snakeparse documentation

Release 0.1.1-dev

Nils Homer

Apr 19, 2018

Contents

Author Nils Homer

Date Apr 19, 2018

Version 0.1.1-dev

Making Snakemake workflows into full-fledged command line tools since 1999.

CHAPTER 1

Documentation Contents

1.1 Snakeparse API

1.1.1 API

Snakeparse: command-line parsing library for Snakemake

This module allows multiple Snakemake workflows to be combined into one command line interface. Arguments to Snakemake and workflow-specific arguments are supported.

This module is inspired by tool-chains like fgbio, Picard, and samtools, that:

- support multiple tools which all can be specified on the same command line
- combines the argument parsing from all tools into one command line
- supports dispatching the tool-specific arguments to sub-parsers
- enable argument parsing for the tool-chain, that can propogate to all tools; in this case, arugenets for Snakemake.

The following is a minimumal usage example for how to use the API:

```
>>> SnakeParse(args=sys.argv[1:]).run()
```

The magic comes from creating a configuration object (*SnakeParseConfig*) that configures the paths to where the Snakemake files (snakefiles) live, as well as various options for how workflows are displayed on the command line. Once the configuration object has been created, it's as simple as:

```
>>> SnakeParse(args=sys.argv[1:], config=config)
```

The given arguments may contain the argument separator `--`. All arguments prior will be passed to Snakemake, while all arguments after will be passed to the specified workflow. Which workflow to run is determined as follows:

1. If the argument separator is present, then if there is only one workflow configured, use that one, otherwise, assume the name of the workflow is specified immediate after the argument separator.

2. If the argument separator is not present, search for the first argument that matches a known workflow name.

If no workflows are configured, but the `-s/--snakefile` option is given before the argument separator, then this workflow is added to the list of workflows, and that workflow will be executed.

For a more typical example, suppose the path to the the snakefiles is `~/snakemake-workflows`, then the following will work:

```
>>> config = SnakeParseConfig(snakefile_globs='~/snakemake-workflows/*')
>>> SnakeParse(args=sys.argv[1:], config=config).run()
```

Below are some example argument lists. Suppose the workflow named `Example` has a single required command line option `--message` that takes a message to be printed.

For running a single workflow:

```
>>> args = ['--snakefile', '/path/to/snakefile', '--', '--message', 'Hello!']
```

If a single workflow has been added via *SnakeParseConfig* object:

```
>>> args = ['Example', '--message', 'Hello!']
```

Alternatively, the workflow name can be omitted when only one workflow has been configured:

```
>>> args = ['--', '--message', 'Hello!']
```

If multiple workflows are configured, then the name must be explictly used.

In some cases, the options to Snakemake take multiple values, so it is ambiguous where the arguments to Snakemake end and the arguments to the workflow begin. Use the `--` argument to explicitly seperate the two lists. The workflow name should be immediately after the `--` seperator:

```
>>> args = ['--force-run', 'rule-1', 'rule-2', '--', 'Example', '--message', 'Hello!
↪']]
```

In the above example, the arguments `['--force-run', 'rule-1', 'rule-2']` are passed to Snakemake, while the arguments `['--message', 'Hello!']` are passed to the SnakeParser for the Example workflow.

Ther are two ways for your snakefile source to receive the parsed arguments: (1) define a concrete subclass of *SnakeParse*, or (2) define a method `snakeparser(**kwargs)` that returns a concrete subclass of *SnakeParse*. For convenience when implementing parsing using the argparse module, the class *SnakeArgumentParser* can be used for (1), while the method *argparser()* can be used for (2). For the `Example` workflow above, an example implementation with a concrete class definition is as follows:

```
>>> from snakeparse.api import SnakeArgumentParser
>>> class Parser(SnakeArgumentParser):
...     def __init__(self, **kwargs):
...         super().__init__(**kwargs)
...         self.parser.add_argument('--message', help='The message.', required=True)
```

Alternatively, a method can be defined in 'example_snakeparser.py'

```
>>> from snakeparse.parser import argparser
>>> def snakeparser(**kwargs):
...     p = argparser(**kwargs)
...     p.parser.add_argument('--message', help='The message.', required=True)
...     return p
```

The module contains the following public classes:

- ***SnakeParser* – The abstract base class that implements the workflow** specific argument parsing. This parser will be invoked by SnakeParse prior to running Snakemake, to ensure that the command line arguments are specified correctly. This parser is likely also used in the Snakemake file to re-instantiate the parsed arguments.
- ***SnakeArgumentParser* – The abstract base class to help argument** parsers that use python's argparse module.
- *SnakeParseException* – The exception raised by this module.
- ***SnakeParseWorkflow* – A container class for basic meta information** about a supported workflow, including to but not limited to the name displayed on the command line, the paths to the snakefile and SnakeParse file, a workflow group to which this workflow belongs, and a short description to display on the commad line.
- ***SnakeParseConfig* – The class used to configure SnakeParse. In** particular, where workflows are located (if they are to be discovered), definitions for the workflow (if they are to be explicitly defined), where SnakeParse files live (either generally or relative to the Snakemake files), the names of the workflow groups, and other miscellaneous options.
- ***SnakeParse* – The main entry point for command-line parsing for** Snakemake. The configuration for SnakeParse will be optionally loaded, then the workflow to run will be parsed, then the workflow arguments will be parsed, and finally the workflow will be run along with all the Snakemake specific arguments.

All other classes in this module are considered implementation details.

class snakeparse.api.**SnakeArgumentParser**(***kwargs: typing.Any*) → None

The abstract base class to help argument parsers that use python's argparse module. Keyword arguments will be passed to the argument parser's constructor.

parse_args(*args: typing.List[str]*) → typing.Any

Parses the command line arguments.

parse_args_file(*args_file: pathlib.Path*) → typing.Any

Parses command line arguments from an arguments file

print_help(*file: typing.Union[typing.IO[str], NoneType] = None*) → None

Prints the help message

class snakeparse.api.**SnakeParse**(*args: typing.List[str] = [], config: typing.Union[_ForwardRef('SnakeParseConfig'), NoneType] = None, debug: bool = False, file: typing.IO[str] = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*) → None

The main entry point for command-line parsing for Snakemake.

Keyword Arguments

- **args** (*List[str]*) – The list of command line arguments.
- **config** (*Optional['SnakeParseConfig']*) – Optionally a SnakeParse configuration object.
- **debug** (*bool*) – Print extra debuggin information in the parser's help message.
- **file** (*TextIOWrapper*) – The file to write any error or help messages, defaults to sys.stdout.

run() → None

Execute the Snakemake workflow

static usage_short(*prog: str = 'snakeparse', workflow_name: str = None*) → str

A one line usage to display at the top of any usage or help message.

class snakeparse.api.**SnakeParseConfig**(*config_path: typing.Union[pathlib.Path, NoneType] = None, prog: str = None, snakemake: typing.Union[pathlib.Path, NoneType] = None, name_transform: str = None, parent_dir_is_group_name: bool = True, workflows: typing.Dict[str, _ForwardRef('SnakeParseWorkflow')] = OrderedDict(), groups: typing.Dict[str, str] = OrderedDict(), snakefile_globs: typing.Union[typing.List[str], NoneType] = []*) → None

The class used to configure SnakeParse.

Keyword Arguments

- **config_path** (*Optional[Path]*) – The path to the SnakeParse configuration file, in JSON, YAML, or HOCON format. Details of its contents are described below.
- **prog** (*str*) – The name of the tool-chain.
- **snakemake** (*Optional[Path]*) – The optional path to the Snakemake executable.
- **name_transform** – An optional method to transform the basename of a workflow's snakefile to the canonical name to use on the command line.
- **parent_dir_is_group_name** (*bool*) – True to use the parent directory of the workflow's snakefile as the workflow's group name. Only applied when searching directories for snakefiles, or when group name is not explicitly given.
- **workflows** (*Dict[str, 'SnakeParseWorkflow']*) – Optionally, the list of workflows as SnakeParseWorkflow objects.
- **groups** (*Dict[str, str]*) – Optionally, one or more key-value pairs, with the key being the canonical workflow group name, and the value being a description for that group to display on the command line.
- **snakefile_globs** (*Optional[List[str]]*) – Optionally, or more glob strings specifying where snakefile files can be found.

NB: the values in the configuration file take precedence over the keyword arguments. In particular, the configuration file may override Worfklows given in the 'workflows' keyword argument.

The following are configuration paths (in HOCON paths) allowed in the configuration file:

- snakemake – the optional path to the Snakemake executable.
- prog – the optional name of the tool-chain.
- name_transform – alias for a built-in method to produce the workflow's name (see the similarly named keyword argument). Either 'snake_to_camel' or 'camel_to_snake' for converting from Snake case to Camel case, or vice versa.
- parent_dir_is_group_name – optional; see the similarly named keyword argument.
- workflows – optionally, a list of configuration objects, one per workflow specified. They object key should be the canonical workflow name to be displayed on the command line, with a dictionary of key value pairs specifying the wofklow configuration with the same names as SnakeParseWorkflow (snakefile, group, and description). Only the snakefile key-value pair is required.
- groups - optional; see the similarly named keyword argument.
- snakefile_globs – optional; see the similarly named keyword argument.

add_group(*name: str*, *description: str*, *strict: bool = True*) → snakeparse.api.SnakeParseConfig
Adds a new group with the given name and description. If strict is `True`, then no group with the same name should already exist.

add_snakefile(*snakefile: pathlib.Path*) → snakeparse.api.SnakeParseWorkflow
Adds a new workflow with the given snakefile. A workflow with the same name should not exist.

add_workflow(*workflow: snakeparse.api.SnakeParseWorkflow*) → snakeparse.api.SnakeParseWorkflow
Adds the workflow to the list of workflows. A workflow with the same name should not exist.

static config_parser(*usage: str = '==SUPPRESS=='*) → argparse.ArgumentParser
Returns an `ArgumentParser` for the configuration options

static name_transfrom_from(*key: str*) → typing.Callable[str, str]
Returns the built-in method to format the workflow's name. Should be either 'snake_to_camel' or 'camel_to_snake' for converting from Snake case to Camel case, or vice versa.

static parser_from(*workflow: snakeparse.api.SnakeParseWorkflow*) → snakeparse.api.SnakeParser
Builds the SnakeParser for the given workflow

exception snakeparse.api.**SnakeParseException**
The exception raised by classes in this module.

class snakeparse.api.**SnakeParseWorkflow**(*name: str*, *snakefile: pathlib.Path*, *group: typing.Union[str, NoneType] = None*, *description: typing.Union[str, NoneType] = None*) → None
A container class for basic meta information about a workflow to be included on the command line.

Keyword Arguments

- **name** (*str*) – The canonical name of the workflow displayed on the command line.
- **snakefile** (*Optional[str]*) – The path to the snakefile file.
- **group** (*(Optional[str]):*) – The name of the workflow group, used to group workflows on the command line.
- **description** (*Optional[str]*) – A short description of the workflow, used when listing the workflows.

class snakeparse.api.**SnakeParser** → None
The abstract base class for implementing the workflow specific argument parsing.

description
A short description of the workflow, used when listing the workflows.

group
The name of the workflow group to which this group belongs.

parse_args(*args: typing.List[str]*) → typing.Any
Parses the command line arguments.

parse_args_file(*args_file: pathlib.Path*) → typing.Any
Parses command line arguments from an arguments file

parse_config(*config: dict*) → typing.Any
Parses arguments from a Snakemake config object. It is assumed the arguments are contained in an arguments file, whose path is stored in the config with key `SnakeParse.ARGUMENT_FILE_NAME_KEY`.

print_help(*file: typing.Union[typing.IO[str], NoneType] = None*) → None
Prints the help message

snakeparse.parser.**argparser**(***kwargs: typing.Any*) → snakeparse.api.SnakeArgumentParser
Returns an SnakeParser that has an initialized member variable parser of type `ArgumentParser`. The keyword arguments are passed to the constructor of `ArgumentParser`.

1.2 Examples

1.2.1 Snakeparse on the Command Line

To get started, run *snakeparse –help*.

Example

1. Using argparse and a custom method named snakeparser.
2. Using argparse and a concrete sub-class of SnakeParser.

For more examples, see this link.

The example below is from (1).

Motivation

Consider this simple Snakemake file (snakefile) that has a required configuration option:

```
message = config['message']

rule all:
    input:
        'message.txt'

# A simple rule to write the message to the output
rule message:
    output: 'message.txt'
    shell: 'echo {message} > {output}'
```

You would need to run snakemake with the `--config` option:

```
$ snakemake --snakefile </path/to/snakefile> --config message='Hello World!'
```

If you forget to add the correct key/value pairs with the `--config` option, you'll get a `KeyError` exception, which is not user-friendly to non-programmers. At that point, you're out of luck to see all the various required and optional config key/value pairs without examining the snakefile (i.e. you want to see a help message). Have fun adding each configuration option one-by-one and gleaning their meaning. Even examining the source, there needs to be clear documentation within your snakefile for each argument for the user to examine. Why can't we just use argparse as we normally would for our command-line python scripts?

Furthermore, if you have multiple snakefiles, setting the `--config` key/value pairs can get quite painful, notwithstanding the fact you need to specify the path to the specific snakefile your interested in each time. Why can't we put all the snakefiles in one place, and have an easy way to specify which to run on the command line? So many other command-line tools do it (ex. bwa, samtools, fgbio, Picard), and even other workflow software do it (ex. dagr), why can't we do it?

This is why I wrote Snakeparse.

Setup

From examples/argparse/method/write_message.smk.

Modify the above snakefile by prepending the following:

```
# Import the parser from snakeparse
from snakeparse.parser import argparser

def snakeparser(**kwargs):
    ''' The method that returns the parser with all arguments added. '''
    p = argparser(**kwargs)
    p.parser.add_argument('--message', help='The message.', required=True)
    return p

# Get the arguments from the config file; this should always succeed.
args = snakeparser().parse_config(config=config)

# NB: you could use `args.message` directly.
message = args.message
```

Execution

Snakeparse Command Line Execution

You can run the installed `snakeparse` utility as follows:

```
$ snakeparse --snakefile examples/argparse/method/write_message.smk -- --message
↪'Hello World!'`
```

or

```
$ snakeparse --snakefile-globs examples/argparse/method/*smk -- WriteMessage --
↪message 'Hello World!'`
```

Programmatic Execution

```
>>> config = SnakeParseConfig(snakefile_globs='~/examples/argparse/method/*smk')
>>> SnakeParse(args=sys.argv[1:], config=config).run()
```

or alternatively `SnakeParse` accepts leading configuration arguments:

```
>>> args = ['--snakefile-globs', '~/examples/argparse/method/*smk'] + sys.argv[1:]
>>> SnakeParse(args=args, config=config).run()
```

1.3 Installation

Python 3.6 or higher is required.

Snakemake 4.4.0 or higher is required.

To clone the repository: `git clone https://github.com/nh13/snakeparse.git`.

To install locally: `python setup.py install`.

The command line utility can be run with `snakeparse`

1.3.1 Conda Recipe

See the conda-recipe on Bioconda.

1.3.2 Testing

To run tests, run `coverage run -m unittest discover -s src`

To obtain test coverage, run `codecov`.

1.4 Releases

1.4.1 Release 0.0.1

This is the first release of snakeparse.

CHAPTER 2

Why Snakeparse?

- I wanted a single command line utility that could organize and execute multiple Snakemake workflows
- I wanted to have my workflow-specific arguments be parsed on the command line (ex. with argparse)
- I wanted an API or library to configure how to group and organize the workflows, and how to display them on the command line.
- I didn't want to write this library more than once.

CHAPTER 3

Demo

CHAPTER 4

Installation

To install the latest release, type:

```
git clone https://github.com/nh13/snakeparse.git
python setup.py install
```

See the Installation notes for details.

4.1 Indices and tables

Contents:

- genindex
- modindex
- search

4.2 References

See also:

Snakemake http://snakemake.readthedocs.io

The python language http://www.python.org

Python Module Index

S

Index