

Week 1: Git and GitHub

Z620: Quantitative Biodiversity, Indiana University

January 16, 2015

Goals

- Learn about Version Control, Git, and GitHub
- Understand how we will be using these tools in QB
- Install and configure Git on your computer
- Create an IU-GitHub account

Version Control

Mismanaging changes to data, manuscripts, and computer code is one of the most dangerous things you can do as a scientist. Yet, this is easily done when renaming and emailing files, when storing files in folders that get forgotten about or on drives that get lost or damaged, and when working in collaboration. Naming a file “Dissertation-Project-StatsCode-Final-v23.R” and then emailing it between people is madness! Funding agencies and many journals now require authors to provide well-managed data and can call on 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 and without losing or overwriting any changes. The individual 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 version control 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 the 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 pushes those changes to the online version. The history of the online version is tracked so the entire history of a project is protected. Likewise, if the online version is directly changed or updated, e.g. by collaborators, then the user merely pulls in the changes 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 projects of all sizes with speed and efficiency. Users install Git onto their local machines (e.g. laptop) but unlike some software you might be familiar with, such as internet browsers, Git *per se* does not have a graphical interface. Instead, we will work with Git 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 interface for viewing and managing a project’s code, data, and text files. 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 and even how to join in and share tools. While many companies, agencies, and governments use GitHub (<https://government.github.com/>), GitHub is a great central location to manage any project. In fact, IU subscribes to GitHub through an Enterprise GitHub system (<https://github.iu.edu>). This is a version of GitHub that is restricted to IU faculty, staff, and students. During this class, we will primarily be using the IU Enterprise version of GitHub.

Basic Git and GitHub glossary:

The terms below will be used throughout this class. Some are commands that you will type into the terminal window. All are defined with respect to how we will be using Git and GitHub in this class.

Term	Meaning
<i>Repository</i>	The collection of files and folders that compose a project.
<i>upstream</i>	Refers to the central repository, e.g., the version managed by your instructors.
<i>origin</i>	Refers to your on-line version of the class repository.
<i>fork</i>	Create a version of the class repository in your IU-GitHub account.
<i>clone</i>	Create a copy of 'origin' on your computer. This is the version you will edit.
<i>fetch</i>	Download changes that have been made to an online repository.
<i>merge</i>	Merge changes with your local version.
<i>pull</i>	'fetch' + 'merge'. 'pull' executes 'fetch' and 'merge' but give less freedom of control.
<i>Staging Area</i>	A file that Git uses to store information about your changes.
<i>add</i>	Having edited, removed, or added files, this command will add your changes to the staging area.
<i>commit</i>	Having added your changes, this command will save a picture of what your files looked like at that moment.
<i>push</i>	Having committed your changes, this command will update 'origin'.
<i>Pull Request</i>	Having updated 'origin', request that these changes be pulled into 'upstream'

Git Installation

If you do not have a current Git installation, please do the following:

1. Open a web browser and navigate 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 incompatibilities 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 your 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 receive 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.