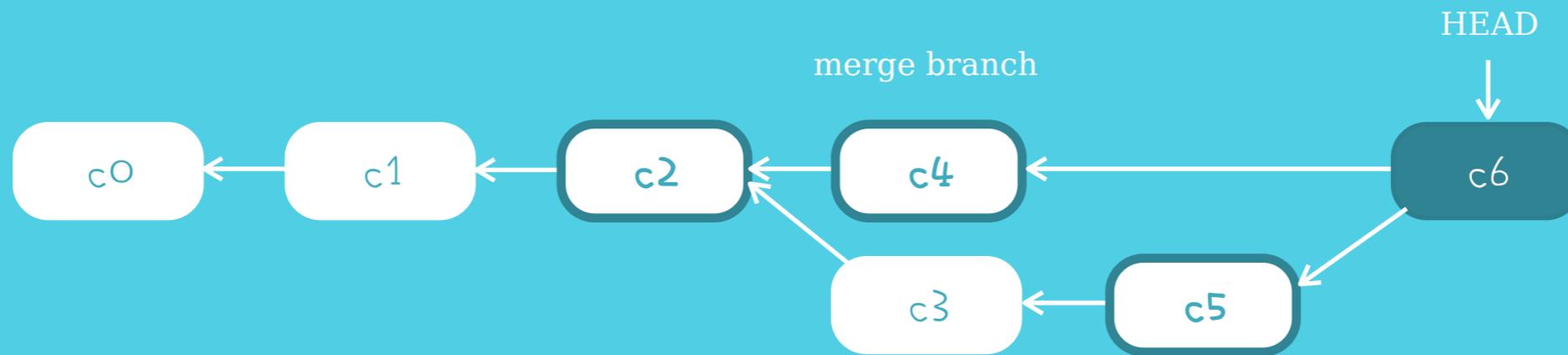


checkout이 svn에서는 내려받는 명령이지만,
git에서는 브랜치를 선택하는 기능입니다.

svn과 git의 브랜치가 어떻게 다른지 정리해볼게요.

	svn	git
만들기	즉시 생성	즉시 생성
내려받기	전체 디렉토리 구조를 내려받음	필요 없음
브랜치 전환	직접 디렉토리를 이동	작업파일들이 실시간 변경됨
병합	위치와 리비전을 명시, 복잡함	빠르고 간편함

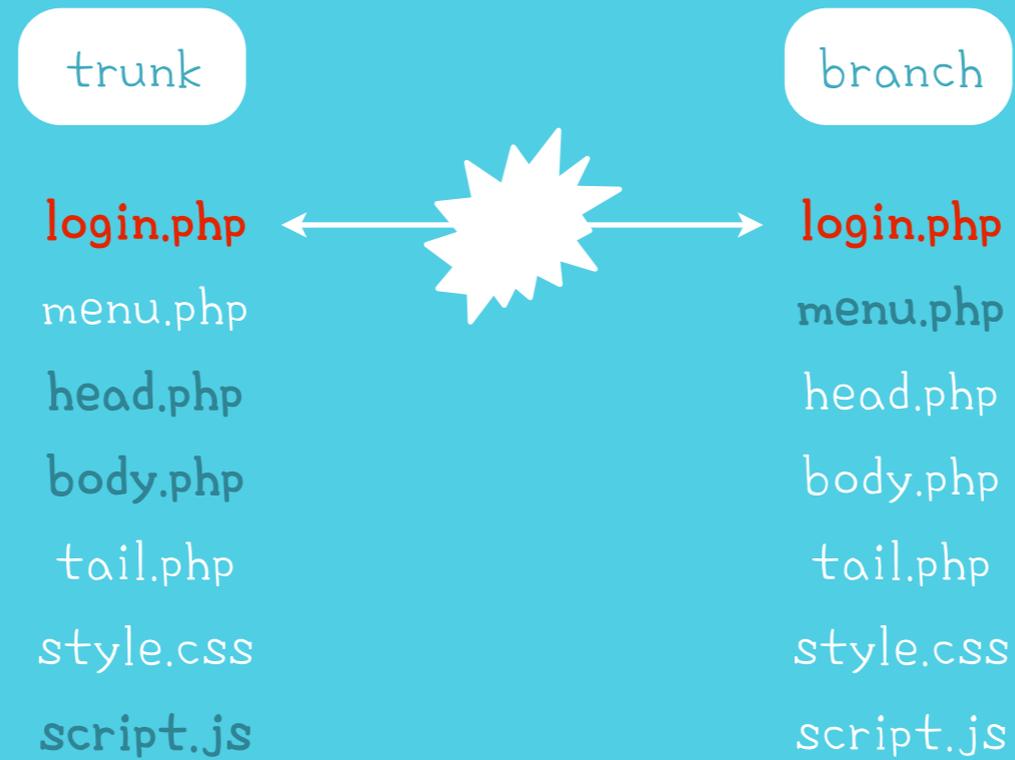
git의 병합



3-way merge : 각 브랜치와 공통 조상 하나를 이용하여 병합

svn의 merge가 위치와 병합 할 리비전을 확인하여 입력하는 등의 절차가 필요했다면, git의 merge는 **브랜치 이름**만으로 모든 것들을 자동으로 해결합니다.

merge중에 충돌은 언제나 일어날 수 있습니다.



충돌에 대한 처리는 svn과 동일합니다.

```
<<<<<<< HEAD:login.php
<div id='footer'>contact : email.support@github.com</div>
=====
<div id='footer'>
  please contact us at support@github.com
</div>
>>>>>>> branch:login.php
```

→ master의 login.php 영역

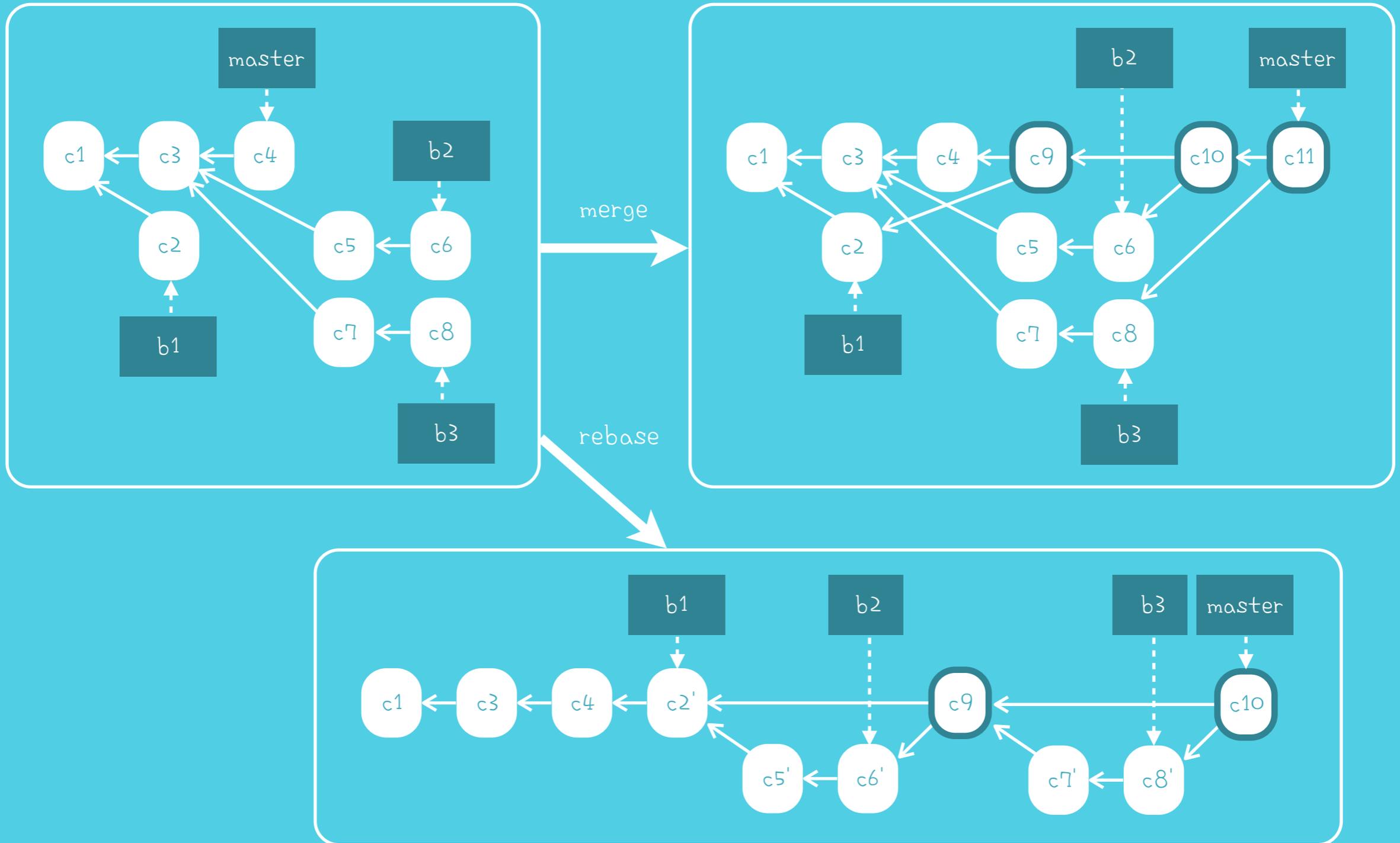
→ branch의 login.php 영역

- ① master와 branch 영역 중 하나를 선택
- ② 혹은 양쪽을 참고하여 새로 작성
- ③ <<<, >>>, === 구분자를 삭제
- ④ 변경 내역을 add 후, commit

git mergetool 명령을 사용하면 충돌을 쉽게 해결하기 위한 merge 도구를 사용가능합니다.

리베이스 rebase

merge와 rebase를 비교합니다.



하지만,
조금이라도 `svn`이 의심 된다면, 조금이라도 `git`가 기대 된다면...

여러분의 프로젝트에  **git**을 시도해보시기를 권해드립니다.

