

## Previous topic

`pathlib` — Object-oriented filesystem paths

## Next topic

`fileinput` — Iterate over lines from multiple input streams

## This Page

Report a Bug  
Show Source

# `os.path` — Common pathname manipulations

**Source code:** [Lib/posixpath.py](#) (for POSIX) and [Lib/ntpath.py](#) (for Windows NT).

This module implements some useful functions on pathnames. To read or write files see `open()`, and for accessing the filesystem see the `os` module. The path parameters can be passed as either strings, or bytes. Applications are encouraged to represent file names as (Unicode) character strings. Unfortunately, some file names may not be representable as strings on Unix, so applications that need to support arbitrary file names on Unix should use bytes objects to represent path names. Vice versa, using bytes objects cannot represent all file names on Windows (in the standard `mbcs` encoding), hence Windows applications should use string objects to access all files.

« Unlike a unix shell, Python does not do any *automatic* path expansions. Functions such as `expanduser()` and `expandvars()` can be invoked explicitly when an application desires shell-like path expansion. (See also the `glob` module.)

**See also:** The `pathlib` module offers high-level path objects.

**Note:** All of these functions accept either only bytes or only string objects as their parameters. The result is an object of the same type, if a path or file name is returned.

**Note:** Since different operating systems have different path name conventions, there are several versions of this module in the standard library. The `os.path` module is always the path module suitable for the operating system Python is running on, and therefore usable for local paths. However, you can also import and use the individual modules if you want to manipulate a path that is *always* in one of the different formats. They all have the same interface:

- `posixpath` for UNIX-style paths
- `ntpath` for Windows paths

*Changed in version 3.8:* `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()`, and `ismount()` now return `False` instead of raising an exception for paths that contain characters or bytes unrepresentable at the OS level.

`os.path.abspath(path)`

Return a normalized absolutized version of the pathname *path*. On most platforms, this is equivalent to calling the function `normpath()` as follows: `normpath(join(os.getcwd(), path))`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.basename(path)`

Return the base name of pathname *path*. This is the second element of the pair returned by passing *path* to the function `split()`. Note that the result of this function is different from the Unix **basename** program; where **basename** for `'/foo/bar/'` returns `'bar'`, the `basename()` function returns an empty string (`''`).

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.commonpath(paths)`

Return the longest common sub-path of each pathname in the sequence *paths*. Raise `ValueError` if *paths* contain both absolute and relative pathnames, the *paths* are on the different drives or if *paths* is empty. Unlike `commonprefix()`, this returns a valid path.

[Availability](#): Unix, Windows.

*New in version 3.5.*

*Changed in version 3.6:* Accepts a sequence of [path-like objects](#).

`os.path.commonprefix(list)`

Return the longest path prefix (taken character-by-character) that is a prefix of all paths in *list*. If *list* is empty, return the empty string (`''`).

**Note:** This function may return invalid paths because it works a character at a time. To obtain a valid path, see `commonpath()`.

```
>>> os.path.commonprefix(['/usr/lib', '/usr/local/lib'])
'/usr/l'

>>> os.path.commonpath(['/usr/lib', '/usr/local/lib'])
'/usr'
```

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.dirname(path)`

Return the directory name of pathname *path*. This is the first element of the pair returned by passing *path* to the function `split()`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.exists(path)`

Return `True` if *path* refers to an existing path or an open file descriptor. Returns `False` for broken symbolic links. On some platforms, this function may return `False` if permission is not granted to execute `os.stat()` on the requested file, even if the *path* physically exists.

*Changed in version 3.3:* *path* can now be an integer: `True` is returned if it is an open file descriptor, `False` otherwise.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.lexists(path)`

Return `True` if *path* refers to an existing path. Returns `True` for broken symbolic links. Equivalent to `exists()` on platforms lacking `os.lstat()`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.expanduser(path)`

On Unix and Windows, return the argument with an initial component of `~` or `~user` replaced by that *user*'s home directory.

On Unix, an initial `~` is replaced by the environment variable `HOME` if it is set; otherwise the current user's home directory is looked up in the password directory through the built-in module `pwd`. An initial `~user` is looked up directly in the password directory.

On Windows, `USERPROFILE` will be used if set, otherwise a combination of `HOME` and `HOMEDRIVE` will be used. An initial `~user` is handled by stripping the last directory component from the created user path derived above.

If the expansion fails or if the path does not begin with a tilde, the path is returned unchanged.

*Changed in version 3.6:* Accepts a [path-like object](#).

*Changed in version 3.8:* No longer uses `HOME` on Windows.

`os.path.expandvars(path)`

Return the argument with environment variables expanded. Substrings of the form `$name` or `${name}` are replaced by the value of environment variable *name*. Malformed variable names and references to non-existing variables are left unchanged.

On Windows, `%name%` expansions are supported in addition to `$name` and `${name}`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.getatime(path)`

Return the time of last access of *path*. The return value is a floating point number giving the number of seconds since the epoch (see the [time](#) module). Raise `OSError` if the file does not exist or is inaccessible.

`os.path.getmtime(path)`

Return the time of last modification of *path*. The return value is a floating point number giving the number of seconds since the epoch (see the [time](#) module). Raise `OSError` if the file does not exist or is inaccessible.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.getctime(path)`

Return the system's ctime which, on some systems (like Unix) is the time of the last metadata change, and, on others (like Windows), is the creation time for *path*. The return value is a number giving the number of seconds since the epoch (see the [time](#) module). Raise `OSError` if the file does not exist or is inaccessible.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.getsize(path)`

Return the size, in bytes, of *path*. Raise `OSError` if the file does not exist or is inaccessible.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.isabs(path)`

Return `True` if *path* is an absolute pathname. On Unix, that means it begins with a slash, on Windows that it begins with a (back)slash after chopping off a potential drive letter.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.isfile(path)`

Return `True` if *path* is an `existing` regular file. This follows symbolic links, so both `islink()` and `isfile()` can be true for the same path.

*Changed in version 3.6:* Accepts a `path-like object`.

`os.path.isdir(path)`

Return `True` if *path* is an `existing` directory. This follows symbolic links, so both `islink()` and `isdir()` can be true for the same path.

*Changed in version 3.6:* Accepts a `path-like object`.

`os.path.islink(path)`

Return `True` if *path* refers to an `existing` directory entry that is a symbolic link. Always `False` if symbolic links are not supported by the Python runtime.

*Changed in version 3.6:* Accepts a `path-like object`.

`os.path.ismount(path)`

Return `True` if pathname *path* is a *mount point*: a point in a file system where a different file system has been mounted. On POSIX, the function checks whether *path*'s parent, *path/..*, is on a different device than *path*, or whether *path/..* and *path* point to the same i-node on the same device — this should detect mount points for all Unix and POSIX variants. It is not able to reliably detect bind mounts on the same filesystem. On Windows, a drive letter root and a share UNC are always mount points, and for any other path `GetVolumePathName` is called to see if it is different from the input path.

*New in version 3.4:* Support for detecting non-root mount points on Windows.

*Changed in version 3.6:* Accepts a `path-like object`.

`os.path.join(path, *paths)`

Join one or more path components intelligently. The return value is the concatenation of *path* and any members of *\*paths* with exactly one directory separator (`os.sep`) following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty. If a component is an absolute path, all previous components are thrown away and joining continues from the absolute path component.

On Windows, the drive letter is not reset when an absolute path component (e.g., `r'\foo'`) is encountered. If a component contains a drive letter, all previous components are thrown away and the drive letter is reset.

Note that since there is a current directory for each drive, `os.path.join("c:", "foo")` represents a path relative to the current directory on drive C: (`c:foo`), not `c:\foo`.

*Changed in version 3.6:* Accepts a [path-like object](#) for *path* and *paths*.

`os.path.normcase(path)`

Normalize the case of a pathname. On Windows, convert all characters in the pathname to lowercase, and also convert forward slashes to backward slashes. On other operating systems, return the path unchanged.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.normpath(path)`

Normalize a pathname by collapsing redundant separators and up-level references so that `A//B`, `A/B/`, `A./B` and `A/foo/../B` all become `A/B`. This string manipulation may change the meaning of a path that contains symbolic links. On Windows, it converts forward slashes to backward slashes. To normalize case, use `normcase()`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.realpath(path)`

Return the canonical path of the specified filename, eliminating any symbolic links encountered in the path (if they are supported by the operating system).

**Note:** When symbolic link cycles occur, the returned path will be one member of the cycle, but no guarantee is made about which member that will be.

*Changed in version 3.6:* Accepts a [path-like object](#).

*Changed in version 3.8:* Symbolic links and junctions are now resolved on Windows.

`os.path.realpath(path, start=os.curdir)`

Return a relative filepath to *path* either from the current directory or from an optional *start* directory. This is a path computation: the filesystem is not accessed to confirm the existence or nature of *path* or *start*.

*start* defaults to `os.curdir`.

**Availability:** Unix, Windows.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.samefile(path1, path2)`

Return `True` if both pathname arguments refer to the same file or directory. This is determined by the device number and i-node number and raises an exception if an `os.stat()` call on either pathname fails.

**Availability:** Unix, Windows.

*Changed in version 3.2:* Added Windows support.

*Changed in version 3.4:* Windows now uses the same implementation as all other platforms.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.sameopenfile(fp1, fp2)`

Return `True` if the file descriptors `fp1` and `fp2` refer to the same file.

**Availability:** Unix, Windows.

*Changed in version 3.2:* Added Windows support.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.samestat(stat1, stat2)`

Return `True` if the stat tuples `stat1` and `stat2` refer to the same file. These structures may have been returned by `os.fstat()`, `os.lstat()`, or `os.stat()`. This function implements the underlying comparison used by `samefile()` and `sameopenfile()`.

**Availability:** Unix, Windows.

*Changed in version 3.4:* Added Windows support.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.split(path)`

Split the pathname `path` into a pair, `(head, tail)` where `tail` is the last pathname component and `head` is everything leading up to that. The `tail` part will never contain a slash; if `path` ends in a slash, `tail` will be empty. If there is no slash in `path`, `head` will be empty. If `path` is empty, both `head` and `tail` are empty. Trailing slashes are stripped from `head` unless it is the root (one or more slashes only). In all cases, `join(head, tail)` returns a path to the same location as `path` (but the strings may differ). Also see the functions `dirname()` and `basename()`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.splitdrive(path)`

Split the pathname *path* into a pair (drive, tail) where *drive* is either a mount point or the empty string. On systems which do not use drive specifications, *drive* will always be the empty string. In all cases, drive + tail will be the same as *path*.

On Windows, splits a pathname into drive/UNC sharepoint and relative path.

If the path contains a drive letter, drive will contain everything up to and including the colon. e.g. `splitdrive("c:/dir")` returns `("c:", "/dir")`

If the path contains a UNC path, drive will contain the host name and share, up to but not including the fourth separator. e.g. `splitdrive("//host/computer/dir")` returns `("//host/computer", "/dir")`

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.splitext(path)`

Split the pathname *path* into a pair (root, ext) such that root + ext == path, and ext is empty or begins with a period and contains at most one period. Leading periods on the basename are ignored; `splitext('.cshrc')` returns `('.cshrc', '')`.

*Changed in version 3.6:* Accepts a [path-like object](#).

`os.path.supports_unicode_filenames`

`True` if arbitrary Unicode strings can be used as file names (within limitations imposed by the file system).