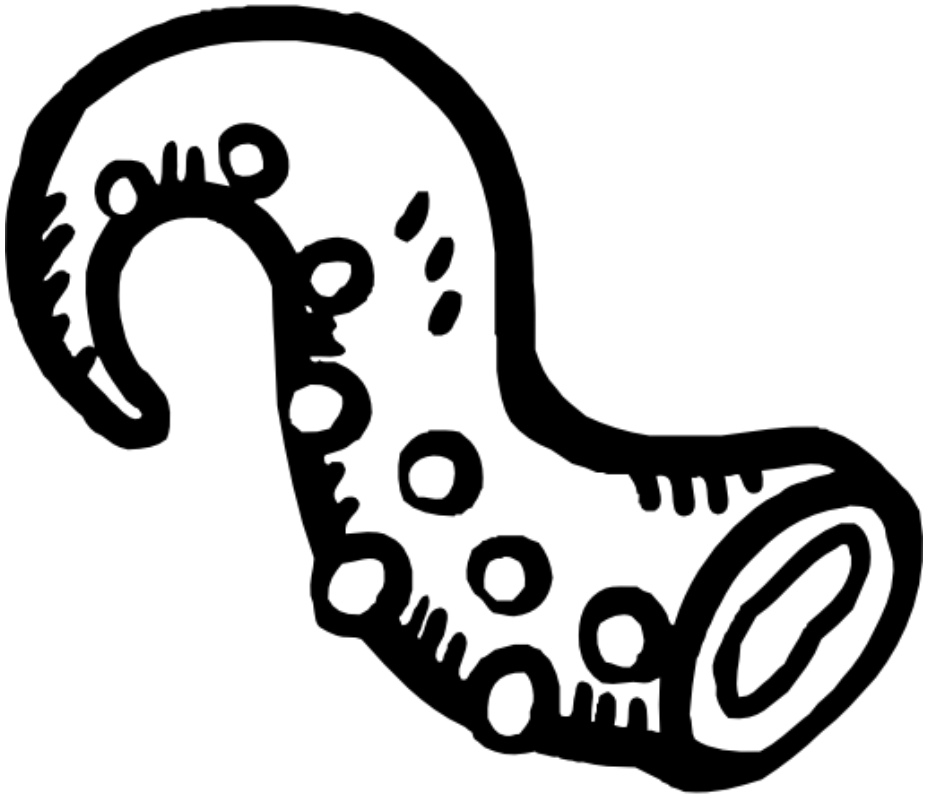
Foliant

Konstantin Molchanov

January 7, 2018



1 Welcome to Foliant!

Foliant is a all-in-one documentation authoring tool. It lets you produce standalone documents in pdf and docx, as well as websites, from single Markdown source.

Foliant is a higher order tool, which means it uses other programs to do its job. For pdf and docx, it uses Pandoc, for websites it uses MkDocs.

Foliant preprocessors let you include parts of documents in other documents, show and hide content with flags, render diagrams from text, and much more.

Logo made by Hand Drawn Goods from www.flaticon.com.

2 Installation

Installing Foliant to your system can be split into three stages: installing Python with your system's package manager, installing Foliant with pip, and optionally installing Pandoc and TeXLive bundle. Below you'll find the instructions for three popular platforms: macOS, Windows, and Ubuntu.

Alternatively, you can avoid installing Foliant and its dependencies on your system by using Docker and the official Foliant Docker images. If that's the path you want to follow, skip straight to the Docker instructions.

2.1 macOS

1. Install Python 3.6 with Homebrew:

    ```
    $ brew install python3
    ```

2. Install Foliant with pip:

    ```
    $ pip3 install foliant
    ```

3. If you plan to bake pdf or docx, install Pandoc and MacTeX with Homebrew:

    ```
    $ brew install pandoc mactex librsvg
    ```

2.2 Windows

0. Install Scoop package manager in PowerShell:

    ```
    $ iex (new-object net.webclient).downloadstring('https://get.scoop.sh')
    ```

1. Install Python 3.6 with Scoop:

    ```
    $ scoop install python
    ```

2. Install Foliant with pip:

```
$ pip install foliant
```

3. If you plan to bake pdf or docx, install Pandoc and MikTeX with Scoop:

```
$ scoop install pandoc latex
```

2.3 Ubuntu

1. Install Python 3.6 with apt. On 14.04 and 16.04:

```
$ add-apt-repository ppa:jonathonf/python-3.6
$ apt update && apt install -y python3 python3-pip
```

   On newer versions:

```
$ apt update && apt install -y python3 python3-pip
```

2. Install Foliant with pip:

```
$ pip3 install foliant
```

3. If you plan to bake pdf or docx, install Pandoc and TeXLive with apt and wget:

```
$ apt update && apt install -y wget texlive-full librsvg2-bin
$ wget https://github.com/jgm/pandoc/releases/download/2.0.5/pandoc-2.0.5
```

3 Docker

1. Install Docker for your platform.
2. Create a Dockerfile in your project directory (if you use `foliant init`, the Dockerfile is created automatically):

```
FROM foliant/foliant
```

   If you plan to bake pdf or docx, use the image with Pandoc and TeXLive:

```
FROM foliant/foliant:pandoc
```

   Pandoc and TeXLive are not included in the default image because they add about 5 GB to the image size. For situations where you're interesing in site generation only, this would add way too much overhead.

3. Build the image for your project:

```
$ docker build -t my-project .
```

4. Run Foliant in Docker:

```
$ docker run --rm -it -v (pwd):/usr/src/app -w /usr/src/app my-project n
```

4 Backends

4.1 MkDocs

MkDocs backend lets you build websites from Foliant projects using MkDocs static site generator.

The backend adds three targets: mkdocs, site, and ghp. The first one converts a Foliant project into a MkDocs project without building any html files. The second one builds a standalone website. The last one deploys the website to GitHub Pages.

4.1.1 Installation

```
$ pip install foliantcontrib.mkdocs
```

4.1.2 Usage

Convert Foliant project to MkDocs:

```
$ foliant make mkdocs -p my-project
 Parsing config
 Applying preprocessor mkdocs
 Making mkdocs with MkDocs

Result: My_Project-2017-12-04.mkdocs.src
```

Build a standalone website:

```
$ foliant make site -p my-project
 Parsing config
 Applying preprocessor mkdocs
 Making site with MkDocs

Result: My_Project-2017-12-04.mkdocs
```

Deploy to GitHub Pages:

```
$ foliant make ghp -p my-project
 Parsing config
 Applying preprocessor mkdocs
 Making ghp with MkDocs

Result: https://account-name.github.io/my-project/
```

4.1.3 Config

You don't have to put anything in the config to use MkDocs backend. If it's installed, Foliant detects it.

To customize the output, use options in backend_config.mkdocs section:

```
backend_config:
  mkdocs:
    mkdocs_path: mkdocs
    use_title: true
    use_chapters: true
    mkdocs.yml:
      site_name: Custom Title
      site_url: http://example.com
      site_author: John Smith
```

`mkdocs_path` Path to the MkDocs executable. By default, mkdocs command is run, which implies it's somewhere in your PATH.

`use_title` If true, use title value from foliant .yml as site_name in mkdocs.yml. It this case, you don't have to specify site_name in mkdocs.yml section. If you do, the value from mkdocs.yml section has higher priority.

If false, you *must* specify site_name manually, otherwise MkDocs will not be able to build the site.

Default is true.

`use_chapters` Similar to use_title, but for pages. If true, chapters value from foliant .yml is used as pages in mkdocs.yml.

`mkdocs.yml` Params to be copied into mkdocs.yml file. The params are passed "as is," so you should consult with the MkDocs configuration docs.

4.1.4 Preprocessor

MkDocs backend ships with a preprocessor that transforms a Foliant project into a MkDocs one. Basically, foliant make mkdocs just applies the preprocessor.

The preprocessor is invoked automatically when you run MkDocs backend, so you don't have to add it in preprocessors section manually.

However, it's just a regular preprocessor like any other, so you can call it manually if necessary:

```
preprocessors:
  - mkdocs:
      mkdocs_project_dir_name: mkdocs
```

`mkdocs_project_dir_name` Name of the directory for the generated MkDocs project within the tmp directory.

4.1.5 Troubleshooting

Fenced Code Is Not Rendered in List Items or Blockquotes MkDocs can't handle fenced code blocks in blockquotes or list items due to an issue in Python Markdown.

Unfortunately, nothing can be done about it, either on MkDocs's or Foliant's part. As a workaround, use indented code blocks.

Paragraphs Inside List Items Are Rendered on the Root Level Check if you use **four-space indentation**. Python Markdown is stern about this point.

4.2 Pandoc

Pandoc is a Swiss-army knife document converter. It converts almost any format to any other format: md to pdf, rst to html, adoc to docx, and so on and so on.

Pandoc backend for Foliant add pdf, docx, and tex targets.

4.2.1 Installation

```
$ pip install foliantcontrib.pandoc
```

You also need to install Pandoc and TeXLive distribution for your platform.

4.2.2 Usage

Build pdf:

```
$ foliant make pdf -p my-project
 Parsing config
 Applying preprocessor flatten
 Making pdf with Pandoc

Result: My_Project-2017-12-04.pdf
```

Build docx:

```
$ foliant make docx -p my-project
 Parsing config
 Applying preprocessor flatten
 Making docx with Pandoc

Result: My_Project-2017-12-04.docx
```

Build tex (mostly for pdf debugging):

```
$ foliant make tex -p my-project
 Parsing config
 Applying preprocessor flatten
 Making docx with Pandoc
```

```
Result: My_Project-2017-12-04.tex
```

4.2.3 Config

You don't have to put anything in the config to use Pandoc backend. If it's installed, Foliant will detect it.

You can however customize the backend with options in backend_config.pandoc section:

```
backend_config:
  pandoc:
    pandoc_path: pandoc
    template: !path template.tex
    vars:
      ...
    reference_docx: !path reference.docx
    params:
      ...
    filters:
      ...
    markdown_flavor: markdown
    markdown_extensions:
      ...
```

`pandoc_path` is the path to pandoc executable. By default, it's assumed to be in the PATH.

`template` is the path to the TeX template to use when building pdf and tex (see "Templates" in the Pandoc documentation).

> **Tip**
>
> Use !path tag to ensure the value is converted into a valid path relative to the project directory.

`vars` is a mapping of template variables and their values.

`reference_docx` is the path to the reference document to used when building docx (see "Templates" in the Pandoc documentation).

> **Tip**
>
> Use !path tag to ensure the value is converted into a valid path relative to the project directory.

`params` are passed to the pandoc command. Params should be defined by their long names, with dashes replaced with underscores (e.g. --pdf-engine is defined as pdf_engine).

`filters` is a list of Pandoc filters to be applied during build.

`markdown_flavor` is the Markdown flavor assumed in the source. Default is markdown, Pandoc's extended Markdown. See "Markdown Variants" in the Pandoc documentation.

`markdown_extensions` is a list of Markdown extensions applied to the Markdown source. See Pandoc's Markdown in the Pandoc documentation.

Example config:

```
backend_config:
  pandoc:
    template: !path templates/basic.tex
    vars:
      toc: true
      title: This Is a Title
      second_title: This Is a Subtitle
      logo: !path templates/logo.png
      year: 2017
    params:
      pdf_engine: xelatex
      listings: true
      number_sections: true
    markdown_extensions:
      - simple_tables
      - fenced_code_blocks
      - strikeout
```

4.2.4 Troubleshooting

Could not convert image ...: check that rsvg2pdf is in path In order to use svg images in pdf, you need to have rsvg−convert executable in PATH.

On macOS, brew install librsvg does the trick. On Ubuntu, apt install librsvg2−bin. On Windows, download rsvg−convert.7z (without fontconfig support), unpack rsvg−convert.exe, and put it anywhere in PATH. For example, you can put it in the same directory where you run foliant make.

5 Preprocessors

5.1 Blockdiag

Blockdiag is a tool to generate diagrams from plain text. This preprocessor finds diagram definitions in the source and converts them into images on the fly during project build. It supports all Blockdiag flavors: blockdiag, seqdiag, actdiag, and nwdiag.

5.1.1 Installation

```
$ pip install foliantcontrib.blockdiag
```

5.1.2 Config

To enable the preprocessor, add blockdiag to preprocessors section in the project config:

```
preprocessors:
  - blockdiag
```

The preprocessor has a number of options:

```
preprocessors:
  - blockdiag:
      cache_dir: !path .diagramscache
      blockdiag_path: blockdiag
      seqdiag_path: seqdiag
      actdiag_path: actdiag
      nwdiag_path: nwdiag
      params:
        ...
```

`cache_dir` Path to the directory with the generated diagrams. It can be a path relative to the project root or a global one; you can use ~/ shortcut.

> **Note**
>
> To save time during build, only new and modified diagrams are rendered. The generated images are cached and reused in future builds.

`*_path` Paths to the blockdiag, seqdiag, actdiag, and nwdiag binaries. By default, it is assumed that you have these commands in PATH, but if they're installed in a custom place, you can define it here.

`params` Params passed to the image generation commands (blockdiag, seqdiag, etc.). Params should be defined by their long names, with dashes replaced with underscores (e.g. −−no−transparency becomes no_transparency); also, −T param is called format for readability:

```
preprocessors:
  - blockdiag:
      params:
        antialias: true
        font: !path Anonymous_pro.ttf
```

To see the full list of params, run blockdiag −h.

5.1.3 Usage

To insert a diagram definition in your Markdown source, enclose it between <blockdiag>...</blockdiag>, <seqdiag>...</seqdiag>, <actdiag>...</actdiag>, or <nwdiag>...</nwdiag> tags (indentation inside tags is optional):

```
Here's a block diagram:

<blockdiag>
  blockdiag {
    A -> B -> C -> D;
    A -> E -> F -> G;
  }
</blockdiag>

Here's a sequence diagram:

<seqdiag>
  seqdiag {
    browser  -> webserver [label = "GET /index.html"];
    browser <-- webserver;
    browser  -> webserver [label = "POST /blog/comment"];
                webserver  -> database [label = "INSERT comment"];
                webserver <-- database;
    browser <-- webserver;
  }
</seqdiag>
```

To set a caption, use caption option:

```
Diagram with a caption:

<blockdiag caption="Sample diagram from the official site">
  blockdiag {
    A -> B -> C -> D;
    A -> E -> F -> G;
  }
</blockdiag>
```

You can override params values from the preprocessor config for each diagram:

```
By default, diagrams are in png. But this diagram is in svg:

<blockdiag caption="High-quality diagram" format="svg">
  blockdiag {
    A -> B -> C -> D;
    A -> E -> F -> G;
  }
```

```
</blockdiag>
```

5.2 Flags

This preprocessors lets you exclude parts of the source based on flags defined in the project config and environment variables, as well as current target and backend.

5.2.1 Installation

```
$ pip install foliantcontrib.flags
```

5.2.2 Config

Enable the propressor by adding it to preprocessors:

```
preprocessors:
  - flags
```

Enabled project flags are listed in preprocessors.flags:

```
preprocessors:
  - flags:
    - foo
    - bar
```

To set flags for the current session, define FOLIANT_FLAGS environment variable:

```
$ FOLIANT_FLAGS="spam, eggs"
```

You can use commas, semicolons, or spaces to separate flags.

Hint

To emulate a particular target or backend with a flag, use the special flags target:FLAG and backend:FLAG where FLAG is your target or backend: $ FOLIANT_FLAGS="target:pdf, backend:pandoc, spam"

5.2.3 Usage

Conditional blocks are enclosed between <if>...</if> tags:

```
This paragraph is for everyone.

<if flags="management">
This parapraph is for management only.
</if>
```

A block can depend on multiple flags. You can pick whether all tags must be present for the block to appear, or any of them (by default, kind="all" is assumed):

```
<if flags="spam, eggs" kind="all">
This is included only if both ‘spam‘ and ‘eggs‘ are set.
</if>

<if flags="spam, eggs" kind="any">
This is included if both ‘spam‘ or ‘eggs‘ is set.
</if>
```

You can also list flags that must *not* be set for the block to be included:

```
<if flags="spam, eggs" kind="none">
This is included only if neither ‘spam‘ nor ‘eggs‘ are set.
</if>
```

You can check against the current target and backend instead of manually defined flags:

```
<if targets="pdf">This is for pdf output</if><if targets="site">This is for

<if backends="mkdocs">This is only for MkDocs.</if>
```

5.3 Flatten

Project Flattener for Foliant

This preprocessor converts a Foliant project source directory into a single Markdown file containing all the sources, preserving order and inheritance.

This preprocessor is used by backends that require a single Markdown file as input instead of a directory. The Pandoc backend is one such example.

```
$ pip install foliantcontrib.flatten
```

Config This preprocessor is required by Pandoc backend, so if you use it, you don't need to install Flatten or enable it in the project config manually.

However, it's still a regular preprocessor, and you can run it manually by listing it in preprocessors:

```
preprocessors:
  - flatten
```

The preprocessor has only one option—flat_src_file_name. It's the name of the flattened file that is created in the tmp directory:

```
preprocessors:
  - flatten:
      flat_src_file_name: flattened.md
```

Default value is ___all___.md.

Note

Flatten preprocessor uses includes, so when you install Pandoc backend, Includes preprocessor will also be installed, along with Flatten.

5.4 Includes

Includes preprocessor lets you reuse parts of other documents in your Foliant project sources. It can include from files on your local machine and remote Git repositories. You can include entire documents as well as parts between particular headings, removing or normalizing included headings on the way.

5.4.1 Installation

```
$ pip install foliantcontrib.includes
```

5.4.2 Config

To enable the preprocessor with default options, add includes to preprocessors section in the project config:

```
preprocessors:
  - includes
```

The preprocessor has a number of options:

```
preprocessors:
  - includes:
      cache_dir: !path .includescache
      recursive: true
      aliases:
        ...
```

`cache_dir` Path to the directory for cloned repositories. It can be a path relative to the project path or a global one; you can use ~/ shortcut.

Note

To include files from remote repositories, the preprocessor clones them. To save time during build, cloned repositories are stored and reused in future builds.

`recursive` Flag that defines whether includes in included documents should be processed.

`aliases` Mapping from aliases to Git repository URLs. Once defined here, an alias can be used to refer to the repository instead of its full URL.

For example, if you set this alias in the config:

```
- includes:
  aliases:
    foo: https://github.com/boo/bar.git
```

you can include README.md file content from this repository using this syntax:

```
<include>$foo$path/to/doc.md</include>
```

5.4.3 Usage

To include a document from your machine, put the path to it between <include>...</include> tags:

```
Text below is taken from another document.

<include>/path/to/another/document.md</include>
```

To include a document from a remote Git repository, put its URL between $s in front of the document path:

```
Text below is taken from a remote repository.

<include>
    $https://github.com/foo/bar.git$path/to/doc.md
</include>
```

If the repository alias is defined in the project config, you can use it instead of the URL:

```
- includes:
    aliases:
      foo: https://github.com/foo/bar.git
```

And then in the source:

```
<include>$foo$path/to/doc.md</include>
```

You can also specify a particular branch or revision:

```
Text below is taken from a remote repository on branch develop.

<include>$foo#develop$path/to/doc.md</include>
```

To include a part of a document between two headings, use the #Start:Finish syntax after the file path:

```
Include content from "Intro" up to "Credits":

<include>sample.md#Intro:Credits</include>

Include content from start up to "Credits":

<include>sample.md#:Credits</include>

Include content from "Intro" up to the next heading of the same level:

<include>sample.md#Intro</include>
```

Options

`sethead` The level of the topmost heading in the included content. Use it to guarantee that the included text doesn't break the parent document's heading order:

```
# Title

## Subtitle

<include sethead="3">
    other.md
</include>
```

`nohead` Flag that tells the preprocessor to strip the starting heading from the included content:

```
# My Custom Heading

<include nohead="true">
    other.md#Original Heading
</include>
```

Default is false.

Options can be combined. For example, use both sethead and nohead if you want to include a section with a custom heading:

```
## My Custom Heading

<include sethead="1" nohead="true">
  other.md#Original Heading
</include>
```

6 CLI Extensions

6.1 Init

This CLI extension add `init` command that lets you create Foliant projects from templates.

6.1.1 Installation

```
$ pip install foliantcontrib.init
```

6.1.2 Usage

Create project from the default "basic" template:

```
$ foliant init
Enter the project name: Awesome Docs
 Generating Foliant project

Project "Awesome Docs" created in /path/to/awesome-docs
```

Create project from a custom template:

```
$ foliant init --template /path/to/custom/template
Enter the project name: Awesome Customized Docs
 Generating Foliant project

Project "Awesome Customized Docs" created in /path/to/awesome-customized-doc
```

You can provide the project name without user prompt:

```
$ foliant init --name Awesome Docs
 Generating Foliant project

Project "Awesome Docs" created in /path/to/awesome-docs
```

Another useful option is `--quiet`, which hides all output except for the path to the generated project:

```
$ foliant init --name Awesome Docs --quiet
/path/to/awesome-docs
```

To see all available options, run `foliant init --help`:

```
$ foliant init --help
usage: foliant init [-h] [-n NAME] [-t NAME or PATH] [-q]

Generate new Foliant project.

optional arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  Name of the Foliant project
```

```
-t NAME or PATH, --template NAME or PATH
                      Name of a built-in project template or path to custor
-q, --quiet           Hide all output accept for the result. Useful for pi
```

6.1.3 Project Templates

A project template is a regular Foliant project, but with `title` value in `foliant.yml` replaced with the `{title}` placeholder. When the project is generated, the placeholder is replaced with the actual project name.

There is a built-in template called "basic." It's used by default if you don' specify the template.

> **Note**
>
> More placeholders and built-in templates may be added in future versions.