

10장 버전 되돌리기와 취소

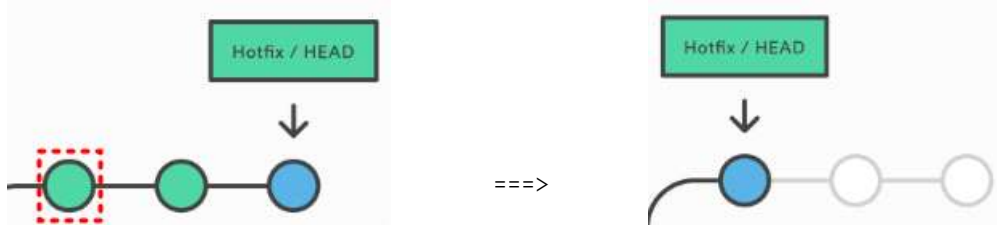
1. 버전 되돌리기

(1) 버전 되돌리기 개요와 준비

특정 커밋으로 되돌리기

깃은 커밋 이력에서 이전 특정 커밋으로 완전히 되돌아가는(roll back) 방법으로 reset 기능을 제공한다. 기능 reset은 시계를 뒤로 맞추는 '타임 머신'과도 같다. 기능 reset으로 특정 커밋으로 완전히 되돌아가면 다음 그림처럼 해당 커밋 이후의 이력은 모두 사라지므로 주의가 필요하다. 기능 reset은 이전 커밋으로 돌아가므로 새로운 커시이 발생하지 않는다. 가능하면 자신의 저장소에 서만 사용하도록 한다. 물론 학습 중에는 아무런 문제가 없으니 걱정하지 말자.

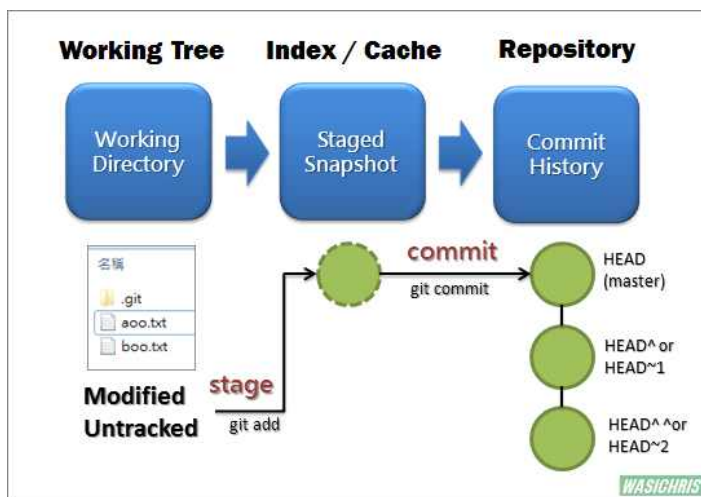
특정 커밋(붉은 상자 부분) 되돌리기 이전



특정 커밋(붉은 상자 부분) 되돌리기 이후

<https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting>

이전 커밋으로 되돌아가는 경우, 깃 저장소는 이전 커밋 내용으로 수정된다. 다만 reset 이전에 있던 작업 폴더와 스테이지 영역을 내용을 어떻게 유지할지가 관건이다. 작업 폴더와 스테이지 영역의 문제로 되돌리기 reset이 다소 복잡하다.



<https://dotblogs.com.tw/wasichris/2016/04/29/225157>

3회의 커밋 이력

다음은 버전 되돌리기 실습을 위해 3회의 커밋 이후 파일 h.py를 수정과 추가, 다시 수정하는 과정을 보이는 깃 명령 이력이다. 다음 깃 명령 이력은 아래부터 위로 순이다. 다음 표에서 붉은 점선 상자의 작업 폴더와 스테이지 영역은 최종 내용만 남으며, 커밋은 붉은 점선 상자의 최종 내용 뿐 아니라 이전의 커밋인 Alphabet과 Numeric 등 모든 이력이 관리된다.

수행 순서와 명령어	작업 디렉토리 (폴더) [h.py]	스테이지(Index) 영역 [h.py]	깃 저장소와 커밋 이력
⑩ \$ echo "print({1, 2})" >> h.py	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	최종 커밋 내용 (h.py) print('AB12') print('ABC') print('123')
⑨ \$ git add h.py	print('123') print('ABC') print('AB12')		
⑧ \$ echo "print([1, 2])" >> h.py	print([1, 2])		
⑦ \$ git commit -am 'Alphanumeric'	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	Alphanumeric print('123') print('ABC') print('AB12') HEAD
⑥ \$ echo "print('AB12')" >> h.py			↓
⑤ \$ git commit -am 'Alphabet'	print('123') print('ABC')	print('123') print('ABC')	Alphabet print('123') print('ABC') HEAD~
④ \$ echo "print('ABC')" >> h.py			↓
③ \$ git commit -m 'Numeric'	print('123')	print('123')	Numeric print('123') HEAD~2
② \$ git add h.py			
① \$ echo "print('123')" > h.py			

위 과정을 처음부터 시작해 보자. 다음은 파일 h.py에 대해 첫 커밋과 'ABC' 출력문을 추가한 이후의 깃 명령 이력이다. 작업 폴더와 스테이지 영역, 최종 커밋 내용을 잘 살펴보자.

수행 순서와 명령어	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소와 커밋 이력
④ \$ echo "print('ABC')" >> h.py	print('123') print('ABC')	print('123')	최종 커밋 내용 (h.py) print('123')
③ \$ git commit -m 'Numeric'			Numeric print('123') HEAD
② \$ git add h.py	print('123')		
① \$ echo "print('123')" > h.py			

항상 깃 저장소를 만들고 버전 관리를 시작하기 전에 다음 명령인 편집기 설정, 원도 자동 줄바꿈과 안전 경고 메시지 설정을 수행하거나 깃 설정 파일에 저장됐는지 확인하고 시작한다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git config --global core.editor 'code --wait'

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git config --global core.autocrlf true

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git config --global core.safecrlf false

```

다음은 위 과정 ❶~❹를 저장소 rback에서 수행한 과정이다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]
$ git init rback
Initialized empty Git repository in C:/[smart Git]/rback/.git/

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]
$ cd rback

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ echo "print('123')" > h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git add h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git commit -m 'Numeric'
[main (root-commit) 50e4a49] Numeric
1 file changed, 1 insertion(+)
create mode 100644 h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline --graph
* 50e4a49 (HEAD -> main) Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ echo "print('ABC')" >> h.py

```

현재 상태는 작업 폴더(Changes not staged for commit:)에 h.py가 수정된 상태임을 알 수 있다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status
On branch main
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:   h.py

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

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status -s
M h.py

```

명령 `git diff`로 스테이지 영역과 비교해서 작업 폴더가 `print('ABC')` 한 줄 더 추가되었음을 알 수 있다. 마찬가지로 명령 `git diff HEAD`로 깃 저장소와 비교해서 작업 폴더가 `print('ABC')` 한 줄 더 추가되었음을 알 수 있다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff
diff --git a/h.py b/h.py
index bec23f4..c40deec 100644
--- a/h.py
+++ b/h.py
@@ -1,2 @@
 print('123')
+print('ABC')

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff HEAD
diff --git a/h.py b/h.py
index bec23f4..c40deec 100644
--- a/h.py
+++ b/h.py
@@ -1,2 @@
 print('123')
+print('ABC')

```

다음은 이후 두 번째 커밋과 'AB12' 출력문을 추가한 이후의 깃 명령 이력이다. 작업 폴더와 스테이지 영역, 최종 커밋 내용을 잘 살펴보자.

명령어	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소와 커밋 이력	
⑥ \$ echo "print('AB12')" >> h.py	print('123') print('ABC') print('AB12')	print('123') print('ABC')	최종커밋 내용 (h.py)	print('ABC') print('123')
⑤ \$ git commit -am 'Alphabet'	print('123') print('ABC')		Alphabet print('ABC') print('123')	HEAD
④ \$ echo "print('ABC')" >> h.py			↓	
③ \$ git commit -m 'Numeric'	print('123')	print('123')	Numeric print('123')	HEAD~
② \$ git add h.py				
① \$ echo "print('123')" > h.py				

다음은 계속해서 위 과정의 두 번째 커밋 ⑤와 ⑥을 수행한 과정이다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git commit -am 'Alphabet'
[main 673af53] Alphabet
1 file changed, 1 insertion(+)

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline --graph
* 673af53 (HEAD -> main) Alphabet
* 50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ echo "print('AB12')" >> h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
print('ABC')
print('AB12')

```

현재 상태는 작업 폴더(Changes not staged for commit:)에 h.py가 수정된 상태임을 알 수 있다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status
On branch main
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:   h.py

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

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status --short
M h.py

```

명령 `git diff`로 스테이지 영역과 비교해서 작업 폴더가 `print('AB12')` 한 줄 더 추가되었음을 알 수 있다. 마찬가지로 명령 `git diff HEAD`로 깃 저장소와 비교해서 작업 폴더가 `print('AB12')` 한 줄 더 추가되었음을 알 수 있다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff
diff --git a/h.py b/h.py
index c40deec..ab8a27a 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,3 @@
 print('123')
 print('ABC')
+print('AB12')

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff HEAD
diff --git a/h.py b/h.py
index c40deec..ab8a27a 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,3 @@
 print('123')
 print('ABC')
+print('AB12')

```

다음은 처음에 살펴본 전체 깃 명령 이력 과정이다.

수행 순서와 명령어	작업 폴더 (h.py)	스테이지 영역 (h.py)	커밋 이력	
⑩ \$ echo "print({1, 2})" >> h.py	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	최종 커밋 내용 (h.py)	print('123') print('ABC') print('AB12')
⑨ \$ git add h.py	print('123') print('ABC') print('AB12')			
⑧ \$ echo "print([1, 2])" >> h.py	print([1, 2])			
⑦ \$ git commit -am 'Alphanumeric'	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD
⑥ \$ echo "print('AB12')" >> h.py			↓	
⑤ \$ git commit -am 'Alphabet'	print('123') print('ABC')	print('123') print('ABC')	Alphabet	HEAD~
④ \$ echo "print('ABC')" >> h.py			↓	
③ \$ git commit -m 'Numeric'	print('123')	print('123')	Numeric	HEAD~2
② \$ git add h.py				
① \$ echo "print('123')" > h.py				

다음은 계속해서 위 과정의 세 번째 커밋 ⑦번부터 ⑩까지 수행한 과정이다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git commit -am 'Alphanumeric'
[main 34cb674] Alphanumeric
1 file changed, 1 insertion(+)

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline --graph
* 34cb674 (HEAD -> main) Alphanumeric
* 673af53 Alphabet
* 50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ echo "print([1, 2])" >> h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git add h.py
```



```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ echo "print({1, 2})" >> h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
print('ABC')
print('AB12')
print([1, 2])
print({1, 2})
```

현재 상태는 작업 폴더(Changes not staged for commit:)에 h.py가 수정된 상태이며, 스테이지 영역(Changes to be committed:)도 수정된 상태(커밋된 내용과 다르다는 것을 의미)임을 알 수 있다.

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

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:   h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status -s
MM h.py
```

현재 상태 파악

현재, 깃의 세 곳 저장소 상태는 다음과 같다. 깃 3 저장소는 작업 폴더와 스테이지 영역, 깃 저장소를 지칭한다.

작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
<pre>print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})</pre>	<pre>print('123') print('ABC') print('AB12') print([1, 2])</pre>	<pre>print('123') print('ABC') print('AB12')</pre>	<div>Alphanumeric</div> <div>print('123') print('ABC') print('AB12')</div> <div>↓</div>	<div>HEAD</div> <div>HEAD~</div> <div>HEAD~2</div>
<div><-----</div> <div>\$ git diff</div>				
	<div><-----</div> <div>\$ git diff --staged</div>		<div>Alphabet</div> <div>print('123') print('ABC')</div> <div>↓</div>	
<div><-----</div> <div>\$ git diff HEAD</div>			<div>Numeric</div> <div>print('123')</div>	

명령 git diff로 스테이지 영역과 비교해서 작업 폴더가 print({1, 2}) 한 줄 더 추가되었음을 알 수 있다. 마찬가지로 명령 git diff HEAD로 깃 저장소와 비교해서 작업 폴더가 두 줄 더 추가되었음을 알 수 있다.

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

```
$ git diff
diff --git a/h.py b/h.py
index 879756d..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -2,3 +2,4 @@ print('123')
print('ABC')
print('AB12')
print([1, 2])
+print({1, 2})
```

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

```
$ git diff HEAD
diff --git a/h.py b/h.py
index ab8a27a..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -1,3 +1,5 @@
 print('123')
print('ABC')
print('AB12')
+print([1, 2])
```

```
+print([1, 2])
```

또한, 명령 `git diff --staged`로 깃 저장소와 비교해서 스테이지 영역은 한 줄 더 추가되었음을 알 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff --staged
diff --git a/h.py b/h.py
index ab8a27a..879756d 100644
--- a/h.py
+++ b/h.py
@@ -1,3 +1,4 @@
 print('123')
 print('ABC')
 print('AB12')
+print([1, 2])
```

명령 checkout으로 시간여행

명령 `checkout`으로 이전 버전의 커밋으로 HEAD를 이동해 그 당시의 파일 `h.py`를 볼 수 있다. 다시 돌아오려면 명령 `checkout [브랜치이름]`으로 브랜치의 최근 커밋으로 다시 이동할 수 있다.

커밋 이력	시간여행 명령 checkout		
Alphanumeric	HEAD		
print('123') print('ABC') print('AB12')			
↓			
Alphabet	HEAD~	\$ git checkout HEAD~	\$ git checkout [commit_ID1]
print('123') print('ABC')		\$ cat h.py \$ git checkout main	\$ cat h.py \$ git checkout main
↓			
Numeric	HEAD~2	\$ git checkout HEAD~2	\$ git checkout [commit_ID2]
print('123')		\$ cat h.py \$ git checkout main	\$ cat h.py \$ git checkout main

명령 `checkout`으로 시간여행을 해 보자. 그런데 `checkout`이 오류가 발생한다. 현재 작업 영역(트리)이 깨끗(clean)하지 않기 때문이다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
print('ABC')
print('AB12')

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git checkout HEAD~
error: Your local changes to the following files would be overwritten by
checkout:
h.py
Please commit your changes or stash them before you switch branches.
Aborting

```

작업 영역(트리)이 깨끗(clean)해지도록 먼저 임시저장 stash한 후 시작하자. 현재, 명령 stash에서 추적되지 않은 파일이 없으므로 옵션 --include-untracked를 빼도 상관없다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash --include-untracked # 옵션 --include-untracked를 빼도 됨
Saved working directory and index state WIP on main: 34cb674 Alphanumeric

```

다음은 명령 checkout으로 위 시간여행을 해 보는 과정이다. 명령 checkout으로 이전 커밋으로 이동하면 분리된 헤드 상태인 'detached HEAD'로 이동된다. 명령 checkout은 HEAD를 이동시키는 것으로 로그 이력이 수정되는 것은 아니다.

```

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git checkout HEAD~
Note: switching to 'HEAD~'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

```

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false

HEAD is now at 673af53 Alphabet

명령 `checkout`로 이전 커밋으로 이동하면 위 결과의 마지막 줄(HEAD is now at 673af53 Alphabet)에 표시되듯이 이동된 커밋이 HEAD가 된다. 명령 `log`의 옵션 `--all`로 브랜치 `main`도 다시 표시할 수 있다. 브랜치 `main` 이전에 표시되는 3개의 로그 이력(6bce263, 5a38603, 216fed4)은 `stash` 임시저장으로 발생한 부분이다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((673af53...))
$ git log --oneline
673af53 (HEAD) Alphabet
50e4a49 Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((673af53...))
$ git log --oneline --all
6bce263 (refs/stash) WIP on main: 34cb674 Alphanumeric
5a38603 index on main: 34cb674 Alphanumeric
216fed4 untracked files on main: 34cb674 Alphanumeric
34cb674 (main) Alphanumeric
673af53 (HEAD) Alphabet
50e4a49 Numeric
```

이동된 커밋에서 파일 `h.py`를 확인하고, 명령 `checkout main`으로 다시 원 상태인 마지막 커밋으로 돌아가자.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((673af53...))
$ cat h.py
```

```
print('123')
print('ABC')
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((673af53...))
```

```
$ git checkout main
```

```
Previous HEAD position was 673af53 Alphabet
```

```
Switched to branch 'main'
```

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

```
$ git log --oneline
```

```
34cb674 (HEAD -> main) Alphanumeric
```

```
673af53 Alphabet
```

```
50e4a49 Numeric
```

이번에는 HEAD~2로 이동해 파일 h.py를 확인하고 돌아오자.

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

```
$ git checkout HEAD~2
```

```
Note: switching to 'HEAD~2'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false

```
HEAD is now at 50e4a49 Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((50e4a49...))
```

```
$ git log --oneline
```

```
50e4a49 (HEAD) Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((50e4a49...))
```

```
$ cat h.py  
print('123')
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback ((50e4a49...))
```

```
$ git checkout main  
Previous HEAD position was 50e4a49 Numeric  
Switched to branch 'main'
```

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

```
$ git log --oneline  
34cb674 (HEAD -> main) Alphanumeric  
673af53 Alphabet  
50e4a49 Numeric
```

다시 처음 상태로 되돌아가기 위해 임시저장 stash 적용

이전에 임시저장 시켜 놓은 stash를 적용해 다음 상태로 되돌아가자.

작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD
			print('123') print('ABC') print('AB12')	
			↓	
			Alphabet	HEAD~
			print('123') print('ABC')	
			↓	
			Numeric	HEAD~2
			print('123')	

임시저장을 확인하고 명령 `stash apply`에서 옵션 `--index`를 사용해 임시저장을 적용한다. 스테이지 영역도 복원하기 위해 옵션 `--index`를 사용한다.

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

```
$ git stash list
```

stash@{0}: WIP on main: 34cb674 Alphanumeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)

\$ git stash apply --index

On branch main

Changes to be committed:

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

modified: h.py

Changes not staged for commit:

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

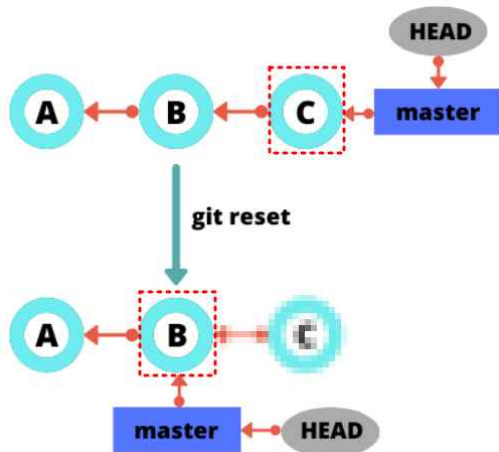
(use "git restore <file>..." to discard changes in working directory)

modified: h.py

(2) 버전 되돌리기 reset

reset 개요

버전 되돌리기 reset은 지정한 이전 특정 커밋(B)으로 버전을 되돌리고 그 이후 커밋(C)은 모두 사라지는 기능이다.



<https://www.studytonight.com/git-guide/git-reset>

버전 되돌리기 reset은 이전 특정 커밋(B)으로 버전을 되돌리므로 깃 저장소(git repository)의 값을 지정한 커밋(B) 내용으로 업데이트한다. 그렇다면 다음 표에서 보듯이 작업 폴더와 스테이지 영역의 내용은 어떻게 처리할까? 이것이 버전 되돌리기 reset에서 3가지 옵션이 있는 이유이다.


작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
<pre>print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})</pre>	<pre>print('123') print('ABC') print('AB12') print([1, 2])</pre>	<pre>print('123') print('ABC') print('AB12')</pre>	Alphanumeric	HEAD
			<pre>print('123') print('ABC') print('AB12')</pre>	
			↓	
			Alphabet	HEAD~
			<pre>print('123') print('ABC')</pre>	
			↓	
			Numeric	HEAD~2
			<pre>print('123')</pre>	

다음은 reset의 3가지 옵션 --hard, --mixed, --soft에 대한 명령과 설명이다.

주요 명령	옵션 --mixed는 옵션이 없는 것과 동일
\$ git reset --hard HEAD~ \$ git reset --hard [commit_ID]	이동되는 커밋의 내용으로 작업 폴더와 스테이지 영역, 깃 저장소를 모두 복사
\$ git reset HEAD~ \$ git reset [commit_ID] \$ git reset --mixed HEAD~ \$ git reset --mixed [commit_ID]	이동되는 커밋의 내용으로 스테이지 영역과 깃 저장소 두 부분에 복사, 작업 폴더는 이전 내용 그대로 남음
\$ git reset --soft HEAD~ \$ git reset --soft [commit_ID]	이동되는 커밋의 내용으로 깃 저장소에만 복사, 스테이지 영역과 작업 폴더는 이전 내용 그대로 남음

reset 옵션 hard

명령 reset의 옵션 --hard는 지정된 커밋 이력 내용으로 깃 저장소는 물론 작업 폴더와 스테이지 영역까지 이전 내용을 무시하고 모두 수정하는 작업을 수행한다. 작업 폴더와 스테이지 영역에 작업 내용이 있다면 모두 사라지므로 주의가 필요하다. 명령 git reset --hard HEAD~로는 다음 표에서 보듯이 지정된 HEAD~(커밋 메시지 Alphabet)의 내용으로 작업 폴더와 스테이지 영역, 깃 저장소가 모두 복사·수정된다. 커밋 메시지 Alphanumeric의 로그 이력과 함께 당시의 작업 폴더와 스테이지 영역, 깃 저장소 내용이 모두 사라진다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print([1, 2])	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD
↓ 	\$ git reset --hard HEAD~			print('123') print('ABC') print('AB12')	
	↓	↓	↓	Alphabet	HEAD~
	↓	↓	↓	print('123') print('ABC')	
reset 이후	print('123') print('ABC')	print('123') print('ABC')	print('123') print('ABC')	Numeric	HEAD~2
				print('123')	

명령 reset 옵션 --hard를 수행해 보자. 작업 폴더의 파일 h.py가 지정된 커밋의 내용으로 수정된 것을 확인할 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

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

```
$ git reset --hard HEAD~  
HEAD is now at 673af53 Alphabet
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ cat h.py  
print('123')  
print('ABC')
```

깃 로그 이력이 커밋 메시지 Alphabet까지 표시되며 git diff는 모두 표시되지 않는다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git log --oneline  
673af53 (HEAD -> main) Alphabet  
50e4a49 Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git diff
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git diff HEAD
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git diff --staged
```

명령 `git reset --hard HEAD~2`로는 다음 표에서 보듯이 지정된 HEAD~2(커밋 메시지 Numeric)의 내용으로 작업 폴더와 스테이지 영역, 깃 저장소가 모두 복사·수정된다. 커밋 메시지 Alphabet 이후의 커밋 내용은 모두 사라진다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	Alphanumeric print('123') print('ABC') print('AB12')	HEAD
↓	\$ git reset --hard HEAD~2			↓	
	↓	↓	↓	Alphabet print('123') print('ABC')	HEAD~
reset 이후				↓	
	print('123')	print('123')	print('123')	Numeric print('123')	HEAD~2

더 알아 봅시다.

되돌리기 명령 reset 이후 다시 이전으로 돌아가기

되돌리기 명령 reset 이후, 다시 이전으로 돌아갈 수 있을까? ORIG_HEAD를 사용하면 간단하다. 깃은 명령에 대한 모든 기록을 남긴다고 해도 과언이 아니다. 깃은 reset 이전의 커밋 ID를 ORIG_HEAD에 저장한다. 그러므로 다음 명령으로 명령 reset 이전으로 바로 돌아갈 수 있다.

□ \$ git reset --hard ORIG_HEAD

다음이 reset 이전으로 바로 돌아가는 과정이다. 다만 --hard이므로 작업 폴더와 스테이지 영역의 내용도 커밋 내용과 같다.

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

```
$ git log --oneline
```

```
673af53 (HEAD -> main) Alphabet
```

```
50e4a49 Numeric
```

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

```
$ git reset --hard ORIG_HEAD
```

```
HEAD is now at 34cb674 Alphanumeric
```

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

```
$ git log --oneline
```

```
34cb674 (HEAD -> main) Alphanumeric
```

```
673af53 Alphabet
```

50e4a49 Numeric

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
print('ABC')
print('AB12')
```

우리가 커밋 메시지 Alphanumeric 커밋한 이후에 작업 폴더와 스테이지 영역을 수정한 내용까지 반영하려면 `git stash apply` 명령으로 임시저장한 내용을 복사해야 한다. 스테이지 영역까지 복사하려면 옵션 `--index`를 사용한다. 물론 이 명령을 수행하려면 이전에 `stash`로 임시저장을 했어야 한다. 우리 이전에 임시저장을 했기 때문에 가능하다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash list
stash@{0}: WIP on main: 34cb674 Alphanumeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash apply --index
On branch main
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified:   h.py

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:   h.py
```

임시저장을 다시 가져온 후 파일을 확인하면 처음과 같다. 명령 `git diff`로 스테이지 영역과 비교해서 작업 폴더가 `print({1, 2})` 한 줄 더 추가되었음을 알 수 있다. 그러므로 스테이지 영역도 처음과 같음을 알 수 있다. 마찬가지로 명령 `git diff HEAD`로 깃 저장소와 비교해서 작업 폴더가 두 줄 더 추가되었음을 알 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
```

```
print('123')
print('ABC')
print('AB12')
print([1, 2])
print({1, 2})
```

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

```
$ git diff
diff --git a/h.py b/h.py
index 879756d..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -2,3 +2,4 @@ print('123')
print('ABC')
print('AB12')
print([1, 2])
+print({1, 2})
```

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

```
$ git diff HEAD
diff --git a/h.py b/h.py
index ab8a27a..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -1,3 +1,5 @@
 print('123')
print('ABC')
print('AB12')
+print([1, 2])
+print({1, 2})
```

이제 되돌리기 명령 `reset` 이전으로 작업 폴더와 스테이지 영역, 깃 저장소가 모두 같아졌다. 정리하면 다음으로 `reset` 이전의 커밋 내용으로 작업 폴더와 스테이지 영역, 깃 저장소를 모두 같게 되돌릴 수 있다.

- `$ git reset --hard ORIG_HEAD`

만일 `reset` 이전 상태가 임시저장 `stash`로 저장되어 있는 경우라면, 다음 두 명령으로 `reset` 이전의 작업 폴더와 스테이지 영역, 깃 저장소를 모두 완벽하게 되돌릴

수 있다.

- \$ git reset --hard ORIG_HEAD
- \$ git stash apply --index

위 [더 알아 봅시다]의 내용으로 저장소 상태를 다시 처음으로 되돌리고 다시 시작한다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --hard ORIG_HEAD
HEAD is now at 34cb674 Alphanumeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash apply --index
On branch main
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified:   h.py

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:   h.py
```

이제 명령 reset 옵션 --hard를 사용해 HEAD~2로 되돌아가 보자. 작업 폴더의 파일 h.py가 지정된 커밋의 내용으로 수정된 것을 확인할 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --hard HEAD~2
HEAD is now at 50e4a49 Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
```

깃 로그 이력이 커밋 메시지 Numeric까지 표시되며 git diff는 모두 표시되지 않는다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
673af53 (HEAD -> main) Alphabet

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff HEAD

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff --staged
```

다음처럼 저장소의 처음 상태로 되돌리고 다시 시작한다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --hard ORIG_HEAD
HEAD is now at 34cb674 Alphanumeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash apply --index
On branch main
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified:   h.py

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:   h.py
```


reset 옵션 mixed

명령 reset의 옵션 --mixed는 지정된 커밋 이력 내용으로 깃 저장소와 스테이지 영역만 이전 내용을 무시하고 수정하는 작업을 수행한다. 스테이지 영역에 작업 내용이 있다면 모두 사라지므로 주의가 필요하다. 명령 git reset --mixed HEAD~로는 다음 표에서 보듯이 지정된 HEAD~(커밋 메시지 Alphanumeric)의 내용으로 스테이지 영역과 깃 저장소가 모두 복사·수정된다. 커밋 메시지 Alphanumeric의 로그 이력과 함께 당시의 스테이지 영역, 깃 저장소 내용이 모두 사라진다. 다만 작업 폴더의 내용은 이전 그대로 남는다. 명령 git reset --mixed는 기본 옵션으로 git reset 과 같다. 즉 옵션 --mixed는 옵션이 없는 것과 같다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print([1, 2])	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	<div>Alphanumeric HEAD</div> <div>print('123') print('ABC') print('AB12')</div>
↓	\$ git reset --mixed HEAD~			↓
	↓	↓	↓	Alphabet HEAD~
				print('123') print('ABC')
reset 이후	print('123') print('ABC') print('AB12') print([1, 2]) print([1, 2])	print('123') print('ABC')	print('123') print('ABC')	<div>Numeric HEAD~2</div> <div>print('123')</div>

명령 reset 옵션 --mixed를 수행해 보자. 결과에서 'Unstaged changes after reset:'은 reset 이후 스테이지 영역이 수정되어 깃 저장소와 같아졌다는 의미이다. 작업 폴더의 파일 h.py은 이전과 변함이 없는 것을 확인할 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --mixed HEAD~ # git reset HEAD~ 동일
Unstaged changes after reset:
M      h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
```

```
print('123')
print('ABC')
print('AB12')
print([1, 2])
print({1, 2})
```

깃 로그 이력이 커밋 메시지 Alphabet까지 표시된다. 명령 `git diff`로 스테이지 영역과 비교해서 작업 폴더가 세 줄 더 추가되었음을 알 수 있다. 마찬가지로 명령 `git diff HEAD`로 깃 저장소와 비교해서 작업 폴더가 세 줄 더 추가되었음을 알 수 있다. 스테이지 영역과 깃 저장소가 같으므로 명령 `git diff --staged`로는 아무것도 표시되지 않는다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
673af53 (HEAD -> main) Alphabet
50e4a49 Numeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff
diff --git a/h.py b/h.py
index c40deec..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,5 @@
 print('123')
 print('ABC')
+print('AB12')
+print([1, 2])
+print({1, 2})
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff HEAD
diff --git a/h.py b/h.py
index c40deec..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,5 @@
 print('123')
 print('ABC')
+print('AB12')
```

```
+print([1, 2])  
+print({1, 2})
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git diff --staged
```


다음으로 저장소 상태를 다시 처음으로 되돌리고 다시 시작한다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git reset --hard ORIG_HEAD  
HEAD is now at 34cb674 Alphanumeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)  
$ git stash apply --index  
On branch main  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
modified:   h.py  
  
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:   h.py
```

reset 옵션 soft

마지막으로 명령 reset의 옵션 --soft는 지정된 커밋 이력 내용으로 깃 저장소만 수정하는 작업을 수행한다. 명령 git reset --soft HEAD~로는 다음 표에서 보듯이 지정된 HEAD~(커밋 메시지 Alphanumeric)의 내용으로 깃 저장소만 복사·수정된다. 커밋 메시지 Alphanumeric의 로그 이력은 사라진다. 그러나 작업 폴더와 스테이지 영역의 내용이 모두 이전 그대로 남는다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD
				print('123') print('ABC') print('AB12')	
	\$ git reset --soft HEAD~			↓	
	↓	↓	↓	Alphabet	HEAD~
	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC')	print('123') print('ABC')	
				Numeric	HEAD~2
reset 이후	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC')	print('123')	

명령 reset 옵션 --soft를 수행해 보자. 작업 폴더의 파일 h.py은 이전과 변함이 없는 것을 확인할 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --soft HEAD~

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ cat h.py
print('123')
print('ABC')
print('AB12')
print([1, 2])
print({1, 2})
```

작업 폴더와 스테이지 영역이 이전과 같으므로 상태는 다음과 같다.

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

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:   h.py

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status -s
MM h.py
```

깃 로그 이력이 커밋 메시지 Alphabet까지 표시된다. 명령 `git diff`로 스테이지 영역과 비교해서 작업 폴더가 한 줄 더 추가되었음을 알 수 있다. 마찬가지로 명령 `git diff HEAD`로 깃 저장소와 비교해서 작업 폴더가 세 줄 더 추가되었음을 알 수 있다. 명령 `git diff --staged`로 깃 저장소와 비교해서 스테이지 영역이 두 줄 더 추가되었음을 알 수 있다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
673af53 (HEAD -> main) Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff
diff --git a/h.py b/h.py
index 879756d..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -2,3 +2,4 @@ print('123')
print('ABC')
print('AB12')
print([1, 2])
+print({1, 2})
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff HEAD
diff --git a/h.py b/h.py
index c40deec..37d4398 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,5 @@
 print('123')
 print('ABC')
+print('AB12')
+print([1, 2])
+print({1, 2})
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git diff --staged
diff --git a/h.py b/h.py
index c40deec..879756d 100644
--- a/h.py
+++ b/h.py
@@ -1,2 +1,4 @@
 print('123')
 print('ABC')
+print('AB12')
+print([1, 2])
```

다음으로 저장소 상태를 다시 처음으로 되돌리자.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --hard ORIG_HEAD
HEAD is now at 34cb674 Alphanumeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git stash apply --index
On branch main
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified:   h.py
```

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: h.py

reset 옵션 정리

되돌리기 reset의 옵션을 정리하면 --hard는 인자인 커밋 깃 저장소의 내용을 작업 폴더와 스테이지 영역, 그리고 깃 저장소 모두에 복사·수정한다. 기본 옵션인 --mixed는 스테이지 영역과 깃 저장소 두 곳에 복사·수정한다. 옵션 --soft는 깃 저장소에만 복사·수정하므로 작업 폴더와 스테이지 영역은 이전 내용이 그대로 남는다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	
reset 이후	↓ \$ git reset --옵션 HEAD~			Alphanumeric HEAD print('123') print('ABC') print('AB12')
	--hard	print('123') print('ABC')	print('123') print('ABC')	↓ Alphabet HEAD~ print('123') print('ABC')
	--mixed 기본 옵션	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC')	↓ Numeric HEAD~2 print('123')
	--soft	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC')	

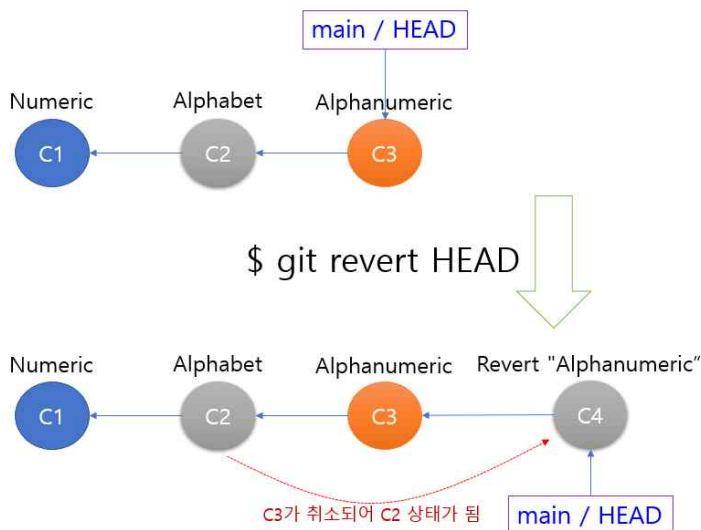
2. 버전 취소 revert

(1) 버전 취소 revert 개요와 준비

이전 커밋 이력 유지하고 특정 커밋 취소 명령 revert

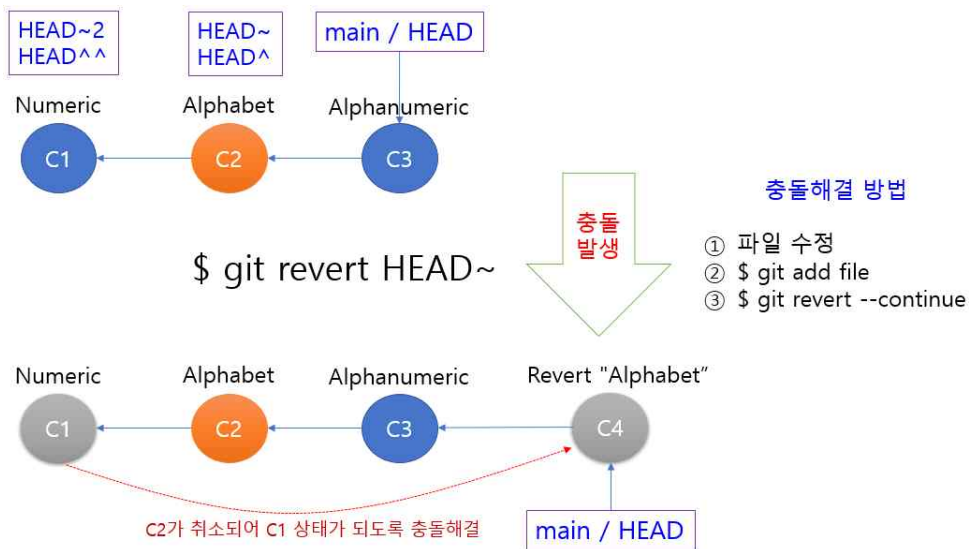
커밋 취소 명령 revert는 워드 프로세서의 undo 기능과 비슷하다. 가장 최근에 실행한 커밋부터 차례로 이전 커밋을 취소하는 데 유용하다. 특정 커밋 취소 명령인 revert는 reset과 다르게 작업 영역이 깨끗(working tree clean)해야 수행할 수 있다.

다음 그림처럼 커밋 취소 명령 revert에서 취소 인자를 HEAD로 사용하면 마지막 커밋인 HEAD를 하나 취소(undo)하고 바로 이전 상태로 돌아가는 새로운 커밋을 마지막에 생성·추가한다. 마치 워드 프로세서에서 [ctrl + z]로 가장 최근에 한 내용을 취소하는 것과 같다. 다음 그림에서 커밋 메시지 Revert "Alphanumeric"이 취소에 대해 새로 추가된 커밋이다. 커밋 취소 명령인 revert는 reset과 비교해서 이전 커밋 이력을 유지하고 이것을 되돌리는 새로운 커밋을 마지막에 생성·추가해 이전 상태로 돌아간다는 특징이 있다.



커밋 취소 명령인 revert에서 취소를 원하는 특정 커밋을 인자로 사용한다. 그런데 명령 revert의 인자가 HEAD~와 같이 HEAD 이전을 취소하면 충돌이 발생할 수 있다. 충돌이 발생하지 않으려면 바로 이전 취소를 여러 번 계속해야 한다. 마치 워드 프로세서에서 취소 [ctrl + z]를 여러 번 하는 것과 같다.

충돌이 발생하면 ① 충돌된 파일 소스를 수정하고, ② 추가 명령 \$ git add file로 수정한 파일을 추가한 후 ③ 커밋 계속 명령 \$ git revert --continue 과정으로 충돌을 해결할 수 있다.



다음은 명령 `revert` 설명이다. 옵션 `--no-edit`을 사용하면 추가되는 커밋 메시지가 자동으로 'Revert "이전 커밋 메시지"'로 지정된다.

주요 명령	
\$ git revert HEAD~ \$ git revert [commit_ID]	이동되는 커밋의 내용으로 작업 폴더와 스테이지 영역, 깃 저장소를 모두 복사하고 커밋 메시지를 수정하도록 편집기가 실행됨
\$ git revert HEAD~ --no-edit \$ git revert [commit_ID] --no-edit	옵션 <code>--no-edit</code> 으로 편집기 실행 없이 커밋 메시지가 자동으로 'Revert "이전 커밋 메시지"'로 지정되어 새로운 커밋 생성

명령 `reset`으로 작업 영역 비우기

현재 상태처럼 작업 영역(working tree)이 깨끗(clean)하지 않으면 다음처럼 오류가 발생해 `revert`가 취소된다. 명령 `revert`는 `reset` 옵션 `--hard`처럼 취소 이전 커밋 내용으로 작업 폴더와 스테이지 영역, 깃 저장소를 모두 수정 · 복사한다.

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric

PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status -s
MM h.py
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git revert HEAD
error: your local changes would be overwritten by revert.
hint: commit your changes or stash them to proceed.
fatal: revert failed
```

작업 영역(working tree)을 깨끗(clean)하게 비우는 방법으로 다음 명령을 활용한다. 명령 `git reset --hard HEAD`로 작업 폴더와 스테이지 영역, 그리고 깃 저장소를 모두 마지막 커밋 내용으로 수정한다.

□ \$ git reset --hard HEAD

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력	
reset 이전	print('123') print('ABC') print('AB12') print([1, 2]) print({1, 2})	print('123') print('ABC') print('AB12') print([1, 2])	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD
				print('123') print('ABC') print('AB12')	
				↓	
	\$ git reset --hard HEAD			Alphabet	HEAD~
	↓	↓	↓	print('123') print('ABC')	
	Working Tree Clean 상태			↓	
reset 이후	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	Numeric	HEAD~2
				print('123')	

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git reset --hard HEAD
HEAD is now at 34cb674 Alphanumeric
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git status
On branch main
nothing to commit, working tree clean
```

```
PC@DESKTOP-482NOAB MINGW64 /c/[smart Git]/rback (main)
$ git log --oneline
34cb674 (HEAD -> main) Alphanumeric
673af53 Alphabet
50e4a49 Numeric
```

간단히 현재 HEAD를 취소해 바로 이전 상태인 HEAD~로 돌아가기

커밋 취소 명령 revert로 바로 이전 상태로 돌아가려면 취소 인자를 HEAD로 사용해야 한다. 다음 표로 명령 git revert HEAD를 설명하고 있다. 명령 git revert 이전 커밋 수 3개에 비해 revert 성공 후 커밋이 하나 추가되어 총 4개가 된다.

구분	작업 폴더 (h.py)	스테이지 영역 (h.py)	깃 저장소 (h.py)	커밋 이력		
revert 이전	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	print('123') print('ABC') print('AB12')	Alphanumeric	HEAD	
				print('123') print('ABC') print('AB12')		
				↓		
				Alphabet	HEAD~	
	\$ git revert HEAD			print('123') print('ABC')		
				↓		
				Numeric	HEAD~2	
				print('123')		
				↓		
				↓		
				↓		
				↓		
				↓		
				↓		
revert 이후	Working Tree Clean 상태			Revert "Alphanumeric"	HEAD	
	print('123') print('ABC')	print('123') print('ABC')	print('123') print('ABC')	print('123') print('ABC')		
	<ul style="list-style-type: none">커밋 Alphanumeric 상태가 취소되어 커밋 Alphabet 상태가 됨이전 커밋이 모두 유지되고 revert에 해당하는 커밋 'Revert "Alphanumeric"'이 마지막으로 생성됨			Alphanumeric	HEAD~	
				print('123') print('ABC') print('AB12')		
				↓		
				Alphabet	HEAD~2	
					print('123') print('ABC')	
					↓	
					Numeric	HEAD~3
					print('123')	
↓						
↓						

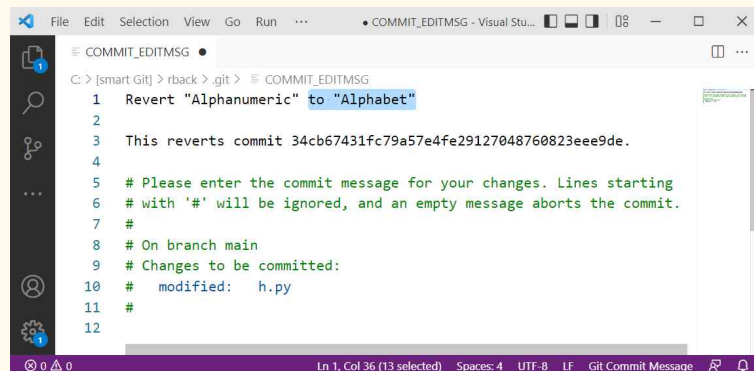
취소 명령 revert가 성공하면 지정된 편집기로 새로운 커밋 메시지를 수정하도록 한다. 자동으로 지정된 커밋 메시지 'Revert "Alphanumeric"'를 'Revert "Alphanumeric" to "Alphabet"'으로 수정한 후, 저장(ctrl + S)하고 닫는(ctrl + F4)다.

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

```
$ git revert HEAD # 다음 vscode에서 편집 후 저장(ctrl + S), 닫기(ctrl + F4)
```

[main 4202413] Revert "Alphanumeric" to "Alphabet"

1 file changed, 1 deletion(-)



최소 revert가 성공하면 새로운 커밋이 입력한 커밋 메시지로 지정되어 생성된다.

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

```
$ git log --oneline
```

```
4202413 (HEAD -> main) Revert "Alphanumeric" to "Alphabet"
```

```
34cb674 Alphanumeric
```

```
673af53 Alphabet
```

```
50e4a49 Numeric
```

최소 revert가 성공해 작업 영역의 파일과 스테이지 영역, 그리고 깃 저장소 내용이 모두 같아진 것을 확인할 수 있다.

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

```
$ cat h.py
```

```
print('123')
```

```
print('ABC')
```

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

```
$ git diff
```

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

```
$ git diff HEAD
```

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

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

(2) 버전 취소 revert의 연속 사용

revert 인자의 순차적 커밋 사용

커밋 취소 명령 revert는 충돌이 발생할 가능성이 많다. 과거의 이전 커밋으로 명령 revert로 이동하려면 여러 번의 revert를 수행해야 한다. 이러한 여러 번 revert 수행을 간편히 제공하는 명령이 인자로 취소가 가능한 순차적 커밋을 여러 번 명시하는 방법이다.

주요 명령	
\$ git revert HEAD HEAD~ [--no-edit] \$ git revert commit_ID1 commit_ID2 [--no-edit]	인자로 취소가 가능한 순차적 커밋을 명시하여 취소를 수행, 만일 명시한 인자의 순차적 취소가 불가능하면 충돌이 발생

revert의 순차적 커밋에서 .. 사용

위에서 학습한 여러 번 커밋을 명시하는 방법보다 간단히 방법을 알아보자.

```
$ git revert HEAD~3..HEAD  
$ git revert HEAD~3..
```

위 명령은 커밋 HEAD에서부터 HEAD~2까지 순차적으로 커밋을 취소하는 명령이다. HEAD는 쓰지 않아도 같다, 위 명령은 다음과 같다.

```
$ git revert HEAD HEAD~ HEAD~2
```

취소 revert에서 --no-edit를 사용해 자동으로 생성되는 커밋 메시지를 활용할 수 있다.

주요 명령	
\$ git revert HEAD~2.. [--no-edit] \$ git revert HEAD~2..HEAD [--no-edit]	인자로 취소가 가능한 순차적 커밋을 HEAD~2..HEAD처럼 명시하여 취소를 수행, 만일 명시한 인자의 순차적 취소가 불가능하면 충돌이 발생
\$ git revert commitID3.. [--no-edit] \$ git revert commitID3..commitID1 [--no-edit]	직접 커밋ID로도 가능

(3) reset과 revert 비교

버전 관리 필요성

항목	reset	revert	비고
기능			
새로운 커밋	없음	있음	
이전 커밋 이력	삭제됨	유지됨	
작업 트리 클린	상관없음	클린해야 함	
충돌 발생	없음	발생할 수 있음	
충돌 해결	-	add 후 commit	

3. vscode로 reset과 revert 수행

(1) reset 수행

버전 관리 필요성

(2) revert 수행

버전 관리 필요성