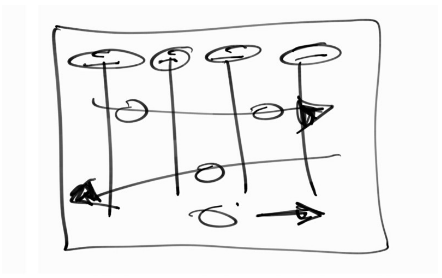
A Visual Introduction to Git

Ashley Keller Follow
Mar 16, 2015 · 7 min read

The lovely world of using Git explained in a simple way with hand drawn pictures. An easy technology to use, a complicated thing to wrap your head around. This article aims to bring fundamental understanding of Git to beginners.

So you want to share some code

Maybe you have found yourself here because you are working on a new team who is using Git or simply because you want to understand more about this awesome technology.


This is how the technical documentation on Git looked to me when I first saw it—very complex and unapproachable for a beginner

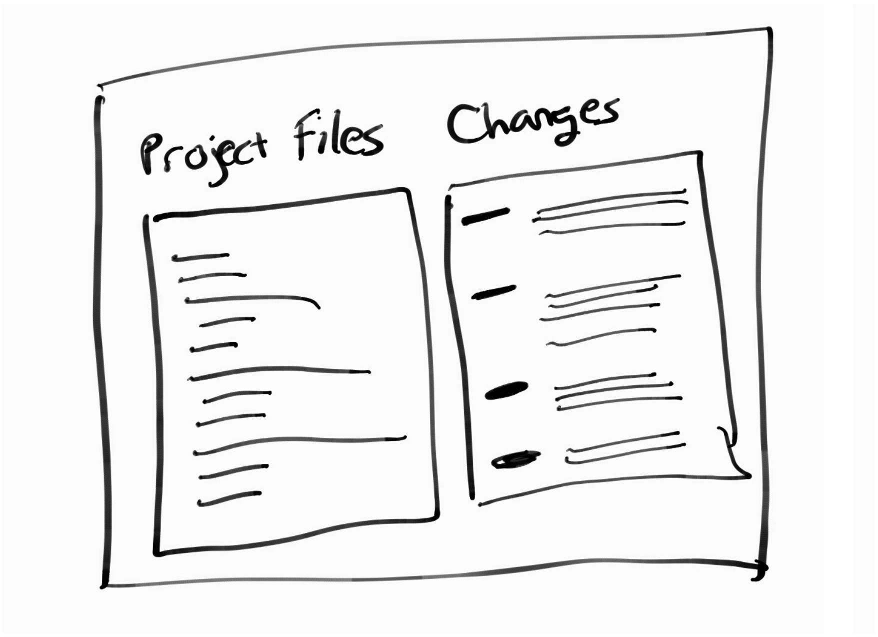
I remember the first time that I looked up a "Git Basics" article. It featured detailed, complex diagrams and a bunch of arrows pointing back and forth—it was a lot to take in. I also found that many of these articles were focused on developers who already had a strong handle on programming theory.

The purpose of this article is to take you on an illustrated journey of how I like to think about Git. Once you have read through everything please feel free to check out this technical documentation that has all the nitty gritty details you need to actually get started with development.

What is Git?

Git is a free and open source version control system for your code. This basically means that it manages changes to a project without overwriting any part of that project. It can be used by individual developers as well as large teams on any size of project.

At its little Git heart, Git is basically a collection of notes on the most recently worked on project files as well as a nice, chronological history of what has changed.



Git is basically a collection of notes on the most recently worked on project files as well as a message history of what has changed

Have you ever tried to work on the same file as someone else at the same time? Have you ever wished you could rollback your work to a previous version? Ever wanted to understand why a developer made certain changes? These are just a few examples of what Git can help you with.

The starting ground

A repository can also be known as a "repo"

This walkthrough will use the perspective of someone who has just downloaded a repository of project files from a source such as GitHub. A repository is fancy speak for a directory of files or storage space where your project lives.

Step one: Change some files

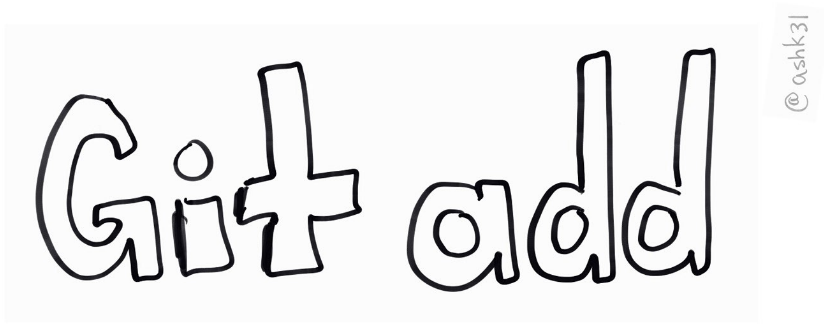
So you have just finished some work on the navigation component of your new project. You may have edited a little bit of the HTML and some CSS styling as well. For the purpose of this article, let's say the files you worked on are called *style.css* and n*avigation.hbs*.

You've made some changes to project files and now you're ready to share them.

When using Git it's always good to think about your work in logical chunks or little packages. Instead of working on a bunch of different things at once, try to discipline yourself to pause after completing specific pieces of your project.

For example, in this case we have worked on the navigation of our project. Before we go ahead and start to work on the footer or add a button below the carousel let's track these changes.

Step two: Collecting your changes



The "git add" command is one of the four basic commands you will use

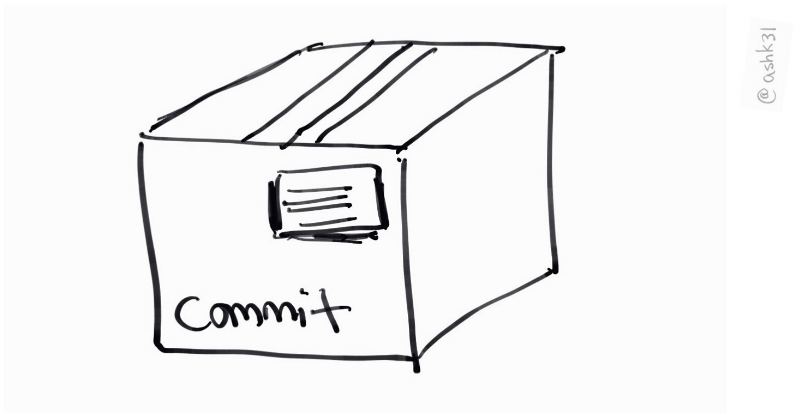
This is where the *git add* command comes into play. Using the command line or a GUI interface like Github for Mac we will add our changes to something called a *commit*. When you use *git add* you are essentially saying "Here are a few files that I've changed" and I like to think of it as placing them into an open package.

style.css
navigation.hbs
@ashk31
Commit

Using the git add command is like placing your changed files into an open package or a little box

Step three: Packing up your commit

Now that you have placed your files into the box we need to seal it up before we can send it on it's way. This is where you can use the handy *git commit* command.

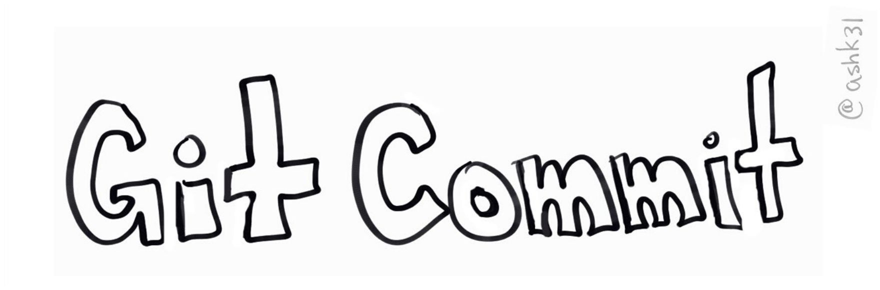


Now that our file changes are packaged up we have to explain why we made these changes

One way to think about using the *commit* command is that you are adding a packing slip to your package that explains what's in the box. When you use *git commit* you get the opportunity to write a brief message that explains what you have worked on and why.

Dear Git,
I made some changes to these files:
1)
2)
and here is why.
♡ me

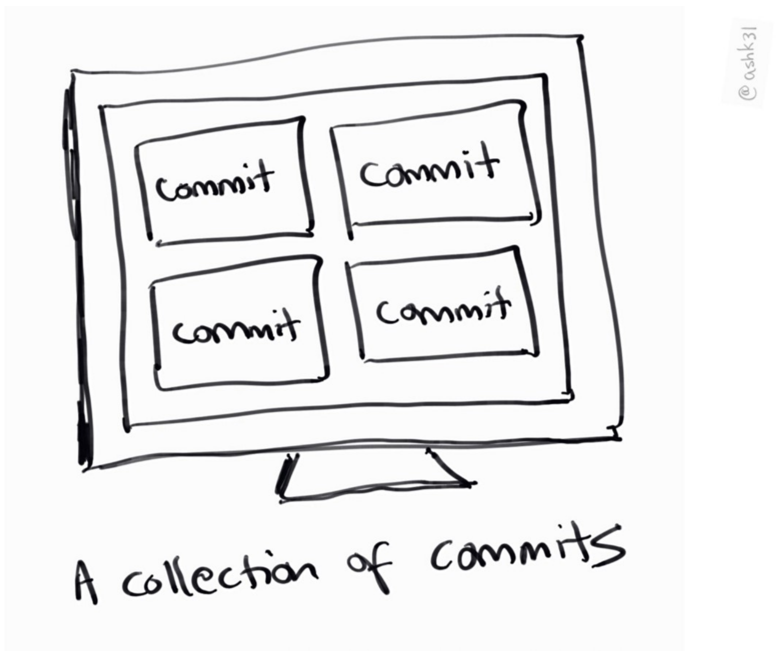
Note: You can create a commit that has only one file or ten files. It doesn't really matter how many files you add to the commit as long as you are doing it purposefully. It is also **best practice** to make many small commits versus completing huge sections of work before packing them up.



Using the git commit command is your opportunity to explain what you've worked on and to "seal up" your changes into a purposeful chunk

When you create a commit the system will file this away into the history of your project with a unique reference code. You can look up this code later and it will pull up the message (or packing slip) that you wrote when you used the *git commit* command.

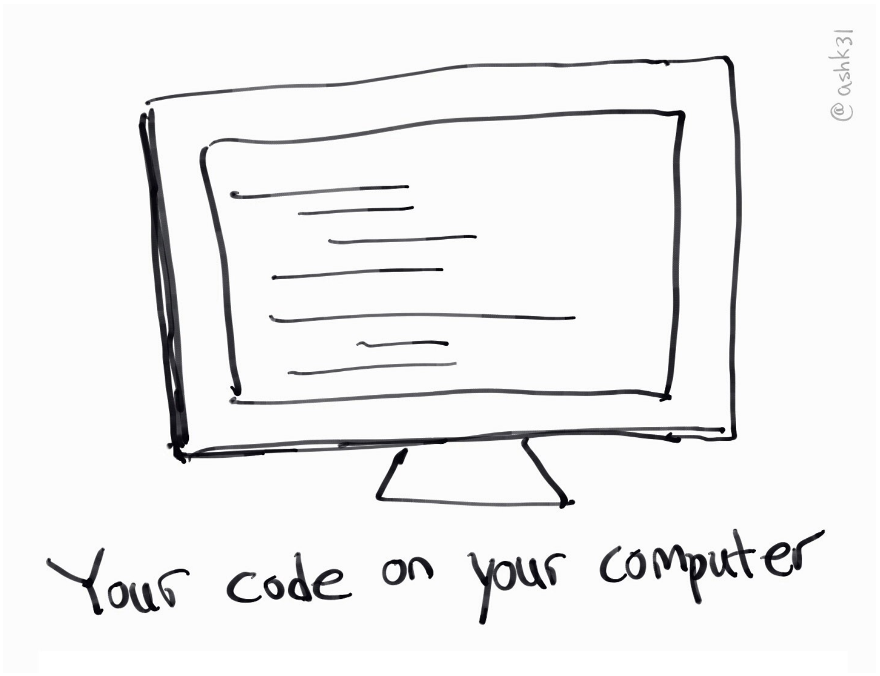
Everything until this point is still local on your machine



It is important to note that up until this point that **everything we have done is still local on our machine**.

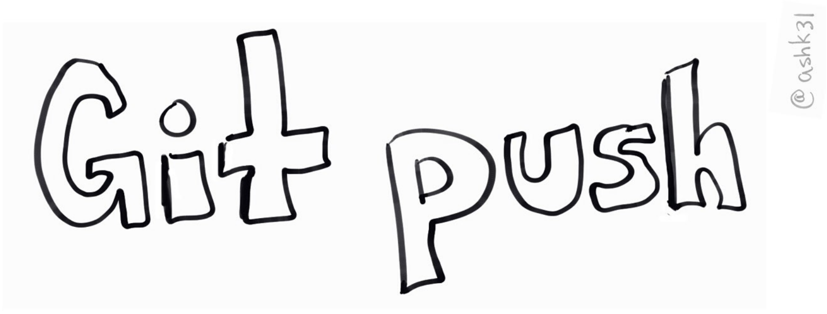
Essentially all of our little commit packages are collected on our computer and are waiting for delivery to the shared project storage space also known as the *project repository*.

This is one of the big advantages of Git over other version control systems such as SVN. Instead of always needing a connection to the file server like you do with SVN, you can work independently on your local machine and still have all the advantages of a file versioning system.



Git can be used on your local machine and does not need a connection to the server for you to still have all the benefits of version control

Step four: Share the changey goodness



Git push is the command you will use to send your changes to the project repository

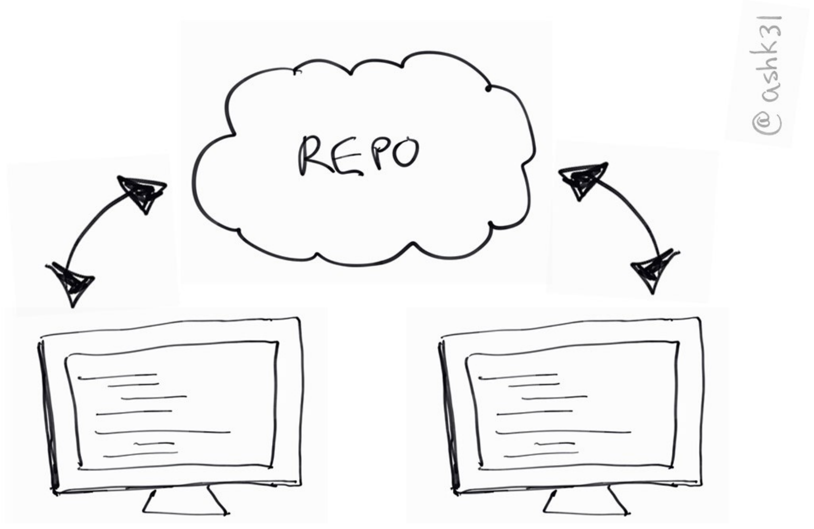
So once we have our commits ready to share with the project repository we can use the *git push* command. I like to think of using the *git push* command as if you are putting your packages on a delivery truck and sending them off. All of your packages are still separated but they will get delivered at the same time.

Beep! Beep! It's the commit delivery. Brought to you by the git push command.

Note: Before you send your commits off to the repository you still have the opportunity to amend the files in the package (add more files or remove some) as well as the mailing slip (your commit message). It is possible to do this after they have "been delivered" but it is a bit trickier. Similar to when Amazon sends you the wrong package. Here's more technical info on that.

Step Five: Continuing the cycle

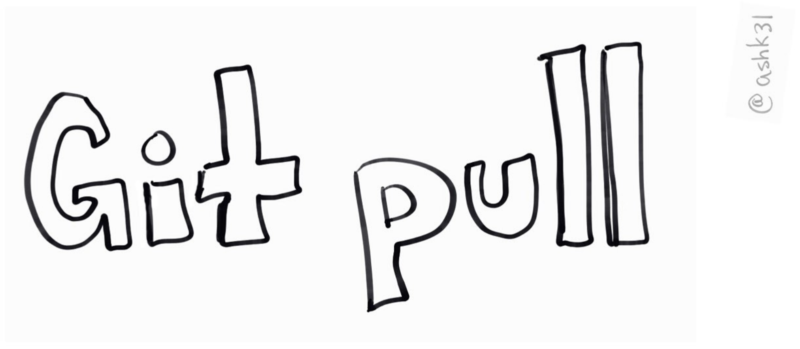
Hurrah! Your changes have been packed up and delivered successfully and now reside in the project repository. This is also sometimes called a *repo*. Now that your changes are in the repo other developers on your team can pull them down into their version of the project when they are ready to.



Now that your changes have been pushed up to the project server, other developers can merge them into what they are working on

A pull you ask? Why yes, that is the last basic command you will want to understand. See below.

Step six: Receiving changes from other developers



Using the git pull command you can receive changes from other developers working on your project

The *git pull* command is how you can send a request to the project repository and ask for all the most recent changes that other developers have committed. A *pull* will not happen automatically.

Doing a pull on the repository basically looks at all of the recent commits from everyone on your team and makes those changes to your project files as well

It's usually a good idea to "do a pull" on the project repository before you start working as well as before you push your changes just in case there are any conflicts that need to be resolved. If there are any issues Git will give you a little warning known as a *merge request*. Here is some more information on resolving merge issues.

The end.

Not really. This article only skims the surface of everything you can do with Git however hopefully it has provided some direction on how the technology works. There are many more best practices and possible commands that you can use to improve your workflow so I encourage you to make use of the extensive documentation and amazing development community to continually refine your Git skills.

Best of luck new Git user, I believe in you!