conda-devenv Documentation

Release 2.1.2.dev4+g43af2fe

Edison Gustavo Muenz

Oct 13, 2020

Contents

1 Overview **1**

2 Contents: **3**

2.1 conda-devenv 3
2.2 Installation 4
2.3 Usage 5
2.4 Contributing 9
2.5 CHANGELOG 11

3 Indices and tables **15**

CHAPTER 1

Overview

With `conda-devenv`, we help you manage software dependencies across Linux, Windows, and macOS operating systems using a *single conda dependency file* `environment.devenv.yml` instead of multiple `environment.yml` files.

See example below:

```
name: mylib

dependencies:
  - cmake
  - eigen
  - pip:
    - sphinx
  - gxx_linux-64=7.3.0  # [linux]
  - ccache              # [unix]
  - clcache             # [win]

environment:
  CPATH:
    - $CONDA_PREFIX/include           # [unix]
    - $CONDA_PREFIX/Library/include   # [win]

  LD_LIBRARY_PATH: $CONDA_PREFIX/lib  # [unix]
```

By executing:

```
conda devenv
```

in the root of the project directory, an `environment.yml` file is produced. We can now activate the conda environment `mylib` as follows:

```
conda activate mylib
```

This will not only resolve the dependencies for the current operating system, but also set environment variables `CPATH` and `LD_LIBRARY_PATH` to the list of paths specified in the `environment.devenv.yml` file. Note, however,

that `LD_LIBRARY_PATH` will only be set in Linux and macOS (Unix) whereas `CPATH` will be set differently in Linux and macOS compared to Windows.

Continue reading to learn more about the full capabilities of `conda-devenv`!

CHAPTER 2

Contents:

2.1 conda-devenv

`conda-devenv` is `conda` extension to work with multiple projects in development mode.

It works by processing `environment.devenv.yml` files, similar to how `conda env` processes `environment.yml` files, with this additional features:

- Jinja 2 support: gives more flexibility to the environment definition, for example making it simple to conditionally add dependencies based on platform.
- `include` other `environment.devenv.yml` files: this allows you to easily work in several dependent projects at the same time, managing a single `conda` environment with your dependencies.
- **Environment variables**: you can define a `environment:` section with environment variables that should be defined when the environment is activated.

Here's a simple `environment.devenv.yml` file:

```
{% set conda_py = os.environ.get('CONDA_PY', '35') %}
name: web-ui-py{{ conda_py }}

includes:
  - {{ root }}/../core-business/environment.devenv.yml

dependencies:
  - gcc  # [linux]
```

(continues on next page)

(continued from previous page)

```
environment:
  PYTHONPATH:
    - {{ root }}/src
  STAGE: DEVELOPMENT
```

To use this file, execute:

```
$ cd ~/projects/web-ui
$ conda devenv
> Executing: conda env update --file environment.yml --prune
Fetching package metadata .........
Solving package specifications: ..........
Linking packages ...
[      COMPLETE      ]|#########################| 100%
#
# To activate this environment, use:
# > source activate web-ui-py35
#
# To deactivate this environment, use:
# > source deactivate web-ui-py35
#

$ source activate web-ui-py35

$ env PYTHONPATH
/home/user/projects/web-ui/src

$ echo $STAGE
DEVELOPMENT
```

2.1.1 Documentation

https://conda-devenv.readthedocs.io.

2.1.2 Development

Please see CONTRIBUTING.

2.1.3 License

Free software: MIT license

2.2 Installation

To install conda-devenv, run this command in your terminal:

```
$ conda install conda-devenv -c conda-forge
```

2.2.1 From sources

The sources for conda-devenv can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ESSS/conda-devenv
```

Or download the tarball:

```
$ curl  -OL https://github.com/ESSS/conda-devenv/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

2.3 Usage

`conda-devenv` works by creating `conda` environments from definitions found in `environment.devenv.yml` files.

`environment.devenv.yml` files are similar to `environment.yml` files processed by `conda env`, with additional support for:

2.3.1 Jinja2

You can use Jinja 2 syntax at will, which lets you do all sort of things like conditionally add dependencies based on the current platform, change the default environment name based python version, etc. For example:

```
{% set conda_py = os.environ.get('CONDA_PY', '35') %}
name: web-ui-py{{ conda_py }}

dependencies:
  - boost
  - cmake
  - gcc     # [linux]
  - ccache  # [not win]
  - clcache # [win] Windows has clcache
```

Note that in the example above, we are able to define dependency requirements that are specific to Linux, macOS, and Windows (e.g., `ccache` is needed in Linux and macOS, whereas `clcache` is needed in Windows). This is one of the most useful capabilities of `conda-devenv`.

Tip: You can actually write any Python expression that evaluates to a boolean inside the brackets following the YAML comment mark `#`. For example, `# [linux]` could be replaced with `# [sys.platform.startswith('linux')]`. This is heavly inspired by conda-build's preprocessing selectors.

The following variables are available in the Jinja 2 namespace:

root	The full path to the directory containing the environment.devenv.yml file.
os	The standard Python module object os obtained with import os.
sys	The standard Python module object sys obtained with import sys.
platform	The standard Python module object platform obtained with import platform.
x86	True if the system architecture is x86, both 32-bit and 64-bit, for Intel or AMD chips.
x86_64	True if the system architecture is x86_64, which is 64-bit, for Intel or AMD chips.
linux	True if the platform is Linux.
linux32	True if the platform is Linux and the Python architecture is 32-bit.
linux64	True if the platform is Linux and the Python architecture is 64-bit.
osx	True if the platform is macOS.
unix	True if the platform is either macOS or Linux.
win	True if the platform is Windows.
win32	True if the platform is Windows and the Python architecture is 32-bit.
win64	True if the platform is Windows and the Python architecture is 64-bit.
is_included	True if the current file is processed because it was included by another file.

Using environment variables in the devenv file

Environment variables that are defined when calling conda devenv can be read normally using os.environ, but for convenience and better error messages, you can use the get_env function:

```
dependencies:
  - python ={{ get_env("PY") }}
```

This will raise an error if PY is not defined. Alternatively, it is also possible to set a default value:

```
dependencies:
  - python ={{ get_env("PY", default="3.6") }}
```

You can also provide a list of allowed or supported values:

```
dependencies:
  - python ={{ get_env("PY", valid=["3.6", "3.7"]) }}
```

This will raise an error if PY is set to a different value.

Note: Environment variables can also be set with the -e/--env-var command line option, see the *Command-line reference* section.

Checking minimum conda-devenv version

If your environment.devenv.yml files make use of features available only in later conda-devenv versions, you can specify a minimum version using the min_conda_devenv_version function at the top of your file:

```
{{ min_conda_devenv_version("1.1") }}
name: web-ui
```

If users are using an old version, they will get then an error message indicating that they should update their conda-devenv version.

It is recommended to use this setting to avoid confusing errors of users updating your software when new conda-devenv features are used.

Note: Unfortunately this feature was added in `conda-devenv 1.1`, so `1.0` users will get a more cryptic message about `min_conda_devenv_version` not being defined.

2.3.2 Environment Variables

It is possible to define environment variables that should be configured in the environment when activated.

```
environment:
  PATH:
    - {{ root }}/bin
  PYTHONPATH:
    - {{ root }}/source/python
  DB_LOCATION: https://localhost/dev
```

Environment variables defined in *list form* (like `PATH` and `PYTHONPATH` above) will **append** to existing environment variables with the values found in the `.devenv.yml` file, using the appropriate separator for the platform (`:` on Linux/OSX and `;` on Windows).

Environment variables defined as a single string (like `DB_LOCATION` above) will **overwrite** an existing environment variable with the value from the `.devenv.yml` file.

`conda-devenv` restores the variables of the environment to their original state upon deactivation.

2.3.3 Includes

It is possible to use *include* directives to include one or more `environment.devenv.yml` files. This merges all `dependencies` and `environment` definitions into a single environment, which makes it a good solution to work in one or more repositories in development mode.

For example:

`/home/user/projects/core/environment.devenv.yml`:

```
name: core
dependencies:
  - numpy
  - pandas
  - pytest
  - invoke
environment:
  PYTHONPATH:
    - {{ root }}/source/python
  DB_LOCATION: https://localhost/dev  # [not is_included]
```

`/home/user/projects/web-ui/environment.devenv.yml`:

```
name: web-ui
includes:
  - {{ root }}/../core/environment.devenv.yml
dependencies:
  - flask
  - jinja2
environment:
  PYTHONPATH:
```

(continues on next page)

(continued from previous page)

```
    - {{ root }}/source/python
  PATH:
    - {{ root }}/bin
  DB_LOCATION: https://localhost/dev
```

In this setup, all the user has to do is executing `conda devenv`:

```
$ cd ~/projects/web-ui
$ conda devenv
```

This will create a `conda` environment named `web-ui` merging all the dependencies and environment variables defined in both files.

However, the same environment variable defined as a single string (like `DB_LOCATION` above) in both files will raise an error unless it is not allowed to 'pass through' using the `# [not is_included]` selector above as an example. In other words, an 'overwrite' situation is not allowed between files.

2.3.4 How it works

Here's how `conda-devenv` works behind the scenes:

1. Generate an `environment.yml` file in the same directory as the `environment.devenv.yml` file. The generated `environment.yml` should **not** be added to VCS.
2. Call `conda env update --prune --file environment.yml`.
3. Generate `devenv-activate{.sh,.bat}` and `devenv-deactivate{.sh,.bat}` scripts in `$PREFIX/etc/conda/activate.d` and `$PREFIX/etc/conda/deactivate.d` respectively which will set/unset the environment variables.

2.3.5 Command-line reference

Default options

- `conda-devenv` creates a file name `environment.yml` at the same directory of the `environment.devenv.yml` file.

Options

```
$ conda devenv --help

usage: conda-devenv [-h] [--file [FILE]] [--name [NAME]] [--print]
                    [--print-full] [--no-prune] [--output-file [OUTPUT_FILE]]
                    [--quiet] [--env-var ENV_VAR] [--verbose] [--version]

Work with multiple conda-environment-like yaml files in dev mode.

optional arguments:
  -h, --help            show this help message and exit
  --file [FILE], -f [FILE]
                        The environment.devenv.yml file to process. The
                        default value is 'environment.devenv.yml'.
  --name [NAME], -n [NAME]
```

(continues on next page)

(continued from previous page)

```
                        Name of environment.
  --print               Prints the rendered file as will be sent to conda-env
                        to stdout and exits.
  --print-full          Similar to --print, but also includes the
                        'environment' section.
  --no-prune            Don't pass --prune flag to conda-env.
  --output-file [OUTPUT_FILE]
                        Output filename.
  --quiet               Do not show progress
  --env-var ENV_VAR, -e ENV_VAR
                        Define or override environment variables in the form
                        VAR_NAME or VAR_NAME=VALUE.
  --verbose, -v         Use once for info, twice for debug, three times for
                        trace.
  --version             Show version and exit
```

`--file`

The input file to be processed

`--print`

Prints the contents of the generated file and exits.

`--no-prune`

Don't pass the `--prune` flag when calling `conda env update`

`--output-file`

Specifies the `conda-env` file which will be created.

`--env-var`

Define or override environment variables in the form `VAR_NAME` or `VAR_NAME=VALUE`. Can be used multiple times for different variables.

2.4 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:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/ESSS/conda-devenv/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" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

conda-devenv could always use more documentation, whether as part of the official conda-devenv 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/ESSS/conda-devenv/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 :)

2.4.2 Get Started!

Ready to contribute? Here's how to set up `conda-devenv` for local development.

1. Fork the `conda-devenv` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/conda-devenv.git
```

3. Create a new conda environment for developing:

```
$ conda create -c conda-forge -n devenv --file requirements_dev.txt
$ source activate devenv
$ pre-commit install
$ pip install -e .
```

4. Run tests to ensure you are starting from a working state:

```
$ pytest tests
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

6. Now make your changes and commit.
7. When you're done making changes, check that your changes still pass the tests:

```
$ pytest tests
```

8. 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
```

9. Submit a pull request through GitHub.

2.4.3 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 3.6, 3.7 and 3.8. Check https://github.com/ESSS/conda-devenv/actions?query=event%3Apull_request and make sure that the tests pass for all supported Python versions.

2.5 CHANGELOG

2.5.1 2.1.1 (2020-08-13)

- Correctly handle editable installs when using pip dependencies (#113).

2.5.2 2.1.0 (2020-06-26)

- New `get_env` function to handle environment variables in Jinja contexts so it conveys better error messages in case of missing/invalid environment variables.

2.5.3 2.0.0 (2020-06-08)

- Drop support for Python 2.7, 3.4, and 3.5.
- Correctly parse relative includes without jinja root.
- New `--verbose` flag which is passed on to `conda`.

2.5.4 1.1.3 (2019-12-26)

- Accept package version specifiers containing upper case letters (#95).

2.5.5 1.1.2 (2019-05-07)

- Correctly support git and mercurial repositories in `pip` dependencies (#92).

2.5.6 1.1.1 (2019-03-22)

- Remove yaml load warnings by using `yaml.safe_load` instead of `yaml.load`.
- Fix `NoneType object is not iterable` error when `includes` is empty.

2.5.7 1.1.0 (2019-02-14)

- New `is_included` jinja variable which is a boolean indicating if the current file was included by another `devenv.yml` file (`True`) or it is the original file passed to `conda devenv` (`False`) (#72).
- Added shortcuts to common jinja 2 checks, for example `win` which is equal to `sys.platform.startswith("win")` (#75).
- Added support conda-build-style YAML line comments (‘‘ # [win]‘‘) (#79).
- New `min_conda_devenv_version` jinja2 function that can be used to specify a minimum conda-devenv version:

```
{{ min_conda_devenv_version("1.1") }}
name: my-environment
```

  This is recommended when using new features so users will be shown a descriptive error message instead of subtle failures (#81).

2.5.8 1.0.4 (2018-09-20)

- Do not fail if history file does not exists (#66).
- Obtain `envs_dir` without using a subprocess (#67).

2.5.9 1.0.3 (2018-06-20)

- Find correct env directory through `envs_dir` instead of matching first in `envs`. This makes environment directory location more reliable in newer conda versions.

2.5.10 1.0.2 (2018-06-07)

- Fix problem with channel specification being wrongly exported (#62).

2.5.11 1.0.1 (2018-06-04)

- Truncate the environment's history file to have the old "prune" behavior when needed (#59).

2.5.12 1.0.0 (2017-09-19)

- Add –version flag (#47).
- Provide a better error message when 'environment.devenv.yml' file is not found (#48).
- Support for pip on dependencies section (#55).

2.5.13 0.9.6 (2017-07-24)

- Applies an "AND" when merging dependencies (#53).
- On Mac generates the same scripts as for Linux (no longer `.bat` files).

2.5.14 0.9.5 (2017-04-24)

- Handle `None` correctly, which actually fixes (#49).

2.5.15 0.9.4 (2017-04-20)

- Fixed major bug where activate/deactivate scripts were not being generated (#49).

2.5.16 0.9.3 (2017-04-10)

- `conda-devenv` no longer requires `conda` to be on `PATH` to work (#45).

2.5.17 0.9.2 (2017-03-27)

- Fix conda-forge package.

2.5.18 0.9.1 (2017-03-22)

- Fix activate and deactivate `bash` scripts: variables not in the environment before activation are now properly unset after deactivation.
- Fix activate and deactivate `bash` scripts: quote variables when exporting them.

2.5.19 0.9.0 (2017-03-17)

- New option `--print-full`, which also prints the expanded `environment:` section.

2.5.20 0.8.1 (2017-03-16)

- Fix entry point call to `main`.

2.5.21 0.8.0 (2017-03-16)

- `conda-devenv` now can receive standard `environment.yml` files, in which case the file will just be forwarded to `conda env update` normally.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search