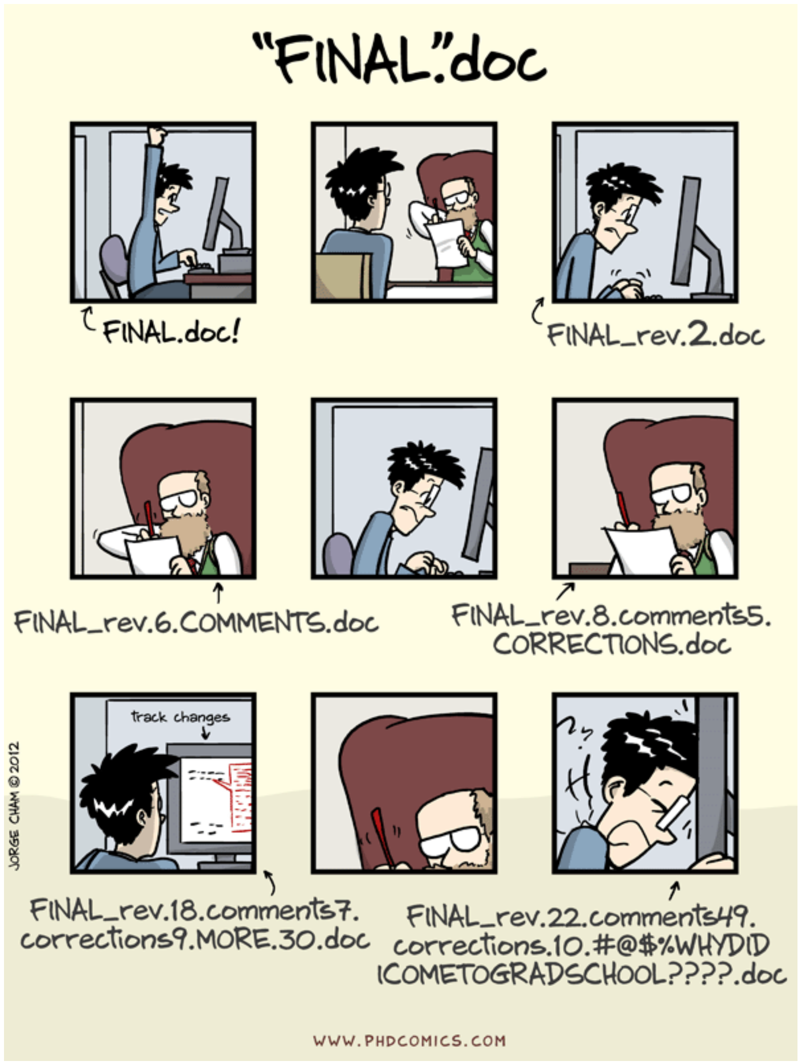
PSY 394U Python Coding for Psychological Sciences
Fall 2016

Lab Note
November 29, 2016

Today's goals

- Learn how to use simple commands for Git
- Learn how to store and manage your Git project on GitHub


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM
http://www.phdcomics.com/comics/archive.php?comicid=1531

0. What is Git?

Git is a version control system. Git keeps track of changes you make to your programs and codes. This is helpful in keeping track of changes that you made on your programs. Without such a version control, you may end up having multiple versions of the same program, with slightly different names (e.g., `analysis1.py`, `analysis2.py`, `analysis3_a.py`, `analysis5_b.py`, etc.) After a while, it becomes very hard to keep track of which programs are which. Another functionality of Git is that it lets you go back to the previous version of the same program, if necessary.

1. Starting Git

Mac

Git comes standard on Mac. You simply need to start the Terminal app, and use Git commands.

Windows

If you have successfully installed Git (https://git-scm.com/downloads), then there is a program called Git BASH. It is a BASH emulator, that lets you type in commands as if you are using a bash terminal on a Unix or Linux system.

Configure your Git environment

At this time, it is a good idea to save your name and email address associated with your Git projects.

```
git config --global user.name "Your Full Name"
git config --global user.email you@somewhere.com
```

2. Initializing a Git

Let's create a directory called `GitTest`, and go to this directory. You can do this by

```
mkdir GitTest
cd GitTest
```

And the command pwd lets you verify which directory you are in. Then let's create a simple program file called Hello-World.py. For those who are not familiar with Python, this program simply prints out "Hello World!"

```
print("Hello World!")
```

Now that we have a program in this directory, we will initialize a Git version control in this directory by typing

```
git init
```

This will produce a message

```
Initialized empty Git repository in
/Users/sh45474/Documents/Research/Tutorials/GitTest/.git/
```

This means that we now have a Git repository (or a directory under a Git version control) in the current directory. At this point, you can type

```
git status
```

And see the status of the files in this Git project. In this particular example, you will get a message

```
On branch master

Initial commit

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

	Hello-World.py

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

Our program Hello-World.py is untracked right now. To keep track of this file, you need to do the following.

```
git add Hello-World.py
git commit -m 'Initial commit'
```

The first command adds the program to be tracked, and the second command commits (or finalizes) the current version to the Git repository. When you commit your changes with `git commit`, you can provide a detailed description of the changes with the `-m` option and a comment. After the program is committed, you see a message

```
[master (root-commit) 2a75ebb] Initial commit
 1 file changed, 2 insertions(+)
 create mode 100644 Hello-World.py
```

At this point, you can check the status again

```
git status
```

Then you will get a message

```
On branch master
nothing to commit, working directory clean
```

This means that the program is successfully registered to the Git repository

Exercise

1. *Create a simple Python program called `Goodbye-World.py` with the following content.*

   ```
   print("Goodbye World!")
   ```

2. *Add and commit this program to the Git repository you created earlier.*

3. Modifying a program

Now, let's edit the program `Hello-World.py` to the following:

```
print("Hello World!!!")
```

In other words, I added two additional exclamation marks. At this point, type the following command,

```
git diff Hello-World.py
```

And you can see the difference between the committed version of the program and the one you just edited.

```
diff --git a/Hello-World.py b/Hello-World.py
index 147d1da..ebcac80 100644
--- a/Hello-World.py
+++ b/Hello-World.py
@@ -1,2 +1,2 @@
-print("Hello World!")
+print("Hello World!!!")
```

Then you type

```
git status
```

It tells you that the program has been modified but not committed.

```
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:   Hello-World.py

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

You can commit the change by

```
git add Hello-World.py
git commit -m 'Added two exclamation marks at the end'
```

When you commit, let's be ***very specific about the changes you made***, so that you know exactly what you did when you review changes made to your program. Once the change has been committed, you can review the difference by

```
git log -p Hello-World.py
```

This command shows all the changes that have been made to the program Hello-World.py since this program was added to the repository.

```
commit b772e179cd4de26ad85c97818913dca12240dd93
Author: Satoru Hayasaka <hayasaka@utexas.edu>
Date:   Fri Nov 25 23:53:37 2016 -0600

    Added two exclamation marks at the end

diff --git a/Hello-World.py b/Hello-World.py
index 147d1da..ebcac80 100644
--- a/Hello-World.py
+++ b/Hello-World.py
@@ -1,2 +1,2 @@
-print("Hello World!")
+print("Hello World!!!")


commit 2a75ebb74923b1333dff4c67a55c5fc6183c482d
Author: Satoru Hayasaka <hayasaka@utexas.edu>
Date:   Fri Nov 25 23:33:25 2016 -0600

    Initial commit

diff --git a/Hello-World.py b/Hello-World.py
new file mode 100644
index 0000000..147d1da
--- /dev/null
+++ b/Hello-World.py
@@ -0,0 +1,2 @@
+print("Hello World!")
+
```

Exercise

1. *Modify the earlier program `Goodbye-World.py` to*

   ```
   print("Goodbye, World!!!")
   ```

   In other words, add a comma after "Goodbye" and add two exclamation marks at the end.
2. *Add and commit the modified program to the Git repository.*

4. Switching between versions

Now, let's edit `Hello-World.py` again.

```
print("Hello World!!!")
name = input("What is your name? ")
print("Hello, " + name)
```

And add and commit this program to the Git repository.

```
git add Hello-World.py
git commit -m 'Now asks a name, and prints out a personalized greeting'
```

Now, after committing this change, you don't like this change. You want to go back to a version prior to this. To do so, you need see which version you want to go back to. To get a list of different versions, you can type

```
git log --oneline Hello-World.py
```

And this produces a list of comments from different commits.

```
b4ba63d Now asks a name, and prints out a personalized greeting
b772e17 Added two exclamation marks at the end
2a75ebb Initial commit
```

Say, you want to go back to the version that doesn't ask for the user's name, but has multiple exclamation marks at the end. So you want to go back to the version `b772e17`. To do so, you type

```
git checkout b772e17 Hello-World.py
```

And if you check `Hello-World.py`, you see that it has gone back to the previous version. One thing to remember is that you have to commit the reverted version again. For example,

```
git commit -m "Reverting back to the version without asking for a name"
```

If you do change your mind again and you want the version that asks for the name of the user, then you can go back to version `b4ba63d` by typing

```
git checkout b4ba63d Hello-World.py
```

Commit often

I highly recommend committing your changes often. When you modify your program and it runs successfully, then you should commit your program to the Git repository. Do not wait until you have the final product. The more intermediate points you have, the more flexible you are when you want to go back to a previous version.

5. What is a GitHub?

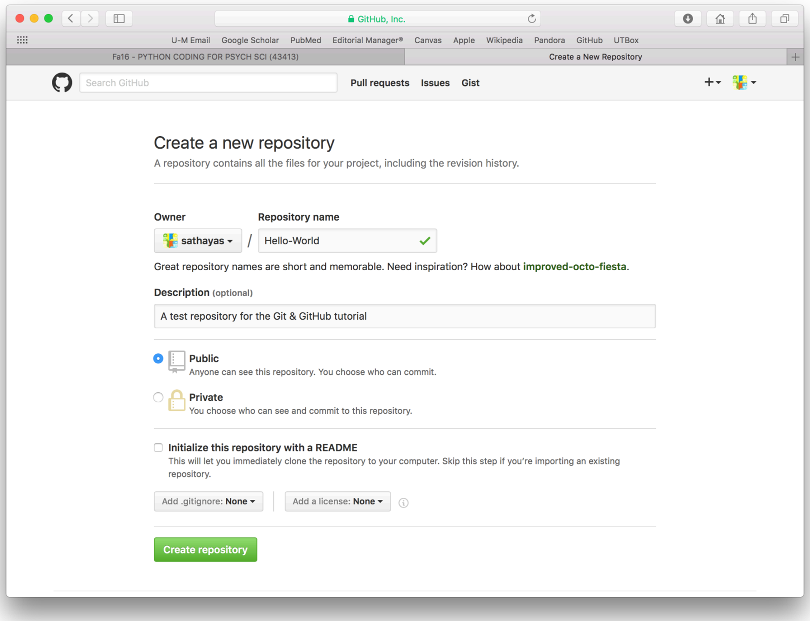
GitHub is a hosting service for Git projects, accessible by a web browser. Any Git project you store on GitHub is available as a *repository* on GitHub. Your repository has its own URL to facilitate the access. Unless you pay a small fee to keep your Git project hidden, anyone can see your Git repository on GitHub. However, only the people who have been granted access as contributors can edit the repository. You can create an account on GitHub for free, if you don't have an account already. You can do so at GitHub's web site at https://github.com and sign up for an account.

GitHub offers an easy way to share your codes with others. This can be beneficial if you are part of a collaborative project. Moreover, it can be used as a platform to disseminate your programs with general public, or at least other researchers in your field.

If you are on a job market and looking for a position that requires extensive programing skills, then chances are your potential employers will check out some of your projects on GitHub. So it is an excellent way to showcase your programming skills!

6. Creating a GitHub repository

To create a new repository on GitHub, you just click on the **New repository** button. Then you can set some details on the new repository.



GitHub, Inc.

U-M Email Google Scholar PubMed Editorial Manager® Canvas Apple Wikipedia Pandora GitHub UTBox

Fa16 - PYTHON CODING FOR PSYCH SCI (43413) Create a New Repository

Search GitHub Pull requests Issues Gist

Create a new repository

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

Owner: sathayas / Repository name: Hello-World

Great repository names are short and memorable. Need inspiration? How about improved-octo-fiesta.

Description (optional): A test repository for the Git & GitHub tutorial

[✓ Public] Anyone can see this repository. You choose who can commit.

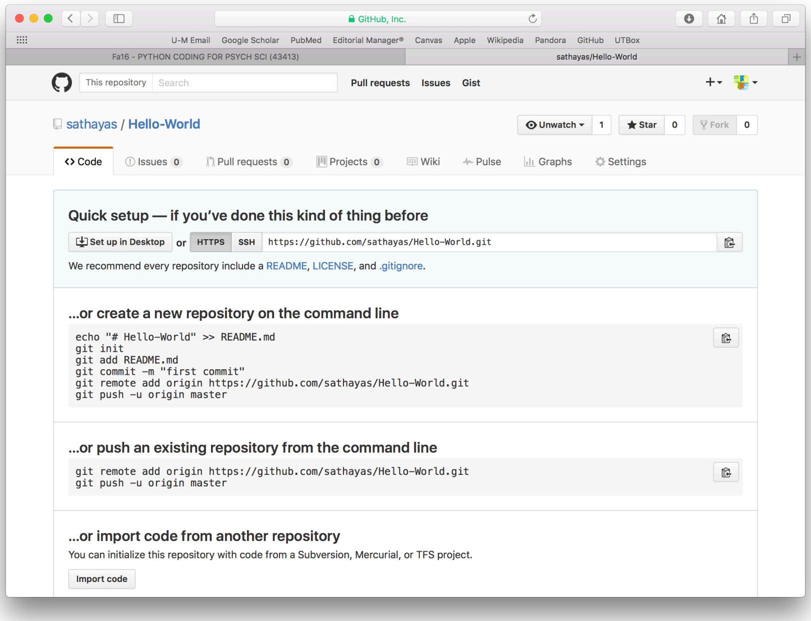
Private: You choose who can see and commit to this repository.

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

Create repository

As an example, I am creating a new repository called Hello-World. It is a publicly accessible repository. Then I click on the **Create repository** button, then it gives me some useful information.



GitHub, Inc.

U-M Email Google Scholar PubMed Editorial Manager® Canvas Apple Wikipedia Pandora GitHub UTBox

Fa16 - PYTHON CODING FOR PSYCH SCI (43413) sathayas/Hello-World

This repository Search Pull requests Issues Gist

sathayas / Hello-World Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/sathayas/Hello-World.git`

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# Hello-World" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/sathayas/Hello-World.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/sathayas/Hello-World.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Here, it shows me two different ways of setting up the repository. The first way is to create a repository from scratch. The second way is to add an existing Git project on my computer. Since we have been working on a Git project for a while, we will do the second approach. So, we need to copy and paste the fillowing command on your computer.

```
git remote add origin https://github.com/sathayas/Hello-World.git
```

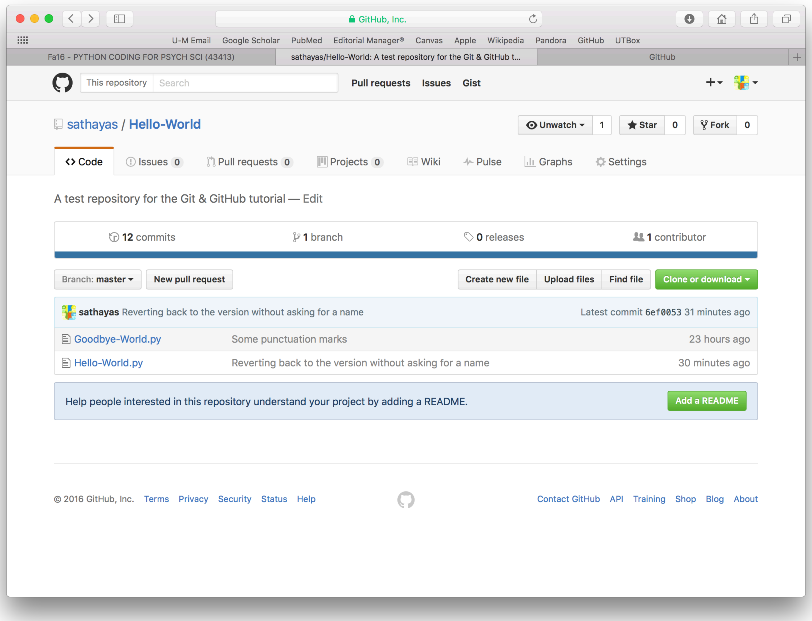
Please note that your username is different from sathayas, so you have to copy and paste the command from the GitHub website (not from this lab note!). This command tells the Git project on your computer that there is a remote GitHub directory out there to store this Git project. To actually add the contents of the Git project from your computer to the GitHub repository, you have to run

```
git push -u origin master
```

It may, or may not, ask for your GitHub username and password, so be ready to provide those. You will get a bunch of messages on your Git window

```
Counting objects: 28, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (22/22), done.
Writing objects: 100% (28/28), 2.98 KiB | 0 bytes/s, done.
Total 28 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/sathayas/Hello-World.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

And on your GitHub repository on shows the content of this Git project.



A test repository for the Git & GitHub tutorial — Edit

sathayas Reverting back to the version without asking for a name   Latest commit 6ef0053 31 minutes ago

Goodbye-World.py	Some punctuation marks	23 hours ago
Hello-World.py	Reverting back to the version without asking for a name	30 minutes ago

Help people interested in this repository understand your project by adding a README.

Congratulations! Your Git project is now on GitHub! You should be able to see two programs under the Git repository: `Hellow-World.py` and `Goodbye-World.py`. If you click either of these, then you will be able to see the source code for the program.

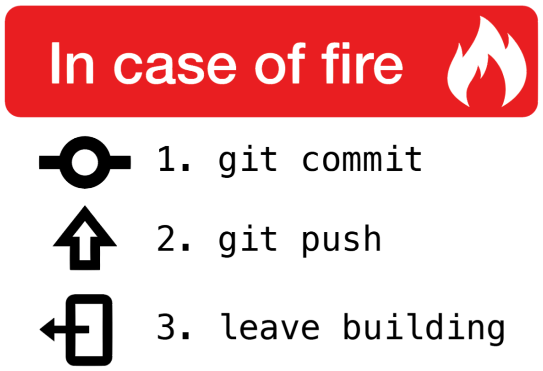
7. Pushing changes to GitHub

Now, let's get back to your Git project on your computer. Make changes to `Hello-World.py` so that

```
print("Hello World!!!")
print("Hello Git!!!")
print("Hello GitHub!!!)
```

Add and commit the changes to the Git project. At this point, whatever the changes you made are still local. In other words, these changes are not shown on your GitHub repository. To update the GitHub repository with these changes, you need to ***push*** the Git project by

```
git push
```



Now click on `Hello-World.py` on your GitHub repository, and you can see that the changes have been incorporated into the GitHub project. It is not necessary to push to GitHub as frequently as you commit your Git project. However, consider the fact that your GitHub repository is in effect a remote backup of your project, just in case something catastrophic happens to your computer.

8. Cloning a GitHub repository

Say, you have been programming a Git project on your laptop computer, but now you would like to work on the same Git project on a desktop computer in your lab. You can do this fairly easily as long as the corresponding GitHub repository is up to date. Let's say, I want to work on the Hello-World project on another computer. Then I simply log on to another computer (*assuming that this computer has Git installed*), then ran

```
git clone https://github.com/sathayas/Hello-World.git
```

The URL for this command can be easily copied when you press the **clone or download** button on GitHub. (***NB**: don't use the URL from this example directly, or otherwise you will be making a copy of my GitHub repository, not yours!*)

Now you should have a directory called `Hello-World` on your other computer, with its contents. If you type `git log` on this directory, you can see that all the history associated with this repository has also been copied to this directory.

You can also use the `git clone` command to clone (i.e., copy everything) someone else's GitHub repository too. For example, you can clone all the example codes from my Python class to your computer by running

```
git clone https://github.com/sathayas/PythonClass2016.git
```

9. Pulling from a GitHub repository

Even if you clone a GitHub repository to your computer, it is possible that the GitHub repository is updated continuously. If that happens, your local copy of the GitHub repository may no longer be up to date. To demonstrate this, let's create a new program called `WhatsUp.py`, with the following.

```
print("What's up!!!")
```

Then add this program, commit it, then push the repository to GitHub.

```
git add WhatsUp.py
git commit -m 'Initial commit'
git push
```

To update your local copy (in some other location), then you need to run the following command.

```
git pull
```

Then you see these messages.

```
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/sathayas/Hello-World
   d11d1c0..8c331cb  master     -> origin/master
Updating d11d1c0..8c331cb
Fast-forward
 WhatsUp.py | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 WhatsUp.py
```

You can see `WhatsUp.py` in this directory now.

10. Resolving conflicts

You update a local copy of a program on your computer. Then you realize that someone else has made another update to the same program on the GitHub repository. What should we do? If this happens, then we need to resolve the conflicting edits.

Say, `Hello-World.py` on GitHub is edited as

```
print("Hello World!!!")
print("Hello Everybody!!!")
```

While you made an edit to your local copy of `Hello-World.py` as

```
print("Hello World!!!")
print("Hey Y'all!!!")
```

And you have added and committed the edit locally on your computer. At this point, you want to sync your local version with the GitHub version with `git pull` command. Here is what happens.

```
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/sathayas/Hello-World
   8c331cb..d158a1d  master     -> origin/master
Auto-merging Hello-World.py
CONFLICT (content): Merge conflict in Hello-World.py
Automatic merge failed; fix conflicts and then commit the result.
```

It tells you that there is a conflict with `Hello-World.py`, and you need to resolve this conflict. Since neither Git nor GitHub can figure out which one you want, you need to manually edit `Hello-World.py`. If you open `Hello-World.py`, you notice

```
print("Hello World!!!")
<<<<<<< HEAD
print("Hey Y'all!!!")
=======
print("Hello Everybody!!!")
>>>>>>> d158a1dc95dbf76fc1c16feb694d8ade22bcd244
```

The two versions are shown, side by side. You edit this program so that you can choose the one you like.

```
print("Hello World!!!")
```

```
    print("Hey Y'all!!!")
```

Then you add and commit the program. If you are OK with this change on your local computer only, then you can leave it now. If you want this change to be made to the one on GitHub, then you need to run `git push` command.

11. More advanced topics

We just scratched the surface of what Git and GitHub can do. If you are interested in the advanced topics on Git or GitHub, there are many tutorials and documentations on the web. As a starter, here are the official documentations on Git and GitHub.

- Git -- https://git-scm.com/doc
- GitHub -- https://help.github.com