Git 도구

리비전 조회하기

커밋을 표현하는 몇가지 방법

- **SHA-1 줄여쓰기**

  git log 명령에 --abbrev-commit 이라는 옵션을 추가하면 짧고 중복되지 않는 해시 값을 보여준다

  $ git log --abbrev-commit ...

- **브랜치로 가리키기**

  브랜치를 사용하는 것이 커밋 나타내는 가장 쉬운 방법임..!

  topic1 브랜치의 최근 커밋을 보고싶은 경우 -

  $ git show topic1

- **RefLog 로 가리키기**

  'RefLog'는 자동으로 브랜치와 HEAD 가 지난 몇 달 동안에 가리켰던 커밋을 모두 기록하는 로그.

  git reflog 를 실행하면 Reflog 를 볼 수 있다.

  @{n} 규칙을 사용하면 HEAD 가 n 번 전에 가리켰던 것을 알 수 있다.

  - 어제날짜의 master 브랜치를 보고싶으면 @{yester}

  $ git reflog

  $ git show HEAD@{n}

  $ git show master@{yesterday}

- **계통 관계로 가리키기**

  이름끝에 ^를 붙이면 해당 커밋의 부모를 찾음.

  HEAD^는 HEAD 의 부모를 의미함 = 바로 이전 커밋을 보여줌

  $ git show HEAD^

  ^뒤에 숫자를 붙이면, 예를들어 2 를 붙이면 두번째 부모를 의미한다.

  첫 번째 부모는 Merge 할 때 Checkout 했던 브랜치를 말하고 두 번째 부모는 Merge 한 대상 브랜치를 의미한다.

  계통을 표현하는 다른 방법 '~'도 있다.

  HEAD~와 HEAD^는 똑같이 첫번째 부모를 가리키지만 그 뒤에 숫자를 사용하면 *달라진다.*

  HEAD~2 는 조부모(첫 번째 부모의 첫 번째 부모)를 가리킨다.

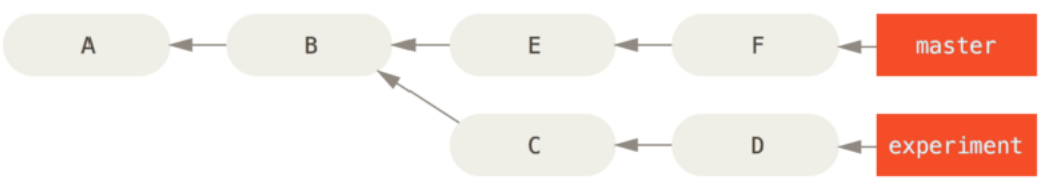
  HEAD~3 은 HEAD^^^와 같은 표현인다. 부모의 부모의 부모 즉, 증조 부모 정도 된다.

- **범위로 커밋 가리키기**

  범위를 사용해서 조회하면 브랜치를 관리할 때 유용하다.

  - Double Dot

    범위를 표현하는 문법으로 Double Dot 을 많이쓴다. Double Dot 은 어떤 커밋들이 한쪽에는 관련되었고 다른 쪽에는 관련되지 않았는지 Git 에게 물어보는 것이다.



  - 세 개 이상의 Refs

refA 나 refB 에는 있지만 refC 에는 없는 커밋을 보려면 아래 중 한 명령을 사용한다.

```
$ git log refA refB ^refC
$ git log refA refB --not refC
```

- Triple Dot

master 와 experiment 의 공통부분은 빼고 다른 커밋만 보고 싶으면 아래와 같이 하면 된다.

```
$ git log master...experiment
```

대화형 명령

Git 에서 대화형 스크립트 제공 -> 명령을 좀 더 쉽게 사용할 수 있다.

git add 명령에 -i 나 --interactive 옵션을 주고 실행하면 Git 은 대화형 모드로 들어간다.

- Staging Area 에 파일 추가하고 추가 취소하기
- 파일의 일부분만 Staging Area 에 추가하기

Stashing 과 cleaning

커밋하지 않고 나중에 다시 돌아와서 작업을 하고 싶은 경우, ***git stash*** 명령으로 해결할 수 있다.

Stash 명령을 사용하면 워킹 디렉토리에서 수정한 파일들만 저장한다. (Stash 는 Modified 이면서 Tracked 상태인 파일과 Staging Area 에 있는 파일들을 보관해두는 장소다.)

아직 끝내지 않은 수정사항을 스택에 잠시 저장했다가 나중에 다시 적용할 수 있다.

- Stash 를 만드는 새로운 방법

  --keep-index 옵션을 이용하면 이미 Staging Area 에 들어 있는 파일을 Stash 하지 않는다.

많은 파일을 변경했지만 몇몇 파일만 커밋하고 나머지 파일은 나중에 처리하고 싶을 때 유용하다.

$ git stash --keep-index

- Stash 를 적용한 브랜치 만들기

  git stash branch 명령을 실행하면 Stash 할 당시의 커밋을 확인한 후 새로운 브랜치를 만들고 여기에 적용한다.

- 워킹 디렉토리 청소하기

  작업하고 있던 파일을 Stash 하지 않고 단순히 그 파일들을 치워버리고 싶을 때가 있다. git clean 명령이 그 일을 한다.

  추적 중이지 않은 모든 정보를 워킹 디렉토리에서 지우고 싶다면 git clean -f -d 명령을 사용하면 된다.

  이 명령을 실행했을 때 어떤 일이 일어날지 미리 보고 싶다면 -n 옵션을 사용하면 된다.

  clean 명령을 대화형으로 실행하려면 -i 옵션을 붙이면 된다.

내 작업에서 서명하기

GPG -> 저장소에 아무나 접근하지 못하게 하고 *진짜로 확인된 사람에게서만* 커밋을 받으려면 GPG 를 이용

- 개인 키 생성

$ gpg --gen-key

- 태그 서명하기
- 태그 확인하기

  git tag -v [tag-name] 명령을 이용해 태그에 서명한 사람이 정말 그 사람이 맞는지 확인한다.

- 커밋에 서명하기

  커밋에 서명하고 싶으면 git commit 명령에 -S 옵션만 붙여주면 된다.

  서명을 확인하려면 git log 명령에 --show-signature 옵션을 붙여주면 된다.

  git log 로 출력한 로그에서 커밋에 대한 서명 정보를 알려면 %G? 포맷을 이용한다.

  Merge 할 때 --verify-signatures 옵션을 붙이면 Merge 할 커밋 중 서명하지 않았거나 신뢰할 수 없는 사람이 서명한 커밋이 있으면 Merge 되지 않는다.

검색

Git 은 데이터베이스에 저장된 코드나 커밋에서 원하는 부분을 빠르고 쉽게 검색하는 도구가 여러가지 있다.

- **Git Grep**

  Git 의 grep 명령을 이용하면 커밋 트리의 내용이나 워킹 디렉토리의 내용을 문자열이나 정규표현식을 이용해 쉽게 찾을 수 있다.

  -n 옵션을 추가하면 찾을 문자열이 위치한 라인 번호도 같이 출력한다.

  어떤 파일에서 몇 개나 찾았는지만 알고 싶다면 --count 옵션을 이용한다.

  매칭되는 라인이 있는 함수나 메서드를 찾고 싶다면 -p 옵션을 준다.

  --and 옵션을 이용해서 여러 단어가 한 라인에 동시에 나타나는 줄 찾기 같은 복잡한 조합으로 검색할 수 있다.

  --break 와 --heading 옵션을 붙여 더 읽기 쉬운 형태로 잘라서 출력할 수도 있다.

  - git grep 명령이 grep 이나 ack 같은 일반적인 검색 도구보다 몇 가지 좋은 점
    - 매우 빠르다.
    - 워킹 디렉토리만이 아니라 Git 히스토리 내의 어떠한 정보라도 찾아낼 수 있다.
- **Git 로그 검색**

어떤 변수가 어디에 있는지를 찾아보는 게 아니라, 히스토리에서 ***언제*** 추가되거나 변경됐는지 찾아볼 수도 있다.

git log 명령을 이용하면 다른 내용도 검색하여 어떤 커밋에서 찾고자 하는 내용을 추가했는지 찾을 수 있다.

- **라인 로그 검색**

  라인 히스토리 검색

  git log 를 쓸 때 -L 옵션을 붙이면 어떤 함수나 한 라인의 히스토리를 볼 수 있다.

  이 명령을 실행하면 함수의 시작과 끝을 인식해서 함수에서 일어난 모든 히스토리를 함수가 처음 만들어진 때부터 Patch 를 나열하여 보여준다.

히스토리 단장하기

사람들과 코드를 공유하기 전에 커밋 히스토리를 **수정**

- **마지막 커밋을 수정하기**
    1. 커밋 메시지를 수정하는 것.
- $ git commit --amend

  이 명령은 자동으로 텍스트 편집기를 실행시켜서 마지막 커밋 메시지를 열어준다. 여기에 메시지를 바꾸고 편집기를 닫으면 편집기는 바뀐 메시지로 마지막 커밋을 수정한다.

    2. 파일 목록을 수정하는 것.
- **커밋 메시지를 여러 개 수정하기**

  히스토리 수정을 위해서는 rebase 명령을 이용하여 수정할 수 있다. 현재 작업하는 브랜치에서 각 커밋을 하나하나 수정하는 것이 아니라 어느 시점부터 HEAD 까지의 커밋을 한 번에 Rebase 한다.

  git rebase 명령에 -i 옵션을 추가하면 대화형 모드로 Rebase 할 수 있다.

- **커밋 순서 바꾸기**

대화형 Rebase 도구로 커밋 전체를 삭제하거나 **순서를 조정**할 수 있다.

- **커밋 합치기**

  Rebase 명령을 이용해서 여러 개의 커밋을 하나로 만들어 버릴 수 있다.

  'pick'이나 'edit'말고 squash'를 입력하면 Git 은 해당 커밋과 바로 이전 커밋을 합칠 것이고 커밋 메시지도 Merge 한다.

- **커밋 분리하기**

  분리한다는 것은 기존의 커밋을 해제하고 Stage 를 여러개로 분리하고 나서 그것을 원하는 횟수만큼 다시 커밋하는 것이다.

  예를 들어

  edit 310154e updated README formatting and added blame

  이 커밋의 'updated README formatting and added blame'을 'updated README formatting'과 'added blame'으로 분리하는 것이다.

- **filter-branch**

  수정해야 하는 커밋이 너무 **많아서** Rebase 스크립트로 수정하기 어려울 것 같으면 **filter-branch** 라는 명령으로 수정할 수 있는데 Rebase 가 삽이라면 이 명령은 포크레인이라고 할 수 있다.

  filter-branch 도 역시 수정하려는 커밋이 이미 공개돼서 다른 사람과 함께 공유하는 중이라면 사용하지 말아야 한다.

  - *모든 커밋에서 파일을 제거하기*

    filter-branch 는 히스토리 전체에서 필요한 것만 골라내는 데 사용하는 도구이다. --tree-filter 라는 옵션을 사용하면 히스토리에서 원하는 파일을 아예 제거할 수 있다.

    --tree-filter 옵션은 프로젝트를 확인한 후에 각 커밋에 명시한 명령을 실행시키고 그 결과를 다시 커밋한다.

    filter-branch 명령에 --all 옵션을 추가하면 모든 브랜치에 적용할 수 있다.

- *하위 디렉토리를 루트 디렉토리로 만들기*

  다른 VCS 에서 코드를 임포트하면 그 VCS 만을 위한 디렉토리가 있을 수 있다. SVN 에서 코드를 임포트하면 trunk, tags, branch 디렉토리가 포함된다. 모든 커밋에 대해 trunk 디렉토리를 프로젝트 루트 디렉토리로 만들 때도 filterbranch 명령이 유용하다.

- *모든 커밋의 이메일 주소를 수정하기*

filter-branch 명령의 --commit-filter 옵션을 사용하여 해당 커밋만 골라서 이메일 주소를 수정할 수 있다.

Reset 명확히 알고 가기!

- 세 개의 트리

  (하는 역할)

  HEAD: 마지막 커밋 스냅샷, 다음 커멧의 부모 커밋

  Index: 다음에 커밋할 스냅샷

  워킹 디렉토리: 샌드박스

- **HEAD**

  HEAD 는 현재 브랜치를 가리키는 포인터이고 브랜치는 브랜치에 담긴 커밋 중 가장 마지막 커밋을 가리킨다.

  HEAD 는 마지막 커밋의 스냅샷이다.

- **Index**

  Index 는 **바로 다음에 커밋할** 것들이다.

  먼저 Index 는 워킹 디렉토리에서 마지막으로 Checkout 한 브랜치의 파일 목록과 파일 내용으로 채워진다. 이후 파일 변경작업을 하고 변경한 내용으로 Index 를

업데이트 할 수 있다. 이렇게 업데이트 하고 git commit 명령을 실행하면 Index 는 새 커밋으로 변환된다.

다음 명령은 현재 Index 가 어떤 상태인지를 확인할 수 있다.

$ git ls-files -s

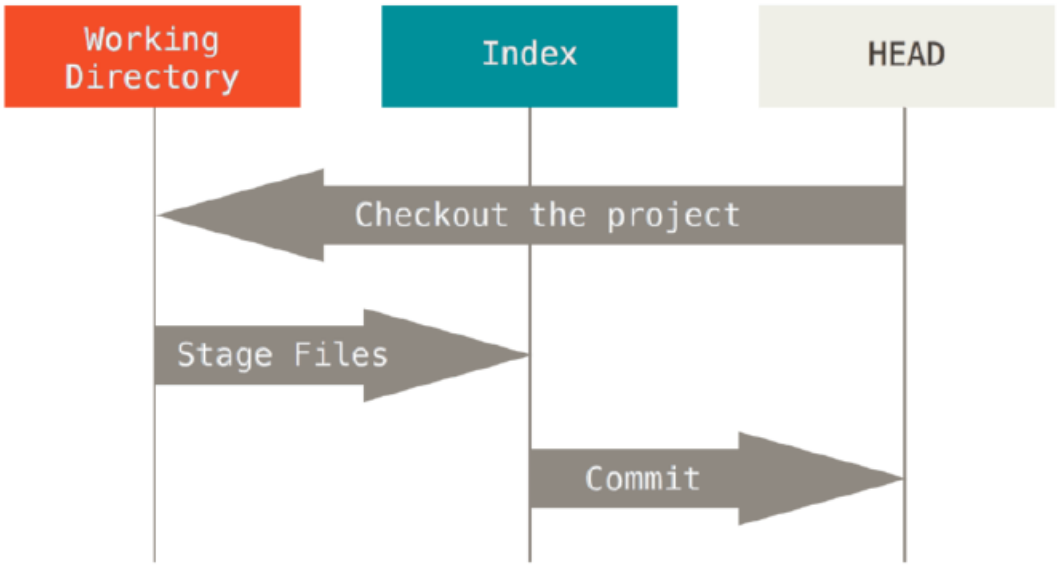
- **워킹 디렉토리**

워킹 디렉토리는 **실제 파일**로 존재한다.

커밋하기 전에는 Index(Staging Area)에 올려놓고 얼마든지 변경할 수 있다.

- 워크 플로

이 과정을 시각화해보도록 하겠다.




HEAD
master
?
Git Repository


HEAD
Index
Working Directory
file.txt v1
file.txt v1

git add

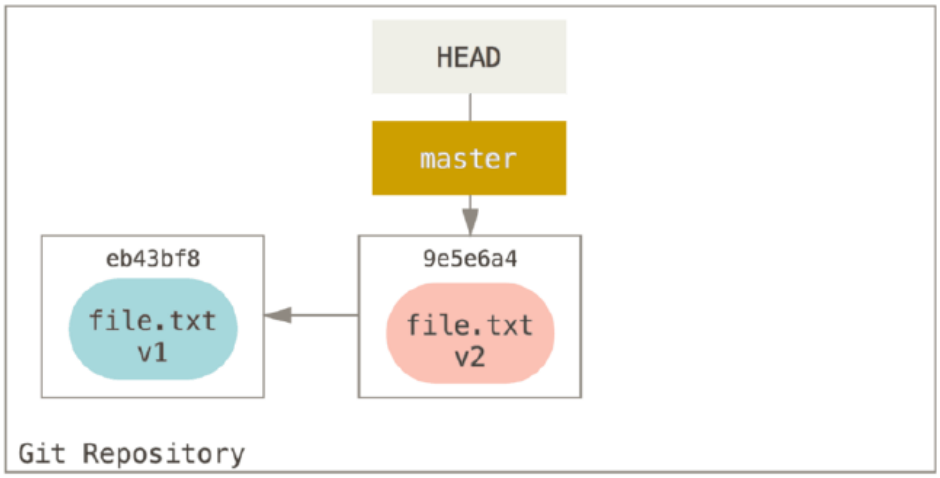

HEAD
master
?
Git Repository


HEAD
Index
Working Directory
file.txt v1

git commit

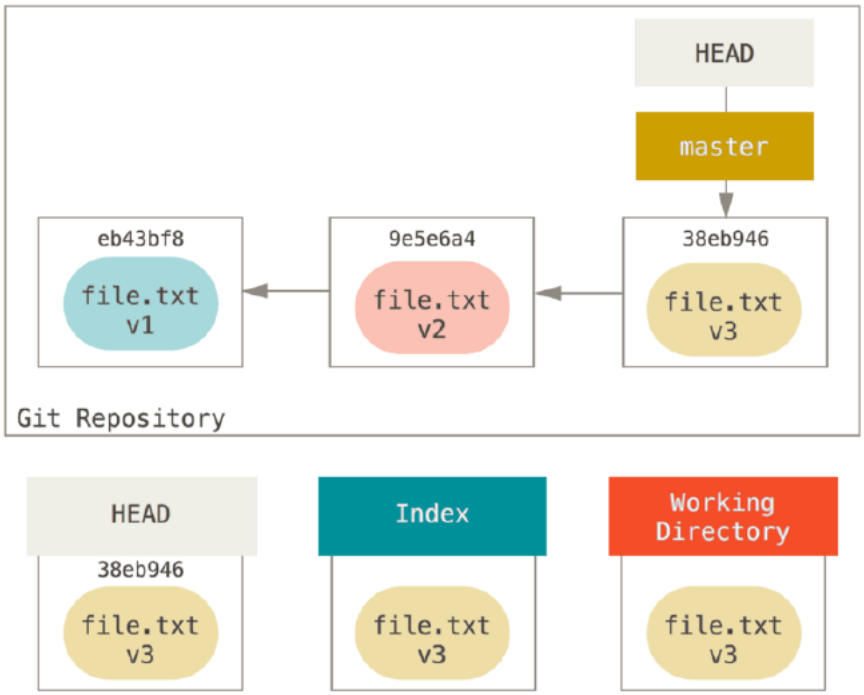
edit file

git add



git commit

file.txt 파일 하나를 수정하고 커밋한다. 이것을 세 번 반복한다. 그러면 히스토리는 다음과 같이 된다.

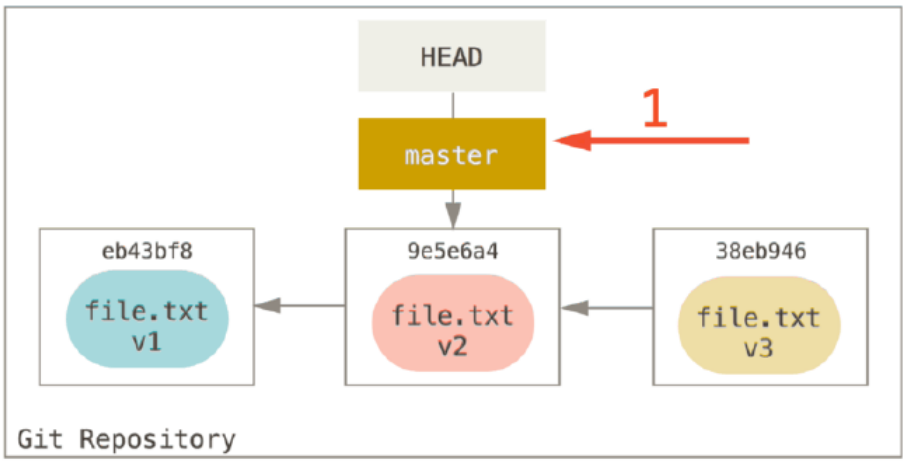


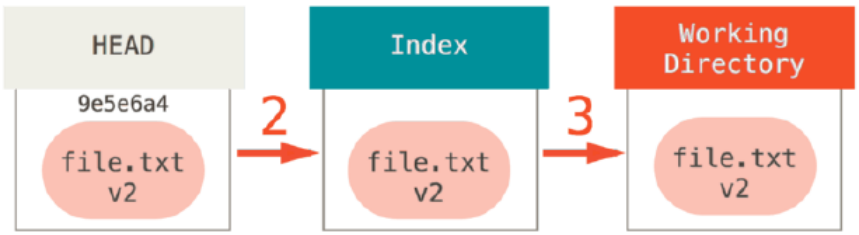
reset 명령은 이 세 트리를 간단하고 예측 가능한 방법으로 조작한다. 트리를 조작하는 동작은 세 단계로 이루어진다.

1 단계: HEAD 이동 (--soft 옵션이 붙으면 여기까지)

2 단계: Index 업데이트 (--hard 옵션이 붙지 않았으면 여기까지)

3 단계: 워킹 디렉토리 업데이트


HEAD
master
1
eb43bf8
file.txt
v1
9e5e6a4
file.txt
v2
38eb946
file.txt
v3
Git Repository


HEAD
9e5e6a4
file.txt
v2
2
Index
file.txt
v2
3
Working
Directory
file.txt
v2