tempfile — Temporary File System Objects

Purpose: Create temporary file system objects.

Creating temporary files with unique names securely, so they cannot be guessed by someone wanting to break the application or steal the data, is challenging. The `tempfile` module provides several functions for creating temporary file system resources securely. `TemporaryFile()` opens and returns an unnamed file, `NamedTemporaryFile()` opens and returns a named file, `SpooledTemporaryFile` holds its content in memory before writing to disk, and `TemporaryDirectory` is a context manager that removes the directory when the context is closed.

Temporary Files

Applications that need temporary files to store data, without needing to share that file with other programs, should use the `TemporaryFile()` function to create the files. The function creates a file, and on platforms where it is possible, unlinks it immediately. This makes it impossible for another program to find or open the file, since there is no reference to it in the file system table. The file created by `TemporaryFile()` is removed automatically when it is closed, whether by calling `close()` or by using the context manager API and `with` statement.

```
# tempfile_TemporaryFile.py

import os
import tempfile

print('Building a filename with PID:')
filename = '/tmp/guess_my_name.{}.txt'.format(os.getpid())
with open(filename, 'w+b') as temp:
    print('temp:')
    print('  {!r}'.format(temp))
    print('temp.name:')
    print('  {!r}'.format(temp.name))

# Clean up the temporary file yourself.
os.remove(filename)

print()
print('TemporaryFile:')
with tempfile.TemporaryFile() as temp:
    print('temp:')
    print('  {!r}'.format(temp))
    print('temp.name:')
    print('  {!r}'.format(temp.name))

# Automatically cleans up the file.
```

This example illustrates the difference in creating a temporary file using a common pattern for making up a name, versus using the `TemporaryFile()` function. The file returned by `TemporaryFile()` has no name.

```
$ python3 tempfile_TemporaryFile.py

Building a filename with PID:
temp:
  <_io.BufferedRandom name='/tmp/guess_my_name.12151.txt'>
temp.name:
  '/tmp/guess_my_name.12151.txt'

TemporaryFile:
temp:
  <_io.BufferedRandom name=4>
temp.name:
  4
```

By default, the file handle is created with mode `'w+b'` so it behaves consistently on all platforms and the caller can write to it and read from it.

```
# tempfile_TemporaryFile_binary.py

import os
import tempfile

with tempfile.TemporaryFile() as temp:
    temp.write(b'Some data')

    temp.seek(0)
    print(temp.read())
```

After writing, the file handle must be “rewound” using `seek()` in order to read the data back from it.

```
$ python3 tempfile_TemporaryFile_binary.py

b'Some data'
```

To open the file in text mode, set mode to `'w+t'` when the file is created.

```
# tempfile_TemporaryFile_text.py

import tempfile

with tempfile.TemporaryFile(mode='w+t') as f:
    f.writelines(['first\n', 'second\n'])

    f.seek(0)
    for line in f:
        print(line.rstrip())
```

The file handle treats the data as text.

```
$ python3 tempfile_TemporaryFile_text.py

first
second
```

Named Files

There are situations where having a named temporary file is important. For applications spanning multiple processes, or even hosts, naming the file is the simplest way to pass it between parts of the application. The `NamedTemporaryFile()` function creates a file without unlinking it, so it retains its name (accessed with the name attribute).

```
# tempfile_NamedTemporaryFile.py

import os
import pathlib
import tempfile

with tempfile.NamedTemporaryFile() as temp:
    print('temp:')
    print('  {!r}'.format(temp))
    print('temp.name:')
    print('  {!r}'.format(temp.name))

    f = pathlib.Path(temp.name)

print('Exists after close:', f.exists())
```

The file is removed after the handle is closed.

```
$ python3 tempfile_NamedTemporaryFile.py

temp:
  <tempfile._TemporaryFileWrapper object at 0x1011b2d30>
temp.name:
  '/var/folders/5q/8gk0wq888xlggz008k8dr7180000hg/T/tmps4qh5zde'
Exists after close: False
```

Spooled Files

For temporary files containing relatively small amounts of data, it is likely to be more efficient to use a `SpooledTemporaryFile` because it holds the file contents in memory using a `io.BytesIO` or `io.StringIO` buffer until they reach a threshold size. When the amount of data passes the threshold, it is “rolled over” and written to disk, and then the buffer is replaced with a normal `TemporaryFile()`.

```python
# tempfile_SpooledTemporaryFile.py

import tempfile

with tempfile.SpooledTemporaryFile(max_size=100,
                                   mode='w+t',
                                   encoding='utf-8') as temp:
    print('temp: {!r}'.format(temp))

    for i in range(3):
        temp.write('This line is repeated over and over.\n')
        print(temp._rolled, temp._file)
```

This example uses private attributes of the `SpooledTemporaryFile` to determine when the rollover to disk has happened. It is not normally necessary to check this status except when tuning the buffer size.

```
$ python3 tempfile_SpooledTemporaryFile.py

temp: <tempfile.SpooledTemporaryFile object at 0x1007b2c88>
False <_io.StringIO object at 0x1007a3d38>
False <_io.StringIO object at 0x1007a3d38>
True <_io.TextIOWrapper name=4 mode='w+t' encoding='utf-8'>
```

To explicitly cause the buffer to be written to disk, call the `rollover()` or `fileno()` methods.

```python
# tempfile_SpooledTemporaryFile_explicit.py

import tempfile

with tempfile.SpooledTemporaryFile(max_size=1000,
                                   mode='w+t',
                                   encoding='utf-8') as temp:
    print('temp: {!r}'.format(temp))

    for i in range(3):
        temp.write('This line is repeated over and over.\n')
        print(temp._rolled, temp._file)
    print('rolling over')
    temp.rollover()
    print(temp._rolled, temp._file)
```

In this example, because the buffer size is so much larger than the amount of data, no file would be created on disk except that `rollover()` was called.

```
$ python3 tempfile_SpooledTemporaryFile_explicit.py

temp: <tempfile.SpooledTemporaryFile object at 0x1007b2c88>
False <_io.StringIO object at 0x1007a3d38>
False <_io.StringIO object at 0x1007a3d38>
False <_io.StringIO object at 0x1007a3d38>
rolling over
True <_io.TextIOWrapper name=4 mode='w+t' encoding='utf-8'>
```

Temporary Directories

When several temporary files are needed, it may be more convenient to create a single temporary directory with `TemporaryDirectory` and open all of the files in that directory.

```
# tempfile_TemporaryDirectory.py

import pathlib
import tempfile

with tempfile.TemporaryDirectory() as directory_name:
    the_dir = pathlib.Path(directory_name)
    print(the_dir)
    a_file = the_dir / 'a_file.txt'
    a_file.write_text('This file is deleted.')

print('Directory exists after?', the_dir.exists())
print('Contents after:', list(the_dir.glob('*')))
```

The context manager produces the name of the directory, which can then be used within the context block to build other file names.

```
$ python3 tempfile_TemporaryDirectory.py

/var/folders/5q/8gk0wq888xlggz008k8dr7180000hg/T/tmp_urhiioj
Directory exists after? False
Contents after: []
```

Predicting Names

While less secure than strictly anonymous temporary files, including a predictable portion in the name makes it possible to find the file and examine it for debugging purposes. All of the functions described so far take three arguments to control the filenames to some degree. Names are generated using the formula:

```
dir + prefix + random + suffix
```

All of the values except `random` can be passed as arguments to the functions for creating temporary files or directories.

```
# tempfile_NamedTemporaryFile_args.py

import tempfile

with tempfile.NamedTemporaryFile(suffix='_suffix',
                                 prefix='prefix_',
                                 dir='/tmp') as temp:
    print('temp:')
    print('  ', temp)
    print('temp.name:')
    print('  ', temp.name)
```

The `prefix` and `suffix` arguments are combined with a random string of characters to build the filename, and the `dir` argument is taken as-is and used as the location of the new file.

```
$ python3 tempfile_NamedTemporaryFile_args.py

temp:
   <tempfile._TemporaryFileWrapper object at 0x1018b2d68>
temp.name:
   /tmp/prefix_q6wd5czl_suffix
```

Temporary File Location

If an explicit destination is not given using the `dir` argument, the path used for the temporary files will vary based on the current platform and settings. The `tempfile` module includes two functions for querying the settings being used at runtime.

```
# tempfile_settings.py

import tempfile

print('gettempdir():', tempfile.gettempdir())
print('gettempprefix():', tempfile.gettempprefix())
print('gettempprefix():', tempfile.gettempprefix())
```

`gettempdir()` returns the default directory that will hold all of the temporary files and `gettempprefix()` returns the string prefix for new file and directory names.

```
$ python3 tempfile_settings.py

gettempdir(): /var/folders/5q/8gk0wq888xlggz008k8dr7180000hg/T
gettempprefix(): tmp
```

The value returned by `gettempdir()` is set based on a straightforward algorithm of looking through a list of locations for the first place the current process can create a file. The search list is:

1. The environment variable `TMPDIR`
2. The environment variable `TEMP`
3. The environment variable `TMP`
4. A fallback, based on the platform. (Windows uses the first available of `C:\temp`, `C:\tmp`, `\temp`, or `\tmp`. Other platforms use `/tmp`, `/var/tmp`, or `/usr/tmp`.)
5. If no other directory can be found, the current working directory is used.

```
# tempfile_tempdir.py

import tempfile

tempfile.tempdir = '/I/changed/this/path'
print('gettempdir():', tempfile.gettempdir())
```

Programs that need to use a global location for all temporary files without using any of these environment variables should set `tempfile.tempdir` directly by assigning a value to the variable.

```
$ python3 tempfile_tempdir.py

gettempdir(): /I/changed/this/path
```

See also

- Standard library documentation for tempfile
- random – Psuedorandom number generators, used to introduce random values into temporary file names

linecache — Read Text Files Efficiently

shutil — High-level File Operations

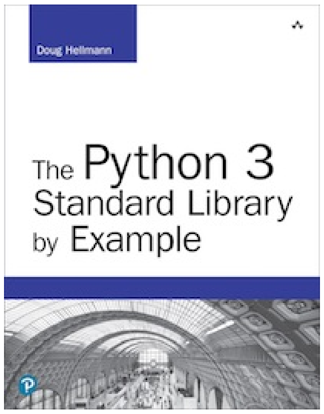
Quick Links

Temporary Files
Named Files
Spooled Files
Temporary Directories
Predicting Names
Temporary File Location

This page was last updated 2016-12-28.

Navigation

linecache — Read Text Files Efficiently
shutil — High-level File Operations



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?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example