Toolbox Documentation

Release 0.5.1

Jeffrey Slort

January 21, 2016

Contents

Contents:

CHAPTER 1

Toolbox

1.1 What is Toolbox ?

Toolbox is a framework to manage a collection of tools. A tool can be anything from a shell script to a python package.

Basically toolbox extends the python argparse.ArgumentParser to enable importing tools from multiple locations in a single executable. Toolbox provides some easy default tools to create , install, uninstall or list (available) tools.

A custom tool needs to implement 2 methods: * prepare_parser : which prepares an argparse.ArgumentParser * execute : which is the main entry point of your tool

Besides the wrapper around argument parsing toolbox as a framework provides some easy to add extra's like persisted configuration, usage of tools within tools and no-configuration logging.

An example of a custom tool:

```
from toolbox.plugin import ToolboxPlugin
from toolbox.mixins import RegistryMixin, ConfigMixin, LogMixin

class CustomPlugin(RegistryMixin, ConfigMixin, LogMixin, ToolboxPlugin)
        name = "custom"
        description = "This is a custom plugin that prints a string"

        def prepare_parser(self,parser):
                parser.add_argument('printable', help="string to print")

        def execute(args):
                # LogMixin
                logger = self.get_logger()
                logger.info("printing {}".format(args.printable)

                # ConfigMixin
                config = self.get_config()
                config['first_print'] = args.printable

                # RegistryMixin
                registry = self.get_registry()
                other_plugin = registry.get_plugin('other')

                print(args.printable)
```

For more info on all the tools the toolbox framework provides check the complete documentation!

- Free software: ISC license

- Documentation: https://tool-box.readthedocs.org.

1.2 Features

- Integrate your own shell scripts etc. with a single command
- Easily integrate your existing python tools with toolbox by wrapping them in ToolboxPlugin class
- Add persisted configuration to your tools
- Use other tools within your own tools
- search tools in the toolbox
- install other tools from PyPI/github with the toolbox commandline

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

At the command line:

```
$ easy_install tool-box
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv toolbox
$ pip install tool-box
```

CHAPTER 3

Usage

3.1 Install a Tool

To install a plugin and register it to the toolbox you can run:

```
tbox install [--external] tool
```

This searches the current working directory for a match , if it can't find it will install it from PyPI or git using pip If you already have a toolbox tool installed and want to register it to the toolbox just add the external flag

3.2 Use a Tool

Using a tool is really simple every installed tool will be available as a subcommand of the toolbox. To use the 'example' tool you execute:

```
tbox example
```

3.3 Register a shell command

already have an awesome script or just want to alias a complex command run the following:

```
tbox create -t shell -i example
```

This will create a new Toolbox tool named example and install it afer asking some basic questions.

3.4 Create a new Tool

To use the Toolbox we need a Tool. Let's create one:

```
tbox create example
```

This sets up the current working directory with an simple template of a basic Tool.

Install the newly created tool as a dev tool by:

```
tbox install --dev ./example
```

A custom package or module needs to expose a 'Plugin' variable or class definition suffixed by 'Plugin' in the main namespace To check if this worked check if your new tool is listed:

```
tbox list
```

3.5 Customizing

Our new tool is not very usefull yet, but that's about to change! An tool should always subclass the toolbox.plugin.ToolboxPlugin . Which essentially means it needs to implement an prepare_parser and execute method. the prepare parser get an instance of an argparse.ArgumentParser. This method sets up the tool for usage on the commandline

the execute method is the main entry point for the commandline and should accept an argparse Namespace.

3.6 Adding to the Mix

The toolbox.mixin module provides some usefull mixins to extend the new custom Tool with basic functionality For example by adding the ConfigMixin to the new tool class the tool gets access to a special plugin dictionary that is persisted between usages.

Tools can use other tools by adding the RegistryMixin which provides access to the toolbox registry from wich other tools can be loaded.

There is also an LogMixin to provide a no-config python logging logger instance.

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/jeff-99/toolbox/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

Toolbox could always use more documentation, whether as part of the official Toolbox 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/jeff-99/toolbox/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 *toolbox* for local development.

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

```
$ git clone git@github.com:your_name_here/toolbox.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 toolbox
$ cd toolbox/
$ 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 toolbox 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/jeff-99/toolbox/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_toolbox
```

CHAPTER 5

Credits

5.1 Development Lead

- Jeffrey Slort <j_slort@hotmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.6.0 (2016-02-01)

- added pip install of tool dependencies by having an requirements.txt

6.2 0.5.0 (2016-01-02)

- added logs tool
- added tests for the registry , scanner and config

6.3 0.4.0 (2016-01-01)

- First stable release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search