PyCTest Documentation

Release 0.0.9

Jonathan R. Madsen

Dec 11, 2018

Contents:

PyCTest is an open-source Python package for select bindings to CMake, CTest, and CPack.

CHAPTER 1

Build Status

CHAPTER 2

Anaconda

CHAPTER 3

Features

- Directly run *cmake*, *ctest*, *cpack*.
- Dynamically generate testing and submit to CDash testing dashboard

CHAPTER 4

Contribute

- Issue Tracker: https://github.com/jrmadsen/pyctest/issues
- Documentation: https://pyctest.readthedocs.io/en/latest/
- Source Code: https://github.com/jrmadsen/pyctest/tree/master

CHAPTER 5

Table of Contents

5.1 About

PyCTest is python bindings of select portions of CMake/CTest package. This enables the generation of CTest test files from Python without a CMake build system.

5.1.1 Available on PyPi and Anaconda

- PyPi has the source distribution
- PyPi installs can take a long time since CMake must be compiled from scratch
- Anaconda has compiled distributions

General Setup

- Create an anaconda environment for PyCTest: `conda create -n pyctest python=3.6 pyctest`
- Activate this environment: `source activate pyctest`
- Write a driver Python script

Example for Python project

The following is an example for a Python code with a compiled C extension that uses `nosetests` for unit-testing:

```
#!/usr/bin/env python

import os, sys, platform
import pyctest.pyctest as pyctest
import pyctest.helpers as helpers

```

(continues on next page)

(continued from previous page)

```
parser = helpers.ArgumentParser("ProjectName", source_dir=os.getcwd(), binary_dir=os.
↪getcwd())
parser.add_argument("-n", "--build", type=str, required=True, help="Build name for␣
↪identification")
args = parser.parse_args()

pyctest.BUILD_NAME = "{}".format(args.build)
pyctest.BUILD_COMMAND = "python setup.py build_ext --inplace"
pyctest.UPDATE_COMMAND = "git"

test = pyctest.test()
test.SetName("unittest")
# insert the command to run the tests for project
test.SetCommand(["nosetests"])

pyctest.generate_config()
pyctest.generate_test_file()
pyctest.run(pyctest.ARGUMENTS)
```

Example for autotools project

```
#!/usr/bin/env python

import os, sys, platform
import multiprocessing as mp
import pyctest.pyctest as pyctest
import pyctest.helpers as helpers

parser = helpers.ArgumentParser("ProjectName", source_dir=os.getcwd(), binary_dir=os.
↪getcwd())
parser.add_argument("-n", "--build", type=str, required=True, help="Build name for␣
↪identification")
args = parser.parse_args()

# CONFIGURE_COMMAND can only run one command so if autogen is required, just execute␣
↪it here
cmd = pyctest.command(["./autogen.sh"])
cmd.SetWorkingDirectory(pyctest.SOURCE_DIRECTORY)
cmd.SetErrorQuiet(False)
cmd.Execute()

pyctest.BUILD_NAME = "{}".format(args.build)
pyctest.UPDATE_COMMAND = "git"
pyctest.CONFIGURE_COMMAND = "./configure"
pyctest.BUILD_COMMAND = "make -j{}".format(mp.cpu_count())

test = pyctest.test()
test.SetName("unittest")
# insert the command to run the tests for project
test.SetCommand(["./run-testing.sh"])

pyctest.generate_config()
pyctest.generate_test_file()
pyctest.run(pyctest.ARGUMENTS)
```

Example for CMake project

```
#!/usr/bin/env python

import os
import sys
import platform
import multiprocessing as mp
import pyctest.pyctest as pyctest
import pyctest.helpers as helpers

binary_dir = os.path.join(os.getcwd(), "build-ProjectName")
parser = helpers.ArgumentParser("ProjectName", os.getcwd(), binary_dir)
parser.add_argument("-n", "--build", type=str, required=True, help="Build name for␣
↪identification")
args = parser.parse_args()

pyctest.BUILD_NAME = "{}".format(args.build)
pyctest.UPDATE_COMMAND = "git"
pyctest.CONFIGURE_COMMAND = "cmake {}".format(pyctest.SOURCE_DIRECTORY)
pyctest.BUILD_COMMAND = "cmake --build {} --target all -- -j{}".format(pyctest.BINARY_
↪DIRECTORY, mp.cpu_count())

test = pyctest.test()
test.SetName("unittest")
# insert the command to run the tests for project
test.SetCommand(["./run-testing.sh"])

pyctest.generate_config(pyctest.BINARY_DIRECTORY)
pyctest.generate_test_file(pyctest.BINARY_DIRECTORY)
pyctest.run(pyctest.ARGUMENTS, pyctest.BINARY_DIRECTORY)
```

Python Modules

- `import pyctest` – global package
- `import pyctest.pyctest` – CTest module
- `import pyctest.pycmake` – CMake module
- `import pyctest.helpers` – Helpers module
  - includes command line arguments (`argparse`) for PyCTest
- NOTES:
  - This document uses `pyctest.<...>` as shorthand for `pyctest.pyctest.<...>` (e.g. `import pyctest.pyctest as pyctest`)
  - It is possible to call CMake from this package but it is generally not the purpose

Benefits

- Integration into continuous integration systems (e.g. Travis, AppVeyor, Jenkins, etc.) and pushing to CDash dashboard will combine all the results in one place
  - The warnings and errors are enumerated in CDash (no more parsing stdout logs for errors)

- Easily create platform-independent testing
- No need to migrate build system to CMake – just specify `pyctest.BUILD_COMMAND`

Standard Configuration Variables

- `pyctest.PROJECT_NAME`
- `pyctest.SOURCE_DIRECTORY`
- `pyctest.BINARY_DIRECTORY`
- `pyctest.SITE`
- `pyctest.BUILD_NAME`
- `pyctest.TRIGGER`
- `pyctest.CHECKOUT_COMMAND`
- `pyctest.BUILD_COMMAND`
- `pyctest.MODEL`
- `pyctest.CUSTOM_COVERAGE_EXCLUDE`
- `pyctest.CUSTOM_MAXIMUM_NUMBER_OF_ERRORS`
- `pyctest.CUSTOM_MAXIMUM_NUMBER_OF_WARNINGS`
- `pyctest.CUSTOM_MAXIMUM_PASSED_TEST_OUTPUT_SIZE`

Setting Arbitrary Variables

```
pyctest.set("CTEST_TOKEN_FILE", "${CMAKE_CURRENT_LIST_DIR}/.ctest-token")
```

Generating a Test

```
test = pyctest.test()
test.SetName("nosetests")
test.SetCommand(["nosetests", "test", "--cover-xml", "--cover-xml-file=coverage.xml"])
# set directory to run test
test.SetProperty("WORKING_DIRECTORY", pyctest.BINARY_DIRECTORY)
test.SetProperty("RUN_SERIAL", "ON")
test.SetProperty("ENVIRONMENT", "OMP_NUM_THREADS=1")
```

Examples

- Basic example
- Advanced example
- includes submission to CDash dashboard

CDash Integration Example

Results from running the TomoPy example can be found at the TomoPy CDash Testing Dashboard @ NERSC

- Python code with C extensions without CMake build system
- The build logs from "python setup.py install" are registered in the "Build" section
- The `nosetests test` command + other are wrapped into CTests

5.1.2 Testing Example

PyCTest can be used to simple execute tests and submit to a dashboard without any configuration, build, etc. steps

```
#!/usr/bin/env python

import os
import sys
import shutil
import argparse
import platform
import traceback

import pyctest.pyctest as pyctest
import pyctest.pycmake as pycmake
import pyctest.helpers as helpers

if __name__ == "__main__":

    directory = os.path.join(os.getcwd(), "pycm-test")

    # these are required
    pyctest.PROJECT_NAME = "PyCTest"
    pyctest.SOURCE_DIRECTORY = directory
    pyctest.BINARY_DIRECTORY = directory

    args = helpers.ArgumentParser(pyctest.PROJECT_NAME,
                                  pyctest.SOURCE_DIRECTORY,
                                  pyctest.BINARY_DIRECTORY).parse_args()

    # set explicitly
    pyctest.MODEL = "Continuous"
    pyctest.SITE = platform.node()

    # create a Test object
    test = pyctest.test()
    test.SetName("list_directory")
    test.SetCommand(["ls", directory])
    test.SetProperty("WORKING_DIRECTORY", os.getcwd())

    # create a second test
    # previous test is already stored by PyCTest
    test = pyctest.test()
    test.SetName("hostname")
    test.SetCommand(["hostname"])
    test.SetProperty("TIMEOUT", "10")

    # generate the CTestConfig.cmake and CTestCustom.cmake
```

(continues on next page)

(continued from previous page)

```
    pyctest.generate_config(directory)

    # generate the CTestTestfile.cmake file
    pyctest.generate_test_file(directory)

    # run CTest -- e.g. ctest -VV ${PWD}/pycm-test
    pyctest.run(pyctest.ARGUMENTS, directory)
```

```
CTest arguments (default): '-V -DSTAGES=Start;Update;Configure;Build;Test;Coverage;
↪MemCheck -S Stages.cmake -j1'
Writing CTest test file: "/Users/jrmadsen/devel/c++/pyctest-master/pycm-test/
↪CTestTestfile.cmake"...
Generating test "list_directory"...
Generating test "hostname"...
-- STAGES = Start;Update;Configure;Build;Test;Coverage;MemCheck
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Running CTEST_START stage...
Run dashboard with model Continuous
   Source directory: /Users/jrmadsen/devel/c++/pyctest-master/pycm-test
   Build directory: /Users/jrmadsen/devel/c++/pyctest-master/pycm-test
   Track: Continuous
   Reading ctest configuration file: /Users/jrmadsen/devel/c++/pyctest-master/pycm-
↪test/CTestConfig.cmake
   Site: JRM-macOS-DOE.local
   Build name: [Darwin macOS 10.13.6 x86_64] [Python 3.6.7]
   Use Continuous tag: 20181129-2118
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_UPDATE stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_CONFIGURE stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_BUILD stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Running CTEST_TEST stage...
Test project /Users/jrmadsen/devel/c++/pyctest-master/pycm-test
    Start 1: list_directory
1/2 Test #1: list_directory ...................   Passed    0.00 sec
    Start 2: hostname
2/2 Test #2: hostname .........................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.01 sec
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_COVERAGE stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_MEMCHECK stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Skipping CTEST_SUBMIT stage...
-- [[Darwin macOS 10.13.6 x86_64] [Python 3.6.7]] Finished Continuous Stages (Start;
↪Update;Configure;Build;Test;Coverage;MemCheck)
```

5.2 Installation

This section covers the basics of how to download and install PyCTest.

Contents:

5.2.1 Supported Environments

PyCTest is tested, built, and distributed for python 2.7 3.5 3.6 on Linux/macOS and python 3.5 3.6 on Windows 10.

5.2.2 Installing from Conda (Recommended)

If you only want to run PyCTest, not develop it, then you should install through a package manager. Conda, our supported package manager, can install PyCTest and its dependencies for you.

First, you must have Conda installed, then open a terminal or a command prompt window and run:

```
$ conda install -c conda-forge PyCTest
```

This will install PyCTest and all the dependencies from the conda-forge channel.

Updating the installation

PyCTest is an active project, so we suggest you update your installation frequently. To update the installation run:

```
$ conda update -c conda-forge PyCTest
```

For some more information about using Conda, please refer to the docs.

5.2.3 Installing from source with Conda

Sometimes an adventurous user may want to get the source code, which is always more up-to-date than the one provided by Conda (with more bugs of course!).

For this you need to get the source from the PyCTest repository on GitHub. Download the source to your local computer using git by opening a terminal and running:

```
$ git clone https://github.com/jrmadsen/pyctest.git
```

in the folder where you want the source code. This will create a folder called *PyCTest* which contains a copy of the source code.

Installing dependencies

You will need to install all the dependencies listed in `requirements.txt` or `meta.yaml` files. For example, requirements can be installed using Conda by running:

```
$ conda install --file requirements.txt
```

After navigating to inside the *PyCTest* directory, you can install PyCTest by building/compiling the shared libraries and running the install script:

```
$ python build.py
$ pip install .
```

Common issues

No issues with the current build system have been reported.

5.2.4 Importing PyCTest

When importing, it is best to import PyCTest before importing numpy. See this thread for details.

5.3 API reference

This section contains the API reference and usage information for PyCTest.

PyCTest Modules:

5.3.1 pyctest.pyctest

5.3.2 pyctest.pycmake

5.3.3 pyctest.helpers

5.3.4 pyctest.cmake

Functions:

5.3.5 pyctest.ctest

Functions:

5.3.6 pyctest.cpack

Functions:

5.4 FAQ

Here's a list of questions.

Questions

5.4.1 How can I report bugs?

The easiest way to report bugs or get help is to open an issue on GitHub. Simply go to the project GitHub page, click on Issues in the right menu tab and submit your report or question.

5.4.2 Which platforms are supported?

PyCTest supports Linux, Mac OS X, and Windows.

5.5 Credits

CHAPTER 6

License

The project is licensed under the MIT license.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P