os.path — Platform-independent Manipulation of Filenames

Purpose: Parse, build, test, and otherwise work on filenames and paths.

Writing code to work with files on multiple platforms is easy using the functions included in the `os.path` module. Even programs not intended to be ported between platforms should use `os.path` for reliable filename parsing.

Parsing Paths

The first set of functions in `os.path` can be used to parse strings representing filenames into their component parts. It is important to realize that these functions do not depend on the paths actually existing; they operate solely on the strings.

Path parsing depends on a few variable defined in os:

- `os.sep` - The separator between portions of the path (e.g., “/” or “\”).
- `os.extsep` - The separator between a filename and the file “extension” (e.g., “`.`”).
- `os.pardir` - The path component that means traverse the directory tree up one level (e.g., “`..`”).
- `os.curdir` - The path component that refers to the current directory (e.g., “`.`”).

The `split()` function breaks the path into two separate parts and returns a `tuple` with the results. The second element of the `tuple` is the last component of the path, and the first element is everything that comes before it.

```python
# ospath_split.py

import os.path

PATHS = [
    '/one/two/three',
    '/one/two/three/',
    '/',
    '.',
    '',
]

for path in PATHS:
    print('{!r:>17} : {}'.format(path, os.path.split(path)))
```

When the input argument ends in `os.sep`, the last element of the path is an empty string.

```
$ python3 ospath_split.py

 '/one/two/three' : ('/one/two', 'three')
'/one/two/three/' : ('/one/two/three', '')
              '/' : ('/', '')
              '.' : ('', '.')
               '' : ('', '')
```

The `basename()` function returns a value equivalent to the second part of the `split()` value.

```python
# ospath_basename.py

import os.path

PATHS = [
    '/one/two/three',
    '/one/two/three/',
    '/',
    '.',
    '',
]
```

```
for path in PATHS:
    print('{!r:>17} : {!r}'.format(path, os.path.basename(path)))
```

The full path is stripped down to the last element, whether that refers to a file or directory. If the path ends in the directory separator (`os.sep`), the base portion is considered to be empty.

```
$ python3 ospath_basename.py

 '/one/two/three' : 'three'
'/one/two/three/' : ''
              '/' : ''
              '.' : '.'
               '' : ''
```

The `dirname()` function returns the first part of the split path:

```
# ospath_dirname.py

import os.path

PATHS = [
    '/one/two/three',
    '/one/two/three/',
    '/',
    '.',
    '',
]

for path in PATHS:
    print('{!r:>17} : {!r}'.format(path, os.path.dirname(path)))
```

Combining the results of `basename()` with `dirname()` gives the original path.

```
$ python3 ospath_dirname.py

 '/one/two/three' : '/one/two'
'/one/two/three/' : '/one/two/three'
              '/' : '/'
              '.' : ''
               '' : ''
```

`splitext()` works like `split()`, but divides the path on the extension separator, rather than the directory separator.

```
# ospath_splitext.py

import os.path

PATHS = [
    'filename.txt',
    'filename',
    '/path/to/filename.txt',
    '/',
    '',
    'my-archive.tar.gz',
    'no-extension.',
]

for path in PATHS:
    print('{!r:>21} : {!r}'.format(path, os.path.splitext(path)))
```

Only the last occurrence of `os.extsep` is used when looking for the extension, so if a filename has multiple extensions the results of splitting it leaves part of the extension on the prefix.

```
$ python3 ospath_splitext.py

       'filename.txt' : ('filename', '.txt')
           'filename' : ('filename', '')
'/path/to/filename.txt' : ('/path/to/filename', '.txt')
```

```
                   '/' : ('/', '')
                    '' : ('', '')
  'my-archive.tar.gz' : ('my-archive.tar', '.gz')
      'no-extension.' : ('no-extension', '.')
```

commonprefix() takes a list of paths as an argument and returns a single string that represents a common prefix present in all of the paths. The value may represent a path that does not actually exist, and the path separator is not included in the consideration, so the prefix might not stop on a separator boundary.

```python
# ospath_commonprefix.py

import os.path

paths = ['/one/two/three/four',
         '/one/two/threefold',
         '/one/two/three/',
         ]
for path in paths:
    print('PATH:', path)

print()
print('PREFIX:', os.path.commonprefix(paths))
```

In this example, the common prefix string is /one/two/three, even though one path does not include a directory named three.

```
$ python3 ospath_commonprefix.py

PATH: /one/two/three/four
PATH: /one/two/threefold
PATH: /one/two/three/

PREFIX: /one/two/three
```

commonpath() does honor path separators, and returns a prefix that does not include partial path values.

```python
# ospath_commonpath.py

import os.path

paths = ['/one/two/three/four',
         '/one/two/threefold',
         '/one/two/three/',
         ]
for path in paths:
    print('PATH:', path)

print()
print('PREFIX:', os.path.commonpath(paths))
```

Because "threefold" does not have a path separator after "three" the common prefix is /one/two.

```
$ python3 ospath_commonpath.py

PATH: /one/two/three/four
PATH: /one/two/threefold
PATH: /one/two/three/

PREFIX: /one/two
```

Building Paths

Besides taking existing paths apart, it is frequently necessary to build paths from other strings. To combine several path components into a single value, use join():

```python
# ospath_join.py

import os.path
```

```
PATHS = [
    ('one', 'two', 'three'),
    ('/', 'one', 'two', 'three'),
    ('/one', '/two', '/three'),
]

for parts in PATHS:
    print('{} : {!r}'.format(parts, os.path.join(*parts)))
```

If any argument to join begins with `os.sep`, all of the previous arguments are discarded and the new one becomes the beginning of the return value.

```
$ python3 ospath_join.py

('one', 'two', 'three') : 'one/two/three'
('/', 'one', 'two', 'three') : '/one/two/three'
('/one', '/two', '/three') : '/three'
```

It is also possible to work with paths that include “variable” components that can be expanded automatically. For example, `expanduser()` converts the tilde (~) character to the name of a user’s home directory.

```
# ospath_expanduser.py

import os.path

for user in ['', 'dhellmann', 'nosuchuser']:
    lookup = '~' + user
    print('{!r:>15} : {!r}'.format(
        lookup, os.path.expanduser(lookup)))
```

If the user’s home directory cannot be found, the string is returned unchanged, as with `~nosuchuser` in this example.

```
$ python3 ospath_expanduser.py

            '~' : '/Users/dhellmann'
   '~dhellmann' : '/Users/dhellmann'
  '~nosuchuser' : '~nosuchuser'
```

`expandvars()` is more general, and expands any shell environment variables present in the path.

```
# ospath_expandvars.py

import os.path
import os

os.environ['MYVAR'] = 'VALUE'

print(os.path.expandvars('/path/to/$MYVAR'))
```

No validation is performed to ensure that the variable value results in the name of a file that already exists.

```
$ python3 ospath_expandvars.py

/path/to/VALUE
```

Normalizing Paths

Paths assembled from separate strings using `join()` or with embedded variables might end up with extra separators or relative path components. Use `normpath()` to clean them up:

```
# ospath_normpath.py

import os.path

PATHS = [
    'one//two//three',
    'one/./two/./three',
    'one/../alt/two/three',
]

for path in PATHS:
    print('{!r:>22} : {!r}'.format(path, os.path.normpath(path)))
```

Path segments made up of `os.curdir` and `os.pardir` are evaluated and collapsed.

```
$ python3 ospath_normpath.py

     'one//two//three' : 'one/two/three'
   'one/./two/./three' : 'one/two/three'
'one/../alt/two/three' : 'alt/two/three'
```

To convert a relative path to an absolute filename, use `abspath()`.

```
# ospath_abspath.py

import os
import os.path

os.chdir('/usr')

PATHS = [
    '.',
    '..',
    './one/two/three',
    '../one/two/three',
]

for path in PATHS:
    print('{!r:>21} : {!r}'.format(path, os.path.abspath(path)))
```

The result is a complete path, starting at the top of the file system tree.

```
$ python3 ospath_abspath.py

                  '.' : '/usr'
                 '..' : '/'
    './one/two/three' : '/usr/one/two/three'
   '../one/two/three' : '/one/two/three'
```

File Times

Besides working with paths, `os.path` includes functions for retrieving file properties, similar to the ones returned by `os.stat()`:

```
# ospath_properties.py

import os.path
import time

print('File         :', __file__)
print('Access time  :', time.ctime(os.path.getatime(__file__)))
print('Modified time:', time.ctime(os.path.getmtime(__file__)))
print('Change time  :', time.ctime(os.path.getctime(__file__)))
print('Size         :', os.path.getsize(__file__))
```

`os.path.getatime()` returns the access time, `os.path.getmtime()` returns the modification time, and `os.path.getctime()` returns the creation time. `os.path.getsize()` returns the amount of data in the file, represented in bytes.

```
$ python3 ospath_properties.py

File         : ospath_properties.py
Access time  : Sun Mar 18 16:21:22 2018
Modified time: Fri Nov 11 17:18:44 2016
```

```
Change time  : Fri Nov 11 17:18:44 2016
Size         : 481
```

Testing Files

When a program encounters a path name, it often needs to know whether the path refers to a file, directory, or symlink and whether it exists. os.path includes functions for testing all of these conditions.

```
# ospath_tests.py

import os.path

FILENAMES = [
    __file__,
    os.path.dirname(__file__),
    '/',
    './broken_link',
]

for file in FILENAMES:
    print('File        : {!r}'.format(file))
    print('Absolute    :', os.path.isabs(file))
    print('Is File?    :', os.path.isfile(file))
    print('Is Dir?     :', os.path.isdir(file))
    print('Is Link?    :', os.path.islink(file))
    print('Mountpoint? :', os.path.ismount(file))
    print('Exists?     :', os.path.exists(file))
    print('Link Exists?:', os.path.lexists(file))
    print()
```

All of the test functions return boolean values.

```
$ ln -s /does/not/exist broken_link
$ python3 ospath_tests.py

File        : 'ospath_tests.py'
Absolute    : False
Is File?    : True
Is Dir?     : False
Is Link?    : False
Mountpoint? : False
Exists?     : True
Link Exists?: True

File        : ''
Absolute    : False
Is File?    : False
Is Dir?     : False
Is Link?    : False
Mountpoint? : False
Exists?     : False
Link Exists?: False

File        : '/'
Absolute    : True
Is File?    : False
Is Dir?     : True
Is Link?    : False
Mountpoint? : True
Exists?     : True
Link Exists?: True

File        : './broken_link'
Absolute    : False
Is File?    : False
Is Dir?     : False
Is Link?    : True
Mountpoint? : False
Exists?     : False
Link Exists?: True
```

See also

- Standard library documentation for os.path
- Python 2 to 3 porting notes for os.path
- `pathlib` – Paths as objects.
- `os` – The os module is a parent of `os.path`.
- `time` – The time module includes functions to convert between the representation used by the time property functions in `os.path` and easy-to-read strings.

The File System pathlib — Filesystem Paths as Objects

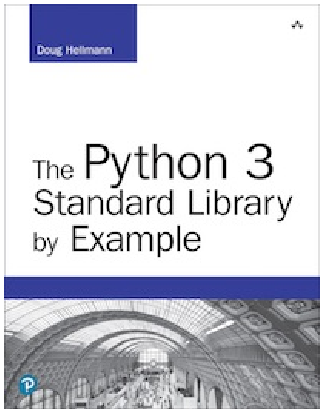
Quick Links

Parsing Paths
Building Paths
Normalizing Paths
File Times
Testing Files

This page was last updated 2018-03-18.

Navigation

The File System
pathlib — Filesystem Paths as Objects



The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?