grec Documentation

Release 0.2.0

Michael Brennan

October 17, 2014

Contents

1 grec **3**
- 1.1 Features . . . 3
- 1.2 Quick Start . . . 3
- 1.3 Command line . . . 4
- 1.4 TODO . . . 5

2 Installation **7**

3 Usage **9**

4 Contributing **11**
- 4.1 Types of Contributions . . . 11
- 4.2 Get Started! . . . 12
- 4.3 Pull Request Guidelines . . . 12

5 Credits **13**
- 5.1 Development Lead . . . 13
- 5.2 Contributors . . . 13

6 History **15**

7 0.2.0 (2014-10-14) **17**

8 0.1.0 (2014-09-26) **19**

9 Indices and tables **21**

Contents:

CHAPTER 1

grec

Colorize terminal text with regular expressions.

- Free software: GPL version 3
- Documentation: https://grec.readthedocs.org.

1.1 Features

grec is similar to *grep*; the difference being that instead of printing matching lines in a file, *grec* colorizes lines. Lines that do not match any pattern are still printed, but without color.

The key feature that separates this utility from other similar ones is that it's possible to colorize a matching string several times without letting previously matched colors mess up following regular expression matches.

1.2 Quick Start

Install with *pip*:

```
pip install grec
```

Print the contents of *log_file.txt* in its entirety but also colorize any occurrences of errors and warnings:

```
grec -m ERROR red -m WARN yellow log_file.txt
```

Or use a pipe:

```
cat log_file.txt | grec -m ERROR red -m WARN yellow -
```

To colorize strings in Python code, use the *Matcher* class from the *grec* package:

```
>>> from grec import Matcher
>>> m = Matcher()
>>> m.add_pattern('ERROR', 'red')
>>> m.add_pattern('WARN', 'yellow')
>>> print m.match('ERROR WARN INFO')
ERROR WARN INFO  (with color)
```

1.3 Command line

The command line interface is the following:

```
grec [-m PATTERN COLOR_INFO] [-g PATTERN [COLOR_INFO ...]] -- file
```

1.3.1 The *-m* argument

This argument takes a regular expression and color information. Here's an example that will make all lines starting with the character "#" have a green color with white background:

```
-m '^#.*' green_on_white
```

Whenever a line matches the regular expression, the part of the line that matched is colorized with the color information. Any number of *-m* arguments can be specified, and colorization will be applied in the order specified on the command line.

The regular expression will be matched by the *re* module. So for each regular expression, only non-overlapping matches will be colorized. To get overlapping matches use several patterns by adding more *-m* arguments.

Color information consists of a foreground and optionally a background. Colorization is performed with the *termcolor* package and thus the following colors are supported: *grey*, *red*, *green*, *yellow*, *blue*, *magenta*, *cyan*, *white*.

To only set the foreground color, simply specify the name of the color. To also set a background color, add it to the foreground color. Use quotes or underlines to prevent the shell from interpreting it as several arguments. Examples:

```
-m <regex> blue_on_yellow
-m <regex> blue_yellow
-m <regex> 'blue on yellow'
-m <regex> 'blue yellow'
```

1.3.2 The *-g* argument

This argument is similar to *-m* but with the difference that instead of colorizing the whole match, this creates a group pattern that only colorizes matched groups of the regular expression.

Because one can have multiple groups within a regular expression, this argument accepts multiple colors. Here's an example which will colorize the first group with green color on white background and the second with yellow foreground:

```
-g '^(#)(.*)' green_on_white yellow
```

If more colors than there are groups in the regular expression are specified, they will be ignored. If the number of colors is less than the groups, the last color specified for the pattern will be used to colorize all of the remaining group matches.

1.3.3 The file argument

This is the file to colorize. If "-" is specified, *stdin* will be read instead and can be used to colorize the output of a pipe.

If no file is given, *stdin* will be used as the default.

1.4 TODO

- Add support for attributes like blinking
- Add support for only changing background color from CLI
- Python 3 support

CHAPTER 2

Installation

At the command line:

```
$ easy_install grec
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv grec
$ pip install grec
```

CHAPTER 3

Usage

To use grec in a project:

```
import grec
```

The *grec* package exports the *Matcher* class. To prepare text colorization, create a new *Matcher* instance and add patterns to it:

```
matcher = grec.Matcher()
matcher.add_group_pattern('(ERROR):(.*)$', ['white', 'red'], ['magenta'])
matcher.add_pattern('WARN', 'yellow')
matcher.add_pattern('[0-9]', 'blue', 'white')
```

This instance is now ready to match for colors in strings passed to it. Do it with the *match* method:

```
result = matcher.match('ERROR: message WARN INFO 123')
```

The result is an instance of *ColoredString*. Its *__str__* method will return a string with ANSI escape codes for all matched colors. Thus, the instance can simply be printed to show the text with colors in the terminal. If no patterns matched the string will be printed without color.

Another way to match is to use the *match_iter* method of the matcher. This method takes an iterable and will return an iterator which returns corresponding *ColoredString* instances for each match of the passed iterable.

```
lines = ['WARN 1', 'ERROR 2', 'INFO 3']
for colored_string in matcher.match_iter(lines):
    print colored_string
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/brisad/grec/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

grec could always use more documentation, whether as part of the official grec docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/brisad/grec/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *grec* for local development.

1. Fork the *grec* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/grec.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv grec
   $ cd grec/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 grec tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7 and for PyPy. Check https://travis-ci.org/brisad/grec/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Development Lead

- Michael Brennan <brennan.brisad@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.2.0 (2014-10-14)

- Added support for colorizing groups in regular expressions

CHAPTER 8

0.1.0 (2014-09-26)

- First release on PyPI.

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*