# Week 1 Exercise: Git and GitHub

*Z620: Quantitative Biodiversity, Indiana University*

*January 16, 2015*

## Goals

- Learn about Version Control, Git, and GitHub
- Install and configure Git on your local computer
- Create an GitHub account using IU's GitHub Enterprise
- Learn what repositories (aka repos) are and how they will be used during Quantitative Biodiversity

## Version Control

One of the riskiest things to do with your data, manuscripts, and computing code is to lose track of various versions. This is easily done when renaming and emailing files, when storing files on drives that get lost or computers that get damaged, and when working in collaborative groups. Naming your files "Dissertation-v43-partA-AdvisorsComments-Rejected.txt" and emailing them between collaborators and committee members is madness! Beyond personal risk, funding agencies and many journals require authors to manage and provide all data, and can call upon authors to provide proof of reproducibility. Consequently, an increasing number of scientists, including those at CERN: https://github.com/cernops and NCBI: https://github.com/ncbi, are using an approach that has been used by tech companies for years, i.e., version control.

Version control is an approach to writing text, managing data, and developing computer code that allows users the ability to examine, comment on, and revert back to changes within the entire life of a document, and without being tied to any single computer. In addition, multiple people in remote locations can collaborate on the same text, code, and data without emailing copies back-and-forth and without losing or overwriting any changes. Likewise, a single graduate student can now step out of the stone age and professionally, cleanly, and safely manage all of their projects, while promoting their own research. Yay for version control!

### How it works, in a nutshell

In short, version control works by centralizing a project in a repository located on a server (e.g. a computer connected to the internet). The individual user never directly edits any code, data, or text in this online repository (aka repo). Instead, the user makes changes to a local version of the project (e.g. on your laptop) and then pushes those changes to the online version. All history of the online version is tracked and so, the entire history of a project is protected. Likewise, if the online version is directly changed or updated, e.g. by collaborators, then all the user has to do is pull the changes in from the online version. All this pushing, pulling, and deciding how different versions from different computers get merged together, is done by version control software. In this class, we will use the most popular and powerful version control software and services out there, i.e. Git and GitHub!

## Git and GitHub

**Git** is a free and open source version control system designed to handle everything from small to very large projects with speed and efficiency. Users install Git onto their local machines (e.g. laptop) and Git is basically the software that does the heavy lifting for you. Unlike many pieces of software you might be used to, such as internet browers, email software, scientific software, and iTunes, Git does not necessarily have a graphical user interface (GUI). In this class, we will work with Git directly through your computer's terminal window.

**GitHub** is a web-based service for hosting projects that use the Git version control system. GitHub provides an attractive user interface for viewing and managing a project's code, data, and text files, whether in collaboration with others or working individually. If your project is visible to others (public), then GitHub also serves as a way to let the world know about the awesome science you're doing (or even that you did it first) and even how to join in and share tools. While many companies, agencies, large laboratries, and governments use GitHub (https://government.github.com/), GitHub is also a great central location for your own personal projects. IU has an Enterprise GitHub system (https://github.iu.edu). That is, a version of GitHub restricted to IU faculty, staff, and students. During this class, we will primarily be using this version of GitHub.

**Very basic Git and GitHub Glossary:**

Like many technical and scientific tools, Git and GitHub have their own vocabulary. Some of the following terms are literally commands that you will type into the terminal window.

| Term | Meaning |
|---|---|
| ***Upstream*** | The central repository. For this class, 'upstream' is the version managed by your instructors. |
| ***Origin*** | Your own IU-GitHub version of a repository. Any versions on your local machine will *origin*ate from the origin. |
| ***Fork*** | Create a personal copy (or version) of a repository in one's own GitHub account. |
| ***Clone*** | Create a local copy of a respository on your computer. This is the version you will directly edit and make various changes to. |
| ***Fetch*** | Download changes that have been made to an online repository: 'fetch' brings in new changes but it doesn't merge them with your local copy. |
| ***Merge*** | Merge changes to the online version with your own local version. |
| ***Pull*** | = 'fetch' + 'merge'. 'pull' accomlishes both 'fetch' and 'merge' but give less freedom of control over either. |
| ***Staging Area*** | A file contained in your local repository. Git uses the file to store information about what you are changing in what files. |
| ***Add*** | After making changes to data, text, or code, and then saving the file as normal, you must 'add' your changes to the staging area. |
| ***Commit*** | Aftering 'add'ing changes, we need to commit them. This will saves a picture of what your files looked like at that moment. |
| ***Push*** | Having 'committed', this command will update your on-line version. |

**upstream**: The central repository. For this class, 'upstream' is the version managed by your instructors.

**origin**: Your own IU-GitHub version of a repository. Any versions on your local machine will **origin**ate from the origin.

**fork**: create a personal copy (or version) of a repository in one's own GitHub account.

**clone**: create a local copy of a respository on your computer. This is the version you will directly edit and make various changes to.

**fetch**: download changes that have been made to an online repository. 'fetch' brings in new changes but it doesn't merge them with your local copy.

**merge**: Merge changes to the online version with your own local version.

**pull**: = fetch + merge. 'pull' accomlishes both 'fetch' and 'merge' but give less freedom of control over either.

**staging area**: A file contained in your local repository. Git uses the file to store information about what you're changing in what files.

**add**: After making changes to data, text, or code, and then saving the file as normal, you must 'add' your changes to the staging area.

**commit**: Aftering 'add'ing changes, we need to commit them. This will saves a picture of what your files looked like at that moment.

**push**: Having 'committed', this command will update your on-line version.

## Git Installation

If you do not have a current Git installation (already installed in the computer lab), please do the following: ***how will they know? first step: check and see if Git is currently insalled***

1. Open a web browser and naviate to git-scm.com/download/
2. Select the appropriate operating system
3. The download should start automatically
4. Open the installer and follow the onscreen directions

**On Mac**: You will need to make sure you have Xcode Command Line Tools installed.
**On Windows**: This process will install Git Bash (msysGit). During installation, you will be asked to adjust your **PATH environment**. We recommend that you select the option to "Use Git from the Windows Command Prompt". This will give you the most flexibility with Git. In addition, we recommend that during installation you select "Use OpenSSH" for your secure shell client with GitBash.

During installation, you will be asked how to configure the line ending conversions **On Mac**: We recommend "Checkout as-is, commit Unix-style line endings" **On Windows**: We recommend "Checkout Windows-style, commit Unix-style line endings"

### Git Test

Before we get started with Git, we first need to test our current installation to make sure there aren't any issues. The easiest way to do this is to determine what Git version is currently installed. We will use **terminal** (GitBash on Windows) to accomplish this.

The first thing we need to do is find and start terminal. On the lab computers, you can find terminal in the **Utilities Folder** in the **Dock** at the bottom of your screen. On your personal computer: **Mac** you can search for terminal with spotlight [Cmd+Space]; **Windows** you can find GitBash in the Start Menu.

1. Find terminal (or GitBash) and open a new window
2. Type the following commands:

```
pwd
ls
git --version
```

## Git User Configuration

1. **Organization:** We recommend that you create a folder in your user directory (> cd ~) called '*GitHub*' to make this and future assignments easier to manage. (**Mac** Users: Do this from Terminal; **Windows** Users: Do this from GitBash)

   ```
   cd ~
   mkdir ./GitHub
   cd ./GitHub
   pwd
   ```

2. **User Configuration:** You will need to configure your local Git installation. We will do this by entering your name and email. We will also set two parameters: push.default and credential.helper

   ```
   git config --global user.name "John Doe"
   git config --global user.email johndoe@example.com
   git config --global push.default simple
   git config --global credential.helper store
   ```

3. The last thing you need to do is configure how Git handles line endings. Line endings are invisible characters that your operating system places at the end of each line in a document. On Unix machines (e.g. Mac), this is the linefeed character (LF). On Windows machines, this is the carriage-return (CR) and linefeed (LF) characters. This difference in line endings between Mac and Windows causes incompatabilities between the two systems. However, Git is enabled to handle the differences by silently converting line endings when repos are push to remote servers. We recommend that you configure this behavior in order to prevent any future issues when collaborating across computer platforms.

   **On Mac**

   ```
   git config --global core.autocrlf input
   ```

   **On Windows**

   ```
   git config --global core.autocrlf true
   ```

**You are now ready to *Git !!!***

## Create User with IU's Enterprise GitHub Service

1. Navigate to https://github.iu.edu
2. Sign in with your IU Username and Passphrase
3. On the top right of your screen, click on your Username
4. Click on the Edit Profile Icon to edit your profile

## Fork and Clone a Repo

1. Navigate to and click on your student repository (repo) on https://github.iu.edu/2015-QuantitativeBiodiversity
2. Fork your repo by clicking on the Fork Icon in the top right of your screen

   
   

   You should now see the repo on your GitHub page.
3. Clone the repo onto your local machine using the command line (terminal or GitBash). Replace "User_Name" with your IU Username and "Repo" with your QB Repository.

   ```
   cd ~/GitHub
   git clone https://github.iu.edu/User_Name/Repo
   cd ./Repo
   git status
   ```

The repo should have downloaded onto your local computer and the status should stay "all up to date". You should also see that the only thing in your repo is a file named README.md

4. Check and update remote repo. The following commands will at your **upstream remote repository**, which is located on the QB Course GitHub. Replace "Repo" with your QB Repository ***clarify?***

   ```
   git remote -v
   git remote add upstream https://github.iu.edu/2015-QuantitativeBiodiversity/Repo
   git remote -v
   ```

   You can copy and paste the URL for your upstream repo from the GitHub website.
5. Open and edit the README.md file:

   This file is a Markdown file. Markdown is markup language for writing and editing text that can be easily converted to other formats (e.g. HTML, PDF, Word). During this semester, we will edit Markdown files using RStudio. On the lab computers, you can find RStudio in the **Analysis & Modeling Folder**. From RStudio, navigate to and open your README.md file. Edit the file as needed (we will demonstrate). Update your '*Student Name*' with your full name. Enter your email address. Write a short Bio about yourself (~ 1 paragraph). List three to five course expectations. Hint: View the Markdown guide to learn about formatting and making ordered lists (https://guides.github.com/features/mastering-markdown/). When you are done, save the close the document.
6. Now we need to add and commit our changes to git. However, before we add anything we want to make sure that we are

   ```
   git status
   git add ./README.md
   git commit -m "Updated README.md with student information"
   ```
7. Now push the changes to GitHub. Before we push our changes, we always want to check for (fetch) and merge in any changes others have made.

   ```
   git fetch upstream
   git merge upstream/master
   git push origin
   git status
   ```

You should now see the repo, including your recent changes, on your GitHub page.

8. Navigate to your GitHub page to make sure that the file was uploaded correctly. If so, submit a Pull Request to submit your file to the course instructors.

   
   

   The course instructors can now merge and see your changes.

9. To get new assignments, you will pull (fetch & merge) your upstream repo. This will allow any updates your instructors have made to be merged with your local documents. In addition to pulling yoru upstream repo, you always want to push any updates to your origin.

   ```
   git status
   git fetch upstream
   git merge upstream/master
   git push origin
   git status
   ```

During this course, you will recieve and submit all assignments using these methods. In addition, you will use Git and GitHub to contribute to assignments in class and on your personal computers.