Jeremy DiBattista Follow

Apr 12 · 5 min read · ✨ · Listen

Easily Automate Your Documentation and Never Touch It Again

Completely automated documentation process in Python using MkDocs

Everyone's favorite aspect of being a developer: maintaining documented code and hosting that documentation on GitHub for their fellow developers.

I am, of course, joking. Documentation is too often a task that gets neglected and makes new members taking over dated codebases a wasteful and painful task. We have all been there — what was the developer before me doing here? And… How does this even work?

No more. With the help of Mkdocs, Mkgendocs, a custom script I have written, and GitHub actions, we can maintain gorgeous documentation like this automatically!

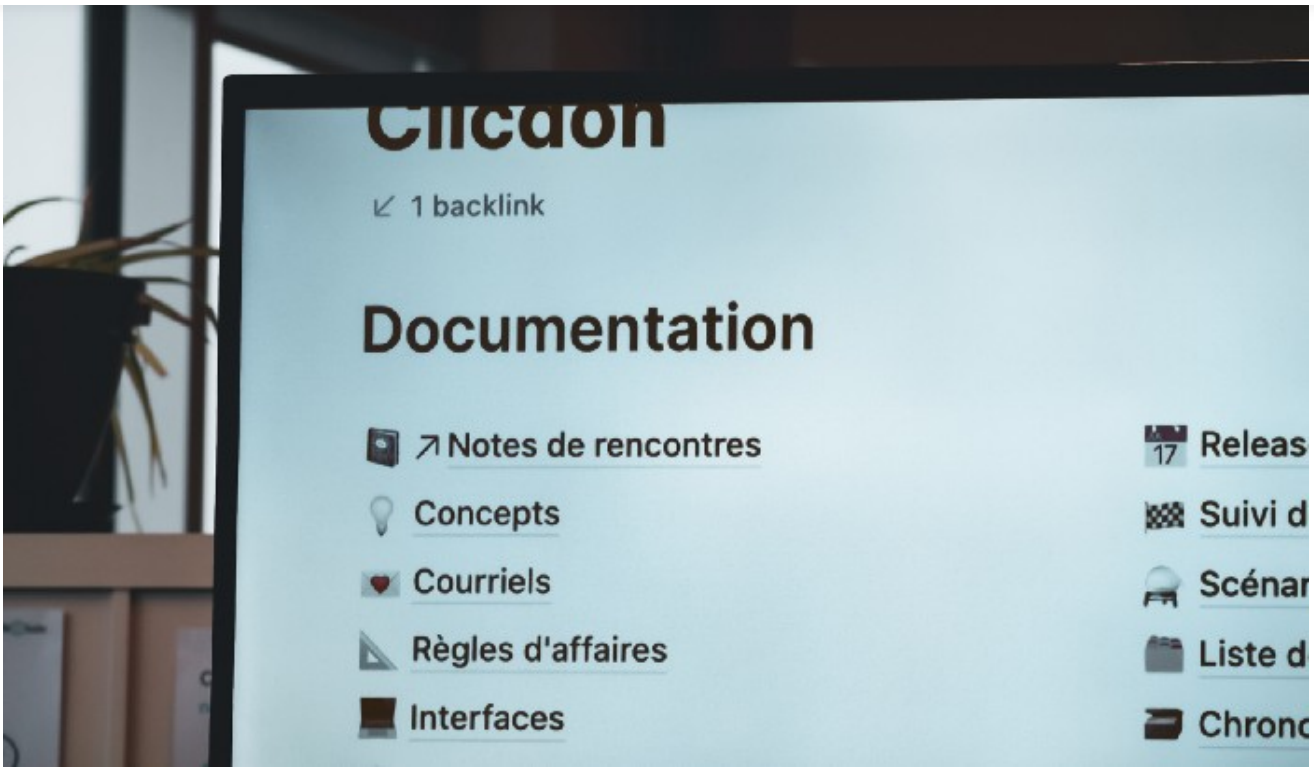

Photo by Sigmund on Unsplash

Step 1 — Make sure your codebase has Google-style python docstrings

The first step in automating is just adding google style docstrings to your python files. For atom text editor users, I recommend the docblock atom extension to automatically do this!

This step is not strictly necessary, but any docstrings added in this manner will be stripped out by the next steps and added to our site! This step can also be done gradually at your leisure. The more documentation you add, the more fruitful your automatic documentation will be!

Step 2— Install necessary packages

It is as simple as installing these two packages!

```
mkgendocs==0.9.0
mkdocs==1.2.3
```

Step 3— Download the starter code and add it to your project root

list of all classes and files we want to include within our documentation. We will automatically create both this mapping, as well the markdown file itself.

mkdocks.yaml — This file takes our documentation files (ex. docs/modules/main.md), and organizes them into site folders so that a sitemap can be created. This will also automatically be done by our script.

automate_mkdocs.py — This script, which is inspired by work from Louis de Brujin, but further extended and optimized for more situations by myself, fills in both the mkgendocs.yaml and mkdocs.yaml files. All files that are present and contain no syntax errors will be included. The style of the directory structure will mirror the way in which your package's folders are set up.

Take the time to fill in the top portions of mkgendocs.yaml and mkdocs.yaml to be unique to your project!

Step 4— Instantly create the documentation!

Run the following commands:

```
python automate_mkdocs.py
```

This will autofill the mkdocs.yaml and mkgendocs.yaml file.

```
gendocs --config mkgendocs.yml
```

This command takes our mkgendocs file, and generates our markdown files! Note: you




```
mkdocs serve
```

When you are happy with how it looks locally, we can deploy it to github with

```
mkdocs gh-deploy
```

It is that easy! A new branch was just created that holds your documentation and autogenerated sitemap! If you are happy with running these commands each time you change your documentation, it is fine to end here, however, there is one further optimization we can make, and that is setting up Github actions to update and deploy our documentation every time we push to our master (or any) branch!

Step 5—Using GitHub actions to auto-deploy documentation

If you have never used GitHub actions before, I recommend familiarizing yourself here first! and creating a base yaml file. From there, we will add commands to:

1. Set up our python version and checkout our code
2. install requirements and update mkdocs and mkgendocs
3. build or update our markdown files
4. deploy our site
5. commit any changes that may have occurred
6. push those changes to the associated branch

This may seem like a lot, but once it is done, it makes life much easier!

Note: your GitHub actions file will likely be similar but will differ a bit based on what version of python you are using, chosen themes, and if you have additional requirements like custom sites or GitHub push restrictions!

Set up our Python version and check out our code. This is fairly straightforward!




```
- name: Set up Python python-version
        uses: actions/setup-python@v1
        with:
          python-version: 3.7
```

Install requirements and update mkdocs and mkgendocs — the requirements.txt file should contain all files for your project, including the ones installed in step 2.

```
- name: Autogenerate new documentation
        continue-on-error: true
        run: |
          pip install -r requirements.txt
          python automate_mkdocs.py
          git add .
```

Build or update markdown files — I added some additional pip files for installation, although these can likely be excluded! I was just personally having issues with the jinja2 package.

```
- name: Update and Build GH Pages
        run: |
          python -m pip install --upgrade pip
          pip install mkdocs==1.2.3
          pip install mkgendocs==0.9.0
          pip install jinja2==2.11
          gendocs --config mkgendocs.yml
```

Deploy our site — This uses a really cool custom GitHub action, and has tons of optional parameters including setting up custom domains!

```
- name: deploy docs
        uses: mhausenblas/mkdocs-deploy-gh-pages@master
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          #CUSTOM_DOMAIN: optionaldomain.com
          CONFIG_FILE: mkdocs.yml
          EXTRA_PACKAGES: build-base
```




Commit any changes — we use “continue on error” so it does not fail if there are no changes!

```
- name: Commit any changes to docs
        continue-on-error: true
        run: |
          git config --local user.name "github-actions[bot]"
          git add ./docs
          git commit -m "Auto-updating the docs"
```

Push our changes — note: you can select which branch to push these changes to

```
- name: Push the changes to master
        continue-on-error: true
        uses: ad-m/github-push-action@master
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          branch: master
          force: true
```

And that is all! Once you iron out any deployment challenges, you now have a completely automated documentation workflow! If this really helped you, here is a link to the creator of mkgendocs’s donation page. I sent him a small donation to show my appreciation for creating that package!

I hope this article helped you in [illegible]rney! This article was one of my favorites to make. Be sure to sav[illegible] you think you will need to come back for reference! If you enjoyed this article, feel free to follow me, and read more of what I write, or use me as a referral so I can continue to make content I love.

The Best Python Sentiment Analysis Package (+1 Huge Common Mistake)

How to get near-perfect performance without training your own