Socialinus

svn 능력자를 위한 git 개념 가이드

Socialinus

svn을 능숙하게 다루던 능력자들
처음 git을 만나면 대게 이런 표정이죠.

Socialinus

하지만 곧 이렇게 됩니다.

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

Socialinus

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

git != svn

Socialinus

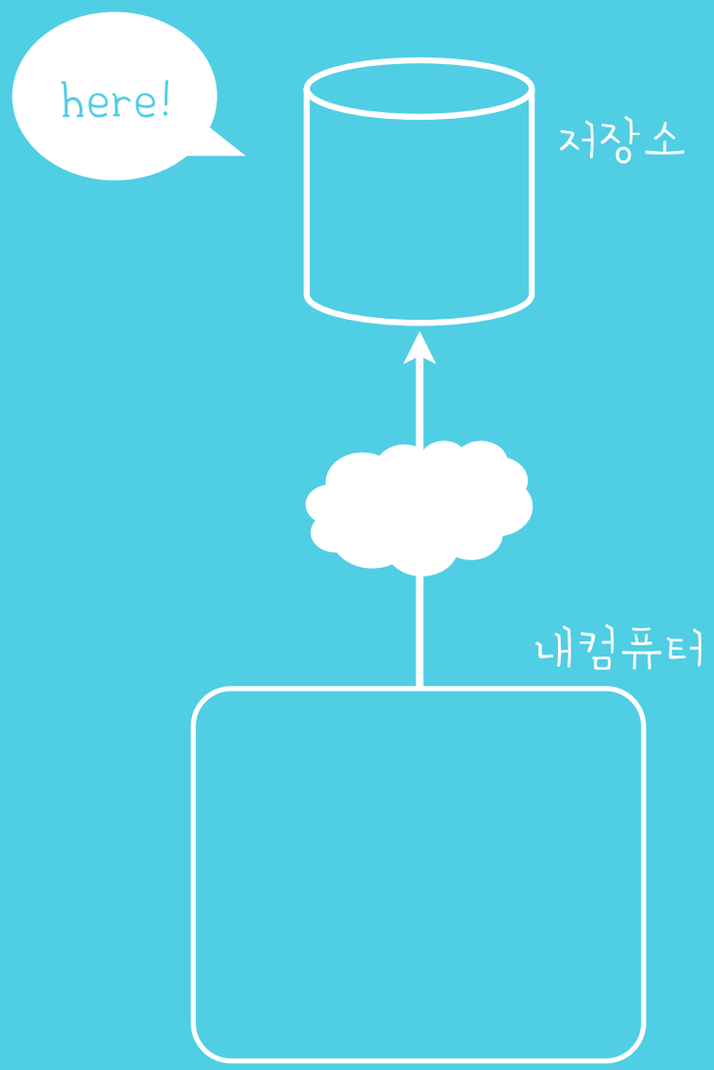
주위에서 흔히 볼 수 있는 git 가이드들은 무척 친절합니다.
하지만, svn 숙련자들에게는 오히려 혼란스럽습니다.

Socialinus

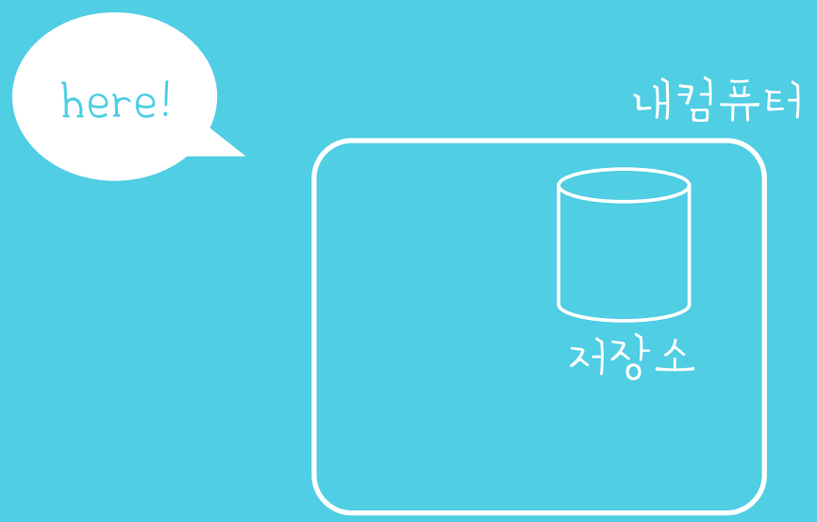
지금부터 svn을 기준으로 git을 살펴보겠습니다.

Socialinus

svn은 보통 저장소가
서버에 있습니다.



git은 저장소가
내컴퓨터에 있습니다. 응?



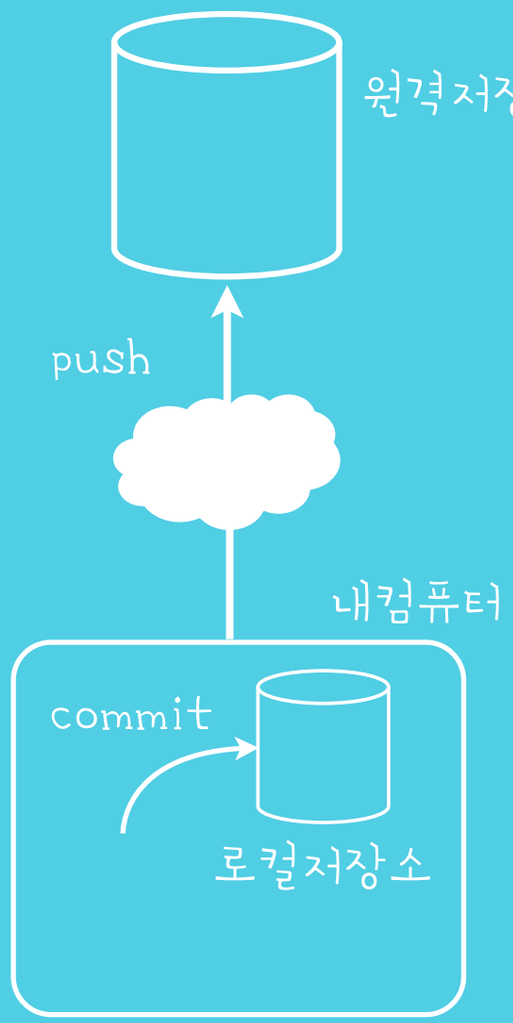
Socialinus

그럼 git은 다른 사람들과 작업을 할 수 없나요?

원격저장소를 만들면 됩니다.

remote repository

Socialinus



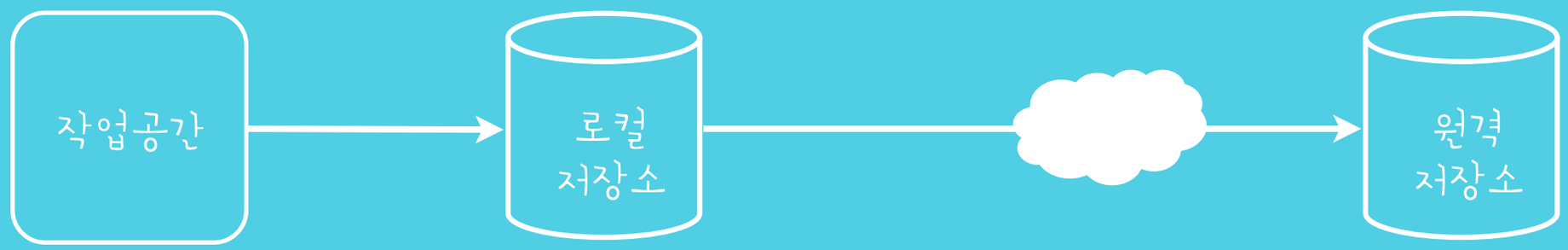
② 다른 사람과 공유할 때 원격 저장소에 **푸쉬**합니다.

① 내 컴퓨터의 저장소에 열심히 작업 내용을 **커밋**하고

Socialinus

svn

git



이렇게 저장소가 분산되는 구조를 분산버전관리시스템(DVCS)라고 합니다.

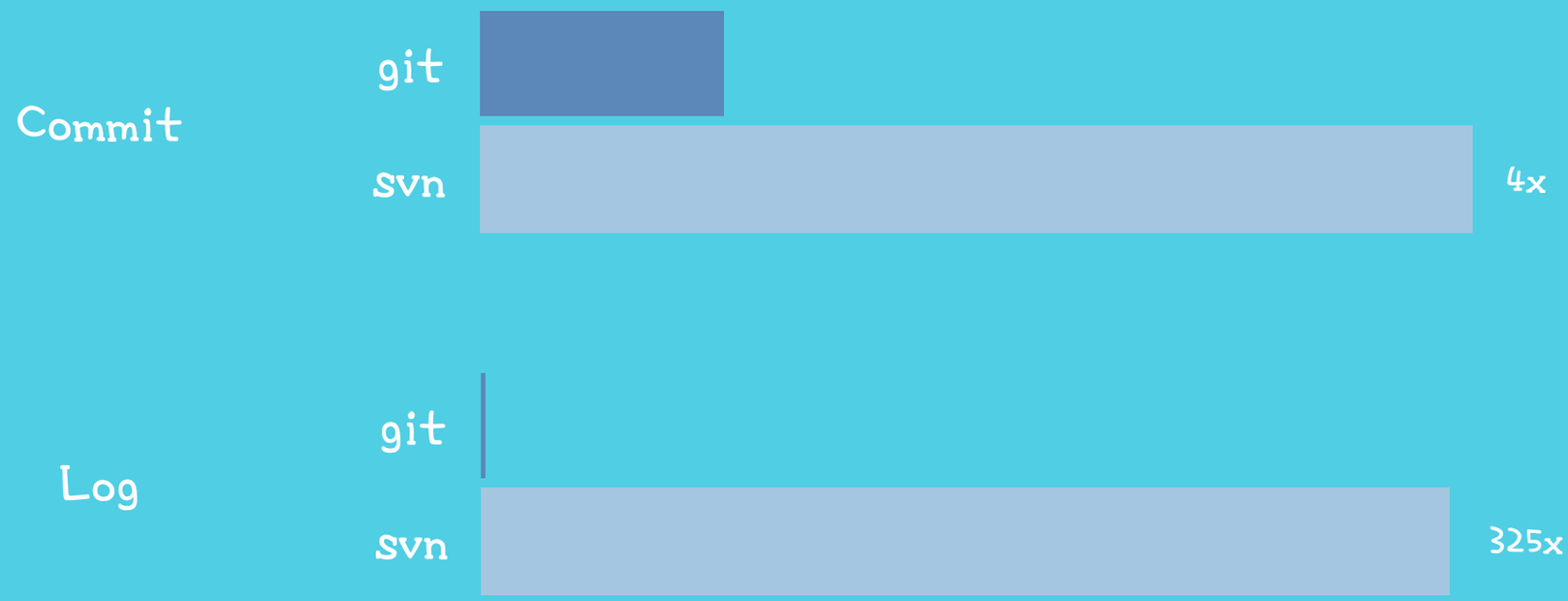
Socialinus

로컬저장소가 따로 있으면 어떤 **장점**이 있을까요?

Socialinus

엄청나게 빠릅니다.

인터넷을 경유할 필요가 없기 때문에 훨씬 빠릅니다.

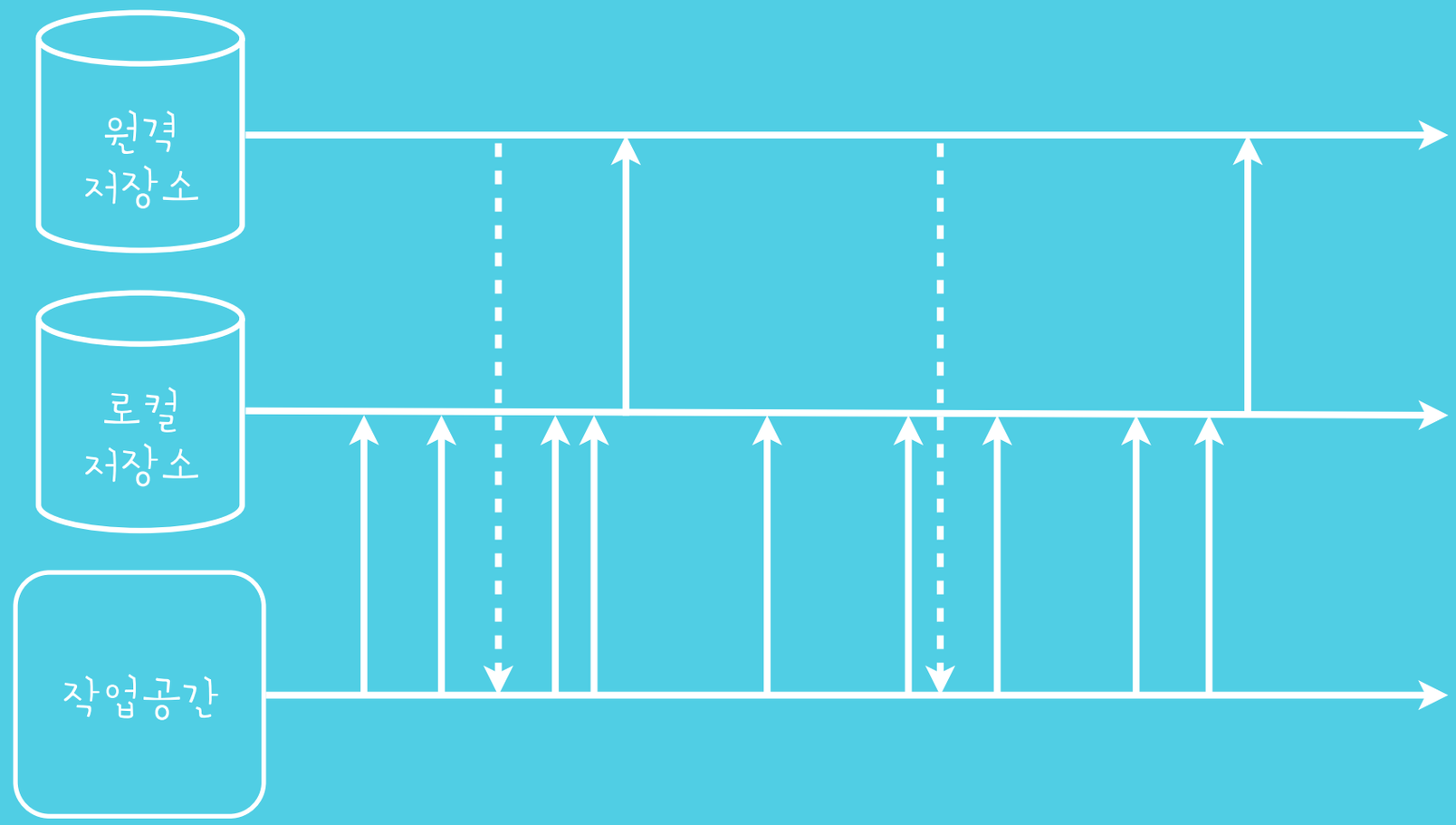


git 공식홈페이지의 벤치마크 결과를 참고하였습니다.

Socialinus

커밋에 부담이 없습니다.

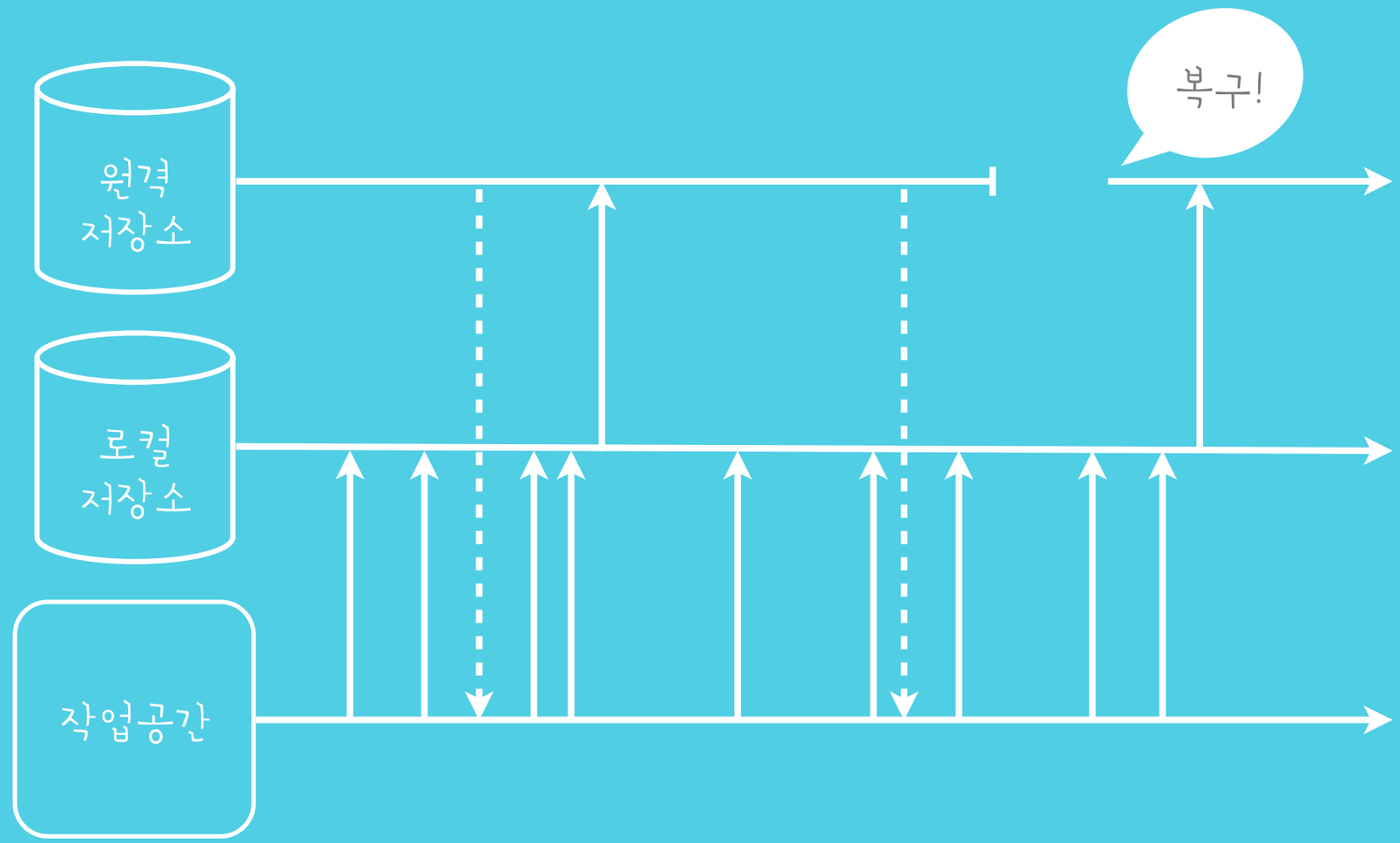
내 로컬 저장소에서 마음대로 실험하고 테스트하세요.



Socialinus

원격저장소와 연결이 끊겨도 계속 버전관리가 가능합니다

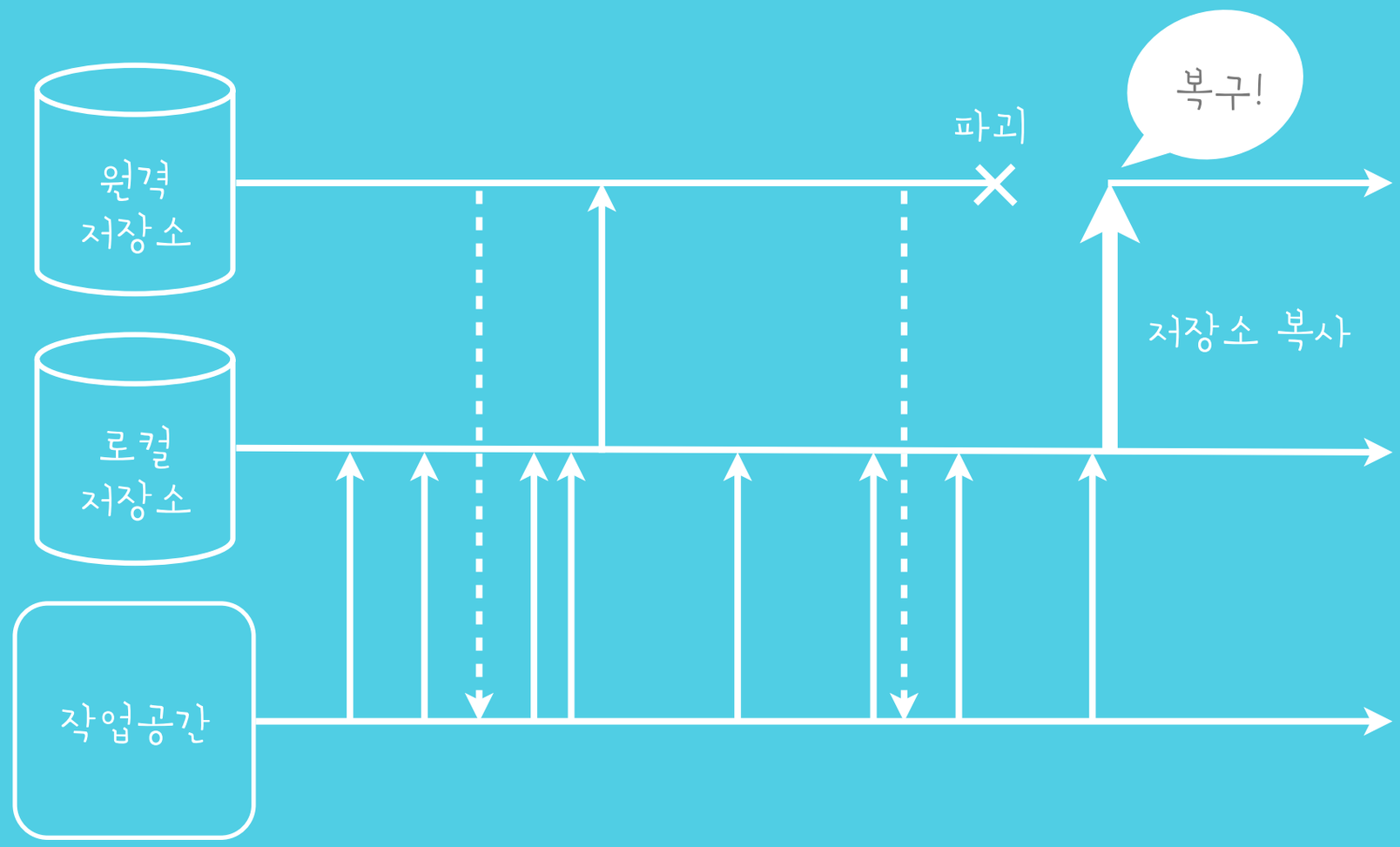
svn은 서버에 문제가 생기면 모든 버전관리가 중단됩니다.



Socialinus

원격저장소가 폭파되어도 로컬저장소로 복원이 가능합니다.

원격저장소와 연결된 모든 로컬저장소는 사본을 보유합니다.

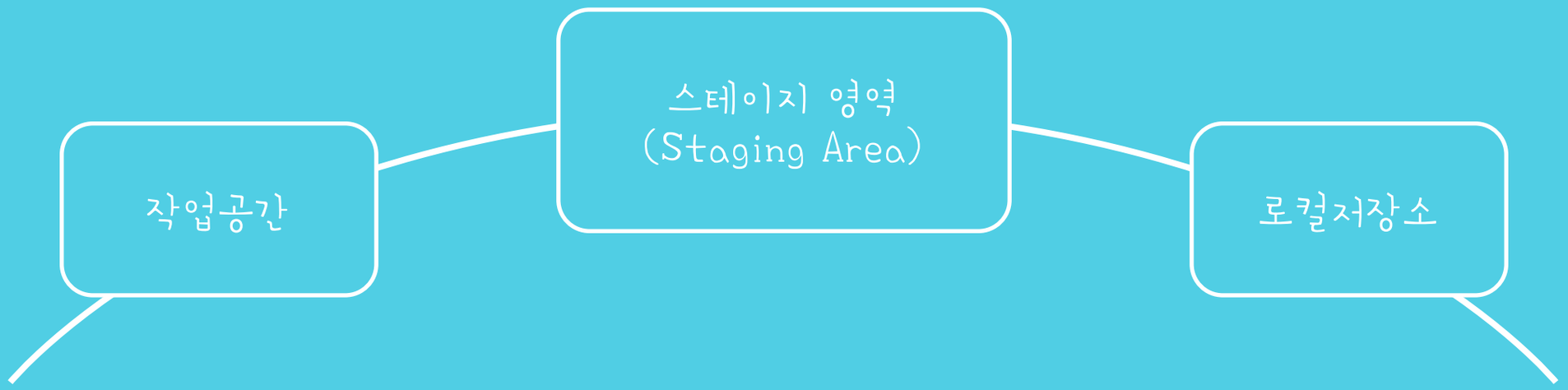


Socialinus

그런데 git에는 로컬저장소에 커밋 전, 하나의 단계가 더 있습니다.

바로 **스테이지 영역**입니다.

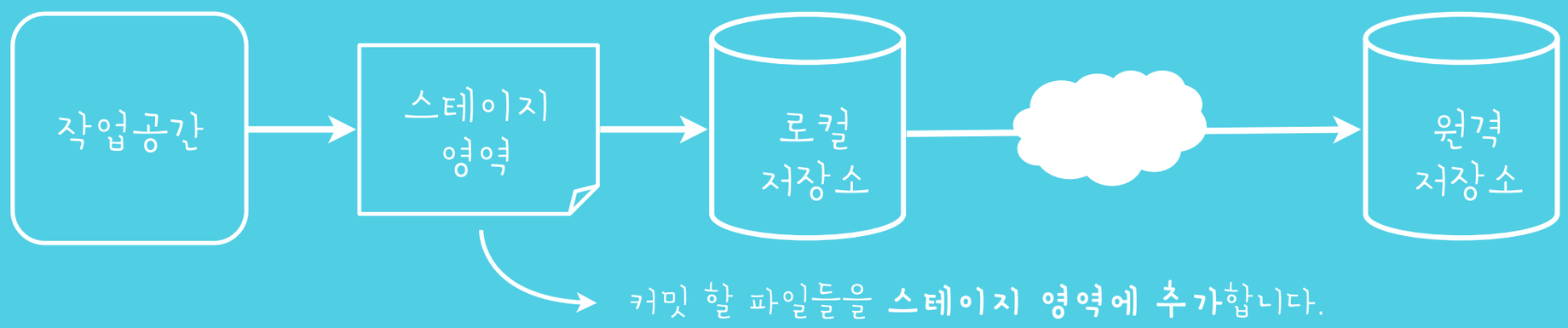
인덱스(index)라고도 부릅니다.



Socialinus

svn

git



Socialinus

스테이지 영역이 어떤 역할을 하는지 알아볼까요?

① 이런 빌드 목표를 가지고 작업을 하고 있습니다.

빌드 목표!

로그인 기능 추가

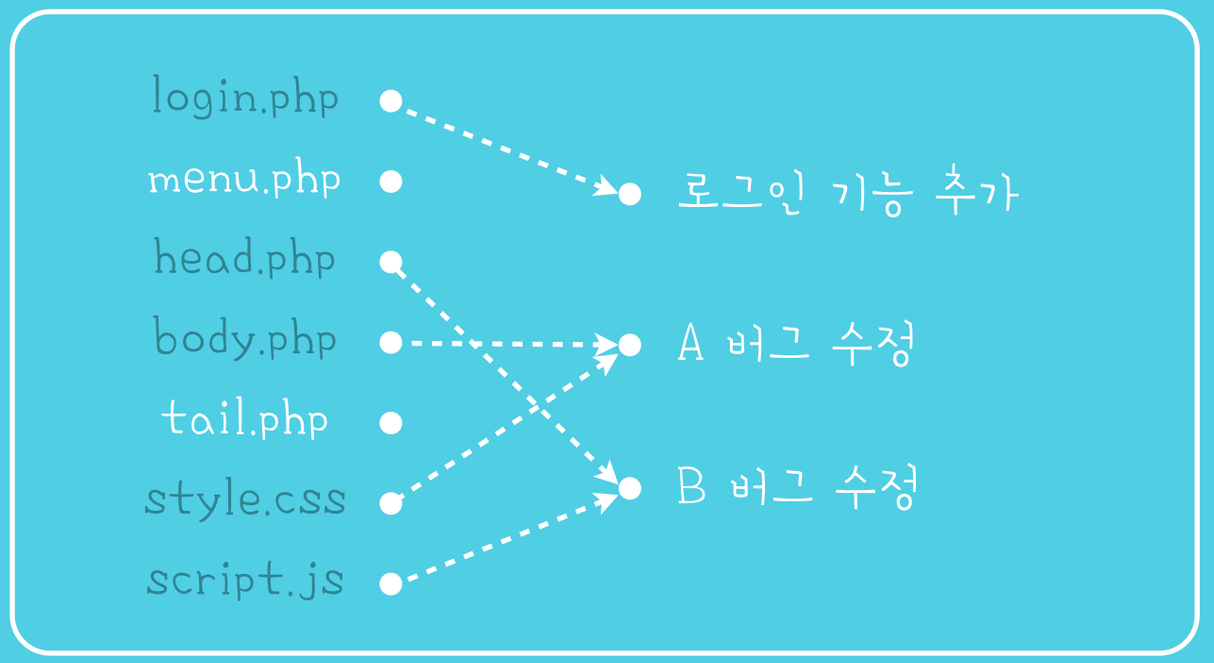
A 버그 수정

B 버그 수정

Socialinus

스테이지 영역이 어떤 역할을 하는지 알아볼까요?

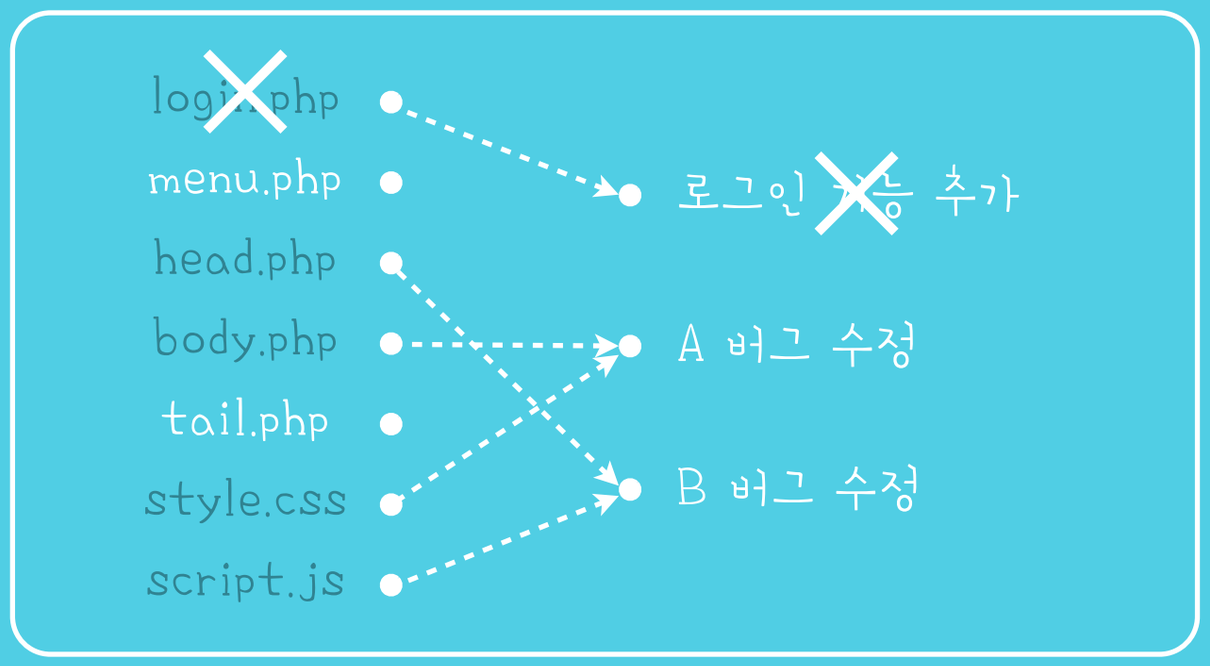
② 다음과 같이 파일들이 수정되었습니다.



Socialinus

스테이지 영역이 어떤 역할을 하는지 알아볼까요?

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



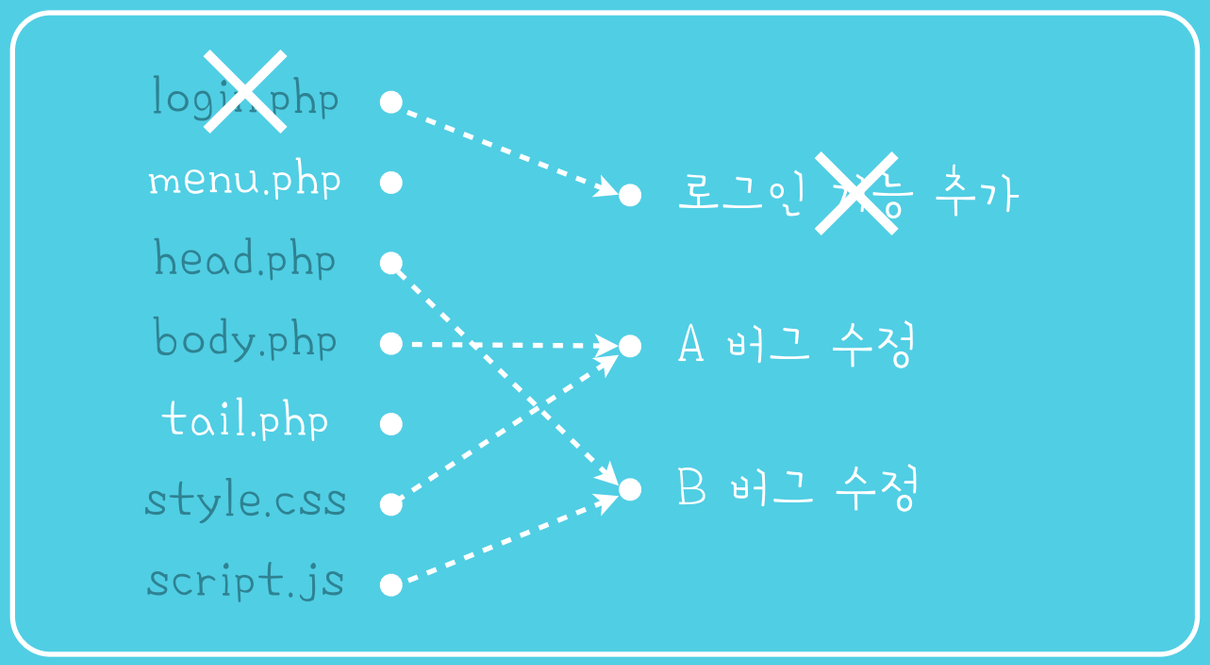
❶ 수정 된 login.php를 어딘가로 백업
❷ svn revert를 이용해 원래 상태로 복원
❸ 수정 내역 전체를 커밋
❹ 백업해두었던 login.php를 다시 복구

변경 된 파일들은 무조건 커밋 대상이
되기 때문에 생기는 문제입니다.

Socialinus

스테이지 영역이 어떤 역할을 하는지 알아볼까요?

④ 그럼 git은 어떻게 하나요?



❶ 커밋 할 파일들만 staging area에 추가

❷ 로컬저장소로 커밋

Socialinus

귀찮을 수 있습니다.

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

git commit -a

Socialinus

커밋을 마쳤으면, 이제 다른 사람들에게 작업물을 공유합니다.

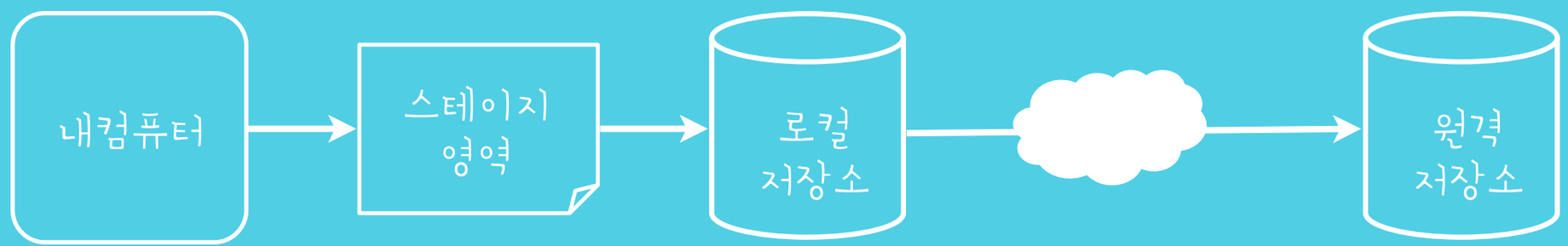
원격저장소에 올리는 명령어는 **push** 입니다.

git push

Socialinus

svn

git



Socialinus

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

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

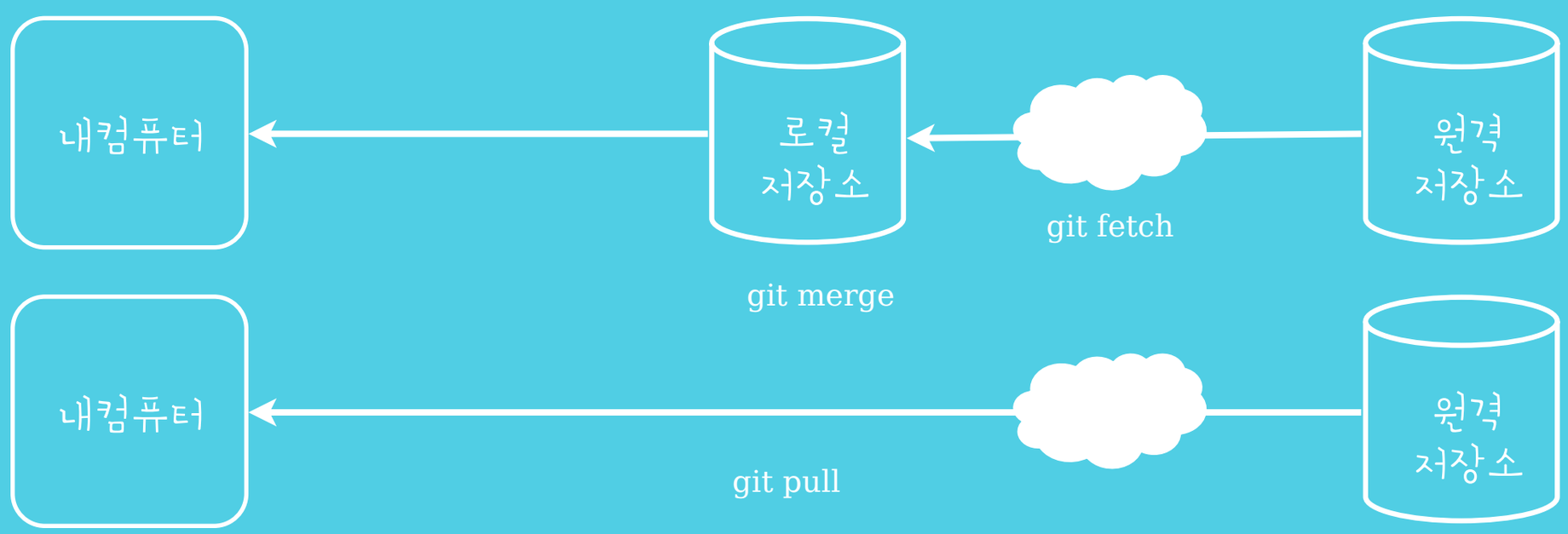
하지만 이 둘을 함께 처리하는 명령은 **pull**

git pull

Socialinus

svn

git



Socialinus

지금까지 알아본 svn과의 차이점은?

① **로컬저장소**가 존재한다.

② 커밋 이전에 **스테이지 영역**에 추가하는 과정이 있다.

③

가장 중요한 차이점이자 git만의 특징이 남았습니다!

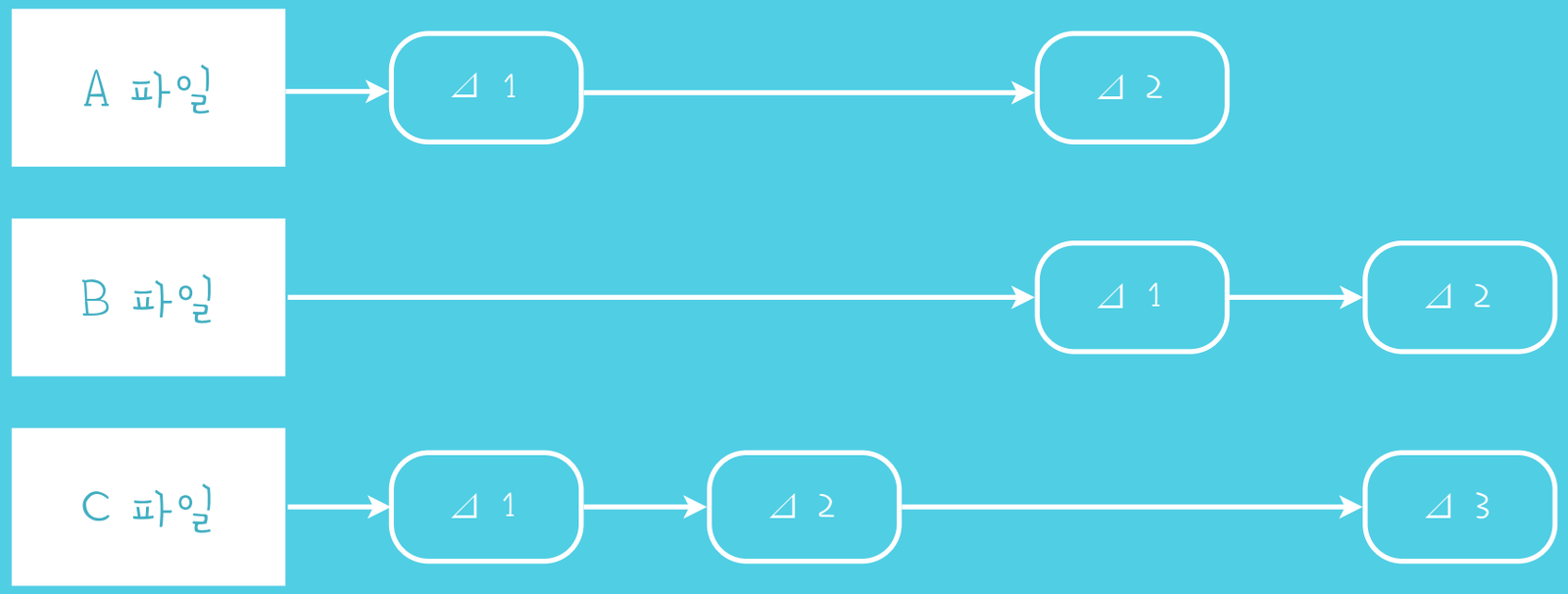
Socialinus

스냅샷 snapshot

git은 각각의 버전을 스냅샷으로 저장합니다.

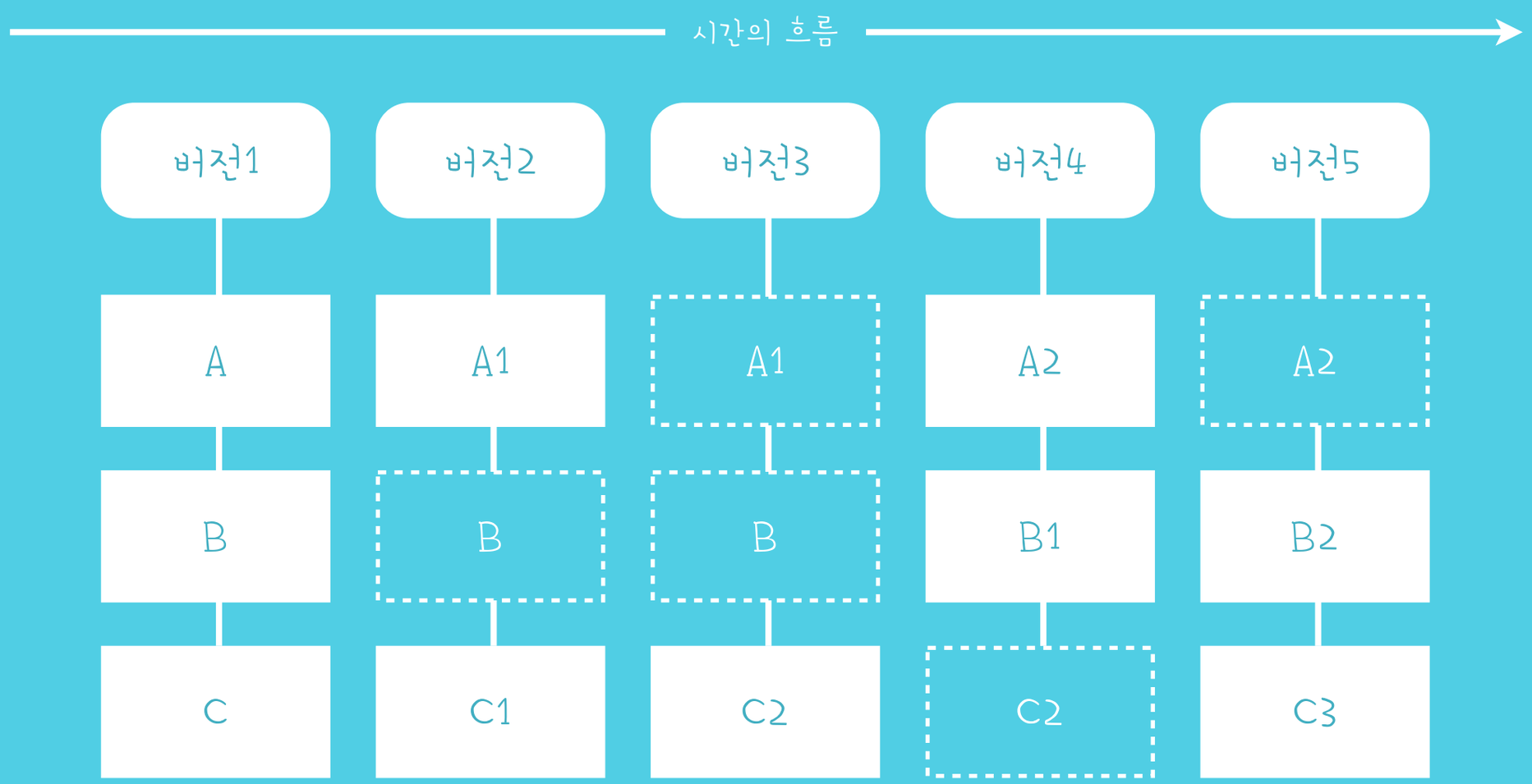
Socialinus

svn은 파일의 변화(차이점)를 저장합니다.



Socialinus

git은 그 순간의 스냅샷으로 저장합니다.



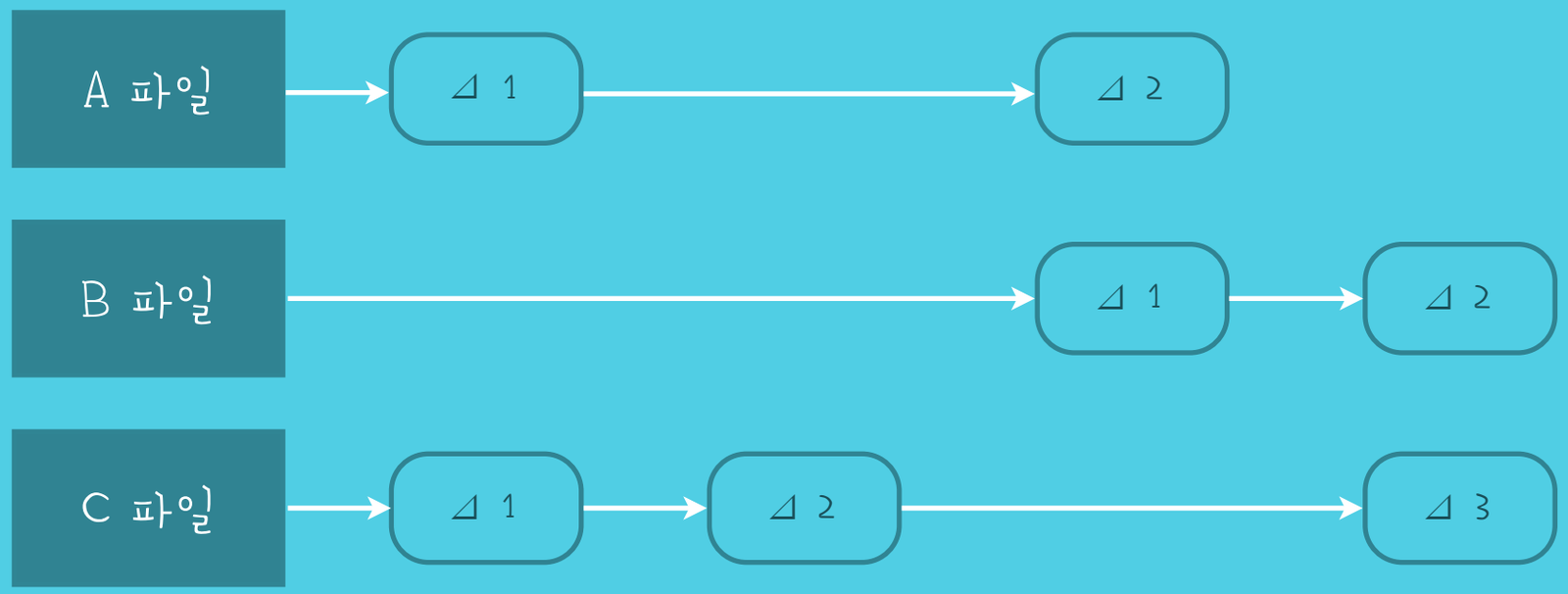
Socialinus

svn에서 **버전5**의 파일들을 가져오겠습니다.

기초가 되는 파일과 함께 모든 변경 내역을 서버로부터 내려받습니다.

시간의 흐름 →

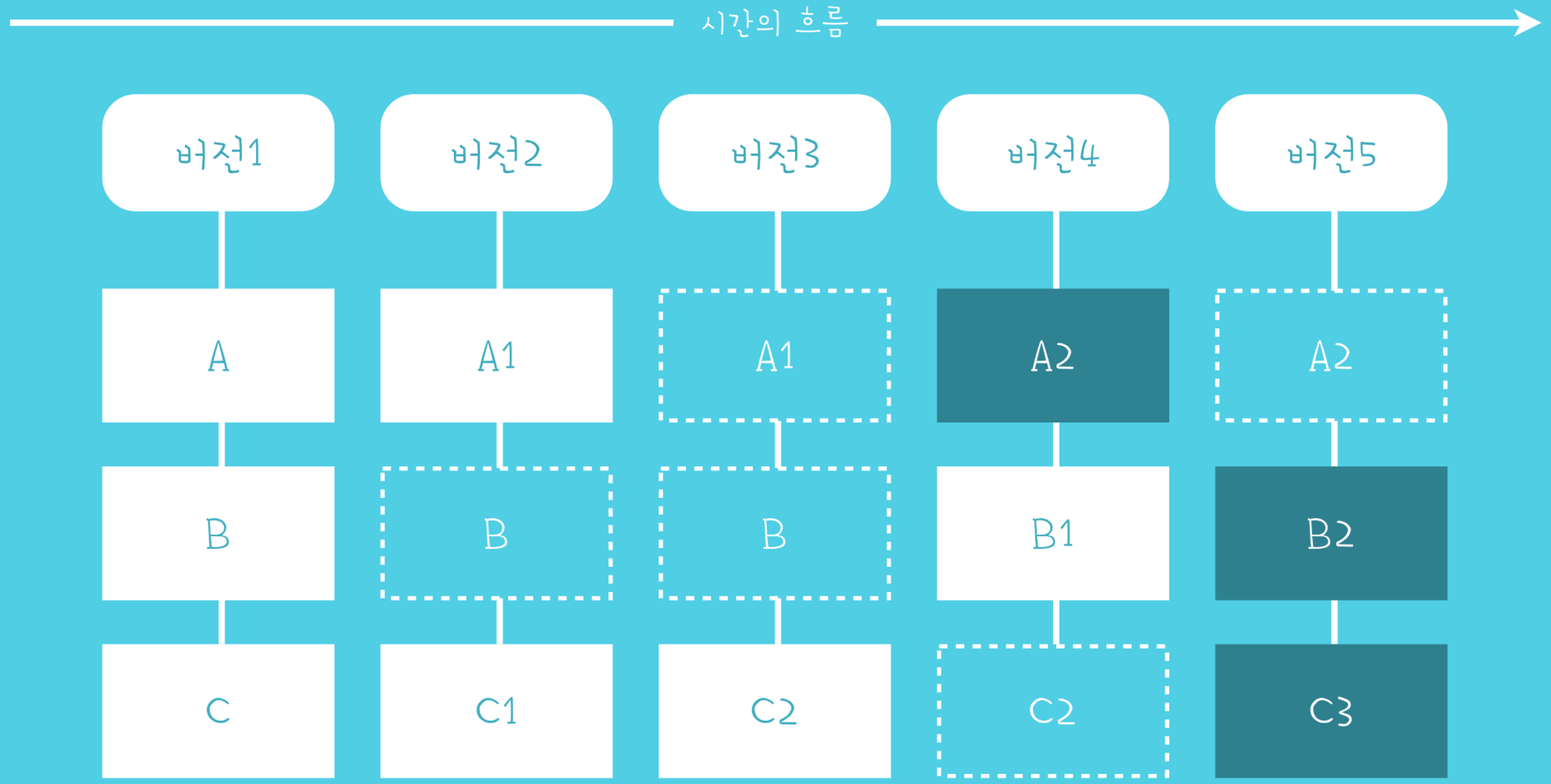
버전1 | 버전2 | 버전3 | 버전4 | 버전5



Socialinus

git에서 **버전5**의 파일들을 가져오겠습니다.

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



Socialinus

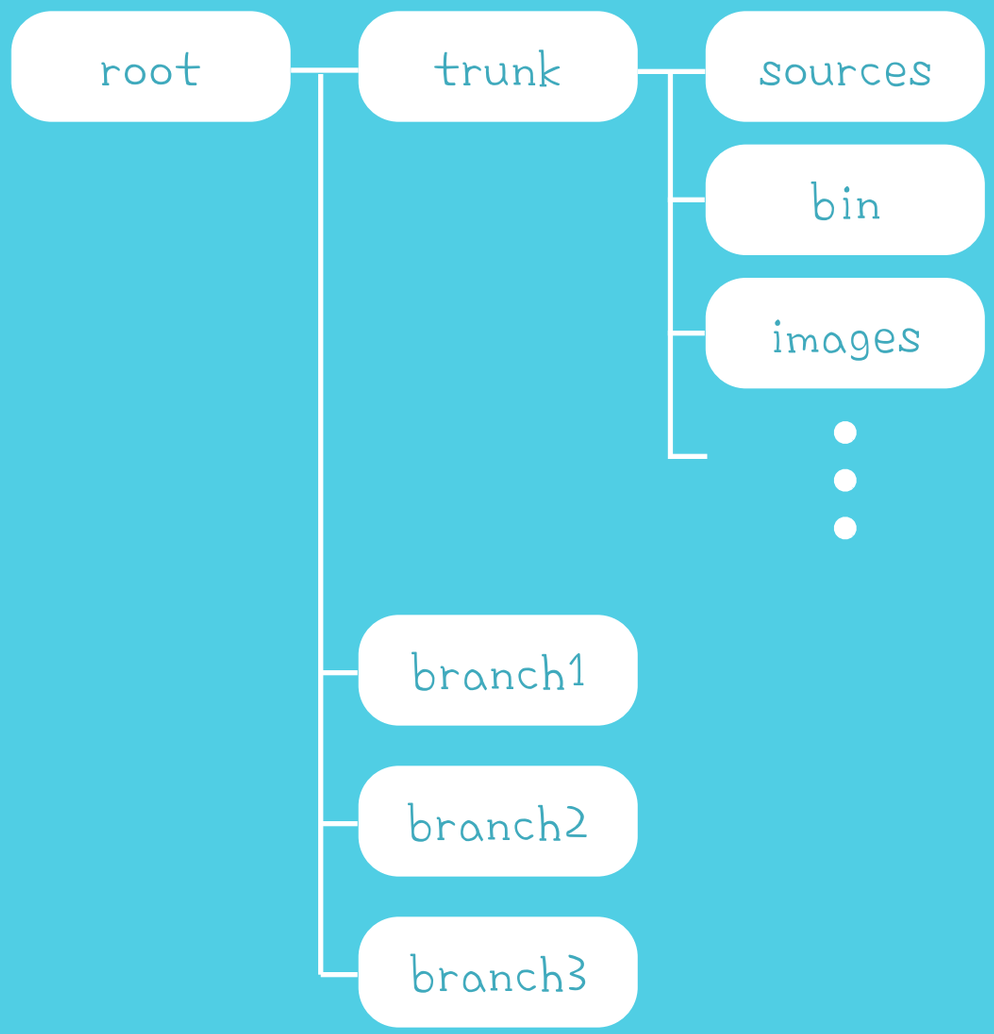
지금까지 스냅샷에 대해서 알아보았습니다.
그런데 왜 이렇게 열심히 설명 한 걸까요?

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

요게 바로 스냅샷 덕분에 구현 될 수 있었답니다.

Socialinus

svn의 브랜치



svn은 간단히 말해 **디렉토리 구조**입니다.

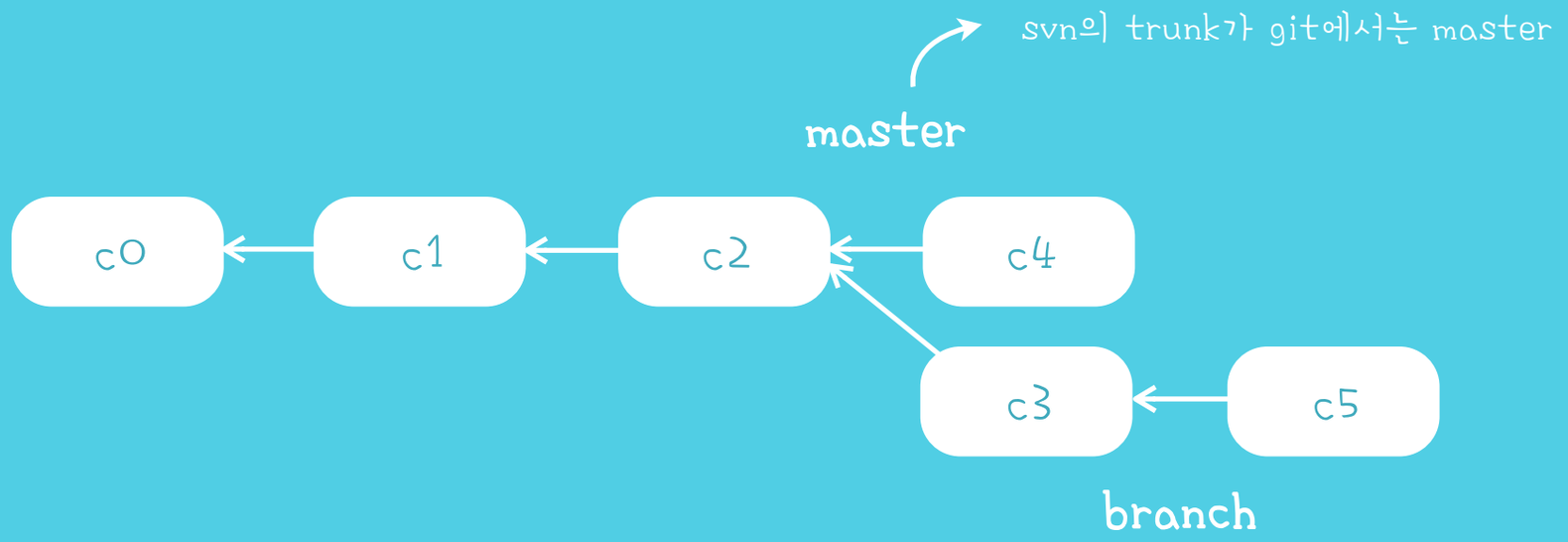
서버는 단지 변경사항만 저장하며
최소한의 자료구조를 유지하지만,

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

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

Socialinus

git의 브랜치

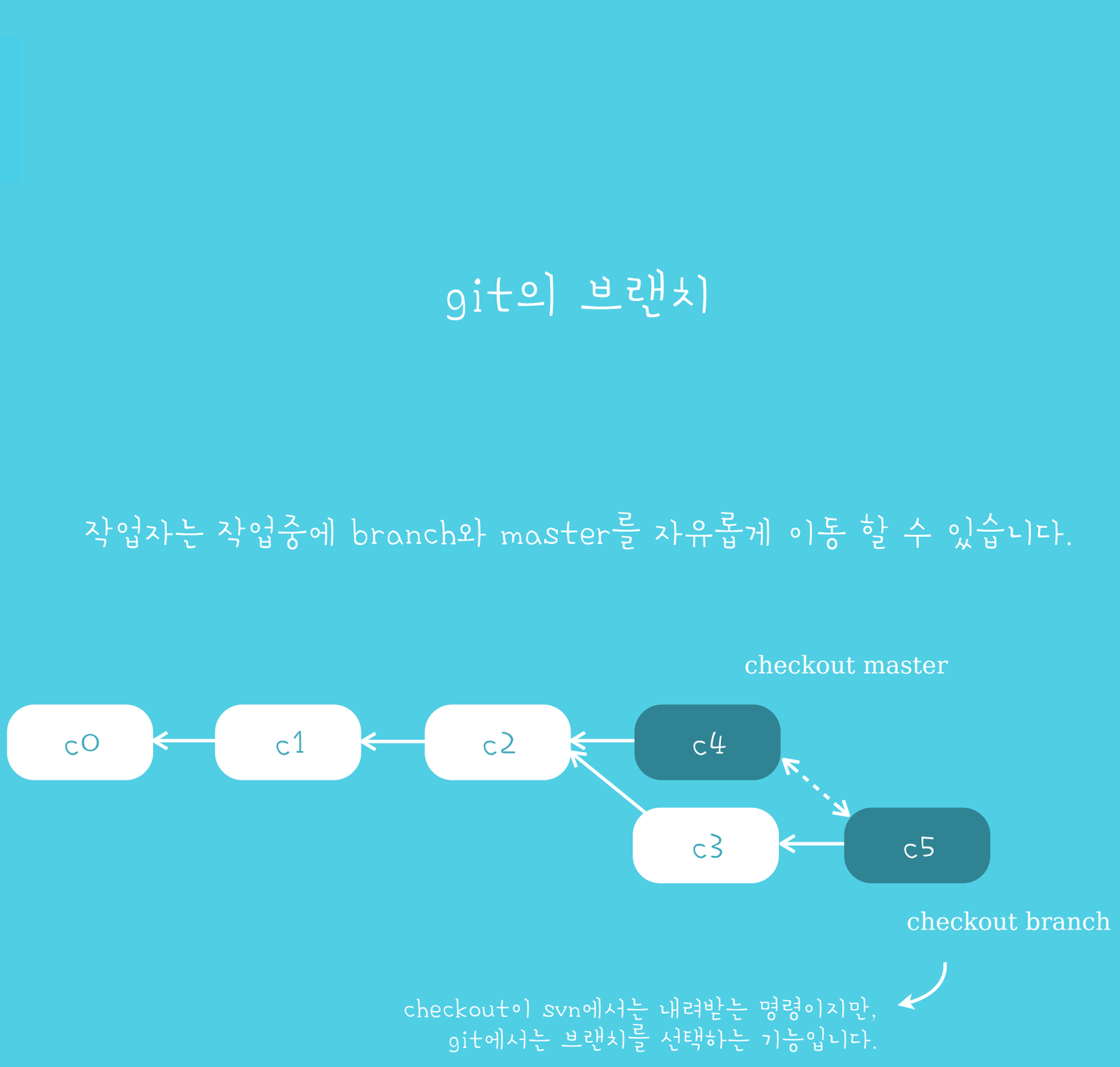


git은 서버에 그런 논리적 디렉토리 구조를 만들지 않습니다.
다만, **연속된 스냅샷이 순차적으로** 이어지다가 가지를 치면서 브랜치가 만들어집니다.

Socialinus

git의 브랜치

작업자는 작업중에 branch와 master를 자유롭게 이동 할 수 있습니다.



checkout이 svn에서는 내려받는 명령이지만,
git에서는 브랜치를 선택하는 기능입니다.

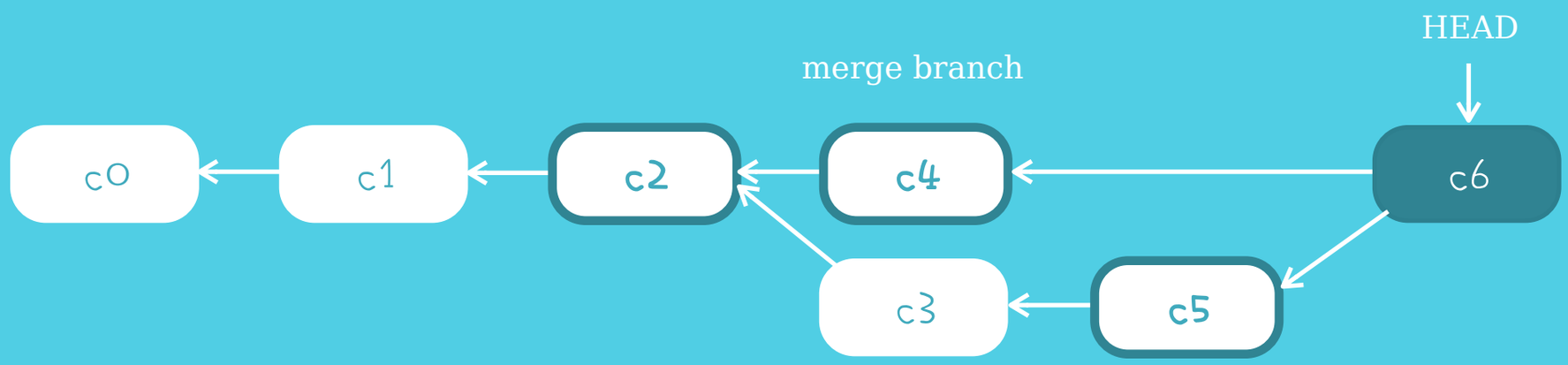
Socialinus

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

Socialinus

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 **자동**으로 해결합니다.

Socialinus

git은 수많은 브랜치가 동시에 돌아가는 것을 장려합니다.

svn이었다면 상상도 못할 일이었겠죠?

① 출시를 위한 **프로덕션 용 브랜치**를 만듭니다.

② **기능 추가** 혹은 **버그 수정**을 위해 그때그때 브랜치를 만듭니다.

③ **아이디어를 실험** 해 보기 위해 브랜치를 만듭니다.

여러 커밋을 거친 branch를 merge하는 작업을
merge commit 이라고 표현하기도 합니다.

Socialinus

git의 브랜치 관련 명령어는 다음과 같습니다:

브랜치 만들기

```
git branch <name>
git checkout <name>
```

→ `git branch -b <name>`

두 명령을 한꺼번에 수행하는 명령

브랜치 지우기

```
git branch -d <name>
```

브랜치 병합

```
git checkout master
git merge branch
```

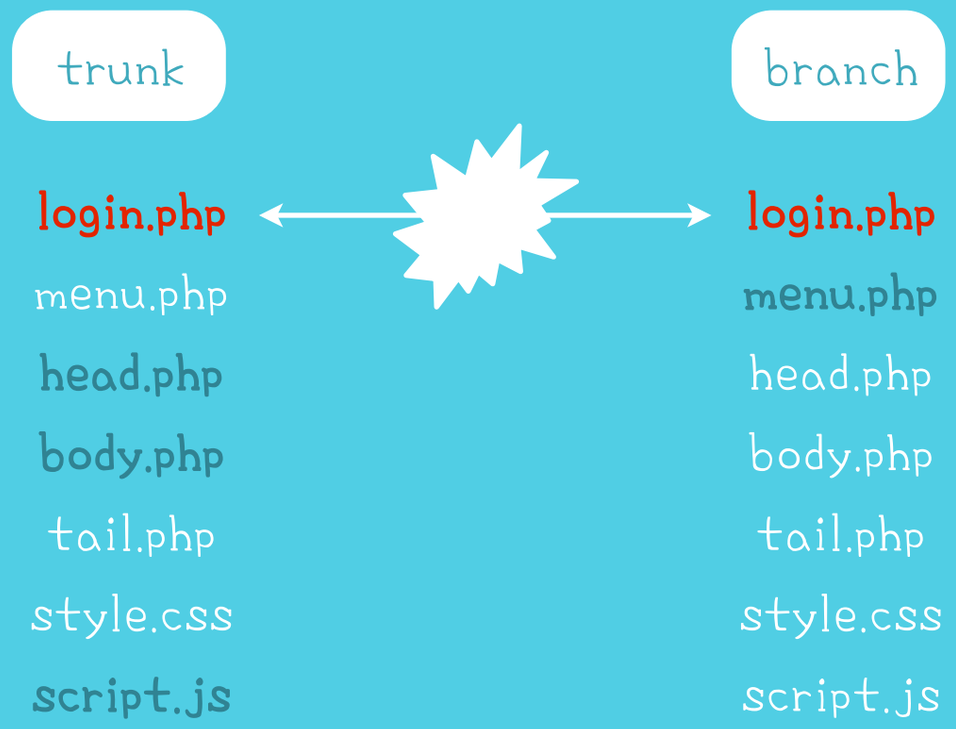
브랜치 목록

```
git branch
git branch --no-merged
git branch --merged
```

→ branch가 자주 생성, 소멸되므로
현재 존재하는 브랜치 중 어떤 것들이
merge가 되었는지, 안되었는지를
구분 할 수 있어야 합니다.

Socialinus

merge중에 충돌은 언제나 일어날 수 있습니다.



Socialinus

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.

Socialinus

리베이스rebase

rebase는 merge와 동일하게 하나의 브랜치를 **다른 브랜치로 병합**하는 기능을 합니다.

단, 차이점은?

두 브랜치의 결과를 병합하는 merge와 달리,
rebase는 한 브랜치에서 이루어진 **작업 히스토리를 다른 브랜치에 차례대로 적용**시킵니다.

왜 쓰는거에요?

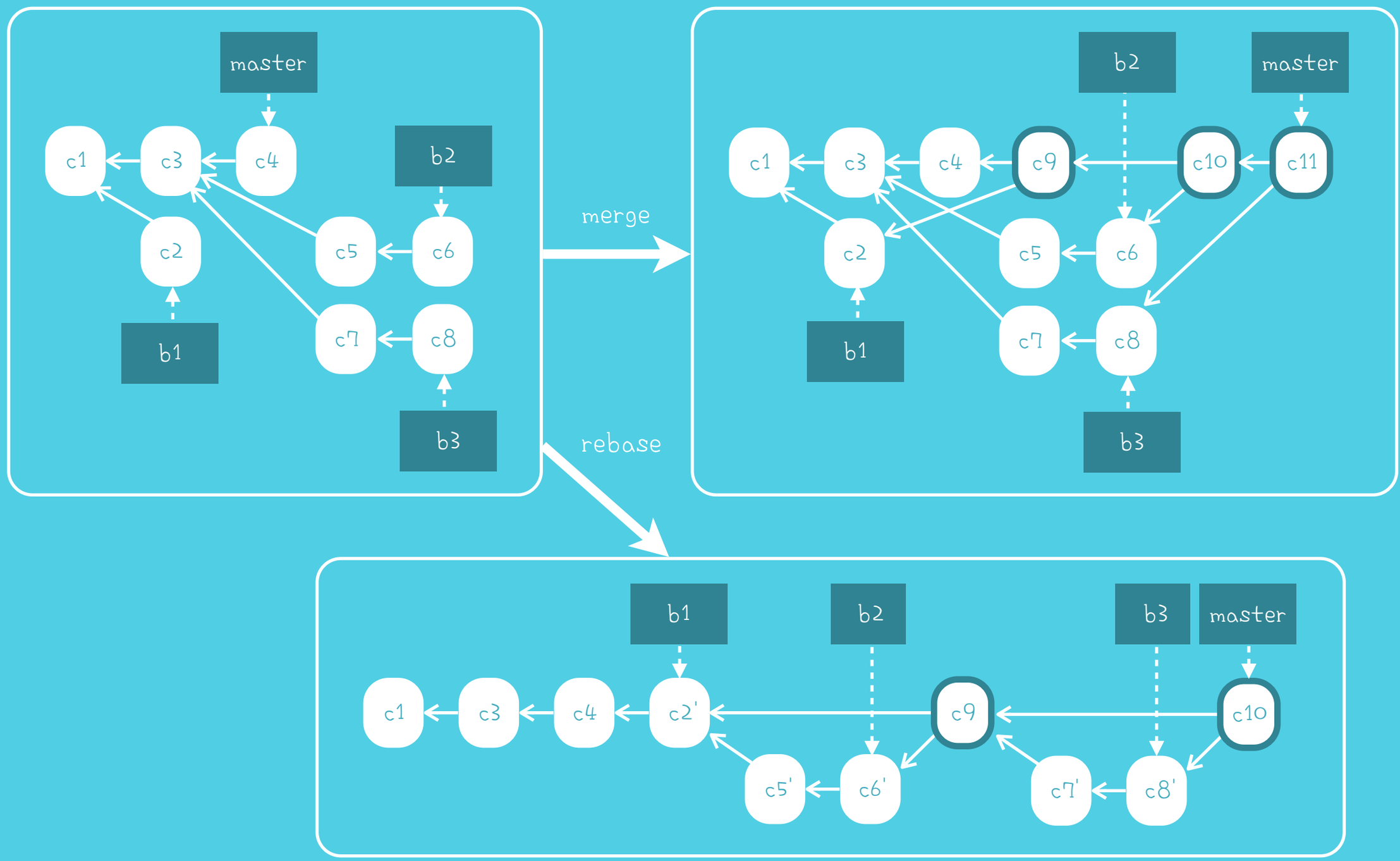
둘의 결과물은 완전히 동일하지만,
rebase는 병합 히스토리를 더욱 깔끔하게 남겨줍니다.

**공동 작업에서 깔끔한 히스토리를 계속 유지해주고 싶다면
rebase를 적극적으로 사용하는 것이 좋습니다.**

Socialinus

리베이스rebase

merge와 rebase를 비교합니다.



Socialinus

태그tag

branch처럼 tag 역시 자체 명령어를 지원하고 있습니다.

태그 만들기

```
git tag <name>
git tag -a <name>
```

→ -a를 붙이면, 작업자, 설명등을 추가로 저장합니다.

태그 지우기

```
git tag -d <name>
```

태그 목록

```
git tag
```

Socialinus

지금까지 git만의 특징들을 알아보았습니다.

정리하면,

① 로컬저장소와 원격저장소의 분리

·····➤ **분산 처리, 안전한 데이터, 빠른 처리 속도**

② 스테이지 영역의 존재

·····➤ **커밋 대상의 분리**

③ 스냅샷을 이용한 버전 관리

·····➤ **빠르고 편리한 브랜치 & 병합 기능**

이어서 알아두어야 할 git의 나머지 기능들을 살펴보겠습니다...

Socialinus

git으로 처음 작업을 시작합니다.

```
git init                          // 내 컴퓨터에 새로 빈 로컬저장소를 만들고,
git remote add <repository>       // 이 로컬저장소를 원격저장소에 연결 한 후,
git pull                          // 원격저장소로부터 데이터를 내려받습니다.
```

이 방식을 이용하면 **로컬저장소와 원격저장소를 수동으로 연결** 할 수 있습니다.
하지만 조금 번거롭죠?

Socialinus

처음 시작하는 이런 방법도 있습니다.

로컬저장소는 원격저장소와 쌍둥이입니다.

git clone <repository> // 한번의 명령으로 원격저장소로부터 데이터를 내려 받습니다.

ex) git clone /local/repository/path

ex) git clone user@hostname:/remote/repository/path

Socialinus

이번에는 실수했을 때 되돌리는 방법입니다.

의외로 많이 사용하는 기능이죠?

svn은 절차가 간단한 반면, git은 더 많은 단계로 인해 좀 더 복잡합니다.

```
git checkout --                          // 수정했지만 아직 스테이지에 추가하지 않은
                                            파일을 되돌릴 때,

git reset HEAD                           // 수정했고 스테이지에 추가한 파일을 다시
                                            스테이지에서제거 할 때,

git fetch origin                         // 수정했고 로컬저장소에 commit 한 파일을
git reset --hard origin/master              되돌릴 때,
```

Socialinus

모르면 아쉬운 git의 특징들

전해드리는 참고사이트에서 더욱 자세하게 살펴보실 수 있습니다.

① 작업 중 브랜치를 변경해야 하는데 커밋하기 애매할 때, 간단히 저장해둘 수 있습니다.
stashing이라고 해요!

② 커밋의 순서를 변경하거나 커밋 메시지 심지어는 커밋한 파일도 변경 할 수 있습니다.

③ 로그를 자기 입맛대로 포맷팅해서 볼 수 있습니다.

④ 이전 커밋에 추가 할 수 있습니다.

Socialinus

만약 이렇다면 git이 필요 없어요.

① 브랜치나 태그 기능은 사용하지 않아요.

② 관리하는 파일들의 규모가 작아서 오래전 버전에 접근하는 데 불편하지 않아요.

③ 지금까지 svn 서버가 문제가 되어 곤란을 겪은 적이 없어요.

④ svn이 느리다는 생각은 한번도 해본 적이 없어요.

Socialinus

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

여러분의 프로젝트에  **git**을 시도해보시기를 권해드립니다.

Socialinus

참조 사이트 일람

Git
http://git-scm.com/

git - 간편 안내서
http://rogerdudler.github.com/git-guide/index.ko.html

Pro Git 서적 번역
http://git-scm.com/book/ko

Git: Rebase는 언제 어떻게 해야 할까?
http://dogfeet.github.com/articles/2012/git-merge-rebase.html