joeyespo / grip

Preview GitHub Markdown files like Readme locally before committing them.

497 commits 2 branches 18 releases 23 contributors MIT

joeyespo committed on GitHub Merge pull request #222 from joeyespo/fix-221 ... Latest commit 04a7b11 on Dec 11, 2016

artwork	Add a favicon.	3 years ago
docs	Add docs directory with TODO.	4 years ago
grip	Manually set USERNAME and PASSWORD defaults to empty string	2 months ago
tests	Regenerate test output files for #193.	6 months ago
.gitignore	Add .cache to .gitignore.	a year ago
.travis.yml	Group pypy in Travis config.	10 months ago
AUTHORS.md	Set version to 4.3.0.	6 months ago
CHANGES.md	Add Development section to CHANGES.	6 months ago
LICENSE	Update LICENSE year.	10 months ago
MANIFEST.in	Fix setup.py:	6 months ago
README.md	Use 'index.html' in --export example for searchability.	5 months ago
pytest.ini	Add API tests, more CLI tests, and GitHub assumption tests.	a year ago
requirements-test.txt	Add API tests, more CLI tests, and GitHub assumption tests.	a year ago
requirements.txt	Allow port 0 (random port).	6 months ago
setup.py	Set version to 4.3.2.	6 months ago

README.md

Grip -- GitHub Readme Instant Preview

Render local readme files before sending off to GitHub.

Grip is a command-line server application written in Python that uses the GitHub markdown API to render a local readme file. The styles come directly from GitHub, so you'll know exactly how it will appear. Changes you make to the Readme will be instantly reflected in the browser without requiring a page refresh.

Motivation

Sometimes you just want to see the exact readme result before committing and pushing to GitHub.

Especially when doing Readme-driven development.

Installation

To install grip, simply:

```
$ pip install grip
```

On OS X, you can also install with Homebrew:

```
$ brew install grip
```

Usage

To render the readme of a repository:

```
$ cd myrepo
$ grip
 * Running on http://localhost:6419/
```

Now open a browser and visit http://localhost:6419. Or run with `-b` and Grip will open a new browser tab for you.

You can also specify a port:

```
$ grip 80
 * Running on http://localhost:80/
```

Or an explicit file:

```
$ grip AUTHORS.md
 * Running on http://localhost:6419/
```

Alternatively, you could just run `grip` and visit localhost:6419/AUTHORS.md since grip supports relative URLs.

You can combine the previous examples. Or specify a hostname instead of a port. Or provide both.

```
$ grip AUTHORS.md 80
 * Running on http://localhost:80/
```

```
$ grip CHANGES.md 0.0.0.0
 * Running on http://0.0.0.0:6419/
```

```
$ grip . 0.0.0.0:80
 * Running on http://0.0.0.0:80/
```

You can even bypass the server and **export** to a single HTML file, with all the styles and assets inlined:

```
$ grip --export
Exporting to README.html
```

Control the output name with the second argument:

```
$ grip README.md --export index.html
Exporting to index.html
```

If you're exporting a bunch of files, you can prevent styles from being inlining to save space with `--no-inline`:

```
$ grip README.md --export --no-inline introduction.html
Exporting to introduction.html
```

Reading and writing from **stdin** and **stdout** is also supported, allowing you to use Grip with other programs:

```
$ cat README.md | grip -
 * Running on http://localhost:6419/


$ grip AUTHORS.md --export - | bcat


$ cat README.md | grip --export - | less
```

This allows you to quickly test how things look by entering Markdown directly in your terminal:

```
$ grip -
Hello **world**!
^D
 * Running on http://localhost:6419/
```

Note: `^D` means `Ctrl+D`, which works on Linux and OS X. On Windows you'll have to use `Ctrl+Z`.

Rendering as user-content like **comments** and **issues** is also supported, with an optional repository context for linking to issues:

```
$ grip --user-content --context=joeyespo/grip
 * Running on http://localhost:6419/
```

For more details and additional options, see the help:

```
$ grip -h
```

Access

Grip strives to be as close to GitHub as possible. To accomplish this, grip uses GitHub's Markdown API so that changes to their rendering engine are reflected immediately without requiring you to upgrade grip. However, because of this you may hit the API's hourly rate limit. If this happens, grip offers a way to access the API using your credentials to unlock a much higher rate limit.

```
$ grip --user <your-username> --pass <your-password>
```

Or use a personal access token with an empty scope (note that a token is *required* if your GitHub account is set up with two-factor authentication):

```
$ grip --pass <token>
```

You can persist these options in your local configuration. For security purposes, it's highly recommended that you **use an access token over a password**. (You could also keep your password safe by configuring Grip to grab your password from a password manager.)

There's also a work-in-progress branch to provide **offline rendering**. Once this resembles GitHub more precisely, it'll be exposed in the CLI, and will ultimately be used as a seamless fallback engine for when the API can't be accessed.

Grip always accesses GitHub over HTTPS, so your README and credentials are protected.

Tips

Here's how others from the community are using Grip.

Want to share your own? Say hello @joeyespo or submit a pull request.

Create a local mirror of a Github Wiki

```
$ git clone https://github.com/YOUR_USERNAME/YOUR_REPOSITORY.wiki.git
$ cd YOUR_REPOSITORY.wiki
$ grip
```

By Joshua Gourneau.

Generate HTML documentation from a collection of linked README files

1. Enter the directory:

   ```
   $ cd YOUR_DIR
   $ export GRIPURL=$(pwd)
   ```

2. Include all assets by setting the `CACHE_DIRECTORY` config variable:

   ```
   $ echo "CACHE_DIRECTORY = '$(pwd)/assets'" >> ~/.grip/settings.py
   ```

3. Export all your Markdown files with Grip and replace absolute asset paths with relative paths:

   ```
   $ for f in *.md; do grip --export $f --inline=False; done
   $ for f in *.html; do sed -i '' "s?$GRIPURL/??g" $f; done
   ```

You can optionally compress the set of HTML files to `docs.tgz` with:

```
$ tar -czvf docs.tgz `ls | grep [\.]html$` assets
```

Looking for a cross platform solution? Here's an equivalent Python script.

By Matthew R. Tanudjaja.

Configuration

To customize Grip, create `~/.grip/settings.py`, then add one or more of the following variables:

- `HOST`: The host to use when not provided as a CLI argument, `localhost` by default
- `PORT`: The port to use when not provided as a CLI argument, `6419` by default
- `DEBUG`: Whether to use Flask's debugger when an error happens, `False` by default
- `DEBUG_GRIP`: Prints extended information when an error happens, `False` by default
- `API_URL`: Base URL for the github API, for example that of a Github Enterprise instance. `https://api.github.com` by default
- `CACHE_DIRECTORY`: The directory, relative to `~/.grip`, to place cached assets (this gets run through the following filter: `CACHE_DIRECTORY.format(version=__version__)`), `'cache-{version}'` by default

- `AUTOREFRESH` : Whether to automatically refresh the Readme content when the file changes, `True` by default
- `QUIET` : Do not print extended information, `False` by default
- `STYLE_URLS` : Additional URLs that will be added to the rendered page, `[]` by default
- `USERNAME` : The username to use when not provided as a CLI argument, `None` by default
- `PASSWORD` : The password or personal access token to use when not provided as a CLI argument (*Please don't save your passwords here.* Instead, use an access token or drop in this code grab your password from a password manager), `None` by default

Environment variables

- `GRIPHOME` : Specify an alternative `settings.py` location, `~/.grip` by default
- `GRIPURL` : The URL of the Grip server, `/__/grip` by default

Advanced

This file is a normal Python script, so you can add more advanced configuration.

For example, to read a setting from the environment and provide a default value when it's not set:

```
PORT = os.environ.get('GRIP_PORT', 8080)
```

API

You can access the API directly with Python, using it in your own projects:

```
from grip import serve

serve(port=8080)
 * Running on http://localhost:8080/
```

Run main directly:

```
from grip import main

main(argv=['-b', '8080'])
 * Running on http://localhost:8080/
```

Or access the underlying Flask application for even more flexibility:

```
from grip import create_app

grip_app = create_app(user_content=True)
# Use in your own app
```

Documentation

serve

Runs a local server and renders the Readme file located at `path` when visited in the browser.

```
serve(path=None, host=None, port=None, user_content=False, context=None, username=None, password=None, renc
```

- `path` : The filename to render, or the directory containing your Readme file, defaulting to the current working directory
- `host` : The host to listen on, defaulting to the HOST configuration variable

- `port` : The port to listen on, defaulting to the PORT configuration variable
- `user_content` : Whether to render a document as user-content like user comments or issues
- `context` : The project context to use when `user_content` is true, which takes the form of `username/project`
- `username` : The user to authenticate with GitHub to extend the API limit
- `password` : The password to authenticate with GitHub to extend the API limit
- `render_offline` : Whether to render locally using Python-Markdown (Note: this is a work in progress)
- `render_wide` : Whether to render a wide page, `False` by default (this has no effect when used with `user_content`)
- `render_inline` : Whether to inline the styles within the HTML file
- `api_url` : A different base URL for the github API, for example that of a Github Enterprise instance. The default is the public API https://api.github.com.
- `title` : The page title, derived from `path` by default
- `autorefresh` : Automatically update the rendered content when the Readme file changes, `True` by default
- `browser` : Open a tab in the browser after the server starts., `False` by default
- `grip_class` : Use a custom Grip class

export

Writes the specified Readme file to an HTML file with styles and assets inlined.

```
export(path=None, user_content=False, context=None, username=None, password=None, render_offline=False, rer
```

- `path` : The filename to render, or the directory containing your Readme file, defaulting to the current working directory
- `user_content` : Whether to render a document as user-content like user comments or issues
- `context` : The project context to use when `user_content` is true, which takes the form of `username/project`
- `username` : The user to authenticate with GitHub to extend the API limit
- `password` : The password to authenticate with GitHub to extend the API limit
- `render_offline` : Whether to render locally using Python-Markdown (Note: this is a work in progress)
- `render_wide` : Whether to render a wide page, `False` by default (this has no effect when used with `user_content`)
- `render_inline` : Whether to inline the styles within the HTML file (Note: unlike the other API functions, this defaults to `True`)
- `out_filename` : The filename to write to, `<in_filename>.html` by default
- `api_url` : A different base URL for the github API, for example that of a Github Enterprise instance. The default is the public API https://api.github.com.
- `title` : The page title, derived from `path` by default
- `grip_class` : Use a custom Grip class

create_app

Creates a Flask application you can use to render and serve the Readme files. This is the same app used by `serve` and `export` and initializes the cache, using the cached styles when available.

```
create_app(path=None, user_content=False, context=None, username=None, password=None, render_offline=False,
```

- `path` : The filename to render, or the directory containing your Readme file, defaulting to the current working directory
- `user_content` : Whether to render a document as user-content like user comments or issues
- `context` : The project context to use when `user_content` is true, which takes the form of `username/project`

- `username` : The user to authenticate with GitHub to extend the API limit
- `password` : The password to authenticate with GitHub to extend the API limit
- `render_offline` : Whether to render locally using Python-Markdown (Note: this is a work in progress)
- `render_wide` : Whether to render a wide page, `False` by default (this has no effect when used with `user_content`)
- `render_inline` : Whether to inline the styles within the HTML file
- `api_url` : A different base URL for the github API, for example that of a Github Enterprise instance. The default is the public API https://api.github.com.
- `title` : The page title, derived from `path` by default
- `text` : A string or stream of Markdown text to render instead of being loaded from `path` (Note: `path` can be used to set the page title)
- `grip_class` : Use a custom Grip class

render_app

Renders the application created by `create_app` and returns the HTML that would normally appear when visiting that route.

```
render_app(app, route='/')
```

- `app` : The Flask application to render
- `route` : The route to render, '/' by default

render_content

Renders the specified markdown text without caching.

```
render_content(text, user_content=False, context=None, username=None, password=None, render_offline=False,
```

- `text` : The Markdown text to render
- `user_content` : Whether to render a document as user-content like user comments or issues
- `context` : The project context to use when `user_content` is true, which takes the form of `username/project`
- `username` : The user to authenticate with GitHub to extend the API limit
- `password` : The password to authenticate with GitHub to extend the API limit
- `render_offline` : Whether to render locally using Python-Markdown (Note: this is a work in progress)
- `api_url` : A different base URL for the github API, for example that of a Github Enterprise instance. This is required when not using the offline renderer.
- `title` : The page title, derived from `path` by default

render_page

Renders the markdown from the specified path or text, without caching, and returns an HTML page that resembles the GitHub Readme view.

```
render_page(path=None, user_content=False, context=None, username=None, password=None, render_offline=False
```

- `path` : The path to use for the page title, rendering `'README.md'` if None
- `user_content` : Whether to render a document as user-content like user comments or issues
- `context` : The project context to use when `user_content` is true, which takes the form of `username/project`
- `username` : The user to authenticate with GitHub to extend the API limit
- `password` : The password to authenticate with GitHub to extend the API limit

- `render_offline` : Whether to render offline using Python-Markdown (Note: this is a work in progress)
- `render_wide` : Whether to render a wide page, `False` by default (this has no effect when used with `user_content`)
- `render_inline` : Whether to inline the styles within the HTML file
- `api_url` : A different base URL for the github API, for example that of a Github Enterprise instance. The default is the public API https://api.github.com.
- `title` : The page title, derived from `path` by default
- `text` : A string or stream of Markdown text to render instead of being loaded from `path` (Note: `path` can be used to set the page title)
- `grip_class` : Use a custom Grip class

clear_cache

Clears the cached styles and assets.

```
clear_cache(grip_class=None)
```

main

Runs Grip with the specified arguments.

```
main(argv=None, force_utf8=True)
```

- `argv` : The arguments to run with, `sys.argv[1:]` by default
- `force_utf8` : Sets the default encoding to `utf-8` in the current Python instance. This has no effect on Python 3 since Unicode is handled by default

Classes

class Grip(Flask)

A Flask application that can serve a file or directory containing a README.

```
Grip(source=None, auth=None, renderer=None, assets=None, render_wide=None, render_inline=None, title=None,
```

default_renderer

Returns the default renderer using the current config. This is only used if renderer is set to None in the constructor.

```
Grip.default_renderer()
```

default_asset_manager

Returns the default asset manager using the current config. This is only used if asset_manager is set to None in the constructor.

```
Grip.default_asset_manager()
```

add_content_types

Adds the application/x-font-woff and application/octet-stream content types if they are missing. Override to add additional content types on initialization.

```
Grip.add_content_types()
```

clear_cache

Clears the downloaded assets.

```
Grip.clear_cache()
```

render

Renders the application and returns the HTML unicode that would normally appear when visiting in the browser.

```
Grip.render(route=None)
```

- `route` : The route to render, `/` by default

run

Starts a server to render the README. This calls Flask.run internally.

```
Grip.run(host=None, port=None, debug=None, use_reloader=None, open_browser=False)
```

- `host` : The hostname to listen on. Set this to `'0.0.0.0'` to have the server available externally as well, `'localhost'` by default
- `port` : The port of the webserver. Defaults to `6419`
- `debug` : If given, enable or disable debug mode. See Flask.debug.
- `use_reloader` : Should the server automatically restart the python process if modules were changed? `False` by default unless the `DEBUG_GRIP` setting is specified.
- `open_browser` : Opens the browser to the address when the server starts

class AlreadyRunningError(RuntimeError)

Raised when `Grip.run` is called while the server is already running.

```
AlreadyRunningError()
```

class ReadmeNotFoundError(NotFoundError or IOError)

Raised when the specified Readme could not be found.

```
ReadmeNotFoundError(path=None, message=None)
```

class ReadmeAssetManager(object)

Manages the style and font assets rendered with Readme pages. This is an abstract base class.

```
ReadmeAssetManager(cache_path, style_urls=None)
```

class GitHubAssetManager(ReadmeAssetManager)

Manages the style and font assets rendered with Readme pages. Set cache_path to None to disable caching.

class ReadmeReader(object)

Reads Readme content from a URL subpath. This is an abstract base class.

```
ReadmeReader()
```

class DirectoryReader(ReadmeReader)

Reads Readme files from URL subpaths.

```
DirectoryReader(path=None, silent=False)
```

class TextReader(ReadmeReader)

Reads Readme content from the provided unicode string.

```
TextReader(text, display_filename=None)
```

class StdinReader(TextReader)

Reads Readme text from STDIN.

```
StdinReader(display_filename=None)
```

class ReadmeRenderer(object)

Renders the Readme. This is an abstract base class.

```
ReadmeRenderer(user_content=None, context=None)
```

class GitHubRenderer(ReadmeRenderer)

Renders the specified Readme using the GitHub Markdown API.

```
GitHubRenderer(user_content=None, context=None, api_url=None, raw=None)
```

class OfflineRenderer(ReadmeRenderer)

Renders the specified Readme locally using pure Python. Note: This is currently an incomplete feature.

```
OfflineRenderer(user_content=None, context=None)
```

Constants

SUPPORTED_TITLES

The common Markdown file titles on GitHub.

```
SUPPORTED_TITLES = ['README', 'Home']
```

- `filename` : The UTF-8 file to read.

SUPPORTED_EXTENSIONS

The supported extensions, as defined by GitHub.

```
SUPPORTED_EXTENSIONS = ['.md', '.markdown']
```

DEFAULT_FILENAMES

This constant contains the names Grip looks for when no file is provided.

```
DEFAULT_FILENAMES = [title + ext
                     for title in SUPPORTED_TITLES
                     for ext in SUPPORTED_EXTENSIONS]
```

DEFAULT_FILENAME

This constant contains the default Readme filename, namely:

```
DEFAULT_FILENAME = DEFAULT_FILENAMES[0]  # README.md
```

DEFAULT_GRIPHOME

This constant points to the default value if the `GRIPHOME` environment variable is not specified.

```
DEFAULT_GRIPHOME = '~/.grip'
```

DEFAULT_GRIPURL

The default URL of the Grip server and all its assets:

```
DEFAULT_GRIPURL = '/__/grip'
```

DEFAULT_API_URL

The default app_url value:

```
DEFAULT_API_URL = 'https://api.github.com'
```

Testing

Install the package and test requirements:

```
$ pip install -e .[tests]
```

Run tests with pytest:

```
$ py.test
```

Or to re-run tests as you make changes, use pytest-watch:

```
$ ptw
```

External assumption tests

If you're experiencing a problem with Grip, it's likely that an assumption made about the GitHub API has been broken. To verify this, run:

```
$ py.test -m assumption
```

Since the external assumptions rely on an internet connection, you may want to skip them when developing locally. Tighten the cycle even further by stopping on the first failure with `-x`:

```
$ py.test -xm "not assumption"
```

Or with pytest-watch:

```
$ ptw -- -xm "not assumption"
```

Contributing

1. Check the open issues or open a new issue to start a discussion around your feature idea or the bug you found
2. Fork the repository, make your changes, and add yourself to Authors.md
3. Send a pull request

If your PR has been waiting a while, feel free to ping me on Twitter.

Use this software often? Please consider supporting Grip.