10.1. `os.path` — Common pathname manipulations

This module implements some useful functions on pathnames. To read or write files see `open()`, and for accessing the filesystem see the `os` module.

> **Note:** On Windows, many of these functions do not properly support UNC pathnames. `splitunc()` and `ismount()` do handle them correctly.

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.)

> **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
> - `macpath` for old-style MacOS paths
> - **`os2emxpath`** for OS/2 EMX paths

`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))`.

New in version 1.5.2.

`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 (`''`).

`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 that this may return invalid paths because it works a character at a time.

`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()`.

`os.path.`**`exists`**(*path*)

Return `True` if *path* refers to an existing path. 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.

`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()`.

New in version 2.4.

`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, **`HOME`** and **`USERPROFILE`** will be used if set, otherwise a combination of **`HOMEPATH`** 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.

`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}`.

`os.path.`**`getatime`**(*path*)
Return the time of last access of *path*. The return value is a number giving the number of seconds since the epoch (see the `time` module). Raise `os.error` if the file does not exist or is inaccessible.

New in version 1.5.2.

Changed in version 2.3: If `os.stat_float_times()` returns `True`, the result is a floating point number.

`os.path.`**`getmtime`**(*path*)
Return the time of last modification of *path*. The return value is a number giving the number of seconds since the epoch (see the `time` module). Raise `os.error` if the file does not exist or is inaccessible.

New in version 1.5.2.

Changed in version 2.3: If `os.stat_float_times()` returns `True`, the result is a floating point number.

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 `os.error` if the file does not exist or is inaccessible.

New in version 2.3.

os.path.**getsize**(*path*)

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

New in version 1.5.2.

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.

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.

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.

os.path.**islink**(*path*)

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

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. 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.

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`.

os.path.**normcase**(*path*)

Normalize the case of a pathname. On Unix and Mac OS X, this returns the path unchanged; on case-insensitive filesystems, it converts the path to lowercase. On Windows, it also converts forward slashes to backward slashes.

`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()`**.

`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).

New in version 2.2.

`os.path.`**`relpath`**(*path*[, *start*])

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: Windows, Unix.

New in version 2.6.

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

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

Availability: Unix.

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

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

Availability: Unix.

`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.

`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()`**.

`os.path.`**`splitdrive`**(*path*)

Split the pathname *path* into a pair `(drive, tail)` where *drive* is either a drive specification 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*.

New in version 1.3.

`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 2.6: Earlier versions could produce an empty root when the only period was the first character.

`os.path.`**`splitunc`**(*path*)

Split the pathname *path* into a pair `(unc, rest)` so that *unc* is the UNC mount point (such as `r'\\host\mount'`), if present, and *rest* the rest of the path (such as `r'\path\file.ext'`). For paths containing drive letters, *unc* will always be the empty string.

Availability: Windows.

`os.path.`**`walk`**(*path*, *visit*, *arg*)

Calls the function *visit* with arguments `(arg, dirname, names)` for each directory in the directory tree rooted at *path* (including *path* itself, if it is a directory). The argument *dirname* specifies the visited directory, the argument *names* lists the files in the directory (gotten from `os.listdir(dirname)`). The *visit* function may modify *names* to influence the set of directories visited below *dirname*, e.g. to avoid visiting certain parts of the tree. (The object referred to by *names* must be modified in place, using **`del`** or slice assignment.)

> **Note:** Symbolic links to directories are not treated as subdirectories, and that **`walk()`** therefore will not visit them. To visit linked directories you must identify them with `os.path.islink(file)` and `os.path.isdir(file)`, and invoke **`walk()`** as necessary.

> **Note:** This function is deprecated and has been removed in Python 3 in favor of **`os.walk()`**.

`os.path.`**`supports_unicode_filenames`**

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

New in version 2.3.