Undoing Things

> "되돌리기"

1. 파일 내용을 수정 전으로 되돌리기

> "Unmodifying a Modified File"
>
> - Modified 파일 되돌리기

- Working Directory에서 파일을 수정했다고 가정해봅시다.
  - 만약 이 파일의 수정 사항을 취소하고, 원래 모습대로 돌리려면 어떻게 해야 할까요?

(1) `git restore`

- `git restore <파일 이름>`의 형식을 사용합니다.
- git의 추적이 되고 있는, 즉 버전 관리가 되고 있는 파일만 되돌리기가 가능합니다.

1. 이미 버전 관리가 되고 있는 test.md 파일을 변경 후 저장(save)합니다.

```
# test.md

Hello
World <- "World"라는 새로운 내용 추가
--------------------------------------
이후 저장
```

2. `test.md`는 modified 상태가 되었습니다.

```
$ git status
On branch master
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:   test.md

no changes added to commit (use "git add" and/or "git commit -a")
```

3. git restore를 통해 수정 전으로 되돌립니다.

```
$ git restore test.md
```

```
# test.md

Hello
-------------------------------------
World가 삭제 되면서, 수정 전으로 되돌아감
```

> **[중요]**
>
> - 원래 파일로 덮어썼기 때문에 수정한 내용은 전부 사라짐
> - 한 번 git restore를 통해 수정을 취소하면, 해당 내용을 복원할 수 없음

> **[참고사항]**
>
> ```
> # Git(2.23.0 이전)에서는 아래 명령어를 사용했음
>
> $ git checkout -- <파일 이름>
> ```

---

2. 파일 상태를 Unstage로 되돌리기

> "Unstaging a Staged File"
>
> Staging Area와 Working Directory 사이를 넘나드는 방법

- git add를 통해서 파일을 Staging Area에 올렸다고 가정해봅시다.
  - 만약 이 파일을 다시 Unstage 상태로 내리려면 어떻게 해야 할까요?
  - 두 가지 상황으로 나누어 살펴보겠습니다.

(1) `git rm --cached`

1. 새 폴더에서 git 초기화 후 진행 `test.md` 파일을 생성하고 git add를 진행

```
$ touch test.md
```

```
$ git add test.md
```

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.md
```

2. Staging Area에 올라간 test.md를 다시 내리기(unstage)

```
$ git rm --cached test.md
rm 'test.md'
```

```
$ git status
On branch master

No commits yet

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

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

(2) `git restore --staged`

- 두번째 상황 전 사전 준비

```
$ git add .
$ git commit -m "first commit"
```

1. `test.md`의 내용을 변경하고 git add를 진행

```
# test.md 파일 변경 후
$ git add test.md
```

```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.md
```

2. Staging Area에 올라간 test.md를 다시 내리기(unstage)

```
$ git restore --staged test.md
```

```
$ git status
On branch master
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:   test.md

no changes added to commit (use "git add" and/or "git commit -a")
```

> **[중요]**
>
> Unstage로 되돌리는 명령어가 다른 이유는 무엇일까?
>
> 1. `git rm --cached <file>`
>   - **기존에 커밋이 없는 경우**
>   - “to unstage and remove paths only from the staging area”
> 2. `git restore --staged <file>`
>   - **기존에 커밋이 존재하는 경우**
>   - “the contents are restored from HEAD”

> **[참고사항]**
>
> ```
> # Git(2.23.0 이전)에서는 아래와 같은 명령어 사용
>
> $ git reset HEAD <파일 이름>
> ```

---

3. 바로 직전 완료한 커밋 수정하기

> 만약 A라는 기능을 완성하고 `"A 기능 완성"`이라는 커밋을 남겼다고 가정해봅시다.
>
> 그런데.. A 기능에 필요한 파일 중 1개를 빼놓고 커밋 했다는 걸 뒤늦게 깨달아 버렸습니다.
>
> 직전 커밋을 취소하고, 모든 파일을 포함해서 다시 커밋 하려면 어떻게 해야 할까요?

(1) `git commit --amend`

- 해당 명령어는 2가지 기능을 가지며 상황별로 동작이 다릅니다.
  1. 커밋 메시지만 수정하기
     - 언제? - 마지막으로 커밋하고 나서 수정한 것이 없을 때 (커밋하자마자 바로 명령어를 실행하는 경우)
  2. 이전 커밋 덮어쓰기
     - 언제? - Staging Area에 새로 올라온 내용이 있을 때

1-1. 커밋 메시지만 수정하는 경우

1. A 기능을 완성하고 커밋합니다.

```
$ git commit -m 'B feature completed'
```

2. 현재 커밋 해시 값 확인해두기

```
$ git log
```

3. 커밋 메시지 수정을 위해 다음과 같이 입력합니다.

```
$ git commit --amend

hint: Waiting for your editor to close the file..[master c01f908] Add no.txt
...
```

4. Vim 편집기가 열리면서 직전 커밋 메시지를 수정할 수 있습니다.

```
B feature completed

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Jan 12 01:25:10 2022 +0900
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   test.txt
```

5. 커밋 메세지를 수정하고 저장하면, 새로운 메세지로 변경되며 커밋 **해시 값 또한 변경됨**

```
$ git log
```

1-2. 커밋 재작성

1. 실수로 bar.txt를 빼고 커밋 해버린 상황까지 만들어 봅시다.

```
$ touch foo.txt bar.txt
$ git add foo.txt
```

```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   foo.txt

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

```
$ git commit -m "foo & bar"

[master 4221af6] foo & bar
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 foo.txt
```

```
$ git status

On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        bar.txt
```

2. 누락된 파일을 staging area로 이동 시킵니다.

```
$ git add bar.txt

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   bar.txt
```

3. `git commit --amend` 를 입력합니다

```
$ git commit --amend
```

4. Vim 편집기가 열립니다. (마찬가지로 커밋 메시지도 수정가능)

```
foo & bar

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Mon Jun 7 22:32:58 2021 +0900
#
# On branch master
# Changes to be committed:
#       new file:   bar.txt
#       new file:   foo.txt
```

5. Vim 편집기를 저장 후 종료하면 직전 커밋이 덮어 씌워집니다.
   (커밋이 새로 추가된 것이 아님) 마찬가지로 커밋 **해시 값 또한 변경됨**

```
$ git commit --amend

[master 7f6c24c] foo & bar
 Date: Mon Jun 7 22:32:58 2021 +0900
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 bar.txt
 create mode 100644 foo.txt
```

6. `git log -p` 를 사용하여 직전 커밋의 변경 내용을 살펴봅니다.

장점

- `--amend` 옵션으로 커밋을 고치는 작업이 주는 장점은 마지막 커밋 작업에서
  뭔가 빠뜨린 것을 넣거나 변경하는 것을 새 커밋으로 분리하지 않고 하나의 커밋에서 처리하는 것
- 예를들면, "앗, 빠진 파일 넣었음", "이전 커밋에서 오타 살짝 고침" 등의 커밋을 만들지 않겠다는 말

> **[중요]**
>
> - 이렇게 `--amend` 옵션으로 커밋을 고치는 작업은, 추가로 작업한 일이 작다고 하더라도
>   이전의 커밋을 완전히 새로 고쳐서 새 커밋으로 변경하는 것을 의미함
> - 이전의 커밋은 일어나지 않은 일이 되는 것이고, 당연히 히스토리에도 남지 않음