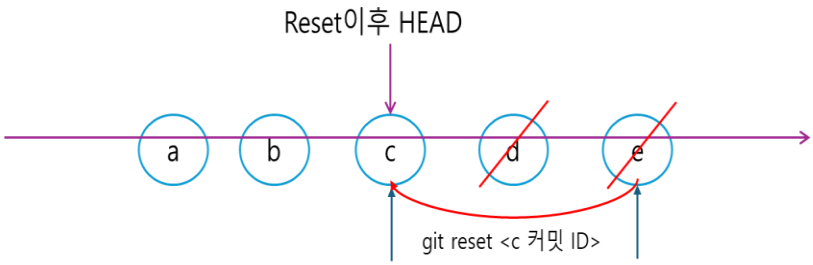
Git04_Reset

Git 온라인 리소스

Git - git-reset Documentation

Reset _ 돌아가려는 커밋으로 리파지토리는 재설정되고, 해당 커밋 이후 이력은 사라짐

Git reset <option><commint-id>



- [상태] 현재 작업 주기상 e 시점까지의 모든 작업이 완료되어 마지막 커밋 e가 생성된 상태
- [문제발생] c 이후의 커밋(d, e)에 문제가 발생하여 해당 변경사항을 취소하고 c 시점으로 돌아가야 하는 상황
- [해결방법] git reset <c 커밋 ID>
- [결과] HEAD 포인터는 c 커밋을 가리키게 되고,
  d와 e 커밋은 더 이상 현재 브랜치의 히스토리에 존재하지 않게 된다.

Fig 1 <<reset>>

위 Fig1 에서 HEAD 포인터는 **c 커밋**을 가리키고 있습니다. HEAD 포인터는 현재 작업 중인 브랜치의 최신 커밋을 가리키는 포인터입니다. git reset 명령어를 통해 HEAD 포인터가 c 커밋으로 이동했음을 알 수 있습니다.

스테이징 영역은 그림에 명시적으로 표시되어 있지는 않지만, HEAD 포인터 바로 다음 영역에 존재한다고 가정할 수 있습니다. 스테이징(index) 영역은 커밋할 변경사항을 미리 준비하는 공간입니다.

`git reset` 명령어의 옵션에 따라 스테이징(index) 영역의 상태가 달라집니다.

- **--soft 옵션:**
  - ✧ HEAD 포인터만 지정된 커밋으로 이동합니다.
  - ✧ 스테이징 영역과 워킹 디렉토리는 변경되지 않고 유지됩니다. 즉, 변경사항이 스테이징된 상태로 남아있습니다.
- **--mixed 옵션 (기본값):**
  - ✧ HEAD 포인터를 지정된 커밋으로 이동합니다.
  - ✧ 스테이징 영역은 지정된 커밋의 상태로 되돌아갑니다. 즉, 스테이징된 변경사항이 취소됩니다.
  - ✧ 워킹 디렉토리는 변경되지 않고 유지됩니다.
- **--hard 옵션:**
  - ✧ HEAD 포인터를 지정된 커밋으로 이동합니다.
  - ✧ 스테이징 영역과 워킹 디렉토리 모두 지정된 커밋의 상태로 완전히 초기화됩니다. 즉, 모든 변경사항이 취소되고 완전히 과거의 시점으로 돌아갑니다(복구불가능).

Git-Reset : hard

- **특정 커밋으로 되돌리기**: `git reset --hard <commit-id>`
  - <commit-id>: 되돌아가려는 특정 커밋의 ID.
- **바로 이전 커밋으로 되돌리기**: `git reset --hard HEAD^`
  - HEAD^: 현재 HEAD 의 바로 이전 커밋

Q1. Git Bash 에서 hard 옵션을 실습해보자.

Q2. 명령을 순서대로 실행하여 파일을 만들고 커밋한다.

git config --global core.autocrlf false 로 줄바꿈 문자를 자동리턴 변환하지 않게 지정 후 아래 내용 실행

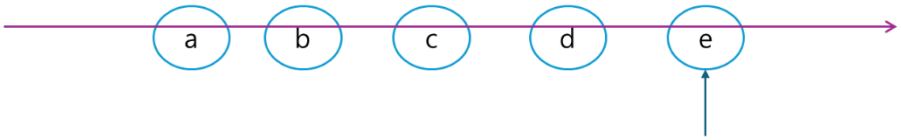
```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git config --global core.autocrlf false
```

```
echo "File A" > a.txt && git add a.txt && git commit -m "Add a.txt"
echo "File B" > b.txt && git add b.txt && git commit -m "Add b.txt"
echo "File C" > c.txt && git add c.txt && git commit -m "Add c.txt"
echo "File D" > d.txt && git add d.txt && git commit -m "Add d.txt"
echo "File E" > e.txt && git add e.txt && git commit -m "Add e.txt"
```

```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ echo "File A" > a.txt && git add a.txt && git commit -m "Add a.txt"
echo "File B" > b.txt && git add b.txt && git commit -m "Add b.txt"
echo "File C" > c.txt && git add c.txt && git commit -m "Add c.txt"
echo "File D" > d.txt && git add d.txt && git commit -m "Add d.txt"
echo "File E" > e.txt && git add e.txt && git commit -m "Add e.txt"
[master (root-commit) e950151] Add a.txt
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
[master 7cf38d1] Add b.txt
 1 file changed, 1 insertion(+)
 create mode 100644 b.txt
[master 3a2b411] Add c.txt
 1 file changed, 1 insertion(+)
 create mode 100644 c.txt
[master 78363a2] Add d.txt
 1 file changed, 1 insertion(+)
 create mode 100644 d.txt
[master 2f3c2b1] Add e.txt
 1 file changed, 1 insertion(+)
 create mode 100644 e.txt
```

Q3. git log --oneline 명령어를 실행하여 현재 커밋 이력을 확인한다.

```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git log --oneline
2f3c2b1 (HEAD -> master) Add e.txt
78363a2 Add d.txt
3a2b411 Add c.txt
7cf38d1 Add b.txt
e950151 Add a.txt
```



2f3c2b1 **(HEAD -> master) Add e.txt**: 이 줄에서 (HEAD -> master)는 HEAD가 2f3c2b1 커밋을 가리키고 있으며, 현재 브랜치가 master임을 나타 낸다

Q4. 워킹 디렉토리와 스테이지 상태 확인을 해보자.

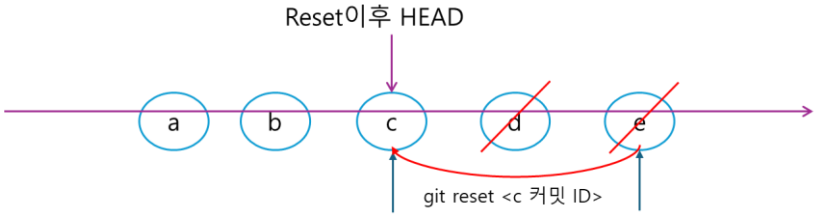
- 각 커밋이 생성된 후에는 스테이지가 비워지고 새로운 커밋이 생성됩니다. 따라서 현재 스테이지는 비어있는 상태입니다.

아직 원격 Git 저장소(예: GitHub, GitLab)에 푸시하지 않았다면, 변경사항은 로컬 저장소에만 존재하는 상태입니다

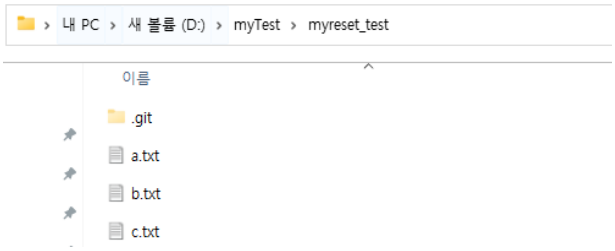
git push 명령어를 실행해야 로컬 저장소의 커밋들이 원격 저장소에 반영됩니다

```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Q5. 아래와 같은 상태로 reset 해보자. 위에서 생성된 파일의 ID를 이용하여 reset한다.




```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reset --hard 3a2b411
HEAD is now at 3a2b411 Add c.txt

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ ls
a.txt  b.txt  c.txt

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git log --oneline
3a2b411 (HEAD -> master) Add c.txt
7cf38d1 Add b.txt
e950151 Add a.txt
```



Q6. git reflog 명령어의 출력 결과는 Git 저장소의 HEAD 변화 이력을 확인하자.

결과는 HEAD가 특정 커밋을 가리키도록 변경된 시점과 변경된 커밋의 ID, 변경 이유를 나타냅니다.

```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reflog
3a2b411 (HEAD -> master) HEAD@{0}: reset: moving to 3a2b411
2f3c2b1 HEAD@{1}: commit: Add e.txt
78363a2 HEAD@{2}: commit: Add d.txt
3a2b411 (HEAD -> master) HEAD@{3}: commit: Add c.txt
7cf38d1 HEAD@{4}: commit: Add b.txt
e950151 HEAD@{5}: commit (initial): Add a.txt
```

Git-Reset : soft

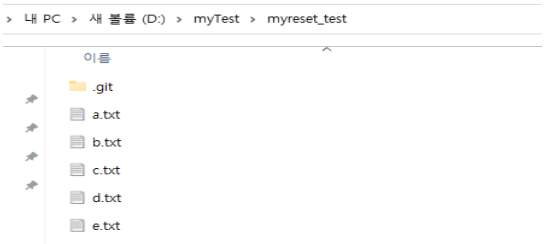
Q1. 동일한 조건으로 myreset_test의 내용을 모두 삭제한 후 soft 옵션을 실행해보자

```
git init
echo "File A" > a.txt && git add a.txt && git commit -m "Add a.txt"
echo "File B" > b.txt && git add b.txt && git commit -m "Add b.txt"
echo "File C" > c.txt && git add c.txt && git commit -m "Add c.txt"
echo "File D" > d.txt && git add d.txt && git commit -m "Add d.txt"
echo "File E" > e.txt && git add e.txt && git commit -m "Add e.txt"
git log --oneline
git status
git reset –-soft  [UserID]
git status
ls
git log --oneline
git reflog
```

```
MINGW64:/d/myTest/myreset_test
265d56a (HEAD -> master) Add e.txt
6bbeb95 Add d.txt
8504343 Add c.txt
db100c4 Add b.txt
1522ba8 Add a.txt
On branch master
nothing to commit, working tree clean

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reset --soft 8504343

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   d.txt
        new file:   e.txt


Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git log --oneline
8504343 (HEAD -> master) Add c.txt
db100c4 Add b.txt
1522ba8 Add a.txt

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reflog
8504343 (HEAD -> master) HEAD@{0}: reset: moving to 8504343
265d56a HEAD@{1}: commit: Add e.txt
6bbeb95 HEAD@{2}: commit: Add d.txt
8504343 (HEAD -> master) HEAD@{3}: commit: Add c.txt
db100c4 HEAD@{4}: commit: Add b.txt
1522ba8 HEAD@{5}: commit (initial): Add a.txt
```

확인

확인



- --soft 옵션은 HEAD 포인터만 이동시키고, 워킹 디렉토리와 스테이징 영역은 그대로 유지합니다.
- 즉, Add d.txt와 Add e.txt 커밋은 히스토리에서 제거되지만, d.txt와 e.txt 파일은 워킹 디렉토리에 남아있고, 스테이징 영역에도 추가된 상태로 남아있습니다.

Git-Reset : mixed

Q1. 동일한 조건으로 myreset_test의 내용을 모두 삭제한 후 mixed 옵션을 실행해보자

```
git init
echo "File A" > a.txt && git add a.txt && git commit -m "Add a.txt"
echo "File B" > b.txt && git add b.txt && git commit -m "Add b.txt"
echo "File C" > c.txt && git add c.txt && git commit -m "Add c.txt"
echo "File D" > d.txt && git add d.txt && git commit -m "Add d.txt"
echo "File E" > e.txt && git add e.txt && git commit -m "Add e.txt"
git log --oneline
git status
git reset –-soft  [UserID]
git status

ls
git log --oneline
git reflog
```

```
Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reset 025077b
                                  c파일 ID reset

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt        워킹 디렉토리 확인

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git status
On branch master     스테이지 영역 제거  /이부분결과가 옵션에 따라 따름
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        d.txt
        e.txt

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

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git log --oneline
025077b (HEAD -> master) Add c.txt
54110f7 Add b.txt
2b0d6cb Add a.txt

Dominica@Dominica MINGW64 /d/myTest/myreset_test (master)
$ git reflog
025077b (HEAD -> master) HEAD@{0}: reset: moving to 025077b
fcf49a8 HEAD@{1}: commit: Add e.txt
f3304d0 HEAD@{2}: commit: Add d.txt
025077b (HEAD -> master) HEAD@{3}: commit: Add c.txt
54110f7 HEAD@{4}: commit: Add b.txt
2b0d6cb HEAD@{5}: commit (initial): Add a.txt
```

- --mixed 옵션은 HEAD 포인터와 스테이징 영역을 이동시키고, 워킹 디렉토리는 유지합니다. (기본 옵션)
- 즉, Add d.txt와 Add e.txt 커밋은 히스토리에서 제거되고, d.txt와 e.txt 파일은 워킹 디렉토리에 남아있지만, 스테이징 영역에서는 제거됩니다.

정리

옵션	HEAD 포인터	스테이징 영역	워킹 디렉토리
--soft	이동	유지	유지
--mixed	이동	이동	유지
--hard	이동	이동	이동

--soft: 과거로 돌아가되, 현재 작업 내용은 그대로 유지하고 싶을 때

--mixed: 과거로 돌아가서 커밋 준비만 다시 하고 싶을 때, 최근 커밋을 취소하고 커밋에 포함할 파일들을 다시 선택하고 싶을 때

--hard: 과거로 완전히 돌아가서 모든 것을 초기화하고 싶을 때