Intro to GIT

Justin Erwin

June 30, 2015

"Git is a distributed revision control and source code management (SCM)", which means it keeps track of changes in text files, and can be used to integrate changes amongst different developers of that code. User can share one remote repository of the code (on a server), checkout individual version of the code (called branches), and then help merge the changes back together.

Installing

Git comes installed on the MAC already,

Installing Git

```
sudo port install git +svn +bash_completion +gitweb
```

I find it a good idea to leave MAC's Git (as well as gcc and python) alone, and to use my own installs for working. (Macports installs in the /opt/local folder and puts it in the PATH ahead of /usr where MAC puts there programs.)

config

We should first set some configuration variables in Git

Configuring Git

```
git config --global user.name "JT Erwin"
git config --global user.email jterwin@lpl.arizona.edu
git config --global core.editor emacs
git config --global merge.tool opendiff
```

The first two are pretty obvious (though you should change the name). The latter two set default applications to edit comments and handle merging, respectively (more on that later).

Starting with Git

- I will outline some basic GIT operations.
- GIT documentation `git-scm.com/book/en/Getting-Started`, which can go into more detail.
- Google is always your friend when asking a particular question.
- Also try out online courses (i.e. `www.codeschool.com/courses/try-git` or `githowto.com/setup`) for some practice.

We being in folder with some code we would like to version

Create repository

```
git init
```

and we add some files to the repository

Adding file

```
git add Makefile main.f90 venus_heating.f90
git status
```

The status will show us any changes since the last commit.

I add some files to this repository, and then add them to the Git repository as

Adding file

```
git add Makefile main.f90 venus_heating.f90
git status
```

Then we commit these changes to the local repository

Committing

```
git commit -am "initial project setup"
git status
```

The two options in the commit are: "a" commit all changes, "m" the message string follows. If you want a longer message, leave out the "m", and the emacs (or the editor that you set in the configuration, vi is default) will pop up and you can edit the message from there.

- Use git mv to move files
- Use git rm to delete files from repository
- git log will display a list of commits with the message
- git checkout file can be used to revert a file to any previous commit
- git tag can be used to assign a sensible name to a commit (i.e. v1.0, paper2014)

- Get use to tree metaphors
- Branches are separate versions of the code we can make but leave a version of the code (the master branch) alone.
- Used for experimenting with changes/updates
- Used for separate development (multi users), with master being the common working copy.
- Changes from branches can be merged into one another!!!

The checkout option is used to switch between different branches.

creating new branch

```
git checkout -b new_branch
git branch --list
```

The -b creates a new branch called *new_branch*, and switches to it. The last command will display all (local) branches.

The checkout option is used to switch between different branches.

creating new branch

```
git commit -am "changed some stuff in new branch"
git status
git checkout master
git status
```

- When switching branches, "Local modifications to the files in the working tree are kept, so that they can be committed to the [branch]". So you should commit before switching branches (checkout) or risk confusion.
- You can have many branches locally and remotely, and remote branches can be checkout out to local one (clarify!!!).

Discarding changes

Lets say you made some changes, but it turned out to be a dead end.

Merging branches

```
git checkout main.f90
```

The file "main.f90" will be reverted to the last commit of the current branch, and it is removed from the staging area.

Merging

After you have finished your changes (and tested them), you will want to merge your code back into the master for others to have in their code.

Merging branches

```
git checkout master
git merge new_branch
```

In the above, we switch to the master branch, merge the branch into the master, and then push the changes to the server for other to use.

Use git diff to view the changes in our files before we commit.

diff-ing

git diff [options] *filename*
git diff [options]

The first will look for changes in just one file, while the second will look for all changes to the files in the git repository. The following slides show *diff* can be used to see changes in the current branch, as well as comparing to past commits or other branches.

Various ways to check your working tree

```
$ git diff              (1)
$ git diff --cached     (2)
$ git diff HEAD         (3)
```

1. Changes in the working tree not yet staged for the next commit.
2. Changes between the index and your last commit; what you would be committing if you run "git commit" without "-a" option.
3. Changes in the working tree since your last commit; what you would be committing if you run "git commit -a"

Comparing with arbitrary commits

```
$ git diff test              (1)
$ git diff HEAD -- ./test    (2)
$ git diff HEAD^ HEAD        (3)
```

1. Instead of using the tip of the current branch, compare with the tip of "test" branch.
2. Instead of comparing with the tip of "test" branch, compare with the tip of the current branch, but limit the comparison to the file "test".
3. Compare the version before the last commit and the last commit.

Comparing branches

```
$ git diff topic master    (1)
$ git diff topic..master   (2)
$ git diff topic...master  (3)
```

1. Changes between the tips of the topic and the master branches.
2. Same as above.
3. Changes that occurred on the master branch since when the topic branch was started off it.

example

```
git diff master..multi_iso hitran.f90
```

This call shows the changes in the file *hitran.f90* from the *master* branch to the *multi_iso* branch.

- So far, the GIT repository has been located on our computer.
- We can located a repository on server (i.e. HIPAS) so that many machines and users can access it.
  - a shared space on data has been made so that anyone in our group can access the code.
- Github is a freemium service to host our repositories, with a GUI to look at the log and manage users/permissions.

- Now I will discuss how I will use git on HIPAS.
- My workflow is to keep the main repository for my code (each code has its own repository) on HIPAS, then I can checkout this repository onto my own machine to edit the code. Then I can push changes back to the server for others to see.
- I can also checkout the code into another folder on HIPAS to run the code on HIPAS, or checkout the code onto HPC to run my code there. This is a nice way to make sure that I am running the most recent code on these different machines.

In my home folder on HIPAS I create a folder *projects*, and then a folder for my code called *venus1.git*:

Creating a remote Git repository

```
ssh jterwin@HIPAS.lpl.arizona.edu
mkdir projects
cd project
mkdir venus1.git
cd venus1.git
git init --bare
```

The "--bare" option is here because we intend to share this repository and only to modify it remotely using "git clone/push/pull/" (more on that soon), and do not intend to modify or run code in this folder. We can checkout the code to a different folder on this machine, and compile and run it there.

On my laptop (or the cluster), I get access to this remote repository

Accessing repository on laptop

```
cd  /Documents/work
git clone jterwin@HIPAS.lpl.arizona.edu:projects/venus1
cd venus1
git status
```

This will copy whatever is in the Git repository on the server (especially all the meta-information) to a directory on my laptop is named *venus1* (without the .git). The *status* will show me whats changed from the last commit (in this case nothing). More detailed information can be displayed by running "git config --list".

As your modifying your code, you can commit every so often with a message of what you have changed. When you commit, you commit to the local repository on your computer. To make these changes available to others we need to push these changes to the server.

Pushing to server

```
git commit -am "changed some stuff"
git push origin master
```

In the above, "origin" lets Git know to send the changes to the server (set in the initial clone, visible with git config --list). "master" is the name of the branch we wish to commit to (more on this later).

To get the newest version of the code, we the pull that information on the server

Pulling server

```
git pull
```

is almost equivalent to

Pulling server

```
git fetch
git merge
```

Basically, *pull* will update the branch info from the server (a *fetch*) and do an automated *merge* without given you info on the merge. This may be dangerous with large project as conflicts can occur. A safer route is to use separate branches for each person or code upgrade, and then do a manual merge where you can individually resolve conflicts.

So, we created a shared folder on HIPAS. It is located at */cdata/ygit*. This is different from the *cdata* folder in you home directory (*~/cdata*) which points to /cdata/yel/jterwin. As I already use *~/projects* as a location for my Git repositories, I move my project to the share folder and then create a symbolic link to the new location in *~/projects*

Sharing Git repository

```
ssh jterwin@HIPAS.lpl.arizona.edu
cd projects
mv venus1.git /cdata/ygit/venus1.git
ln -s /cdata/ygit/venus1.git venus1.git
```

And then on my laptop I can clone this remote Git repository like before.

THE UNIVERSITY
OF ARIZONA