고용노동부 한국산업인력공단 국가인적자원개발컨소시엄

Git과 GitHub 입문

2023. 6. 21

백명숙

과정개요

■ 개인 프로젝트부터 현업 개발까지, 팀 개발에 필수적인 기술인 Git과 GitHub을 익힙니다.
Git CLI버전, GUI 버전 두가지 방식을 모두 사용합니다.

러닝 맵

Learning Object(학습모듈) 및 커리큘럼

LO	커리큘럼
Git과 버전관리	- 버전관리의 개념 - 환경설정
Git과 GitHub feat CLI	- Git Bash 활용 - 로컬저장소 생성 및 원격저장소에 올리기 - CLI로 Add, Commit, Push, Pull, Branch생성 하기
Git과 GitHub feat GUI	- SourceTree 활용 - GUI로 Add, Commit, Push, Pull 하기 - Branch 생성 및 Merge 하기 (CLI, GUI 두가지 방식) - Branch merge 할 때 conflict (충돌) 해결하기
Git의 추가 명령어	- rebase - amend - reset, revert - stash - cherry-pick
Fork와 PR(pull request)	- Repository Fork 하기 - Branch와 Fork 비교하기 - Pull Request(PR) 로 Contribution 하기

Git & GitHub 입문

백명숙(ms.baek@hucloud.co.kr)

왜 Git과 GitHub을 익혀야 하나요?

- 개발팀 프로젝트에서 버전 관리와 클라우드 저장소는 필수 불가결한 기술입니다.
- 버전 관리의 Git과 클라우드 저장소의 GitHub을 익히면 팀프로젝트와 회사에서 효율적인 개발 협업을 할 수 있습니다.

사전 조사

- 버전 관리가 무엇인지 들어 보기만 했다.
- Git 기본 기능을 써봤다.(commit, push, pull, merge, branch)
- Git의 동작원리를 알고 있다. (로컬 저장소, 스테이지, diff…)
- GitHub을 써봤다.(fork, pull request, code review)

버전 관리가 무엇인가요?

두명의 개발자가 하나의 서비스를 만들려면?

- 1. 각자 이름의 폴더를 만들고 각자 개발하다가 메일로 보내서 합친다.
- 2. 혹시 에러 날 수 있으니 백업본을 만들어 둔다.
- source_200101.zip / soure_200102.zip
- 3. 상대방이 작업중인 파일을 고치고 싶으면 메일에 따로 적어준다.
- 4. 코드를 합칠 때 이를 확인해서 내 코드에도 반영한다.

- **따로 조금씩 작업하다 내가 원할 때 코드를 합칠 수 있는 방법이 없나?**
  백업도 좀 쉽게 하고…=>

Git은 무엇인가?

- 깃(Git)은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 **분산 버전 관리 시스템**이다.
- 위키백과의 Git

깃


```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBY5Y7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 crate mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

저장소 생성, 파일 추가, 원격 동기화를 표시하는 명령줄 세션

원저자	리누스 토르발스[1]
개발자	주니오 하마노(Junio Hamano), 리누스 토르발스 등[2]

GitHub은 무엇인가?

- 깃허브(GitHub, 원래 이름: Logical Awesome LLC)는 분산 버전 관리 툴인 깃 (Git) 저장소 호스팅을 지원하는 웹 서비스이다.
- 위키백과의 GitHub

깃허브 (GitHub)

버전 관리 시스템 : Git

- 코딩할 때 단순히 ctrl+z를 눌러 이전 상태로 되돌리는 것이 아니라, 원하는 시점마다 깃발을 꽂고 (버전을 만들고) 이들 간에 자유롭게 돌아다닐 수 있다.

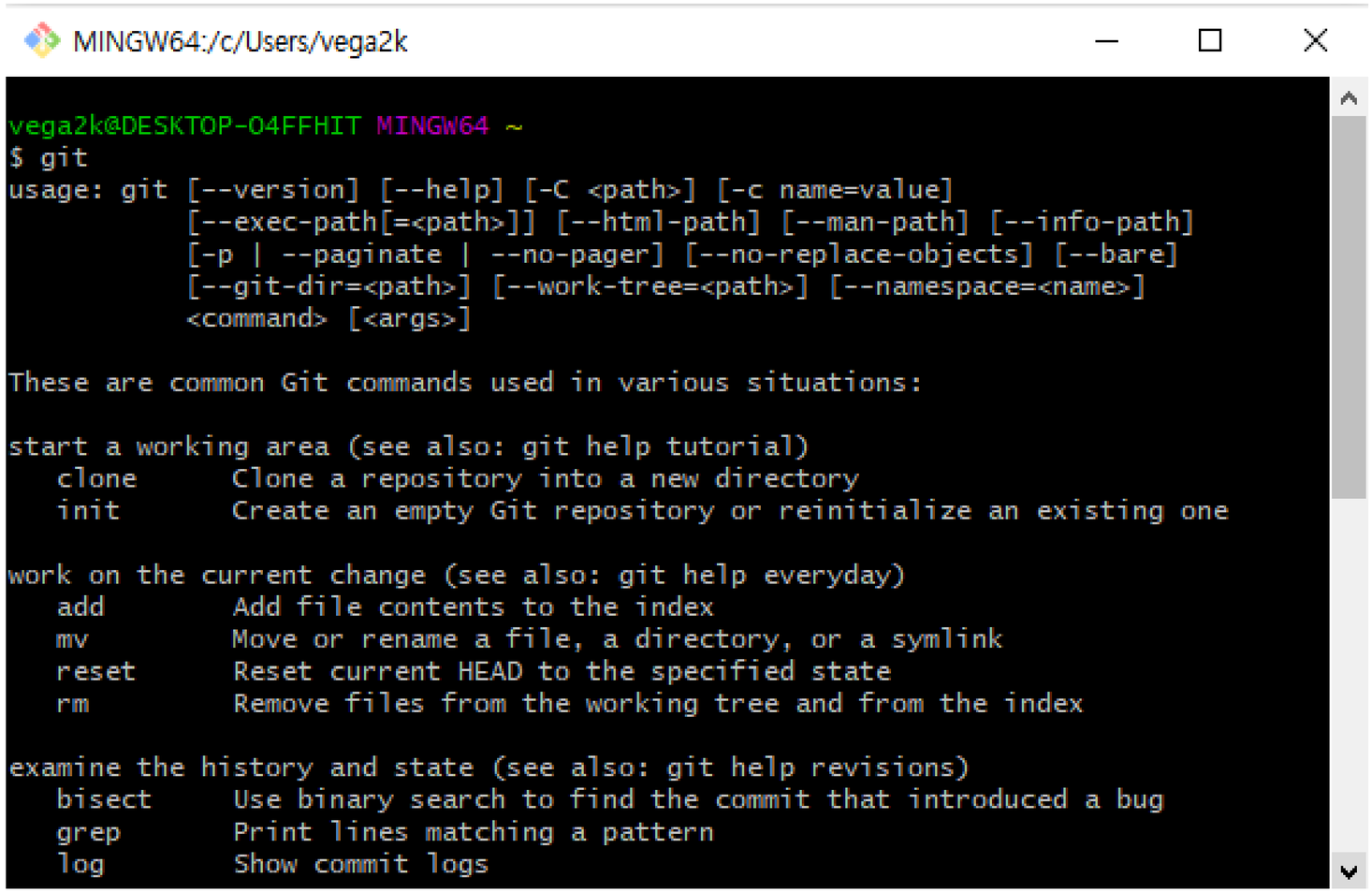
- 내가 만든 버전 뿐 아니라 동료가 만든 버전으로 이동할 수 있고, 동료와 내버전을 비교해서 최신본으로 코드를 업데이트를 할 수 있다.

Git을 사용하려면 무엇이 필요하나요?

- 저장할 공간만 있다면 어디서나 사용 가능
- 1. 개인 컴퓨터
- 2. 회사 서버
- 3. 클라우드(GitHub, BitBucket, GitLab …)

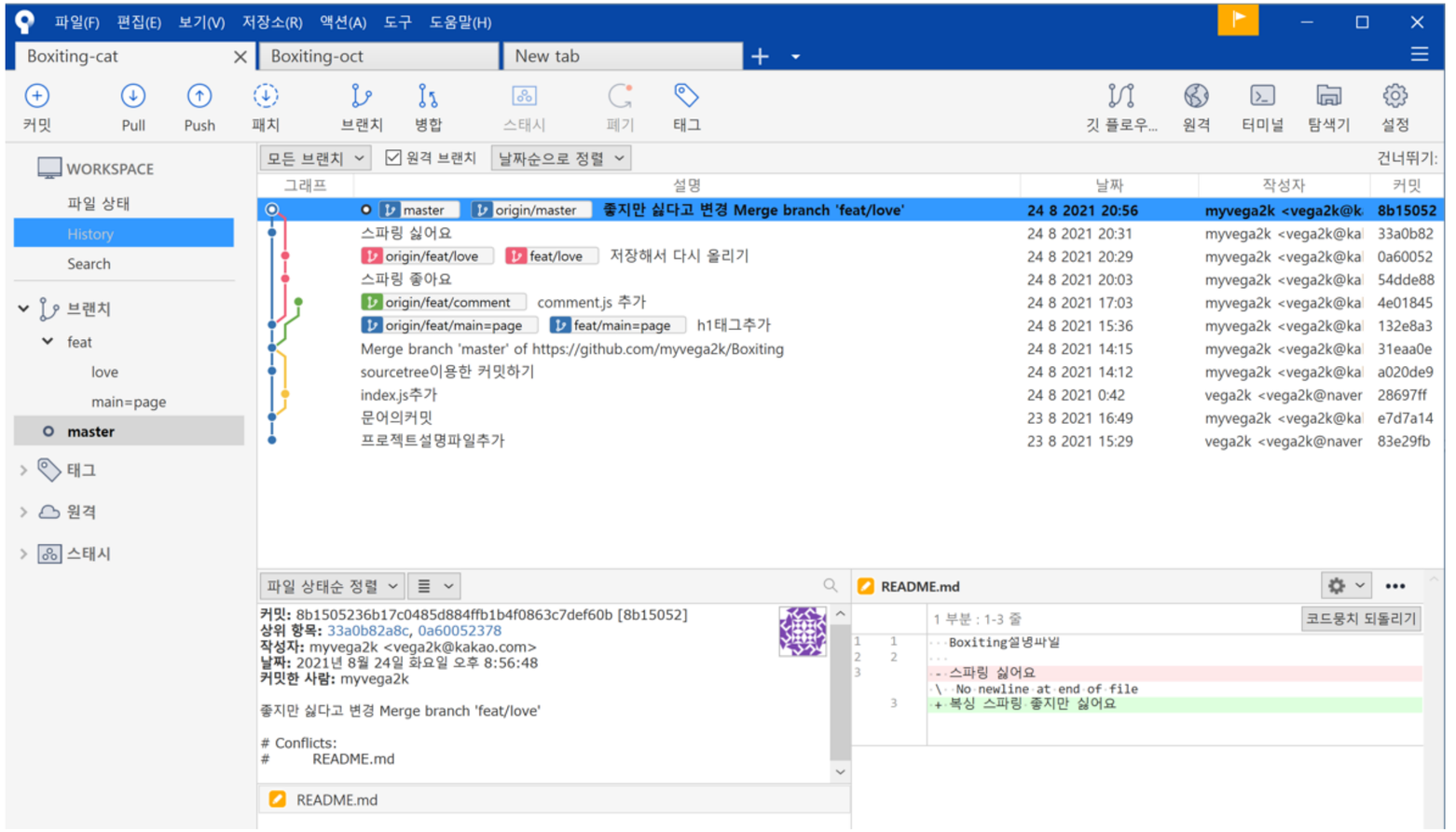
Git을 사용하는 두 가지 방법

- CLI 방식



Git을 사용하는 두 가지 방법

- GUI 방식



GitHub에 코드를 올리는 과정

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어 주기
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub 저장소 주소 알려주기
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기

GitHub에 코드를 올리는 과정

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령 `git init`
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택 `git add`
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어 주기 `git commit -m "첫페이지 제작"`
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub 저장소 주소 알려주기 `git remote add`
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기 `git push`

Git & GitHub 환경 설정하기

Git Bash 설치 (CLI 환경)

- Git 설치
- https://git-scm.com/downloads
- 명령프롬프트(Windows)/Terminal(Mac)에서 git 을 입력 해보고 내 컴퓨터에 Git이 설치 되어 있는지 확인하기

```
git
```

Visual Studio Code 설치

- VS Code 설치
- https://code.visualstudio.com/
- MS에서 제작한 무료 코드 에디터 Visual Studio Code 설치

  문법 강조, 코드 자동완성, Git 연동 등 유용한 기능이 많다.

GitHub 가입

- Git으로 버전 관리한 코드를 올릴 수 있는 클라우드 서버

Git 호스팅 사이트	모기업	특징	가격
GitHub	GitHub Inc (Microsoft에서 인수)	사용자 2,800만의 세계 최대 규모의 Git 호스팅 사이트	• 공개저장소 무료, • 비공개 저장소는 3인 이하인 경우는 무료 • 설치형 버전인 엔터프라이즈는 월21달러
GitLab	GitLab Inc	NASA, Sony 등 10만개 이상의 조직이 사용. GitLab 자체가 오픈소스인 특징	• 공개, 비공개 저장소 생성 무료
BitBucket	Atlassian	사용자 600만명. 지라(Jira)와 연동 쉽다	• 5명 이하 팀이면 공개 및 비공개 저장소 생성 무료

Git 초기화와 로컬 저장소

Git 용어 정의

1. Repository
  : Local Repository (로컬 컴퓨터의 .git 폴더), RemoteRepository (서버의 저장소)
2. Origin
  : Remote Repository를 Origin 이라고 부름
3. Master or Main
 : Branch들 중 가장 중요하고 핵심인 원본 Branch.
 : 기존에는 master라는 이름을 사용하였지만 깃허브가 main으로 기본 이름을 바꾸면서 main으로 사용함.
4. Working Tree(workspace)
 : Staging Area 이전의 단계 실질적으로 Source를 작성하는 폴더. Add 하기 이전의 공간
5. Index(Staging area)
 : Working Tree의 파일들이 Add 명령어를 통해 Staging Stage에 파일들을 올려야만 Commit이 가능함.
6. Commit
 : Staging Area의 파일들을 local repo에 버전으로 기록. Push 하면 해당 버전이 서버로 올라감.
7. HEAD
 : Local Repository의 작업 버전과 현재 branch 위치. 일종의 커서 역할.

Git 용어 정의

1. Working Tree와 Index

작업중인 Source 폴더를 'Working Tree'라고 부르며 Commit을 실행하기 전의 저장소와 Working Tree 사이에 존재하는 공간을 'Index'라고 합니다. Commit 작업은 Working Tree에 있는 변경 내용을 저장소에 바로 기록하는 것이 아니라 그 사이 Index에 파일 상태를 기록 (Staging 한다고 표현)하게 되어 있습니다. 저장소의 변경 사항을 기록하기 위해서는, 기록하고자 하는 모든 변경 사항들이 'Index'에 존재해야 합니다. 하지만 Index라는 공간이 있기 때문에 작업 트리 안에 있는 Commit이 필요없는 파일들을 Commit에 포함하지 않을 수 있고, 원하는 변경 사항에 대해서만 Commit 할 수 있습니다.

2. PUSH

: push를 실행하면, 원격 저장소에 내 변경 이력이 업로드 되어, 원격 저장소와 로컬 저장소가 동일한 상태가 됩니다.

3. Clone

: Remote Repository를 Local Repository에 복사하는 명령어

4. Pull

: Remote Repository의 최신 변경 사항을 Local Repository와 합치는 명령어.

5. Fetch

: 정보를 받아오고 Merge까지 해주는 Pull과 달리 Origin의 main Branch "정보만" 받아옴.

GitHub에 코드를 올리는 과정 (again)

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령 `git init`
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어주기
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub저장소 주소 알려주기
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기

이 폴더에서 Git으로 버전관리를 하고 싶어요!

- 로컬 저장소 생성

이 폴더에서 Git으로 버전관리를 하고 싶어요!

- 1.원하는 폴더에서 Git 초기화를 하면 그때부터 가능하다.

```
git  init
```

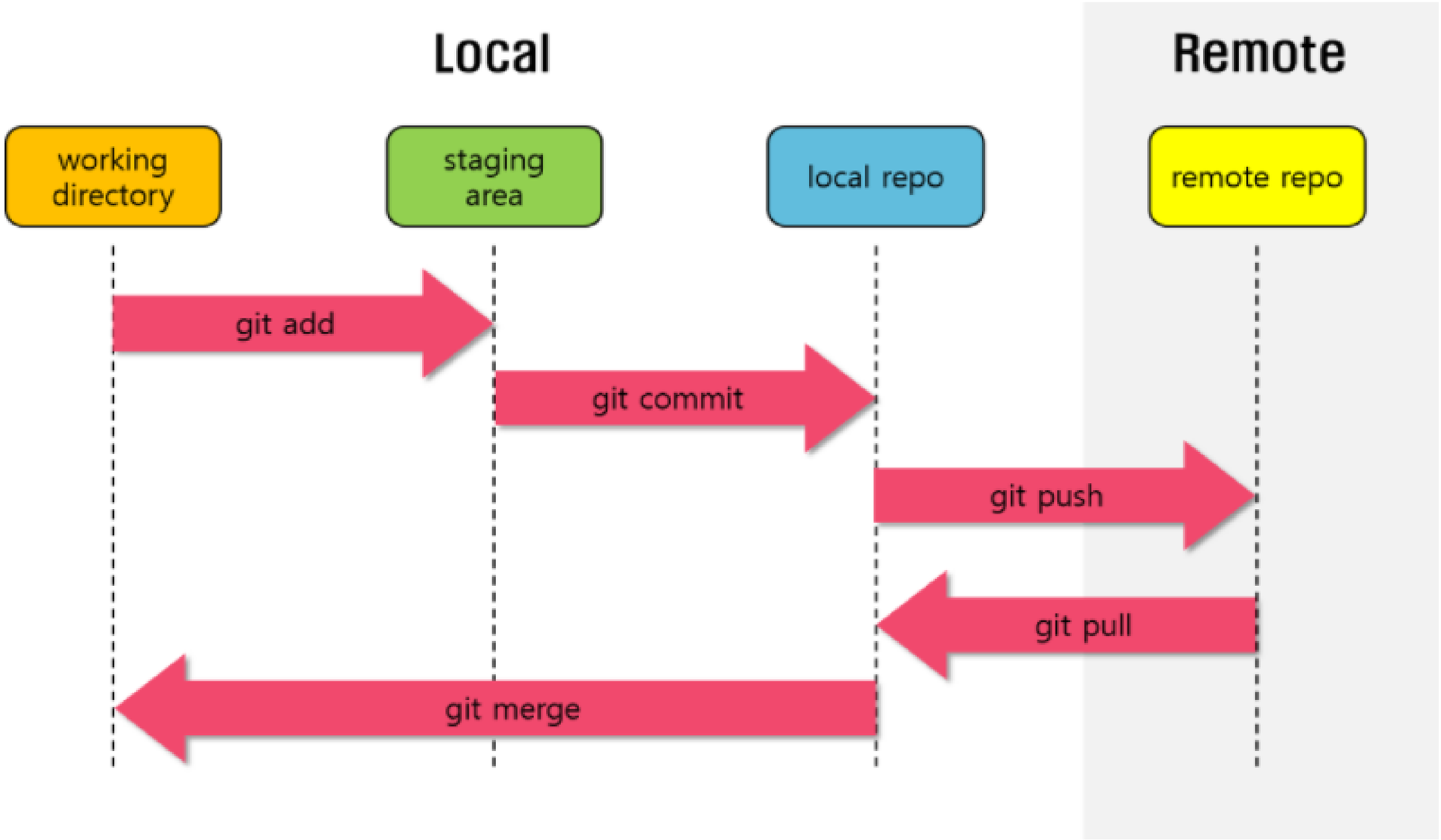
- 2.Git 초기화를 하면 .git 이라는 숨겨진 폴더가 로컬저장소이다.
- 3.로컬저장소에 내가 만든 버전 정보, 원격 저장소 주소 등이 저장된다.
- 4.원격저장소에서 내 컴퓨터로 코드를 받아오면 로컬저장소가 자동으로 생성된다.
- 5.한 폴더에 하나의 로컬저장소만 유지해야 한다.

로컬 저장소 생성 실습

- 1. 내 컴퓨터에서 MyApp-Octocat 폴더 생성
- 2. Git Bash로 생성한 폴더로 이동하기
- 3. git init 으로 로컬 저장소 생성

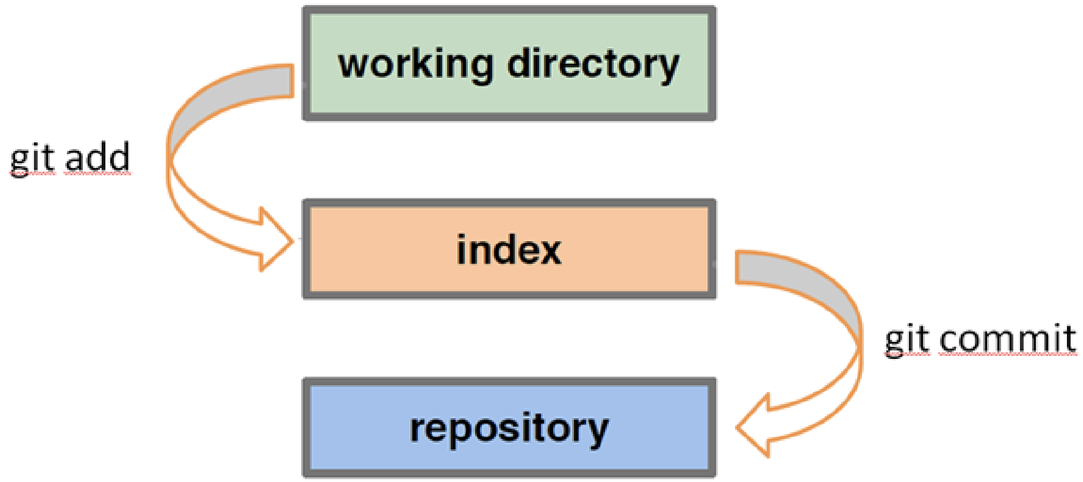
첫 번째 버전 만들기

Git Workflow



Git Workflow

- Working directory : 버전으로 만들어 지기 전 파일을 수정하는 곳
- Staging area(Index) : 버전 관리를 원하는 파일을 모으는 곳
- Repository : 만들어지는 버전들이 최종적으로 모이는 곳



GitHub에 코드를 올리는 과정 (again)

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택 git add
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어주기 git commit -m "첫페이지 제작"
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub저장소 주소 알려주기
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기

덩어리가 뭐예요? : 커밋(commit) = 하나의 버전

1. 페이지 1,2,3 작성

덩어리가 뭐예요? : 커밋(commit) = 하나의 버전

1. 페이지 1,2,3 작성

commit

옥토캣 1
페이지 1, 2, 3

2. 페이지 2 수정

commit

옥토캣 2
페이지
1, 2(수정본), 3

덩어리가 뭐예요? : 커밋(commit) = 하나의 버전

1. 페이지 1,2,3 작성

commit

옥토캣 1
페이지 1, 2, 3

최신버전이 이상하네..
옥토캣1 버전으로
돌아가서 다시 시작
해야지

2. 페이지 2 수정

commit

옥토캣 2
페이지
1, 2(수정본), 3

Git 사용자 정보 등록

- 1.Git 전역 사용자 설정(GitHub의 username과 email을 입력하세요).

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
git config --list
```

- 2.Git 전역 사용자 설정 초기화.

```
git config --global --unset user.name
git config --global --unset user.email
```

버전 생성 실습

- 1. VS Code에서 README.md, index.html 파일 생성
- 2. 원하는 파일만 추가하기

```
git  add README.md
```

- 3. 메시지를 달아서 커밋하기

```
git commit -m "프로젝트 설명 파일 추가"
```

- 4. 생성한 커밋 목록보기

```
git  log
```

커밋에 대하여

- 1. 커밋은 '의미있는 변동사항'을 묶어서 만든다.
- 2. 버튼 클릭 버그를 고치는데 5가지 파일을 수정했다면 5가지 파일들을 묶어서 하나의 커밋으로 만든다.
- 3. 동료개발자(혹은미래의나)가 '버튼클릭버그'를 고치는데 어떤 파일을 수정했는지 손쉽게 파악 가능하다.
- 4. 커밋 메시지 적는게 귀찮아도 시간을 들여서 자세하게 적어주세요.

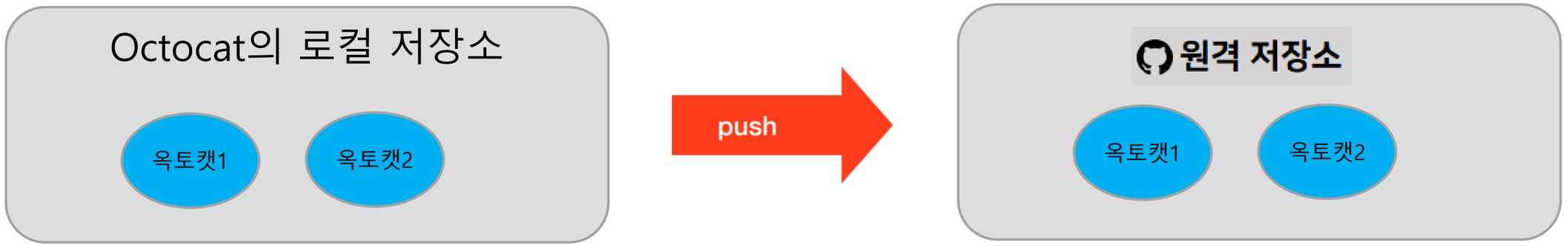
만든 버전 GitHub에 올리기

GitHub에 코드를 올리는 과정(again)

- 1. 내 컴퓨터 프로젝트 폴더에 '여기에서 Git을 쓰겠다!' 라고 명령
- 2. 즐겁게 코딩
- 3. 내가 변경한 파일 중 올리길 원하는 것만 선택
- 4. 선택한 파일들을 한 덩어리로 만들고 설명 적어주기
- 5. GitHub 사이트에서 프로젝트 저장소(Repository) 만들기
- 6. 내 컴퓨터 프로젝트 폴더에 GitHub 저장소 주소 알려주기 git remote add
- 7. 내 컴퓨터에 만들었던 덩어리 GitHub에 올리기 git push

로컬 저장소와 원격 저장소

- 내 컴퓨터의 로컬 저장소에서 버전 관리가 완벽하게 되고 있어.
- 이제 GitHub에 올려서 다른 사람들과 함께 버전 관리를 할래!



원격 저장소 GitHub에서 만들고 커밋 푸시하기

- 1. GitHub에 로그인해서 MyApp-Octocat 저장소 생성
- 2. 내 컴퓨터 MyApp-Octocat 폴더에 GitHub 저장소 주소 알려주기

```
git remote add origin https://github.com/아이디/이름.git
```

- 3. 만든 커밋 푸시하기

```
git push origin master
```

- 4. GitHub 사이트에서 올라간 커밋 확인

다른 사람이 만든 저장소 받아오기

원격 저장소를 내 컴퓨터에 받아오기: 클론(clone)

- 클론(clone)을 하면 원격저장소의 코드를 내 컴퓨터에 받아올 수 있습니다.
- 로컬 저장소(.git폴더)도 자동으로 생깁니다.

원격 저장소를 내 컴퓨터에 받아오기: 푸시(push)

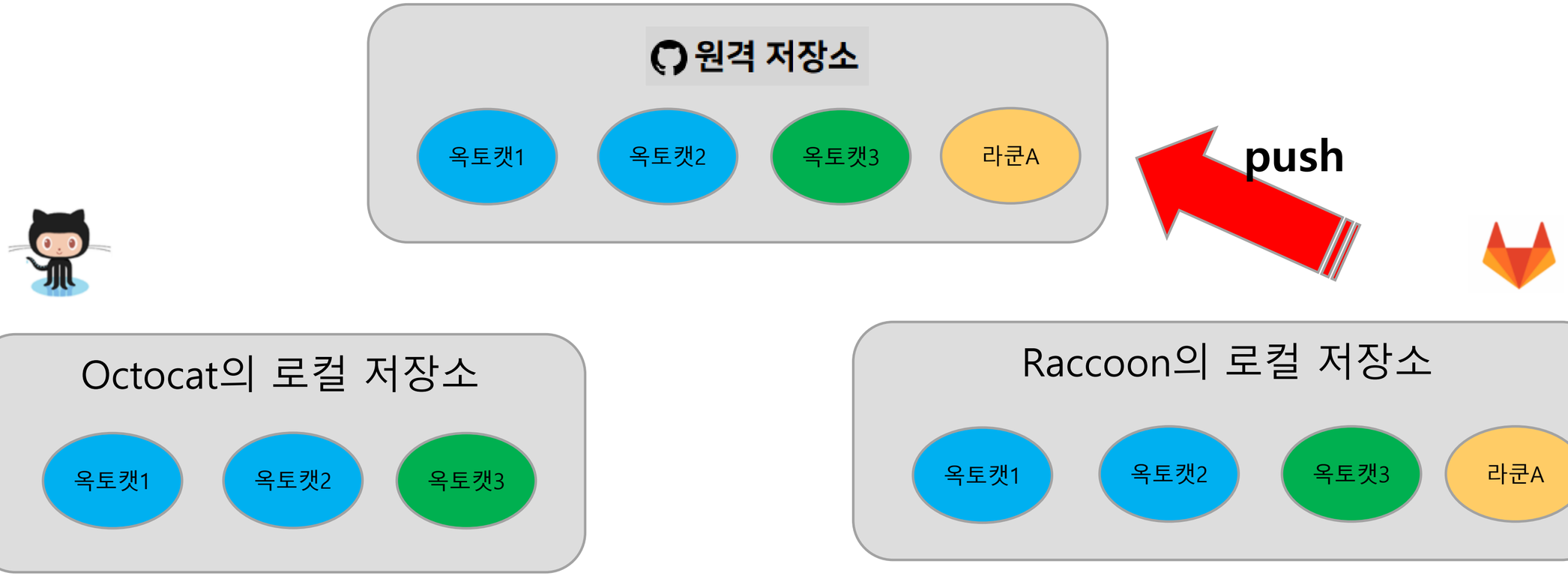
- 옥토캣이 새로운 버전 '옥토캣3'을 만들어 원격저장소에 push 했어요.

원격 저장소를 내 컴퓨터에 받아오기: 풀(pull)

- 라쿤은 업데이트 된 데이터는 풀(pull) 명령어로 받아올 수 있습니다.

원격 저장소를 내 컴퓨터에 받아오기: 푸시(push)

- 라쿤도 커밋을 만들어서 원격저장소로 푸시(push)를 할 수 있습니다.
  (원격 저장소에 푸시 권한이 있을 경우)



GitHub 저장소를 내 컴퓨터에 받아오기: 클론(Clone)

- 1. 내 컴퓨터에 MyApp-Raccoon 폴더를 만들고
  GitHub의 MyApp-Octocat 저장소 받아오기

```
git clone https://github.com/아이디/이름.git .
```

점을 찍어줘야 현재 폴더에 내려받습니다.
안 찍으면 새 폴더 생성!

- 2. app.js 파일 생성 후 add -> commit -> push
- 3. GitHub에서 새 커밋 확인하기

GitHub 저장소의 변경사항 내 컴퓨터에 받아오기

- 1. 내 컴퓨터에 MyApp-Octocat 폴더로 이동 후 app.js 없는 것을 확인
- 2. pull 명령어로 라쿤이 새로 올린 커밋 받아오기

```
git pull origin master
```

- 3. 새로 생긴 app.js 확인하기

Git의 원리

Git에서의 commit 이란?

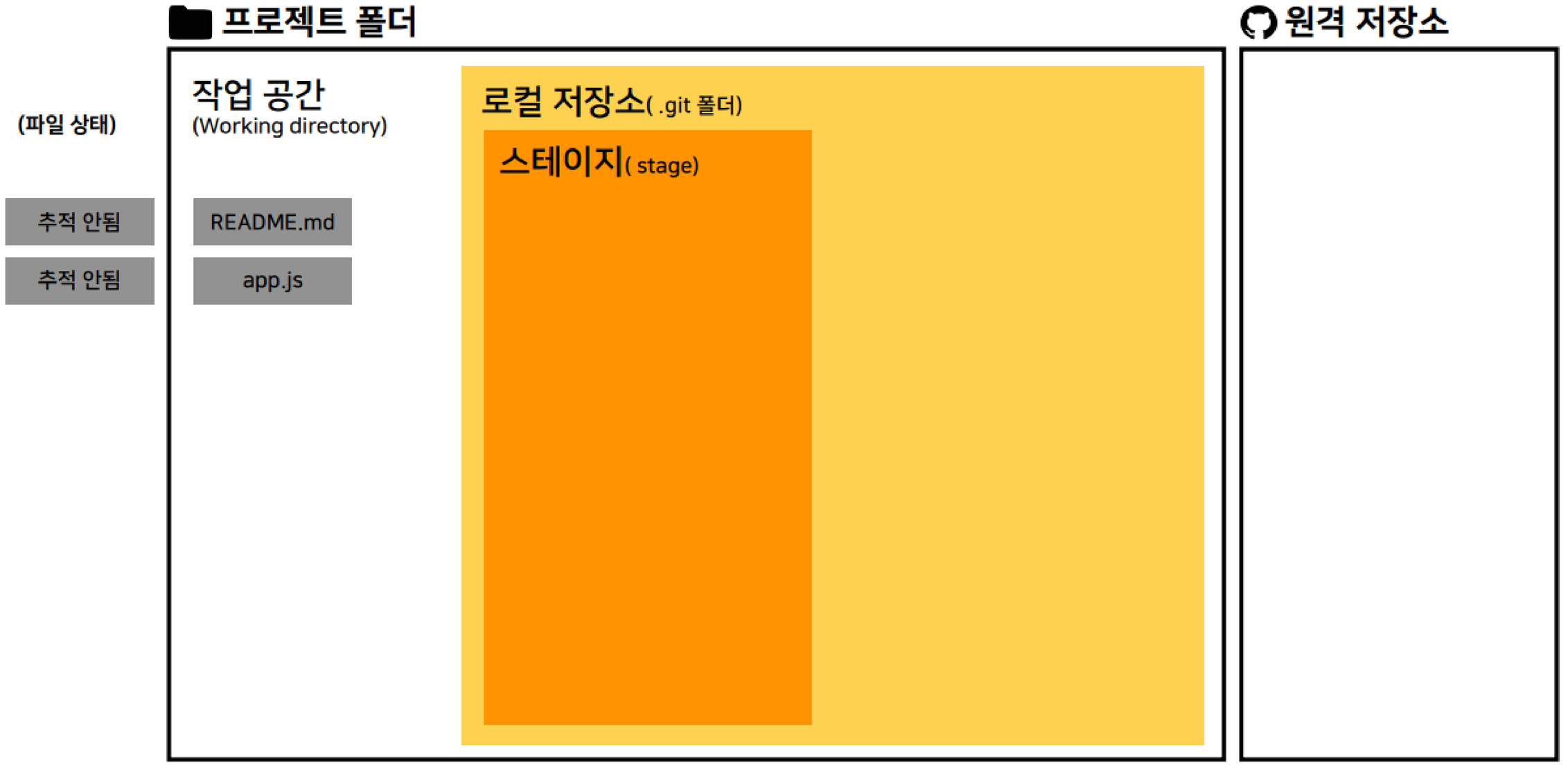
- 1. 버전은 의미 있는 변화를 뜻하며, 작업이 완결된 상태이어야 합니다.
- 2. 이렇게 의미 있는 변화에 대하여 기록하는 것이 바로 commit 입니다.

- **git add를 사용하는 이유**

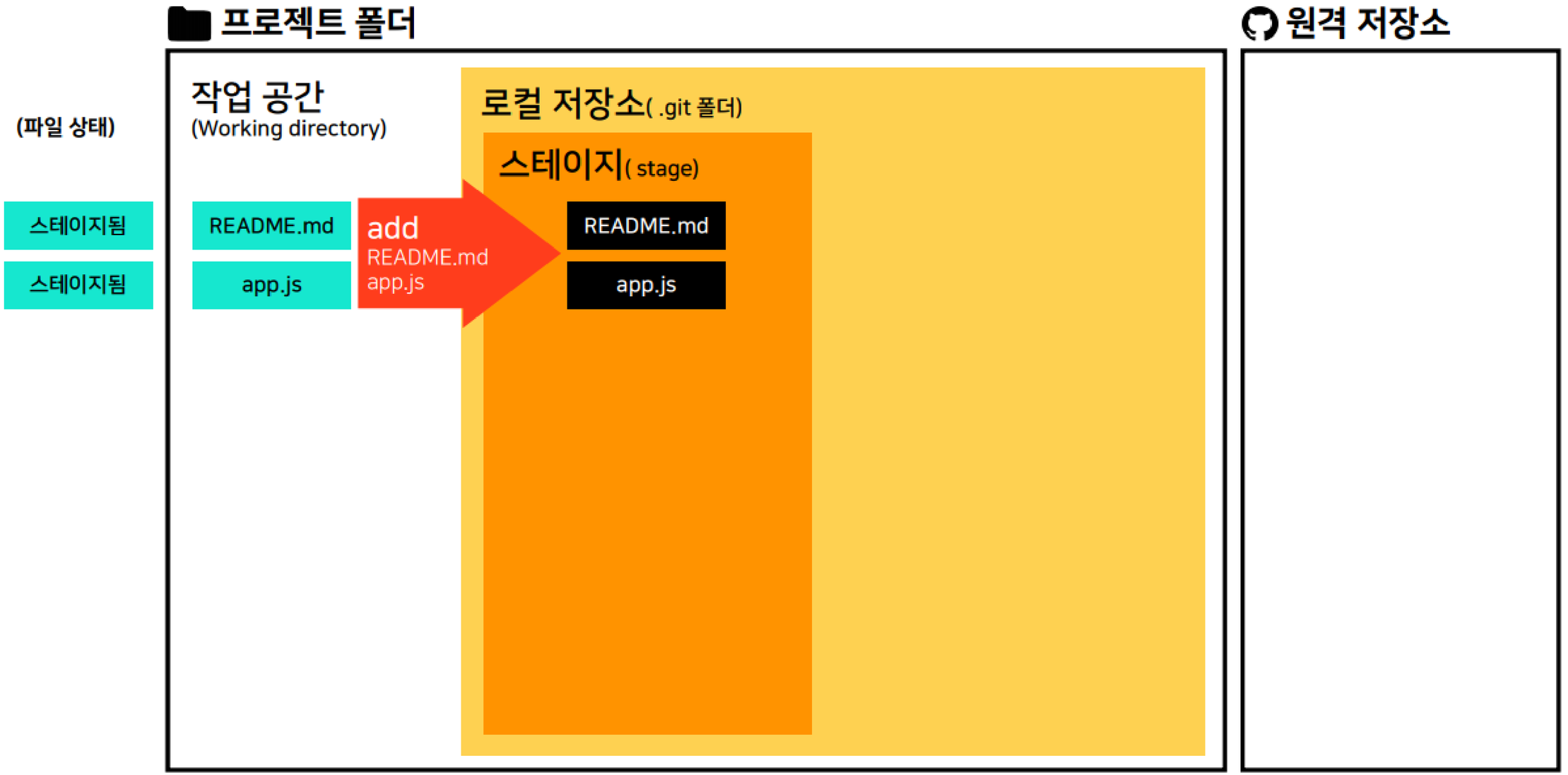
: git add 명령어는 원하는 기능만 add 해서 그것들만 따로 commit을 할 수 있습니다.

: add를 해서 commit 단계 이전의 공간을 stage area 라고 합니다.

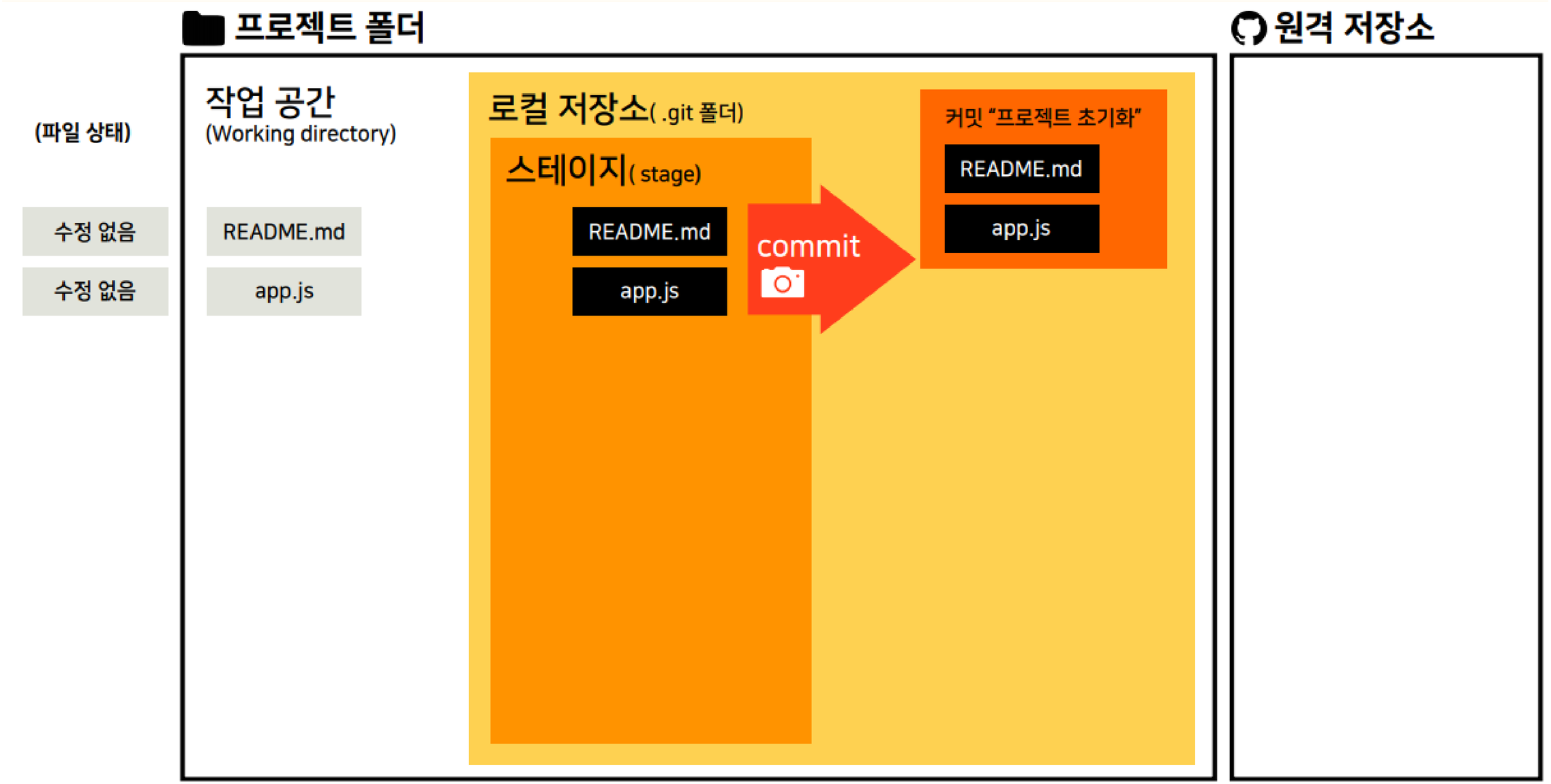
0. 맨 처음 로컬 저장소를 만들었을 때



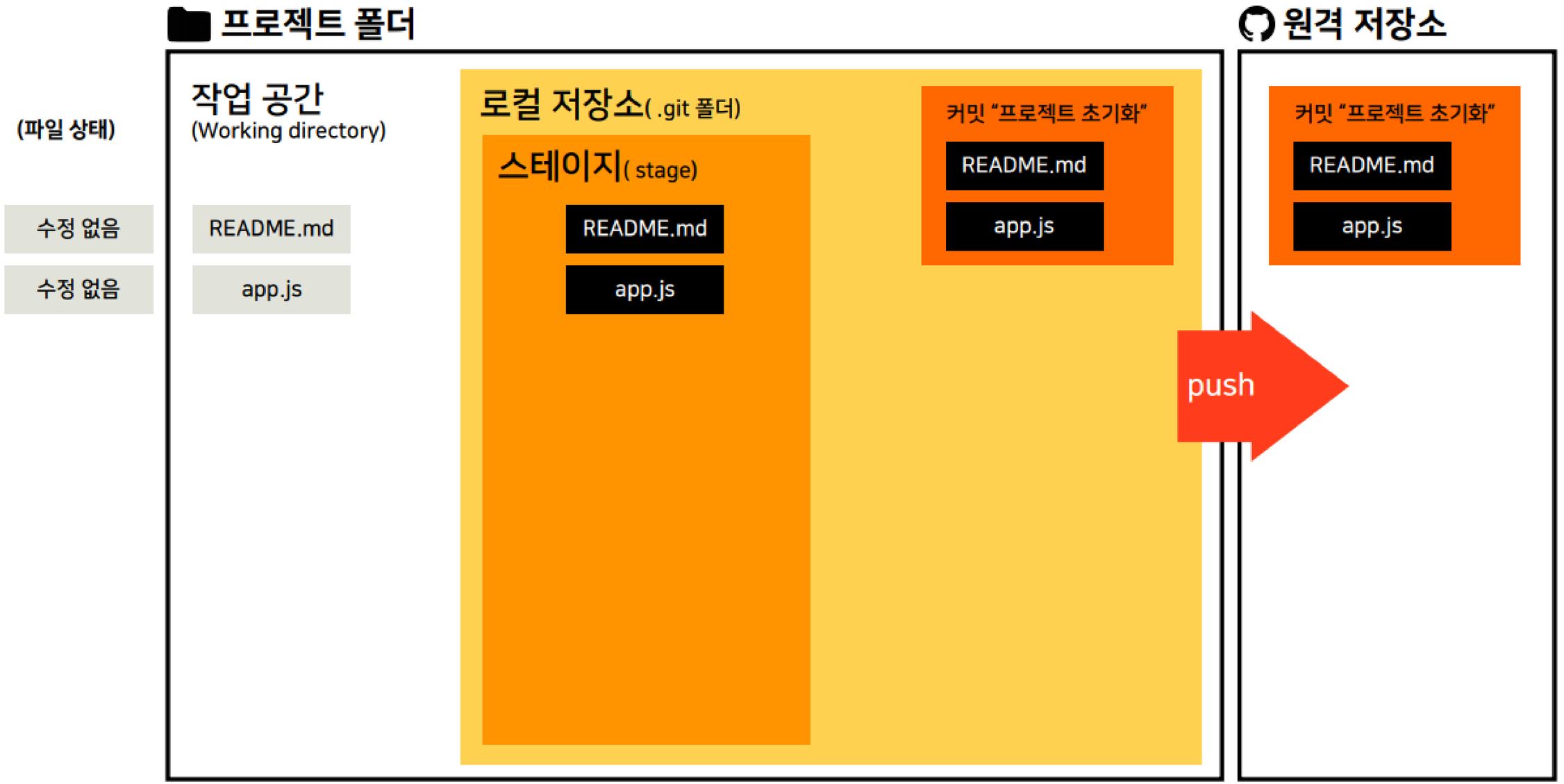
1. 두 파일 스테이지에 올리기 git add



2. 스테이지 사진 찍어 남기기 : commit

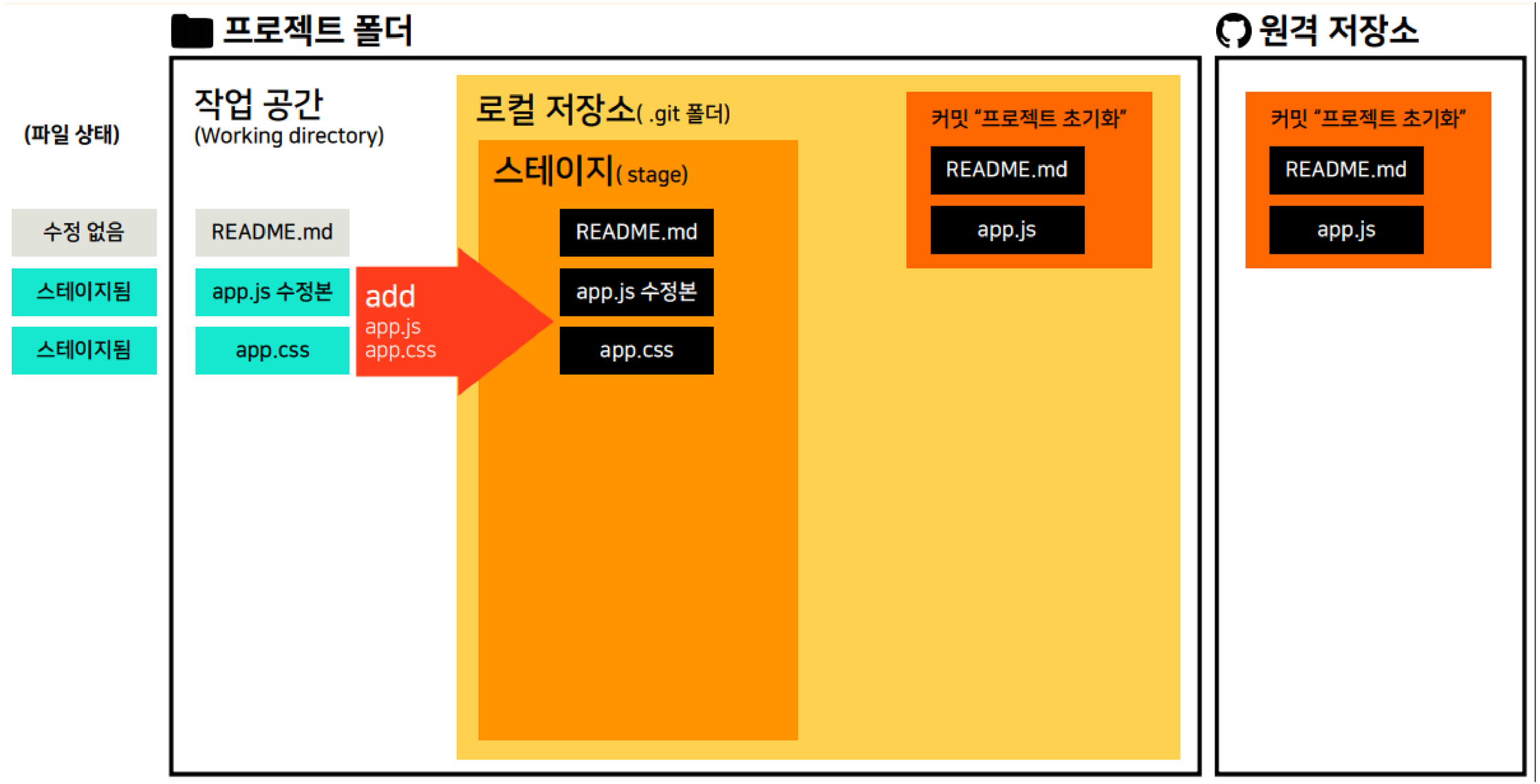


3. 커밋을 원격 저장소에 올리기 : push

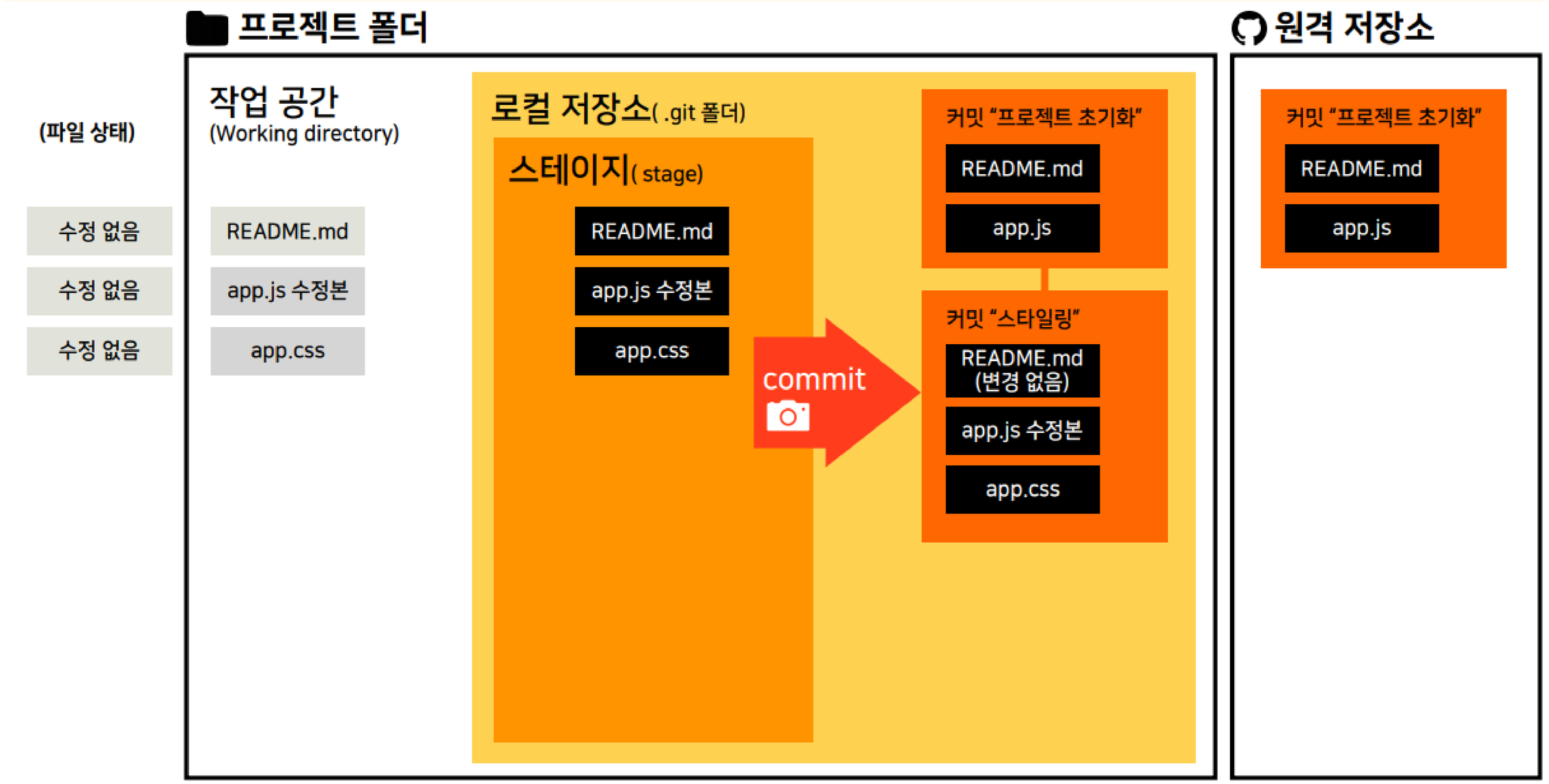


4. app.js 수정, app.css 추가

4. app.js 수정, app.css 스테이지에 올리기



5. 스테이지 사진 찍어 남기기: 커밋



6. 커밋을 원격 저장소에 올리기: 푸시

요약

- 1. Git으로 추적하는 파일의 4가지 상태

- 2. 작업공간(Working Directory)에 있는 수정함 추적 안됨 파일을 스테이지로 올려서 스테이지됨 으로 변경한다.
- 3. 커밋을 하면 수정 없음 상태로 돌아가서 다시 파일을 수정할 수 있다.

Git로 add, commit, push, pull 하기

Git에서의 커밋이란?

- 1. MyApp-Octocat 폴더의 파일 수정 후 add, commit, push 하기
- 2. MyApp-Raccoon 폴더를 소스트리에 추가하고 pull 받아오기

간단한 협업 시나리오: 7가지 개념 리뷰

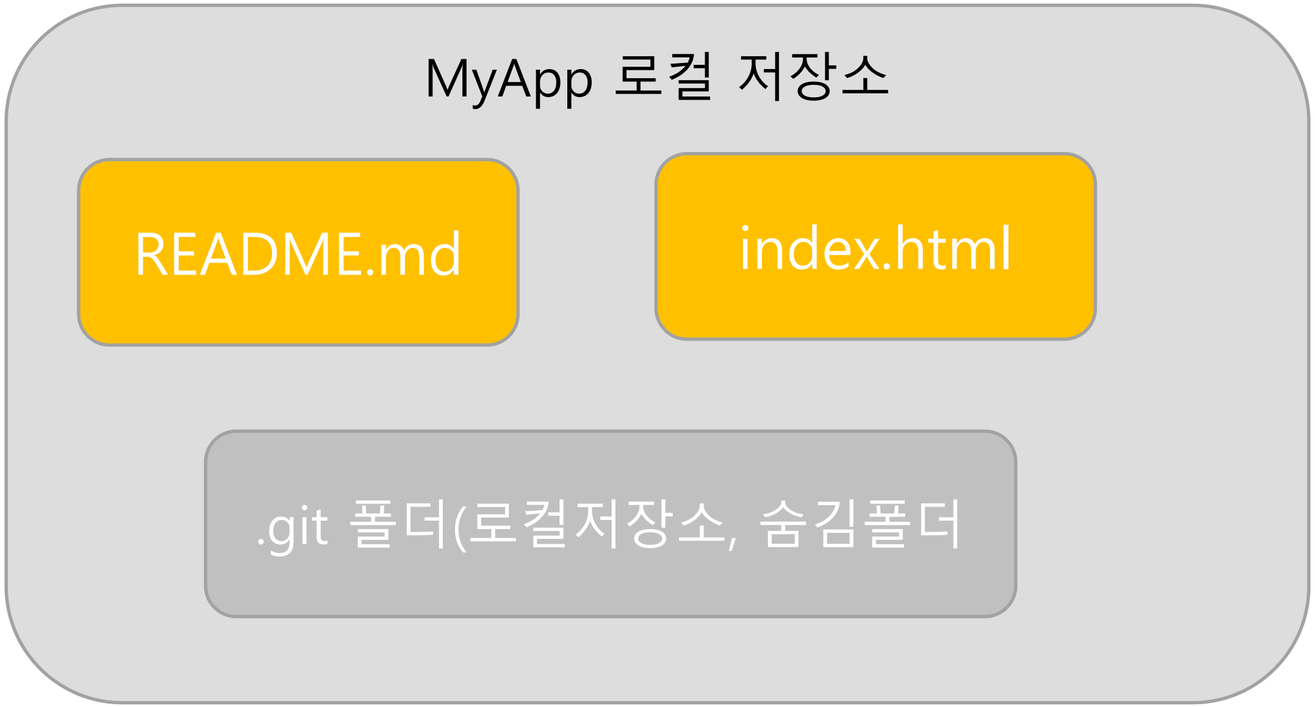
7가지 개념 리뷰

- 1. 해커톤에 참가한 개발자 옥토캣은 개발자 라쿤과 짝이 됩니다.

7가지 개념 리뷰: 로컬 저장소 생성

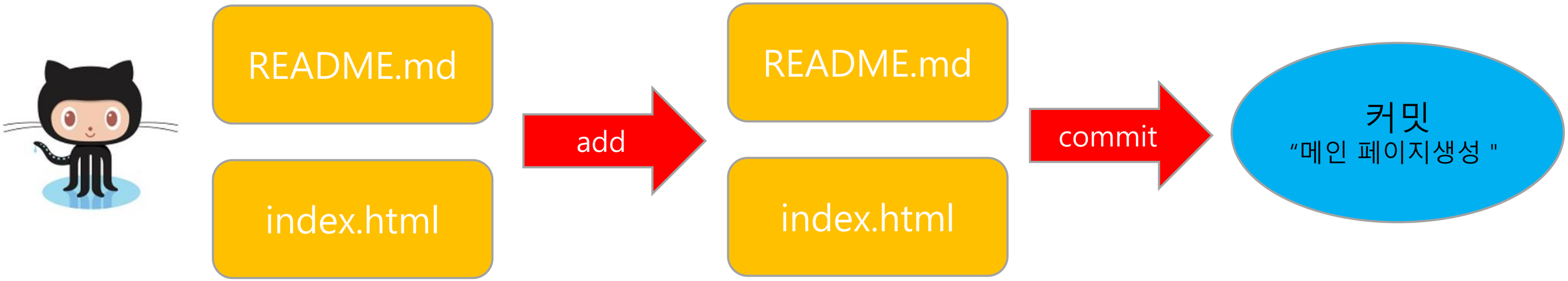
- 2. 옥토캣이 먼저 프로젝트 폴더를 만듭니다.

  먼저 컴퓨터에 MyApp 폴더를 만들고, `git init` 으로 로컬 저장소를 생성합니다.



7가지 개념 리뷰: add, commit

- 3. 옥토캣이 먼저 메인 페이지 제작을 맡기로 했습니다.
  VS Code에서 파일을 만들어 모두를 git add 한 다음 “메인 페이지 생성”
  이라는 커밋을 만들었습니다. git commit -m “첫페이지 제작”



7가지 개념 리뷰: remote add, push

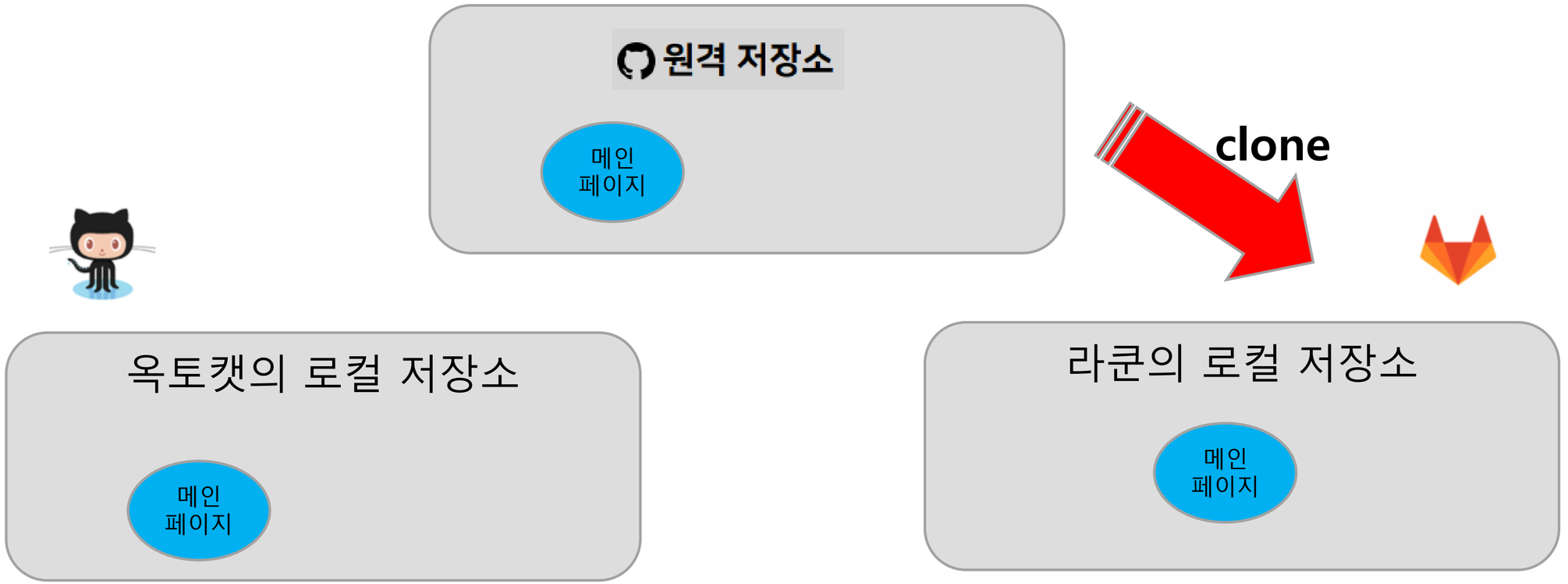
- 4. GitHub에 원격 저장소를 만들어 원격주소를 로컬 저장소에 알려준 후에

  `git remote add origin 저장소주소`

  로컬 저장소에서 만든 커밋을 원격 저장소에 푸시 합니다. `git push`

7가지 개념 리뷰: clone

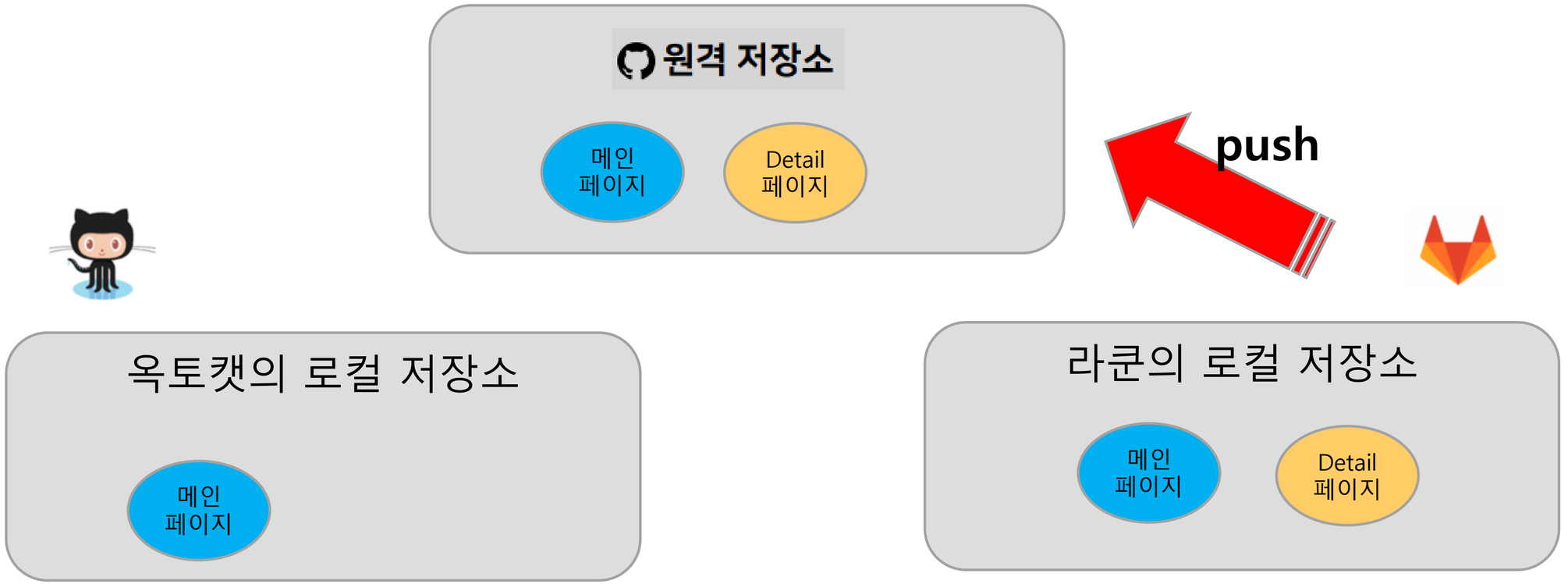
- 5. 라쿤도 개발을 시작하기 위해 옥토캣이 올려준 원격 저장소를 본인 컴퓨터에 받아 옵니다. git clone 저장소주소



7가지 개념 리뷰: push

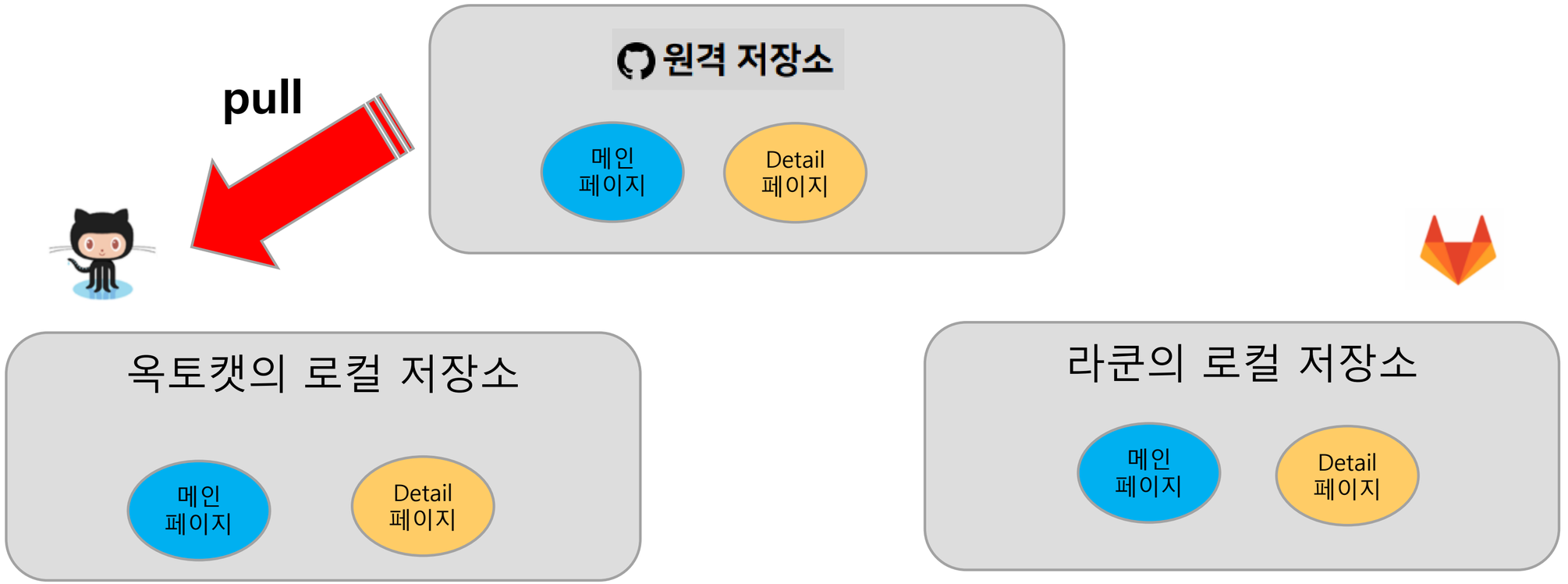
- 6. 라쿤은 Detail 페이지 제작을 맡았습니다.

  GitHub 저장소에 라쿤 계정을 Collaborator로 추가한 후 새 커밋을 push 합니다.



7가지 개념 리뷰: pull

- 7. 라쿤이 새로 추가한 “Detail 페이지” 커밋을 옥토캣 컴퓨터에 받아오고 각자 개발한 버전(커밋)을 언제든지 원격저장소에 올리고 받아 올 수 있습니다.



SourceTree GUI로 Git 다지기

궁금한 점들

- 1. 커밋 객체에는 무엇이 저장 되나요?
- 2. 두 사람이 병렬로 커밋을 만들고 싶으면 어떡하나요?
- 3. 두 사람이 만든 버전을 합칠 수 있나요?
- 4. 다른 사람이 만든 오픈소스에는 어떻게 기여 할 수 있나요?

추가적으로 학습할 내용들

- 1. Git GUI 소스트리로 로컬 저장소 추가하기
- 2. 애드(Add)와 커밋(Commit)이 무엇인지 스테이지 개념과 함께 이해하기
- 3. 브랜치(Branch)로 평행세계 나누기
- 4. 머지(Merge)로 두 브랜치 합치기
- 5. 두 브랜치가 충돌(Conflict) 났을 때 해결하기
- 6. 예의 바른 병합 요청(Pull Request) 보내기
- 7. 다른 사람의 저장소 통째로 복제하기: 포크(Fork)

SourceTree 설치 - GUI 환경 구축

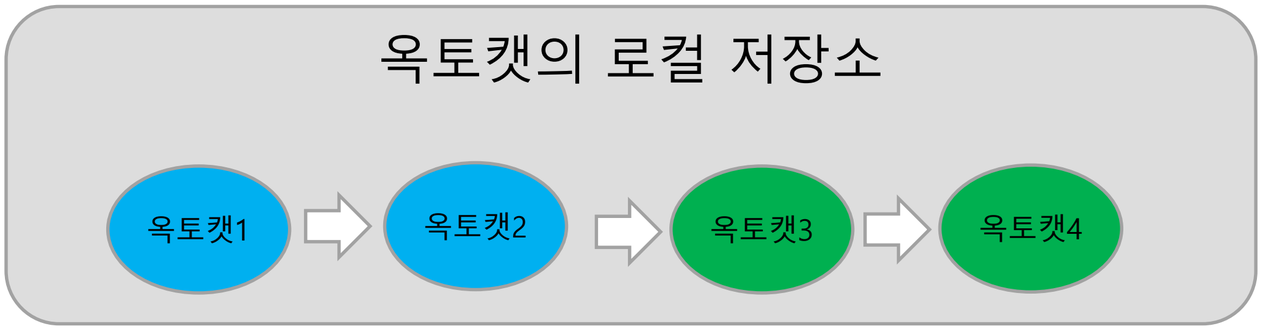
- 1. 버튼 클릭으로 Git 명령을 실행할 수 있는 도구, 소스트리 설치

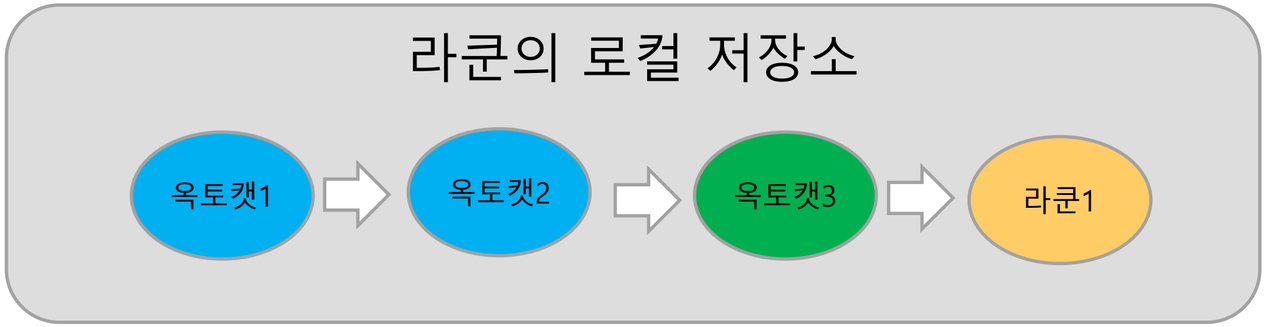
  Git 개념을 시각적인 그래프로 볼 수 있어 편리하다.

  https://www.atlassian.com/ko/software/sourcetree
- 2. 설치한 소스트리에 내 컴퓨터에서 이미 만든 로컬 저장소 추가하기

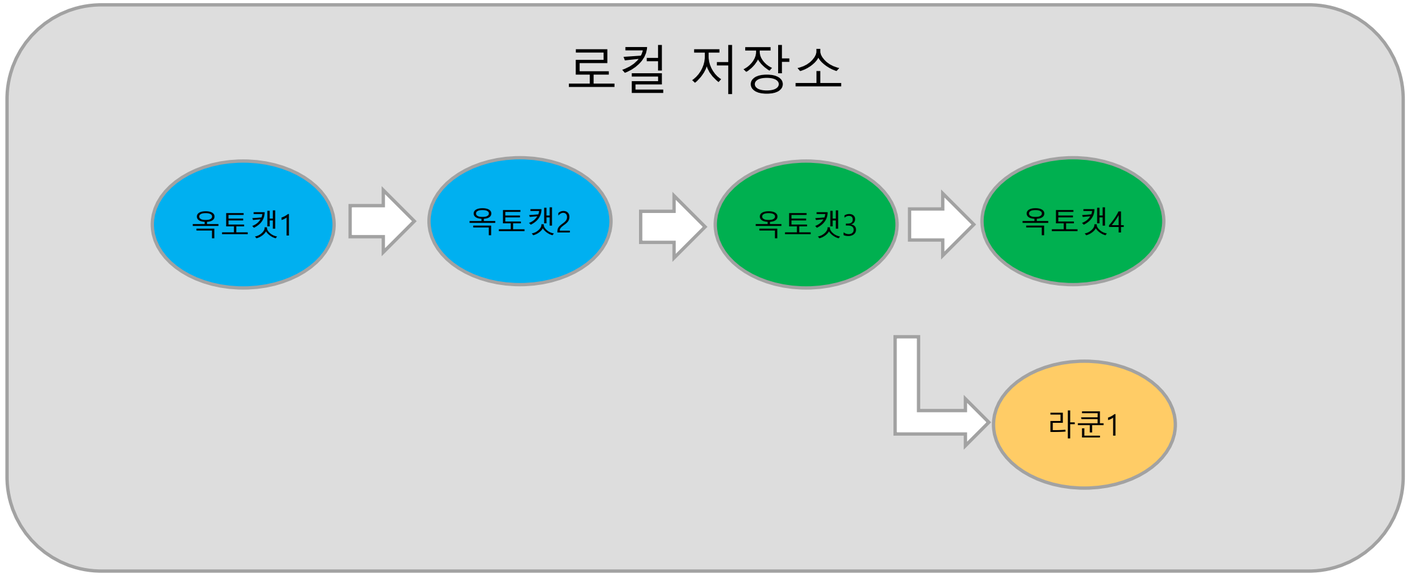
평행세계 나누기: 브랜치(branch)

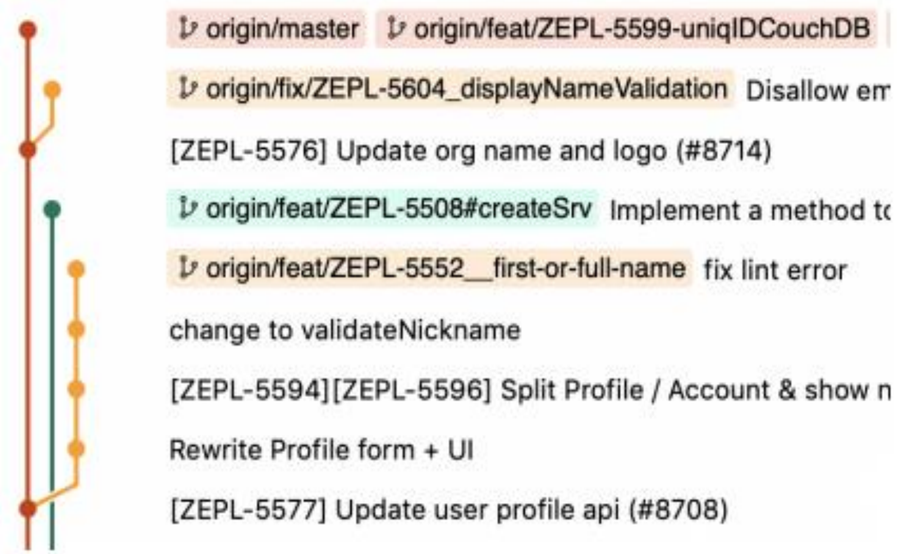
Problem: 한 줄로 커밋을 쌓으면 둘이 겹치지 않나요?





Solved: 여러 줄로 쌓으면 됩니다.



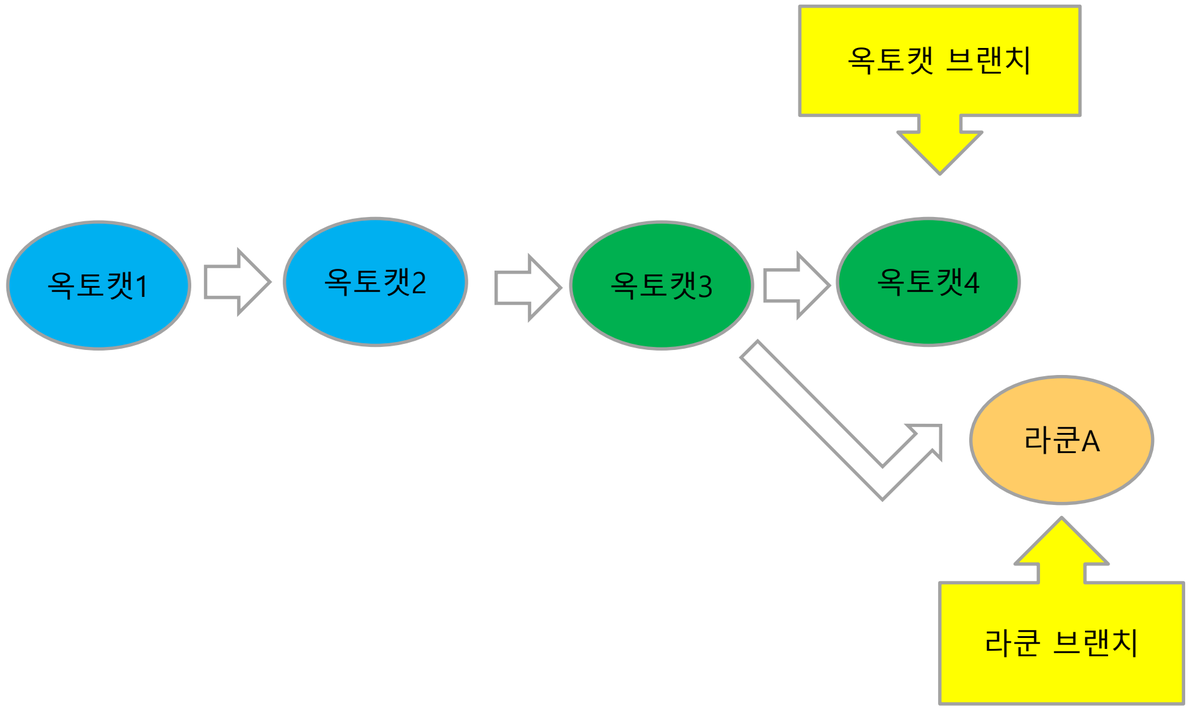


여러 줄로 커밋을 쌓는다?

- Q1. 왜 같이 작업 하려면 여러 줄로 커밋을 쌓아야 하나요?
- A1. 한 줄에서 작업하면 충돌이 날 수 있습니다. 똑같은 코드를 동시에 고칠 가능성이 있죠.

- Q2.그럼 n줄로 쌓고 나중에 합치나요?
- A2.네.그럼 충돌이 나더라도 합치는 시점에 명시적으로 충돌을 해결할 수 있습니다.

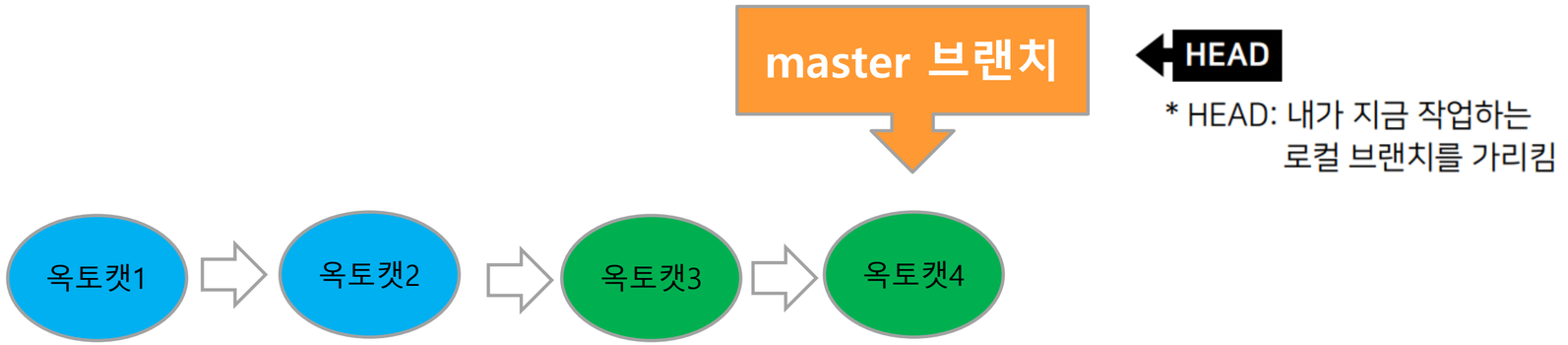
브랜치(Branch – 가지) 개념



사실 이미 브랜치는 있었어요.

```
git push origin master
```

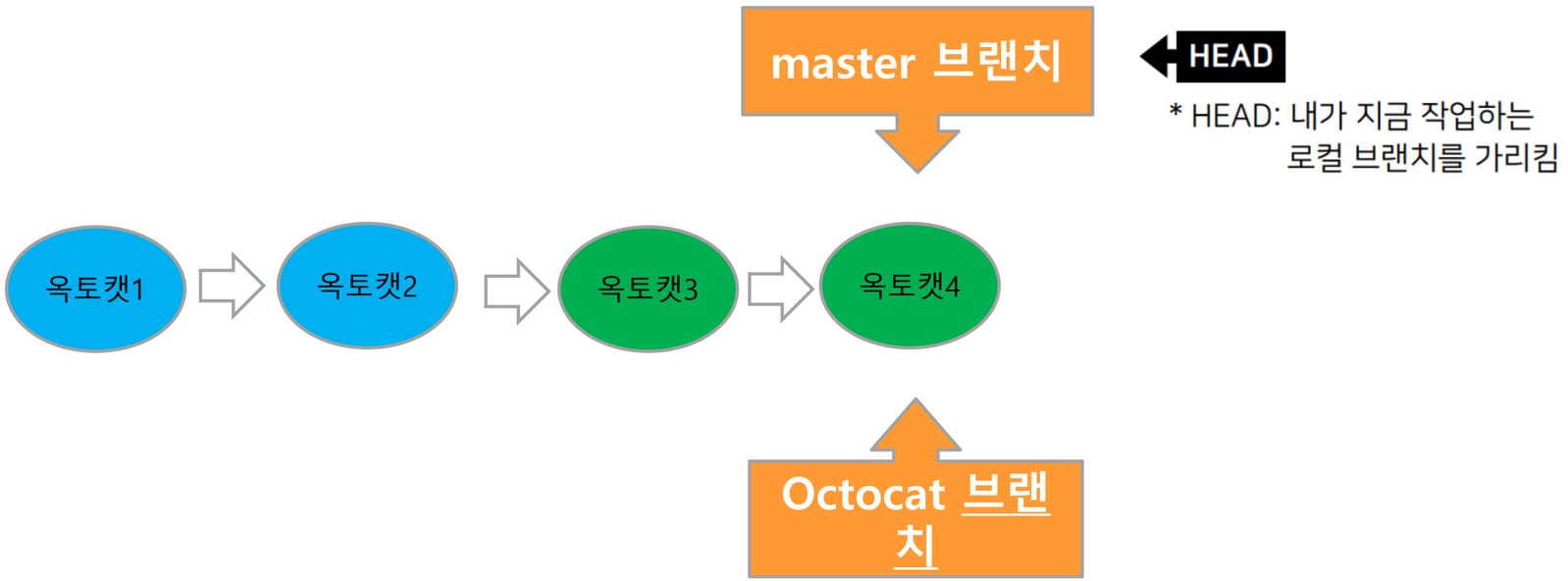
이 명령어는 master 브랜치(기본으로 만들어져 있음)에 커밋을 푸시해라 라는 뜻이었습니다.



브랜치 만들기 git branch

```
git branch Octocat
```

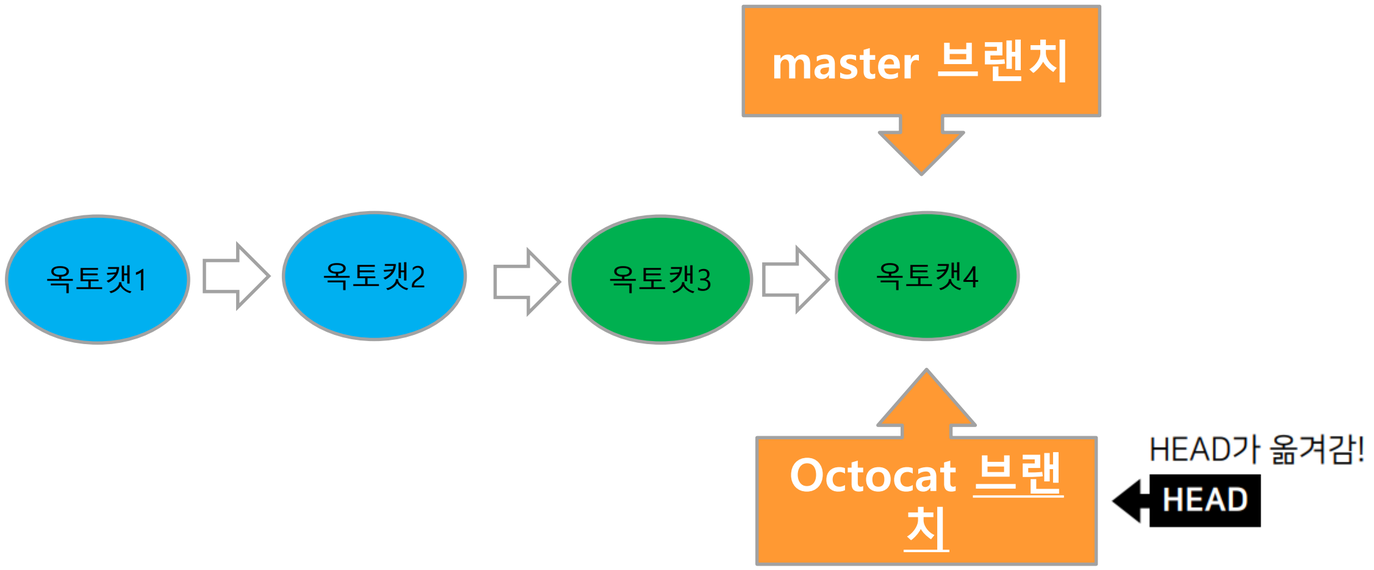
- Octocat 브랜치를 현재 시점에 만들어라!



만든 브랜치로 이동하기 git checkout

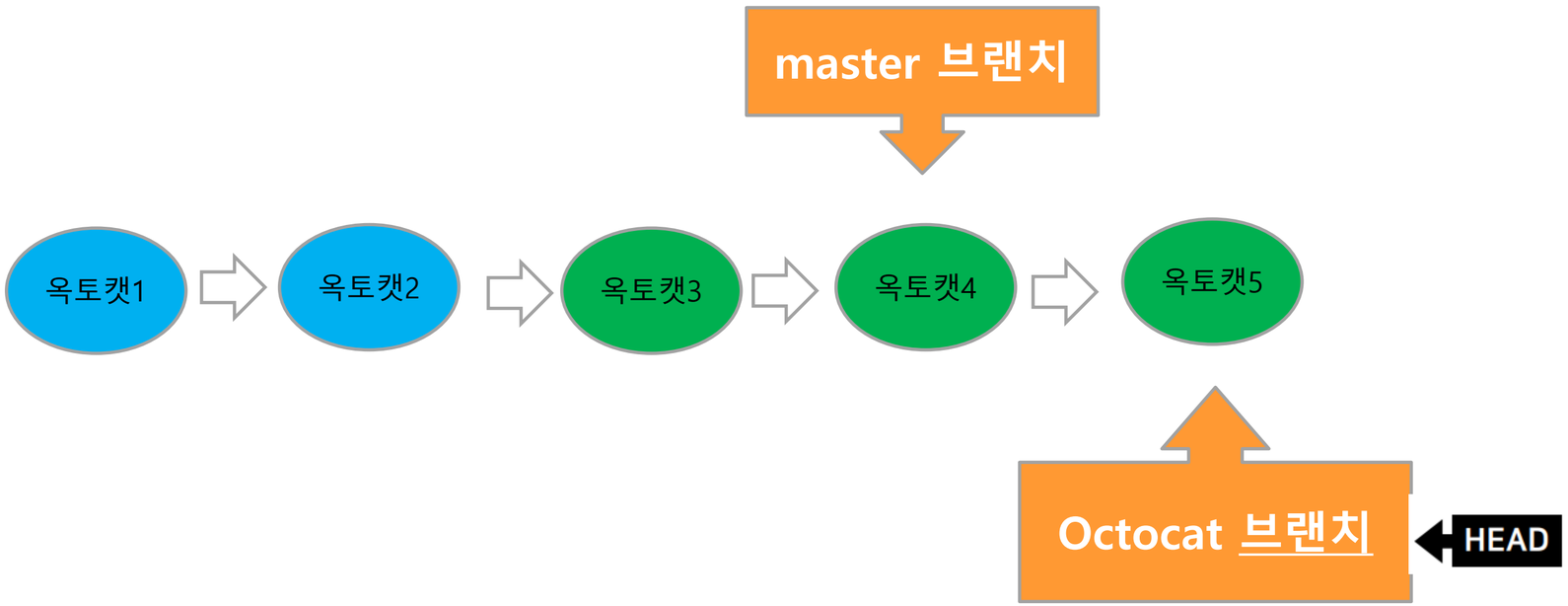
```
git checkout Octocat
```

- Octocat 브랜치로 이동해라!



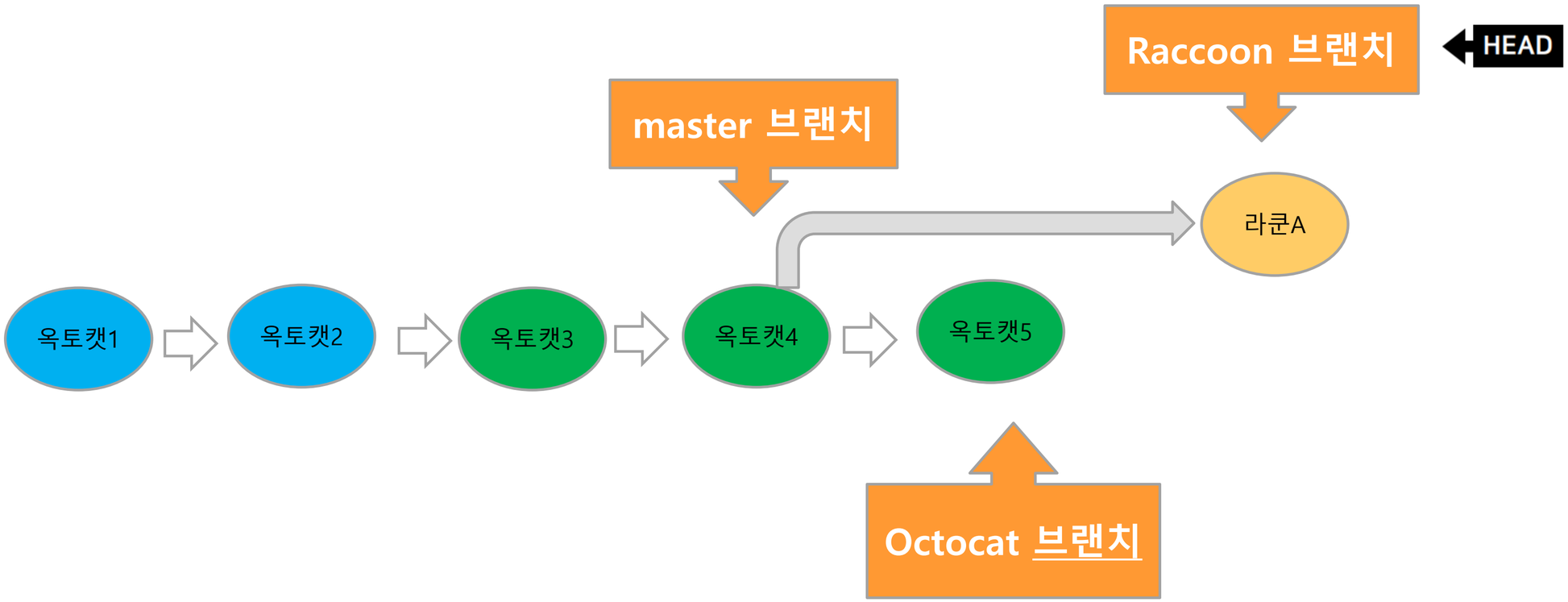
Octocat 브랜치에 커밋을 추가하면?

- master 브랜치는 아직 과거 커밋을, Octocat 브랜치는 새 커밋을 가리킨다.



master로 이동하고, Raccoon 브랜치 만들고, 커밋

- master 브랜치의 최신 커밋(옥토캣4)을 기점으로 Octocat 브랜치, Raccoon 브랜치가 나누어진다.



브랜치 생성 실습

- 1. [MyApp-Octocat 저장소] master에서 feat/main-page 브랜치 생성
- 2. 커밋 추가
- 3. [MyApp-Raccoon 저장소] pull받기
- 4. [MyApp-Raccoon 저장소] master에서 feat/comment 브랜치 생성
- 5. 커밋 추가

브랜치 생성 실습

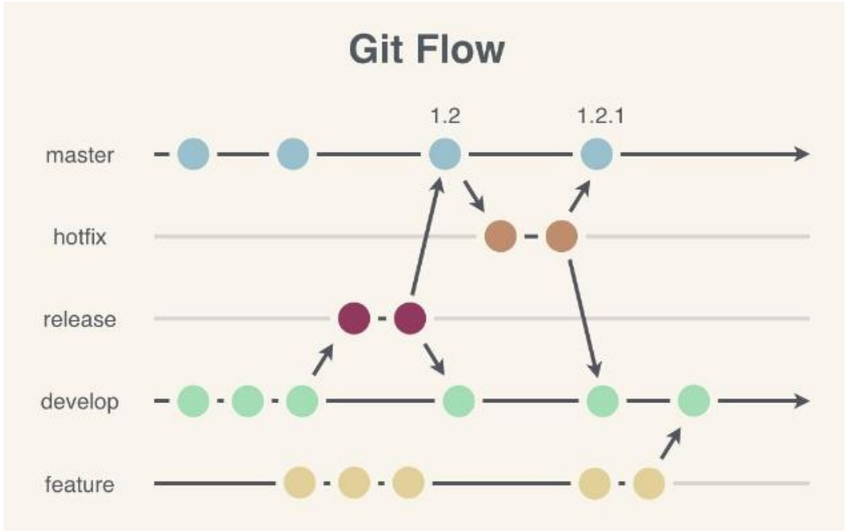
- 1.CLI로 branch 생성 및 삭제하기.

```
git branch feat/login          //branch 생성
git checkout feat/login        //생성한 branch로 이동
git push origin feat/login     //생성한 branch에 push 하기
git checkout master            //master branch로 이동
git merge feat/login           //master와 생성한 branch를 merge하기
git push origin master         //master branch push 하기
```

```
git branch –d feat/login                    //로컬 branch 삭제하기
git branch origin --delete feat/login       //원격 branch 삭제하기
```

Git-flow 전략

- 항상 유지되는 메인 브랜치들(master, develop)과 일정 기간 동안 만 유지되는 보조 브랜치들(feature, release, hotfix)을 포함하여 5가지 브랜치가 있다.



- Master branch : 운영 서버에 제품(Product)으로 출시 될 수 있는 브랜치
- Develop branch : 다음 출시 버전을 대비하여 개발하는 브랜치
- Feature branch : 추가 기능을 개발하는 브랜치
- Release branch : 이번 출시 버전을 준비하는 브랜치
- Hotfix branch : Master 브랜치에서 발생한 버그를 수정하는 브랜치

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

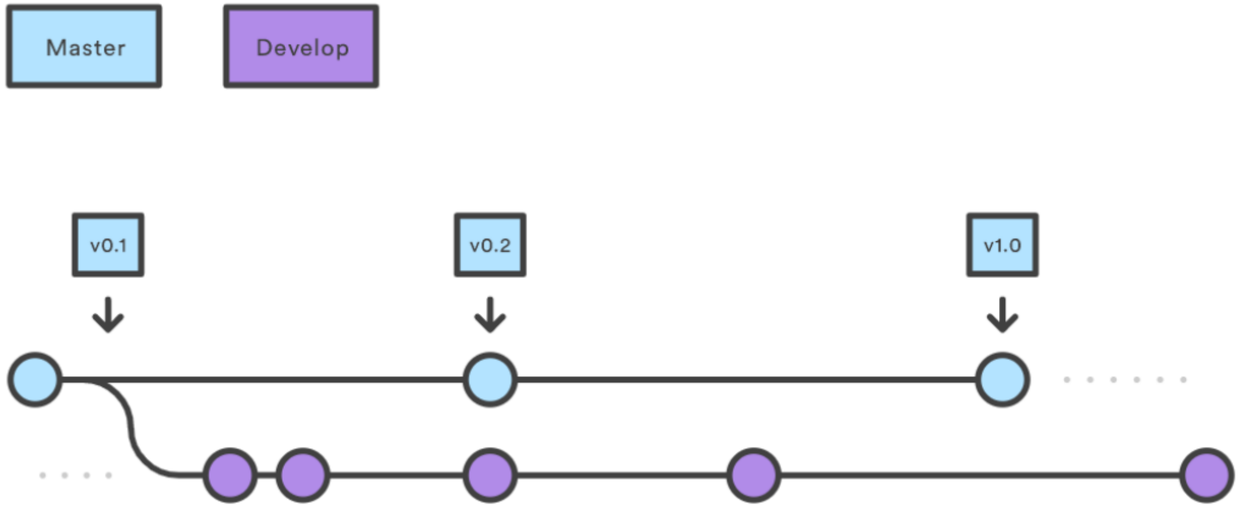
Git branching model

- 1. Master branch

: 배포(release) 이력을 관리하기 위해 사용한다. 즉, 배포 가능한 상태만을 관리한다.

- 2. Develop branch

: 기능 개발을 위한 브랜치들을 병합하기 위해 사용한다. 모든 기능이 추가되고 버그가 수정되어 배포 가능한 안정적인 상태이면 develop 브랜치를 master 브랜치에 병합(merge)한다. 평소에는 이 브랜치를 기반으로 개발을 진행한다.



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

Git branching model

- 3. Feature branch : 기능을 개발하는 브랜치

: feature 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때 마다 develop 브랜치로 부터 분기한다.

: feature 브랜치에서의 작업은 공유할 필요가 없기 때문에 로컬 저장소에서 관리한다. 개발이 완료되면 develop 브랜치로 병합(merge) 하여 다른 사람들과 공유한다.

: feature/기능요약 형식 이나, feature/{issue-number}-{feature-name} 형식을 사용한다.

=> feature/login, feature/1-init-project, feature/2-build-gradle-script-write

- 4. Release branch : 출시 버전을 준비하는 브랜치

: release 브랜치는 배포를 위한 최종적인 버그 수정, 문서추가 등 배포와 관련된 작업을 수행한다.

: develop 브랜치에서 배포할 수 있는 기능이 모이거나 정해진 배포 일정이 되면 release 브랜치를 분기한다.

: release-RB_* 또는 release-*

=> release-1.2

Git branching model

- 5. hotfix branch : 출시 버전에서 발생한 버그를 수정하는 브랜치

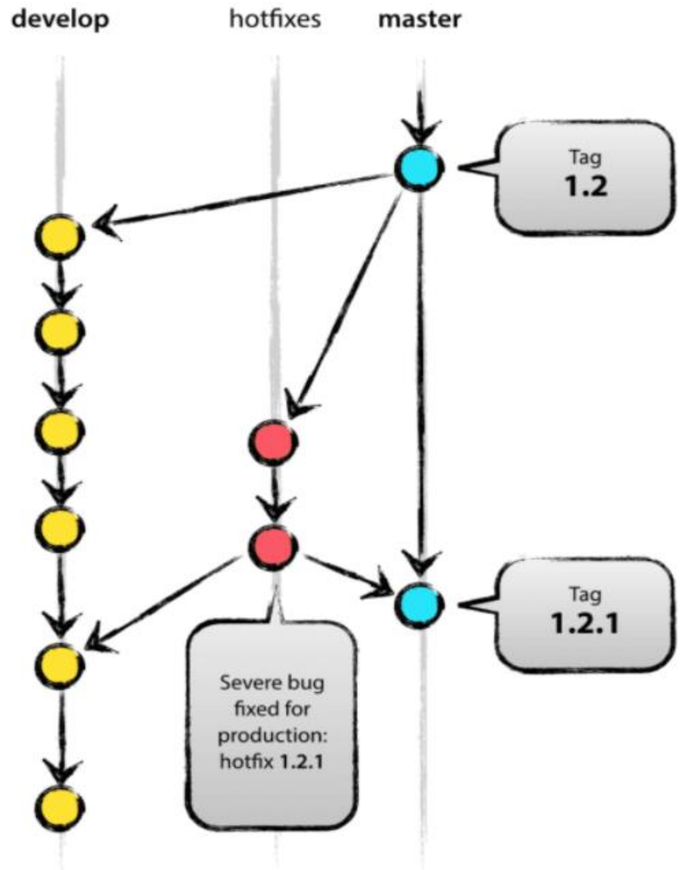
: release 한 버전에 긴급하게 수정을 해야 할 필요가 있을 경우에 master 브랜치에서 분기하는 브랜치이다.

1) 배포한 버전에 긴급하게 수정할 부분 발생 => master 브랜치에서 hotfix 브랜치를 분기한다.

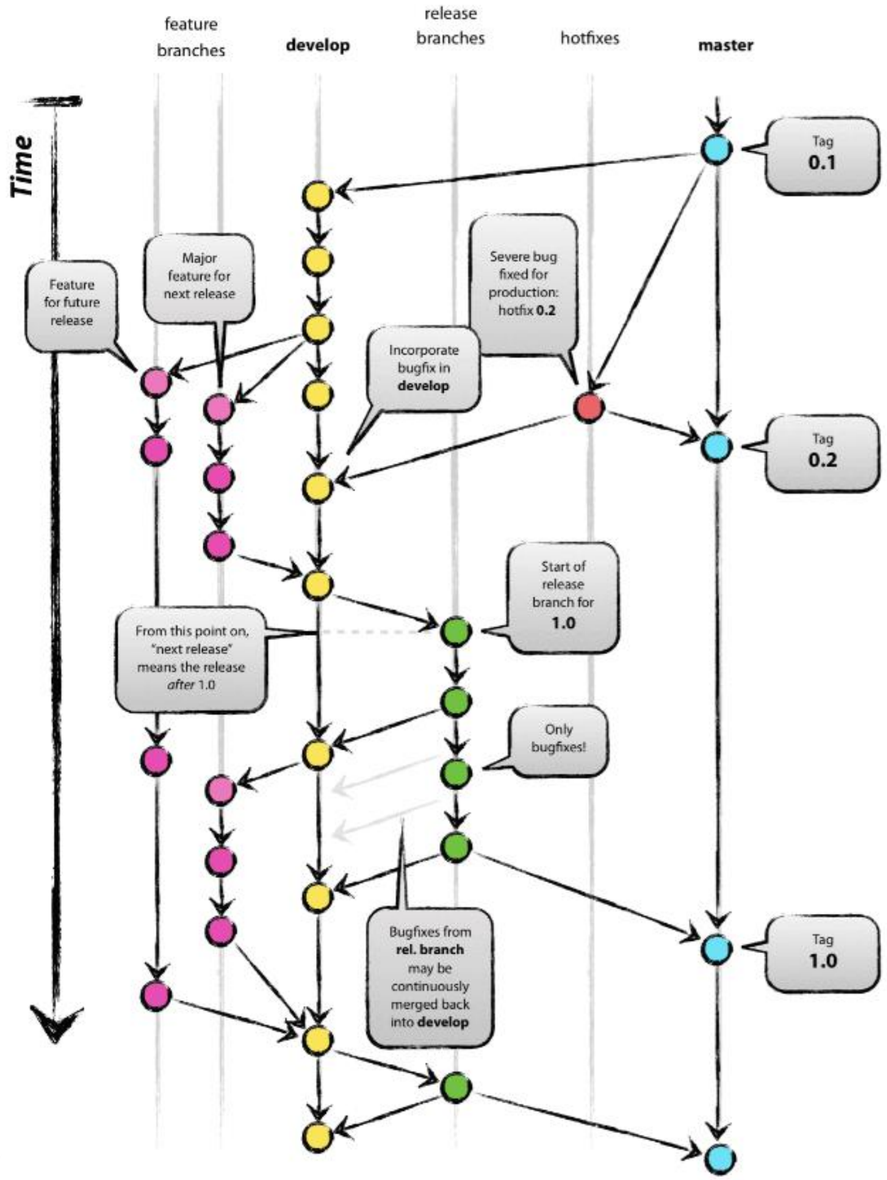
2) 문제가 되는 부분 수정 후, master 브랜치에 merge 하고 배포한다.

3) hotfix 브랜치에서의 변경 사항은 develop 브랜치에도 merge 한다.

4) hotfix는 master 브랜치에서 분리한 임시 브랜치이다.



Git branching model



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

Git-Flow

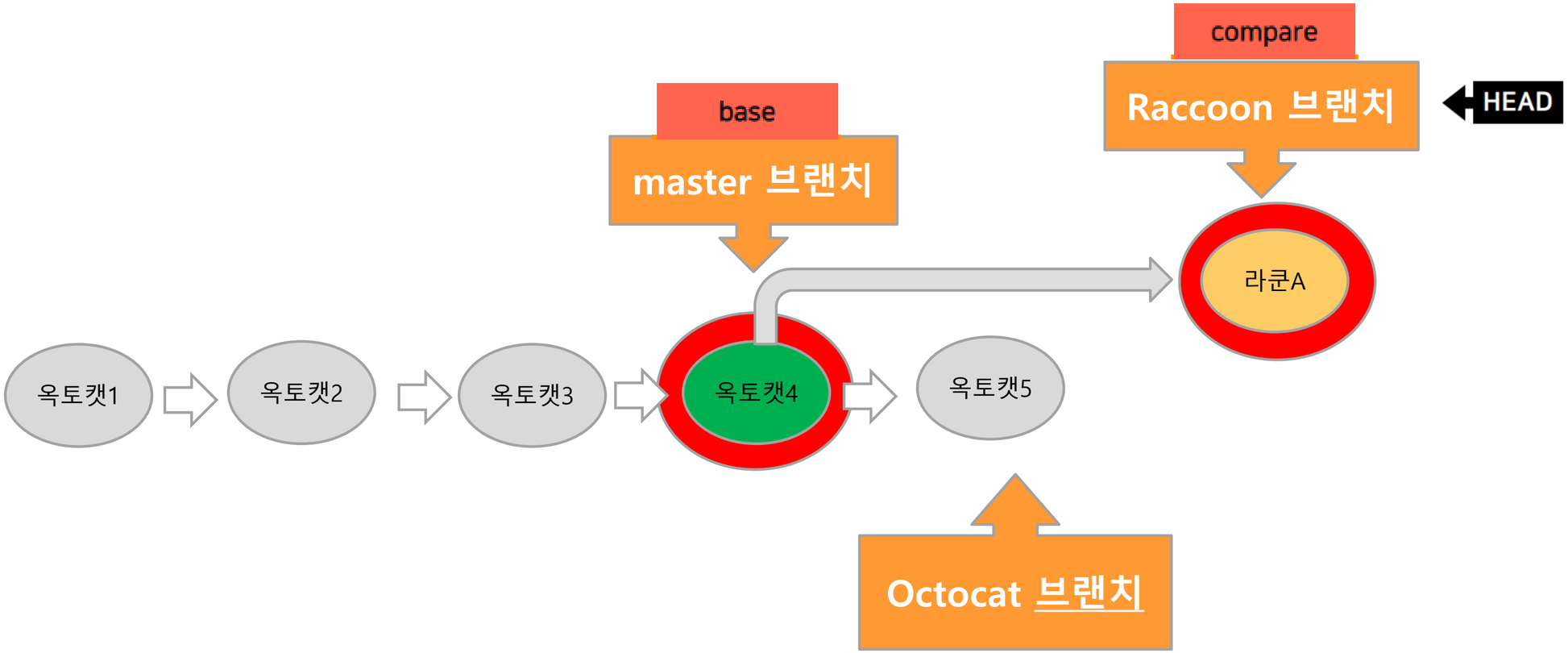
- Git-Flow를 사용하여 개발하는 순서
  - 처음에 Master(Main)와 Develop 생성
  - 새로운 추가 작업은 Develop에서 Feature Branch 생성
  - Feature는 Develop으로 Merge (이때 Develop이 최신 상태인지 확인해야함)
  - QA를 위해서 Develop에서 Release Branch 생성
  - QA에서 발생한 버그는 Release에서 수정
  - QA가 끝나면 Release에서 Develop / Master(Main)으로 각각 Merge
  - Hotfix는 Master에서 시작하여 수정 후 Master / Develop에 Merge

Git-Flow를 좀 더 간소화 한 브랜칭 전략이 GitHub-Flow이다.

두 버전 합치기: 머지(merge)

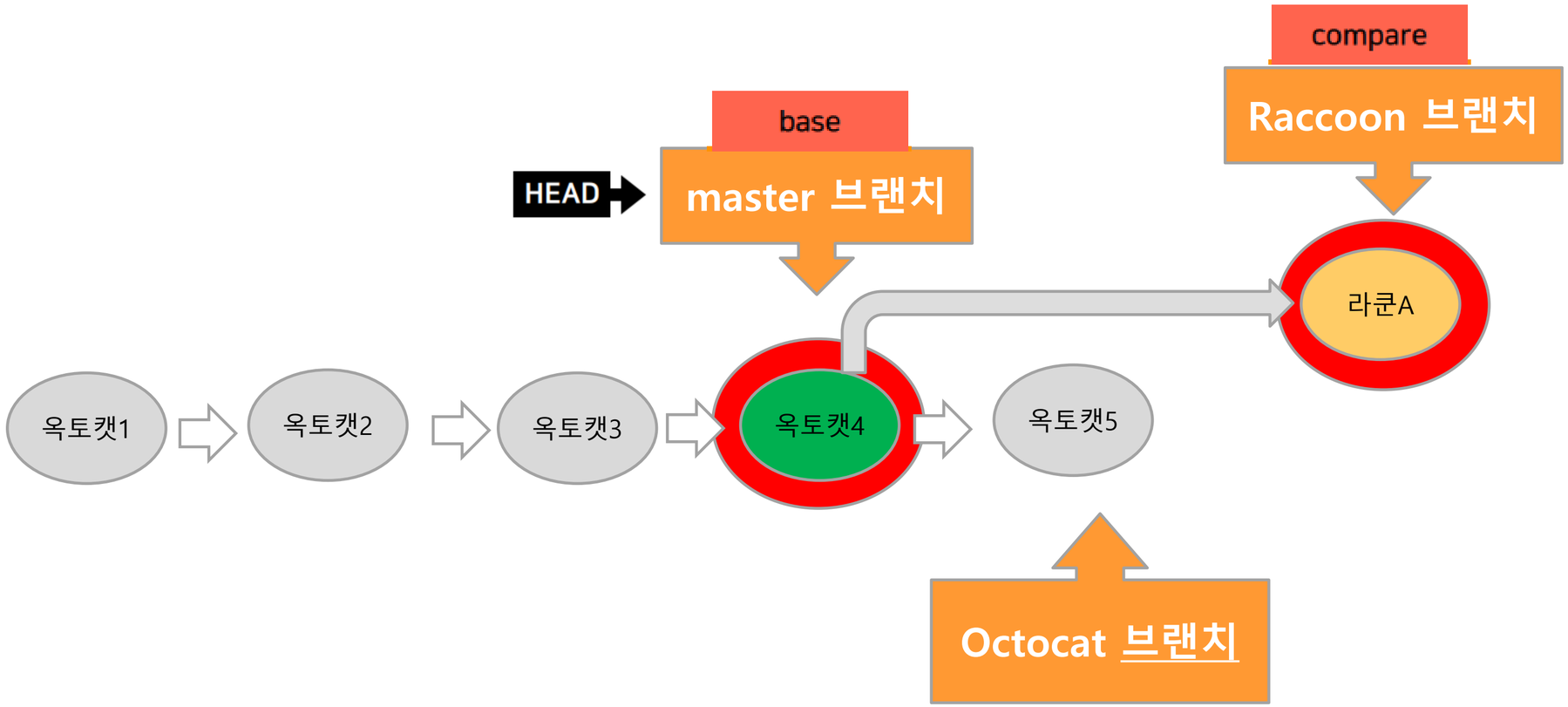
Feat 브랜치를 master 브랜치에 합치기

- master 브랜치의 최신 커밋에(base) Raccoon 브랜치의 최신 커밋(compare)을 합치려고 한다.



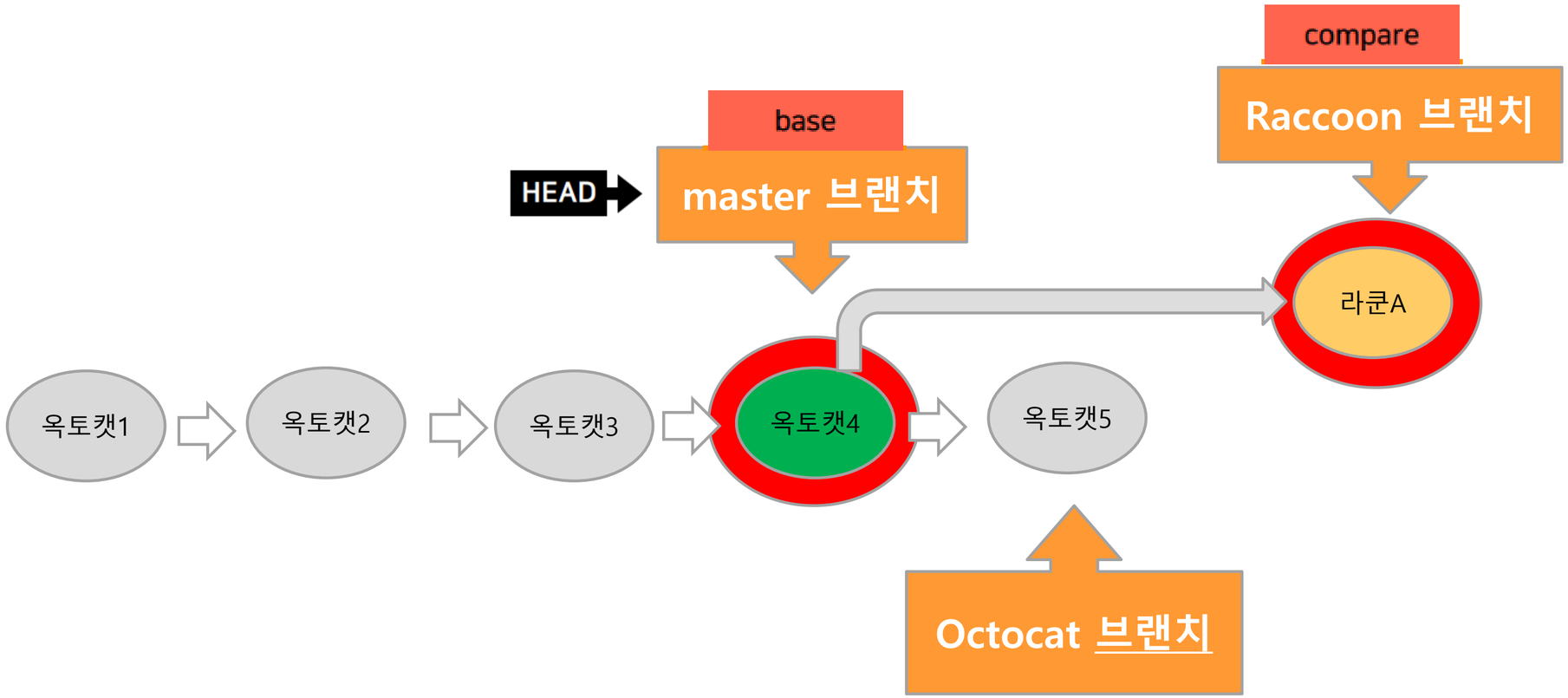
1. 먼저 base가 될 master 브랜치로 이동

```
git checkout master
```



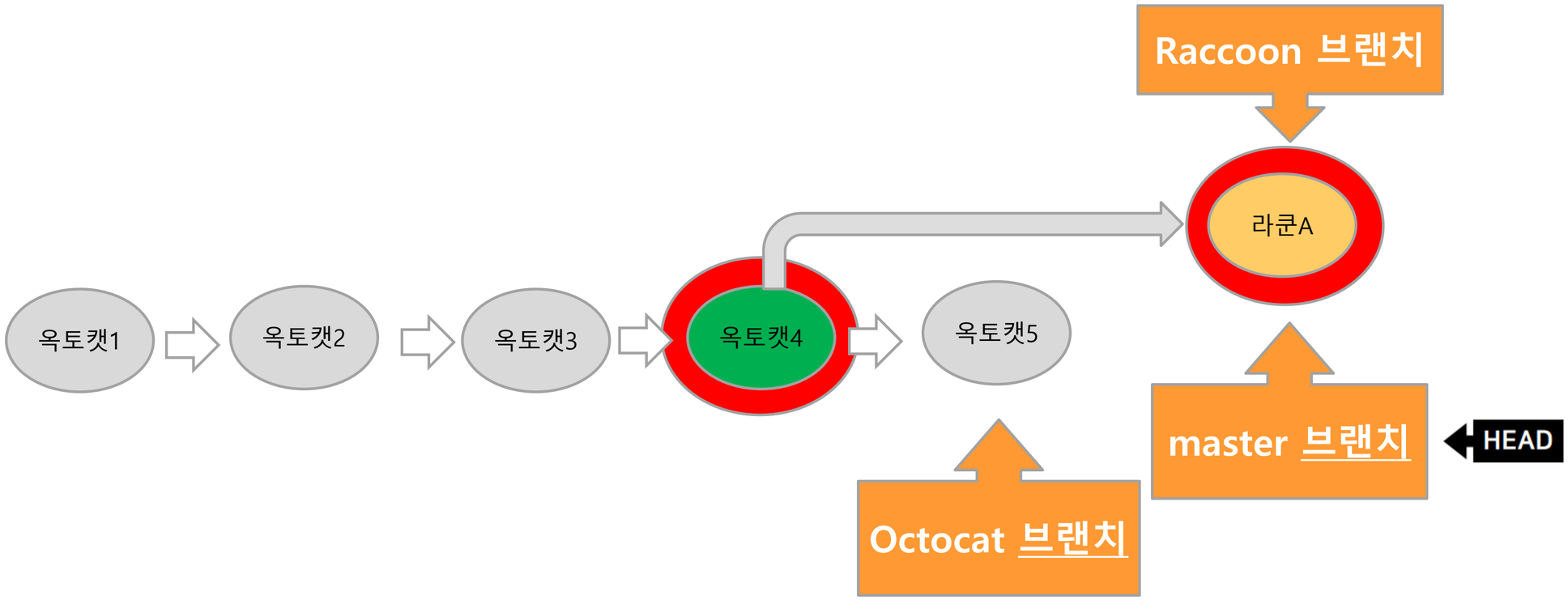
2. compare 브랜치인 Raccoon를 합치고 싶다라고 명령

```
git merge Raccoon
```



3. 합쳐진 결과는 라쿤A 커밋

- 라쿤A는 옥토캣4 + a 이므로 둘이 합친 결과물은 당연히 라쿤A.
- Raccoon 브랜치와 master 브랜치 모두 라쿤A를 가리킨다.

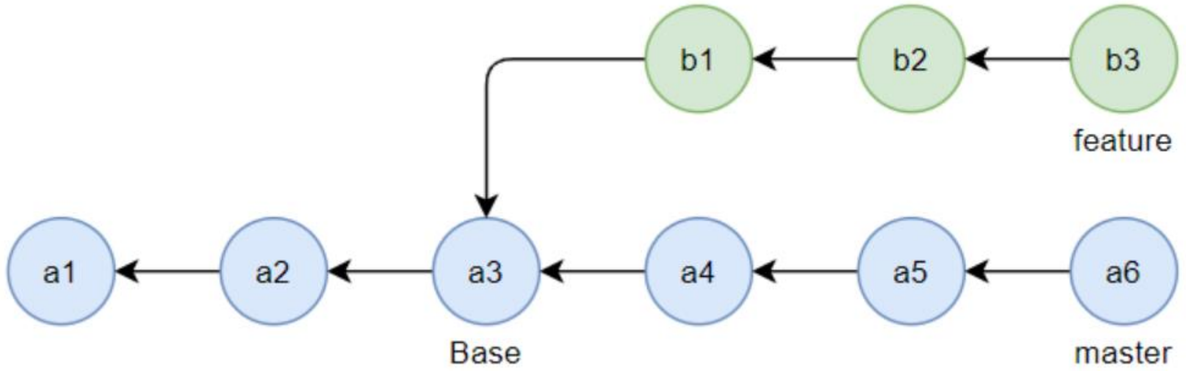


머지(merge) 실습

- 1. [MyApp-Octocat 저장소] master 브랜치로 이동
- 2. feat/main-page 브랜치와 현재 브랜치(master)를 머지(merge)
- 3. master 브랜치에 feat/main-page의 최신 커밋이 반영된 것을 확인

Rebase

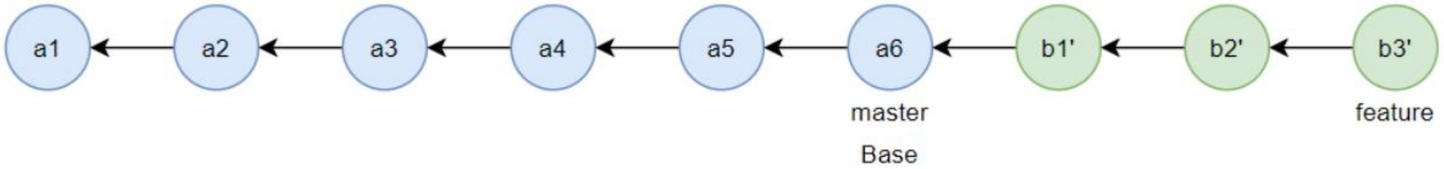
- Rebase는 이름 그대로 브랜치의 base를 다시 설정한다는 의미입니다.



- feature브랜치에서 a3 커밋을 베이스로 작업을 진행하고 있고, master 브랜치는 a4,a5,a6이 진행되었습니다.
- 이때 최신 master 브랜치의 작업 내용을 본인이 현재 작업 중인 feature 브랜치에 적용하고 싶을 때 Rebase 명령어를 사용합니다.
- Rebase를 사용하면 커밋 히스토리를 간결하게 유지할 수 있습니다.

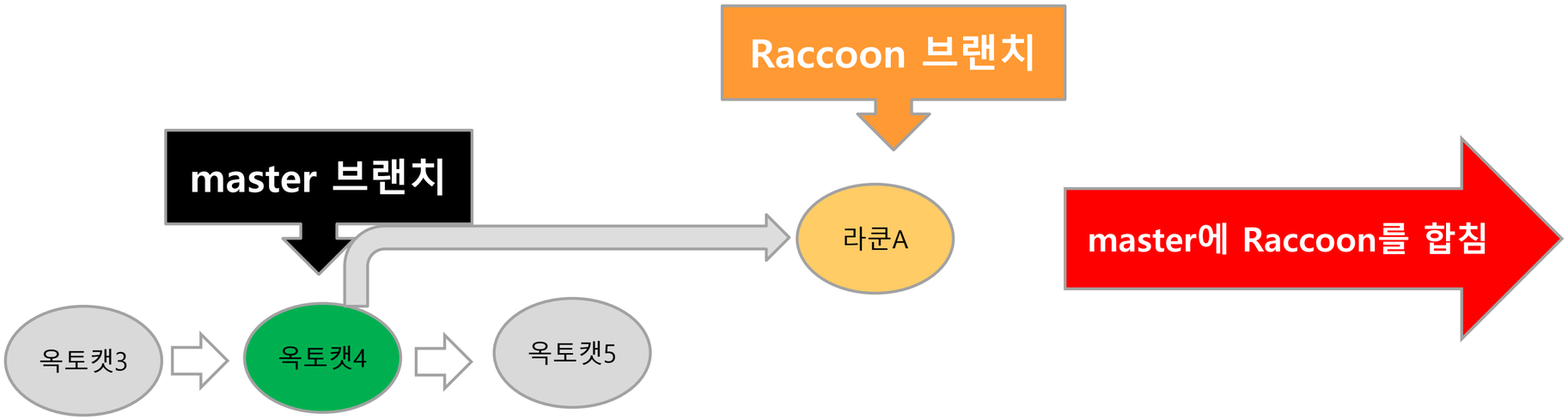
```
# feature브랜치로 이동
git checkout feature
# master를 base로 rebase 명령어 실행
git rebase master
```

- 위 명령어를 실행하면 git log가 아래와 같아집니다.



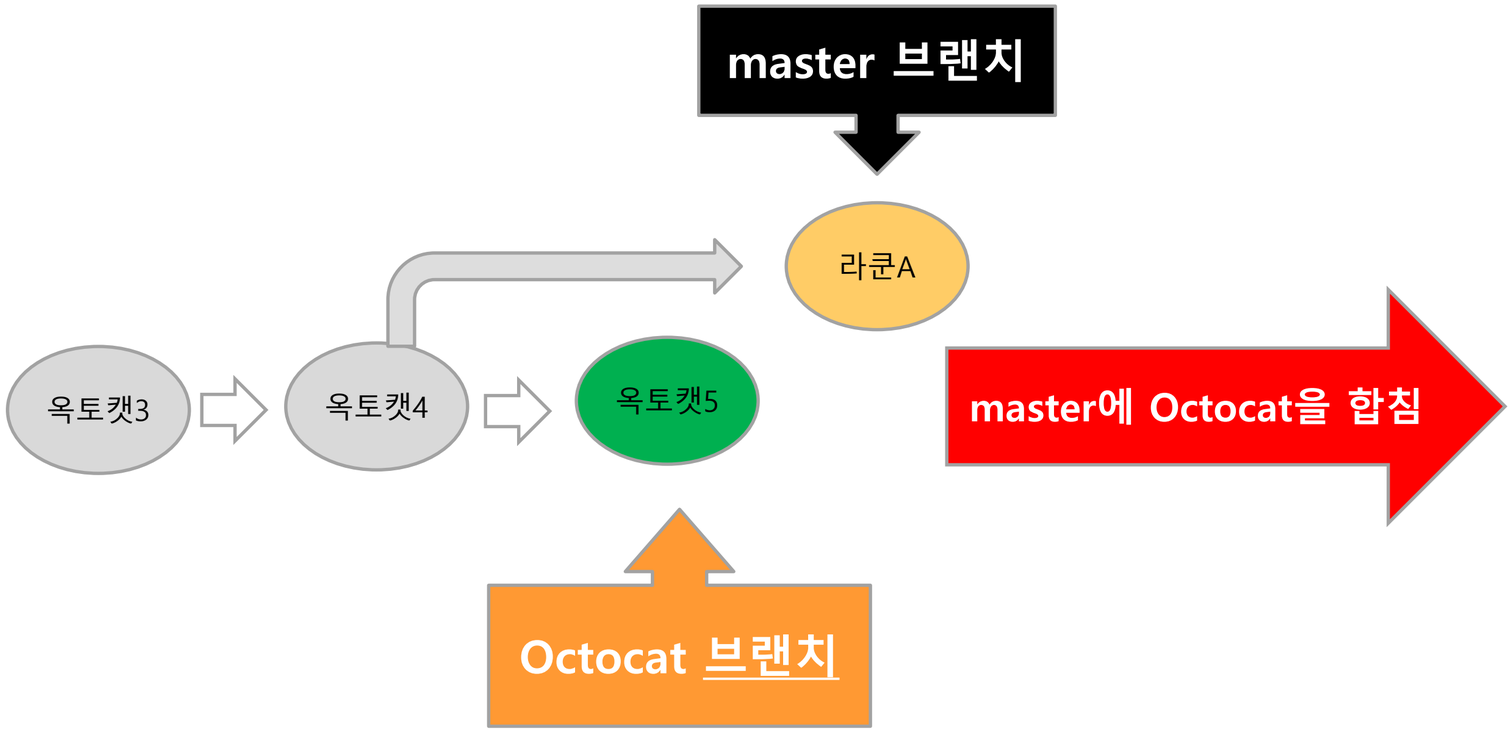
합치다가 충돌이 났어요 : 컨플릭트(conflict)

Case1 : 빨리감기(Fast-forward) 된 Merge



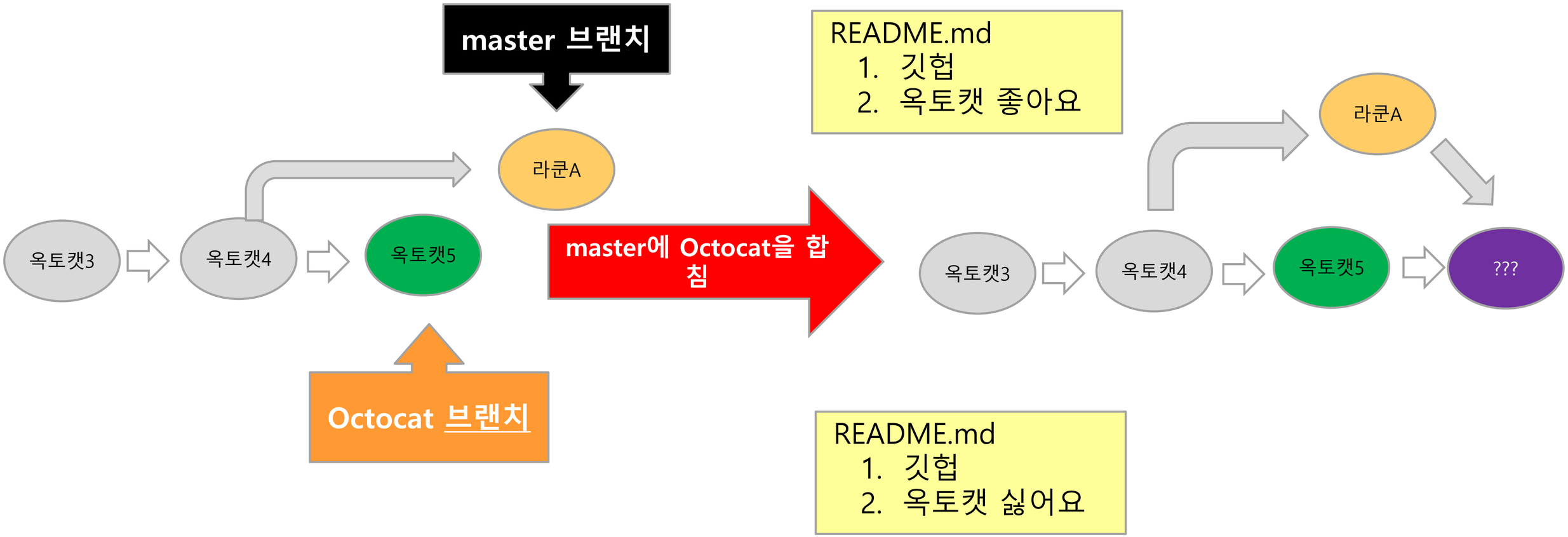
Case1 : 빨리감기(Fast-forward) 된 Merge

Case2 : 새로운 커밋이 만들어지는 Merge



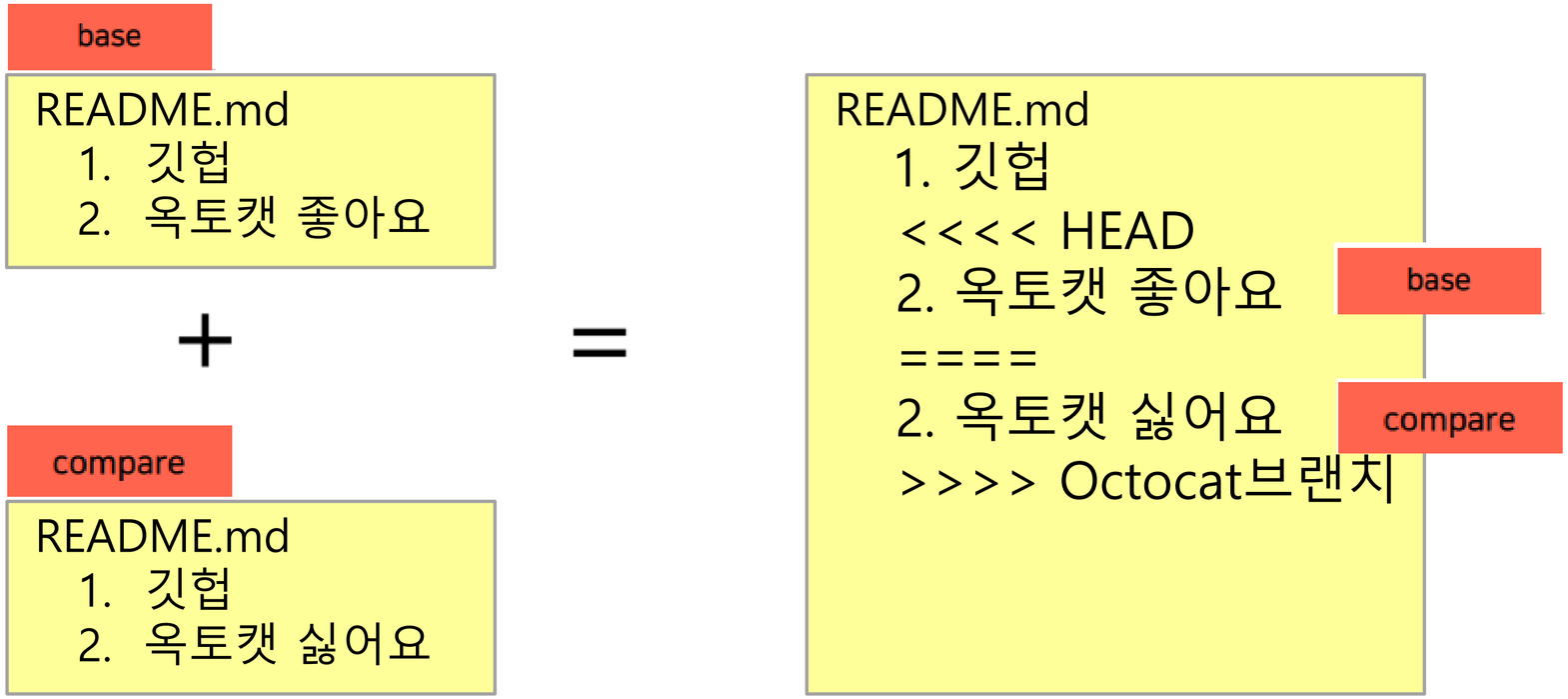
Case2-a : 새로운 커밋이 만들어지는 Merge

Case2-b : 만약에 충돌이 났다면?



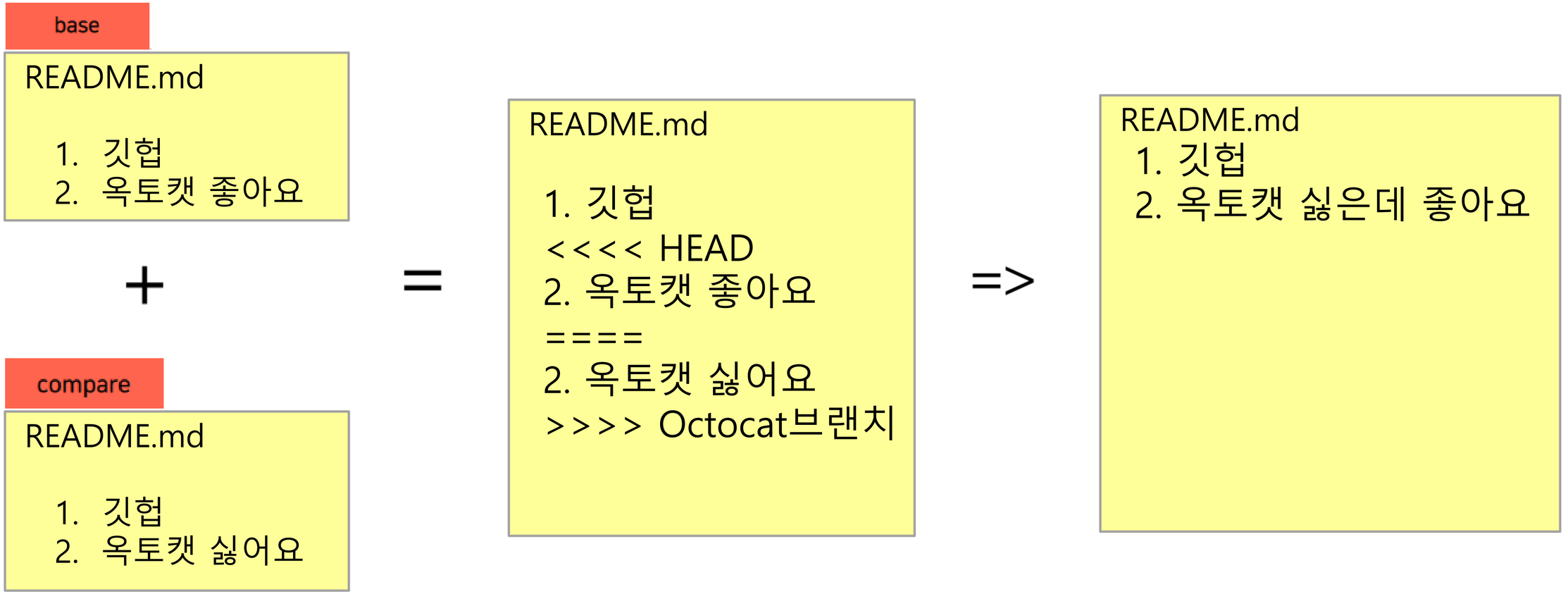
컨플릭트(conflict) 해결

- merge 할 때 두 버전이 같은 곳을 수정했다면 수동으로 고쳐 주어야 한다.

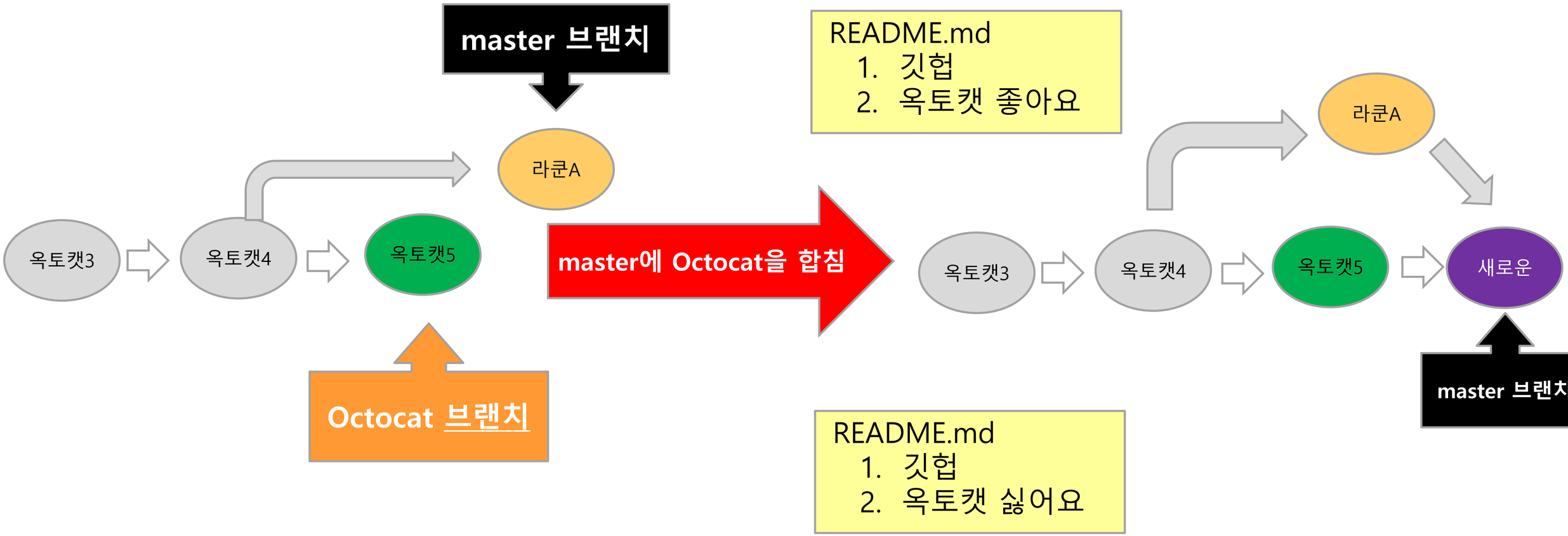


컨플릭트(conflict) 해결

- merge 할 때 두 버전이 같은 곳을 수정 했다면 수동으로 고쳐 주어야 한다.



Case2-b : 충돌이 해결된 Merge 커밋 생성



머지(merge) 컨플릭트 해결 실습

- 1. [MyApp-Octocat 저장소] master 브랜치를 기점으로 feat/love 브랜치 추가하기
- 2. feat/love 브랜치에서 README.md 파일에 ‘옥토캣 좋아요‘ 텍스트 추가 후 커밋
- 3. master 브랜치로 이동 후 README.md 파일에 ‘옥토캣 싫어요‘ 텍스트 추가
- 4. master에 feat/love 의 수정본 merge 할 때 컨플릭트 발생
- 5. 컨플릭트 해결

Git의 추가 명령어
: amend, cherry-pick, reset, revert, stash

개발자로 취업에 성공한 옥토캣

master브랜치에 커밋이 잘못 푸시되었네요.
두 커밋 이전 상태로 돌려주시겠어요?

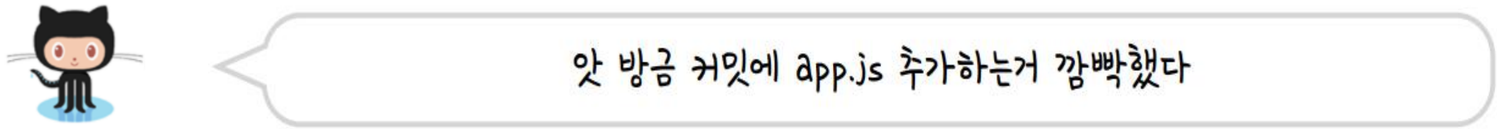
네! (큰일났다··· 어떻게 하지?)

latest 브랜치에서 빨리 고쳐야 하는 버그가 보이네요. master에서
hotfix 브랜치 따서 개발해주시고 나중에 latest에 체리픽 해주세요.

체리···픽?

1. amend

- 깜빡하고 수정 못한 파일이 있어요. 방금 만든 커밋에 살짝 추가하고 싶어요.



- 1. 새로운 커밋 생성
- 2. 코드 업데이트
- 3. 기존 커밋에 amend 해서 커밋 수정 & push
- 4. 커밋 메시지만 수정
- 5. force push

2. stash

- 변경사항을 잠시 keep 해두고 싶어요, 아직 커밋은 안 만들래요.

급한 버그가 생겼어요! ㅠㅠ
지금 하시던거 중단하고 이거 먼저 고쳐주세요

네~
(다른 브랜치로 가야겠네··· 지금 브랜치에서 아직 커밋 못한 파일들은
어떡하지? 커밋하긴 애매한데 날려버릴수도 없고)

2. stash

- 1. 여러 파일 변경사항 만들기
- 2. stash로 서랍에 넣어두기 (tracked 인 것만 들어감)

  $ git stash save "저장할이름"
- 3. stash를 복원하기

  $ git stash apply

3. reset

- 이전 커밋으로 브랜치를 되돌리고 싶어요.

요구사항이 바뀌었어요 ㅠㅠ
지금 만드신 '싫어요 버튼'은 빼고 좋아요 버튼까지만 만들어주세요

넴 (이럴 줄 알고 커밋을 잘 만들어놨지... 옛날 코드로 리셋하자)

3. reset

- 1. feat/b 브랜치의 커밋을 하나 전으로 soft reset

  (커밋 이력만 되돌리고, Source 코드는 이전 상태도 되돌리지 않고 그대로 유지됨)

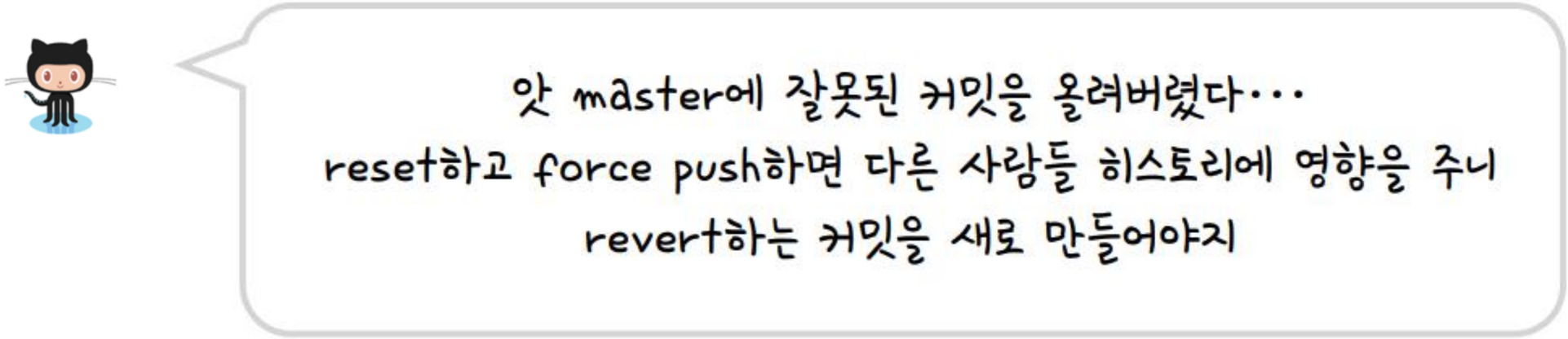
  $ git reset --soft HEAD^
- 2. feat/b 브랜치의 커밋을 다른 커밋으로 hard reset

  (커밋 이력도 되돌리고, Source 코드는 이전 상태로 되돌린다.)

  $ git reset --hard HEAD^

4. revert

- 이 커밋의 변경사항을 되돌리고 싶어요.



- 1. 새로운 커밋 생성
- 2. 방금 만든 커밋 revert 하는 새로운 커밋 생성 (SourceTree 명령은 Reverse)

  $ git revert HEAD

5. cherry-pick

- 저 커밋 하나만 떼서 지금 브랜치에 붙이고 싶어요.

어제 릴리즈한 latest브랜치에 버그가 있어!
일단 fix/text-bug 브랜치에서 버그를 고쳐서 master에 머지했다...

master에 다른 수정사항도 너무 많아서 latest랑 당장 머지할 수 없겠네요.
그래도 릴리즈된 latest 브랜치에 버그수정 커밋은 들어가야 해요!

그럼 제가 고친 코드가 있는 커밋을 latest에 똑 떼서 붙일게요.
이게 체리픽이죠? 후후후

5. cherry-pick

- 1. fix/bug 브랜치 생성 -> 커밋 2개(버그수정, 일반기능) 추가
- 2. master로 체크 아웃
- 3. master에서 fix/bug의 버그수정 커밋 만 체리픽 하기

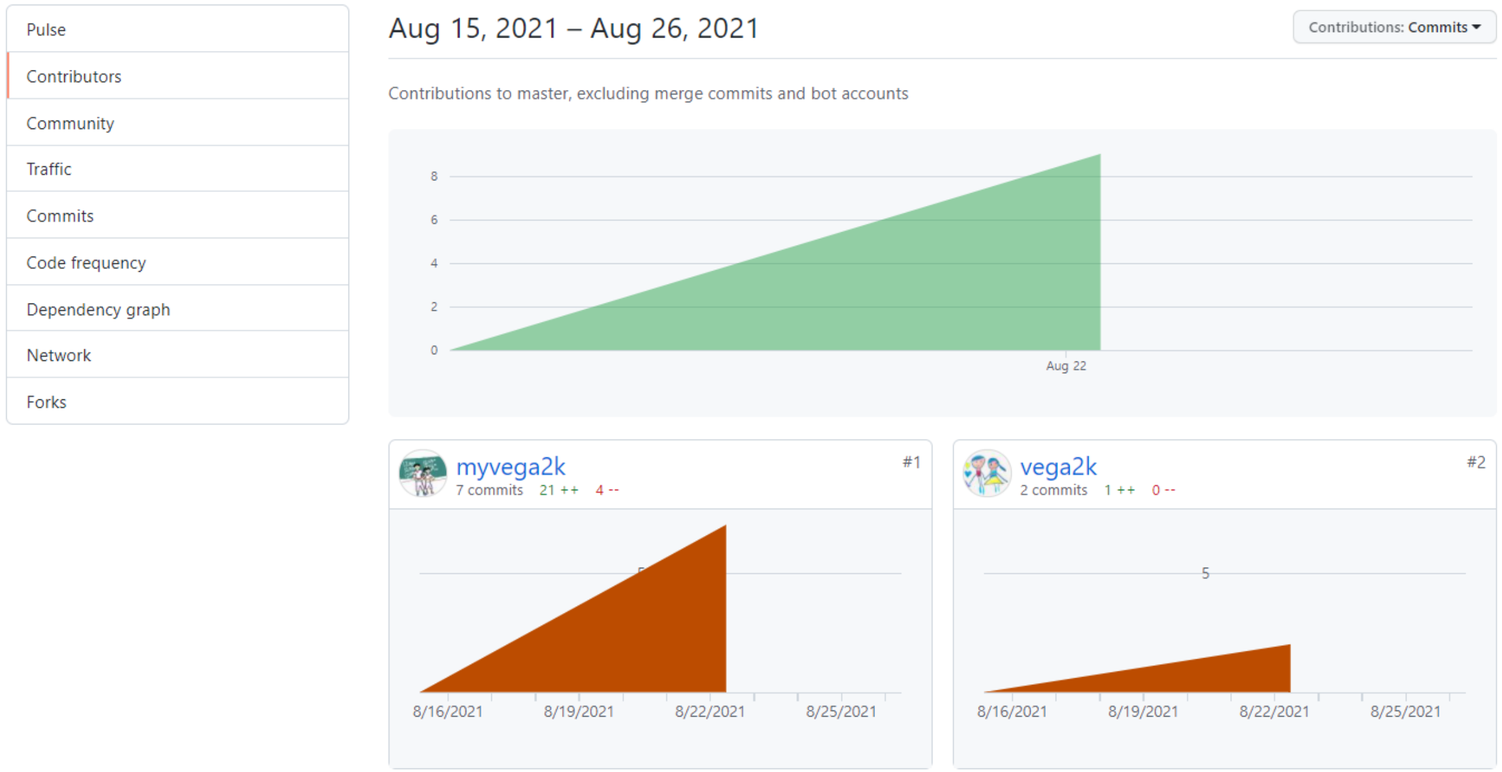
```
$ git cherry-pick fix/bug~1
```

https://stackoverflow.com/questions/9339429/what-does-cherry-picking-a-commit-with-git-mean

저장소 통째로 복제하기 : 포크(fork)

오픈 소스에 기여하기

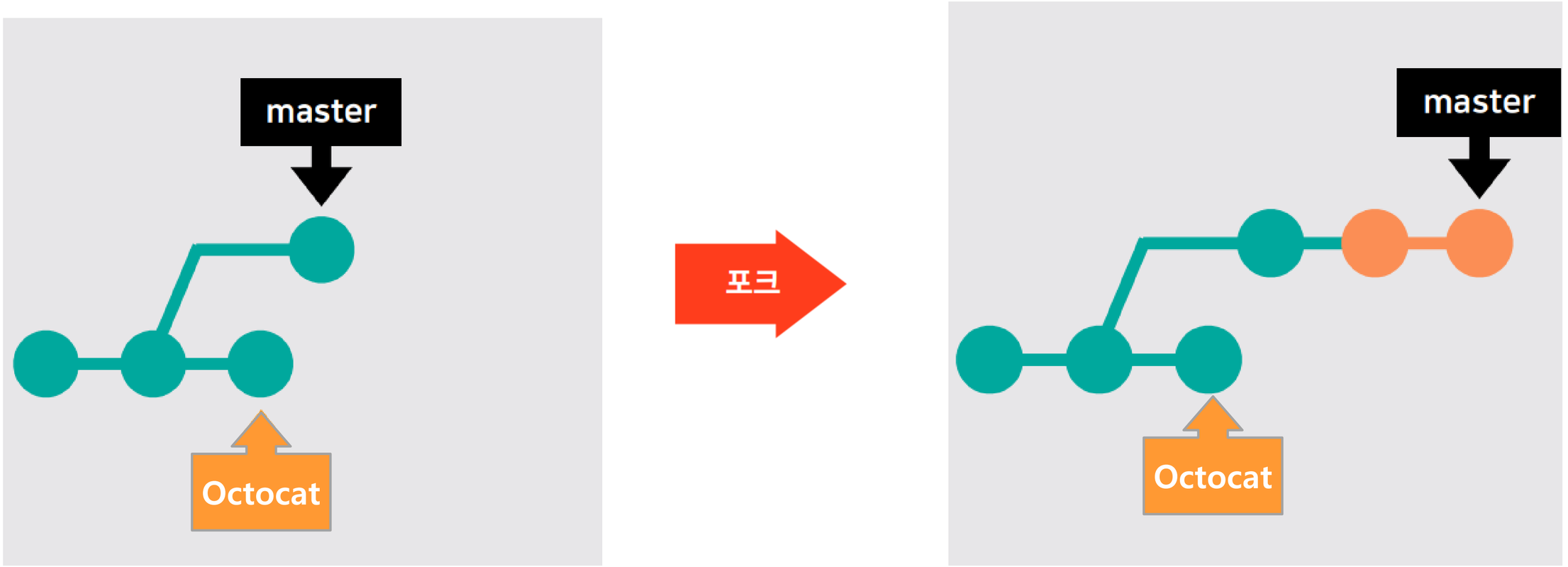
- 다른 사람이 만든 오픈소스에 다른 개발자가 커밋을 하고 싶어 합니다.
- 오픈소스에 기여를 하기 위해서는 커밋 전에 Collaborator 등록을 부탁해야 합니다.



포크(fork): 저장소를 통째로 복제

- 1. 내 저장소에 해당 오픈소스를 복제(fork) 한다.
- 2. 그 저장소(내꺼)에 자유롭게 커밋, 푸시를 하고
- 3. 내 저장소의 브랜치와 오픈소스의 브랜치를 merge 해 달라고 요청하면 됩니다.

포크(fork): 저장소를 통째로 복제



브랜치 vs 포크

- 두 가지 모두 코드를 협업하기 위해서 분기점을 나누는 방식이지만
  특성이 다르므로 내 프로젝트 상황에 맞게 사용하면 됩니다.

	기능	편리한 점	불편한 점
Branch	하나의 원본 저장소에서 분기를 나눈다.	하나의 원본저장소에서 코드 커밋 이력을 편하게 볼 수 있다.	다수의 사용자가 다수의 브랜치를 만들면 관리하기 힘들다.
Fork	여러 개의 원격저장소를 만들어 분기를 나눈다.	원본저장소에 영향을 주지 않으므로 자유롭게 코드를 수정할 수 있다.	원본저장소의 이력을 보려면 따로 주소를 추가해 주어야 한다.

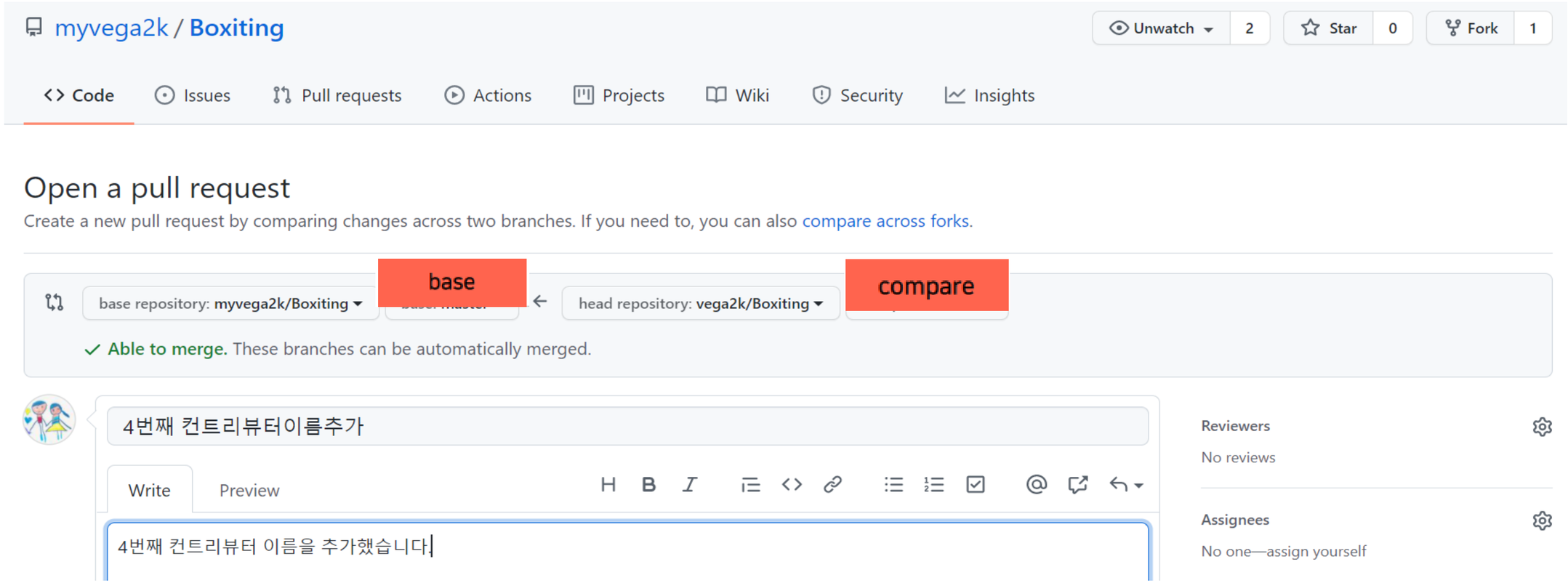
포크 실습

- 1. 원본 저장소를 포크하기
- 2. 포크한 저장소 내컴퓨터에 클론하기
- 3. README.md에 내 이름 추가
- 4. 원본 저장소를 원격(remote)에 upstream이란 이름으로 추가

내 코드를 merge 해주세요 : pull request

Pull Request

- 1. merge 하고 싶은 두 브랜치를 선택하기
- 2. 어떤 변경을 했는지 제목과 내용을 적으면 됩니다.
- 3. fork 한 저장소에서도 요청을 보낼 수 있습니다.



Pull Request : merge 요청 보내기

- 1. 코드를 함께 작성하는 팀원이 있다면, 최대한 직접 merge 하는 것은 피하고 모든 merge를 pull request를 통해서 하세요.
- 2. 동료가 내 Pull Request(PR)을 보고 코드를 리뷰 할 수 있습니다.
- 3. 동료의 PR에 수정이 필요하면 댓글을 달아 change request를 보낼 수 있습니다.
- 4. 오픈소스에 PR을 보낼 때는 '기여 안내문서(contribution guideline)'을 반드시 참고해야 합니다.

Tip : 브랜치 관리하기

- 1. 보통 feat/기능이름 로 한 사람이 개발하는 기능 브랜치를 만듭니다.

  또는 fix/버그이름 hotfix/급한버그
- 2. 작업이 끝나면 dev (혹은 master)브랜치로 PR을 보냅니다.
- 3. dev 브랜치에서 작업이 모두 merge 되면 release (혹은 latest)브랜치로 merge 시키고 이를 운영 서버에 배포합니다.
- 4. 직접 커밋은 feat(혹은 fix, hotfix) 브랜치 에만 합니다.
- 5. dev나 master, release 브랜치에서는 직접 커밋하지 말고 merge만 합니다.

Pull Request 실습

- 1. 포크한 저장소에서 원본 저장소로 Pull Request 보내기

한국소프트웨어기술진흥협회
KOSTA Korea Software Technology Association

판교 | 경기도 성남시 분당구 삼평동 대왕판교로 670길 유스페이스2 B동 8층 T. 070-5039-5805
가산 | 서울시 금천구 가산동 371-47 이노플렉스 1차 2층 T. 070-5039-5815
웹사이트 | http://edu.kosta.or.kr **팩스 |** 070-7614-3450