GIT101

Contributing to gitMechanics

Basics of LaTeX, git, and How You Can Help

Arizona State University

Brian Chevalier

Updated: October 15, 2018

Contents

1 Introduction to git

Git is an open-source, distributed *version control system* originally built by Linus Torvalds for managing contributions to Linux. Git is now one of the world's most used version control systems (all of Windows is not managed with git), one of the primary places git repositories are stored is on github.com (which is where this document lives). At it's core all git does is keep track of changes in files, but particularly plain text files. Plain text files can have many file extensions such as txt, md, tex, etc. We'll talk about these more later.

Git can be used through two primary ways: via a graphical user interface (GUI) or through the command line. Git is available on all platforms (even iOS, android, and chromeOS). Before we talk about actually using git we'll look at some basic terminology.

1.1 Vernacular

There are a few concepts that you'll want to understand in general before getting started with the particulars of using git. Some of these words can be very confusing for beginners so we're going to lay out some words that you'll want to be familiar with.

Repository Also called a "repo". This is a directory that stores your source files. There is a special folder called ".git" which contains the metadata about the repo.

Clone When you see a project on github you can clone that project. This is a full copy of a repository. Every repo is a clone which is why git is a distributed version control system.

Commit After you make changes to a repo you'll want to commit those changes. This is essentially like a bookmark in your repo that holds the state of your repo. Every commit must be made with a message (make sure it's useful!) and optional additional comments.

Stage Before you actually make a commit you will will have to "stage" or decide which files will be included with the commit. If you're using a GUI git tool this may just be a step while making a commit. On the command line you have to stage the files you are going to commit (this is done with git add filename or git add -a for all files in the repo to be added) and then commit them.

Fetch When you're working with a repo you typically have a version of that repo on a server that you want to route your changes through. If someone else has made changes to that repo or you made changes on another device you have to fetch changes, then merge them into your repo and push those changes back to the server.

Push You push changes to a server. You can push multiple commits at a time, so you don't have to push every time you make a commit or change a file.

Pull A pull is a combination of fetch and merge. You'll typically want to avoid pulling changes into your repo because you want to manually fetch and then merge in changes.

Pull Request This is a request made by someone to suggest changes to a repo that they do not have write access to. This is very common for open source projects so changes to the canonical source code is vetted by project maintainers.

Branch A repo starts with one branch called the master. You can create as many branches as you want. For instance, you may want to make a new branch that includes

Merge Conflict This is the error that occurs when two that git attempts to combine have different text on the same line number that it cannot reconcile. This requires manual fixing but does not often happen.

Diff A diff is to show the differences between two or more files. Depending on your knowledge, a GUI is likely better for seeing diffs since it will be easier to understand.

Checkout The power of git is that you can checkout previous commits of a repo. As long as all the files in your repo are already committed, you can checkout another branch or another commit. This eliminates the fear of making destructive changes. As long as you commit changes regularly and make descriptive commit messages you will always be able to return to a known change.

Cherrypick Sometimes when you're working with multiple branches you'll want to choose a commit from one branch and apply it onto another. This is what a cherrypick does. You will not likely need to do this.

1.2 Git on the Command Line

If you choose to use a command line tool for git it is very important you understand what these words mean as many of them are the names of commands you'll want to use when managing your repo. The most common are clone, add, commit, fetch and push. The typical steps you make when working on a repo is: clone someone else's repo. Make changes. Fetch changes from the server to see if any changes have been made there and merge in the changes. Add your files to a commit, then make the commit. Finally, you can push your files to the server. This can be accomplished by the following commands:

```
git clone [URL]
git fetch
git merge
git add --all
git commit -m "commit message"
git push
```

Note that you'll want to replace [URL] with the URL of the repo that you are cloning. You will enter each line for each part of the process.

Here is a great video on YouTube with an in depth example on using git from the command line. Even if you don't use the command line it is great for understanding how git works.

`https://youtu.be/0fKg7e37bQE`

2 Introduciton to LaTeX

LaTeX (pronounced lay-teck) is a free, open-source high-quality document preparation system (this document is written and compiled with LaTeX!). TeX was written and designed by Donald Knuth in 1978. LaTeX builds on top of the work of Knuth providing more functionality through a more advanced set of macros.

2.1 Vernacular

Macro A macro is a LaTeX command that can take input and does something with it. You can define any number of custom macros (also called commands) that can speed

up your document preparation and make your style more consistent.

Package A package is a set of predefined LaTeX macros packaged up for your use. Users can create their own packages. gitMechanics relies on a few custom packages.

Document class There are many document classes that you can use, and you can even define your own classes. Classes include: article, standalone, beamer (slide presentation format), report, book, etc. Each class provides a set of commands to typeset your work.

Environments Text in an environment is processed in a particular way by the compiler. Inside an environment particular types of commands are available for use. The entire body of the document is wrapped in the document environment. There are also list environments, math environments, floating environments, and of course you can define custom environments.

2.2 Getting LaTeX on Your Machine

Your first step is to install a distribution of LaTeX. This will be different depending on your platform, however it is available on all platforms (you can use the Linux distribution on ChromeOS. On iOS you can get a dedicated app, I reccomend TeX Writer). The following URL includes links on where to install the different distributions.

`https://www.latex-project.org/get/`

Most also come with a GUI editor and built in methods for compiling your documents so you don't have to touch the command line. Each editor will be slightly different so you'll have to spend a while getting to know your editor. Note that you can also opt to use your own text editor and compile from the command line using pdflatex. Regardless, make sure your text editor has good code completion support for LaTeX otherwise it will make learning commands a nightmare.

2.3 Compiling Your First Document

The most basic document needs two things. First you have to define your "document class". As discussed before, there are many classes. Document classes can also take optional inputs in square brackets before the actual document class. This document is an article class document and the options are landscape, twocolumn, and 12pt. The options must be separated by commas. The example code below typsets a blank document. Note that the article declaration is on the second line, which is completely valid, but is also done to fit within the column margins. Also note that your document will be within the document environment, and that the % sign indicates a comment.

```
\documentclass[landscape, twocolumn, 12pt]
{article}

\begin{document}
% Your document goes here
\end{document}
```

2.4 Document Structure

2.5 Loading Packages

Standard LaTeX is good, however, you will likely need to load additional packages to extend the language. You can import a package like the following:

```
\usepackage{amsmath}
```

You will want to typically use the AMSMath package since it adds many commonly used math symbols. You can find full documentation on the package at:

`https://www.ctan.org/pkg/amsmath`

The Comprehensive TeX Archive Network (CTAN) has many packages and documentation for packages that are typically already included with your LaTeX distribution. Most packages have the full documentation in PDF format available on CTAN.

2.6 Equations in LaTeX

If there's one thing LaTeX is known for it's typesetting math and equations. We're going to get into that now. For math to be added to your document you have to be in some sort of math mode. This means you could be in a math environment or have your math inline with delimiters.

Math environments include: align, equation, or a few more. Equations in this environment will automatically be centered and numbered. You can suppress numbering of these environments by adding "*" after the environment name. The following equation is typeset using the equation environment:

$$\int_0^L f(x)dx \tag{1}$$

```
\begin{equation}
\int_{0}^{L} f(x) dx
\end{equation}
```

Note that the braces around the limits of integration are not strictly needed for single character limits, however it is best practice to keep them in braces to avoid confusion and be consistent for more complicated equations.

The next important math environment is the align environment. The align environment allows typesetting multiple consecutive equations aligned. The following is an example of that:

$$y = mx + b \tag{2}$$
$$= \frac{\Delta y}{\Delta x}x + b \tag{3}$$

The syntax looks like:

```
\begin{align}
y &= mx + b\\
  &= \frac{\Delta y}{\Delta x}x + b
\end{align}
```

Things to note: the equations are aligned based on the location of the "&" and new lines are entered with "\". Greek letters and other commands also require a space after the command name otherwise it will throw an error.

2.6.1 Lableing and Referencing

Hyperlinking

2.6.2 Greek Letters

Most Greek letters can be included in math mode typically by typing "\" followed by the name of the letter such as typesetting the letter beta: β. The following PDF lists many Greek letters and math mode symbols.

`https://wch.github.io/latexsheet/latexsheet.pdf`

2.6.3 Operators

3 Drawing in LaTeX

LaTeX produces truly stunning PDF output, but as the great philosopher, Beyoncé once said, "Pretty Hurts".

GUI Options There are some options to avoid manually marking up drawings. One method is to simply produce figures and drawings in another application, and export as a PDF, or if you like the pixelated compression artifacts, JPEG. You can then save the file in the Figures subdirectory of your project and include something like the following:

```
\begin{figure}[H]
\centerline{
\includegraphics[width=\columnwidth]
{Figures/FIGNAME}}
\caption{CAPTION}
\label{fig:LABEL}
\end{figure}
```

You can also use a GUI editor that will output to markup formats and include those directly into your source code. You can export to TikZ format which we will discuss in the next section. A great option for this is GeoGebra:

`https://www.geogebra.org`

3.1 TikZ

While including external images works, it can get out of control pretty quickly if you have many figures. It also eliminates the possibility of using a set of macros to keep drawings consistent throughout many documents. Using generated TikZ still runs into this problem. Therefore, the only good option is to learn TikZ markup. For gitMechanics, I have already begun building a library of macros that will hopefully make the processs more straight-forward and consistent between documents.

4 Introduciton to Jekyll

Jekyll is a blog-aware static site generator. Github supports hosting websites with custom domains using the Jekyll blogging platform. The major benefit to this (other than that it is free), is that anytime the master branch of the github repo is updated via a commit, github automatically rebuilds the site using Jekyll and makes it live. This is great because all of the source documents can be stored in the same repository as the source files that build the website. It also has the added benefit of automatically rebuilding itself if someone makes a pull request on the repo and it is accepted.

5 Making a Pull Request