linecook Documentation

Release 0.4.0

Tony S. Yu

Jul 19, 2019

Contents:

`linecook` is a command-line tool that transforms lines of text into a form that's pleasant to consume.

The core goal of `linecook` is to make it easy to create your own transforms to parse whatever text you have. For example, take an `app.log` file that looks like:

```
$ tail app.log

2018-06-09 13:55:26 INFO Dependencies loaded successfully
2018-06-09 13:55:26 WARN Could not find version number for app
2018-06-09 13:55:27 INFO Starting app...
2018-06-09 13:55:27 ERROR SyntaxError: invalid syntax
    >>> while True print('Hello world')
       File "<stdin>", line 1
          while True print('Hello world')
                           ^
    SyntaxError: invalid syntax
```

If you want to highlight the log type and mute the dates/times, then you can create a custom recipe in one of your *Configuration files* like the following:

```
from linecook import patterns as rx
from linecook.transforms import color_text

LINECOOK_CONFIG = {
    'recipes': {
        'my-logs': [
             color_text(rx.any_of(rx.date, rx.time), color='blue'),
             color_text('INFO', color='cyan'),
             color_text('WARN', color='grey', on_color='on_yellow'),
             color_text('ERROR', on_color='on_red'),
        ],
    },
}
```

To use this recipe, you can just pipe the log output to `linecook` with your new recipe as an argument:

```
$ tail app.log | linecook my-logs

2018-06-09 13:55:26 INFO Dependencies loaded successfully
2018-06-09 13:55:26 WARN Could not find version number for app
2018-06-09 13:55:27 INFO Starting app...
2018-06-09 13:55:27 ERROR SyntaxError: invalid syntax
    >>> while True print('Hello world')
       File "<stdin>", line 1
          while True print('Hello world')
                           ^
    SyntaxError: invalid syntax
```

That's all there is to it!

CHAPTER 1

Configuration

1.1 Configuration files

The following configuration files are loaded, in order:

- `linecook.config.core`: Package configuration
- `~/.linecook/config.py`: User configuration
- `./.linecook/config.py`: Local configuration

Each file should define a dictionary named `LINECOOK_CONFIG` containing keys such as `transforms` and `recipes`.

Files loaded later (lower on the list) override values loaded earlier. Note that the overriding happens at the *second* level of dictionaries. For example, if `~/.linecook/config.py` is defined as:

```
from linecook.transforms.core import color_text

LINECOOK_CONFIG = {
    'transforms': {
        'warn_color': color_text(' WARN ', color='yellow'),
        'error_color': color_text(' ERROR ', on_color='on_red'),
    },
    'recipes': {
        'logs': ['warn_color', 'error_color'],
        'default': ['warn_color', 'error_color'],
    },
}
```

And then, `./.linecook/config.py` is defined as:

```
from linecook.transforms.core import filter_line

LINECOOK_CONFIG = {
    'recipes': {
```

(continues on next page)

(continued from previous page)

```
        'default': [filter_line(' DEBUG '), 'error_color']
    },
}
```

The loaded result would roughly translate to:

```
from linecook.transforms.core import color_text, filter_line

LINECOOK_CONFIG = {
    'transforms': {
        'warn_color': color_text(' WARN ', color='yellow'),
        'error_color': color_text(' ERROR ', on_color='on_red'),
    },
    'recipes': {
        'logs': ['warn_color', 'error_color'],
        'default': [filter_line(' DEBUG '), 'error_color']
    },
}
```

You'll notice that `recipes` doesn't match the `recipes` in the second config file: Instead, the second file only overrode the `'default'` value in the first config file, but preseved the `'logs'` value.

CHAPTER 2

Developer's Guide

2.1 Prerequisites

The `linecook` package uses poetry for dependency management and distribution. You call install `poetry` using:

```
curl -sSL https://raw.githubusercontent.com/sdispater/poetry/master/get-poetry.py |␣
↪python
```

2.2 Setup

Clone from github:

```
git clone https://github.com/tonysyu/linecook.git
```

Install development requirements:

```
cd linecook
poetry install
```

For building the documentation locally, you'll also need to run:

```
poetry install --extras "docs"
```

2.3 Development

For local development, you'll also want to install pre-commit hooks using:

```
poetry run pre-commit install
```

By default, this will run the black code formatter on *changed* files on every commit. To run black on all files:

```
poetry run pre-commit run --all-files
```

2.4 Running tests

The test suite can be run without installing dev requirements using:

```
$ tox
```

To run tests with a specific Python version, run:

```
$ tox --env py36
```

You can isolate specific test files/functions/methods with:

```
tox PATH/TO/TEST.py
tox PATH/TO/TEST.py::TEST_FUNCTION
tox PATH/TO/TEST.py::TEST_CLASS::TEST_METHOD
```

2.5 Documentation

Documentation is built from within the docs directory:

```
cd docs
make html
```

After building, you can view the docs at `docs/_build/html/index.html`.

2.6 Debugging

It turns out that breakpoints are a bit tricky when processing streamed input. A simple `pdb.set_trace()` will fail, so you'll need to try one of the solutions described on StackOverflow[1],[2] (answer that worked for me).

Better yet, if you can use a single line of text can be passed in to test an issue, you can use the `--text` (`-t`) flag instead of piping text:

```
linecook <RECIPE> --text 'Line of text to test'
```

2.7 Release

A reminder for the maintainers on how to deploy.

- Update the version and push:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

[1] https://stackoverflow.com/questions/17074177/how-to-debug-python-cli-that-takes-stdin
[2] https://stackoverflow.com/questions/9178751/use-pdb-set-trace-in-a-script-that-reads-stdin-via-a-pipe

- Build release, deploy to PyPI, and clean

```
$ make release
$ make clean
```

CHAPTER 3

linecook

3.1 linecook package

3.1.1 Subpackages

linecook.config package

Submodules

linecook.config.core module

class linecook.config.core.**LineCookConfig**(*config_dicts*)
Bases: object

Configuration for linecook parsed from known configuration files.

transforms
Named transforms available for recipes.

Type dict

recipes
Named recipes, which are simply text transforms, or sequences of transforms, that are applied to each line of text.

Type dict

See load_config for a description of known configuration files, and hierarchical configuration works.

linecook.config.core.**load_config**()
Return LineCookConfig reduced from all known configuration files.

The following configuration files are loaded, in order:

- linecook.config.core

- `~/.linecook/config.py`
- `./.linecook/config.py`

Each file should define a dictionary named `LINECOOK_CONFIG` containing keys such as `transforms` and `recipes`.

Files loaded later (lower on the list) override values loaded earlier. Note that the overriding happens at the *second* level of dictionaries. For example, if `~/.linecook/config.py` is defined as:

```
from linecook.transforms.core import color_text

LINECOOK_CONFIG = {
    'transforms': {
        'warn_color': color_text(' WARN ', color='yellow'),
        'error_color': color_text(' ERROR ', on_color='on_red'),
    },
    'recipes': {
        'logs': ['warn_color', 'error_color'],
        'default': ['warn_color', 'error_color'],
    },
}
```

And then, `./.linecook/config.py` is defined as:

```
from linecook.transforms.core import filter_line

LINECOOK_CONFIG = {
    'recipes': {
        'default': [filter_line(' DEBUG '), 'error_color']
    },
}
```

The loaded result would roughly translate to:

```
LINECOOK_CONFIG = {
    'transforms': {
        'warn_color': color_text(' WARN ', color='yellow'),
        'error_color': color_text(' ERROR ', on_color='on_red'),
    },
    'recipes': {
        'logs': ['warn_color', 'error_color'],
        'default': [filter_line(' DEBUG '), 'error_color']
    },
}
```

You'll notice that `recipes` doesn't match the `recipes` in the second config file: Instead, the second file only overrode the `'default'` value in the first config file, but preseved the `'logs'` value.

linecook.config.parsers module

linecook.config.parsers.**collect_recipes**(*config_dicts*, *transforms_registry*)

Return recipe dictionary from a list of configuration dictionaries.

Parameters

- **config_dicts** (*list(dict)*) – Unparsed configuration dictionaries. For each dictionary, only use the 'recipes' value, which itself is a dictionary, where the keys are recipe names and values are lists of transform functions or transform names.

- **transforms_registry** (*dict*) – Dictionary containing named transform functions. See also `collect_tranforms`, which build this registry.

`linecook.config.parsers.`**collect_tranforms**(*config_dicts*)
Return transform dictionary from a list of configuration dictionaries.

Parameters **config_dicts** (*list(dict)*) – Unparsed configuration dictionaries. For each dictionary, this applies parsers registered with `register_transform_parser` that convert configuration data into named transform functions.

`linecook.config.parsers.`**get_value_from_each**(*key*, *dict_list*)
Return list of values for `key` in a list of dictionaries.

`linecook.config.parsers.`**parse_colorizers**(*colorizers_dict_seq*)
Return dictionary of transforms based on `colorizers` in config_dict.

This converts `colorizers` field in a configuration dict into color transforms. For example, take the following configuration:

```
'colorizers': {
    'warn_color': {
        'match_pattern': ' WARN ',
        'on_color': 'on_yellow',
    },
},
```

That configuration is parsed to return the transform:

```
from linecook.transforms.core import color_text

color_text(' WARN ', on_color='on_yellow')
```

`linecook.config.parsers.`**parse_transforms**(*transforms_dict_seq*)
Return dictionary of transforms based on `transforms` in config_dict.

All this really does is merge the transforms defined in multiple configuration dictionaries.

`linecook.config.parsers.`**register_transform_parser**(*config_type*)
Add parser to registry of known linecook config parsers.

Parameters **config_type** (*str*) – The config type that is parsed by this decorated function The resulting output will be stored in the parsed config under this name.

A config parser takes a sequence representing updated the an object containing the parsed data.

You can register the a parser with the same name multiple times, which will simply override older instances.

`linecook.config.parsers.`**resolve_recipe**(*recipe*, *transforms_registry*)

Module contents

linecook.recipes package

Submodules

linecook.recipes.dpkg_log module

linecook recipe for dpkg.log.

linecook.recipes.dpkg_log.**emphasize_dpkg_actions**()
Return transform that emphasizes packaging actions

linecook.recipes.python module

Example of linecook recipe for python code.

This is a toy example: Actual syntax highlighting isn't possible since linecook doesn't (easily) store state between different lines, which prevents proper highlighting of things like multi-line strings.

Module contents

linecook.transforms package

Submodules

linecook.transforms.core module

Text formatters match the signature of basic regex functions, taking a text/regex match pattern and an input string.

class linecook.transforms.core.**CountLines**(*line_template='{count_label} {line}', count_template='{count:>3}:', color_kwargs={'attrs': ['bold'], 'color': 'grey'}*)
Bases: object

Tranformation returning line of text with line count added.

reset()

linecook.transforms.core.**color_text**(*match_pattern='.*', color=None, on_color=None, attrs=None*)
Return color transform that returns colorized version of input string.

Parameters

- **color** (*str*) – Text color. Any of the following values grey, red, green, yellow, blue, magenta, cyan, white.
- **on_color** (*str*) – Background color. Any of the following values on_grey, on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white
- **attrs** (*list(str)*) – Text attributes. Any of the following values: bold, dark, underline, blink, reverse, concealed.

linecook.transforms.core.**delete_text**(*match_pattern*, *, *replacement=''*)

linecook.transforms.core.**filter_line**(*match_pattern*, *on_match=None*, *on_mismatch=None*)
Return transform that filters lines by match pattern.

If neither on_match or on_mismatch are given, return input string if it matches the given pattern. Otherwise, the input string is passed to those callback functions and the output is returned.

Parameters

- **on_match** (*callable*) – A text transform that is called with the input string if the string matches the match_pattern.

- **on_mismatch** (*callable*) – A text transform that is called with the input `string` if the string does not match the `match_pattern`.

`linecook.transforms.core.`**partition**(*match_pattern*, *on_match=None*, *on_mismatch=None*)
Return line partitioned by pattern and re-joined after transformation.

Parameters

- **on_match** (*callable*) – A text transform that is called with each substring that matches the `match_pattern`.
- **on_mismatch** (*callable*) – A text transform that is called with each substring that does not match the `match_pattern`.

`linecook.transforms.core.`**replace_text**(*match_pattern*, *replacement*)

`linecook.transforms.core.`**split_on**(*match_pattern*, *, *replacement='\n'*)

linecook.transforms.logging module

linecook.transforms.parse module

Transforms that parse data from text.

`linecook.transforms.parse.`**json_from_qs**(*flatten=True*)
Return tranform that outputs json string from query string.

Module contents

`linecook.transforms.`**delete_text**(*match_pattern*, *, *replacement=''*)

`linecook.transforms.`**split_on**(*match_pattern*, *, *replacement='\n'*)

3.1.2 Submodules

3.1.3 linecook.cli module

`linecook` cli to prepare lines of text for easy consumption.

`linecook.cli.`**build_parser**()

`linecook.cli.`**main**()

`linecook.cli.`**print_available_recipes**(*linecook_config*)

`linecook.cli.`**recipe_not_found_msg**(*recipe_name*)

`linecook.cli.`**run**(*args*)

3.1.4 linecook.parsers module

`linecook.parsers.`**create_regex_factory**(*format_string=None*, *regex_type=None*, *ignore_case=False*)
Return a `create_regex` function that compiles a pattern to a regex.

`linecook.parsers.`**resolve_match_pattern**(*pattern*)
Return a compiled regex, parsing known regex shorthands.

3.1.5 linecook.patterns module

linecook.patterns.**any_of**(**args*)
Return regex that matches any of the input regex patterns.

The returned value is equivalent to writing:

```
r'(<arg1>|<arg2>|...)'
```

linecook.patterns.**anything = '.*'**
Pattern matching any text

linecook.patterns.**bounded_word**(*string*)
Return regex that matches the input string as a bounded word.

The returned value is equivalent to writing:

```
r'\b<string>\b'
```

linecook.patterns.**date = '\\b\\d{4}-\\d{2}-\\d{2}\\b'**
Pattern matching calendar dates in ISO 8601 format (YYYY-MM-DD)

linecook.patterns.**day = '\\d{2}'**
Pattern matching a numeric day

linecook.patterns.**double_quoted_strings = '(?<!["\\w])"[^"]*"(?!["\\w])'**
Pattern matching strings surrounded by double-quotes

linecook.patterns.**exact_match**(*string*)
Return regex that matches the input string exactly.

The returned value is equivalent to writing:

```
r'^<string>$'
```

linecook.patterns.**exact_template = '^{}$'**
Template string to match text exactly (start-to-end)

linecook.patterns.**first_word = '^\\s*\\w+'**
Pattern matching first word

linecook.patterns.**indent = '^\\s*'**
Pattern matching indent at start of string

linecook.patterns.**month = '\\d{2}'**
Pattern matching a numeric month

linecook.patterns.**num_float = '\\b[+-]?(\\d*[.])?\\d+\\b'**
Pattern matching floating point number

linecook.patterns.**num_int = '\\b[+-]?\\d\\b'**
Pattern matching integers

linecook.patterns.**number = '(\\b[+-]?\\d\\b|\\b[+-]?(\\d*[.])?\\d+\\b)'**
Pattern matching integers or floats

linecook.patterns.**single_quoted_strings = "(?<!['\\w])'[^']*'(?!['\\w])"**
Pattern matching strings surrounded by single-quotes

linecook.patterns.**start = '^'**
Pattern matching start of string

linecook.patterns.**strings = '((?<![\'\\w])\'[^\']*\'(?![\'\\w])|(?<!["\\w])"[^"]*"(?!["\\w**
Pattern matching strings surrounded by single- or double-quotes

linecook.patterns.**time = '\\b(\\d{2}:\\d{2}(:\\d{2})?)\\b'**
Pattern matching numeric time

linecook.patterns.**time_ms = '\\b\\d{2}:\\d{2}:\\d{2}(,|.)\\d{3}\\b'**
Pattern matching numeric time with milliseconds

linecook.patterns.**whitespace = '\\s*'**
Pattern matching any whitespace

linecook.patterns.**word_template = '\\b{}\\b'**
Template string to match text surrounded by word boundaries

linecook.patterns.**year = '\\d{4}'**
Pattern matching a numeric year

3.1.6 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

P

R

S

T

W

Y