Git - First Steps

- Clone the repository for this presentation:

  `git clone https://github.com/steviep42/BrownBag`

- You could also download the folder as a ZIP file if you don't want to mess with git at this time.

- This creates a folder on your hard drive called "BrownBag" which contains a folder called "GIT". Go there and checkout a file called "git_commands.txt" It has the command I will be typing in the live part of this presentation.

- You might also like to go to https://www.youtube.com/user/biorsph/playlists and select the "Git" playlist

Git - Managing Change

Git - Managing Change

git --distributed-is-the-new-centralized

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git - Obtaining and Installing - Windows

* Installing git on Windows is easy. Go to http://msysgit.github.io then download and double click the installer.

* After that you will have both a command line version and a GUI as well as a SSH client that you will need to interact with services such as GitHub and BitBucket.

* As part of the installation you will also get a BASH emulator that behaves similarly to the BASH shell on Linux systems (where git was first developed).

* This is useful if you wish to run git from the command line though most people want to use the GUI.

* Windows users can also right-click on a folder in Windows Explorer to access the BASH or GUI

Git - Obtaining and Installing - Apple

- Installing git on Apple computers is also pretty easy. Go to

- http://git-scm.com/download/mac then download and double click the installer.

- Since Apple computers have a built-in UNIX operating system you can crank up the Terminal application to enter git commands.

- Of course most people will want to use a GUI in which case you can head over to https://mac.github.com/ to get a copy of **GitHub**.

Git - Tracking Change

* Git can help you manage changes to your projects that involve the development of code for an assignment, thesis, or a research project.

* Do you write code in SAS, R, Perl, Python, SPSS, Java, C, C++, FORTRAN, Ruby, or any other language ?

* Do you write "perfect" code upon your first attempt ? If so then please tell us how you do it.

* If you are like most people you fumble around a while trying different things before you get something that "kinda works" and then you spend more time improving it. Even after you are "done" then someone asks you to change it to do something different. And so on and so on.....

Git - Motivations for Using

* Example case. I was once asked to write a function to perform a mathematical operation called "convolution" on two input vectors in a way that was more efficient than commonly available methods.

* The reference algorithm is well known and can be found in many texts and on Wikipedia.

* It took some number of changes, (maybe 10), for me to code up the slow version of the algorithm and debug it to the point where it produced the expected results.

* This was version "1.0" and served as the benchmark. My goal was to tweak or replace the algorithm to make it work much faster.

Git - Motivations for Using

* So then the real work began as I tried out different approaches and "tricks" to get the algorithm to work faster even if the input was huge. This took many iterations like maybe 70 in all before I found something that looked promising.

* After testing it on lots of different input I committed to the approach and then make lots of changes to do error checking and validation of input. Let's say another 15-20 changes.

* Then I went back and wrote in comments that would be useful for a user of the code to know about. Another 10 or so changes.

Git - Motivations for Using

* Which of these files represents the final version ? Who really knows ? I don't !

```
$ ls
FFT notes2.doc                   conv2.R.exp
RProg.pdf                        conv2.R.exp.GOOD
conv.R                           conv2v1.R
conv1.R                          conv2v1.R.DONT_LOSE_THIS
conv2.R                          conv2v2.R
conv2.R.BESTSOFAR                conv2v3.R
conv2.R.BEST_with_for_loops      convolveV.R
conv2.R.GOOD                     convolveV.R.v1
conv2.R.REALLYGOOD               fn.R
$
```

* Okay I could figure it out in a few minutes but it's better to work out of one file , (or a fixed set of files), and use a tool like git that monitors changes over time. That way when I come back years after the fact and know what's going on

Git - Motivations for Using

* It might be better for the file structure to look like this:

```
$ ls
Readme.md          convolution.R    test.data
$ 
```

* One source file, (but you can have more if you need them), with all changes to that file being tracked over time.

* A "readme" file that describes things

* Maybe some test.data (as long as it isn't huge)

Git - History
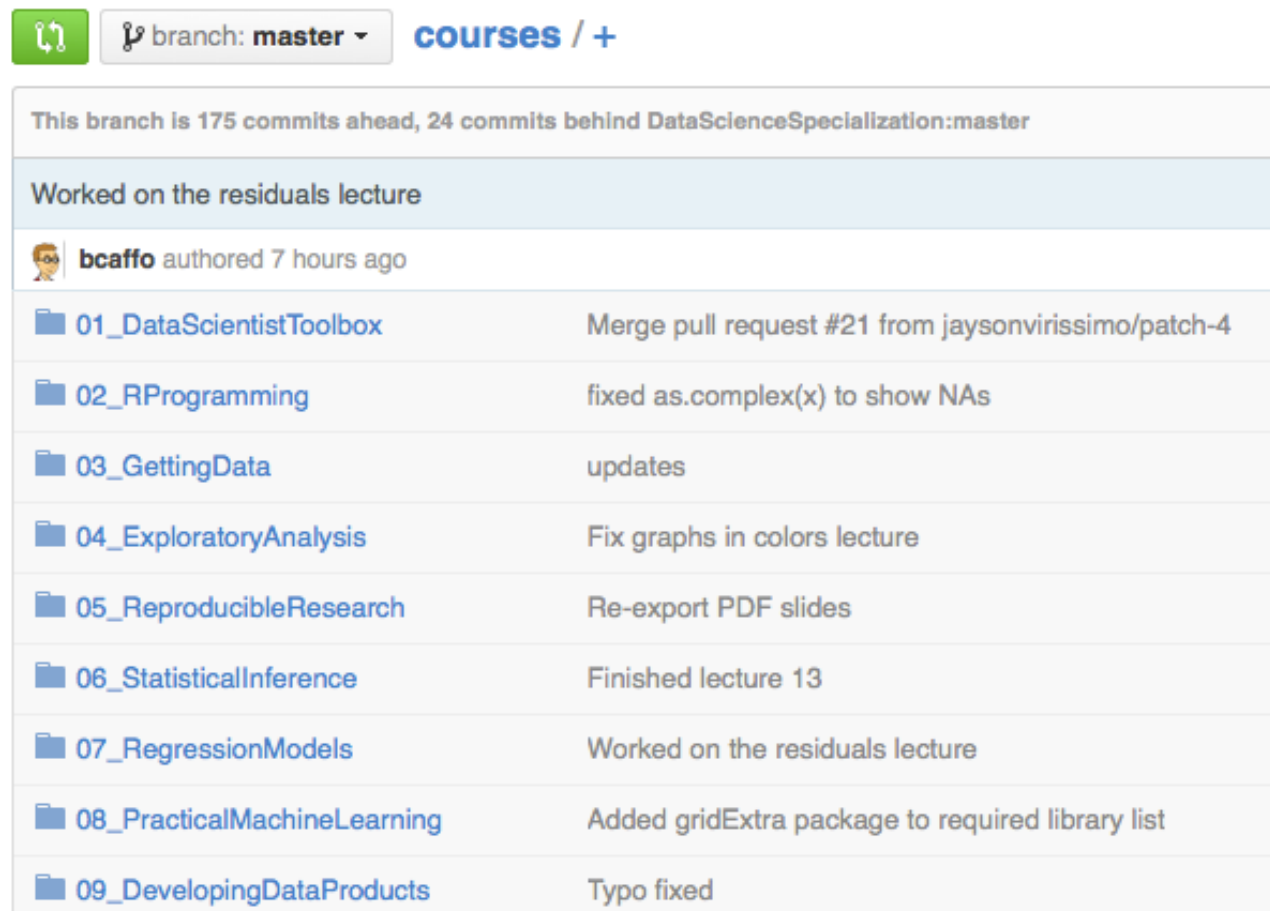
*   Originally developed for Linux kernel development in 2005

*   Used to track changes to documents and program code over time with the ability to "retreat" to earlier versions or "branch off"

*   Enables collaboration: more than one person can work on a shared project and then "merge" their changes into a "production" version.

*   Efficient handling of very small or very large projects

*   Can be used on all major operating systems either from a "command line" or from a graphical client

Git - Use Scenarios

* You are taking a class and the teacher/author has directed you to GitHub to **fetch** a **git repository**. (Like Edx, Coursera, or some open source project).

* On your laptop you are developing some programs for a class project or your thesis. This can be languages such as SAS, R, SPSS, C, C++, Java, Python, etc. The code changes over time and you want to capture revisions with the ability to "back out" of changes that introduce bugs in your code.

* You want to share your code with others via a repository and have them make changes or develop new code that can then be integrated back into the repository.

* You develop slightly different versions of your code over time in response to new assignments or requests so you want to create different "branches" for each version.

Git - Pulling in Courses

Coursera hosts eight courses targeted to Data and Statistical Analysis. These are created by Faculty at Johns Hopkins. The material for all 8 courses is maintained as a Git repository on Github.

Git - The Rise of Remote Repositories

1. The Data Scientist's Toolbox
2. R Programming
3. Getting and Cleaning Data
4. Exploratory Data Analysis
5. Reproducible Research
6. Statistical Inference
7. Regression Models
8. Practical Machine Learning
9. Developing Data Products

Git - Pulling in Courses

You can clone this repository using the same approach as above (although it's big ~1.7 GB)

```
$ git clone https://github.com/bcaffo/courses
Cloning into 'courses'...
remote: Counting objects: 6769, done.
remote: Compressing objects: 100% (4578/4578), done.
remote: Total 6769 (delta 2003), reused 6751 (delta 1990)
Receiving objects: 100% (6769/6769), 900.33 MiB | 1.19 MiB/s, done.
Resolving deltas: 100% (2003/2003), done.
Checking connectivity... done.
Checking out files: 100% (8093/8093), done.
$
$ du -sh courses
1.7G    courses
$ 
```

Git - Need for Backup

- Note that **git** does a great job of managing changes to your code but it is **NOT** a backup system.

- Some people create repositories on their DropBox folder.

- Others create repositories on shared drives.

- Others will **push** their git repository to a remote service such as **GitHub** and/ or **Bitbucket** where the code is backed up and can also be downloaded by others.

Git - Key Concepts

- **REPOSITORY** - A Folder that has been placed under git control.

- **ORIGIN** - Every repository folder has an "origin", which could be on a remote server like gitHub or BitBucket or another user's folder.

- **BRANCH** - Think of a tree. It has a "trunk" and "branches" off of the "trunk". In git the "master" branch is the "trunk". You can create branches off of the trunk to do experimental work without changing the master branch (trunk)

- **TRACKING -** Tracking files is a way to have git observe all changes to files you edit/create/remove. "Tracked" files are files you feel are "close" to being useful and ready for sharing with others.

- **COMMIT -** Commits are done when you have a file or set of files that are ready to be "committed" and shared with others. Each project will usually have multiple commits over time. There are no limits on the number of commits.

Git - Key Concepts

- **REPOSITORY** - A Folder that has been placed under git control.

At it's most basic this is a folder/directory on your local hard drive that you have placed under git control by using either the "git init" command or the "git clone" command.

```
$ git init TestRepos
Initialized empty Git repository in /Users/fender/TestRepos/.git/
$
$ git clone https://github.com/steviep42/Cool_Software.git
Cloning into 'Cool_Software'...
remote: Counting objects: 29, done.
remote: Total 29 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (29/29), done.
Checking connectivity... done.
$ 
```

The folder contents remain intact and can be altered using any operating system command.Repositories can be shared with others or kept private. You are not obligated to share it but many people do.

Git - Key Concepts

- **ORIGIN** - Every repository folder has an "origin", which could be on a remote server like gitHub or BitBucket or another user's folder. The origin could simply be a folder you created.

Use the "git remote" command to see if your repository is from some place on the Internet. You probably already know if it is or isn't but if you have forgotten then this example will help:

```
$ git clone https://github.com/steviep42/Cool_Software.git
Cloning into 'Cool_Software'...
remote: Counting objects: 29, done.
remote: Total 29 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (29/29), done.
Checking connectivity... done.
$
$ cd Cool_Software/
$
$ git remote -v
origin  https://github.com/steviep42/Cool_Software.git (fetch)
origin  https://github.com/steviep42/Cool_Software.git (push)
$ 
```

Git - Key Concepts

- **BRANCH** - Think of a tree. It has a "trunk" and "branches" off of the "trunk". In git the "master" branch is the "trunk". You can create branches off of the trunk to do experimental work without changing the master branch (trunk)
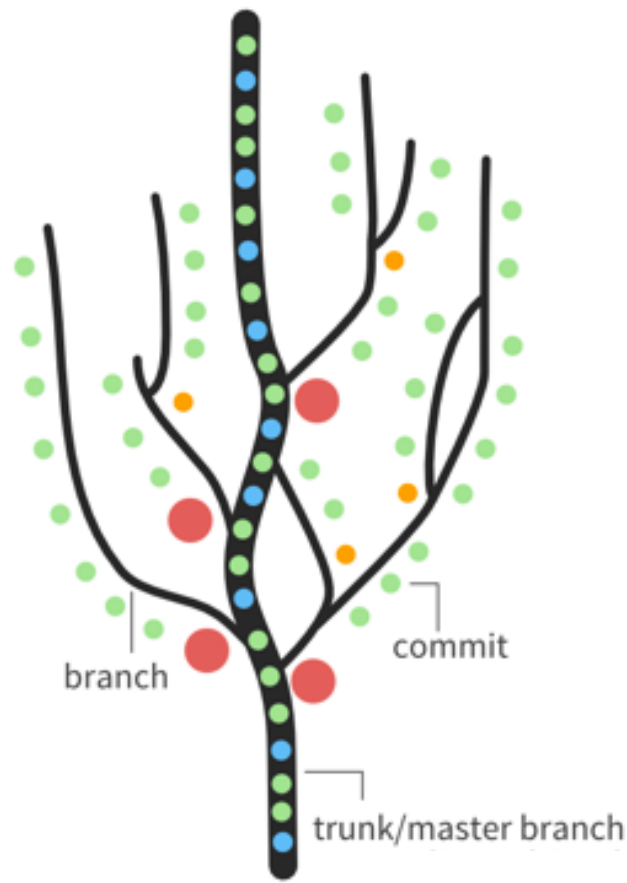
Git - Key Concepts

- **TRACKING -** Tracking files is a way to have git observe all changes to files you edit/create/remove. "Tracked" files are files you feel are "close" to being useful and ready for sharing with others. Use the "**git add**" command to track files.

```
$ ls
Assignment_1     README.md        Script1.R        Script2         test
$
$ git add Script2
$
```

Git - Key Concepts

- **COMMIT -** Commits are done when you have a file or set of files that are ready to be "committed" and shared with others. Each project will usually have multiple commits over time. There are no limits on the number of commits.

- Use the **"git commit <filename>"** or **"git commit -a"** commands (or variation thereof) to accomplish the commit:

```
$ git add Script2
$
$ git commit Script2
[master da04fb7] I made a commit
 1 file changed, 1 insertion(+)
$
```

Git - Common git commands

To create a repository: `git init`

To get a copy of a repository:
`git clone /path/to/repos`
`git clone https://github.com/steviep42/Cool_software.git`

To pull changes from a remote repository:
`git pull`

To add/stage files to the repository:
`git add <filename(s)>`

To commit files to the repository:
`git commit -m "Commit Message"`

To see what git "thinks" about things:
`git status`

To list commit history:
`git log pretty=oneline`

Git - Workflows

The Basic Git workflow goes something like this:

1) Create the repository (use "git init" or "git clone")

2) You modify/create/edit files in your working folder/repository

3) You add file(s) which allows you to track changes to the files.

4) When you feel your files represent a logical point of progress then commit them.

Repeat 2-4 many times until your project is complete or at a logical stopping point.

Git - First Steps

- Clone the repository for this presentation:

      git clone https://github.com/steviep42/BrownBag

- This creates a folder on your hard drive called "BrownBag" which contains a folder called "GIT". Go there and checkout a file called "git_commands.txt" It has the command I will be typing in the live part of this presentation.

- You might also like to go to https://www.youtube.com/user/biorsph/playlists and select the "Git" playlist

Git - First Steps

- One of the first things you should do is configure your name and email so git will know who you are. It will use this information when recording changes.

- While your userid is arbitrary make it something relevant as you will probably use it to set up a GitHub or BitBucket account at some point in the future.

```
git config –global user.name "Your Name"
git config –global user.email "My Email"
```

```
$
$ git config --global user.name steviep42
$ git config --global user.email ticopittard@gmail.com
$
$ git config --get-all user.name
steviep42
$ git config --get-all user.email
ticopittard@gmail.com
$
```

Git - Cloning a Remote Repository

- Many people don't know about git until they are directed to use it by a teacher, a collaborator, or a research paper.

- Usually you are asked to to clone a repository from GitHub or BitBucket, which are services that allow people to register and distribute code.

- "repository" is just a fancy word for a folder structure.

- Usually you will be given a URL for the repository.

Git - Cloning a Remote Repository

- Usually you will be given a URL for the repository. As an example let's say I directed you to **clone** one of my repositories. Here is an actual URL:

  https://github.com/steviep42/Cool_Software.git

- We'll just enter the command:

git clone https://github.com/steviep42/Cool_software.git

Git - Cloning a Remote Repository

```
$ pwd
/Users/fender
$ git clone https://github.com/steviep42/Cool_software.git
Cloning into 'Cool_software'...
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 15 (delta 2), reused 12 (delta 1)
Unpacking objects: 100% (15/15), done.
Checking connectivity... done.
$ cd Cool_software/
$ ls -a
.                  ..                  .git          README.md          Script1.R
$
```

Notice that once we **clone** the repository we have a folder called **Cool_Software** that contains some files: 1) an R program file and 2) a README file.

Git - Cloning a Remote Repository

Well that was okay but what are the advantages of this approach ? For starters you could modify the files you've just cloned and, given permission, you could **push** the changes up to the repository such that when other users clone the repository then your changes would be captured.

```
$ ls -a
.                    ..                    .git            README.md        Script1.R
$ git remote -v
origin   https://github.com/steviep42/Cool_software.git (fetch)
origin   https://github.com/steviep42/Cool_software.git (push)
$
```

Notice that when we are in the folder we can use the git remote -v command to see what remote repositories we have available.

How does git know this ? Well if you look in the Cool_Software folder you will see a sub directory called .git that has lots of information in it.

Git - Cloning a Remote Repository

If you look in the **Cool_Software** folder you will see a sub directory called **.git** that has lots of information in it. It's a good policy not to mess with these files except maybe for the **description** file which will talk about in a few slides.

```
$ ls -a
.                    ..                 .git              README.md           Script1.R
$ more .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/steviep42/Cool_software.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
$
```

Git - Cloning a Remote Repository

- This also implies that you can have any number of active git repositories on your local machine, (or a remote machine), although it is best to start out with only one or two until you get some experience.

- It is important to understand that for each folder that has a git repository there will always be a subfolder within that repository folder called **.git** which contains important info. Note that you should never modify files in this folder by hand.

- You can look at some of the files such as **config** or **description.**

- Once we have cloned a repository we can pull down any changes that have been made to it. As an example let's say that I updated the Cool_Software repository by adding a class assignment into it. (I'll show how I would do this later).

- Instead of having to re download everything again we can simply type **git pull** to get only the new additions and changes.

Git - Pulling in Updates

- So you will now see a file called **Assignment_1**

```
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/steviep42/Cool_software
   8cb842a..aa241df  master      -> origin/master
Updating 8cb842a..aa241df
Fast-forward
 Assignment_1 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Assignment_1
$ ls
Assignment_1     README.md        Script1.R
$
```

- Had there been no available updates we would have seen something like:

```
$ git pull
Already up-to-date.
$
```

Git - Creating a Local Repository

- While you can fetch remote repositories it's also quite common to create them on your personal computer. Some prefer to create repositories within their DropBox folders so that the content is backed up.

- In the following examples I am using Apple OSX but this sequence of commands would work on Linux or a Windows machine that has git installed.

- We create a folder called **MyProject**

- Next we initialize a git repository. Note that the **.git** folder is there but aside from that there are no other files or sub folders.

```
$ mkdir MyProject
$ cd MyProject/
$ git init
Initialized empty Git repository in /Users/fender/MyProject/.git/
$ ls -a
.       ..       .git
$
```

Git - Creating a Local Repository

- Within the **MyProject** folder let's create a file called **Readme.txt.** After that if we do a **git status** we'll see that the file's presence has been noticed ! You are being "watched" but don't be paranoid !

```
$ vi Readme.txt
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

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

        Readme.txt

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

- Note that we are on the **branch master**, which at this time is the **only branch available.** The results of **git status** also tells us that if we want for git to manage Readme.txt then we'll need to **add** it (also known as **staging).**

Git - Creating a Local Repository

- We **added** the file and the **status** command now tells us we have some changes waiting to be **committed.**

- This means that git has **staged** the file but has yet to **commit** it into the repository.

```
$
$ git add Readme.txt
$ git status
On branch master

Initial commit

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

        new file:    Readme.txt

$
```

- We could **unstage** the addition if we wanted to by doing **git rm —cached Readme.txt** But let's commit the file.

Git - Creating a Local Repository

- We accomplish the commit by doing **`git commit -a`** which then prompts us to enter a comment after which we get conformation.

- We can also do a **git log** command to see what our commit history looks like.

```
$ git commit -a
[master (root-commit) 91efc2b] This is my first commit.
 1 file changed, 1 insertion(+)
 create mode 100644 Readme.txt
$
$ git log
commit 91efc2ba416487bbe13dbf5da900b695b52d8de8
Author: steviep42 <ticopittard@gmail.com>
Date:   Fri Aug 8 22:43:58 2014 -0400

    This is my first commit.
$
```

- We can see the exact date and time of the update as well as who did the commit in addition to the accompanying comment.

Git - Creating a Local Repository

- Next we'll create another new file called **program.R** and then issue the **git status** command to see what git thinks is going on.

```
$ vi program.R
$ more program.R
data(mtcars)
mylm <- lm(mpg~wt,data=mtcars)
$
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        program.R

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

- So **git** let's us know that we have a new **untracked** file called **program.R** This is similar to what happened before. See you are fast becoming an expert !

Git - Creating a Local Repository

- So let's add the new file. We'll use the wildcard character ***** to indicate this. When we do **git commit -a** it will prompt us to enter some comments.

```
$ git add *
$ git commit -a
[master 74daa42] Adding in an R program
 1 file changed, 2 insertions(+)
 create mode 100644 program.R
$
$ git log
commit 74daa425f6caccd5d308bf41cf4cf9d4a84fface
Author: steviep42 <ticopittard@gmail.com>
Date:    Fri Aug 8 22:51:07 2014 -0400

    Adding in an R program

commit 91efc2ba416487bbe13dbf5da900b695b52d8de8
Author: steviep42 <ticopittard@gmail.com>
Date:    Fri Aug 8 22:43:58 2014 -0400

    This is my first commit.
$
```

Git - Creating a Local Repository

- **git log** has many possible formats to make viewing easier.

```
$ git log --pretty=oneline
74daa425f6caccd5d308bf41cf4cf9d4a84fface Adding in an R program
91efc2ba416487bbe13dbf5da900b695b52d8de8 This is my first commit.
$
$ git log --pretty=format:'%h %cd %s (%an)'
74daa42 Fri Aug 8 22:51:07 2014 -0400 Adding in an R program (steviep42)
91efc2b Fri Aug 8 22:43:58 2014 -0400 This is my first commit. (steviep42)
$
```

- In the second example the numbers on the left are known as "**hash numbers**" that are associated with a specific commit.

- These are unique identifiers which make it possible to retrieve a previous version of our repository (if needed).

Git - Creating a Local Repository

- We certainly don't need to add and commit files one at a time. We can add many files including folders all at once.

- Think of staging files as being a "pre-commit". Staging files let's you track changes to files without committing the changes to the repository. You can always un stage files.

- Keep in mind that working inside a **git** repository is no different than working inside any other folder or directory.

- While it is true that **git** is monitoring changes to files it doesn't do anything to them. It just notes that you have changed a file.

- **Git** doesn't validate or debug your code. In fact it doesn't really care what language you are using. It's job is to help manage changes over time.

Git - Creating a Local Repository

Git can show differences between the current contents of a file vs it's content at the time of a previous commit (or staging). The **Readme.txt** file has the following:

> **This is a readme file. It is our first addition to the repository**

Edit this file to add another line after it:

> **I'm adding another line**

Git will of course notice that we have made changes but we can also use git to show us the differences:

```
$ git diff
diff --git a/Readme.txt b/Readme.txt
index 75dd3dc..abe1dcf 100644
--- a/Readme.txt
+++ b/Readme.txt
@@ -1 +1,2 @@
 This is a readme file. It is our first addition to the repository
+I'm adding another line
$
```

Git - Branching Out

- One of the best features of git is that you can create **branches** of your production code. Remember that we currently have only one branch which is named **master.**

```
$ git branch
* master
$
```

- Think of branches as being "safe copies" of your committed code that you can experiment with without fear of corrupting the **master branch,** which is where the "good stuff" is.

- If you wind up liking the results of your experiment then you can merge changes back into the master branch.

- If you don't like the results of your experiment then you can simply switch back to the master branch and delete the experimental branch.

- You can create as many branches as you want but it's best, in my opinion anyway, to work with as few as possible.

Git - Branching Out

- The **git branch** command lists all currently available branches.

- **git branch <branchname>** creates a new branch.

- **git checkout <branchname>** will checkout the contents of the master branch into the branch named **<branchname>**:

```
$ git branch
* master
$
$ git branch experiment
$ git branch
  experiment
* master
$ git checkout experiment
Switched to branch 'experiment'
$ ls
Readme.txt      program.R
```

Git - Branching Out

- Let's **add** and **commit** another file here called **program2.R** Remember that we are currently in the **experiment** branch

```
$ vi program2.R
$ git add program2.R
$ git commit -a
[experiment b0aea66] Added a second program to do linear modeling using the di
amonds data frame from the ggplot2 package
 1 file changed, 5 insertions(+)
 create mode 100644 program2.R
$
$ ls
Readme.txt        program.R        program2.R
$ git branch
* experiment
  master
```

- What next ? Well we can continue to test our new program and maybe even add a few more. We can delete files if we want.

- Remember that whatever we do in the experimental branch will NOT impact the **master** branch. We can switch back and forth between the two.

Git - Branching Out

- We can switch back to the **master branch** at any time without losing our work in the **experimental** branch.

```
$ git branch
* experiment
  master
$ ls
Readme.txt        program.R        program2.R
$
$ git checkout master
Switched to branch 'master'
$ ls
Readme.txt        program.R
```

- So observe that when we switch back to the **master branch** we don't see the **program2.R** file since we didn't yet **merge** it back into the **master** branch.

Git - Branching Out

- Don't worry we can easily merge in those changes from the **experiment** branch

```
$ git branch
  experiment
* master
$
$ ls
Readme.txt        program.R
$ git merge experiment
Updating 205865e..b0aea66
Fast-forward
 program2.R | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 program2.R
$ ls
Readme.txt        program.R        program2.R
```

- Notice we are in the **master** branch. We issue the merge command referencing our **experiment** branch, which brings on the changes we made in that branch. We only made one - adding in the file program2.R

https://github.com/steviep42/BrownBag/   Steve Pittard wsp@emory.edu

Git - Branching Out

- Once we have merged in the changes we can then delete the **experimental** branch if we want. But we don't have to.

- In fact we might want to keep it around for more testing

- Even if we do delete it we can create one or more new branches for future testing.

```
$ git branch
  experiment
* master
$
$ git branch -d experiment
Deleted branch experiment (was b0aea66).
$
$ git branch
* master
```

Git - Cloning a Local Repository

- If you are working on a multiuser computer, (e.g. a Linux, Apple, or Windows computer), then other users can clone your repository.

- You could also make the repository available via DropBox or network sharing.

- Note that the user must have appropriate access permissions to see the folder.

- As an example let's say that user **guest** wants to clone the **MyProject** repository on my Apple system. They login and do:

```
Guest$ pwd
/Users/Guest
Guest$
Guest$ git clone /Users/fender/MyProject
Cloning into 'MyProject'...
done.
Guest$ cd MyProject/
Guest$ ls
Readme.txt      program.R      program2.R
Guest$
```

Git - Using GitHub

Git - Using GitHub

- So if you want to share your work with the world, (or a subset thereof), then you can use free services such as those offered by GitHub and BitBucket.

- These services also provide backup. There is also support for a builtin Wiki to document things and solicit input from users.

- Getting a GitHub account is free as are repositories ~~BUT only **public** repositories are free. If you want privacy you have to pay a token amount.~~

- Go to https://github.com/ and sign up. If you have already selected a **git** userid locally then use that if possible when creating your **github** userid. Once you have done that you are ready to go.

Git - Using GitHub

Additionally there are some very cool free features if you host your repository on GitHub.

- Wiki
- Activity graphs (shows download activity)
- Issue Tracker (let's people flag issues in your code)

The Wiki capability alone is great since you and your collaborators/users can openly document ideas and examples of the code for all to see.
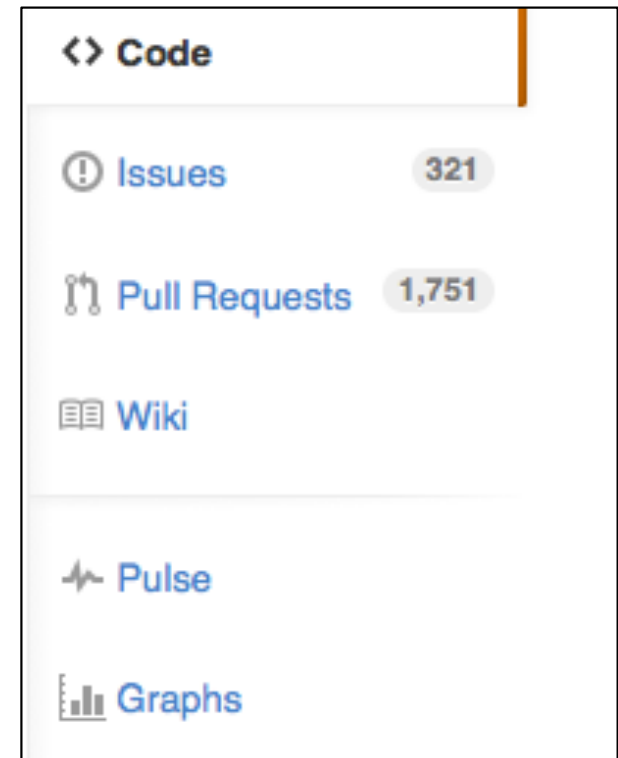
Git - Using GitHub - Create a Remote Repository

Log in to github and you will see a screen like the following. At this point you want to create a repository so click the "+" character in the upper right corner.
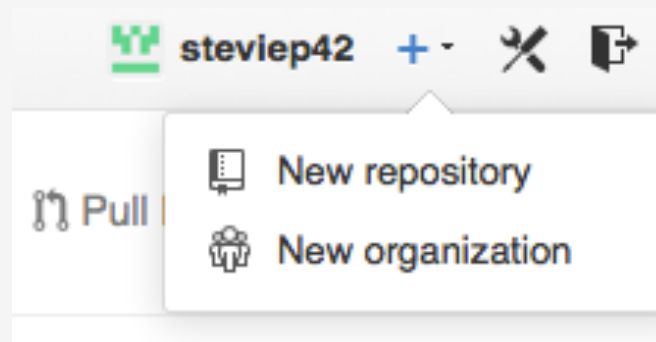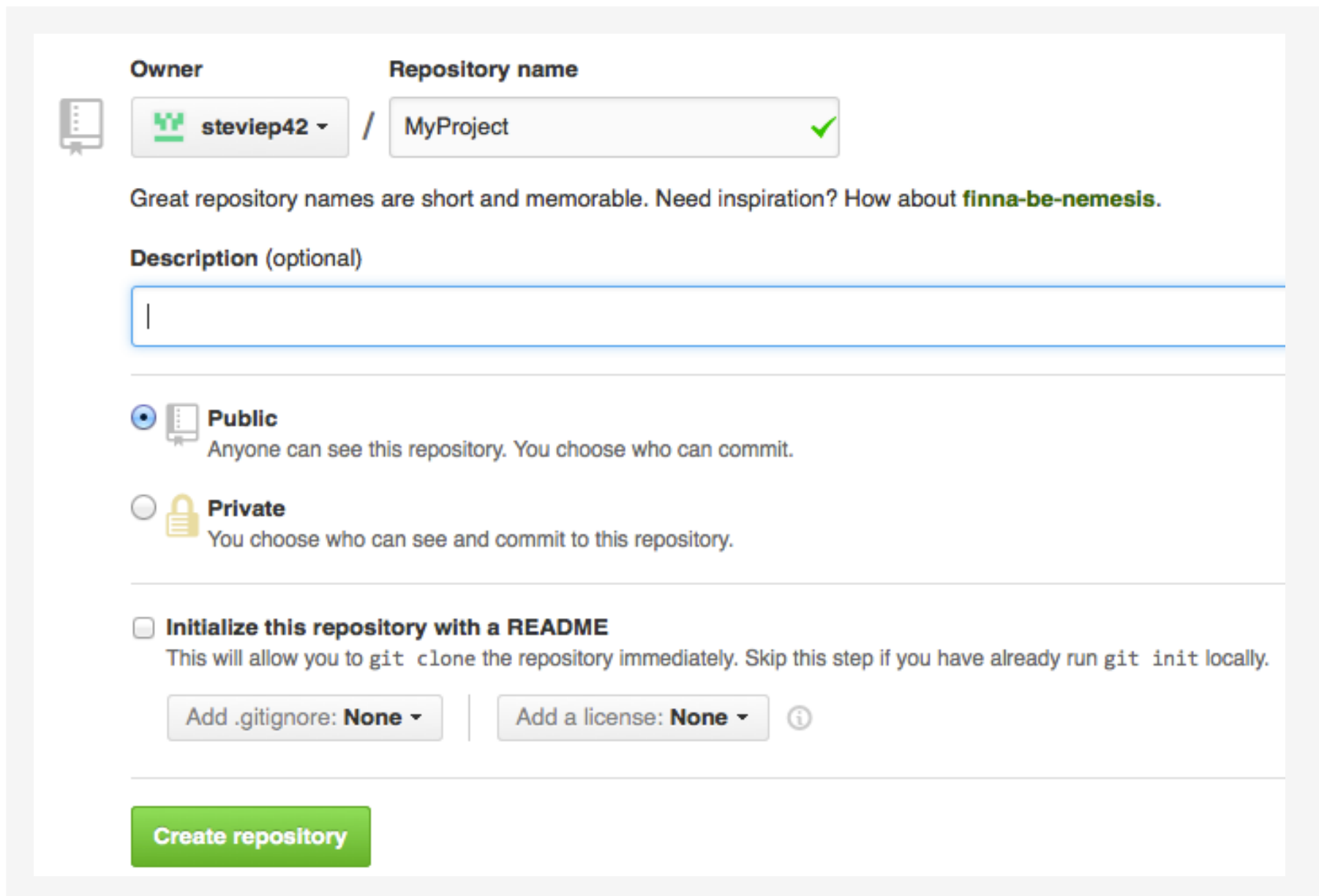
So you will then be prompted to enter the name of the repository. Here I am calling it "test". Also I don't want to initialize the repository since I already have a repository locally on my hard drive that I want to **push up** to this repository.

So I don't select the **"Initialize this repository with a README"** option

Then click the **Create Repository** button to complete the process.

Git - Using GitHub - Create a Remote Repository

Git - Using GitHub - Create a Remote Repository

After the creation of the repository we will see a screen with the following directions. Since we have a local repository already, MyProject, we now want to **push** it up to GitHub.

Push an existing repository from the command line

```
git remote add origin https://github.com/steviep42/MyProject.git
git push -u origin master
```

Create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/steviep42/MyProject.git
git push -u origin master
```
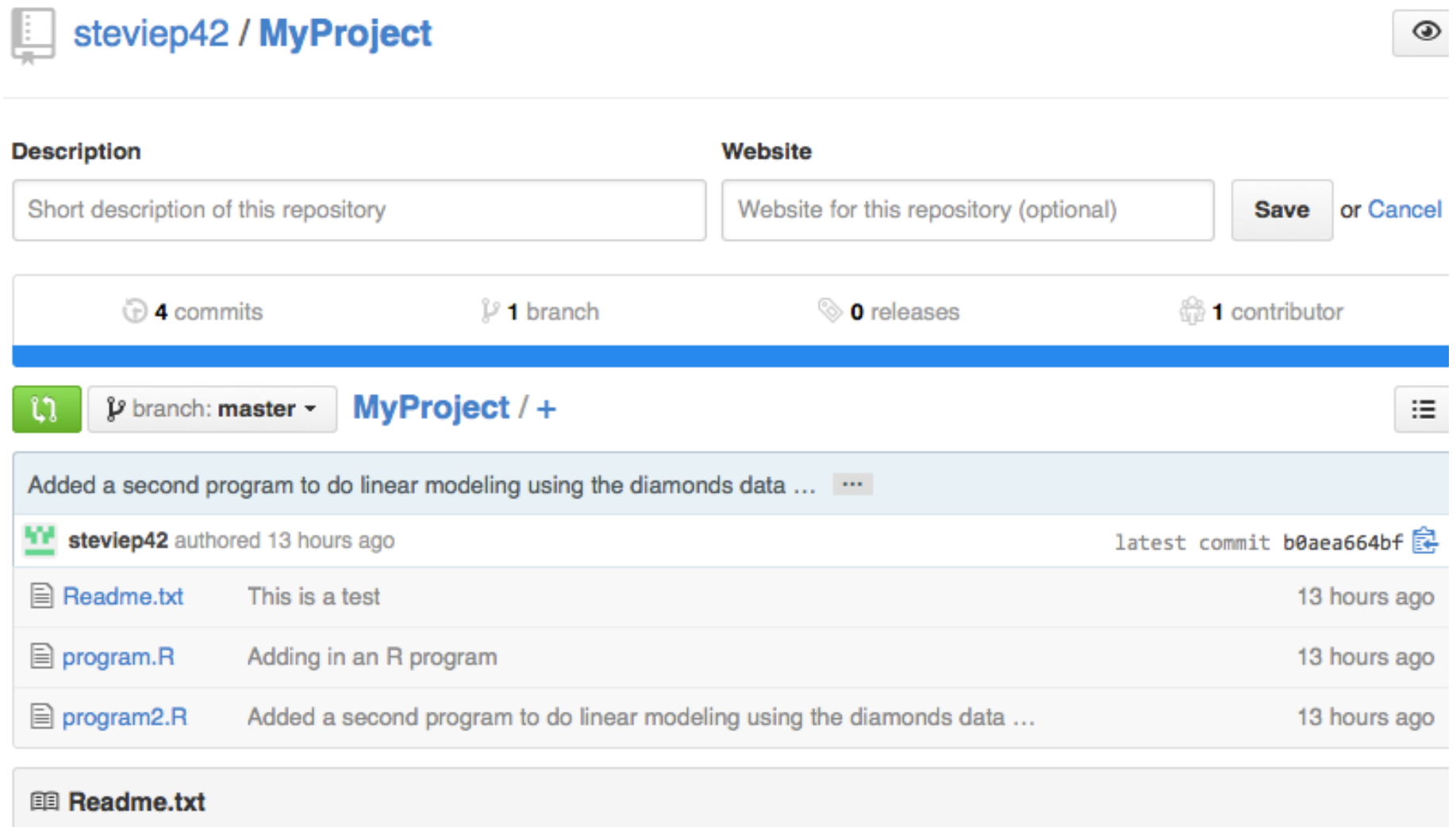
Git - Using GitHub - Create a Remote Repository

So back in our MyProject directory/repository we have to follow the given directions to **push** up the files. We are adding a remote reference called **origin** that will be associated with the URL for the GitHub repository.

```
$ pwd
/Users/fender/MyProject
$ ls
Readme.txt         program.R         program2.R
$ git remote add origin https://github.com/steviep42/MyProject.git
$ git push -u origin master
Counting objects: 12, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 1.13 KiB | 0 bytes/s, done.
Total 12 (delta 2), reused 0 (delta 0)
To https://github.com/steviep42/MyProject.git
 * [new branch]       master -> master
Branch master set up to track remote branch master from origin.
$ git remote -v
origin  https://github.com/steviep42/MyProject.git (fetch)
origin  https://github.com/steviep42/MyProject.git (push)
$
```

Git - Using GitHub - Create a Remote Repository

If we look on the GitHub site we will see our files are now there.

steviep42 / **MyProject**

Description

Short description of this repository

Website

Website for this repository (optional)

Save or Cancel

4 commits　　1 branch　　0 releases　　1 contributor

branch: **master** ▾　**MyProject** / +

Added a second program to do linear modeling using the diamonds data ...

steviep42 authored 13 hours ago　　latest commit b0aea664bf

Readme.txt	This is a test	13 hours ago
program.R	Adding in an R program	13 hours ago
program2.R	Added a second program to do linear modeling using the diamonds data ...	13 hours ago

Readme.txt

Git - Using GitHub - Create a Remote Repository

- So now what ?

- We still do our edits, adds, and commits as before. That procedure has not changed at all.

- But now when we have a series of commits we can then **push** them up to the **remote** repository.

- We do this only when we have some solid, well tested changes and additions to make.

- So let's add another program file. We add and commit it like we've been doing. The changes to the repository remain local until we **push** it up to the remote using a command like: `git push -u origin master`

Git - Using GitHub - Create a Remote Repository

```
$ pwd
/Users/fender/MyProject
$ ls
Readme.txt       program.R       program2.R
$ vi program3.R
$ git add .
$ git commit
[master f62e165] I added in a third program
 1 file changed, 3 insertions(+)
 create mode 100644 program3.R
$
$ git push -u origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/steviep42/MyProject.git
   b0aea66..f62e165  master -> master
Branch master set up to track remote branch master from origin.
$ ▉
```

Git - Using GitHub - Create a Remote Repository

So let's see what our github page looks like now. It should show the additional file that we just pushed up.

Git - Using GitHub - Create a Remote Repository

- Okay so now we can do our adds and commits and then push up to GitHub anytime we want to get our changes reflected.

- But as another example let's say we wanted to first create a placeholder repository on GitHub after which we will **clone** it locally and then push up changes. This is the more common scenario. It's not drastically different from what we did before.

- Let's create a repository called **ThesisCode** This could be a repos for stashing coide you develop in support of a potential thesis project.

- In this case we will select the option that says **"Initialize this repository with a README"** since we don't have a pre-existing repository sitting on our hard drive.
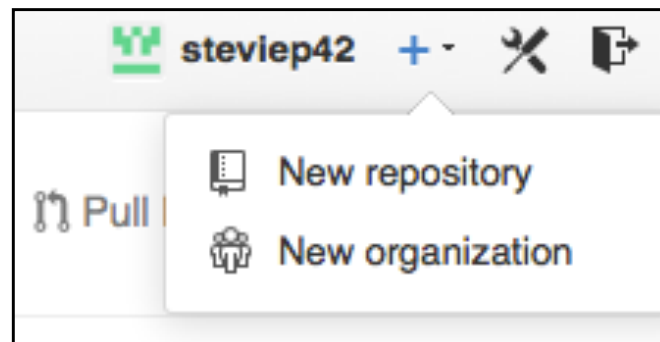
Steve Pittard wsp@emory.edu

Git - Using GitHub - Create a Remote Repository

Git - Using GitHub - Create a Remote Repository

Repository for R and SAS code in support of potential thesis project — Edit

⊕ 1 commit	⅄ 1 branch	🏷 0 releases	👥 1 contributor

⟲ ⅄ branch: **master** ▾   **ThesisCode** / +   ☰

Initial commit

steviep42 authored just now   latest commit **3d8012c823** 📋

📄 README.md   Initial commit   just now

📖 **README.md**

So now we can **clone** this locally and start working.

Git - Using GitHub - Create a Remote Repository

```
$ cd ~
$ pwd
/Users/fender
$ git clone https://github.com/steviep42/ThesisCode
Cloning into 'ThesisCode'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
$ cd ThesisCode/
$ ls -a
.                    ..                   .git         README.md
$ 
```
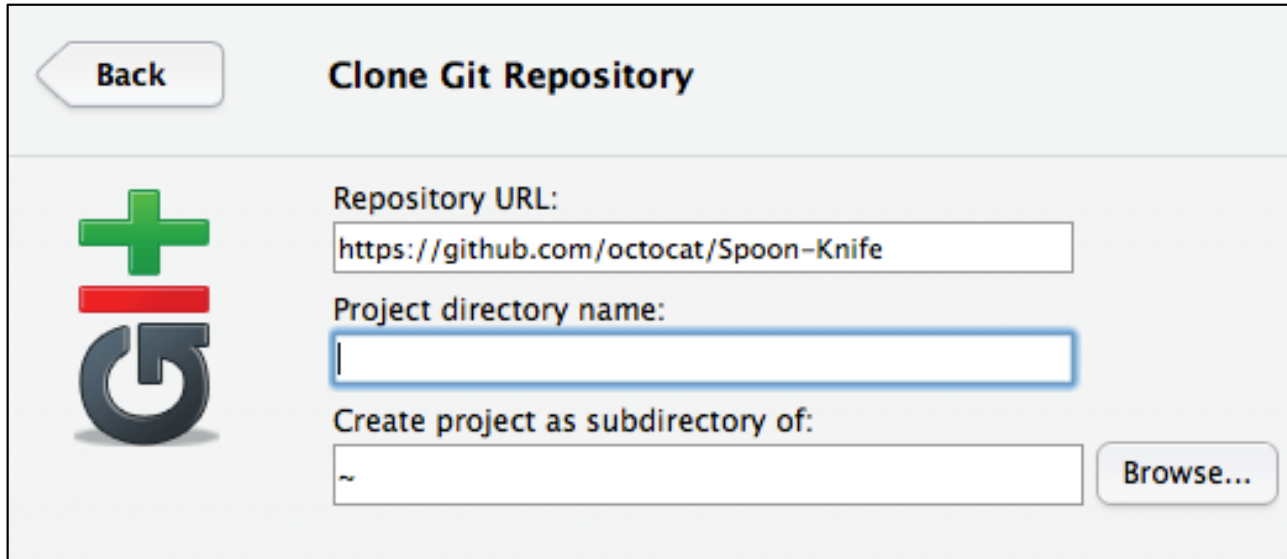
Git - Using RStudio to Manage Repositories

- Rstudio has the ability to manage git projects for you although in comparison to the GUI tools it is somewhat limited (at least in my opinion). Here is a tour of how to do some of the basics.

- Let's use Rstudio to clone an existing remote repository. To create a **Git** managed project select File -> New Project from the Rstudio menu. When presented with the following panel select **Version Control** and then select **Git** when prompted to select a repository type/source.
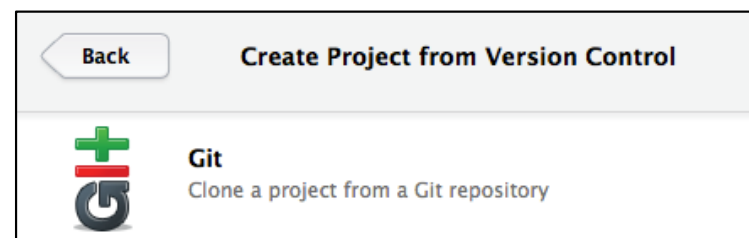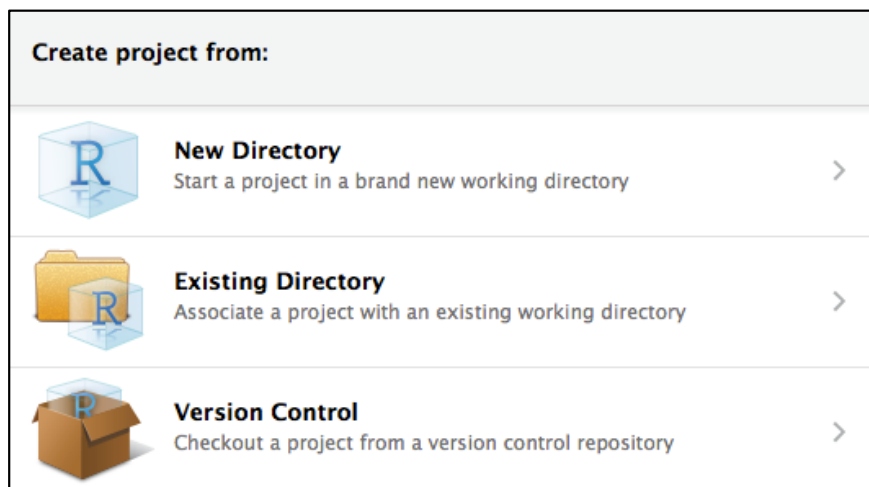
Git - Using RStudio to Manage Repositories

- The next window you encounter prompts you to enter a URL of a Git hosted repository. In this case we'll use a new one. Enter the URL `https://github.com/octocat/Spoon-Knife`

- If you want to change the local destination to something **other** than your home directory then click the **Browse** button and select a different folder. Then click the **Create Project** button.

Git - Using RStudio to Manage Repositories

Rstudio will then close the existing window and open a new one that contains the remote repository. It will also be placed under **git** control. The two right panes will look like the following. Note that Rstudio puts the question mark characters next to files that it has noticed are not currently part of the repository.
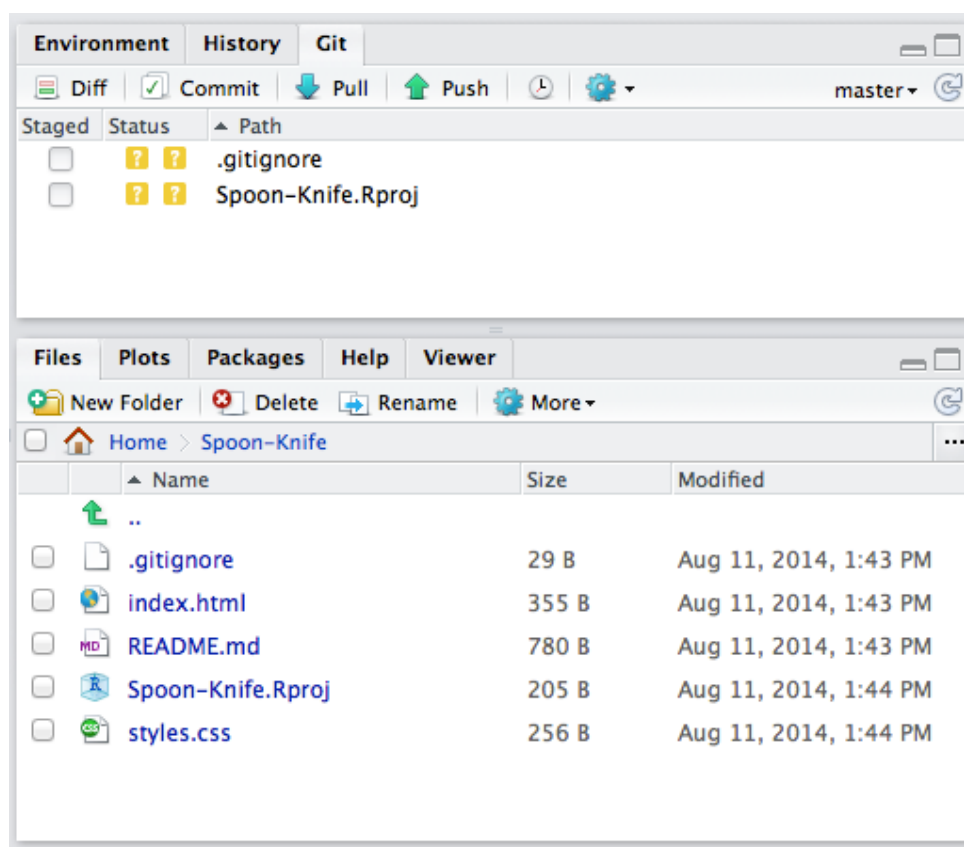
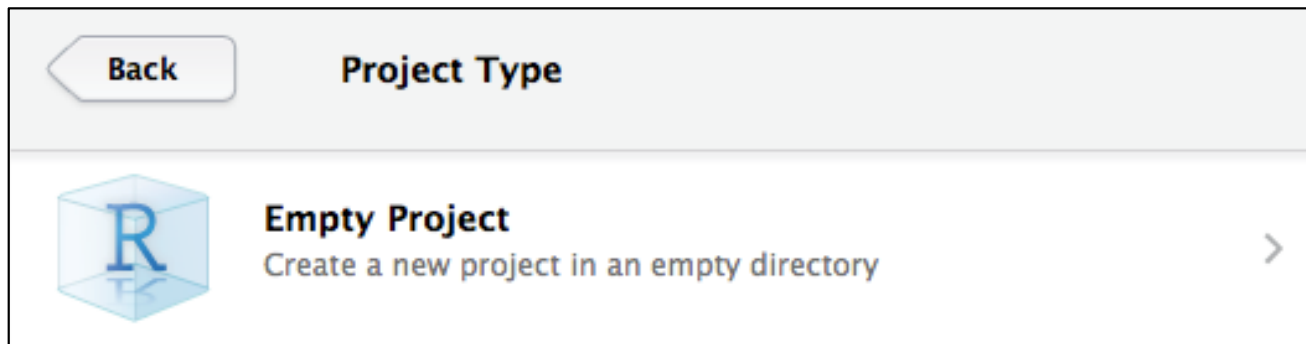# Git - Using RStudio to Manage Local Repositories

To create a local project under **git** control is very similar. You create a new project though when you get to this window:
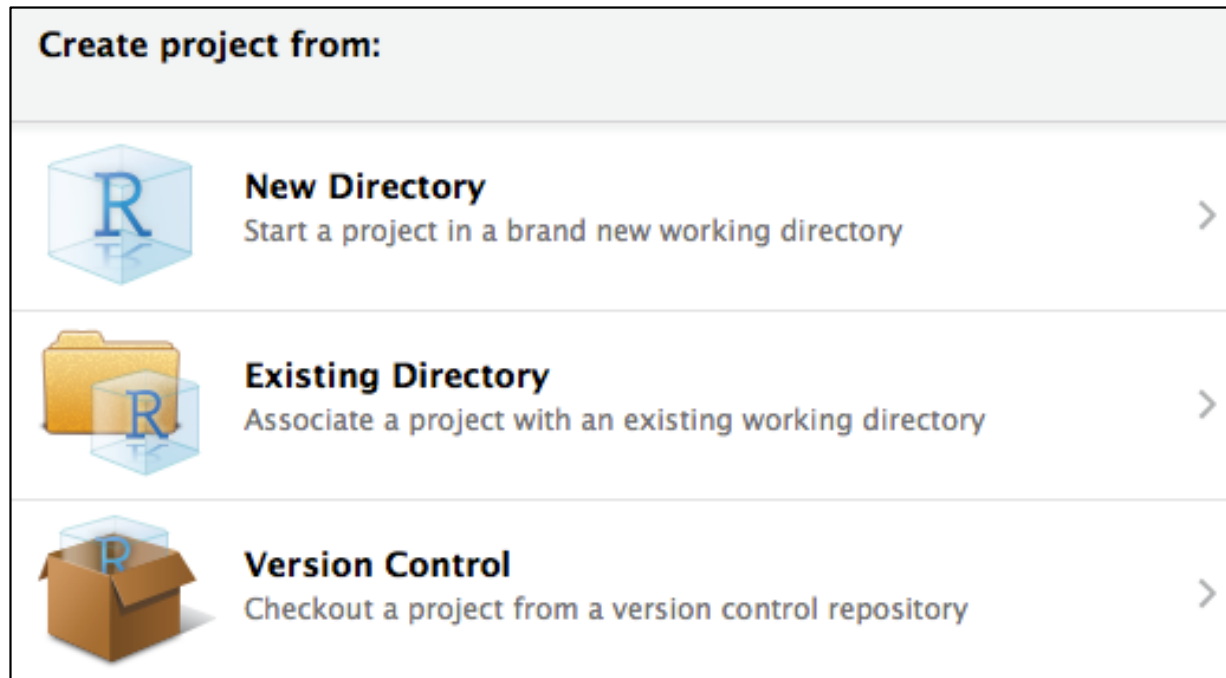
Select the "New Directory" option.

Git - Using RStudio to Manage Local Repositories

Then from the next window select "Empty Project":

This will then proceed to another window. Type in the name of a new folder and check **Create a git repository** to put the new folder under git control. Finally, click **Create Project.**
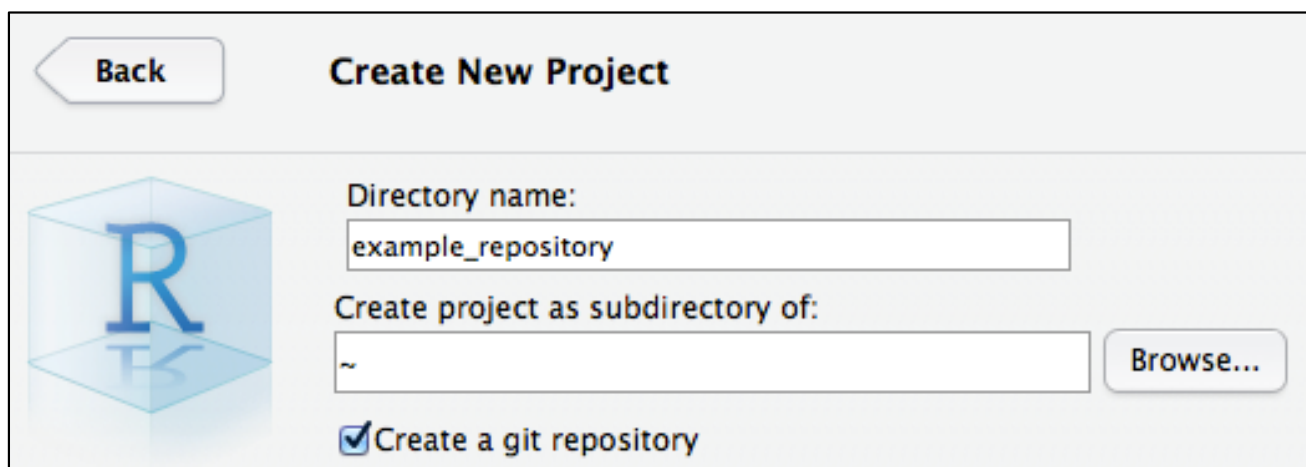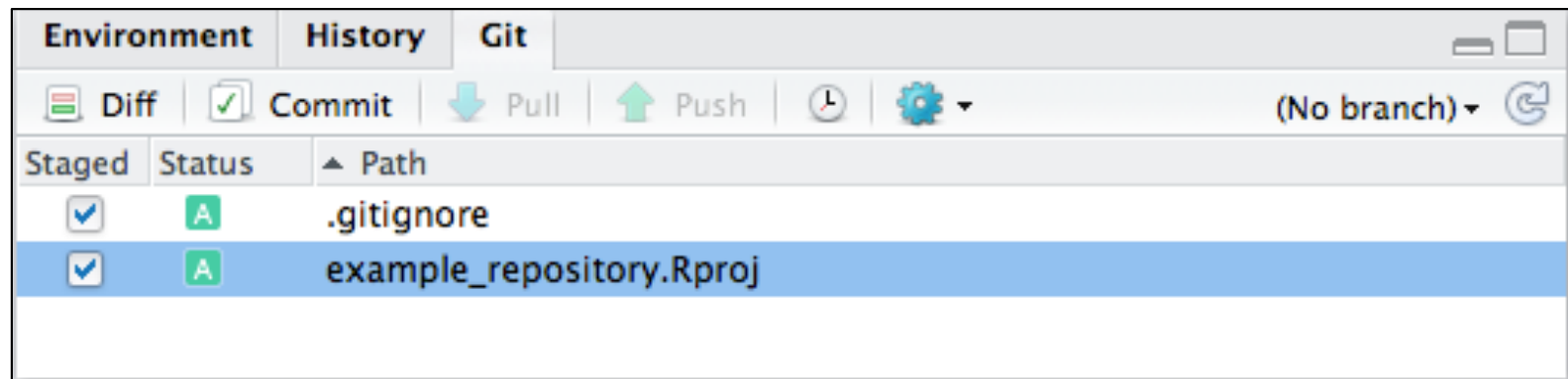
Git - Using RStudio to Manage Local Repositories

And finally you will see that RStudio has opened your new project. The two right panes will look something like this (assuming you used the name **example_repository** as the name of your repository.
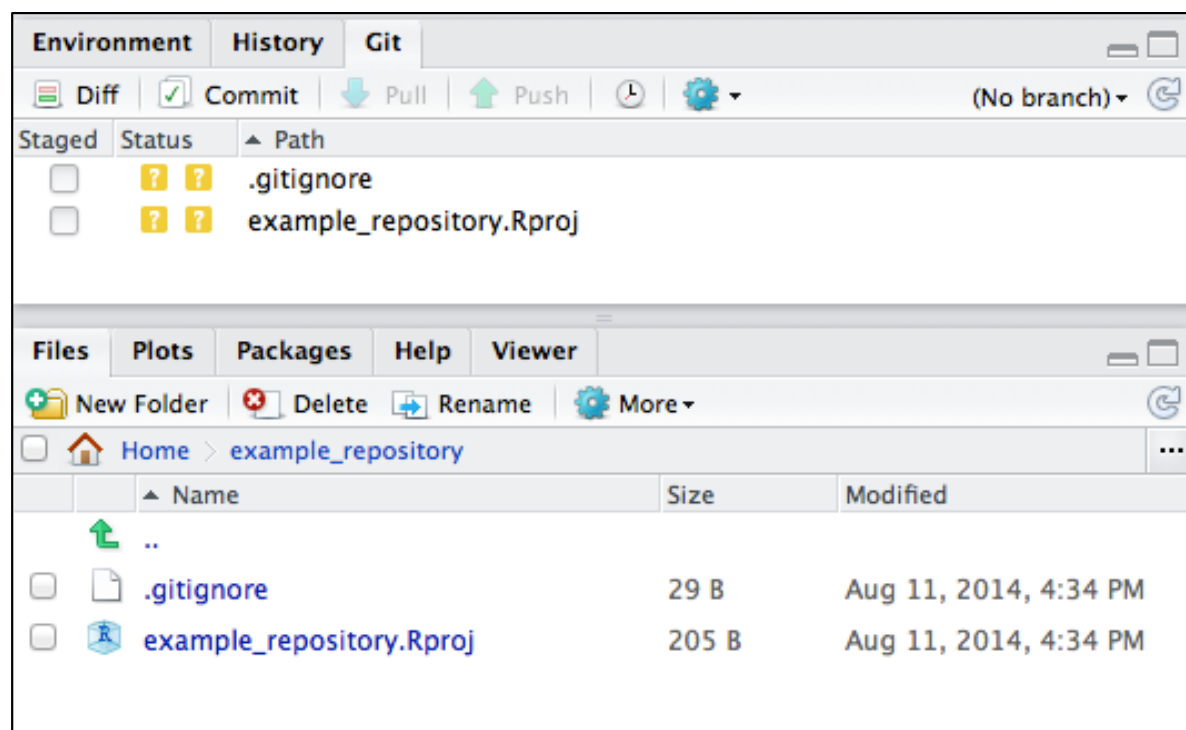
The **question marks** in the upper pane tell us that **git** notices the two files in the folder and that they haven't yet been staged or committed to the project.
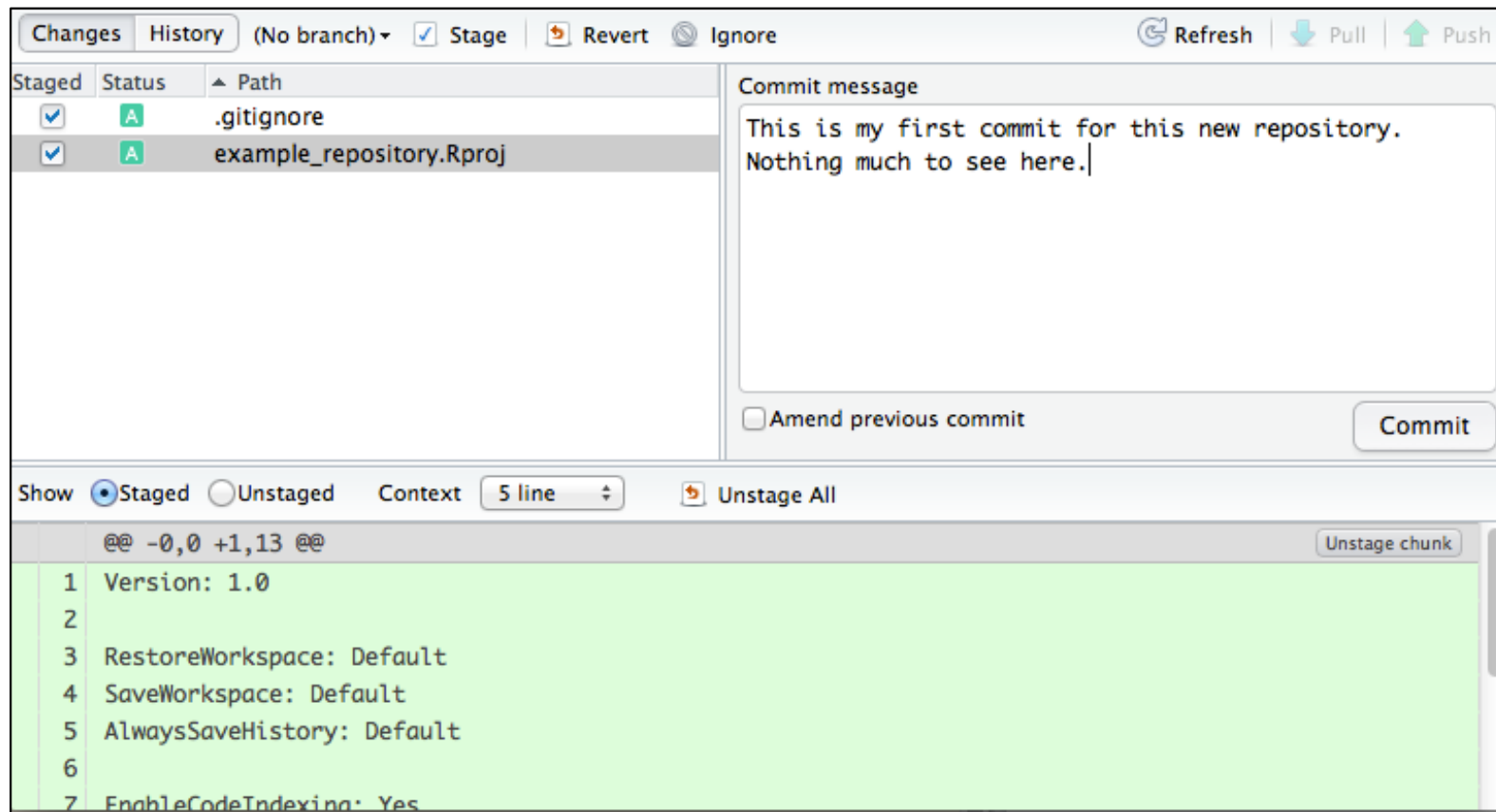
Git - Using RStudio to Manage Local Repositories

If we click the boxes next to the two files in the **Staged** column then git will start tracking changes. Note that under the **Status** column you will a green **A** letting you know that the files have been added.

Environment	History	Git										(No branch) ▾
▤ Diff	☑ Commit		⬇ Pull		⬆ Push		⏱	⚙ ▾				
Staged	Status	▲ Path										
☑	A	.gitignore										
☑	A	example_repository.Rproj										

If we then hit the **commit** tab then we will be prompted to enter a commit message just like we did when we managed our repository from the command line.
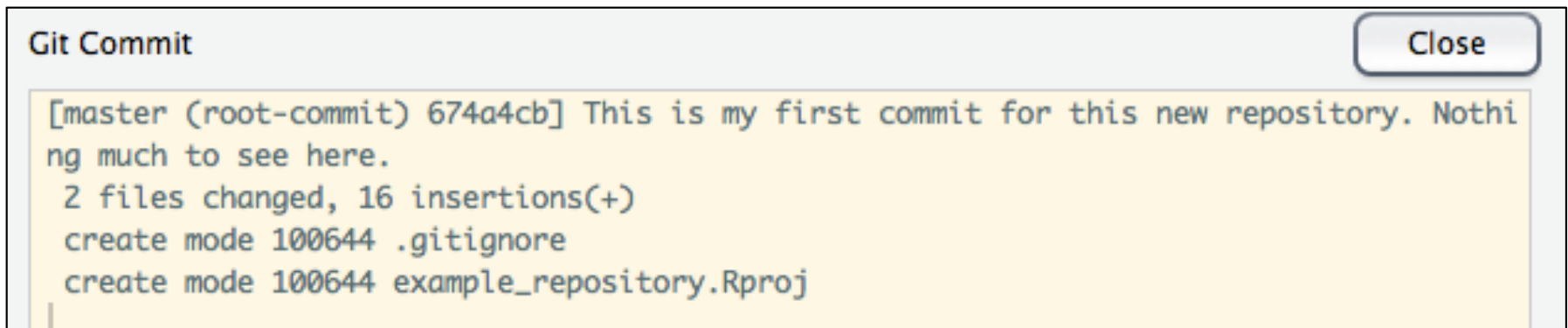
Git - Using RStudio to Manage Local Repositories

Once we hit commit we will see a window like below. Enter a commit message and hit the **Commit** button.

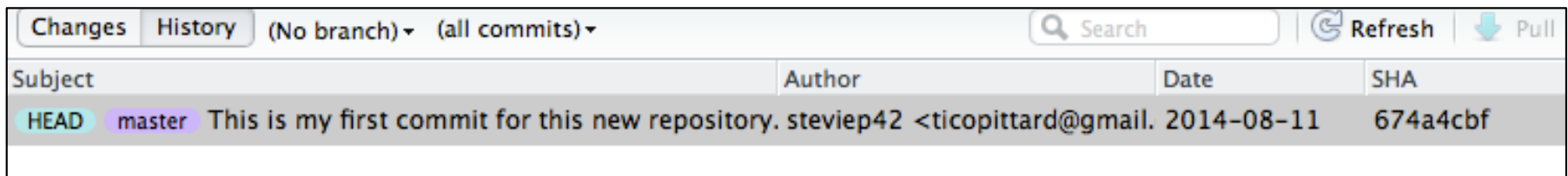# Git - Using RStudio to Manage Local Repositories

After hitting the **Commit** button you will get a confirmation message that the **Master** branch has been created. **Close** this window.

```
Git Commit                                                    Close

[master (root-commit) 674a4cb] This is my first commit for this new repository. Nothi
ng much to see here.
 2 files changed, 16 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 example_repository.Rproj
```

If you click **refresh** on the previous window you will see more confirmation:

Changes	History	(No branch) ▾	(all commits) ▾		🔍 Search		🔄 Refresh	⬇ Pull
Subject				Author		Date	SHA	
HEAD	master	This is my first commit for this new repository.		steviep42 <ticopittard@gmail.		2014-08-11	674a4cbf	

Git - Using RStudio to Manage Local Repositories

Back in the Rstudio window you will now see:

If you click **refresh** on the previous window you will see more confirmation:

Git - Using RStudio to Manage Local Repositories

Return to the Rstudio window and do **File -> New -> R Script** and create a new R script. Put in some arbitrary contents and save it as **Script1**
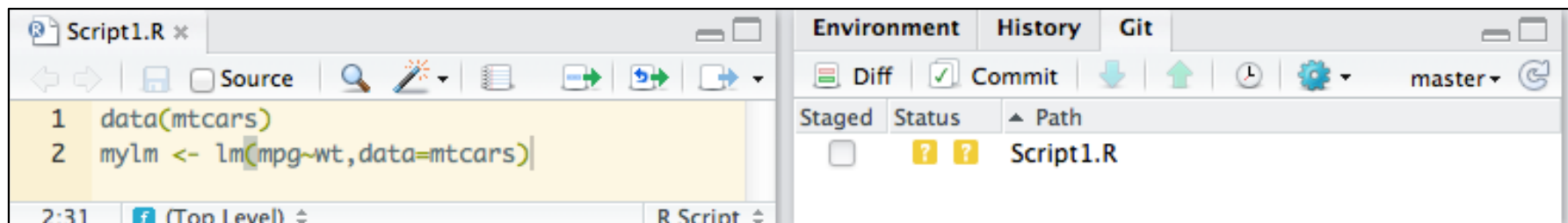
Git will recognize the new file and we can then **add** it and **commit** it if we want to. The procedure is similar to that as before.

To do anything much beyond **adds, commits,** and **diffs** then you need to launch a shell, which then requires the procedures we learned about in the first part of this presentation wherein we did things via the command line.

Git - Using RStudio to Manage Local Repositories

To do anything much beyond **adds, commits,** and **diffs** then you need to launch a shell, which then requires the procedures we learned about in the first part of this presentation wherein we did things via the command line (create branches, push to remote repositories).