Git log, status, and diffs

Reuven M. Lerner, PhD
reuven@lerner.co.il • http://lerner.co.il/

Looking at the log

- “git log” shows the commit history
- Right now, there’s not much to show!
- Let’s look at a few ways to view the log
- This is one of the most important commands
- On the command line, logs are displayed using "more" or "less" — quit from it using "q"

Simple log

```
$ git log

commit b53dd549c4891f094ab427852899da958fb9ae21

Author: Reuven Lerner <reuven@lerner.co.il>

Date:   Sat Sep 17 23:12:33 2011 +0300


    I added a file
```

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

One-liner

```
$ git log --pretty=oneline

b53dd549c4891f094ab427852899da958fb9ae21 I added a file
```

With diffs

```
$ git log -p
commit b53dd549c4891f094ab427852899da958fb9ae21
Author: Reuven Lerner <reuven@lerner.co.il>
Date:   Sat Sep 17 23:12:33 2011 +0300

    I added a file

diff --git a/hello.rb b/hello.rb
new file mode 100644
index 0000000..3ad1879
--- /dev/null
+++ b/hello.rb
@@ -0,0 +1 @@
+print "Hello, world"
```

Log of one file

- Normally, I get the log of the entire repository:

```
git log
```

- But I can ask for the log of a particular file:

```
git log foo.rb
```

- Or of the files in a particular directory:

```
git log config
```

OK, that's about it!

- You have now done the most common actions in a Git repository:
  - Staging a file ("git add")
  - Committing a file ("git commit")
  - Looking at the log of commits ("git log")
- These three commands are probably 90% of my work with Git.

Let’s change our file

```
print "Hello again, world"

print "And goodbye for now"
```

Check the status

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   hello.rb
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    hello.rb~
no changes added to commit (use "git add" and/or "git commit -a")
```

"Not staged for commit"

- Notice that "hello.rb", which was already committed, is "not staged for commit."
- In other words: We have modified hello.rb. If we want to commit the new version, we must stage it, using "git add hello.rb"
- (Or we can just commit it, either with "git commit hello.rb" or "git commit -a")

Check the status

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   hello.rb
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    hello.rb~
no changes added to commit (use "git add" and/or "git commit -a")
```

Untracked file?!?

- `hello.rb~` is a backup file from Emacs
- We don't want Git to track backup files, so let's tell Git to ignore them
- Create a file called .gitignore, and put it at the top of your repository.
- Each line contains a file-globbing pattern to ignore or a comment (starting with #) that goes to the end of the line

Our .gitignore file

```
*~
```

Now, recheck status

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   hello.rb
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   .gitignore
```

What's going on?

- .gitignore is working, so we ignore *~
- But .gitignore is a file like any other, and thus needs to be staged and committed
- Let's stage it and commit it, without committing hello.rb

Add, commit .gitignore

```
$ git add .gitignore

$ git commit .gitignore -m 'Added .gitignore'

[master a4d9e4f] Added .gitignore

 1 files changed, 1 insertions(+), 0 deletions(-)

 create mode 100644 .gitignore
```

The log

```
$ git log --pretty=oneline
a4d9e4f924765a1abec8a3ea711c657e8952aff5 Added .gitignore
b53dd549c4891f094ab427852899da958fb9ae21 I added a file
```

The (short) status

```
$ git status --short

 M hello.rb



$ git status -s

 M hello.rb



$ git status --short --ignored

 M hello.rb

!! hello.rb~
```

"git status -s" symbols

(empty)	unmodified
M	modified
A	added
D	deleted
R	renamed
C	copied
U	updated but not merged

Commit our file

```
$ git commit hello.rb -m 'Updated to say bye'

[master f7310fd] Updated to say bye

 1 files changed, 2 insertions(+), 1 deletions(-)
```

Log

```
$ git log --pretty=oneline
f7310fd5064c4cc649f96c9d12faedb8327fa296 Updated to say bye
a4d9e4f924765a1abec8a3ea711c657e8952aff5 Added .gitignore
b53dd549c4891f094ab427852899da958fb9ae21 I added a file
```

Log of a single file

```
$ git log --pretty=oneline hello.rb
f7310fd5064c4cc649f96c9d12faedb8327fa296 Updated to say bye
b53dd549c4891f094ab427852899da958fb9ae21 I added a file

$ git log --pretty=oneline .gitignore
a4d9e4f924765a1abec8a3ea711c657e8952aff5 Added .gitignore
```

Log since ...

```
$ git log --pretty=oneline --since='last week'
```

Log since ...

```
$ git log --pretty=oneline --since='last month'
93399cbfc5f1d48e49c95429ff58d792f01c66b7 Updated with Joy fix
4ef392744d3b277a10da122b952d445c7222be16 Fixed JavaScript error
3c7cf0fe41fe274f04e32892897bbc0a52cbdfd5 Added updates from Joy
c9e23f038c7fd5f5640e7886f83643a0d077ac28 Made fix that Joy suggested, and updated pub ID
305aa3f0f38480aa3e9a1ea67cd0c02fe9d2a565 Added stupid btnsubmit_x
c8acaa85b490f7d61719d77b290892ddb5a14e2a Maybe PHP will work now?
2bfe3ee04006ce9a778a3d35254e464818ebd4fa Overcame PHP bug?
16bf9eb8baff13cf2ccc818fe9b246f66673c67e Updated textarea
58a47f37ff8a7aa2e3792e2864e5c0d5d6567d51 Huh?  He used GET?!?
87e00ca5c905ca9391cd41f121adf4877c5ba1c3 Use search
b9c16c07409a133be90ab75f45b638ea3b235d14 Try out changing form entry
```

Log since ... until ...

```
$ git log --pretty=oneline --since='june 2011' --until='august 2011'
4e00a6a53ab18154d6656596e4d819f9c85263ea Added Damians changes
59893c912119b966d86e5d8d32f092cf3dcbe52a Got rid of new stuff
f19dc691c27bc0cef0ff2b4a33707671d83b2dd8 Renamed buyback to newbuyback
94fe4939c86aff470199b1bcf0c8d04cc39c54b1 Added Indian stuff
1df1944b43a0d73ee5a1899b3c11c2e8e63a4a7b Updated sitemap with Mike\s additions
43ba11f23dc38885932bd05c50c01db79a1670e2 Removed buyback box from FAW
a59b81c4392ce7d40acfea8af3ed7015e553bfb3 Switched label
88a23aaa107e29de14be9bb3c76afa3d971e0c14 Switched to SellBackYourbook
```

Log from a particular user

```
git log --author='reuven'

git log --author='reuven' --since="2 months"
```

Visualizing

- Even command-line people often want to see a visual representation of their commit history
- "gitk" uses the Tk toolkit to show things
- You can also use SourceTree, from Atlassian (http://www.sourcetreeapp.com/)

Simple visualizing

- You can also use the --graph option to "git log", which shows a simple graph along the left side

Diffs

- “Diffs” are our way of seeing changes that have happened to files
- By default, we see the difference between our working directory and the latest commit

No changes? No diffs!

```
$ git status
# On branch master
nothing to commit (working directory clean)

$ git diff
$
```

Diff with a commit

```
$ git diff b53d
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..b25c15b
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+*~
diff --git a/hello.rb b/hello.rb
index 3ad1879..94da73f 100644
--- a/hello.rb
+++ b/hello.rb
@@ -1 +1,2 @@
-print "Hello, world"
+print "Hello again, world"
+print "And goodbye for now"
```

Diff of two commits

```
$ git diff a4d9..b53d hello.rb


$ git diff f731..a4d9 hello.rb
diff --git a/hello.rb b/hello.rb
index 94da73f..3ad1879 100644
--- a/hello.rb
+++ b/hello.rb
@@ -1,2 +1 @@
-print "Hello again, world"
-print "And goodbye for now"
+print "Hello, world"
```

What are those @@ signs?

```
--- a/hello.rb
+++ b/hello.rb
@@ -1,2 +1 @@
```

- The @@ signs mean:
- - start of original document
- , separating old from new
- + start of new document
- Each range is firstnum, lastnum

Which lines?

```
@@ -1,2 +1 @@
```

- The above shows that from line 1 to line 2 in the old file was turned into line 1 (through line 1) in the new file

Changes to a file since a particular commit

```
$ git diff b53d hello.rb
diff --git a/hello.rb b/hello.rb
index 3ad1879..94da73f 100644
--- a/hello.rb
+++ b/hello.rb
@@ -1 +1,2 @@
-print "Hello, world"
+print "Hello again, world"
+print "And goodbye for now"
```

git difftool

- "git difftool" is a front end to "git diff" that takes the same arguments, but lets you use a different diff tool, including a graphical one
- To see what tools are available on your system, invoke:

```
git difftool --tool-help
```

- You can configure your personal preference with a global configuration:

```
git config --global diff.tool TOOLNAME
```

- To choose a particular tool once, use the "--tool" option:

```
git difftool --tool=emerge
```

Modify our file again

```
print "Hello once again, world"
print "And goodbye for now"
print "But wait, one more thing..."
```

Stage it and diff...

```
$ git add hello.rb

$ git diff

$
```

Huh?

- By default, “diff” compares your working directory with the latest commit
- In other words, once you have staged a file ("git add"), diff won't show change since the last commit
- The solution? Use --cached with "git diff" (You can also say --staged, if you prefer.)

To summarize

- "git diff" compares the current working directory with the most recent commit
- "git add" creates a commit (sort of), which means that once you have staged something, "git diff" will compare the working directory with what you have staged… which won't show anything
- To compare the working directory with the latest (real) commit, use --cached or compare with HEAD:

```
git diff HEAD
```

Once again...

```
$ git diff --cached
diff --git a/hello.rb b/hello.rb
index 94da73f..aa93483 100644
--- a/hello.rb
+++ b/hello.rb
@@ -1,2 +1,3 @@
-print "Hello again, world"
+print "Hello once again, world"
 print "And goodbye for now"
+print "But wait, one more thing..."
```

Diffs between commits

- Compare between working directory and a commit:

```
git diff ba14
```

- Compare between any two commits:

```
git diff cf3e..ba14
```

- Compare between a commit and a branch:

```
git diff cf3e master
```

Ignore whitespace

- Using the "-w" flag to diff ignores whitespace
- This is great if you want to ignore indentation changes, or what new blank lines have been added/removed

Patches

- A "patch" is a file that you can e-mail to other people, describing the differences between two commits
- Sending a patch is the old, slow, non-connected way of doing a pull

Creating a patch

- A patch is a diff from the current state to a previous state. *You almost always want to do this relative to another branch!*
- Create a new branch with your feature.
- Make some commits.
- Create a patchfile:

```
git format-patch master --stdout > new-feature.patch
```

Applying patches

- Let's assume that someone has sent you a patch. Now what?

```
git apply --stat new-feature.patch
```

```
git apply --check new-feature.patch
```

- Use "git am" to sign off on the patch!

```
git am --signoff < new-feature.patch
```