CachePath Documentation

Release 1.1.1

Hayden Flinner

Dec 09, 2018

Contents

A small package for pythonic parameterized cache paths.

CHAPTER 1

Getting Started

Install: `pip install cachepath`

Import: `from cachepath import CachePath, TempPath, Path`

Docs: ReadTheDocs | API doc is here

Why?

1. Integrates `pathlib` with `tempfile.gettempdir` and `shutil.rmtree` by providing `TempPath` and `Path.rm()`:

```
path = TempPath()
path.rm()
# or would you rather..
path = None
with tempfile.NamedTemporaryFile(delete=False) as f:
    path = Path(f.name)
# Only now can we use Path. If we tried using it within the With
# (for example for path.read_text()), we'd break on Windows
path.unlink()  # only if file, doesn't work on folders
```

2. Wraps `pathlib` import for Py2/3 compat. (not in `six`!):

```
from cachepath import Path
# or
try: from pathlib import Path; except ImportError: from pathlib2 import Path
```

3. Provides CachePath, which lets you quickly get a parameterized temp filename, with all folders automatically created:

```
r = CachePath(date, userid, 'expensive_results.txt')
assert (r == Path('/tmp/', date, userid, 'expensive_results.txt')
        and r.parent.exists())
r.rm()  # File remove
r.parent.rm()  # Symmetric with folder remove!
```

(continues on next page)

(continued from previous page)

```
# Without cachepath
p = Path(tempfile.gettempdir(), date, userid, 'expensive_results.txt').
# Don't update timestamp if it already exists so that we don't cause
# Make-like tools to always think something's changed
if not p.parent.exists():
    p.parent.mkdir(parents=True, exist_ok=True)

p.unlink()  # Why is it .unlink() instead of .remove()?
# Why .remove and .unlink, but mkdir instead of createdir?
p.parent.remove()
# .remove() might throw because there was another file in the folder,
# but we didn't care, they're tempfiles!
import shutil
shutil.rmtree(p.parent)
```

Why, but longer:

Do you need a temp path to pass to some random tool for its logfile? Behold, a gaping hole in `pathlib`:

```
import tempfile
import os
try: from pathlib import Path; except ImportError: from pathlib2 import Path
def get_tempfile():
    fd, loc = tempfile.mkstemp()
    os.close(fd)  # If we forgot do this, it would stay open until process exit
    return Path(loc)

# Easier way
from cachepath import TempPath
def get_tempfile():
    return TempPath()  # Path('/tmp/213kjdsrandom')
```

But this module is called cachepath, not temppath, what gives?

Suppose I'm running that same imaginary tool pretty often, but I'd like to skip running it if I already have results for a certain day. Just sticking some identifying info into a filename should be good enough. Something like `Path('/tmp/20181204_toolresults.txt')`

```
# try: from pathlib import Path; except ImportError: from pathlib2 import Path
# We'll cheat a little to get py2/3 compat without so much ugliness
from cachepath import Path
import tempfile
def get_tempfile(date):
    filename = '{}_toolresults.txt'.format(date)
    return Path(tempfile.gettempdir(), filename)

# Easier to do this...
from cachepath import CachePath
def get_tempfile(date):
    return CachePath(date, suffix='.txt')
```

Not bad, but not great. But our requirements changed, let's go a step further.

Now I'm running this tool *a lot*, over a tree of data that looks like this:

```
2018-12-23
    person1
    person2
2018-12-24
    person1
2018-12-25
    person1
```

I want my logs to be structured the same way. How hard can it be?

```
2018-12-23/
    person1_output.txt
    person2_output.txt
2018-12-24/
    person1_output.txt
2018-12-25/
    person1_output.txt
```

Let's find out:

```
# Let's get the easy way out of the way first :)
def get_path(date, person):
    return CachePath(date, person, suffix='_output.txt')
    # Automatically ensures /tmp/date/ exists when we create the CachePath!

# Now the hard way
def get_path(date, person):
    personfilename = '{p}_output.txt'.format(p=person)
    returning = Path(tempfile.gettempdir())/date/personfilename
    # Does this mkdir update the modified timestamp of the folders we're in?
    # Might matter if we're part of a larger toolset...
    returning.parent.mkdir(exist_ok=True, parents=True)
    return returning
```

Suppose we hadn't remembered to make the `$date/` folders. When we passed the Path out to another tool, or tried to .open it, we may have gotten a Permission Denied error on Unix systems rather than the "File/Folder not found" you might expect. With CachePath, this can't happen. Creating a CachePath implicitly creates all of the preceding directories necessary for your file to exist.

Now, suppose we found a bug in this external tool we were using and we're going to re-run it for a day. How do we clear out that day's results so that we can be sure we're looking at fresh output from the tool? Well, with CachePath, it's just:

```
def easy_clear_date(date):
    CachePath(date).clear()  # rm -r /tmp/date/*
```

But if you don't have cachepath, you'll find that most Python libs play it pretty safe when it comes to files. Path.remove() requires the folder to be empty, and doesn't provide a way to empty the folder. Not to mention, what if our results folders had special permissions, or was actually a symlink, and we had write access but not delete? Oh well, let's see what we can do:

```
def hard_clear_date(date):
    # We happen to know that date is a folder and not a file (at least in our
    # current design), so we know we need some form of .remove() rather than
    # .unlink(). Unfortunately, pathlib doesn't offer one for folders with
    # files still in them. If you google how to do it, you will find plenty of
    # answers, one of which is a pure pathlib recursive solution! But we're lazy,
```

(continues on next page)

(continued from previous page)

```
    # so lets bring in yet another module:
    p = Path(tempfile.gettempdir(), date)
    import shutil
    if p.exists():
        shutil.rmtree(p)
    p.mkdir(exist_ok=True, parents=True)
    # This still isn't exactly equivalent to CachePath.clear(), because we've
    # lost whatever permissions were set on the date folder, and if it were
    # actually a symlink to somewhere else, that's gone now.
```

Convinced yet? `pip install cachepath` or copy the source into your local `utils.py` (you know you have one.)

API doc is here.

By the way, as a side effect of importing `cachepath`, all Paths get the ability to do `rm()` and `clear()`.

CHAPTER 2

Shameless Promo

Find yourself working with paths a lot in cmd-line tools? You might like invoke and/or magicinvoke!

2.1 Usage

To use CachePath in a project:

```
from cachepath import CachePath, Path
```

2.1.1 Changing Storage Location

```
import cachepath
cachepath.location = './cache'  # Once, anywhere. Default is tempfile.gettempdir()

# The order that you import / assign to .location doesn't matter yet
from cachepath import CachePath, Path
```

2.1.2 Indepth Example

Lets hack together a cache for a website scraper. This could be useful if you were working on your parsing logic or want the files used in the process to to be available on disk for later debugging instead of just staying in memory until a crash.

If this seems like a lot, try reading *Getting Started* first.

```
from cachepath import CachePath, Path
from itertools import takewhile

def get_scraped_ebay_stats(user, product_id):
```

(continues on next page)

(continued from previous page)

```
    # '/tmp/ebay/user/product_id.html'
    return dumb_parser(CachePath('ebay', user, product_id, suffix='html'))

def clear_cache(user=None, product_id=None):
    args = takewhile(lambda x: x != None, ('ebay', user, product_id))
    return CachePath(*args).clear()

def get_tempfile():
    return CachePath()


# Without cachepath
try:
   from pathlib import Path
except:  # py2
   from pathlib2 import Path

# If we don't put everything under a folder, we'll try to rm -rf /tmp/ later..
import tempfile
ebay = Path(tempfile.gettempdir())/'ebay'

import shutil  # Needed to remove a folder recursively

def get_scraped_ebay_stats(user, product_id):
    p = Path(ebay, user, product_id).with_suffix('html')
    p.parent.mkdir(exist_ok=True, parents=True) # Does this update timestamp
    # and thus break Make-like tools? Turns out no (on Linux at least),
    # but who would have known?
    dumb_parser(p)

def clear_cache(user=None, product_id=None):
    if product_id:
        # if product_id ever starts pointing to a folder, this will break
        return (ebay/user/product_id).unlink()
    p = ebay/user if user else ebay
    # Not exactly equivalent: cachepath.clear() just removes the contents of
    # a folder, it doesn't remove and recreate. Helpful to avoid messing up
    # permissions or requiring a personal folder to place things.
    shutil.rmtree(p)  # No rm  -rf for folder paths in pathlib

def get_tempfile():
    return tempfile.mkstemp()

def dumb_parser(html_path):
    if not p.exists():
        sh('wget {}'.format(p))
    return myprocess(p.read_text())
```

2.2 cachepath module

Package that provides CachePath, as well as exporting a Python2/3 compatible Path.

`cachepath.`**`TempPath`**(*cls*, **args*, ***kwargs*)

See CachePath for more details:

```
TempPath('x', 'y', suffix='.z')
# Is safer, easier, and explains your intent to always have a new file better than
CachePath('x', 'y', 'randomstringhere', suffix='.z')

# However,
TempPath() == CachePath()  # for convenience
```

class cachepath.**CachePath**

Bases: `pathlib2.Path`

Construct a CachePath from one or several strings/Paths.

Constructing a CachePath automatically creates the preceding folders necessary for the file to exist, if they're not already there.

CachePaths also have a few helper methods:

`CachePath().clear()`

`CachePath().rm()`

By accident, these methods are also attached to regular Paths after constructing a CachePath, but it's not recommended to depend on this behavior.

Examples

Basic Usage:

```
CachePath() == '/tmp/xyz123randomfile'
CachePath('myfilename') == '/tmp/myfilename'
CachePath('myfolder', dir=True) == '/tmp/myfolder/'
TempPath('myfolder') == '/tmp/myfolder/zsdskjrandom'
```

Multi-component Paths:

```
p = CachePath('date/processed_data', dir=True)
# Or, Alternate constructor to avoid {}/{}.format()
p = CachePath('date', 'processed_data', dir=True)
```

For an example of real usage, here's a quick cache for an arbitrary function/arg combo

```
def get_scraped_ebay_stats(product_id):
    p = CachePath('ebay_results/{}'.format(product_id))
    if not p.exists():
       sh('wget {}'.format(p))
    return parser.parse(p.read_text())
```

Parameters

- **args** (*[str], optional*) – List of strings to join for Path. If None, `getempfile` is used.
- **dir** (*bool, optional*) – Is the path intended to be a directory? Useful when you just need a tempdir for lots of files, and you don't want to make a CachePath out of each.

```
d = CachePath(date, dir=True)
(d/'tool1results').touch()
```

(continues on next page)

(continued from previous page)

```
(d/'tool2results').touch()
list(d.iterdir()) == ['tool1results', 'tool2results']
```

- **suffix** (*str, optional*) – Appended to the last path in *args, i.e. CachePath('a', 'b', suffix='_long_suff.txt') == '/tmp/a/b_long_suff.txt'
- **mode** (*int, optional, default=0o777*) – Mode to create folder with, if it doesn't already exist.

`cachepath.clear(path)`

Clear the file/dir, leaving it empty (dir) or 0 length (file).

Monkey-patched onto all Paths on import. Creates file if path doesn't exist.

`cachepath.rm(path)`

Delete the file/dir, even if it's a dir with files in it.

Monkey-patched onto all Paths on import. Does nothing if path doesn't exist.

2.3 History

2.3.1 1.0.0 (2018-12-08)

- Big doc updates. 1.0.0 to symbolize SemVer adherence.

2.3.2 0.1.0 (2018-12-08)

- First release on PyPI. Adds CachePath, TempPath, Path.

2.4 Contributing

Have an idea to contribute, or hungry to fix bugs? https://github.com/haydenflinner/cachepath/issues.

2.4.1 Setting Up for Development

Here's how to set up *cachepath* for local development.

1. Fork the *cachepath* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/cachepath.git
```

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

```
$ mkvirtualenv cachepath
$ cd cachepath/
$ 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:

```
$ py.test
$ tox
```

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.

2.4.2 Tips

To run a subset of tests:

```
$ py.test tests.test_cachepath
```

2.4.3 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

2.5 All Tests

Useful to see how CachePaths behave or to see if a case you're concerned about is tested for.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""Tests for `cachepath` package."""

import pytest
from cachepath import CachePath, Path

@pytest.fixture
def cachepath(tmpdir):
    import cachepath
    cachepath.location = tmpdir
    return cachepath
```

(continues on next page)

(continued from previous page)

```
def test_works(cachepath):
    p = cachepath.CachePath('lolfile')
    p.open('w').writelines([u'hi'])
    assert 'hi' == p.read_text()

def test_rm_clear_file(cachepath):
    p = cachepath.CachePath()
    p.write_text(u'lol')
    p.clear()
    assert p.read_text() == ''
    p.rm()
    assert not p.exists()

def test_clear_folder(cachepath):
    p = cachepath.CachePath('lolfolder', dir=True)
    # We would get surprising behavior if / created CachePaths given the side
    # effecting constructor, so don't do that!
    (p/'file').touch()
    p.clear()
    assert len(list(p.iterdir())) == 0
    assert not (p/'file').exists()
    p.rm()
    assert not p.exists()

def test_tmp(cachepath, tmpdir):
    p = cachepath.TempPath('folder/path/here')
    p.touch()
    assert str(tmpdir) in str(p.parent)

def test_can_change_location():
    # Old test, now the rest of the tests depend on this to work, but can't hurt.
    import cachepath
    cachepath.location = './dummy'
    assert cachepath.CachePath('innerfile') == Path('./dummy/innerfile')

def test_import_side_effects(tmpdir):
    # This isn't encouraged to depend on, but if someone does, would rather not break␣
↪them.
    import cachepath
    tmpdir = str(tmpdir)
    Path(tmpdir).clear()
    Path(tmpdir, 'test_file').rm()

@pytest.mark.xfail
def test_tmp_removes_self(cachepath, tmpdir):
    # TODO
    p = cachepath.TempPath('hi', delete=True)
    with p:
        pass
    assert not p.exists()  # Might throw?
```

Python Module Index

C

Index