Intro to Git and SourceTree

Contents

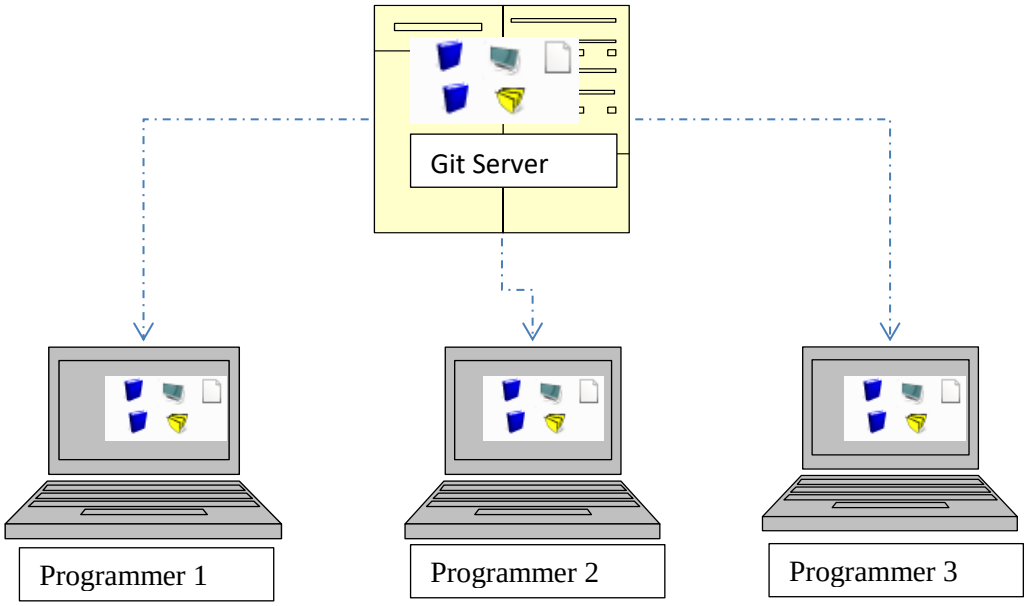
What are they?

Git is a *source code control system* aka *version control system*. It tracks changes to your programs and lets groups of people collaborate on the same program. Git is a command line program with no user interface.

SourceTree is nice user interface that is built on top of Git. It uses Git commands under the hood.

What are they good for?

When multiple programmers are all making changes to the same set of files it easy for changes to get lost. How do you know who has the most up to date version of each file? How do you know that when you get a copy of a file that someone else worked that you won't overwrite a change that you made that they hadn't gotten a copy of before?



Git solves this by keeping a master copy of all the source code files on a server. Each programmer downloads a copy of the files from the server and edits them on their own computer. When a programmer is done with a set of changes, they push their changes back up to the server. Others can then pull those changes down from the server and merge them with their local version.

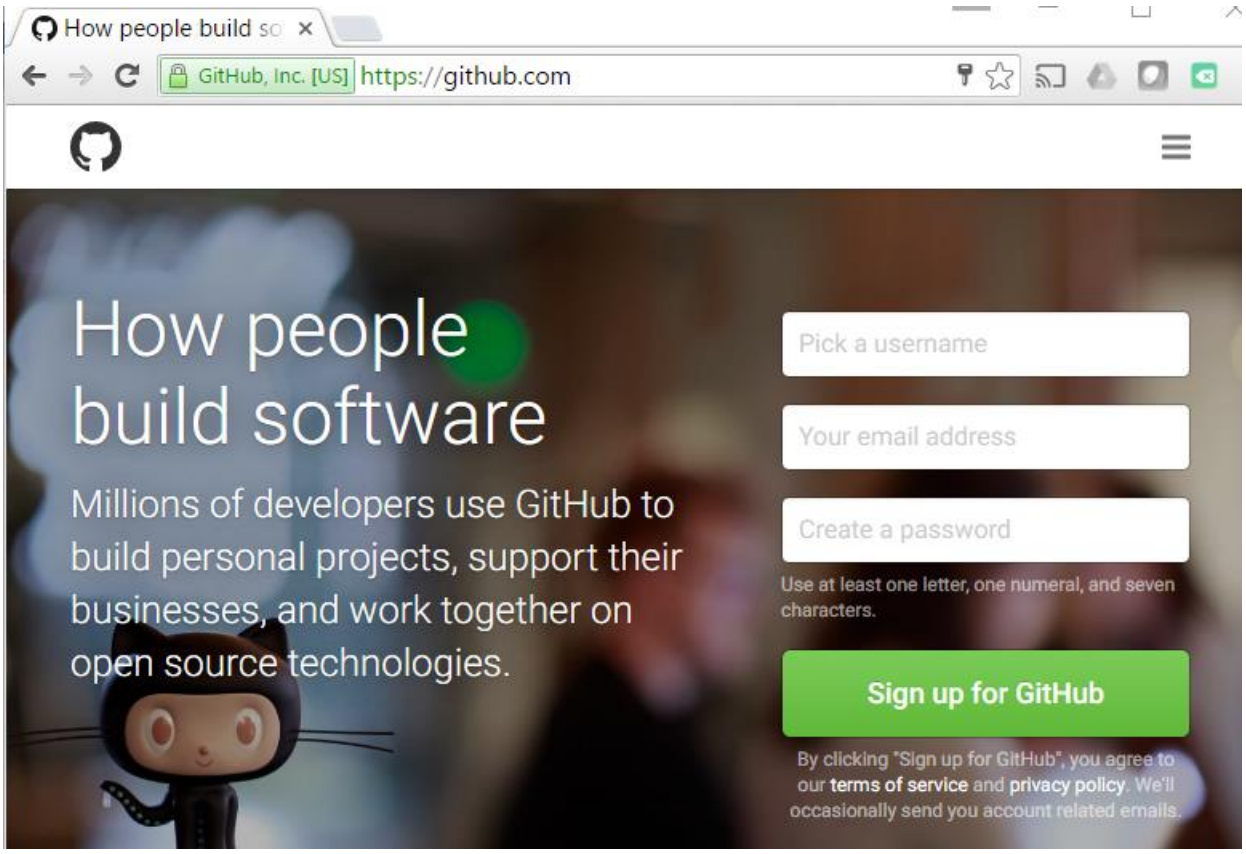
Setup

There are many different Git servers available. The most popular are publicly available servers that anyone can use use such as http://github.com and http://bitbucket.com. They both offer free and paid plans. Github is free as long as you make your code publicly viewable, Bitbucket is free for teams of up

to 5 users. There are also free Git hosting packages that you can install on your own server such as Bonobo Git Server and Gogs. For the training exercise we will use Github.

Sign up for an account

Go to https://github.com/ and sign up for an account.



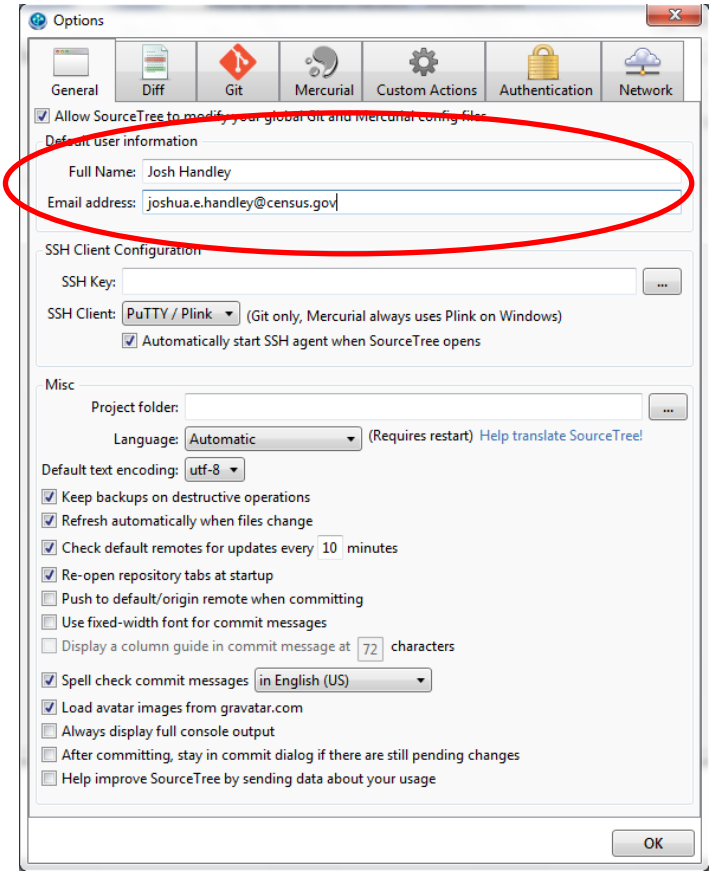
Once you have logged in let Josh know so that he can add you as a collaborator on the training repository.

Install Source Tree

To use Git to work with shared code you will first need to install SourceTree on your laptop. You can download and install it from: http://www.sourcetreeapp.com/

Set your name and email address

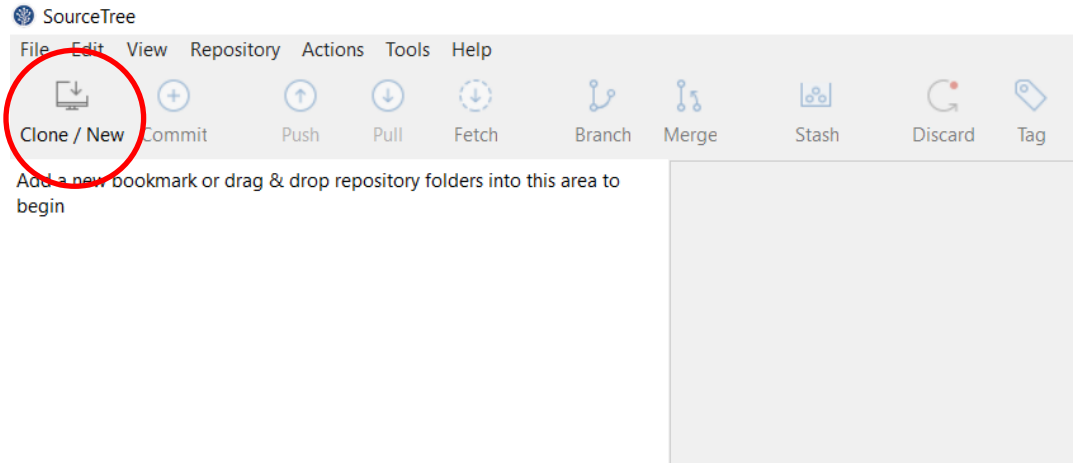
Before you can use SourceTree to interact with the Git server, you will need to enter your name email so it will be able to identify you. Go to the *Tools* menu and choose *Options*. Enter your first and last name and your email in the dialog under *Default User Information*.



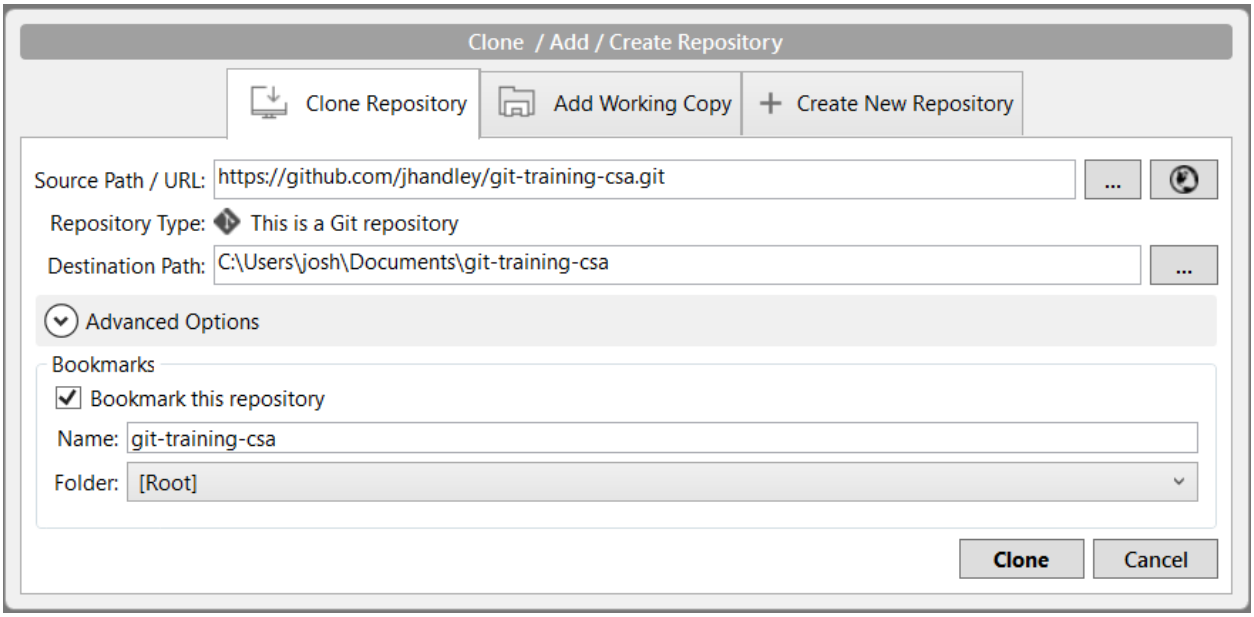
Clone the repository from the server

Now that SourceTree is configured, you will need to get an initial copy of the source code (the repository) from the server. In Git, you create a repository for each project. A repository is just a collection of files and their modification history. A copy of the repository will be stored on the server and another copy will be stored locally on your PC. To get a copy of the repository onto your PC you need to “clone it”.

To clone the repository, click on the *Clone/New* button in the top-left corner of the SourceTree window. We will start by cloning a simple repository containing a few text files that we will use for training. Later we will create and clone a repository for the census application.

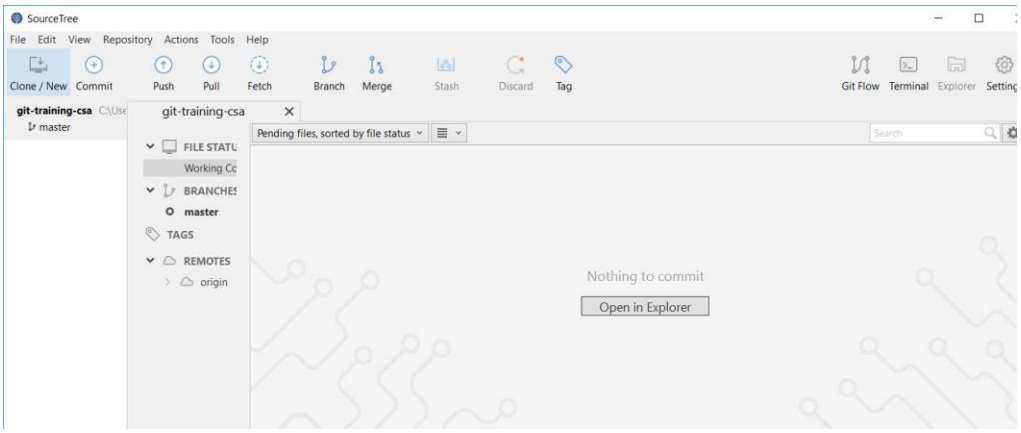


This will bring up the following dialog box:



Enter https://github.com/jhandley/git-training-csa.git for the Source Path/URL. Note that you can copy this from the Github website. Enter the location on your PC where you want to store the files for the *Destination Path* and click *Clone*. You will be prompted to enter your user name and password for Github. Use the name and password that you supplied when creating your account on Github.

Once the operation completes you should have a local copy of all the source code and your SourceTree window should look like this:

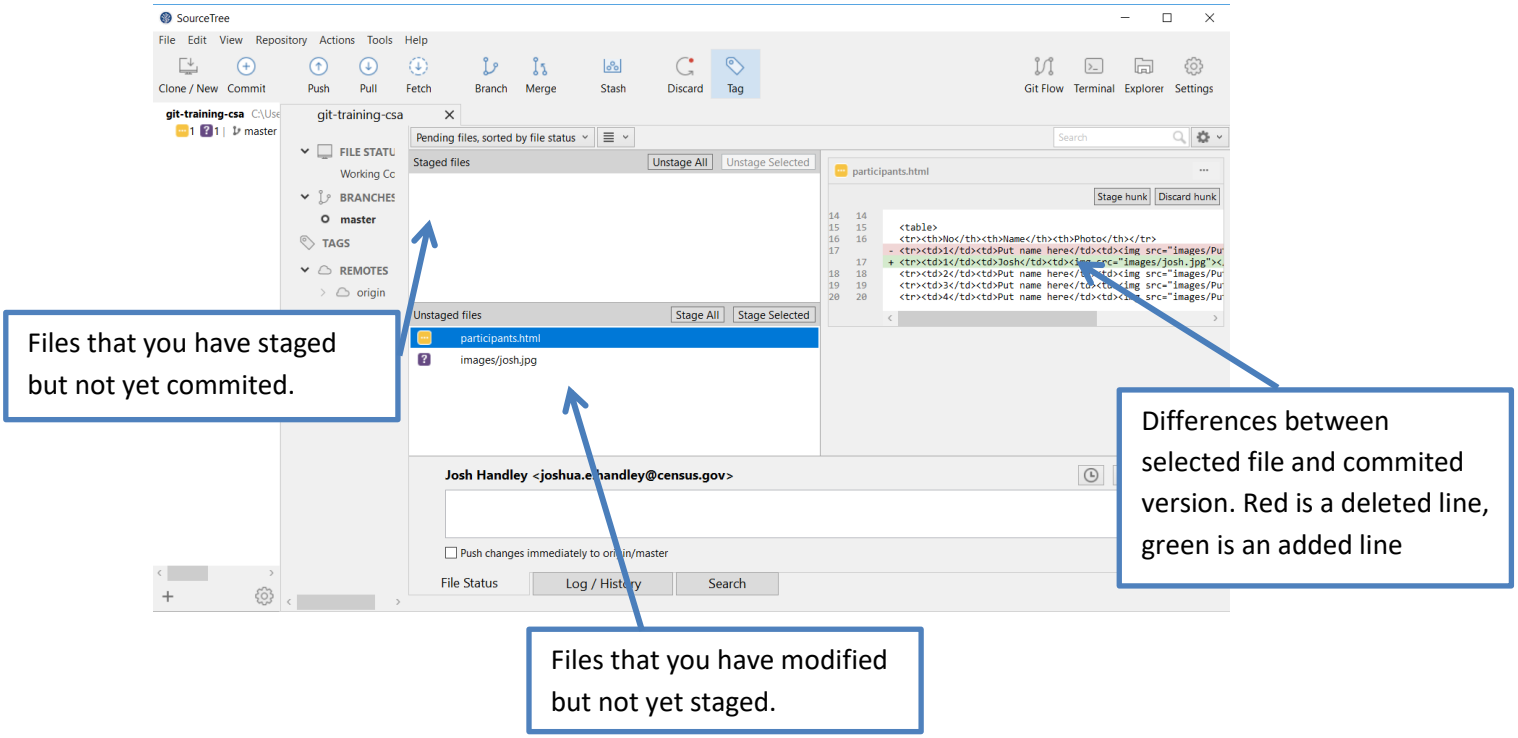


If you click on "Open in Explorer" you will see the files that you downloaded from the repository on your computer. You can now work with these files as you normally would. Once you have made changes you can go back to source to push your updates up to the server.

The SourceTree User Interface

SourceTree has a lot of different windows and commands. We will focus on just a few key functions to get started. There are a lot of advanced options that are needed when working with huge teams and huge projects that we will not need.

The most important parts of the user interface are the toolbar along the top where you find buttons for the commands you will use and the staged and unstaged file areas that you will use when committing changes that you have made.



Getting the Latest Version of the Code

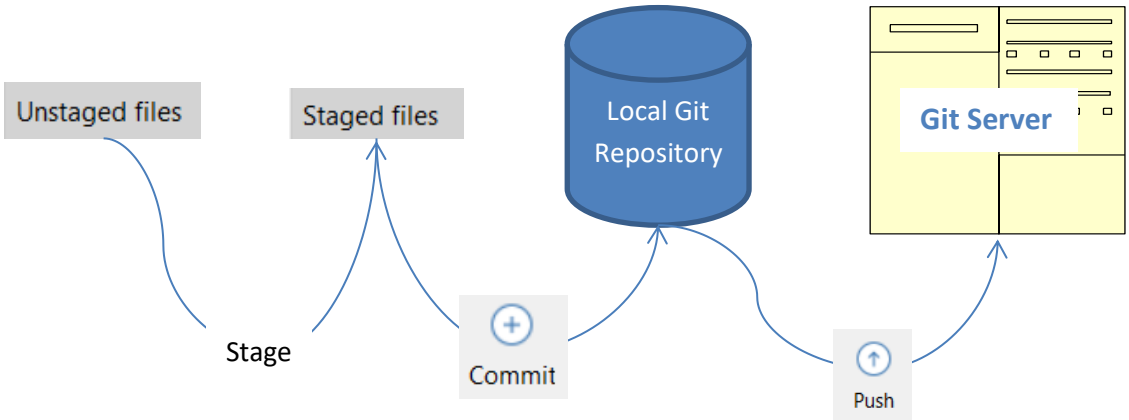
To get the latest version of the code from the server use the *pull* button on the toolbar.

This will retrieve any changes that have been made to the code since the last time you did a *pull*. It is a good idea to do a pull anytime you are about to start to make changes to any of the programs so that you won't be working with out of date files.

In the case where you have made changes to one of the files on your PC and someone else has already pushed changes to the *same* file to the server, Git will try to merge their changes into yours. Usually it will merge this automatically, however sometimes if you have both changed the same line of code then you will be asked to manually resolve the conflicting lines in order to complete the merge.

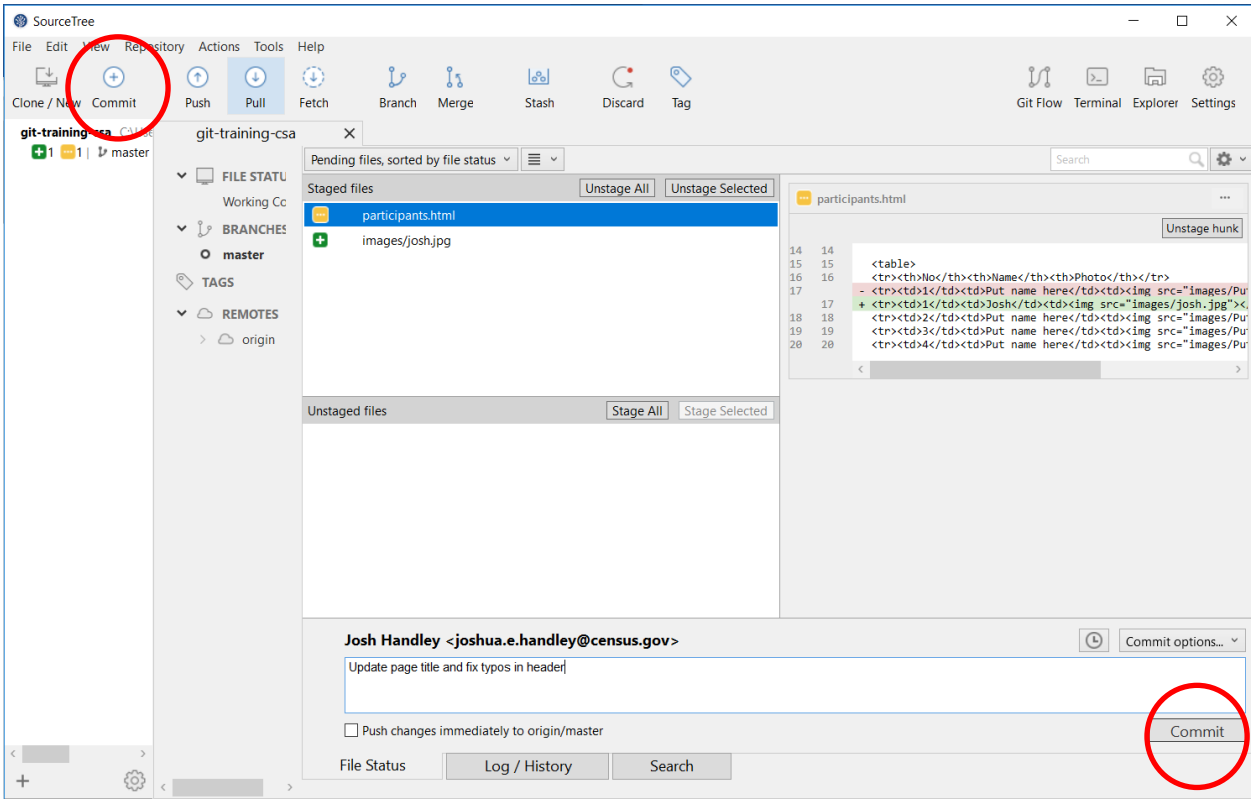
Committing and Pushing Your Changes

Putting your changes up to the server is a multistage process that involves three steps: *stage*, *commit*, and *push*. For the most part you will just do all three at the same time.



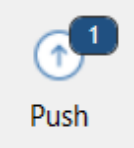
First stage the files you want to push by selecting them in the unstaged files window and clicking "Stage Selected". You may want to examine the differences between the files you have modified and those that were last committed by clicking on each file and looking in the differences area of the Window to verify that the changes that you are about to commit are the changes you really want to make. Once you have all the changed files together in the staged area and are confident that they are all correct, click the

commit button on the toolbar. Clicking commit will commit all the staged files but not any of the unstaged files. This way you can commit a subset of the files that you have changed if you wish.



When you click the commit button a dialog will pop up at the bottom of the SourceTree window with an area where you can add comments describing what you have changed. You should **ALWAYS** add at least a sentence or two describing the changes that you made so that other people can understand what you have done. If you have made a lot of changes or made a major change that others need to be aware of then feel add to add more comments. Once you have entered your comments, click the commit button in the bottom right corner to save your changes.

At this point your changes are comitted locally but they have not yet been sent to the server. They only get sent to the server when you click the push button. If you have commited files but not pushed them to the server SourceTree indicates so in the tool bar. If you have one unpushed commit it will display the number one on top of the *Push* button.

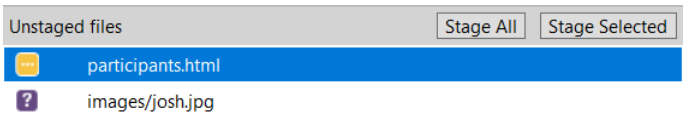


Note that you can stage and commit files even if you do not have an internet connection however you must have a connection to push or pull.

If someone else has already pushed changes to the file that you are trying to push, the server will refuse your changes. You will need to first pull their changes and merge them into yours before pushing your changes to the server.

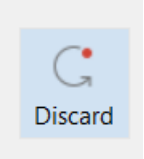
Adding New Files

Committing new files is the same as committing modified files. Just create the new file as you would normally and SourceTree will automatically add it to the unstaged files area. It will be marked with a question mark icon to show that it is a new file as opposed to a modification to an existing file. To commit new files, you just select them and click "Stage selected files" as you would with modified files. You can mix new and modified files in the same commit.



Reverting Local Changes

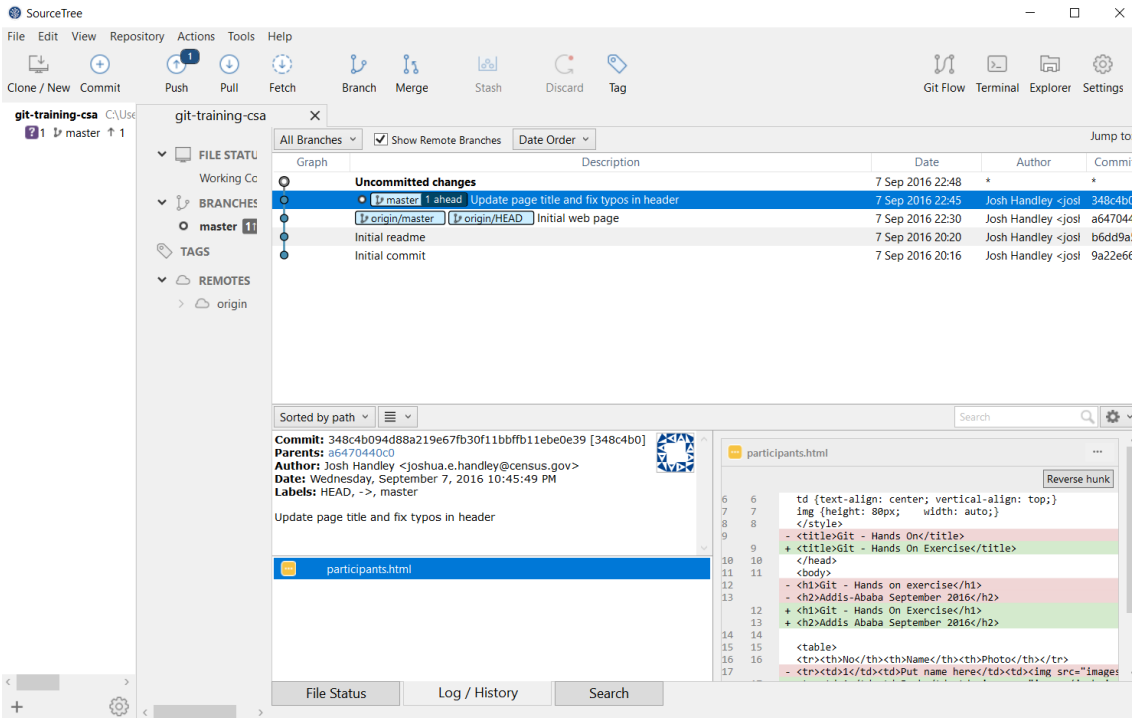
If you have unstaged or staged files and you want to revert back to the old version of the file you can select the files to revert by checking the boxes next to them in the unstaged or staged files area and then clicking on the *Discard* button. This will revert all selected files back to the way the way that they were as of the last commit.



Looking at the Commit History

Often times it is useful to see what changes you and others have made. You can do this by clicking on the Log/History tab on the bottom of the SourceTree window. This shows a list of all commits that have been made to the code starting with the most recent first. You can click on any commit to see the details of what was done. This will show the comments made when the commit was done plus a list of all files changed. Clicking on any of the changed files will display the differences between the before and after

of selected file.



The Typical Workflow

Typically when working together on a project each programmer uses the following workflow when working on adding a new feature or a bug fix to the software:

1) Pull to get the latest code onto your machine
2) Code the new feature or bug fix
3) Once coding is done, test locally
4) Pull again before committing in case anyone else has changed code
5) Retest code to make sure the pull/merge didn't break anything
6) Commit and push code changes

Practicing with the training repository

The training repository contains a web page with a list of the names of all the workshop participants and a placeholder for a picture of each participant. Your task is to clone the training repository to your local machine, edit the file participants.html to correct the spelling of your name since Josh inevitably spelled it incorrectly, add a picture of yourself to the images folder and replace the image name "unknown.jpg" in the row of workshop-participants.html with your name in it, with the name of your image file. For the row for Josh should change from:

```
<tr><td>1</td><td>Josh</td><td><img src="unknown.jpg"></td></tr>
```

To

```
<tr><td>1</td><td>Josh</td><td><img src="josh.jpg"></td></tr>
```

Open the html file in your browser to verify that your name and image are correct and then use SourceTree to stage, commit and finally push your changes to the server.

The Github Web Interface

Github itself has a web interface that offers a limited set of functionality. It allows you to view files and commit history but you cannot edit or commit anything through the web interface. You will probably use SourceTree rather than the web interface for most tasks that you do.

Annex: Git Glossary

There is a lot of jargon that you will run across with Git. Here are some definitions:

Blame

The "blame" feature in Git describes the last modification to each line of a file, which generally displays the revision, author and time. This is helpful, for example, in tracking down when a feature was added, or which commit led to a particular bug.

Branch

A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.

Clone

A clone is a copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy. With your clone you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online. It is, however, connected to the remote version so that changes can be synced between the two. You can push your local changes to the remote to keep them synced when you're online.

Collaborator

A collaborator is a person with read and write access to a repository who has been invited to contribute by the repository owner.

Commit

A commit, or "revision", is an individual change to a file (or set of files). It's like when you save a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.

Contributor

A contributor is someone who has contributed to a project by having a pull request merged but does not have collaborator access.

Diff

A diff is the difference in changes between two commits, or saved changes. The diff will visually describe what was added or removed from a file since its last commit.

Fetch

Fetching refers to getting the latest changes from an online repository (like GitHub.com) without merging them in. Once these changes are fetched you can compare them to your local branches (the code residing on your local machine).

Git

Git is an open source program for tracking changes in text files. It was written by the author of the Linux operating system.

Merge

Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another.

Pull

Pull refers to when you are fetching in changes and merging them. For instance, if someone has edited the remote file you're both working on, you'll want to pull in those changes to your local copy so that it's up to date.

Push

Pushing refers to sending your committed changes to a remote repository hosted on a server. For instance, if you change something locally, you'd want to then push those changes so that others may access them.

Remote

This is the version of something that is hosted on a server, most likely GitHub.com.

Repository

A repository is a set of files in git that are tracked together. A repository contains all of the files for a single project.