**Further Research**

[Version Control on Wikipedia](https://en.wikipedia.org/wiki/Version_control)

**VCS Info**

There are [a number of Version Control Systems out there](https://en.wikipedia.org/wiki/List_of_version_control_software). This alone should prove that version control is incredibly important. Three of the most popular version control systems are:

* [Git](https://git-scm.com/)
* [Subversion](https://subversion.apache.org/)
* [Mercurial](https://www.mercurial-scm.org/)

There are two main types of version control system models:

* the *centralized model* - all users connect to a central, master repository
* the *distributed model* - each user has the entire repository on their computer

**Further Research**

* [Centralized vs. DVCS from the Atlassian Blog](http://blogs.atlassian.com/2012/02/version-control-centralized-dvcs/)
* [Distributed version control on Wikipedia](https://en.wikipedia.org/wiki/Distributed_version_control)

**Recap**

Remember that the main point of a version control system is to help you maintain a detailed history of the project as well as the ability to work on different versions of it. Having a detailed history of a project is important because it lets you see the progress of the project over time. If needed, you can also jump back to any point in the project to recover data or files.

In this course, we'll be using Git which is a distributed version control system. You might be surprised to discover that you're already using version control all the time!

**Daily Use**

**Version Control Is Everywhere**

My job revolves around working with documents. I create new documents all the time, I fill them with information (hopefully informative information!) and then edit...edit...edit! Is your job like this? Perhaps it's not documents of text, but you're probably working with data in some form or another that changes over time.

Now you might not think that you're using version control when working with documents, and you'd be right...sort of. You're not *actively* maintaining different versions of a document as you write it. But that doesn't mean there aren't different versions of the document. The computer is keeping track of the different versions for you!

Don't believe me? Aside from pondering your propensity towards doubting, let's prove I'm right:

* open up your favorite text editor/code editor
* type some content (how about "version control is dull!")
* change one of the words in you wrote (e.g. change "dull" to "life-changing awesome")
* now (here it comes…!) press cmd + z or ctrl + z

💥 Version control in action! (See?...told you I wasn't lying) I bet you use the "undo" command all the time. I know that I sure do!

Practically every application I've ever used has an undo feature. You can think of this as a form of version control, but it's a rather limited form of version control. Let's look at a more powerful form by checking out a Google Docs document.

If you've ever written in a Google doc, have you noticed the small gray text at the top that tells you about the status of the document? Ever noticed that as you type, it's actively saving the document? Then, when you finish typing, it tells you that the document has saved.

The *real* question is, did you know that is a link that you can click on? Wanna see for yourself? Try it out in one of your own Google Docs.

Clicking on the link takes you to a "Revision history" page. (Ooo! Did you notice the word "revision"? The word "version" is a synonym for "revision"!)

**Revision History Isn't Powerful Enough**

Google Docs' Revision history page is incredibly powerful! I've used it on several occasions to salvage text that I'd written at one point, erased, and then realized I actually *did* want to keep.

But for all its ability, it's not as powerful as we'd like. What's it missing? A few that I can think of are:

* the ability to label a change
* the ability to give a detailed explanation of why a change was made
* the ability to move between different versions of the same document
* the ability to undo change A, make edit B, then get back change A without affecting edit B

The version control tool, Git, can do all of those things - *and more*!!! (bet you didn't see *that* coming!) So have I sold you yet on the awesomeness that is Git? I hope so, cause we're about to dive into it in the next section.

## Terminology

## Version Control System / Source Code Manager

A **version control system** (abbreviated as **VCS**) is a tool that manages different versions of source code. A **source code manager** (abbreviated as **SCM**) is another name for a version control system.

Git is an SCM (and therefore a VCS!). The URL for the Git website is <https://git-scm.com/> (see how it has "SCM" directly in its domain!).

## Commit

Git thinks of its data like a set of snapshots of a mini filesystem. Every time you **commit** (save the state of your project in Git), it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. You can think of it as a save point in a game - it saves your project's files and any information about them.

Everything you do in Git is to help you make commits, so a commit is the fundamental unit in Git.

## Repository / repo

A **repository** is a directory which contains your project work, as well as a few files (hidden by default on Mac OS X) which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer. A repository is made up of commits.

## Working Directory

The **Working Directory** is the files that you see in your computer's file system. When you open your project files up on a code editor, you're working with files in the Working Directory.

This is in contrast to the files that have been saved (in commits!) in the repository.

When working with Git, the Working Directory is also different from the command line's concept of the current working directory which is the directory that your shell is "looking at" right now.

## Checkout

A **checkout** is when content in the repository has been copied to the Working Directory.

## Staging Area / Staging Index / Index

A file in the Git directory that stores information about what will go into your next commit. You can think of the **staging area** as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repository.

## SHA

A **SHA** is basically an ID number for each commit. Here's what a commit's SHA might look like: e2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6.

It is a 40-character string composed of characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. "SHA" is shorthand for "Secure Hash Algorithm". If you're interested in learning about hashes, check out our [Intro to Computer Science course](https://www.udacity.com/course/intro-to-computer-science--cs101).

## Branch

A **branch** is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

Going back to the example of save point in a game, you can think of a branch as where you make a save point in your game and then decide to try out a risky move in the game. If the risky move doesn't pan out, then you can just go back to the save point. The key thing that makes branches incredibly powerful is that you can make save points on one branch, and then switch to a different branch and make save points there, too.

With this terminology in mind, let's take a high-level look at how we'll be using Git by looking at the typical workflow when working with version control.