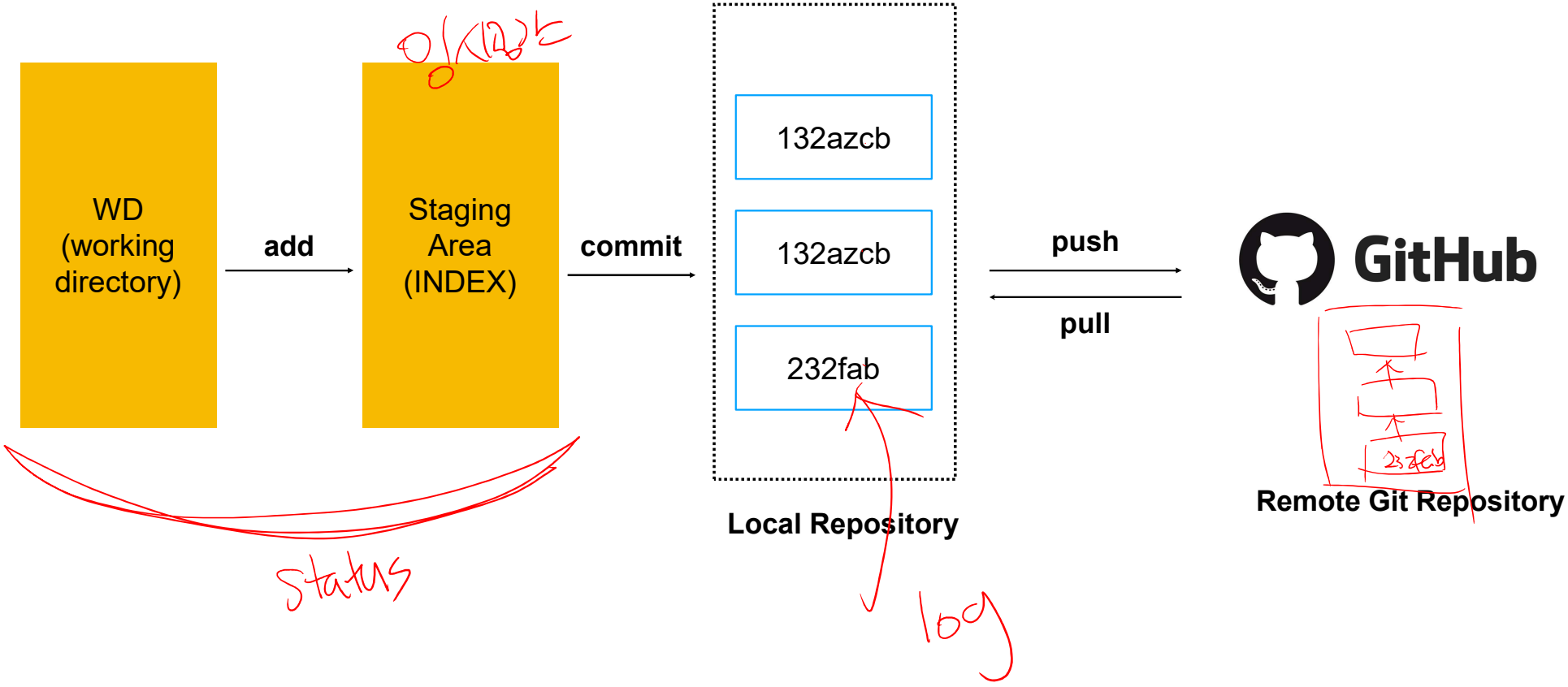
Branch, Github Flow

- Github으로 협업하는 흐름

Git (이미 알고 있어야하는 개념 정리)



Git 들어가기전에..

Git을 CLI(Command Line Interface)에서 활용하기 위해서는 아래의 명령어는 필수적이다!

항상 모든 명령어 뒤에 상태를 확인하자.

```
$ git status
```

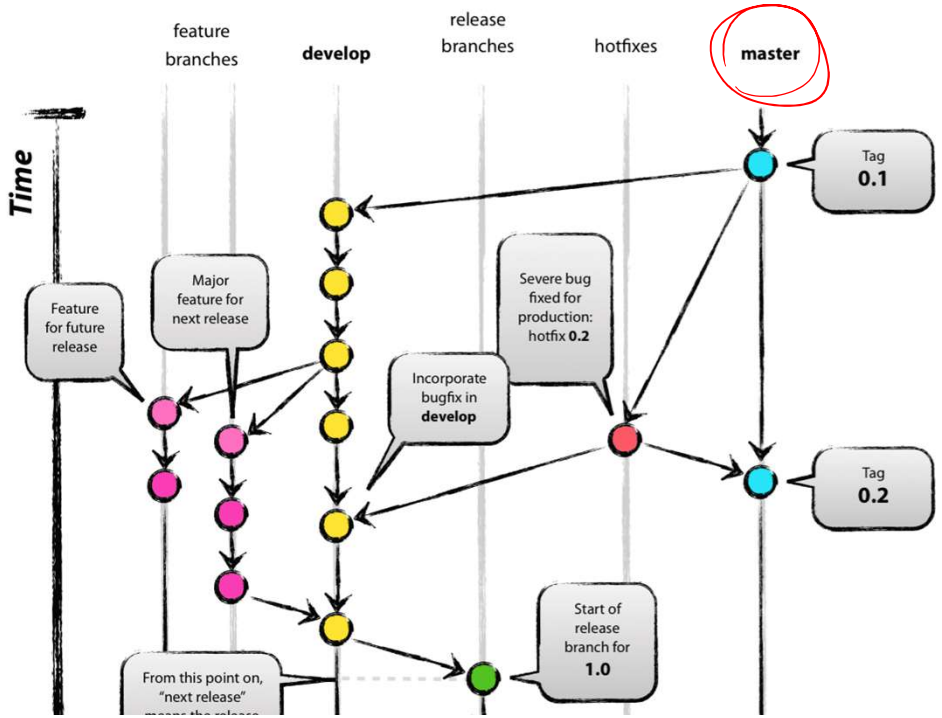
$ git add .

현재 디렉토리 내
→ 변경된 파일들만
(status)

Git Flow

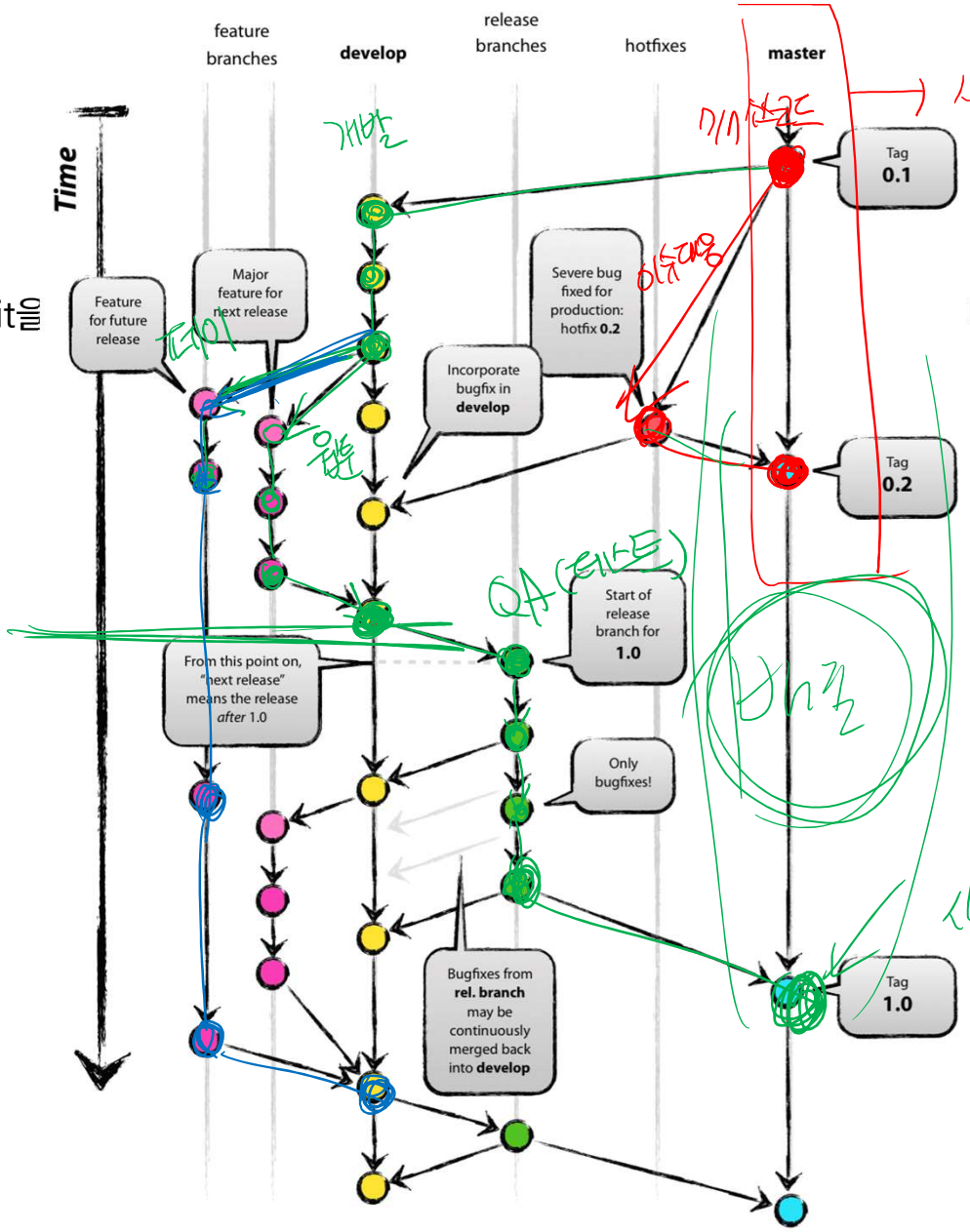
Git을 활용하여 협업하는 흐름으로 branch를 활용하는 전략을 의미한다.

가장 대표적으로 활용되는 전략은 다음과 같다.



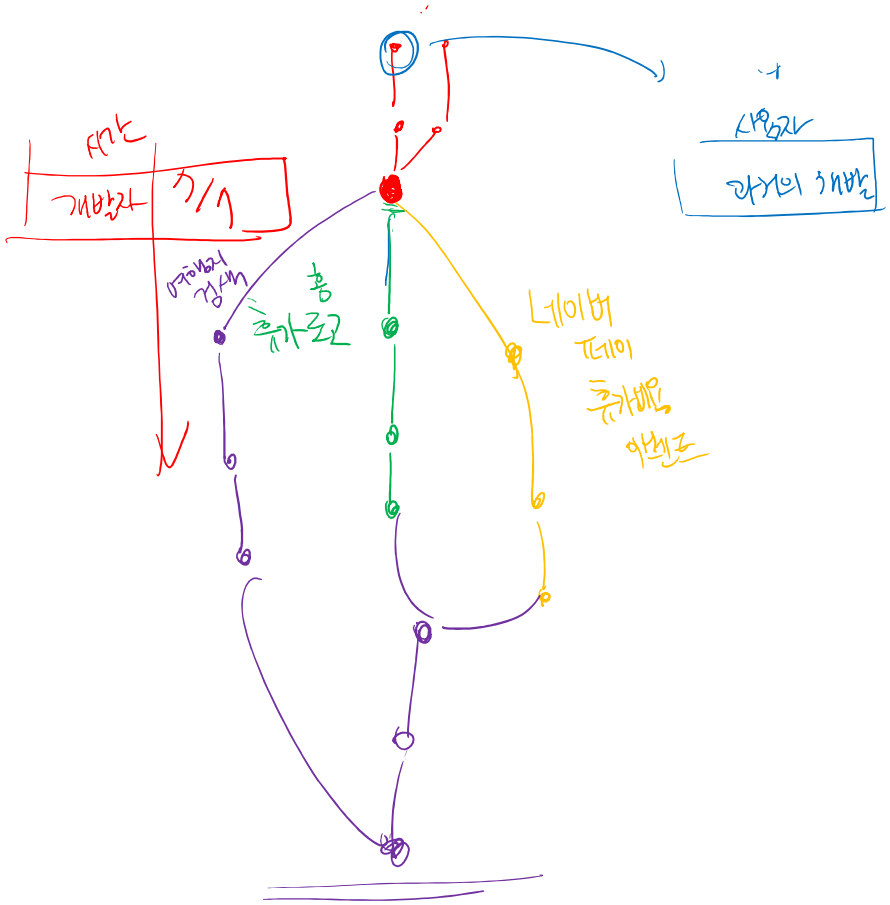
https://nvie.com/posts/a-successful-git-branching-model/

Git을



다.

https://nvie.com/posts/a-successful-git-branching-model/



Git Flow

branch	주요 특징	예시
master (main)	* 배포 가능한 상태의 코드	LOL 클라이언트 라이브 버전 (9.23.298.3143)
develop (main)	* feature branch로 나뉘어지거나, 발생된 버그 수정 등 개발 진행 * 개발 이후 release branch로 갈라짐.	다음 패치를 위한 개발 (9.24)
feature branches (supporting)	* 기능별 개발 브랜치(topic branch) * 기능이 반영되거나 드랍되는 경우 브랜치 삭제	개발시 기능별 예) 신규챔피언 세나, 드래곤 업데이트
release branches (supporting)	* 개발 완료 이후 QA/Test 등을 통해 얻어진 다음 배포 전 minor bug fix 등 반영	9.24a, 9.24b, …
hotfixes (supporting)	* 긴급하게 반영 해야하는 bug fix * release branch는 다음 버전을 위한 것이라면, hotfix branch는 현재 버전을 위한 것	긴급 패치를 위한 작업

https://nvie.com/posts/a-successful-git-branching-model/

Git Flow

Git Flow는 정해진 답이 있는 것은 아니다.

Github Flow, Gitlab Flow 등의 각 서비스별 제안되는 흐름이 있으며,

변형되어 각자의 프로젝트/회사에서 활용 되고 있다.

간단하게 브랜치를 활용하는 명령어를 알아보고,

프로젝트에 활용할 수 있는 간단한 버전의 브랜치 전략을 배워보자.

Branch basic commands

1. 브랜치 생성

```
(master) $ git branch {branch name}
```

2. 브랜치 이동

```
(master) $ git checkout {branch name}
```

3. 브랜치 생성 및 이동

```
(master) $ git checkout -b {branch name}
```

4. 브랜치 목록

```
(master) $ git branch
```

5. 브랜치 삭제

```
(master) $ git branch -d {branch name}
```

Branch merge

각 branch에서 작업을 한 이후 이력을 합치기 위해서는 일반적으로 merge 명령어를 사용한다.

병합을 진행할 때, 만약 서로 다른 이력(commit)에서 동일한 파일을 수정한 경우 충돌이 발생할 수 있다.

이 경우에는 반드시 직접 수정을 진행 해야 한다.

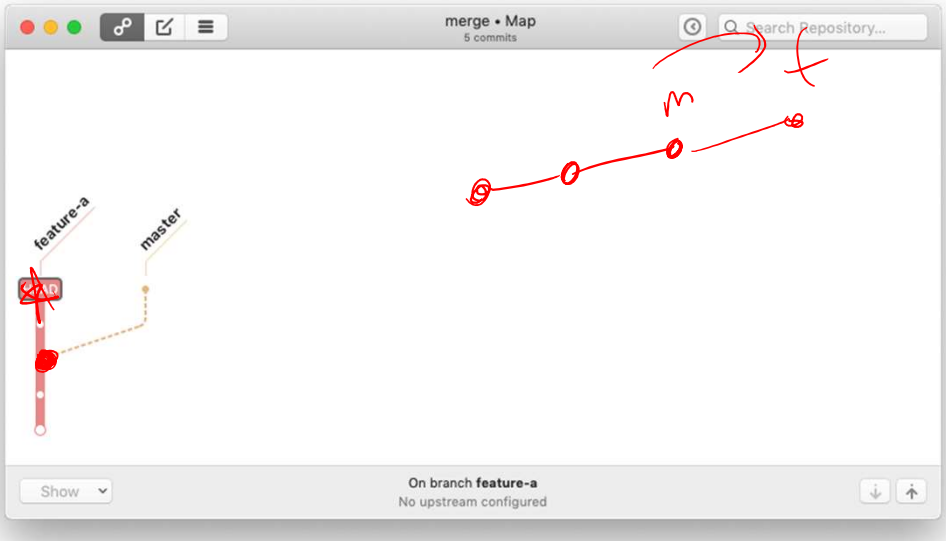
충돌이 발생한 것은 오류가 발생한 것이 아니라 이력이 변경되는 과정에서 반드시 발생할 수 있는 것이다.

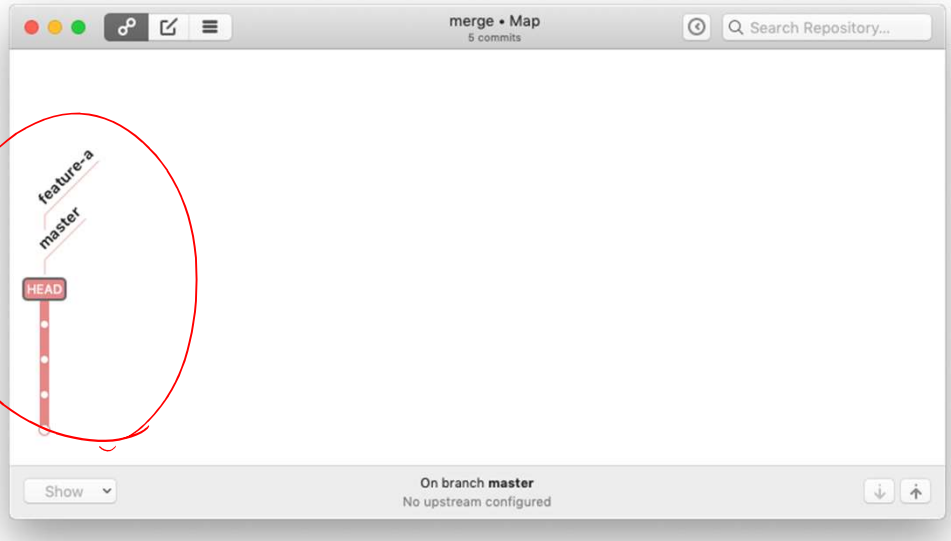
Branch merge - fast-forward

기존 master 브랜치에 변경사항이 없어 단순히 앞으로 이동

```
(master) $ git merge feature-a
Updating 54b9314..5429f25
Fast-forward
```

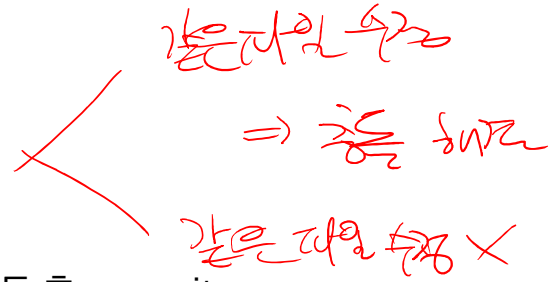
1. feature-a branch로 이동 후 commit
2. master 별도 변경 없음
3. master branch로 병합





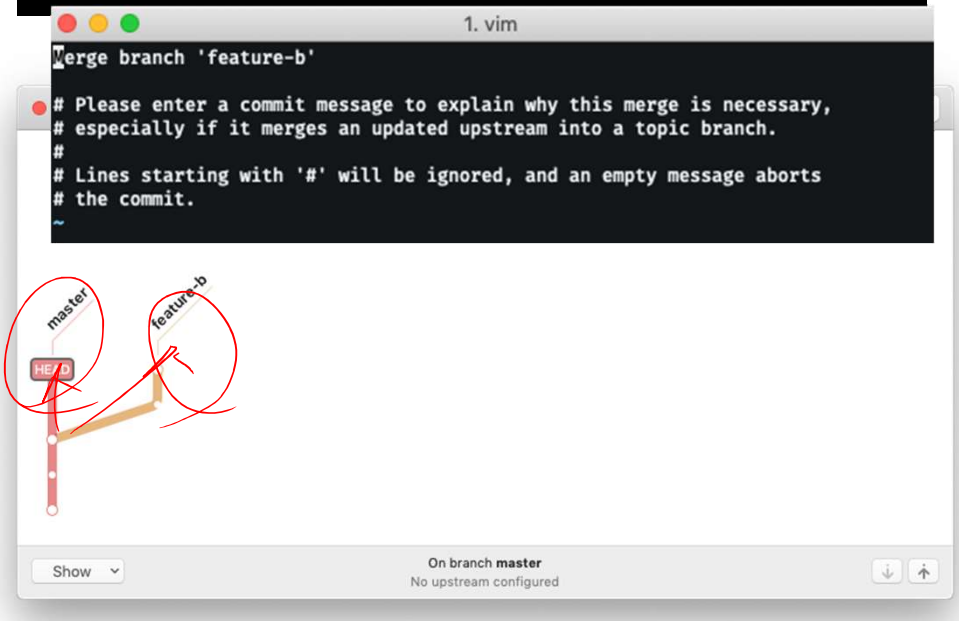
Branch merge – merge commit

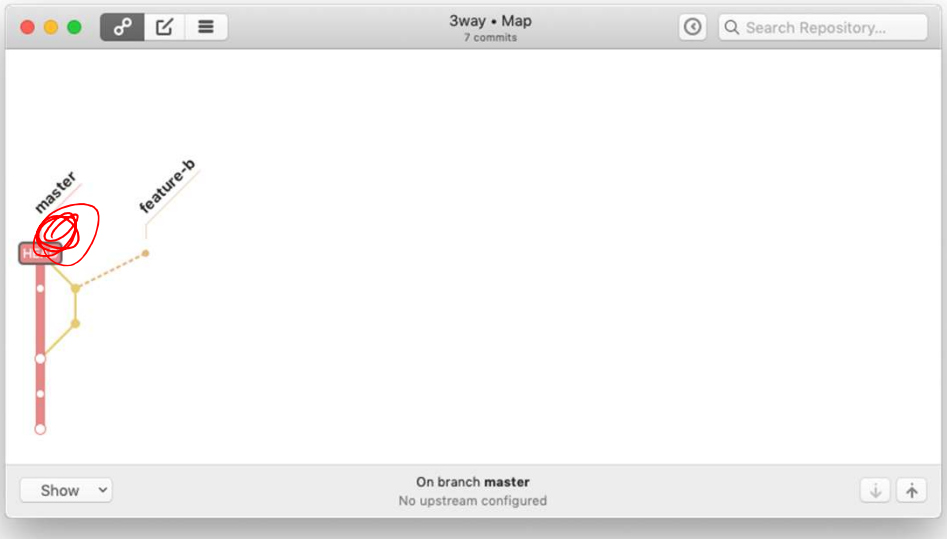
기존 master 브랜치에 변경사항이 있어 병합 커밋 발생



```
(master) $ git merge feature-a
Already up to date!
Merge made by the 'recursive' strategy.
```

1. feature-a branch로 이동 후 commit
2. master branch commit
3. master branch로 병합





Github Flow 기본 원칙

Github Flow는 Github에서 제안하는 브랜치 전략으로 다음과 같은 기본 원칙을 가지고 있다.

1. master branch는 반드시 배포 가능한 상태여야 한다.
There's only one rule: anything in the master branch is always deployable.

2. feature branch는 각 기능의 의도를 알 수 있도록 작성한다.
Your branch name should be descriptive,
so that others can see what is being worked on.

3. Commit message는 매우 중요하며, 명확하게 작성한다.
Commit messages are important. By writing clear commit messages,
you can make it easier for other people to follow along and provide feedback.

4. Pull Request를 통해 협업을 진행한다.
Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories.

5. 변경사항을 반영하고 싶다면, master branch에 병합한다.
Now that your changes have been verified in production, it is time to merge your code into the master branch.

1) Feature Branch Workflow

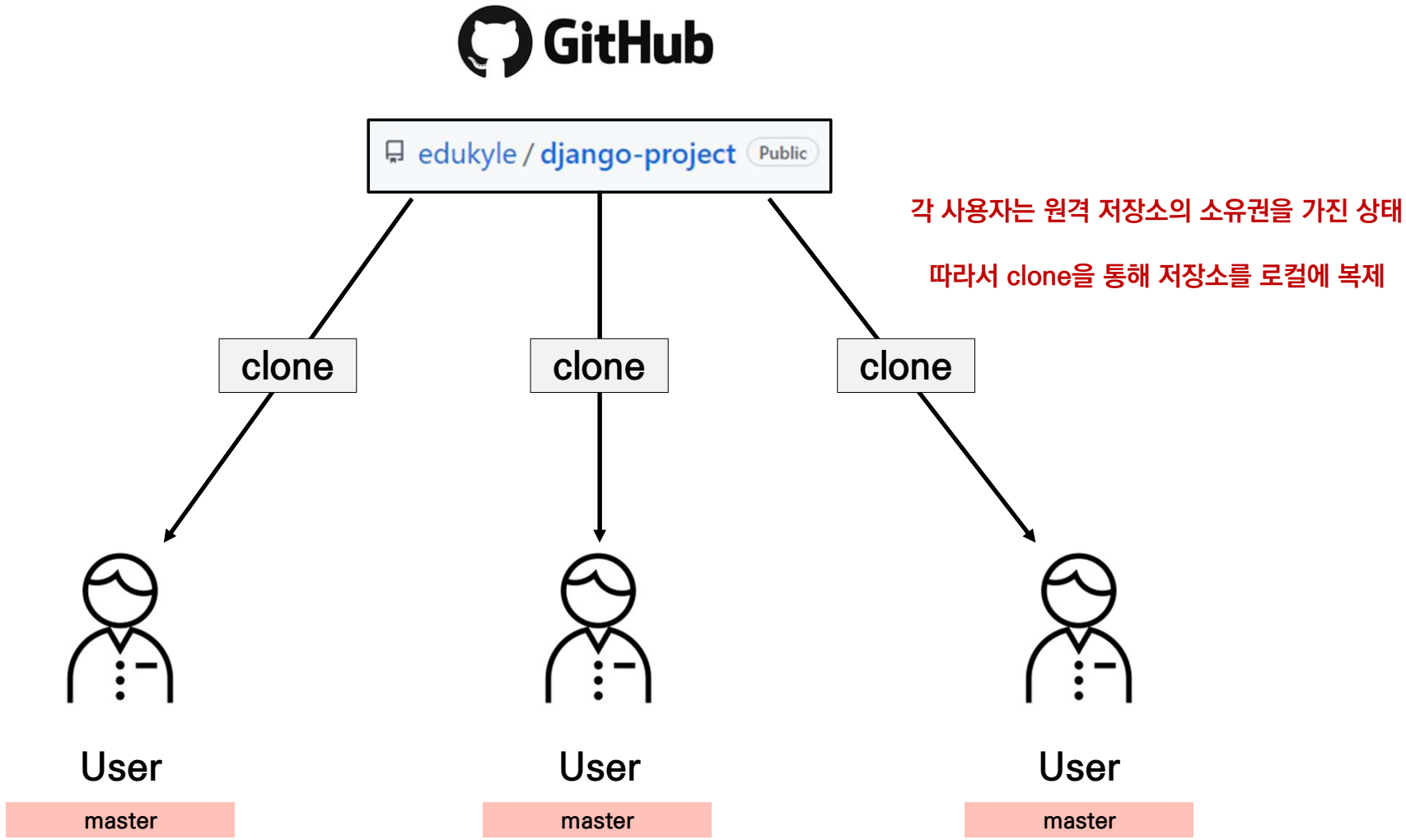
Shared repository model (저장소의 소유권이 있는 경우)

2) Forking Workflow

Fork & Pull model (저장소의 소유권이 없는 경우)

1) Feature Branch Workflow

1) Feature Branch Workflow (1 / 10)



각 사용자는 원격 저장소의 소유권을 가진 상태

따라서 clone을 통해 저장소를 로컬에 복제

1) Feature Branch Workflow (2 / 10)

GitHub

edukyle / django-project Public

기능 추가를 위해 branch 생성 및 기능 구현

User

master

feature/login

User

master

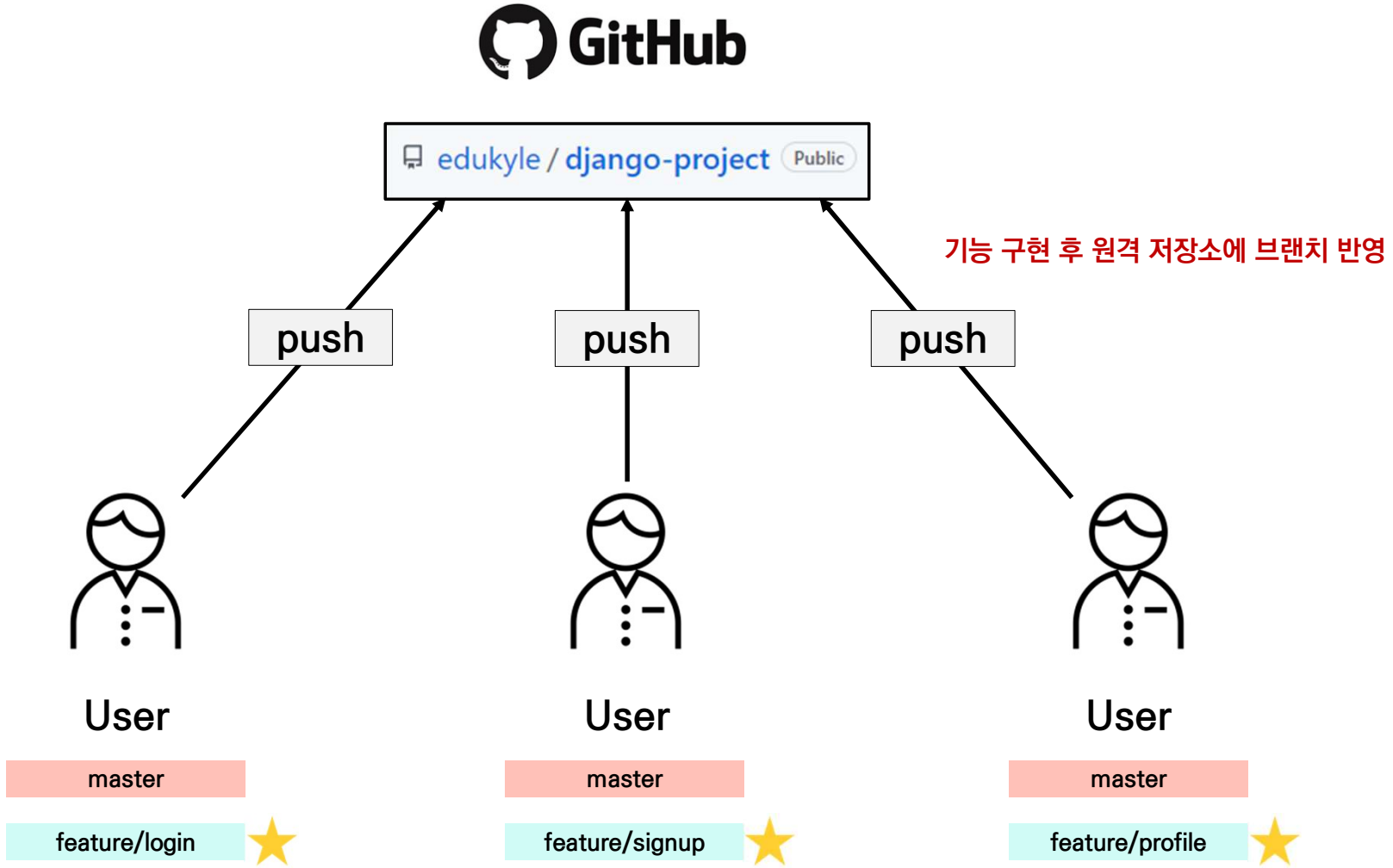
feature/signup

User

master

feature/profile

1) Feature Branch Workflow (3 / 10)



1) Feature Branch Workflow (4 / 10)

edukyle / django-project Public

master

feature/login

feature/signup

feature/profile

User

master

feature/login

User

master

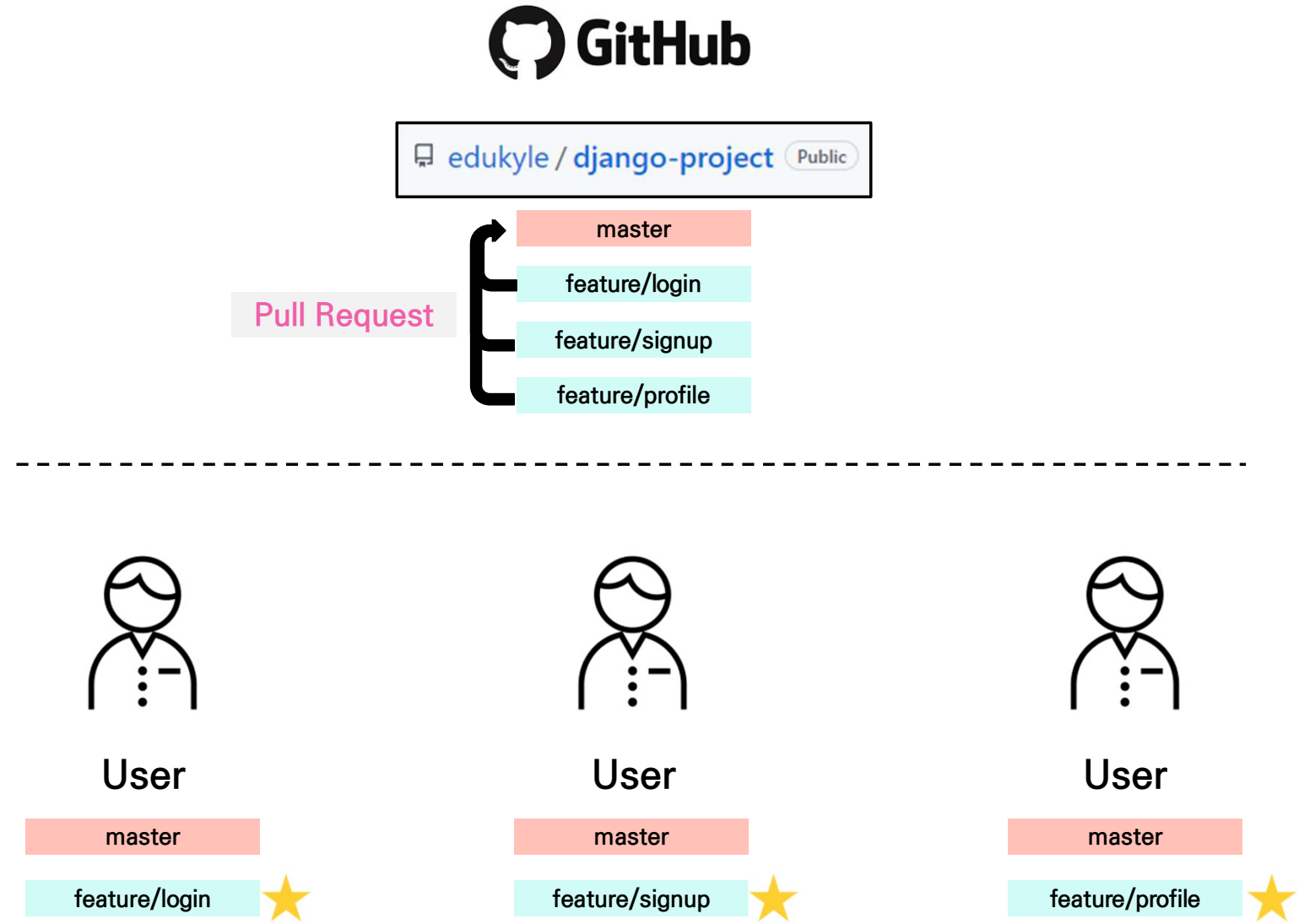
feature/signup

User

master

feature/profile

1) Feature Branch Workflow (5 / 10)



1) Feature Branch Workflow (6 / 10)

edukyle / django-project Public

master 병합 완료

feature/login

feature/signup

feature/profile

병합 완료 된 브랜치 삭제

User

master

feature/login ★

User

master

feature/signup ★

User

master

feature/profile ★

1) Feature Branch Workflow (7 / 10)

edukyle / django-project Public

master

master 브랜치로 switch

User

master

feature/login

User

master

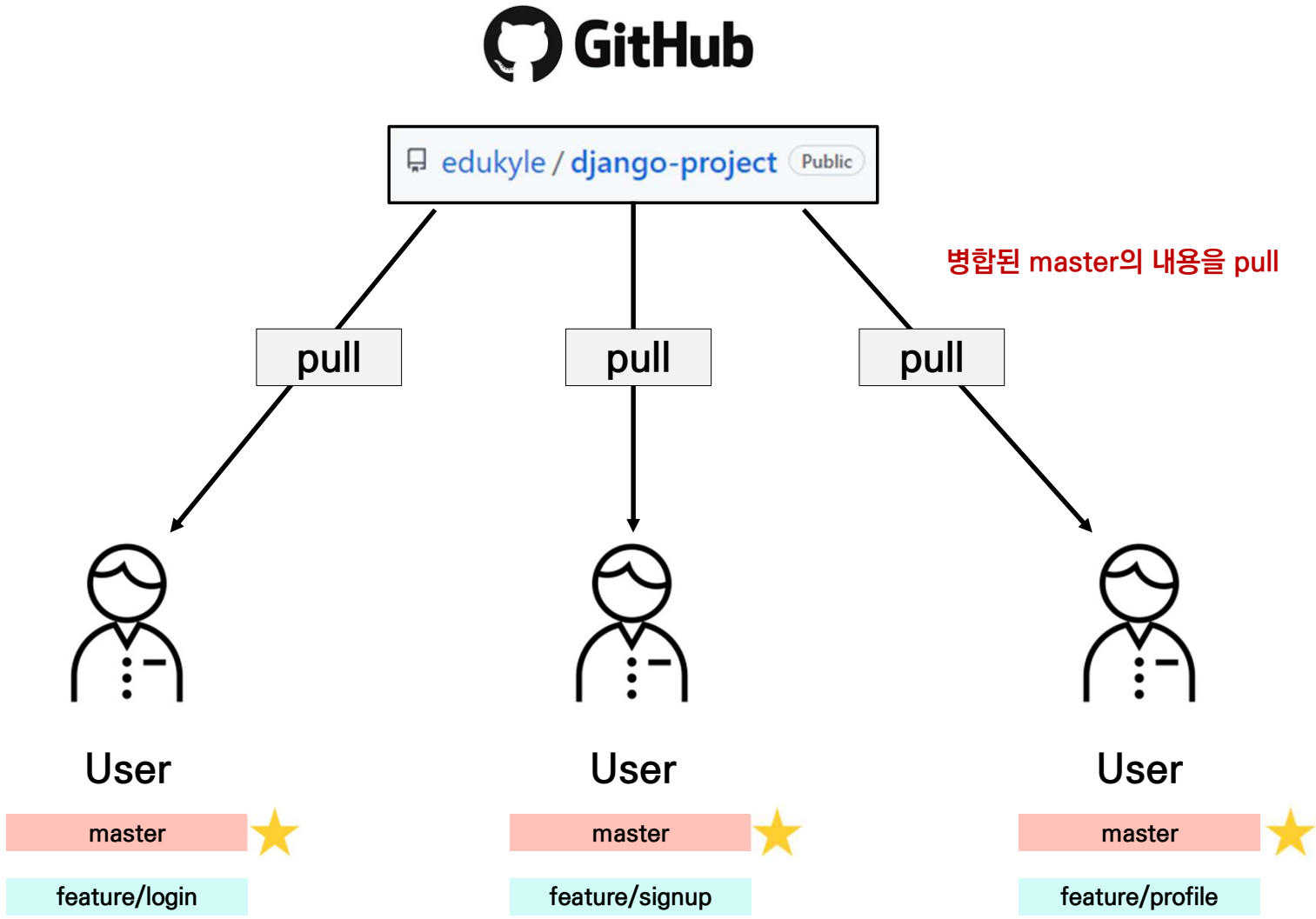
feature/signup

User

master

feature/profile

1) Feature Branch Workflow (8 / 10)



1) Feature Branch Workflow (9 / 10)

edukyle / django-project Public

원격 저장소에서 병합 완료 된 로컬 브랜치 삭제

User

master

User

master

User

master

1) Feature Branch Workflow (10 / 10)

edukyle / django-project Public

새로운 기능 추가를 위해 branch 생성 및 과정 반복

User

master

feature/pay

User

master

feature/delete

User

master

feature/post

2) Forking Workflow

2) Forking Workflow (1 / 10)

원본

AlexKwonPro / kakao_clone Public

fork

복제

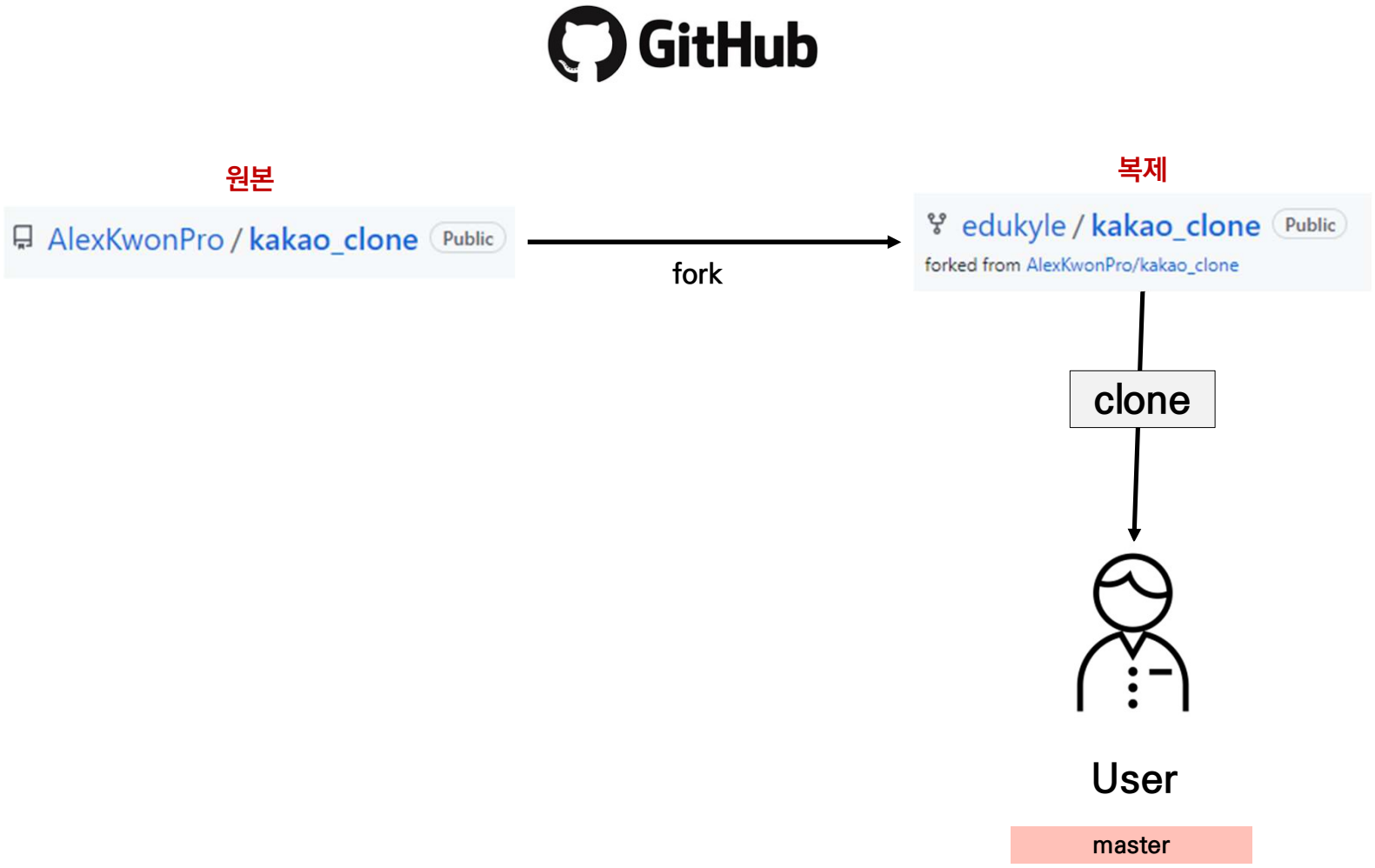
edukyle / kakao_clone Public

forked from AlexKwonPro/kakao_clone

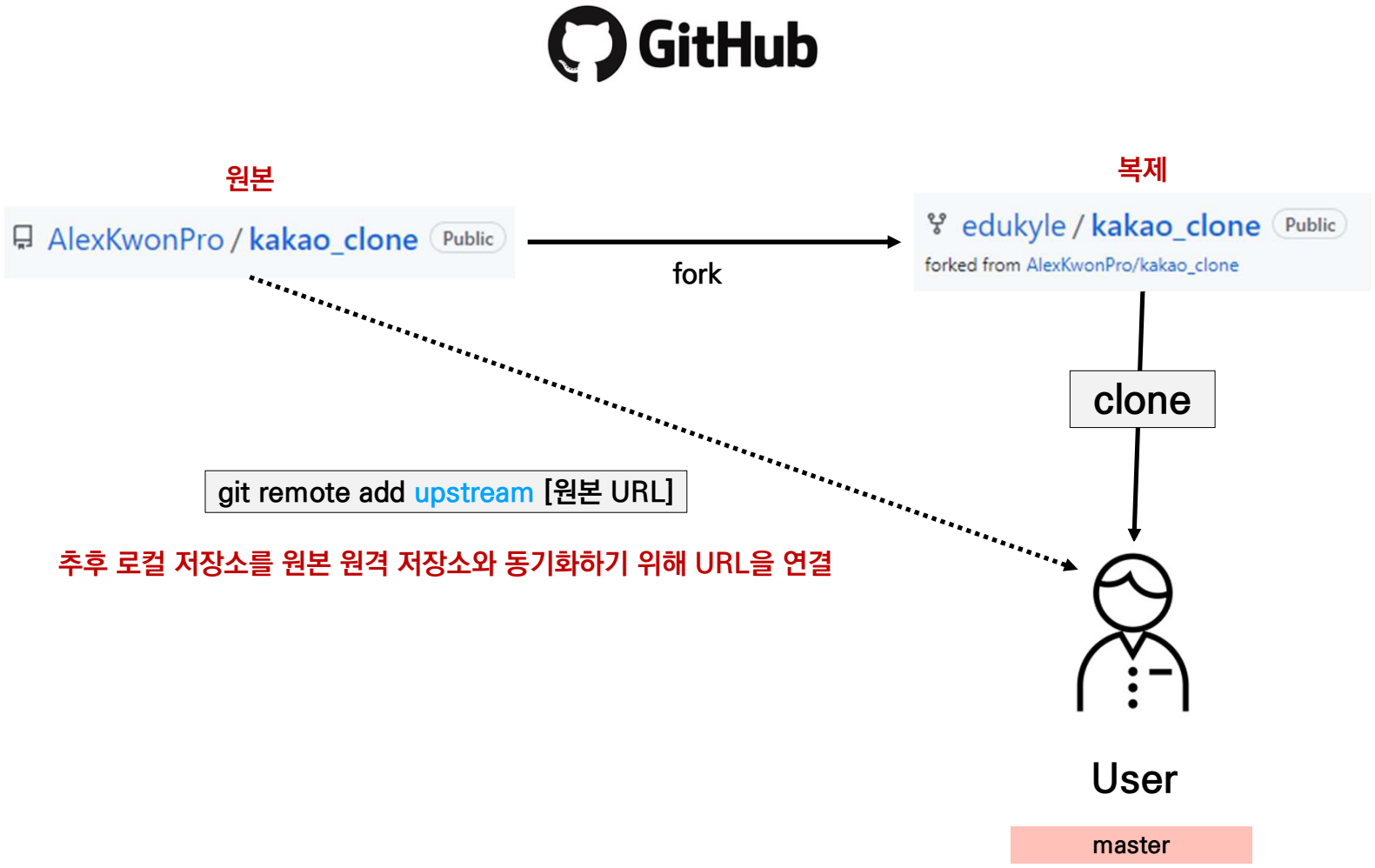
소유권이 없는 원격 저장소를 fork를 통해 복제

User

2) Forking Workflow (2 / 10)



2) Forking Workflow (3 / 10)



2) Forking Workflow (4 / 10)

원본 (upstream)

AlexKwonPro / kakao_clone Public

fork

복제 (origin)

edukyle / kakao_clone Public

forked from AlexKwonPro/kakao_clone

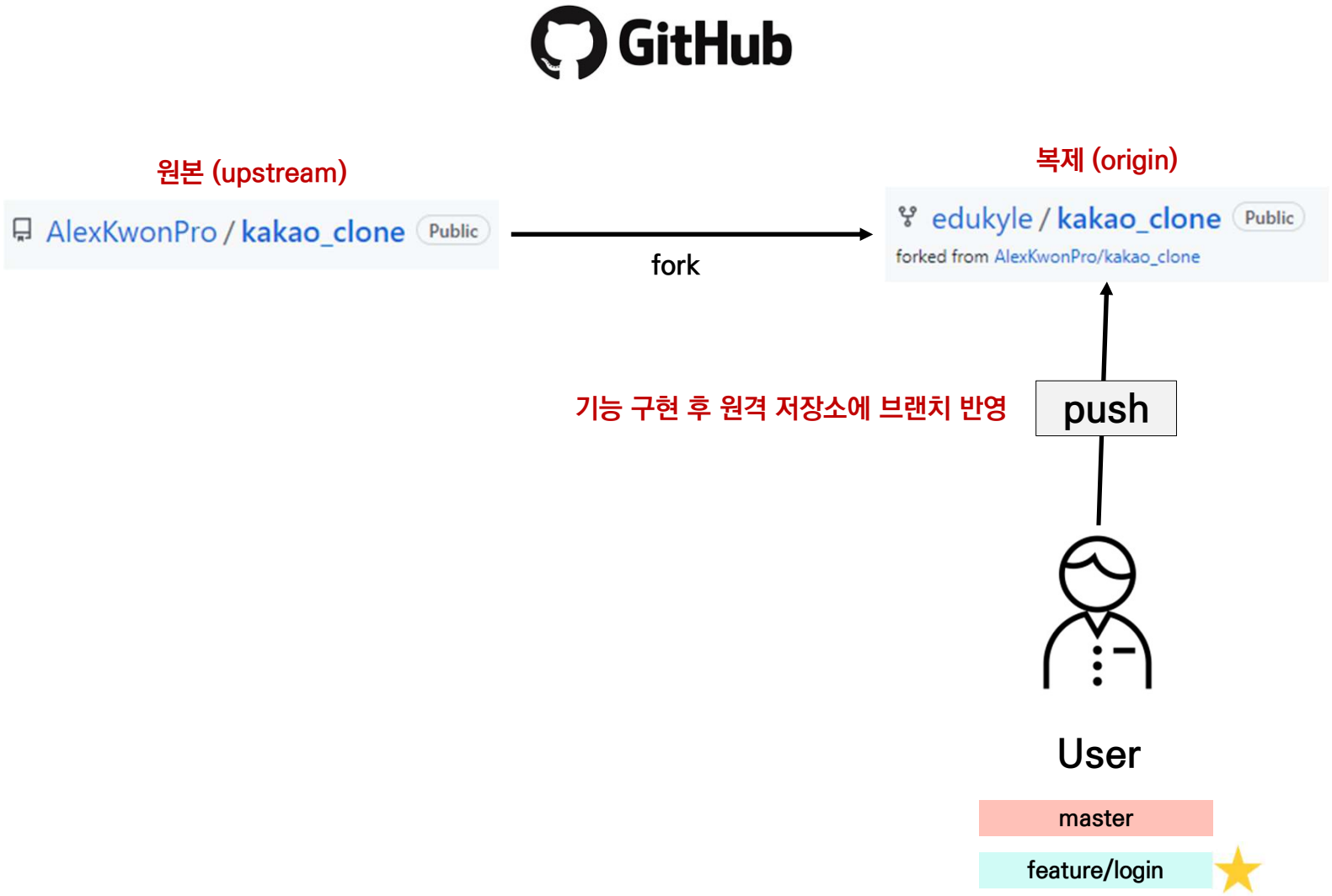
User

master

feature/login

기능 추가를 위해 branch 생성 및 기능 구현

2) Forking Workflow (5 / 10)



2) Forking Workflow (6 / 10)

원본 (upstream)

AlexKwonPro / kakao_clone Public

fork

복제 (origin)

edukyle / kakao_clone Public

forked from AlexKwonPro/kakao_clone

master

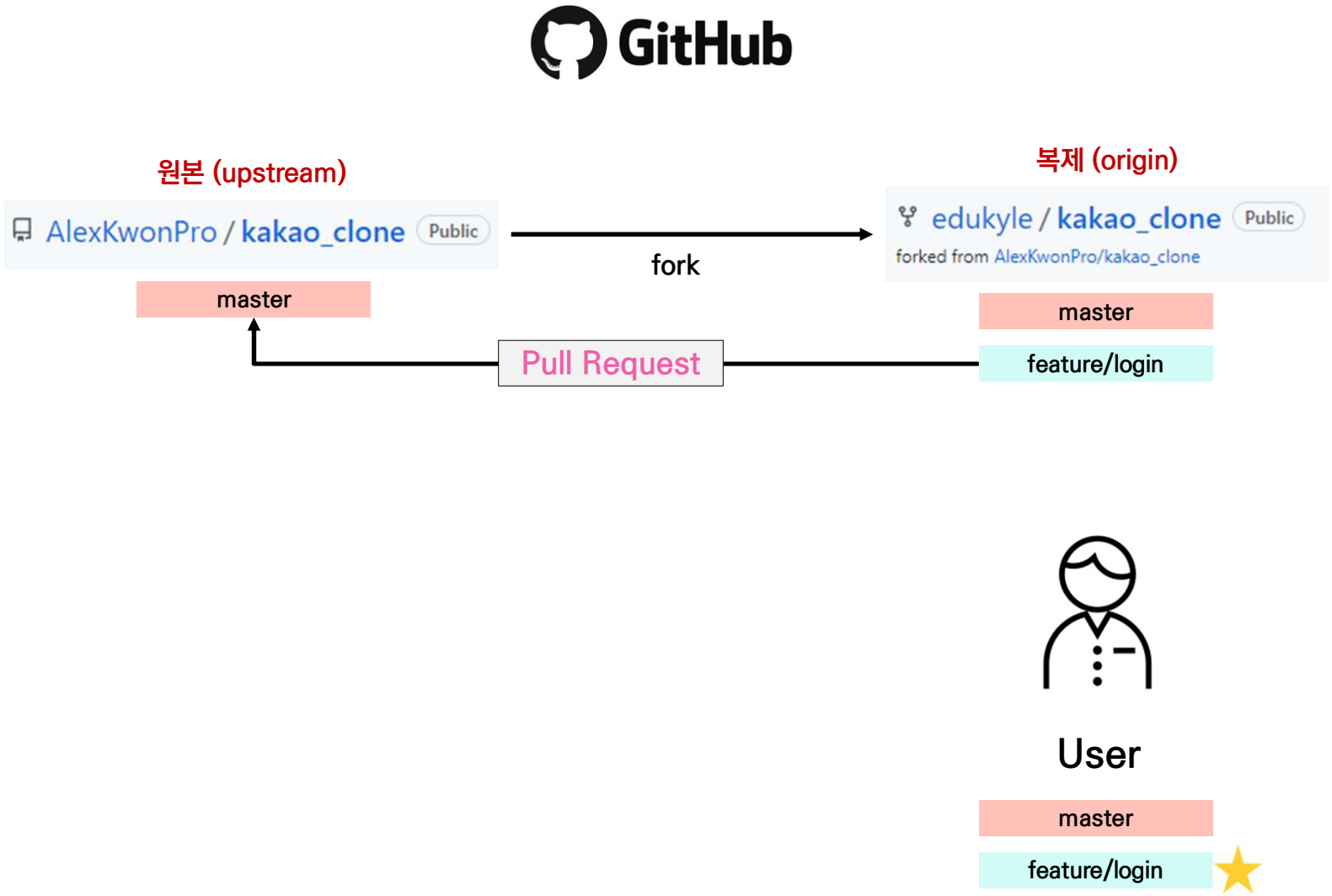
feature/login

User

master

feature/login

2) Forking Workflow (7 / 10)



2) Forking Workflow (8 / 10)

원본 (upstream)

AlexKwonPro / kakao_clone Public

병합 완료 master

fork

복제 (origin)

edukyle / kakao_clone Public

forked from AlexKwonPro/kakao_clone

master

feature/login 병합 완료 된 브랜치 삭제

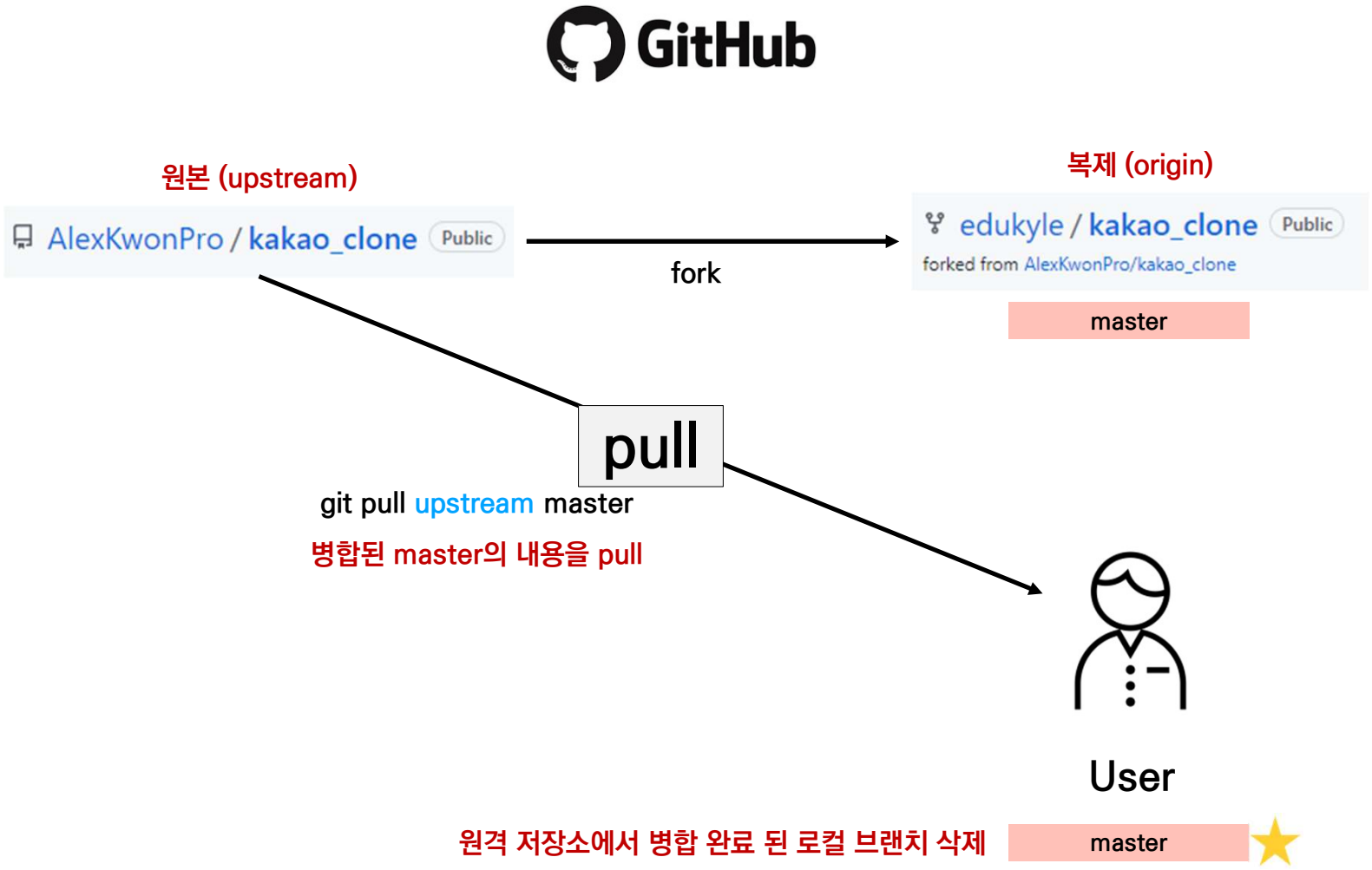
User

master 브랜치로 switch

master

feature/login

2) Forking Workflow (9 / 10)



2) Forking Workflow (10 / 10)

원본 (upstream)

AlexKwonPro / kakao_clone Public

fork

복제 (origin)

edukyle / kakao_clone Public

forked from AlexKwonPro/kakao_clone

master

User

master

새로운 기능 추가를 위해 branch 생성 및 과정 반복

feature/signup

Github Flow Models

앞서 설명된 기본 원칙 아래 Github에서 제시하는 방법이 2가지가 있다.

* Shared Repository Model

* Fork & Pull Model

이 두 모델의 가장 큰 차이점은 내(작업자)가 해당 프로젝트 저장소에 직접적인 push 권한이 있는지 여부!!

https://guides.github.com/