A Path Less Traveled

DAVID BEAZLEY

David Beazley is an open source developer and author of the *Python Essential Reference* (4th Edition, Addison-Wesley, 2009). He is also known as the creator of Swig (http://www.swig.org) and Python Lex-Yacc (http://www.dabeaz.com/ply.html). Beazley is based in Chicago, where he also teaches a variety of Python courses dave@dabeaz.com

If you're like me, you've probably written a Python script or two that had to manipulate pathnames. For that, you've probably used the much beloved `os.path` module—and perhaps the `glob` module. And let's not forget some of their friends such as `fnmatch`, `shutil`, `subprocess`, and various bits of functionality in `os`. Aw, let's face it, who are we kidding here? Pathname handling in Python is an inexplicable mess, has always been a mess, and will always continue to be a mess. Or will it?

In this installment, I take a look at the new `pathlib` standard library module added to Python 3.4 [1]. More than 10 years in the making, it aims to change the whole way that you manipulate files and pathnames—hopefully, for the better.

Classic Pathname Handling

In programs that need to manipulate files and pathnames, certain tasks seem to arise over and over again. For example, splitting pathname components apart, joining paths together, dealing with file extensions, and more. To further complicate matters, POSIX and Windows systems don't agree on basic features such as the path separator (/ vs. \) or case sensitivity. So if you try to write all of the code yourself, it quickly becomes a mess. For these tasks, the `os.path` module is usually the recommended solution. It mainly provides common operations that you might apply to strings containing file names and does so in a platform-independent manner. For example:

```
>>> filename = '/Users/beazley/Pictures/img123.jpg'
>>> import os.path

>>> # Get the base directory name
>>> os.path.dirname(filename)
'/Users/beazley/Pictures'

>>> # Get the base filename
>>> os.path.basename(filename)
'img123.jpg'

>>> # Split a filename into directory and filename components
>>> os.path.split(filename)
('/Users/beazley/Pictures', 'img123.jpg')

>>> # Get the filename and extension
>>> os.path.splitext(filename)
('/Users/beazley/Pictures/img123', '.jpg')
>>>

>>> # Get just the extension
>>> os.path.splitext(filename)[1]
'.jpg'
>>>
```

In practice, using these functions gets a bit a more messy. For example, suppose you want to rewrite a file name and change its extension. To do that, you might write code like this:

```
>>> filename
'/Users/beazley/Pictures/img123.jpg'
>>> dirname, basename = os.path.split(filename)
>>> base, ext = os.path.splitext(basename)
>>> newfilename = os.path.join(dirname, 'thumbnails',
base+'.png')
>>> newfilename
'/Users/beazley/Pictures/thumbnails/img123.png'
>>>
```

Actually, all of that code is probably embedded inside some sort of larger task. For example, processing all of the images in an entire directory:

```
import os.path
import glob

def make_thumbnails(dirname, pat):
    filenames = glob.glob(os.path.join(dirname, pat))
    for filename in filenames:
        dirname, basename = os.path.split(filename)
        base, ext = os.path.splitext(basename)
        newfilename = os.path.join(dirname, 'thumbnails',
                                   base+'.png')
        print('Making thumbnail %s -> %s' % (filename, newfilename))
        out = subprocess.check_output(['convert', '-resize',
                                   '100x100', filename, newfilename])

# Example
make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

Here's a more complicated example that recursively walks an entire directory structure, making directories, and launching subprocesses:

```
import os
import os.path
import subprocess
from fntmatch import fnmatch

def make_thumbnails(topdir, pat):
    for path, dirs, files in os.walk(topdir):
        filenames = [filename for filename in files
                        fnmatch(filename, pat)]
        if not filenames:
            continue

        newdirname = os.path.join(path, 'thumbnails')
        if not os.path.exists(newdirname):
            os.makedir(newdirname)

        for filename in filenames:
            base, _ = os.path.splitext(filename)
            newfilename = os.path.join(newdirname, base+'.png')
            origfilename = os.path.join(path, filename)
            print('Making thumbnail %s -> %s' % (origfilename,
            newfilename))
              out = subprocess.check_output(['convert', '-resize',
                          '100x100', origfilename, newfilename])

if __name__ == '__main__':
    make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

Again, if you've written any kind of Python code that manipulates files, you're probably already pretty well familiar with this sort of code (for better or worse).

Past Efforts to Improve Path Handling

Complaints about Python's pathname handling in `os.path` are varied but tend to focus on a couple of common themes. First, there is the fact that the interface doesn't really match other parts of Python, which are usually more object-oriented. Second, a lot of the useful functionality concerning files tends to be spread out over many different standard library modules. As such, file-name handling code becomes more messy than it probably needs to be.

Efforts to improve Python's path handling apparently go back nearly 15 years. To be honest, this is not an aspect of Python that has garnered much of my own attention, but the rejected PEP 355 cites discussions about the matter going as far back as 2001 [2]. The third-party `path` module, created by Jason Orendorff, may be the best-known attempt to clean up some of the mess [3]. With `path`, you create path objects and manipulate them in a more object-oriented manner:

```
>>> from path import path
>>> filename = path('/Users/beazley/Pictures/img123.
jpg')

>>> # Get the base directory name
>>> filename.parent
path(u'/Users/beazley/Pictures')

>>> # Get the base filename
>>> filename.name
path(u'img123.jpg')

>>> # Get the base filename without extension
>>> filename.namebase
u'img123'

>>> # Get the file extension
>>> filename.ext u'.jpg'
>>>
```

path objects can be joined together using the / operator in a way that mimics its use on the file system itself. For example:

```
>>> filename.parent / 'thumbnails' / (filename.
namebase + '.png')
path(u'/Users/beazley/Pictures/thumbnails/img123.png')
>>>
```

path objects include a large variety of other methods related to manipulating files, including globbing, reading, writing, and more. For example:

```
>>> # Read the file as bytes
>>> data = filename.bytes()
>>>

>>> # Remove the file
>>> filename.remove()
path(u'/Users/beazley/Pictures/img123.jpg')
>>>

>>> # Check for existence
>>> filename.exists()
False
>>>

>>> # Walk a directory tree and produce .JPG files
>>> for p in path('/Users/beazley/Pictures').walk('*.
JPG'):
...     print(p)
...
/Users/beazley/Pictures/Foo/IMG_0001.JPG
/Users/beazley/Pictures/Foo/IMG_0002.JPG
/Users/beazley/Pictures/Foo/IMG_0003.JPG
...
/Users/beazley/Pictures/Bar/IMG_1024.JPG
/Users/beazley/Pictures/Bar/IMG_1025.JPG
```

Here is a revised version of the image thumbnail code that uses path.

```
from path import path
import subprocess

def make_thumbnails(topdir, pat):
  topdir = path(topdir)
  for filename in topdir.walk(pattern=pat):
      newdirname = filename.parent / 'thumbnails'
      if not newdirname.exists():
          newdirname.mkdir()
      newfilename = newdirname / (filename.namebase + '.png')
      print('Making thumbnail %s -> %s' % (filename,
                         newfilename))
      out = subprocess.check_output(['convert', '-resize',
                           '100x100', filename, newfilename])

if __name__ == '__main__':
    make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

For various reasons, the path module was never incorporated into the standard library. The main reason may have been the kitchen-sink aspect of the whole implementation. Under the covers, the path object inherits directly from the built-in string type and adds more than 120 additional methods. As a result, it's a kind of "god object" that combines all of the functionality of strings, pathnames, files, and directories all in one place. To emphasize this point, there is the potential for confusion between string and path methods. For example:

```
>>> # A string method
>>> filename.split('/')
  [u'', u'Users', u'beazley', u'Pictures', u'img123.jpg']

>>> # A path method
>>> filename.splitpath()
(path(u'/Users/beazley/Pictures'), u'img123.jpg')
>>>
```

There are even methods for features you might not expect such as cryptographic hashing:

```
>>> filename.read_md5()
'\x98\x05\xdd\x97\xe0\xd3\x1f\xedH*xb\x179\xbf\x18'
>>>
```

It's a legitimate concern to wonder whether it's appropriate for a single object to contain every possible operation that one might think to do with a file—probably not.

Introducing pathlib

Starting in Python 3.4, a new standard library module pathlib was added to manipulate paths. It is the work of Antoine Pitrou and is described in some detail in PEP 428 [4]. As with previous efforts, it takes an object-oriented approach as before by defining a Path class. However, this class no longer derives from built-in strings. It's also much more refined in that it only focuses on functionality related to paths, and not everything that someone might want to do with a file in general.

To illustrate, here are some earlier examples redone using pathlib:

```
>>> from pathlib import Path
>>> filename = Path('/Users/beazley/Pictures/img123.
jpg')

>>> # Get the base directory name
>>> filename.parent
PosixPath('/Users/beazley/Pictures')

>>> # Get the base filename
>>> filename.name
'img123.jpg'
```

```
>>> # Get the file extension
>>> filename.suffix
'.jpg'

>>> # Get the file stem
>>> filename.stem
'img123'

>>> # Get the parts of the filename
>>> filename.parts
('/', 'Users', 'beazley', 'Pictures', 'img123.jpg')
>>>
```

Path also allows the / operator to be used to easily form new pathnames:

```
>>> filename.parent / 'thumbnails' / (filename.stem +
'.png')
PosixPath('/Users/beazley/Pictures/thumbnails/img123.png')
>>>
```

Common operations for replacing/changing parts of the file name are also provided:

```
>>> filename.with_suffix('.png')
PosixPath('/Users/beazley/Pictures/img123.png')
>>> filename.with_name('index.html')
PosixPath('/Users/beazley/Pictures/index.html')
>>>
```

You will notice that in these examples an object of type PosixPath is created. This is system dependent—on Windows an object of type WindowsPath is created instead. Differences in the path implementation are used to support features such as case-sensitivity on the file system. For example, on Windows, you'll find that path comparison works as expected even if the file names have varying case:

```
>>> # Windows case-insensitive path comparison (only works on
Windows)
>>> a = Path('pictures/img123.jpg')
>>> b = Path('PICTURES/IMG123.JPG')
>>> a == b
True
>>>
```

Last, but not least, pathlib provides a few basic functions for querying, directory walking, and other similar operations. For example, you can test whether a file matches a glob pattern as follows:

```
>>>> filename.match('*.jpg')
True
>>>
```

Here is a recursive glob over a directory structure:

```
>>> topdir = Path('/Users/beazley/Pictures')
>>> for filename in topdir.rglob('*.JPG'):
...  print(filename)
...
/Users/beazley/Pictures/Foo/IMG_0001.JPG
/Users/beazley/Pictures/Foo/IMG_0002.JPG
/Users/beazley/Pictures/Foo/IMG_0003.JPG
...
/Users/beazley/Pictures/Bar/IMG_1024.JPG
/Users/beazley/Pictures/Bar/IMG_1025.JPG
...
```

Putting this all together, here is an example of the thumbnail script using pathlib.

```
from pathlib import Path
import os
import subprocess

def make_thumbnails(topdir, pat):
  topdir = Path(topdir)
  for filename in topdir.rglob(pat):
     newdirname = filename.parent / 'thumbnails'
     if not newdirname.exists():
        print('Making directory %s' % newdirname)
        newdirname.mkdir()
     newfilename = newdirname / (filename.stem + '.png')
     out = subprocess.check_output(['convert', '-resize',
              '100x100', str(filename), str(newfilename)])

if __name__ == '__main__':
  make_thumbnails('/Users/beazley/PhotoLibrary', '*.JPG')
```

On the whole, I think you'll find the script to be bit cleaner than the original version using os.path. If you've used the third-party path module, there are a few potential gotchas stemming from the fact that Path objects in pathlib do not derive from strings. In particular, if you ever need to pass paths to other functions such as the subprocesss.check_output() function in the example, you'll need to explicitly convert the path to a string using str() first.

Final Words

I'll admit that I've always been a bit bothered by the clunky nature of the os.path functionality. Although this annoyance has been minor (in the grand scheme of things, there always seemed to be bigger problems to deal with), pathlib is a welcome addition. Now that I know it's there, I think I'll start to use it. If you're using Python 3, it's definitely worth a look. A backport to earlier versions of Python can be found at https://pypi.python.org/pypi/pathlib/.

References

[1] pathlib documentation: https://docs.python.org/3/library/pathlib.html.

[2] PEP 355: http://legacy.python.org/dev/peps/pep-0355/.

[3] path.py package: https://pypi.python.org/pypi/path.py.

[4] PEP 428: http://legacy.python.org/dev/peps/pep-0428/.