

K-Digital Training KDT 풀스택 웹 개발자 양성 부트캠프 3기

# Git

• WITH 팀 리쳐드



# Git

- Git 이란?

소스 코드를 효율적으로 관리하기 위해 만들어진 **“분산형 버전 관리 시스템”**

- 사용 이유?

소스 코드의 **변경 이력**을 쉽게 확인  
특정 시점에 저장된 **버전과 비교**하거나 **특정 시점으로 돌아가기**  
위해

# Github 회원 가입 및 로그인


# Github에서 Repository 만들기



## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

 linda-spr ▾

Repository name \*

/

Great repository names are short and memorable. Need inspiration? How about [potential-octo-robot?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

### Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

### Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None** ▾

### Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

# Git 설치하기 ( Window )



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

## Download for Windows

[Click here to download](#) the latest (2.36.1) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 24 days ago, on 2022-05-09.

### Other Git for Windows downloads

Standalone Installer

[32-bit Git for Windows Setup.](#)

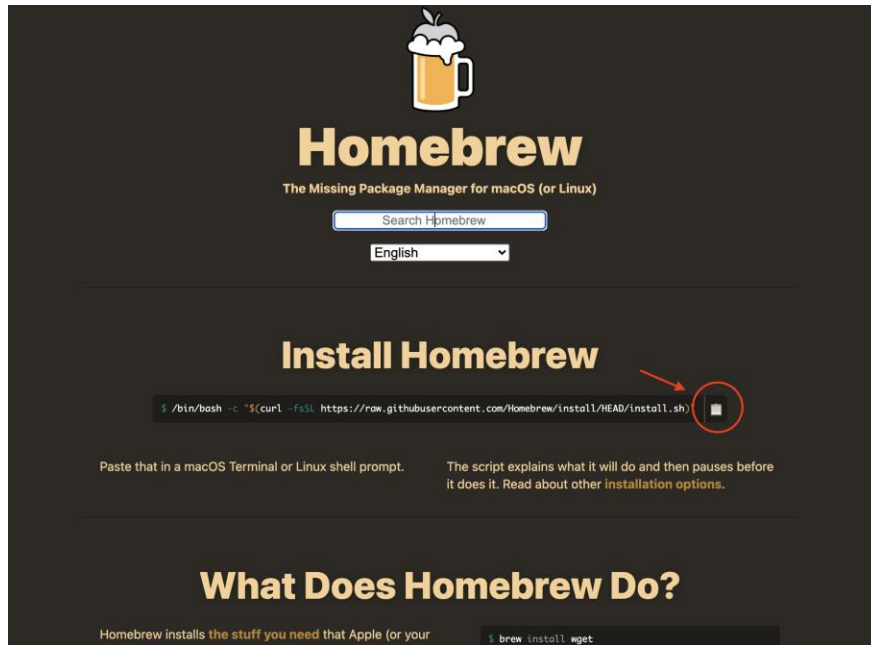
[64-bit Git for Windows Setup.](#)

Portable ("thumbdrive edition")

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

# Git 설치하기 ( MAC )



## 1. Homebrew 설치하기

참고! Homebrew란? 소프트웨어 패키지 관리 시스템  
Homebrew는 프로그램 설치를 명령어로 손쉽게 할 수 있다  
~

## 2. Homebrew로 Git 설치하기

[https://brew.sh/index\\_ko](https://brew.sh/index_ko)

**brew install git**

# Git 설정

- `git config --global init.defaultBranch main`
- 개행 문자 관련 처리(윈도우는 CR, LF / 맥은 LF 만)
  - `git config --global core.autocrlf true` (**Window**, CRLF → LF)
  - `git config --global core.autocrlf input` (**Mac**, LF 만 사용)
- `git config --global user.name` “프로필 이름”
- `git config --global user.email` “이메일 주소”
- `git config --global --list`

# [잠깐!] 터미널 명령어 정리1

- **cd** (change directory): 디렉터리(폴더) 이동
  - `cd /` : 최상위(루트) 디렉터리로 이동
  - `cd ./` : 현재 디렉터리 (/ 생략 가능)
  - `cd ../` : 부모 디렉터리로 이동 (/ 생략 가능)
  - `cd ./폴더명` : 현재 위치의 [폴더명]으로 이동

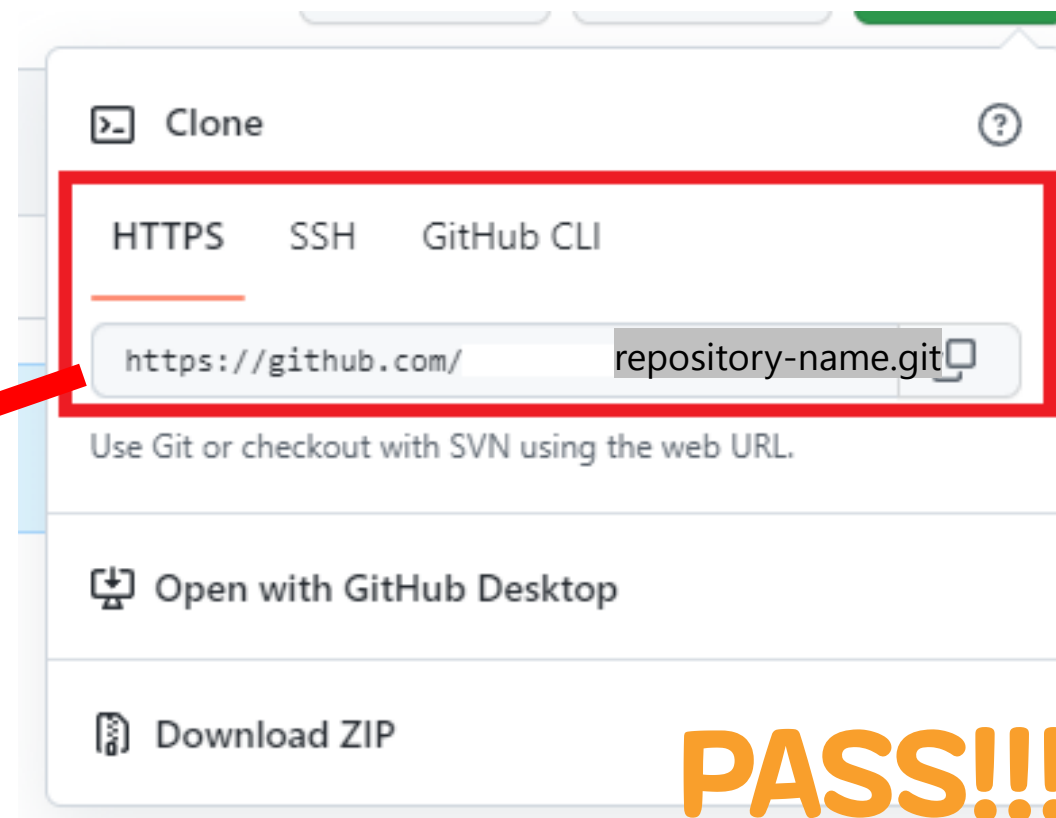


# [잠깐!] 터미널 명령어 정리2

- **mkdir** (make directory): 디렉터리(폴더) 생성
- **ls** (list segments): 현재 위치의 파일 목록 조회
  - ls : 단순 목록 확인
  - ls -l : 파일 상세 정보
  - ls -a : 숨김 파일 표시
  - ls -al : 숨김 파일 표시 & 파일 상세 정보 확인

# 방법1. Github Repository 받아오기

- mkdir [폴더명]
- cd [폴더명]
- git init
- git clone [깃 저장소 주소]
- cd [이동할 디렉토리]
- code .



이 방법은 github에서 new repository를 생성하고,  
빈 프로젝트를 내 local(내 PC) 로 가져오는 방법임

## 방법2. Local과 Github Repository 연결하기

- **cd [폴더명]**

- **git init** : 버전관리 시작하겠다고 등록하는 것

- **git remote add origin [깃 저장소 주소]**

: 원격 저장소 주소를 origin이라는 이름으로 추가

- **git remote -v** : remote repository 조회

- **code .**



**Local(내 PC)에 이미 프로젝트 구조가 만들어져 있을 때,  
github의 new repository와 연결하는 방법임**

# VSCode 확장 프로그램 설치



## Auto Commit Message

A VS Code extension to generate a smart commit message based on file changes

MichaelCurrin



## GitLens — Git supercharged

Supercharge Git within VS Code — Visualize code authorship at a glance via Git blame annotations and CodeL...

 GitKraken

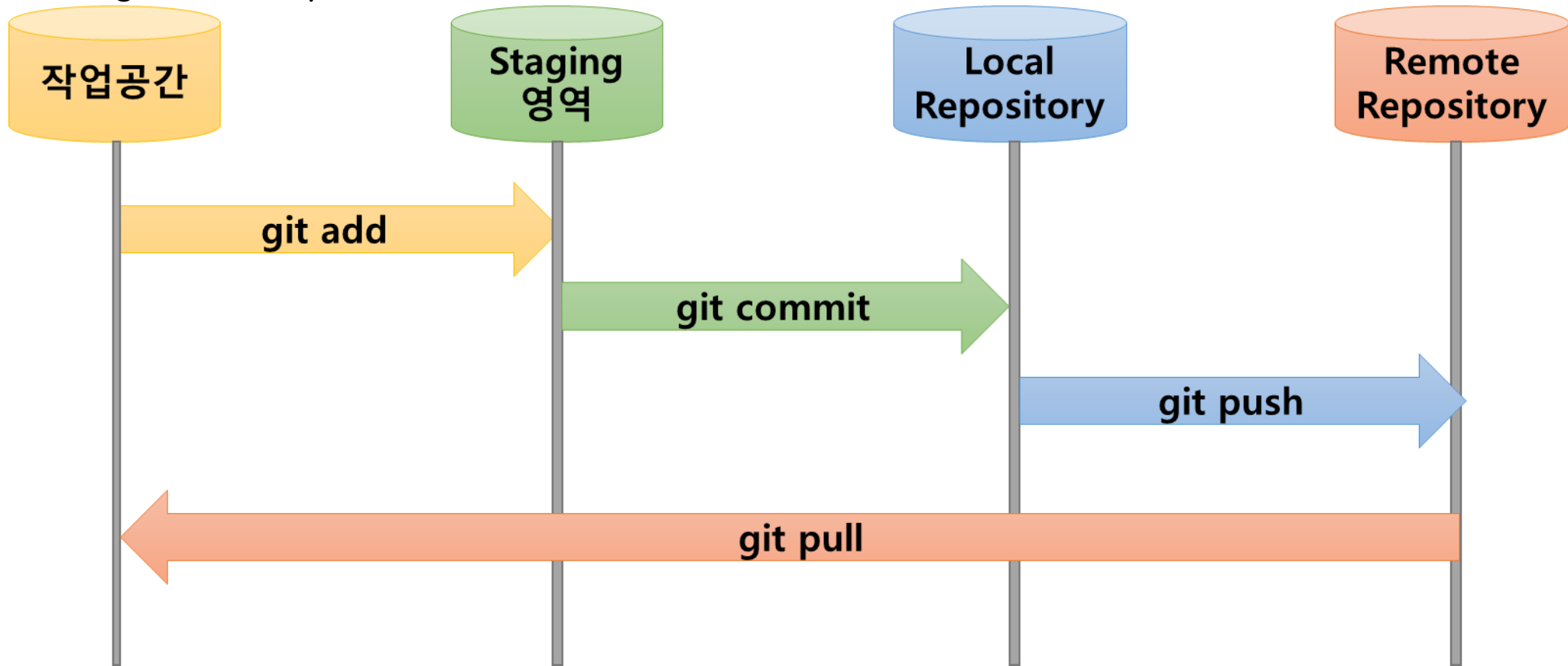
🕒 32ms



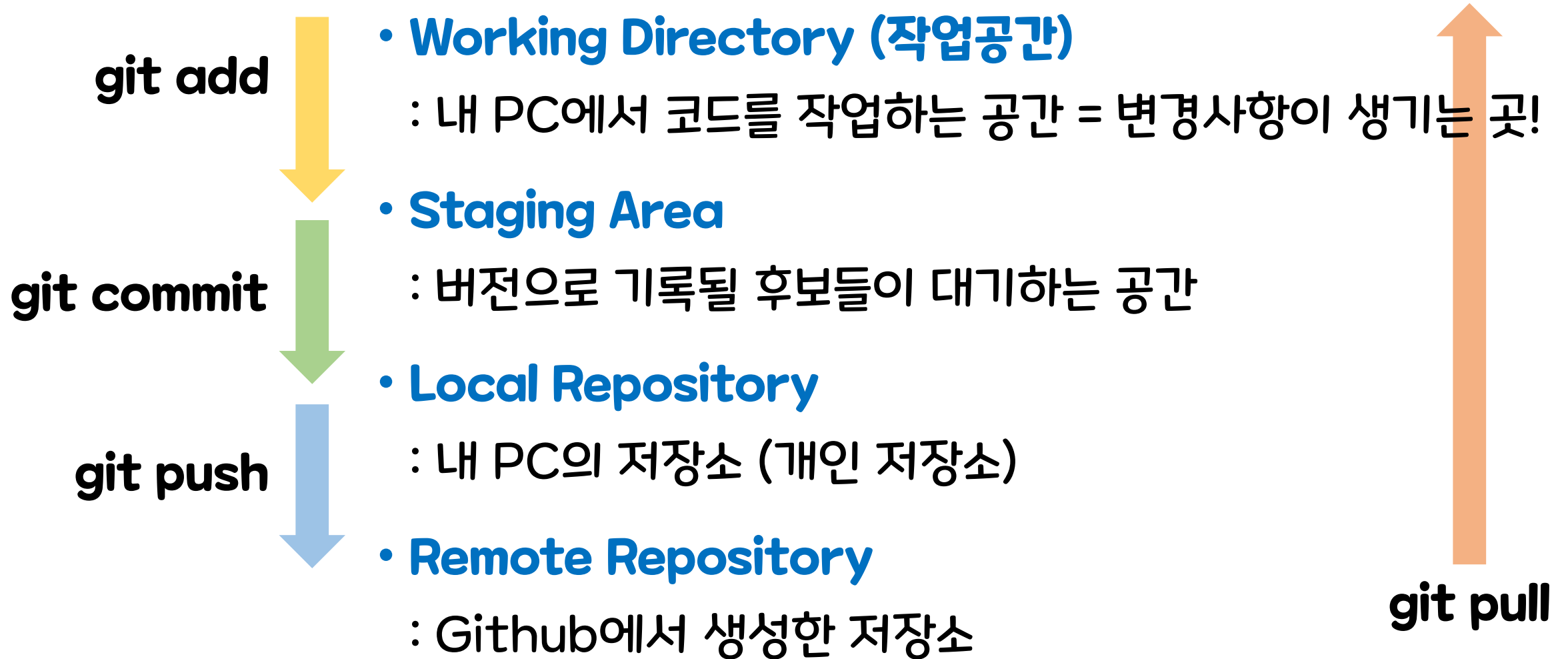
# Git 사용 세팅 끝!

# Git 흐름 이해하기

= working directory



# [잠깐!] 용어 정리





# Git 사용해보기

“ 그동안 작성한 수업 폴더를 Github에 올려보자! ”

- **git status** (지금 상황은?)
- **git add .** (working directory -> staging area)
- **git status** (지금 상황은?)
- **git commit -m “study: first commit”**  
(staging area -> local repository)
- **git log** (지금까지의 커밋 기록은?)
- **git push origin main** (local repository -> remote repository)

# 간단 실습 Time

Bootstrap 실습 코드에서  
Bootstrap Components에 있는 기능 추가하여,  
**add -> commit -> push까지 진행해보기**

# Git Skill Up!

# Branch(브랜치)

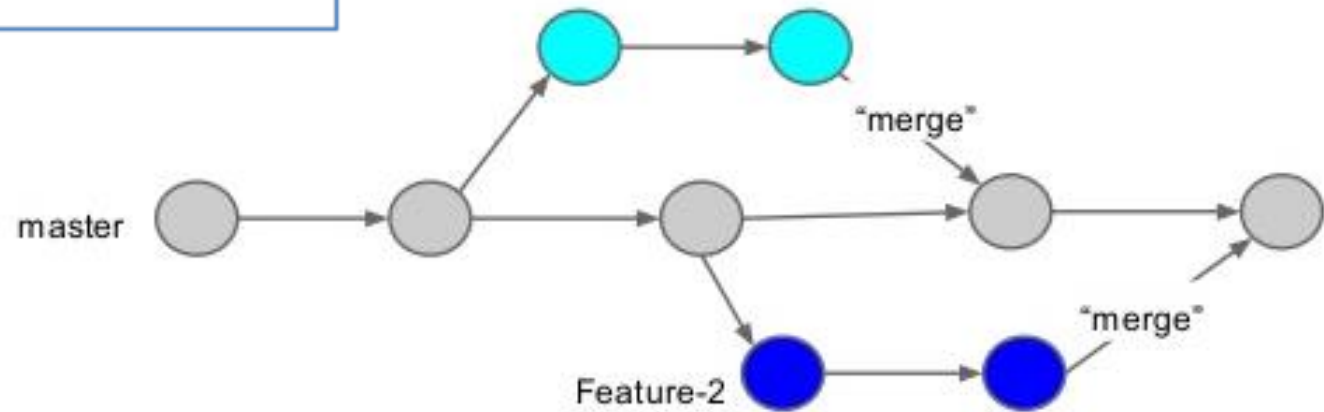
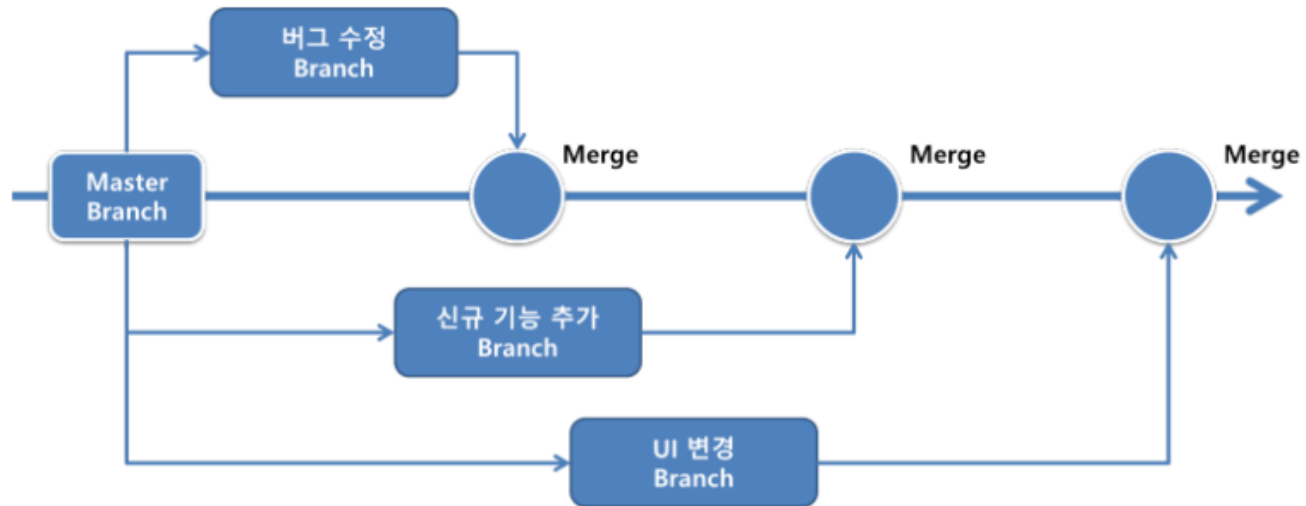


# Branch

독립적으로 어떤 작업을 하기 위해 필요한 개념

Ex) A라는 사람이 “로그인” 기능을 만들고, B라는 사람이 “버그 수정” 을 할 때  
A와 B는 **최초 Branch에서 파생한 각각의 Branch**를 만들어 작업을 진행하고  
최초 Branch 로 Merge를 통해 각자가 작업한 것을 합칠 수 있다.

# Branch



# Branch 생성하기

```
git branch # local branch 목록 확인
```

```
git branch "브랜치명" # 현재 branch에서 새로운 branch 생성
```

```
git checkout "전환 브랜치명" # branch 이동
```

```
git branch -d "브랜치명" # bran 삭제
```

```
# (단, 삭제할 branch가 현재 branch에 합쳐져 있을 경우에만)
```

# 새로운 branch 생성 & 이동 동시에

```
git checkout -b "만들 브랜치명"
```



# Merge

git branch를 다른 branch로 **합치는 과정**

Ex) a 브랜치에 b 브랜치를 합치고 싶은 경우

```
git checkout a
```

← a 브랜치로 이동

```
git merge b
```

← b 브랜치와 merge 진행

# Merge

**Case1.** a 브랜치와 b 브랜치에서 서로 다른 파일을 수정했을 때

a 브랜치

b 브랜치

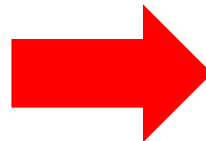
이름

123.txt

이름

987.txt

```
git checkout a
git merge b
```



a 브랜치

이름

123.txt

987.txt

알아서 합쳐진다! 😊👍

# Merge

Case2. 서로 같은 파일에서 다른 부분을 수정했을 때

원본

```
<html>
  <head>
    <title>원본</title>
  </head>
  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

a 브랜치

```
<html>
  <head>
    <title>a에서 수정</title>
  </head>
  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

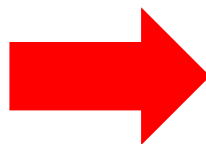
b 브랜치

```
<html>
  <head>
    <title>원본</title>
  </head>
  <body>
    <h1>b에서 수정</h1>
  </body>
</html>
```

# Merge

Case2. 서로 같은 파일에서 다른 부분을 수정했을 때

```
git checkout a
git merge b
```



a 브랜치

```
<html>
  <head>
    <title>a에서 수정</title>
  </head>
  <body>
    <h1>b에서 수정</h1>
  </body>
</html>
```

알아서 합쳐진다! 😊 🙌

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

원본

```
<html>
  <head>
    <title>원본</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

a 브랜치

```
<html>
  <head>
    <title>a에서 수정</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

b 브랜치

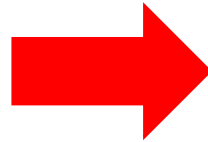
```
<html>
  <head>
    <title>b에서 수정</title>
  </head>

  <body>
    <h1>반가워요</h1>
  </body>
</html>
```

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```
git checkout a  
git merge b
```



Conflict  
( 충돌 )



**수동으로 해결해야 한다!!**

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```

You, 3초 전 | 1 author (You)
1  <html>
2  |   <head>
3  |   <title>a에서 수정</title>
4  |   </head>
5  |   <body>
6  |   <h1>반가워요</h1>
7  |   </body>
8  |   </html>
9  |
10 |
11 |
12 |
13 |
You, 2분 전 • study: Prepare git merge lesson

Accept Current Change
<title>a에서 수정</title>
</head>
<title>b에서 수정</title>
>>>>> b (Incoming Change)
</head>
<body>
<h1>반가워요</h1>
</body>
</html>

```

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```

You, 3초 전 | 1 author (You)
1  <html>
2  |   <head>
3  |   Accept Current Change Accept Incoming Change Accept Outgoing Change
4  |   <<<<<< HEAD (Current Change)
5  |   |   <title>a에서 수정</title>
6  |   =====
7  |   |   <title>b에서 수정</title>
8  |   >>>>>> b (Incoming Change)
9  |   |   </head>
10 |   |
11 |   |   <body>
12 |   |   |   <h1>반가워요</h1>
13 |   |   </body>
    |   </html>
    You, 2분 전 • study: Prepare git merge lesson
  
```

```

merge
<head>
|   <title>b에서 수정</title>
</head>
  
```



# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```

You, 3초 전 | 1 author (You)
1  <html>
2  |   <head>
3  |   Accept Current Change | Accept Incoming Change | Accept Both Changes | C
4  |   <<<<<< HEAD (Current Change)
5  |   |   <title>a에서 수정</title>
6  |   =====
7  |   |   <title>b에서 수정</title>
8  |   >>>>>> b (Incoming Change)
9  |   |   </head>
10 |   |   <body>
11 |   |   |   <h1>반가워요</h1>
12 |   |   </body>
13 |   </html>
You, 2분 전 • study: Prepare git merge lesson

```

```

<head>
|   <title>a에서 수정</title>
|   <title>b에서 수정</title>
</head>

```

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```

You, 3초 전 | 1 author (You)
1 <html>
2   <head>
3   <<<<<< HEAD (Current Change)
4   <title>a에서 수정</title>
5   =====
6   <title>b에서 수정</title>
7   >>>>>> b (Incoming Change)
8   </head>
9
10  <body>
11  | <h1>반가워요</h1>
12  </body>
13 </html>
You, 2분 전 • study: Prepare git merge lesso

<> test.html
1 <html>
2   <head>
3-  <title>a에서 수정</title>
4   </head>
5
6   <body>
7   | <h1>반가워요</h1>
8   </body>
9  </html>

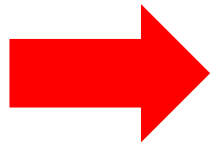
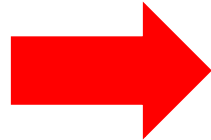
<> test.html: Current Changes ↔ Incoming Changes X
1 <html>
2   <head>
3+  <title>b에서 수정</title>
4   </head>
5
6   <body>
7   | <h1>반가워요</h1>
8   </body>
9  </html>

```

# Merge

Case3. 서로 같은 파일이고 같은 부분을 수정했을 때

```
git checkout a
git merge b
```



해결 후 파일 저장 -> **add** -> **commit** -> **push**

수동으로 코드를 머지하면 된다! 😁 🙌

**대형 프로젝트를 위해 알아 두면 좋은 지식!**

# Branch의 종류

main(master)

develop

feature

release

hotfix

# Branch – main(master)

- 제품으로 출시될 수 있는 브랜치
- 배포(Release) 이력을 관리하기 위해 사용
- 배포 가능한 상태만을 관리하는 브랜치

# Branch - develop

- 다음 출시 버전을 개발하는 브랜치
- 기능 개발을 위한 브랜치들을 병합하기 위해 사용
- 평소 개발을 진행하는 브랜치

# Branch - feature

- 기능 개발을 진행하는 브랜치
- 새로운 기능 개발 및 버그 수정을 할 때마다 'develop' 에서 분기
- 공유할 필요가 없어 로컬에서 진행 후 develop 에 merge 해 공유
- 이름 : feature/~~

```
git checkout -b feature/이름 develop
```

```
/* 개발_이름 */
```

```
|
```

```
git checkout develop
```

```
git merge --no--ff feature/이름
```

```
git branch -d feature/이름
```

```
git push origin develop
```



# Branch – release

- 출시 버전을 준비하는 브랜치
- 배포를 위한 전용 브랜치
- 이름 : release-0.0

```
git checkout -b release-1.2 develop
```

```
/* 배포 사이클 */
```

```
git checkout master
```

```
git merge --no--ff release-1.2
```

```
git tag -1 1.2
```

```
git checkout develop
```

```
git merge --no--ff release-1.2
```

```
git branch -d release-1.2
```

# Branch – hotfix

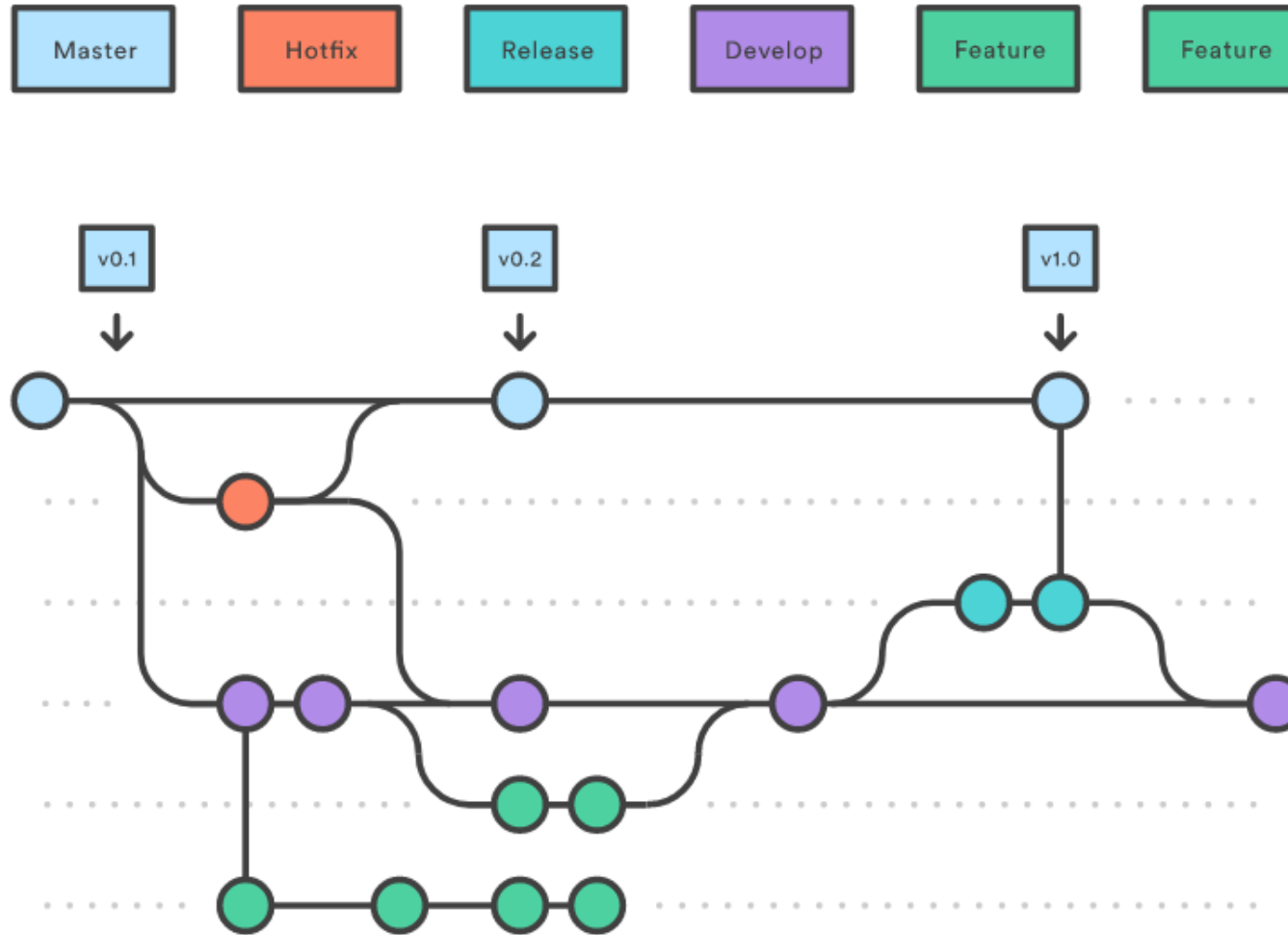
- 출시 버전에서 발생한 버그 수정 브랜치
- 배포한 버전에 긴급하게 수정해야 할 필요가 있는 경우 사용
- Master에서 분기
- 이름 : hotfix-0.0.0

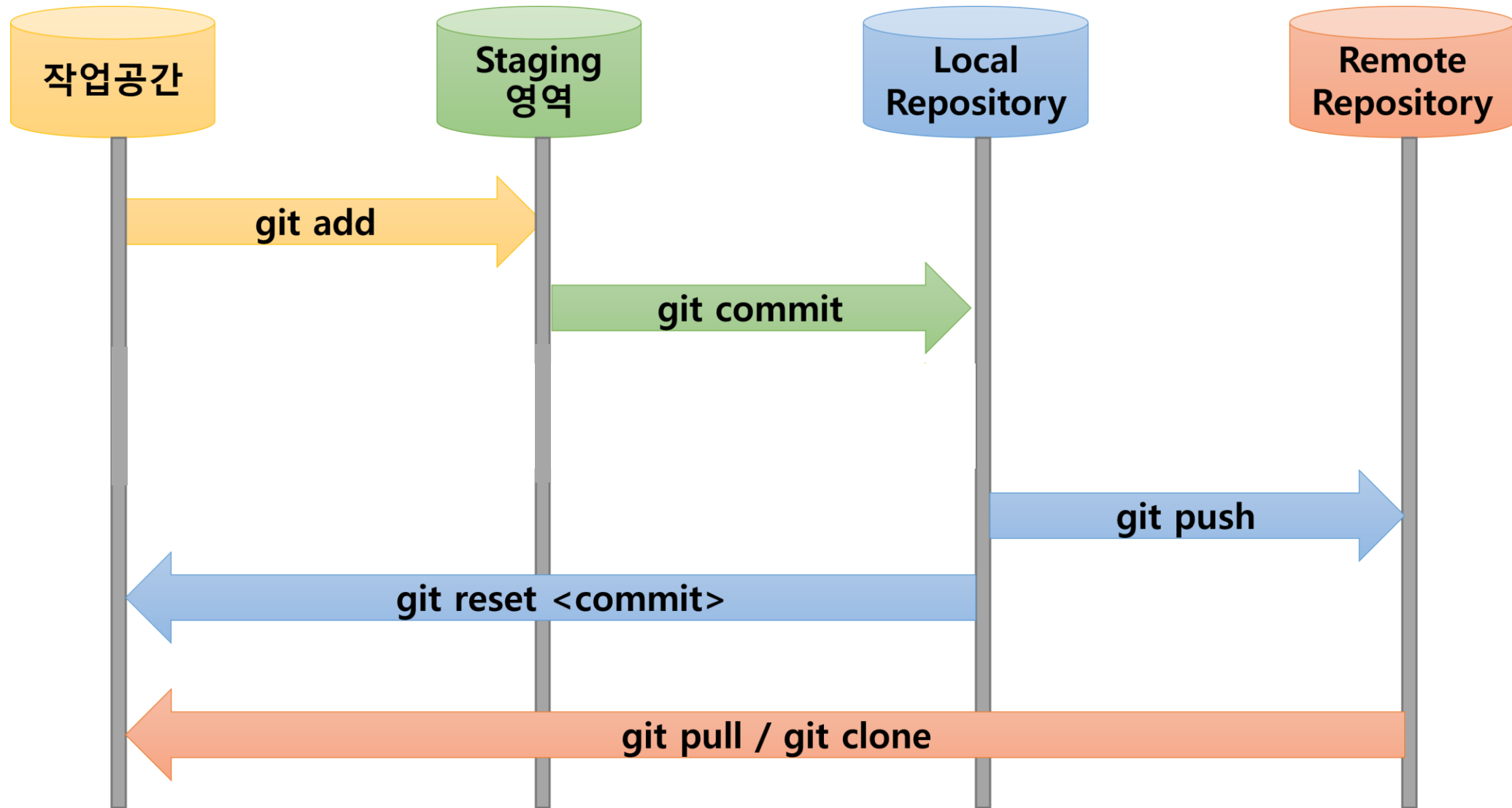
```
git checkout -b hotfix-1.2.1 master
|
/* 문제 수정 */

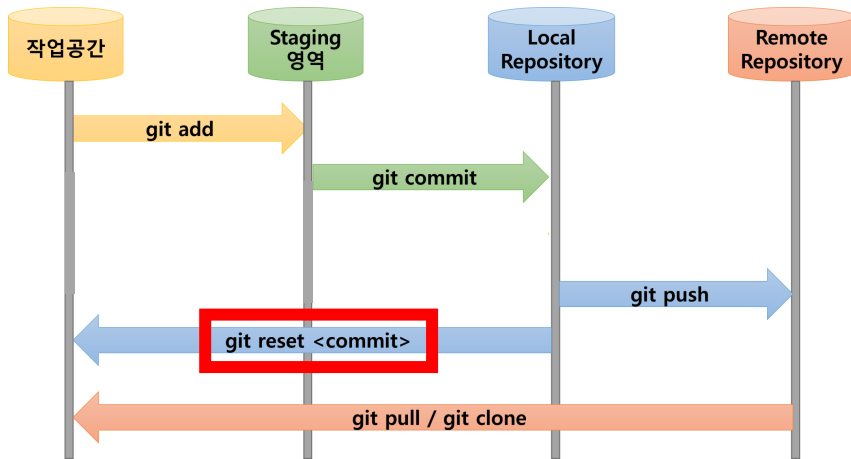
git checkout master
git merge --no--ff hotfix-1.2.1
git tag -a 1.2.1

git checkout develop
git merge --no--ff hotfix-1.2.1
git branch -d hotfix-1.2.1
```

# Branch 종류







## 가장 최근 커밋 취소

```
git reset HEAD^
```

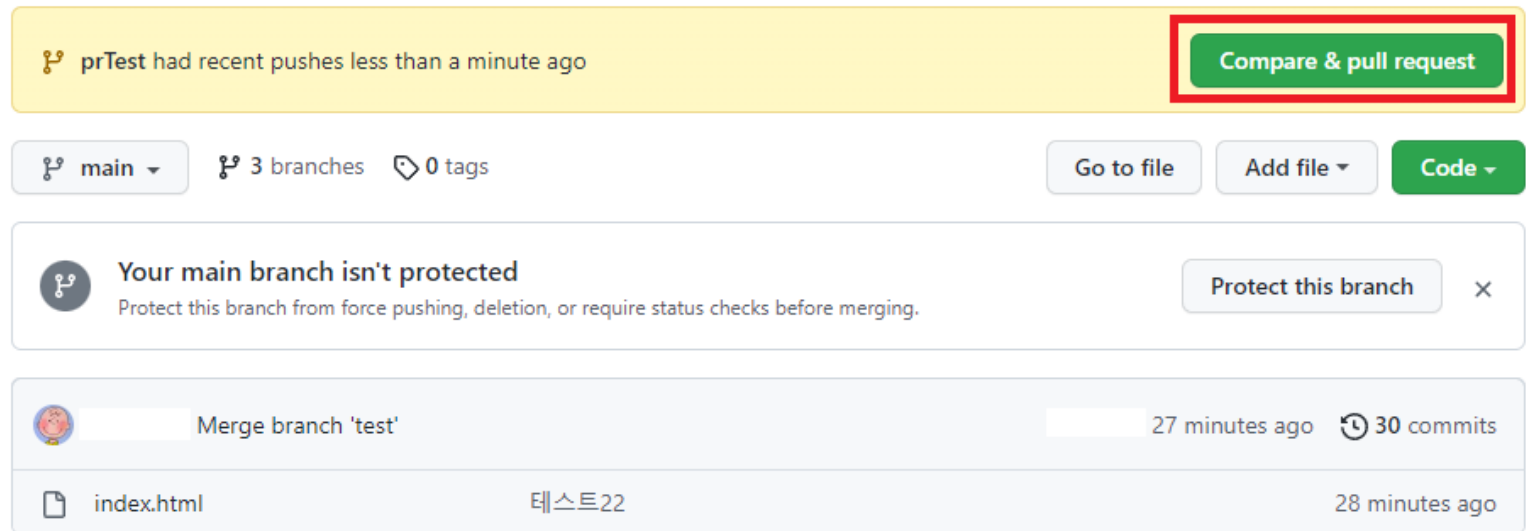
## 특정 커밋 취소

```
Linda@Linda MINGW64 /e/Development/github/SeSAC4_
$ git log
commit df72409a0cd5864061c73a28a3e4896d08f6e2cb (HEAD)
```

```
git reset --hard df72409a0cd5864061c73a28a3e4896d08f6e2cb
```

# Pull Request

- Push 권한이 없는 오픈 소스 프로젝트에 기여할 때 많이 사용함.
- “ 내가 수정한 코드가 있으니 내 branch를 가져가 검토 후 병합(merge) 해주세요!! ”
- 당황스러운 코드 충돌을 줄일 수 있음



# Pull Request

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main
←
compare: prTest
✓ Able to merge. These branches can be automatically merged.

github pr 테스트

Write

Preview

H B I ≡ <> ↻ ≡ ≡ ☑ @ ↻ ↶

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

## Merge pull request

You can also [open this pull request](#)

### ✓ Create a merge commit

All commits from this branch will be added to the base branch via a merge commit.

### Squash and merge

The 1 commit from this branch will be added to the base branch.

### Rebase and merge

The 1 commit from this branch will be rebased and added to the base branch.

# Pull Request

test3에서 코드 수정 #1

Merged

merged 1 commit into `main` from `test3` 2 minutes ago

merge 완료!!

Conversation 0

Commits 1

Checks 0

Files changed 1

commented 4 minutes ago

Owner

코드 수정했으니까 검토 후 머지해주세요!!

test3에서 코드 수정 57eae68

merged commit into `main` 2 minutes ago

Revert

Pull request successfully merged and closed

You're all set—the `test3` branch can be safely deleted.

Delete branch



# .gitignore

## .gitignore?

- Git 버전 관리에서 **제외할 파일 목록을 지정**하는 파일
- Git 관리에서 특정 파일을 제외하기 위해서는 git에 올리기 전에 .gitignore에 파일 목록을 미리 추가해야 한다.

# .gitignore

**\*.txt** → 확장자가 txt로 끝나는 파일 모두 무시

**!test.txt** → test.txt는 무시되지 않음.

**test/** → test 폴더 내부의 모든 파일을 무시 ( b.exe와 a.exe 모두 무시 )

**/test** → (현재 폴더) 내에 존재하는 폴더 내부의 모든 파일 무시 ( b.exe무시 )

```
(base) [07:25 PM] cwjcsk:~/99_test/99_tmp$ tree -a
.
├── .gitignore
├── test
│   └── b.exe
└── tmp
    ├── test
    │   └── a.exe
```

3 directories, 3 files