scikit-ci Documentation

Release 0.21.0.post0.dev0+gb348833

The scikit-build team

May 22, 2019

User guide

scikit-ci enables a centralized and simpler CI configuration for Python extensions.

By having `appveyor.yml`, `azure-pipelines.yml`, `circle.yml` and `.travis.yml` calling the scikit-ci command-line executable, all the CI steps for all service can be fully described in one `scikit-ci.yml` configuration file.

CHAPTER 1

Installation

1.1 Install package with pip

To install with pip:

```
$ pip install scikit-ci
```

1.2 Install from source

To install scikit-ci from the latest source, first obtain the source code:

```
$ git clone https://github.com/scikit-build/scikit-ci
$ cd scikit-ci
```

then install with:

```
$ pip install .
```

or:

```
$ pip install -e .
```

for development.

1.3 Dependencies

1.3.1 Python Packages

The project has a few common Python package dependencies. The runtime dependencies are:

```
pyfiglet
ruamel.yaml>=0.15;python_version == '2.7'
ruamel.yaml>=0.15,<=0.15.94;python_version == '3.4'
ruamel.yaml>=0.15;python_version > '3.4'
```

The development dependencies (for testing and coverage) are:

```
codecov==2.0.15
coverage==4.5.1
flake8==3.5.0
pytest==3.6.3
pytest-cov==2.5.1
pytest-runner==4.2
wheel>=0.29.0
```

CHAPTER 2

Usage

The scikit-ci command line executable allows to execute commands associated with steps described in a scikit-ci *configuration file*.

2.1 Executing scikit-ci steps

Invoking scikit-ci will execute all steps listed in a scikit-ci *configuration file*:

```
ci
```

This command executes in order the steps listed below:

- before_install
- install
- before_build
- build
- test
- after_test

It also possible to execute a given step and its dependent steps:

```
ci build
```

In that case, the executed steps will be:

- before_install
- install
- before_build
- build

Note: Remember that:

- steps are executed following a specific *ordering*
- scikit-ci *keeps track* of previously executed steps.
- environment variables set in `step(n)` will be available in `step(n+1)`. For more details, see *Environment variable persistence*

2.2 Calling scikit-ci through `python -m ci`

You can invoke scikit-ci through the Python interpreter from the command line:

```
python -m ci [...]
```

This is equivalent to invoking the command line script `ci [...]` directly.

2.3 Getting help on version, option names

```
ci --version    # shows where ci was imported from
ci -h | --help # show help on command line
```

CHAPTER 3

Configuration file

The configuration file is read by the scikit-ci executable to find out which commands to execute for a given step.

The configuration file should named `scikit-ci.yml` and is usually added to the root of a project.

It is a YAML file that can be validated against scikit-ci-schema.yml.

3.1 Concept of Step

A step consist of a list of `commands` and optional key/value pairs describing the `environment`.

More specifically, a step can be described using the following structure:

```yaml
before_install:
  environment:
    FOO: bar
  commands:
    - echo "Hello world"
```

where `before_install` can be replaced by any of these:

- `before_install`
- `install`
- `before_build`
- `build`
- `test`
- `after_test`

3.2 Mapping with Appveyor, Azure Pipelines, CircleCI and TravisCI steps

scikit-ci do not impose any particular mapping.

Documentation specific to each services is available here:

- Appveyor build pipeline
- Azure pipelines
- CircleCI configuration 2.0
- CircleCI configuration 1.0 (deprecated)
- TravisCI build lifecycle

Reported below are some recommended associations that are know to work.

- `appveyor.yml`:

```
install:
  - python -m ci install

build_script:
  - python -m ci build

test_script:
  - python -m ci test

after_test:
  - python -m ci after_test
```

Note: Since on windows the `ci` executable is installed in the `Scripts` directory (e.g *C:\Python27\Scripts\ci.exe*) which is not in the `PATH` by default, the `python -m ci` syntax is used.

- `azure-pipelines.yml`:

```
      - script: python -m ci install
        displayName: Install

      - script: python -m ci build
        displayName: Build

      - script: python -m ci test
        displayName: Test

      - powershell: |
          if ($env:CODECOV_TOKEN -ne $null -And $env:BUILD_REASON -ne
↪"PullRequest") {
            python -m ci after_test
          }
        displayName: After Test
        env:
          CODECOV_TOKEN: $(CODECOV_TOKEN)
```

- `.circleci/config.yml` (CircleCI 2.0):

```
  steps:
    - checkout
    - run:
        <<: *initialize-venv
    - run:
        name: Install scikit-ci
        command: |
          . ../venv/bin/activate
          BOOTSTRAP_BRANCH=$CIRCLE_BRANCH
          BOOTSTRAP_REPO_SLUG=$CIRCLE_PROJECT_USERNAME/$CIRCLE_PROJECT_
↪REPONAME
          if [[ $CIRCLE_PR_USERNAME != "" ]]; then
            BOOTSTRAP_BRANCH=$(curl -s https://api.github.com/repos/${CIRCLE_
↪PROJECT_USERNAME}/${CIRCLE_PROJECT_REPONAME}/pulls/${CIRCLE_PR_NUMBER} |␣
↪jq -r '.head.ref')
            BOOTSTRAP_REPO_SLUG=$CIRCLE_PR_USERNAME/$CIRCLE_PR_REPONAME
          fi
          echo "BOOTSTRAP_BRANCH:$BOOTSTRAP_BRANCH"
          echo "BOOTSTRAP_REPO_SLUG:$BOOTSTRAP_REPO_SLUG"
          git clone git://github.com/$BOOTSTRAP_REPO_SLUG -b $BOOTSTRAP_
↪BRANCH ../bootstrap-scikit-ci
          pip install -U ../bootstrap-scikit-ci
    - run:
        name: Install dependencies
        command: |
          . ../venv/bin/activate
          ci install
    - run:
        name: Flake8
        command: |
          . ../venv/bin/activate
          ci before_build
    - run:
        name: Build
        command: |
          . ../venv/bin/activate
          ci build
    - run:
        name: Test
        command: |
          . ../venv/bin/activate
          ci test
    - run:
        name: Coverage
        command: |
          . ../venv/bin/activate
          ci after_test
```

- `circle.yml` (CircleCI 1.0):

```
dependencies:
  override:
    - |
      BOOTSTRAP_BRANCH=$CIRCLE_BRANCH
      BOOTSTRAP_REPO_SLUG=$CIRCLE_PROJECT_USERNAME/$CIRCLE_PROJECT_REPONAME
      if [[ $CIRCLE_PR_USERNAME != "" ]]; then
        BOOTSTRAP_BRANCH=$(curl -s https://api.github.com/repos/${CIRCLE_
↪PROJECT_USERNAME}/${CIRCLE_PROJECT_REPONAME}/pulls/${CIRCLE_PR_NUMBER} |␣
↪jq -r '.head.ref')
```

(continues on next page)

(continued from previous page)

```
        BOOTSTRAP_REPO_SLUG=$CIRCLE_PR_USERNAME/$CIRCLE_PR_REPONAME
      fi
      echo "BOOTSTRAP_BRANCH:$BOOTSTRAP_BRANCH"
      echo "BOOTSTRAP_REPO_SLUG:$BOOTSTRAP_REPO_SLUG"
      git clone git://github.com/$BOOTSTRAP_REPO_SLUG -b $BOOTSTRAP_BRANCH ..
↪/bootstrap-scikit-ci
      pip install -U ../bootstrap-scikit-ci

    - ci install

test:
  override:
    - ci test

deployment:
  master:
    branch: master
    commands:
      - ci after_test
```

- `.travis.yml`

```
install:
  - ci install

script:
  - ci test

after_success:
  - ci after_test
```

3.3 Order of steps

scikit-ci execute steps considering the following order:

1. `before_install`
2. `install`
3. `before_build`
4. `build`
5. `test`
6. `after_test`

This means that the *mapping specified* in the continuous integration file has to be done accordingly.

3.4 Automatic execution of dependent steps

Considering the *step ordering*, executing any `step(n)` ensures that `step(n-1)` has been executed before.

3.5 Keeping track of executed steps

scikit-ci keeps track of executed steps setting environment variables of the form `SCIKIT_CI_<STEP_NAME>` where `<STEP_NAME>` is any of the step name in upper-case.

Note: Specifying the command line option `--force` allows to force the execution of the steps ignoring the values of the `SCIKIT_CI_<STEP_NAME>` environment variables.

3.6 Environment variable persistence

Environment variable defined in any given step are always guaranteed to be set in steps executed afterward.

This is made possible by serializing the environment on the filesystem.

Note: After executing steps, a file named `env.json` is created in the current directory along side `scikit-ci.yml`. This is where the environment is cached for re-use in subsequent steps.

Specifying the command line option `--clear-cached-env` allows to execute steps after removing the `env.json` file.

3.7 Step specialization

For any given step, it is possible to specify `commands` and `environment` variables specific to each continuous integration service.

Recognized services are:

- `appveyor`
- `azure`
- `circle`
- `travis`

3.7.1 Commands

`commands` common to all services are executed first, then `commands` specific to each services are executed.

For example, considering this configuration used on CircleCI and TravisCI:

```
before_install:
  commands:
    - echo "Hello Everywhere"

  circle:
    commands:
      - echo "Hello on CircleCI"

  travis:
```

(continues on next page)

(continued from previous page)

```
    linux:
      commands:
        - echo "Hello on TravisCI"
```

The output on the different service will be the following:

- CircleCI:

```
Hello Everywhere
Hello on CircleCI
```

- TravisCI:

```
Hello Everywhere
Hello on TravisCI
```

Note: Sections *Command Specification* and *Python Command Specification* describe the different types of command.

3.7.2 Environment

Similarly, `environment` can be overridden for each service.

For example, considering this configuration used on CircleCI and TravisCI:

```
before_install:

  circle:
    environment:
      CATEGORY_2: 42

  travis:
    linux:
      environment:
        CATEGORY_1: 99

  environment:
    CATEGORY_1: 1
    CATEGORY_2: 2

  commands:
    - echo "CATEGORY_1 is ${CATEGORY_1}"
    - echo "CATEGORY_2 is ${CATEGORY_2}"
```

The output on the different service will be the following:

- on CircleCI:

```
CATEGORY_1 is 1
CATEGORY_2 is 42
```

- on TravisCI:

```
CATEGORY_1 is 99
CATEGORY_2 is 2
```

3.8 Reserved Environment Variables

- `CI_NAME`: This variable is automatically set by scikit-ci and will contain the name of the continuous integration service currently executing the step.

3.9 Environment variable usage

To facilitate the use of environment variable across interpreters, scikit-ci uses a specific syntax.

Environment variable specified using `$<NAME_OF_VARIABLE>` in both commands and environment variable will be expanded.

For example, considering this configuration used on Appveyor, CircleCI and TravisCI:

```
before_install:

  appveyor:
    environment:
      TEXT: Windows$<TEXT>

  travis:
    linux:
      environment:
        TEXT: LinuxWorld

  environment:
    TEXT: World

  commands:
    - echo $<TEXT>
```

The output on the different service will be the following:

- on Appveyor:

```
WindowsWorld
```

- on CircleCI:

```
World
```

- on TravisCI:

```
LinuxWorld
```

Note: On system having a POSIX interpreter, the environment variable will **NOT** be expanded if included in string start with a single quote.

class ci.driver.**Driver**

static expand_command(*command*, *environments*, *posix_shell=True*)
Return an updated `command` string where all occurrences of `$<EnvironmentVarName>` (with a corresponding env variable set) have been replaced.

If `posix_shell` is True, only occurrences of `$<EnvironmentVarName>` in string starting with double quotes will be replaced.

See https://www.gnu.org/software/bash/manual/html_node/Double-Quotes.html and https://www.gnu.org/software/bash/manual/html_node/Single-Quotes.html

3.10 Command Specification

Specifying command composed of a program name and arguments is supported on all platforms.

For example:

```
test:
  commands:
    - echo "Hello"
    - python -c "print('world')"
    - git clone git://github.com/scikit-build/scikit-ci
```

On unix based platforms (e.g CircleCI and TravisCI), commands are interpreted using `bash`.

On windows based platform (e.g Appveyor), commands are interpreted using the windows command terminal `cmd.exe`.

Since both interpreters expand quotes differently, we recommend to avoid single quoting argument. The following table list working recipes:

	CircleCi, TravisCI	Appveyor
scikit-ci command	**bash output**	**cmd output**
`echo Hello1`	Hello1	Hello1
`echo "Hello2"`	Hello2	“Hello2”
`echo 'Hello3'`	Hello3	‘Hello3’
`python -c "print('Hello4')"`	Hello4	Hello4
`python -c 'print("Hello5")'`	Hello5	`no output`
`python -c "print('Hello6\'World')"`	Hello6’World	Hello6’World

And here are the values associated with `sys.argv` for different scikit-ci commands:

```
python program.py --things "foo" "bar" --more-things "doo" 'dar'
```

Output on CircleCi, TravisCI:

```
arg_1 [--things]
arg_2 [foo]
arg_3 [bar]
arg_4 [--more-things]
arg_5 [doo]
arg_6 [dar]
```

Output on Appveyor:

```
arg_1 [--things]
arg_2 [foo]
arg_3 [bar]
arg_4 [--more-things]
```

(continues on next page)

(continued from previous page)

```
arg_5 [doo]
arg_6 ['dar']     # <-- Note the presence of single quotes
```

```
python program.py --things "foo" "bar" --more-things "doo" 'dar'
```

Output on CircleCi, TravisCI:

```
arg_1 [--the-foo=foo]
arg_2 [-the-bar=bar]
```

Output on Appveyor:

```
arg_1 [--the-foo=foo]
arg_2 [-the-bar='bar']     # <-- Note the presence of single quotes
```

Note: Here are the source of `program.py`:

```
import sys
for index, arg in enumerate(sys.argv):
    if index == 0:
        continue
    print("arg_%s [%s]" % (index, sys.argv[index]))
```

3.11 Python Command Specification

New in version 0.10.0.

The `python` commands are supported on all platforms.

For example:

```
test:
  commands:
    - python: print("single_line")
    - python: "for letter in ['a', 'b', 'c']: print(letter)"
    - python: |
              import os
              if 'FOO' in os.environ:
                  print("FOO is set")
              else:
                  print("FOO is *NOT* set")
```

Note: By using `os.environ`, they remove the need for specifying environment variable using the `$<NAME_OF_VARIABLE>` syntax described in *Environment variable usage*.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

4.1 Types of Contributions

You can contribute in many ways:

4.1.1 Report Bugs

Report bugs at https://github.com/scikit-build/scikit-ci/issues.

If you are reporting a bug, please include:

- Any details about your CI 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

The scikit-ci project could always use more documentation. We welcome help with the official scikit-ci docs, in docstrings, or even on blog posts and articles for the web.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/scikit-build/scikit-ci/issues.

If you are proposing a new 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 *scikit-ci* for local development.

1. Fork the *scikit-ci* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/scikit-ci.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed (*pip install virtualenvwrapper*), this is how you set up your cloned fork for local development:

   ```
   $ mkvirtualenv scikit-ci
   $ cd scikit-ci/
   $ 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
   $ python setup.py test
   $ tox
   ```

   If needed, you can get flake8 and tox by using *pip install* to 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 2.7, and 3.3, 3.4, 3.5 and PyPy. Check https://travis-ci.org/scikit-build/scikit-ci/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest tests/test_scikit_ci.py::test_expand_environment
```

CHAPTER 5

Credits

Please see the GitHub project page at https://github.com/scikit-build/scikit-ci/graphs/contributors

CHAPTER 6

History

scikit-ci was initially developed in May 2016 by Omar Padron to facilitate the continuous integration of the scikit-build project.

At that time, it already consisted of a driver script calling methods specific to each continuous integration service. By having each CI service calling the same driver script, there was no need to deal with implementing install/test/build steps over and over in different scripting languages (power shell, shell or windows batch). Instead all code was implemented in python code leveraging the subprocess module.

Later in early September 2016, with the desire to setup cross-platform continuous integration for other project and avoid duplication or maintenance hell, a dedicated repository was created by Jean-Christophe Fillion-Robin. By simply cloning the repository, it was possible to more easily enable CI for other projects.

While this was an improvement, all the steps were still hardcoded in the driver scripts, the project was not easily customizable. More could be done to improve the user experience.

Finally, in late September 2016, all hardcoded code was moved into standalone executable python scripts. Then, Jean-Christophe came up with the concept of scikit-ci.yml configuration file. This configuration file allows to describe the commands and environment for each step (install, test and build) specific to a project and associated continuous integration services.

CHAPTER 7

Release Notes

This is the list of changes to scikit-build between each release. For full details, see the commit logs at http://github.com/scikit-build/scikit-ci

7.1 Scikit-ci 0.21.0

- Fix installation of using Python 3.4

7.2 Scikit-ci 0.20.0

- Support environment file *env.json* update from within step.

7.3 Scikit-ci 0.19.0

- Streamline use of *ci.driver.Driver.save_env* ensuring provided dictionary is stringified.

7.4 Scikit-ci 0.18.0

- Add support for Azure Pipelines

7.5 Scikit-ci 0.17.0

- Add support for ruamel.yaml >= 0.15.52 and fix *AttributeError: 'CommentedMap' object has no attribute 'replace'* error.

CHAPTER 8

Making a release

A core developer should use the following steps to create a release *X.Y.Z* of **scikit-ci** on PyPI.

8.1 Prerequisites

- All CI tests are passing on AppVeyor, CircleCI and Travis CI.
- You have a GPG signing key.

8.2 Documentation conventions

The commands reported below should be evaluated in the same terminal session.

Commands to evaluate starts with a dollar sign. For example:

```
$ echo "Hello"
Hello
```

means that `echo "Hello"` should be copied and evaluated in the terminal.

8.3 Setting up environment

1. First, register for an account on PyPI.
2. If not already the case, ask to be added as a `Package Index Maintainer`.
3. Create a `~/.pypirc` file with your login credentials:

```
[distutils]
index-servers =
  pypi
  pypitest

[pypi]
username=<your-username>
password=<your-password>

[pypitest]
repository=https://test.pypi.org/legacy/
username=<your-username>
password=<your-password>
```

where `<your-username>` and `<your-password>` correspond to your PyPI account.

8.4 PyPI: Step-by-step

1. Make sure that all CI tests are passing on AppVeyor, CircleCI and Travis CI.
2. Download the latest sources

```
$ cd /tmp && \
  git clone git@github.com:scikit-build/scikit-ci && \
  cd scikit-ci
```

3. List all tags sorted by version

```
$ git fetch --tags && \
  git tag -l | sort -V
```

4. Choose the next release version number

```
$ release=X.Y.Z
```

Warning: To ensure the packages are uploaded on PyPI, tags must match this regular expression: `^[0-9]+(\.[0-9]+)*(\.post[0-9]+)?$`.

5. In *README.rst*, update PyPI download count after running this big table query and commit the changes.

```
$ git add README.rst && \
  git commit -m "README: Update download stats [ci skip]"
```

Note: To learn more about *pypi-stats*, see How to get PyPI download statistics.

6. In *CHANGES.rst* replace `Next Release` section header with `Scikit-ci X.Y.Z` and commit the changes.

```
$ git add CHANGES.rst && \
  git commit -m "Scikit-ci ${release}"
```

7. Tag the release

```
$ git tag --sign -m "Scikit-ci ${release}" ${release} master
```

Warning: We recommend using a GPG signing key to sign the tag.

8. Create the source distribution and wheel

```
$ python setup.py sdist bdist_wheel
```

9. Publish the both release tag and the master branch

```
$ git push origin ${release} && \
  git push origin master
```

10. Upload the distributions on PyPI

```
twine upload dist/*
```

Note: To first upload on TestPyPI , do the following:

```
$ twine upload -r pypitest dist/*
```

11. Create a clean testing environment to test the installation

```
$ pushd $(mktemp -d) && \
  mkvirtualenv scikit-ci-${release}-install-test && \
  pip install scikit-ci && \
  ci --help
```

Note: If the `mkvirtualenv` command is not available, this means you do not have virtualenvwrapper installed, in that case, you could either install it or directly use virtualenv or venv.

To install from TestPyPI, do the following:

```
$ pip install -i https://test.pypi.org/simple scikit-ci
```

12. Cleanup

```
$ popd && \
  deactivate  && \
  rm -rf dist/* && \
  rmvirtualenv scikit-ci-${release}-install-test
```

13. Add a `Next Release` section back in *CHANGES.rst*, commit and push local changes.

```
$ git add CHANGES.rst && \
  git commit -m "CHANGES.rst: Add \"Next Release\" section [ci skip]" && \
  git push origin master
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

CHAPTER 10

Resources

- Free software: Apache Software license
- Documentation: http://scikit-ci.readthedocs.org
- Source code: https://github.com/scikit-build/scikit-ci
- Mailing list: https://groups.google.com/forum/#!forum/scikit-build

Index