cuv'ner Documentation

Release 0.0.0

meejah

May 30, 2017

Contents

"A commanding view of your test-coverage".

The tool `cuv` provides console tools to show coverage data for you Python project nicely in the console. It uses colour and unicode characters.

`cuv` can:

- graph coverage of all files in your project;
- colour-ize individual source-code files (via `less`) by their coverage;
- colour-ize `diff` (or e.g. `git diff master..HEAD`) by coverage
- diff `.coverage` files themselves

CHAPTER 1

Cuv'ner

"A commanding view of your test-coverage" The command-line tool `cuv` provided by this package gives some useful tools to visualize your project's coverage data. This means you must first run coverage against your project's test-suite.

Once you have a `.coverage` file, you can use the commands documented below (or just type `cuv` to explore the help).

We utilize several quality open-source packages to achieve this:

- coverage by Ned Batchelder
- Click by Armin Ronacher / pocoo
- pygments by Georg Brandl / pocoo
- ansicolors by Giorgos Verigakis
- unidiff by Matias Bordese
- Source Code Pro: the best programming font

Code: https://github.com/meejah/cuvner Docs: https://cuvner.readthedocs.org

Background + Terminology

This started out as some experiments in "whole-project coverage visualization", and then also grew some console tools that I find useful when working with Python code.

I have abandoned the pixel/graphical visualization ideas and proofs-of-concepts into a branch and now this tool is *just* the console visualizations – which are very useful when working on Python code.

As far as my setup, I am using Debian with a 256-color and unicode capable shell using Solarized Dark color schemes. There are probably bugs with other setups, and to a reasonable extent I'm happy to accept pull-reqeusts fixing these. That said, a unicode-capable shell is a must.

Other Neat Visualizations

Other nice "coverage visualization" tools I've run across:

- of course, Coverage.py itself comes with a nice HTML visualization
- emacs-coverage
- codecov.io browser extension shows coverage live while browsing github

Notes on Tox

If you're using tox to run tests (and you should, it's great!) your coverage files will – depending upon setup – end up in `.tox/envname/.coverage` or similar. So, you will either need to use `--coverage` to point cuv'ner at the right file, or simply move it to the top-level of your project for ease-of-use.

Pro Tip(tm)

The "uncovered" lines start with a slightly different unicode character than the "covered" lines, so if you're trying to write tests for uncovered things, you can do this on the "next file that has uncovered things":

```
cuv src/file.py | less -p  -j 4
```

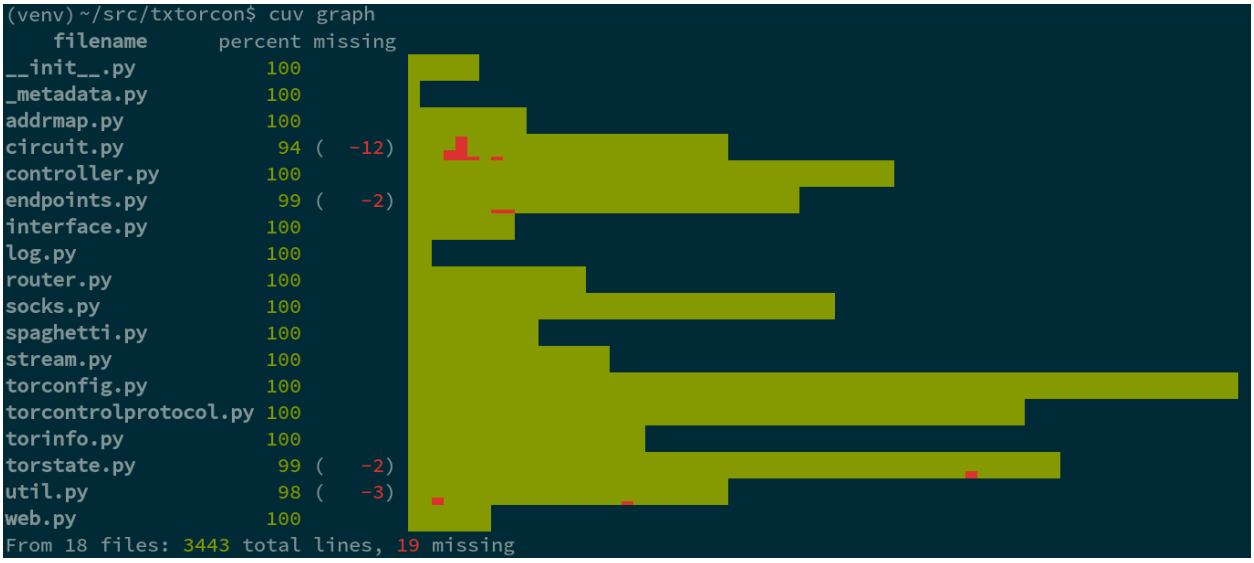
That says (since `cuv lessopen` is the "default" command) to run `cuv lessopen` on `src/file.py` and then , in less, jump to the first uncovered line and place it 4 lines below the top of the screen. You can then hit `n` in less to go to the next one.

This is precisely what the `cuv next` command does.

Console Visualizations

The two main tools usable directly in the console are `cuv graph` and `cuv lessopen` (which can be tied directly into `less` via the `LESSOPEN` environment variable). For determining coverage of branches or individual commits, use `cuv diff`. `cuv spark` can provide a quick overview of a project's coverage.

cuv graph



This displays all the files in your project and a histogram-like graph of their coverage. Each character represents 8 lines of code, and uses a group of unicode characters (0x2580 through 0x2587) to draw a little graph. So, if those 8 lines are not covered at all, the graph will be all red; if they're all covered, it will be all green. If 2 out of the 8 lines are covered, there will be about 25% green and the rest red.

The total size of each file can thus be easily seen (by the length of the histogram part, which wraps to subsequent lines if needed) and an idea of which parts are covered is given.

TODO:

- testing on more terminal types
- how does it look when using something besides Solarized Dark?
- useful, beyond eye-candy?

cuv lessopen

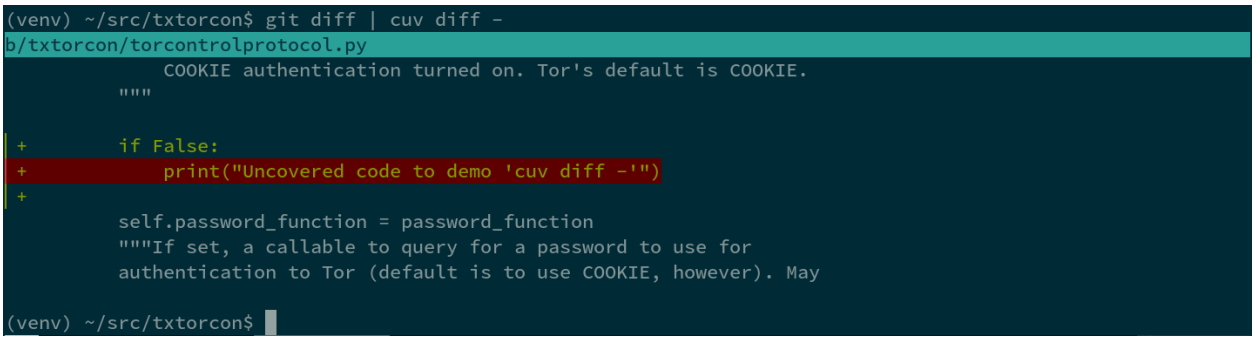
This command is intended to be used via the `LESSOPEN` environment variable, which lets you pre-process files that are opened with `less`. So, once set up (see the help via `cuv lessopen --help`) you can simply run `less` on any file in your project, and it will get syntax-highlighted and show you the line-by-line coverage with a leading green or red mini-verical bar and red background (for uncovered lines).

A header appears at the top showing the total coverage for this particular file.

TODO:

- probably the "proper" way to do this is via a Pygments plugin or extension of some sort
- option to change which Pygments style is used
- dark/light background option?

cuv diff



You can pipe a `git diff` to this and see a colorized version of the diff in your console. I am using a library called unidiff to read the actual diff, which so far works quite nicely. That said, I've only tried against the output of Git, like so:

```
git diff | cuv diff | less
```

TODO:

- colors are unsatisfying, since for added lines they're pretty much the same as Git's colored output
- maybe make it look more like the "real" underlying diff? (e.g. re-create the @@ and so forth things)
- does it work with merge commits?

cuv spark

This shows a "spark-line" sort of thing in the console. It's not very useful for big projects (e.g. Twisted), but gives a very quick overview of the coverage in a small amount of space. Using the same unicode characters as `cuv graph`, this represents each file as a single character, and its percentage coverage is graphed (so you only get granularity down to about 12.5%).

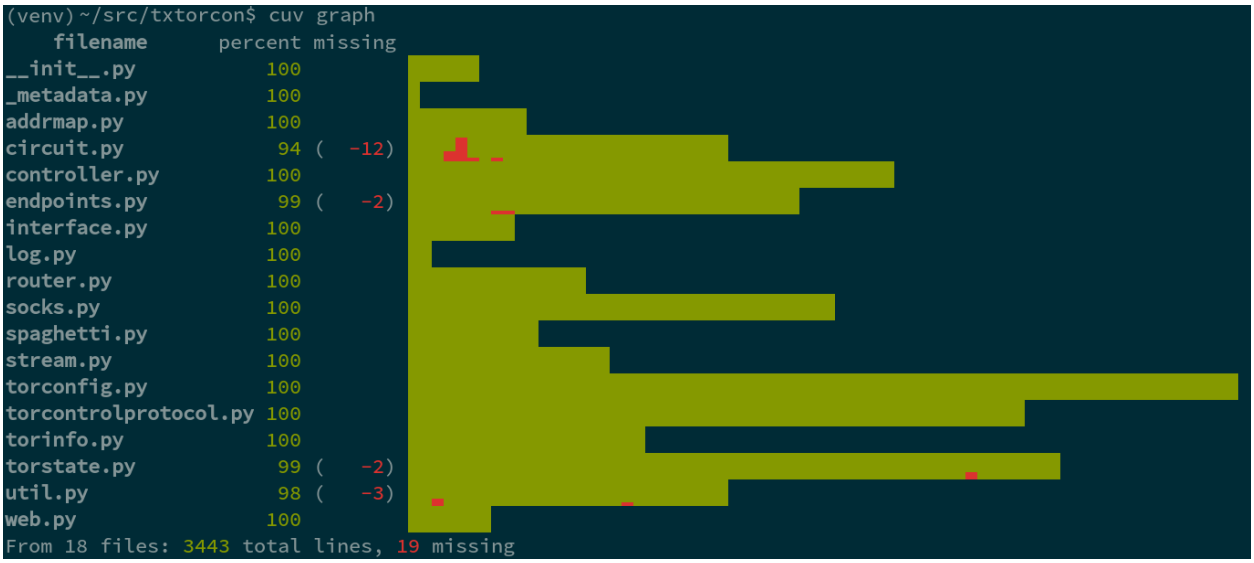
CHAPTER 2

Command Documentation

`cuv graph` Usage: `cuv graph [OPTIONS] [KEYWORD]...`

Console graph of each file's coverage.

- `--help`: Show this message and exit.



`cuv lessopen` Usage: `cuv lessopen [OPTIONS] [INPUT_FILE]`

Syntax + coverage highlighting in console.

Set 'less' up to use this via the LESSOPEN var:

```
export LESSOPEN='| cuv lessopen %s'
```

or if you prefer:

```
export LESSOPEN='| python -m cuv lessopen %s'
```

You may need to provide the full path to 'cuv'. Now, whenever you'less' a file within a project that has coverage data, it will besyntax-highlighted and coloured according to coverage.

- `--help`: Show this message and exit.

`cuv diff` Usage: `cuv diff [OPTIONS] [INPUT_FILE]`

Color a diff by its coverage.

This prints out the whole diff as you would expect, but any added("+") lines in the diff get a red background if they are notcovered.

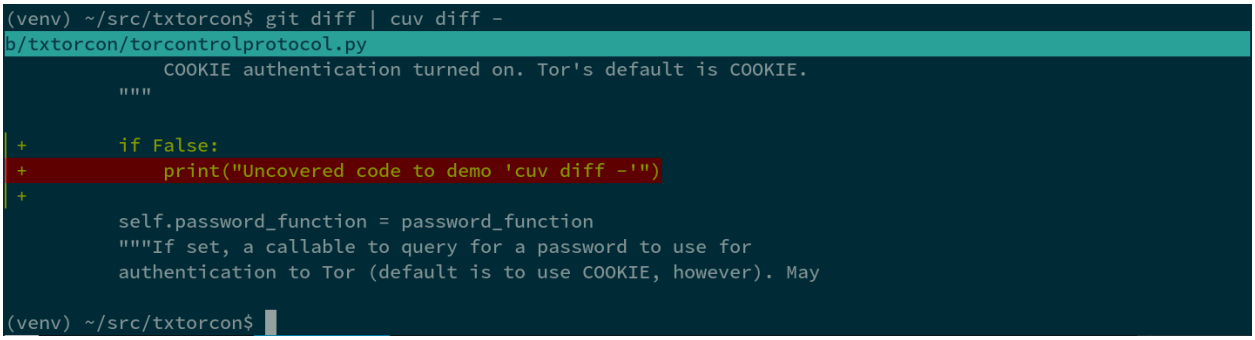
For example, to see if your local changes are covered in a Gitcheckout:

```
git diff | cuv diff -
```

To see if your whole branch is covered:

```
git diff master...HEAD | cuv diff -
```

- `--help`: Show this message and exit.



`cuv next` Usage: `cuv next [OPTIONS]`

Display the next uncovered chunk.

This finds the next file that has some uncovered lines and thenruns:

```
cuv lessopen <filename> | less -p \u258c -j 4
```

- `--ignore TEXT`:
- `-N, --line-numbers`:
- `--help`: Show this message and exit.

```
(venv)~/src/txtorcon$ cuv next --ignore circuit.py
        """

        def progress(*args):
            progress.target(*args)
        config = get_global_tor(
            reactor,
            control_port=control_port,
            progress_updates=progress
        )
        # config is a Deferred here, but endpoint resolves it in
        # the listen() call
        r = TCPHiddenServiceEndpoint(
            reactor, config, public_port,
            hidden_service_dir=hidden_service_dir,
            local_port=local_port,
            stealth_auth=stealth_auth,
        )
        progress.target = r._tor_progress_update
        return r
```

`cuv spark` Usage: `cuv spark [OPTIONS] [KEYWORD]...`

Single-line terminal graph of coverage.

- `--sort / --no-sort`:
- `--help`: Show this message and exit.

```
(venv) ~/src/txtorcon$ cuv spark

(venv) ~/src/txtorcon$
```

`cuv readme` Usage: `cuv readme [OPTIONS]`

View the README

- `--help`: Show this message and exit.