11.6. `tempfile` — Generate temporary files and directories¶

This module generates temporary files and directories. It works on all supported platforms.

In version 2.3 of Python, this module was overhauled for enhanced security. It now provides three new functions, `NamedTemporaryFile()`, `mkstemp()`, and `mkdtemp()`, which should eliminate all remaining need to use the insecure `mktemp()` function. Temporary file names created by this module no longer contain the process ID; instead a string of six random characters is used.

Also, all the user-callable functions now take additional arguments which allow direct control over the location and name of temporary files. It is no longer necessary to use the global *tempdir* and *template* variables. To maintain backward compatibility, the argument order is somewhat odd; it is recommended to use keyword arguments for clarity.

The module defines the following user-callable functions:

`tempfile.TemporaryFile`([*mode='w+b'*[, *bufsize=-1*[, *suffix=''*[, *prefix='tmp'*[, *dir=None*]]]]])¶

Return a file-like object that can be used as a temporary storage area. The file is created using `mkstemp()`. It will be destroyed as soon as it is closed (including an implicit close when the object is garbage collected). Under Unix, the directory entry for the file is removed immediately after the file is created. Other platforms do not support this; your code should not rely on a temporary file created using this function having or not having a visible name in the file system.

The *mode* parameter defaults to `'w+b'` so that the file created can be read and written without being closed. Binary mode is used so that it behaves consistently on all platforms without regard for the data that is stored. *bufsize* defaults to `-1`, meaning that the operating system default is used.

The *dir*, *prefix* and *suffix* parameters are passed to `mkstemp()`.

The returned object is a true file object on POSIX platforms. On other platforms, it is a file-like object whose `file` attribute is the underlying true file object. This file-like object can be used in a `with` statement, just like a normal file.

`tempfile.NamedTemporaryFile`([*mode='w+b'*[, *bufsize=-1*[, *suffix=''*[, *prefix='tmp'*[, *dir=None*[, *delete=True*]]]]]])¶

This function operates exactly as `TemporaryFile()` does, except that the file is guaranteed to have a visible name in the file system (on Unix, the directory entry is not unlinked). That name can be retrieved from the `name` member of the file object. Whether the name can be used to open the file a second time, while the named temporary file is still open, varies across platforms (it can be so used on Unix; it cannot on Windows NT or later). If *delete* is true (the default), the file is deleted as soon as it is closed.

The returned object is always a file-like object whose `file` attribute is the underlying true file object. This file-like object can be used in a `with` statement, just like a normal file.

New in version 2.3.

New in version 2.6: The *delete* parameter.

`tempfile.SpooledTemporaryFile`([*max_size=0*[, *mode='w+b'*[, *bufsize=-1*[, *suffix=''*[, *prefix='tmp'*[, *dir=None*]]]]]])¶

This function operates exactly as `TemporaryFile()` does, except that data is spooled in memory until the file size exceeds *max_size*, or until the file's `fileno()` method is called, at which point the contents are written to disk and operation proceeds as with `TemporaryFile()`.

The resulting file has one additional method, `rollover()`, which causes the file to roll over to an on-disk file regardless of its size.

The returned object is a file-like object whose `_file` attribute is either a `StringIO` object or a true file object, depending on whether `rollover()` has been called. This file-like object can be used in a `with` statement, just like a normal file.

New in version 2.6.

`tempfile.mkstemp`([*suffix=''*[, *prefix='tmp'*[, *dir=None*[, *text=False*]]]])¶

Creates a temporary file in the most secure manner possible. There are no race conditions in the file's creation, assuming that the platform properly implements the `os.O_EXCL` flag for `os.open()`. The file is readable and writable only by the creating user ID. If the platform uses permission bits to indicate whether a file is executable, the file is executable by no one. The file descriptor is not inherited by child processes.

Unlike `TemporaryFile()`, the user of `mkstemp()` is responsible for deleting the temporary file when done with it.

If *suffix* is specified, the file name will end with that suffix, otherwise there will be no suffix. `mkstemp()` does not put a dot between the file name and the suffix; if you need one, put it at the beginning of *suffix*.

If *prefix* is specified, the file name will begin with that prefix; otherwise, a default prefix is used.

If *dir* is specified, the file will be created in that directory; otherwise, a default directory is used. The default directory is chosen from a platform-dependent list, but the user of the application can control the directory location by setting the *TMPDIR*, *TEMP* or *TMP* environment variables. There is thus no guarantee that the generated filename will have any nice properties, such as not requiring quoting when passed to external commands via `os.popen()`.

If *text* is specified, it indicates whether to open the file in binary mode (the default) or text mode. On some platforms, this makes no difference.

`mkstemp()` returns a tuple containing an OS-level handle to an open file (as would be returned by `os.open()`) and the absolute pathname of that file, in that order.

New in version 2.3.

`tempfile.mkdtemp`([*suffix=''*[, *prefix='tmp'*[, *dir=None*]]])¶

Creates a temporary directory in the most secure manner possible. There are no race conditions in the directory's creation. The directory is readable, writable, and searchable only by the creating user ID.

The user of `mkdtemp()` is responsible for deleting the temporary directory and its contents when done with it.

The *prefix*, *suffix*, and *dir* arguments are the same as for `mkstemp()`.

`mkdtemp()` returns the absolute pathname of the new directory.

New in version 2.3.

`tempfile.mktemp`([*suffix=''*[, *prefix='tmp'*[, *dir=None*]]])¶

Deprecated since version 2.3: Use `mkstemp()` instead.

Return an absolute pathname of a file that did not exist at the time the call is made. The *prefix*, *suffix*, and *dir* arguments are the same as for `mkstemp()`.

Warning

Use of this function may introduce a security hole in your program. By the time you get around to doing anything with the file name it returns, someone else may have beaten you to the punch. `mktemp()` usage can be replaced easily with `NamedTemporaryFile()`, passing it the `delete=False` parameter:

```
>>> f = NamedTemporaryFile(delete=False)
>>> f
<open file '<fdopen>', mode 'w+b' at 0x384698>
>>> f.name
'/var/folders/5q/5qTPn6xq2RaWqk+1Ytw3-U+++TI/-Tmp-/tmpG7V1Y0'
>>> f.write("Hello World!\n")
>>> f.close()
>>> os.unlink(f.name)
>>> os.path.exists(f.name)
False
```

The module uses two global variables that tell it how to construct a temporary name. They are initialized at the first call to any of the functions above. The caller may change them, but this is discouraged; use the appropriate function arguments, instead.

`tempfile.tempdir`¶

When set to a value other than `None`, this variable defines the default value for the *dir* argument to all the functions defined in this module.

If `tempdir` is unset or `None` at any call to any of the above functions, Python searches a standard list of directories and sets *tempdir* to the first one which the calling user can create files in. The list is:

1. The directory named by the **TMPDIR** environment variable.
2. The directory named by the **TEMP** environment variable.
3. The directory named by the **TMP** environment variable.
4. A platform-specific location:
   - On RiscOS, the directory named by the **Wimp$ScrapDir** environment variable.
   - On Windows, the directories `C:\TEMP`, `C:\TMP`, `\TEMP`, and `\TMP`, in that order.
   - On all other platforms, the directories `/tmp`, `/var/tmp`, and `/usr/tmp`, in that order.
5. As a last resort, the current working directory.

`tempfile.gettempdir`()¶

Return the directory currently selected to create temporary files in. If `tempdir` is not `None`, this simply returns its contents; otherwise, the search described above is performed, and the result returned.

New in version 2.3.

`tempfile.template`¶

Deprecated since version 2.0: Use `gettempprefix()` instead.

When set to a value other than `None`, this variable defines the prefix of the final component of the filenames returned by `mktemp()`. A string of six random letters and digits is appended to the prefix to make the filename unique. The default prefix is `tmp`.

Older versions of this module used to require that `template` be set to `None` after a call to `os.fork()`; this has not been necessary since version 1.5.2.

`tempfile.gettempprefix`()¶

Return the filename prefix used to create temporary files. This does not contain the directory component. Using this function is preferred over reading the *template* variable directly.

New in version 1.5.2.

Previous topic

11.5. `filecmp` — File and Directory Comparisons

Next topic

11.7. `glob` — Unix style pathname pattern expansion

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 11. File and Directory Access »