

[About](#)
[Documentation](#)
[Reference](#)
[Book](#)
[Videos](#)
[External Links](#)
[Downloads](#)
[Community](#)

 This book is available in [English](#).

Full translation available in

[Български](#)
[Español](#)
[Français](#)
[Ελληνικά](#)
[日本語](#)
[한국어](#)
[Nederlands](#)
[Русский](#)
[Slovenčina](#)
[Tagalog](#)
[Українська](#)
[简体中文](#)

Partial translations available in

[Čeština](#)
[Deutsch](#)
[Македонски](#)
[Polski](#)
[Српски](#)
[Ўзбекса](#)
[繁體中文](#)

Translations started for

[azərbaycan dili](#)
[Беларуская](#)
[ଓଡ଼ିଆ](#)
[Indonesian](#)
[Italiano](#)
[Bahasa Melayu](#)
[Português \(Brasil\)](#)
[Português \(Portugal\)](#)
[Türkçe](#)

 The source of this book is hosted on [GitHub](#).
 Patches, suggestions and comments are welcome.

[Chapters](#) • 2nd Edition

2.3 Git Basics - Viewing the Commit History

Viewing the Commit History

After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened. The most basic and powerful tool to do this is the `git log` command.

These examples use a very simple project called "simplegit". To get the project, run

```
$ git clone https://github.com/schacon/simplegit
```

When you run `git log` in this project, you should get output that looks something like this:

```
$ git log
commit c82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@geek-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number
```

```
commit 085bb3bc6008e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@geek-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test
```

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@geek-mail.com>
Date: Sat Mar 15 10:31:28 2008 -0700

    first commit
```

By default, with no arguments, `git log` lists the commits made in that repository in reverse chronological order, that is, the most recent commits show up first. As you can see, this command lists each commit with its SHA-1 checksum, the author's name and email, the date written, and the commit message.

A huge number and variety of options to the `git log` command are available to show you exactly what you're looking for. Here, we'll show you some of the most popular.

One of the more helpful options is `-p` or `--patch`, which shows the difference (the `patch` output) introduced in each commit. You can also limit the number of log entries displayed, such as using `-2` to show only the last two entries.

```
$ git log -p -2
commit c82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@geek-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 +5,7 @@ require 'rake/gempackagetask'
spec = Gem::Specification.new do |s|
  s.platform = Gem::Platform::RUBY
  s.name = "simplegit"
-  s.version = "0.1.0"
+  s.version = "0.1.1"
  s.author = "Scott Chacon"
  s.email = "schacon@geek-mail.com"
  s.summary = "A simple gem for using Git in Ruby code."
```

```
commit 085bb3bc6008e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@geek-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700
```

removed unnecessary test

```
diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ee..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
```

This option displays the same information but with a diff directly following each entry. This is very helpful for code review or to quickly browse what happened during a series of commits that a collaborator has added. You can also use a series of summarizing options with `git log`. For example, if you want to see some abbreviated stats for each commit, you can use the `--stat` option:

```
$ git log --stat
commit c82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@geek-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 085bb3bc6008e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@geek-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700
```

removed unnecessary test

```
lib/simplegit.rb | 5 -----
1 file changed, 5 deletions(-)
```

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@geek-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(+)

As you can see, the `--stat` option prints below each commit entry a list of modified files, how many files were changed, and how many lines in those files were added and removed. It also puts a summary of the information at the end.

Another really useful option is `--pretty`. This option changes the log output to formats other than the

`default`. A few useful options are available for you to use. The `format` option prints each commit as

git log --pretty=oneline

The most interesting option is `format`, which allows you to specify your own log output format. This is especially useful when you’re generating output for machine parsing—because you specify the format explicitly, you know it won’t change with updates to Git.

```
$ git log --pretty=format:"%h %an : %s"
ca82a6d - Scott Chacon, 6 years ago : changed the version number
085bb3b - Scott Chacon, 6 years ago : removed unnecessary test
a11bf0a - Scott Chacon, 6 years ago : first commit
```

Useful options for `git log --pretty=format` lists some of the more useful options that `format` takes.

Table 1. Useful options for `git log --pretty=format`

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

You may be wondering what the difference is between `author` and `committer`. The author is the person who originally wrote the work, whereas the committer is the person who last applied the work. So, if you send in a patch to a project and one of the core members applies the patch, both of you get credit—you as the author, and the core member as the committer. We’ll cover this distinction a bit more in [Distributed Git](#).

The `oneline` and `format` options are particularly useful with another log option called `--graph`. This option adds a nice little ASCII graph showing your branch and merge history:

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHILD on trap
* 5e3e11l 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 xmleschema
* 1ld191e Merge branch 'defunkt' into local
```

This type of output will become more interesting as we go through branching and merging in the next chapter.

Those are only some simple output-formatting options to `git log`—there are many more. [Common options to git log](#) lists the options we’ve covered so far, as well as some other common formatting options that may be useful, along with how they change the output of the log command.

Table 2. Common options to `git log`

Option	Description
<code>-p</code>	Show the patch introduced with each commit.
<code>--stat</code>	Show statistics for files modified in each commit.
<code>--shortstat</code>	Display only the changed/insertions/deletions line from the <code>--stat</code> command.
<code>--name-only</code>	Show the list of files modified after the commit information.
<code>--name-status</code>	Show the list of files affected with added/modifed/deleted information as well.
<code>--abbrev-commit</code>	Show only the first few characters of the SHA-1 checksum instead of all 40.
<code>--relative-date</code>	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
<code>--graph</code>	Display an ASCII graph of the branch and merge history beside the log output.
<code>--pretty</code>	Show commits in an alternate format. Options include <code>oneline</code> , <code>short</code> , <code>full</code> , <code>fuller</code> , and <code>format</code> (where you specify your own format).
<code>--oneline</code>	Shorthand for <code>--pretty=oneline --abbrev-commit</code> used together.

Limiting Log Output

In addition to output-formatting options, `git log` takes a number of useful limiting options; that is, options that let you show only a subset of commits. You’ve seen one such option already—the `-2` option, which displays only the last two commits. In fact, you can do `-nN`, where `N` is any integer to show the last `n` commits. In reality, you’re unlikely to use that often, because Git by default pipes all output through a pager so you see only one page of log output at a time.

However, the time-limiting options such as `--since` and `--until` are very useful. For example, this command gets the list of commits made in the last two weeks:

```
$ git log --since=2.weeks
```

This command works with lots of formats—you can specify a specific date like “2008-01-15”, or a relative date such as “2 years 1 day 3 minutes ago”.

You can also filter the list to commits that match some search criteria. The `--author` option allows you to

itter on a specific author, and the `--grep` option lets you search for keywords in the commit messages.

Note

You can specify more than one instance of both the `--author` and `--grep` search criteria, which will limit the commit output to commits that match any of the `--author` patterns and any of the `--grep` patterns; however, adding the `--all-match` option further limits the output to just those commits that match all `--grep` patterns.

Another really helpful filter is the `-S` option (colloquially referred to as Git's "pickaxe" option), which takes a string and shows only those commits that changed the number of occurrences of that string. For instance, if you wanted to find the last commit that added or removed a reference to a specific function, you could call:

```
$ git log -S function_name
```

The last really useful option to pass to `git log` as a filter is a path. If you specify a directory or file name, you can limit the log output to commits that introduced a change to those files. This is always the last option and is generally preceded by double dashes (`--`) to separate the paths from the options.

In [Options to limit the output of git log](#) we'll list these and a few other common options for your reference.

Table 3. Options to limit the output of `git log`

Option	Description
<code>--n</code>	Show only the last n commits
<code>--since</code> , <code>--after</code>	Limit the commits to those made after the specified date.
<code>--until</code> , <code>--before</code>	Limit the commits to those made before the specified date.
<code>--author</code>	Only show commits in which the author entry matches the specified string.
<code>--committer</code>	Only show commits in which the committer entry matches the specified string.
<code>--grep</code>	Only show commits with a commit message containing the string.
<code>-S</code>	Only show commits adding or removing code matching the string

For example, if you want to see which commits modifying test files in the Git source code history were committed by Junio Hamano in the month of October 2008 and are not merge commits, you can run something like this:

```
$ git log --pretty=format:"%h - %s" --author='Junio C Hamano' --since="2008-10-01" \
--before="2008-11-01" --no-merges -- t/
5e10e3b - Fix testcase failure when extended attributes are in use
acd3be - Enhance hold_lock_file_for_(update,append)() API
f5d3754 - demonstrate breakage of detached checkout with symbolic link HEAD
dia43f2 - reset --hard/read-tree --reset -u: remove unmerged new paths
51a94af - Fix "checkout --track -b newbranch" on detached HEAD
b0ad11e - pull: allow "git pull origin $something:$current_branch" into an unborn branch
```

Of the nearly 40,000 commits in the Git source code history, this command shows the 6 that match those criteria.

Tip

Preventing the display of merge commits

Depending on the repository used in your repository, it's possible that a sizable percentage of the commits in your log history are just merge commits, which typically aren't very informative. To prevent this large volume of merge commits cluttering up your log history, simply add the log option `--no-merges`.

[prev](#) | [next](#)