# picky-conda Documentation

***Release 2.0.4***

**Chris Withers**

**Oct 17, 2018**

# Contents

Picky is a tool for making sure that the packages you have installed in a conda environment match those you have specified.

The recommended workflow is to have an *environment.yaml* that contains your abstract requirements and an *environment.lock.yaml* that contains your concrete requirements.

An abstract set of requirements contains the minimal amount of information to install all the packages your projects needs and any maximum or minimum version requirements there may be. For example:

```
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3
  - pandas < 0.19
  - pip:
    - testfixtures >= 6.0.0
```

Concrete requirements contain all the detail needed to exactly reproduce the current environment. Given the abstract requirements above, concrete requirements would look something like:

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - appnope=0.1.0=py27hb466136_0
  - numpy=1.11.3=py27h8a80b8c_4
  - pandas=0.18.1=np111py27_0
  ...
  - pip:
    - testfixtures==6.0.1
```

So, the rough workflow of a project starting from scratch would be:

1. `conda create -n yourproject -c simplistix python=X.Y picky-conda`
2. Add abstract dependencies, including your major Python version and picky-conda, along with any packages that need to be installed with pip, to *environment.yaml*. After each modification of *environment.yaml*, run `conda env update` as follows:

   ```
   $ conda env update --file environment.yaml
   ```

   If you use `conda install` or `pip install` to install packages, you will need to remember to add those packages to your *environment.yaml* afterwards.
3. When you want to release to production or share your environment with other developers, lock your current exact dependencies with:

   ```
   $ picky lock
   ```

   The resultant *environment.lock.yaml* should be version controlled.
4. When setting up the environment elsewhere, the concrete requirements should be used:

   ```
   $ conda env create --force --file environment.lock.yaml
   ```

   If this is as part of an automated testing process, you should also do the following once the environment is set up to make sure there are no unpinned dependencies:

```
$ picky check
```

CHAPTER 1

# Configuring Picky

Picky can be configured by placing a file called *picky.yaml* in the same directory as your *environment.yaml*. Example content could be:

```
detail: version
develop:
  mypackage: .
ignore:
  # mac-only:
  - appnope
```

Each of the sections is optional and explained below.

## 1.1 Lock Detail Level

The *build* part of a conda version is the bit after the second = and is specific to the target operating system. If your *environment.lock.yaml* will be used on multiple operating systems, you can exclude the build part of versions by placing `detail: version` in your *picky.yaml*.

If this is done, your concrete requirements will end up looking like the following:

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - appnope=0.1.0
  - numpy=1.11.3
  - pandas=0.18.1
  ...
  - pip:
    - testfixtures==6.0.1
```

## 1.2 Ignoring Packages

Some packages are only installed on a particular operating system. `conda env` does not currently support operating-specific dependencies. To work around this, you can include package names in the *ignore* section of *picky.yaml* and they will not be included in any generated *environment.lock.yaml*.

If the sample *picky.yaml* was used, then the resultant concrete requirements produced by `picky lock` would be:

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - numpy=1.11.3
  - pandas=0.18.1
  ...
  - pip:
    - testfixtures==6.0.1
```

Note the absence of the `appnope` package.

## 1.3 Installing Packages with pip in Development Mode

It's often handy to install an application's source using `pip install -e` in order to get python packages importable and console scripts set up. However, `conda env export` currently reports packages installed in this way incorrectly.

To help with all this, you can include a mapping of packages to be installed in this way to their paths on disk in *picky.yaml* and will use that to create appropriate entries in your *environment.lock.yaml*.

If the sample *picky.yaml* was used, then the resultant concrete requirements produced by `picky lock` would be:

```
name: myproject
channels:
  - conda-forge
  - defaults
dependencies:
  - numpy=1.11.3
  - pandas=0.18.1
  ...
  - pip:
    - testfixtures==6.0.1
    - "-e ."
```

CHAPTER 2

# Further Documentation

## 2.1 Installation Instructions

The best way to install picky is with conda:

```
conda install -c simplistix picky-conda
```

Of course, once it's installed, make sure it's in your `environment.yml`!

## 2.2 Development

If you wish to contribute to this project, then you should fork the repository found here:

https://github.com/Simplistix/picky-conda/

Once that has been done, you can follow these instructions to perform various development tasks:

### 2.2.1 Setting up the environment

All development requires that you have a conda environment set up, this can be created by doing the following from within a checkout of the above repository, assuming you have installed conda by following its instructions:

```
$ conda env create -n picky-conda python=3.6 --file environment.yaml
$ conda activate picky-conda
```

### 2.2.2 Running the tests

Once you have set up and activated your conda environment, the tests can be run from the root of your checkout as follows:

```
$ pytest
```

### 2.2.3 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ cd docs
$ make html
```

### 2.2.4 Making a release

To make a release, just update the version in `setup.py`, update the change log, tag it and push to https://github.com/Simplistix/picky-conda/ and Travis CI should take care of the rest.

## 2.3 Changes

### 2.3.1 2.0.4 (17 Oct 2018)

- Attempt to get Travis to build conda packages that do not require a specific version of Python, like it used to.

### 2.3.2 2.0.3 (17 Oct 2018)

- All packages are sorted by name when serializing. This prevents spurious differences being reported when the order of packages differs as reported by conda.

### 2.3.3 2.0.2 (2 May 2018)

- Channel ordering is now ignored ignored by `picky check`.

### 2.3.4 2.0.1 (22 April 2018)

- Fix bugs in handling of packages installed in *develop* mode.

### 2.3.5 2.0.0 (20 April 2018)

- Re-write to target just conda environments and based on *environment.yaml*, *environment.lock.yaml* and *picky.yaml*.

### 2.3.6 0.9.2 (1 July 2015)

- check to see if `pip` takes `--disable-pip-version-check` before using it.

### 2.3.7 0.9.1 (23 June 2015)

- correct the dependency specification of `argparse` so it only occurs on Python 2.6

### 2.3.8 0.9 (22 June 2015)

- Python 3 support
- Fixed handling of package 'extras' in pip output and specifications.
- Fixed handling of arbitrary equality clauses in pip output and specifications.

### 2.3.9 0.8 (22 June 2015)

- Initial Release

## 2.4 License

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search