
Open source Software
오픈소스 소프트웨어
11. 파일 삭제 rm과 복원 restore

학습 개요

1. $ rm f
   - 리눅스 파일 삭제
2. $ git rm f
3. $ git rm --cached f
4. $ git restore f
5. $ git restore --staged f

학습 목표

1. 깃의 3 영역 중에서 작업 디렉토리와 스테이징 영역에서 파일을 삭제할 수 있다.
2. 특정 파일에서 스테이징 영역의 상태를 작업 디렉토리에 복원할 수 있다.
3. 특정 파일에서 깃 저장소 상태를 스테이징 영역에 복원할 수 있다.
4. 특정 파일에서 깃 저장소 상태를 작업 디렉토리에 복원할 수 있다.
5. 특정 파일에서 깃 저장소 상태를 스테이징 영역과 작업 디렉토리에 함께 복원할 수 있다.

LESSON 01

파일 삭제 rm

1 파일 삭제 rm

파일 삭제

리눅스 명령 파일 삭제

- $ rm [file]
  - 작업 디렉토리에서 file 삭제

깃 명령 파일 삭제

- $ git rm [file]
  - 작업 디렉토리와 스테이징 영역에서 모두 file 삭제
    - ➡ 다음 커밋에서 지정한 file을 삭제하겠다는 의미
    - ➡ Tracked 상태의 파일을 제거하여 Untracked 상태로 만듦
- $ git rm --cached [file]
  - 스테이징 영역에서 file 삭제 ➡ 작업 디렉토리에서는 삭제되지 않음
  - $ git ls-files 결과에서 보이지 않음 ➡ 기본적으로 스테이징 영역의 파일 목록을 표시

1 파일 삭제 rm

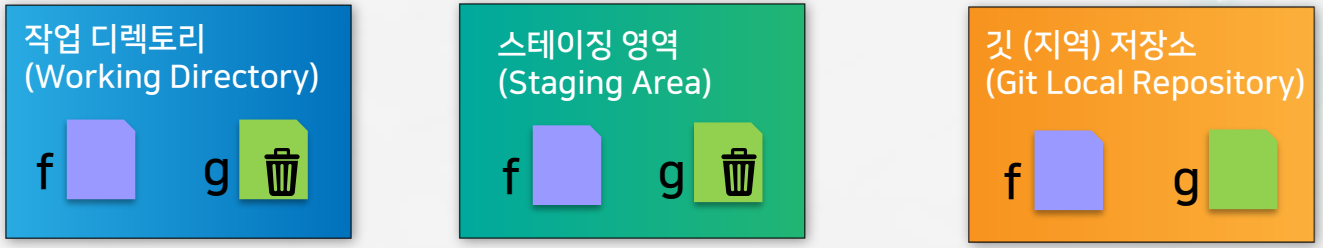
작업 디렉토리와 스테이징 영역에서 파일 삭제

현재 상태

- 3영역(WD와 SA, GR) 모두에 파일 f,g가 있는 상태

작업 디렉토리와 스테이징 영역세서 파일 g 삭제

- $ git rm g



1 파일 삭제 rm

$ git rm g 이후 상태

삭제된 이후 상태

배시

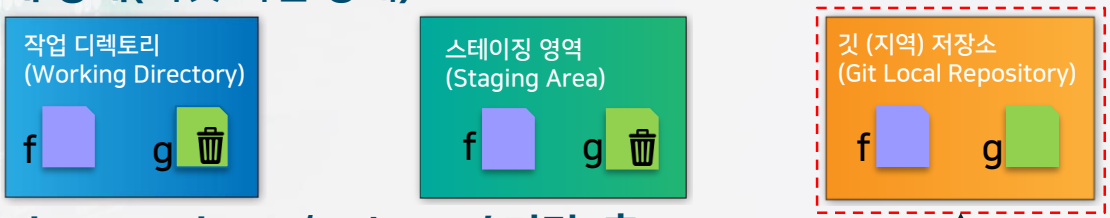
```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ ls
f

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git ls-files
f

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    g

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status -s
D  g
```

녹색은 스테이징 영역의 표시이며 깃 저장소와 비교해 삭제됨을 의미

1 파일 삭제 rm

$ git commit –m 'Delete g'

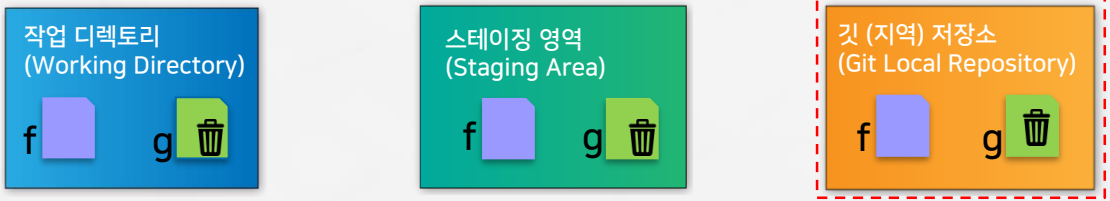
현재 상태(커밋 이전 상태)



$ git commit –m 'Delete g' 커밋 후

- 파일 g가 삭제된 상태에서 커밋
  - 이전 커밋과 차이는 g가 삭제된 것, 커밋 이후 log show

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/grm (main)
$ git show
commit 49b0613f77ea47f169ed01b56af5e92b0d7f7dbb (HEAD -> main)
Author: ai7dnn <ai7dnn@gmail.com>
Date:   Sat Jan 21 15:57:23 2023 +0900

    Delete g

diff --git a/g b/g
deleted file mode 100644
index e69de29..0000000
```

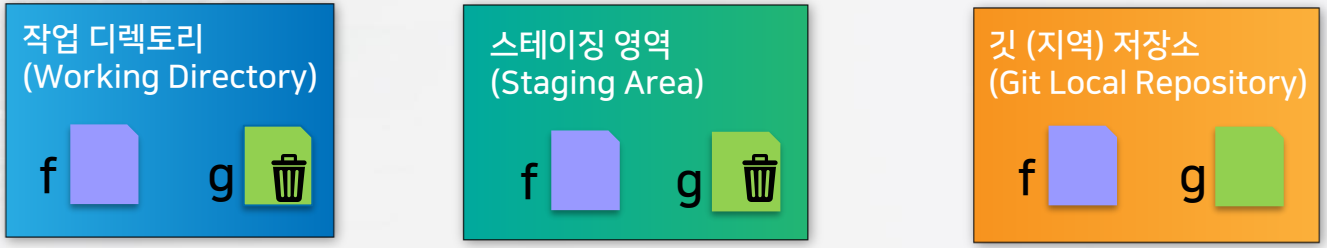


1 파일 삭제 rm

스테이징 영역에서만 파일 삭제

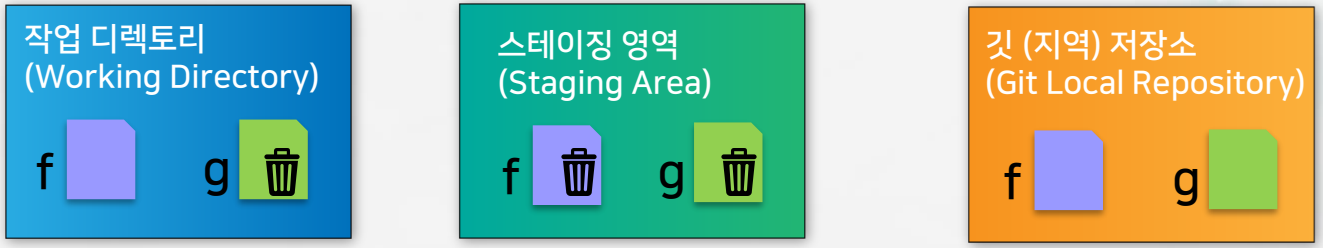
현재 상태

- WD와 SA에 파일 f가 있는 상태



스테이징 영역에서 파일 f 삭제

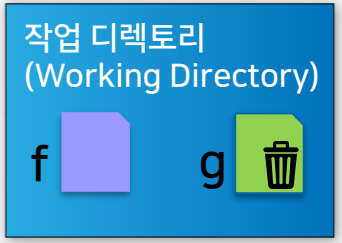
- $ git rm --cached f



1 파일 삭제 rm

$ git rm --cached f 이후 상태

현재 상태




```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ ls
f

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git ls-files

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status
On branch main
Changes to be committed:
 (use "git restore --staged <file>..." to unstage)
      deleted:    f
      deleted:    g

Untracked files:
 (use "git add <file>..." to include in what will be committed)
     f
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status -s
D  f
D  g
?? f
```

??은 Untracked를 의미하며 스테이징 영역에서 파일이 삭제되어 추적되지 않는 파일이 된 상태임

1 파일 삭제 rm

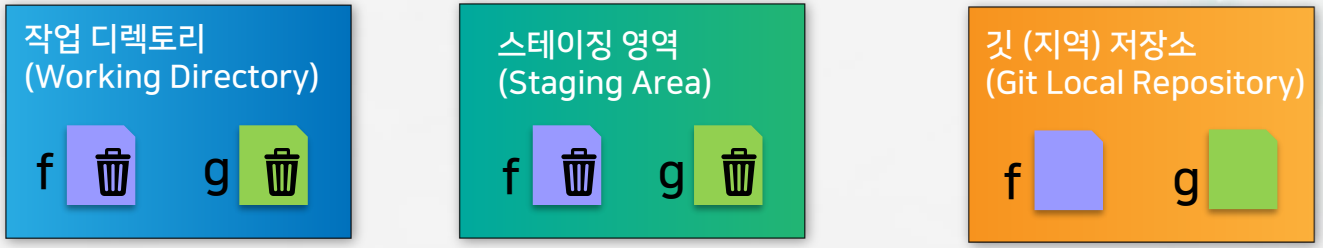
작업 디렉토리만 파일 삭제

현재 상태

- WD에만 파일 f가 있는 상태

작업 디렉토리에서 파일 f 삭제, 리눅스 명령 rm

- $ rm f

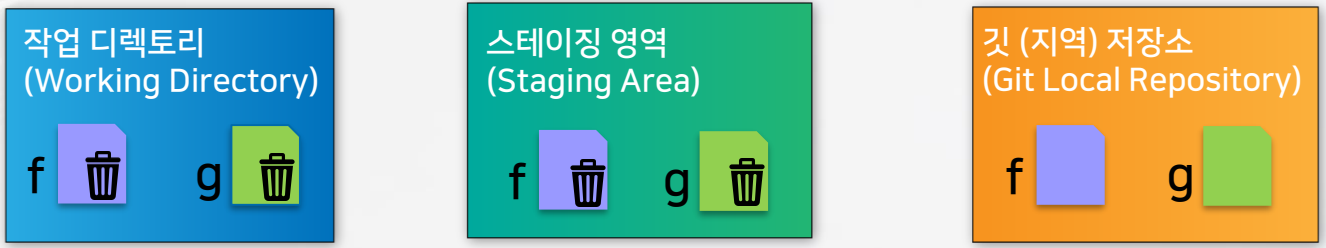


1 파일 삭제 rm

$ rm f 이후 상태

현재 상태

- WD와 SA에서 파일 f, g가 모두 삭제된 상태



배시

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ ls

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git ls-files

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    f
        deleted:    g


PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rm (main)
$ git status -s
D  f
D  g
```

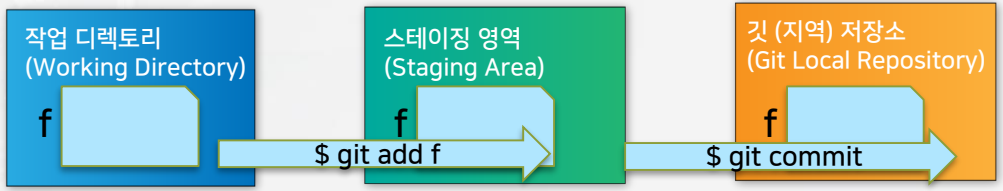
LESSON 02

파일 복원 restore

3 영역에서 파일이 모두 다른 상태 준비

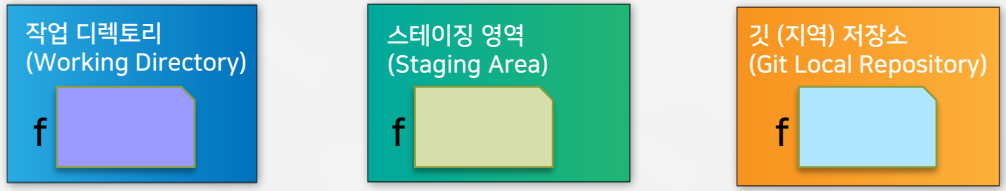
커밋 과정 복습, 마지막은 2 영역의 내용이 모두 다른 상태

- add, commit



- 파일 f 수정 후 다시 add

- 다시 파일 f 수정

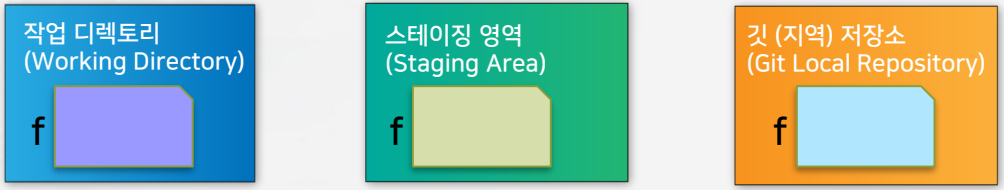


2 파일 복원 restore

$ git restore [file]

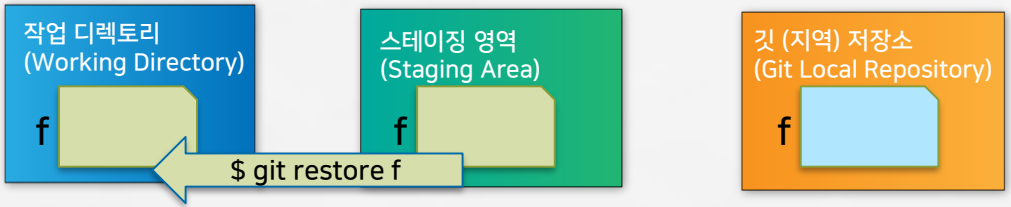
현재 상태

- 3 영역에서 파일 f가 모두 다른 상태



작업 디렉토리의 파일 f를 스테이징 영역의 파일 상태로 복구

- 작업 디렉토리에 있던 f 내용이 사라지므로 유의
- 3 영역에서 파일 f가 작업 디렉토리와 스테이징 영역이 같은 상태가 됨
  - $ git restore f

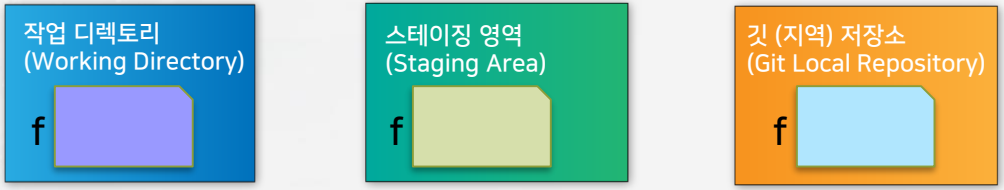


2 파일 복원 restore

$ git restore --staged [file]

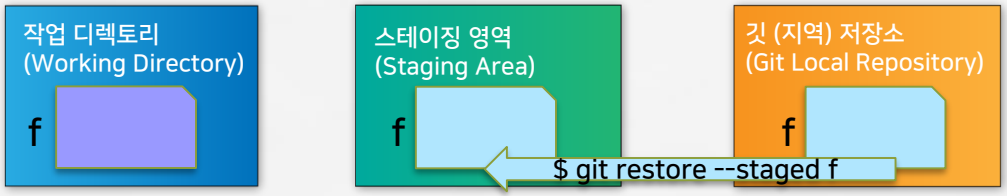
현재 상태

- 3 영역에서 파일 f가 모두 다른 상태



깃 저장소의 최신 커밋 상태의 파일 f를 스테이징 영역에 복구

- 스테이징 영역에 있던 f 내용이 사라지므로 유의
- 3 영역에서 스테이징 영역의 파일 f가 깃 저장소 영역과 같은 상태가 됨
  - $ git restore --staged f

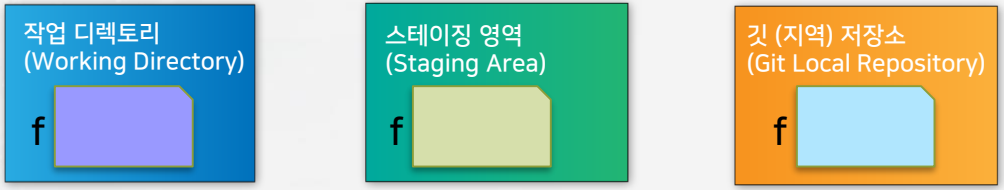


2 파일 복원 restore

깃 저장소 내용으로 한번에 모두 복원

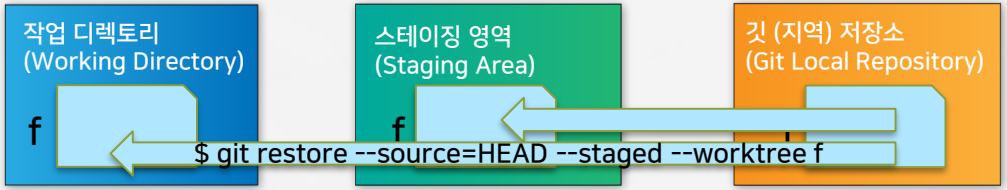
현재 상태

- 3 영역에서 파일 f가 모두 다른 상태



깃 저장소의 최신 커밋 상태의 파일 f를 작업 디렉토리와 스테이징 영역에 한번에 복구

- 파일 f가 현재 커밋 상태의 내용으로 작업 디렉토리와 스테이징 영역 모두 같은 상태가 됨
  - $ git restore --source=HEAD --staged --worktree f

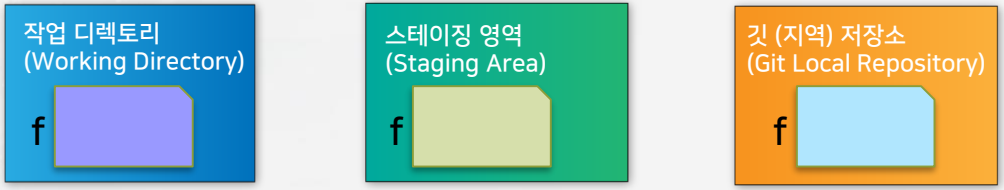


2 파일 복원 restore

HEAD 내용으로 작업 디렉토리 복원

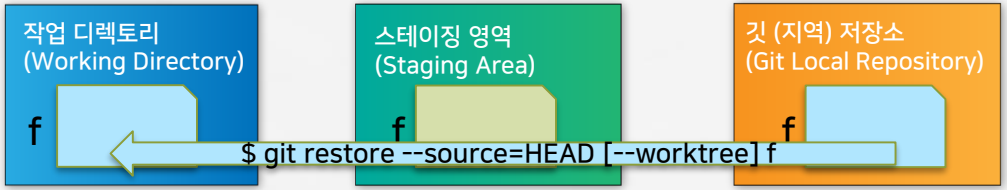
현재 상태

- 3 영역에서 파일 f가 모두 다른 상태



깃 저장소의 최신 커밋 상태의 파일 f를 작업 디렉토리에 복원

- 파일 f가 현재 커밋 상태의 내용으로 작업 디렉토리에 복사되어 동일하게 됨
  - $ git restore --source=HEAD f



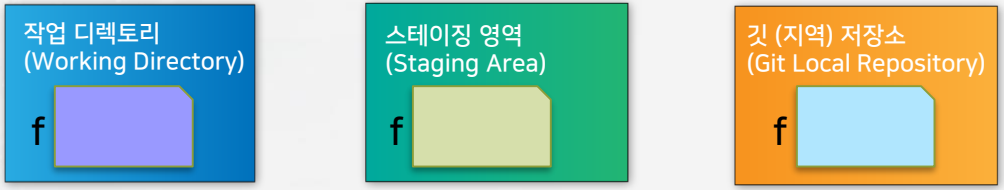
2 파일 복원 restore

$ git restore --source=HEAD^ --staged [file]

복구에 사용되는 파일이 위치한 커밋을 지정, HEAD~은 이전 커밋의 내용으로 복구한다는 의미

현재 상태

- 3 영역에서 파일 f가 모두 다른 상태

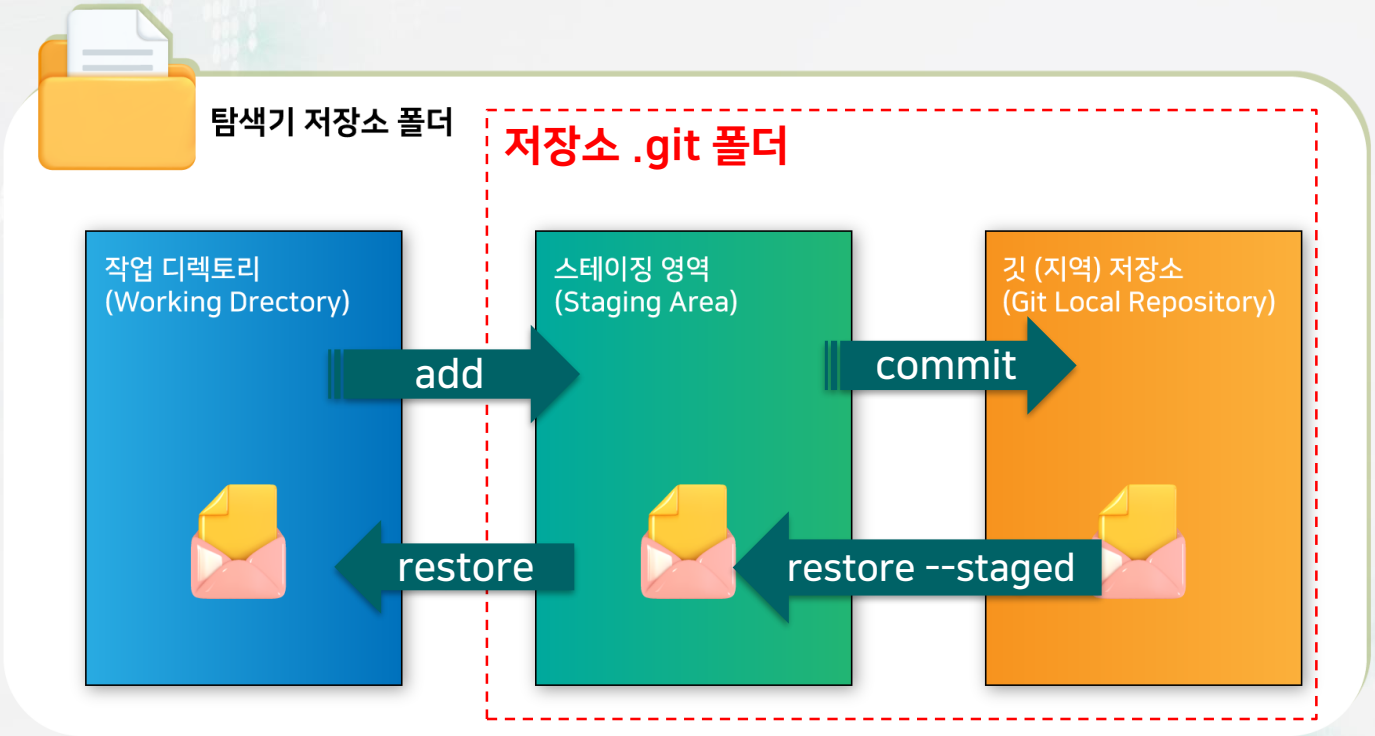


깃 저장소의 최신 커밋 상태의 파일 f를 스테이징 영역에 복구

- 스테이징 영역에 있던 f 내용이 사라지므로 유의
- 3 영역에서 스테이징 영역의 파일 f가 깃 저장소 영역과 같은 상태가 됨
  - $ git restore --staged f

2 파일 복원 restore

복원 restore 정리



Summary

작업 디렉토리에서 파일 삭제

- $ rm f

작업 디렉토리와 스테이징 영역에서 파일 삭제

- $ git rm f

스테이징 영역에서만 파일 삭제

- $ git rm --cached f

스테이징 영역의 상태를 작업 디렉토리에 복원

- $ git restore f

Summary

깃 저장소 상태를 스테이징 영역에 복원

- $ git restore --staged f

깃 저장소 상태를 작업 디렉토리에 복원

- $ git restore --source=HEAD --worktree f
- $ git restore --source=HEAD f

깃 저장소 상태를 스테이징 영역과 작업 디렉토리에 함께 복원

- $ git restore --source=HEAD --staged --worktree f

--staged는 -S
--worktree는 -W
짧게 가능(둘 다 대문자)