Learning the Git Basics

This information draws heavily from the Executive Program in Applied Data Analytics (www.applieddataanalytics.org/). If you are interested in an excellent course on data analytics, I highly recommend this class.

Table of Contents

1. Introduction

- back to Table of Contents

This notebook introduces you to using git for version control. The tutorial outlines the basic workflow for commmitting and syncing changes with a git remote repository. The github flow reviews a procedure for working on a group project.

a. Learning objectives

- Know the minimum set of git commands needed to use git for version control.
- Provide some additional commands and context so you better understand how to use git.
- Provide an overview of what git is and how it works.

b. Git Overview

The git overview will give a quick overview of what git is and how to get setup

You will need to have access to Git (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git) and Terminal (http://swcarpentry.github.io/shell-novice/).

Git in a nutshell

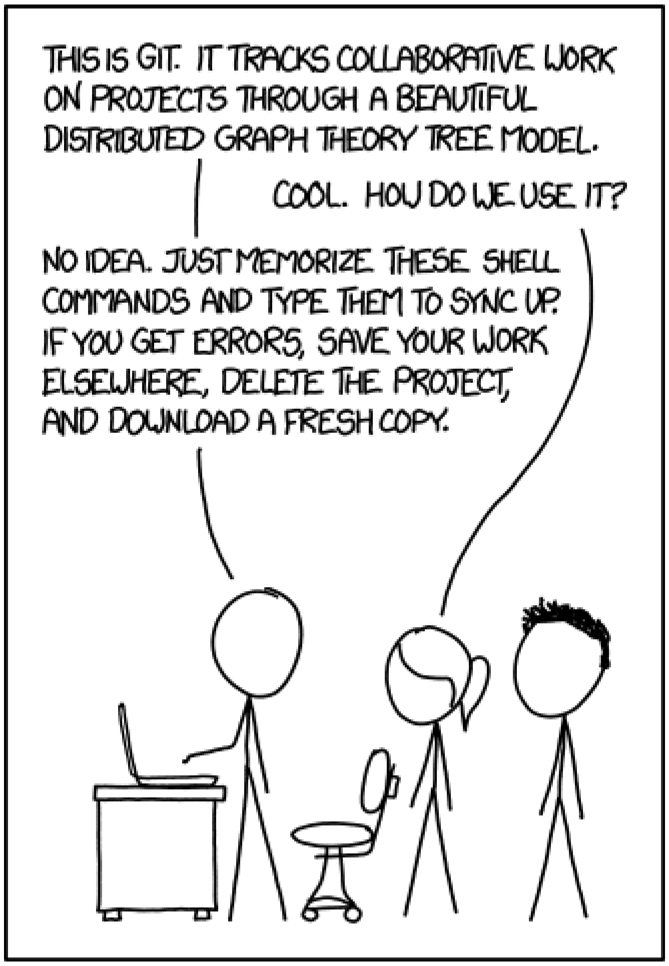

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Image source (https://xkcd.com/1597/)

Git is a *version control system* which helps you keep track of changes you make to files during the development of your project. You can think of it as an undo button, a lab notebook, and a tool to safely and efficiently collaborate with others on a shared project, all rolled into one. **All serious software projects use version control.**

While git is mostly used in software development, it can be used for anything you like writing books (https://www.gitbook.com/), for example, or even tableau files as long as your files are plain text (e.g., source code, $\LaTeX$ files).

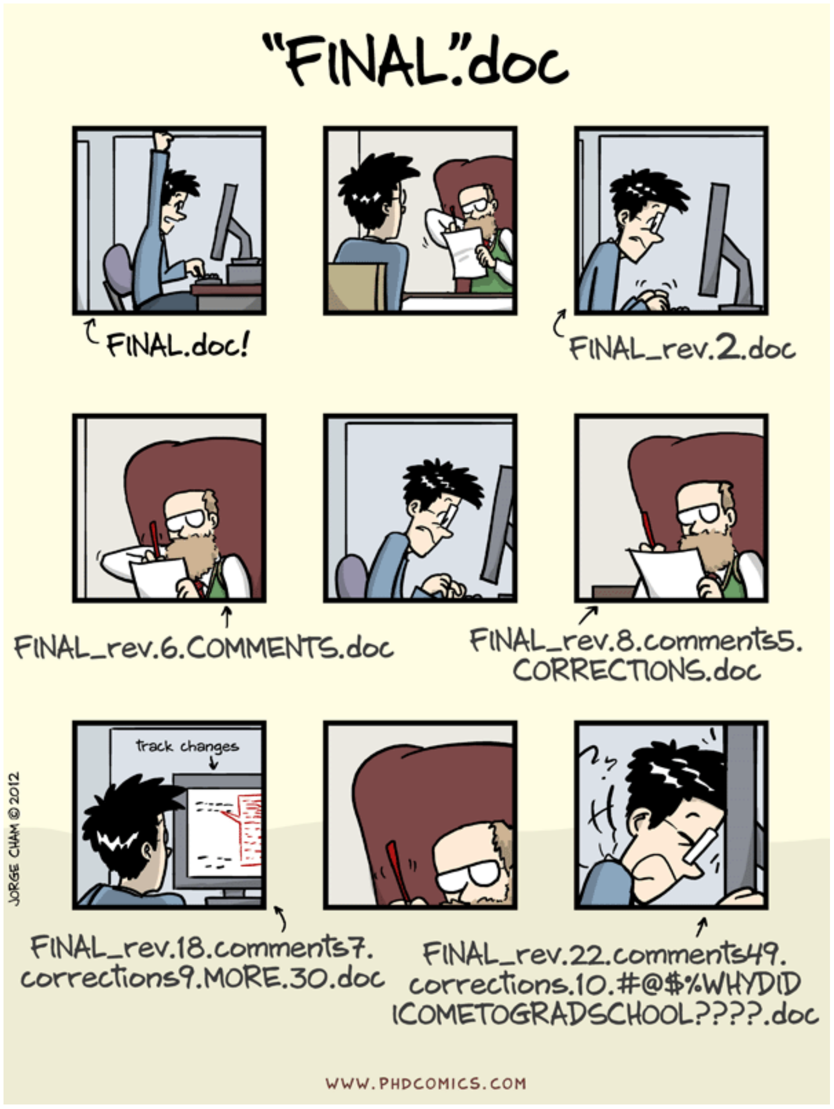
Simply speaking, git saves snapshots of your work called `commits`; after a `commit` is created, you can go back and forth through different commits in your project -- maybe you were experimenting with some new function and realized the old function was better, no problem, you can bring back anything! The collections of commits together with their associated metadata (like who made the changes, and when) form the `repository` of your project.

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Image source (https://xkcd.com/1296/)

The entire development of your project, the `repository`, is stored on your computer. but we know that's dangerous, so you can also host a `remote` copy on a hosting service like GitHub, Bitbucket, or GitLab. Hosting a project's `repository` on GitHub also allows for the distribution of your work and collaboration with others. This prevents endless emailing of source code and the following situation:


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDIDICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

git sounds awesome! How do I get it?

Chances are, git is already installed on your computer. To check, open up a terminal and type `git`. If not, you can get it here (https://git-scm.com/).

OS X users can use `homebrew` to install git.

Can I get buttons and stuff?

`git` is a command line tool, which means it doesn't have a native graphical user interface. Using the `git` CLI is the most flexible way of working with `git`, and if you are working on a remote server you will unlikely be able to use a GUI.

However, if you *really* want a point-and-click interface on your computer, here are some options:

- Options for Mac (https://git-scm.com/download/gui/mac)
- GitKraken (https://www.gitkraken.com/) (Windows and Mac)
- Github Desktop (https://desktop.github.com/)

Keep in mind that if you are logging into a remote machine, such as AWS, a GUI may not be an option.

Sidenote: How to Configure your Git Profile

First things first: you need to configure your `git` client, so that your commits are correctly attributed to you, and so you get pretty output. Do the following:

Open up a terminal: Go to your desktop on ADRF, left-click and select Open Terminal.

```
# How my git configuration currently looks
$ git config --list
```

switch the names to your information

```
# Adding some customization
$ git config --global user.name "Erlich Bachman"
$ git config --global user.email "E_bachman@aviato.com"
$ git config --global color.ui "auto"
```

Also do the following (important):

```
$ git config --global push.default current
```

This defines the action taken if no push references are given. (push the current branch to update the remote branch of the same name) You now have your `git` client configured. Next we will create our first repository.

2. Git Tutorial

This tutorial will walk through how to create a repository. You will add work to the repository. You will then commit this work. Finally, you will pull any new work that others working on the project have completed. Finally, you will push your commited work. **The git command** always is used by first calling git, then telling it what action you want to perform, then passing it additional arguments to tell it how you want it to carry out the requested action:

```
git <action> <parameters>
```

Example with action "add", adding "`my_file.txt`":

```
git add my_file.txt
```

Create a Git Repo

From a new project

```
cd ~\myproject
git init # intialize the repo
git add . # add the folder
```

From an existing repo

```
git clone git://host.org/myproject.git # an external GitHub Repo though HTTPS
git clone ~/some/repo.git ~/new/repo.git # through the filesystem
```

- Let's say our project is to work on the Not_Hotdog app. You need to first clone down the Not_Hotdog repository:

```
git clone https://github.com/ktpatrick/Not_Hotdog.git
```

First, go into the directory of the git repository in which you are working:

```
cd Not_Hotdog
```

> Sidenote: One of the things `git clone` does is create a `.git` directory. To see this, we can *list* all the files in this directory using the `dir` command. We include the `/a` flag to tell the command to display *hidden* files, or those that begin with a `.`, in the list.

```
dir /a
```

> We see that there is a `.git` directory in the `Not_Hotdog` directory. *Unless you really know what you are doing* **DO NOT EVER** *modify anything in this directory. If you delete this directory, the entire history of your project will be gone.*

- Use your programming skills to create the beginnings of the next big app and save that code in the Not_Hotdog repo.
- run git status to see what changes have been made:

  ```
  git status
  ```

- Add any files or directories that are new or have been changed:

  ```
  git add <file_name>

  git add <directory_name>

  git add README.md

  git add *.py   # you can use wild cards
  ```

- run git status now to see what's changed:

  ```
  git status
  ```

- Once you've added all the files, it has been "staged" to be committed. We can now make our first commit!

  ```
  git commit -m "Your sweet comment goes here"
  ```

  > As part of commit, if you don't include the -m paramenter it will ask you to enter a commit message. This will open up your default shell text editor.

- After commit, you sync with the github remote repository.
  - first pull, to receive changes that are on the server, not on your computer.

    ```
    git pull
    ```

  - then, push your changes to github

    ```
    git push
    ```

When you are collaborating with a team of developers, pulls sometimes force you to manually reconcile changes made to the same bits of code. If it is just you working alone in a repository, however, chances are your pull won't result in any changes or merges. It will just tell you there aren't any changes.

*When you push, depending on how you cloned your repository, you will likely have to log in to github.

The most basic procedure is then:

```
1. clone repository
2. do work
3. add
2. commit
3. pull
4. push
```

You'll see in a minute how to take this to the next level with github workflow. But before we do that, I'm going to cover a couple other useful git commands.

Stashing - Moving changes to the side

The git stash command lets you put aside a set of changes so that you can pull updated code from a remote. You can then either re-apply your stash of changes to the updated code files or discard them.

```
git stash -- save modified and staged changes and them remove them from current bra
nch.
git stash list -- list stack-order of stashed file changes
git stash apply -- reverts to most recent stash and keeps the stash
git stash pop -- reverts to most recent stash and drops the stash from the stack
git stash drop -- discard the changes from top of stash stack
```

Other Useful Commands

```
git <command> --help #pull up documentation for a <command>

git status -- check which files have been changed in the working directory

git log -- get a history of changes

git checkout <somefile> HEAD --revert to a the state of a file at the last commit

git reset --hard Revert back to the last state WARNING THIS CANNOT BE UNDONE

git rm <somefile> -- remove the file from your git repository and the file system,
 adds the removal to the next commit.

git mv <old_file_path> <new_file_path> -- move a file that is already versioned in
 git from one location to another, adding the change to the next commit.
```

3. GitHub Flow

So far we have been doing the "solo" workflow, which looks something like the following:

```
git clone https://github.com/ktpatrick/some-repo.git
touch some_file.py
pound the keys, do some work
git add some_file.py
git commit -m "Working with some awesome idea"
git pull
git push origin master
```

As you might have guessed, this workflow is just fine when you are working by yourself. But, when you're working in a team, it's useful to have a more structured workflow. Here I'll talk very briefly about the Github flow.

In the GitHub flow, we never code anything unless there is a need to. When something needs to be done, we create an issue on the GitHub repository for it. Good issues:

1. Are clear
2. Have a defined output
3. Are actionable (written in the imperative voice)
4. Can be completed in a few days (at most)

Here are some examples:

> Good: Fix the bug in ...
> Good: Add a method that does ...
> Bad: Solve the project
> Bad: Some error happened

Here is how to create a GitHub issue. (https://guides.github.com/features/issues/) Once an issue exists, we'll pull from the repo and create a branch. A branch is a copy of the code base separate from the main master branch where we can work on our issue (e.g, fixing a bug, adding a feature) without affecting the master branch during our work and then ultimately merge our change into the master branch.

The flow goes something like this:

1. Pull from the repo

```
git pull
```

2.Decide what you want to do and create a branch

```
git checkout -b meaningful_name_for_branch
```

The command git checkout -b creates a new branch (in this case called "meaningful_name_for_branch") and switches to that branch. We can see what branch we are on by using the command git branch, which displays all the branches in the local repository with a * next to the branch we are currently on.

branch - a branch is a set of code changes that are kept separate from the main code base (or trunk) in a git repository. A branch can be worked on in isolation until one wants to merge the changes back into the trunk. Git makes it easy to create branches both in your local repository and in a remote. Standard branches:

- master -- default development branch
- HEAD -- current branch
- HEAD^ -- parent of head
- HEAD~4 -- the great-grandfather of head

3.Try to solve issue

```
pound the keys,pound more keys, make some changes, add/rm files, commit
```

4.Push to the repo and create a remote branch

```
git push
```

5.Create a pull request and describe your work (Suggest/add a reviewer)

Here are more details on how to create a GitHub pull request (https://help.github.com/articles/creating-a-pull-request/)

6.Someone then reviews your code

7.The pull-request is closed and the remote branch is destroyed

8.Switch to master locally

```
git checkout master
```

9.Pull the most recent changes (including yours)

```
git pull
```

10.Delete your local branch

```
git branch -d meaningful_name_for_branch
```

This is just the tip of the git/github iceberg.

Github training videos (https://www.youtube.com/user/GitHubGuides)