withenv Documentation

Release 0.7.0

Eric Larson

Apr 21, 2017

Contents

Withenv is a tool to help manage your environment variables consistently.

The idea behind withenv is to have an easy way set environment variables prior to running some program that depends on them, leaving your shell in a sane state.

Contents:

CHAPTER 1

withenv

We use environment variables all the time, but they can be painful to maintain because a shell is sticky. It is too easy to set an environment variable in your shell, only to have that variable stick around when you change projects.

withenv aims to help this problem by providing a simple way to prefix commands targeting YAML files that will be added to the environment prior to the command running.

See the docs for more info.

- Free software: BSD license
- Documentation: https://withenv.readthedocs.org

CHAPTER 2

Installation

At the command line:

```
$ easy_install withenv
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv withenv
$ pip install withenv
```

This will install the *we* command line tool.

CHAPTER 3

Usage

The *withenv* package installs the *we* executable. Here is the basic usage.

```
$ we --env foo.yml printenv
```

The YAML in *foo.yml* gets loaded and applied to the environment. If the value already exists in the environment, that value will be overwritten.

You can also use a directory of YAML files.

```
$ we --dir myenv printenv
```

The files will be applied to the environment in alphabetical order.

You can shorten the flags as well as mixing files and directories.

```
$ we -e foo.yml -d bar -e baz.yml printenv
```

Each flag will be applied in order from left to right.

YAML Format

You can use a hash or list of hashes in your YAML file. For example:

```
---
FOO: bar
BAR: hello $FOO
```

It is not recommended to use a hash in this format because the order cannot be gauranteed, although, it will probably work just fine. If you need explicit ordering within your file, use a list of hashes.

```
---
- FOO: bar
- BAR: hello $FOO
```

Here we see the *$FOO* variable is used within the value of *$BAR*.

Environment Files

Withenv also makes an effort to include environment files. Specifically, you can include a file that use the format:

```
export $VARNAME=$VALUE
```

Each line is parsed as an entry. This can be a typical shell script as lines that don't start with *export* will be ignored. With that in mind, functions defined in the script will not be available.

Command Substitutions

Sometimes you want to replace a variable based on the result of a command. Say for example, you wanted to grab a value from a chef environment. We can use the knife and jq to grab the value and inject into our environment value.

```
---
- CHEF_ENV: dev
- TOKEN: "`knife environment show $CHEF_ENV -Fj | jq --raw-output .default_attributes.
↪token`"
```

The knife command will go to our chef server and grab the environment's configuration and output it as JSON. This output is piped to the *jq* command where we are able to use JSONPath to grab the field value we need. The *–raw-output* will ensure we don't have any quotes around the value.

We could then use this in a commmand.

```
$ we -e token.yml curl -H 'X-Auth-Token: $TOKEN' http://example.com/api/
```

Currently, *withenv* supports this dynamic substitution when the value starts and endswith a backtick.

Creating an Alias

Sometimes you'll find that your environment is composed of a suite of details. Say for example, you were deploying an application via some script that uses environment variables to choose what region, cloud account and process to run.

```
$ we -d envs/apps/foo \
     -e envs/acct/dev.yml \
     -e envs/regions/us-east \
     -E TAG=foo
     ./create-app-server
```

We can create an alias for this by creating an alias YAML file.

```
# myalias.yml
---
- directory: envs/apps/foo
- file: envs/acct/dev.yml
- file: envs/regions/us-east
- override: "TAG=foo"
```

We can then run our command with a shortened *we* command.

```
$ we -a myalias create-app-server
```

Loading Defaults

Withenv will look for a default alias file called *.werc*. The *we* command will look in the current directory and walk the filesystem until it finds a *.werc* file. If it finds a *.werc*, it will load it as an alias file prior to any command line arguments. If no *.werc* is found, *we* continues normally.

For example, lets say that you had a some projects for different clients. Each client provided credentials to a cloud account and you want to use the specific client when running commands.

The *.werc* might look like this:

```
# .werc
---
- file: client.yml
- file: ~/projects/clients/$CLIENT/creds.yml
```

The *client.yml* would add the *$CLIENT* env var. Now you could see what instances your client has running.

```
$ we ec2-describe-regions
# or for rackspace
$ we rack servers instance list
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/ionrock/withenv/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.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

withenv could always use more documentation, whether as part of the official withenv docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ionrock/withenv/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 :)

Get Started!

Ready to contribute? Here's how to set up *withenv* for local development.

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

```
$ git clone git@github.com:your_name_here/withenv.git
```

3. Install your local copy into a virtualenv. This is how you set up your fork for local development:

```
$ cd withenv/
$ make bootstrap
```

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:

```
$ make test-all
```

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.

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, and 3.4, and for PyPy. Check https://travis-ci.org/ionrock/withenv/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests/test_my_test.py::TestClass::test_func
```

CHAPTER 5

Credits

Development Lead

- Eric Larson <eric@ionrock.org>

Contributors

None yet. Why not be the first?

CHAPTER 6

Indices and tables

- genindex
- modindex
- search