

Lab 1.5: Introduction to Version Control in R

Updated January, 2016.

Table of Contents

What is Version Control?

All scientists use version control in one way or another, from data collection to manuscript preparation [1]. The “Track Changes” feature of your favorite word processor is an example of a version control system, commonly used in academic collaboration. Often, personal version control is ad-hoc and informal, where different revisions of papers, code, etc. are saved as duplicates with uninformative file names (e.g. *draft_1.docx*, *draft_2.docx*, *draft_final.docx*). Formal Version Control Systems (VCS), refer to dedicated programs or features that track the changes and revisions of a file in a standardized manner.

These systems generally fall into two families: centralized and distributed revision control systems. Centralized systems generally follow the server-client paradigm, where a user (client system) makes changes (i.e. “commits”) against a centrally stored copy (on a server somewhere). CVS and Subversion (SVN) are examples of centralized VCS. In distributed systems, all users keep a copy of the full version history of a project on their client systems. Users may still choose to maintain a central project authority, and in this case changes are made by moving local changes to the central repository (“pushing”) or by moving remote changes to the local repository (“pulling”). Bazaar, Git, and Mercurial are examples of distributed VCS.

Why Git?

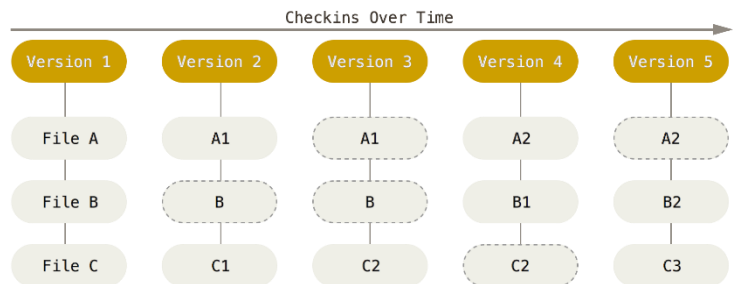
Different VCS have different strengths and weaknesses, and there exist many comparisons on the web [3] [4]. While different software projects and standards wax and wane in popularity over time, Git is currently the most popular and well-supported VCS. Originally created as a VCS for the Linux kernel development, many popular software projects have migrated to Git including: Android, Qt, Perl5, Python, and Swift. RStudio has built-in provisions to integrate Git with your R projects. Git is a very powerful tool with many levels of complexity, however in line with the Parteo principle, most of the power of Git comes from understanding a few core commands/principles.

Core Git Concepts

The definitive manual to Git can be found on the Git website [5]. A few core concepts from the manual are useful to understand and are reproduced below:

Git makes snapshots

Git thinks of its data more like a set of snapshots of a miniature filesystem (your project folder). Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a **stream of snapshots**.



Nearly every operation is local

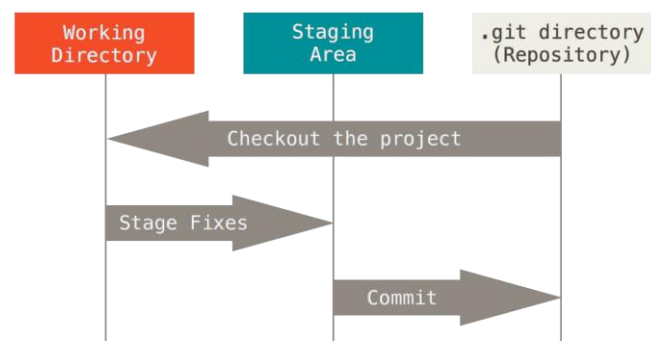
Most operations in Git only need local files and resources to operate – generally no information is needed from another computer on your network. For example, to browse the history of the project, Git doesn't need to go out to the server to get the history and display it for you – it simply reads it directly from your local database. This means you see the project history almost instantly.

Git has integrity

Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy. You can't lose information in transit or get file corruption without Git being able to detect it.

The three states

This is the main thing to remember about Git if you want the rest of your learning process to go smoothly. Git has three main states that your files can reside in: committed, modified, and staged. **Committed** means that the data is safely stored in your local database. **Modified** means that you have changed the file but have not committed it to your database yet. **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.



The Git repository for your project exists entirely in the single `.git` directory in your project folder (project root). This is where all metadata and internal structures are stored. You usually never need to edit the contents of this folder manually.

The basic Git workflow goes something like this:

1. You modify files in your working directory.

2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the Git directory, it's considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

Installing Git

To begin, download Git from the official source for Windows and OS X at the following URL: <http://git-scm.com/downloads>. Instructions for Linux users are also found at this URL.

Windows and OS X users can accept all defaults when installing Git.

Next, you will need to assign a name and email address that will be used to label your commits. When working in groups, it's best to use a unique name and email address, so that it is clear who made each commit. Open a Git shell and type the following:

(Note that you are not to enter the leading dollar sign into your terminal. The dollar sign just a convention for saying "enter the following into your terminal prompt".)

```
$ git config --global user.name "YOUR FULL NAME"
$ git config --global user.email "YOUR EMAIL ADDRESS"
```

You can double-check if your information was saved by running the following command in a git shell:

```
$ git config --global --list
```

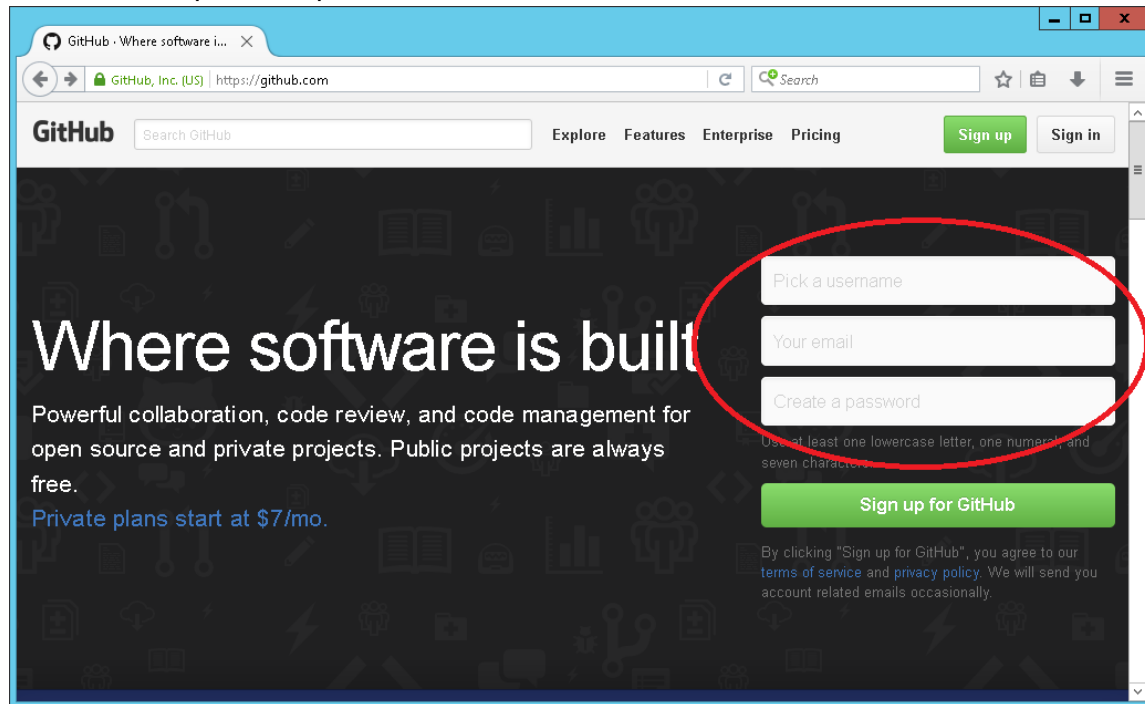
Choosing a Git host

As with other types of web services, there exist many web-based Git hosting services that can host a copy of your project repository online. This offers you a second location to copy your projects (a backup) and provides others with a location to upload (i.e. push) contributions back to your project. As of 2016, the three most popular web Git hosting services are GitHub [6], Bitbucket [7], and GitLab [8]. There also exist open-source tools for creating your own Git web host, on a server that you personally control. See GitWeb [9] and GitLab [8] for examples of popular, open source self-host tools.

For the purpose of this exercise, we will provide instructions for using GitHub as an online host. If you choose to use a different host, the local commands should be no different for you (except for changing certain URLs), however the web interfaces will be different and there may be some differences in terms used.

Make an account

Create an account on GitHub (<https://github.com/>). The free plan allows you to create as many projects as you want, provided that they are publicly accessible. GitHub offers free access to a basic personal plan for students (5 private repositories for 2 years). Bitbucket and GitLab allow for the creation of private repositories with their free accounts.



Generate an SSH key

Secure Shell (SSH) allows you to securely connect to your GitHub repository without having to enter a username and password every time you want to commit. SSH is also more secure than using username/password authentication and also enables you to do many, many other things that are beyond the scope of this course. In short, when you generate an SSH key, a public/private key-pair is generated. Your public key is analogous to a lock that is created to fit your private key.

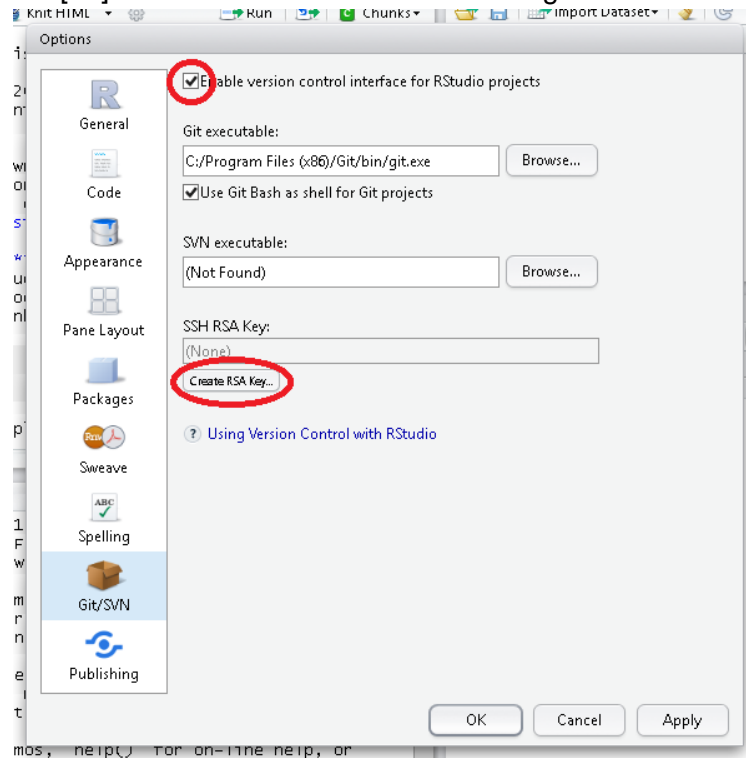
First check to see if you already have an SSH key-pair generated by running the following command in Git Bash, or your terminal:

```
$ ls -la ~/.ssh
```

If you see any file ending in .pub in the resulting output, then you already have a key. If the terminal complains about missing files/directories then you need to make a key.

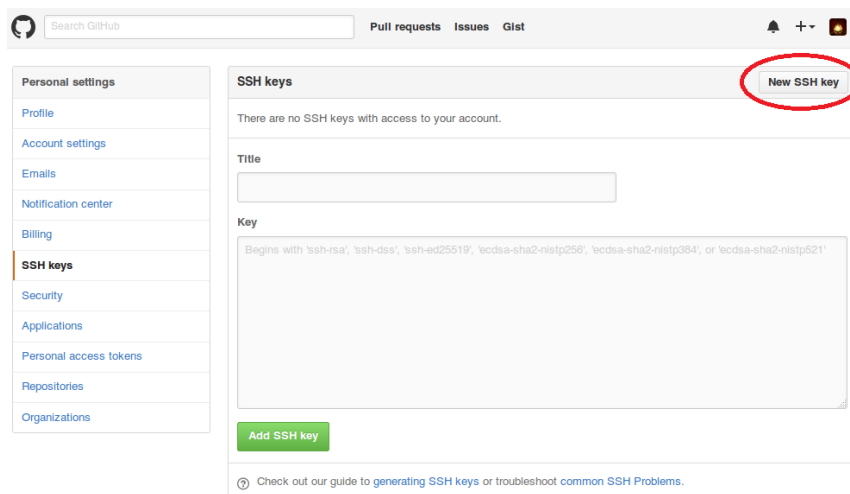
To generate an SSH key-pair, you can either follow the instructions on GitHub (<https://help.github.com/articles/generating-a-new-ssh-key/>), or you can have Rstudio make a key for you.

To have RStudio make the key for you, go to your menu bar and select **Tools** → **Global Options** . . . then go to **Git/SVN** and ensure that the top box is checked and that there is the “**Git Executable**” field is populated. Click “**Create RSA Key...**” and click “**Create**” *without* entering a passphrase [10]. Go ahead and close the following window.



Back in the Options window, click on the text that says “**View public key**”, and copy the entire contents of the following window to your clipboard.

Next, open a window in your web browser and navigate to the following URL: <https://github.com/settings/ssh> (or the SSH key settings of your preferred host). Paste the contents of your clipboard, your public key, into the “**Key**” field, and title it with something memorable. Press “**Add SSH Key**”.



You should now be ready to track projects with Git.

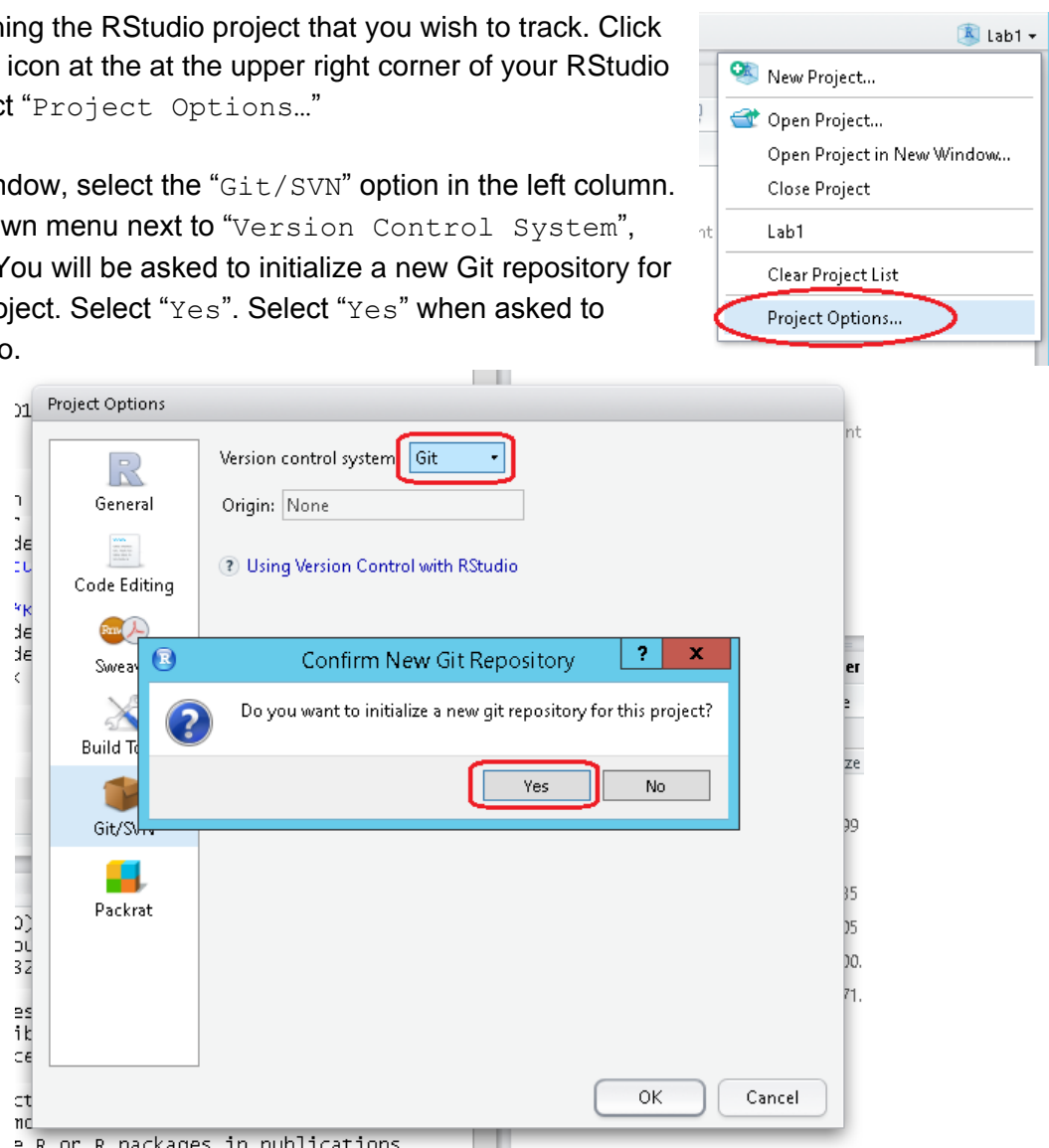
Linking RStudio Projects with Git

Now that we have installed Git, configured RStudio, and exchanged credentials with GitHub, we can now use Git to track changes in an existing R project, and synchronize changes with GitHub.

Initializing a Git repository with RStudio

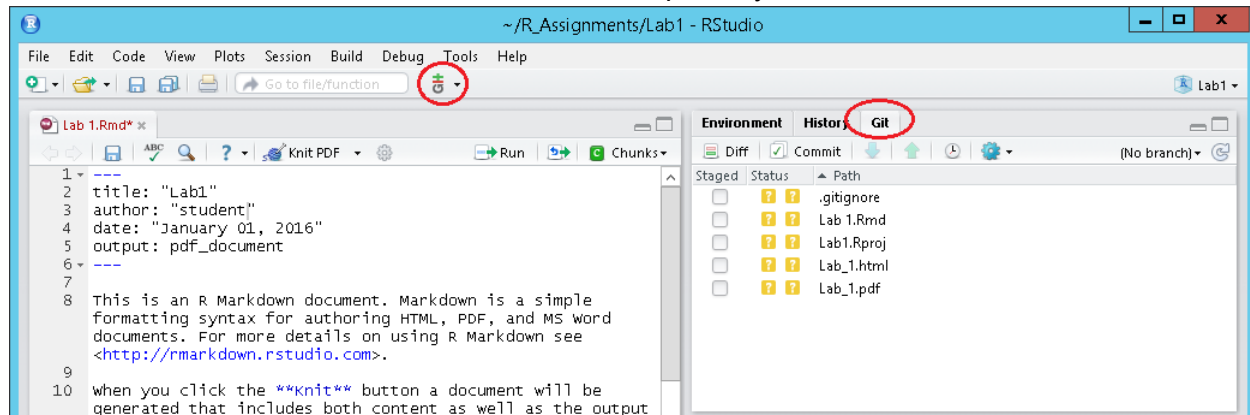
Begin by opening the RStudio project that you wish to track. Click on the Project icon at the upper right corner of your RStudio window. Select “Project Options...”

In the next window, select the “Git/SVN” option in the left column. In the drop-down menu next to “Version Control System”, select “Git”. You will be asked to initialize a new Git repository for the current project. Select “Yes”. Select “Yes” when asked to restart RStudio.



You should now notice a couple of additions to the RStudio environment. The first is a new pane that tracks the status of changed files in your R project in relation to your local Git repository.

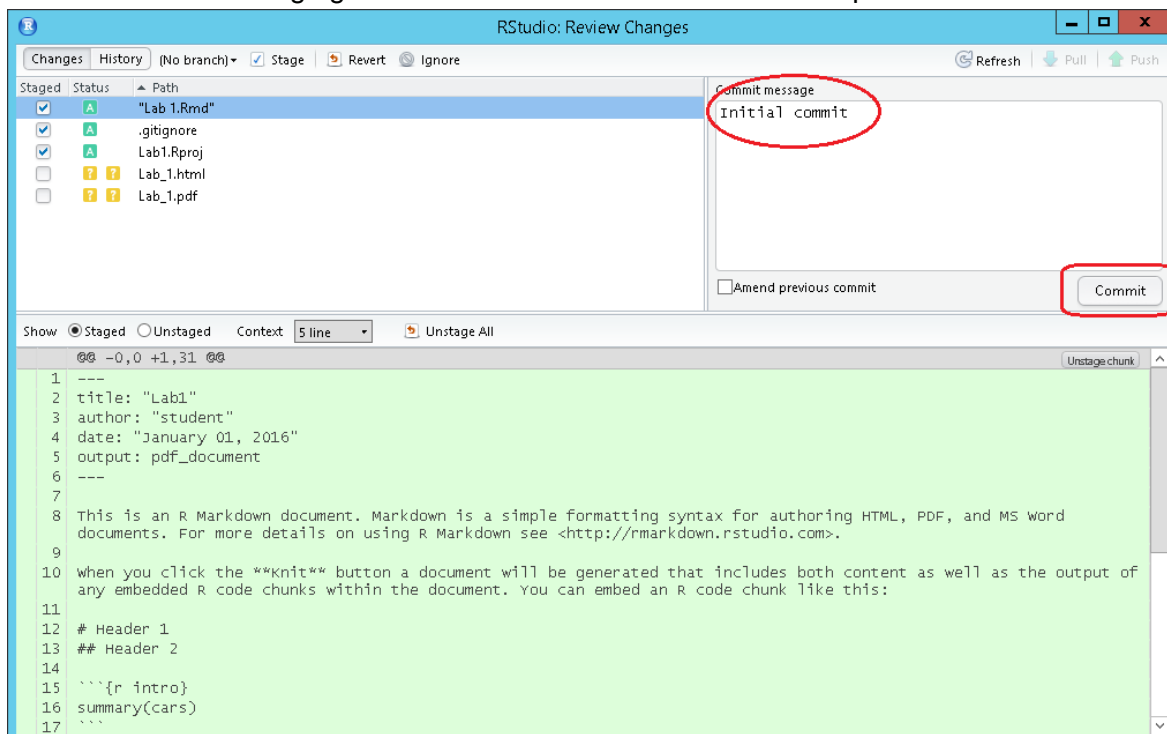
Currently, the yellow question marks beside each file in your project signify that the files are **untracked** in Git. Let's commit them to our local repository.



Check the box next to every file that you would like to commit. Let's start with your lab RMarkdown file, your .Rproj file and your .gitignore file (if you have one). The icon next to each file now should change to a green A, to indicate that the files are now **added** to your staging area.

Now let's commit this snapshot of our workspace to Git as our first commit. Save your RMarkdown file, and click on the "Commit" icon in the Git pane.

The following window allows you to review changes before you record the current changes to your local repository. Here, you can add additional files to your current commit/staging area, and remove files from the staging area. Let's add a useful comment and press "Commit".

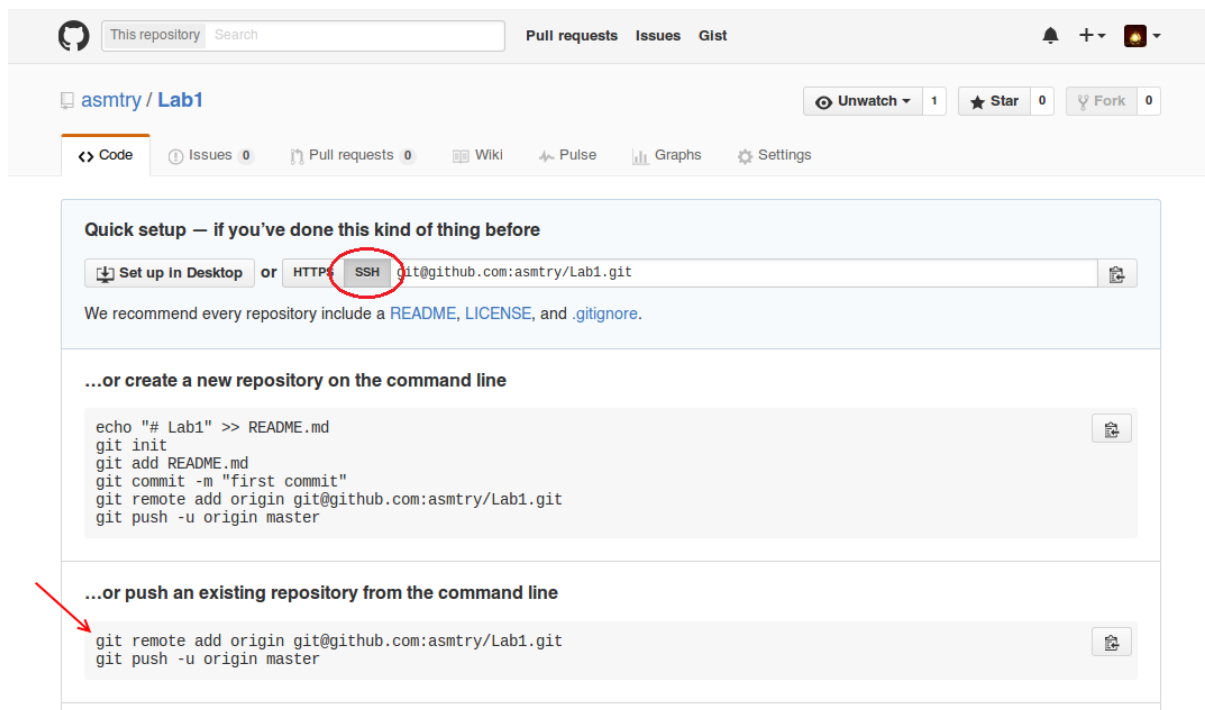



You have successfully created your first commit in Git! To view a history of previous commits, you can click on the History (clock) icon in the Git pane.

Pushing your local repository to a web remote

Now, let's send the initial snapshot of our local repository to GitHub. First, navigate to <https://github.com/new> and create a new repository. For consistency, give the GitHub repository the same name as your R project and create a description if you wish. Ignore the other options for now and click "Create Repository".

In the following screen, make sure to select SSH (see the image below), to generate the appropriate commands to enter into Git Bash (or your terminal). In our case, we are going to push an existing repository to GitHub from the command line. Note that you will only need to do this step once.



Switch back to RStudio and navigate to the Git pane. Click on the  icon and select "Shell..." This will either open a shell (via Git Bash or your terminal emulator) in your repository's main directory. Enter the two commands from the GitHub new repository page into the command prompt (see the arrow in the above image).

The first command tells Git that your local repository has a remote version on GitHub and the second command pushes all of your current work to the remote repository. *Note: You may be asked if you want to connect to the host 'github.com' in a prompt similar to the following:*

```
The authenticity of host 'github.com (<some IP address>)' can't be
established.
RSA key fingerprint is <some long hex string>.
```



```
Are you sure you want to continue connecting (yes/no)?
```

If you are shown this prompt, type `yes` and press enter.

You have now pushed your first commit to GitHub! Note that two new icons have appeared in the Git pane of RStudio. Now that you have established a Git remote, these icons allow you to push changes (and pull changes) directly from RStudio without having to use the command line.



A good resource for more detail on the integration of Git, Github, and RStudio, is this document written by an RStudio developer (<http://r-pkgs.had.co.nz/git.html>) [11]. A PDF version is included with this exercise.

Homework Exercise (4 Points total)

These exercises rely on the lab data found in the resource section. Download `Trees.csv` from CatCourses and save to your R workspace.

Your document should have the following sections, and provide written explanations formatted in R Markdown that explains your code, output and graphics in the following format:

NAME
CLASS
DATE

Homework Assignment 1

Objective Statement: [What are you trying to accomplish?]

Methods: [In general terms, what analyses are you doing?]

Data: [What are the data and where did they come from?]

Code: [In specific terms, what is the code that was used to conduct the analysis?]

Results: [What do the results show? Numerical evidence and graphic evidence are required.]

Discussion: [What do the results mean?]

Limitations: [What are the limitations, caveats, and assumptions of the analysis?]

External Libraries in R

R Packages and Libraries

Packages are a collection of R functions, data and pre-compiled code. The location where the packages are stored are called Libraries. One of R's strengths is its ability to dynamically install packages from online repositories. The Comprehensive R Architecture Network (CRAN) is the

most common R repo that includes many packages that are well documented and mostly built on statistical publications.

Installing Packages

Use the `install.packages` function to install new packages. For instance, if you wish to install the `ggplot2` package, you would execute the following:

```
install.packages("ggplot2")
```

This will download the relative files for the package and it will become available for use in R.

Importing Libraries

For using libraries from packages, they must be imported using the `library` function. After installing the `ggplot2` package, you can import it by executing the following:

```
library("ggplot2")
```

Functions from the `ggplot2` library are now available.

Regression Analysis

Regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables. More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed. Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables – that is, the average value of the dependent variable when the independent variables are fixed. Less commonly, the focus is on a quantile, or other location parameter of the conditional distribution of the dependent variable given the independent variables. In all cases, the estimation target is a function of the independent variables called the regression function. In regression analysis, it is also of interest to characterize the variation of the dependent variable around the regression function which can be described by a probability distribution[3].

Descriptive Statistics

Descriptive statistics is the discipline of quantitatively describing the main features of a collection of information,[1] or the quantitative description itself. Descriptive statistics are distinguished from inferential statistics (or inductive statistics), in that descriptive statistics aim to summarize a sample, rather than use the data to learn about the population that the sample of data is thought to represent. This generally means that descriptive statistics, unlike inferential statistics, are not developed on the basis of probability theory.[2] Even when a data analysis

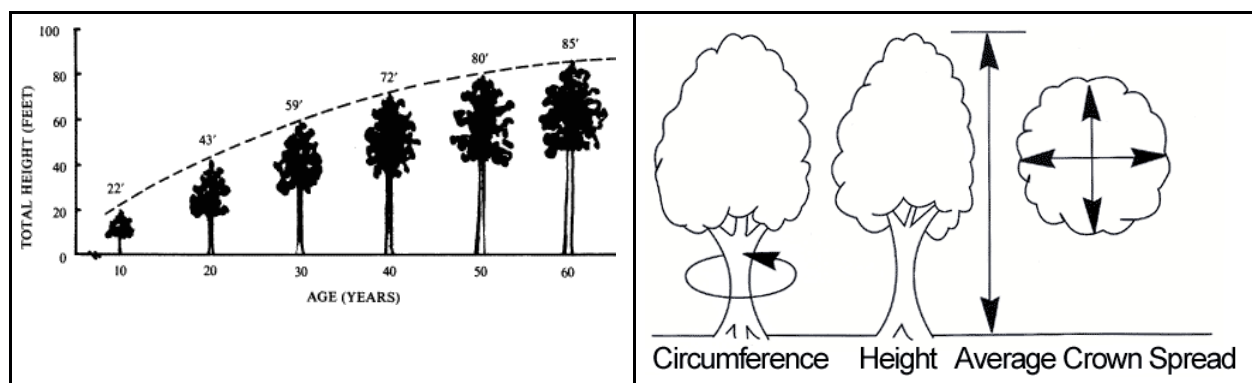
draws its main conclusions using inferential statistics, descriptive statistics are generally also presented [2].

OBJECTIVE

Understanding the dimensional relationships among objectives is key to making inferences about unmeasured characteristics. An emerging area of environmental data analysis is in Carbon Markets to offset GHG emissions. To better estimate standing carbon in forests, allometric equations are used to estimate biomass (ie volumes) of trees. Your objective is to understand the nature of conifers in the Sierra Nevada using measurements extracted from LiDAR (tree crown heights and radii). Your objective is report out the descriptive statistics of the dataset and determine if tree heights are positively or negatively correlated with crown radii. Test the null hypothesis that there is zero correlation. Advanced users are challenged to create a map that shows the relative position and sizes of tree crowns [Pro tip: use the `ggplot2` package].

The objective of Homework 1.5 is to further analyze `Trees.csv` data to determine if a linear model is sufficient to predict crown radii as a function of tree height. Use a subset of the data as a working model to develop a regression analysis to better understand any variability in the data. Fitting the linear model to these subsetted data, describe the fit of the model using diagnostic measures, and describe any assumptions or caveats to using this model for the rest of the data set.

As before we will be using the `Trees.csv` data of lidar to examine point data, but then we run through additional diagnostics with plotting and use Git for versioning.



Step 1 - DEVELOP WORKFLOW and PSEUDOCODE; IMPORT DATA FRAME

Critical to any successful data analysis is sketching out your workflow. Develop pseudocode as appropriate. What are the critical steps? Use the template to form an outline and comment your steps as you go using `#comment`.

R has many built in functions for importing data, the easiest of which is loading 'comma-separated value' text files. These data are often denoted by `.csv` file extensions. Load `Trees.csv` into a matrix using the `read.csv` function. (Hint: You can get help with the `read.csv` function using the help flag `?`) Print the dimensions `dim()` and structure `str()` of the data frame. Do the data seem to be correctly formatted?

Step 2 - RUN DATA DIAGNOSTICS

R has a built in summary function that is useful for obtaining a quick summary of a data set, but to better understand how these built in functions work, step through some diagnostics on your own. The following are some hints:

```
#What are your data?
data(Trees)
dim(Trees)
ncol(Trees)
nrow(Trees)
str(Trees)
head(Trees)
tail(Trees)
#Explore Trees[,2:5] columns as x
#min(x), mean(x), median(x), max(x), range(x), iqr(x), etc.
```

R also has a histogram function for visualizing data distributions.

```
hist(Trees[,1]) #first column is just ID, but you get the point...
```

Create data summaries, data distribution histograms for tree heights and crown radii. Does the data appear to meet assumptions of normality? Create log transformed histograms and reexamine.

Step 3 - EXAMINE CORRELATION BETWEEN VARIABLES

Find the (or “Pearson”) correlation for tree heights and crown radii and create a pairwise plot for tree heights and crown radii. The plot should have a xy scattergram as the main figure, and boxplots on margins like Figure 3.18 in Kabacoff text. What is the r value? Use `cor.test` to determine the probability of rejecting the null hypothesis.

Step 4 - CREATE A FUNCTION TO SPEED UP ANALYSIS

You can create your own functions in R. Use the following template to create a function:

```
function_name = function(x) {
  #do something, such as square a passed value of x
  sqr = x^2
  return(sqr) #if applicable, return something
}
```

Create a function similar to the summary function called EDA (exploratory data analysis) that accepts a list of values as an input and outputs the following in a human readable way:

- Minimum
- Mean
- Median
- Maximum
- Range
- Standard Deviation

- Coefficient of Variation

Additional functionality, such as including plots and transformations could embed this function within an R script. We'll save R scripts for next time.

Step 5 - Subsets

Sometimes it is useful to work with a subset of your data. Why? Import `Trees.csv` into R Studio and take a subset (see `?subset`) using the following limits, which are in meters (m) as per the Universal Transverse Mercator (UTM) coordinate system:

- $708000 < x < 708200$
- $4334000 < y < 4334200$

What property of the data are we taking a subset of? Why could this be a good representation for all the data? Summarize the new dataset (similar to Homework 1) including histograms and plots and compare to your results in the first homework; is the subset really a good representation of all the data?

Step 6 - Spatial Plotting

Produce a visual of the subset by plotting their locations (using `x` and `y`) as dots with representations of their heights and crown radii (all on the same plot). It is best to use the more advanced plotting function, `ggplot`, where color is a factor of height and the size of the dot is a function of crown radius.

Step 7 - Descriptive Statistics

Install and import the “`psych`” library (see above about installing packages). Analyze the subset using the `describe` function from the `psych` library. How does this compare to the summary function? What are some new metrics given by `describe` and what do they mean?

Step 8 - Regression Modeling

Regression modeling can be used to find data tendencies. This is often visualized by plot this tendency against your data (also known as a trend line). The simplest model (and probably the most used) is a linear model. Create a pairwise plot for tree heights and crown radii. Create a linear model using the `lm` function for heights and crown radii and add it to your plot. (See `?abline` for adding lines to an existing plot or use `par`).

Install and import the `car` library. Find residuals using the results from your linear model and display a summary. Summarize homoscedasticity using the `ncvTest` and `spreadLevelPlot` functions. What do these values mean (residual and homoscedasticity)? How could they be useful? Create diagnostic plots for the linear models and describe them.

Step 9 - Outliers

Detect Outliers using the `car` library and explain the results. What are possible reasons for the outliers? (Advanced users are encouraged to highlight the outliers in a data plot).

Step 10 - Commit to Git

Resources

Data

- Trees.csv on CATCOURSES (Lab Data→Lab1)

References

- [R Reference Card](#)
- [R Operators and a few Functions](#)
- [An Introduction to Data Analysis and Graphics with R](#)

Citations

- [1] Ram, K. Git can facilitate greater reproducibility and increased transparency in science. Src Code for Bio Med (2013).
- [2] Atlassian: [What is Version Control: Centralized vs. DVCS](#)
- [3] Google Scholar search: [comparison of Git, Mercurial and Bazaar](#)
- [4] Search engine search: [comparison of Git Mercurial and Bazaar](#)
- [5] [Pro Git – git-scm.com](#)
- [6] <https://github.com/>
- [7] <https://bitbucket.org/>
- [8] <https://gitlab.com/>
- [9] <https://git-scm.com/book/en/v2/Git-on-the-Server-GitWeb>
- [10] The point of an SSH key passphrase is to protect your private key in the event that someone is able to read your key. However, if someone is able to read your private key, you have bigger issues to worry about!
- [11] Git and GitHub - <http://r-pkgs.had.co.nz/git.html>

Keywords

R, RStudio, RMarkdown, cor, plot, ggplot, knitr, lm