Write the Docs Documentation

Release 1.0

Eric Holscher, Troy Howard

May 13, 2016

Contents

Welcome! We are excited that you are going down the path of creating more wonderful documentation in the world. This guide exists to provide both novice and expert writers a best practice handbook for building, structuring, and writing software documentation.

This is a living, breathing guide. If you'd like to contribute, fork us on GitHub! Also feel free to send us any additions in any format to guide@writethedocs.org.

Now, let's get started.

CHAPTER 1

Getting Started with Documentation

1.1 Starting off

So, you want to write the docs for your project?

> *Where on earth do I start?*

Well, you've come to the write [sic] place.

The goal here is to explain how to write docs, how to get the time to write docs, and what you should be striving for with those docs. A lot of projects are different, and a lot are the same. We will be covering the tactics for a few different groups.

1.1.1 Open Source Developers

You have some awesome open source code, but nobody knows how to use it. The chance of someone discovering and using your awesome code goes up greatly with good documentation.

Start with the *A beginners guide to writing documentation*, which provides practical advice on getting started.

1.1.2 Developers at a company

You need help justifying writing documentation for your project. It seems the timelines you're given hardly allow any time to write tests, and docs always get put off until the end. Well, here are some ways that you can show value to your boss, and hopefully get the time to write the docs.

Get started with *Building mindshare in a company*, which should give you a blueprint for how you can implement a documentation solution in your company.

1.1.3 Enterprise Users

You are a SAAS or Services company and you have developers as your target audience. If you don't have great documentation, your competitors will leave you in the dust. It will also jack up your support costs, because customers can't help themselves.

This page of *Interesting approaches to documentation* is a good starting point to see some documentation that is well done.

1.2 A beginners guide to writing documentation

Note: This is a write up of a *Presentation*. Please provide feedback to @ericholscher. You can view the source on GitHub.

> Camera pans from stage left.
>
> It shows a text editor, open to a blank page.
>
> A person hunched in front, head to desk.

The scene above is well known to everyone who writes for a living; the mixed emotions of a blank page. Full of excitement, fresh with a new beginning. Yet also full of despair, where do you even start?

I am here to stop this scene from playing out.

This is a guide to documenting your first project. The first time is always the hardest, and I hope this guide will get you started down the righteous path. At the end, you should have a project that is ready for public release.

Feel free to read this document straight through, or simply use it as a reference.

1.2.1 Why write docs

You will be using your code in 6 months

I find it best to start off with a selfish appeal. The best reason to write docs is because you will be using your code in 6 months. Code that you wrote 6 months ago is often indistinguishable from code that someone else has written. You will look upon a file with a fond sense of remembrance. Then a sneaking feeling of foreboding, knowing that someone less experienced, less wise, had written it.

As you go through this selfless act of untangling things that were obvious or clever months ago, you will start to empathize with your users. If only I had written down why I had done this. Life would be so much simpler. Documentation allows you to transfer the *why* behind code. Much in the same way code comments explain the *why*, and not the *how*, documentation serves the same purpose.

Sidebar on open source

There is a magical feeling that happens when you release your code. It comes in a variety of ways, but it always hits you the same. *Someone is using my code?!* A mix of terror and excitement.

> I made something of value!
>
> What if it breaks?!
>
> I am a real open source developer!
>
> Oh god, someone else is using my code..

Writing good documentation will help alleviate some of these fears. A lot of this fear comes from putting something into the world. My favorite quote about this is something along these lines:

> Fear is what happens when you're doing something important.
>
> If you are doing work that isn't scary,
>
> it isn't improving you or the world.

Congrats on being afraid! It means you're doing something important.

You want people to use your code

You have written a piece of code, and released it into the world. You have done this because you think that others might find it useful. However, people need to understand why your code might be useful for them, before they decide to use it. Documentation tells people that this project is for them.

> If people don't know why your project exists,
> they won't use it.
> If people can't figure out how to install your code,
> they won't use it.
> If people can't figure out how to use your code,
> they won't use it.

There are a small number of people who will source dive and use any code out there. That is a vanishingly small number of people, compared to people who will use your code when properly documented. If you really love your project, document it, and let other people use it.

You want people to help out

Open source is this magical thing right? You release code, and the code gnomes come and make it better for you.

Not quite.

There are lots of ways that open source is amazing, but it doesn't exist outside the laws of physics. You have to put work in, to get work out.

> You only get contributions after you have put in a lot of work.
> You only get contributions after you have users.
> You only get contributions after you have documentation.

Documentation also provides a platform for your first contributions. A lot of people have never contributed before, and documentation changes are a lot less scary than code changes. If you don't have documentation, you will miss out on a whole class of contributors.

It makes your code better

There is an old truth that writing things down helps you think. It's really easy to have an idea in your head that sounds perfect, but the act of putting words to paper requires a distillation of thought.

Writing documentation improves the design of your code. Talking through your API and design decisions on paper allows you to think about them in a more formalized way. A nice side effect is that it allows people to contribute code that follows your original intentions as well.

You want to be a better writer

Writing documentation is a different form of writing than most people have experience with. Technical writing is an art that doesn't come naturally. Writing documentation will start you down the road to being a better technical writer, which is a useful skill to have as a programmer.

Writing also becomes easier over time. If you don't write for many months, it is a lot harder to start writing again. Keeping your projects documented will keep you writing at a reasonable cadence.

1.2.2 What technology

Starting simple is the best way to achieve actual results. I will present a well-paved path to walk down, and after you have the basic idea, you can expand your scope. The tools should be powerful and easy to use. This removes obstacles to actually putting words on the page.

Sidebar on markup languages.

The examples in this document are both valid Markdown and reStructuredText. reStructuredText is a bit harder to use, but is more powerful. I recommend that you check them both out, and decide which you want to use going forward.

Version controlled plain text

As programmers we live in a world of plain text. Our documentation tooling should be no exception. We want tools that turn plain text into pretty HTML. We also have some of the best tooling available for tracking changes to files. Why would we forgo using those tools when writing documentation? This workflow is powerful, and familiar to developers.

Basic Example

```
Resources
---------

* Online documentation: http://docs.writethedocs.org/
* Conference: http://conf.writethedocs.org/
```

This will render into a header, with a list underneath it. The URLs will be hyperlinked automatically. It's easy to write, still makes sense as plain text, and renders nicely into HTML.

README

Your first steps in documentation should go into your README. Code hosting services will render your README into HTML automatically if you provide the proper extension. It is also the first interaction that most users will have with your project. So having a solid README will serve your project well.

Some people even go as far as to start your project with a README

1.2.3 What to write

Now we're getting down to the brass tacks. Making sure that you give your users all the information that they need, but not too much.

First, you need to ask yourself who you're writing for. At first, you generally just need to appeal to two audiences:

- Users
- Developers

Users are people who simply want to use your code, and don't care how it works. Developers are people who want to contribute back to your code.

What problem your project solves

A lot of people will come to your docs trying to figure out what exactly your project is. Someone will mention it, or they'll google a phrase randomly. You should explain what your project does and why it exists. Fabric does a great job of this.

A small code example

Show a telling example of what your project would normally be used for. Requests does a great example of this.

A link to your code & issue tracker

People like to browse the code sometimes. They might be interested in filing bugs against the code for issues they've found. Make it really easy for people who want to contribute back to the project in any way possible. I think the Python Guide does a good job with the link to the code portion.

Frequently Asked Questions (FAQ)

A lot of people have the same problems. If things happen all the time, you should probably fix your documentation or the code, so that the problems go away. However, there are always questions that get asked about your project, things that can't be changed, etc. Document those, and **keep it up to date**. FAQs are generally out of date, but when done well, they are a golden resource. Tastypie did a great job with this, with their "Cookbook" concept.

How to get support

Mailing list? IRC Channel? Document how to get help and interact with the community around a project. Django does a great job with this.

Information for people who want to contribute back

People usually have standards for how they expect things to be done in their projects. You should document these so that if people write code, they can do things in the norm of the project. Open Comparison does a great job of this.

Installation instructions

Once people figure out whether they want to use your code or not, they need to know how to actually get it and make it run. Hopefully your install instructions should be a couple lines for the basic case. A page that gives more information and caveats should be linked from here if necessary. I think at Read the Docs we do a good job with this.

Your project's license

BSD? MIT? GPL? This stuff might not matter to you, but the people who want to use your code will care about this a whole lot. Think about what you want to accomplish with your license, and please only pick one of the standard licenses that you see around the web.

1.2.4 Next Steps

After you follow the above guide, we know your project will be successful! For further reading, check out this post on how to maintain an open source project.

1.2.5 Template

A simple template for you to start with, for your `README`. Name the file `README.md` if you want to use markdown, or `README.rst` if you want to use reStructuredText. More information about these can be found in the *sidebar on markup*.

```
$project
========

$project will solve your problem of where to start with documentation,
by providing a basic explanation of how to do it easily.

Look how easy it is to use:

    import project
    # Get your stuff done
    project.do_stuff()

Features
--------

- Be awesome
- Make things faster

Installation
------------

Install $project by running:

    install project

Contribute
----------

- Issue Tracker: github.com/$project/$project/issues
- Source Code: github.com/$project/$project

Support
-------

If you are having issues, please let us know.
We have a mailing list located at: project@google-groups.com

License
-------

The project is licensed under the BSD license.
```

CHAPTER 2

Writing Great Documentation

- Structuring your Documentation
- Style Guides
- *Interesting approaches to documentation*

CHAPTER 3

Documentation Culture at your Company

3.1 Building mindshare in a company

Having a culture of documentation inside of a company is a great thing. Building a culture around documentation however is a hard thing to do. This will be a guide with some information and suggestions for how to go about bringing good docs into a company.

3.1.1 State the problem

There are a lot of nebulous problems in communication inside of organizations. You want to start small with something that you can nail. When I've done this in the past, it looks something like this:

> We will solve the issue of documentation around internal project documentation in our Engineering organization.

Which brings me to the next point.

3.1.2 Start in engineering

Starting in the engineering organization I think is the simplest way to affect change on documentation culture. There are lots of other parts of an organization, but it's easier to get permission to work on just the internal stuff to engineering. Think things like:

- Project documentation
- **Operations documentation**
  - Playbooks
  - Monitoring Information
- Best Practices
- Style Guides

3.1.3 Gather Input

You will want members from all of the teams in your organization to be invited to the planning meetings. They don't need to come, but you need to make sure it's an open and public discussion. This validates the outcome of the planning, and makes sure that you are taking into account all of the use cases of your team.

3.1.4 Build a taxonomy

A big problem with a lot of documentation systems is that they have been organized organically, AKA have no organization. If you build out a structure for people to start, and keep it consistent across projects, it will make everyones lives easier.

Note: You need to have an answer to the "Where do I put it?" question.

3.1.5 Make it easy

It should be as easy and straight-forward getting started as possible. Most people don't like change, and introducing new tooling and things will already rouse a negative response in some. This is why you need to make it as simple as possible to get started.

Build templates

It should take about 5 seconds to get a basic outline of the documentation for a project started.

Also have a standard hierarchy for the docs, so that people know where to look for things in any project.

3.1.6 Have regular meetings throughout the process

Having a weekly standing meeting where you keep track of the progress along all the stages is important. This gives people a known place to go and discuss issues or ideas. It also provides a sense that the project is moving forward. At the beginning these meetings will be used to plan and track implementation. After implementation, it will be a place to drive adoption and gather feedback.

Make it long term

Something like a good documentation system also needs pretty constant care and feeding, reorganizing, and other maintained work. If you view this project as getting the tools in place, without a long term commitment, it will fail just like your last system.

- Integrating Tech Writers into the product process
- Producing documentation inside a Support team

CHAPTER 4

Inspiration

It's always good to see the amazing things other folks have done. This gets us fired up. Check out these resources that we get inspired by:

- https://github.com/RichardLitt/awesome-docs
- Your link here

CHAPTER 5

Markup Languages

- **Markdown**
  - Cheatsheet
  - Tutorial
- **Asciidoc**
  - Cheatsheet
  - Tutorial
- **reStructuredText**
  - Cheatsheet
  - Tutorial

CHAPTER 6

Tools of the Trade

6.1 Tools for documentation writing

Writing documentation requires good tools. There are a bunch of different tools in the world, but we figure we should give you the sharpest. Below we talk about some of our favorites and the ones we find do the job the best.

6.1.1 Introduction to Sphinx

Philosophy

Sphinx is what is called a documentation generator. This means that it takes a bunch of source files in plain text, and generates a bunch of other awesome things, mainly HTML. For our use case you can think of it as a program that takes in plain text files in reStructuredText format, and outputs HTML.

```
reST -> Sphinx -> HTML
```

So as a user of Sphinx, your main job will be writing these text files. This means that you should be minimally familiar with reStructuredText as a language. It's similar to Markdown in a lot of ways. It's a lot more powerful than Markdown, but with that power comes increased complexity. Just know that some of the awkward syntax allows you to do more interesting things further down the line. In particular, it is extensible: it has a formal way of adding markup directives that allow more sophisticated parsing. For example, Sphinx includes directives to relate documentation of modules, classes and methods to the corresponding code.

Installing Sphinx

The first step to getting going is installing Sphinx. Sphinx is a python project, so it can be installed like any other python library. Several Operating Systems (Mac OS X, Major Versions of Linux/BSD) have Python pre-installed, so you should just have to run:

```
sudo easy_install Sphinx
```

Instructions for installing Python and Sphinx on Windows can be found at the Sphinx install page.

Note: Advanced users can install this in a virtualenv if they wish. Also, `pip install Sphinx` works fine if you have Pip.

Getting Started

You'll want to read the Sphinx Tutorial, as it provides an introduction to a lot of the basic ideas. For the most part documentation follows a standard structure for our documentation repository:

```
project/
    docs/
        conf.py
        index.rst
        Makefile
```

We have a top-level `docs` directory in the main project directory. Inside of this is:

`index.rst`: This is the index file for the documentation, or what lives at `/`. It normally contains a *Table of Contents* that will link to all other pages of the documentation.

`conf.py`: which allows for customization of the output. For the most part this shouldn't need to be changed.

`Makefile`: This ships with sphinx, and is the main interface for local development, but shouldn't be changed.

Other `*.rst` files for specific subsections of documentation.

Writing docs

Where you write your documentation will vary based on how the project is layed out. Generally major topics will go in an aptly named file in the top-level docs directory. If a topic gets larger, it can then be broken out into multiple files in a directory. When you write a document, figure out if there is already a place for it in the project, otherwise feel free to start a new file.

> **Warning:** If you make a new file, make sure it is included in the *Table of Contents* in `index.rst`.

reStructuredText

To write nice looking documentation you will need to have a basic understanding of RST as a language. The reStructuredText Primer is a great place to start reading, and it covers most of the syntax you will care about. The main parts you will need at first are:

- **Inline Markup**
- **Source Code**
- **Hyperlinks**
- **Sections**
- **Directives**

Note: You can live-preview RST on the web: http://rst.ninjs.org/ . Note that it won't understand Sphinx-specific markup though.

Feel free to play around with RST a bit to make sure that you understand how it works.

> **Warning:** RST is white-space sensitive in places. If it is acting weirdly, make sure you indent lines that are part of the same content similarly.

Building docs

Once you have your documentation written and want to turn it into HTML, it's pretty simple. Simply run:

```
# Inside top-level docs/ directory.
make html
```

This should run Sphinx in your shell, and output HTML. At the end, it should say something about the documents being ready in `_build/html`. You can now open them in your browser by typing:

```
open _build/html/index.html
```

6.1.2 Sphinx Themes

These are the Sphinx themes that we recommend. If there are any others you like, feel free to open a pull request to add them.

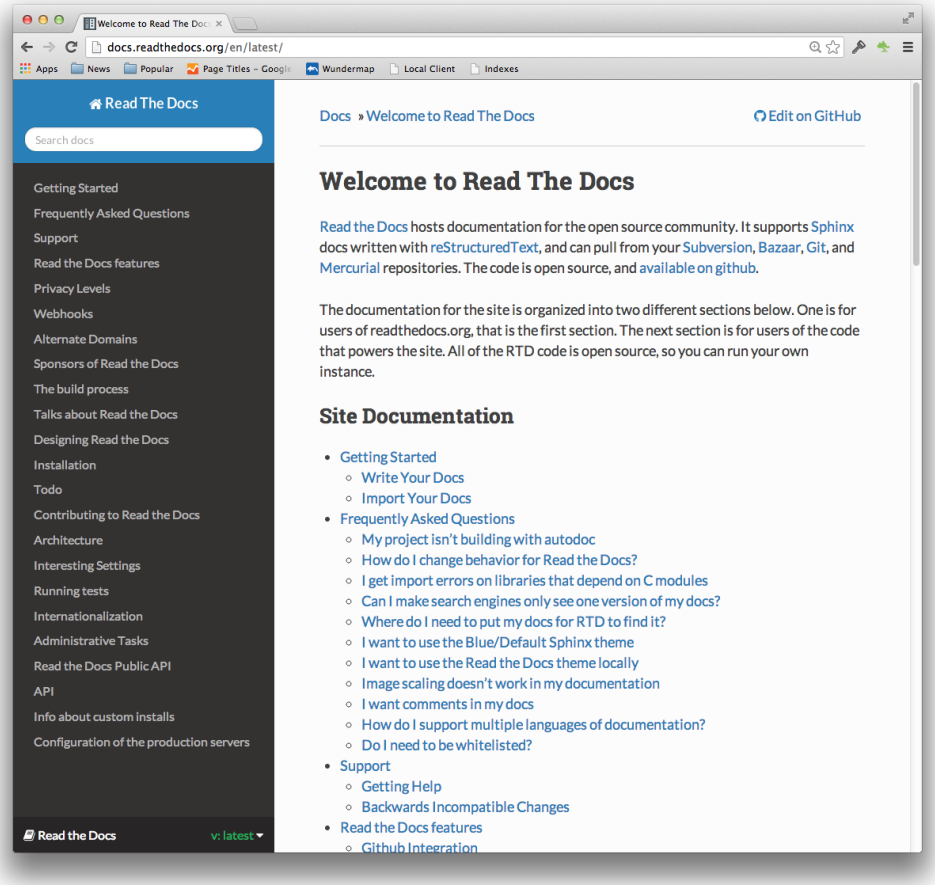
Requirements to be included on this list:

- Mobile Ready
- Nice fonts and typography
- Installable as a Python module
- Maintained and documented

Read the Docs Theme

The official theme for Read the Docs. It features beautiful typography and a nice blue color scheme. It looks great on mobile, and provides a menu of all the pages on the left-hand side.

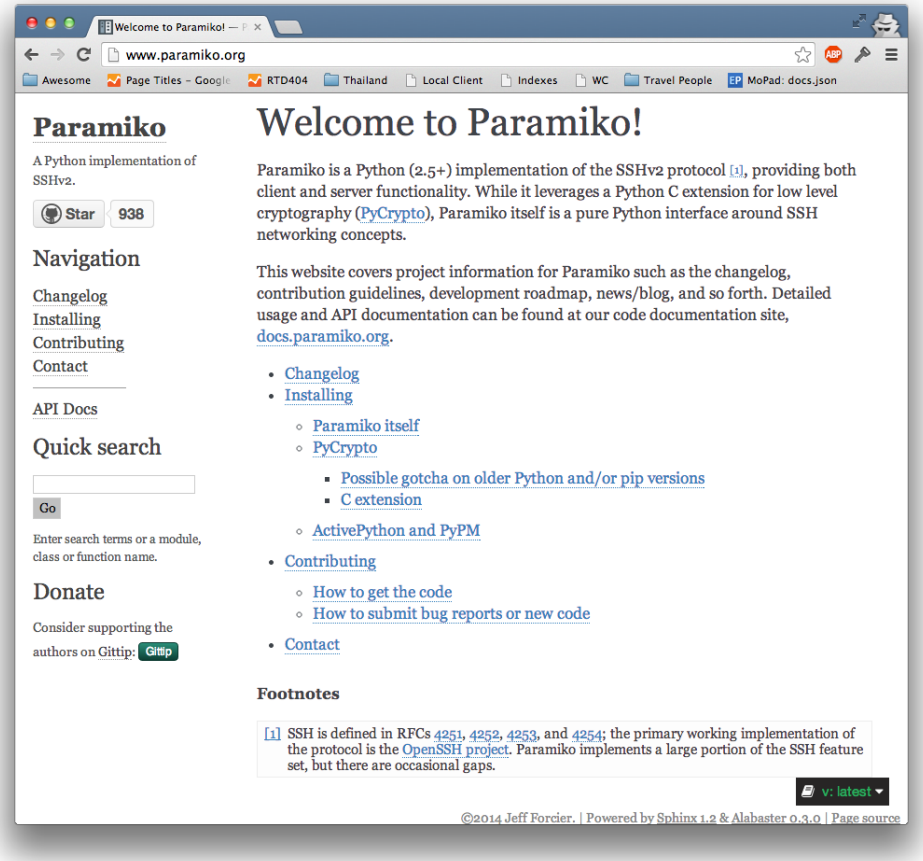
- https://github.com/snide/sphinx_rtd_theme



Alabaster

Based off the original Flask and KR themes, this is a more extensible version of the prior. It is what this site uses, and provides very minimal markup. It's great for text content where you just want to make the words front and center.

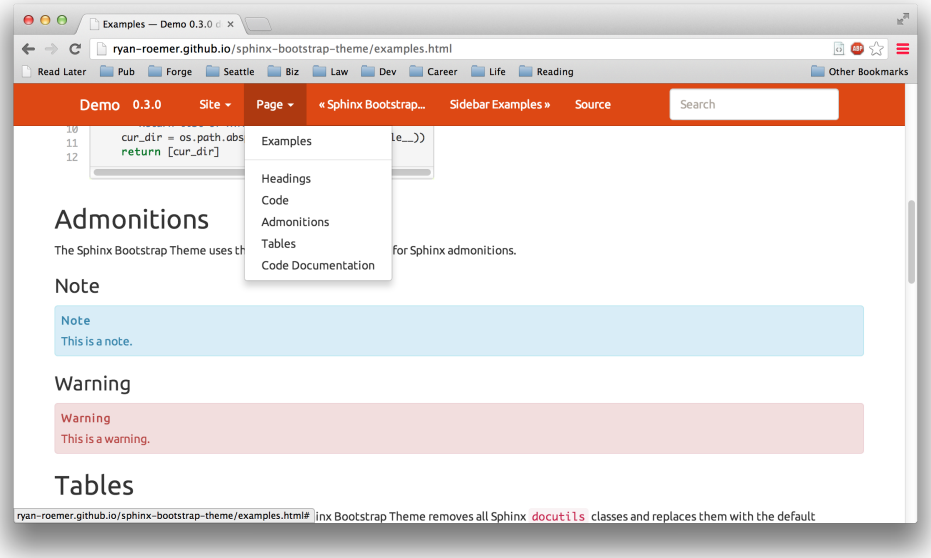
- https://github.com/bitprophet/alabaster



Sphinx Bootstrap Theme

A basic Sphinx theme that uses Bootstrap for nice styling. It is a great start for any site that uses Bootstrap, or just wants a simple good looking theme.

- https://github.com/ryan-roemer/sphinx-bootstrap-theme



- Keeping your documentation up to date
- **Automated testing**
  - Link checking
  - Code Testing
- Sphinx
- Jekyll

CHAPTER 7

API Documentation

A section dedicated to all things API. Whether it's a Python or a REST API, you need to be documenting your code.

- **REST API Tools**
  - Swagger
- **API Documentation**
  - Sphinx
  - Automated reference docs

CHAPTER 8

Distribution

- Common Output formats
- **Documentation Styles**
  - Slate
  - Read the Docs Theme
  - Alabaster

CHAPTER 9

Writing Environments

- Text Editors
- IDE's
- Live Previews

CHAPTER 10

Talks & slide decks

10.1 Presentations

These presentations are available for anyone to give at a local user group or conference. We hope that you can use them to evangalize the idea of writing documentation.

If you have feedback or want to make improvements, please submit them to our repository on GitHub: https://github.com/writethedocs/docs/issues

10.1.1 Theme

We ship a theme that allows your presentation to have the Write the Docs theme. They require a set of fonts that you'll need as well.

- `Keynote Master Slides`
- Fonts

10.1.2 Beginner Presentations

- **Writing Docs: A beginners guide to writing documentation**
  - `Keynote`
  - `PDF`
  - `Zipped HTML`

10.1.3 Intermediate Presentations

Note: Coming soon

Outline:

- IA/Structure
- Things to include

CHAPTER 11

Additional Notes

11.1 About Write the Docs

Here are information about the Write the Docs project itself.

11.1.1 Vision

There exists a tribe of documentarians in the world. Up until this point, they haven't had a central place to meet each other, and coalesce into a community. We are providing the space to allow this to happen, both in person and online.

The time for all technical people to care about documentation is now. We've lived too long with awful instructions for the tools we use everyday. People should demand solid instruction of things that they use.

Writers

Documentation is how you share your creations with the world. If you want people to benefit from your work, they have to be able to use it. Help me, help you. Do something and change something.

We believe it should be easy for people to start writing documentation. There should be straight-forward guides to getting started with good tools.

Students

People who want to learn should be given the best possible tools for this job. We want to raise the standard of documentation to make it easier for people to learn about the things they want to do.

We believe that there should be best practices around documentation. They should be simple, concise and easy to follow practical guidelines.

We believe that knowledge should be available to all people, regardless of their language of choice. Documentation should be written in a way that allows it to be translated easily

11.1.2 Interesting approaches to documentation

- Riak does a side by side comparison of the Dynamo paper and their implementation: http://docs.basho.com/riak/1.2.0/references/dynamo/
- Live notes of conferences as they happen: http://pydanny-event-notes.readthedocs.org/en/latest/

- Teaching / Books:
  - http://ericholscher.com/blog/2012/dec/1/interesting-projects-read-docs-teaching/
- This GitHub repo has some great examples of documentation done well:
  - https://github.com/PharkMillups/beautiful-docs

11.1.3 License

11.1.4 Documentation Community

We are bringing together a community around documentation. Communities need a Third Place to gather, and we hope to be that place. We currently have an IRC channel and mailing list for people to connect on:

- #writethedocs on Freenode IRC
- Write the Docs Mailing List
- Write the Docs on Twitter
- Write the Docs on GitHub

Contributing

A community without people who help maintain standards and advance the state of the art isn't worth having. Write the Docs on GitHub is the place to contribute to this site and other parts of the community. If you have any wild, crazy, mundance, or old-hat ideas, we'd love to consider and appreciate them.

11.1.5 Authors and contributors

Primary authors

- Eric Holscher <eric@ericholscher.com>

Contributors

By alphabetical order...

- Benoît Bryon <benoit@marmelune.net>
- Chris McDonald <xwraithanx@gmail.com>
- Eric Holscher <eric@ericholscher.com>
- Rich Morin <rdm@cfcl.com>
- Troy Howard <thoward37@gmail.com>

CHAPTER 12

Write the Docs Resources

- Online documentation: http://docs.writethedocs.org/
- Conference: http://www.writethedocs.org/
- Slack: Write the Docs
- Twitter: http://twitter.com/writethedocs
- Issues & feature requests: https://github.com/writethedocs/docs/issues
- Source repository: https://github.com/writethedocs/docs