Overview

Homework 2 involves taking your homepage you made in Homework 1, and making it easier to manage with scripting, both in Bash and in Python. You also learn git and industry best-practices with GitHub.

As with learning many skills, it is always more beneficial to learn coding "the hard way" first, then learn how to do it the more elegant way. Coding your homework projects will be like that. [1]

Your Homework 1 assignment was to create several web pages by hand. Imagine you want to add another link to your site, or update content common to all pages: You'd have to go to each page and update it, or risk the pages not looking correct! This is a tedious process, and more importantly one that is prone to error. The solution is to use a Static Site Generator to automatically generate your pages based on commonalities (templates) and differences (content).

1 Requirements

1.1 Submission requirements

- **Must be deployed and functioning to GitHub Pages**
- **Must be built in two phases (bash and python), with each phase having a separate branch, and a separate pull request for that branch on GitHub.**
- Must have a `README.md` that indicates to users that they must run either `build.sh` or `build.py` to generate page
- Must move your static assets (that is to say, CSS, images, etc) into your `docs/` directory

1.2 Code requirements

In the end, your repo will look like this:

```
- templates/
    - top.html
    - bottom.html
- content/
    - index.html
    - contact.html
    - blog.html
    - projects.html
- docs/
    - css/           (CSS files, etc)
    - img/           (images, etc)
    - index.html
    - contact.html
    - blog.html
    - projects.html
- build.sh
- build.py
- README.md
```

- Both the Bash script `build.sh` AND the Python script `build.py` should do the same thing

[1] In a way, we are also recapitulating the evolution of web coding itself! It started with plain HTML, then developed CSS, then people started scripting it, and eventually they started building interactive web apps (what we will get to finally).

- Input files must be in `templates/` and `content/` directory
- Output files must go into `docs/` directory
- This builds on your Homework 1, so your site should already have at least 3 pages. If not, catch up now.
- The `templates` and the `content` directory are the "input" and not the final pages. The final pages that users actually see get generated by the scripts you will be writing and go into the `docs/` directory.

2 Steps

2.1 Launching HW 1 with git

Your first task is to launch homework 1 with Git. You can do this as soon as you learn about Git.

1. Create a new repo on GitHub (Make sure to select creating a README.md) file
2. Clone that onto your computer
3. Add, commit, and push all of HW 1 files to it
4. Using the "settings" ensure it is published

2.2 Writing Bash script site generator

The next step is to improve your site to use a Bash-based *static site generator*. A "static site generator" is a type of command-line tool that automates tedious tasks when managing a bunch of HTML files.

1. Create a new Git branch called `ssg-bash`
2. Create new directories `templates/` and `docs/`
3. Split your index.html file into 3 files:
   - The top part, up until your main content, should be in `templates/top.html`
   - The "main content" of your HTML file should go into `content/index.html`
   - Finally, the remainder of the page should go into `templates/bottom.html`
4. Create a bash script called `build.sh` that combines these three files, and output the result into a similarly named file in `docs/` (e.g. `templates/top.html + content/index.html + templates/bottom.html = docs/index.html`)
5. Rinse and repeat for all your pages.

Note: For this homework, and all subsequent ones, with any substantial addition or change to your work, you should get in the habit of doing a git add, git commit, and git push! This will save your work at each step.

Hint: For Bash, think about using `cat`, e.g.: `cat templates/top.html content/index.html templates/bottom.html > docs/index.html`

2.3 Creating the pull request

In "the real world" at companies that use GitHub or equivalent, new changes to a project (such as this) are typically integrated via "Pull Requests", a feature of GitHub that allows review and discussion of a branch before it is "merged" into the master branch.

1. Push your branch to GitHub if you haven't already
2. Create a new Pull Request for `ssg-bash` to `master`
3. "Accept" your Pull Request to combine your code with the master branch – don't delete your old branch, leave it around so it can be graded!

2.4 Writing Python script site generator

The next step is to replicate the same functionality you built with Bash, except in Python.

1. Switch back to the `master` branch, and create a new branch called `ssg-python`
2. Create a Python script file called `build.py`
3. Fill in Python code in this file to replicate the same thing you did for `build.sh`

Note: As with before, make a commit after every addition or change. You should have at least half a dozen commits by the end of this homework.

2.5 Create pull request for Python script

Follow the same steps as above to create and merge in a Pull Request, except for the `ssg-python` branch.

2.6 Publishing again

The "output" HTML should be in the `docs/` directory, so after we finish the other steps in the assignment, we need to configure GitHub Pages to use this directory instead.

Also, be sure to run `build.py` (or `build.sh`) to generate your actual pages, and then include those in git and push up those files, before you publish it! Remember, the `templates` and the `content` directory are the "input" and not the final pages. The final pages that users actually see get generated by the script.

3 Submission

Submit all of the following:

1. A link to your GitHub project
2. A link to the deployed, functioning site at yourname.github.io/projectname/
3. One link for each of the two pull requests for your `bash` and `python` branches.

4 Bonus

4.1 Site improvements

Same bonus section as last week - improvements to style since and content from last week, maybe more fully realizing your ideal style!

4.2 Buy a domain name

Buy a domain name for yourself from https://www.namecheap.com/ and research how to point it to your GitHub account.

4.3 Rudimentary templating

So far, your auto-generated site will have a defect: You cant make the <`title`> tag change from page to page, or the navigation change color or style when you are already viewing that page (e.g., with many bootstrap components, using the `active` class).

How might you fix this? One idea is to add text like {{`title`}} to your template files that gets replaced by Bash or Python with different things for each file.

We'll be going deeper into this next week as we further descend into Python!