NYC DATA SCIENCE ACADEMY

Introduction to Git & Github

Data Science Bootcamp

Outline

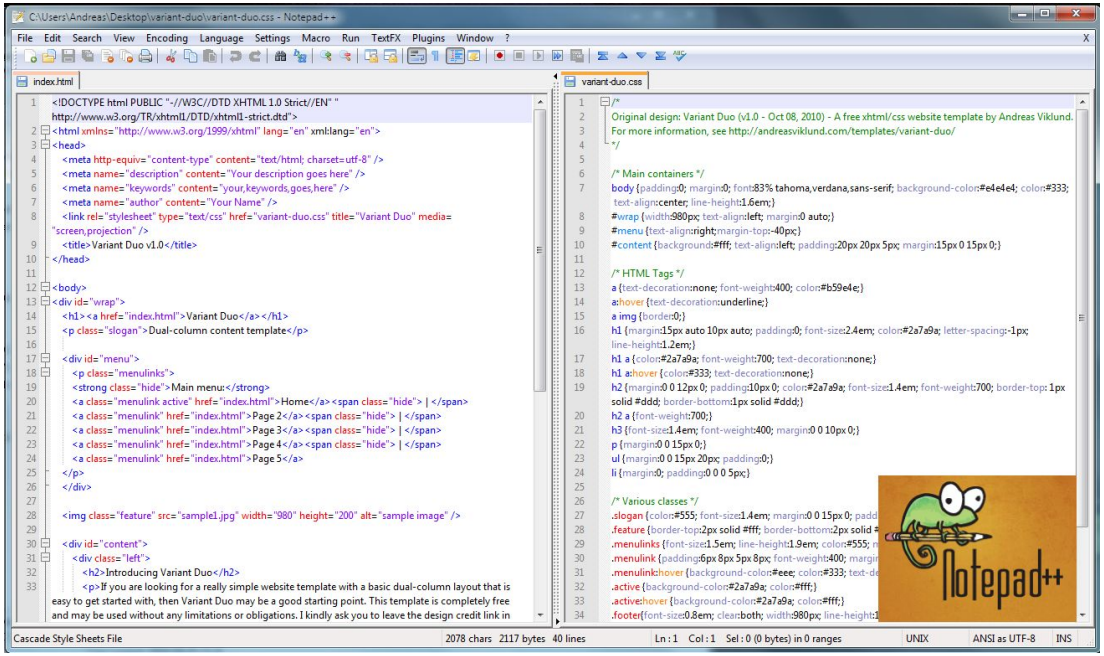
- **Set up Git and GitHub**
  - **Introduction to Git**
    - **Creating a Git Repository**
    - **Manipulating files**
  - **Introduction to GitHub**
    - **Lightning Tour of Github**
    - **Create a Remote Repository**

Pre-requisites

- Patience and willingness to keep learning
- Have a good text editor for editing plain text files. Good examples are:
  - Notepad++
  - TextWrangler
  - Sublime
  - Emacs
  - Vim
  - LightTable
- Syntax highlighting and side-by-side editing are also useful features.

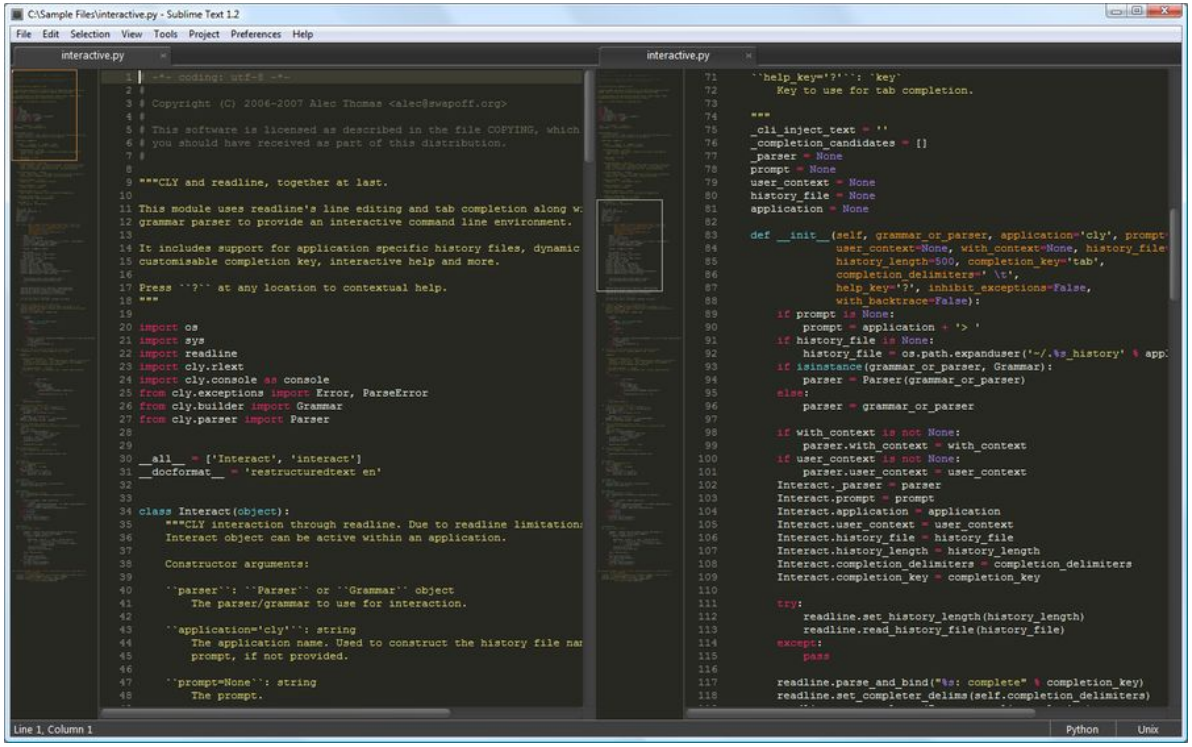
Pre-requisites for Windows

- Notepad++: https://notepad-plus-plus.org/download/
- The default is plain UTF-8 encoding, without a byte-order mark (BOM).



Pre-requisites for Windows

- Sublime Text: http://www.sublimetext.com/3
- Heavy-weight IDE: RStudio, Eclipse, Visual Studio, PyCharm

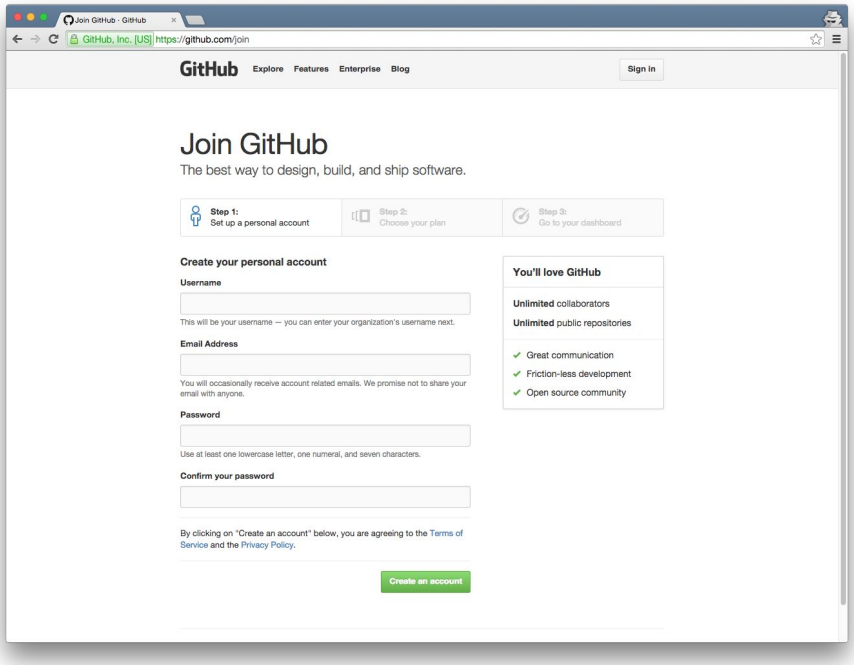


Pre-requisites for Mac

- Sublime Text: http://www.sublimetext.com/3
- TextMate: http://macromates.com/download
- LightTable: http://lighttable.com/
- MacVim: https://github.com/macvim-dev/macvim
- Many other options ...

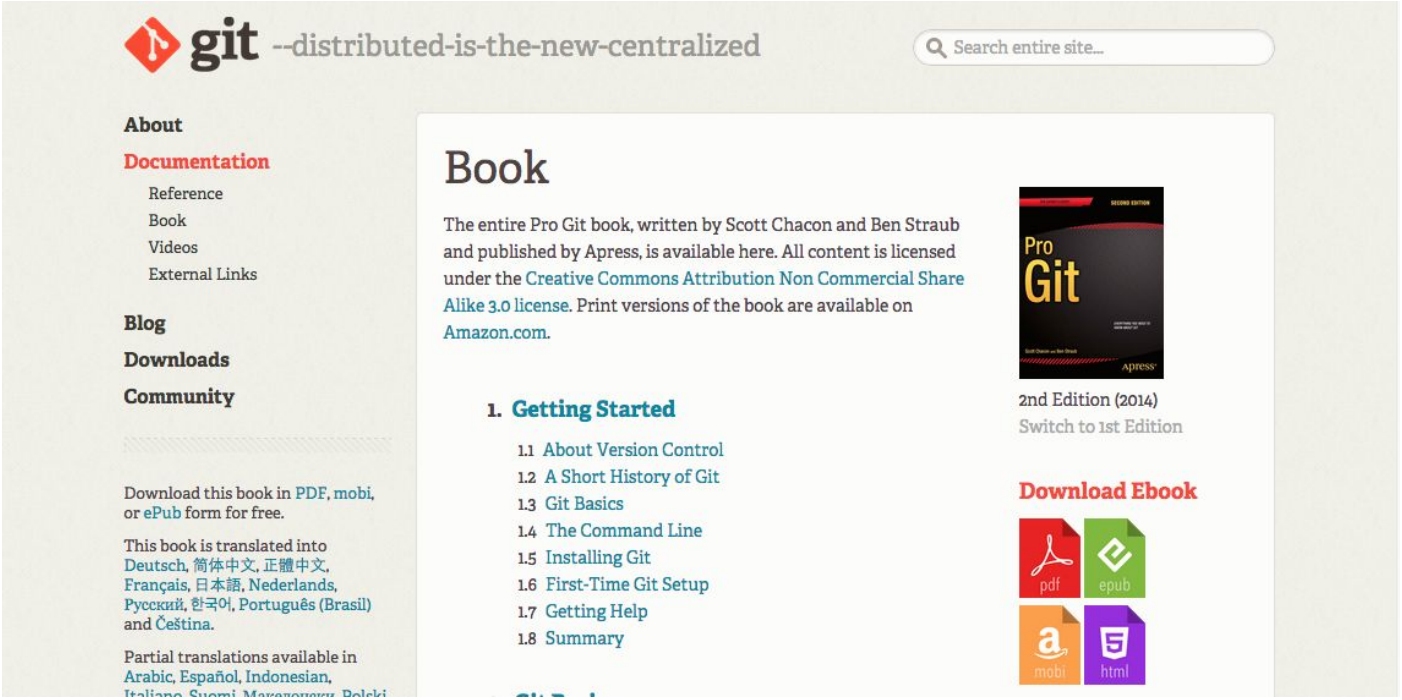
Pre-requisites

- Create a personal GitHub account.
- Choose the free option: get unlimited open repositories, but no private repositories.
- Save your credentials in a safe place.



Pre-requisites

- Bookmark the free Pro Git book http://git-scm.com/book/en/v2/

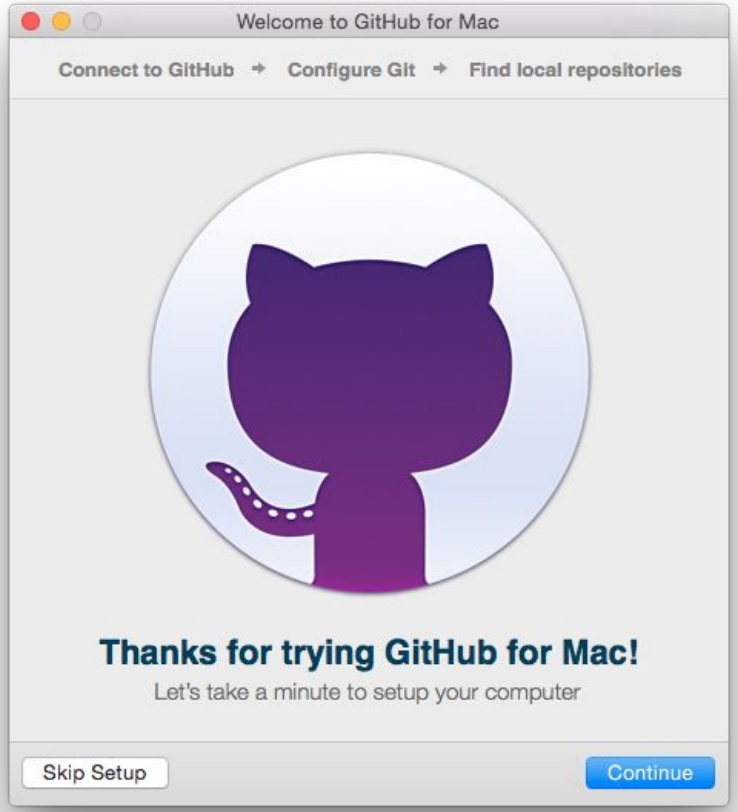


Download and Install Github GUI For Your OS (Optional)

- Easy method: Download and install GitHub's application. May not work for all versions of your OS. If this doesn't work, it's ok, let an instructor know.
- Windows: https://windows.github.com/
- Mac: https://mac.github.com/

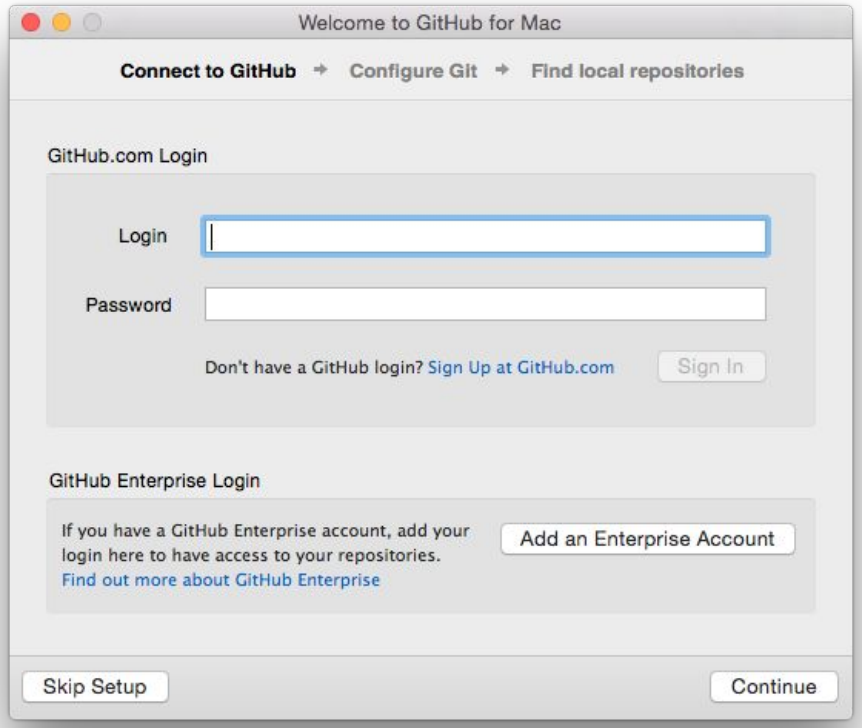
Download and Install Github GUI For Your OS (Optional)

- After you install the GUI, find the application in the start menu or Applications, and open it.



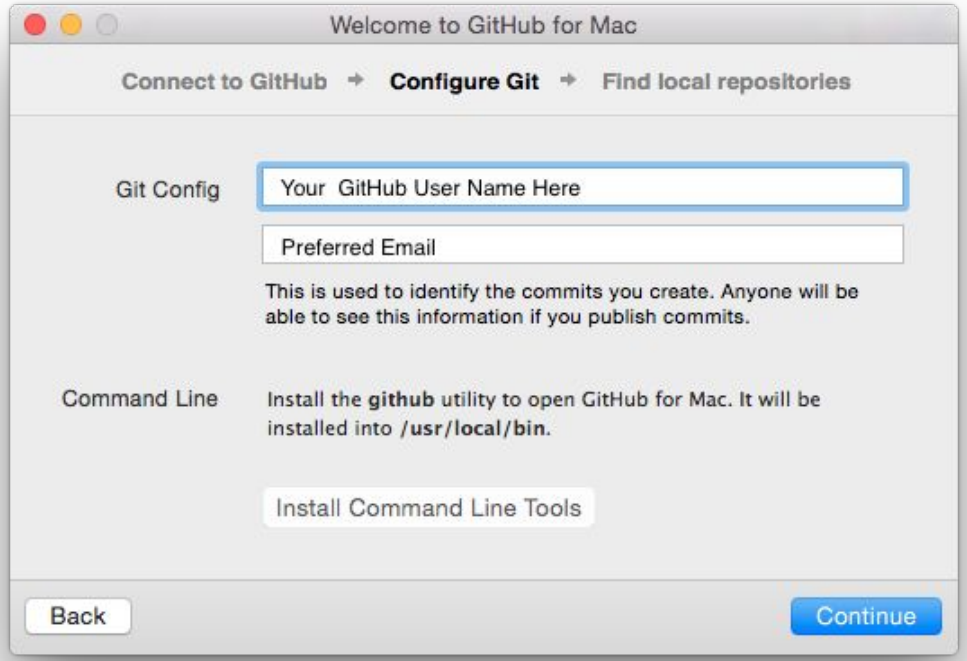
Download and Install Github GUI For Your OS (Optional)

- Enter your GitHub credentials through the setup wizard. This step should automatically configure your SSH keys for identification.



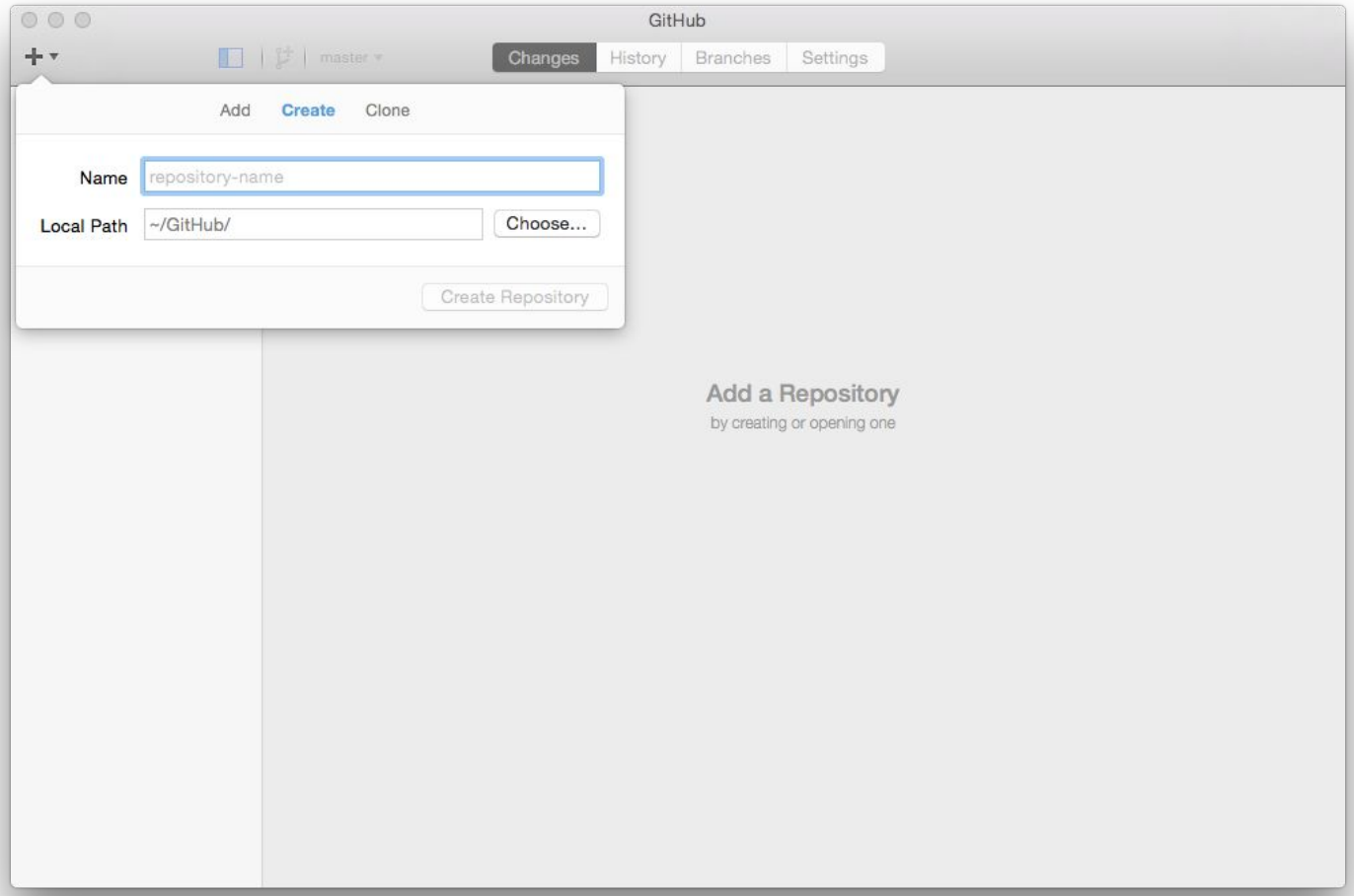
Download and Install Github GUI For Your OS (Optional)

- Enter the github username you created, and the preferred email you used as well. Your commits will be identified by these details, so make sure it's okay to share.



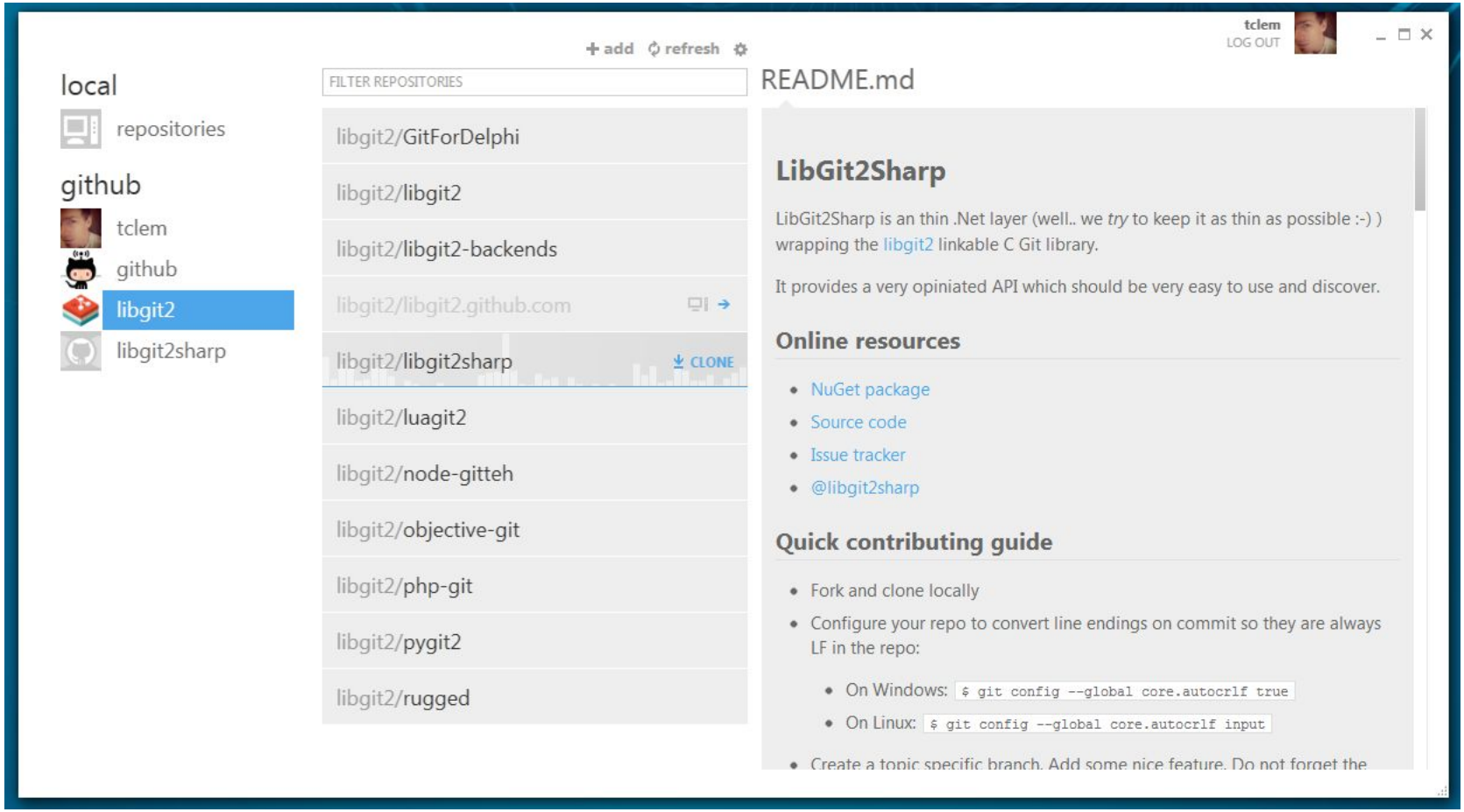
Download and Install Github GUI For Your OS (Optional)

- Example Application for Mac



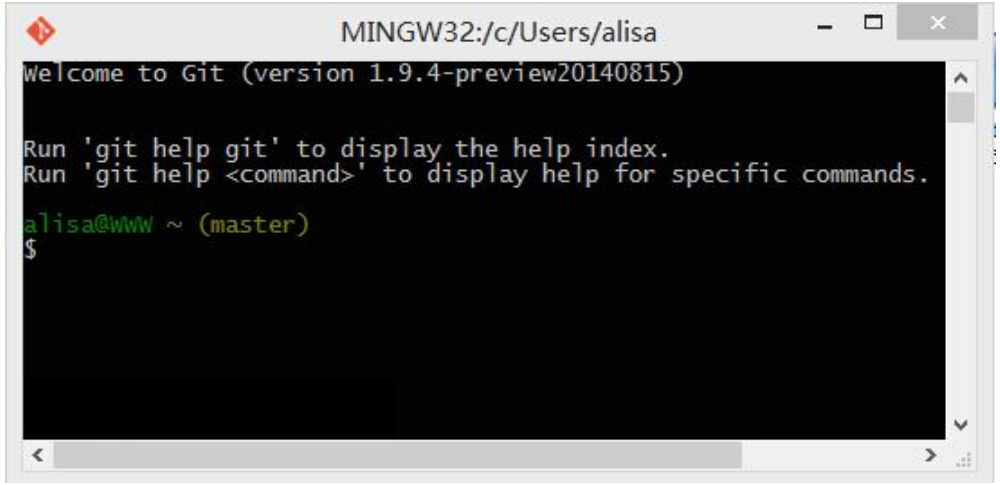
Download and Install Github GUI For Your OS (Optional)

- Example Application for Windows



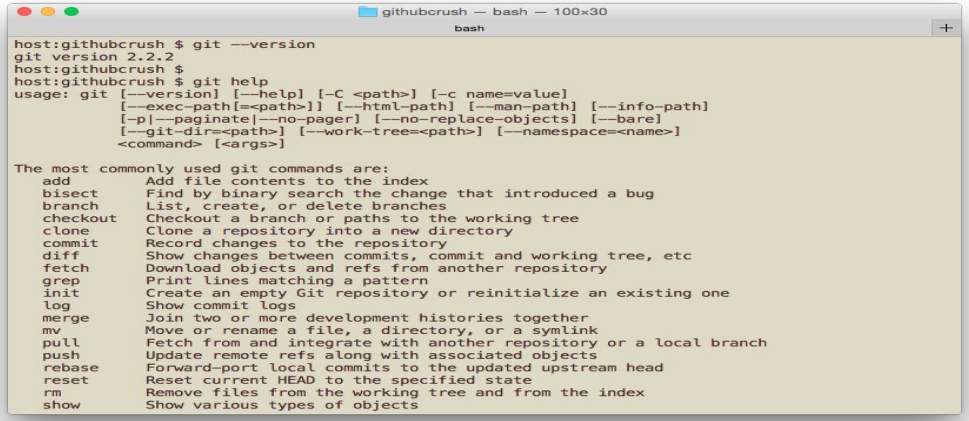
Command-Line For Windows

- Download the package msysgit from https://git-for-windows.github.io/ and install it. This wizard will take you through the steps to install it.
- Assuming the default settings, open your 'My Documents' folder to find the root folder of the command-line environment that msysgit created.
- On your computer, find Git in the start menu, and open GitBash. If you find something like the following window, the installation is successful!



Command-Line for Mac

- Three alternative install methods:
  - Use the pre-packaged version of git provided by OS X
  - Directly install it from a package http://git-scm.com/download/mac
  - Install the Homebrew package manager http://brew.sh/ and then open your terminal type `$ brew install git`
- Within Applications, find the Utilities folder, and open the Terminal application. If you can successfully type git help at the terminal, then installation was successful!



View All Configuration Information

- Start the terminal program for your OS. Type in this command to list all the known global Git configuration. If you don't see a **user.name** or **user.email** setting in your global config, then it has to be manually configured.

```
$ git config --list --global
...
user.name='your username'
user.email=your email
...
```

Configuration for Git

- View the version information and verify that the installation was successful. E.g. on Windows:

```
$ git --version
git version 1.9.4.msysgit.1
```

- Set your user information in the command window/terminal

```
$ git config --global user.name "username"
$ git config --global user.email "your email"
```

- Note : The parameter --global in the git config command indicates that all repositories on this host will use this configuration. For setting a different configuration on an individual repository, use the --local parameter.

View Specific Configuration Information

- If you want to view a particular setting, like user.email, specify that parameter after the git config command.

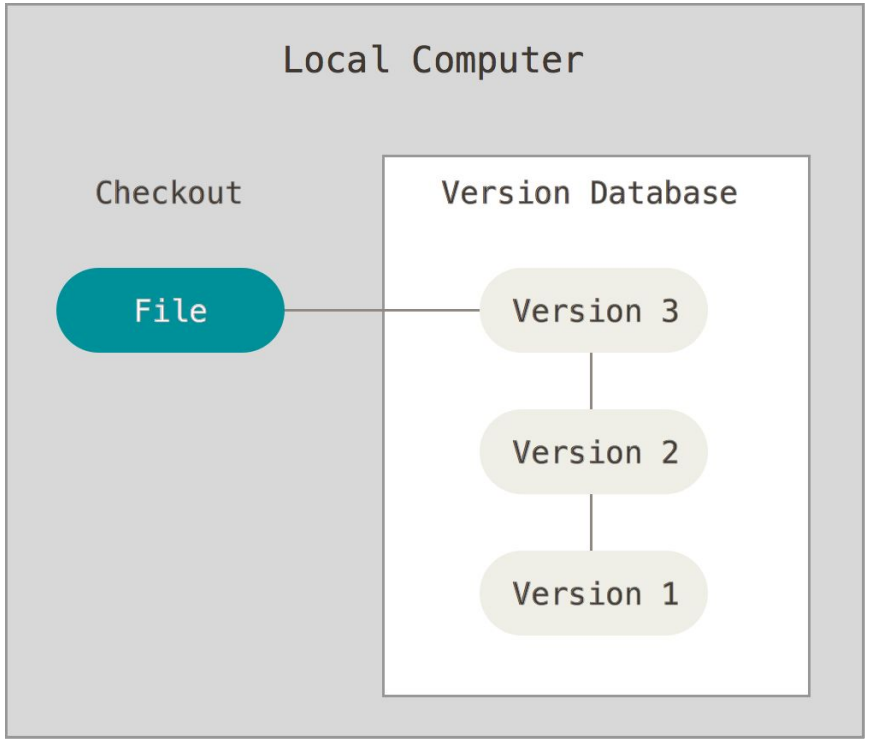
```
$ git config user.email
my.email@domain.com
```

Outline

- Set up Git and GitHub
- **Introduction to Git**
  - Creating a Git Repository
  - Manipulating files
- Introduction to GitHub
  - Lightning Tour of Github
  - Create a Remote Repository

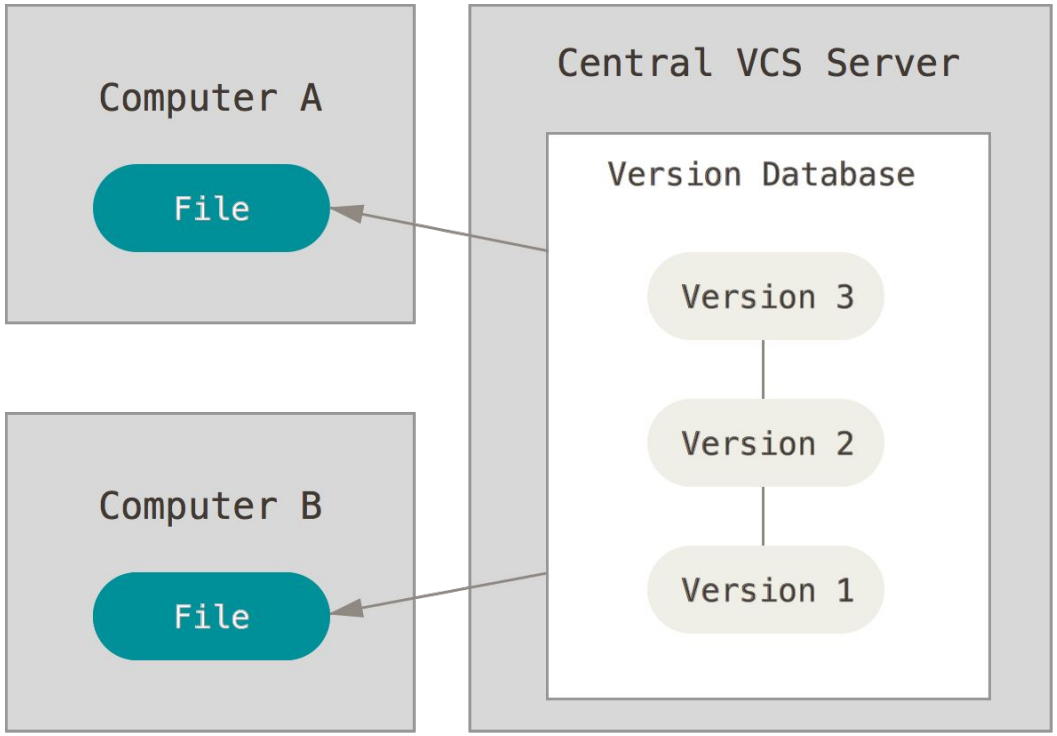
What is Version Control?

- Version control allows us to capture and refer back to a known state of a file across a series of changes.
- E.g. Mac OS X File System, MS Office App 'Track Changes' feature



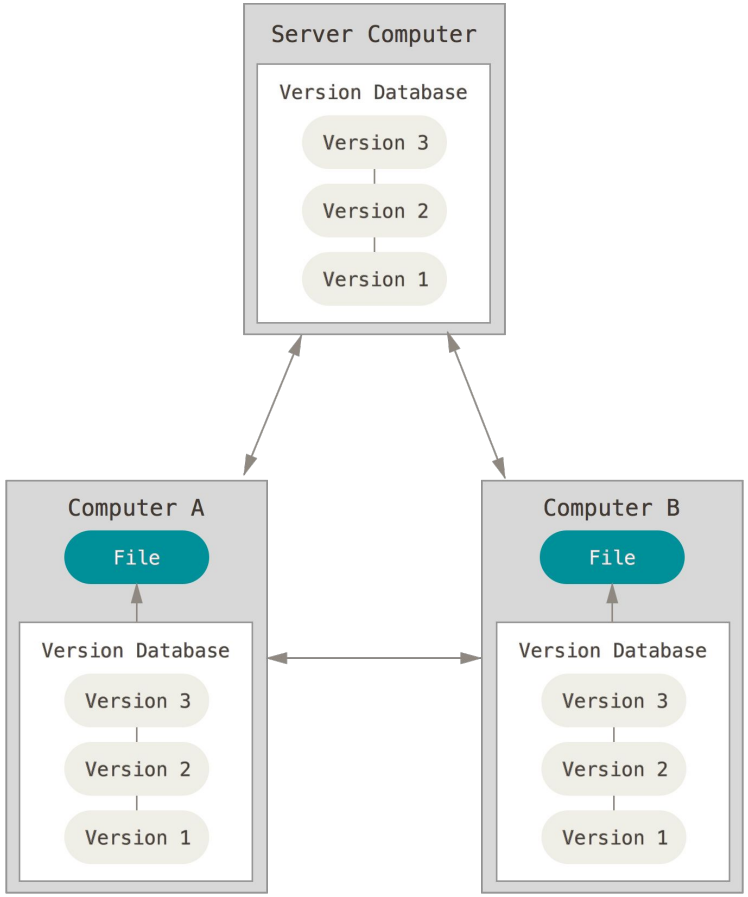
What is Centralized Version Control?

- Sharing changes between hosts is done through a central authority (the source of truth).
- E.g. standard Google spreadsheet



What is Distributed Version Control?

- Sharing changes between hosts is done using a peer-to-peer model (no real source of truth).
- E.g. Git, offline Google spreadsheet



Introduction to Git

- Distributed version control system, great for collaboration over source code
- Linus Torvalds developed Git in 2005 to help manage the Linux kernel development across thousands of volunteers
- Fast and robust performance
- Wide community adoption
- Allows massive, parallel branch development
- Has the ability to efficiently manage large scale projects

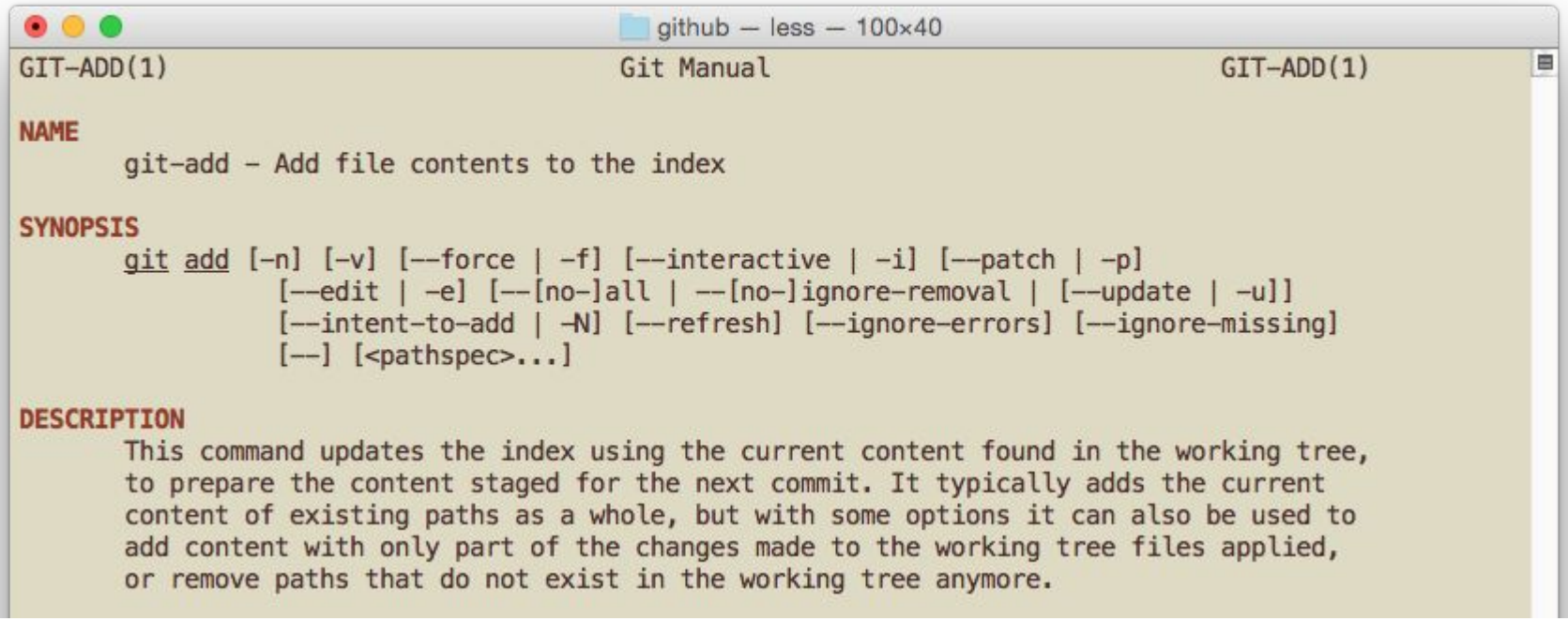
Introduction to Git

- If you're coming from another Version Control System (VCS), note that Git has some differences:
  - Doesn't really work like CVS, Perforce, i.e. centralized version control.
  - Instead of encoding deltas between one version and the next, Git works with files as a whole.
  - Treats the history as a series of snapshots, which are then efficiently stored in the local database.
  - Many commands in Git work on primarily the local data. No need to talk to a remote server, unless you want to exchange history, branches, and/or tags.

Help with Git Commands

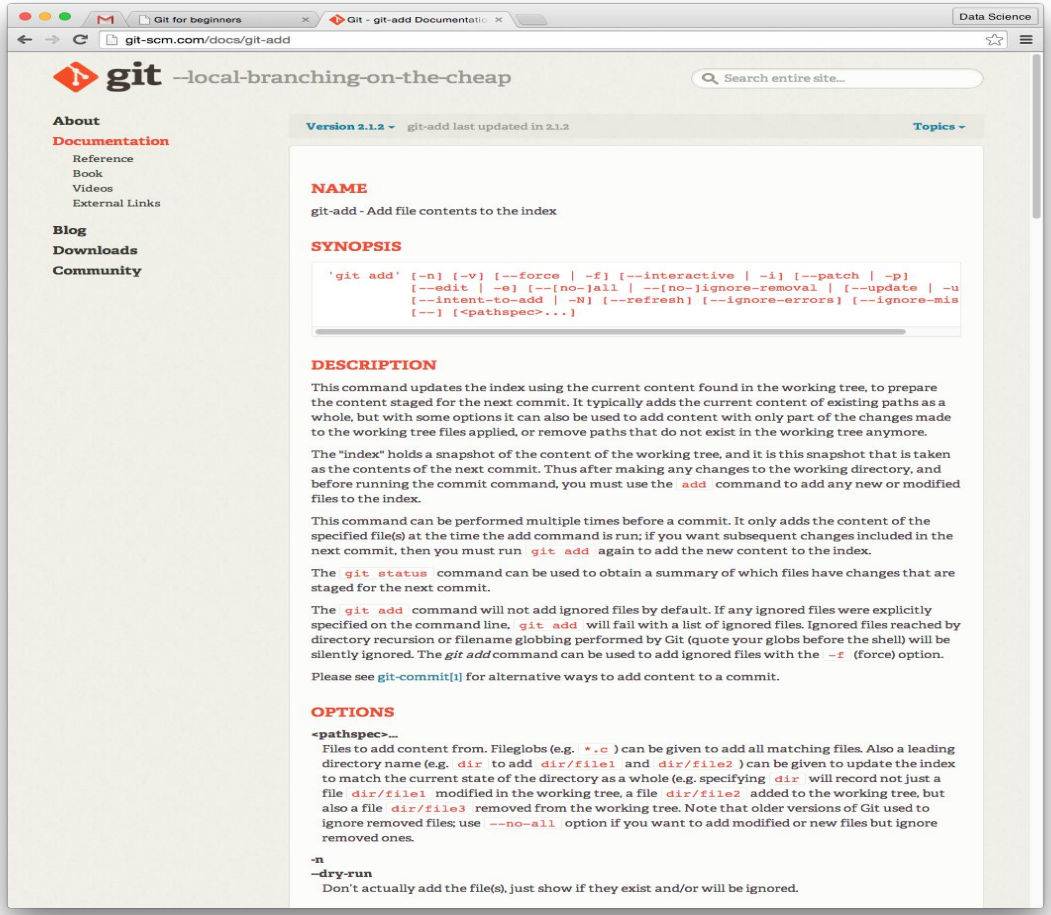
- For more information about Git's many commands, consult the help command.

```
$ git add --help
$ git help add
```



Help on the Web

- Many online resources, but here is one: http://git-scm.com/docs/git-add



Outline

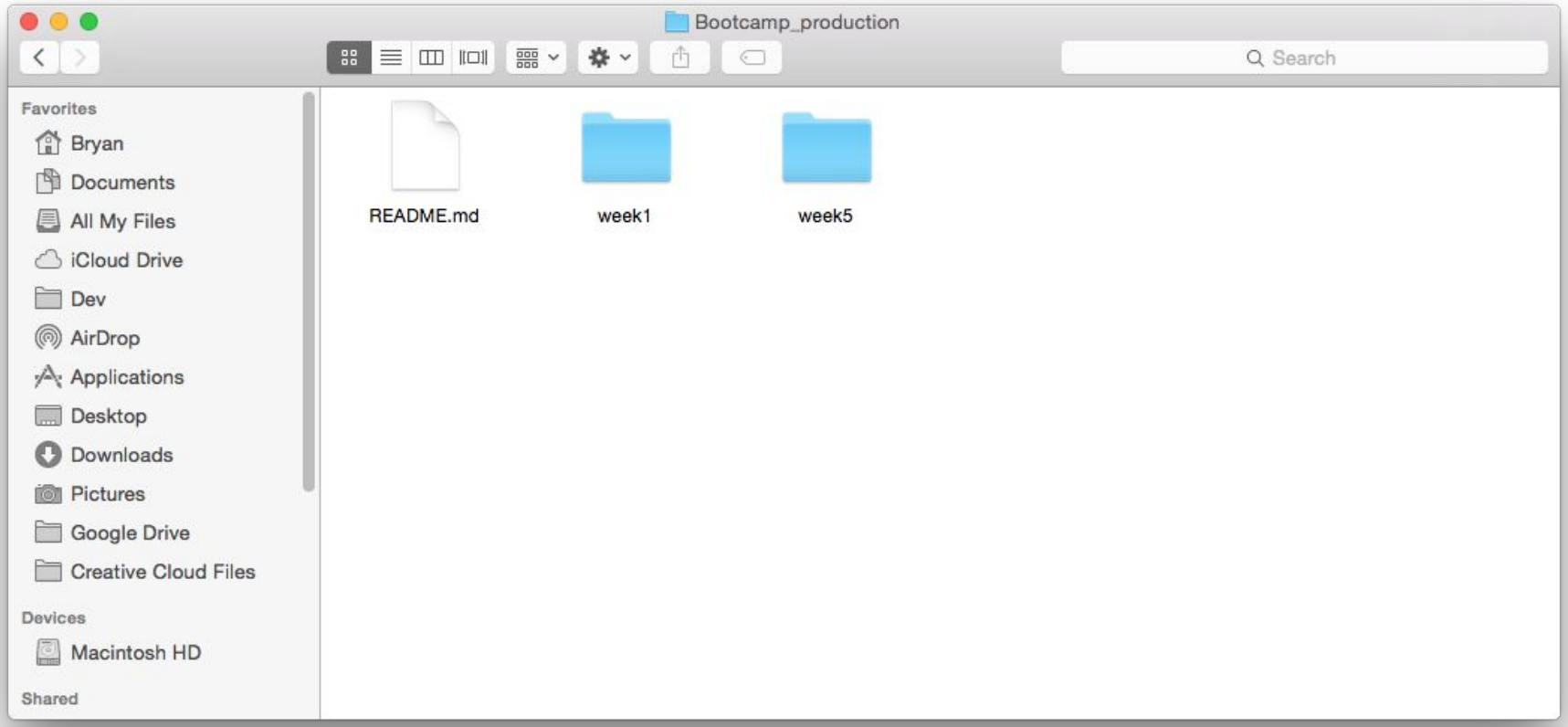
- Set up Git and GitHub
- Introduction to Git
  - **Creating a Git Repository**
  - Manipulating files
- Introduction to GitHub
  - Lightning Tour of Github
  - Create a Remote Repository

What is a Git Repository?

- View the repository as a controlled working area (a directory) with a version database.
  - Some or all of the files in this directory can be managed by Git:
    - add and commit files to the repository
    - move or delete files
    - view updated files
    - view history
    - cancel local changes
  - Git can even follow renamings (or moving) of files!

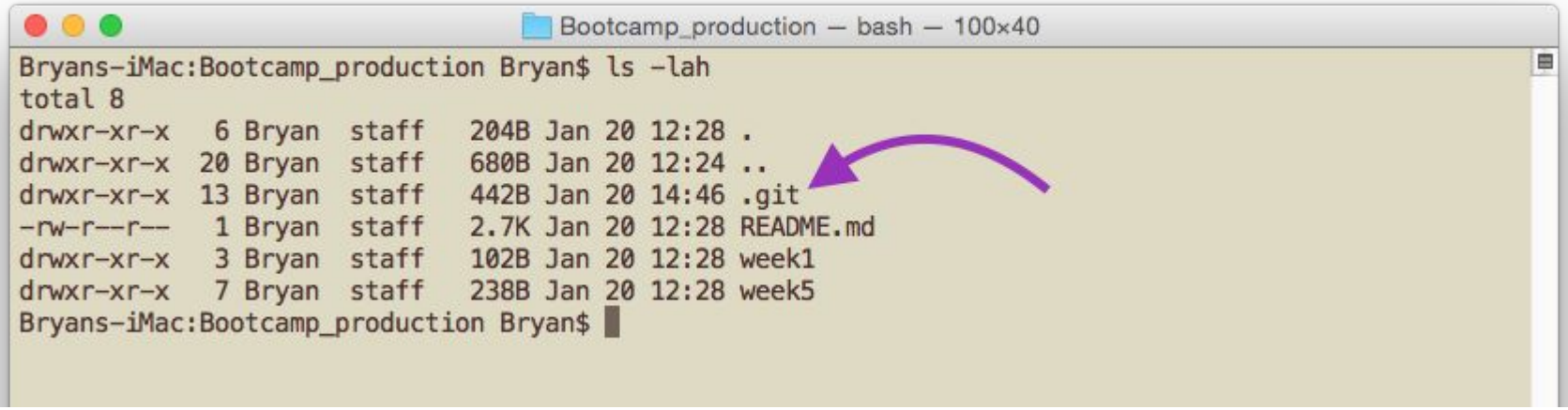
What is a Repository?

- A repository is a directory that contains other files and subdirectories on some volume.



What is a Repository?

- Plus the local, sometimes hidden .git database within.



- Note: On Windows, the .git folder is usually visible. On Linux and Mac, all '.name' folders are hidden by default.

Create a Repository

- There are two ways to create a repository:
  - Create a local repository directly:
    - Create a repository locally using the git init command.
    - Later, add a remote repository reference to the configuration, and push data to that remote.
  - Clone a remote repository to your local machine.

Create a Local Repository

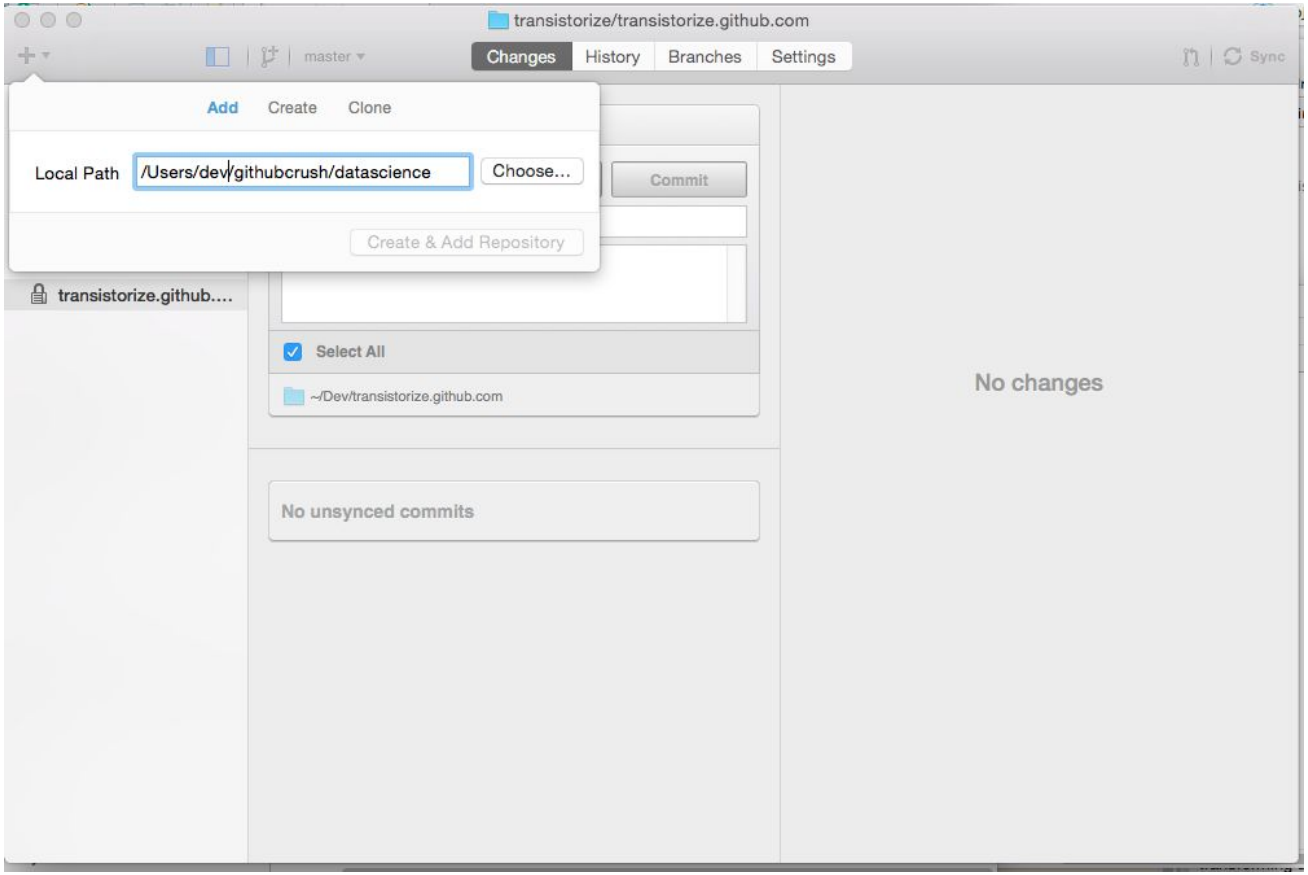
- **Step 1**: Choose a working directory and initialize an empty directory
  - Type the following commands one at a time. Everything after the double forward slash (//) are just comments.

```
$ pwd                  // show current directory
/c/Users/dev/githubcrush
$ mkdir datascience    // create empty directory
$ cd datascience       // enter the directory
$ git init             // initial new repository, an empty repository
Initialized empty Git repository in c:
/Users/dev/githubcrush/datascience/.git/
```

- Note: The git init command is used to change a local directory into a repository managed by Git.

Create a Local Repository

- Optionally, add the repository to the GitHub UI.



Add Content to the Local Repository

- **Step 2**: Create a new text file using your text editor, create a file named data.txt within the datascience directory you created before. Add the following content to this file:

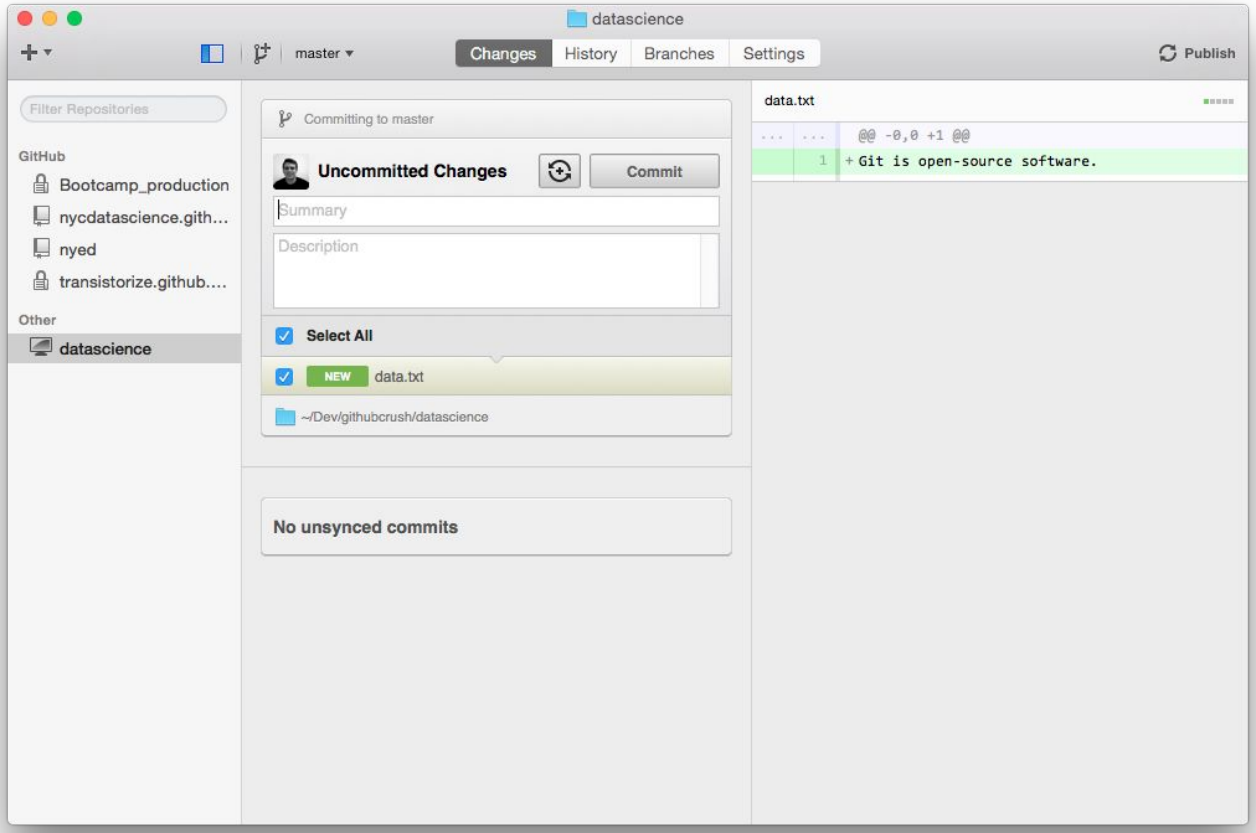
```
Git is open-source software.
```

- Note: on Windows, make sure Notepad++ is configured with the settings: UTF-8 without BOM.
- FYI: On Linux and Mac, you can create a file via the command line.

```
$ touch data.txt        // creates an empty file
$ echo "Git is open-source software." > data.txt
$ ls -a                 // lists files in the directory
.   ..      .git        data.txt
```

Status of the Local Repository

- Verify in the UI that a new file has been created.

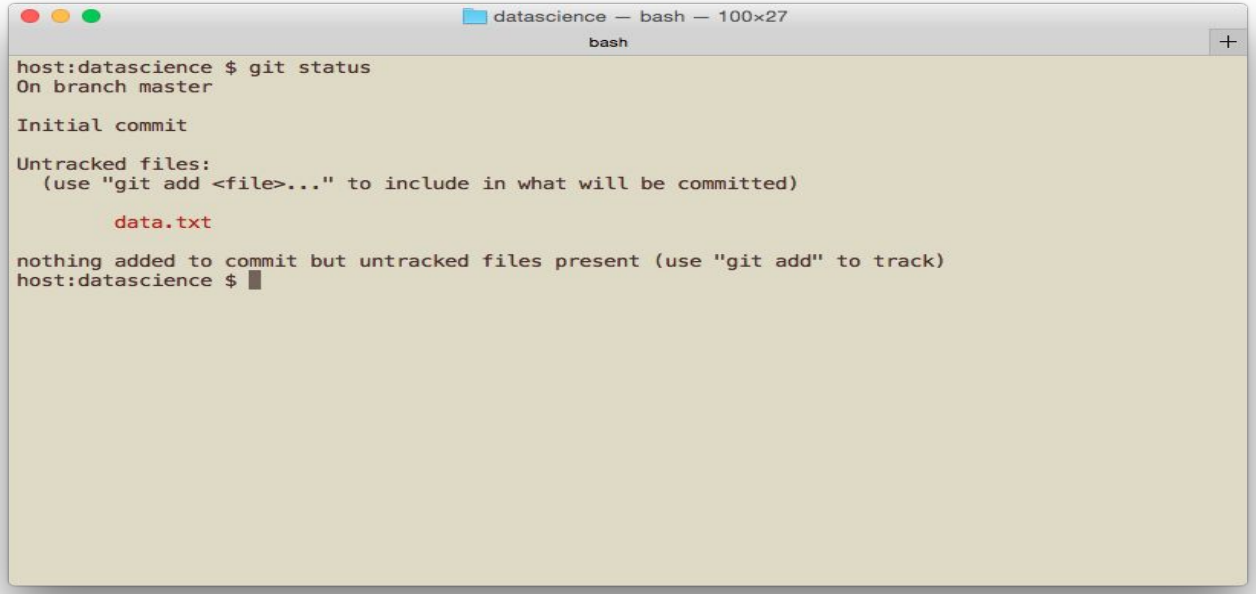


Status of the Local Repository

- The status command is used to show what changes git has detected in the local working directory.

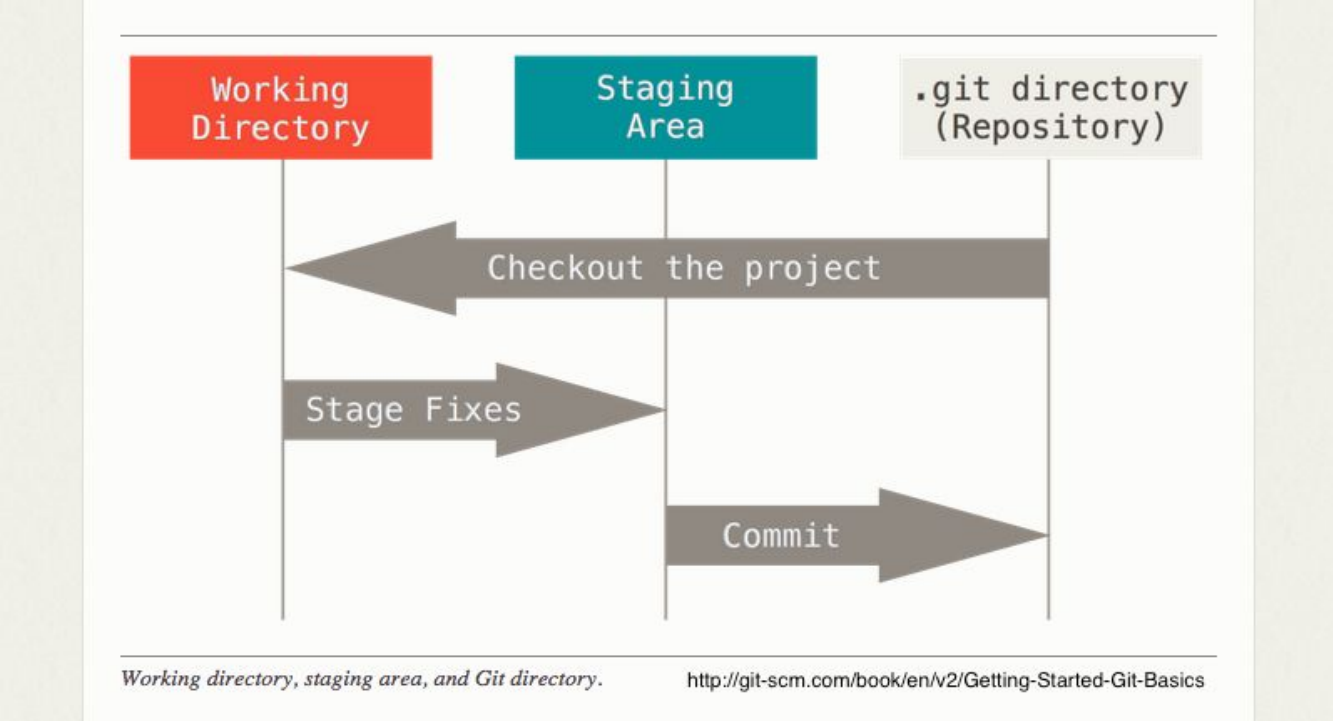
```
$ git status          //examine git's point of view
```

- What's going on here? What does it mean to be untracked?



Git Model: Working Directory

- Untracked or modified items have to be added to the index file in the .git repository, a.k.a. the staging area



Working directory, staging area, and Git directory. http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

- Note: Later, we will learn how to go straight from creating or modifying a file to commit it.

Add a file to the staging area

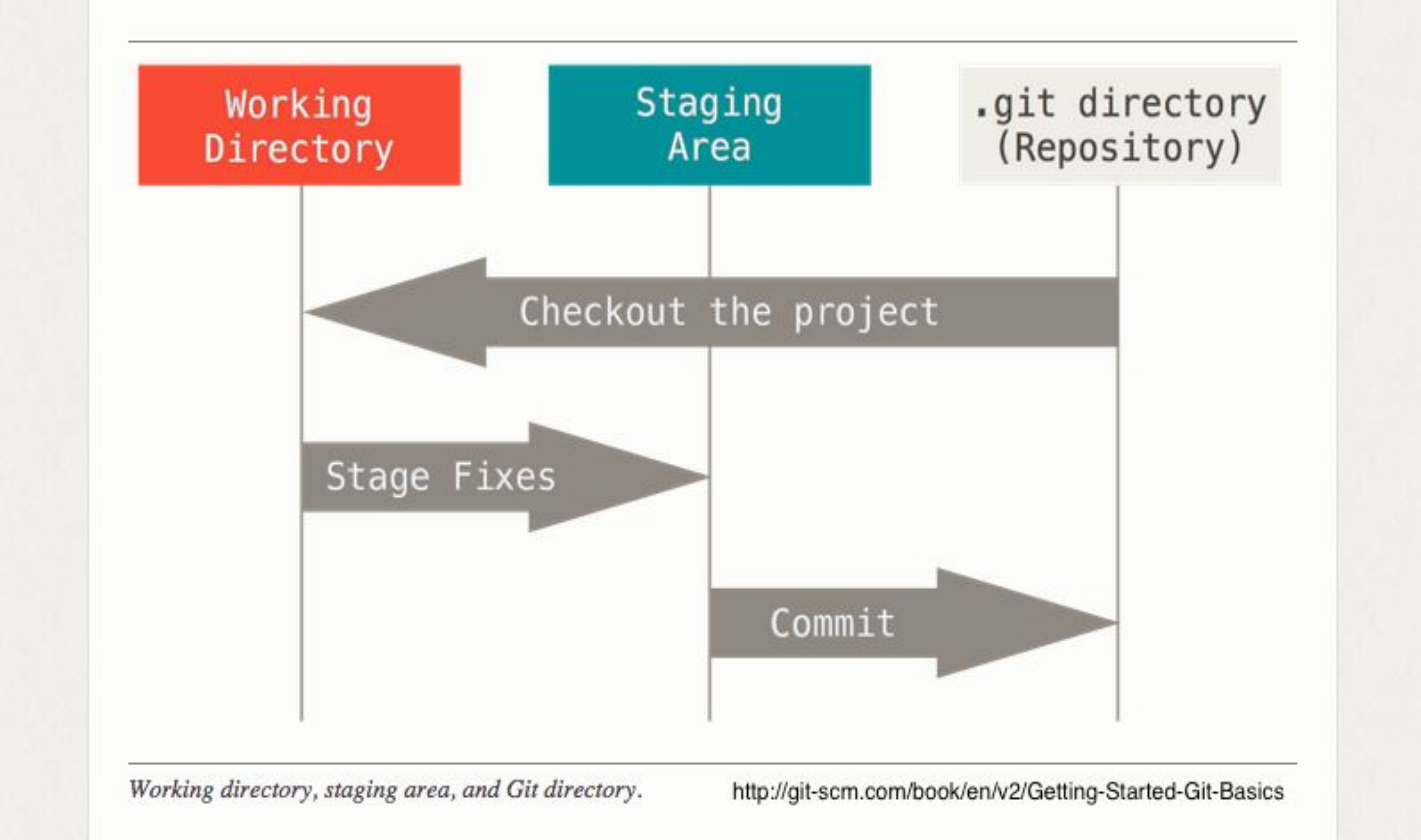
- **Step 3**: Once the file is created, we must add it to the staging area and then commit the file to the repository.

```
$ git add data.txt     // add file to staging
$ git status           // view status of current working
directory
On branch master       // name of branch currently checked out
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   data.txt
```

- Note : The command git status tells us a new file was added and reminds us to commit it. Also, we can cancel the tracked status of the file by using the command `git rm --cached`

Git Model: Staging Area (Index)

- The staging area (index) tracks all the modified file content.



Working directory, staging area, and Git directory. http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Commit Content to the Repository

- **Step 4**: Commit the file to the local repository

```
$ git commit -m "Add new data."
[master (root-commit) 2a6487f] Add new data to the project.
 1 file changed, 1 insertion(+)
 create mode 100644 data.txt
$ git status
On branch master
nothing to commit, working directory clean
```

- Note : After the parameter -m in git commit, provide a short, clear description of this change. It's makes it easier to find and understand changes in the history log. Also, we know the commit ID of the file on the master branch begins with 2a6487f. The description of this file submitted is 'Add new data to the project'.

Commit History

- View the commit log to see the recent history

```
$ git log --graph --decorate
* commit 2a6487f3bd883297451f3c6412b81555ef45942f
  Author: user.name <my.email@domain.com>

      Add new data.
```

- Note: Try `$ git log --oneline` You can mix flags to find your output preference. See git help log for more details.

Exercise 1

- Create a new directory that is a sibling to the datascience directory. Call it exercise_1. Initialize that directory to be another git repository. If it helps, add that second repository to the UI tool.
- Create a new empty file in the exercise_1 directory. Add and commit this new file to your second repository.
- What branch did you commit this change to?
- Change directories back to the datascience directory. What is the status of the datascience repository?

Change Content in the Repository

- **Step 1**: Create multiple files using a text editor.
  - Modify data.txt file, and append this line to the file: **R is open-source software.**
    - Create the new file www.txt to have the string 'www'
    - Create the new file test.txt to have the string 'test'
    - Run `$ git status` Are all the files treated the same?

Change Content in the Repository

- **Step 2**: Try adding multiple files simultaneously to the staging area
- Using the --all parameter, add all files to the repository simultaneously.

```
$ git add --all
```

- Run `$ git status` What do you find?

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:   data.txt
  new file:   test.txt
  new file:   www.txt
```

Reset a File from Staging

- **Step 3**: Reset the files you just added, out of the staging area.

```
$ git reset HEAD data.txt test.txt www.txt

$ git status
```

- Note: In this case, this command does not modify the actual files. For now, ignore the mention of the HEAD reference.

Reset a File from Staging

- **Step 4**: Re-add all the files you just reset, back into the staging area.

```
$ git add data.txt test.txt www.txt

$ git status
```

Change Content in the Repository

- **Step 5**: Commit all the files as one change. Pick an appropriate message.

```
$ git commit -m "..."
```

- When nothing is modified or untracked, it is said you have a clean working directory.

```
$ git status
On branch master
nothing to commit, working directory clean
```

Change Content in the Repository

- Without looking at the slides, what are the steps to creating or modifying content within a repository?
- Could I add an image to the datascience repository?
- When you touch a file, does git consider that a modification? How can you tell?

Commit a Modified File in One Command

- 'Skipping' the add-to-staging part
- Adding the -a parameter to git commit will commit all known files to the repository simultaneously. A file is known if it has been added and committed to the git index before. Still, be careful when using this combination.
- Modify test.txt to read as follows:

```
test! test! test!
```

- Then execute:

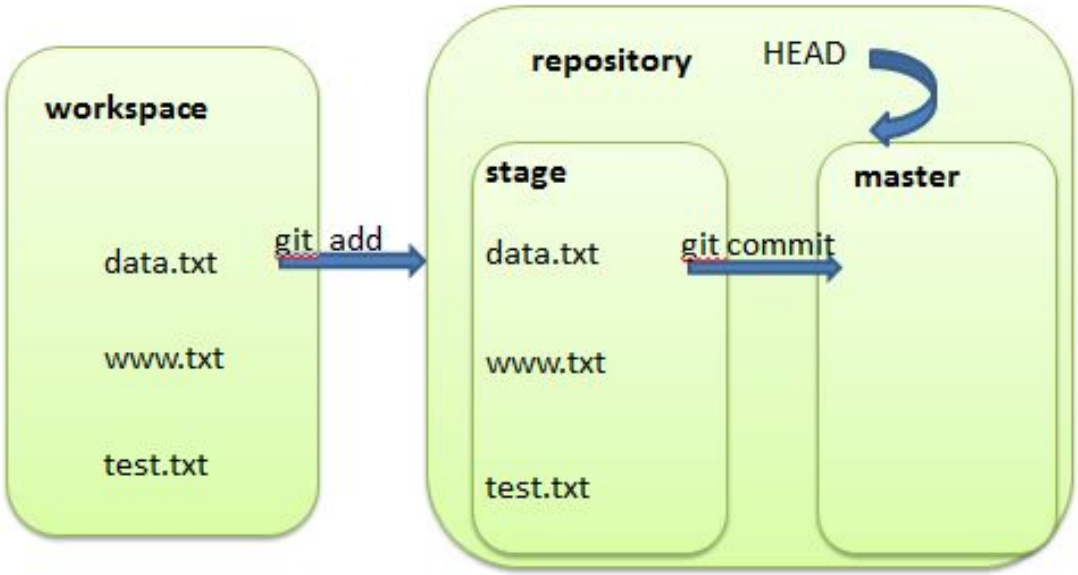
```
$ git commit -a -m "Commit more tests"
[master ed1d0f8] Commit more tests
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Quick Review

- In order to better understand the purpose of Git commands, keep the following model in mind:
  - **Branch**: A stream of file snapshots, like a tree branch. Only one branch is active at a time in a repository, often defaults to the master branch.
  - **Workspace**: What we see in the directory, the datascience folder created before is the working area.
  - **Staging**: Used to prepare a commit of one or more files, and you can review the change list.
  - **Repository**: There is a hidden directory called .git in the workspace, that is the 'real' repository. Keeps track of snapshots, history, the index, the HEAD pointer (references), branches, and more.

Quick Review - Workflow

- The workflow figure of adding and committing files to repository.
  - Firstly, git add adds files in the workspace to staging.
  - Secondly, git commit commits files in staging to the current branch.



Outline

- Set up Git and GitHub
- Introduction to Git
  - Creating a Git Repository
  - **Manipulating files**
- Introduction to GitHub
  - Lightning Tour of Github
  - Create a Remote Repository

Change a Local Repository - Delete a file

- We take the test.txt file, committed under datascience, for example.
  - the delete command is rm [file name]

```
$ rm test.txt

$ ls -a
.   ..      .git        data.txt    www.txt
```

Change a Local Repository - Delete a file

- When you delete the test.txt file, view the current status of the workspace.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -
a")
```

Change a Local Repository - Delete a file

- See the difference between the current workspace and the latest version of the repository.

```
$ git diff HEAD -- test.txt
diff --git a/test.txt b/test.txt
deleted file mode 100644
index e69de29..0000000
```

Change a Local Repository - Delete a file

- Command rm test.txt just deleted the file in the workspace, but did not remove it from the repository, so:
  - If you want to restore files to the workspace, use the command:

```
$ git checkout -- test.txt
```

- Note : Command git checkout -- [filename] can restore the file to the workspace, so you can undo accidental, non-commited changes. The --, in this case, is a generic way to refer to the current branch.

Change a Local Repository - Delete a file

- If you really want to remove the file from the repository, use the git rm command:

```
$ git rm test.txt
rm 'test.txt'

$ git commit -m "delete test.txt"
[master b81549a] delete test.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 test.txt
```

- Note: When you delete a file, commit the updated status of the file. When you do this, you are removing the file from future snapshots, but the previous versions of the file are still in the history where it can be retrieved.

Change a Local Repository - View Changes

- We completely overwrite the contents of data.txt, by changing its contents to:

```
R is software.
R is free software.
```

- Before adding file to the staging area, view its status :

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)
        modified:   data.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Change a Local Repository - View Changes

- After modifying the file within the workspace, view the status of the file.
  - if the file is unstaged, git diff views the difference between the current workspace and last known snapshot:

```
$ git diff -- data.txt
diff --git a/data.txt b/data.txt
index 15b81a8..a5c20b1 100644
--- a/data.txt
+++ b/data.txt
@@ -1,2 +1,2 @@
-Git is open-source software.
-R is open-source software.
+R is software.
+R is free software.
```

Change a Local Repository - View Changes

- After modifying the file within the workspace, view the status of the file.
  - if the file is staged already, have to add the --cached parameter:

```
$ git add data.txt

$ git diff data.txt

$ git diff --cached data.txt
diff --git a/data.txt b/data.txt
index 15b81a8..a5c20b1 100644
--- a/data.txt
+++ b/data.txt
@@ -1,2 +1,2 @@
-Git is open-source software.
-R is open-source software.
+R is software.
+R is free software.
```

Change a Local Repository - View history

- Firstly, we add the modified file data.txt and commit it to the repository.

```
$ git add data.txt
$ git commit -m "Add free to data"
```

- Note：Under Windows, sometimes adding a file to staging outputs the following warning：

```
$ git add data.txt
warning: LF will be replaced by CRLF in data.txt.
The file will have its original line endings in your working
directory.
```

- It doesn't matter, just ignore it. You can enforce this behavior by issuing the command:

```
$ git config core.autocrlf true
```

Change a Local Repository - View history

- If we don't want too much information, output can be controlled by the parameters --oneline, or to see the full SHA-1 commit IDs, use --pretty=oneline.

```
$ git log --pretty=oneline
96f01db1d6e026b7284a7d1267c1adb70b20aa42 Add free to data
063624c7165dc5a6c6f5949ab7cc82a7d8a28036 delete test.txt
ed1d0f8ad270606bda51290a1bfed37f40f15469 Commit more tests
4d8db1c6129c836e48c36d265bcb938bc07abf21 Create three more
files.
a638a8aab1a396c6cbd8ab425e8a8976af498dfb Add new data.
```

- Note: Output information may not look exactly as above, the important part is that only the commit IDs and description are printed.

Change a Local Repository - Rename a file

- Git can track file renames and also handle moving ('mv') files and directories about.

```
$ git mv www.txt www2.txt  // make sure www.txt was committed
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  renamed:    www.txt -> www2.txt
...
```

- Note: Note how the rename automatically added itself to the staging area. Now commit it.

Hide files from Git

- Sometimes, we want to tell Git to ignore certain files. These could be generated files, files with secret keys, or special IDE files that should not be committed. The .gitignore exists to hide these kind of files and prevent mistakes.

```
$ echo ".DS_Store" > .gitignore      // Ignore common OSX file.
$ echo "Thumbs.db" > .gitignore      // Ignore common Windows
file.
$ git add .gitignore; git commit -m "add gitignore file"
$ git status
```

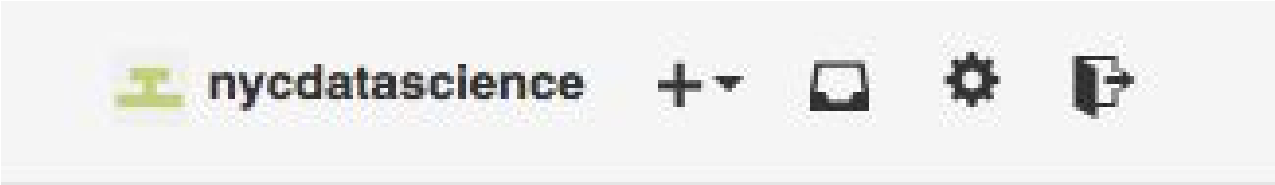
- Note: .gitignore files support a rich set of matching patterns, to match multiple files at any folder depth.

Outline

- Set up Git and GitHub
- Introduction to Git
  - Creating a Git Repository
  - Manipulating files
- Introduction to GitHub
  - **Lightning Tour of Github**
  - Create a Remote Repository

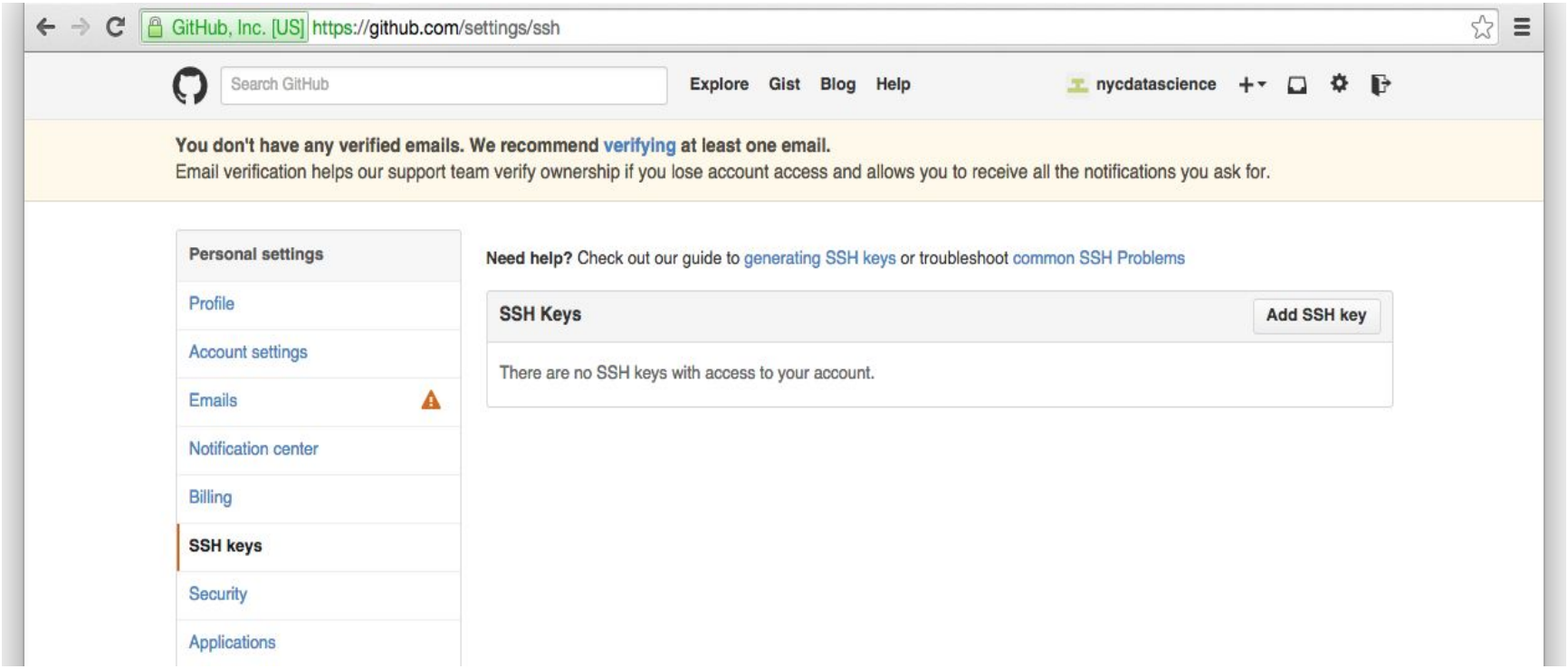
Sharing Secrets

- If sticking with the command line, need to manually configure SSH keys: https://github.com/settings/ssh/audit
- Or, click on the gear icon in the upper corner:

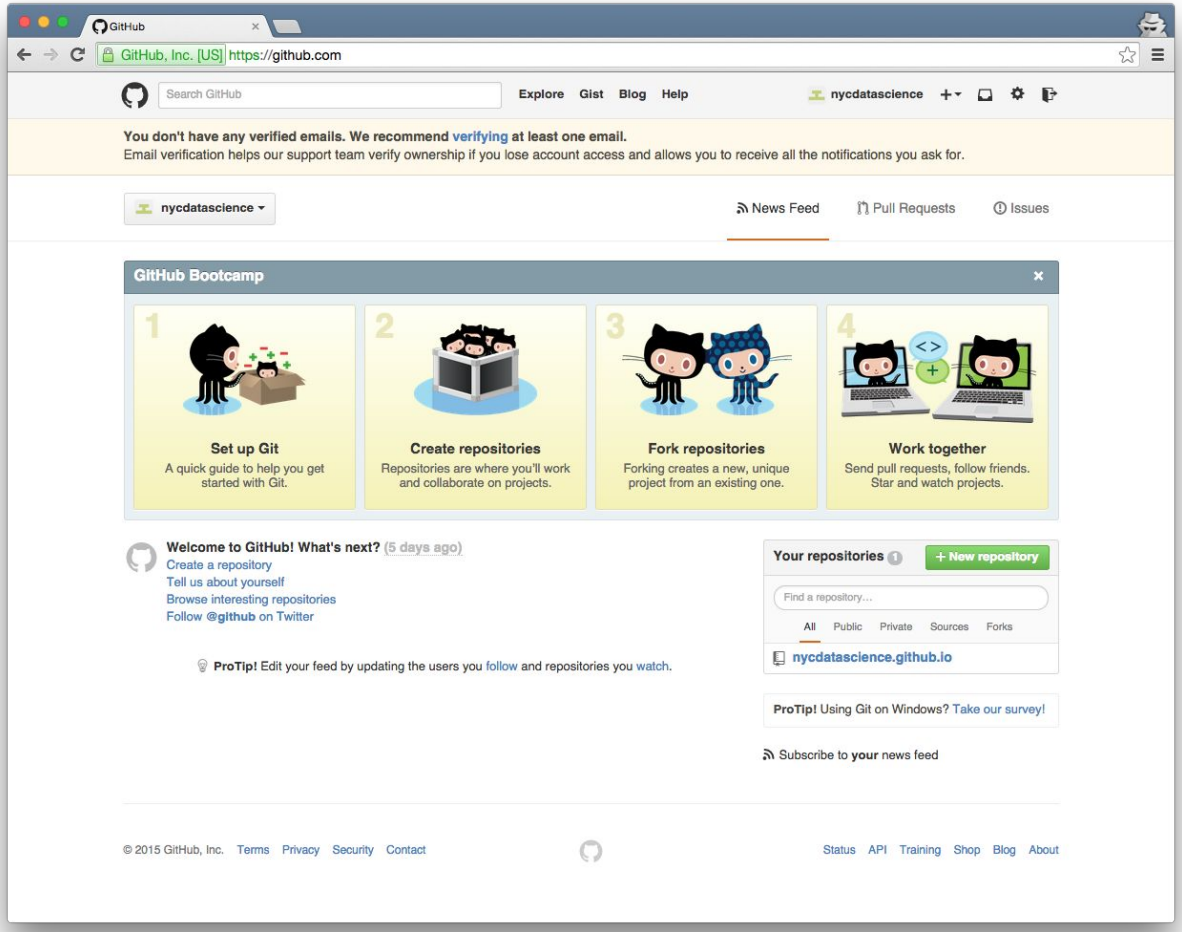


Adding SSH Keys to Github

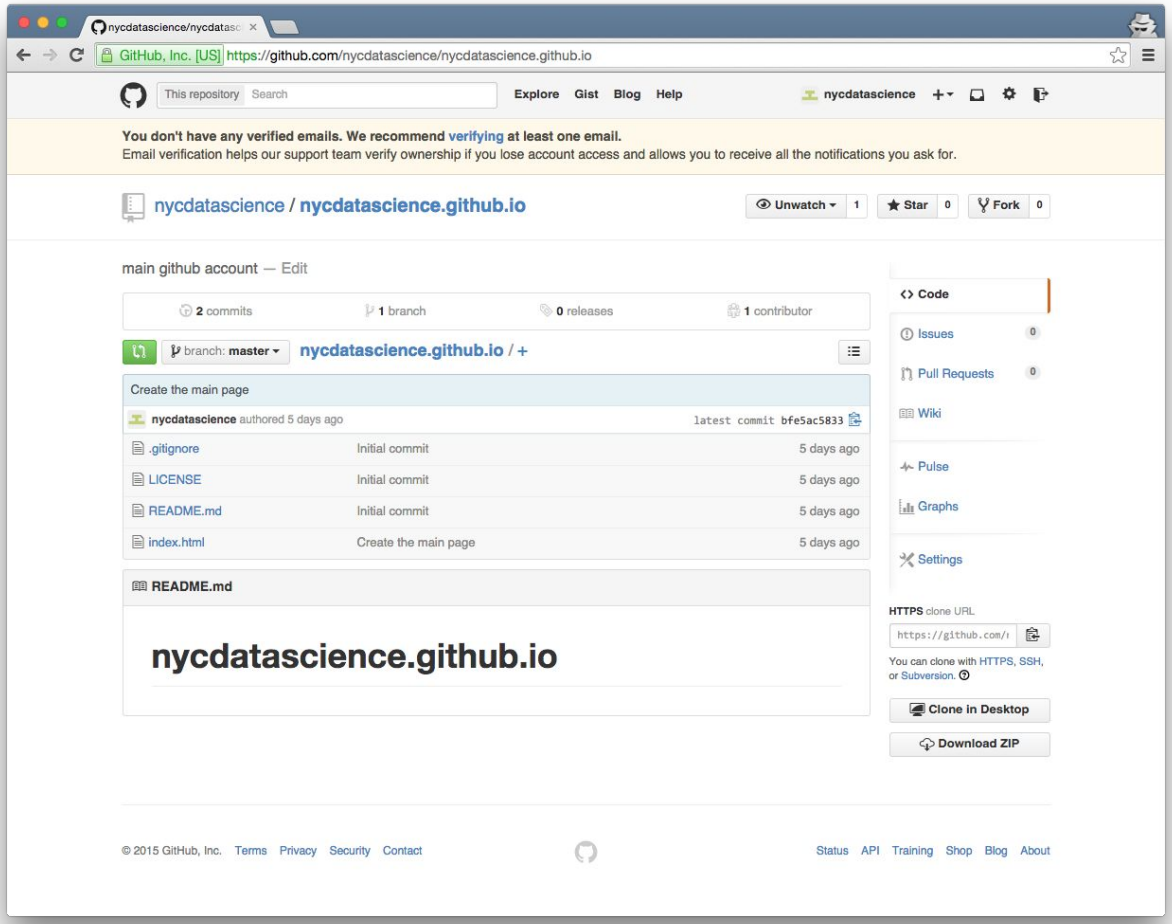
- Click 'generating SSH keys' link in the settings UI.



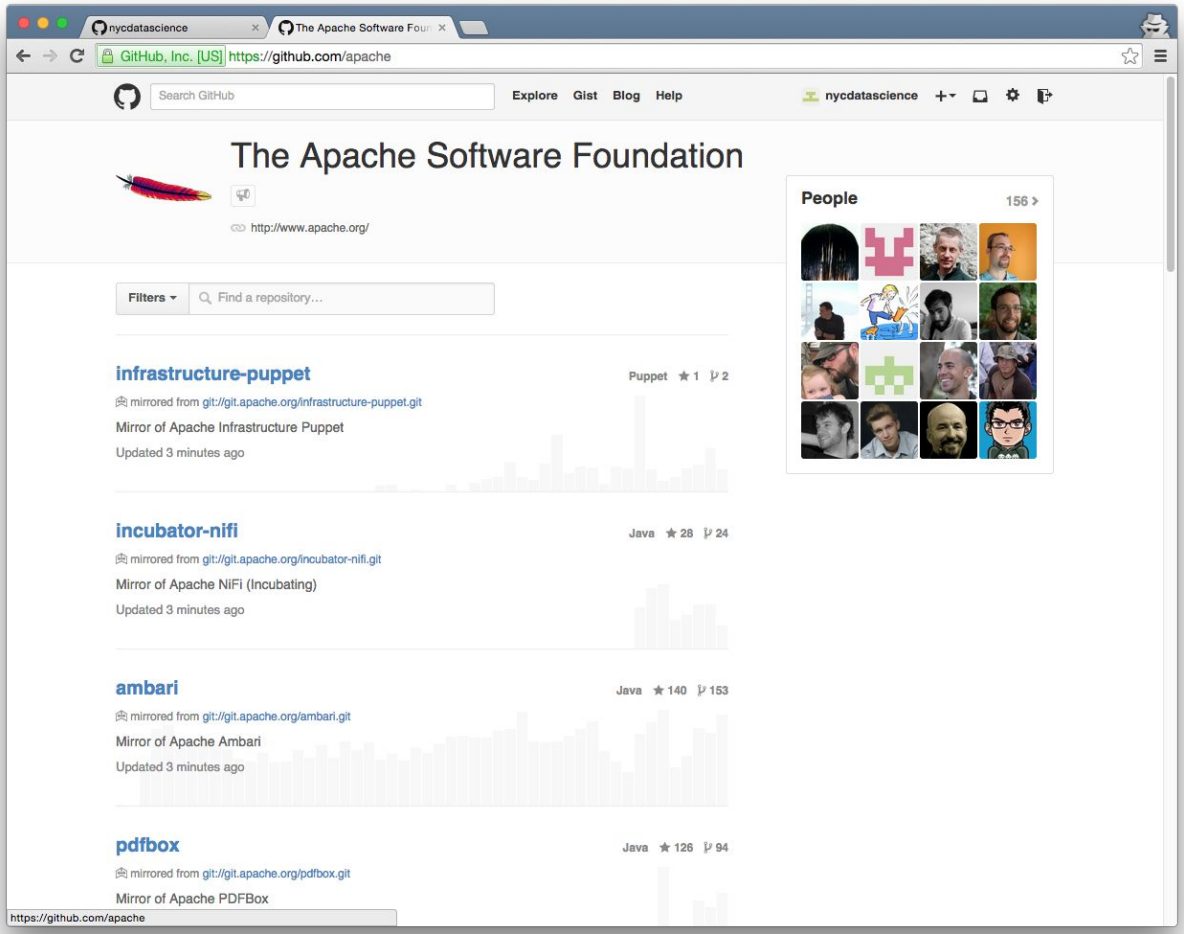
Quick Feature Tour - Main Page



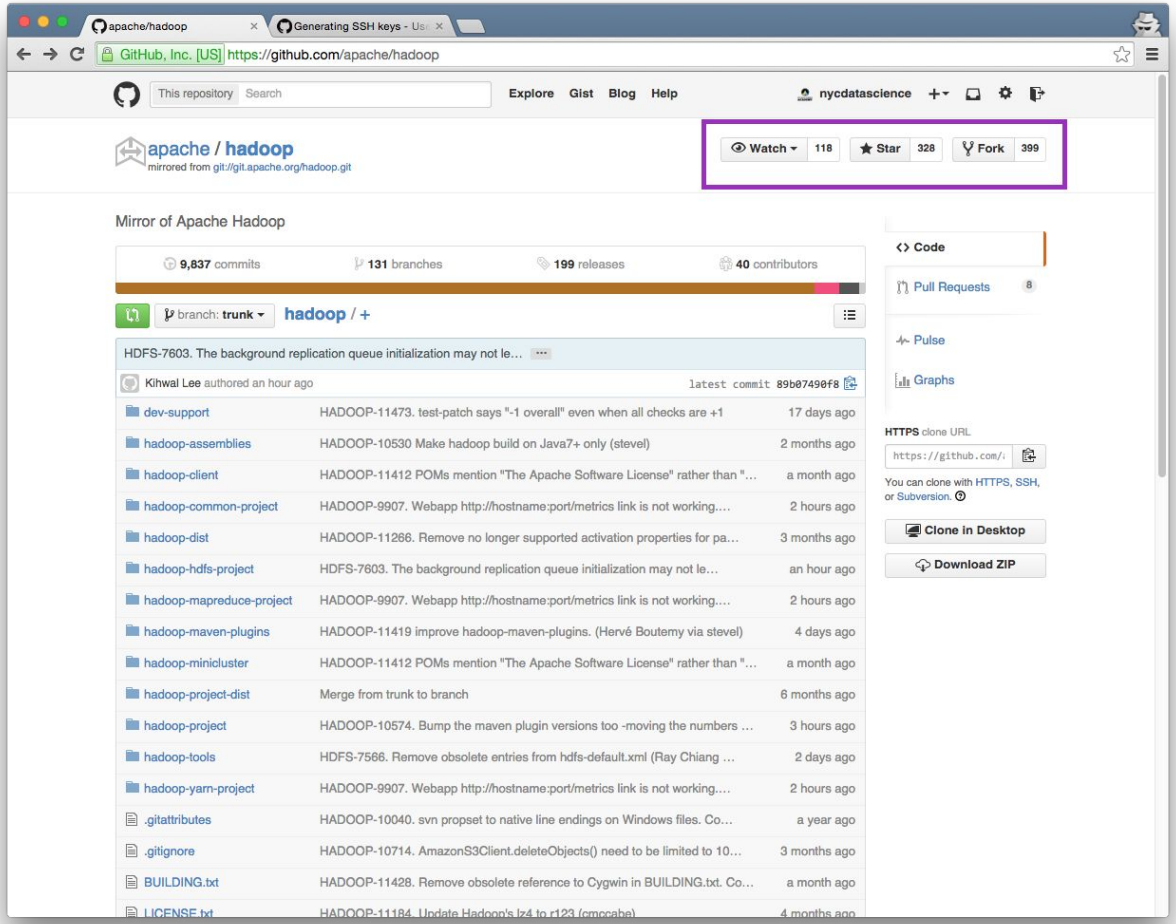
Quick Feature Tour - Personal GitHub Page



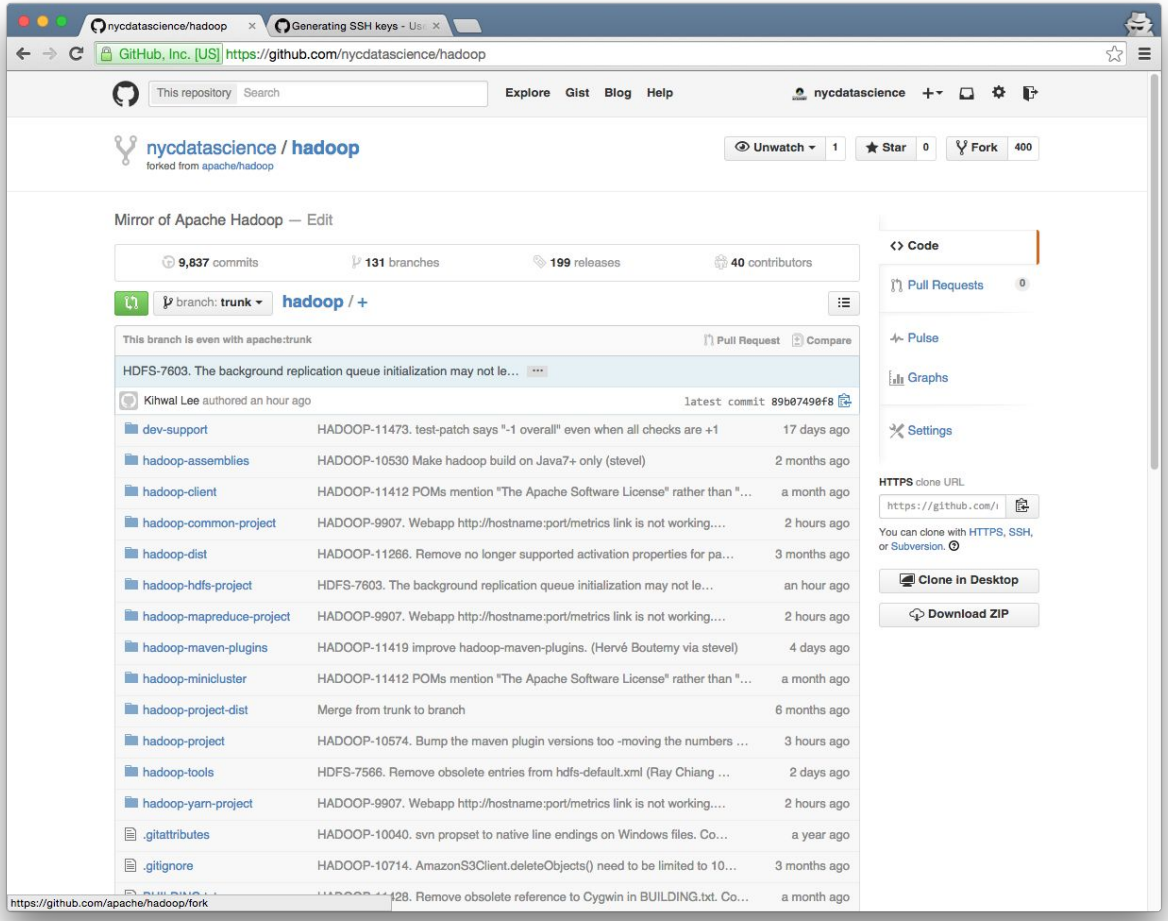
Quick Feature Tour - Organization Page



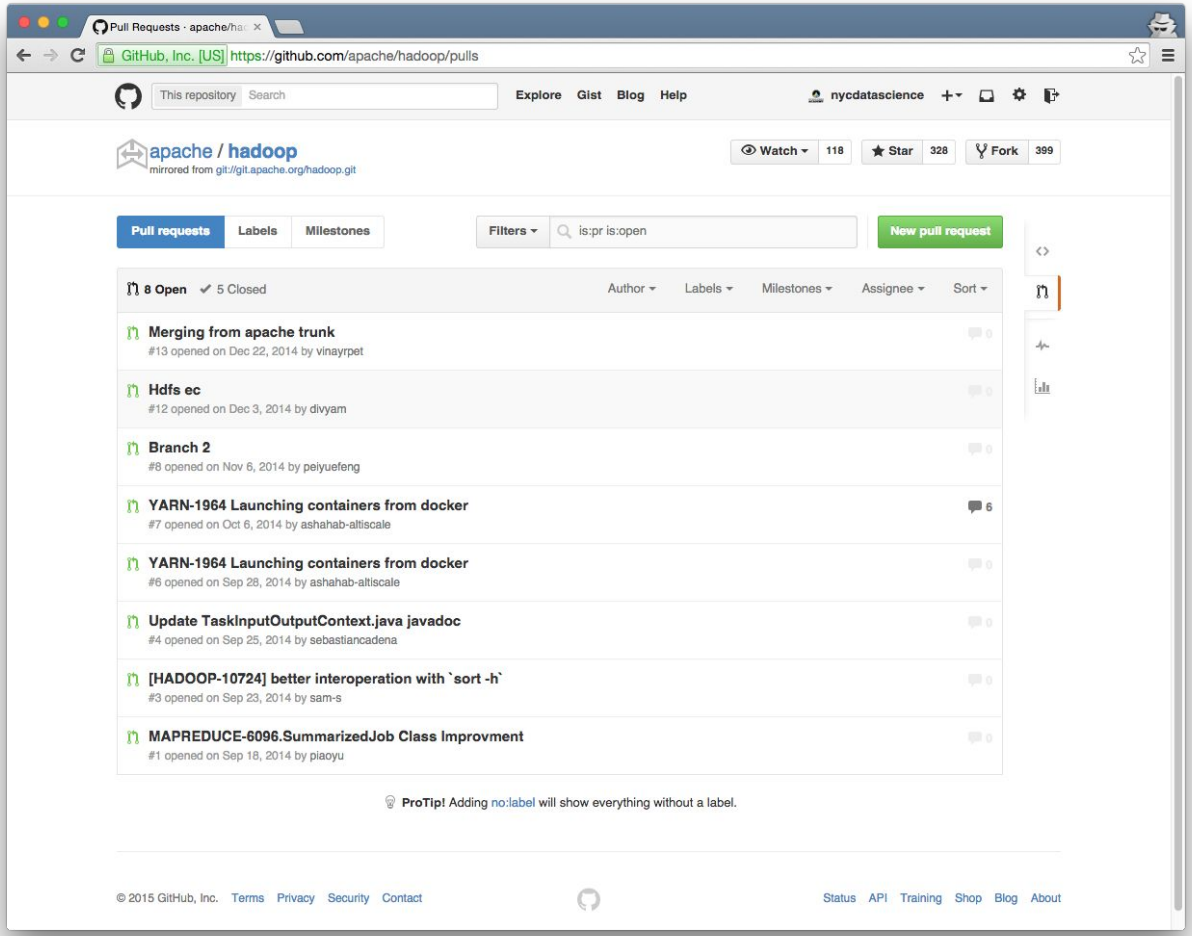
Quick Feature Tour - Forking Projects



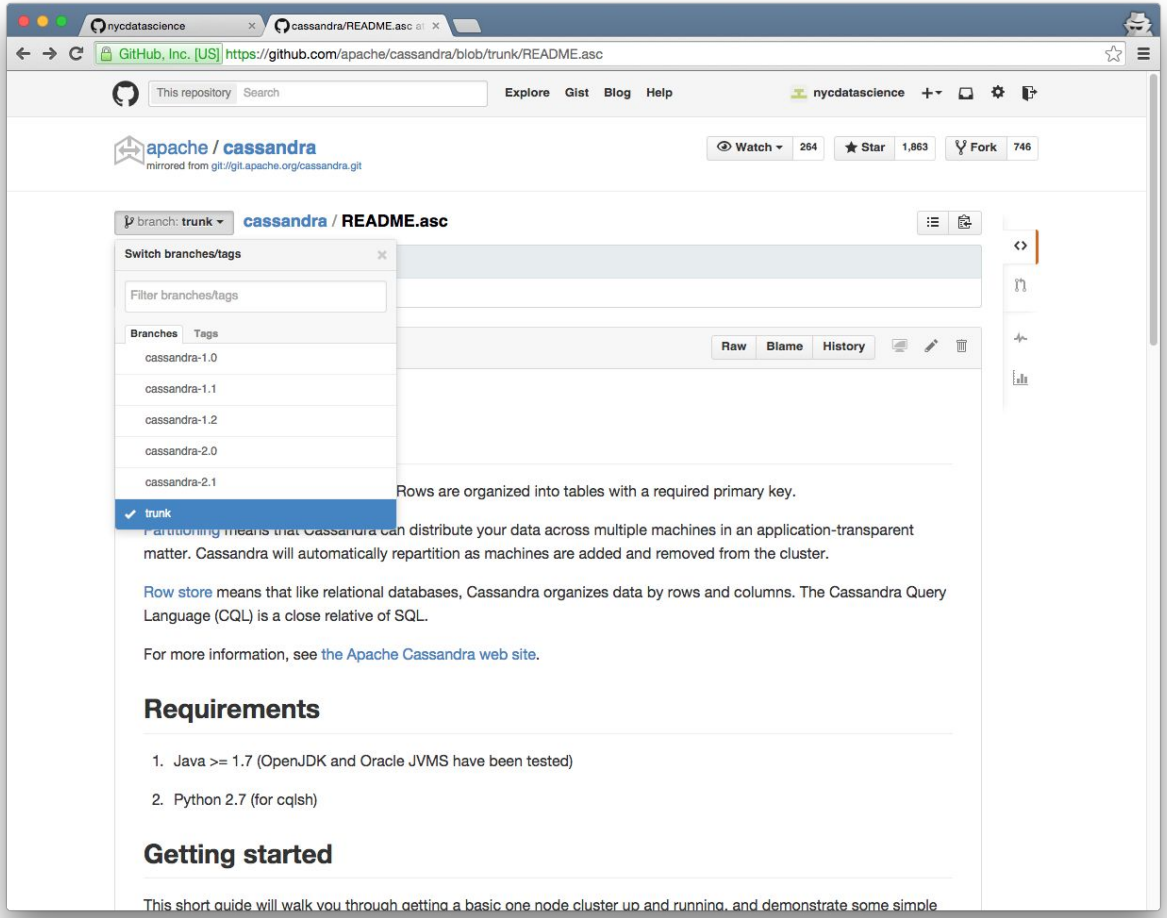
Quick Feature Tour - Forking Projects



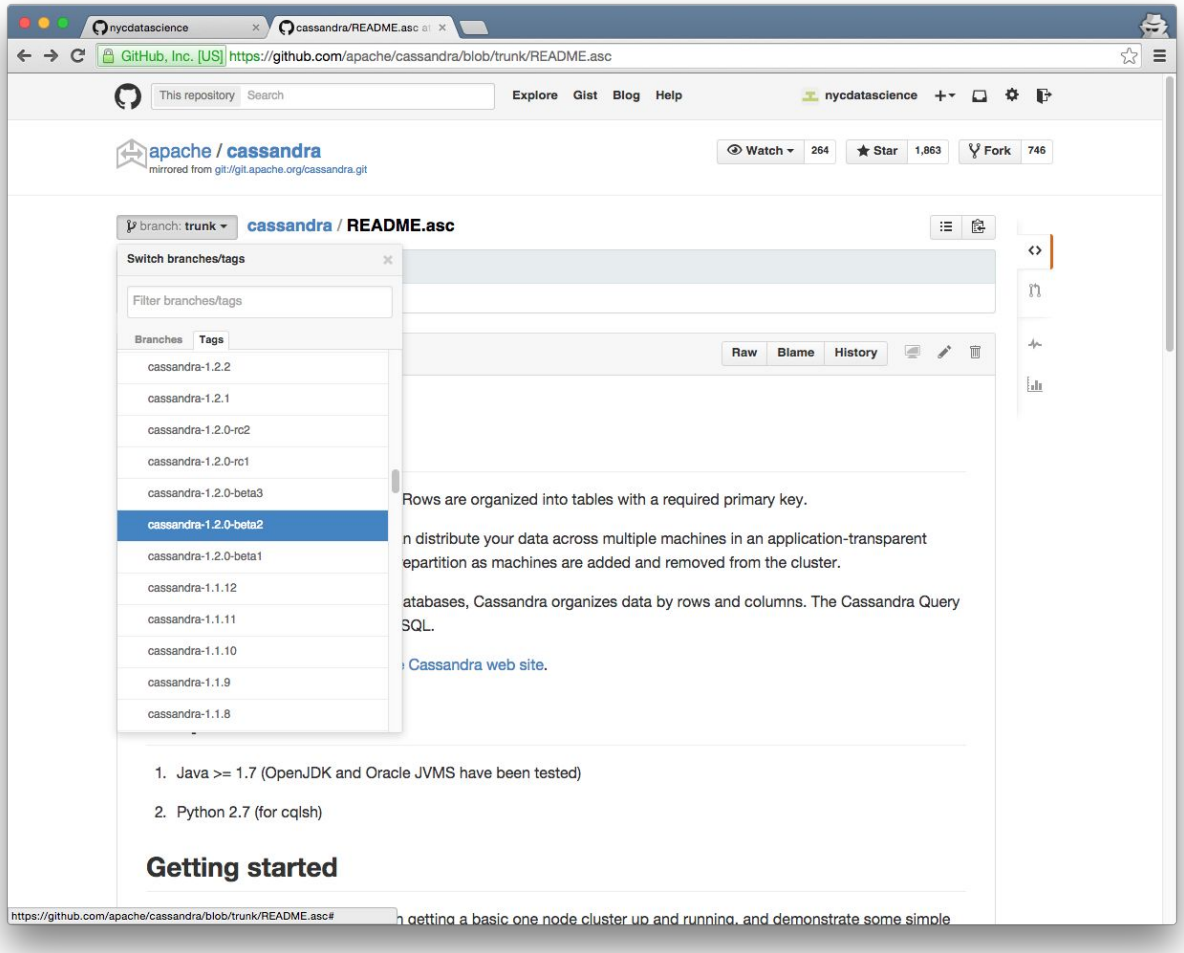
Quick Feature Tour - Pull Requests



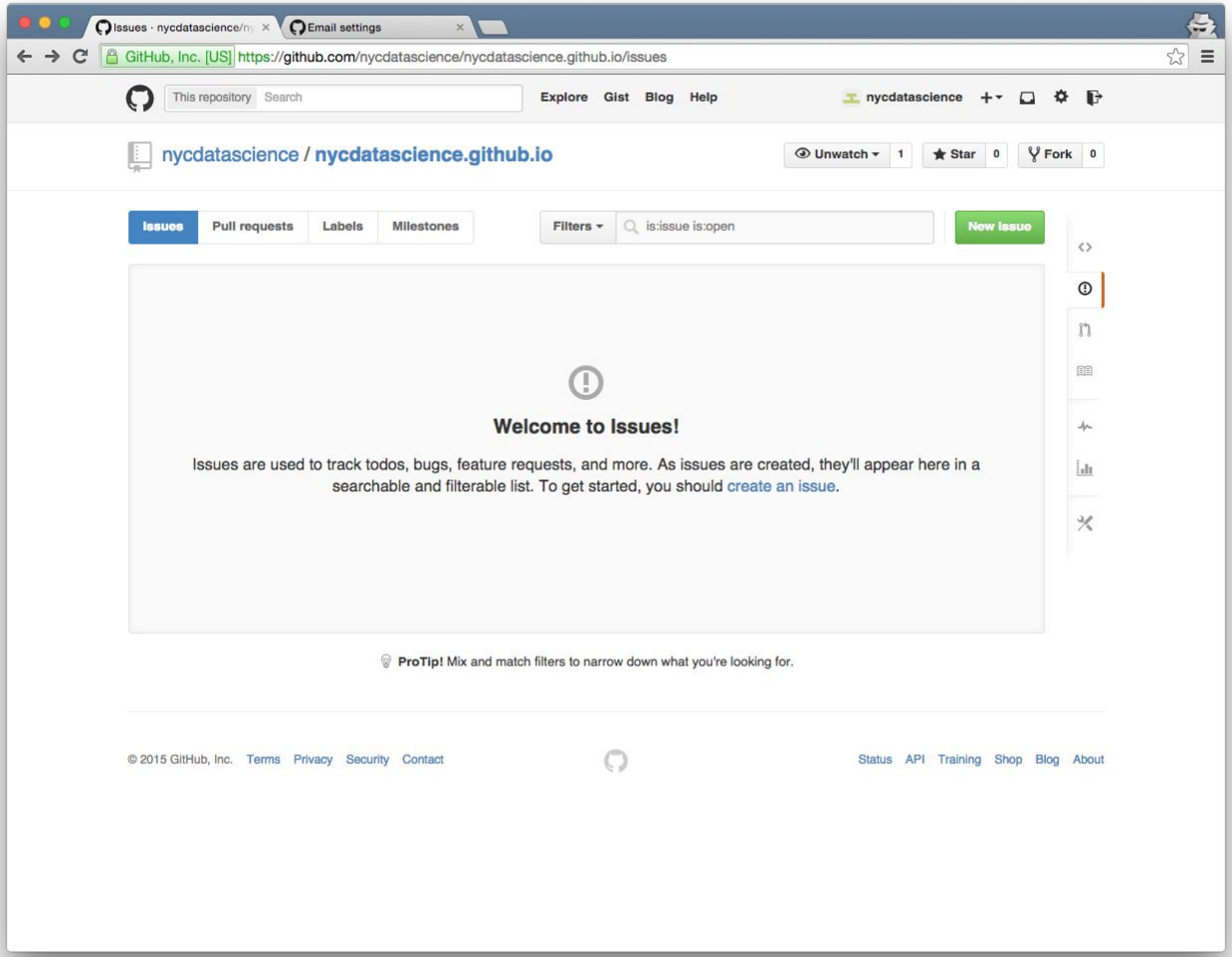
Quick Feature Tour - Example Branching Structure



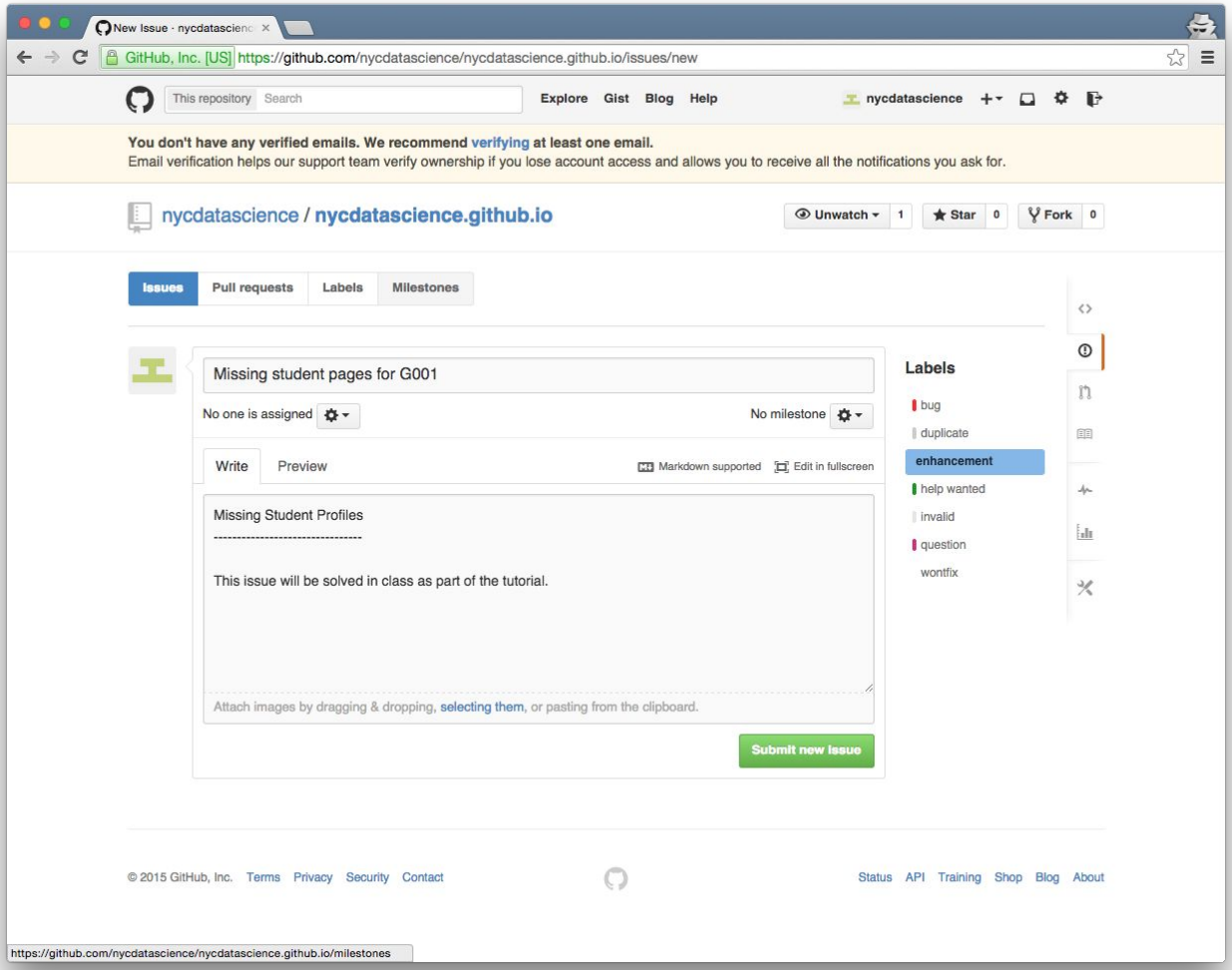
Quick Feature Tour - Example Tagging Structure



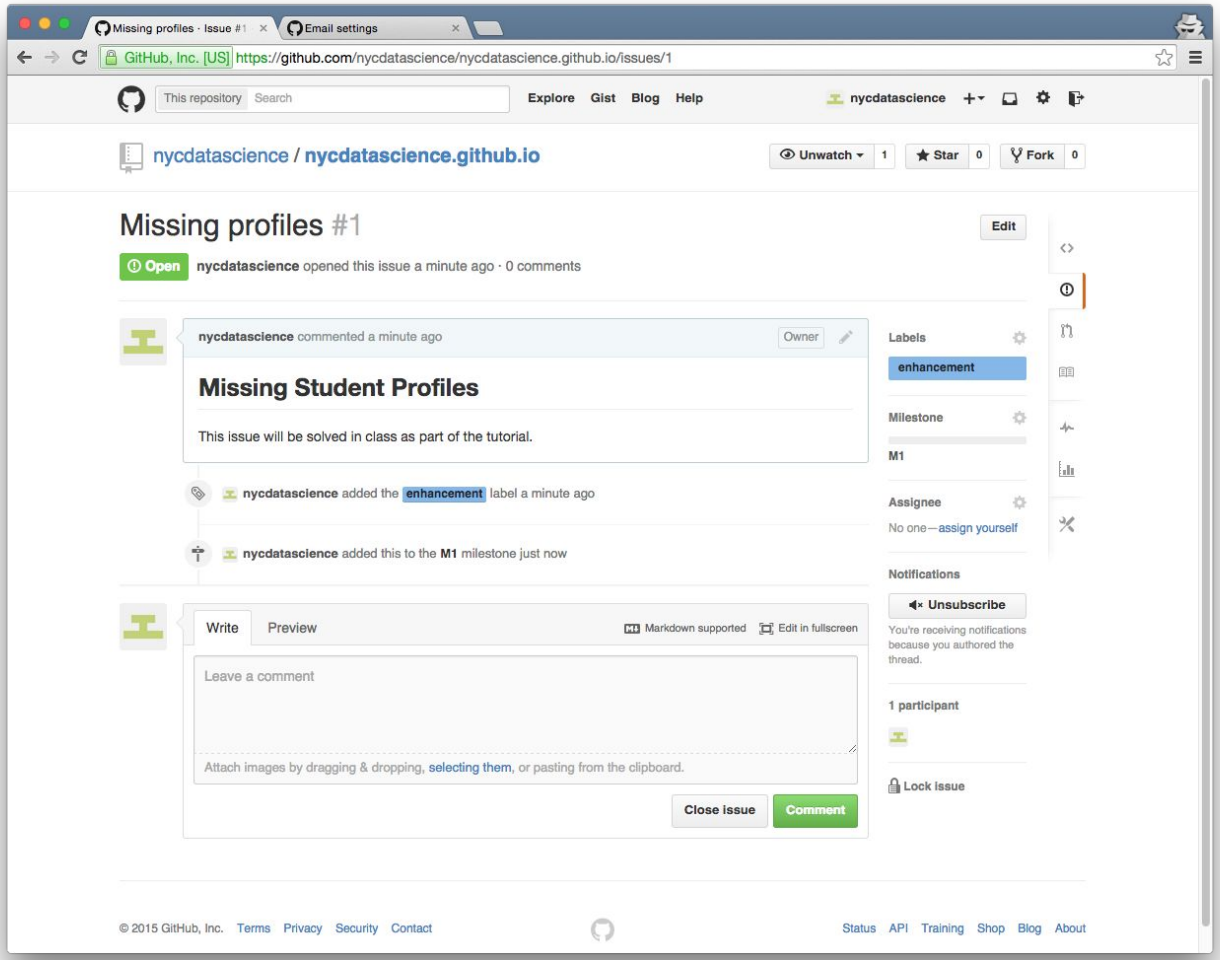
Quick Feature Tour - Issue Tracking



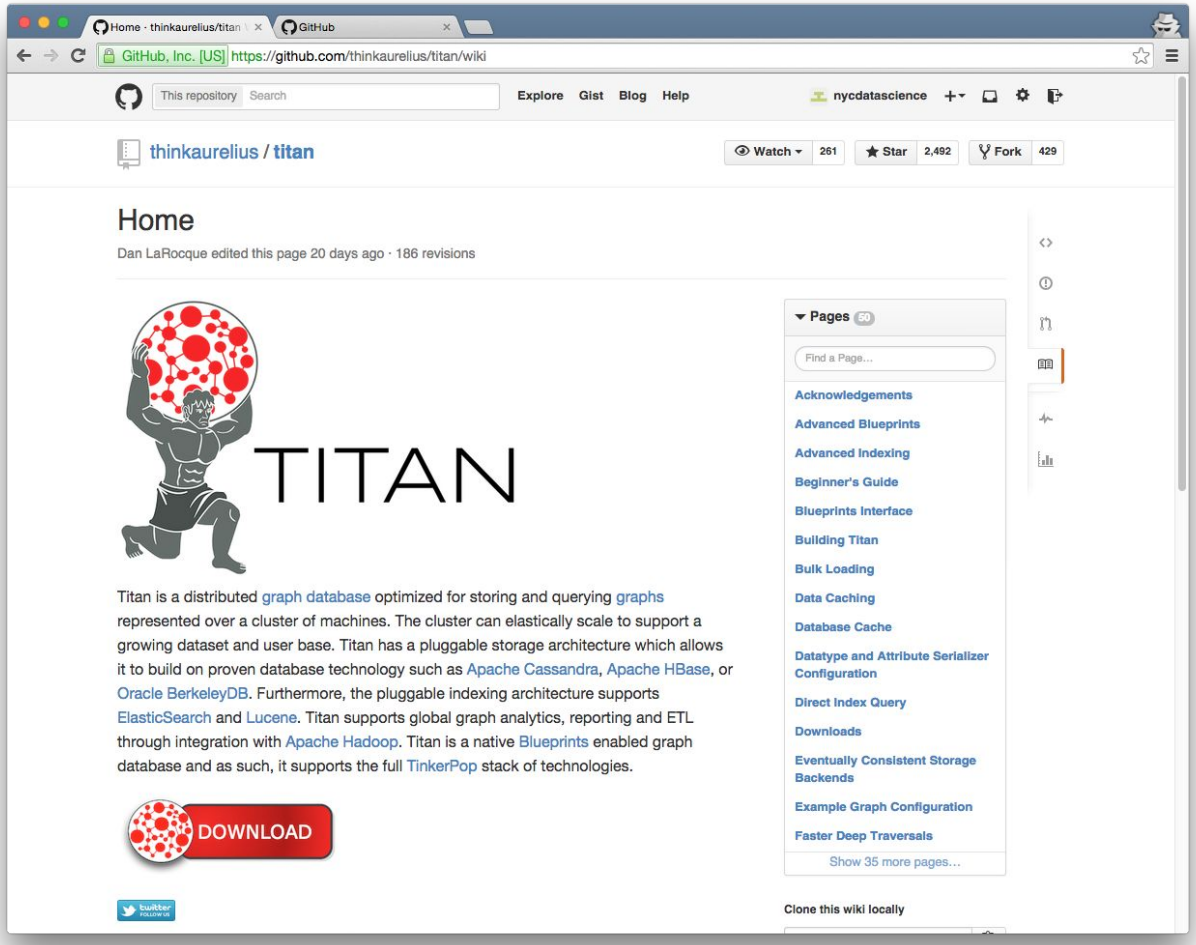
Quick Feature Tour - Issue Tracking



Quick Feature Tour - Issue Tracking

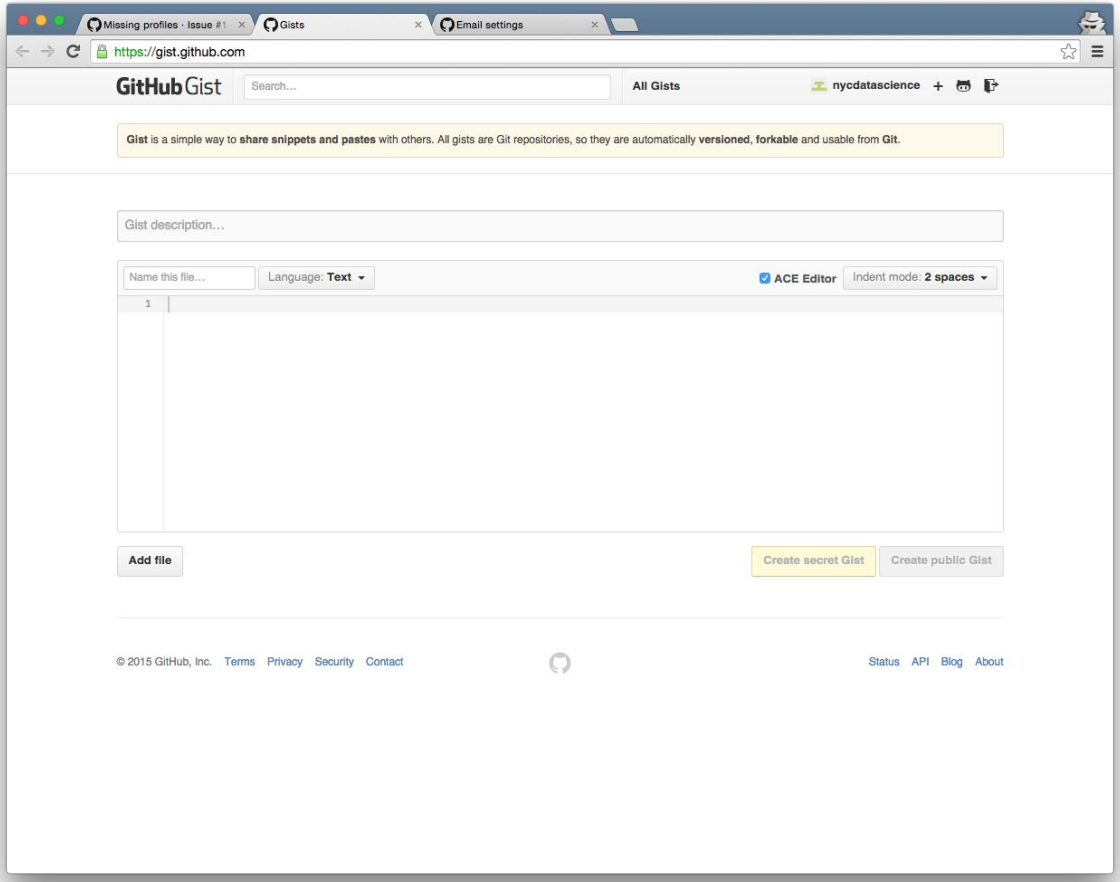


Quick Feature Tour - Wiki Pages



Quick Feature Tour - Gist

- Gist: Sharing quick snippets of code - https://gist.github.com/



Disk Quotas

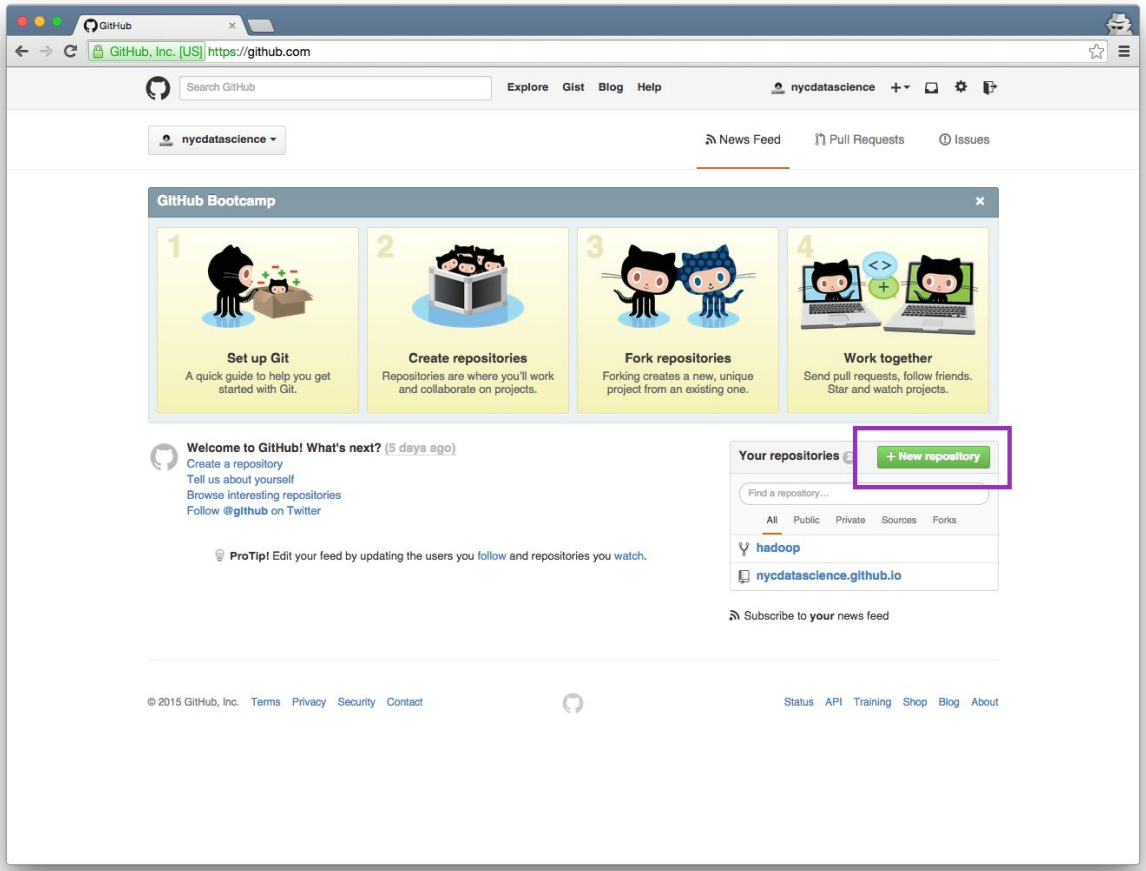
- How much information can you store on GitHub?
- Try to keep large datasets out of Git, and try to keep repositories below 100 MB. Instead, link to where a person can find a secure copy. See this help article.
- Also, for really large data sets, consider using BitTorrent, GoogleDrive, or DropBox with SHA-256 checksums for security.

Outline

- Set up Git and GitHub
- Introduction to Git
  - Creating a Git Repository
  - Manipulating files
- Introduction to GitHub
  - Lightning Tour of Github
  - **Create a Remote Repository**

Create a Remote Repository

- Look for the **+ New repository** button.



Create a Remote Repository

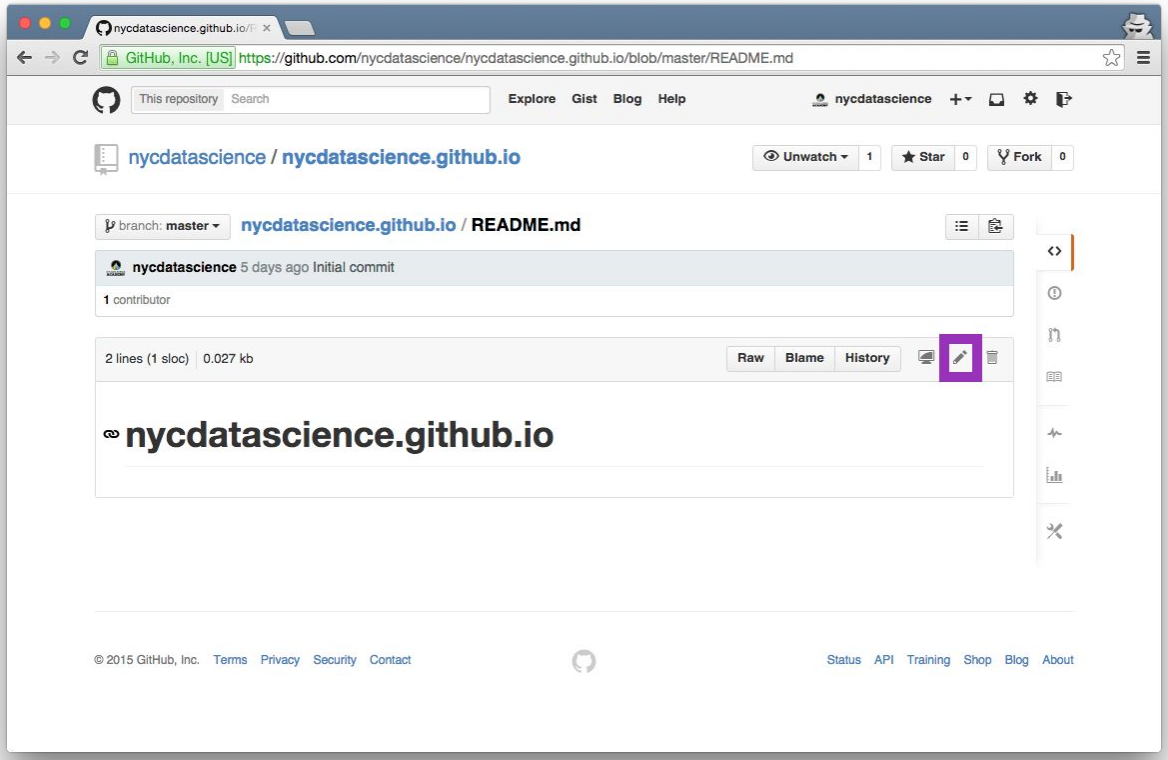
- For the repository name, create the domain name for your personal page, using your personal github username, in the following format your_username.github.io
- username: nycdatascience

```
repository name = nycdatascience.github.io
```

- Click on the checkbox Initialize this repository with a README. The license choice is up to you.
- Click the Create repository button.

Edit From the Site

- Once the new repository has been created, click on your repository's README.md file. For example, this was the NYC Data Science README:



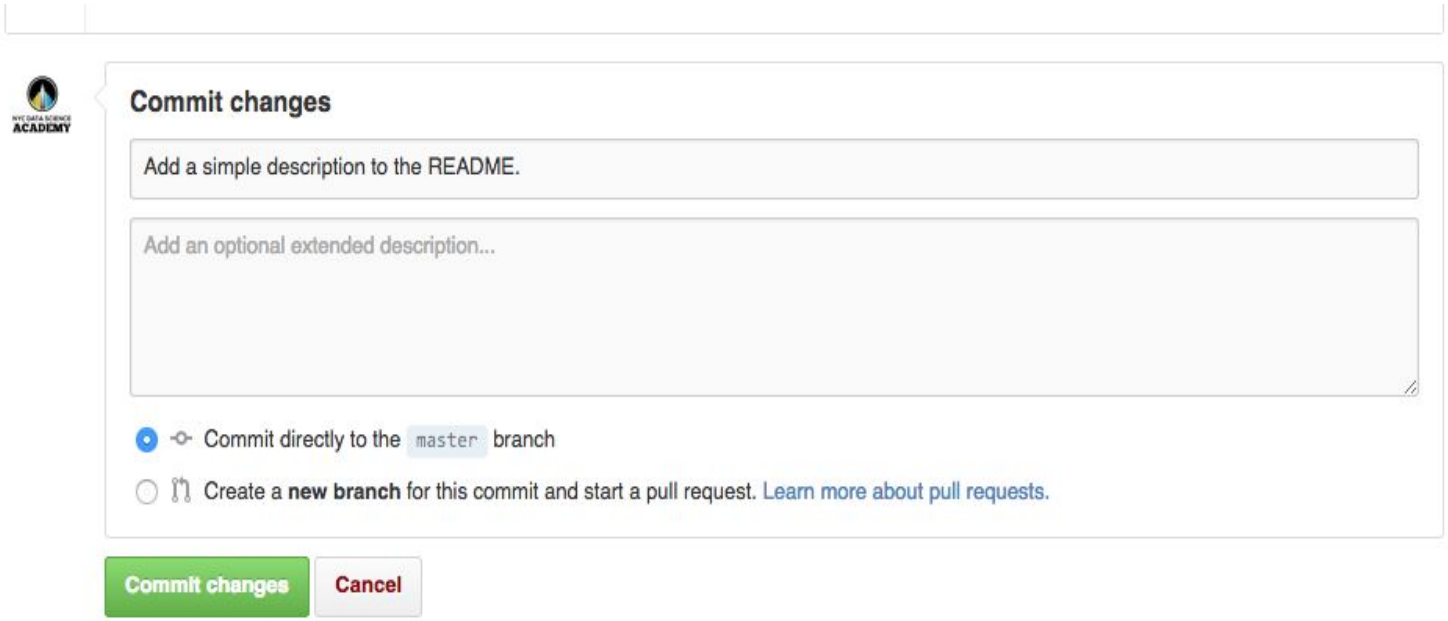
Edit From the Site

- Markdown https://help.github.com/articles/github-flavored-markdown/
- Change the contents of the README to have a good description, optionally using markdown like in the following:

```
# my_user_name.github.io

 Copyright @ My Name

## Description

This will be the main portfolio page for the My Name. I am
currently located in New York City.
```

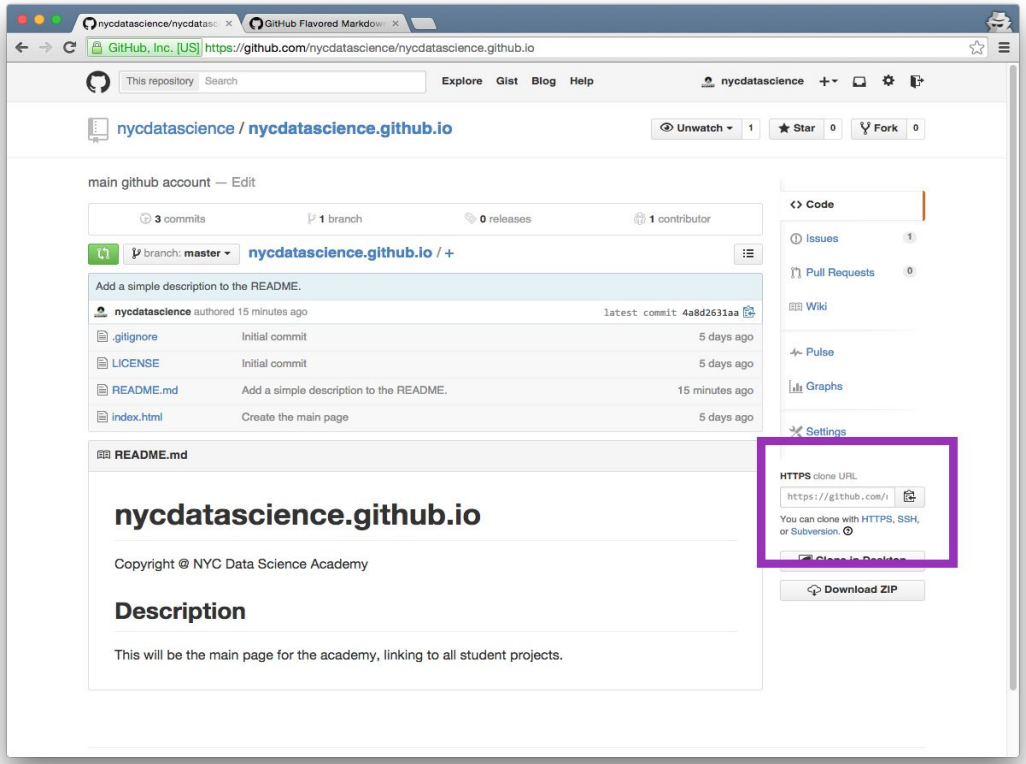
Edit From the Site

- Scroll down, add a commit message, and hit the **Commit changes** button. Leave the change to be made against the master branch.



Cloning

- Go back to the main page for the repository, and look for the remote HTTPS/SSH address of your repository:



Cloning

- We want to clone the repository using one of the following addresses, but your repository.
- HTTPS:

```
https://github.com/nycdatascience/nycdatascience.github.io.git
```

- Alternatives:

```
SSH: git@github.com:nycdatascience/nycdatascience.github.io.git
Subversion: https://github.com/nycdatascience/nycdatascience.
github.io
```

Cloning

- Copy the HTTPS address (the default), and open the local command line.

```
$ git clone https://github.com/<username>/<username>.github.io.
git
Cloning into 'nycdatascience.github.io'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 11 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

New Local Repository

- Change directories into the folder that was cloned locally. Be sure to substitute your username in the command below.

```
$ cd <username>.github.io
$ ls
README.md
```

Remotes

- Repositories cloned from GitHub automatically have the 'remote' setting put into the local configuration.

```
$ git remote -v
origin  https://github.com/nycdatascience/nycdatascience.github.
io.git (fetch)
origin  https://github.com/nycdatascience/nycdatascience.github.
io.git (push)
```

- Note: Run a git status to see what the set branch is.

Remotes

- When the local repository and remote repository are both created independently, how can you associate them?
- Associate the local and remote database versions, by specifying a simple name instead of the full remote address. Use the command
  - `$ git remote add [simple name] [url]`
- In the previous example, this was not necessary as the origin was set already.

```
// https protocol
$ git remote add origin https://github.
com/nycdatascience/nycdatascience.github.io.git
// or ssh protocol
$ git remote add origin git@github.com:
nycdatascience/nycdatascience.github.io.git
```

Pull

- Say you want check for the latest changes on the remote server. First, go back to GitHub, and modify the README.md file again. Then type one of the following.

```
$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/nycdatascience/nycdatascience.github.io
   4a8d263..f527c76  master     -> origin/master
Updating 4a8d263..f527c76
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Push

- Now we want to publish a local edit back to the remote GitHub repository. Create a new file in the local repository called `index.html`, with some sample content.

```
Hello, World!
```

Push

- Commit the new content to your local repository.

```
$ git add index.html
$ git commit -m "Add the primary webpage for the portfolio."
```

Push

- Commit the new content to your remote personal repository's master branch.

```
$ git push origin master
```

- If there are errors, then there is a problem with your setup. If not, wait a few seconds, and then refresh your GitHub page. Visit your actual portfolio page at https://username.github.io/ where username is your Github username.