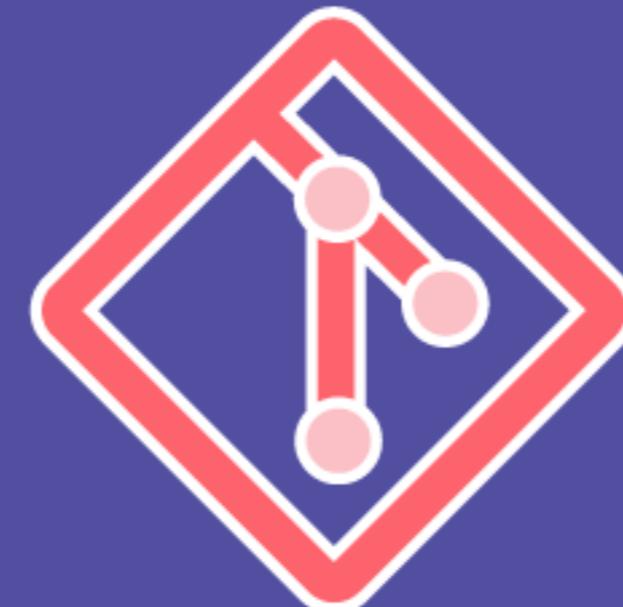


Git을 사용한 버전 관리

Git 이란?



/* elice */

커리큘럼

1

Git이란?

프로젝트 협업시 많이 사용되는 Git에 대해 알아보고,
Git을 사용하는 이유에 대해 알아봅니다.

2

Git 시작하기

Git 명령어에 대해 배우고,
이를 활용하여 나만의 Git 저장소를 만들어 봅니다.

3

Git 가지 치기

동일한 소스코드를 함께 공유하고 다루게 도와주는
Git의 가지 치기에 대해 알아보고,
마주칠 수 있는 문제에 대해 알아보겠습니다.

커리큘럼

4

Git 원격 저장소

원격 저장소를 연결하여 이용하는 방법을 배워봅시다.

이를 이용해, 서버에 있는 저장소를 이용해 다른 사람과 협업하는 방법을 배워보겠습니다.

*

Git 파고들기

부록

Git에 대한 심층적 이해를 돋기 위한 부록입니다.

수강 대상



정보기술 직무 종사자



Git을 통한 **프로젝트 관리**를 하고자
하는 개발자



Github와 같은 Git 호스팅 서비스를
이용하여 **활용**하고자 하는 개발자

수강 목표

Git을 사용하는 이유를 설명할 수 있습니다.

Git 환경을 구성할 수 있습니다.

Git을 사용하여 프로젝트를 효율적으로 관리할 수 있습니다.

목차

1. Git을 사용하는 이유
2. Git의 특징
3. Git 설치와 초기 설정
4. Git 저장소 생성

효율적인 협업



효율적인 협업

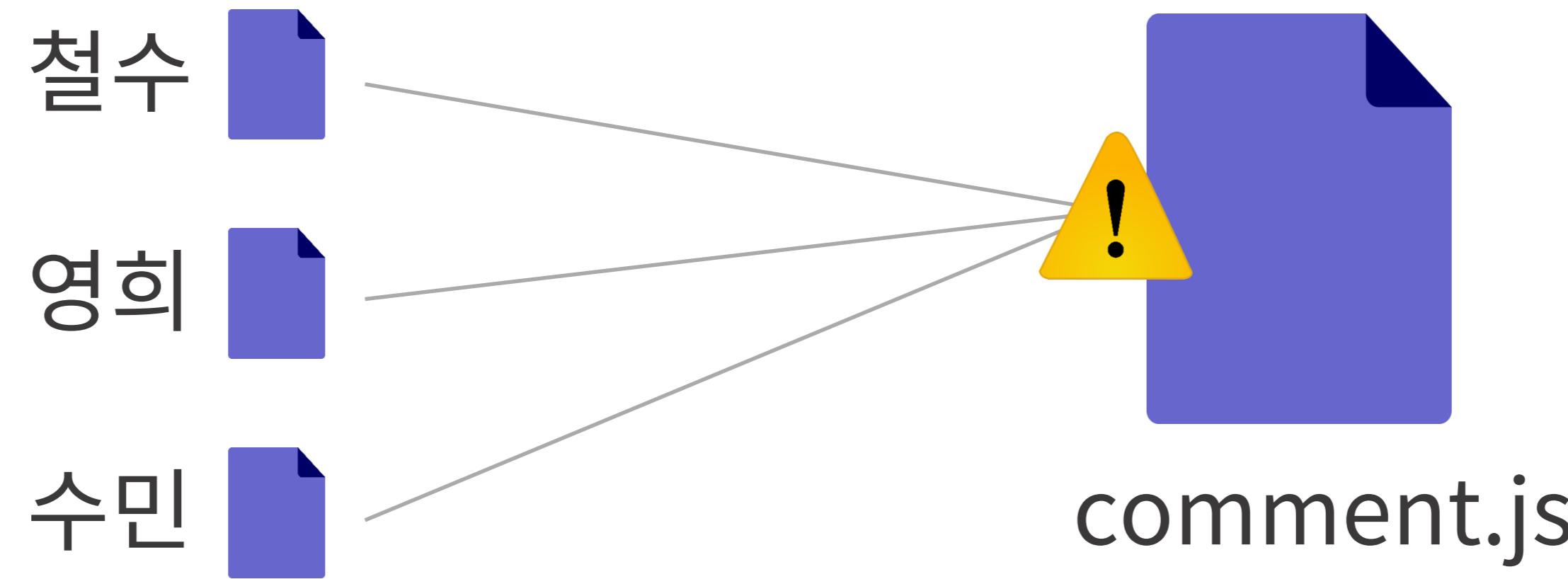
철수

comment.js 오류 사항 수정 했어.

영희

어? 내가 수정한 부분은 다 어디갔지?

효율적인 협업



쉬운 버전관리



comment.js



comment.js.bak



comment.js.bak2



comment.js.bak-180201

쉬운 버전관리



comment.js

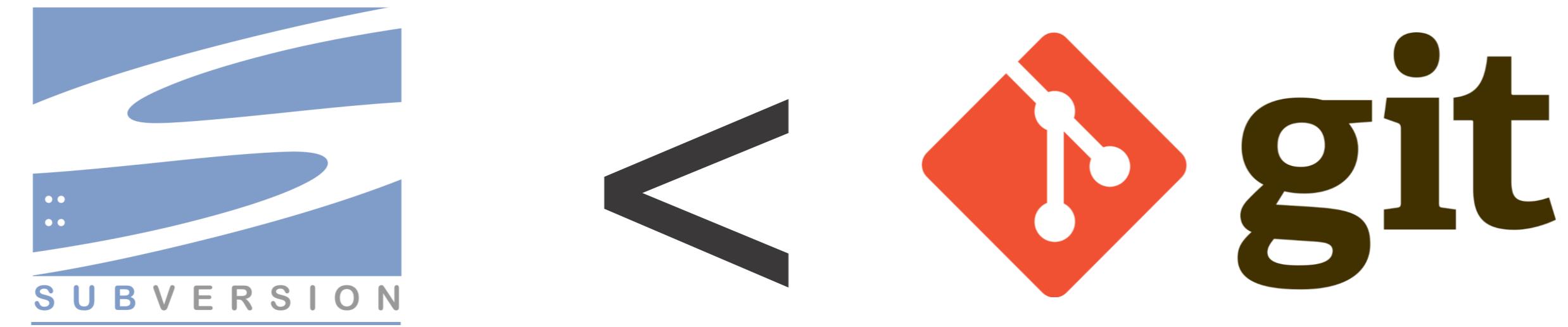
Git의 특징

1. 가지 치기와 병합



그림 git-scm.com

2. 가볍고 빠르다



3. 분산 작업

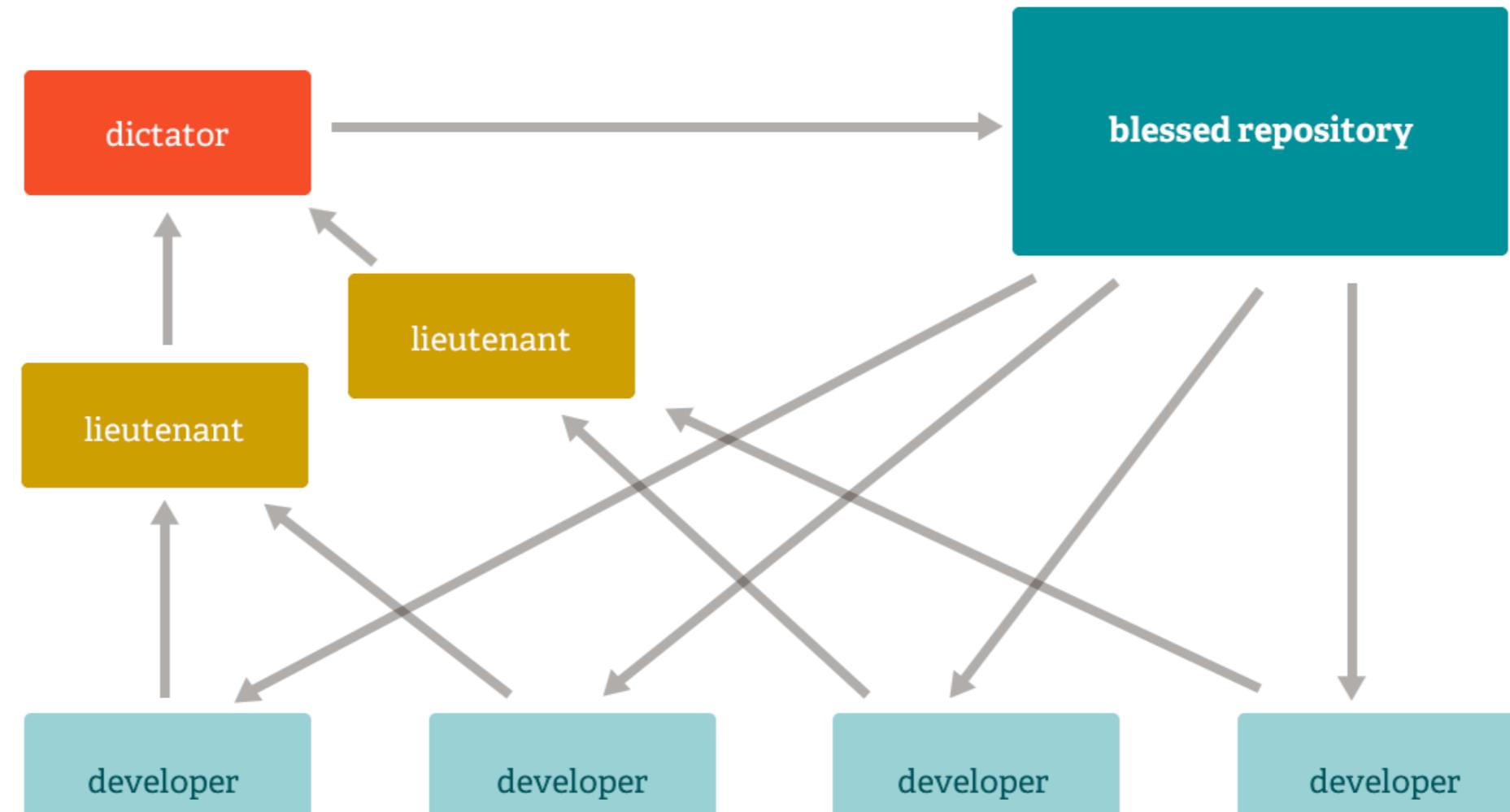


그림 git-scm.com

4. 데이터 보장

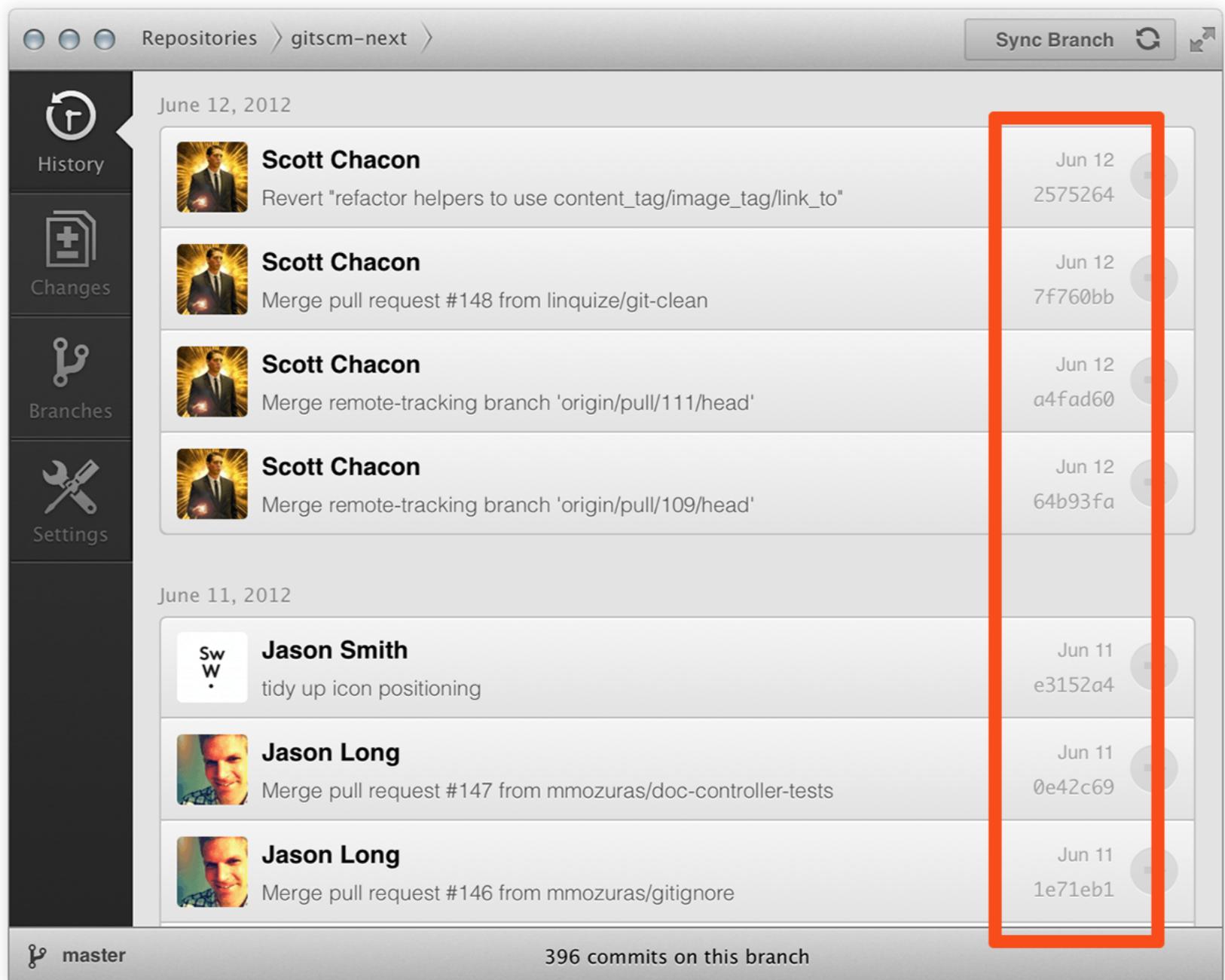


그림 git-scm.com

5. 준비 영역 (Staging area)

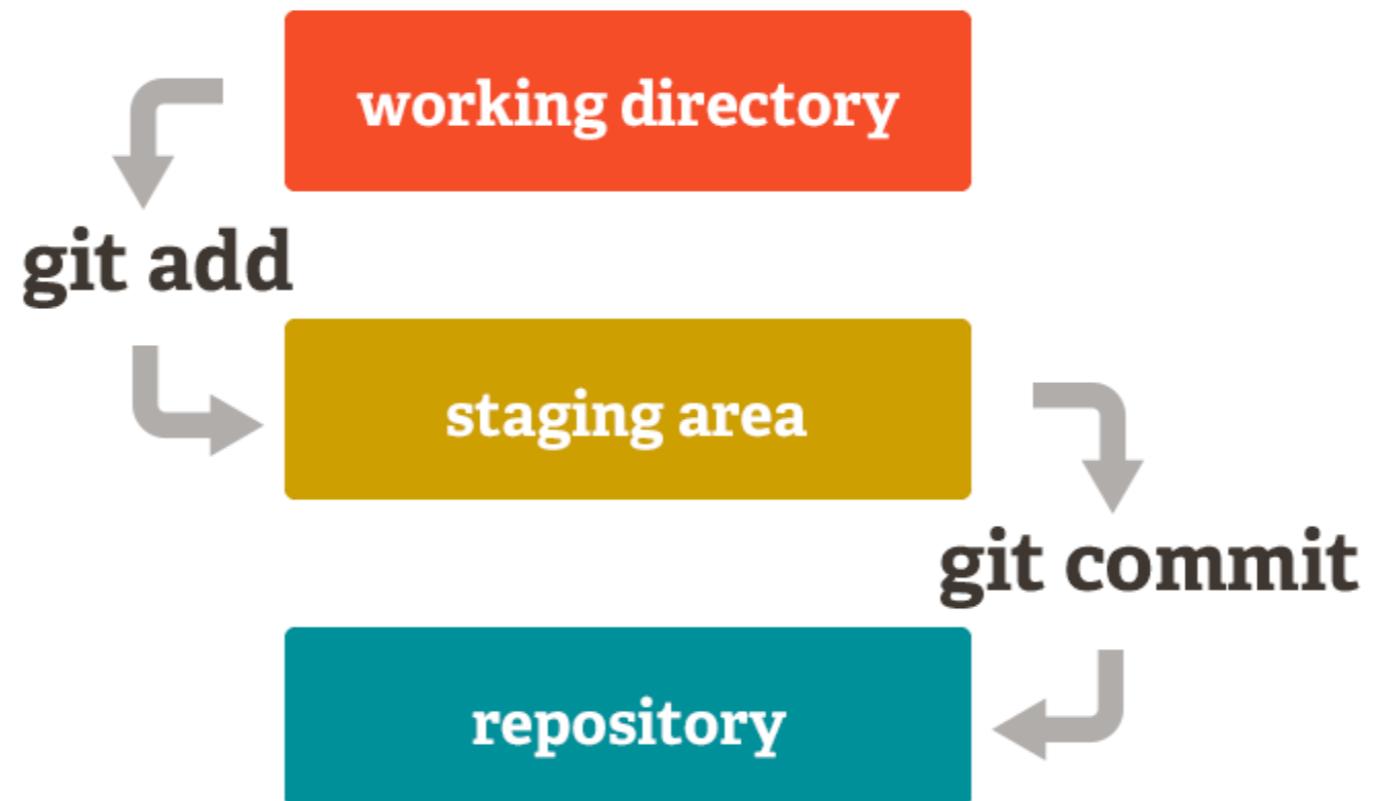


그림 git-scm.com

6. 오픈 소스



Git 호스팅 서비스



Git 설치와 초기 설정

Git 설치

1. Git 설치 여부 확인

- Linux 또는 macOS 환경에서는 대부분 이미 Git이 설치되어 있습니다.
- 다음 화면을 참고하여 Terminal 실행 후 Git 명령어를 실행해보세요.
- Windows 환경에서는 다음 단계로 넘어가주세요.

Git 설치

Terminal 실행, `git` 입력 후 엔터

```
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

These are common Git commands used in various situations:

위와 같이 실행되지 않는다면 다음 단계에서
Git을 설치해주세요

Git 설치

2. Git 설치

- 아래의 사이트 접속 후 본인에게 맞는 설치 파일 다운로드

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

- 다운로드 된 설치 파일 실행

Git 설치

3. Git 실행

- Linux 또는 macOS
Terminal 실행
- Windows
시작 메뉴 → Git → Git Bash 실행

Git 설치

4. Git 설치 확인

- Git이 정상적으로 설치되었나요?

아래의 명령어를 입력하여 Git 버전을 확인해보세요.
(정확한 버전은 아래와 달라도 상관 없습니다)

```
$ git --version  
git version 2.17.2 (Apple Git-113)
```

Git 초기 설정

1. 사용자 정보 설정

- 저장소에 코드를 반영할 때 등록될 사용자 정보를 설정합니다.

```
$ git config --global user.name "elice"  
$ git config --global user.email gitaccount@elice.com
```

Git 초기 설정

1. 사용자 정보 설정

- 프로젝트마다 다른 사용자 정보를 지정하고 싶으면
저장소 생성 후 `--global` 옵션을 빼고
실행해주세요.

```
$ git config user.name "elice"  
$ git config user.email gitaccount@elice.com
```

Git 초기 설정

2. 설정 정보 확인

- 아래의 명령을 실행하여 앞에서 설정한 내용을 확인해보세요.

```
$ git config --list  
credential.helper=osxkeychain  
user.name=elice  
user.email=gitaccount@elice.com  
...
```

Git 저장소 생성

Git 저장소 생성

Git init

기존의 디렉토리를 git repository로 설정

기존 디렉토리 사용

Git을 사용할 프로젝트 폴더로 이동 후
아래의 명령어를 실행해주세요.

```
$ git init
Initialized empty Git repository in /Users/heeguchi/git_test/.git/
```

기존 디렉토리 사용

프로젝트 디렉토리에 `.git` 디렉토리가 생성되며
저장소 생성이 완료되었습니다. 간단하죠?

```
$ ls -al
drwxr-xr-x  3 heeguchi  staff   96 11 11 21:45 .
drwxr-----+ 5 heeguchi  staff  160 11 11 21:45 ..
drwxr-xr-x 10 heeguchi  staff  320 11 11 21:45 .git
```

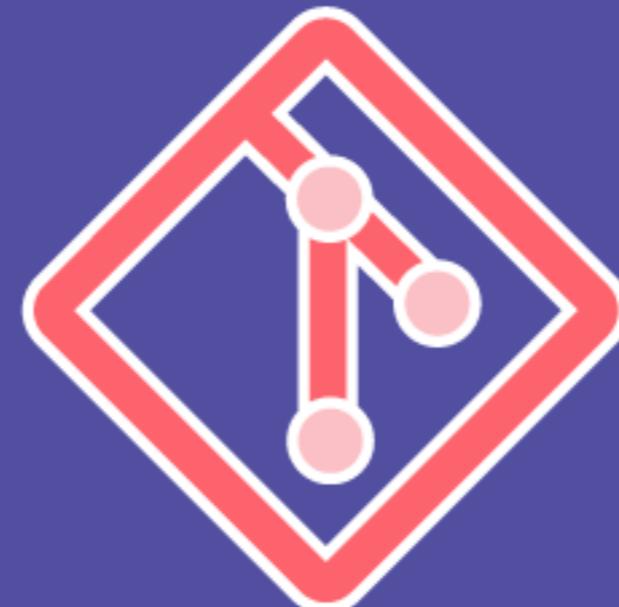


/* elice */

contact@elice.io

Git을 사용한 버전 관리

Git 시작하기



/* elice */

수강 목표

Git저장소에 작업 내용을 반영할 수 있습니다.

Git저장소의 세 가지 영역을 알 수 있습니다.

Git저장소의 현재 상태를 파악할 수 있습니다.

목차

1. Git 파일 생성
2. Git 저장소
3. Git 관리 상태 확인

Git 파일 생성

새로운 파일 생성

저장소 생성 완료 후,

새로운 `comment.js` 파일 작업을 완료하였습니다.

이 파일을 저장소에 어떻게 반영할 수 있을까요?

파일 영역의 라이프 사이클

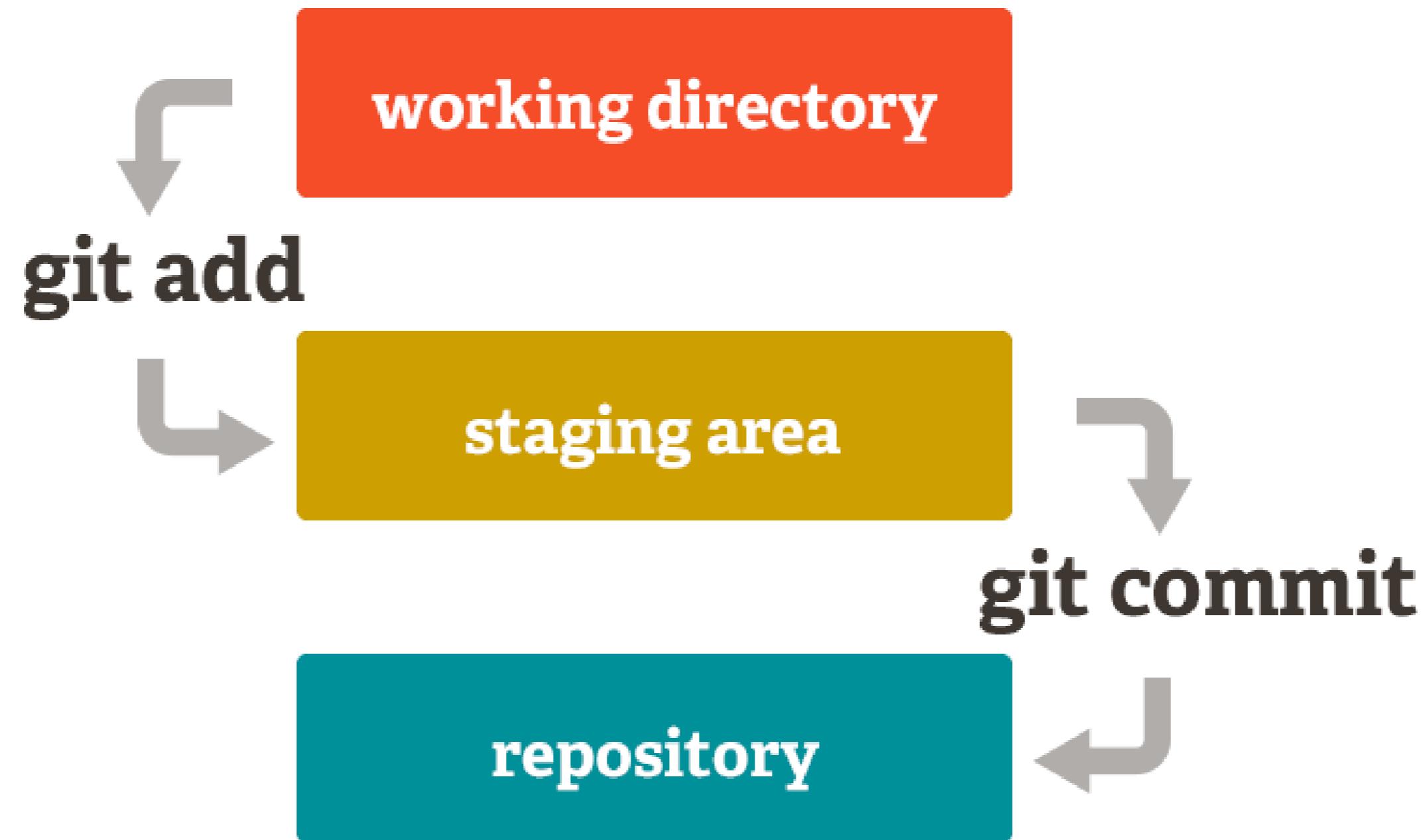


그림 git-scm.com

파일의 상태 라이프 사이클

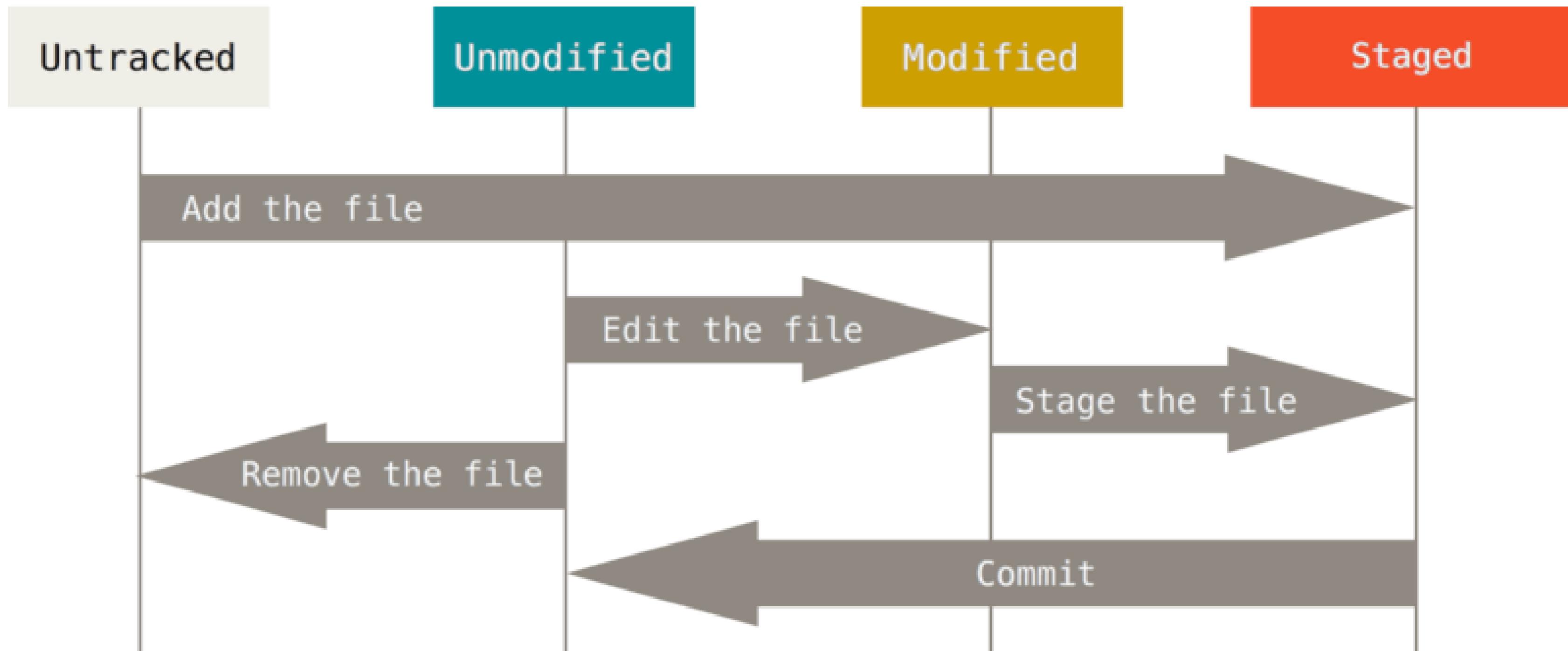


그림 git-scm.com

새로운 파일 생성

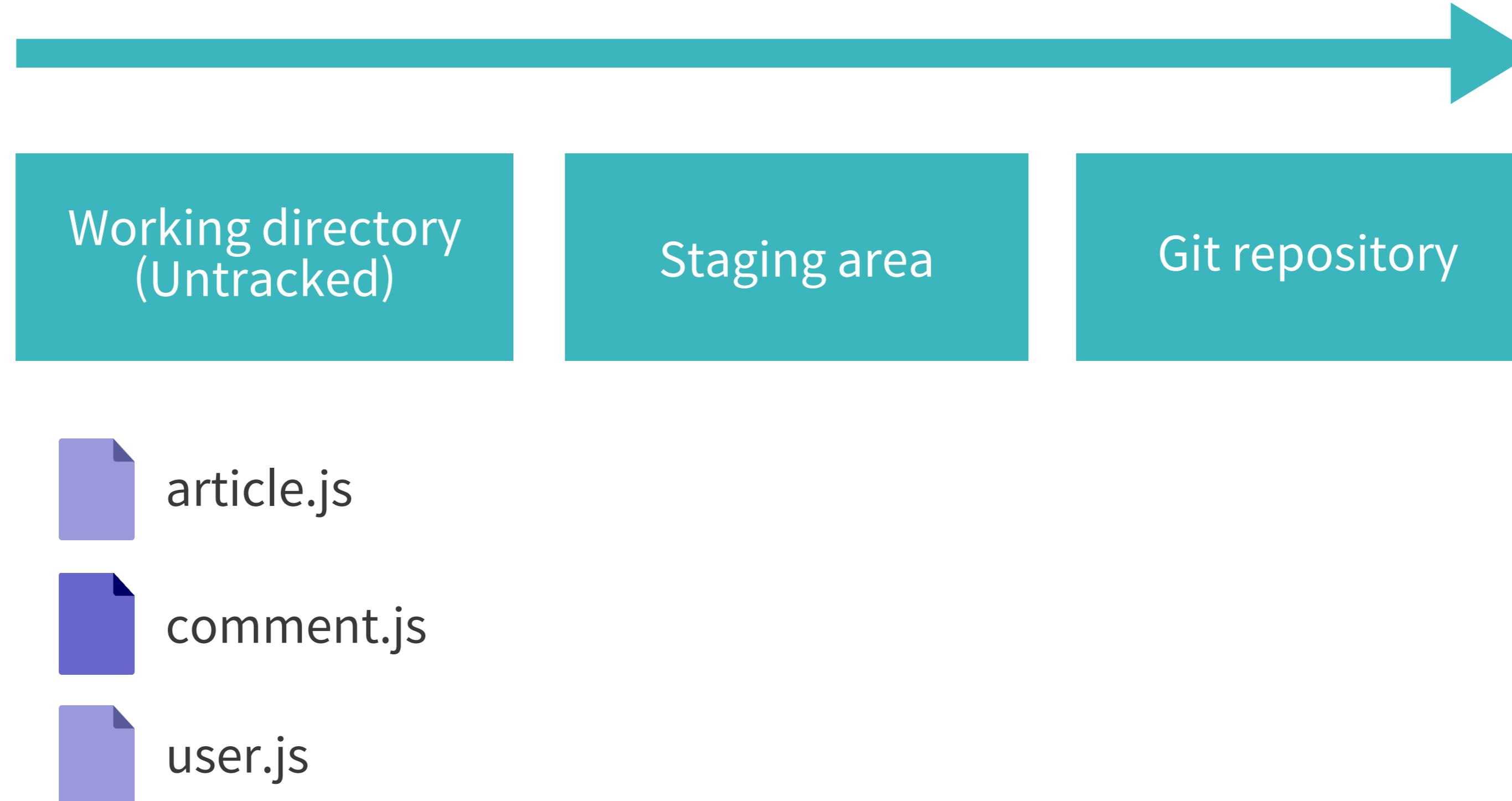
먼저, `comment.js` 파일을

준비영역으로 보내야합니다

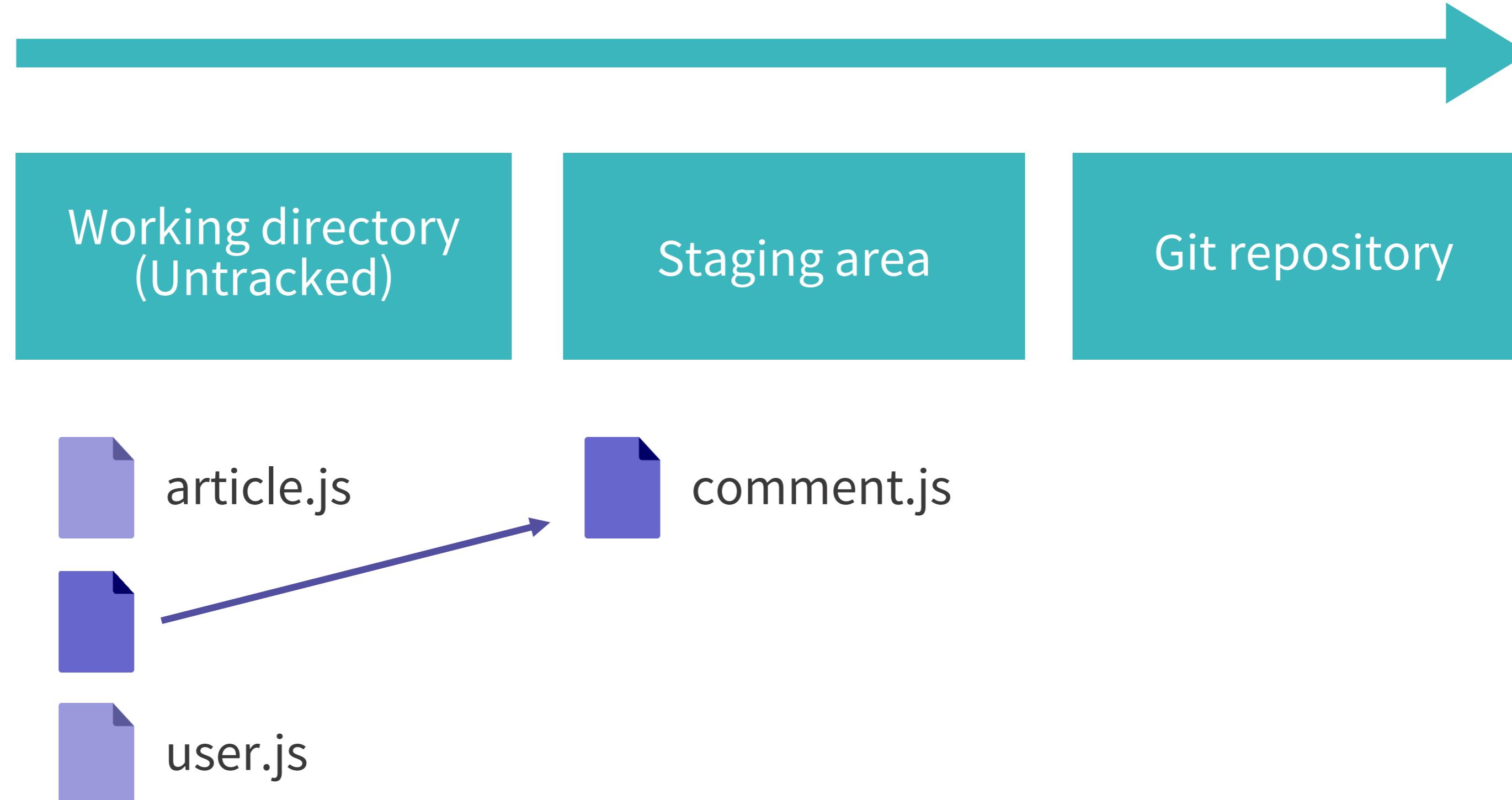
`git add` 명령어를 사용합니다

```
$ git add comment.js
```

세 가지 영역



세 가지 영역

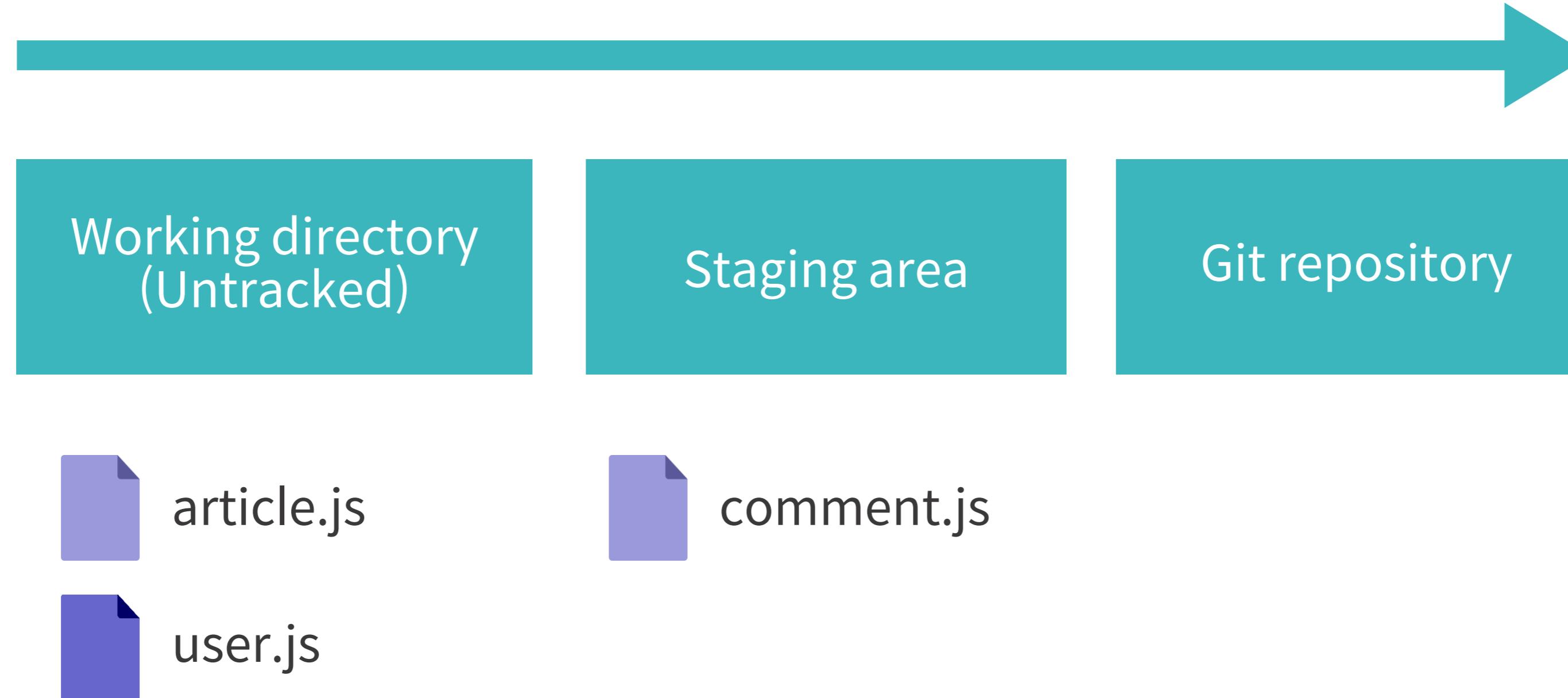


새로운 파일 생성 (1)

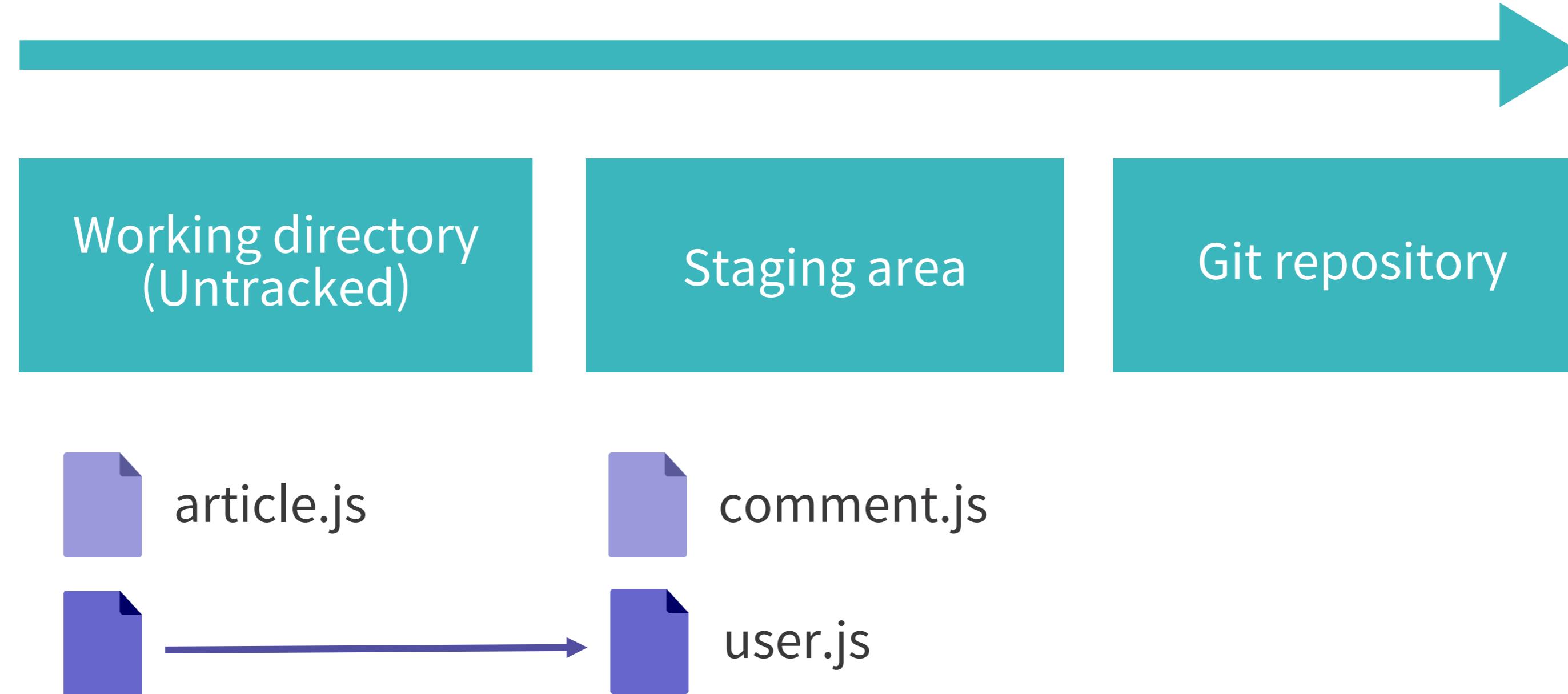
추가할 파일이 있다면 계속해서 더 추가할 수 있습니다

```
$ git add user.js
```

세 가지 영역



세 가지 영역



새로운 파일 생성 (2)

한 번에 추가할 파일이 너무 많다면
현재 폴더를 대상으로 지정할 수도 있습니다

```
$ git add .
```

Staging 상태 확인

`git status` 명령어로 Staging area의 어떤 파일이 변경되었는지 등의 파일의 상태를 확인 할 수 있습니다

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   comment.js
```

Git 저장소 반영

Git 저장소 반영

comment.js 파일 작업을 staging 하였으므로
이제 무엇을 수정하고 추가했는지 메시지를 남겨
저장소에 저장하는 작업을 진행합니다

Git 저장소 반영

git commit

.git 저장소 내에 staging 파일 저장

파일 영역의 라이프 사이클

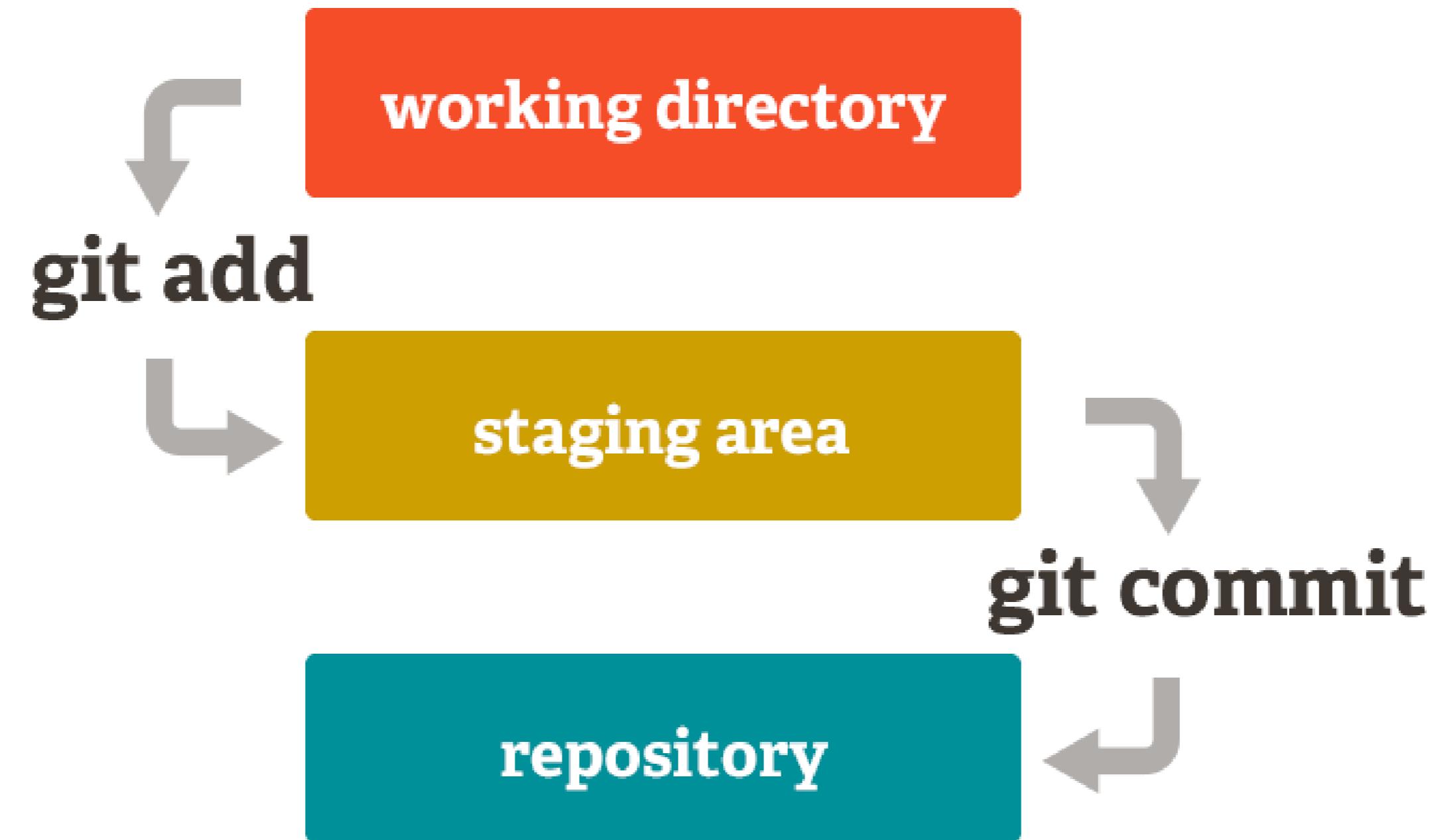
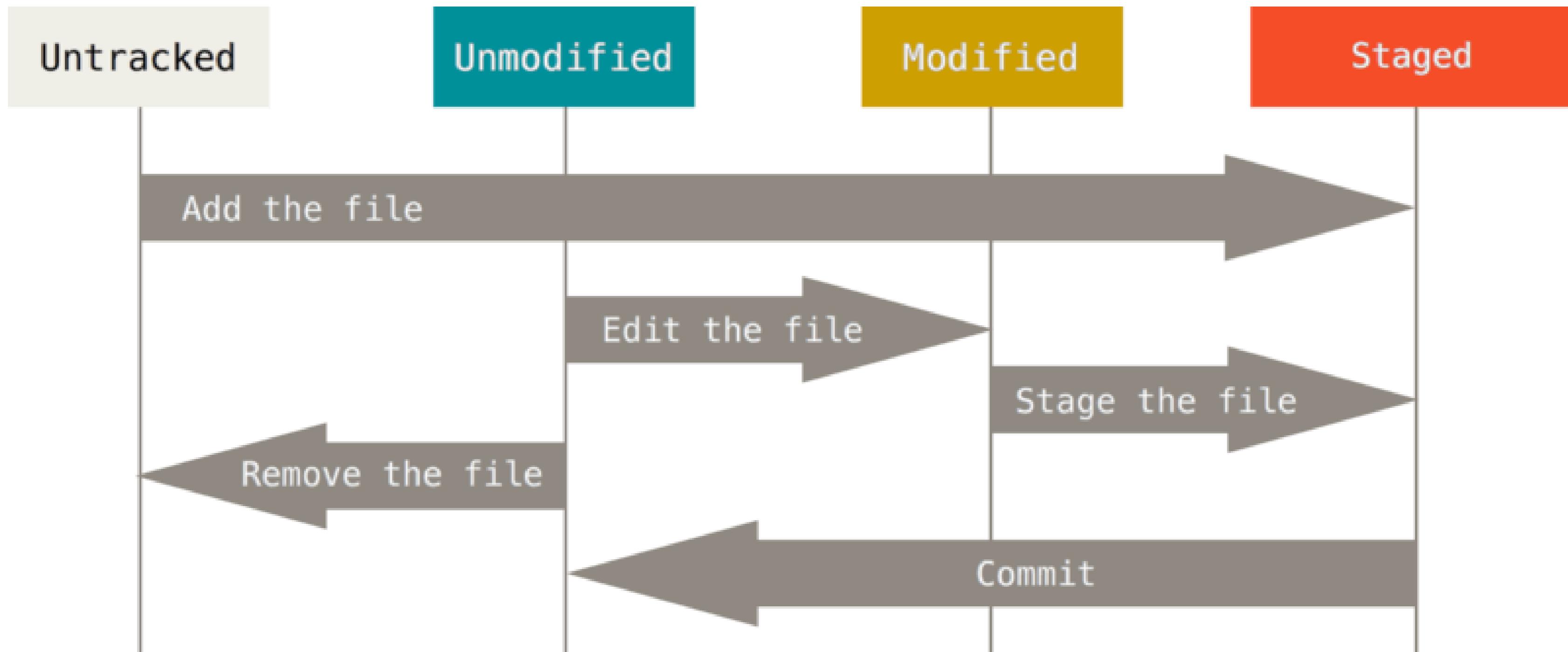


그림 git-scm.com

파일 상태의 라이프 사이클

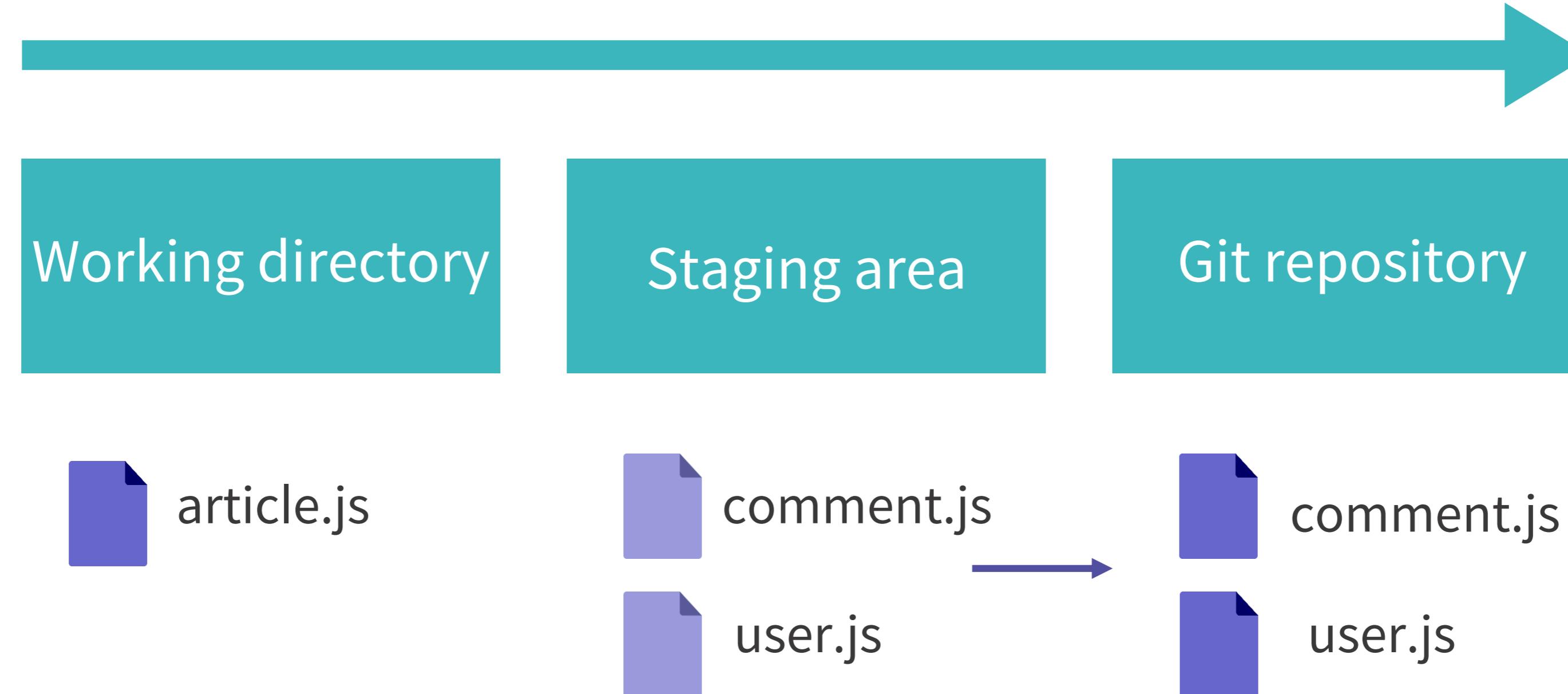


저장소 반영

준비 영역에 있는 파일들을 저장소에 반영하겠습니다.
생략 가능하지만 반영한 내용을 추후에 쉽게 알 수 있도록
적절한 메세지를 넣어주세요.

```
$ git commit -m "Initial commit"  
master (root-commit) d78ade4] Initial commit  
 2 files changed, 4 insertions(+)  
  create mode 100644 comment.js  
  create mode 100644 user.js
```

세 가지 영역



저장소 반영 내용 변경

앞에서 적은 메세지에 오타가 있거나
누락된 파일이 있을 경우 걱정하지 마세요.

```
$ git commit --amend
```

저장소 반영 내용 변경

텍스트 편집기가 실행되고,
수정하고 싶은 부분을 수정 후 저장하면 그대로 반영됩니다.

Initial commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
```

...

저장소 반영 내역

저장소 반영 내역이 궁금하시다면
아래 명령어를 사용해보세요.

```
$ git log  
commit d78ade4c54fbfe333a466c78f4c2ca66d63d6053  
Author: Elicer <elice@elice.io>  
Date:   Tue Dec 11 22:23:13 2018 +0900
```

Initial commit

Git history

Commit 된 내용은 다음과 같습니다.

d78ad

Commit size

Tree

Parent

Author

Committer

Initial commit

Git 관리 상태 확인

파일 상태 확인

Git 저장소까지 저장을 완료하였습니다.

파일들의 상태와 history를 볼 수 있는

`git status`, `git log`에 대해 자세히 살펴보겠습니다.

Git 관리 상태 확인

git status

Staging file들의 상태 확인

git log

.git repository에 존재하는 history 확인

파일 영역의 라이프 사이클

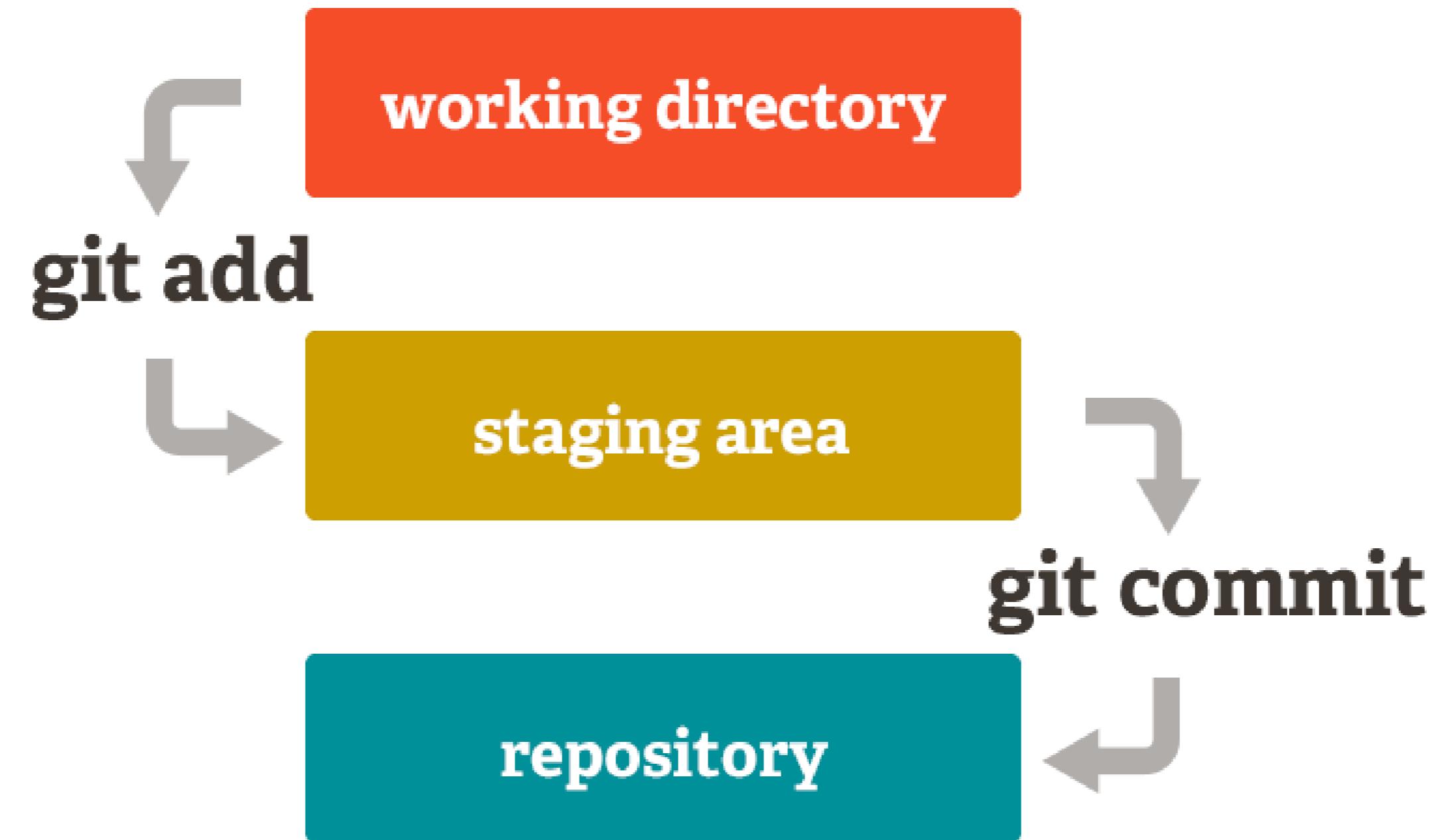
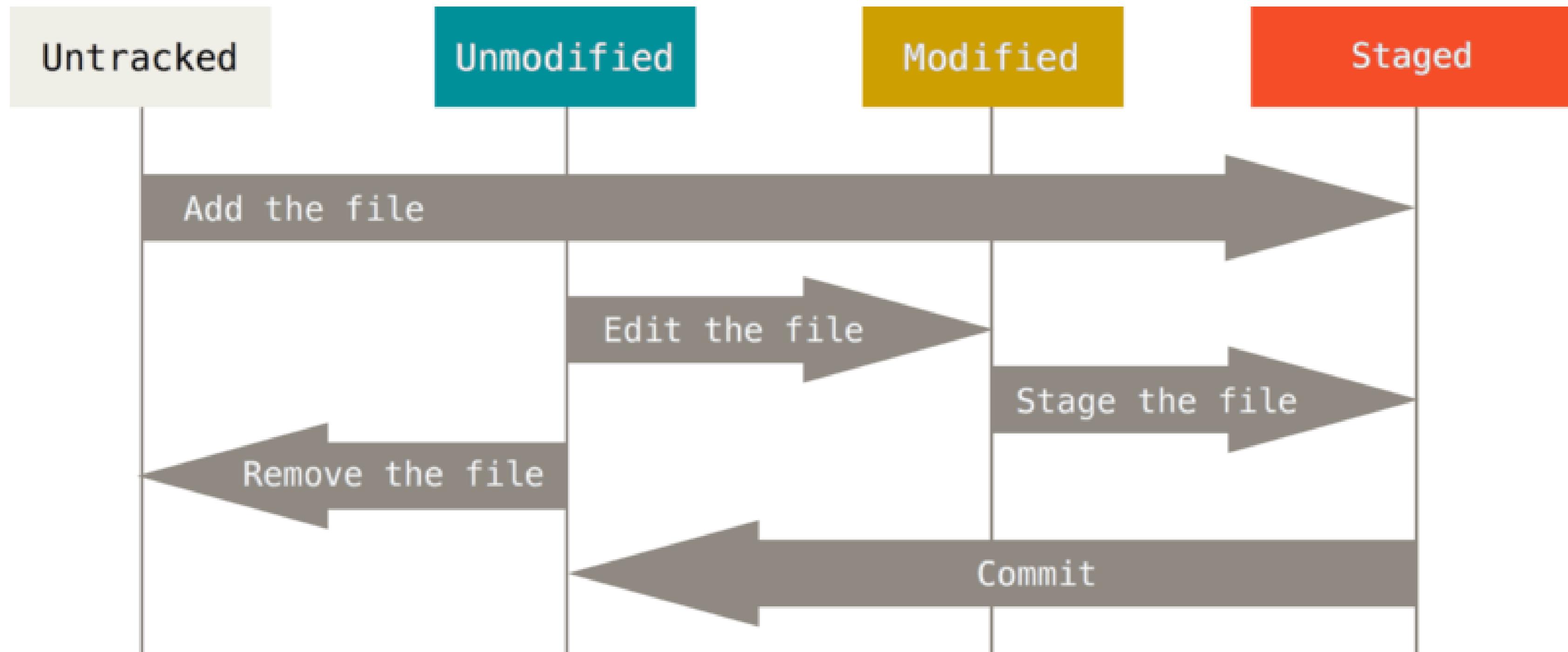
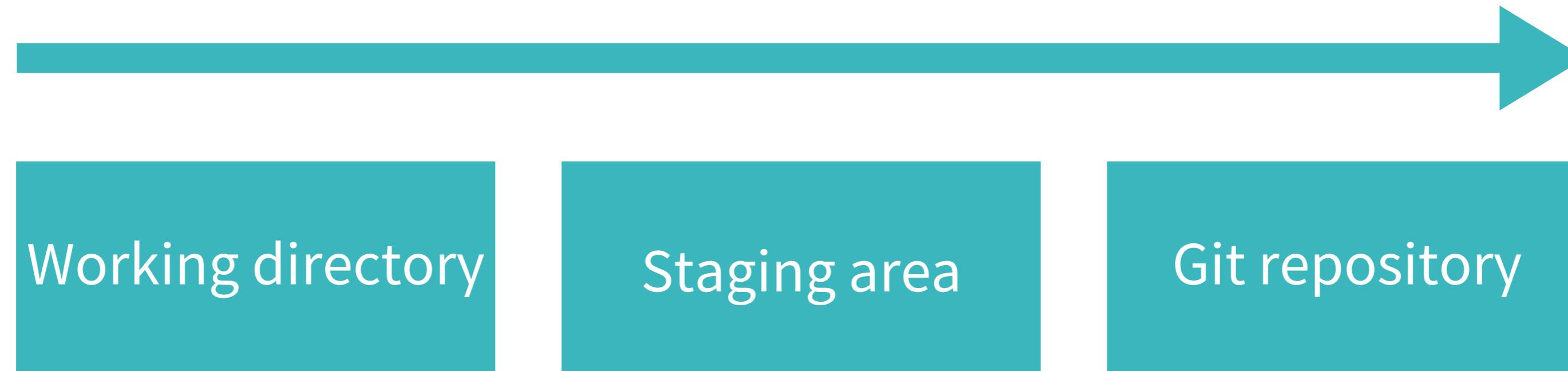


그림 git-scm.com

파일 상태의 라이프 사이클



세 가지 영역



 comment.js

 article.js

 user.js

status

Untracked : add로 staging되지 않은 파일 들

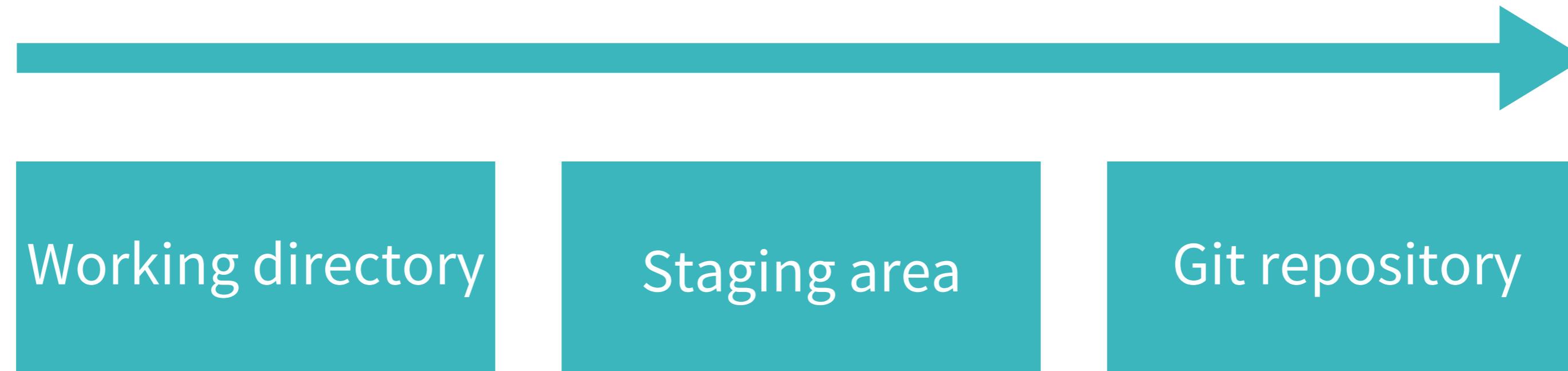
```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be
committed)
    article.js
    comment.js
    user.js

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

세 가지 영역

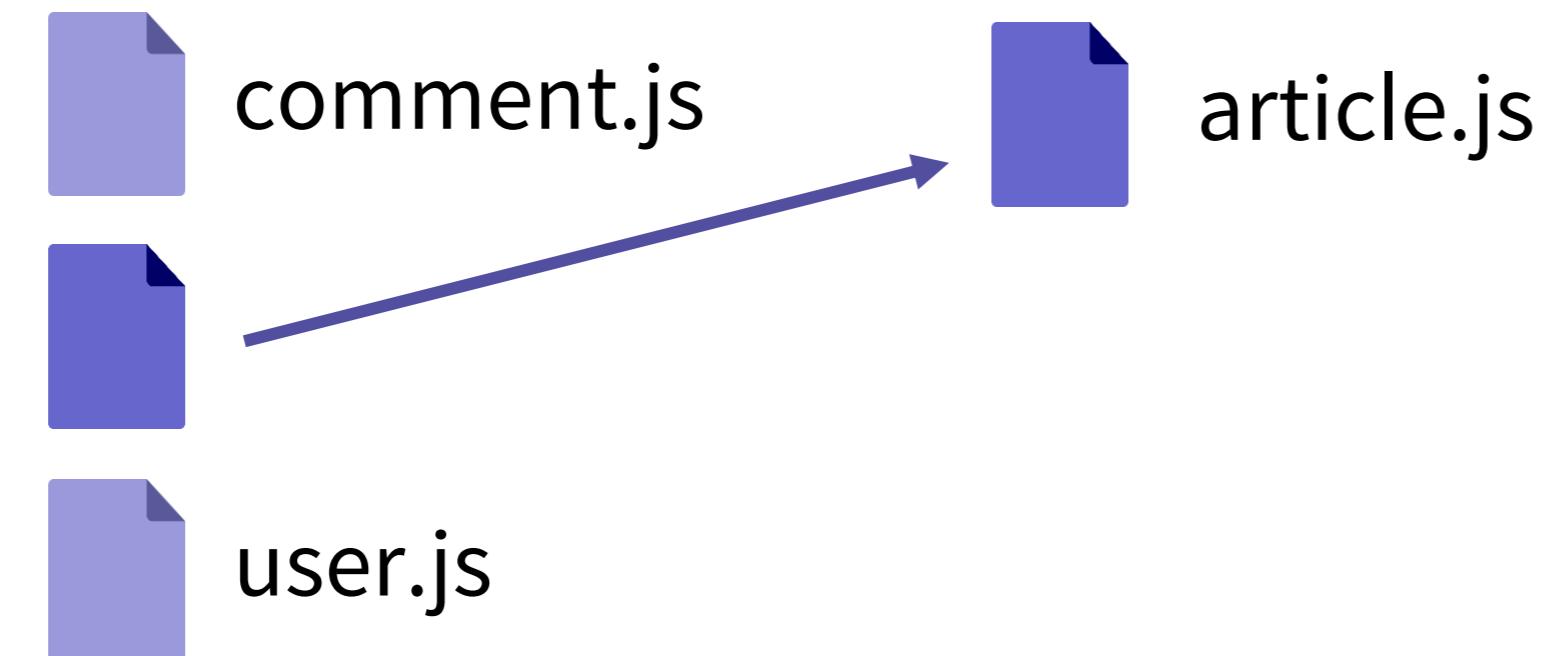
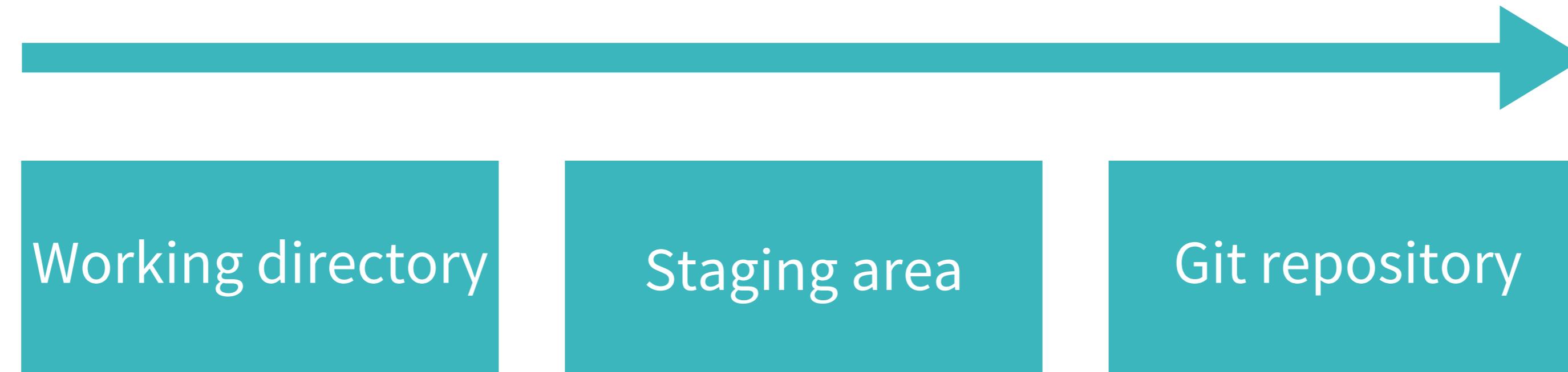


 comment.js

 article.js

 user.js

세 가지 영역



status

```
$ git status  
On branch master
```

No commits yet

```
changes to be commited:  
(use "git rm --cached <file>..." to unstage)  
      new file : article.js
```

untracked files:

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

status

```
git reset <file_name>
```

위의 명령을 이용하여 add 명령을 취소할 수 있습니다.

이제 commit된 파일을 수정할 경우 git이 파일을 어떤 상태로 인식하는지 확인해 보겠습니다.

```
article.js 를 commit 해주세요
```

세 가지 영역



Working directory

Staging area

Git repository



comment.js



user.js



(수정된)article.js



article.js (a522b)

status

modified : commit 된 파일 중 수정된 파일이 있을 경우

```
$ git status
On branch master
changes not staged for commit:
(use "git add<file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working ...)
modified : article.js
```

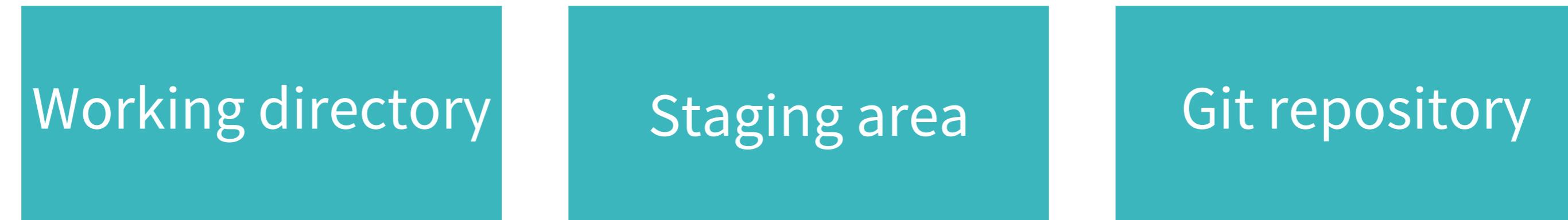
no changes added to commit (use "git add" and/or "git commit -a")

status

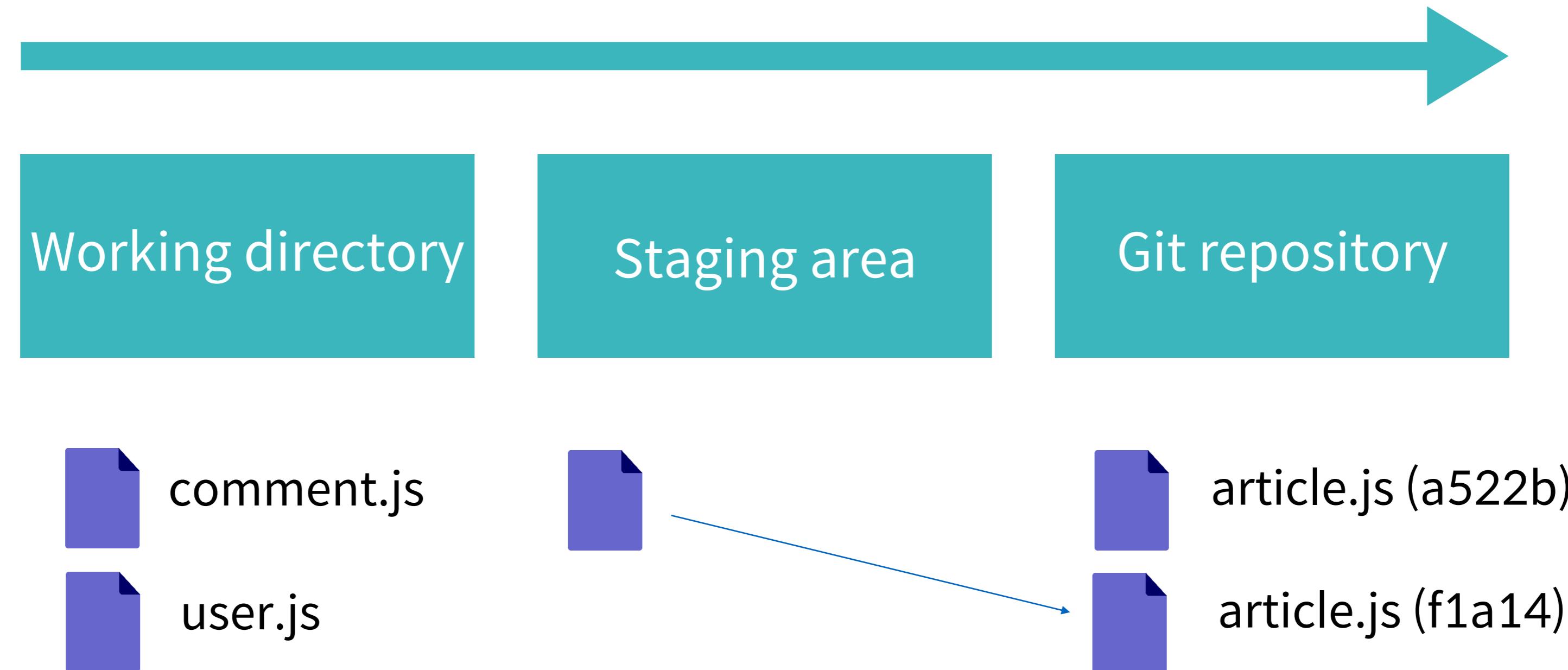
git diff : commit 된 파일 중 변경된 사항을 비교할때

```
$ git diff  
diff --git a/article.js b/article.js  
index e87ce30..a752541 100644  
--- a/article.js  
+++ b/article.js  
@@ -1 +1 @@  
-#let  
+#trysome
```

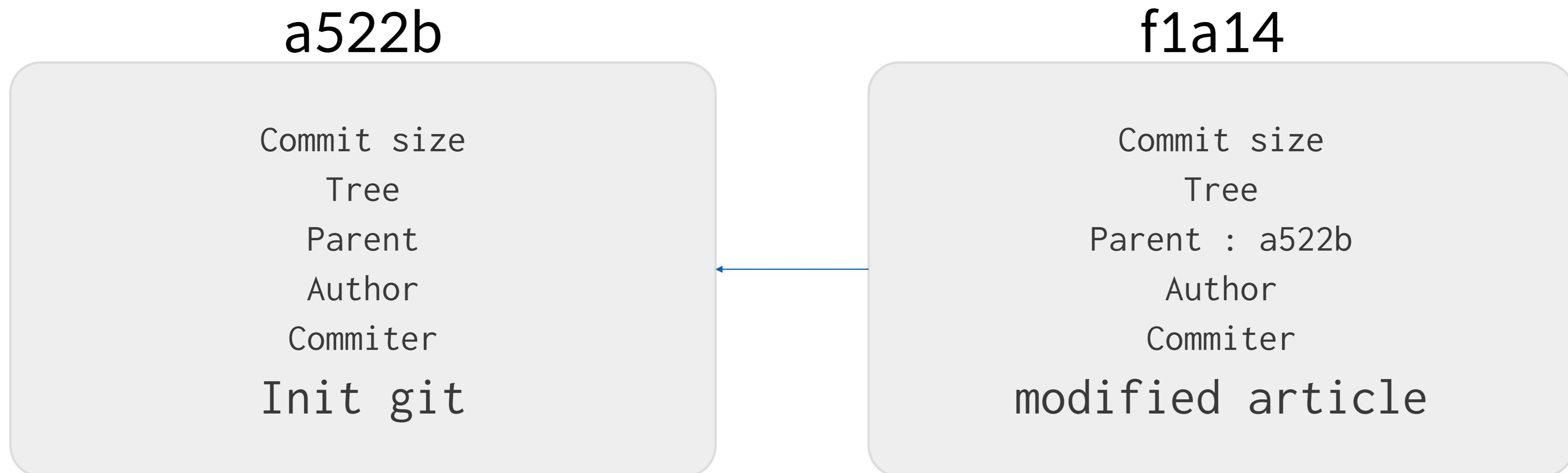
세 가지 영역



세 가지 영역



Git history



log

저장소 반영 내역을 확인하기 위하여 log 명령어를 사용합니다

```
$ git log  
commit f1a14ec47cf07fcf915e161e30f4a7b7b6752fec  
Author: unknown <unknown@unknown.me>  
Date:   Tue Jul 30 14:52:43 2019 +0900
```

modified article

```
commit a522b90e409b5f957e41c1e3e0a0206869a84696  
Author: unknown <unknown@unknown.me>  
Date:   Tue Jul 30 14:17:04 2019 +0900
```

init git

대표적인 log 옵션들

```
git log -p -2
```

-p, --patch : 각 commit의 수정 결과를 보여주는 diff와 같은 역할을 수행합니다

-n : 상위 n개의 commit만 보여줍니다

대표적인 log 옵션들

```
git log --stat
```

--stat : 어떤 파일이 commit에서 수정되고 변경되었는지,
파일 내 라인이 추가되거나 삭제되었는지 확인

```
git log --pretty=oneline
```

--pretty=oneline : 각 commit을 한 줄로 출력

대표적인 log 옵션들

```
git log --graph
```

--graph: commit간의 연결된 관계를
아스키 그래프로 출력한다.

```
git log -S function_name
```

-S: 코드에서 추가되거나 제거된 내용 중 특정 텍스트
(위에서는 function_name)가 포함되어 있는지 검사



/* elice */

contact@elice.io

Git을 사용한 버전 관리

Git 가지 치기



/* elice */

수강 목표

Git에서 **브랜치**를 만들 수 있습니다.

여러 작업을 **독립적으로** 진행할 수 있습니다.

브랜치의 내용을 **병합**할 수 있습니다.

목차

1. Git Branch
2. fast forward
3. Merge
4. conflict 해결

Git Branch

Git Branch?

독립적으로 어떤 작업을 진행하기 위한 개념

각각의 Branch는 다른 Branch의 영향을 받지 않음

Git Branch?



그림 git-scm.com

Git Branch 종류

메인 Branch

배포할 수 있는 수준의
안정적인 Branch

토픽 Branch

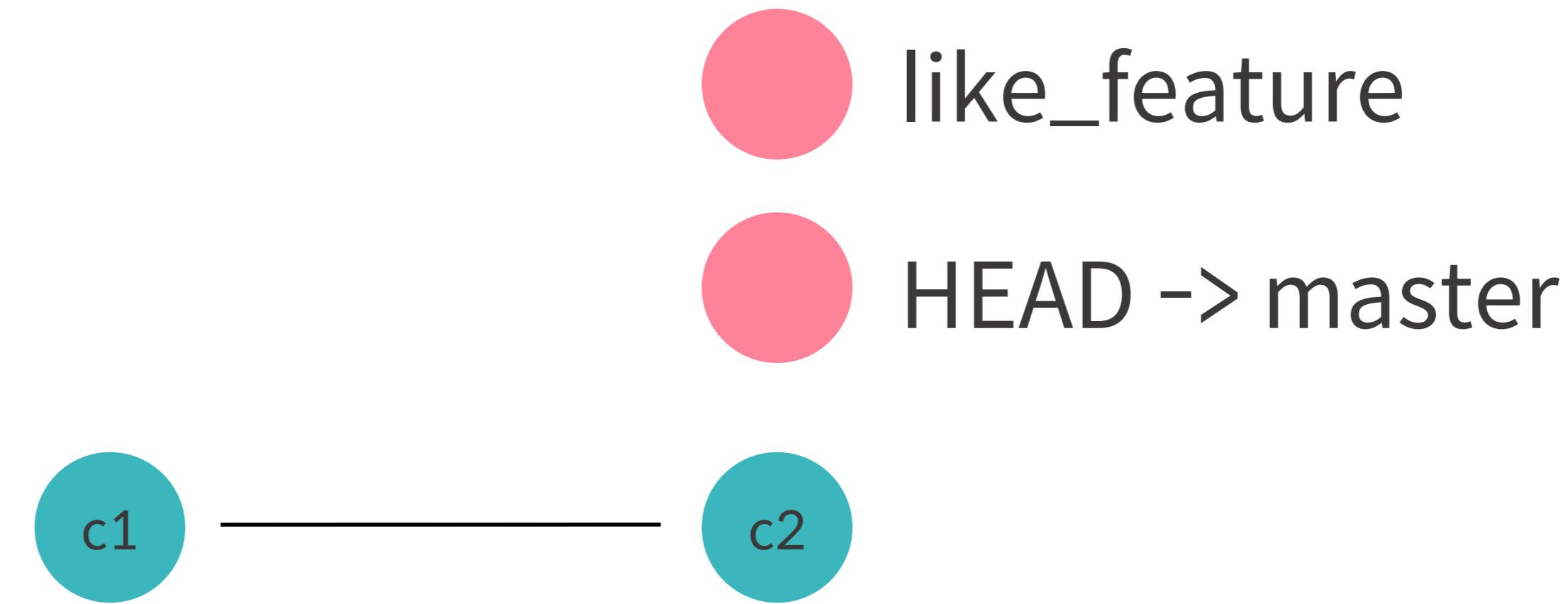
기능 추가나 버그 수정과 같은
단위 작업을 위한 Branch

Git Branch 생성

Branch는 아래의 명령어로 생성할 수 있습니다

```
$ git branch like_feature
```

Git Branch 생성



Git Branch 전환 (1)

현재의 Branch는 아래의 명령어를 통해 확인할 수 있습니다

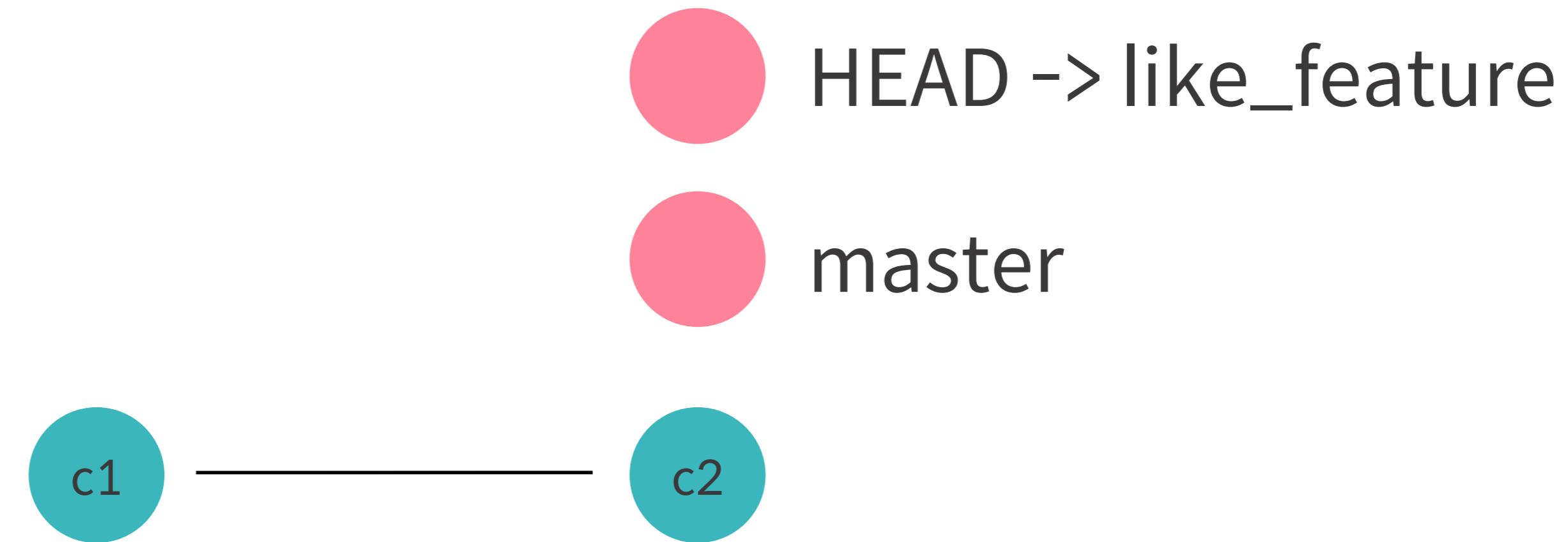
```
$ git branch  
  like_feature  
* master
```

Git Branch 전환 (2)

Branch 전환은 아래의 명령어를 통해 할 수 있습니다.

```
$ git checkout like_feature  
Switched to branch 'like_feature'
```

Git Branch 생성



Git Navigation

checkout은 branch를 전환하는데 사용할 수도 있고
아래와 같이 `git log`로 확인한 **snapshot**을 넘나들때도
사용이 가능합니다

```
git checkout <snapshot hash>
```

Git Navigation

```
$ git log --pretty=oneline  
e4abb6f... (HEAD -> master) this is master  
d97d387... another snapshot  
  
$ git checkout d97d38  
.  
HEAD is now at d97d38 another snapshot  
  
$ git log --pretty=oneline  
e4abb6f... (master) this is master  
d97d387... (HEAD) another snapshot
```

Git Navigation

이렇게 snapshot의 hash값을 이용하여
과거의 파일 내용을 확인 할 수 있습니다.

fast - forward

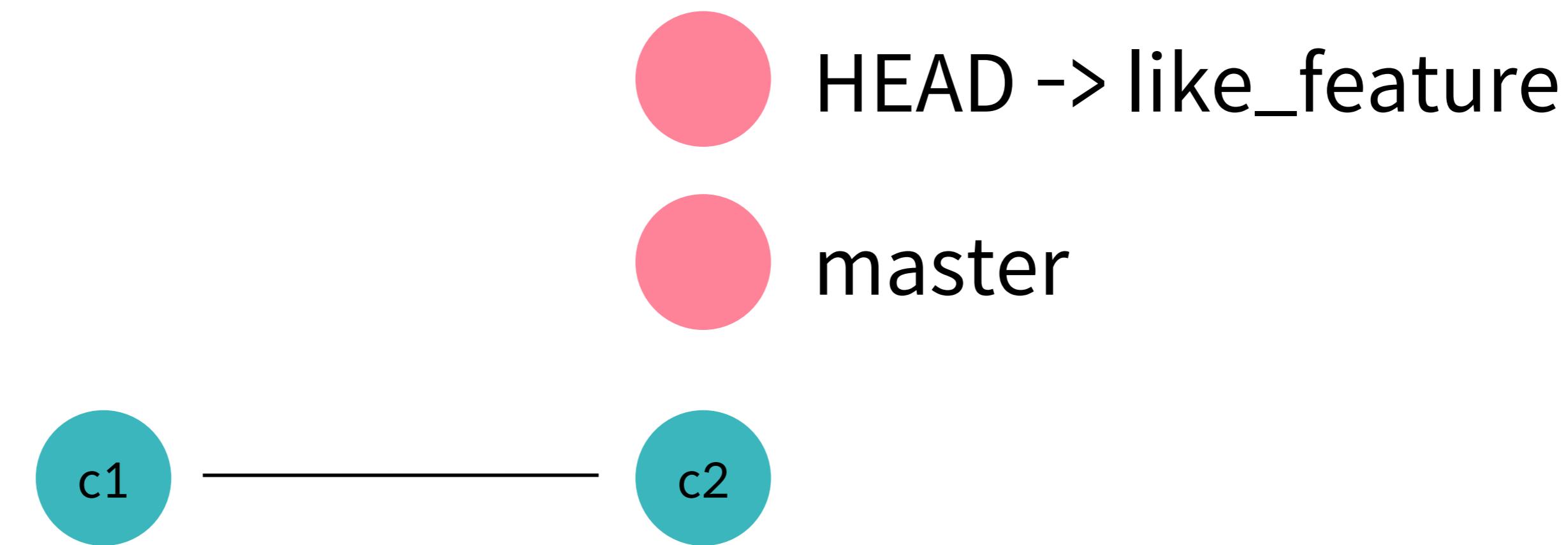
fast - forward

like_feature Branch의 working directory에서
새로운 정보를 넣어 commit해보겠습니다.

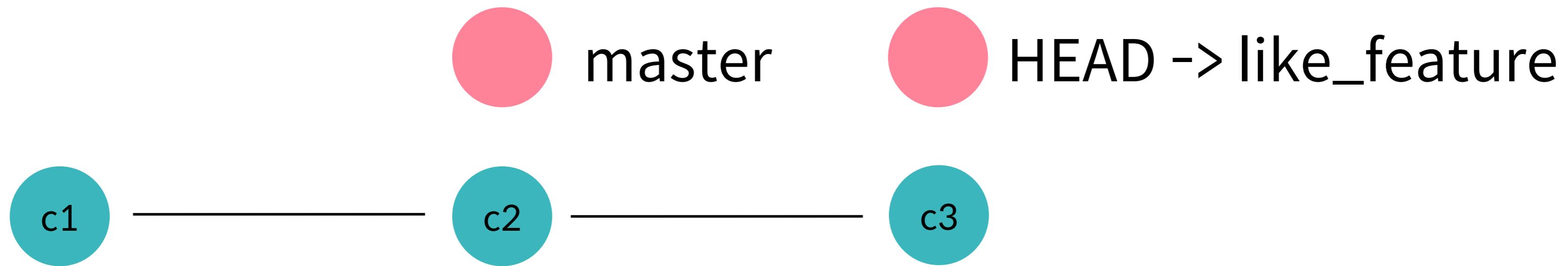
먼저 like_feature로 위치를 이동시켜볼까요?

```
git checkout like_feature
```

fast - forward



fast - forward



Git Merge

Git Merge

like_feature

Branch 에서의 작업을 끝마치고,

master

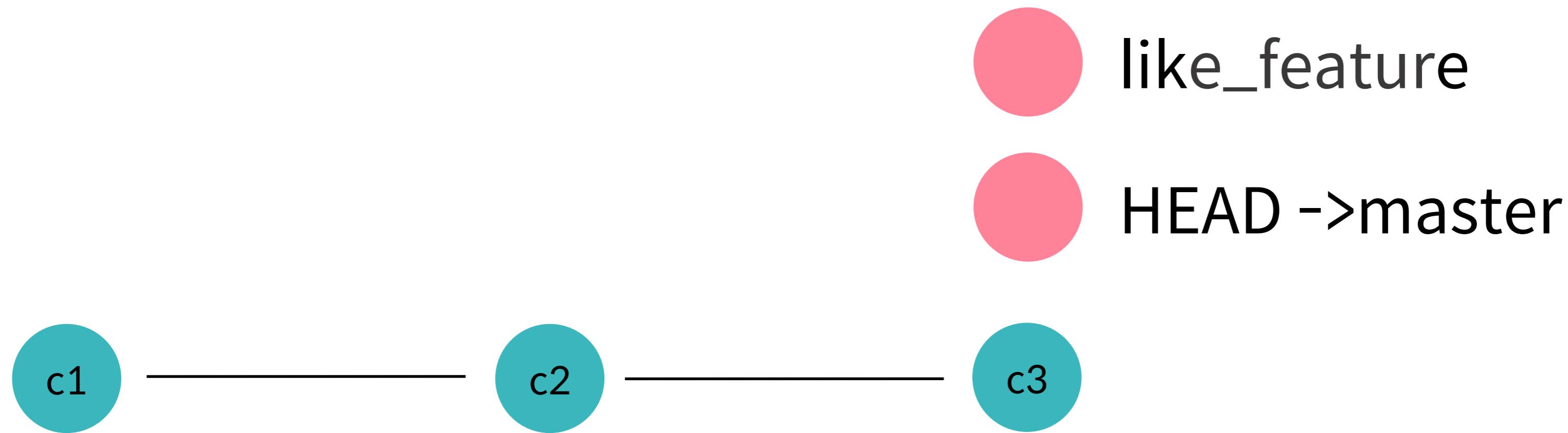
Branch 로 통합합니다

Git Merge

먼저 master Branch로 이동 하여
like_feature Branch를 병합합니다

```
$ git checkout master
Switched to branch 'master'
$ git merge like_feature
Updating d78ade4..a63hec2
Fast-forward
  comment.js | 3 +-+
  1 file changed, 2 insertions(+), 1 deletion(-)
```

fast - forward



fast - forward

`like_feature` Branch의 내용이 `master` Branch에서 업데이트 된 내용이기 때문에 **곧바로** merge가 되는 것을 확인 할 수 있습니다

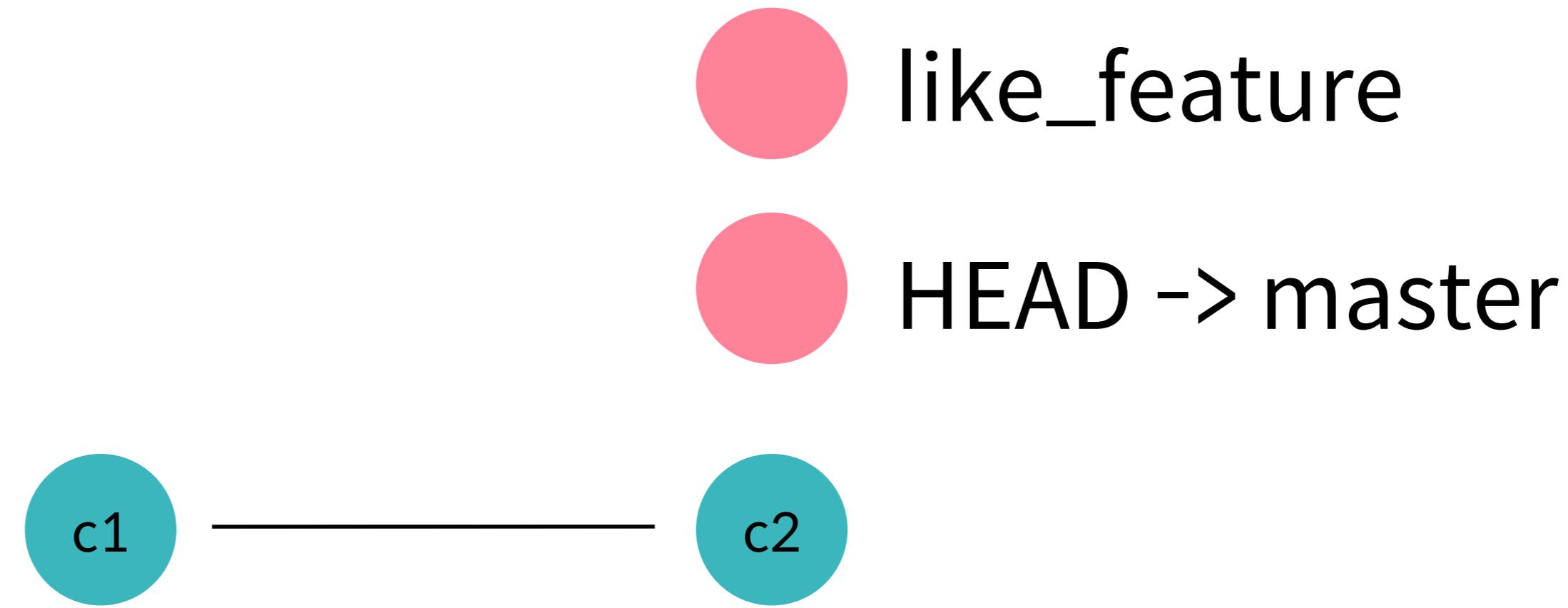
이렇게 merge가 이루어지는 것을 **fast-forward** 라고 부릅니다

갈라지는 branch

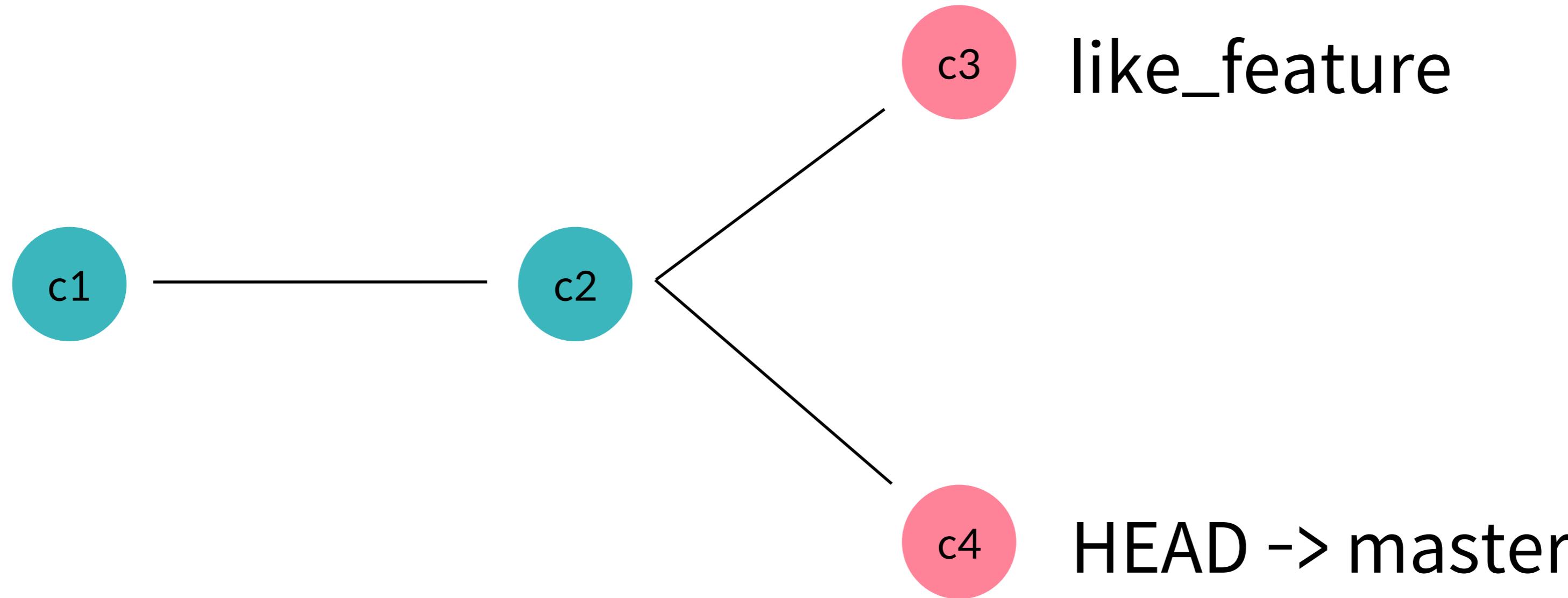
이번에는 각각의 Branch의 working directory에서
같은 파일의 내용을 다르게 수정해보겠습니다

명심하세요! 각각의 Branch는 다른 Branch의
영향을 받지 않기 때문에, 여러 작업을 동시에 진행 할 수 있습니다

갈라지는 branch



갈라지는 branch



갈라지는 branch

`git log --graph --all` 을 사용하면

commit graph를 확인 할 수 있습니다

추가로 옵션을 주어서 더 깔끔하게 볼 수도 있겠죠?

```
$ git log --pretty=oneline --graph --all
* c3360a...(HEAD->master) add master
| *782d90...(like_feature) add like_feature
| /
* 1c7881... init git
```

갈라지는 branch

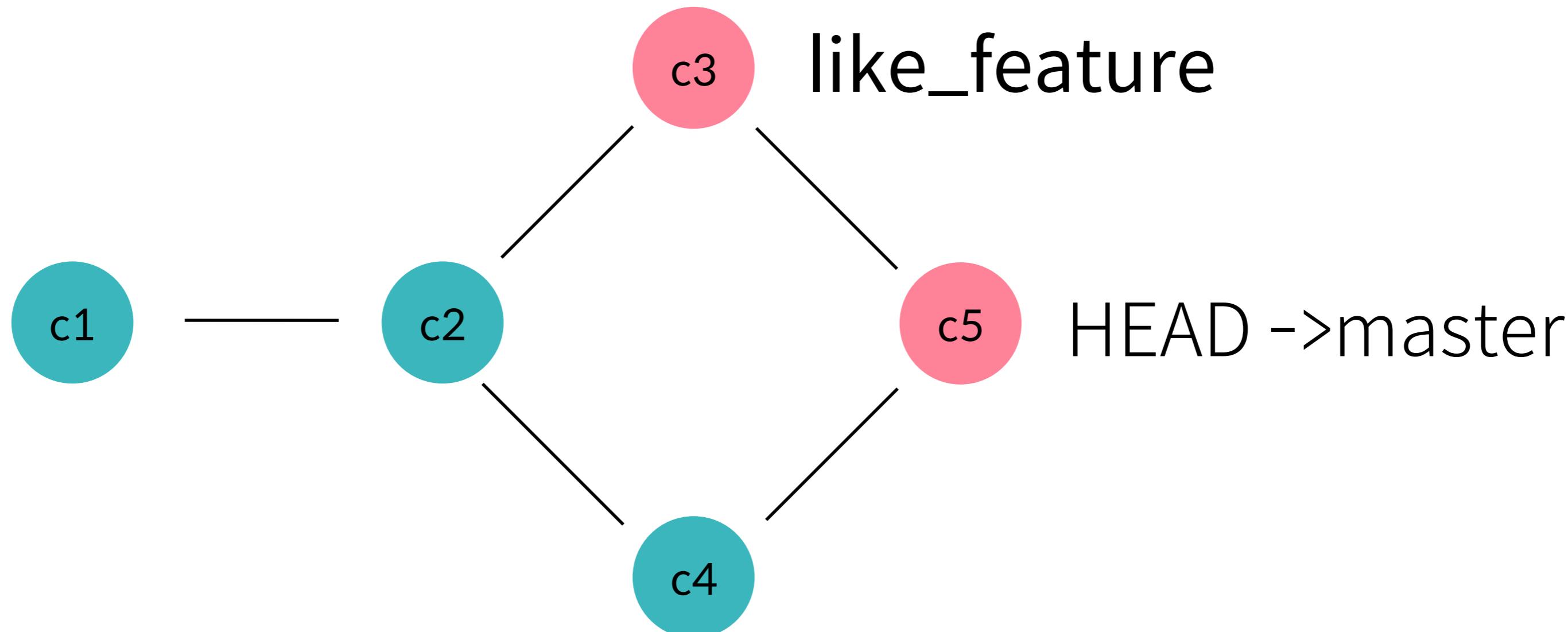
`git checkout master` 을 이용하여 master로 checkout한 후

`git merge like_feature` 로 merge 해보겠습니다

갈라지는 branch

```
$ git checkout master
switched to branch 'master'
$ git merge like_feature
[main 2019-07-31T08:58:15.648Z] update#setState idle
Merge made by the 'recursive' strategy.
  checkout.txt | 2 +-  
1 file changed, 1 insertion(+), 1 deletion(-)
```

갈라지는 branch



Git Branch 삭제

아래의 명령어는 Merge된 Branch를 볼 수 있습니다

```
$ git branch --merged  
like_feature  
* master
```

Git Branch 삭제

사용을 마친 branch는 `git branch -d <branch name>` 을
이용하여 삭제할 수 있습니다

```
$ git branch -d like_feature
Deleted branch like_feature (was 782d900).

$ git log --graph --pretty=oneline --all
*   3bf1a8... merging(HEAD -> master)
|\ \
| * 782d90... add like_feature
* | c3360a... add master
| /
* 1c7881 init git
```

Git Merge conflict

Merge conflict

Merge한 두 Branch에서
같은 파일을 변경했을 때 충돌이 발생합니다

Merge conflict

```
...
$(".comment-good").text($(".comment-good").val()-1);
...
```

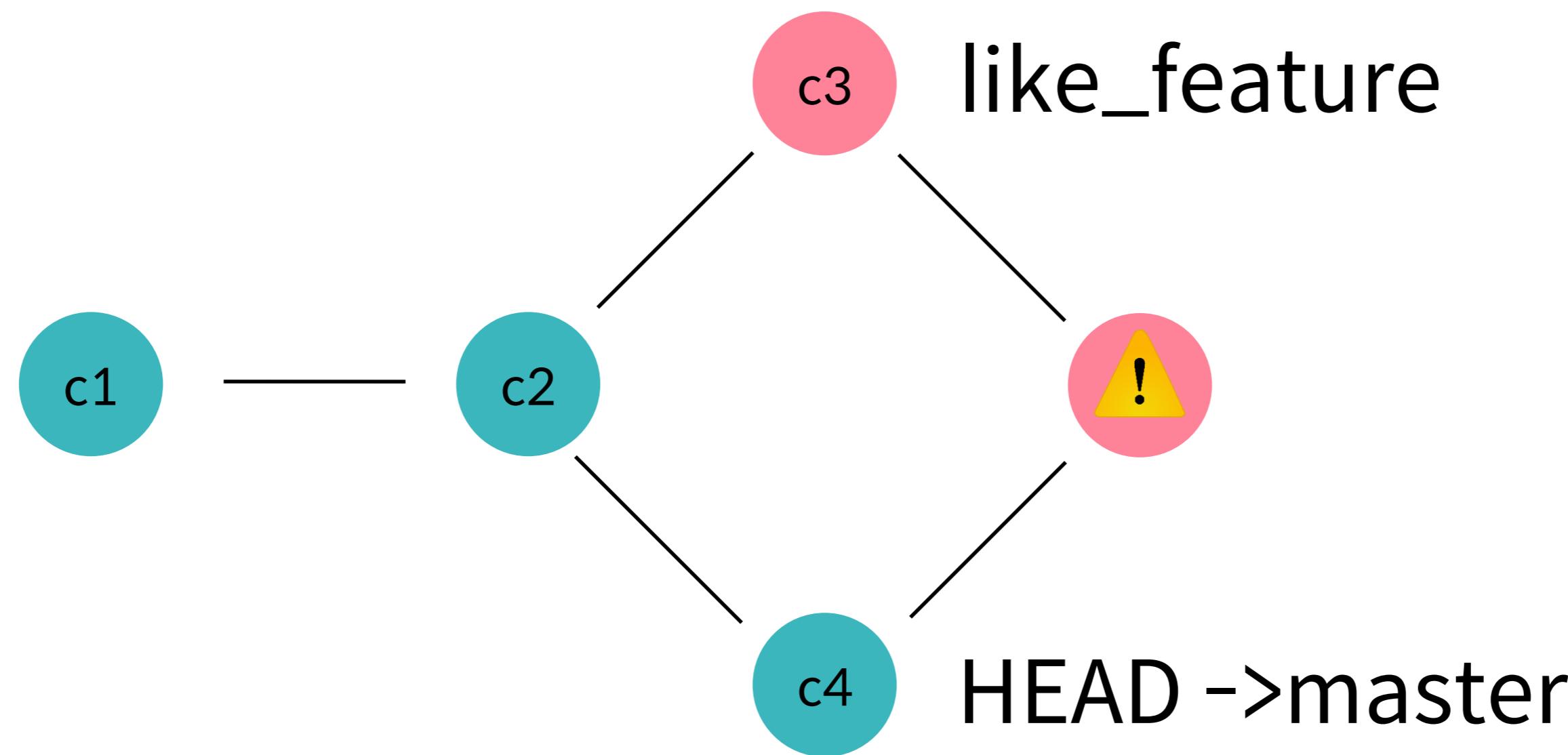
```
...
$(".comment-good").text($(".comment-good").val()-1 < 0 ? 0 :
$(".comment-good").val()-1);
...
```

Merge conflict

git merge like_features 명령을 수행했을 때
아래와 같이 충돌이 발생했습니다.

```
$ git merge like_feature
Auto-merging comment.js
CONFLICT (content): Merge conflict in comment.js
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflict



Merge conflict

git status 명령어로

어느 파일에서 충돌이 발생했는지 확인합니다.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      comment.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Git Merge 충돌 해결

충돌이 일어난 `comment.js` 파일을 열어봅니다.

```
<<<<<< HEAD
$( ".comment-good" ).text( $( ".comment-good" ).val() - 1 );
=====
$( ".comment-good" ).text( $( ".comment-good" ).val() - 1 < 0 ? 0 :
$( ".comment-good" ).val() - 1 );
>>>>> like_feature
```

Git Merge 충돌 해결

수정 완료 후,

'<<<<<<', '====', '>>>>>'가 포함된 행을 삭제해줍니다.

```
$(".comment-good").text($(".comment-good").val()-1 < 0 ? 0 :  
$(".comment-good").val()-1);
```

Git Merge 충돌 해결

수정 완료 후 `git add`, `git commit` 과정을 거쳐
다시 Merge 해줍니다.

```
$(".comment-good").text($(".comment-good").val()-1 < 0 ? 0 :  
$(".comment-good").val()-1);
```

```
$ git merge like_feature  
[like_feature a63hec2] Merge branch 'master' into like_feature
```

Git Merge 충돌 방지

master Branch의 변화를

지속적으로 가져와서

충돌이 발생하는 부분을 제거

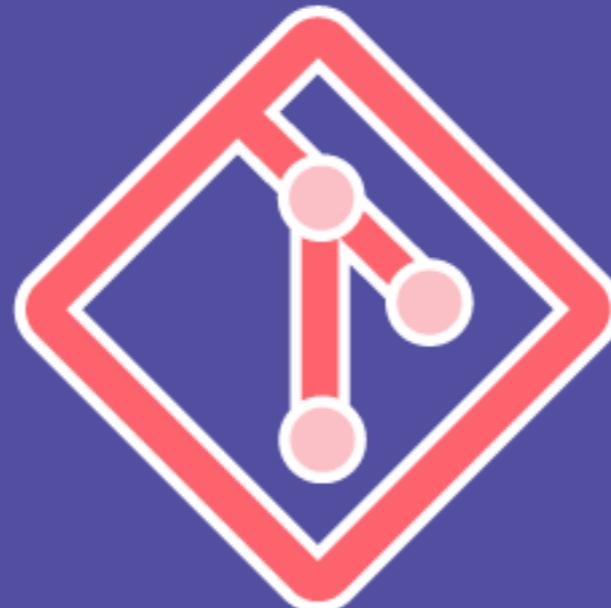


/* elice */

contact@elice.io

Git을 사용한 버전 관리

Git 원격 저장소



/* elice */

수강 목표

로컬 저장소를 원격 저장소와 연결할 수 있습니다.

원격 저장소에서 작업 내용을 받아올 수 있습니다.

원격 저장소에 작업 내용을 반영할 수 있습니다.

목차

1. 원격 저장소 받아오기
2. 원격 저장소 동기화
3. Origin이란?

원격 저장소 받아오기

원격 저장소?

인터넷이나 네트워크
어딘가에 있는 저장소

원격 저장소?

The screenshot shows a GitHub repository page for 'TheAlgorithms / Python'. The repository title is 'TheAlgorithms / Python'. It has 2,879 stars, 52,706 forks, and 15,473 issues. The 'Code' tab is selected. Below the tabs, there are several filters: 'python', 'algorithm', 'algorithms-implemented', 'algorithm-competitions', 'algorithms', 'sorts', 'searches', 'sorting-algorithms', 'education', 'learn', 'practice', 'community-driven', and 'interview'. A search bar contains the text 'Topic: community-driven'. Key statistics shown are 1,123 commits, 10 branches, 0 releases, 278 contributors, and an MIT license. The commit history lists recent changes:

- cclauss Travis CI: Run each failing pytest in allow_failures mode (#1087) ... Latest commit 561a414 7 hours ago
- .github Update FUNDING.yml last month
- arithmetic_analysis print() is a function just like every other function (#1101) 17 hours ago
- backtracking Add combinations (#1015) 24 days ago
- boolean_algebra Boolean algebra pytests (#1097) 2 days ago
- ciphers print() is a function just like every other function (#1101) 17 hours ago

Git 원격 저장소 받아오기

Git clone

기존의 git repository를 복사

원격 저장소 url로 받아오기

Gitlab 또는 Github에서 원하는 프로젝트에서
clone 버튼을 누릅니다.

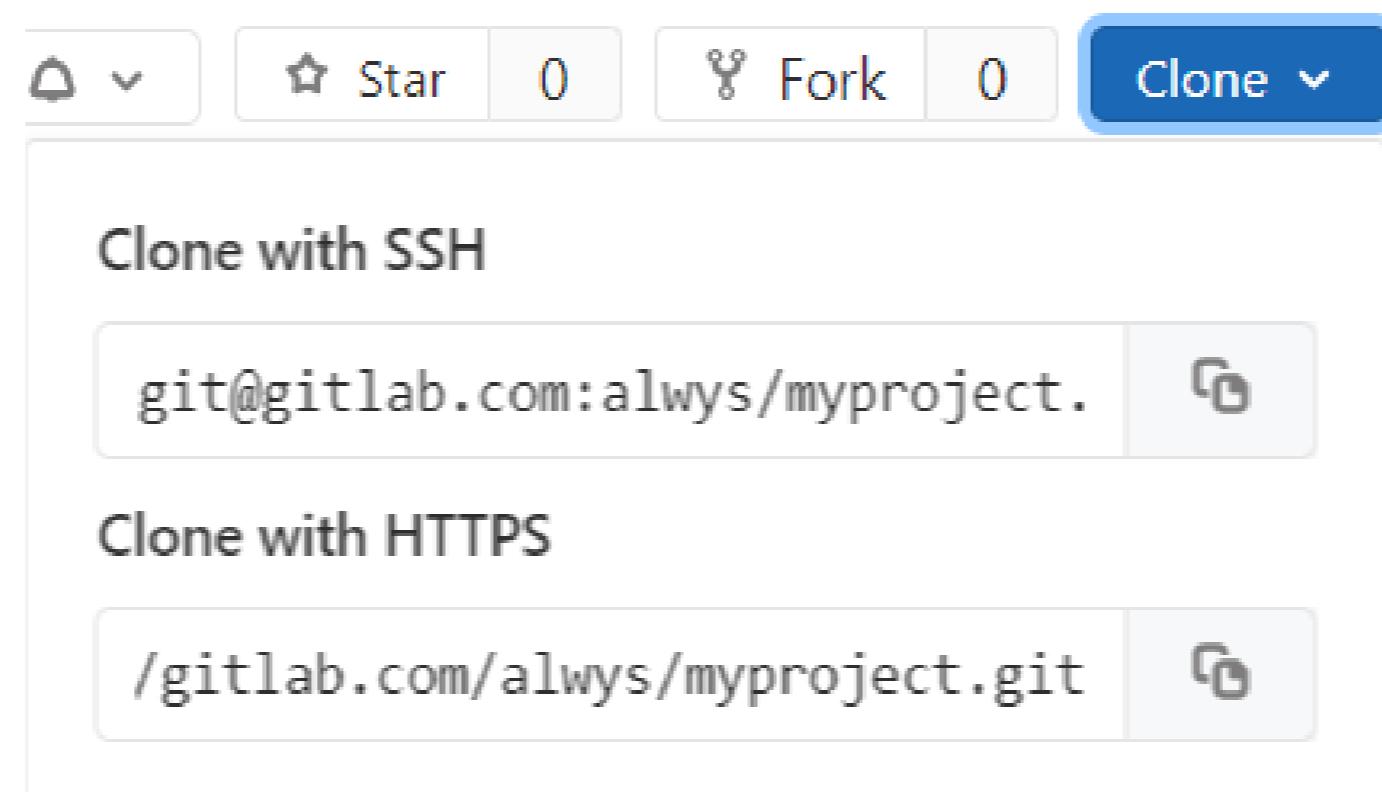
hanium > Myproject > Details

The screenshot shows a project named 'Myproject' with a lock icon, Project ID 13265120. It has 1 Commit, 1 Branch, 0 Tags, and 133 KB Files. A prominent 'Clone' button is visible in the top right. Below the project name, there's an 'Auto DevOps' section with a cloud icon and a 'Enable in settings' button. At the bottom, there's a navigation bar with 'master', 'myproject / +', 'History', 'Find file', 'Web IDE', and other icons.

원격 저장소 url로 받아오기

아래와 같이 2개의 옵션을 볼 수 있습니다.

여기서 **Clone with HTTPS** 옵션으로
Clone을 하게 됩니다.



원격 저장소 url로 받아오기

Git clone 뒤에 clone 버튼으로 확인한
원격저장소의 주소를 넣어줍니다.

```
$ git clone https://gitlab.com/always/myproject.git
```

원격 저장소 추가 (1)

원격 저장소는 아래의 명령어로 연결할 수 있어요.

```
$ git remote add origin https://gitlab.com/group/project
```

원격 저장소 추가 (2)

저장소 주소는 다음과 같이 구성됩니다.

```
$ git remote add origin https://gitlab.com/group/project
```

웹 호스트 서비스

그룹 명

프로젝트 명



원격 저장소 추가 (3)

연결된 원격 저장소를 확인해보세요.

```
$ git remote  
origin
```

원격 저장소 살펴보기

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/.../git_test.git
  Push  URL: https://github.com/.../git_test.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

원격 저장소 이름 변경

원격 저장소 단축 이름을
origin에서 git_test으로 변경

```
$ git remote rename origin git_test
```

원격 저장소 삭제

주소가 변경되었거나, 필요 없어진 저장소는
아래의 명령어로 삭제할 수 있어요.

```
$ git remote rm git_test
```

원격 저장소 동기화

저장소 강신

Pull

원격 저장소에서 데이터 가져오기 + 병합(Merge)

Fetch

원격 저장소에서 데이터 가져오기

저장소 갱신 - Pull

원격 저장소에서 데이터를 가져와
로컬 데이터와 병합합니다.

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 ...
Unpacking objects: 100% (3/3), done.
From https://github.com/.../git_test
  0daf5d2..c61952d  master -> origin/master
Updating 0daf5d2..c61952d
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

```
$ git log --all
commit f7b775d (HEAD->master, origin/master)
commit 725fc8b
commit fad4a9a
commit 1418700 init repository
```

저장소 갱신 - Fetch

원격 저장소에서 데이터를 가져오지만,
병합하지는 않습니다.

진행중인 작업을 마무리하고 병합해주어야 합니다.

```
$ git fetch  
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 ...  
Unpacking objects: 100% (3/3), done.  
From https://github.com/heeguchi/git_test  
  c61952d..932dd32 master -> origin/master
```

```
$ git log --all  
commit f7b775d (HEAD->master)  
commit 725fc8b (origin/master)  
commit fad4a9a  
commit 1418700 init repository
```

저장소 갱신 - Fetch

git log

명령어로 변경된 파일을 확인하고
Merge 해줍니다.

```
$ git log origin/master  
commit 8a9ed03b4d750ca0ef58e795af237cb32aa461b7  
(origin/master, origin/HEAD)  
Author: Elice <elice@elice.noreply.github.com>  
Date:   Mon Aug 12 11:15:58 2019 +0900
```

Update README.md

...

저장소 갱신 - Fetch

git log

명령어로 변경된 파일을 확인하고
Merge 해줍니다.

```
$ git merge origin/master
Updating 932dd32..8a9ed03
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

저장소 발행

로컬 저장소에서 작업한 내용을 원격 저장소에 반영합니다.

다른 사람이 먼저 **Push한 상태에서는 Push할 수 없어요.**

다른 사람이 작업한 것을 **Merge부터** 해주세요.

```
$ git push origin master
```

요약

1. `git remote add origin` (또는 다른 원격저장소 이름)으로 로컬저장소와 **연결합니다**.
2. `git fetch` 또는 `git pull` 을 이용하여 원격저장소의 내용을 동기화합니다.
3. `fetch`를 실행한 경우 `git merge origin/master` 로 병합을 완료해줍니다.
4. `git push origin master` 를 이용하여 변경된 사항을 원격 저장소에 **전달해줍니다**.

Origin이란?

origin/master

내 컴퓨터에 저장되어 있는 저장소와
원격저장소를 연결하기 위해서 아래와 같은 명령을 사용했습니다.

```
git remote add origin https://github.com/group/project
```

이는 원격저장소의 단축이름을 **origin**으로 지정한다는 의미입니다.

origin/master

origin 이 아닌 다른 이름으로 원격 저장소의 이름을
지정해 줄 수도 있습니다.

```
git remote add myproject https://github.com/group/project
```

origin/master

기본적으로 만들어진 원격저장소의 이름은 origin이 default값입니다.
때문에 clone으로 복사해온 저장소의 이름은
origin으로 통일되게 됩니다.

-v 옵션을 사용하면

지정한 저장소의 이름과 주소를 함께 볼 수 있습니다.

```
git remote -v
```

origin/master

```
git remote -v
```

```
$ git remote -v
origin https://gitlab.com/group/project (fetch)
origin https://gitlab.com/group/project (push)
```

CREDIT

코스 매니저
지동준

강사
김경민

콘텐츠 제작에 기여하신 분
장준호, 지동준

영상 제작에 기여하신 분
김경민, 박수광

검수와 자문에 도움주신 분
김경민, 문승현, 장석준

/* elice */

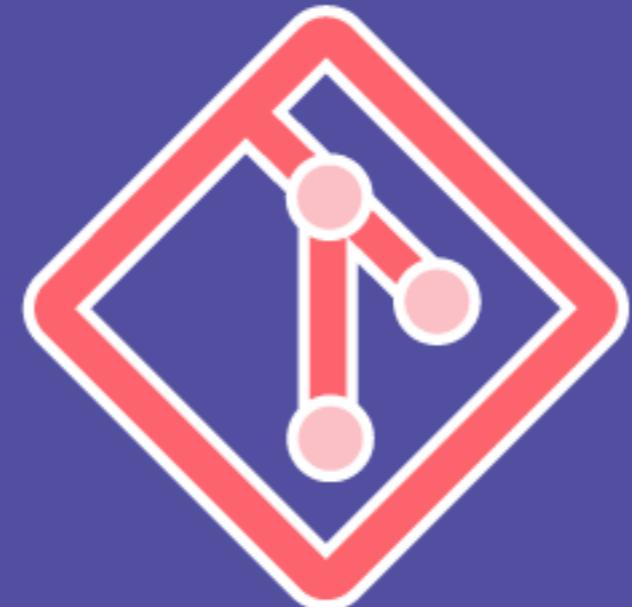


/* elice */

contact@elice.io

Git을 사용한 버전 관리

Git HEAD pointer



/* elice */

HEAD

파일들을 staging시키고 commit하여 git repository에
저장하는 방법까지 알아보았습니다.

그렇다면 git은 어떤 방식으로 commit된 내용으로
버전을 효율적으로 관리 할 수 있을까요?

파일의 영역 라이프 사이클

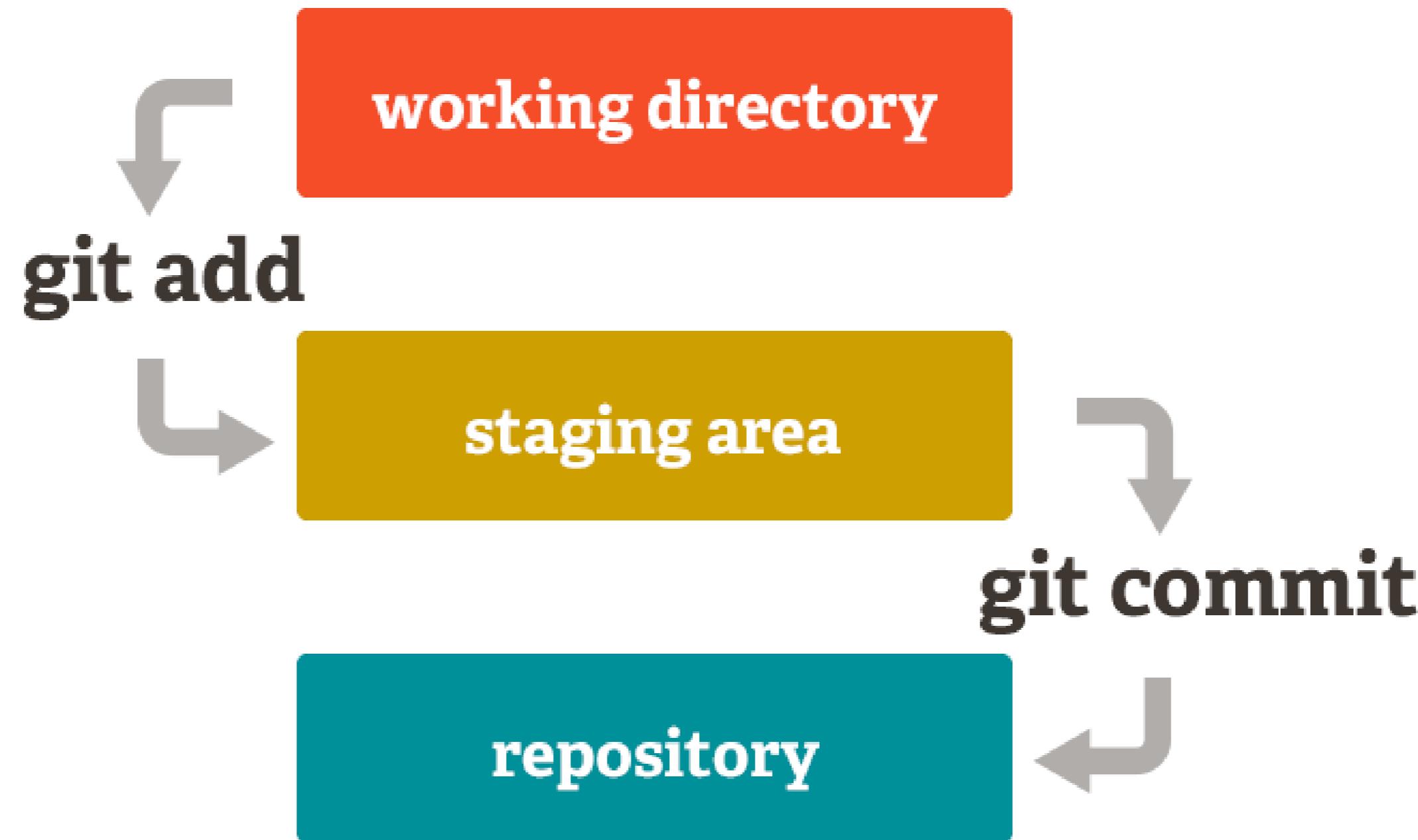


그림 git-scm.com

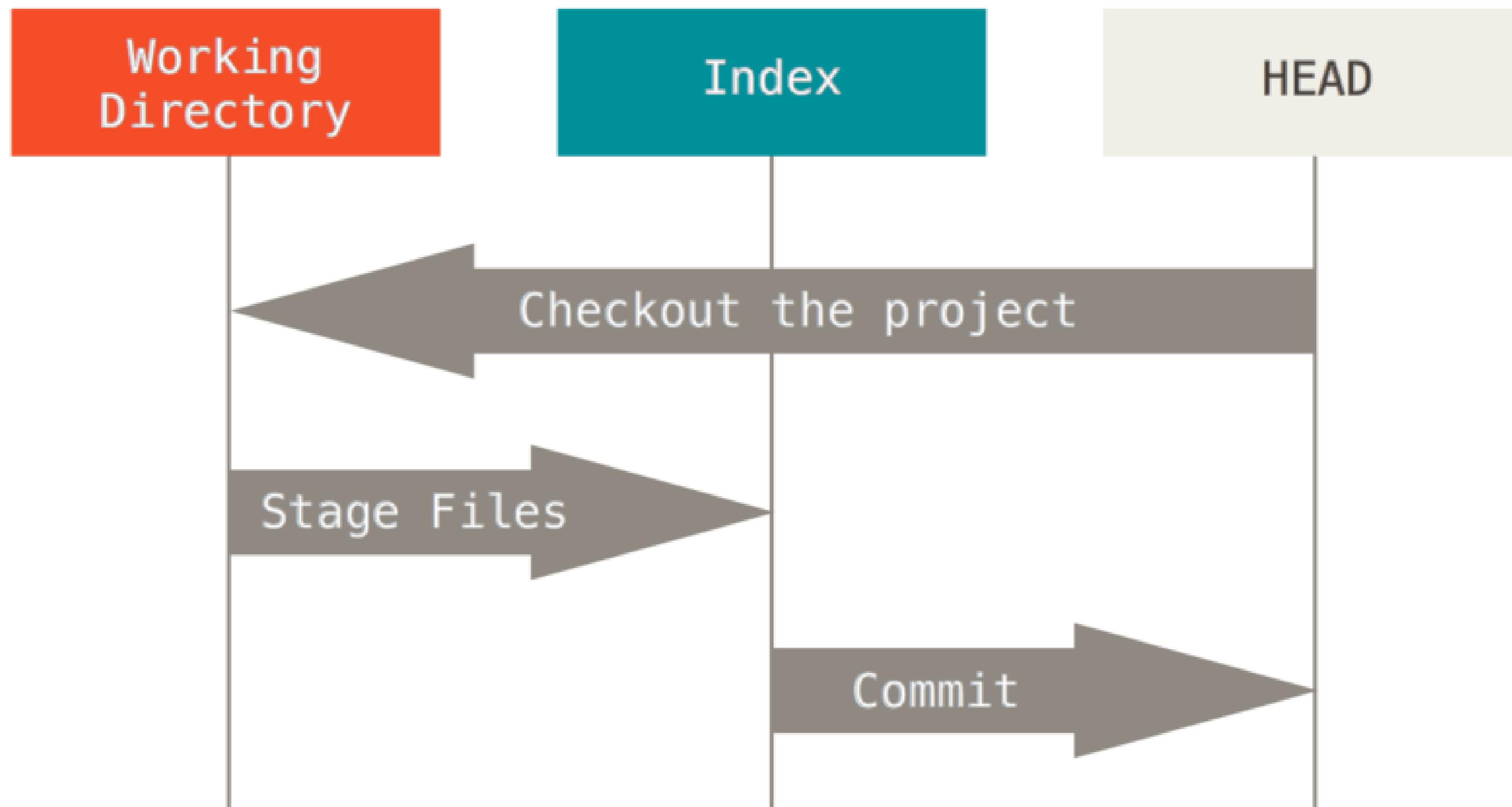
git 영역의 구성

HEAD

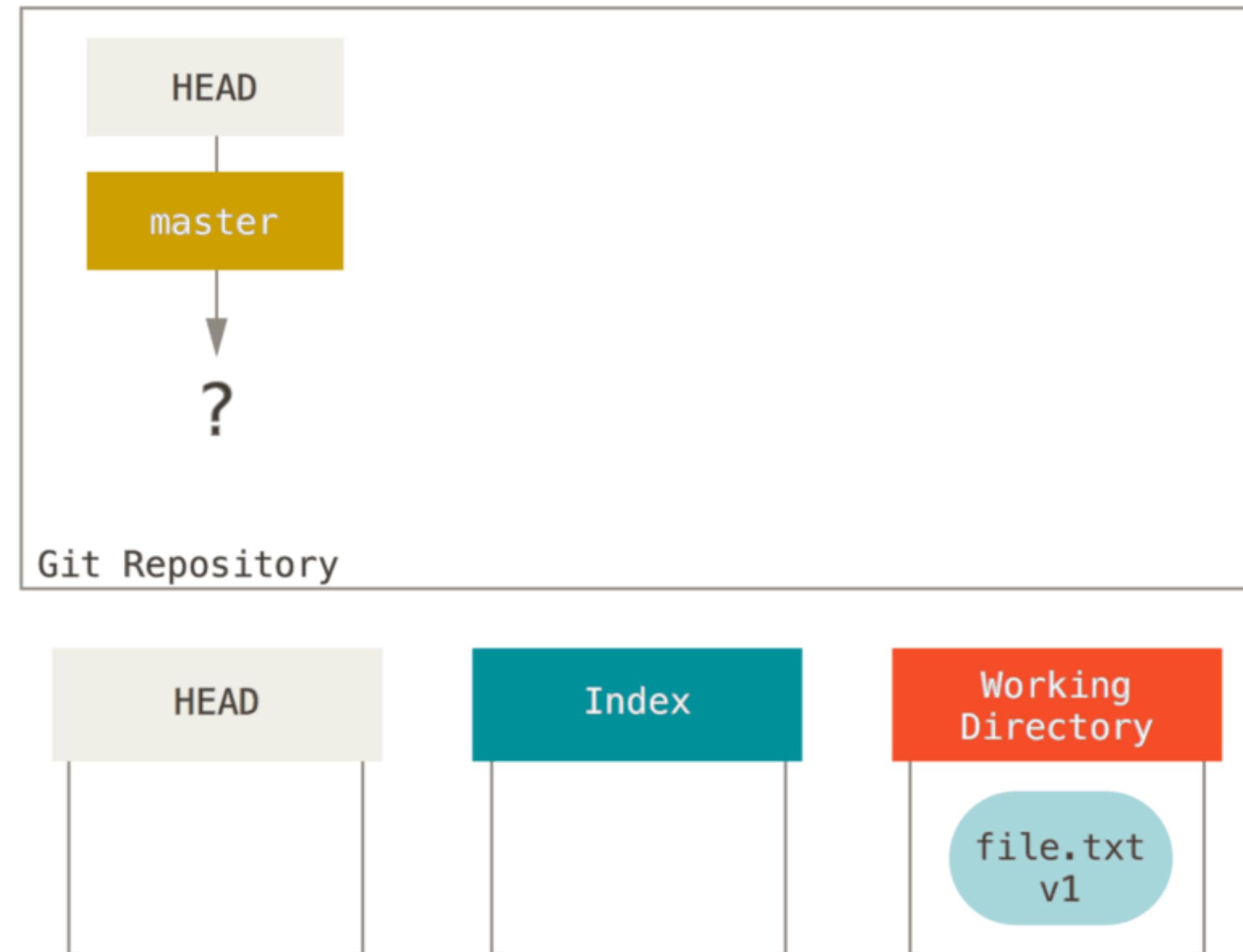
Index

working directory

파일의 워크플로



파일의 워크플로



Index

git add 로 Staging 되어 있는 데이터들

HEAD

마지막 커밋을 가리키는 snapshot



snapshot

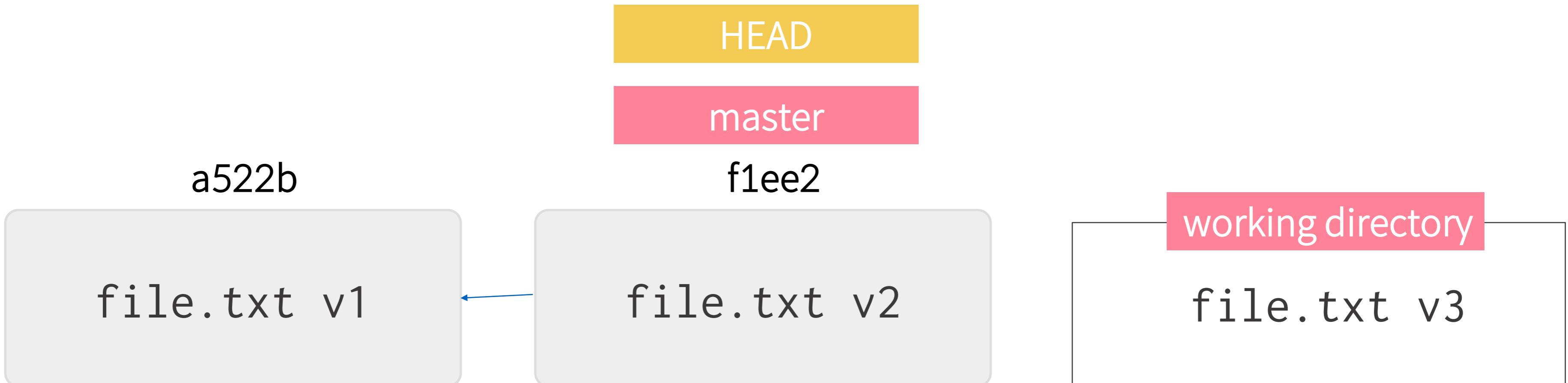
staging 되어진 데이터들(Index)을
commit 명령으로 영구히 저장한 것을
snapshot 이라 부릅니다.

HEAD 역시 snapshot이며 후에 배울 branch에
서 현재 우리가 위치해 있는 branch를 가리키
는 포인터의 역할을 같이 수행하게 됩니다.

HEAD를 이용한 버전 관리

`git reset --<option> HEAD~` 을 이용하여

HEAD를 전의 snapshot으로 이동시킬 수 있습니다.



--soft(default)

git reset --soft HEAD~ 을 이용하여
예전 버전으로 돌아가 commit을 수정할 수 있습니다.



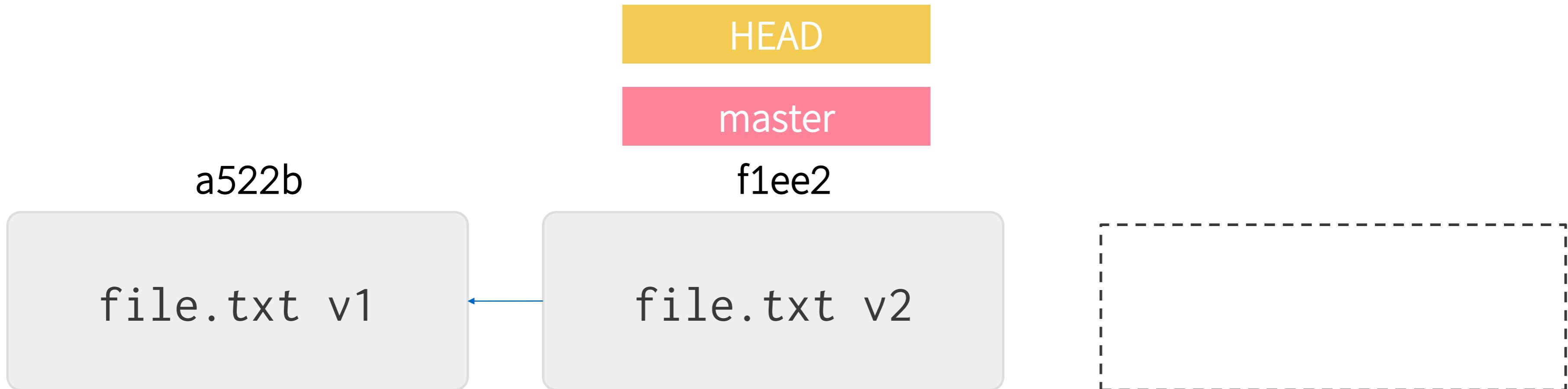
--soft(default)

```
$ git log --pretty=oneline  
eba263 (HEAD->master) add change  
d3c3d init commit
```

```
$ git reset --soft HEAD~  
$ git log --pretty=oneline  
d3c3d (HEAD->master) init commit  
$ git commit -m "modified message"  
$ git log --pretty=oneline  
39ae2 (HEAD->master) modifies me..  
d3c3d init commit
```

--hard

git reset --hard HEAD~ 을 이용하면
working directory에 존재하는 파일도 사라집니다.





/* elice */

contact@elice.io

Git을 사용한 버전 관리

Git rebase

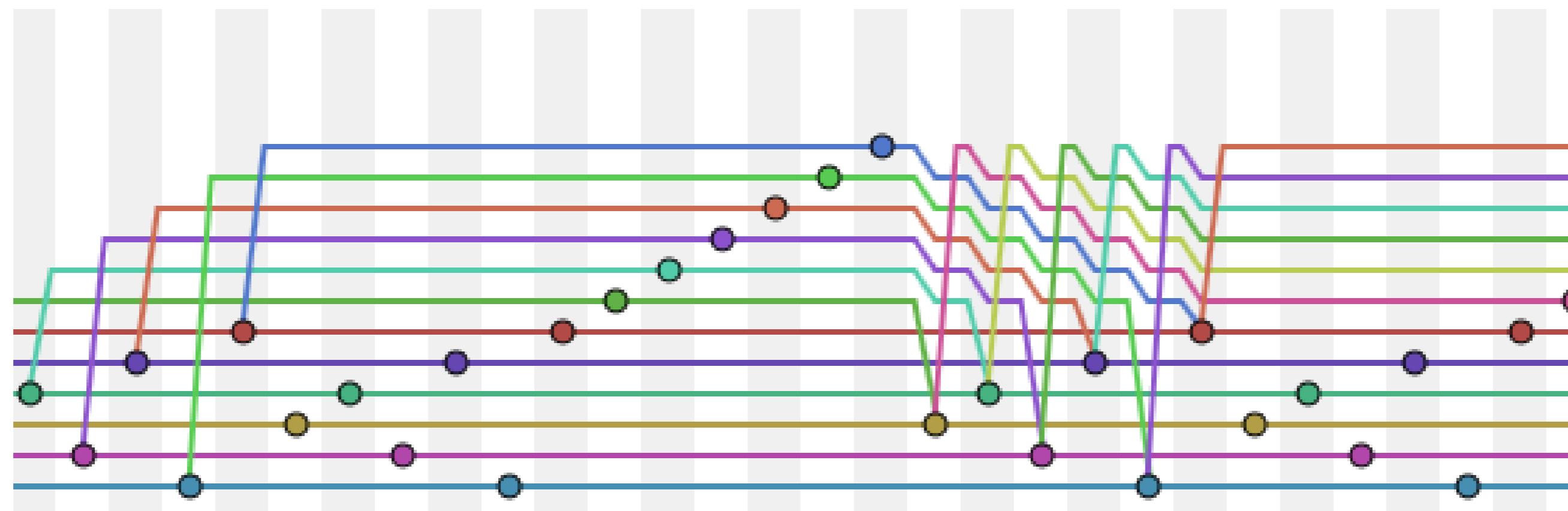


/* elice */

rebase

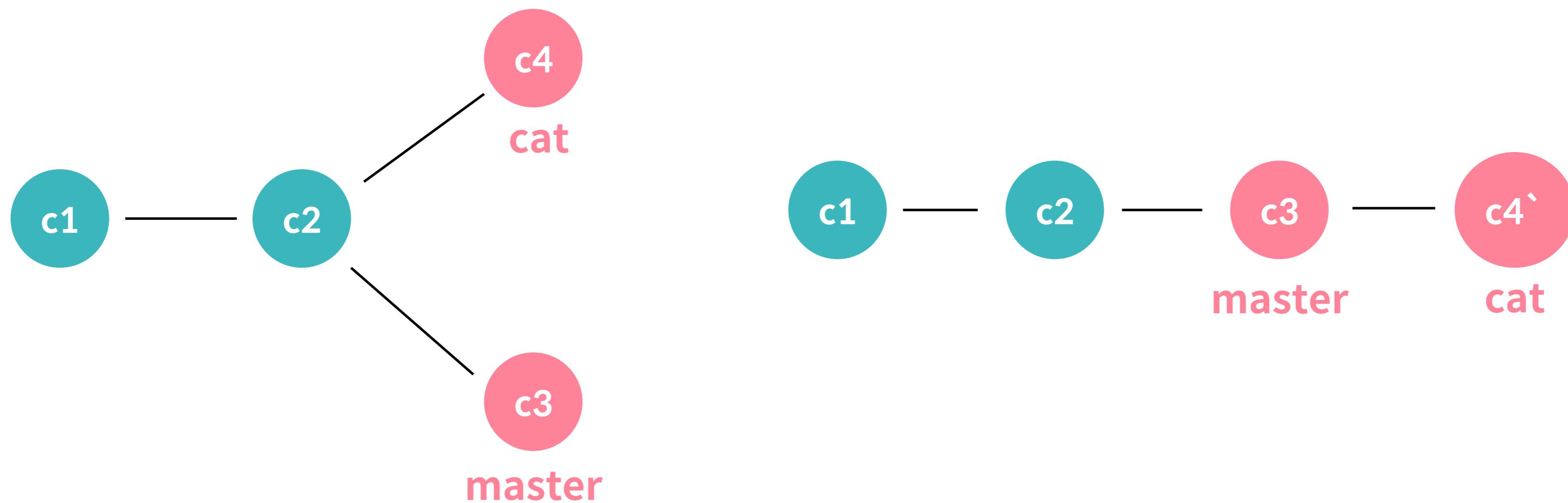
rebase

rebase는 브랜치가 너무 많아져서
history 정리가 필요한 상황에 사용합니다.



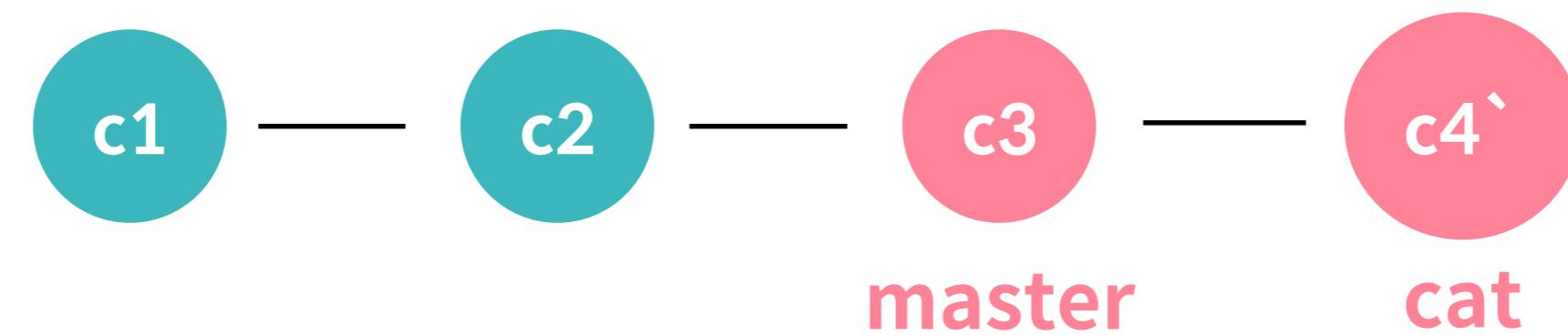
rebase

그래프를 선형으로 만듭니다.



rebase

rebase는 공통된 부모까지의 commit을 가져와
원하는 브랜치 옆에 이어붙입니다.



```
git checkout cat
```

```
git rebase master
```

rebase conflict 해결하기

rebase conflict 해결

merge와 마찬가지로 rebase 과정 중에서도
충돌이 발생할 수 있습니다.

```
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: 2
Using index info to reconstruct a base tree...
M      client_file
Falling back to patching base and 3-way merge...
Auto-merging client_file
CONFLICT (content): Merge conflict in client_file
error: Failed to merge in the changes.
```

rebase conflict 해결

status로 파일의 상태를 확인하면 다음과 같습니다.

```
$ git status
rebase in progress; onto 1522887
You are currently rebasing branch 'cat' on '1522887'.
(fix conflicts and then run "git rebase --continue")
(use "git rebase --abort" to check out the original branch)
Unmerged paths:
(use "git restore --staged <file>..." to unstage)
(use "git add <file>..." to mark resolution)
    both modified: client_file
no changes added to commit (use "git add" and/or "git commit -a")
```

rebase conflict 해결

client_file에서 발생한 충돌을 해결하고
git add 명령을 수행해서 client_file을 staging 해줍니다.

```
$ git add client_file
$ git status
rebase in progress; onto 1522887
You are currently rebasing branch 'cat' on '1522887'.
(all conflicts fixed: run "git rebase --continue")
```

```
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified:   client_file
```

rebase conflict 해결

staging 되었다면 이를 `git rebase --continue` 를 이용해서
rebase를 마무리 해줍니다.

```
$ git rebase --continue
Applying: 2
$ git log --all --graph --oneline
* 83e40a4 (HEAD -> cat) cat commit
|
* 1522887 (master) master commit
|
* e44bc57 init repo
```

rebase conflict 해결

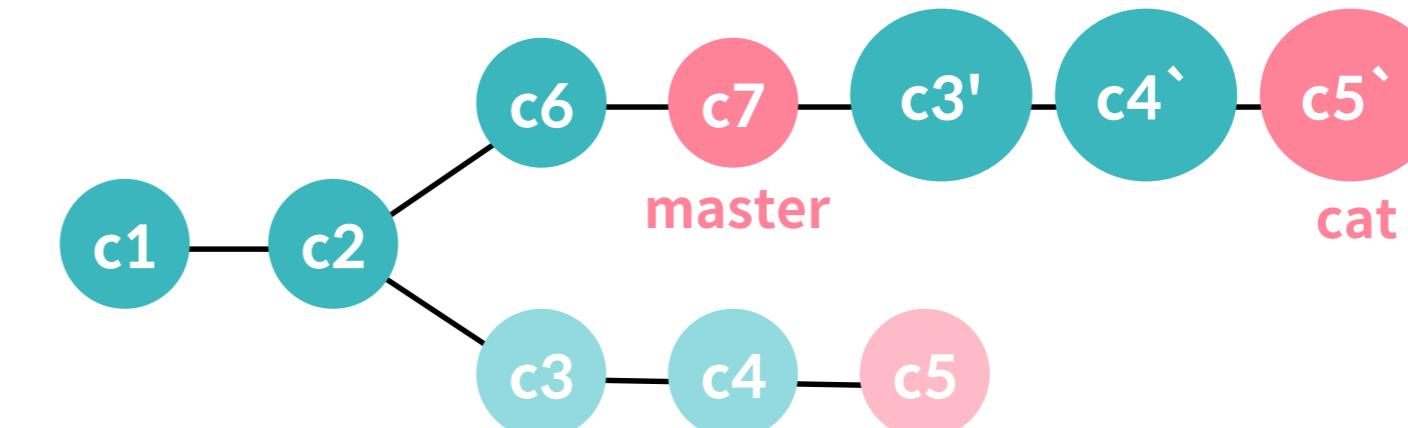
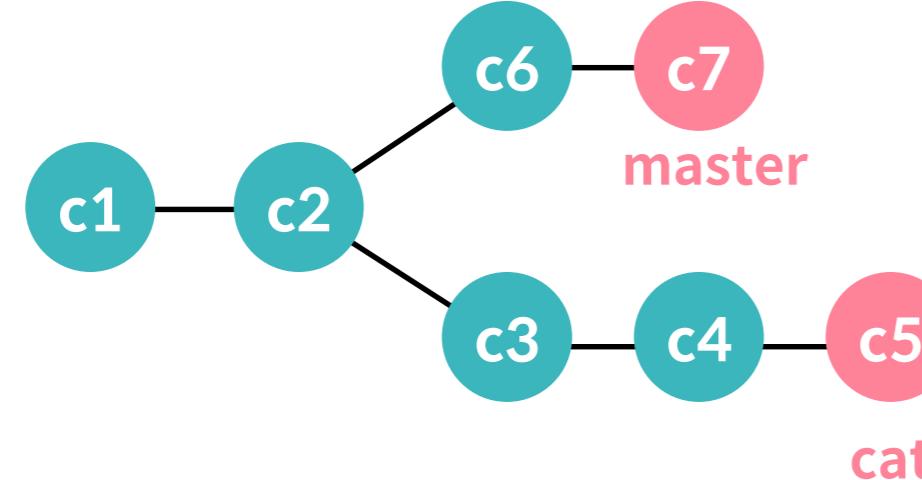
후에 master를 cat으로 merge 뿐만 아니라 rebase로도
fast-forward 시켜 줄 수 있습니다.

```
$ git checkout master
$ git rebase cat
First, rewinding head to replay your work on top of it...
Fast-forwarded master to cat.
$ git log --online
* 83e40a4 (HEAD -> master, cat) cat commit
* 1522887 master commit
* e44bc57 init repo
```

결론

git rebase master

위의 명령어를 요약하자면 현재 위치해 있는 브랜치의 commit부터 master의 공통된 부모 전까지의 commit 을 master 옆에 이어 붙인다는 의미입니다.





/* elice */

contact@elice.io

Git을 사용한 버전 관리

pull과 push의 자세한 이해



/* elice */

저장소 동기화 파헤치기

로컬 저장소와 원격저장소를 서로 동기화 시키기 위해서는 아래의 2개의 과정을 거쳐야 합니다.

1. remote 저장소의 이름을 지정해 줍니다.

```
git remote add origin https://github.com/group/project
```

2. 트래킹 브랜치를 설정해서 push 또는 pull을 합니다.

```
git pull origin master
```

```
git push --set-upstream origin master
```

```
git push origin master
```

그냥 git pull, git push를
하면 안되는 걸까요?

저장소 동기화 파헤치기

최초로 원격저장소를 pull 하게 되면
다음과 같은 메시지를 볼 수 있습니다.

```
$ git pull
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>
If you wish to set tracking information for this branch you can do so with:
    git branch --set-upstream-to=origin/<branch> master
```

저장소 동기화 파헤치기

최초로 원격저장소를 push하게 되면
다음과 같은 메시지를 볼 수 있습니다.

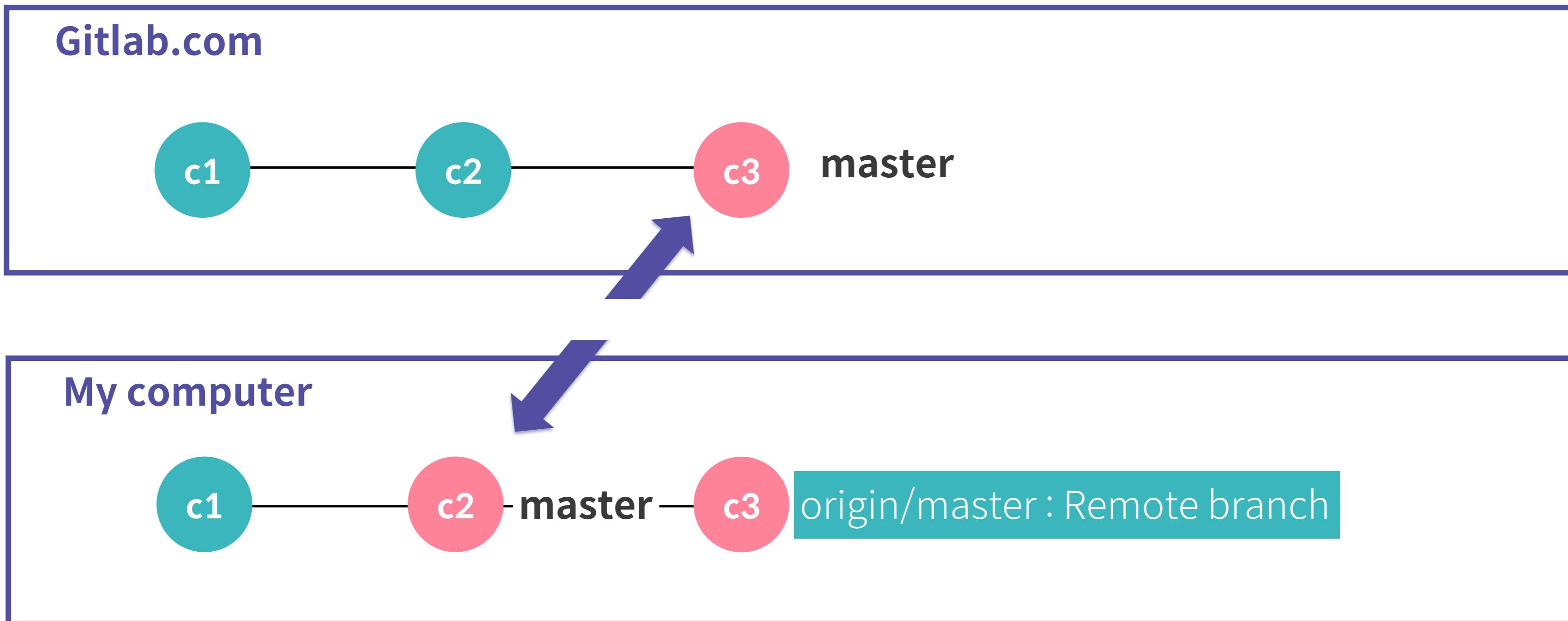
```
$ git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

git push --set-upstream origin master
```

--set-upstream

--set-upstream

앞에서 다룬 메시지는 원격저장소에 위치한 브랜치를 추적하기 위한
로컬 저장소의 브랜치가 정해지지 않았다는 의미입니다.



pull push 파헤치기

remote 등록

first라는 원격저장소를 등록해 보겠습니다.

```
git remote add first https://github.com/group/project1
```

```
$ git remote -v
first https://gitlab.com/group/project1 (fetch)
first https://gitlab.com/group/project1 (push)
```

pull 파헤치기

pull을 할 때의 명령어를 파헤쳐 볼까요?
pull은 원격저장소로 부터 데이터를 가져오며
병합하는 명령어입니다.

```
git pull origin master
```

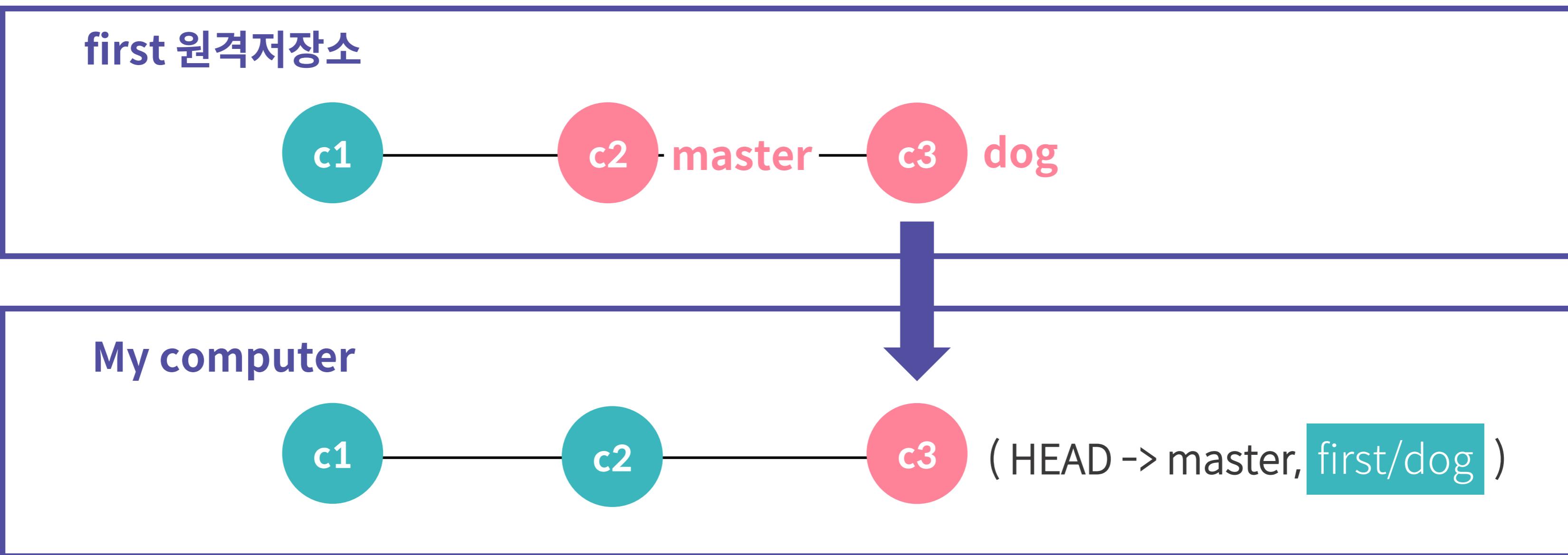
pull 파헤치기

pull 옆에 우리가 받아오고자 하는
원격 저장소의 이름과
원격 저장소에서 받아오고자 하는 브랜치의
이름을 적어주면 됩니다.

```
git pull first dog
```

pull 파헤치기

pull 을 했을 때 다음과 같이 트래킹 브랜치가 생성되며 원격저장소에서의 dog 브랜치가 위치한 위치를 보여줍니다.



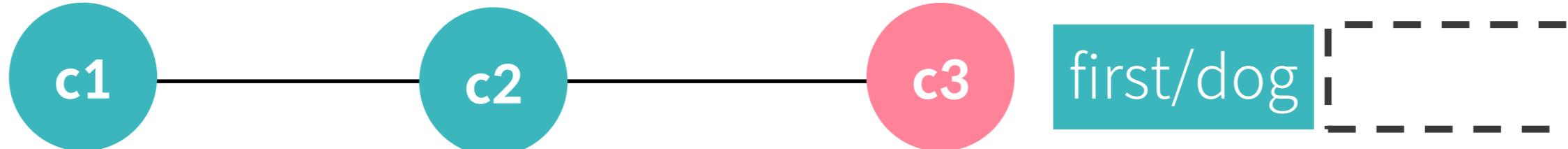
pull 파헤치기

트래킹 브랜치가 생성되었지만 로컬저장소에서
트래킹 브랜치와 연결된 브랜치가 존재하지 않습니다.

first 원격저장소



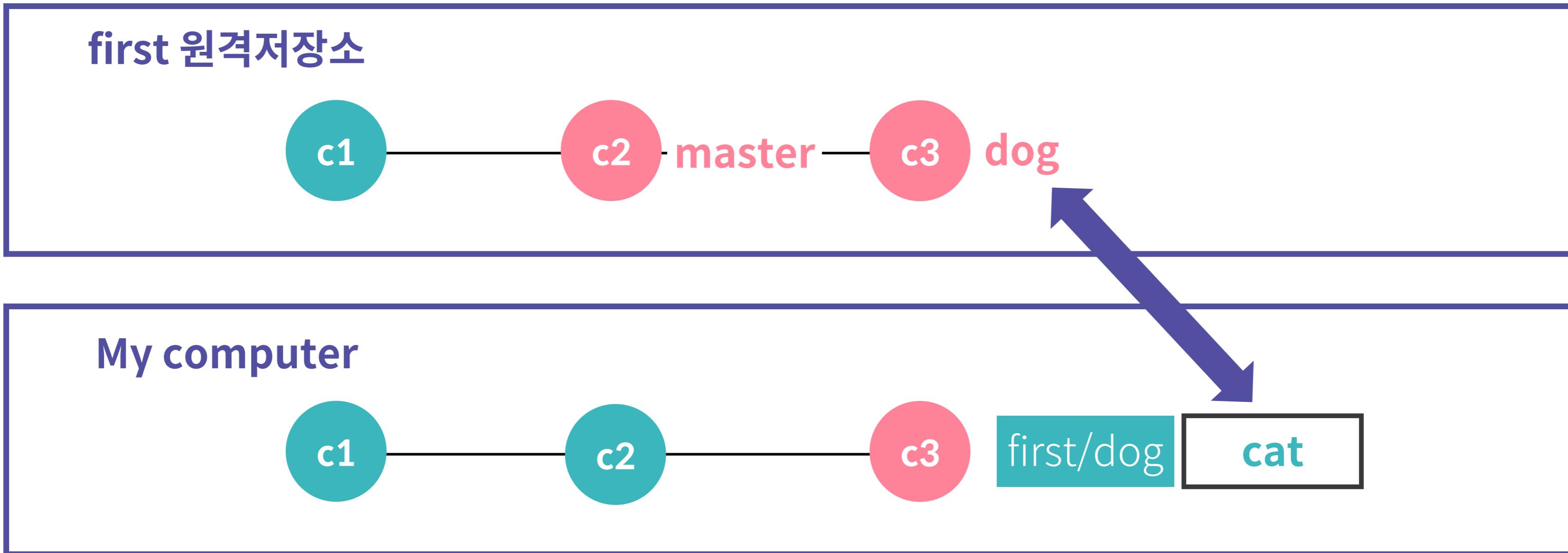
My computer



pull 파헤치기

cat이라는 브랜치를 만들고 아래의 명령으로 트래킹 브랜치와 연결시키겠습니다.

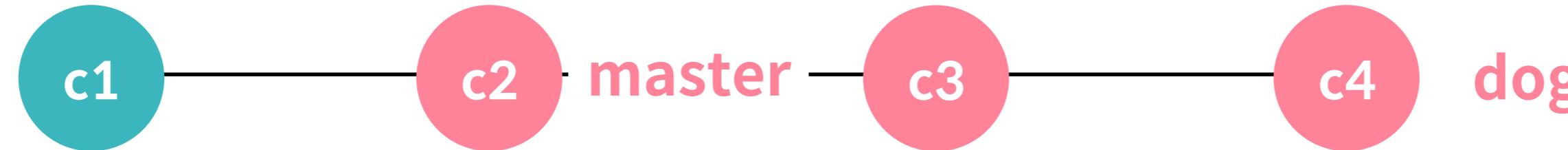
```
git branch --set-upstream-to=first/dog cat
```



pull 파헤치기

만약 원격저장소의 dog 브랜치에 새로운 commit이 올라오게 되더라도
cat 브랜치에서 git pull을 실행하면
트래킹 브랜치가 위치해 있는 곳까지 fast-forward가 이루어집니다.

first 원격저장소



My computer



push 파헤치기

이번에는 push를 할 때의 명령어를 파헤쳐 볼까요?
push는 원격저장소로 데이터를 전송하는 명령어입니다.

```
git push
```

push 파헤치기

`git clone` 명령어로 저장소를 복사해온 것이 아니면
처음 push 할 때 실제로는 다음과 같은 명령을 사용해야 합니다.

`git push origin master`

push 파헤치기

push 옆에 데이터를 전송하고자 하는 원격저장소와
push하기 희망하는 로컬저장소의 브랜치 이름을 적어줍니다.

```
git push second cat
```

remote 등록

새로운 second라는 원격저장소를 등록해 보겠습니다.

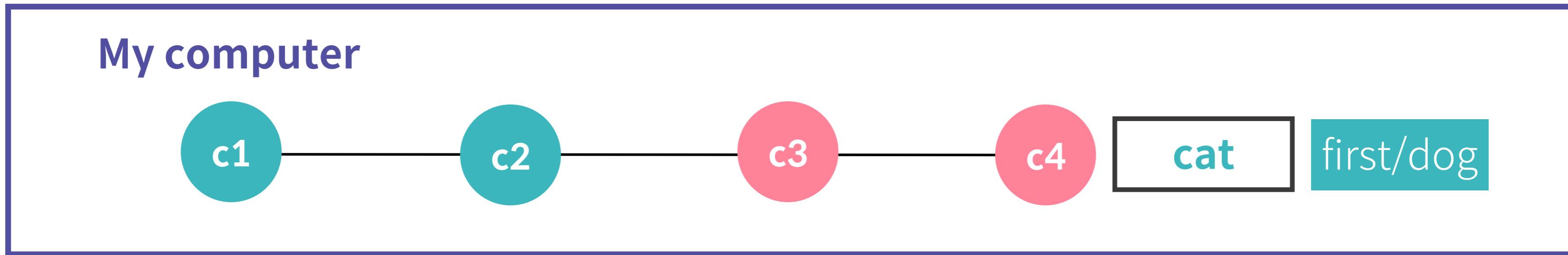
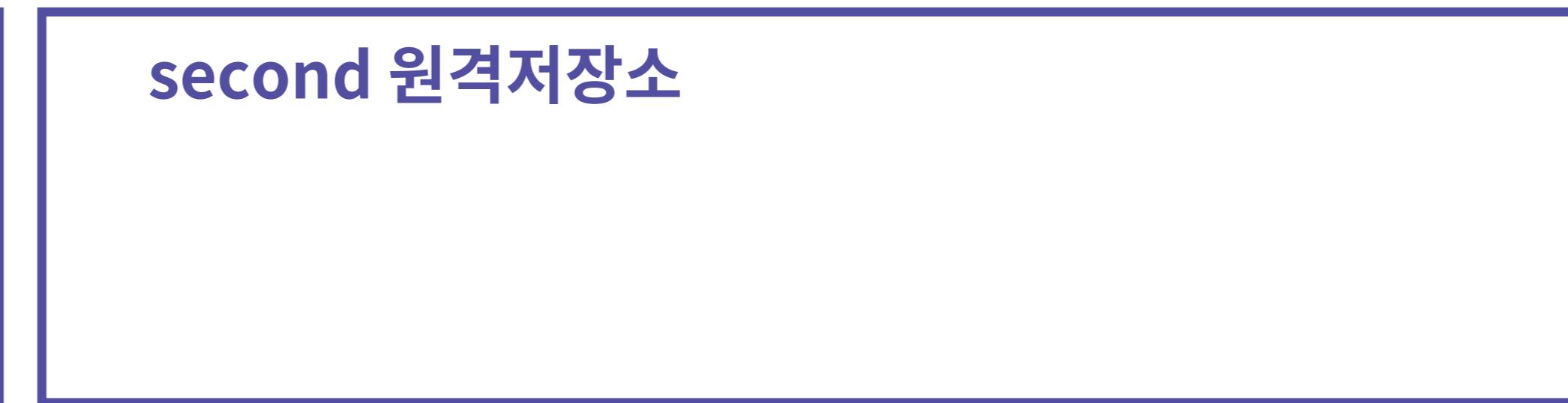
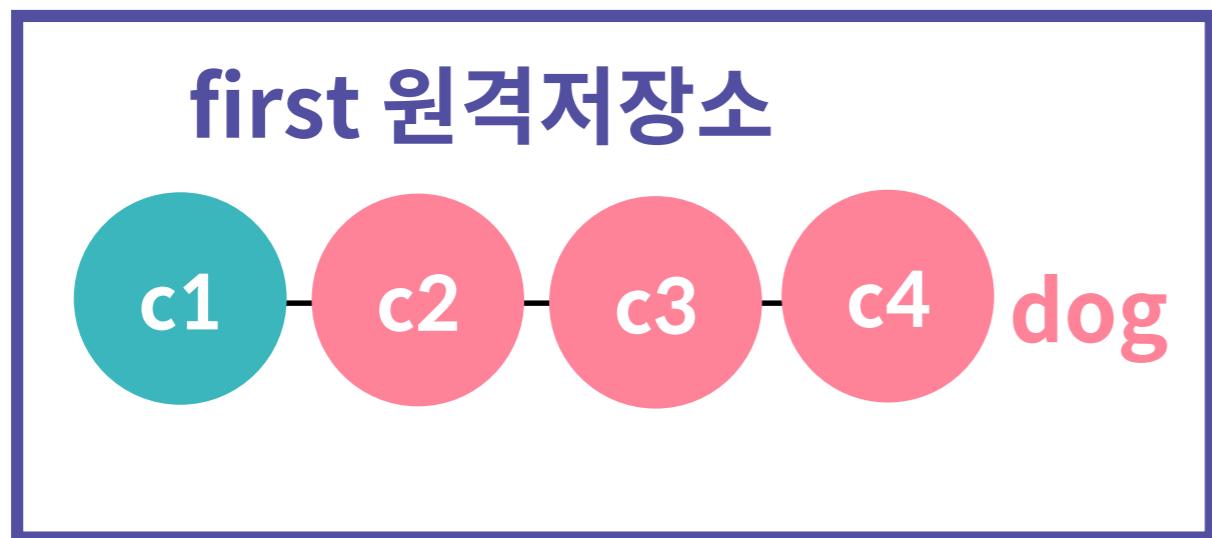
```
git remote add second https://github.com/group/project2
```

```
$ git remote -v
first https://gitlab.com/group/project1 (fetch)
first https://gitlab.com/group/project1 (push)
second https://gitlab.com/group.project2 (fetch)
second https://gitlab.com/group.project2 (push)
```

push 파헤치기

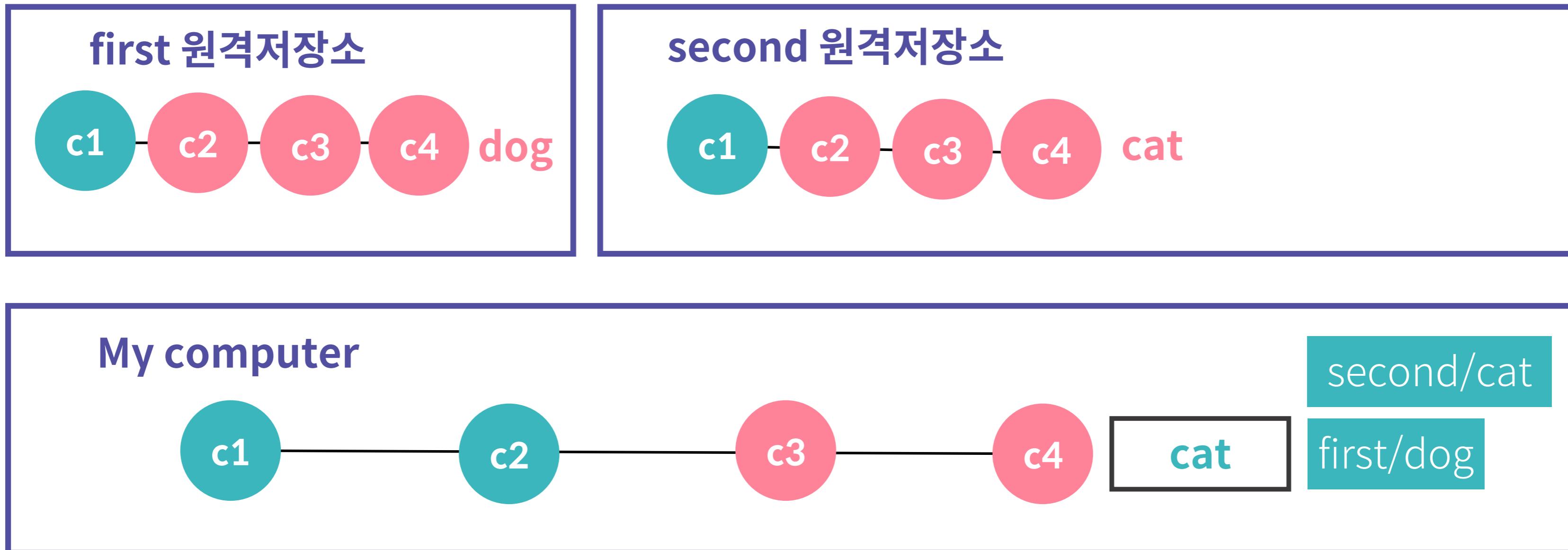
second 원격저장소는 현재 비어있는 상태입니다.

git remote second cat 명령으로 push를 진행 해보겠습니다.



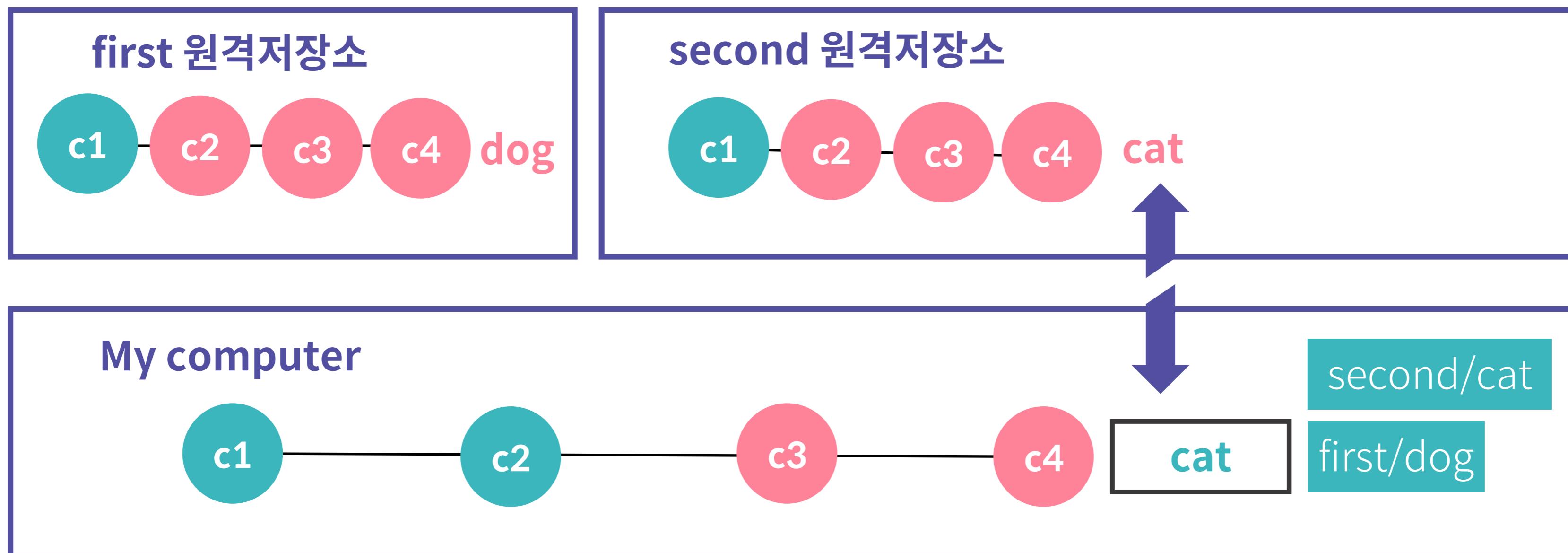
push 파헤치기

cat 브랜치 까지의 내용이 second 저장소로 push 됩니다.



push 파헤치기

아직 로컬저장소의 cat 브랜치가 원격저장소의 cat 브랜치와
트래킹 되지 않은 상태입니다.



push 파헤치기

로컬과 원격 브랜치가 아직 서로 연결이 안되어 있기 때문에 바로
git push **second** 로 명령어를 사용할 수 없습니다.

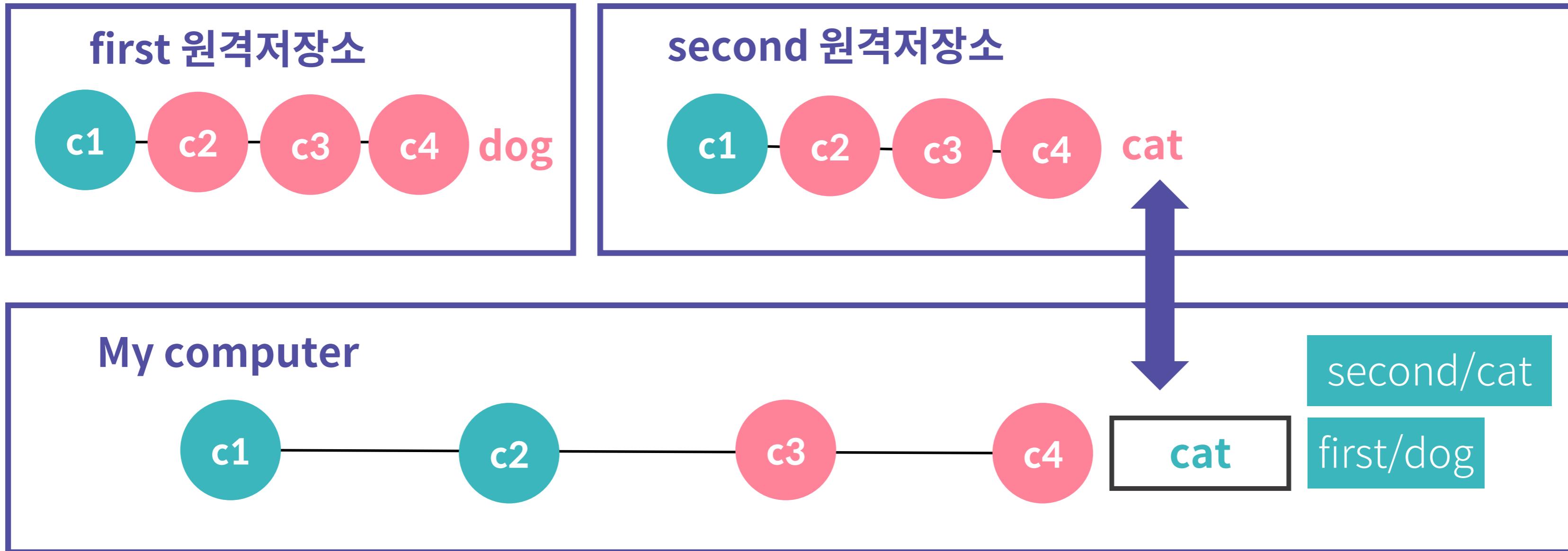
```
elice@Desktop MINGW64 ~/local_repo (cat)
$ git remote -v
fatal: The current branch cat has no upstream branch.
To push the current branch and set the remote as upstream
```

```
git push --set-upstream second cat
```

push 파헤치기

cat 브랜치를 트래킹 해주기 위해서 아래 명령어를 사용하겠습니다.

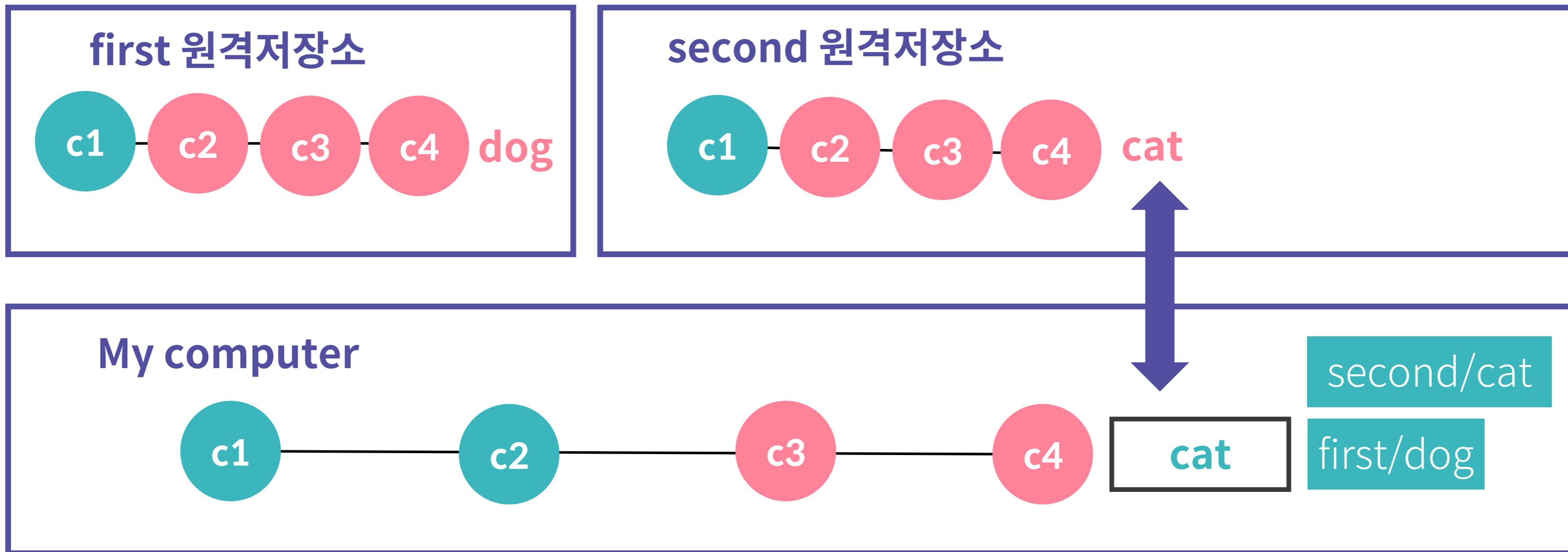
```
git branch -set-upstream-to=second/cat cat
```



push 파헤치기

또는 아래와 같이 push와 동시에 트래킹을 설정할 수 있습니다.

```
git push --set-upstream second cat
```



결론

트래킹 브랜치는 저장소에 존재하는 특정 브랜치의 위치를 알려줍니다.

트래킹 브랜치의 정보를 이용해서 내가 원하는 브랜치와 서로 트래킹을
시켜주고 지속적으로 관리할 수 있습니다.

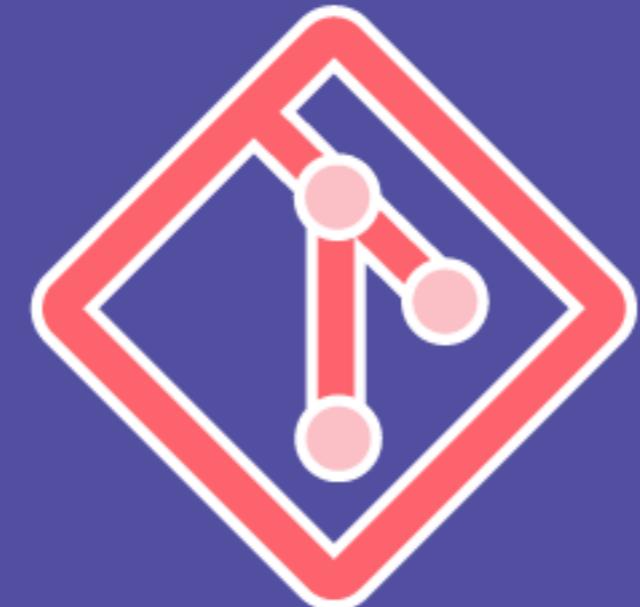


/* elice */

contact@elice.io

Git을 사용한 버전 관리

동기화 오류 모음집(pull편)



/* elice */

동기화와 관련된 오류들

pull 오류

첫 번째 pull 오류

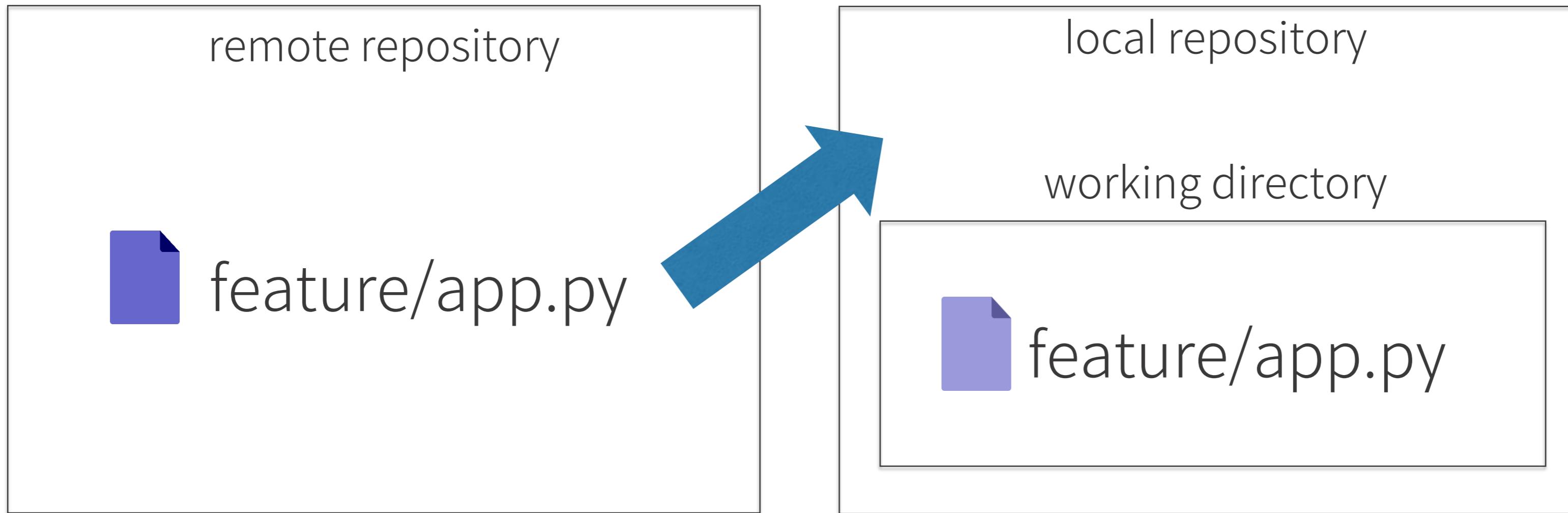
```
$ git pull  
error: The following untracked working tree files would be overwritten by merge:  
      <file_name>  
Please move or remove them before you merge.  
Aborting
```

두 번째 pull 오류

```
$ git pull  
error: Your local changes to the following files would be overwritten by merge:  
      <file_name>  
Please move or remove them before you merge.  
Aborting
```

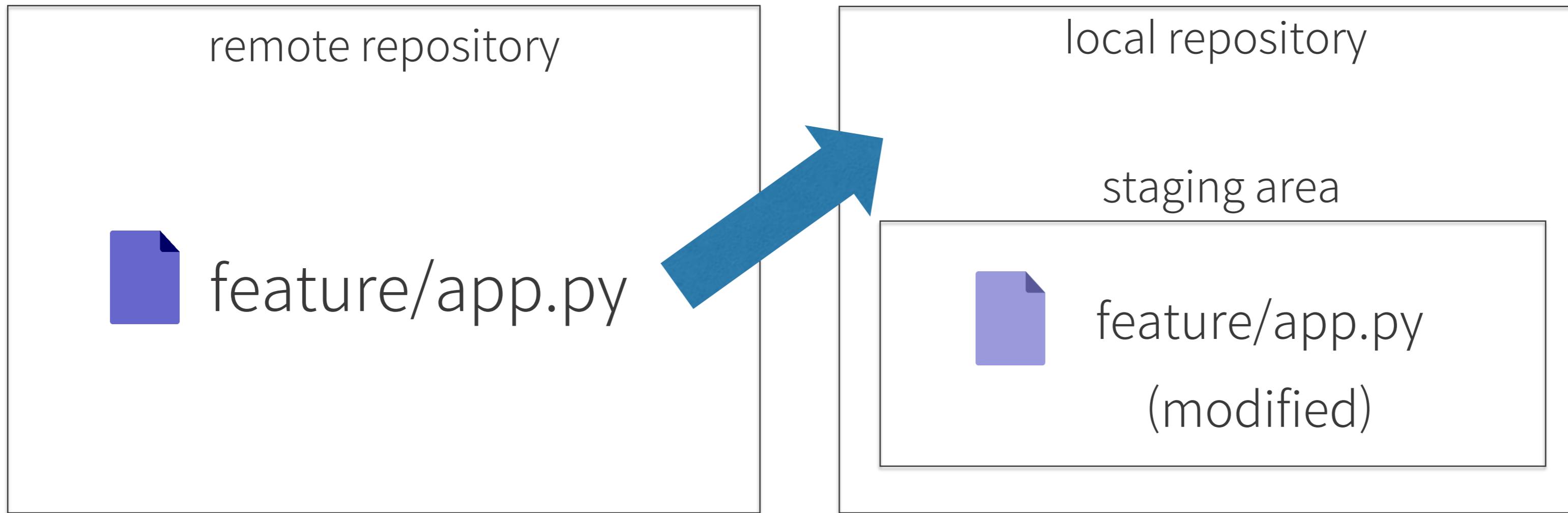
pull 오류 (1)

첫 번째는 pull을 했을 때 받아온 저장소의 파일이 디렉토리 내의 untracked 파일 중 같은 이름을 가질 때 발생하는 오류입니다.



pull 오류 (2)

두 번째는 pull을 했을 때 받아온 저장소의 파일이 로컬 git 저장소에서 수정된 파일과 같은 이름을 가질 때 발생하는 오류입니다.



pull 오류 해결(1)

1. 충돌되는 파일을 삭제 또는 다른 임시 디렉토리로 이동시킨다.
2. 워킹 디렉토리를 초기화 시킨다.

git clean -f -d -x

오류가 발생되는 파일이 한 두개 라면 충돌되는 파일을 다른 디렉토리로
이동시키면 됩니다. 그러나..

충돌이 발생되는 파일이 너무 많아 일일이
이동시킬 수 없을 경우가 발생할 수 있습니다.

git clean -f -d -x

```
$ git pull origin master  
error: The following untracked working tree files would be overwritten by merge:  
.gradle/4.6/fileChanges/last-build.bin  
.gradle/vcsworkingDirs/gc.properties  
app/build/generated/not_namespaced_r_class_soruces1/...  
app/build/generated/not_namespaced_r_class_soruces2/...  
app/build/generated/not_namespaced_r_class_soruces3/...  
...  
Please move or remove them before you merge.  
Aborting
```

git clean -f -d -x

git clean -f -d -x 명령어는

git이 추적 중이지 않은 워킹 디렉토리에 존재하는 파일들과
.gitignore로 git의 관리를 받지 않는 파일들까지
모두 깨끗하게 지워버리는 명령어입니다.

git clean -f -d -x

따라서 수정 중이던 내용 중 필요한 부분이 있다면
커밋으로 수정된 상태를 저장하거나
아예 수정하고 있는 파일을 다른 디렉토리에 옮겨 저장하는 것이
pull 오류를 해결하는 가장 확실한 방법입니다.

pull 오류 해결(2)

1. git stash를 사용하여 작업한 내용을 커밋하지 않고 임시저장 한다.
2. 충돌되는 파일을 삭제 또는 다른 임시 디렉토리로 이동시킨다.

git stash

다음과 같이 작업된 내용이 아직 커밋되지 않고 있다고 가정하겠습니다.

```
$ git status
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    modified: app.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
    modified: lib/myproject.py
```

git stash

git stash 명령어를 이용하여 내용을 임시저장 해보겠습니다.

```
$ git stash  
Saved working directory and index state  
        "WIP on master: 049d078 added the index file"  
HEAD is now at 049d078 added the index file  
(To restore them type "git stash apply")
```

git stash

working directory가 비워진 것을 확인 할 수 있습니다.

```
$ git status  
On branch master  
nothing to commit, working directory clean
```

git stash

git stash list 로 임시저장한 내역을 확인 할 수 있습니다.

```
$ git status list  
stash@{0}: WIP on master: 049d078 added the index file
```

git stash

pull을 완료하고 `git stash apply` 로
가장 최근에 저장한 stash를 불러 올 수 있습니다.

```
$ git stash apply
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working directory)
  modified: app.py
  modified: lib/myproject.py
```

pull 오류 해결

한 가지 주의할 점은 stash는 staging되었던 상태의 파일들을
불러올 때 다시 staging시켜 주지 않습니다.

staging을 유지시켜 주기 위해서는 `--index` 옵션을 주면 됩니다.

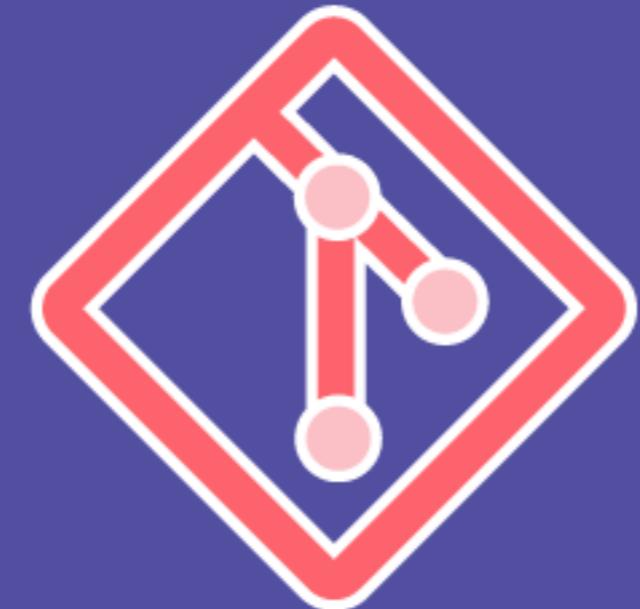


/* elice */

contact@elice.io

Git을 사용한 버전 관리

동기화 오류 모음집(로그인 편)



/* elice */

동기화와 관련된 오류들

로그인 오류

로그인 오류

push/pull 명령어 사용시 패스워드를 잘못 입력하였을 때,
git이 패스워드를 다시 묻지 않고
Authenticate 오류를 발생시키는 경우가 있습니다.

```
$ git push (or) $ git pull
remote: Invalid username or password.
fatal: Authentication failed for 'https://git@gitlab.com/...'
```

GCM(git credential manager)

보통 http는 ssh의 인증방식과 다르게 한 번 git 웹 호스팅 서비스에 등록을 하면 자동으로 인증되는 방식이 아니라
매번 원격저장소에 접근 시
아이디와 패스워드를 입력해 줘야하는 단점이 있습니다.

GCM(git credential manager)

credential 저장소는 이러한 인증정보를 저장해두고
자동으로 입력해주는 시스템입니다.

이때 잘못된 패스워드를 자동으로 git이 입력하기 때문에
앞선 오류가 발생하게 됩니다.

GCM(git credential manager)

```
git config --system --unset credential.helper
```

위의 명령어는 잘못 지정한 인증정보를 다시 초기화 해줍니다.

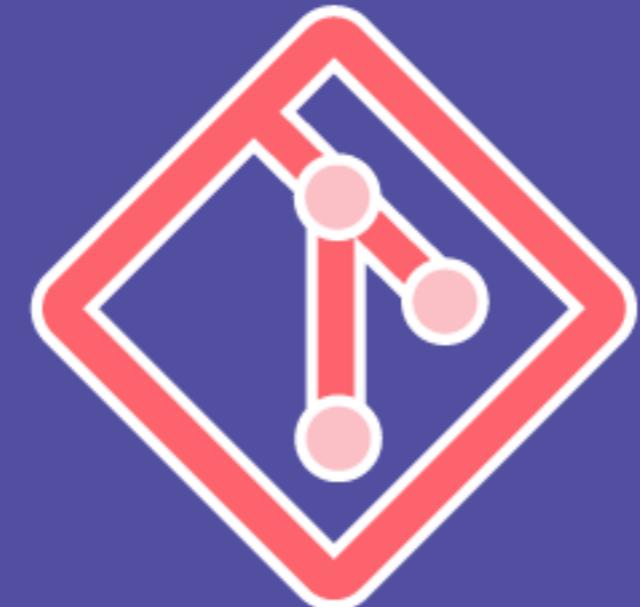


/* elice */

contact@elice.io

Git을 사용한 버전 관리

트래킹 브랜치



/* elice */

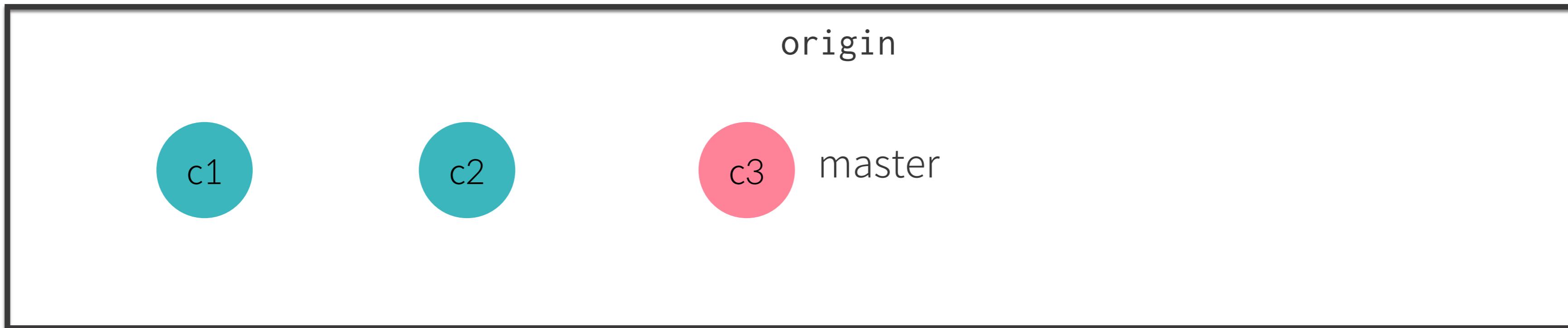
트래킹 브랜치

origin/master

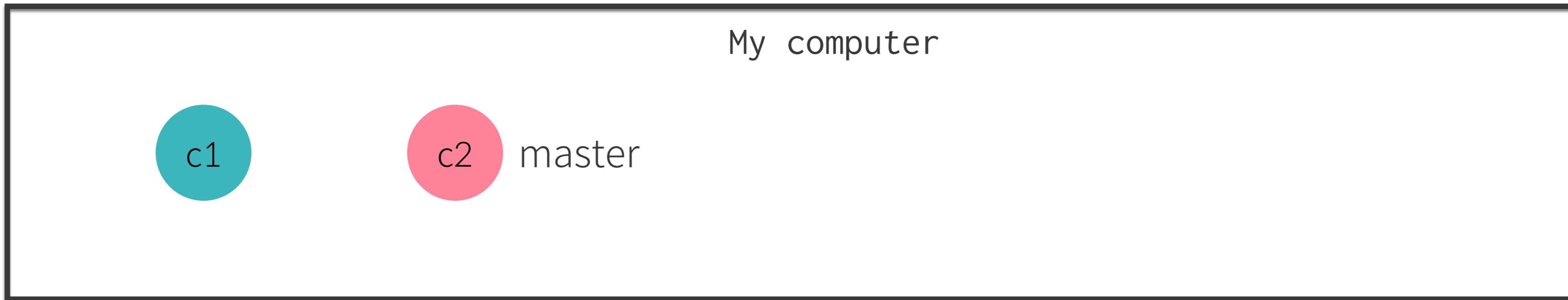
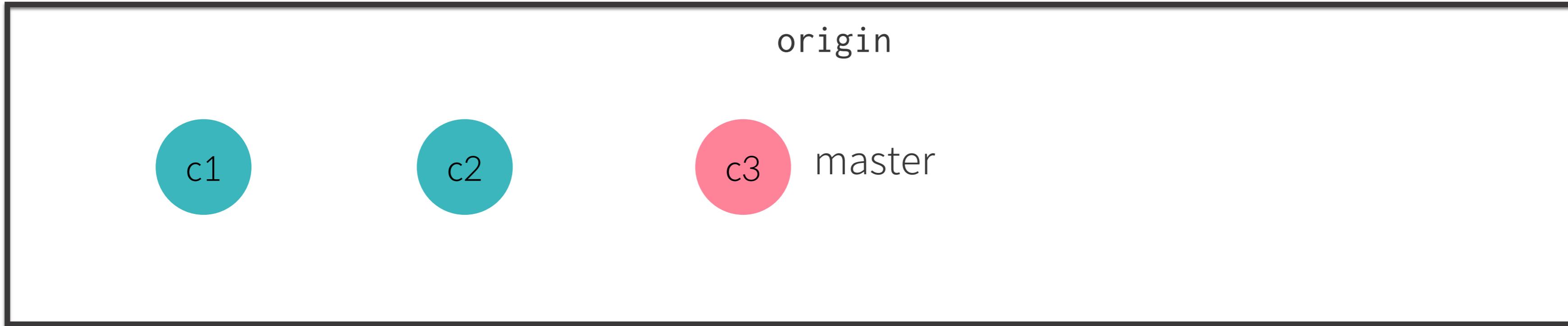
원격저장소 역시 **git 저장소**입니다.

다만, working directory가 존재하지 않으며 이를
bare 저장소라고 부릅니다.

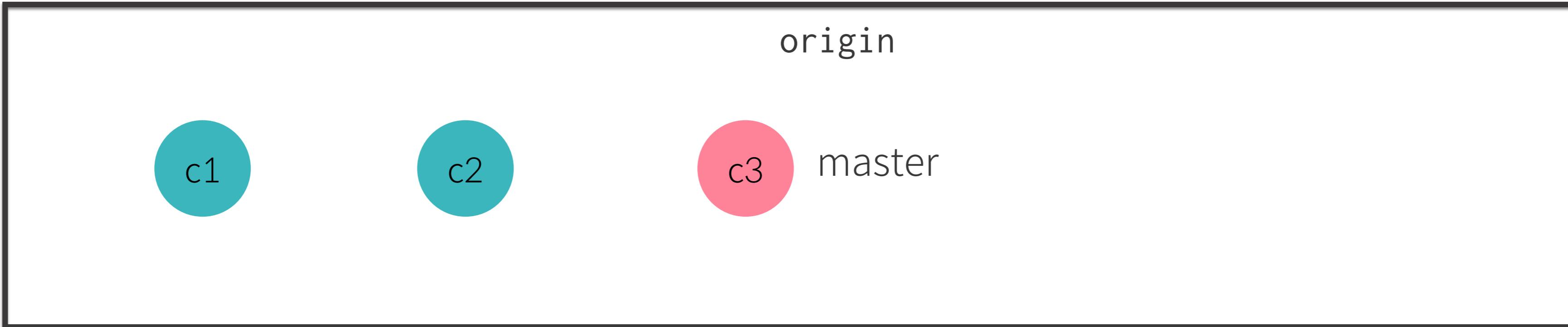
아래와 같이 master 브랜치가 존재합니다.



origin/master



origin/master



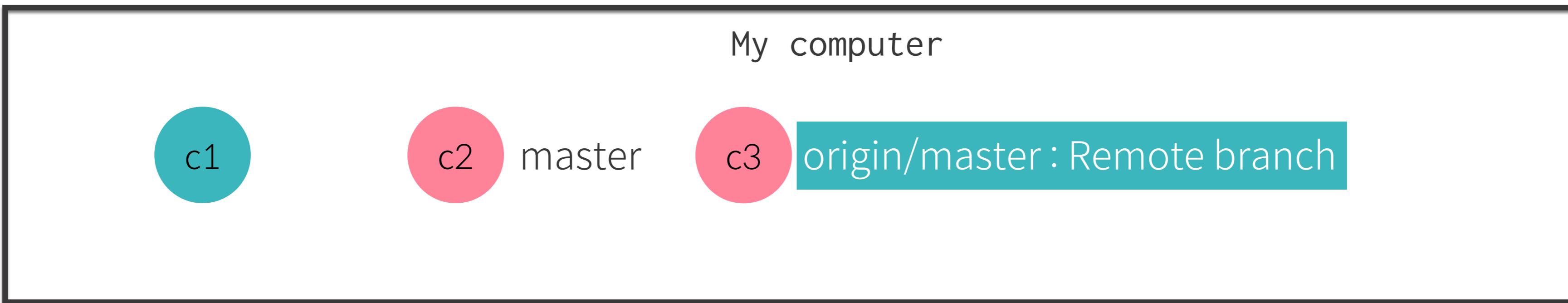
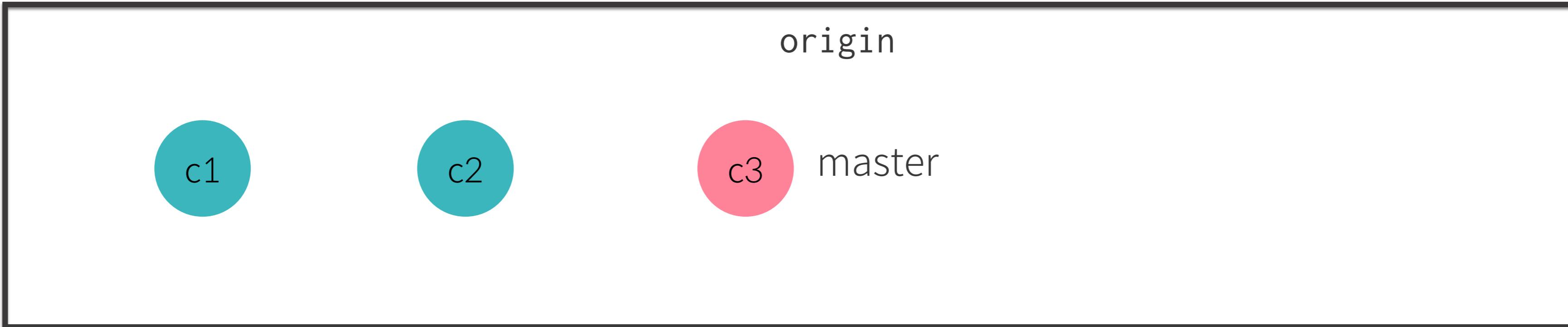
git fetch 또는 git pull 을 이용해서

원격저장소를 동기화 하게 되면

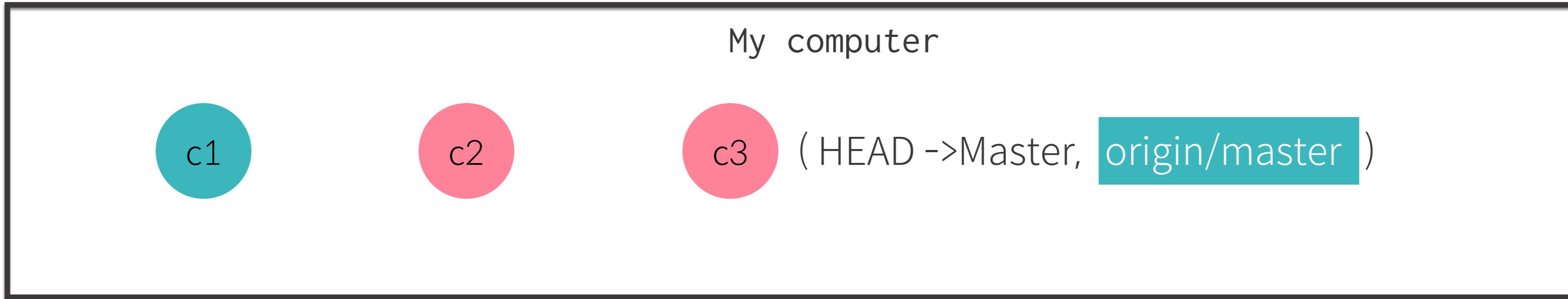
origin/master 브랜치가 생기게 됩니다.

이 브랜치를 **트래킹 브랜치**라고 부릅니다.

origin/master



origin/master



트래킹 브랜치는 원격저장소의 브랜치가 어디에 위치하는지를 보여줍니다.

트래킹 브랜치는

1. 브랜치이지만 checkout 할 수 없습니다.
2. 가리키고 있는 commit의 위치를 강제로 변경할 수 없습니다.
3. `git merge origin/master` 로 merge가 가능합니다.



/* elice */

contact@elice.io