Git Overview

Scott Jackson

Last updated: March 25, 2013

1 What is git?

Git is a type of Version Control Software (VCS). It is open-source, cross-platform, and free, meaning you can download it at no cost for Windows, Mac, or Linux from the website at http://git-scm.com/, but you can also download the source code, and see exactly how it ticks (and modify it for your own use, or contribute changes that might get incorporated in future versions, etc.), if you're interested.

2 Why version control for research?

2.1 Great for programmers, great for researchers

The current notion of version control mainly revolves around the needs of software developers. Making software can be complicated business, especially when there are multiple people working on a piece of software. The term "version" refers to the fact that software is released with a version number or label (like Windows 7 or Microsoft Word 2010 or git 1.8.1.3). Even if people continue to work on developing Word or git or Windows or Emacs or whatever, versions represent some kind of stable point that works a certain way. It may not be perfect, so people work on fixing bugs, and then release a new version that includes those bug fixes. Maybe someone adds a new feature or improves an existing one. New version. And so on.

What VCS software does is help manage all these changes, even when there are many different people making the changes. It allows developers to track the changes, so they can undo things if some new change is introduced, but then causes other problems or otherwise doesn't work out. It allows people to work on parallel versions without disrupting the "main" branch of development. Good VCS software enables more complex operations as well, but the basic idea is that it gives you a master Undo and Track Changes button.

So why would a researcher want to use a VCS, if they're not developing software? Read on...

2.2 Universal "Undo" button

Have you ever used Undo in a program? Maybe you accidentally deleted a big section of text, or messed up some formatting. Hitting Undo gets you back to a

previous state, and it can be a real life-saver. But in many programs, this only goes back so far, and it's kind of limited. In most programs, if you save your work, close the program, and maybe even send the document via email to yourself on another computer, you lose the ability to Undo anything you did previously. What if you deleted something important, and only realize the next day what happened? Also, Undos are usually not labelled. You just hit Undo until you get back to where you think you'd like to be, but you don't have the ability to make notes about particular states to help find these earlier points. You also can't selectively Undo something. If you accidentally delete something, but then write a new section, and only realize the deletion later, it can be a pain, or outright impossible, to both keep your new work and recover the old section.

One thing that a VCS provides is a kind of universal Undo button with all of the functionality mentioned above, and more. The basic workflow is that after you do some work on your files, you use the VCS to "commit" the changes to the project history. This usually involves making a few notes (as simple or as detailed as you like) and entering a command in the VCS. This creates a point (a "version", though you don't have to number them, necessarily) that you can always go back to later. So if you realize that there was something in that older version that you need, or you just need to reconstruct what happened, you can "rewind" to that older version. That older version includes ALL the files in your project, not just a single file. For example, imagine that you have a "version" that represents the draft you send to your advisor to look at. It takes your advisor a while to get back to you, and in the meantime you've changed some parts of the analysis, or added some data or something. Then when he gives you comments on some of the older, modified sections, you can easily "rewind" to see the exact version of the paper that he saw, including any of the other additional files, like stimuli, sounds, figures, whatever. It becomes less like an Undo button and more like a time machine!

Furthermore, VCS's typically have good facilities to merge and branch the histories. So you can make a "branch" of your project to try out a different way of doing things, and easily switch back and forth between the parallel versions. Or if you realize that you need to recover something from something several versions ago, you can go back to recover just the change you want, without having to undo all the other work you've done since then.

2.3 Universal "Track Changes"

One of the more useful features of Microsoft Word is the "track changes" feature when you are collaborating with other people. The whole idea of the VCS is to track your changes to your entire research project in parallel, not just for a single document. There are plenty of tools, both within the VCS and in editors like Emacs that enable you to go through two different versions, comparing the differences, and allowing you to keep the ones you want and reject others for the next version. But unlike Word, all these changes are tracked, and so everything is recoverable. For example, maybe a co-author makes a large change to a section using track changes, and maybe as first author you disagree, so you reject those changes. But then later you become convinced that they were the right thing to do. How can you recover

those changes? Maybe your co-author kept a copy somewhere, or you go try to recover a version from an email, or something like that. A good VCS will let you easily recover that earlier change. In other words, *every* change is tracked, not just the ones you make while the special "track changes" button is on.

2.4 Reproducibility and efficiency

By now, I hope you are starting to see how this could be useful for your research, even if you don't program a line of code. The key here is that a VCS provides a foundation for reproducible research by helping you keep track of *exactly* what you did to get to where you are, and by helping you to not repeat effort. That great introduction you wrote a month a go, but then later deleted? You can easily recover it. That alternative analysis that didn't quite work out? You can keep it safely in the history of the project and return to it to remind yourself why it didn't work, if you need to re-convince yourself or a co-author or a reviewer. But because it's in the *history*, and not some other version of a file that's floating around in your folder system, you won't ever accidentally grab it when you meant to grab the current version. Yes, it's possible to construct and maintain a system for keeping track of a lot of this kind of information using filenaming conventions, separate folders for different versions, etc., but this is not an efficient system, and it's too easy to make mistakes, especially when collaborating. It's like saying, "why would I want to use a computer to typeset my manuscript, when I could do it myself with the simplicity of movable type?" I won't stop you, and if it works, fantastic. I'm just here to tell you that there's another option.

3 Why start with a VCS?

I highly recommend that if you are interested in making your research more reproducible, you start with the VCS called **git**. Here's why:

- Learning the basics of git for everyday use is very simple.
- You don't have to change *anything else* about your current workflow to use git. You can continue writing documents in Word, doing analysis in SPSS, or whatever. Git does not make you change everything about your work habits.
- Using git *does* force you to make a few simple changes to your work habits, but these are good habits to have, and are habits that will greatly improve the reproducibility of your work. They are habits that will take practice, and git gives you the opportunity to practice, without penalizing you if you slip up now and then.
- If you're not used to command-line programs, git gives you some good practice at that in a very simple context.

In short, starting to use a VCS like git is the easiest way to *immediately* start making your research more reproducible.

4 Why git?

Git is not the only VCS, not by a long shot. There are many, many other alternatives out there, and while some are expensive proprietary software packages aimed at large companies, some good options are also free and open-source, like Subversion or Mercurial. I am still a relative novice to git and VCS myself, and I have *not* done anything like a comprehensive review of VCS's. But here's my take on why I (and other people, who know more than I do) recommend using git, and not some other software package.

1. Git is *fast.* Some VCS's take some time to perform even the standard operations. One of the big advantages of git compared to many other systems is that it is very very fast. I believe this is critical to practical use, especially for researchers, because it means there is relatively little "overhead" to using the system. If the VCS slows you down, then you will not be encouraged to use it. If you get a stroke of inspiration and want to start a branch, the last thing you want is to have to sit around waiting for your VCS to do its thing before you can start working on it.

2. Git is *local.* One of the main reasons that git is fast is that it does not require talking to a server. Some VCS's require you to store a "master" version on a server somewhere, and when you want to update the history with some new work, you have to connect to the server to upload your changes. There are many remote hosting options for git (more on that in another tutorial), but normally all the changes are encapsulated on your own machine, and you can use it for your personal uses without *ever* having to access the internet (once you download and install git for the first time, of course). This has a lot of practical advantages, one being that you can work anywhere and use every function of git, even when you don't have an internet connection.

3. Git is *flexible.* Git differs from lots of other VCS's in its basic structure. I won't go into that now, but the bottom line is that one result of this structure is that it allows you to pick and choose how to use it to best fit your needs. Some people work on large, complex software projects where they have many many collaborators, but they need a way to keep tabs on changes, and not just allow every person to make whatever changes they want. Software developers have come up with many different models for how to deal with these kinds of interactions, and different companies and communities do things in different ways. Likewise, in research, people differ in how they like to work, and will have different ideas about what kinds of workflows will work for them. With git, you are not committed to very much at all, and you can pretty much come up with a workflow that fits how you would like to work. In my mind, this is a huge advantage when it comes to fitting the needs of researchers.

4. Git is *powerful.* While the basic operations of git are simple enough to learn and start using in a single sitting, it has a lot of depth. You may change your tastes and later discover that a different VCS works better for you. But you will

never "outgrow" git. It is used in very large, commercial-grade projects that exceed the complexity of most research projects by several orders of magnitude (ever hear of Linux?). It's very unlikely that you will get to a point in your own research career when you will need to switch to a system that does *more* than git. And if that does happen, you will probably not find that need covered by another system, and will probably have to code up your own system. And since git is open-source, you might be able to add that functionality yourself, or encourage the community to do so.

5 Why these tutorials?

If you visit the git site, you will immediately see a variety of links promising to show you about git or teach you. Indeed, another good thing about git is that a large user community means a large variety and depth in tutorials and help. So why am I bothering to write another tutorial?

The reason is that as I mentioned above, git (and all other VCS's) are geared towards programmers. To take a simple example, in the online Pro Git book, the example of "basic branching and merging" is an example involving web development, and incorporating a "hotfix" (fixing a bug on a live web site) with a branch of not-yet-live development. It's a nice example, but miles away from typical use that a researcher might need. So while most git tutorials have the needs (and skills) of programmers in mind, I'm aiming to address the needs of researchers. I think git is very accessible and easy to use, and brings enormous benefits, but approaching it can be difficult, simply because nearly all the documentation assumes a different audience. So I hope that with the audience of researchers in cognitive and social sciences in mind, these tutorials will fill a helpful niche.

That said, I do encourage you to check out the various other tutorials and resources throughout the web. There are many excellent and helpful things out there, and the main git website is an excellent place to start to find other resources.

If you're ready to start, [go here for the first tutorial!]