Github Pages Tutorial

Ted Laderas (laderast@ohsu.edu), Eric Leung, and Robin Champieux

4/21/2017

What is GitHub Pages?

GitHub Pages is a system built into GitHub that allows you to build websites directly from a GitHub Repository.

There are two kinds of GitHub pages:

- Personal websites (associated with an account)
- Repository Specific Websites (webpages for software)

We'll mostly cover the personal website for today.

Why GitHub Pages?

GitHub is a highly dependable webhost, and you get one personal webpage account for free.

Because GitHub pages is compatible with markdown, we'll be using markdown formatting to build our webpage.

Our goal for the day

We're going to build a personal webpage which will live at USERNAME.github.io, and have blogging features and will be easily customizable.

There's an example of a blog here: https://biodata-club.github.io/githubPagesTutorial/

What You Need for Today

You don't need to know how to code to setup a GitHub page. We'll learn some Markdown basics so that we can add content to our site.

- A GitHub Account
- GitHub Desktop
- Text Editor (we prefer Sublime, but Atom is also good)
- File Manager (to rename files)
  - Windows Explorer or Mac OS Finder (whatever you have)
- Something to say
- Pictures (optional)

You may want to spend a little time learning how to use markdown, as it'll make putting your content together faster. There is a tutorial here: The Magic of Markdown

What We're Going to do Today

1) Download the tutorial repository
2) Make a local repo with the `websiteFolder`
3) Publish your Repo to GitHub

4) Look at your pretty website!
5) Change and Personalize Your Site with YAML
6) Start Editing Content with Markdown
7) Adding Links
8) Adding Images
9) Add a Blog Post
10) Customize your sidebar with new links
11) Look at your pretty website!

1. Download the Repo(istory) From GitHub

Clone the the repo at: https://github.com/BioData-Club/githubPagesTutorial/ (click the "Clone or Download" button to open it in your GitHub Desktop).

2. Make a local repo

Using a file manager, take out the `websiteFiles` folder from the `intermediate` folder and rename it to USERNAME.github.io, where USERNAME is your GitHub username.

I like to have a code specific folder to store my code, such as "c:/code/" (windows) or "~/Code/" (Mac/Linux) so that it has a permanent place to stay.

This will be important when you push changes to your website. So create a code-specific folder and put the USERNAME.github.io folder in it.

3. Publish Your Repo to Github

In GitHub Desktop, add the repo using the "+" symbol and use the "Add" tab to add your folder. "Publish" the folder to GitHub using the "Publish" button.

Check that your repo exists (it will be at http://github.com/USERNAME/USERNAME.github.io/) before moving on.

4. Look at your website

Once you've published the repository, try your personal webpage: http://USERNAME.github.io

Now we'll start customizing your webpage.

5. Editing the Name of Your Webpage and Customizing

The first step to customizing your webpage is to edit the blog's name to include your information. To do this, you will be editing a YAML file, which is a simple configuration file. Open the `_config.yml` file using a text editor (such as Sublime) to update. Change the *title*, *tagline*, *description*, *url*, and *name* fields.

For now, don't modify anything below`#You don't need to modify below this line`.

Here's an example `_config.yml` file.

```
# Personalize your website info here
title:            Anew Guy
tagline:          'Professional Website for Anew Guy'
description:      'Website and Blog for Anew Guy'
```

```
url:               http://anewguy.github.io
author:
  name:            'Anew Guy'
```

After you have edited your `_config.yml` file, save it. Now you'll update your website by commiting these changes and pushing them to the website. To update the local files you've changed and push them to your GitHub, you will need to create a commit.

In GitHub Desktop, click the "Uncommitted Change" button. Create a commit summary, which is short description the changes you've made. Commit descriptions are good practice, and allow you to better identify and even revert changes in the future. Then push the "Sync" button to update your repository and website.

Now take another look at your peronal webpage: http://USERNAME.github.io Note that because GitHub is processing your file in the background, it may take up to 30 seconds for your changes to appear.

6a. Make content using Markdown

We'll be adding our content by using Markdown Files. Markdown is a lightweight way to format text that is much easier to manipulate than working with HTML files directly.

GitHub has a "translation engine" that will transform and format your files for the website.

This means you can concentrate on making content rather than obsessing about every little site detail. There are lots of designers who are good at that if you aren't.

6b. Markdown Basics

A markdown file generally has an `.md` or a `.markdown` extension. Let's open the file in `_posts/2012-02-07-example-content.m` in your text editor and you'll see some examples of how to write in markdown.

Way more info about writing markdown can be found here: https://github.com/laderast/magic-of-markdown

And this markdown cheatsheet is always handy if you forget how to do things. https://enterprise.github.com/downloads/en/markdown-cheatsheet.pdf

Try changing the `about.md` or the `cv.md` files with your info. When you're done, save your changes, commit, and push. Confirm that your changes worked.

7. Let's Add Some Links

Links, or URLs and web addresses, you see on websites can link to other webpages.

In your `about.md` file, you can add a link by enclosing the text for your link in square brackets (`[text here]`) and the underlying link in parentheses (`(http://link-here.com)`) right next to it. So altogether, it should look like this:

```
[link to Google](https://www.google.com)
```

The output looks like this: link to Google

If you just want to show the link, you can paste it as is into your file.

```
https://www.google.com
```

https://www.google.com

8. Let's Add an Image

You can also add images to your webpage. GitHub has a mascot called Octocat, so let's add a picture of one to our site. Similar to how we added a link, you just need to add an exclamation mark in front of the square bracket and parentheses.

An example looks like this

```
![Jetpacktocat](https://octodex.github.com/images/jetpacktocat.png)
```

The output looks like this:

Figure 1: Jetpacktocat

Note: the text within the square brackets should generally be a description of image, but you can leave it blank if you want so the following would also work.

```
![](https://octodex.github.com/images/jetpacktocat.png)
```

Storing images in your website

Where do you put images? You can make an `images/` folder in your repository. Then you can refer to them like this:

```
![](/images/jetpacktocat.png)
```

9. Add a Blog Entry

Your current repository is setup for blog entries. In your newly renamed `websiteFiles/` folder, you'll find another folder named `_posts`. This is where you'll put new blog posts.

Each post you make should be in the form `YEAR-MONTH-DAY-title.md`, so for example, `2017-04-21-github-pages-tutorial.m` You'll also want a `.yaml`-like text near the top.

```
---
layout: post
title: Awesome Blog Post
---

Text for your post goes here.
```

When you're done, commit and push the file into GitHub and confirm that your new blog post is up. You can also delete the sample blog posts so they don't show up. (Make sure to commit your deletions.)

10. Modifying the Menu

If you take a look at the example site, https://biodata-club.github.io/githubPagesTutorial/, you'll notice a menu side bar on the left. Near the bottom left corner, you'll see links to other pages on the site. We can add and remove these links and their respective pages fairly easily.

Looking at the files in your main directory (which are the contents from `websiteFiles/`), the menu is automatically generated based on the files that end in `.md` (except `LICENSE.md` and `README.md`). If you take a look, you can see `about.md`, `cv.md`, and `about.md`. Similarly named links should be in the menu.

To create a new page, you need to have the following at the top and just change the text for the title.

```
---
layout: page
title: Coursework
---

Text for your page goes here.
```

Try adding a `coursework.md` file in the main directory by pasting the text above and then writing some content, such as classes you've taken.

You can add some bullets by using the `+` symbol like this:

```
+ Basket Weaving
+ Cooking
+ GitHub Class
```

Which will show up like this:

- Basket Weaving
- Cooking
- GitHub Class

When you're done, commit, and load your website again. Confirm that "Classes" shows up in the menu sidebar. Neat! (If you don't like this page, you can always delete it when you're done).

11. Look at your pretty website

Congratulations on making it this far! This is just the start of your adventures with GitHub Pages.

Have fun!

More Stuff

The following are things you might like to do to your website.

Making folder links using a `Index.md` file

You might be wondering how to generate links that work with a trailing slash. For example, http://laderast.github.io/DS4BS/

The first step is to create a folder with that name, in the top level folder. For example, I just created a folder called `DS4BS` in my `laderast.github.io` folder. Then I just created a file called `index.md` in that folder.

Be sure to add the correct YAML at the start of your `index.md` file:

```
---
layout: page
title: New Page
---
```

If you have a file named `index.md` in your folder, your link will be automatically translated and served as a webpage. For example, if you added a folder named "Stuff", and put an `index.md` in the folder, the link USERNAME.github.io/stuff/ will open that `index.md` file.

Using Custom Themes

You might not like the layout of your page (that's fine). You can transfer your markdown files into a different Jekyll Theme and use that as the basis for your website.

http://jekyllthemes.org

Many of these themes you can directly fork and make a website directly.

If you know HTML

The cool thing about markdown is that you can add HTML directly into markdown files and the HTML should integrate pretty seamlessly. This is great when you want to customize placement of things.

For example, maybe we want a smaller version of octocat. We can just add an `<img>` tag to customize it, and change the `height` attribute.

```
<img src="https://octodex.github.com/images/jetpacktocat.png"
height="300">
```

Useful Links

- GitHub Pages: https://pages.github.com
- Jekyll Themes: http://jekyllthemes.org

Advanced Topic: Jekyll and Liquid Templating

If you're interested in setting up a more complicated site, you'll have to learn a little bit about Jekyll and the liquid templating language. Liquid tags are defined by putting them between curly brackets with percents `{% %}`.

Liquid tags are basically instructions for Jekyll so you can dynamically change the content of your site. For example, the sidebar on our page uses the following liquid script (look in `includes/sidebar.html`). Basically, it looks for any file that has the YAML entry of `layout: page` and generates a link to that.

```
{% comment %}
    The code below dynamically generates a sidebar nav of pages with
    `layout: page` in the front-matter. See readme for usage.
  {% endcomment %}

  {% assign pages_list = site.pages %}
  {% for node in pages_list %}
    {% if node.title != null %}
      {% if node.layout == "page" %}
        <a class="sidebar-nav-item{% if page.url == node.url %} active{% endif %}" href="{{ node.ur
      {% endif %}
    {% endif %}
  {% endfor %}
```

Most of the time, you will not put liquid script in your markdown files, but rather in the `.html` files that are used in the basis for your Jekyll Template.

Liquid is a super powerful way to make your site portable across hosts. For that reason, it's worth knowing a couple of liquid variables, such as `site` and `page`. For more info, please look here: https://jekyllrb.com/docs/variables/

`_layouts`

Did you wonder what the `layout: default` YAML refers to? If you look in the `_layouts` folder, you will see a file called `default.html`, which is the default layout for your Jekyll installation.

With a little knowledge and tweaking, you make your own layout files using liquid templating.

include

You can add content from the `_includes` folder by using the include tag. For example, `index.html` in the base folder adds the content of `footer.html`

```
{% include footer.html %}
```

for loops

Let's look at `index.html` in the base folder of the installation. You'll see it's a mix of HTML and liquid, and that it takes a list from `paginator`, called `paginator.posts`. Using liquid, we pull out the `url` and `title` values from each post to make the `post-title` heading.

```
<div class="posts">
  {% for post in paginator.posts %}
  <div class="post">
    <h1 class="post-title">
      <a href="{{ post.url }}">
        {{ post.title }}
      </a>
    </h1>

    <span class="post-date">{{ post.date | date_to_string }}</span>

    {{ post.content }}
  </div>
  {% endfor %}
</div>
```

Advanced Topic: Repository Pages

As we mentioned before, each repository has it's own page. These can be customized under the **Settings** page on your repo.

Basically, you will need to make a new branch called "gh-pages" that contains your website (*do not merge this branch with your other code*!). When you have created this branch, your repo website will appear at http://USERNAME.github.io/repoName (where repoName is the name of your repo).

Note that once you have your "gh-pages" branch, you can change the jekyll theme under "Settings". GitHub Pages offers a number of built-in Jekyll Themes that might be a good place to start.

Way more info can be found here: https://help.github.com/articles/creating-a-github-pages-site-with-the-jekyll-theme-chooser/

https://help.github.com/articles/adding-a-jekyll-theme-to-your-github-pages-site/

http://stackoverflow.com/questions/31327045/switch-theme-in-an-existing-jekyll-installation