커밋 히스토리 조회하기

새로 저장소를 만들어서 몇 번 커밋을 했을 수도 있고, 커밋 히스토리가 있는 저장소를 Clone했을 수도 있다. 어쨌든 가끔 저장소의 히스토리를 보고 싶을 때가 있다. Git에는 히스토리를 조회하는 명령어인 git log가 있다.

이 예제에서는 simplegit이라는 매우 단순한 프로젝트를 사용한다. Simplegit은 Git을 설명하는데 자주 사용하는 예제다. 아래와 같이 이 프로젝트를 Clone한다.

git clone git://github.com/schacon/simplegit-progit.git

이 프로젝트 디렉토리에서 git log 명령을 실행하면 아래와 같이 출력된다.

$ git log

commit ca82a6dff817ec66f44342007202690a93763949

Author: Scott Chacon <schacon@gee-mail.com>

Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 10:31:28 2008 -0700

first commit

특별한 아규먼트 없이 git log 명령을 실행하면 저장소의 커밋 히스토리를 시간순으로 보여준다. 즉, 가장 최근의 커밋이 가장 먼저 나온다. 그리고 이어서 각 커밋의 SHA-1체크섬, 저자이름, 저자이메일, 커밋한 날짜, 커밋 메시지를 보여준다.

원하는 히스토리를 검색할 수 있도록 git log 명령은 매우 다양한 옵션을 지원한다. 여기에서는 자주 사용하는 옵션을 설명한다.

-p가 가장 유용한 옵션 중 하나다. –p는 각 커밋의 diff 결과를 보여준다. 게다가 -2는 최근 두 개의 결과만 보여주는 옵션이다.

$ git log -p -2

commit ca82a6dff817ec66f44342007202690a93763949

Author: Scott Chacon <schacon@gee-mail.com>

Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

diff --git a/Rakefile b/Rakefile

index a874b73..8f94139 100644

--- a/Rakefile

+++ b/Rakefile

@@ -5,7 +5,5 @@ require 'rake/gempackagetask'

spec = Gem::Specification.new do |s|

s.name = "simplegit"

- s.version = "0.1.0"

+ s.version = "0.1.1"

s.author = "Scott Chacon"

s.email = "schacon@gee-mail.com"

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test code

diff --git a/lib/simplegit.rb b/lib/simplegit.rb

index a0a60ae..47c6340 100644

--- a/lib/simplegit.rb

+++ b/lib/simplegit.rb

@@ -18,8 +18,3 @@ class SimpleGit

end

end

-

-if $0 == \_\_FILE\_\_

- git = SimpleGit.new

- puts git.show

-end

\ No newline at end of file

이 옵션은 직접 diff를 실행한 것과 같은 결과를 출력하기 때문에 동료가 무엇을 커밋했는지 리뷰하고 빨리 조회하는데 유용하다.

가끔은 diff결과를 줄 단위로 보기보다는 단어 단위로 보는 것이 좋을 떄도 있다. git log –p와 같은 명령에 –word-diff 옵션을 사용하면 줄 단위 대신 단어 단위로 변경사항을 보여준다. 단어 단위로 다른 부분을 확인하는 것은 소스코드에 별로 유용하지 않다. 책이나 에세이 같이 문장이 긴 글을 쓸 때는 단어 단위로 보는 것이 편하다. –word-diff 옵션은 다음과 같이 사용한다.

또 git log 명령에는 히스토리의 통계를 보여주는 옵션도 있다. --stat옵션으로 각 커밋의 통계 정보를 조회할 수 있다.

$ git log --stat

commit ca82a6dff817ec66f44342007202690a93763949

Author: Scott Chacon <schacon@gee-mail.com>

Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

Rakefile | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test code

lib/simplegit.rb | 5 -----

1 file changed, 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 10:31:28 2008 -0700

first commit

README | 6 ++++++

Rakefile | 23 +++++++++++++++++++++++

lib/simplegit.rb | 25 +++++++++++++++++++++++++

3 files changed, 54 insertions(+)

이 결과에서 –stat 옵션은 어떤 파일이 수정됐는지, 얼마나 많은 파일이 변경됐는지, 또 얼마나 많은 줄을 추가하거나 삭제했는지 보여준다. 요약 정보는 가장 뒤쪽에 보여준다.

다른 또 유용한 옵션은 –pretty 옵션이다. 이 옵션을 통해 log의 내용을 보여줄 떄 기본 형식 이외에 여러 가지 중에 하나를 선택할 수 있다. oneline 옵션은 각 커밋을 한 줄로 보여준다. 이 옵션은 많은 커밋을 한 번에 조회할 떄 유용하다. 추가로 short, full, fuller 옵션도 있는데 이것은 정보를 조금씩 가감해서 보여준다.

$ git log --pretty=oneline

ca82a6dff817ec66f44342007202690a93763949 changed the version number

085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code

a11bef06a3f659402fe7563abf99ad00de2209e6 first commit

가장 재밌는 옵션은 format옵션이다. 나만의 포맷으로 결과를 출력하고 싶을 때 사용한다. 특히 결과를 다른 프로그램으로 파싱하고자 할 때 유용하다. 이 옵션을 사용하면 포맷을 정확하게 일치시킬 수 있기 때문에 Git을 새 버전으로 바꿔도 결과 포맷이 바뀌지 않는다.

$ git log --pretty=format:"%h - %an, %ar : %s"

ca82a6d - Scott Chacon, 11 months ago : changed the version number

085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code

a11bef0 - Scott Chacon, 11 months ago : first commit

표 2-1 형식에서 사용하는 유용한 옵션들.

| **Option** | **Description of Output** |
| --- | --- |
| %H | Commit hash |
| %h | Abbreviated commit hash |
| %T | Tree hash |
| %t | Abbreviated tree hash |
| %P | Parent hashes |
| %p | Abbreviated parent hashes |
| %an | Author name |
| %ae | Author e-mail |
| %ad | Author date (format respects the --date= option) |
| %ar | Author date, relative |
| %cn | Committer name |
| %ce | Committer email |
| %cd | Committer date |
| %cr | Committer date, relative |
| %s | Subject |

저자와 커미터를 구분하는 것이 조금 이상해 보일 수 있다. 저자는 원래 작업을 수행한 원작자이고 커미터는 마지막으로 이 작업을 적용한 사람이다. 만약 당신이 어떤 프로젝트에 패치를 보냈고 그 프로젝트의 담당자가 패치를 적용했다면 두 명의 정보를 모두 알 필요가 있다. 그래서 이 경우 당신이 저자고 그 담당자가 커미터다. 5장에서 이 주제에 대해 자세히 다룰 것이다.

Oneline과 format 옵션은 –graph 옵션과 함께 사용할 때 더 빛난다. 이 명령은 브랜치와 머지 히스토리를 보여주는 아스키 그래프를 출력한다. 이 명령을 Grit 프로젝트 저장소에서 사용해보면 아래와 같다.

$ git log --pretty=format:"%h %s" --graph

\* 2d3acf9 ignore errors from SIGCHLD on trap

\* 5e3ee11 Merge branch 'master' of git://github.com/dustin/grit

|\

| \* 420eac9 Added a method for getting the current branch.

\* | 30e367c timeout code and tests

\* | 5a09431 add timeout protection to grit

\* | e1193f8 support for heads with slashes in them

|/

\* d6016bc require time for xmlschema

\* 11d191e Merge branch 'defunkt' into local

git log 명령의 기본적인 옵션과 출력물의 형식에 관련된 옵션을 살펴보았다. git log 명령은 앞서 살펴본 것보다 더 많은 옵션을 지원한다. 표 2-2는 지금 설명한 것과 함께 유용하게 사용할 수 있는 옵션이다. 각 옵션으로 어떻게 log 명령을 제어할 수 있는지 보여준다.

| **옵션** | **설명** |
| --- | --- |
| -p | 각 커밋에 적용된 패치를 보여준다. |
| --word-diff | diff 결과를 단어 단위로 보여준다. |
| --stat | 각 커밋에서 수정된 파일의 통계정보를 보여준다. |
| --shortstat | --stat 명령의 결과 중에서 수정한 파일, 추가된 줄, 삭제된 줄만 보여준다. |
| --name-only | 커밋 정보중에서 수정된 파일의 목록만 보여준다. |
| --name-status | 수정된 파일의 목록을 보여줄 뿐만 아니라 파일을 추가한 것인지, 수정한 것인지, 삭제한 것인지도 보여준다. |
| --abbrev-commit | 40자 짜리 SHA-1 체크섬을 전부 보여주는 것이 아니라 처음 몇 자만 보여준다. |
| --relative-date | 정확한 시간을 보여주는 것이 아니라 2 주전처럼 상대적인 형식으로 보여준다. |
| --graph | 브랜치와 머지 히스토리 정보까지 아스키 그래프로 보여준다. |
| --pretty | 지정한 형식으로 보여준다. 이 옵션에는 oneline, short, full, fuller, format이 있다. format은 원하는 형식으로 출력하고자 할 때 사용한다. |
| --oneline | --pretty=oneline --abbrev-commit 옵션을 함께 사용한 것과 동일하다. |

조회 제한 조건

출력 형식과 관련된 옵션을 살펴봤지만 git log 명령은 조회 범위를 제한하는 옵션들도 있다. 히스토리 전부가 아니라 부분만 조회한다. 이미 최근 두 개만 조회하는 -2 옵션은 살펴봤다. 실제 사용법은 -<h>이고 n은 최근 n 개의 커밋을 의미한다. 사실 이 옵션은 잘 쓰이지 않는다. Git은 기본적으로 출력을 pager 류의 프로그램을 거쳐서 내보내므로 한 번에 한 페이지씩 보여준다.

반면 –since나 --until같은 시간을 기준으로 조회하는 옵션은 매우 유용하다. 지난 2주동안 만들어진 커밋들만 조회하는 명령어는 아래와 같다.

$ git log --since=2.weeks

이 옵션은 다양한 형식을 지원한다. 2008-01-15 같이 정확한 날짜도 사용할 수 있고 2 years 1 day 3 minutes ago 같이 상대적인 기간을 사용할 수도 있다.

또 다른 기준도 있다. –author옵션으로 저자를 지정하여 검색할 수도 있고 –grep 옵션으로 커밋 메시지에서 키워드를 검색할 수도 있다. (author와 grep 옵션을 함께 사용하면 모두 만족하는 커밋을 찾는다).

마지막으로 파일 경로로 검색하는 옵션이 있는데 이것도 정말 유용하다. 디렉토리나 파일 이름을 사용하여 그 파일이 변경된 log의 결과를 검색할 수 있다. 이 옵션은 – 와 함께 경로 이름을 사용하는데 명령어 끝 부분에 쓴다. (역주, git log—path1, path2)

표 2-3은 조회 범위를 제한하는 옵션들이다.

| **옵션** | **설명** |
| --- | --- |
| -(n) | 최근 n 개의 커밋만 조회한다. |
| --since, --after | 명시한 날짜 이후의 커밋만 검색한다. |
| --until, --before | 명시한 날짜 이전의 커밋만 조회한다. |
| --author | 입력한 저자의 커밋만 보여준다. |
| --committer | 입력한 커미터의 커밋만 보여준다. |

아래 예제는 2008년 10월에 Junio Hamano가 커밋한 히스토리를 조회하는 것 이다. 그 중에서 테스트 파일을 수정한 커밋 중에서 머지 커밋이 아닌 것들만 조회한다.

$ git log --pretty="%h - %s" --author=gitster --since="2008-10-01" \

--before="2008-11-01" --no-merges -- t/

5610e3b - Fix testcase failure when extended attribute

acd3b9e - Enhance hold\_lock\_file\_for\_{update,append}()

f563754 - demonstrate breakage of detached checkout wi

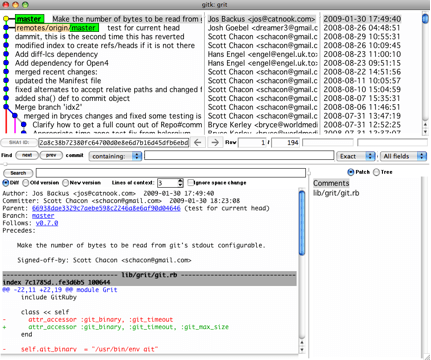
d1a43f2 - reset --hard/read-tree --reset -u: remove un

51a94af - Fix "checkout --track -b newbranch" on detac

b0ad11e - pull: allow "git pull origin $something:$cur

GUI 도구로 히스토리를 시각화하기

GUI 도구로 커밋 히스토리를 시각화하고 싶다면 gitk를 사용할 수 있다. gitk 는 Tcl/TK 프로그램이고 git log 명령을 시각화해주는 도구다. gitk는 git log 명령이 지원하는 필터링 옵션을 거의 모두 지원한다. 프로젝트 디렉토리에서 gitk를 실행하면 다음과 같이 보일 것이.



위쪽 반을 차지하는 윈도에서는 히스토리를 그래프로 예쁘게 보여준다. 아래쪽 반을 차지하는 윈도는 diff 결과를 보여주는데 위쪽 윈도에서 선택한 커밋에 대한 diff 결과를 보여준다.