CME 211: Homework 0

Due: Friday, September 29, 2017 at 2:30pm (Pacific Daylight Time)

Background

This tutorial style assignment will guide you through all of the steps needed to get up and running for CME211.

User accounts

This homework documentation will refer to your GitHub username and Stanford username (also known as the SUNetID) in various places.

- `[github_user]` needs to be replaced with *your* GitHub username.
- `[sunet_id]` needs to be replaced with *your* Stanford username (SUNetID). This is the id that comes before `@stanford.edu` in your email address. Note that email aliases will not work here.

Create a GitHub account

CME211 uses GitHub for submission of the homework assignments. Thus, all students in CME211 will need a GitHub account. If you do not already have a GitHub account, please visit https://github.com/join to create a new account.

(Optional): Once you have an account, it is a good idea to request an education discount:

- https://education.github.com/discount_requests/new

This will allow you to maintain private repositories on your GitHub account while you are a student. Free (non-education) GitHub accounts are limited to public repositories.

Generate CME211 homework directory

CME211 has a GitHub organization (https://github.com/CME211) which maintains all student code repositories. Each student will be granted a separate private code repository. The only people that can access the repository are the student and course staff. To create your repository please visit the following link and login with your GitHub account credentials:

- Create CME211 homework repository (link)

The location of the created repository is:

- `https://github.com/CME211/cme211-[github_user]`

Here, `[github_user]` is your GitHub username. For example, Nick's GitHub username is `nwh` so his repository address is `https://github.com/CME211/cme211-nwh`.

At this point it is a good idea to visit the webpage for your repository and look around. You will see a line "We recommend every repository include a README, LICENSE, and .gitignore." in your repository. Please create the README and .gitignore. The `README.md` file contains a short description of the repository. The `.gitignore` file tells `git` which files to ignore. For now .gitignore can be empty.

Now that your repository is created, it is time to login to `corn.stanford.edu` and clone the repository so that you can do some work. Please follow these steps.

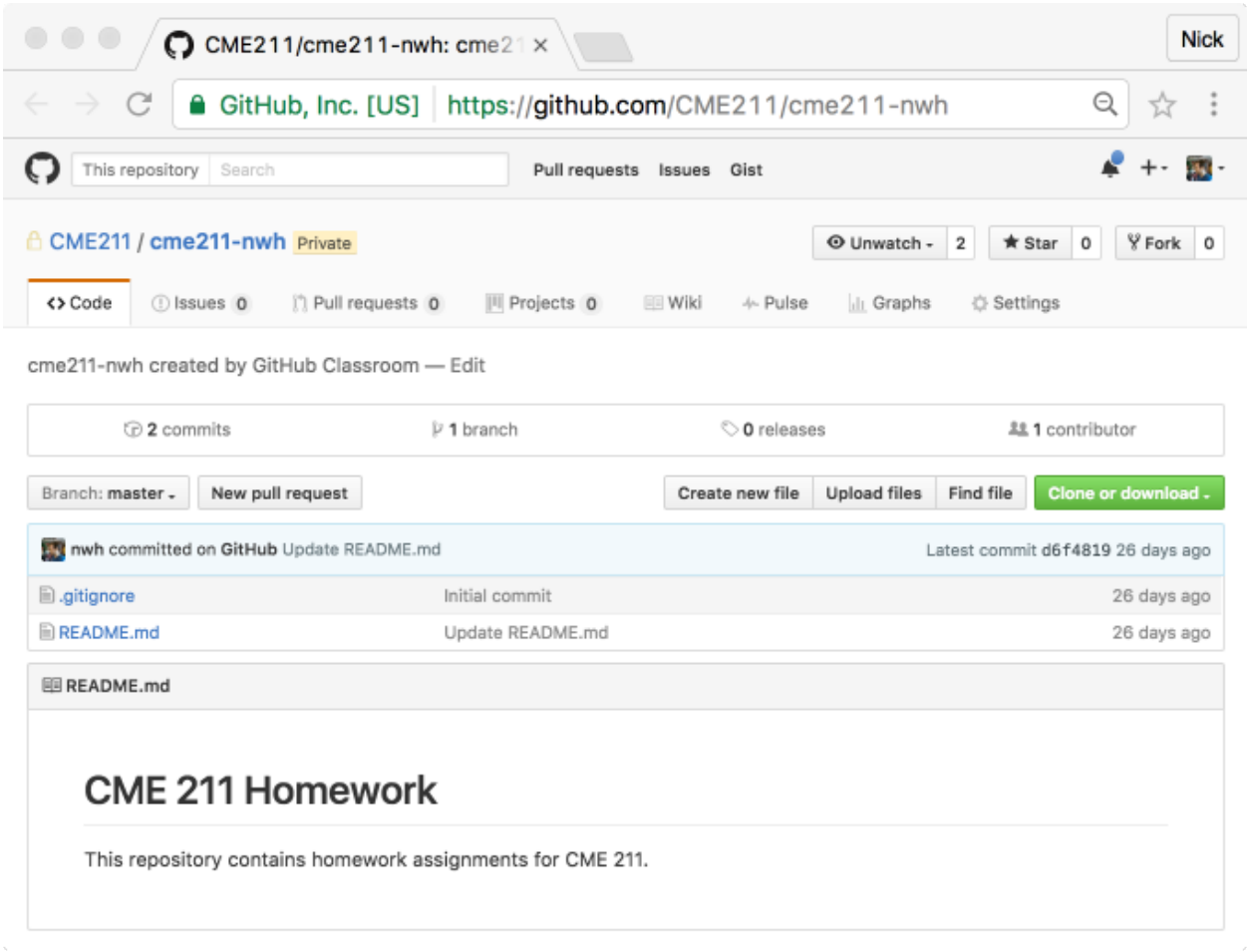

Figure 1: Empty GitHub repository for CME211

Log into `corn.stanford.edu`

Access `corn.stanford.edu` via `ssh` with an appropriate terminal. On macOS this can be accomplished in `Terminal.app`. In the shell, enter the command (replacing `nwh` with your SUNetID):

```
$ ssh nwh@corn.stanford.edu
```

After 2-factor authentication and a welcome message, you will see a command prompt looking something like this:

```
nwh@corn22:~$
```

This says that user `nwh` is logged into server `corn22` and the shell is currently focused on the users home directory. Remember `~` is an alias for the home directory. Note that your command prompt may look different. In the instructions that follow `$` indicates a shell command.

Configure `git`

We need to tell `git` who we are. If you have not yet configured `git` on `corn`. Please do the following:

```
# on corn
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Replace the name and email (email associated with your github account) with your information. It is also a good idea to tell `git` which editor you want to use for commit messages. For new users, I recommend using `nano` for this purpose:

```
$ git config --global core.editor nano
```

These commands store information in the user's `git` configuration file, located at `~/.gitconfig`. You can inspect the contents of the file with `cat`. Here is mine:

```
$ cat ~/.gitconfig
[user]
    name = Nick Henderson
    email = nwh@stanford.edu
[core]
    editor = nano
```

See: https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

Clone repo

Clone your CME211 repository to your Farmshare user directory on `corn`. This is accomplished with the commands:

```
$ cd /farmshare/user_data/[sunet_id]
$ git clone https://github.com/CME211/cme211-[github_user].git
```

with `[sunet_id]` replaced by your Stanford username and `[github_user]` replaced by your GitHub username. Nick's GitHub username turns out to be the same as his Stanford username. Therefore, Nick would execute the commands:

```
$ cd /farmshare/user_data/nwh
$ git clone https://github.com/CME211/cme211-nwh.git
Cloning into 'cme211-nwh'...
Username for 'https://github.com': nwh
Password for 'https://nwh@github.com':
```

```
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
Checking connectivity... done.
```

Please note that if you have enabled GitHub 2-factor authentication, you will have to create a personal access token and use that at the password prompt above.

Check the contents

Use the `ls` command to list the contents of your home directory to make sure the `cme211-[github_user]` directory exits. Use `cd` to enter the directory and then list (with `ls`) the contents in the directory. Here is what the process looks like for Nick:

```
$ pwd
/farmshare/user_data/nwh
$ ls
cme211-nwh
$ cd cme211-nwh/
/cme211-nwh$ ls
README.md
$
```

Create and save `STUDENT` file

The CME211 grading tools will look at the `STUDENT` file in your homework directory to determine who you are. Create and open a text file named `STUDENT` at the top level of your homework directory. For example, Nick could accomplish this with the `nano` text editor with the following commands:

```
$ cd /farmshare/user_data/nwh/cme211-nwh
$ nano STUDENT
```

Nick's `STUDENT` file has the contents:

```
[cme211-student]
name = Nick Henderson
stanford_email = nwh@stanford.edu
stanford_id = 01234567
github_user = nwh
```

Your `STUDENT` file must maintain the same header (`[cme211-student]`) and variable names (`name`, `stanford_email`, `stanford_id`, `github_user`). Note that these are all case-sensitive. You must replace Nick's information with your own. The `stanford_id` must be 8 digits, so include the leading `0` if you have one. You can check that the file exists with the following commands from your homework directory:

```
$ ls
README.md   STUDENT
$ cat STUDENT
# contents of STUDENT file displayed in terminal
```

Commit `STUDENT` and push to GitHub

Now let's walk through the process of committing the `STUDENT` file to the `git` repository and pushing the changes up to GitHub. First let's check the repository status (make sure that your shell is focused on your

homework directory):

```
$ 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)

    STUDENT

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

Here, `git` is telling us that there is a new ("untracked") file in the directory. We can add this to the repository with the command:

```
$ git add STUDENT
```

If all goes well, there will be no output after this command. We can again check the repository status:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   STUDENT
```

This tells us that `git` now knows about the new file. We can commit this to the local repository with the command:

```
$ git commit -m "add STUDENT file"
[master 88c188a] add STUDENT file
 1 file changed, 5 insertions(+)
 create mode 100644 STUDENT
```

Let's break this down:

- `git` is the command for the version control tool
- `commit` is a `git` argument saying that we want to commit to the local repo
- `-m "add STUDENT file"` let's us add the commit message on the command line. If you don't add a commit message on the command line, `git` will open a text editor where you can insert a message. All commits must have a message.

Now let's push the local changes up to GitHub:

```
$ git push origin master
# {authentication}
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 414 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:CME211/cme211-test-nwh.git
   d6f4819..88c188a  master -> master
```

Let's break this down again:

- `git` is the command for the version control tool
- `push` is a `git` argument saying that we push commits from local repo to a remote repo (GitHub)
- `origin` is a label for the remote repository (see `$ git remote -v`)

- `master` is the name of the main local branch

Now check the website for your GitHub repo to make sure the `STUDENT` file is here.

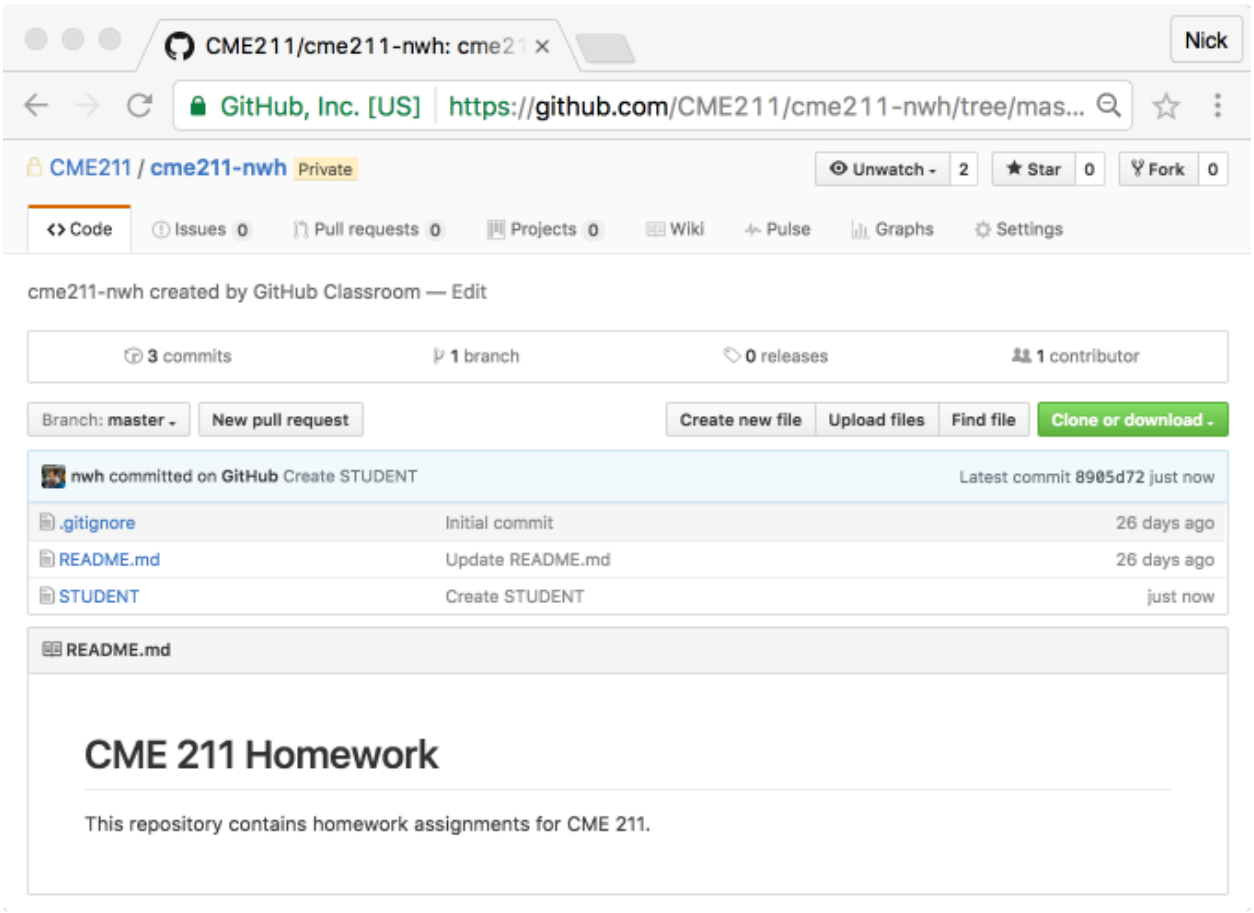

Figure 2: GitHub repository for CME211 with `STUDENT` file

Create first Python script

Let's get started with Python! First, we are going to begin by creating the `hw0` directory. For CME211, all assignments will go in an appropriately named directory in your homework repository. Let's create the `hw0` directory and `cd` into it:

```
# first, check the working directory
$ pwd
/farmshare/user_data/nwh/cme211-nwh
# create hw0 directory
$ mkdir hw0
# move into the new directory
$ cd hw0
$ pwd
/farmshare/user_data/nwh/cme211-nwh/hw0
```

Note: `hw0` must be lower case. Let's create our first Python program with the filename `hello_world.py`:

```
# check the working directory (should be hw0)
$ pwd
/farmshare/user_data/nwh/cme211-nwh/hw0
# create and open file with nano
$ nano hello_world.py
```

Insert the following line of code into the file then close the editor:

```
print("Hello world!")
```

To execute from shell:

```
$ python3 hello_world.py
Hello world!
```

Awesome, you've made a (very simple) Python program!

Create second Python script

All of the programs you write for CME211 will take some input from the command line. In this exercise you will write a Python scripts that reads command line arguments and echos them back to the terminal with the corresponding index.

The program should print a helpful "usage" message if no command line arguments are provided:

```
$ python3 echo_commands.py
Usage:
  $ python3 echo_commands.py [arguments]
```

If command line arguments are provided, then they are displayed to the terminal, in order, one argument per line, and prefixed by the index and a space. Index `0` corresponds to the name of the Python script. Here are a few examples

```
$ python3 echo_commands.py A B C
0 echo_commands.py
1 A
2 B
3 C
$ python3 echo_commands.py cme211 is great
0 echo_commands.py
1 cme211
2 is
3 great
```

We haven't taught you how to do this yet, so here is the code for `echo_commands.py`:

```
import sys

if __name__ == "__main__":
    if len(sys.argv) <= 1:
        # no argumemnts, print usage message
        print("Usage:")
        print("  $ python3 echo_commands.py [arguments]")
        sys.exit(0)
    # echo arguments
    for i in range(len(sys.argv)):
        print(i,sys.argv[i])
```

This will be a recurring pattern in all of your Python programs for CME211. It is important to become familiar with it early. Note that if you attempt to copy the code from the PDF, the formatting is likely to be wrong. Each indentation level is 4 spaces. Run the echo_commands.py to see if the results match the above three examples.

Commit changes, push to GitHub

We now need to commit the new Python files to the local repository and push to GitHub. First, let's `cd` to the top level of the homework directory and check the status. For Nick, the command sequence would be:

```
$ cd /farmshare/user_data/nwh/cme211-nwh
$ git status
Your branch is up-to-date with 'origin/master'.

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

    hw0/

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

This tells us that `hw0/` is an untracked directory. Let's add it to the local repository:

```
$ git add hw0
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   hw0/echo_commands.py
    new file:   hw0/hello_world.py
```

Note how adding the directory automatically adds all of the contents. Now, let's commit and push:

```
# commit to local repository
$ git commit -m "add hw0 python files"
[master 86b1c83] add hw0 python files
 2 files changed, 13 insertions(+)
 create mode 100644 hw0/echo_commands.py
 create mode 100644 hw0/hello_world.py
# push to remote (GitHub)
$ git push origin master
# {authentication}
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 566 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To git@github.com:CME211/cme211-test-nwh.git
   88c188a..86b1c83  master -> master
```

It is a good idea to check the website for your GitHub homework repository to verify that the files have been pushed.

Checklist

In summary, the requirements of Homework 0 are:

- Create a GitHub account if you don't already have one
- Visit provided link to create an empty CME211 GitHub repository for your homework
- Log into `corn.stanford.edu` via `ssh`
- Configure `git` with the `$ git config` command
- Clone your GitHub homework repository into `/farmshare/user_data/[sunet_id]`
- In your homework repository (`/farmshare/user_data/[sunet_id]/cme211-[github_user]`):
- Create a `STUDENT` file with your information in specified format
- Create a `hw0` directory (must be lower case)
- Create a `hw0/hello_world.py` script according to instructions above
- Create a `hw0/echo_commands.py` script according to instructions above
- Commit all created files to the local repository and push to GitHub