Testing mkdocs

Workshop documentation from Markdown

Brian Innes

Table of contents

1. MkDoc Test

This site show how to use MkDocs and GitHub Actions to convert project documentation written in Markdown to a static web site, which can be hosted on GiHub Pages.

This provides a much better user experience than GitHib rendered Markdown.

In addition to the HTML formatted site, MkDocs can also generate pdf documentation. The pdf generated for this site can be found here

The default themes supplied with MkDocs creates a site that is reactive to screen size, so works on all devices automatically. However, selecting one of the community themes can add additional features, such as:

- **copy to clipboard** on all code blocks
- links to GitHub repo as well as an **edit on GitHub** link on pages
- different navigation options for site and current page
- link to pdf documentation in footer

Combining **MkDocs**, **GitHub Actions** and **GitHub Pages** enables an automatic process of keeping the documentation site on GitHub Pages up to date. You only need to focus on creating the documentation in Markdown.

When the Markdown pages are pushed to the master branch of the GitHub repository, a GitHub Action will automatically be triggered to generate and publish the documentation to GitHub Pages. If your git infrastructure doesn't provide Actions then you can manually generate and push the documentation to GitHub Pages.

These facilities are available on both Free and paid GitHub accounts, though some Enterprise GitHub installations may not have the GitHub Actions feature.

To enable this capability in your project, you need to have a local clone of your GitHub project on your system, then:

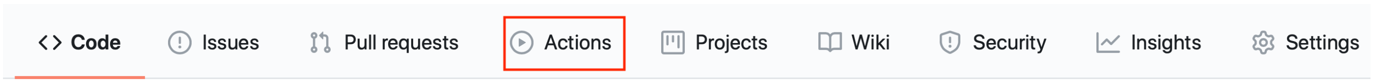
1. Add the .github directory, and all the content within the directory and sub-directories, from this project to your own project (covered in more details in the GitHub Actions section)
2. Add the mkdocs.yml to the root of your project and configure it for your project (covered in more details in the MkDocs section)
3. Configure your GitHub repository for GitHub Pages (covered in more details in the GitHub Pages section)
4. Push all the changes to the master branch of your GitHub repository
5. Update the GitHub Pages configuration to point to the generated site

2. GitHub Actions

GitHub Actions are an automation feature build into GitHub. They allow actions to be triggered as a result of activity, such as a push or pull request, on a repository

GitHub Actions are easy to set up and are available on both the free and paid GitHub plans. Note: Actions are not yet available on all Enterprise Git sites.

You can easily tell if your git project is enabled for Actions by seeing if there is an Action section:



Full documentation is available for GitHub actions, but follow these simple steps and use this repository as a reference. The action used in this project publishes the project documentation (in the **docs** folder of the repo) to a GitHub pages site using the MkDocs static site generator.

1. Create a **.github** directory in your repository
2. Add an **actions** directory inside the **.github** directory
3. Create an action to generate and publish the site using MkDocs (*see below for instructions*)
4. Add a **workflows** directory inside the **.github** directory
5. Create a workflow to checkout the code from git and then run the action created in step 3 to generate the site

2.1 Actions

An action is an individual task, which can be run as part of a workflow initiated when something happens on a GitHub repository. There is a marketplace of actions available or you can create your own private actions.

In this example a private action is created.

Private actions can exist in the git repository where they will be used. I recommend adding them within the .github directory to separate the GitHub action configuration with the primary purpose of the github repository.

See the documentation for all the options for creating actions, but this example uses a container to perform the operations of the action.

An action is defined by a configuration file, called **action.yml**, again the syntax of the config file is described in the documentation. The **action.yml** file for this project contains the following:

```
# action.yml
name: 'Deploy to GitHub Pages'
description: 'Publish Markdown docs as GitHub Pages static site'
runs:
  using: 'docker'
  image: 'Dockerfile'
```

Here you see a name and description, then a **runs** section which uses a Docker container to perform the work. The Docker container image can be specified using a **Dockerfile** - the path is relative to the **action.yml** file, so in this case in the same directory. It is also possible to specify an image from a public image repository, such as Docker Hub.

The Docker file for this project contains the following:

```
FROM python:alpine

RUN apk add --no-cache \
    build-base \
    git \
    git-fast-import \
    openssh

RUN pip install --no-cache-dir mkdocs

COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

which simple installs the prerequisite packages to install and build the mkdocs runtime, then copies in the script which will run when the contain is run.

The entrypoint script contains the following:

```
#!/bin/sh

mkdocs gh-deploy --config-file "${GITHUB_WORKSPACE}/mkdocs.yml" --force
```

which simple runs the mkdocs command, with the **gh-deploy** option to deploy to github pages. Notice the use of the **GITHUB_WORKSPACE** environment variable, which the GitHub Actions runtime initialises at the container startup.

2.2 Workflow

The workflow defines when a GitHub action will run and what actions will be run. For a project the workflow files must exist in the **.github/workflows** directory structure at the root of the project.

The workflow file (**build.yml**) for this project contains the following:

```
name: GenerateSite

on:
  push:
    branches: [master]
  pull_request:
    branches: [master]
jobs:
  generate:
    name: 'Run mkdocs gh-deploy'
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository
        uses: actions/checkout@v2

      - name: generate site
        uses: ./.github/actions/
```

Here you can see the workflow has a name then the **on** section defines when this workflow will run. Here I specified that this workflow will only run when something is pushed to the master branch or a pull request is made to the master branch. Activity on other branches will not trigger this workflow.

Info

GitHub as renamed the default branch from **master** to **main** in all new repositories

When the workflow triggers a single job is run within a Ubuntu Linux container. The job has 2 steps that are run sequentially.

The first one uses an action from the marketplace to checkout the latest code from the repository (after the push or pull request that triggered the workflow has taken place).

The second one is our private action to generate the MkDocs site and publish it to GitHub Pages. Notice the uses property points to the directory containing the **action.yml** file, which is relative to the project root directory.

You can use the **Actions** section of the GitHub web UI to check on the progress of actions and see the status of all Action triggered and provides access to the logs generated by all actions that have previously run.

You can add a badge to your project README.md, to give a visual indication if the latest Action completed successfully. The badge for this site is created using the following:

```
GitHub Pages : ![GenerateSite](https://github.com/binnes/mkdocTest/workflows/GenerateSite/badge.svg?branch=master)
```

2.3 User Limits

GitHub Actions are available on most GitHub accounts, though there are usage limitations, but for documentation formatting the limitations should not pose an issue.

3. MkDocs

MkDocs is an open source project that takes Markdown documents and generates a static web site. The static web site does have some useful features, such as navigation and search.

MkDocs is easy to install on all common operating systems and provides a good developer experience, where you can see real time changes as you work on a document.

You also have multiple built-in or community themes available and have the option to customize an existing theme or create a new theme the get the documentation layout and style exactly as you want. For this project the default theme is used.

3.1 Installing MkDocs

Before you can use MkDocs you need to install it. The documentation provides instructions for installation of the python 3 prerequisites and MkDocs

This project uses some additional plugins and extensions to the core MkDocs, which are discussed later in this section. You also need to install the extensions and their prerequisites if you want to use the additional features.

3.2 Configuring MkDocs

MkDocs is configured using a file named **mkdocs.yml** located in the root directory of your project. The minimum configuration is simply a site name.

```
site_name: Testing mkdocs
site_description: >-
  Automatically create formatted documentation from Markdown files in a GitHub repository.
  The Markdown is automatically formatted using Docs, generated when a GitHub pull request or push
  is made to the repository using GitHub Actions and hosted on GitHub Pages.
site_url: https://binnes.github.io/mkdocTest
site_author: Brian Innes
repo_name: binnes/mkdocTest
repo_url: https://github.com/binnes/mkdocTest
edit_uri: ""
use_directory_urls: false
theme:
    name: material
plugins:
  - search
  - with-pdf:
      cover_subtitle: Workshop documentation from Markdown
      output_path: pdf/mkdocs.pdf
markdown_extensions:
  - attr_list
  - admonition
  - toc:
      permalink: true
extra_css:
    - css/extra.css
    - css/pdf-print.css
nav:
  - Home: index.md
  - Actions: GitHubActions.md
  - MkDocs: MkDocs.md
  - Pages: GitHubPages.md
google_analytics:
  - 'UA-172132667-1'
  - 'binnes.github.io/mkdocTest'
```

You can see all the configuration options in the MkDocs User Guide, but some things to note:

- The site_name is a mandatory field and is used as the title for the site.
- The site_description field provides the description header metadata, which is used by a number of tools to summarise the content of a site
- The default location MkDocs looks for docs to render is the **docs** directory in the root of your project. This can be changed by adding the **docs_dir** configuration option.
- The default location MkDocs will write the rendered site is the **site** directory. This can be changed by adding the **site_dir** configuration option.
- MkDocs automatically adds the **search** plugin to generate a search capability on a generated site, but it must be specified when adding a **plugins** section to use additional plugins
- The default theme is **mkdocs**, but this can be altered using the **theme** configuration options.
- The **attr_list** Markdown extension has been enabled to allow additional HTML attributes to be added to markdown. This is used to allow links to be opened in a new tab or window rather than leaving the site by adding the **target attribute**. So the Markdown for the link in this section is specified in Markdown as : `[additional HTML attributes](https://python-markdown.github.io/extensions/attr_list/){target=_blank}`
- You control the navigation options of the published site using the **nav** configuration option.
- MkDocs is enabled for Google Analytics, simply specify your details to start collecting data on a site.

This project is using GitHub Actions to generate the site, so the site directory shouldn't be pushed manually into the GitHub repository, so this project has added the **site/** directory to the **.gitignore** file, to prevent it being added to the repository.

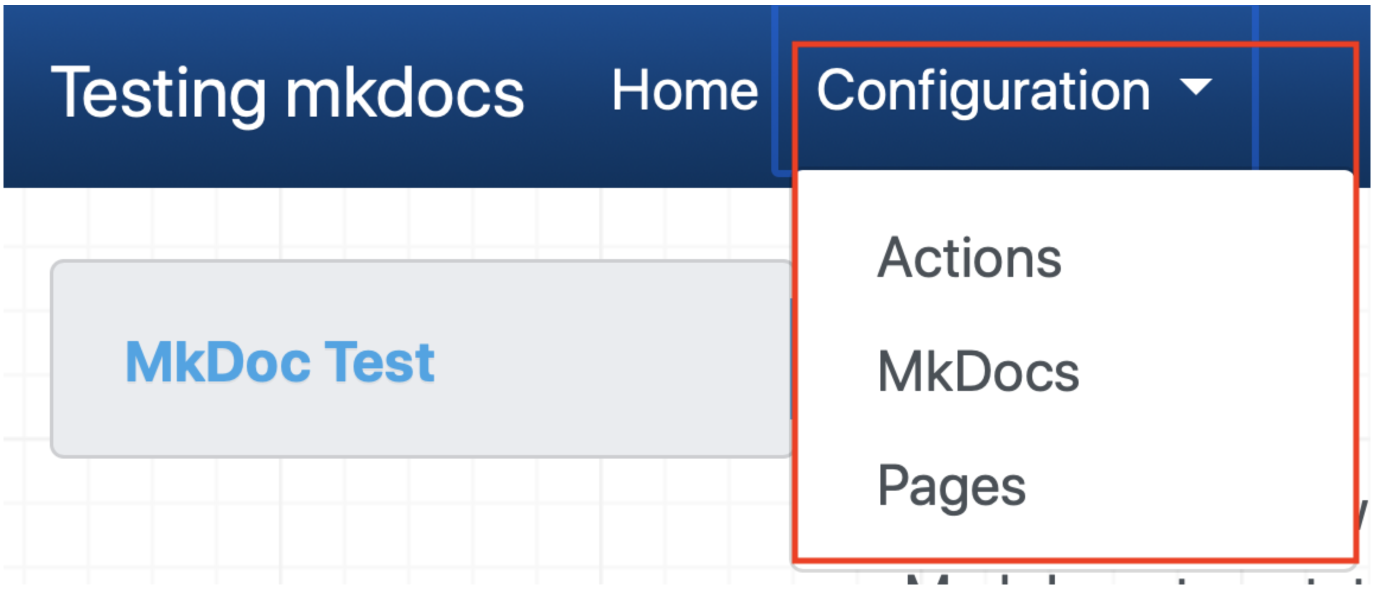
3.3 Controlling site navigation

The **nav** configuration controls the navigation on the site. Different themes can support different levels of navigation nesting and offer different navigation options on the page.

You can control the navigation nesting by adding a section in the configuration. If I changed the navigation for this project to:

```
nav:
- Home: index.md
- Configuration:
  - Actions: GitHubActions.md
  - MkDocs: MkDocs.md
  - Pages: GitHubPages.md
```

the navigation now shows the top level **Configuration** option as a drop down list containing the 3 sub-level entries.



3.4 Publishing the site

MkDocs has a built in option for deploying to GitHub Pages.

The default option is to push the generated site to a branch in the GitHub repository called **gh-pages**. This default can be changed using the **remote_branch** configuration option.

As this project uses a GitHub Action to automatically publish the site when changes are pushed to the master branch it shouldn't be necessary to publish manually.

To manually publish documentation to GitHub Pages:

1. Open a terminal window
2. Navigate to the root directory of your cloned GitHub repository
3. Run the **mkdocs** command:

```
mkdocs gh-deploy
```

3.5 Options for customisation

MkDocs works pretty well with the default theme and configuration, but it is possible to change the styling and functionality of the generated site or add more complex Markdown formatting options.

3.5.1 Options for changing the generated site look and feel

The default **mkdocs** and **readthedocs** themes provide good options for the layout, navigation and style of the generated site, but the community provided themes offer additional options. This site is using the **Material** theme from the community.

Using a community theme is usually pretty easy:

1. install the theme - this is usually done using the pip package manager e.g. `pip install mkdocs-material`
2. configure **mkdocs.yml** to set the theme e.g.

   `yaml`
   `theme:`
   `    name: material`

However if none of the provided themes are exactly what is needed then there are 2 options:

1. Customise an existing theme - which is documented in the MkDocs user guide
2. Generate your own custom theme - which is also documented in the MkDocs user guide

Note

If you choose a community theme or generate your own theme, then you might need to modify the Dockerfile for the GitHub Action if additional packages are needed for the theme.

3.5.2 Markdown extensions

MkDocs uses Python Markdown to translate the Markdown files into HTML, which supports extensions. You can modify the default set of extensions that MkDocs uses to add support for additional Markdown features using the **markdown_extensions** configuration options in the mkdocs.yml configuration file.

This project has the **attr_list** extension enabled to allow additional HTML attributes to be added when formatting pages. In this project it is primarily used to add the **target** attributes to external links. An additional use is to resize an image by specifying a width tag `{width=600}`

Note

MkDocs supports HTML attributes, but if you want to generate a pdf of the site, then you need to avoid HTML attributes and use CSS to control presentation. So the {width=600} HTML attribute should be rewritten to use CSS, so becomes {style="width: 600"}

If you choose add a Markdown extension, then you may need to modify the Dockerfile for the GitHub Action to ensure the extension is installed. The officially supported extensions are usually installed by default, but third party extensions will need to be installed so they are available when the action is run to generate the site.

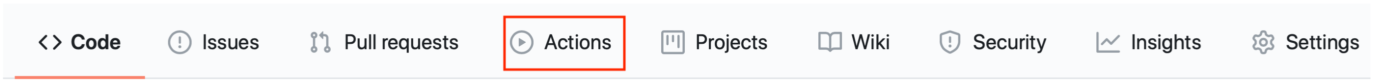
Example of customisation

Adding the **attr_list** extension we can alter the appearance of images.

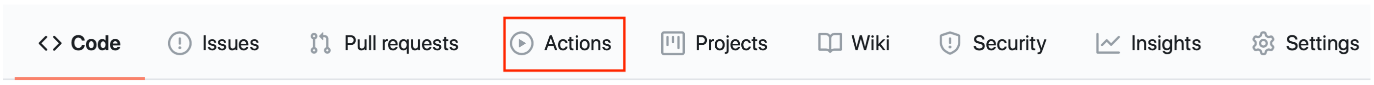
Warning

The additional markup used for formatting by Markdown extensions will not be rendered by the GitHub Markdown renderer. The extended Markdown will appear on pages viewed through the GitHub repository pages.

The Markdown `![menu](images/menu.png)` will generate the image below:

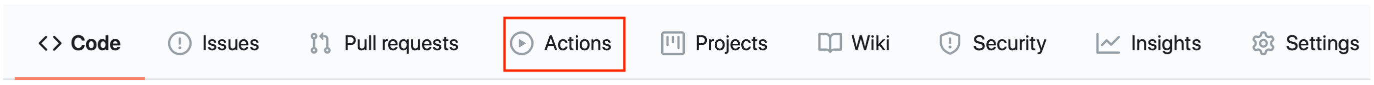


However, you may want to change the size of the image, so we can use any of the CSS options for specifying a size (px, em, rem, %, ...). The following will make the image 50% of the width of the column containing the image: `![menu](images/menu.png){style="width: 50%"}` which produces the following output:



You can also add additional styles to an element, so to centre the above image the following can be used: `![menu](images/menu.png){style="width: 50%; display: block; margin: 0 auto;"}`

which creates the following:



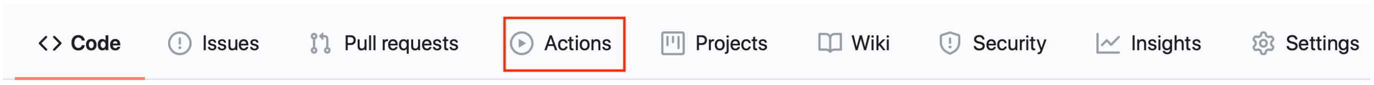
Note

You can combine customising the style using additional CSS with the attr_list plugin to create a custom style to apply to Markdown. If the theme provides a suitable style then you can also use it rather than defining a new custom style where appropriate.

If I create a file within a css folder inside the docs folder called **extra.css** containing:

```
img.center {
  display: block;
  margin: 0 auto;
}
```

and configure MkDocs to use that file by adding the **extra_css** configuration to the **mkdocs.yml** configuration file, then I can just apply the class to the image: `![menu](images/menu.png){style="width: 50%" .center}` which produces the same output as using the style attribute above.



You may also have noticed the **Note** section above. This is created using the **admonition** Markdown extension and the following Markdown text:

```
!!! note
    You can combine customising the style using additional CSS with the attr_list plugin to create a custom style to apply to Markdown.  If the theme provides a
suitable style then you can also use it rather than defining a new custom style where appropriate.
```

The default **mkdocs** theme supports admonition tags of **note**, **warning** and **danger**. You can add additional tags by creating custom CSS in the **extra.css** file.

3.5.3 Plugins

Plugins provide more advanced customisation within MkDocs, such as providing the search capability within the generated static site. You can create your own plugins or use one of the community provided plugins to add additional capability to the generated site.

Again any plugins used in a site will need to be installed, so the Docker file for the GitHub Action will need to be modified to ensure all plugins are available when the action runs to generate the site.

3.6 Generating PDF documentation for your site

In additional to a static web site, MkDocs can also generate a PDF file containing all the documents in the site combined into a single PDF. To do this you need to use a plugin. There are a number of plugins available that will generate a pdf, but the one I use is called **MkDocs with pdf**. Before you can use it you need to install it along with the prerequisites. The installation is documented in the plugin project README file.

Once the plugin is installed it can be added to the MkDocs configuration file, in the plugin section:

```
plugins:
  - search
  - with-pdf:
      cover_subtitle: Workshop documentation from Markdown
      output_path: pdf/mkdocs.pdf
```

Note

You need to add the search plugin when adding the plugins section. The search plugin is added by default if no plugins section is added, but needs to be manually added when a plugins section is created in the configuration file.

You will see there is an option to provide a subtitle to be placed on the cover page.

Once the plugin has been added to the MkDocs config file it will be run when a **mkdocs build** command is issued, so each time the static HTML site is built then a new pdf will be created. When using **mkdocs serve** the pdf will not be built.

The default location for the pdf is within the site folder. A new folder named **pdf** is create and the pdf file is named **document.pdf**. These defaults can be modified by providing the **output_path** configuration property.

The **Material** theme automatically adds a link to the generated pdf at the bottom of each page of the static website. If you're reading this served from GitHub pages, then check the bottom of this page to see the link to the pdf version of this documentation.

3.6.1 Explicitly define all Markdown pages

When generating a pdf, you need to place all Markdown documents you want included in the pdf in the navigation configuration within the **mkdocs.yaml** configuration file.

When generating a static site MkDocs will include Markdown pages that are not specified in the site navigation, so long as they are linked to by a page defined in the navigation.

However, a pdf document needs to have everything explicitly declared as it specifies the order in which the Markdown pages appear in the pdf. Any page not explicitly declared within the **nav** configuration will not be included in the pdf, so any links to those pages will not work in the pdf.

A common approach to include content which would otherwise not appear in the site navigation is to have an **Appendix** or **Additional Resources** section at the end of the pdf.

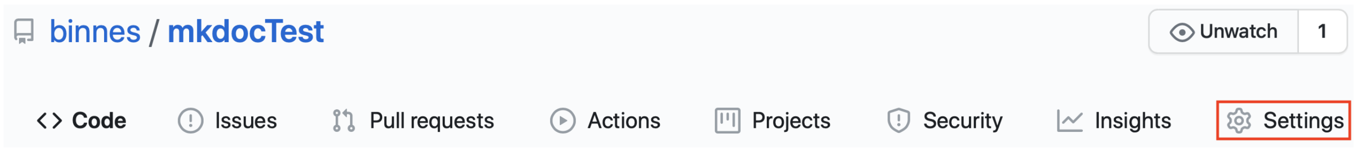
4. GitHub Pages

GitHub Pages provide the option for GitHub to host a static website for a project.

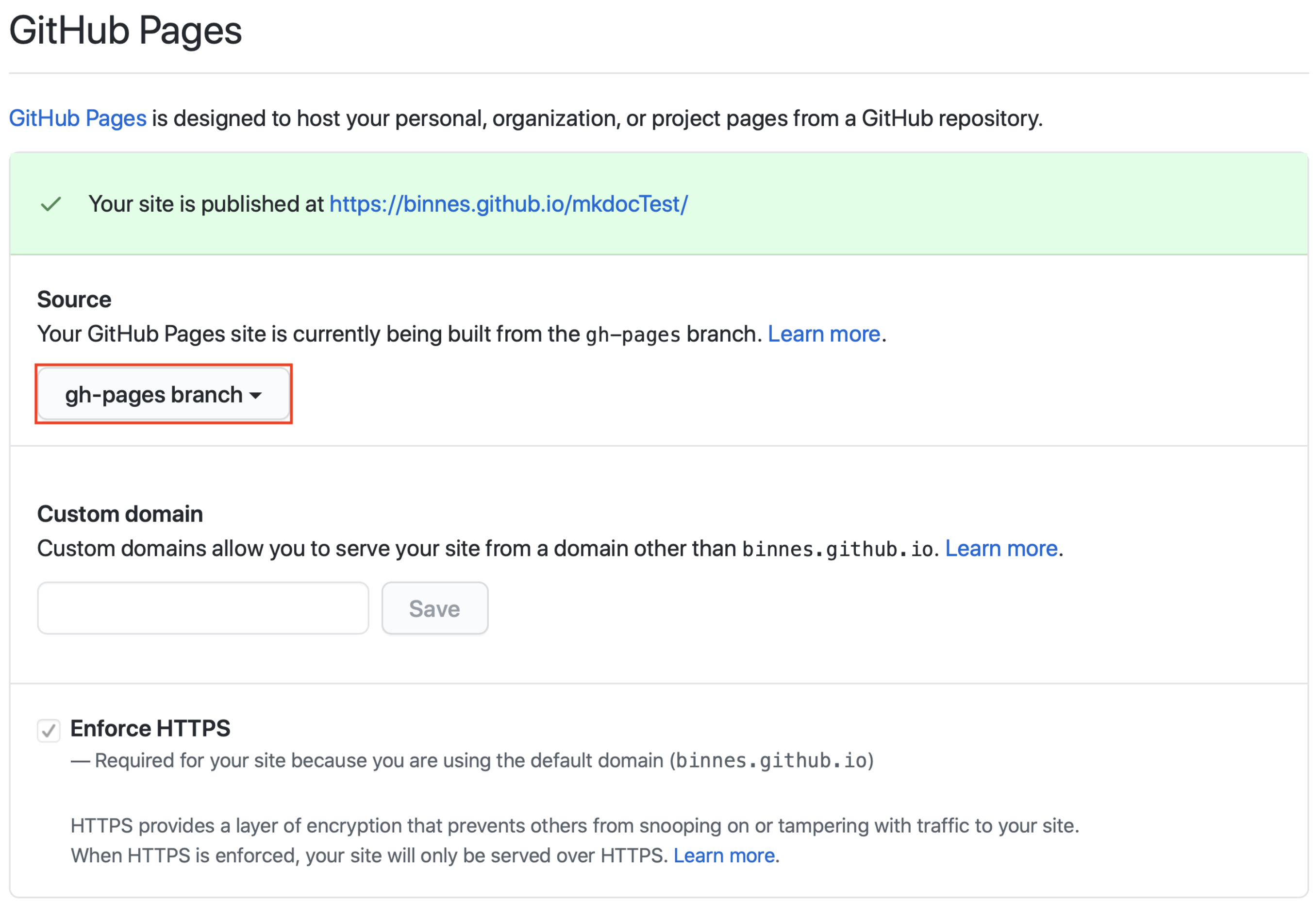
By default when you create a new repository there is no GitHub Pages site creates, but you can easily configure a site.

To configure GitHub Pages you need to use the GitHub web console and navigate into your project.

1. Select the Settings section.



2. Scroll down until you see the GitHub Pages section. There select the gh-pages branch as the Source for your site. This option will only be available after the `mkdocs gh-deploy` has been run.



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is published at https://binnes.github.io/mkdocTest/

Source
Your GitHub Pages site is currently being built from the `gh-pages` branch. Learn more.

gh-pages branch

Custom domain
Custom domains allow you to serve your site from a domain other than `binnes.github.io`. Learn more.

Save

Enforce HTTPS
— Required for your site because you are using the default domain (`binnes.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site.
When HTTPS is enforced, your site will only be served over HTTPS. Learn more.

Once configured you will see the URL for your site