

Data Science Without Fear

Jeff Jacobs

2023-07-20

Table of contents

Preface

This book serves as an introduction to data science, written with the goal of meeting you wherever you're at in your educational/career journey, regardless of your background in math or computer science. That is to say, we don't assume any background in STEM, only the desire to learn and the work ethic to put in the hours needed to become comfortable with each topic!

It is the companion to the course DSAN5000: Data Science and Analytics, taught at Georgetown University in the Fall of 2023.

1 Introduction

Setting Up Your Environment

Welcome to the world of data science! Before we can get to the fun stuff, learning about the world through data, we'll need to set up a **workflow**: the toolbox you'll use when you sit down to do some data science. Given our goal of meeting you wherever you're at in terms of data science knowledge, we'll start from the basics: operating systems, programming languages, setting up an **Integrated Development Environment (IDE)**, and using **version control** to make sure you can always “undo” if you make a change that breaks your code.

1.1 Operating Systems

- **Linux: (Free!)**
 - The “standard” OS for developers! You can expect instructions to use Linux commands: `ls`, `cd`, `touch`, `mv` ...
- **Windows: BUT**, proprietary shell (constantly googling “PowerShell equivalent”)

Linux	PowerShell
<code>ls</code>	<code>Get-ChildItem</code>
<code>cd</code>	<code>Set-Location</code>
<code>touch</code>	<code>New-Item</code>
<code>mv</code>	<code>Move-Item</code>

- **OSX: BUT**, OSX and Linux both built on Unix
⇒ if you know Terminal you know Linux!

1.2 Git vs. GitHub

Despite the confusingly similar names, it is important to keep in mind the distinction between **Git** and **GitHub**!

- **Git** is a command-line program, which on its own just runs on your local computer and keeps track of changes to your code.
- **GitHub**, on the other hand, is a **website** which allows you to take your **Git** repositories and store them **online**, whether privately or publicly.

This means, for example, that (if your repository is public) once you **push** your code to GitHub, others can view it and download it for themselves.

Git

- Command-line program
- `git init` in shell to create
- `git add` to track files
- `git commit` to commit changes to tracked files

GitHub

- Code hosting website
- Create a **repository** (repo) for each project
- Can **clone** repos onto your local machine

`git push`/`git pull`: The link between the two!

1.3 Git Diagram

1.4 Initializing a Repo

Let's make a directory for our project called `cool-project`, and initialize a Git repository for it:

```
user@hostname:~$ mkdir cool-project
user@hostname:~$ cd cool-project
user@hostname:~/cool-project$ git init
Initialized empty Git repository in /home/user/cool-project/.git/
```

This creates a hidden folder, `.git`, in the directory:

```
user@hostname:~/cool-project$ ls -lah
total 12K
drwxr-xr-x  3 user user 4.0K May 28 00:53 .
drwxr-xr-x 12 user user 4.0K May 28 00:53 ..
drwxr-xr-x  7 user user 4.0K May 28 00:53 .git
```

1.5 Adding and Committing a File

We're writing Python code, so let's create and track `cool_code.py`:

```
user@hostname:~/cool-project$ touch cool_code.py
user@hostname:~/cool-project$ git add cool_code.py
user@hostname:~/cool-project$ git status
On branch main

No commits yet

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

user@hostname:~/cool-project$ git commit -m "Initial version of cool_code.py"
```

```
[main (root-commit) b40dc25] Initial version of cool_code.py
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 cool_code.py
```

1.6 The Commit Log

View the commit log using `git log`:

```
user@hostname:~/cool-project$ git log
commit b40dc252a3b7355cc4c28397fefe7911ff3c94b9 (HEAD -> main)
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date:   Sun May 28 00:57:16 2023 +0000
```

Initial version of cool_code.py



1.7 Making Changes

```
user@hostname:~/cool-project$ git status
On branch main
nothing to commit, working tree clean
user@hostname:~/cool-project$ echo "1 + 1" >> cool_code.py
user@hostname:~/cool-project$ more cool_code.py
1 + 1
user@hostname:~/cool-project$ git add cool_code.py
user@hostname:~/cool-project$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   cool_code.py

user@hostname:~/cool-project$ git commit -m "Added code to cool_code.py"
```

```
[main e3bc497] Added code to cool_code.py
1 file changed, 1 insertion(+)
```

The output of the `git log` command will show the new version:

```
user@hostname:~/cool-project$ git log
commit e3bc497acbb5a487566ff2014dcd7b83d0c75224 (HEAD -> main)
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date:   Sun May 28 00:38:05 2023 +0000

    Added code to cool_code.py

commit b40dc25b14c0426b06c8d182184e147853f3c12eassets/img/gh_history.png
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date:   Sun May 28 00:37:02 2023 +0000

    Initial version of cool_code.py
```

1.8 More Changes

```
user@hostname:~/cool-project$ echo "2 + 2" >> cool_code.py
user@hostname:~/cool-project$ more cool_code.py
1 + 1
2 + 2
user@hostname:~/cool-project$ git add cool_code.py
user@hostname:~/cool-project$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cool_code.py

user@hostname:~/cool-project$ git commit -m "Second version of cool_code.py"
[main 4007db9] Second version of cool_code.py
1 file changed, 1 insertion(+)
```


1.9 And the git log

```
user@hostname:~/cool-project$ git log
commit 4007db9a031ca134fe09eab840b2bc845366a9c1 (HEAD -> main)
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:39:28 2023 +0000
```

Second version of cool_code.py

```
commit e3bc497acbb5a487566ff2014dcd7b83d0c75224
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:38:05 2023 +0000
```

Added code to cool_code.py

```
commit b40dc25b14c0426b06c8d182184e147853f3c12e
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:37:02 2023 +0000
```

Initial (empty) version of cool_code.py

1.10 Undoing a Commit I

First check the `git log` to find the **hash** for the commit you want to revert back to:

```
commit e3bc497acbb5a487566ff2014dcd7b83d0c75224
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:38:05 2023 +0000
```

Added code to cool_code.py

1.11 Undoing a Commit II

- This is irreversible!

```
user@hostname:~/cool-project$ git reset --hard e3bc497ac
HEAD is now at e3bc497 Added code to cool_code.py
user@hostname:~/cool-project$ git log
commit e3bc497acbb5a487566ff2014dcd7b83d0c75224 (HEAD -> main)
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:38:05 2023 +0000
```

Added code to cool_code.py

```
commit b40dc25b14c0426b06c8d182184e147853f3c12e
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>
Date: Sun May 28 00:37:02 2023 +0000
```

Initial (empty) version of cool_code.py

1.12 Onwards and Upwards

```
user@hostname:~/cool-project$ echo "3 + 3" >> cool_code.py
user@hostname:~/cool-project$ git add cool_code.py
user@hostname:~/cool-project$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cool_code.py

user@hostname:~/cool-project$ git commit -m "Added different code to cool_code.py"
[main 700d955] Added different code to cool_code.py
1 file changed, 1 insertion(+)
```

1.13

The final git log:

```
user@hostname:~/cool-project$ git log
commit 700d955faacb27d7b8bc464b9451851b5e319f20 (HEAD -> main)
```

```
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>  
Date: Sun May 28 00:44:49 2023 +0000
```

```
Added different code to cool_code.py
```

```
commit e3bc497acbb5a487566ff2014dcd7b83d0c75224  
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>  
Date: Sun May 28 00:38:05 2023 +0000
```

```
Added code to cool_code.py
```

```
commit b40dc25b14c0426b06c8d182184e147853f3c12e  
Author: Jeff Jacobs <jjacobs3@cs.stanford.edu>  
Date: Sun May 28 00:37:02 2023 +0000
```

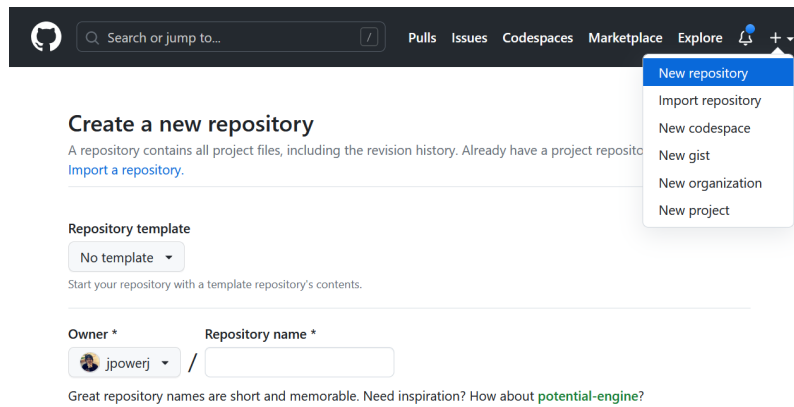
```
Initial (empty) version of cool_code.py
```

1.14 But Why These Diagrams?

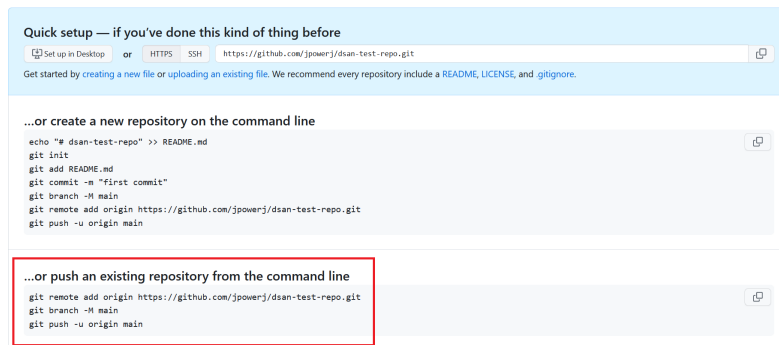
Even the simplest projects can start to look like:



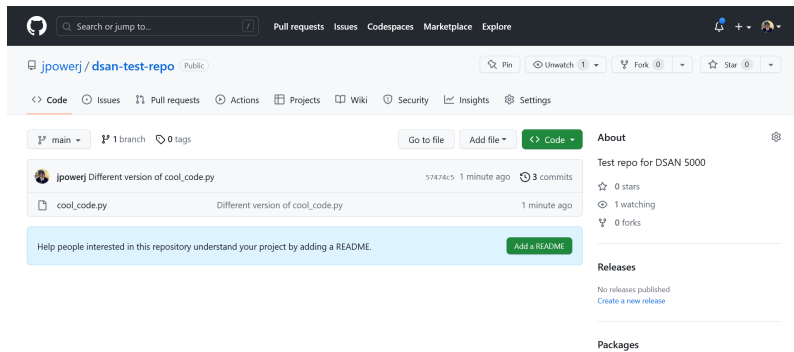
1.15 The GitHub Side: Remote



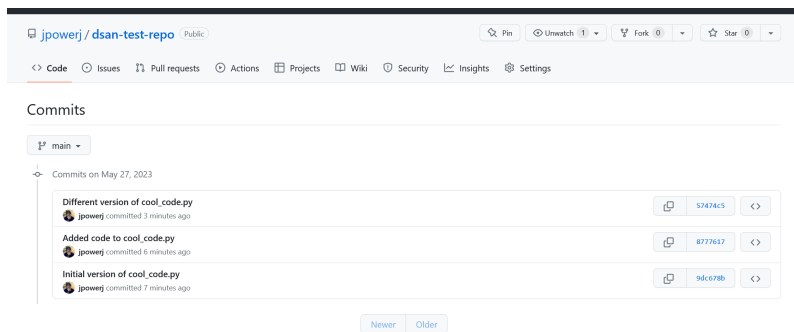
1.16 An Empty Repo



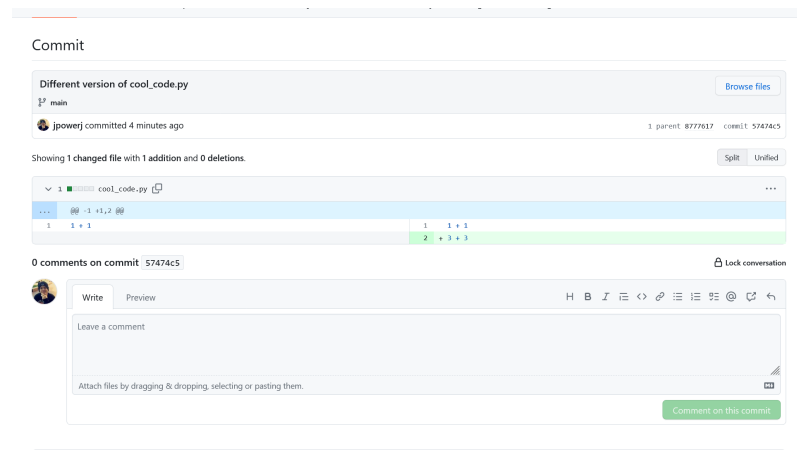
1.17 Refresh after git push



1.18 Commit History



1.19 Checking the diff



1.20 Web Development

	Frontend	Backend
Low Level	HTML/CSS/JavaScript	GitHub Pages
Middle Level	JS Libraries	PHP, SQL
High Level	React, Next.js	Node.js, Vercel

Frontend icons: UI+UI elements, what the user sees (on the screen), user experience (UX), data visualization
Backend icons: Databases, Security

1.21 Getting Content onto the Internet


Step 1: index.html

Step 2: Create GitHub repository


Step 3: git init, git add -A ., git push



Step 4: Enable GitHub Pages in repo settings

Step 5: <username>.github.io!


 **General**


Access



 Collaborators



 Moderation options 


Code and automation


 Branches


 Tags


 Rules Beta 

 Actions 

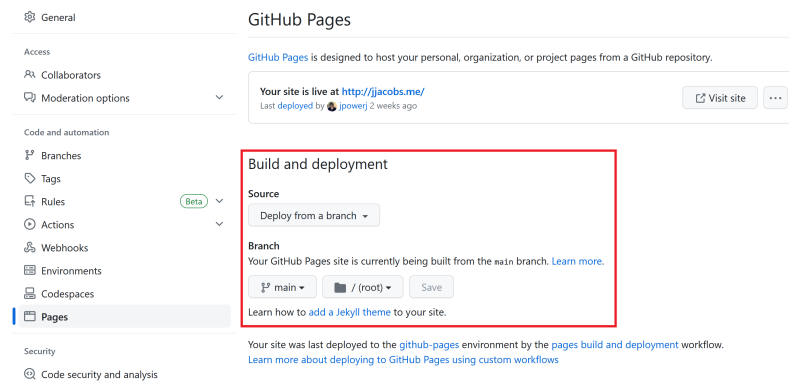
 Webhooks

 Environments

 Codespaces

 Pages

1.22 Deploying from a Branch/Folder



1.23 R vs. RStudio vs. Quarto

- GUI wrapper around R (Integrated Development Environment = IDE)
- Run **blocks** of R code (`.qmd` **chunks**)

The R Language

- Programming language
- Runs scripts via `Rscript <script>.r`

+

- GUI wrapper around Python (IDE)
- Run **blocks** of Python code (`.ipynb` **cells**)

The Python Language

- Scripting language
- On its own, just runs scripts via `python <script>.py`