Introduction to Git

Kevin Johnson

November 20, 2015

Contents

1 What Is Git?

Git is a version control system originally created a very long time ago by Linus Torvalds for use in development of the Linux Kernel. Today git is by far the most popular version control system for programmers. It provides utilities to allow users to track changes in a set of files over time. The main use case of git is for teams of programmers to work on a single code base without screwing everything up.

The basic idea behind git is you make a change to your code and the commit that change to the git repository for that project. Every time you commit a change to git it takes a snapshot of your code. Your files can be in three states: committed, modified, and staged. Committed means that the file is stored in the git database. Modified means that you have changed the file but you have not yet committed it to your database. Staged means that you have marked a modified file to be added to the database on your next commit.

A git repository consists of a git directoy, a working directory, and a staging area. The git directory stores the database for your project. It includes the entire history of all of your commits to that project. The working directory is a version of your code files that has been extracted from your databse for you to modify. The staging area contains information on what files will go into your next commit.

In most circumstances your workflow with git will consist of the following three steps:

1. Modify files
2. Stage files
3. Commit changes to database

That's really all you need to know to get started.

2 Installing Git

2.1 Windows

The official Git for Windows program can be downloaded from `http://git-scm.com/download/win`. It comes with a command line program that includes all of the necessary git-related commands. Frankly, if you're on Windows I would seriously consider switching to OS X or Linux. Pretty much everything related to data science works nicer on OS X or Linux. Windows can be a real pain sometimes.

2.2 OS X

If you're not already using Homebrew then I recommend you start now. It's a command line utility for installing software on OS X, modeled after the package management systems available in most Linux distributions. You can install it and install git with the following commands:

```
ruby -e "$(curl -fsSL \
    https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew update
brew install git
```

2.3 Linux

Simply use whatever package manager your distribution uses to install git. For Ubuntu this would be

```
sudo apt-get install git
```

3 Configuration

The first thing you should do after installing git is set a few global options. You need to tell git what your name is and where you can be contacted. The purpose of this is so that every change made to the code has a name attached to it along with an email so you can send them a message about how their code has ruined your life.

The following commands will set these options for you:

```
git config --global user.name "Kevin Johnson"
git config --global user.mail "kjohnson63@gatech.edu"
```

You can also set your preferred text editor if you want. If you don't then git will use your system's default text editor.

```
git config --global core.editor nano
```

4 Basic Commands

4.1 Initializing a Repository

The first thing you need to do when starting a new project is create the git repository that will store your code.

```
git init
```

When you run that command you will get a file structure that looks like this:

```
kjohnson@kjohnson-desktop:~/tmp$ tree -a --charset=ascii
.
`-- .git
    |-- branches
    |-- config
    |-- description
    |-- HEAD
    |-- hooks
    |   |-- applypatch-msg.sample
    |   |-- commit-msg.sample
    |   |-- post-update.sample
    |   |-- pre-applypatch.sample
    |   |-- pre-commit.sample
    |   |-- prepare-commit-msg.sample
    |   |-- pre-push.sample
    |   |-- pre-rebase.sample
    |   `-- update.sample
    |-- info
    |   `-- exclude
    |-- objects
    |   |-- info
    |   `-- pack
    `-- refs
        |-- heads
        `-- tags

10 directories, 13 files
```

You can safely ignore pretty much all of these files. Everything in the `.git` folder will be taken care of by your git commands.

You can also clone a current repository that you want to work on. This copies 100% of the data in the git database to your machine, including the full history of commits.

```
git clone https://github.com/kevjohnson/healthdata
```

4.2 Adding a File

Let's make a README file and add it to the repository.

```
touch README
```

Now that we have a file in our working directory we can use the `git status` command to check on the status of our repository.

```
kjohnson@kjohnson-desktop:~/tmp$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

For git to start tracking a new file we need to add it to the database using the `add` command

```
git add README
```

4.3 Making Changes to a File

Let's take a look at our status now.

```
kjohnson@kjohnson-desktop:~/tmp$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README
```

Our README file is now in the staging area ready to be committed to the database. Let's go ahead and do that now.

```
git commit -m 'Added README to repository'
```

The `-m` flag is a required summary message to go along with the commit. The standard guidelines say to keep this message under 50 characters. The purpose is to give a brief overview of what has changed in the code in this commit. If you do not include the `-m` flag then your text editor will pop up allowing you to create a more extensive commit message. See this blog post for more information on best practices for commit

messages. Some people are very passionate about high quality commit messages, but in reality most people write terrible commit messages.

We've now created a repository, created a file, and committed that file to the repository. This is all you need to know to get up and running right now using git. Everything else after this is just managing the results of what we've already done.

5 Useful Things to Know

5.1 Viewing the Commit History

You can view the history of commits for your project using the `git log` command.

```
kjohnson@kjohnson-desktop:~/tmp$ git log
commit a611d493b2d5e2e92c663fc0a813f0372dea91f7
Author: Kevin Johnson <kjohnson63@gatech.edu>
Date:   Mon Nov 15 10:54:07 2015 -0500

    Added README to repository
```

It will give you an ID for each commit along with the author, date, and commit message. As you can imagine the output of this command can get very complicated very quickly. To manage that there are a ton of options you can pass to control the output of the command. See here for full documentation. The only one I will mention here is the `-n` flag where `n` is the number of previous commits you want to see. For example, `git log -5` will show you the last five commits.

5.2 Undoing Your Mistakes

Let's say you push a commit to the repository but forget to include one of your files in that commit. You can use `git commit -amend` to add the files currently in the staging area to the previous commit.

```
touch script.py
git add script.py
git commit --amend
```

If you want to remove a file from the staging area (thus not including it in your next commit) you can use the `git reset HEAD` command.

```
touch badfile.py
git add badfile.py
git reset HEAD badfile.py
```

I recommend not worrying to much about reset and what it does. This example is the most common use case for reset so it's best to just remember it and move on. If you're really interested you can check this out.

5.3 Unmodifying a File

What if we modify README but then later decide we don't like our modifications? We can use the `git checkout` command to get the latest version of that file from the git database. This overwrites any changes we have made to it since our last commit.

```
git checkout -- README
```

5.4 Working with Servers

A typical git workflow involves hosting the repository on a remote server so all of your collaborators can connect to it and add/retrieve code. This can get a bit complicated so I'm just going to give you the basics to get started.

I'm going to use one of my own repositories as an example.

```
git clone https://github.com/kevjohnson/mymaps.git
```

When you clone a repository git automatically adds the server you cloned it from as origin.

```
kjohnson@kjohnson-desktop:~/tmp$ git remote -v
origin  https://github.com/kevjohnson/mymaps.git (fetch)
origin  https://github.com/kevjohnson/mymaps.git (push)
```

If you did not clone the repository you can add a remote server using `git remote add`.

```
git remote add origin https://github.com/kevjohnson/mymaps.git
```

When you're ready to add your changes to the remote server you can use the `git push` command. This takes all of the commits in your git database and sends them to the server. The server then takes the commits it doesn't already have and adds them to its own database.

```
git push origin master
```

Origin refers to the remote server that you want to push to, and master refers to the branch you want to push to. I wouldn't worry about branches too much as you're starting out. Just know that master is the name of the default branch in the repository and branches are used so people can work on features separately from the main code base. You can create a branch for a feature you want and then merge the branch back into master when you're done. This way you don't impact the code in the master branch while you're working. Again, just use master for everything for now.

If you want to get the latest code from the server you can use the `git pull` command.

```
git pull origin master
```

This does not overwrite any changes you have made to your version of the code. It will attempt to automatically merge the files from the server with your own files, but often you will have to manually go through and merge the files yourself.

If, for whatever reason, you want to burn everything to the ground and start over with the most recent version of the code on the server you can use a combination of `git fetch` and `git reset`. The `fetch` command is basically the same thing as `pull` except it doesn't attempt to merge differences with your local code.

```
git fetch origin
git reset --hard origin/master
```

6 GitHub

GitHub is the most popular website for hosting and sharing code using git. It provides free hosting for all of your code and built in tools to make working with git easier. I recommend creating a profile if you don't already have one and putting all of your work up on GitHub. GitHub often operates like a portfolio for programmers. Georgia Tech has an enterprise subscription to GitHub that gives you some extra features, most notably free private repositories. You can find it at `https://github.gatech.edu/`. GitHub also provides a GUI client for Windows and OS X which can be handy if you're not comfortable with the command line.