Aman Agarwal (言煜鹰) Follow
In the business of helping others.
Nov 19, 2016 · 10 min read

GitHub for Chimpanzees (and a short rant against scientists)

· · ·

<Rant>

I have a degree in engineering, and have studied science *a lot*. Yet I have realised that I find it easier to read something written in *Spanish*, than most scientific papers written in *English* (and I've never even fucking learned Spanish). Do you know why scientific research papers are so hard to read?

Because the authors of these papers, the "global scientific community", ***predominantly*** consists of people whose life depends on one thing: whether they can coat their words with enough bullshit that the real scientists don't figure out how insignificant their work really is.

Now wait - to those who are offended by this claim, please allow me to expand before you write a nasty comment. You people are mostly a bunch of ass-witted graduate students who think that writing fat tongue twisters in your paper will make you seem smart enough to hide whatever mistake it was that got you selected for your current position in the first place. **(Now you can write that comment)**

I'm not even alone—I know **plenty** of senior scientists and experienced engineers who share this opinion. Now, I *do* understand that you can't write papers simply using high-schooler vocabulary. I also acknowledge the need for the required pre-requisite knowledge while reading a scientific paper. There is indeed always a need for *some* technical jargon, because in general the target audience of these papers are also experts in their niche fields. I'm not advocating that scientists simply replace every single "difficult sounding word". This is about the *other* practice of adding additional fluff to make even simple things sound more sophisticated on paper. I would argue that **a marginal increase in readability would increase a paper's *readership* by at least 5 times**. So improving readability by a factor of 2x is more than enough and that's all I ask for—**not suggesting that you turn every paper into a textbook**.

> ***Read Albert Einstein's paper if you think it can't be done.***

I don't blame you for following the herd—your entire system forces you to do that, and I have been in the same situation—my own supervisor in college once told me that my project reports should use more *sophisticated* English and passive voice and what not, so that it "sounds like real research". That whenever I have to choose between two words, I should use the more fancy term so that people take my work seriously.

Guess what, this culture of trying to describe *everything* in a more sophisticated way is what is in part responsible for destroying the world. **Because of this tough lingo, students get put off from pursuing science and it also puts a language barrier to scientists from other countries. By crypting your knowledge you are**

basically hoarding it from the common people who might have taken an interest in science, which makes the whole society dumber as a result. Talk about strangling innovation… And then you will get on your own faux Neil Tyson chair and complain about people who don't understand science, deny climate change and do stupid things. Well guess what, if you tried to make science accessible to people, maybe they would actually take interest. (*For the record, I greatly respect Dr Tyson and his efforts to educate common people about science. But far too many "scientists" echo his sentiments while also hypocritically bloating their papers with unnecessary jargon.*)

> ***Allowing blind people to use elevators is one thing; but adding Braille to the elevator buttons is where real humanity lies.***

Documentation for programmers has reached the same unprecedented level of hyper-jargon bullshit. Sometimes I am amazed that a document which is supposed to provide *clear and EASY instructions* can be harder to understand for a beginner than a hieroglyphic mural in the pyramid of Giza.

A great challenge in reducing income inequality and saving the world is to level the playing field and educate the damn masses. This is why I'm on a quest to democratise specialized knowledge in my own way.

</Rant>

. . .

My previous article, a dummies' guide to Google App Engine, shocked me by attracting a non-zero audience of people who shared the same sentiment as I do.

So today I will clean out one of the largest piles of cow-shit that new developers have to deal with. **Learning how to store projects on GitHub.**

Edit: I have to point out that this is more of a "git" tutorial; git and GitHub are technically two different things. Moreover, git can be used for *waaayy* many more things than just storing your projects online. It is a 'version control system', and you can Google what that means. If you want to learn the more advanced stuff, take this online course. Thanks to Reddit-folk for pointing this out.

I looked at some other git tutorials online which are supposedly for beginners. But after reading this and even THIS article, I have reached the conclusion that my intelligence is sub-human, and therefore only a tutorial meant for chimpanzees and dumber would work for me.

> *Assuming that there are other people on the planet who fall into the same category, I am writing a GitHub tutorial for chimpanzees.*

By the way, the **best GitHub tutorial** I've found so far is this:

I highly recommend watching this if you have the time!

Don't get me wrong here— I like GitHub. But I find it hard to read the fancy documentation written by people who have forgotten the boundary between English and Java.

Sorry— I digressed again. No more rants.

I'll cut the remaining intro paragraphs and jump right into it. By the way, PLEASE let me know if you encounter any problem following this tutorial.

. . .

ZERO-BULLSHIT GITHUB GUIDE

1. **What is a repository? I keep seeing this word in every git tutorial.**

A repository is a fancy word used by sadists to describe what is basically an online folder which stores all the code files of a particular project. So for example if you're building an app called "Monkey species identifier", you'll have one big folder which has all the sub-folders and files related to that app project right? That is called a "repository" and it will have a name like "Monkey-species-identifier". For the record, I swear this name wasn't inspired by the title of this tutorial. *Maybe I just ate too many bananas as a kid…*

2**. Download or install Git**

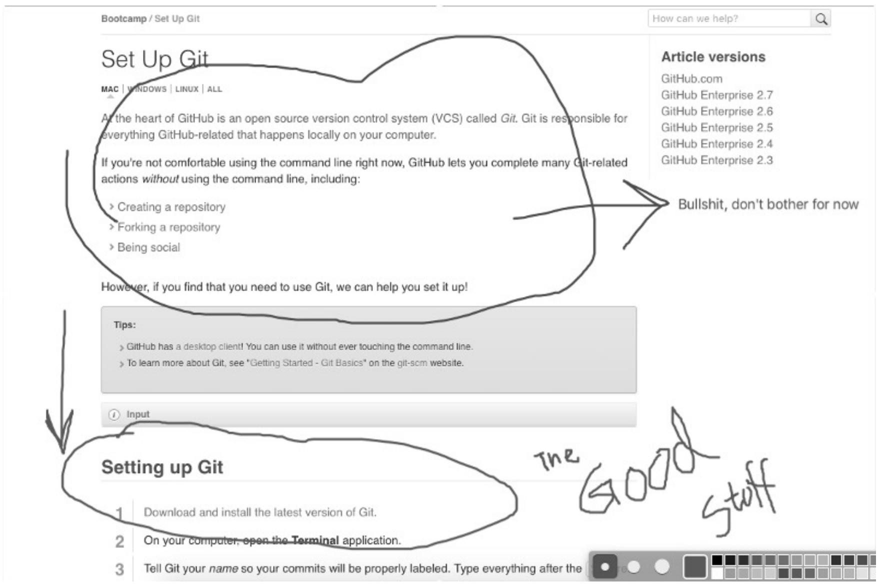
There are two ways you can do this. **If you're using a Mac** (like I do), don't bother surfing around the web looking for things to download. Just open your Terminal, and type "git". The Terminal will itself tell you that you don't have git installed, whether you'd like to install it, yada yada yada and basically take you through the process on its own. And by the end you'll suddenly have git on your system!

If you use Windows, you'll have to download git and then you might need something called a Powershell, which looks just like the command prompt but maybe has a different color.

If you use Linux, I'm not sure what you should do, because you're probably a geek who knows this stuff already? Why else do you even have Linux?

Anyway, the official documentation for this step can be found here: https://help.github.com/articles/set-up-git/

Scroll down the mumbo-jumbo words like "forking" and "repository" etc, and go straight to the *good shit, w*here they use familiar words like “download” and “install”.



Bootcamp / Set Up Git

How can we help?

Set Up Git

MAC | WINDOWS | LINUX | ALL

At the heart of GitHub is an open source version control system (VCS) called *Git*. Git is responsible for everything GitHub-related that happens locally on your computer.

If you're not comfortable using the command line right now, GitHub lets you complete many Git-related actions *without* using the command line, including:

- Creating a repository
- Forking a repository
- Being social

However, if you find that you need to use Git, we can help you set it up!

Tips:

- GitHub has a desktop client! You can use it without ever touching the command line.
- To learn more about Git, see "Getting Started - Git Basics" on the git-scm website.

Input

Setting up Git

1. Download and install the latest version of Git.
2. On your computer, open the **Terminal** application.
3. Tell Git your *name* so your commits will be properly labeled. Type everything after the

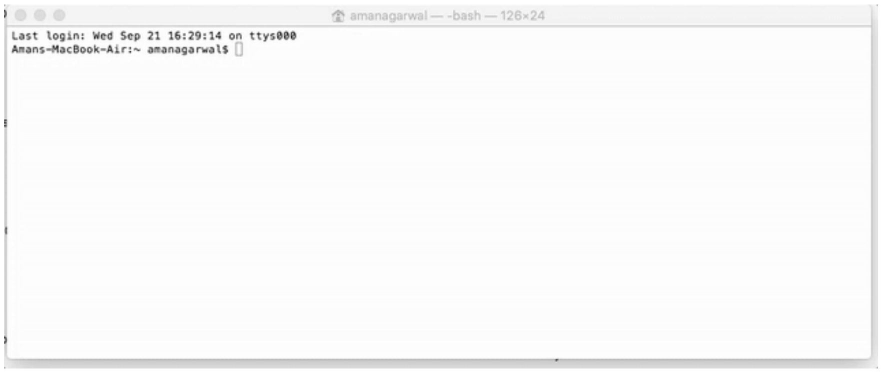
Article versions

GitHub.com
GitHub Enterprise 2.7
GitHub Enterprise 2.6
GitHub Enterprise 2.5
GitHub Enterprise 2.4
GitHub Enterprise 2.3

Bullshit, don't bother for now

The Good stuff

3. **Once you have git, open your Terminal and do this to check if it works fine:**



```
Last login: Wed Sep 21 16:29:14 on ttys000
Amans-MacBook-Air:~ amanagarwal$ 
```

Type ‘git’ and press enter.

After this, it doesn’t matter if you have Windows/Linux/Mac etc. It’s all the same once you have installed GitHub.

. . .

FIRST—Learn how to use the command line to navigate between folders.

It’s actually quite simple! I don’t have time to create my own tutorial, so I’ll just refer you to this ***beautiful*** 3-minute video which explains how to use the command line very easily, in a way that a chimpanzee like me could understand. It's not my own video, I found it online and it's excellent.

Click on the image link.

Please invest 3 minutes into watching this video!

. . .

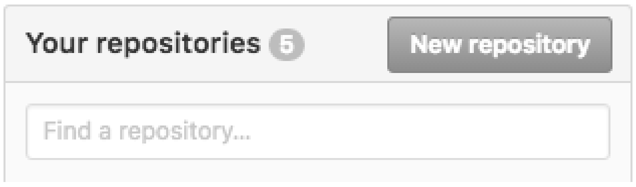
Now, how to use GitHub?

You basically need to remember only FOUR commands to get up and running: ***clone, add, commit*** and ***push***.

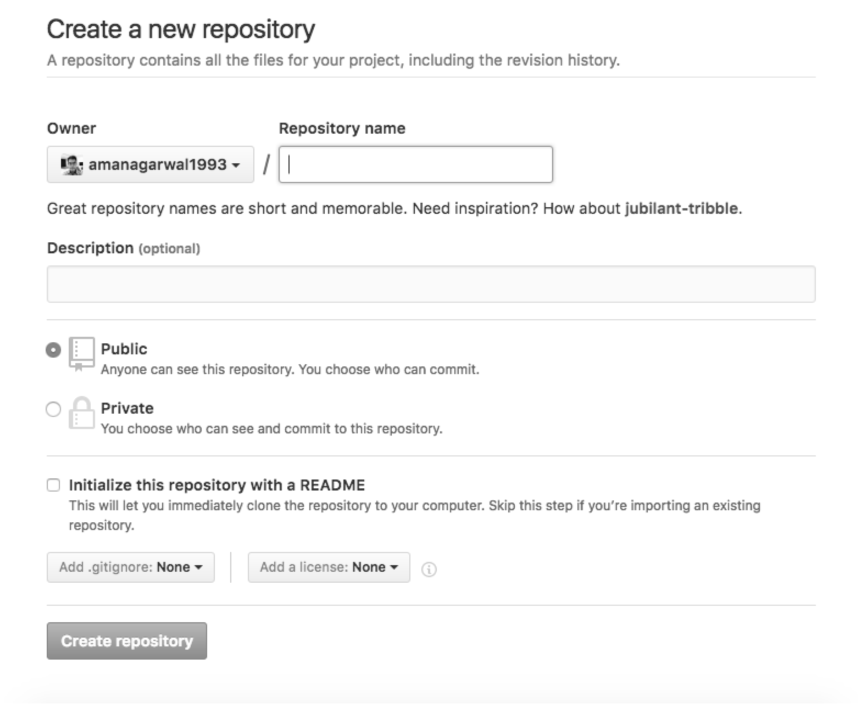
Creating a new project

1. Go to github.com and sign-in.

2. Find and click on the 'New Repository' button. You will see something like this in the **right-hand side** of your window, somewhere.



3. You will see something like this:



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: amanagarwal1993 / Repository name:

Great repository names are short and memorable. Need inspiration? How about **jubilant-tribble**.

Description (optional)

● Public
Anyone can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

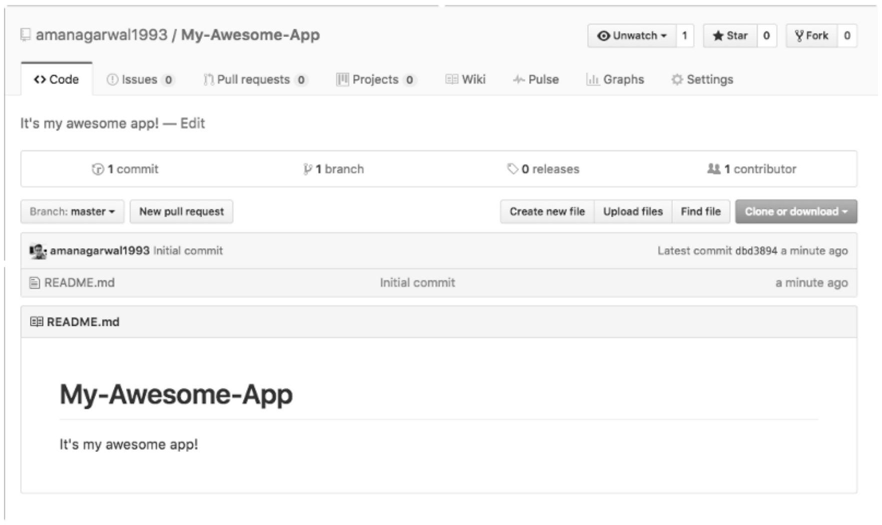
4. Create a repository name. There should be no spaces or punctuation in the name. For example, "My-Awesome-App".

5. Choose the 'Public' option for now. To choose 'Private', you'll need to upgrade your account to a paid account. It doesn't really make a difference unless you're working in a company and your code is meant to be kept secret from the world or something like that.

6. Check 'Initialize this repository with a README'.

7. Click on 'Create Repository'.

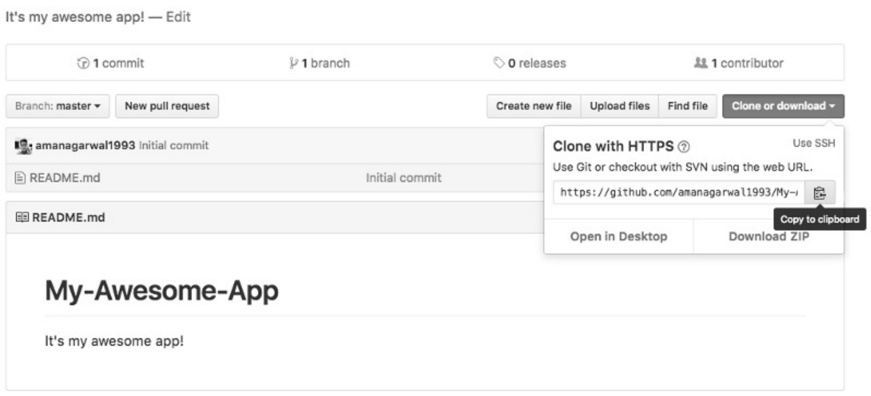
8. Now you'll see this:



You have created a new repository called 'My-Awesome-App'.

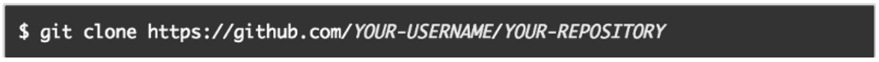
Currently your new Repository has only one file, which is the README.md. It was created because you chose the option to initialize the repo with a readme. :)

9. Then, go to your GitHub repo, and click on "Clone or download". Copy the https url.



11. Now open your terminal/powershell/whatever. Navigate (using cd and ls functions) to the place where you want to set up your project folder.

12. Type "git clone" and then paste the url you copied in step #10. Press enter, as shown below:



```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

The url is the one you copied earlier.

13. You will see the folder My-Awesome-App being created. Now, add the project files (your code files) to this folder.

Bingo! You have now created what is called a "local repository", which is a clone of your repository on github.com.

Now—how to store your code online and make changes to it

The next step is to sync these changes with your online repository on github.com. This requires 2 steps:

1. Tracking changes in the local repository
2. Updating the online repository to reflect those changes

For the first step, tracking changes locally, we use 2 commands, **add** and **commit**. You don't have to understand how exactly they work (for now). Basically it's just the way the git system is created, so we shall roll with that.

1. If your terminal is not already inside your My-Awesome-App folder, use **cd** to get inside it through the command line.

2. Do this:

git add *

This tells GitHub that you have added some new files to your project, that you want to make a part of the project. You will use the add command every time you add a new file to that folder.

17. Now, we will 'commit' those changes. Type the following command: *git commit -m "short comment"*. For example,

git commit -m "Added new files"

18. Almost there! Now your local repository is up to date with the latest changes. Time to sync it with the online repository.

19. Type: **git push origin master**

20. By executing the above command, your terminal will upload the local files to the online repo, and now your code is stored on github!

If you go to github.com now and check your repository, you will see all your code files there. It wasn't too difficult, right? :)

. . .

"What do I do next time?"

The above tutorial was all about your *first time* using GitHub. But what if you make changes to that project and want to sync your updates again? Well, here's what you do:

1. Make whatever changes to your code on your computer in the local repository, and save.
2. Command: **git add** *
3. Use command **git commit -m "comment"**. This allows you to document the change you just made to the code. You know, like telling someone "hey I added a new feature". (If you want to record different changes to 3 different files, you will use the command thrice, by using a different comment for each file and

adding *FILENAME* at the end, like '***git commit -m "comment" FILENAME***'
This way, you will be able to document the latest changes for *each* file of code.)

4. Sync the changes with **git push origin master**.

Makes sense? Now go ahead and try it out.

Do let me know what you think of the tutorial. Even if it's bad. I post all my essays on Reddit for the community to rip apart, so don't worry I have thick skin. :)

Want to stay in touch with me?

This is not a newsletter. I don't write for profit, so you'll only get genuine, humane emails from me. I write in-depth, easy-to-read essays on technology and business, every couple months.

yourname@example.com Sign up