

Open source Software

오픈소스 소프트웨어

12. [실습] 파일 diff와 rm, restore



학습 개요

1. 깃 3 영역의 파일을 비교
2. 커밋 간의 파일 비교
3. 파일 삭제와 복구



학습 목표

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

LESSON 01

파일 비교 diff





1 파일 비교 diff

[실습] 저장소 생성 1회 커밋

저장소 gfile, 파일 f

```
$ git init gfile
Initialized empty Git repository in C:/[smart git]/gfile/.git/
```

```
$ cd gfile
```

```
$ echo aaa > f
```

```
$ git add f
```

```
$ git commit -m A
[main (root-commit) 5afb35b] A
1 file changed, 1 insertion(+)
create mode 100644 f
```

```
$ git log --oneline
5afb35b (HEAD -> main) A
```

작업 디렉토리	스테이징 영역	깃 저장소
aaa	aaa	aaa



1 파일 비교 diff

파일 수정 후 비교

작업 디렉토리만 다른 상태

```
$ echo bbb >> f
```

```
$ git status -s
M f
```

```
$ git diff
diff --git a/f b/f
index 72943a1..dbec026 100644
--- a/f
+++ b/f
@@ -1,2 @@
aaa
+bbb
```

```
$ git diff --staged
```

```
$ git diff HEAD
diff --git a/f b/f
index 72943a1..dbec026 100644
--- a/f
+++ b/f
@@ -1,2 @@
aaa
+bbb
```

작업 디렉토리	스테이징 영역	깃 저장소
aaa bbb	aaa	aaa



1 파일 비교 diff

🔧 파일 추가 후 비교

☑️ 깃 저장소만 다른 상태

```
$ git add f
```

```
$ git status -s
M f
```

```
$ git diff
```

```
$ git diff --staged
diff --git a/f b/f
index 72943a1..dbee026 100644
--- a/f
+++ b/f
@@ -1,2 @@
aaa
+bbb
```

```
$ git diff HEAD
diff --git a/f b/f
index 72943a1..dbee026 100644
--- a/f
+++ b/f
@@ -1,2 @@
aaa
+bbb
```

작업 디렉토리	스테이징 영역	깃 저장소
aaa bbb	aaa bbb	aaa

1 파일 비교 diff

🔧 파일 커밋 후 비교

✅ 3 영역이 모두 동일

```
$ git commit -m B  
[main 7704d66] B  
1 file changed, 1  
insertion(+)
```

```
$ git log --oneline  
7704d66 (HEAD -> main) B  
5afb35b A
```

```
$ git status  
On branch main  
nothing to commit, working  
tree clean
```

```
$ git diff
```

```
$ git diff --staged
```

```
$ git diff HEAD
```

작업 디렉토리

aaa
bbb

스테이징 영역

aaa
bbb

깃 저장소

aaa
bbb

3 영역이 동일 상태,
빈 상태,
깨끗한 상태



1 파일 비교 diff

버전 간의 파일 비교

✓ HEAD~와 HEAD 비교

✓ 이번 버전과 작업 디렉토리, 스테이징 영역 비교

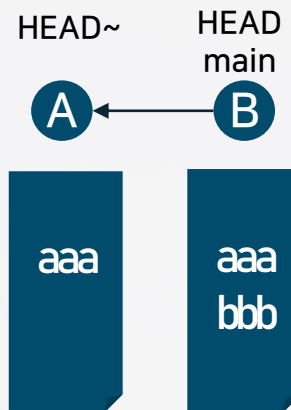
작업 디렉토리	스테이징 영역	깃 저장소
aaa bbb	aaa bbb	aaa bbb

```
$ git diff HEAD~ HEAD
diff --git a/f b/f
index 72943a1..dbee026 100644
--- a/f
+++ b/f
@@ -1,1,2 @@
   aaa
+bbb
```

```
$ git diff HEAD HEAD~
diff --git a/f b/f
index dbee026..72943a1 100644
--- a/f
+++ b/f
@@ -1,2,1 @@
   aaa
-bbb
```

```
$ git diff --staged HEAD~
diff --git a/f b/f
index 72943a1..dbee026 100644
--- a/f
+++ b/f
@@ -1,1,2 @@
   aaa
+bbb
```

```
$ git diff HEAD~
diff --git a/f b/f
index 72943a1..dbee026 100644
--- a/f
+++ b/f
@@ -1,1,2 @@
   aaa
+bbb
```



LESSON 02

파일 삭제 rm과 복구 restore



2 파일 삭제 rm과 복구 restore

명령 별칭(git alias) 생성

계속 사용하는 git 명령을 짧게 다른 이름으로 만드는 방법

- \$ git config --global alias.별칭이름 '원명령어 --긴옵션 -짧은옵션'
 - 설정 이후에는 다음 명령으로 가능 → \$ git 별칭이름

예

- \$ git config --global alias.ss 'status -s' → --global은 모든 프로젝트에서 공통적으로 사용하고자 하는 설정, 사용자의 홈디렉토리의 .gitconfig파일에 아래와 같이 추가
- 다음은 동일한 명령어
 - \$ git status -s
 - \$ git ss
- \$ git config --global alias.s status
 - \$ git s
- \$ git config --global alias.co checkout
 - \$ git co
- \$ git config --global alias.br branch
 - \$ git br
- \$ git config --global alias.c commit
 - \$ git c

[alias]

```
ss = status -s  
s = status  
co = checkout  
br = branch  
c = commit
```

2 파일 삭제 rm과 복구 restore

파일 g 생성 후 커밋

명령 별칭 ss 생성

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git config --global alias.ss 'status -s'
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ echo 111 > g
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git add f g
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git commit -m 1
[main 59bd4e1] 1
1 file changed, 1 insertion(+)
create mode 100644 g
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git log --oneline
59bd4e1 (HEAD -> main) 1
7704d66 B
5afb35b A
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ ls
f g
```

2 파일 삭제 rm과 복구 restore

🔧 파일 g 삭제

✅ 파일 g를 작업 디렉토리와 스테이징 영역에서 함께 삭제

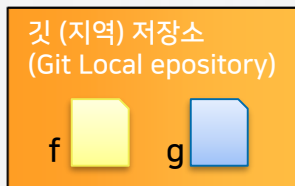
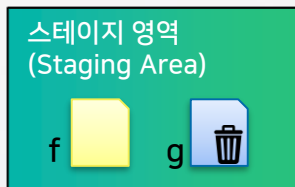
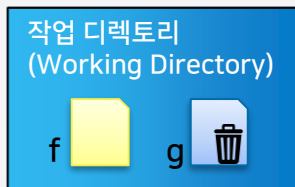
```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git ss # git status와 동일
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git rm g
rm 'g'
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git ss
D g
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ ls
f
```

```
파이썬@DESKTOP-8TN3J1L MINGW64 /c/[smart git]/gfile (main)
$ git ls-files
f
```



**\$ git rm g
이후 상태**

2 파일 삭제 rm과 복구 restore

현재 상태 커밋

파일 g를 삭제한 상태를 커밋

```
$ git ss
D g

$ git commit -m 'Delete g'
[main b5eb2ab] Delete g
1 file changed, 1 deletion(-)
delete mode 100644 g
```

```
$ git ss
```

```
$ ls
f
```

```
$ git ls-files
f
```

```
$ git log --oneline
b5eb2ab (HEAD -> main) Delete g
59bd4e1 1
7704d66 B
5afb35b A
```

```
$ git show
commit
b5eb2abb5296ecef3dde1921bd328748
81310162 (HEAD -> main)
Author: ai7dnn
<ai7dnn@gmail.com>
Date: Thu Jan 26 12:07:12 2023
+0900
```

Delete g

```
diff --git a/g b/g
deleted file mode 100644
index 58c9bdf..0000000
--- a/g
+++ /dev/null
@@ -1 +0,0 @@
-111
```

\$ git commit -m 'Delete g'
이후 상태

작업 디렉토리
(Working Directory)



스테이지 영역
(Staging Area)



깃 (지역) 저장소
(Git Local epository)



2 파일 삭제 rm과 복구 restore

스테이징 영역에서 파일 g 복원

✓ 파일 g는 HEAD~에 그 내용이 있는 상태,

• 스테이징 영역에 HEAD~의 파일 g를 복사

■ \$ git restore --source=HEAD~ --staged g

```
$ git ls-files
f
```

```
$ git restore --source=HEAD~ --staged g
```

```
$ git status
On branch main
```

Changes to be committed:

```
(use "git restore --staged <file>..." to unstage)
    new file:   g
```

Changes not staged for commit:

```
(use "git add/rm <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    deleted:    g
```

```
$ git ss
AD g
```

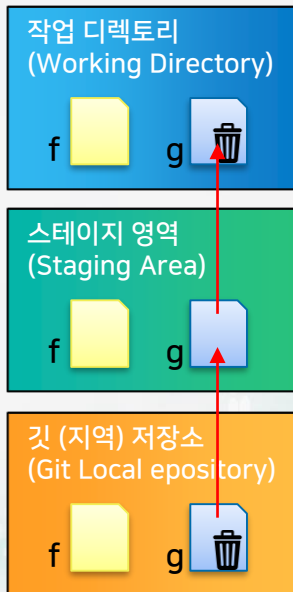
```
$ git ls-files
f
g
```

```
$ ls
f
```

만일 HEAD에 파일 g가 있다면
다음으로 가능
\$ git restore --staged g

깃 저장소와 비교해 없던 파일이
새로 추가 되었으므로 new file

스테이징 영역과 비교해 작업
디렉토리에는 삭제된 파일
이므로 deleted



2 파일 삭제 rm과 복구 restore

⚙️ 작업 디렉토리 파일 g 복원

☑️ 스테이징 영역의 파일 g를 작업 디렉토리에 복사

- \$ git restore g

```
$ git ls-files
```

```
f
```

```
g
```

```
$ ls
```

```
f
```

```
$ git restore g
```

```
$ ls
```

```
f g
```

```
$ git status
```

```
On branch main
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
new file:   g
```

```
$ git ss
```

```
A g
```

\$ git restore g
이전 상태

작업 디렉토리
(Working Directory)



스테이징 영역
(Staging Area)



깃 (지역) 저장소
(Git Local epository)



\$ git restore g
이후 상태

작업 디렉토리
(Working Directory)



스테이징 영역
(Staging Area)



깃 (지역) 저장소
(Git Local epository)



2 파일 삭제 rm과 복구 restore

스테이징 영역에서만 삭제

파일 g를 스테이징 영역에서만 삭제

- \$ git rm --cached g

```
$ git ls-files  
f  
g
```

```
$ git rm --cached g  
rm 'g'
```

```
$ git ls-files  
f
```

```
$ git status  
On branch main  
Untracked files:
```

(use "git add <file>..." to include in what will be committed)

g

nothing added to commit but untracked files present (use "git add" to track)

```
$ git ss  
?? g
```

스테이징 영역과 비교해 작업
디렉토리에만 있는 파일이므로
untracked file

\$ git rm --cached g
이후 상태



2 파일 삭제 rm과 복구 restore

작업 디렉토리에서만 삭제

파일 g를 작업 디렉토리에서만 삭제

리눅스 명령어

\$ rm g

\$ ls

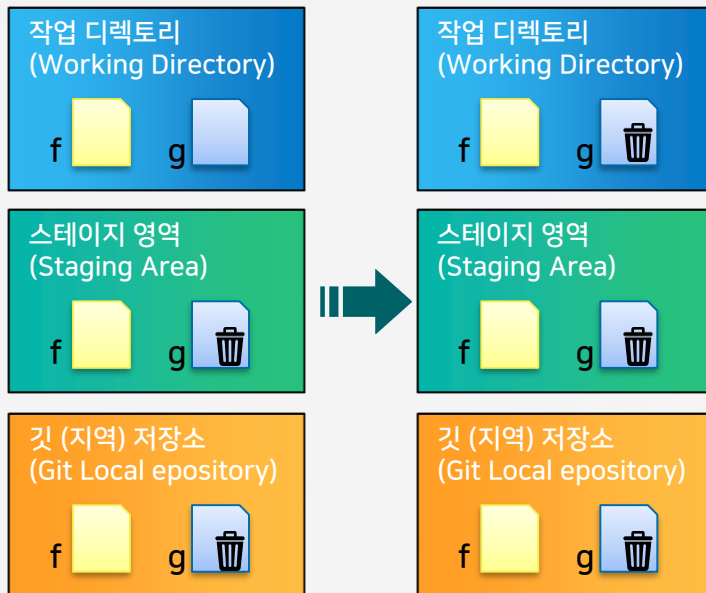
f g

\$ git ss

?? g

\$ rm g

\$ git ss



2 파일 삭제 rm과 복구 restore

두 영역에서 파일 f 삭제 후 작업 디렉토리만 복원

✓ 먼저 파일 f를 작업 디렉토리와 스테이징 영역에서 삭제

• \$ git rm f

✓ 깃 저장소의 최신 커밋 상태의 파일 f를 작업 디렉토리에 복원

• \$ git restore --source=HEAD f

```
$ git status
On branch main
nothing to commit, working tree clean
```

```
$ ls
f
```

```
$ git ls-files
f
```

```
$ git rm f
rm 'f'
```

```
$ git ss
D f
```

깃 저장소와 비교해
스테이징 영역에서
삭제된 파일이므로 **deleted**

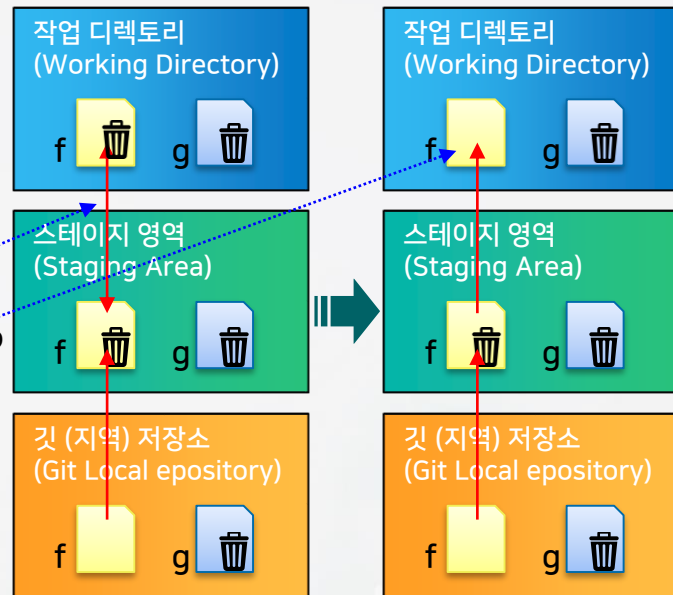
스테이징 영역과 비교해
작업 디렉토리에만
있는 파일이므로 **untracked**

```
$ git restore --source=HEAD
```

```
$ git ss
D f
?? f
```

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   f
```

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



2 파일 삭제 rm과 복구 restore

⚙️ 두 영역에서 파일 f 삭제 후 모두 복원

✓ 깃 저장소의 최신 커밋 상태의 파일 f를 두 영역에 모두 복원

• 먼저 다시 파일 f 삭제

■ \$ rm f

• 깃 저장소의 최신 커밋 상태의 파일 f를 두 영역에 모두 복원

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

```
$ ls
f
```

```
$ git ls-files
```

```
$ rm f
```

```
$ ls
```

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

```
$ ls
f
```

```
$ git ls-files
```

```
f
```

```
$ git ss
```

```
$ git status
```

```
On branch main
```

```
nothing to commit, working tree clean
```



Summary

» 스테이징 영역 기준으로 작업 디렉토리 파일 비교

- ◆ \$ git diff

» 깃 저장소 기준으로 스테이징 영역 파일 비교

- ◆ \$ git diff --staged HEAD

» 깃 저장소 기준으로 작업 디렉토리 파일 비교

- ◆ \$ git diff HEAD

