Git for neuroscientists

Release 0.1

The UNIC version control evangelists

September 23, 2016

Contents

Authors: **The UNIC version control evangelists**

Version 0.1

September 23, 2016

CHAPTER 1

Basic ideas

Any time multiple versions of a document exist, whether due to a document changing over time, or because multiple authors are working on it, some kind of version control is needed.

Version control allows:

- accessing any version from the original to the most recent;
- seeing what has changed from one version to the next;
- giving a label of some kind to distinguish a particular version.

1.1 Examples of version control systems

The simplest method of version control is probably the most widely used in science: changing the file name.

(http://www.phdcomics.com/comics.php?f=1323)

(http://www.phdcomics.com/comics.php?f=1531)

Fig. 1.1: from *"Piled Higher and Deeper" by Jorge Cham* www.phdcomics.com

Other examples include:

- "track changes" in Microsoft Word
- Time Machine in Mac OS X
- versioning in Dropbox, Google Drive
- formal version control systems such as CVS, Subversion, Mercurial, Git

1.2 The importance of tracking projects, not individual files

Early version control systems, such as CVS, track each file separately - each file has its own version number. The same is true of Dropbox, Microsoft Word.

This is a problem when you make changes to several files at once, and the changes in one file depend on changes in another.

In modern version control systems, and in backup-based systems such as Time Machine, entire directory trees are tracked as a unit, which means that each version corresponds to the state of an entire project at a point in time.

1.3 Advantages of formal version control systems

- explicit version number for each version
- easy to switch between versions
- easy to see changes between versions
- tools to help merge incompatible changes

In the next sections, we will use Git (http://git-scm.com/), probably the most widely used modern version control system, to introduce the principles of version control. We will demonstrate both Git's command-line interface and a graphical interface, SourceTree (http://www.sourcetreeapp.com).

CHAPTER 2

Getting started with Git

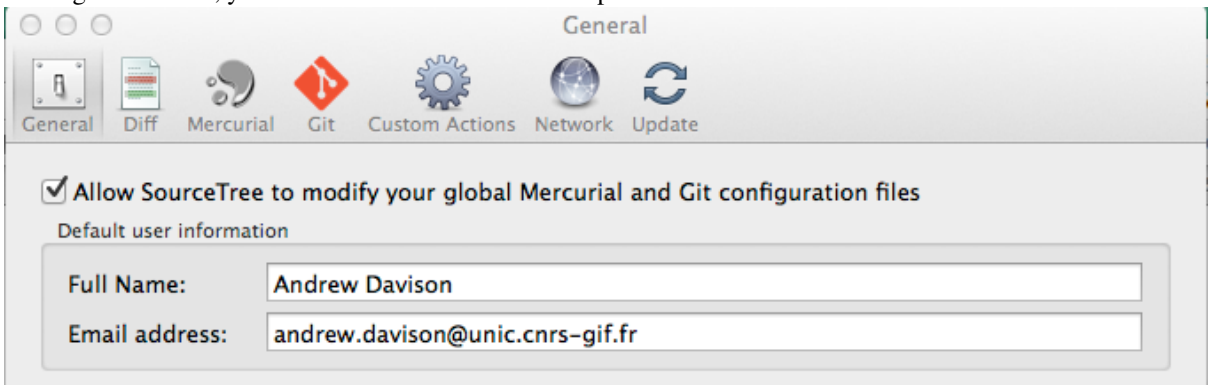
2.1 Installing Git

Git is available for Linux, Mac OS X, and Windows. For Linux, it will certainly be available in your package manager. For Windows and Mac OS X, download from http://git-scm.com/downloads. If you're using SourceTree (download from http://www.sourcetreeapp.com), Git is included.

(http://www.sourcetreeapp.com) Once you've installed it, you should tell Git about yourself:

```
git config --global user.name  Andrew Davison
git config --global user.email andrew.davison@unic.cnrs-gif.fr
```

If using SourceTree, you can set this information in the preferences.



2.2 Creating a repository

We start by introducing three concepts:

Working copy the set of files that you are currently working on

Index a staging area

Repository a "database" containing the entire history of your project (all versions)

As an example, we will use the Brian code from this paper:

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, et al. (2007) Simulation of networks of spiking neurons: A review of tools and strategies. *J Comp Neurosci* **23**:349-98

available from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=83319

```
$ unzip destexhe_benchmarks.zip
$ cd destexhe_benchmarks
$ cp -r Brian ~/my_network_model
$ cd ~/my_network_model
$ ls
COBA.py             COBAHH.py       CUBA.py         README.txt
```

We're going to take this code as the starting point for our own project, and we want to keep track of the changes we make.

The first step is to create a repository, where all the versions will be stored. This is very simple:

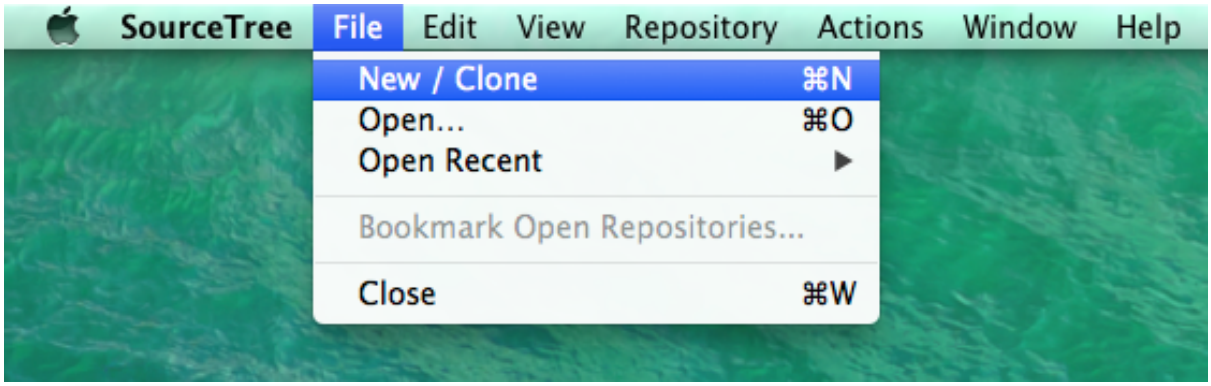
```
$ git init
Initialized empty Git repository in /Volumes/USERS/andrew/my_network_model/.git/
```

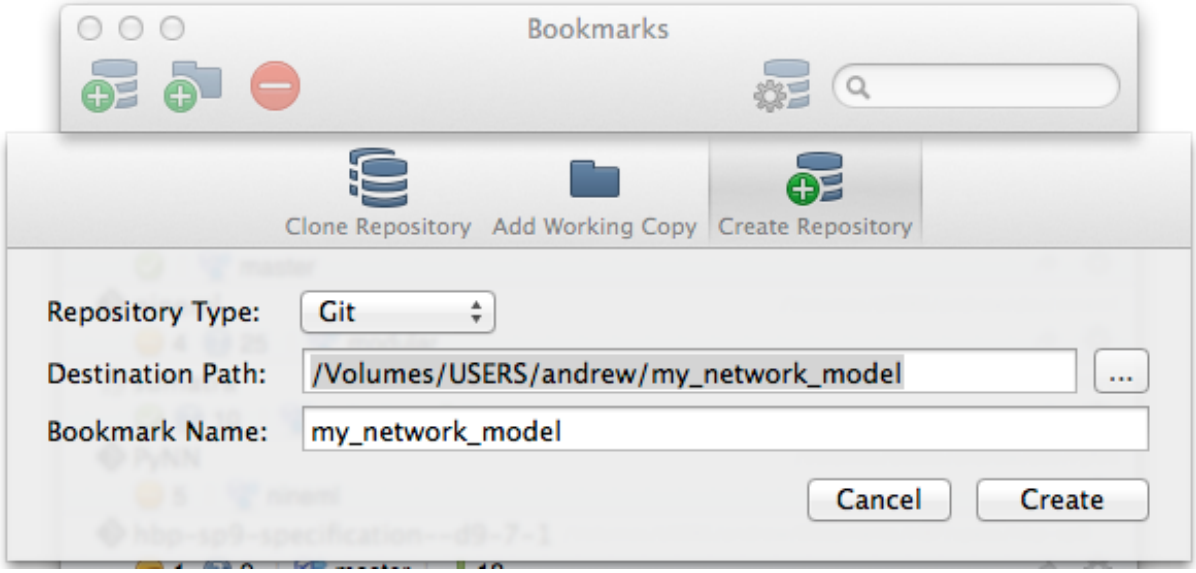
Nothing seems to have happened. In fact, the **`git init`** command has created a new subdirectory:

```
$ ls -a
.           ..              .git            COBA.py         COBAHH.py       CUBA.py         READM
```

You almost never need to care about what is in this directory: this is where Git will store all the information about the repository.

With SourceTree, you create a new repository via the File menu:





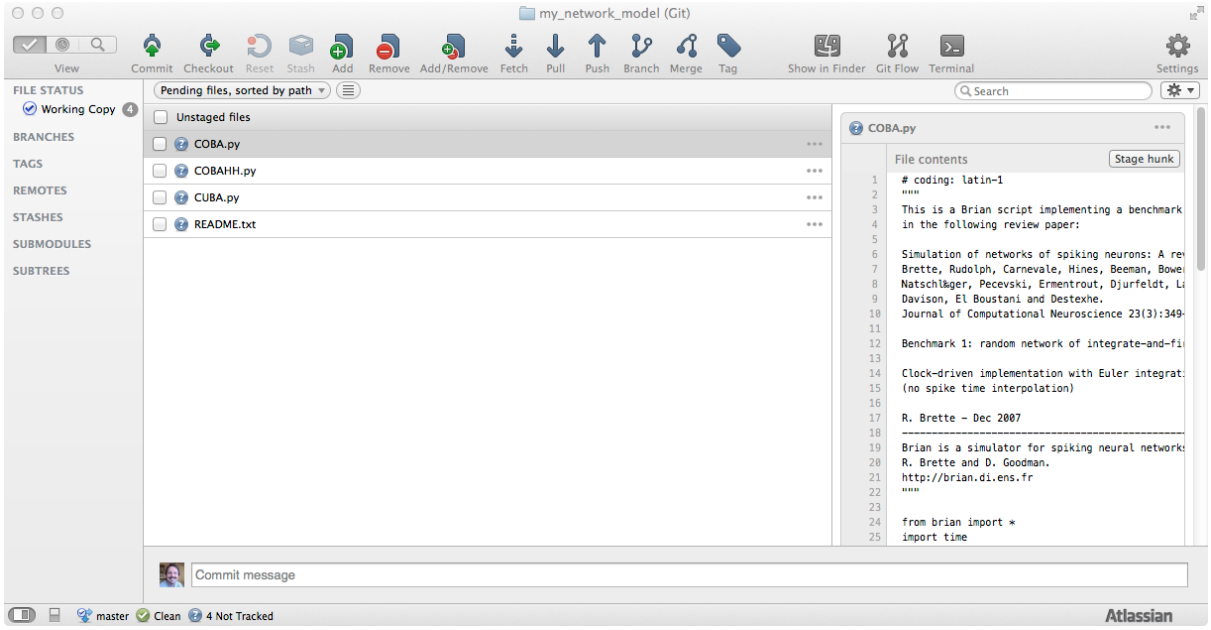
2.3 Adding files to the repository

Now we need to tell Git which files are part of our project. On the command line:

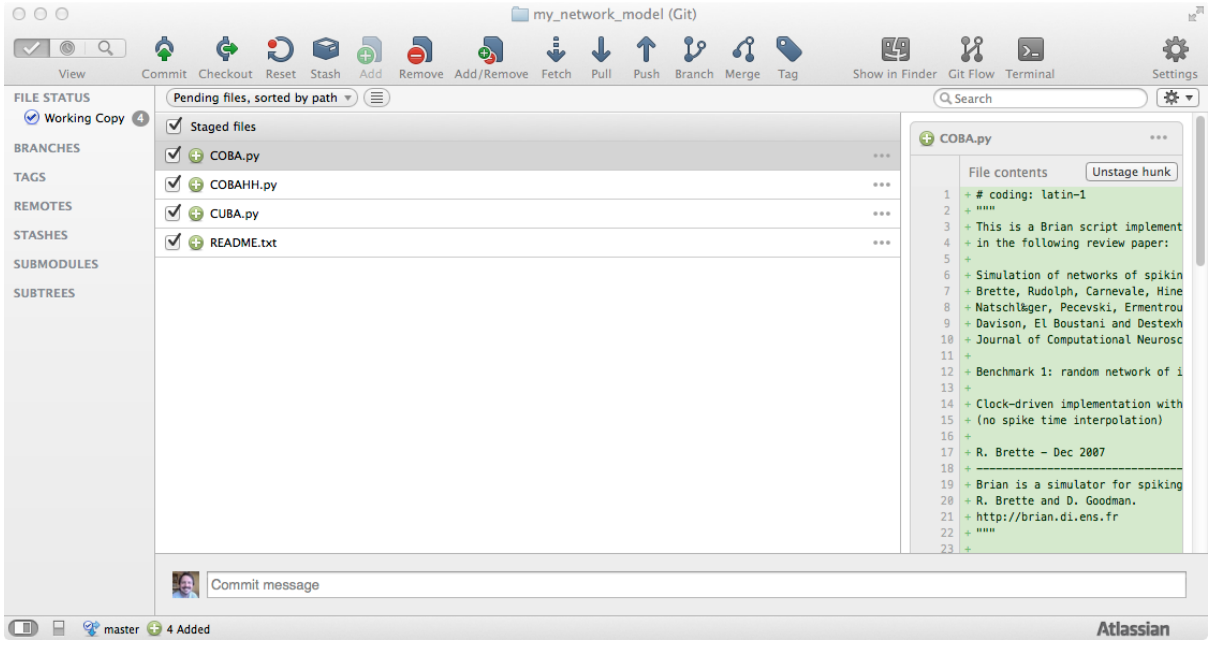
```
$ git add *
```

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   COBA.py
        new file:   COBAHH.py
        new file:   CUBA.py
        new file:   README.txt
```

In SourceTree, we can see our files listed as “Unstaged”



When we check the boxes, they are listed as "Staged".

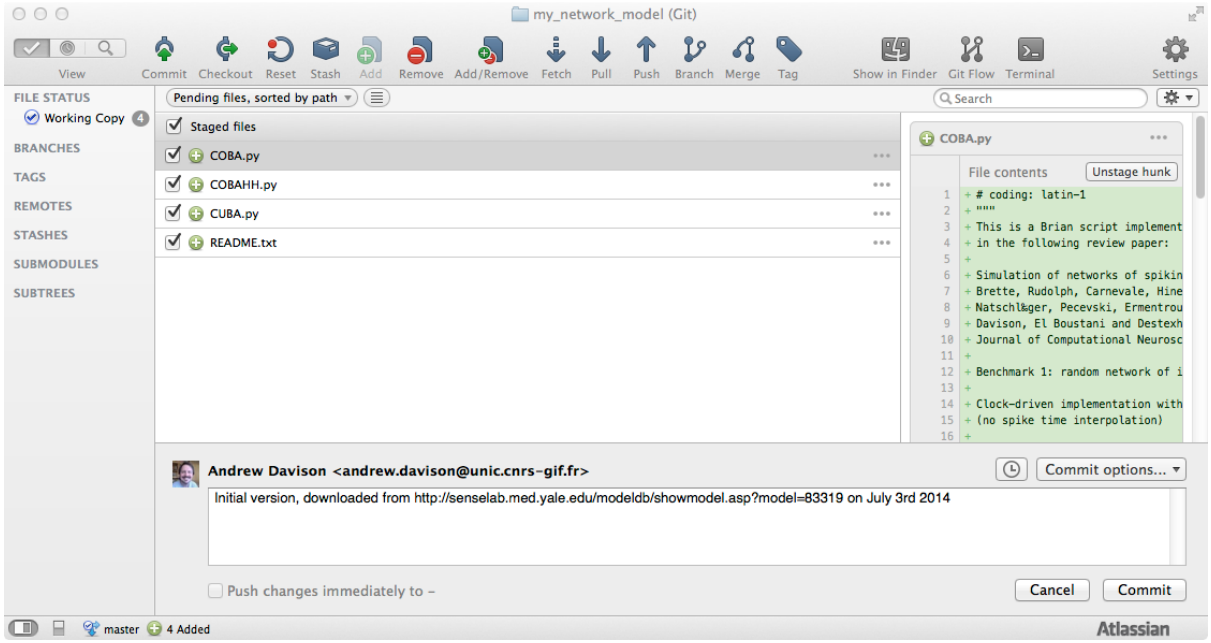


2.4 Committing changes

These files are now *queued* or *staged* to be added to the repository, but they are not yet there. Nothing is definitive until we make a *commit* (also known as a "*check-in*"). Via the command line:

```
$ git commit -m "Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.a
[master (root-commit) b0c5d2c] Initial version, downloaded from http://senselab.med.yale.edu/model
 4 files changed, 259 insertions(+)
 create mode 100644 COBA.py
 create mode 100644 COBAHH.py
 create mode 100644 CUBA.py
 create mode 100644 README.txt
```

Via SourceTree; in both cases we have to give a short message summarizing what changes were made.

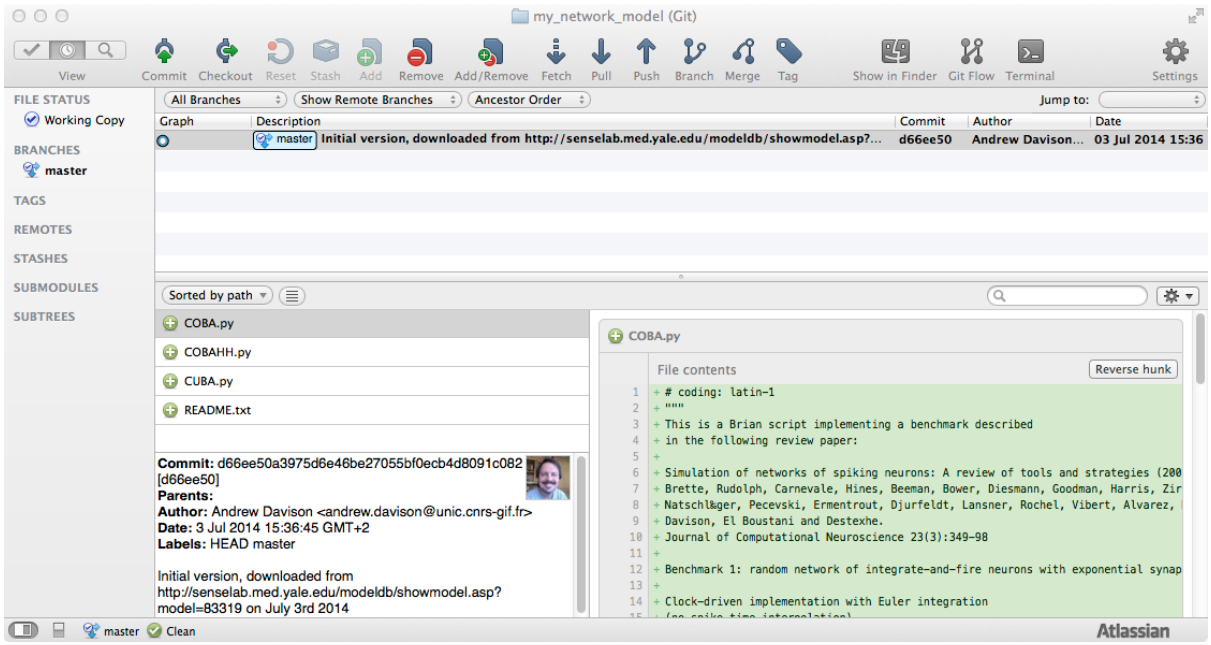


2.5 Viewing the history of changes

The log command lists all the different versions stored in the repository. For now, of course, we have only one:

```
$ git log
commit d66ee50a3975d6e46be27055bf0ecb4d8091c082
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:36:45 2014 +0200

    Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=833
```

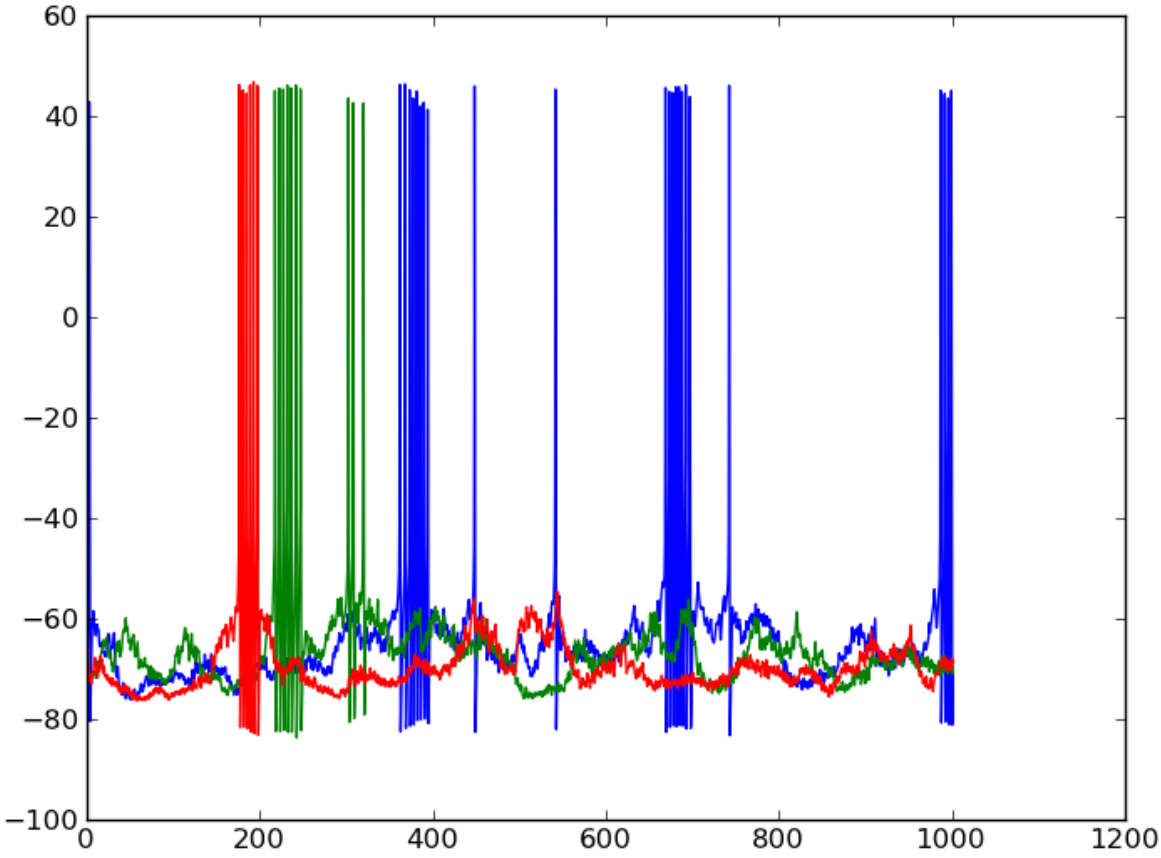
In SourceTree, we use the "Log" view.



Now let's run the code:

```
$ python COBAHH.py
Network construction time: 0.814524173737 seconds
Simulation running...
Simulation time: 45.7264661789 seconds
126014 excitatory spikes
29462 inhibitory spikes
```

This pops up a window with the following figure:



We'd prefer to save the figure to a file for further use, rather than work with the model interactively, so let's change the last lines of the script from:

```
plot(trace.times/ms, trace[1]/mV)
plot(trace.times/ms, trace[10]/mV)
plot(trace.times/ms, trace[100]/mV)
show()
```

to

```
plot(trace.times/ms, trace[1]/mV)
plot(trace.times/ms, trace[10]/mV)
plot(trace.times/ms, trace[100]/mV)
savefig("COBAHH_output.png")
```

2.6 Seeing what's changed

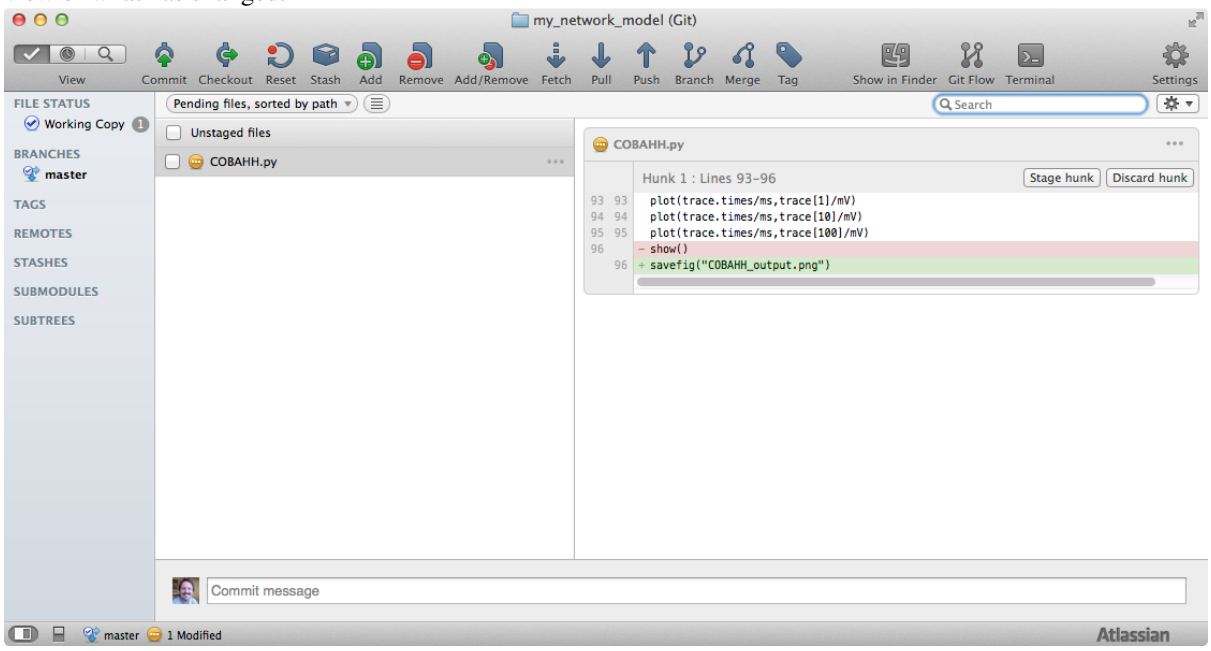
Now if we run **`git status`** we see:

```
$ 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:   COBAHH.py

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

```
$ git diff
diff --git a/COBAHH.py b/COBAHH.py
index d9eda02..22bcf1a 100644
--- a/COBAHH.py
+++ b/COBAHH.py
@@ -93,4 +93,4 @@ print Mi.nspikes,"inhibitory spikes"
 plot(trace.times/ms,trace[1]/mV)
 plot(trace.times/ms,trace[10]/mV)
 plot(trace.times/ms,trace[100]/mV)
-show()
+savefig("COBAHH_output.png")
```

This is even easier in SourceTree: we automatically see a list of files that have been changed, and a colour-coded view of what has changed.



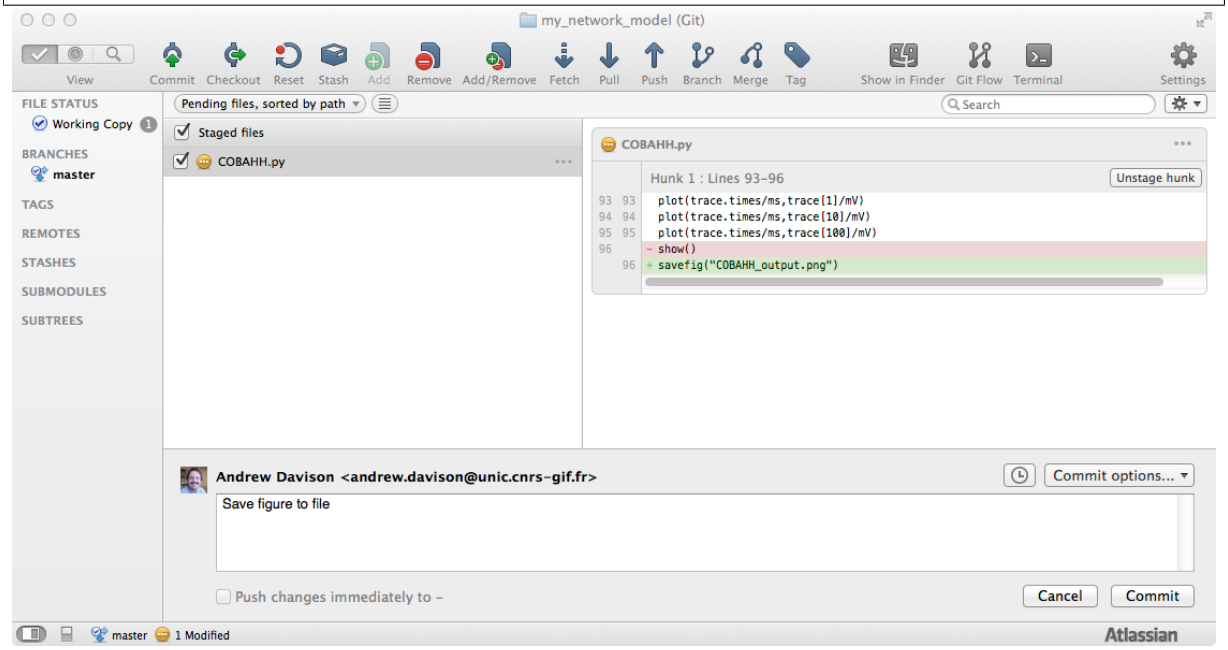
Now let's commit the changes...

```
$ git add COBAHH.py
```

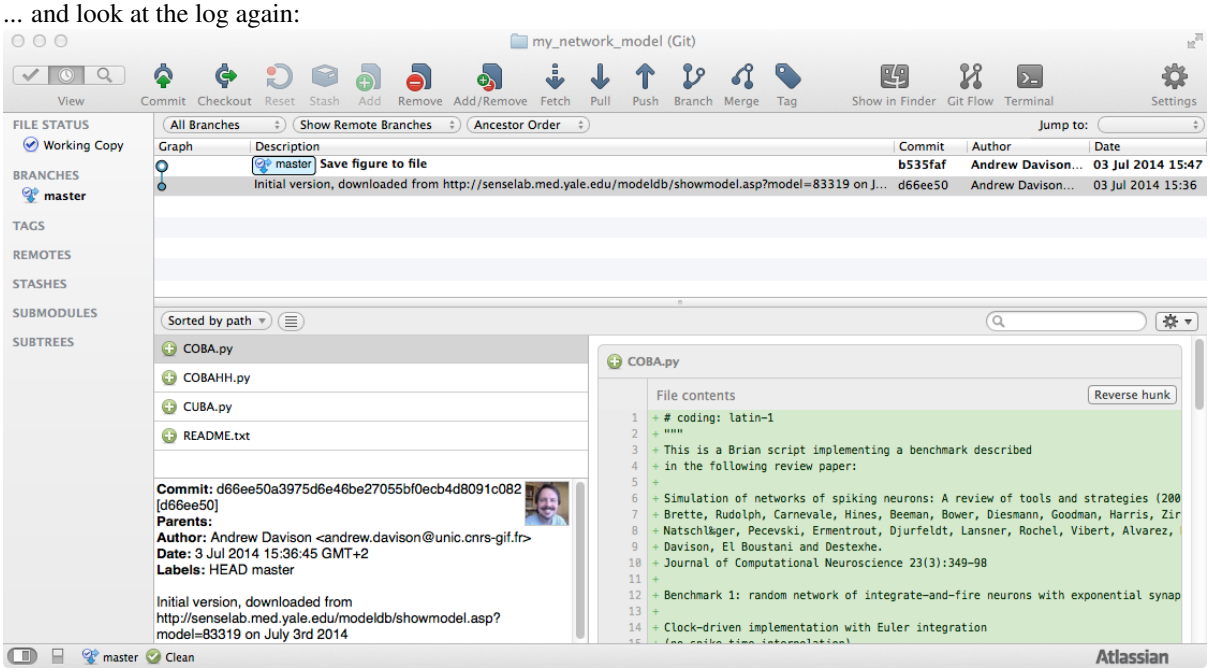
```
$ git commit -m 'Save figure to file'
[master 1c5d37f] Save figure to file
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git log
commit b535fafe86c0caeb2b38ef30f0e96c9904ff56d6
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:47:45 2014 +0200

    Save figure to file

commit d66ee50a3975d6e46be27055bf0ecb4d8091c082
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:36:45 2014 +0200
```

```
    Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=833
```



... and look at the log again:



We see both versions, with commit messages, the author name and the date.

2.7 Switching between versions

To switch between versions (you should not do this if you have modified any of the files - commit your changes first), use **`git checkout`**:

```
$ git checkout d66ee50a3975d6e46be27055bf0ecb4d8091c082
Note: checking out 'd66ee50a3975d6e46be27055bf0ecb4d8091c082'.

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 performing another checkout.
```

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

  git checkout -b new_branch_name

HEAD is now at d66ee50... Initial version, downloaded from http://senselab.med.yale.edu/modeldb/sl
```

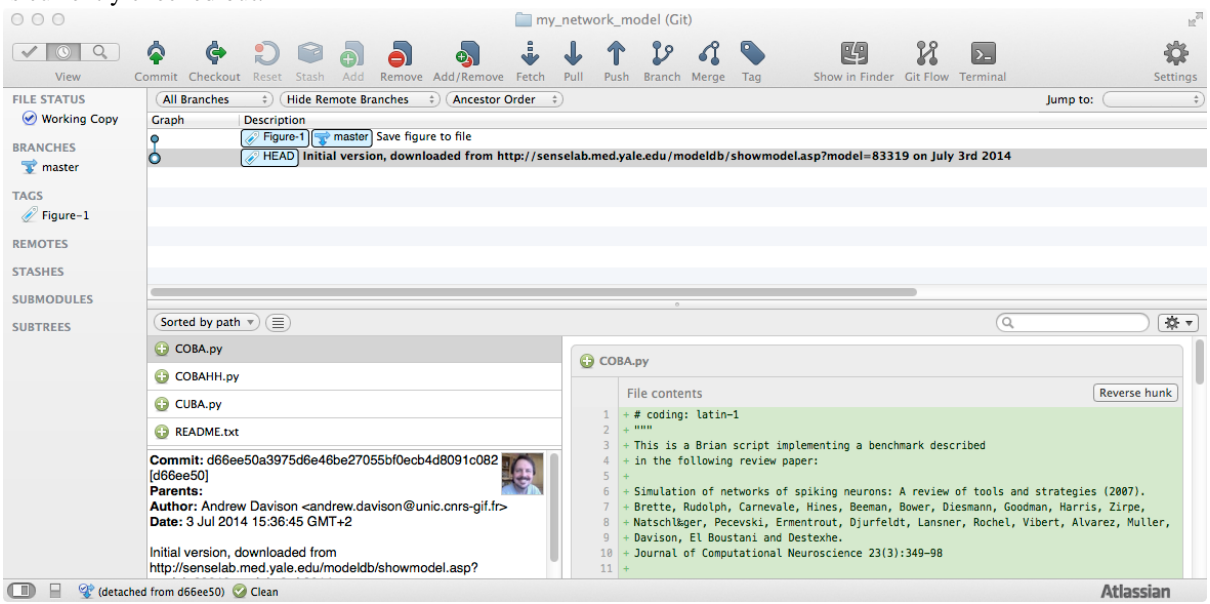
This will change the files in your working copy to reflect the state they had when you committed that particular version. (You can ignore the message about 'detached HEAD' for now). We can check that our working copy has really changed by looking at the end of the COBAHH.py file: we see that we are back to using the "show()" command instead of "savefig()".

```
$ tail COBAHH.py
run(1000*msecond)
duration=time.time()-start_time
print "Simulation time:",duration,"seconds"
print Me.nspikes,"excitatory spikes"
print Mi.nspikes,"inhibitory spikes"

plot(trace.times/ms,trace[1]/mV)
plot(trace.times/ms,trace[10]/mV)
plot(trace.times/ms,trace[100]/mV)
show()
```

Using **`git branch`** we can see which version we are currently using:

```
$ git branch
* (detached from d66ee50)
  master
```

In SourceTree, just click on the version you want to use. This will add a label "HEAD" to show you which version is currently checked out.



To switch to the most recent version, use *git checkout master*

```
$ git checkout master
Previous HEAD position was bfb2cd7... Initial version, downloaded from http://senselab.med.yale.ed
Switched to branch 'master'
```

```
$ git branch
* master
```

2.8 Giving informative names to versions

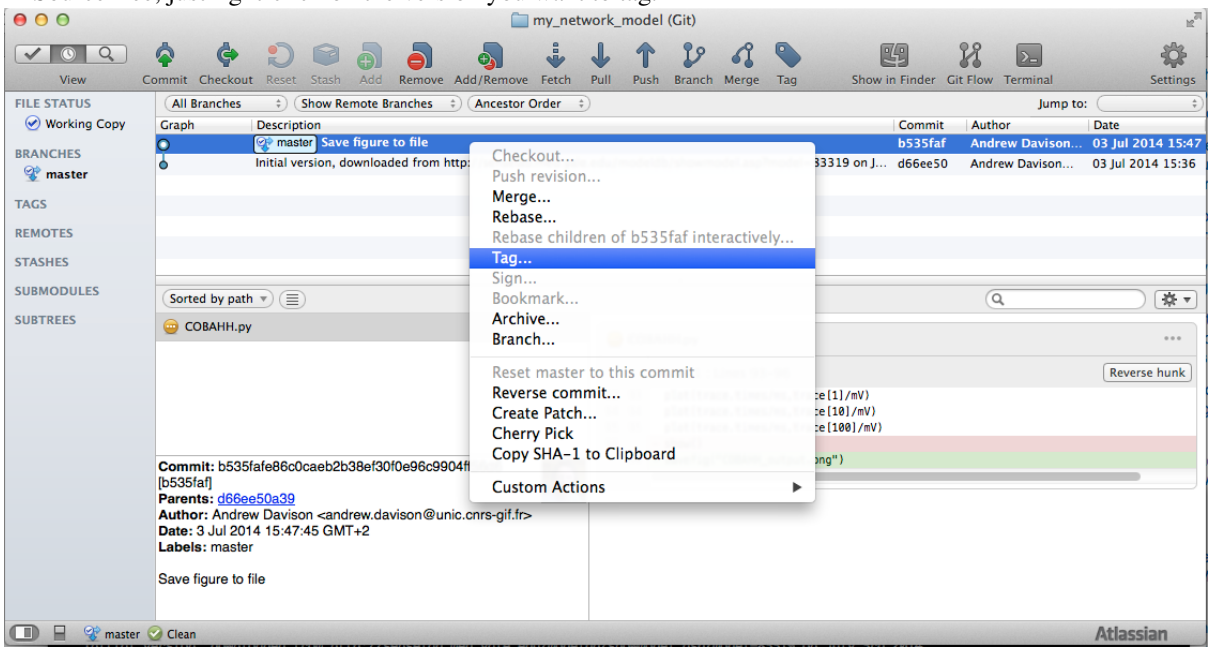
Remembering the version number for a particular version of interest (for example, the version used to generate a particular figure in your manuscript) can be difficult. For this reason, the **`git tag`** command can be used to give descriptive and memorable names to significant versions:

```
$ git tag Figure-1
```

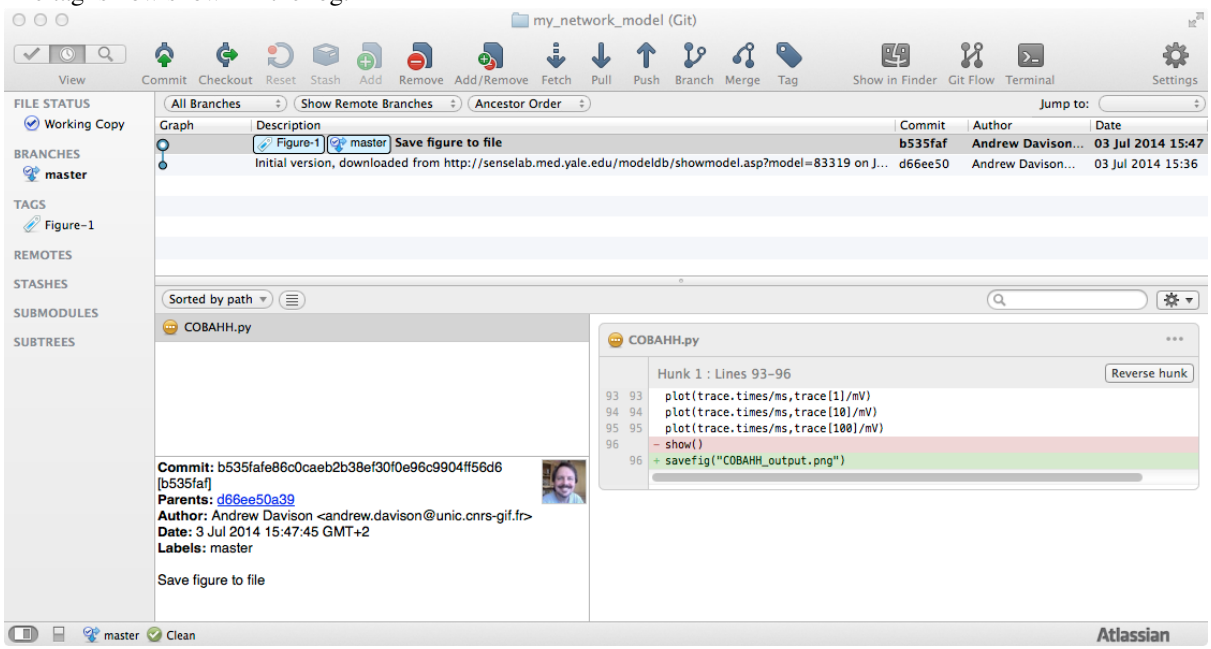
You can now switch to a tagged version using the tag name:

```
$ git checkout Figure-1
```

In SourceTree, just right-click on the version you want to tag.



The tag is now shown in the log.



2.9 Recap #1

So far, we have learned how to:

- Create a repository
- Add files to a repository
- Commit changes
- Move your code-base backwards and forwards in time

These operations are so easy and so useful that there is no reason not to use them for almost any work you do as a scientist. Any time I start a new project, whether writing code or writing a paper with LaTeX, I now run **`git init`** as soon as I've created a new directory for the project.

CHAPTER 3

Making backups

As well as helping to keep track of different versions of a project, version control systems are hugely useful for keeping backups of your code with minimal hassle.

Making a copy of your repository is as simple as moving to the location where the backup will be, and then using the **`git clone`** command.

```
$ cd /Volumes/USB_DRIVE
$ git clone ~/my_network_model
```

```
$ cd ~/Dropbox
$ git clone ~/my_network_model
```

```
$ ssh cluster.example.edu
(cluster)$ git clone ssh://my_laptop.example.edu/my_network_model
```

You can then keep the backup in-sync with the main repository by either using **`git pull`** in the backup location, or using **`git push`** in your working directory:

```
$ cd ~/my_network_model
$ git push /Volumes/USB_DRIVE/my_network_model master
Everything up-to-date
```

Why is this better than just copying the files? Because you no longer have to worry about versions or about over-writing your changes. i.e., no more:

> *"Now, is the version on my USB key newer than my local version? Did I change any files on my laptop?"*

or:

(http://www.phdcomics.com/comics.php?f=1323)

CHAPTER 4

Working on multiple computers

As an extension of the idea of backups, version control systems are excellent for keeping code in sync between multiple computers. Suppose you have a copy of your repository on your laptop, and you were working on the code in the airport.

```
(laptop)$ git diff
diff --git a/CUBA.py b/CUBA.py
index 84f75be..2bd2fa1 100644
--- a/CUBA.py
+++ b/CUBA.py
@@ -72,4 +72,4 @@ print "Simulation time:",duration,"seconds"
 print Me.nspikes,"excitatory spikes"
 print Mi.nspikes,"inhibitory spikes"
 plot(M.times/ms,M.smooth_rate(2*ms,'gaussian'))
-show()
++savefig("CUBA_output.png")
```

```
(laptop)$ git add CUBA.py
```

```
(laptop)$ git commit -m 'CUBA script now saves figure to file'
```

The log on your laptop now looks like this:

```
(laptop)$ git log
commit 4f568a24ffc4d8695d666bec90c5533dc4db3206
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 16:10:24 2014 +0200

    CUBA script now saves figure to file

commit 4b18663c05fd120e101aa14bfe8abea65f7c1700
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:50:06 2014 +0200

    Save figure to file

commit bfb2cd73ad8073770b27c5ed4938e915baf256f3
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:49:42 2014 +0200

    Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=833
```

Meanwhile, you've started running some simulations on a local cluster, and you're investigating the effect of changing some parameters:

```
(cluster)$ git diff
diff -r 416ac8894202 CUBA.py
--- a/CUBA.py       Thu Jul 12 14:28:19 2012 +0200
+++ b/CUBA.py       Thu Jul 12 15:19:49 2012 +0200
```

```
@@ -25,9 +25,9 @@
 import time

 start_time=time.time()
-taum=20*ms
-taue=5*ms
-taui=10*ms
+taum=15*ms
+taue=3*ms
+taui=5*ms
 Vt=-50*mV
 Vr=-60*mV
 El=-49*mV
```

```
(cluster)$ git add CUBA.py
```

```
(cluster)$ git commit -m 'Changed time constants in CUBA model'
```

```
(cluster)$ git log
commit 5a13e7c0ad7a21a22e91d2359767a5429f0b0c54
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 16:14:01 2014 +0200

    Changed time constants in CUBA model

commit b535fafe86c0caeb2b38ef30f0e96c9904ff56d6
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:47:45 2014 +0200

    Save figure to file

commit d66ee50a3975d6e46be27055bf0ecb4d8091c082
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:36:45 2014 +0200

    Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=833
```

Now the repositories on the two machines are out of sync. The first two commits are the same on both, but the third is different on the two machines. Note that the first two have the same hexadecimal version number on both machines, but that the third has a different hex number:

Laptop	Cluster
0:d66ee50	0:d66ee50
1:b535faf	1:b535faf
2:b1092b1	2:5a13e7c

So, how do we get the two machines in sync? This can be done from either machine. Here, we'll do it from the laptop.

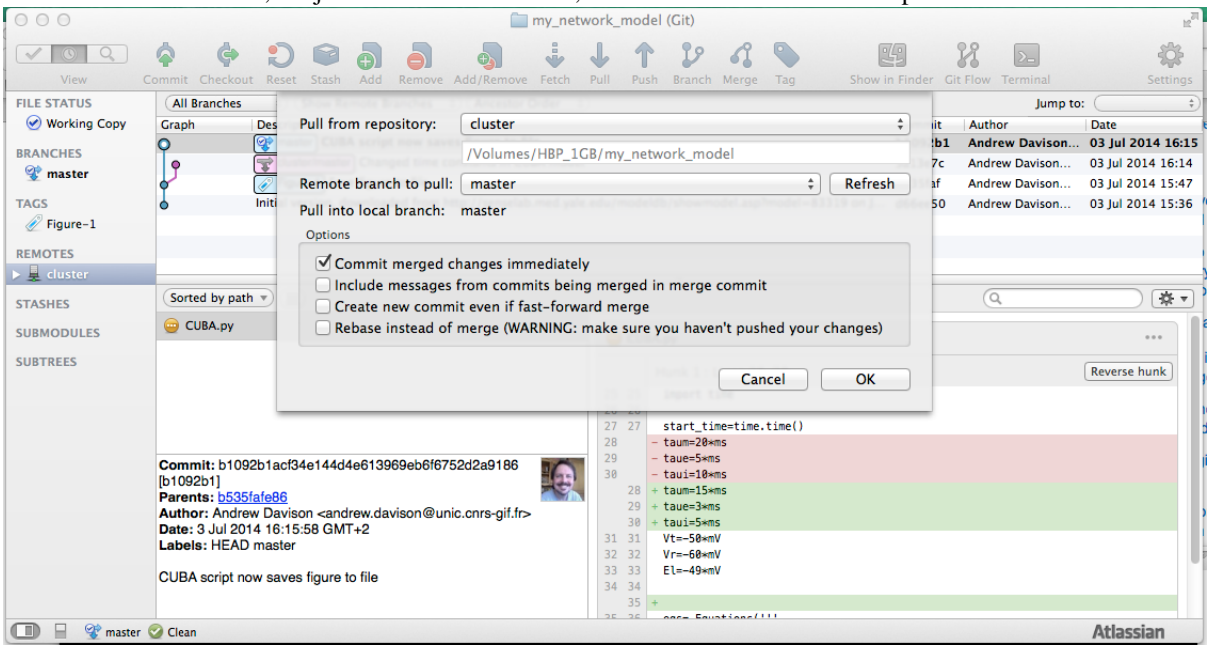
```
(laptop)$ git pull ssh://cluster.example.edu/my_network_model
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /Volumes/HBP_1GB/my_network_model2
 * branch            HEAD       -> FETCH_HEAD
Auto-merging CUBA.py
Merge made by the 'recursive' strategy.
 CUBA.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Note that **`git pull`** is equivalent to running **`git fetch`** followed by **`git merge`**. *"Fetch"* pulls changes into the local *repository*, but does not change the *working copy*, i.e. it does not change your files. *"Merge"* is the

part that changes your files.

Even though we made two different changes to the same file on different machines, Git is clever enough to realize that we'd edited different parts of the file `CUBA.py`, it can automatically merge the two changes. If there was a conflict (if we'd edited the same lines on both machines), the merge would fail and we'd have to manually merge the files (see below).

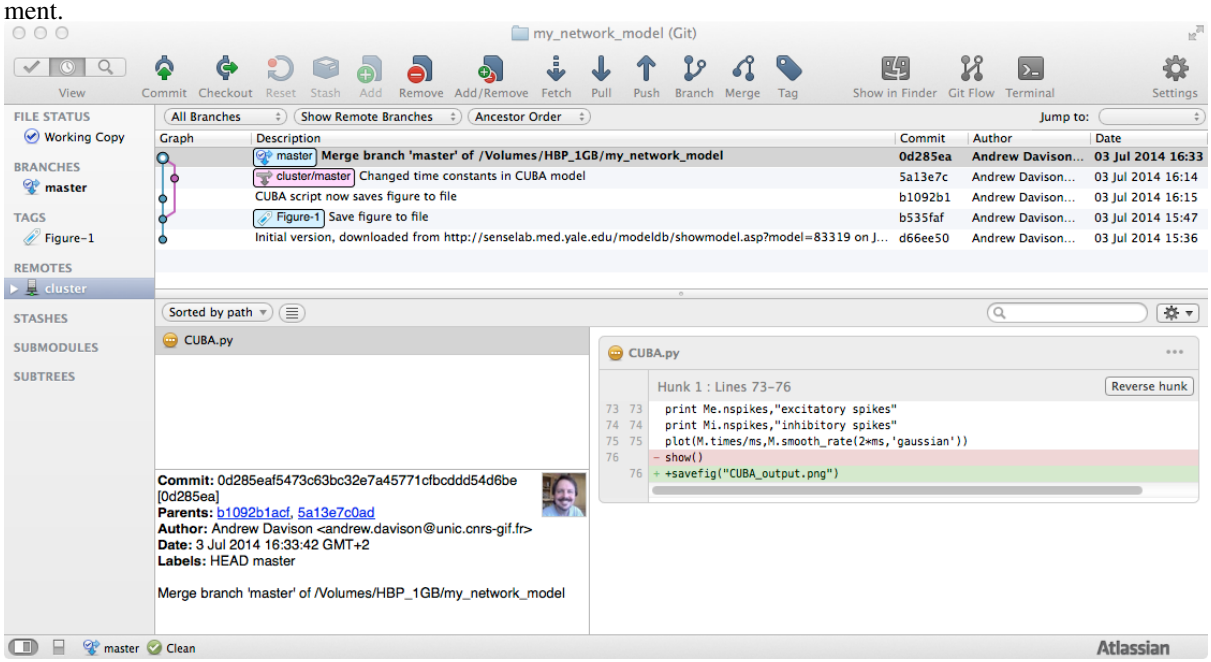
To do this in SourceTree, we just click the Pull button, then select where we want to pull from:



Now we can see the full history, with all changes:

```
(laptop)$ git log
commit 0d285eaf5473c63bc32e7a45771cfbcddd54d6be
Merge: b1092b1 5a13e7c
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 16:33:42 2014 +0200

    Merge branch 'master' of /Volumes/HBP_1GB/my_network_model

commit b1092b1acf34e144d4e613969eb6f6752d2a9186
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 16:15:58 2014 +0200

    CUBA script now saves figure to file

commit 5a13e7c0ad7a21a22e91d2359767a5429f0b0c54
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 16:14:01 2014 +0200

    Changed time constants in CUBA model

commit b535fafe86c0caeb2b38ef30f0e96c9904ff56d6
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:47:45 2014 +0200

    Save figure to file

commit d66ee50a3975d6e46be27055bf0ecb4d8091c082
Author: Andrew Davison <andrew.davison@unic.cnrs-gif.fr>
Date:   Thu Jul 3 15:36:45 2014 +0200

    Initial version, downloaded from http://senselab.med.yale.edu/modeldb/showmodel.asp?model=833
```

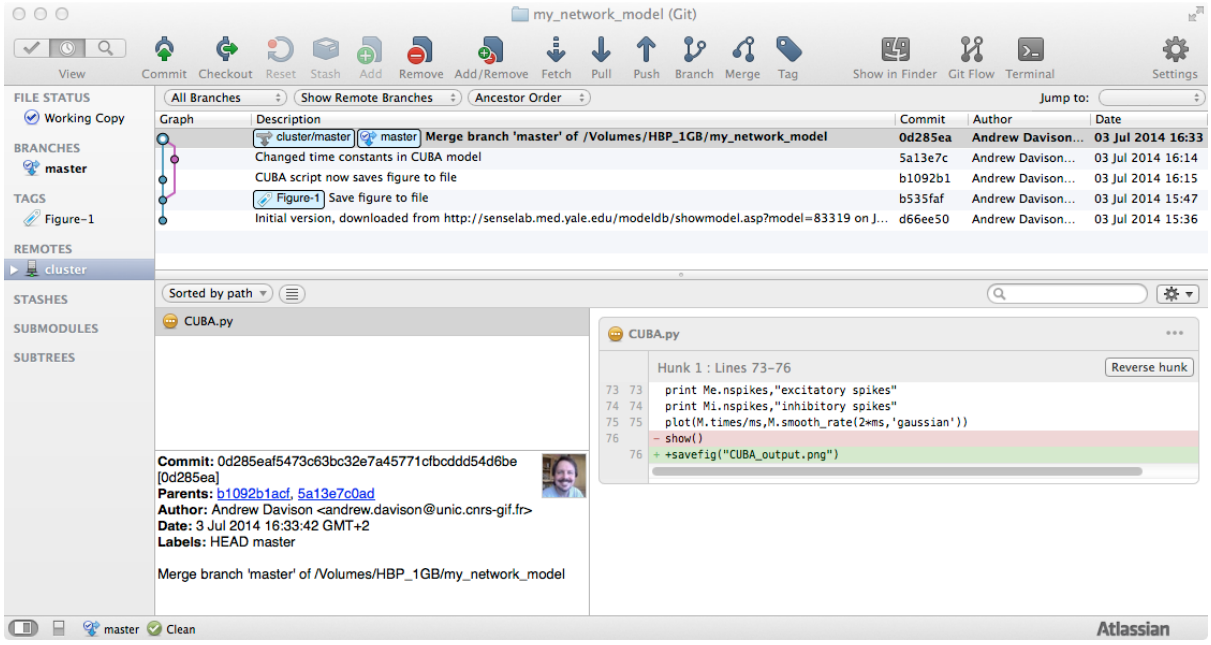
This is much clearer in SourceTree, which presents a graph with coloured lines, showing the process of development.



To complete the sync, we now pull the changes to the cluster:

```
(cluster)$ git pull ssh://my_laptop.example.edu/my_network_model
From ssh://my_laptop.example.edu/my_network_model
 * branch            HEAD       -> FETCH_HEAD
Updating b2351e1..b2405ed
Fast-forward
 CUBA.py | 7 ++++---
 1 file changed, 4 insertions(+), 3 deletions(-)
```

In SourceTree, notice that the label “cluster/master” is now at the same version as the local “master”:



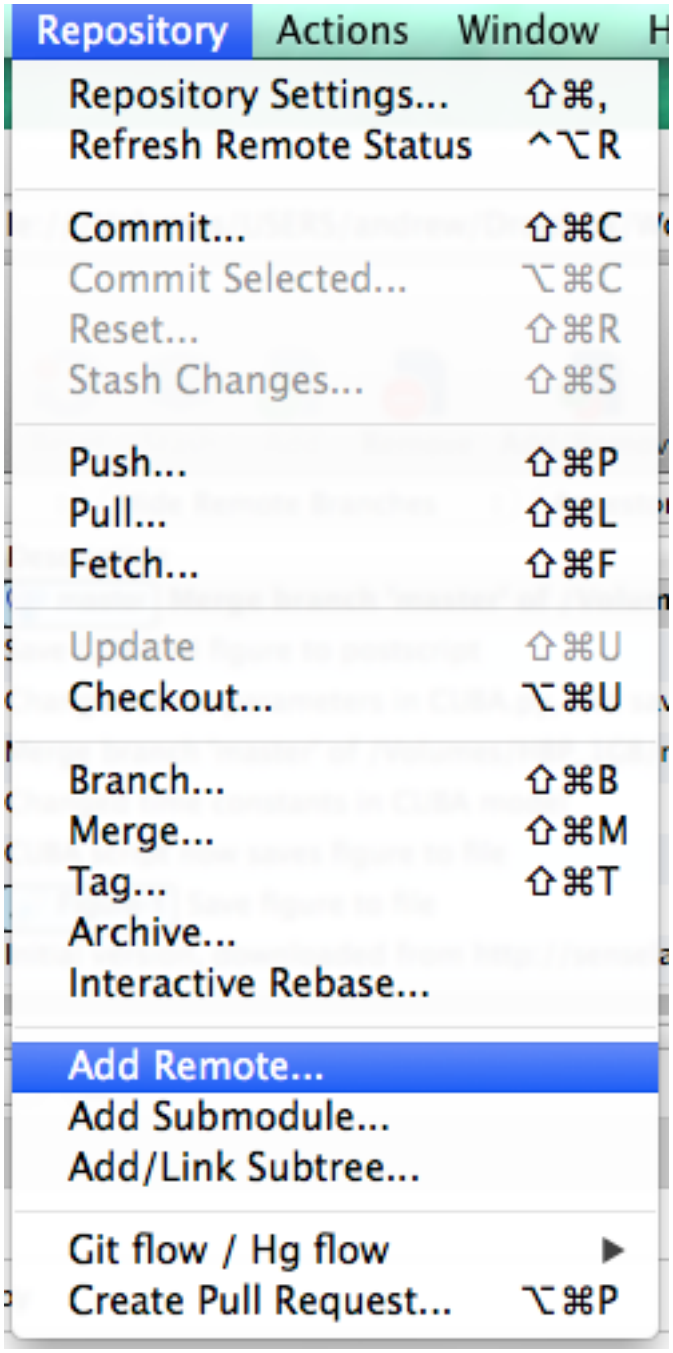
CHAPTER 5

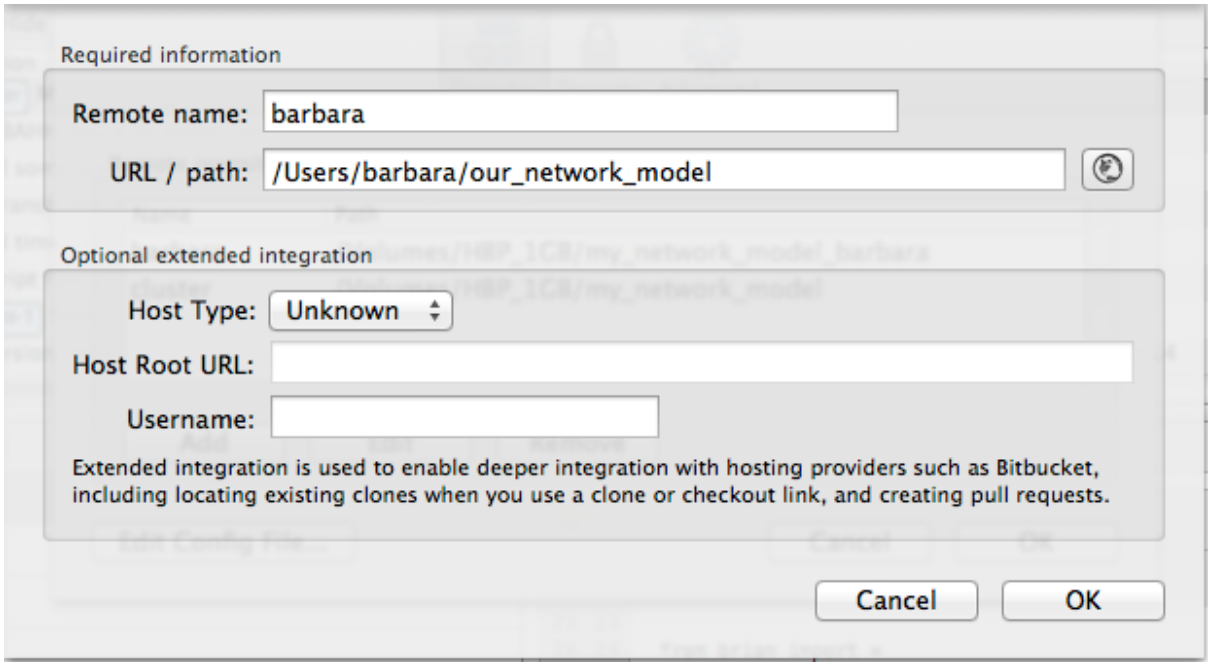
Collaborating with others

Using version control systems to collaborate with others is essentially no different to working solo on multiple machines, except that you perhaps have less knowledge of exactly what changes have been made by others.

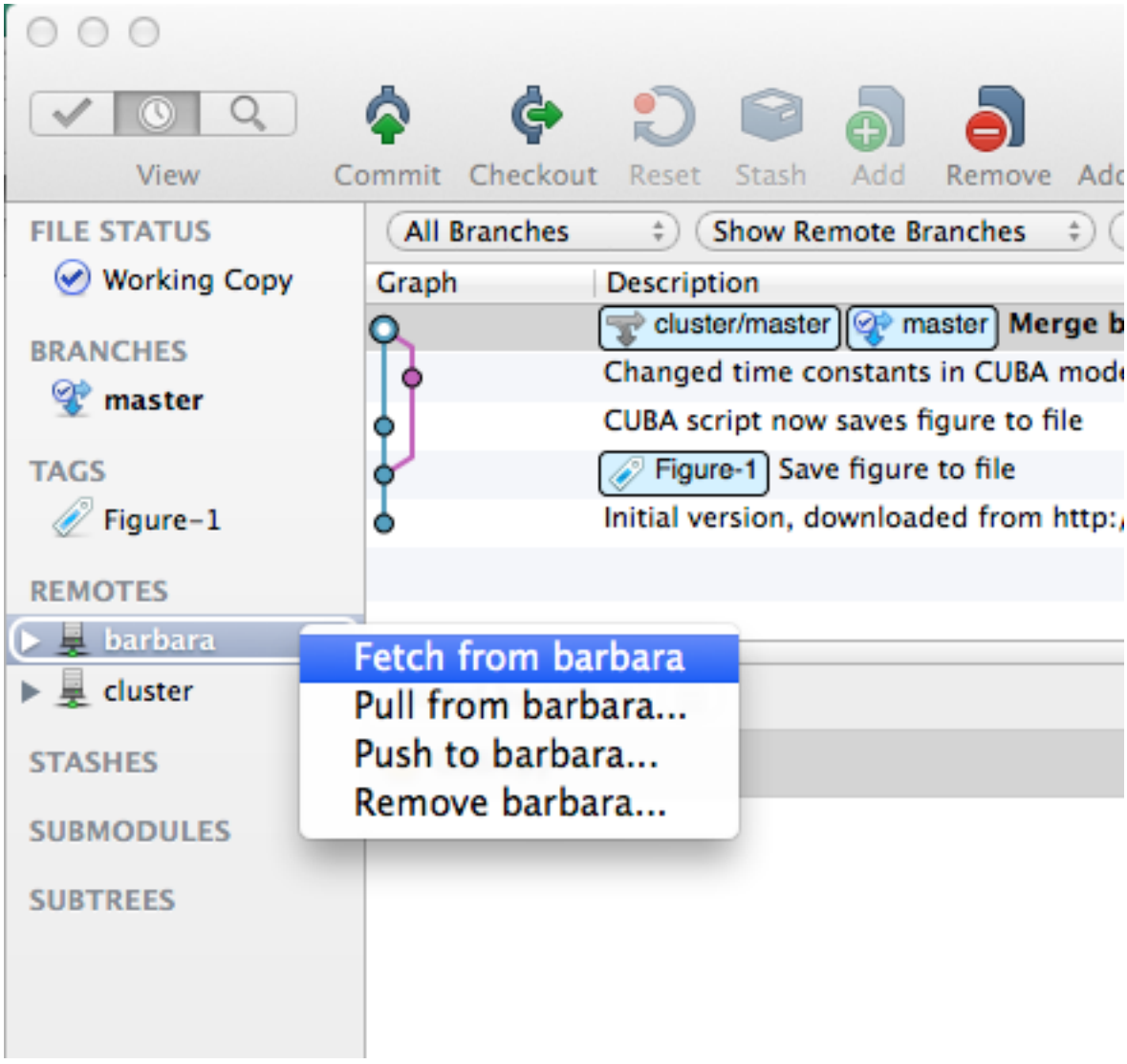
Suppose my colleague Barbara has also been working on the same code: she cloned my repository at version 0, and since then has been working independently. I'm a little wary of pulling in her changes, so first I can take a look at what she's changed:

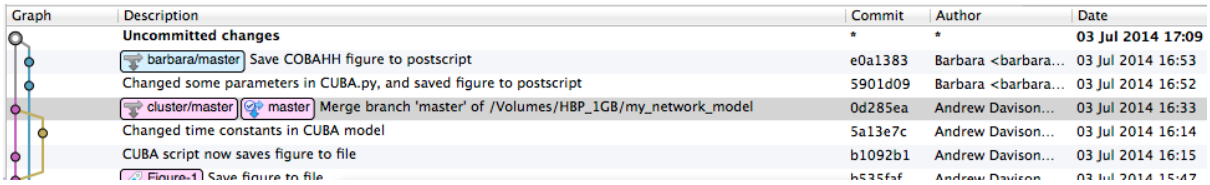
First I add Barbara's repository as a "remote":


Repository
Actions
Window
Repository Settings... ⇧⌘,
Refresh Remote Status ^⌥R
Commit... ⇧⌘C
Commit Selected... ⌥⌘C
Reset... ⇧⌘R
Stash Changes... ⇧⌘S
Push... ⇧⌘P
Pull... ⇧⌘L
Fetch... ⇧⌘F
Update ⇧⌘U
Checkout... ⌥⌘U
Branch... ⇧⌘B
Merge... ⇧⌘M
Tag... ⇧⌘T
Archive...
Interactive Rebase...
Add Remote...
Add Submodule...
Add/Link Subtree...
Git flow / Hg flow
Create Pull Request... ⌥⌘P



Then I can fetch her changes to look at them, *without* merging them into my code.

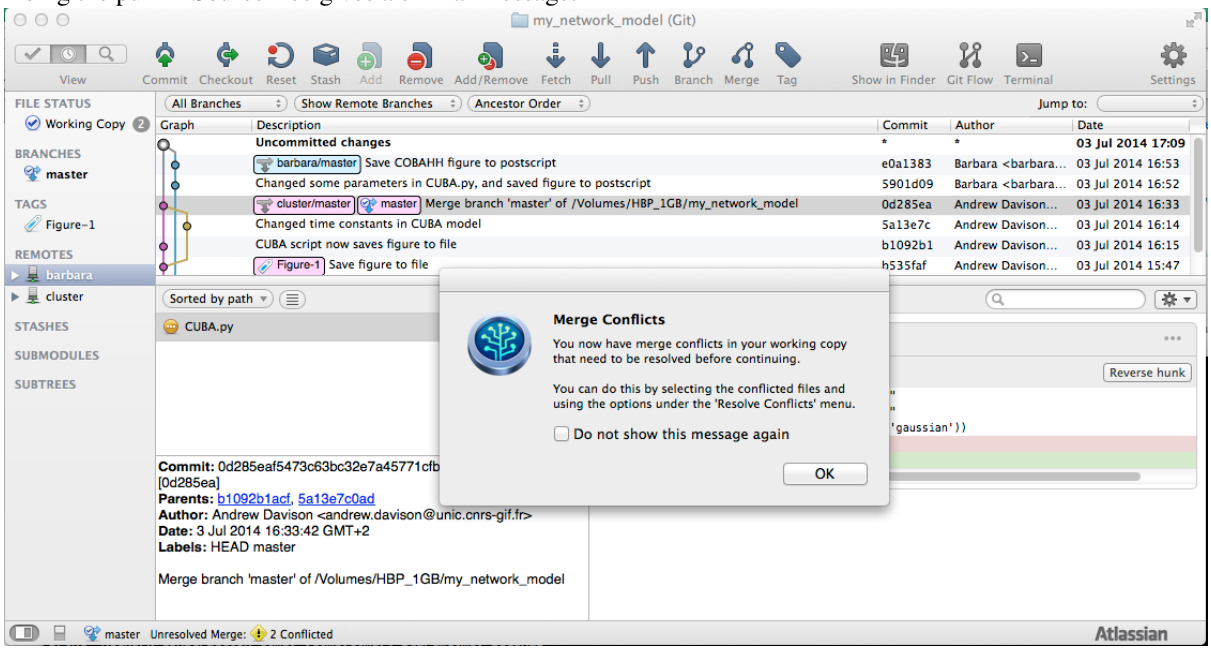




Looks like there may be some problems, since I've also changed parameters in CUBA.py, and I'm saving figures to PNG format, not postscript. Oh, well, deep breath, let's plunge in:

```
$ git pull /Users/barbara/our_network_model
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 3), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From /Users/barbara/our_network_model
 * branch            HEAD       -> FETCH_HEAD
Auto-merging CUBA.py
CONFLICT (content): Merge conflict in CUBA.py
Auto-merging COBAHH.py
CONFLICT (content): Merge conflict in COBAHH.py
Automatic merge failed; fix conflicts and then commit the result.
```

Doing the pull in SourceTree gives a similar message:



Unlike last time, when our changes were in different parts of the file, and so could be merged automatically, here Barbara has changed some of the same lines as me, and Git can't choose which changes to keep.

5.1 Dealing with conflicting changes

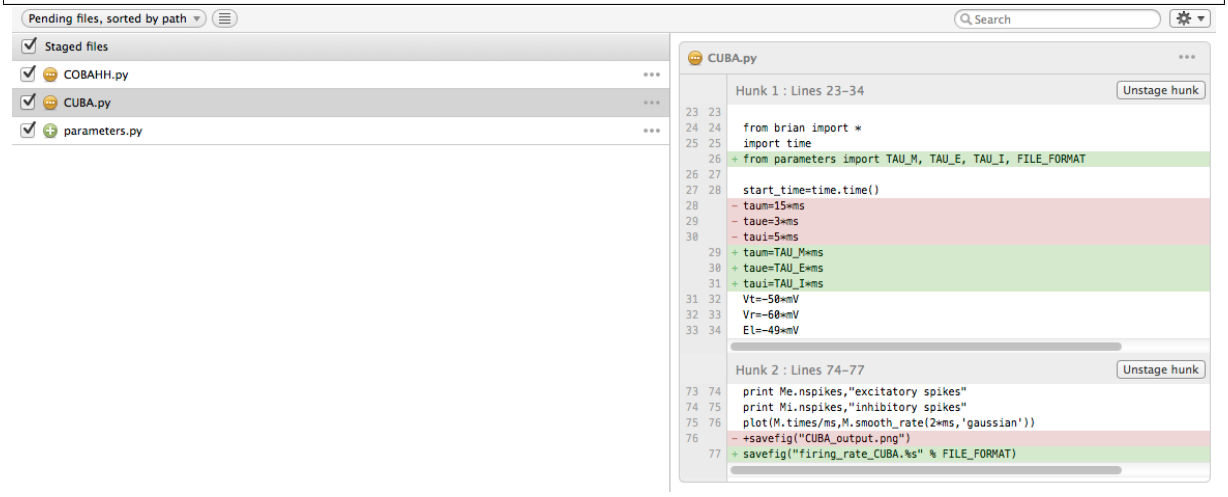
If we now look at `CUBA.py`, we can see the conflicts marked with `<<<<<<<` and `>>>>>>>`:

```
...
from brian import *
import time

start_time=time.time()
<<<<<<< HEAD
taum=15*ms
taue=3*ms
```

```
taui=5*ms
=======
taum=25*ms
taue=5*ms
taui=10*ms
>>>>>>> 58a6d1659c88502e66e2aa27396a92949be24172
Vt=-50*mV
Vr=-65*mV
El=-49*mV

...

<<<<<<< HEAD
+savefig("CUBA_output.png")
=======
savefig("firing_rate_CUBA.eps")
>>>>>>> 58a6d1659c88502e66e2aa27396a92949be24172
```

Well, it makes sense for both me and Barbara to explore different parameters, and it makes sense to allow different file formats, so let's move the parameters into a separate file, and parameterize the file format. The file now looks like this:

```
...
from brian import *
import time
from parameters import TAU_M, TAU_E, TAU_I, FILE_FORMAT

start_time=time.time()
taum = TAU_M*ms
taue = TAU_E*ms
taui = TAU_I*ms
Vt=-50*mV
Vr=-65*mV
El=-49*mV

...

assert FILE_FORMAT in ('eps', 'png', 'jpg')
savefig("firing_rate_CUBA.%s" % FILE_FORMAT)
```

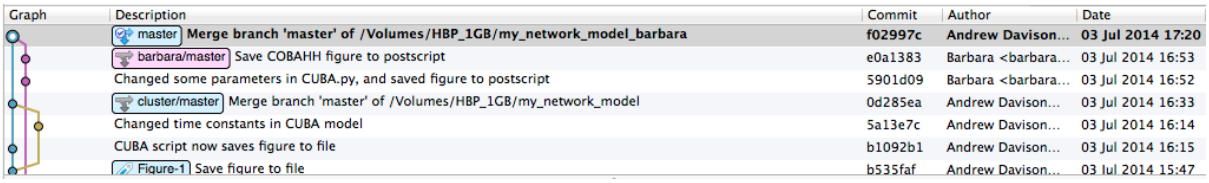


After manually editing `COBAHH.py` as well, I can now do a commit:

```
$ git add COBAHH.py CUBA.py parameters.py
```

```
$ git commit -m "Merged Barbara's changes; moved parameters to separate file"
```

SourceTree shows us how the graph has changed.

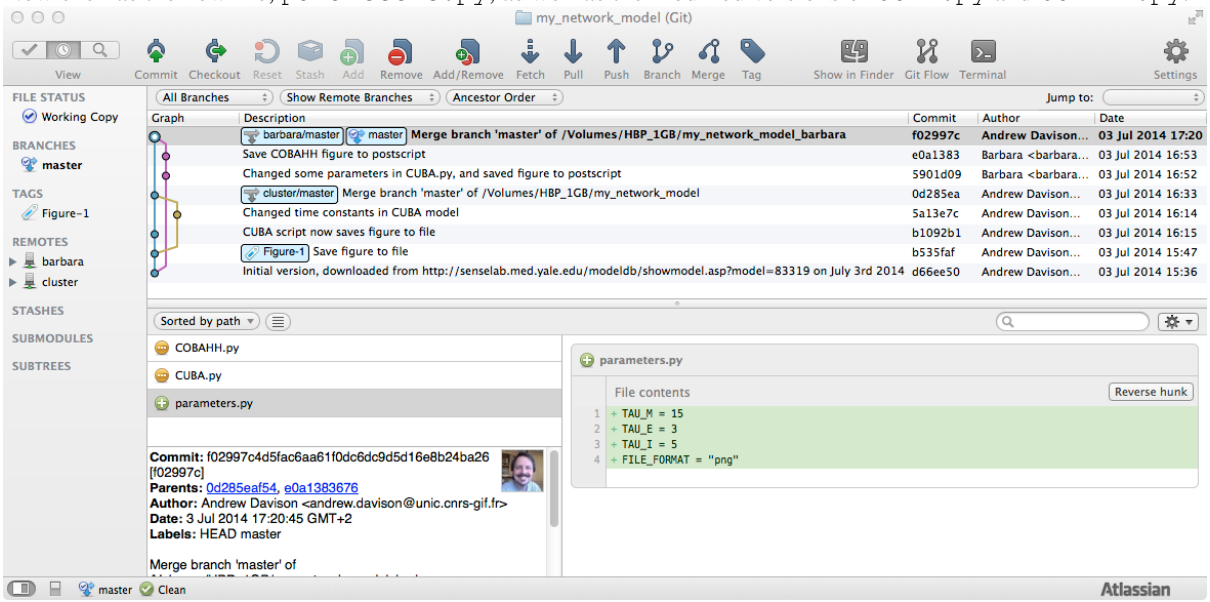


Note: I decided to add the new `parameters.py` to the repository. This means Barbara and I will still have conflicts in future if we're using different parameters, but at least the conflicts will be localized to this one file. It might have been better not to have `parameters.py` under version control, since it changes so often, but then we need another mechanism, in addition to version control, to keep track of our parameters.

I send Barbara an e-mail to tell her what I've done. Now all she has to do is run **`git pull`**.

```
(barbara)$ cd ~/our_network_model
(barbara)$ git pull /Volumes/USERS/andrew/my_network_model
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
From /Volumes/USERS/andrew/my_network_model
   0d285ea..f02997c  master     -> origin/master
Updating e0a1383..f02997c
Fast-forward
 COBAHH.py      |  4 ++--
 CUBA.py        | 10 ++++++----
 parameters.py |  4 ++++
 3 files changed, 12 insertions(+), 6 deletions(-)
 create mode 100644 parameters.py
```

Now she has the new file, `parameters.py`, as well as the modified versions of `CUBA.py` and `COBAHH.py`.



5.2 Recap #2

You should now be able to use Git for:

- quick and easy backups of your code
- keeping your work in sync between multiple computers

- collaborating with colleagues

CHAPTER 6

Working with branches

Todo

section on branching

CHAPTER 7

Licence

CHAPTER 8

Sources

https://bitbucket.org/apdavison/git-for-neuroscientists/ - feel free to fork the repository!