1. Installing and Initializing git

After you've installed and built a fresh new copy of Sage, you need to install and initialize git. This is the software that allows your local copy of sage to communicate with the global sage project through Sagetrac.

(1) Download and install the appropriate version of git

`http://www.sagemath.org/doc/developer/git_setup.html`.

In fact you can get everything I'm about to say from that website, but this is just the bare-bones version.

(2) Once this is installed you need to configure your local git repository. Open a terminal window, and from the ∼ directory, type the following:

```
$ gitconfig --global user.name "Anna Haensch"
$ gitconfig --global user.email annahaensch@gmail.com
```

Except obviously replace my information with your own. You can check that this made its way to your configuration file ∼`/.gitconfig`.

(3) Now you need to link your local system to Sagetrac using the Trac username and password that you registered with. To do this, you need to introduce yourself to Sagetrac by sending your public ssh key along with your Trac login information. From where you are, enter `cd sage 6.2`, and then execute sage with `./sage`, now sage should be running, and you will enter the following.

```
sage:  dev.upload_ssh_key()
The trac git server requires your SSH public key to be able to identify you.
Upload "/Users/annahaensch/.ssh/id_rsa.pub" to trac?  [Yes/no] y
File not found:  "/Users/annahaensch/.ssh/id_rsa.pub"
Create new ssh key pair?  [Yes/no] y
Generating ssh key.
Trac username:  annahaensch
# Your trac username has been written to a configuration file for future
# sessions.  To reset your username, use "dev.trac.reset_username()".
Trac password:
You can save your password in a configuration file.  However, this file might be
readable by privileged users on this system.
Save password in file?  [yes/No] n
Your key has been uploaded.
```

(4) Now that you are initialized and communicating with git, you want to make sure that your files are all up to date with the main hub. From your ∼ directory, run the following.

```
$ git clone git://github.com/sagemath/sage.git
$ cd sage
$ make
```

This process could take about 15 minutes or several hours.

Now you should have your original sage version built, it should look like ∼`/sage-6.2` and an equivalent mirror path to sage, labeled ∼`/sage`. In the following sections, we will be working through the latter, but both are perfectly fine ways to execute sage and access the source files.

2. Sending Code to Trac

Now you are ready to start writing and uploading code, and reviewing tickets on Sagetrac. For this part, let's suppose you have a bunch of code ready to add to the library. We'll want to do everything form inside the sage clone that you built in section 1, so before starting move there with `cd sage`.

(1) Go to Sagetrac and start a new ticket, filling out the necessary information. Once you submit the ticket, it will be given a number. Let's suppose that number is #16379.

(2) From inside Sage you can created new branches. These are just what the sound like, new offshoots of your current install that you can muck around with, and not change anything downstream. To create a branch which is automatically associated with your ticket, run sage with `./sage` and then enter the git command

```
sage:  dev.checkout(16379)
```

This will create, and place you in, a local branch called `ticket/16379` which is uniquely associated to ticket #16379. To double check that you are in the right branch, follow the steps in section 3 item 1.

(3) Suppose you have some code that you've added into the library and you'd like to get your current copy of Sage to recognize it. You need to commit those changes to the library.

```
sage:  dev.commit()
Commit your changes to branch "ticket/16379"?  [Yes/no] y
# Use "dev.push()" to push your commits to the trac server once you are done.
```

This commit will redirect you to a screen where you write a two-line commit message, the instructions are self explanatory. When your message is written, hit `ctrl+X` and then `enter` to bring you back to the `sage:` prompt. At this point, your changes to the library are recognized by your local copy of sage, meaning that if you wrote a new function foo(), you can freely use foo() on your copy of Sage.

(4) It may take a few commits to feel like your code is in shape to send it to Sagetrac, but as soon as you are ready, you can push it all to Sagetrac from within Sage.

```
sage:  dev.push()
Local commits that are not on the remote branch "u/annahaensch/ticket/16379":
 471f157:  updated examples from last commit ←first line of commit message
Push to remote branch?  [Yes/no] y
```

Now you can go to the ticket on Sagetrac, and all of your changes should be up there.

3. Some Important Commands

Here's a basic breakdown of some frequently used commands, and what they can do for you.

(1) From ~/sage, you can always double-check which branch you are working in with

```
$ git branch
master
*ticket/16379
```

and your current branch is denoted with a *. To change to a new branch, in other words, to "check out" a different branch, use

```
$ git checkout master
Switched to branch 'master'
$ git branch
* master
 ticket/16379
```

In this way it's clear how to switch between branches.

(2) Suppose you are reviewing a patch for ticket 16397. Then you start by building a local branch, with

```
$ ./sage -dev checkout --ticket 16379
$ ./sage -b
```

Now you may have two local branches `master` and `ticket/16397`. To see the difference between them, use

```
$ git diff --color master...ticket/16379
```

This will give you a color-coded output which also identifies trailing white space. The three dots are important. Always remove all trailing whitespace before pushing code to Trac.

(3) Suppose you'd like to create a branch to play around in without actually associating it to a ticket, then from within ./sage, execute

```
dev.checkout(branch="test")
```

(Alternatively, from ~/sage you can execute $ `git checkout -b test`). If you want to play around in the .py file and change/add functions you can do that. To get your branch to recognize them, you need to first commit from ~/sage as in section 2 item (3),

```
$ ./sage -dev commit
```

and then rebuild you branch by quitting sage, and typing

```
$ ./sage -b
```

(4) If you are finished with a branch and want to delete it, you can switch back to the master branch, and use

```
$ git branch -D ticket/16379
```