Git Basics for LaTeX Users

Ben O. Smith*
School of Economic Sciences
Washington State University

What's Version Control? Why am I doing this?

When you write a document, you have a few basic steps: you start the document and save, revise the document and save then finally publish the document. But, you repeat the second step many times.

Here's the problem, as you revise the document, your previous versions are lost (or at a minimum difficult to access). So, what do you do? A lot of people start doing "ad-hoc" version control. You might create duplicates of the file at certain points, or put a previous version of a paragraph in a comment or depend on in-application versioning. None of these are particularly satisfactory and tend to make more of mess than anything else.

As messy as "ad-hoc" versioning is with one person, it is far worse when you have multiple authors. Consider a paper with three authors (figure 1). Some authors might make a large number of changes in a short period of time, others take a long period of time to do less edits. And these edit periods overlap. If you were doing this with Word and E-Mail, this would result in multiple versions of the document that then would have to be hand merged. Not so with LaTeX and Git.

Figure 1: Some edits take different periods of time and the edits overlap

*tazz_ben@ad.wsu.edu

1 Git Concepts

Git is distributed version control system. It's "distributed" because a complete copy of the version history resides on your computer and every other contributors' computer in a hidden directory. This means you can work on your paper online or offline and Git is really fast.

I'm going to walk you though using Git with the command line. You will probably decide to use a GUI client (link in section 3.1), but if you understand the concepts of what is going on, using the GUI tool will be much easier.

I have a directory called "myproject" that is empty, I want to write a paper here and use Git for version control. First I need to initiate the repository:

```
> git init
```

Initialized empty Git repository in /Users/tazz/myproject/.git/

Great, now I can work on my file just like I normally would. I'm creating a basic LaTeX document for another handout. I've created the initial version and I want to commit it to the repository.

```
> git add basic.tex
```

The changes (which in this case is the entire file) of basic.tex are now "staged". The staging area is where you select which set of changes you want to group together as a single commit. If I type:

```
> git status
```

I get the following information:

```
# On branch master
#
# Initial commit
#
```

```
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
# new file:   basic.tex
#
```

Ok, but I have some other work I want to do before I commit. I need to create a readme file in the same directory. I've created the file, so I'm now going to stage it as well.

```
> git add README.md
```

I should also note that if you want to stage all changes in a directory you can type the following:

```
> git add .
```

Ok, now I'm ready to commit:

```
> git commit -m "Initial version of LaTeX document and Readme"
```

Git returns:

```
[master (root-commit) 1c583ce] Initial version of LaTeX document and Readme
 2 files changed, 13 insertions(+)
 create mode 100644 README.md
 create mode 100644 basic.tex
```

Now, what's really great about this is that now all of that work I did is safe. I can delete both of those files in my working directory and I'd still be able to retrieve them.

So, the basic concept is that you work on files in your working directory, stage the changes and finally commit those changes to the repository. Visually, it might look like figure 2:

Suppose I've made a change to "basic.tex" I want to commit.

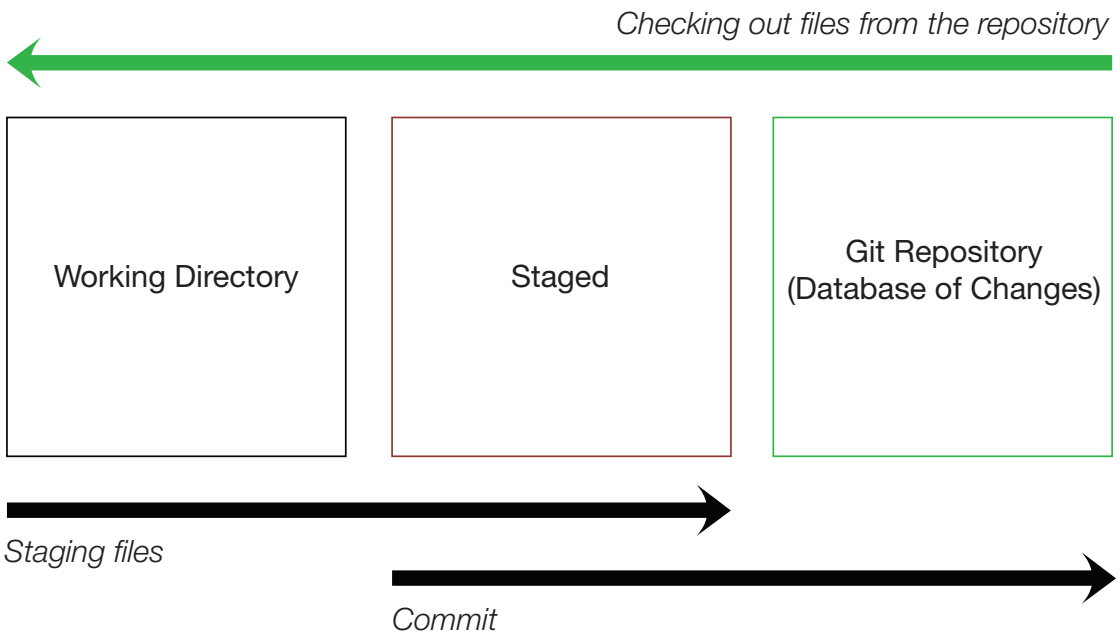

Figure 2: You first stage a set of files, then once you are done, you commit

```
> git add basic.tex
> git commit -m "Added fullpage to basic.tex document"
```

Git returns:

```
[master dd367b7] Added fullpage to basic.tex document
 1 file changed, 1 insertion(+)
```

You can see the history of commits by typing:

```
> git log
```

Git returns:

```
commit dd367b7cfd99c3a865d2e655938a8fa8c5e72fd6
Author: Ben Smith
Date:   Mon Oct 1 14:27:40 2012 -0700
```

```
    Added fullpage to basic.tex document


commit 1c583cecf971e0ef0bc155ed153275c3307e365e
Author: Ben Smith
Date:   Mon Oct 1 14:21:34 2012 -0700


    Initial version of LaTeX document and Readme
```

2 Git Remotes

But Git can also work with remote servers ("remotes"). As an example, let's clone the EconScripts repository off of GitHub.

```
> git clone git://github.com/tazzben/EconScripts.git
```

Now, on my computer, I have the full EconScripts repository. I can make changes to the files, stage the files and commit the files **exactly** as I did in section 1. However, there is one difference, once I'm ready for others to see those changes, I can "push" them to the remote.

```
> git fetch
> git push
```

If I want to pull in the changes others have made, I simply type:

```
> git fetch
> git pull
```

But, what about figure 1 with the multiple authors? What if multiple people are working on the same file at once? It isn't a problem. Git commits *changes* so you and someone else can work on the same file, make additions to out-of-date files and when it gets pushed up to the remote they will be reconciled.

3 Git Resources

3.1 Git Packages

- Downloads[1]
- GUI Clients[2]

3.2 References

- Pro Git[3]

[1] http://git-scm.com/downloads
[2] http://git-scm.com/downloads/guis
[3] http://git-scm.com/book