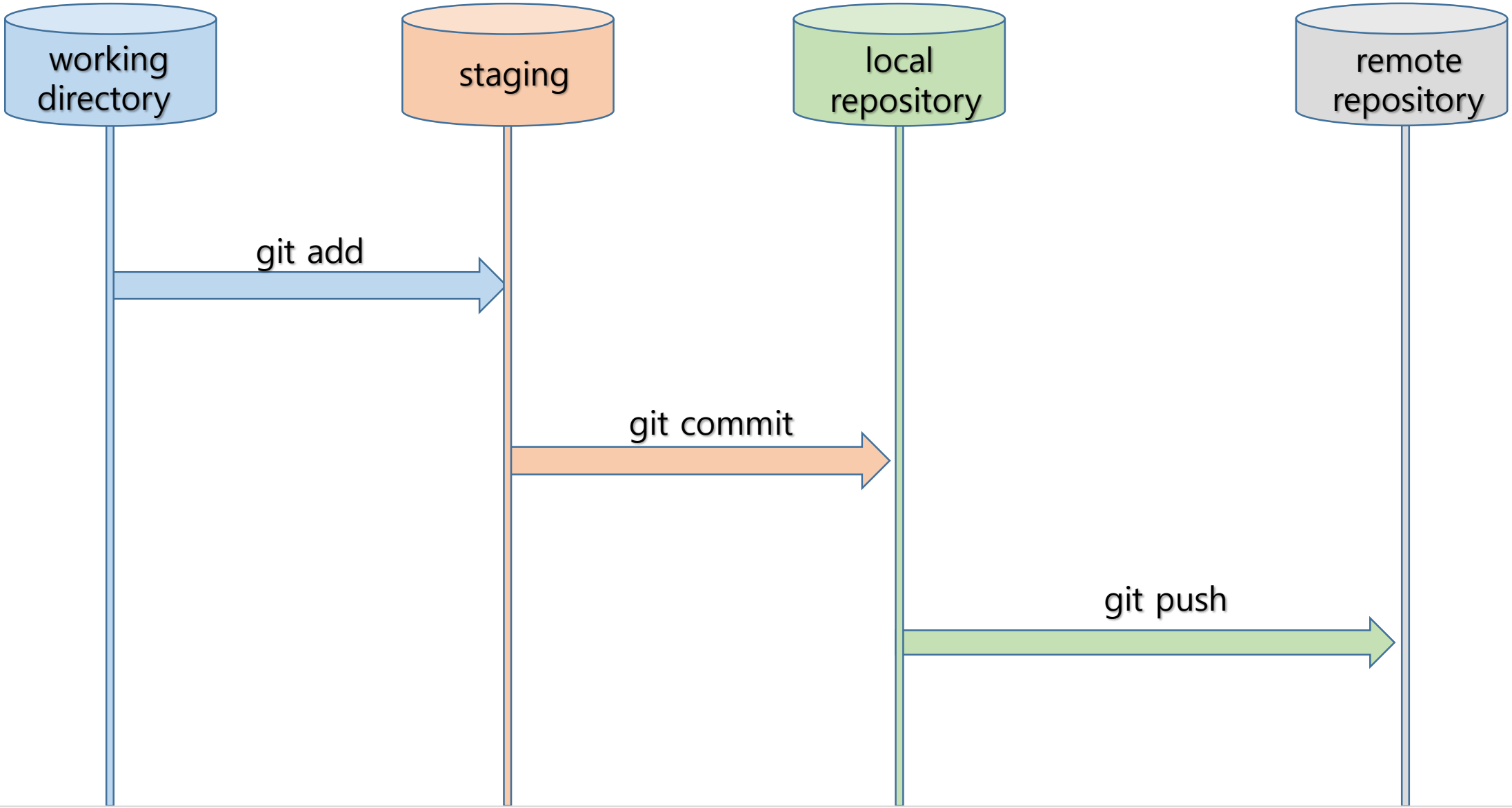
Git Area

Git Area


working directory
staging
local repository
remote repository
git add
git commit
git push

Git Area

- **Working Directory**

내가 작업중인 디렉토리(폴더)가 있는 영역
Working Directorry에는 Untracked와 Tracked가 구분됨

1. Unteacked : 파일이 ignore 되었거나 새로 추가된 파일(git add 된 적이 없는 파일)
2. tracked : 수정된 파일 (git add 된 적 있는 파일)

Git Area

- **Staging Area**

파일이 git add되어 commit을 하기 전 준비 상태인 영역
Staging Area에는 크게 아래와 같이 구분됨

1. Unmodified :Statging 영역에서 Commit까지 모두 마친 상태 또는 add한 상태(new file)
   → git status에서 확인 불가 (nothing to commit, working tree clean)
2. Modified : Git에서 관리되고 있던 파일이 수정되었을 때의 상태
3. Staged : 커밋 직전 단계

Git Area

- **Staging Area**

빨간 박스 : Staged 영역에 올라간 커밋 직전 상태
파란 박스 : Staged 영역에 올라가지 않은 상태
→ git add해서 Staging Area에 올리지 않은 상태

```
user@DESKTOP-I9RHACC MINGW64 /d/gitWorkspace (kimjaeseop)
$ git status
On branch kimjaeseop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   borad.html
        modified:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   area.html
```

Git Area

- **Local Repository(=Git Directory)**

  커밋이 완료된 영역

Git Area

- **깃 영역을 활용하기 위한 예시 상황 - 1**

3가지의 파일(index.html, board.html, area.html) 중 3가지를 모두 add 하여 staging area에 들어갔을 경우 이번 commit에서는 area.html을 제외하고 진행하고 싶음

이를 위해서는 staging area에서 working directory로 이동시켜야 함

git restore 명령어를 사용하여 가능

Git Area

- 깃 영역을 활용하기 위한 예시 상황 - 2

3개 파일의 내용을 변경하고 add

```
user@DESKTOP-I9RHACC MINGW64 /d/gitWorkspace (kimjaeseop)
$ git add .

user@DESKTOP-I9RHACC MINGW64 /d/gitWorkspace (kimjaeseop)
$ git status
On branch kimjaeseop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   area.html
        modified:   borad.html
        modified:   index.html
```

Git Area

- 깃 영역을 활용하기 위한 예시 상황 - 3

git restore --staged [파일명] 을 입력하여 staging area 영역에서 제외된 것을 확인
→ working directory에 있기 때문에 작업한 내용은 남아있음

```
user@DESKTOP-I9RHACC MINGW64 /d/gitWorkspace (kimjaeseop)
$ git restore --staged area.html

user@DESKTOP-I9RHACC MINGW64 /d/gitWorkspace (kimjaeseop)
$ git status
On branch kimjaeseop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   borad.html
        modified:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   area.html
```

Git Area

- **Q1. git restore 명령에서 staged 옵션을 제외하고 어떻게 되는지 확인 해보세요.**

1. 작업(수정 등)한 내용이 없어지는지 확인 해보세요.
2. git status 명령을 사용하여 어떠한 변화가 있는지 확인 해보세요.

Git 브랜치 전략 개요

Git 브랜치 전략의 종류

- **Git 브랜치 전략**

  여러명의 개발자가 소스 코드를 한 곳(저장소)에 모아놓은 환경에서 효과적으로 활용하기 위한 전략

- **Git 브랜치 전략의 종류**

  대표적으로 4가지의 전략이 있음

  1. Git Flow
     → 가장 대표적인 브랜치 전략
  2. Github Flow
  3. GitLab Flow
  4. Trunk-based

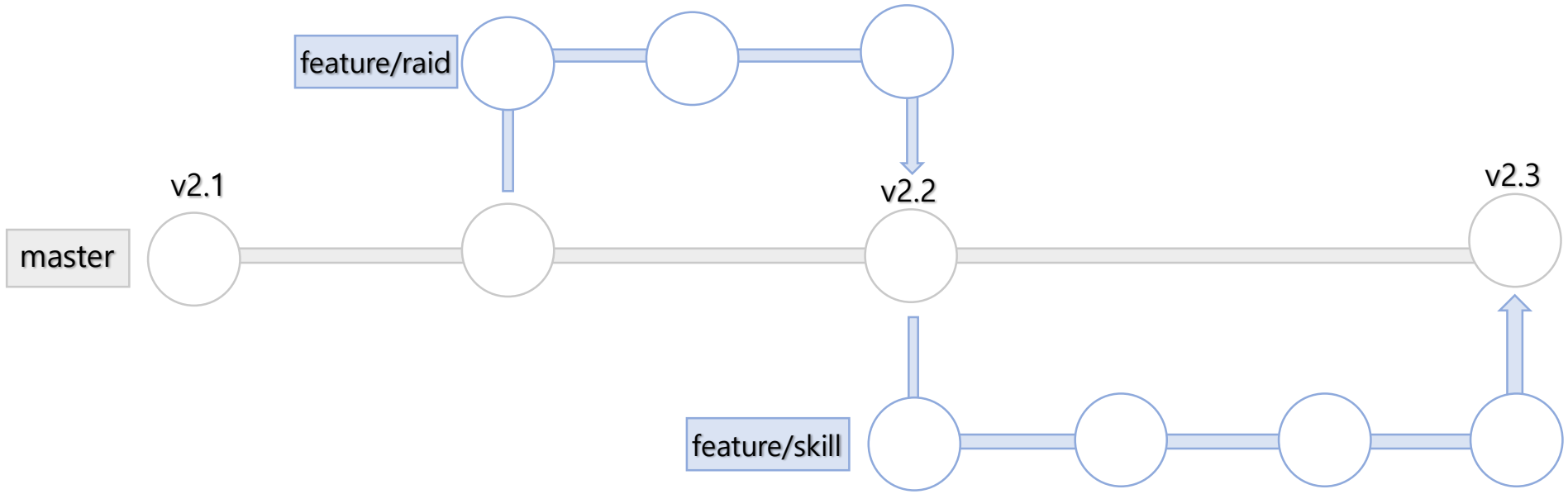
Git 브랜치 전략의 종류

- **Git 브랜치**

브랜치	설명
master	유저들이 사용중인 실제 서비스중인 소스코드가 있는 브랜치
develop	다음 출시 버전을 개발하기 위한 브랜치
feature/{스펙명}	새로운 기능을 개발하는 브랜치
release	출시 전 테스트, QA를 진행하기 위한 브랜치
hotfix	master의 코드에 버그가 발생할 경우 긴급 수정하기 위한 브랜치

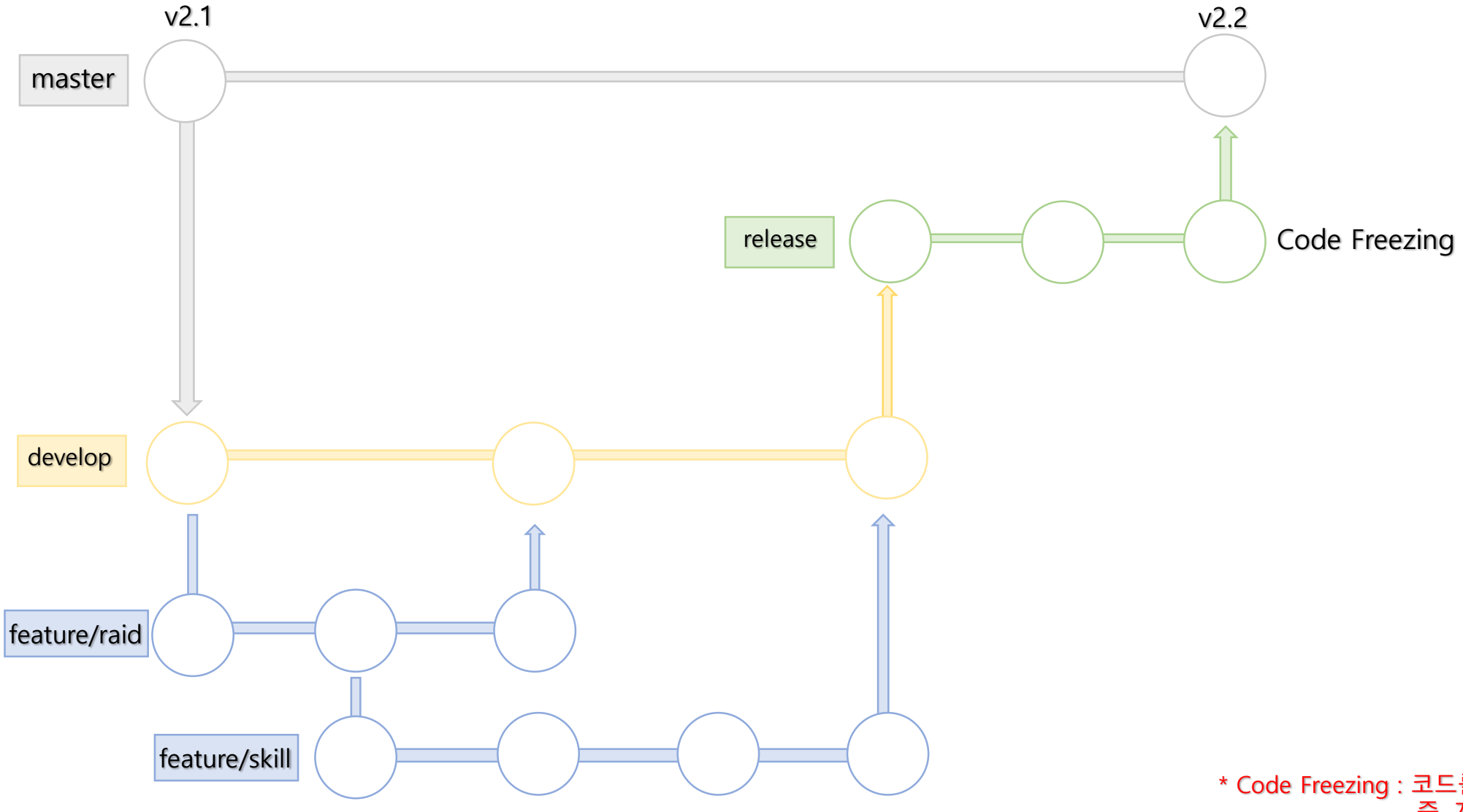
Git 브랜치 전략 - 1
(Trunk-based)

Trnk-based



Git 브랜치 전략 - 2
(Git Flow)

Git Flow - 1



* Code Freezing : 코드를 더 이상 수정하지 않는 상황
즉, 개발 작업을 마무리하는 단계

Git Flow - 2