An Introduction to Git and Github for UMD Freshmen

Frequently Asked Questions

Q: What is Git?
A: Git is a piece of software that allows you to track, document, and revert changes to any documents or files that you add, edit, or remove.
Q: What is GitHub?
A: GitHub is a website that integrates with Git, allowing you to save, document, and retrieve your code using an online database.
Q: What do I need to know in order to learn Git?
A: Nothing! Git doesn't require any prior knowledge. However, it does have specific terminology and commands that you will need to learn to effectively use Git.

Introduction: Why learn Git?

You're on your fifth hour working on this programming assignment. It's 1:00 am, and you just had your program working, but you changed one line somewhere that you don't remember, and now everything is broken. What do you do? You need a piece of software that tells you all of the changes you've made to a project and seamlessly allows you revert them. Thankfully, such software is already created; they are called version control systems, or VCS for short. Dr. Richard Maclin, a computer science professor at the University of Minnesota Duluth, notes that the software engineering industry "wouldn't be what it is today without the aid of [VCS]." While many VCS exist, Git is the most popular choice among companies, and having proficiency in Git is sure to add an edge to your resume, as well as save hours of your life sifting through code looking for changes.

Installing Git on Windows

For installing on Windows, you can download Git at: http://git-scm.com/download/win. Follow the walkthrough prompt, read and agree to the terms of service, and click the install button.

Installing Git on Mac

For installing on Mac, you can use Xcode Command Line Tools. On version 10.9 or above you can simply run git from the terminal to automatically install it:

```
$ git --version
```

Installing Git on Linux

For installing Git on Linux, you can usually use your distribution package management tool.

If you're using a Debian distribution, such as Ubuntu or Mint, you can use apt:

```
$ sudo apt install git-all
```

If you're using a RPM-based distribution, such as Fedora, RHEL or CentOS, you can use dnf:

```
$ sudo dnf install git-all
```

For any other distributions, the Git website provides instructions at: http://git-scm.com/download/linux

Signing up for GitHub

GitHub allows you to sync your changes that you make with Git. For example, you can write a program on your laptop, save it using GitHub, and then work on it on a library computer without needing to copy the changes back and forth. Are you not sure how to register a GitHub account? You already have one! The University of Minnesota provides a professionally licenced GitHub account which you can login to at https://github.umn.edu/login using your UMD login.

Step 1: Create a repository

When using Git, projects you work on are stored in a *repository*. Repositories are folders you use to put everything you need for a project to work into. There are two ways to start a repository. You can create one from scratch, or *initialize* one. You can also access a repository that you or someone else has already initialized. This is known as *cloning* a repository. You can clone a repository from your local computer, or from a remote server. To clone a repository you will need a "path" to it, which is generated when you create a repository. You can think of a path to a project similar to a path to a friend's house: You need to know where to go to access their house, and Git needs to know where to go to access the repository.

Initializing a repository from scratch:

```
$ git init
```

Cloning a repository from a local repository (on your computer):

```
$ git clone /path/to/repository
```

Cloning a repository from a remote repository (on a server):

```
$ git clone username@host:/path/to/repository
```

For example, I have a username of "karls071" and a repository named "MazePathFinding", which I can clone using the command:

```
$ git clone https://github.umn.edu/karls071/MazePathFinding.git
```

Step 2: Understanding updating files

Now you have a folder that uses the Git versioning control system. You can move or create any type of files in this folder, and they are considered part of this "project" or repository. Now, consider you programed a file called "myProgram.cpp". You added an important function to this program and saved it. You will need to save those changes to your repository. In order to do this, you must *add* those files to the list of files you wish to update, *commit* those changes, and then *push* them to the remote server. Let us break down each step.

Step 3: Adding files

Any files that you wish to update must be added to the list of files to update. Thus, you can use the add command. If you wish to add all changed files, you can refer to everything as "*". Git is smart enough to only add files that you have changed.

```
$ git add myProgram.cpp
```

Or to add all changed files:

```
$ git add *
```

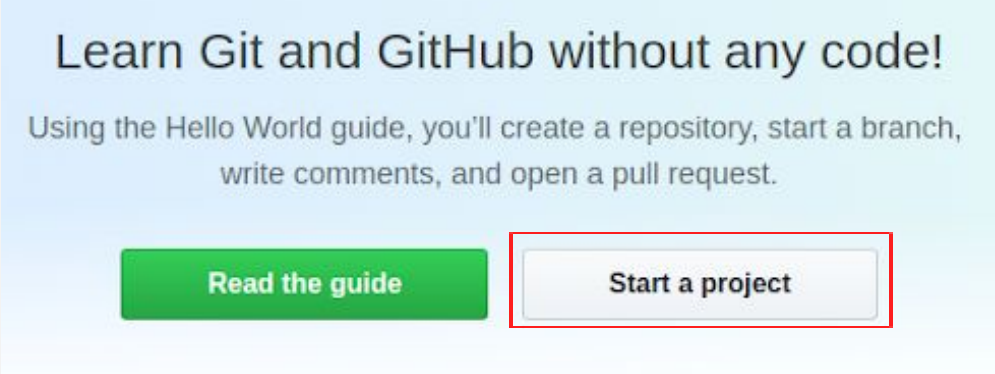
Step 4: Committing files

Now that you have added the files you have edited, you can commit them. Committing files will package all of the changes in the files you've added together, bundled with a time, a message about what the changes are, and a pinpoint that you can always revert to. You use the '-m' option to specify you are making a message with your commit. You should always leave comprehensive comments, so you remember what changes you made in each commit.

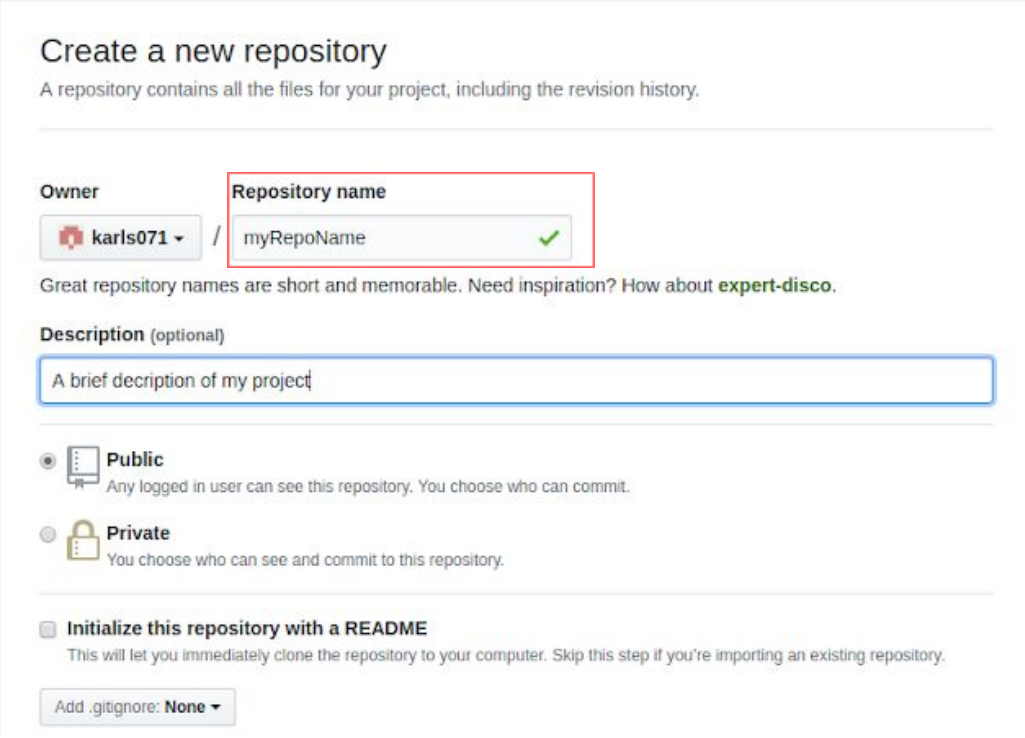
```
$ git commit -m "This is a commit comment. Here I should talk about what changes I made to myProgram.cpp"
```

Step 5: Creating and syncing an online repository

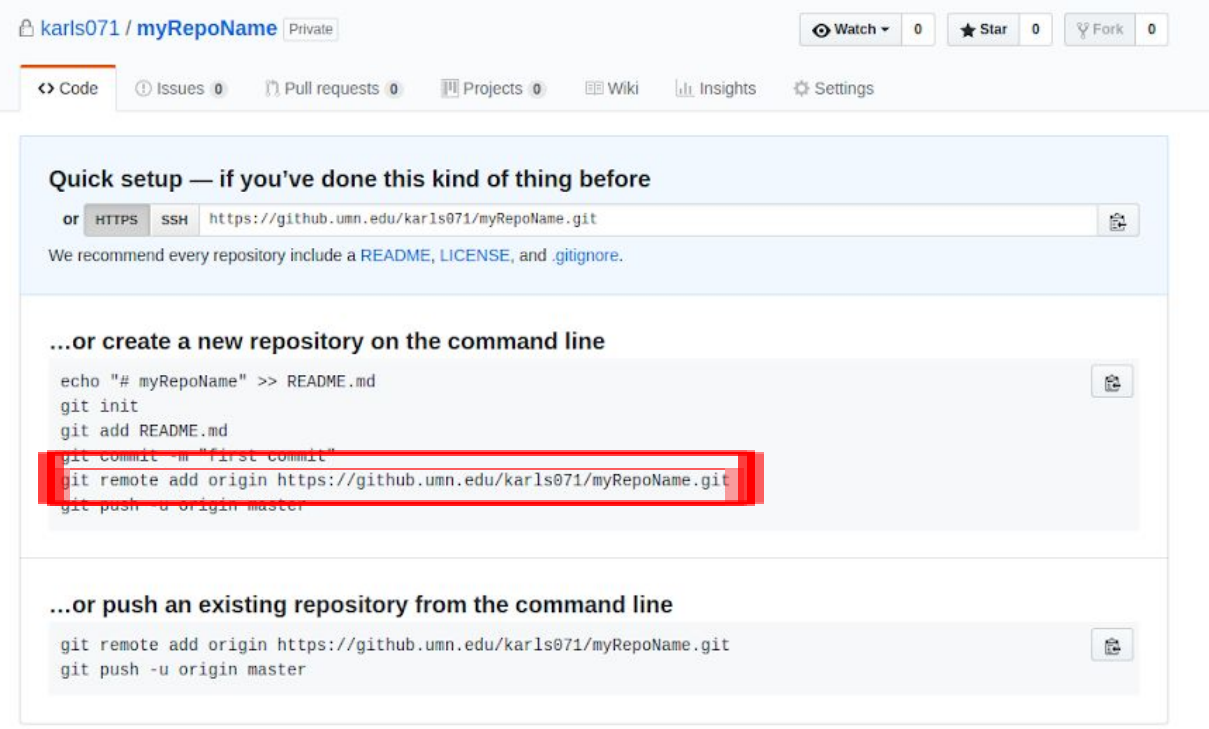
Using these steps so far, you are now able to store and document changes on your local machine. However, if you want others to be able to see your work, or if you want to work on this project from another computer, you will need to *push* your changes. In Git, you have the ability to "push" and "pull" changes you make to and from a server. If you imagine your program is a box of code, you can push it over to a storage unit. Later when you need it, you can pull the box back from the storage unit (server), open it up, and continue to make changes to it. The most popular server for storing code is GitHub.com. After logging in to your UMD account here, you will see two buttons, one for a quick reference manual for basic Git commands, and the other is for creating an online repository to sync with your local one.



After selecting the "start a project" button, you will be brought to the following page:



Here you will decide a repository name, description, and accessibility. Public repositories are accessible to the public, and private are only available to you or people you choose. Caution: *Never* make a repository containing academic material public unless an instructor *explicitly* tells you to do so. If a student uses your repository to cheat, you *both* will be held accountable for academic dishonesty, regardless of whether or not you intended them to use it. After deciding a repository name, description, and accessibility, you will be brought to the following page:



This page gives you the address of your repository along with the command to remotely add it, which I boxed in red. Using the address, back in your Git directory, you can sync this remote repository with your local one by using the remote add command that the page provides you with. It will look different based on your account and repository name, but it will follow the format of:

```
$ git remote add origin https://github.umn.edu/accountName/repositoryName.git
```

Step 6: Pushing files

Now that you synced your local repository with the remote one, you can push your changes using the push command:

```
$ git push origin master
```

"Origin" is your local code, and "master" is the code currently saved on GitHub. You can think of the relationship of origin and master similar to a rough draft and final copy of an essay. You start out with room for improvement in your rough draft (origin), but you steadily improve it. Once you are satisfied, you submit your rough draft as a finished copy (master). Thus, you are pushing your "original" changes to your "master" program. Note: While you are working on projects, you should always remember to push your changes so you can access them from other devices if you need to.

Step 7: Reverting changes

In order to get a list of all changes, you can use the git log command:

```
$ git log
commit 25939f9e840d1d629eef340393da96c9851e1abc
Author: karls071 <karls071@d.umn.edu>
Date:   Sat Sep 22 14:24:24 2018 -0500
    Fixed formatting issue with maze text output
commit e59f0748ec142ef373c9ff5474ecac24abb762d2
Author: karls071 <karls071@d.umn.edu>
Date:   Sat Sep 22 00:01:11 2018 -0500
    Updated documentation
commit c0949590c07be48c72c83872cf27fec63fd60aae
Author: karls071 <karls071@d.umn.edu>
Date:   Fri Sep 21 23:57:25 2018 -0500
    Initial commit
```

This is an example of a project that I am working on. It shows each commit's "name" which is the long string of alphanumeric characters. It also shows each commits author, date, and message. All you need to do is use the git revert command, which takes the commit's "name" as an argument:

```
$ git revert 25939f9e840d1d629eef340393da96c9851e1abc
```

Note: This will only revert the changes made in that specific commit, so even if it was made four commits ago, it will not revert any of the changes made after it.

Conclusion: Further uses

This tutorial has only scratched the surface of what versioning control systems can do, so if you are curious about anything else Git has to offer, or if you want more examples or documentation, you can find further documentation on the Git official website at https://git-scm.com/docs.