PyCTest Documentation

Release 0.0.12

Jonathan R. Madsen

Jan 16, 2019

Contents:

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

CHAPTER 1

Build Status

CHAPTER 2

Anaconda (conda-forge)

CHAPTER 3

Anaconda (jrmadsen)

CHAPTER 4

Features

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

CHAPTER 5

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 6

Table of Contents

6.1 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.

6.1.1 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
import sys
import platform
import pyctest.pyctest as pyctest
```

(continues on next page)

(continued from previous page)

```
import pyctest.helpers as helpers

parser = helpers.ArgumentParser("ProjectName", source_dir=os.getcwd(),binary_dir=os.
↪getcwd(), vcs_type="git")
args = parser.parse_args()

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

test = pyctest.test()
test.SetName("unittest")
test.SetCommand(["nosetests"])

pyctest.run()
```

Example for autotools 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

parser = helpers.ArgumentParser("ProjectName", source_dir=os.getcwd(), binary_dir=os.
↪getcwd(),
                                vcs_type="git")
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.CONFIGURE_COMMAND = "./configure"
pyctest.BUILD_COMMAND = "make -j{}".format(mp.cpu_count())

# alternate test declaration format
pyctest.test("unittest", ["./run-testing.sh"])

pyctest.run()
```

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

project = "PyCTestDemo"
binary_dir = os.path.join(os.getcwd(), "{}-build".format(project))
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())

pyctest.test("unittest", ["./run-testing.sh"])

pyctest.run()
```

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

Direct Access to CMake/CTest/CPack Executables

- `python -m pyctest.cmake <ARGS>` == `cmake <ARGS>`
- `python -m pyctest.ctest <ARGS>` == `ctest <ARGS>`
- `python -m pyctest.cpack <ARGS>` == `cpack <ARGS>`

Following Python code:

```
from pyctest.ctest import CTest
from pyctest.cmake import CMake
from pyctest.cpack import CPack
```

(continues on next page)

(continued from previous page)

```
CMake({"CMAKE_BUILD_TYPE":"Release"}, os.getcwd(), "-G", "Ninja")
CTest("--build-and-test", os.getcwd(), "-VV")
CPack("-G", "TGZ")
```

is equivalent to the following shell commands:

```
cmake -DCMAKE_BUILD_TYPE=Release ${PWD} -G Ninja
ctest --build-and-test ${PWD} -VV
cpack -G TGZ
```

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

6.1.2 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 platform

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()
```

(continues on next page)

(continued from previous page)

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

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

    # create a second test
    pyctest.test("hostname", ["hostname"], {"TIMEOUT": "10"})

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

```
##############################################
#  _____  _  _  ___  _____  _____  _____  _____  #
# (  _  \( \/ )/ __)(_  _)(  __)/ ___)(_  _)  #
#  ) __/ )  /( (__   )(   ) _) \___ \  )(   #
# (__)  (__/  \___) (__) (_____)(____/ (__)  #
#                                            #
##############################################

PyCTest args: []
CTest args: []
CMake args: []
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/docs/pycm-test/
↪CTestTestfile.cmake"...
Generating test "list_directory"...
Generating test "hostname"...
-- STAGES = Start;Update;Configure;Build;Test;Coverage;MemCheck
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Running CTEST_START stage...
Run dashboard with model Continuous
Source directory: /Users/jrmadsen/devel/c++/pyctest-master/docs/pycm-test
Build directory: /Users/jrmadsen/devel/c++/pyctest-master/docs/pycm-test
Track: Continuous
Reading ctest configuration file: /Users/jrmadsen/devel/c++/pyctest-master/docs/pycm-
↪test/CTestConfig.cmake
Site: JRM-macOS-DOE.local.dhcp.lbl.gov
Build name: [Darwin macOS 10.14.2 x86_64] [Python 3.7.0]
Use Continuous tag: 20190116-2239
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_UPDATE stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_CONFIGURE stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_BUILD stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Running CTEST_TEST stage...
Test project /Users/jrmadsen/devel/c++/pyctest-master/docs/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
```

(continues on next page)

(continued from previous page)

```
Total Test time (real) =   0.01 sec
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_COVERAGE stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_MEMCHECK stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Skipping CTEST_SUBMIT stage...
-- [[Darwin macOS 10.14.2 x86_64] [Python 3.7.0]] Finished Continuous Stages (Start;
↪Update;Configure;Build;Test;Coverage;MemCheck)
```

6.2 Installation

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

Contents:

6.2.1 Supported Environments

PyCTest is tested, built, and distributed for python 2.7, 3.6, and 3.7 on Linux/macOS through conda-forge. Windows support is possible but Anaconda compiler issues within conda-forge with respect to `std::unique_ptr` are currently causing issues.

6.2.2 Installing from Conda (Recommended)

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.

6.2.3 Installing from source

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.

Source installation is also available through PyPi:

```
$ pip install -vvv pyctest
```

Installing dependencies

You will need a C compiler, C++ compiler, CMake, Git, OpenSSL, and Curl on your system. Generally, these packages already exist on your system. The C++ compiler requires support for C++11, in particular it needs to support lambdas and `std::unique_ptr`.

After navigating to inside the *pyctest* directory, you can install PyCTest by building/compiling the shared libraries and either of the following standard Python installation commands:

```
$ pip install -vvv .
$ python setup.py install
```

Common issues

- Lack of full C++11 support, particularly *std::unique_ptr*

6.2.4 Importing PyCTest

In general, the base module is not utilized directly. The following import scheme is generally simplest:

```
import pyctest as _pyctest
import pyctest.pyctest as pyctest
import pyctest.pycmake as pycmake
import pyctest.helpers as helpers
from pyctest.cmake import CMake
from pyctest.ctest import CTest
from pyctest.cpack import CPack

print(_pyctest.version)
CMake('--version')
CTest('--version')
CPack('--version')
```

6.3 API reference

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

PyCTest Modules:

6.3.1 pyctest.pyctest

—

pyctest.pyctest.**GetGitBranch**(*dir: str=None*) → str
Get the branch name of a git repo

pyctest.pyctest.**add_note**(*dir: str=''*, *file: str=''*, *clobber: bool=False*) → None
Add a note to the dashboard

pyctest.pyctest.**add_presubmit_command**(*dir: str=''*, *cmd: list=[]*, *clobber: bool=False*) → None
Add a command to be executed before submission

pyctest.pyctest.**add_test**(*arg0: object*) → None
Add a test

class pyctest.pyctest.**cache**
Bases: pybind11_builtins.pybind11_object

Cache types

BOOL = cache.BOOL

FILEPATH = cache.FILEPATH

INTERNAL = cache.INTERNAL

NONE = cache.NONE

PATH = cache.PATH

STRING = cache.STRING

class pyctest.pyctest.**command**
Bases: pybind11_builtins.pybind11_object

Run a command – works like execute_process(...)

AddCommand(*self: object*, *arg0: list*) → None
Add a command

Command(*self: object*) → str
Get the argument list

Error(*self: object*) → str
Get the error string

Exec(*self: object*, *args: list=[]*) → None
Execute (i.e. run)

Execute(*self: object*, *args: list=[]*) → None
Execute (i.e. run)

Output(*self: object*) → str
Get the output string

Result(*self: object*) → str
Get the result (return code) string

Results(*self: object*) → str
Get the results

SetEncoding (*self: object*, *arg0: pyctest.pyctest.encoding*) → None
Set the process encoding

SetErrorFile (*self: object*, *arg0: str*) → None
Set the error file

SetErrorQuiet (*self: object*, *arg0: bool*) → None
Suppress error

SetErrorStripTrailingWhitespace (*self: object*, *arg0: bool*) → None
Strip trailing whitespace from error

SetInputFile (*self: object*, *arg0: str*) → None
Set the input file

SetOutputFile (*self: object*, *arg0: str*) → None
Set the output file

SetOutputQuiet (*self: object*, *arg0: bool*) → None
Suppress output

SetOutputStripTrailingWhitespace (*self: object*, *arg0: bool*) → None
Strip trailing whitespace from output

SetTimeout (*self: object*, *arg0: str*) → None
Set the process timeout

SetWorkingDirectory (*self: object*, *arg0: str*) → None
Set the working directory

pyctest.pyctest.**copy_files** (*files: list=[]*, *from_dir: str=''*, *target_dir: str=''*) → None
Helper method to copy files over to binary dir

class pyctest.pyctest.**encoding**
Bases: pybind11_builtins.pybind11_object

Encoding types

ANSI = encoding.ANSI

Auto = encoding.Auto

None = encoding.None

OEM = encoding.OEM

UTF8 = encoding.UTF8

pyctest.pyctest.**exe_path** () → str
Path to ctest executable

pyctest.pyctest.**execute** (*args: List[str]=[]*) → int
Directly run ctest

pyctest.pyctest.**find_test** (*arg0: str*) → pyctest.pyctest.test
Find a test by name

pyctest.pyctest.**generate_config** (*output_directory: str=''*) → None
Generate CTestConfig.cmake, CTestCustom.cmake, and copy over PyCTest CMake files

pyctest.pyctest.**generate_test_file** (*output_directory: str=''*) → None
Generate a CTestTestfile.cmake

pyctest.pyctest.**git_checkout** (*repo_url: str*, *source_dir: str*, *branch: str='master'*, *update: bool=True*) → None
Perform git checkout a code and optionally update if already exists

class pyctest.pyctest.**ostream_redirect**
Bases: pybind11_builtins.pybind11_object

pyctest.pyctest.**remove_test**(*arg0: object*) → None
Remove a test

pyctest.pyctest.**run**(*args: List[str]=[]*, *working_directory: str=''*) → None
Run CTest

class pyctest.pyctest.**set**
Bases: pybind11_builtins.pybind11_object

Set a variable – works like set(...)

class pyctest.pyctest.**test**
Bases: pybind11_builtins.pybind11_object

CTest test object – works like add_test(...)

GetCommand(*self: object*) → list
Get the command for the test

GetName(*self: object*) → str
Get test name

GetProperty(*self: object*, *arg0: str*) → str
Get a test property

GetPropertyAsBool(*self: object*, *arg0: str*) → bool
Get property as boolean

SetCommand(*self: object*, *arg0: list*) → None
Set the command for the test

SetName(*self: object*, *arg0: str*) → None
Set test name

SetProperty(*self: object*, *arg0: str*, *arg1: str*) → None
Set a test property

6.3.2 pyctest.pycmake

—

class pyctest.pycmake.**cmake**
Bases: pybind11_builtins.pybind11_object

pyctest.pycmake.**exe_path**() → str
Path to cmake executable

pyctest.pycmake.**execute**(*args: List[str]=[]*) → int
Directly run cmake

class pyctest.pycmake.**ostream_redirect**
Bases: pybind11_builtins.pybind11_object

pyctest.pycmake.**run**(*args: List[str]=[]*, *binary_dir: str='.'*) → None
Run CMake

6.3.3 pyctest.helpers

—

class pyctest.helpers.**Conda**
Bases: object

Utility class for getting relevant conda environment variables

pyctest.helpers.**FindExePath**(*name*, *path=None*)
Function for finding the path to an executable

pyctest.helpers.**RemovePath**(*path*)
Function for safely removing a folder

pyctest.helpers.**Cleanup**(*path=None*, *extra=[]*, *exclude=[]*)
This function removes PyCTest files/folders that are copied into or generated in the given path during running PyCTest

Parameters

- **path** (*str, optional*) – alternative path (default=pyctest.SOURCE_DIRECTORY)
- **extra** (*list, optional*) – any additional files/folders to remove
- **exclude** (*list, optional*) – use this to exclude deleting files/folders

pyctest.helpers.**GetHomePath**()
Get the user's home drive for the operating system

pyctest.helpers.**GetSystemVersionInfo**()
Version info for operating system

class pyctest.helpers.**ArgumentParser**(*project_name, source_dir, binary_dir, vcs_type=None, cleanup_extra=[], cleanup_exclude=[], drop_method='https', drop_site='cdash.nersc.gov', drop_location='/submit.php?project=${CTEST_PROJECT_NAME}', drop_site_user=None, drop_site_password=None, use_launchers=False, submit_retry_count=1, submit_retry_delay=30, curl_options=None, cdash_version=None, nightly_start_time='01:00:00 UTC', jobs=1, submit=False, stages=['Start', 'Update', 'Configure', 'Build', 'Test', 'Coverage', 'MemCheck'], trigger='None', site='build-8403268-project-257458-pyctest', ctest_args=['-V'], mode='Stages', checkout_command=None, update_command=None, configure_command=None, build_command=None, coverage_command=None, memcheck_command=None, python_exe='/home/docs/checkouts/readthedocs.org/user_builds/pyctest/conda/ build_type=None, model='Continuous', build_choices=['Release', 'RelWithDebInfo', 'Debug', 'MinSizeRel'], prog=None, usage=None, description="PyCTest argparse. Arguments after first '–' are passed directly to CTest, arguments after second '–' are passed directly to CMake", epilog=None, parents=[], formatter_class=<class 'pyctest.helpers.LineWrapRawTextHelpFormatter'>, prefix_chars='-', fromfile_prefix_chars=None, argument_default=None, conflict_handler='resolve', add_help=True*)

Bases: argparse.ArgumentParser

Adds common command-line options for PyCTest

Note: Inherit from this class to create custom options

Parameters

- **project_name** (*str*) – Name of the project
- **source_dir** (*str*) – relative path to the source code
- **binary_dir** (*str*) – relative path to the binary (build) directory
- **mode** (*str='Continuous', optional*) – Workflow setting Options: Start, Update, Configure, Build, Test, Coverage, MemCheck, Submit, Stages
- **stages** (*list=['Start', 'Update', 'Configure', 'Build', 'Test', 'Coverage', 'MemCheck']*) – In "Stages" mode, these are the workflow components to execute If a command for the stage is not set, e.g. CTEST_COVERAGE_COMMAND is not set, then the stage is skipped
- **trigger** (*str='None', optional*) – Deprecated, has no effect
- **model** (*str='Continuous', optional*) – Submission track. Common choices: Continuous, Nightly, Experimental
- **site** (*str = platform.node(), optional*) – The ID of the submission site
- **jobs** (*int = 1, optional*) – Number of tests to execute in parallel

- **submit** (*bool, optional*) – Enable/disable submission by default
- **vcs_type** (*str, optional*) – Version control type: bzr, cvs, git, hg, p4, svn
- **checkout_command** (*str, optional*) – Define the default checkout command
- **update_command** (*str, optional*) – Define the default update command
- **configure_command** (*str, optional*) – Define the default configure command
- **build_command** (*str, optional*) – Define the default configure command
- **coverage_command** (*str, optional*) – Define the default coverage command
- **memcheck_command** (*str, optional*) – Define the default memory check command
- **build_choices** (*str, optional*) – If running CMake, define the build type [Release, RelWithDebInfo, Debug, MinSizeRel]
- **use_launchers** (*bool, optional*) – For build trees generated by CMake using one of the Makefile Generators or the Ninja generator, specify whether the CTEST_USE_LAUNCHERS feature is enabled by the CTestUseLaunchers module (also included by the CTest module). When enabled, the generated build system wraps each invocation of the compiler, linker, or custom command line with a "launcher" that communicates with CTest via environment variables and files to report granular build warning and error information. Otherwise, CTest must "scrape" the build output log for diagnostics.
- **cleanup_extra** (*list, optional*) – When *–pyctest-cleanup* or *–pyctest-clean-first* is invoked, these additional files/folders will be removed
- **cleanup_exclude** (*list, optional*) – When *–pyctest-cleanup* or *–pyctest-clean-first* is invoked, these additional files/folders will be excluded from removal
- **cdash_version** (*str=None, optional*) – Specify the version of CDash on the server
- **submit_retry_count** (*int=1, optional*) – Specify a number of attempts to retry submission on network failure
- **submit_retry_delay** (*int=30, optional*) – Specify a delay before retrying submission on network failure
- **curl_options** (*list=None, optional*) – Curl options: 'CURLOPT_SSL_VERIFYPEER_OFF' and 'CURLOPT_SSL_VERIFYHOST_OFF'
- **drop_location** (*str="/submit.php?project={}".format(project_name), optional*) – The path on the dashboard server to send the submission
- **drop_method** (*str="https", optional*) – Specify the method by which results should be submitted to the dashboard server. The value may be cp, ftp, http, https, scp, or xmlrpc (if CMake was built with support for it)
- **drop_site** (*str="cdash.nersc.gov", optional*) – The dashboard server name (for ftp, http, and https, scp, and xmlrpc)
- **drop_site_password** (*str=None, optional*) – The dashboard server login password, if any (for ftp, http, and https).
- **drop_site_user** (*str=None, optional*) – The dashboard server login user name, if any (for ftp, http, and https).
- **nightly_start_time** (*str="01:00:00 UTC", optional*) – In the Nightly dashboard mode, specify the "nightly start time". With centralized version control systems (cvs and svn), the Update step checks out the version of the software as of this time so that multiple clients

choose a common version to test. This is not well-defined in distributed version-control systems so the setting is ignored.

parse_args (**args*, ***kwargs*)
Parse the arguments

Note: Arguments following first double-dash ("–") are passed directly to *ctest*

Note: Arguments following second double-dash ("–") are passed directly to *cmake*

process_args (*args*)
Process PyCTest args

6.3.4 pyctest.cmake

Direct interface to CMake executable:

```
python -m pyctest.cmake [ARG [ARG]]
```

---

pyctest.cmake.**CMake** (**args*, ***kwargs*)
Function for direct access to CMake

Parameters

- **args** (*list*) – List of CMake arguments (added to cmd-line in order after *kwargs*)
- **kwargs** (*dict*) – List of CMake variable definitions (added before *args*)

Example

```
pyctest.cmake.CMake("--build", os.getcwd(), "--target", "all")
```

```
_kwargs['CMAKE_BUILD_TYPE'] = 'Debug'
_kwargs['CMAKE_INSTALL_PREFIX'] = '/usr/local'
_kwargs['BUILD_SHARED_LIBS'] = 'ON'
_args = [ os.path.dirname(os.getcwd()), '-G', 'Ninja' ]
pyctest.cmake.CMake(_args, _kwargs)
```

6.3.5 pyctest.ctest

Direct interface to CTest executable

```
python -m pyctest.ctest [ARG [ARG]]
```

---

pyctest.ctest.**CTest** (**args*, ***kwargs*)
Function for direct access to CTest

Parameters

- **args** (*list*) – List of CTest arguments (added to cmd-line in order after *kwargs*)
- **kwargs** (*dict*) – List of CTest variable definitions (added before *args*)

Example

```
pyctest.ctest.CTest('-V', '-S', 'CTestScript.cmake', '-j1')
```

```
_kwargs['STAGES'] = 'Start;Update;Build;Test;Submit'
_args = [ '-V', '-S', 'Stages.cmake', '-j1' ]
pyctest.ctest.CTest(_args, _kwargs)
```

6.3.6 pyctest.cpack

Direct interface to CPack executable:

```
python -m pyctest.cpack [ARG [ARG]]
```

---

pyctest.cpack.**CPack**(**args*, ***kwargs*)
Function for direct access to CPack

Parameters

- **args** (*list*) – List of CPack arguments (added to cmd-line in order after *kwargs*)
- **kwargs** (*dict*) – List of CPack variable definitions (added before *args*)

Example

```
pyctest.cpack.CPack("-P", package_name, "-V")
```

```
_kwargs['CPACK_PACKAGE_NAME'] = 'pyctest'
_kwargs['CPACK_PACKAGE_VERSION'] = '0.0.12'
_args = [ '-G', 'TGZ' ]
pyctest.cpack.CPack(_args, _kwargs)
```

PyCTest is an extension that directly hooks into the CMake/CTest libraries. Instead of generating CMake/CTest files, it use the library itself. Since the CMake version is directly controlled as a git submodule, there are no external compatibility issues.

version_info	List of version fields
build_info	Build information
version	Version string
cmake_executable	Path to CMake executable
ctest_executable	Path to CTest executable
cpack_executable	Path to CPack executable

pyctest.**print_display**()
Prints the PyCTest banner at first import

Set PYCTEST_VERBOSE=<INT> or PYCTEST_BANNER=<BOOL or INT> to control banner settings. Default is ON.

pyctest.**version_info = (0, 0, 12)**
List of version fields

pyctest.**build_info = {'build_type': 'MinSizeRel', 'compiler': '/home/conda/feedstock_root**
Build information

pyctest.**version = '0.0.12'**
Version string

pyctest.**cmake_executable = '/home/docs/checkouts/readthedocs.org/user_builds/pyctest/conda/**
Path to CMake executable

pyctest.**ctest_executable = '/home/docs/checkouts/readthedocs.org/user_builds/pyctest/conda/**
Path to CTest executable

pyctest.**cpack_executable = '/home/docs/checkouts/readthedocs.org/user_builds/pyctest/conda/**
Path to CPack executable

6.4 FAQ

Here's a list of questions.

Questions

6.4.1 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.

6.4.2 Which platforms are supported?

PyCTest supports Linux, Mac OS X, and Windows.

6.5 Credits

CHAPTER 7

License

The project is licensed under the MIT license.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

T

U

V