hubugs

Release 0.18.0

Apr 02, 2017

Contents

Warning: I don't really use GitHub anymore, so this package is somewhat orphaned. It should still work *and* I will fix bugs when they're pointed out, but `hubugs` is frankly quite a low priority for me.

hubugs is a very simple client for working with GitHub's issue tracker from the command line.

It is written in Python, and works with Python 2.6 and newer(including Python 3). *hubugs* is released under the GPL v3

Git repository https://github.com/JNRowe/hubugs/

Issue tracker https://github.com/JNRowe/hubugs/issues/

Contributors https://github.com/JNRowe/hubugs/contributors/

CHAPTER 1

Contents

Background

GitHub provide a great issue tracker for small projects. It is simple, light and fast. There's only one problem for me, I prefer command line tools for these tasks.

Luckily, the issue tracker exposes your data via a thoroughly documented API, so alternative interfaces are only a SMOP (Small Matter of Programming) away.

The requirements

I live at the command line, data that isn't easily manageable from there may as well not exist in my eyes. Therefore:

> A simple command line interface is priority one.

It is nice to customise the views of data, and improve the look and feel of a view over time. Therefore:

> Views must be customisable via simple templates

Data display must be fast, and require the minimum possible network access. Therefore:

> Network data must be cached and compressed where possible

Installation

You can install *hubugs* either via PyPI (Python Package Index) or from source.

Using PyPI

To install using pip:

```
$ pip install hubugs  # to install in Python's site-packages
$ pip install --install-option="--user" hubugs  # to install for a single user
```

To install using easy_install:

```
$ easy_install hubugs
```

From source

If you have downloaded a source tarball you can install it with the following steps:

```
$ python setup.py build
# python setup.py install  # to install in Python's site-packages
$ python setup.py install --user  # to install for a single user
```

hubugs depends on following packages, all of which are available from PyPI:

- click an excellent package for building command line tools in Python
- configparser for configuration file handling
- html2text is used formatting HTML for the terminal
- httplib2 for HTTP communications
- Jinja2 for templating
- misaka is used for converting issue text to HTML
- Pygments for syntax highlighting in template output

Configuration

Before **hubugs** can operate on issues you must generate an OAuth token. **hubugs** provides functionality to do this:

```
 hubugs setup
GitHub user? [JNRowe]
GitHub password? <password>
Support private repositories? (Y/n) y
Configuration complete!
```

Note: You can revoke the generated token at any time from the GitHub settings page.

If you wish to set the authorisation token from the command line you can use the HUBUGS_TOKEN environment variable. For example:

```
 HUBUGS_TOKEN=xxx hubugs open
```

Usage

The **hubugs** script is the main workhorse of *hubugs*.

Let's start with some basic examples:

```
 hubugs list  # List the open bugs for the current project
Id Title
 5 Handle some GitHub markdown extensions [feature]
 6 Sphinx documentation tree [task]
 7 Support per project templates [feature]

3 open bugs found

 hubugs search markdown  # Search for bugs matching markdown
Id Title
 5 Handle some GitHub markdown extensions [feature]

1 open bug found matching markdown

 hubugs show 6  # Show bug number 6
          Id: 6
       Title: Sphinx documentation tree
      Labels: task
     Created: yesterday by JNRowe
     Updated: yesterday
       State: open
    Comments: 0
Pull request: No

This project deserves some real user documentation, not just a few notes in
`README.rst`.

 hubugs comment 6  # Comment on bug 6 using your editor

 hubugs comment -m"New comment." 6  # Add comment from command line
```

Options

--version
show program's version number and exit

-h, --help
show program's help message and exit

--pager <pager>
pass output through a pager

--no-pager
do not pass output through pager

-p <project>, **--project**=<project>
GitHub project to operate on

-u <url>, **--host-url**=<url>
host to connect to, for GitHub Enterprise support

Note: You can set a default value for the `--pager` and `--host-url` options by defining `hubugs.pager` or `hubugs.host-url` respectively in your `git` configuration files. Both global and project local settings are supported, see *`git-config(1)`* for more information.

Commands

`setup` - Generate a new GitHub access token

```
hubugs.py setup [-h] [--local]
```

--local
set access token for local repository only

`list` - List bugs for a project

```
hubugs list [-h] [-s {open,closed,all}] [-l label]
    [-o {number,updated}]
```

-s <state>, **--state**=<state>
state of bugs to operate on

-l <label>, **--label**=<label>
list bugs with specified label

-o <order>, **--order**=<order>
sort order for listing bugs

-p <number>, **--page** <number>
page number

-r, --pull-requests
list only pull requests

`search` - Search bugs reports in a project

```
hubugs search [-h] [-s {open,closed,all}]
    [-o {number,updated}]
    term
```

-s <state>, **--state**=<state>
state of bugs to operate on

-o <order>, **--order**=<order>
sort order for listing bugs

`show` - Show specific bug(s) from a project

```
hubugs show [-h] [-f] [-p] bugs [bugs ...]
```

-f, --full
show bug including comments

-p, --patch
display patches for pull requests

-o, --patch-only
display only the patch content of pull requests

-b, --browse
open bug in web browser

open - Open a new bug in a project

```
hubugs open [-h] [-a label] [--stdin] [title] [body]
```

-a label, **--add** label
add label to issue

--stdin
read message from standard input

comment - Comment on an existing bug in a project

```
hubugs comment [-h] [--stdin] [-m MESSAGE] bugs [bugs ...]
```

--stdin
read message from standard input

-m <text>, **--message**=<text>
comment text

edit - Edit an existing bug in a project

```
hubugs edit [-h] [--stdin] [title] [body] bugs [bugs ...]
```

--stdin
read message from standard input

close - Close an existing bug in a project

```
hubugs close [-h] [--stdin] [-m MESSAGE] bugs [bugs ...]
```

--stdin
read message from standard input

-m <text>, **--message**=<text>
comment text

reopen - Reopen a previously closed bug in a project

```
reopen [-h] [--stdin] [-m MESSAGE] bugs [bugs ...]
```

--stdin
read message from standard input

-m <text>, **--message**=<text>
comment text

`label` - Perform labelling actions on an existing bug in a project

```
hubugs label [-h] [-a label] [-r label] bugs [bugs ...]
```

-a <label>, **--add**=<label>
add label to issue

-r <label>, **--remove**=<label>
remove label from issue

`milestone` - Add an issue to a milestone

```
hubugs milestone [-h] milestone [bugs [bugs ...]]
```

`milestones` - Manage repository milestones

```
hubugs milestones [-h] [-o {due_date,completeness}] [-s {open,closed}]
    [-c milestone] [-l]
```

-o <order>, **--order**=<order>
sort order for listing bugs

-s <state>, **--state**=<state>
state of bugs to operate on

-c <name>, **--create**=<name>
create new milestone

-l, **--list**
list available milestones

Templates

Output is produced from templates using Jinja. Before writing your own templates you should read the awesome Jinja template designer documentation.

Note: If you create some cool templates of your own please consider posting them in an issue or pushing them to a fork on GitHub, so that others can benefit.

Template locations

Templates are loaded from directories in the following order:

- If it exists, `${XDG_DATA_HOME:~/.local/share}/hubugs/templates`
- Any `hubugs/templates` directory in the directories specified by `XDG_DATA_DIRS`
- The package's `templates` directory

For information on the usage of `XDG_DATA_HOME` and `XDG_DATA_DIRS` read XDG Base Directory Specification.

Note: For OS X users there is a fallback to `~/Library/Application Support`, if `XDG_DATA_HOME` is unset.

Precedence

The first name match in the order specified above selects the template, so a `view/list.txt` in `$XDG_DATA_HOME/hubugs/templates` overrides the `view/list.txt` provided in the *hubugs* package.

Template sets

You can specify the template set to use by defining a *hubugs.templates* setting in your git configuration files. For example:

```
git config --global hubugs.templates my_templates
```

You can also set project specific template sets by editing a repository's config. See *git-config(1)*.

Naming

Templates are separated in to two groups. The first group, `view`, is for templates used in directly producing consumable output(such as from the `list` subcommand). The second group, `edit`, is for templates used to generate input files for editing text(such as in the `open` subcommand).

`view` group templates currently include:

- `issue.txt` for formatting a single bug
- `list.txt` for formatting list output

`edit` group templates currently include:

- `default.mkd` for general use, such as in commenting on a bug
- `open.mkd` for opening(or editing) bugs

Data

The following variables are available for use in templates

View group

`columns` (*int*)
The width of the current terminal window

`list.txt` data

`project` (*Repository*)
The current project's repository data. See *Repository objects*.

`bugs` (*list*)
Contains the sorted list of bugs to display, if any. See *Bug objects*.

`id_len` (*int*)
Set to the maximum length of the bug IDs to display

`state` (*str*)
The bug states being searched/listed

`order` (*str*)
The display order

`term` (*str*)
The search term being listed, if any

`issue.txt` data

`project` (*Repository*)
The current project's repository data. See *Repository objects*.

`bug` (*list*)
Contains the sorted list of bugs to display, if any. See *Bug objects*

`comments` (*list*)
When displaying a single bug this contains the list of comments associated with a bug, if any. See *Comment objects*

`full` (*bool*)
True, if the user provided the *`hubugs show -f`* option

`patch` (*str*)
The content found at the location in `Bug.patch_url`, if the user provided the *`hubugs show -p`* option

`patch_only` (*bool*)
True, if the user provided the *`hubugs show -o`* option

Edit group

`title` (*str*)
The current bug title in `edit` subcommand sessions. See *`Bug.title`*

`body` (*str*)
The current bug body in `edit` subcommand sessions, if any. See *`Bug.body`*

`comment_char` (*str*)
The character to use for comments in templates, defaults to '#'. See `core.commentchar` in *`git-config(1)`*

All groups

Jinja templates support object attribute and method access, so an individual `bug` object's *`created_at`* attribute can be called with a `strftime()` method for custom date output. For example, `{{ bug.created_at.strftime("%a, %e %b %Y %H:%M:%S %z") }}` can be used to output an **RFC 2822**-style date stamp.

If you're authoring your own templates and you find you need extra data for their generation open an issue.

Filters

`hubugs` defines the following filters beyond the huge range of excellent built-in filters in Jinja:

Note: If you write extra filters that you believe could be of use to other *`hubugs`* users please consider posting them in an issue or pushing them to a fork on GitHub, so that others can benefit from your work.

---

colourise

This filter applies a colour to text, if possible.

For example, to show a bug's `title` attribute in red:

```
{{ bug.title | colourise('red') }}
```

or to display black text on a red background:

```
{{ bug.title | colourise('black on red') }}
```

Note: This filter is also available under the synonym `colorize`.

---

highlight

This filter highlights text using Pygments. You can specify the lexer to be used, and also the formatter.

For example, to highlight a chunk of text as Python:

```
{{ text | highlight('python') }}
```

To do the same using the 256-colour mode of Pygments:

```
{{ text | highlight('python', 'terminal256') }}
```

See the output of **`pygmentize -L`** for the list of available lexers and formatters.

html2text

This filter converts HTML to a plain text representation using html2text.

markdown

The purpose of this filter is to convert the Markdown formatted text from a GitHub issue to html. The excellent misaka package is used to provide the conversion.

In the default templates it is used to render bug bodies:

```
{{ comment.body | markdown | html2text }}
```

Note: We ping-pong the conversion from Markdown to HTML as it produces a prettier text representation of the comment. We benefit from uniform newline usage and clean word wrapping of the output.

relative_time

This filter is used to generate a human-readable relative timestamp from a `datetime` object.

For example, to display a bug's `created_at` attribute as a relative time:

```
{{ bug.created_at | relative_time }}
```

which could produce output such as:

```
about two months ago
```

hubugs

Simple client for GitHub issues

SYNOPSIS

hubugs [option]... <command>

DESCRIPTION

hubugs is a very simple client for working with GitHub's issue tracker. It allows you to perform all the issue related tasks you'd normally perform from the command line.

OPTIONS

--version show program's version number and exit

-h, --help show program's help message and exit

--pager <pager> pass output through a pager

--no-pager do not pass output through pager

-p <project>, --project=<project> GitHub project to operate on. You can supply just `<project>` if you wish to work on one of your own projects, or `<user>/<project>` to operate on another user's repository. Default is derived from the `hubugs.project` setting in the git config or the current repository, if possible.

-u <url>, --host-url=<url> host to connect to, for GitHub Enterprise support

COMMANDS

setup

Generate a new GitHub access token

--local set access token for local repository only

list

List bugs for a project

-s <state>, --state=<state> state of bugs to operate on

-l <label>, --label=<label> list bugs with specified label

-o <order>, --order=<order> sort order for listing bugs

-p <number>, --page <number> page number

-r, --pull-requests list only pull requests

search

Search bugs reports in a project

-s <state>, --state=<state> state of bugs to operate on

-o <order>, --order=<order> sort order for listing bugs

show

Show specific bug(s) from a project

-f, --full show bug including comments

-p, --patch display patches for pull requests

-o, --patch-only display only the patch content of pull requests

-b, --browse open bug in web browser

open

Open a new bug in a project

-a label, --add label add label to issue

--stdin read message from standard input

comment

Comment on an existing bug in a project

--stdin read message from standard input

-m <text>, --message=<text> comment text

edit

Edit an existing bug in a project

--stdin read message from standard input

close

Close an existing bug in a project

--stdin read message from standard input

-m <text>, --message=<text> comment text

reopen

Reopen a previously closed bug in a project

--stdin read message from standard input

-m <text>, --message=<text> comment text

label

Perform labelling actions on an existing bug in a project

-a <label>, --add=<label> add label to issue

-r <label>, --remove=<label> remove label from issue

milestone

Add an issue to a milestone

milestones

Manage repository milestones

-o <order>, --order=<order> sort order for listing bugs

-s <state>, --state=<state> state of bugs to operate on

-c <name>, --create=<name> create new milestone

-l, --list list available milestones

CONFIGURATION

You can specify the template set to use by defining a `hubugs.templates` setting in your git configuration files. For example:

```
git config --global hubugs.templates my_templates
```

You can also set project specific template sets by editing a repository's config. See *git-config(1)*.

You can set a default value for the `--pager` and `--host-url` options by defining `hubugs.pager` or `hubugs.host-url` respectively in your `git` configuration files.

BUGS

None known.

AUTHOR

Written by James Rowe

RESOURCES

Home page, containing full documentation: http://hubugs.rtfd.org/

Issue tracker: https://github.com/JNRowe/hubugs/issues/

COPYING

Frequently Asked Questions

Do you accept template contributions?

Yes, if they are somewhat general. And, it is a great way to have me maintain template compatibility for you in case something changes in a future version.

Either open an issue or push them to a fork on GitHub.

Why is the wrapping broken in comments I make?

Unfortunately, GitHub have crippled the newline behaviour of Markdown. If you wrap your comments for readability then you'll be creating new paragraphs with every single line. There is very little that can be done locally to fix this.

The easiest way to workaround the issue is to disable wrapping in your text editor for template files. The files `hubugs` creates are easy to match for automating this within your editor, just use `$TMPDIR/hubugs-*.mkd`.

How do I create headlines when lines beginning with # are scrubbed?

Markdown syntax supports two different heading formats, and only one of them requires a `#` at the start of a line. Using the alternative setext format is simple:

```
Heading
=======

Sub-heading
-----------
```

The other possibility, although not recommended, is to set a custom value for *core.commentchar* in your `git` configuration settings. program:*hubugs* will honour the set value, and strip any lines that begin with that character. You should note however that this is a recent addition to git, and other tools may not work correctly with it set to a non-default value.

I don't like your choice of template language

[It isn't really a question, but it has come up a couple of times.]

The use of Jinja should only be an issue if you wish to author your own templates, if you're using the built-in templates you shouldn't notice Jinja at all. That said...

The use of Jinja seems to be an entry barrier to some people, but it isn't going to change. For the same – invariably pointless and religious – reasons people prefer other templating engines *I* prefer Jinja.

Note: With all that said, I probably wouldn't be opposed to accepting patches supporting additional *optional* engines ;)

Alternatives

Before diving in and spitting out this package I looked at the alternatives below. If I have missed something please drop me a mail.

Both the `ghi` and `github-cli` packages listed below are very useful, and quite usable, clients for GitHub issues. You should definitely try them out before making a decision on what to use, and this would also allow you to highlight any possible bias I may have shown in comparing them ;)

`ghi`

ghi is a great issues client, written in Ruby.

I personally didn't like the "feel" of the command line interface, and wanted slightly different output.

hubugs provides the following advantages:

- Easily customisable output
- On-disk cache support

and has the following disadvantages:

- *hubugs* requires a significant number of external packages, which may be a problem for some users

github-cli

github-cli is a fantastic command-line client for GitHub's issues, written in Python. I really liked `github-cli` to the extent of authoring a few patches, but just didn't find it comfortable to use in the long run[1].

hubugs provides the following advantages:

- Easily customisable output
- HTTP compression support
- On-disk cache support
- Pull request integration
- Uses the same method as `git` for choosing an editor to use, which will only affect you if you a custom editor for interacting with `git`

and has the following disadvantages:

- *hubugs* requires a significant number of external packages, which may be a problem for some users

Release HOWTO

Test

In the general case tests can be run via `nose2`:

```
$ nose2 -vv tests
```

When preparing a release it is important to check that *hubugs* works with all currently supported Python versions, and that the documentation is correct.

Prepare release

With the tests passing, perform the following steps

- Update the version data in `hubugs/_version.py`
- Update `NEWS.rst`, if there are any user visible changes
- Commit the release notes and version changes
- Create a signed tag for the release
- Push the changes, including the new tag, to the GitHub repository

Update PyPI

Create and upload the new release tarballs to PyPI:

```
$ ./setup.py sdist --formats=bztar,gztar register upload --sign
```

Fetch the uploaded tarballs, and check for errors.

You should also perform test installations from PyPI, to check the experience *hubugs* users will have.

[1] If you look at the timeline for the hubugs repository and my github-cli fork you'll see I was using `github-cli` for a few months after spiking *hubugs*.

API documentation

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

Command line

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

hubugs.**setup**()
Setup GitHub access token.

hubugs.**list_bugs**()
Listing bugs.

hubugs.**search**()
Searching bugs.

hubugs.**show**()
Displaying bugs.

hubugs.**open_bug**()
Opening new bugs.

hubugs.**comment**()
Commenting on bugs.

hubugs.**edit**()
Editing bugs.

hubugs.**close**()
Closing bugs.

hubugs.**reopen**()
Reopening closed bugs.

hubugs.**label**()
Labelling bugs.

hubugs.**milestone**()
Issue milestones.

hubugs.**milestones**()
Repository milestones.

hubugs.**report_bug**()
Report a new bug against hubugs.

hubugs.**main**()
Main command-line entry point.

Return type int

Returns Exit code

models

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

hubugs.models.**object_hook**(*d*, *name='unknown'*)
JSON object hook to create dot-accessible objects.

Parameters

- **d** (*dict*) – Dictionary to operate on
- **name** (*str*) – Fallback name, if dict has no `type` key

hubugs.models.**_v2_conv_timestamp**(*s*)
Parse API v2 style timestamps.

Parameters **s** (*str*) – Timestamp to parse

hubugs.models.**from_search**(*obj*)
Support legacy API search results as API v3 issues(-ish).

This is an awful hack to workaround the lack of search support in API v3. It needs to be removed at the first possible opportunity.

Return type `Issue`

Returns API v2 issue mangled to look like a API v3 result

Template

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

hubugs.template.**get_template**(*group*, *name*)
Fetch a Jinja template instance.

Parameters

- **group** (*str*) – Template group identifier
- **name** (*str*) – Template name

Return type `jinja2.environment.Template`

Returns Jinja template instance

Jinja filter support

hubugs.template.**jinja_filter**(*func*)
Simple decorator to add a new filter to Jinja environment.

Parameters **func** (*func*) – Function to add to Jinja environment

Return type `func`

Returns Unmodified function

hubugs.template.**colourise**(*text*, *fg=None*, *bg=None*, ***kwargs*)
Colourise text.

Returns text untouched if colour output is not enabled

Parameters

- **text** (*str*) – Text to colourise
- **fg** (*str*) – Foreground colour
- **bg** (*str*) – Background colour
- **kwargs** (*dict*) – Formatting to apply to text

Return type str

Returns Colourised text, when possible

hubugs.template.**highlight**(*text*, *lexer='diff'*, *formatter='terminal'*)
Highlight text with pygments.

Returns text untouched if colour output is not enabled

Parameters

- **text** (*str*) – Text to highlight
- **lexer** (*str*) – Jinja lexer to use
- **formatter** (*str*) – Jinja formatter to use

Return type str

Returns Syntax highlighted output, when possible

hubugs.template.**html2text**(*html*, *width=80*, *ascii_replacements=False*)
HTML to plain text renderer.

Parameters

- **text** (*str*) – Text to process
- **width** (*int*) – Paragraph width
- **ascii_replacements** (*bool*) – Use psuedo-ascii replacements for Unicode

Return type str

Returns Rendered text

hubugs.template.**relative_time**(*timestamp*)
Format a relative time.

Taken from bleeter. Duplication is evil, I know.

Parameters **timestamp** (*datetime.datetime*) – Event to generate relative timestamp against

Return type str

Returns Human readable date and time offset

User interface support

hubugs.template.**display_bugs**(*bugs*, *order*, ***extras*)
Display bugs to users.

Parameters

- **bugs** (list` of ``models.Issue) – Bugs to display
- **order** (*str*) – Sorting order for displaying bugs
- **extras** (*dict*) – Additional values to pass to templates

Return type str

Returns Rendered template output

hubugs.template.**edit_text**(*edit_type='default'*, *data=None*)
Edit data with external editor.

Parameters

- **edit_type** (*str*) – Template to use in editor
- **data** (*dict*) – Information to pass to template

Return type str

Returns User supplied text

Raises

- ***EmptyMessageError*** – No message given
- ***EmptyMessageError*** – Message not edited

Utilities

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

hubugs.utils.**setup_environment**(*project*, *host_url*)
Configure execution environment for commands dispatch.

Convenience functions

hubugs.utils.**check_output**(*args*, ***kwargs*)
Simple check_output implementation for Python 2.6 compatibility.

Parameters **args** (*list*) – Command and arguments to call

Return type str

Returns Command output

Raises **subprocess.CalledProcessError** – If command execution fails

hubugs.utils.**get_editor**()
Choose a suitable editor.

See *git-var(1)* for details.

Return type list of str

Returns Users chosen editor, or vi if not set

hubugs.utils.**pager**(*text*, *pager=False*)
Pass output through pager.

Parameters

- **text** (*str*) – Text to page
- **pager** (*bool*) – Pager to use

Git/GitHub support

hubugs.utils.**get_github_api**()
Create a GitHub API instance.

Return type httplib2.Http

Returns GitHub HTTP session

hubugs.utils.**get_git_config_val**(*key*, *default=None*, *local_only=False*)
Fetch a git configuration value.

Parameters

- **key** (*str*) – Configuration value to fetch
- **default** (*str*) – Default value to use, if key isn't set
- **local_only** (*bool*) – Fetch configuration values from repo config only

Return type str

Returns Git config value, if set

hubugs.utils.**set_git_config_val**(*key*, *value*, *local_only=False*)
Set a git configuration value.

Parameters

- **key** (*str*) – Configuration value to fetch
- **value** (*str*) – Value to set
- **local_only** (*bool*) – Set configuration values from repo config only

hubugs.utils.**get_repo**()
Extract GitHub project name from git/hg config.

We check the git config for hubugs.project, and then fall back to remote.origin.url. If both of these fail we check a mercurial root, to satisfy the hg-git users.

Return type str

Returns GitHub project name, including user

hubugs.utils.**sync_labels**(*globs*, *add*, *create*)
Manage labels for a project.

Parameters **globs** (*AttrDict*) – Global argument configuration

Return type list

Returns List of project's label names

Text formatting

hubugs.utils.**_colourise**(*text*, *colour*)
Colour text, if possible.

Parameters

- **text** (*str*) – Text to colourise
- **colour** (*str*) – Colour to display text in

Return type str

Returns Colourised text, if possible

hubugs.utils.**success**(*text*)
Output a success message.

Parameters **text** (*str*) – Text to format

Return type str

Returns Bright green text, if possible

hubugs.utils.**fail**(*text*)
Output a failure message.

Parameters **text** (*str*) – Text to format

Return type str

Returns Bright red text, if possible

hubugs.utils.**warn**(*text*)
Output a warning message.

Parameters **text** (*str*) – Text to format

Return type str

Returns Bright yellow text, if possible

Errors

Note: The documentation in this section is aimed at people wishing to contribute to *hubugs*, and can be skipped if you are simply using the tool from the command line.

exception hubugs.template.**EmptyMessageError**
Error to raise when the user provides an empty message.

exception hubugs.utils.**HttpClientError**(*message*, *response*, *content*)
Error raised for client error status codes.

exception hubugs.utils.**RepoError**
Error raised for invalid repository values.

Appendix

`Bug` objects

`Bug.`**`assignee`** (*User*)
The user the issue is assigned to, if any.

`Bug.`**`body`** (*str*)
The full body of the issue.

`Bug.`**`body_html`** (*str*)
The full body of the issue rendered as HTML.

`Bug.`**`closed_at`** (*datetime.datetime*)
The date and time the issue was closed.

`Bug.`**`closed_by`** (*User*)
The user the issue was closed by, if closed.

`Bug.`**`comments`** (*int*)
The number of comments made on the issue.

`Bug.`**`created_at`** (*datetime.datetime*)
The date and time the issue was created.

`Bug.`**`labels`** (*list*)
`Label` objects associated with the issue.

`Bug.`**`milestone`** (*str*)
The milestone name associated with the issue, if any.

`Bug.`**`number`** (*int*)
The issue's number.

`Bug.`**`pull_request`** (*PullRequest*)
Pull request data associated with the issue.

`Bug.`**`state`** (*str*)
State of the issue, either `open` or `closed`.

`Bug.`**`title`** (*str*)
The issue's title.

`Bug.`**`updated_at`** (*datetime.datetime*)
The date and time when the issue was last updated.

`Bug.`**`user`** (*User*)
The GitHub user that created the issue.

`Comment` objects

`Comment.`**`body`** (*str*)
The full text of the comment.

`Comment.`**`body_html`** (*str*)
The full text of the comment rendered as HTML.

`Comment.`**`created_at`** (*datetime.datetime*)
The date and time the comment was created.

`Comment`.**updated_at** (*datetime.datetime*)
The date and time when the comment was last updated.

`Comment`.**user** (*User*)
The GitHub user that created the comment.

`Label` objects

`Label`.**color**
The colour value for the given label.

`Label`.**name**
The name given to the label.

`PullRequest` objects

`PullRequest`.**diff_url** (*str*)
URL for **diff** output associated with the issue.

`PullRequest`.**patch_url** (*str*)
URL for the **git format-patch** output associated with the issue.

`Repository` objects

`Repository`.**clone_url** (*str*)
Clone URL for fetching via HTTP.

`Repository`.**created_at** (*datetime.datetime*)
The date and time the repository was created.

`Repository`.**description** (*str*)
The description given to the repository

`Repository`.**fork** (*bool*)
Whether the repository is a fork of another on GitHub

`Repository`.**forks** (*int*)
The number of forks on GitHub

`Repository`.**git_url** (*str*)
Clone URL for fetching via `git` protocol.

`Repository`.**has_downloads** (*bool*)
Whether the repository has downloads enabled

`Repository`.**has_issues** (*bool*)
Whether the repository has issues enabled

`Repository`.**has_wiki** (*bool*)
Whether the repository has the wiki enabled

`Repository`.**homepage** (*str*)
The homepage of the repository

`Repository`.**html_url** (*str*)
The main project page on GitHub

`Repository.`**`language`** (*str*)
The programming language used

`Repository.`**`master_branch`** (*str*)
The repository's defined master branch

`Repository.`**`mirror_url`** (*str*)
The original location of a repository, if a mirror

`Repository.`**`name`** (*str*)
The repository's name

`Repository.`**`open_issues`** (*int*)
The number of open issues in a repository

`Repository.`**`owner`** (*User*)
The owner of the repository

`Repository.`**`private`** (*bool*)
Whether the repository is set as private

`Repository.`**`pushed_at`** (*datetime.datetime*)
The last time the repository's content was updated

`Repository.`**`size`** (*int*)
The size of the repository

`Repository.`**`watchers`** (*int*)
The number of watchers of the repository

`User` objects

`avatar_url` (*str*)
The location of the user's avatar

`gravatar_id` (*str*)
The hash identifier for fetching images from gravatar

`login` (*str*)
The user's login name on GitHub

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C