Beginning Git, ISTA-420

Charles Carter

June 16, 2017

1 Beginning Git (and Github)

Version control is hard. This isn't a function of version control software, but a function of the complexity of software, the nature of collaboration, and the sheer quantity of source code. Many developers (and others) use `git` for version control. It's fairly simple to use, effective for the intended purpose, and (being open source) is both freely available and extensible.

This introduction seeks to simply introduce `git` and Github — this is *not* in any sense a tutorial. It should be sufficient for new users to "dip their toe in the water." New users can find many good tutorials using their favorite internet search engine, and new users *should* take the time to work through two or three of these early in their version control career.

The first thing to recognize, and to remember, is that `git` is a *distributed* version control system. This means that every user has an exact copy of the central repository on their local machine. The workflow I follow for a one person project is to maintain a local working directory, and to update the remote repository (on Github) from my local repository. For me, if I have to pull from the central repository to update a local directory, something has gone wrong. Of course, if you work away from your usual machine, you can pull (or clone) your remote repository, and create a new repository *exactly* identical to your usual machine . . . assuming, of course, that you have been diligent in updating your remote repository from your local machine.

This introduction follows a typical work flow. As always, you should read the technical documentation *before* you run the command. RTFM!!! An initial work session may look something like this:

Listing 1: Workflow

```
> git config --global user.name ''Charles Carter''
> git config --global user.email cartec22@erau.edu
> git config --global core.editor gvim
> git init
> git add *.c
> git status
> git commit -m ''first commit''
> git log
> git push origin master
```

2 `git` Workflow

2.1 Prerequisites

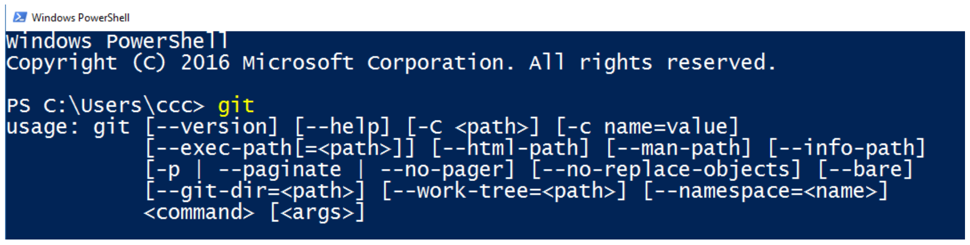
There are three prerequisites to this introduction: (1) getting `git`, (2) using PowerShell, and (3) establishing a Github account.

1. `git` for Windows can be obtained from `https://git-scm.com/download/win`. Download the appropriate version for your machine and install it. Everything should be pretty straight forward.

2. This introduction uses Windows PowerShell. PowerShell should be installed on current Windows machines. One way to invoke Powershell is to enter `powershell` in the run box. You can use the DOS commands you already know, such as `dir, mkdir, cd, del, copy, move,` etc.

3. If you don't already have a Github account, go to `https://github.com/` and establish a Github account.

Once you have installed `git`, start PowerShell and enter `git` at the command prompt. You should see something like figure 1. If you don't, you have not installed `git` correctly.



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\ccc> git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

Figure 1: Successful `git` install

2.2 Configuration

Before you create your first local repository, you should configure your local `git` system. At a minimum, you should configure your user name and your email address. You can configure your favorite text editor to work with your system. You can set other options, but this introduction does not discuss those. See lines 1, 2, and 3 above in listing 1. RTMF!!!

2.3 Initialization

When you begin a new project, you always create a new directory to hold your project files. You frequently will have various subdirectories in your main project directory to organize your working files by type. Immediately after you create your new project directory, you should initialize a `git` repository. Running the command on line 4 in listing 1 initializes your new repository and creates a hidden subdirectory (named **.git**). This subdirectory maintains a persistent record of your versions.

2.4 Adding Files

To add files to your local repository, use the `git add` command. See line 5 in listing 1. The usual file globs work, e.g., the asterisk (*) for zero to many characters. Be sure to include the path for all files contained in subdirectories. For example, if you place your documentation in `/project/documentation` as PDF files, you should `git add ./documentation/*pdf`.[1] You only have to add a file once. If you edit the file, `git` picks up the edits. However, if you create new files (which you undoubtedly will), you will have to add them to your repository. RTFM!!!

2.5 Checking Status

After you add files, you can check the status of your staging area. See line 6 in listing 1.

2.6 Committing Files

When you commit files to your repository, run the command on line 7 in listing 1. The flag `- m` indicates that your commit message follows on the command line. You cannot commit to your repository without a commit message. If you fail to include a message on the command line, `git` will throw you into your default text editor to type a message. It's easier to include your message on the command line,

[1]Please pay attention to the difference between the backslash character (\) Microsoft uses, and the UNIX based forward slash (/).

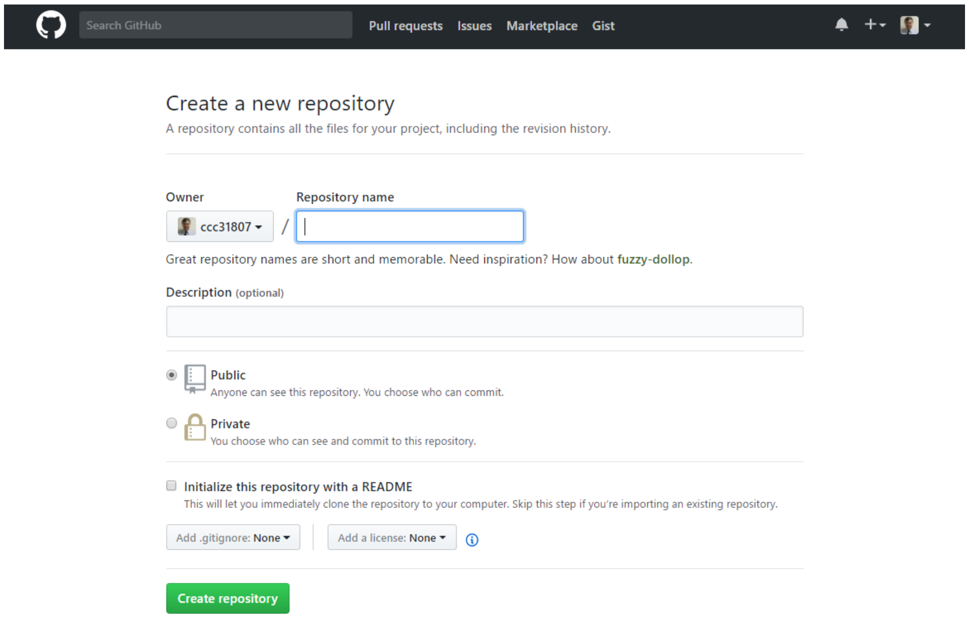
If you have not added any new files, only edited files already committed, you can combine adding and committing with this command: `git commit -am 'message'`. The flag `-a` indicates that you want to add all the files that have changed since the last commit. RTFM!!!

2.7 Activity Logs

`git` shows your activity with the command shown on line 8 in listing 1.

2.8 Creating a Remote Repository

Create a new repository on Github by clicking the plus symbol in the toolbar on the upper right of the screen. If you select "new repository", you will be taken to the set up page, shown in figure 2. Choose a short name (which cannot be changed), a description (which can be changed), and create the repository. The resulting page will give you a series of commands to connect your local repository to your remote repository. Just copy those commands, paste them into your PowerShell instance, and run them. These settings will be written to your local repository, and you will not have to do this again.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: ccc31807 / Repository name:

Great repository names are short and memorable. Need inspiration? How about fuzzy-dollop.

Description (optional)

[✓ Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with a README — This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Figure 2: Creating a remote repository

2.9 Pushing to a Remote Repository

Finally, to synch your local repository with your remote repository, run the command on line 9 in listing 1. Unless you change the names of your local and/or remote repositories, this will do everything you need to do.

3 Afterword

Version control is hard. This isn't a function of version control software, but a function of the complexity of software, the nature of collaboration, and the sheer quantity of source code. However, it's essential for anyone who deals with different versions of documents and files. `git` is a popular, and reasonable, approach to version control. This very brief introduction just gets your little toe in the water, there's a vast ocean before you. Good luck, have fun versioning, and as always, read the friggin' manual.