**Biblioteca OS Python**

os.**name**

The name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'java'.

os.**ctermid**()

Return the filename corresponding to the controlling terminal of the process.

os.**environ**

A [mapping](https://docs.python.org/3/glossary.html#term-mapping) object where keys and values are strings that represent the process environment. For example, environ['HOME'] is the pathname of your home directory (on some platforms), and is equivalent to getenv("HOME") in C. This mapping is captured the first time the [os](https://docs.python.org/3/library/os.html#module-os) module is imported, typically during Python startup as part of processing site.py. Changes to the environment made after this time are not reflected in [os.environ](https://docs.python.org/3/library/os.html#os.environ), except for changes made by modifying [os.environ](https://docs.python.org/3/library/os.html#os.environ) directly. This mapping may be used to modify the environment as well as query the environment. [putenv()](https://docs.python.org/3/library/os.html#os.putenv) will be called automatically when the mapping is modified. On Unix, keys and values use [sys.getfilesystemencoding()](https://docs.python.org/3/library/sys.html#sys.getfilesystemencoding) and 'surrogateescape' error handler. Use [environb](https://docs.python.org/3/library/os.html#os.environb) if you would like to use a different encoding.

os.**environb**

Bytes version of [environ](https://docs.python.org/3/library/os.html#os.environ): a [mapping](https://docs.python.org/3/glossary.html#term-mapping) object where both keys and values are [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) objects representing the process environment. [environ](https://docs.python.org/3/library/os.html#os.environ) and [environb](https://docs.python.org/3/library/os.html#os.environb) are synchronized (modifying [environb](https://docs.python.org/3/library/os.html#os.environb) updates [environ](https://docs.python.org/3/library/os.html#os.environ), and vice versa). [environb](https://docs.python.org/3/library/os.html#os.environb) is only available if [supports\_bytes\_environ](https://docs.python.org/3/library/os.html#os.supports_bytes_environ) is True.

os.**fsencode**(*filename*)

Encode [path-like](https://docs.python.org/3/glossary.html#term-path-like-object) *filename* to the [filesystem encoding and error handler](https://docs.python.org/3/glossary.html#term-filesystem-encoding-and-error-handler); return [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) unchanged.

[fsdecode()](https://docs.python.org/3/library/os.html#os.fsdecode) is the reverse function.

os.**fsdecode**(*filename*)

Decode the [path-like](https://docs.python.org/3/glossary.html#term-path-like-object) *filename* from the [filesystem encoding and error handler](https://docs.python.org/3/glossary.html#term-filesystem-encoding-and-error-handler); return [str](https://docs.python.org/3/library/stdtypes.html#str) unchanged.

[fsencode()](https://docs.python.org/3/library/os.html#os.fsencode) is the reverse function.

os.**fspath**(*path*)

Return the file system representation of the path.

If [str](https://docs.python.org/3/library/stdtypes.html#str) or [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) is passed in, it is returned unchanged. Otherwise [\_\_fspath\_\_()](https://docs.python.org/3/library/os.html#os.PathLike.__fspath__) is called and its value is returned as long as it is a [str](https://docs.python.org/3/library/stdtypes.html#str) or [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) object. In all other cases, [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) is raised.

*class*os.**PathLike**

An [abstract base class](https://docs.python.org/3/glossary.html#term-abstract-base-class) for objects representing a file system path, e.g. [pathlib.PurePath](https://docs.python.org/3/library/pathlib.html#pathlib.PurePath).

os.**getenv**(*key*, *default=None*)

Return the value of the environment variable *key* if it exists, or *default* if it doesn’t. *key*, *default* and the result are str. Note that since [getenv()](https://docs.python.org/3/library/os.html#os.getenv) uses [os.environ](https://docs.python.org/3/library/os.html#os.environ), the mapping of [getenv()](https://docs.python.org/3/library/os.html#os.getenv) is similarly also captured on import, and the function may not reflect future environment changes. On Unix, keys and values are decoded with [sys.getfilesystemencoding()](https://docs.python.org/3/library/sys.html#sys.getfilesystemencoding) and 'surrogateescape' error handler. Use [os.getenvb()](https://docs.python.org/3/library/os.html#os.getenvb) if you would like to use a different encoding.

os.**getenvb**(*key*, *default=None*)

Return the value of the environment variable *key* if it exists, or *default* if it doesn’t. *key*, *default* and the result are bytes. Note that since [getenvb()](https://docs.python.org/3/library/os.html#os.getenvb) uses [os.environb](https://docs.python.org/3/library/os.html#os.environb), the mapping of [getenvb()](https://docs.python.org/3/library/os.html#os.getenvb) is similarly also captured on import, and the function may not reflect future environment changes.

[getenvb()](https://docs.python.org/3/library/os.html#os.getenvb) is only available if [supports\_bytes\_environ](https://docs.python.org/3/library/os.html#os.supports_bytes_environ) is True.

os.**get\_exec\_path**(*env=None*)

Returns the list of directories that will be searched for a named executable, similar to a shell, when launching a process. *env*, when specified, should be an environment variable dictionary to lookup the PATH in. By default, when *env* is None, [environ](https://docs.python.org/3/library/os.html#os.environ) is used.

os.**getegid**()

Return the effective group id of the current process. This corresponds to the “set id” bit on the file being executed in the current process.

os.**geteuid**()

Return the current process’s effective user id.

os.**getgid**()

Return the real group id of the current process.

os.**getgrouplist**(*user*, *group*)

Return list of group ids that *user* belongs to. If *group* is not in the list, it is included; typically, *group* is specified as the group ID field from the password record for *user*.

os.**getgroups**()

Return list of supplemental group ids associated with the current process.

os.**getlogin**()

Return the name of the user logged in on the controlling terminal of the process. For most purposes, it is more useful to use [getpass.getuser()](https://docs.python.org/3/library/getpass.html#getpass.getuser) since the latter checks the environment variables LOGNAME or USERNAME to find out who the user is, and falls back to pwd.getpwuid(os.getuid())[0] to get the login name of the current real user id.

os.**getpgid**(*pid*)

Return the process group id of the process with process id *pid*. If *pid* is 0, the process group id of the current process is returned.

os.**getpgrp**()

Return the id of the current process group.

os.**getpid**()

Return the current process id.

os.**getppid**()

Return the parent’s process id. When the parent process has exited, on Unix the id returned is the one of the init process (1), on Windows it is still the same id, which may be already reused by another process.

os.**getpriority**(*which*, *who*)

Get program scheduling priority. The value *which* is one of [PRIO\_PROCESS](https://docs.python.org/3/library/os.html#os.PRIO_PROCESS), [PRIO\_PGRP](https://docs.python.org/3/library/os.html#os.PRIO_PGRP), or [PRIO\_USER](https://docs.python.org/3/library/os.html#os.PRIO_USER), and *who* is interpreted relative to *which* (a process identifier for [PRIO\_PROCESS](https://docs.python.org/3/library/os.html#os.PRIO_PROCESS), process group identifier for [PRIO\_PGRP](https://docs.python.org/3/library/os.html#os.PRIO_PGRP), and a user ID for [PRIO\_USER](https://docs.python.org/3/library/os.html#os.PRIO_USER)). A zero value for *who* denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process.

os.**getresuid**()

Return a tuple (ruid, euid, suid) denoting the current process’s real, effective, and saved user ids.

os.**getresgid**()

Return a tuple (rgid, egid, sgid) denoting the current process’s real, effective, and saved group ids.

os.**getuid**()

Return the current process’s real user id.

os.**initgroups**(*username*, *gid*)

Call the system initgroups() to initialize the group access list with all of the groups of which the specified username is a member, plus the specified group id.

os.**putenv**(*key*, *value*)

Set the environment variable named *key* to the string *value*. Such changes to the environment affect subprocesses started with [os.system()](https://docs.python.org/3/library/os.html#os.system), [popen()](https://docs.python.org/3/library/os.html#os.popen) or [fork()](https://docs.python.org/3/library/os.html#os.fork) and [execv()](https://docs.python.org/3/library/os.html#os.execv). Assignments to items in [os.environ](https://docs.python.org/3/library/os.html#os.environ) are automatically translated into corresponding calls to [putenv()](https://docs.python.org/3/library/os.html#os.putenv); however, calls to [putenv()](https://docs.python.org/3/library/os.html#os.putenv) don’t update [os.environ](https://docs.python.org/3/library/os.html#os.environ), so it is actually preferable to assign to items of [os.environ](https://docs.python.org/3/library/os.html#os.environ). This also applies to [getenv()](https://docs.python.org/3/library/os.html#os.getenv) and [getenvb()](https://docs.python.org/3/library/os.html#os.getenvb), which respectively use [os.environ](https://docs.python.org/3/library/os.html#os.environ) and [os.environb](https://docs.python.org/3/library/os.html#os.environb) in their implementations.

os.**setegid**(*egid*)

Set the current process’s effective group id.

os.**seteuid**(*euid*)

Set the current process’s effective user id.

os.**setgid**(*gid*)

Set the current process’ group id.

os.**setgroups**(*groups*)

Set the list of supplemental group ids associated with the current process to *groups*. *groups* must be a sequence, and each element must be an integer identifying a group. This operation is typically available only to the superuser.

os.**setpgrp**()

Call the system call setpgrp() or setpgrp(0, 0) depending on which version is implemented (if any). See the Unix manual for the semantics.

os.**setpgid**(*pid*, *pgrp*)

Call the system call setpgid() to set the process group id of the process with id *pid* to the process group with id *pgrp*. See the Unix manual for the semantics.

os.**setpriority**(*which*, *who*, *priority*)

Set program scheduling priority. The value *which* is one of [PRIO\_PROCESS](https://docs.python.org/3/library/os.html#os.PRIO_PROCESS), [PRIO\_PGRP](https://docs.python.org/3/library/os.html#os.PRIO_PGRP), or [PRIO\_USER](https://docs.python.org/3/library/os.html#os.PRIO_USER), and *who* is interpreted relative to *which* (a process identifier for [PRIO\_PROCESS](https://docs.python.org/3/library/os.html#os.PRIO_PROCESS), process group identifier for [PRIO\_PGRP](https://docs.python.org/3/library/os.html#os.PRIO_PGRP), and a user ID for [PRIO\_USER](https://docs.python.org/3/library/os.html#os.PRIO_USER)). A zero value for *who* denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process. *priority* is a value in the range -20 to 19. The default priority is 0; lower priorities cause more favorable scheduling.

os.**setregid**(*rgid*, *egid*)

Set the current process’s real and effective group ids.

os.**setresgid**(*rgid*, *egid*, *sgid*)

Set the current process’s real, effective, and saved group ids.

os.**setresuid**(*ruid*, *euid*, *suid*)

Set the current process’s real, effective, and saved user ids.

os.**setreuid**(*ruid*, *euid*)

Set the current process’s real and effective user ids.

os.**getsid**(*pid*)

Call the system call getsid(). See the Unix manual for the semantics.

os.**setsid**()

Call the system call setsid(). See the Unix manual for the semantics.

os.**setuid**(*uid*)

Set the current process’s user id.

os.**strerror**(*code*)

Return the error message corresponding to the error code in *code*. On platforms where strerror() returns NULL when given an unknown error number, [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) is raised.

os.**supports\_bytes\_environ**

True if the native OS type of the environment is bytes (eg. False on Windows).

os.**umask**(*mask*)

Set the current numeric umask and return the previous umask.

os.**uname**()

Returns information identifying the current operating system. The return value is an object with five attributes:

sysname - operating system name

nodename - name of machine on network (implementation-defined)

release - operating system release

version - operating system version

machine - hardware identifier

For backwards compatibility, this object is also iterable, behaving like a five-tuple containing sysname, nodename, release, version, and machine in that order.

Some systems truncate nodename to 8 characters or to the leading component; a better way to get the hostname is [socket.gethostname()](https://docs.python.org/3/library/socket.html#socket.gethostname) or even socket.gethostbyaddr(socket.gethostname()).

os.**unsetenv**(*key*)

Unset (delete) the environment variable named *key*. Such changes to the environment affect subprocesses started with [os.system()](https://docs.python.org/3/library/os.html#os.system), [popen()](https://docs.python.org/3/library/os.html#os.popen) or [fork()](https://docs.python.org/3/library/os.html#os.fork) and [execv()](https://docs.python.org/3/library/os.html#os.execv).

os.**fdopen**( *fd* , *\* args* , *\*\* kwargs* )

Retorna um objeto de arquivo aberto conectado ao descritor de arquivo *fd* . Este é um alias da [open()](https://docs.python.org/3/library/functions.html#open)função interna e aceita os mesmos argumentos. A única diferença é que o primeiro argumento de [fdopen()](https://docs.python.org/3/library/os.html#os.fdopen)deve ser sempre um inteiro.

os.**closerange**( *fd\_low* , *fd\_high* )

Feche todos os descritores de arquivo de *fd\_low* (inclusive) a *fd\_high* (exclusivo), ignorando erros. Equivalente a (mas muito mais rápido que):

os.**copy\_file\_range**( *src* , *dst* , *contagem* , *offset\_src = Nenhum* , *offset\_dst = Nenhum* )

Copia os bytes de *contagem do descritor de arquivo src* , a partir do deslocamento *offset\_src* , para o descritor de arquivo *dst* , a partir do deslocamento *offset\_dst* . Se *offset\_src* for Nenhum, então *src* será lido da posição atual; respectivamente para *offset\_dst* . Os arquivos apontados por *src* e *dst* devem residir no mesmo sistema de arquivos, caso contrário, um [OSError](https://docs.python.org/3/library/exceptions.html#OSError)é gerado com [errno](https://docs.python.org/3/library/exceptions.html#OSError.errno)definido como [errno.EXDEV](https://docs.python.org/3/library/errno.html#errno.EXDEV).

os.**device\_encoding**( *fd* )

Retorna uma string descrevendo a codificação do dispositivo associado a *fd* se estiver conectado a um terminal; senão retornar [None](https://docs.python.org/3/library/constants.html#None).

os.**dup**( *fd* )

Retorna uma duplicata do descritor de arquivo *fd* . O novo descritor [de arquivo não é herdável](https://docs.python.org/3/library/os.html#fd-inheritance) . No Windows, ao duplicar um fluxo padrão (0: stdin, 1: stdout, 2: stderr), o novo descritor de arquivo é [herdável](https://docs.python.org/3/library/os.html#fd-inheritance) .

os.**dup2**( *fd* , *fd2* , *herdável = True* )

Duplique o descritor de arquivo *fd* para *fd2* , fechando o último primeiro, se necessário. Retorna *fd2* . O novo descritor de arquivo é [herdável](https://docs.python.org/3/library/os.html#fd-inheritance) por padrão ou não herdável se *herdável* for False.

os.**fchmod**( *fd* , *modo* )

Altere o modo do arquivo fornecido por *fd para o modo* numérico . Consulte os documentos para [chmod()](https://docs.python.org/3/library/os.html#os.chmod)possíveis valores de *mode* . A partir do Python 3.3, isso é equivalente a .os.chmod(fd, mode)

os.**fchown**( *fd* , *uid* , *gid* )

Altere o proprietário e o ID do grupo do arquivo fornecido por *fd para o uid* e *gid* numéricos . Para deixar um dos ids inalterado, defina-o como -1. Veja [chown()](https://docs.python.org/3/library/os.html#os.chown). A partir do Python 3.3, isso é equivalente a .os.chown(fd, uid, gid)

os.**fdatasync**( *fd* )

Força a gravação do arquivo com o descritor de arquivo *fd* no disco. Não força a atualização de metadados.

os.**fpathconf**( *fd* , *nome* )

Return system configuration information relevant to an open file. *name* specifies the configuration value to retrieve; it may be a string which is the name of a defined system value; these names are specified in a number of standards (POSIX.1, Unix 95, Unix 98, and others). Some platforms define additional names as well. The names known to the host operating system are given in the pathconf\_names dictionary. For configuration variables not included in that mapping, passing an integer for *name* is also accepted. If *name* is a string and is not known, [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) is raised. If a specific value for *name* is not supported by the host system, even if it is included in pathconf\_names, an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised with [errno.EINVAL](https://docs.python.org/3/library/errno.html#errno.EINVAL) for the error number.

os.**fstat**(*fd*)

Get the status of the file descriptor *fd*. Return a [stat\_result](https://docs.python.org/3/library/os.html#os.stat_result) object.

os.**fstatvfs**(*fd*)

Return information about the filesystem containing the file associated with file descriptor *fd*, like [statvfs()](https://docs.python.org/3/library/os.html#os.statvfs). As of Python 3.3, this is equivalent to os.statvfs(fd).

os.**fsync**(*fd*)

Force write of file with filedescriptor *fd* to disk. On Unix, this calls the native fsync() function; on Windows, the MS \_commit() function. If you’re starting with a buffered Python [file object](https://docs.python.org/3/glossary.html#term-file-object) *f*, first do f.flush(), and then do os.fsync(f.fileno()), to ensure that all internal buffers associated with *f* are written to disk.

os.**ftruncate**(*fd*, *length*)

Truncate the file corresponding to file descriptor *fd*, so that it is at most *length* bytes in size. As of Python 3.3, this is equivalent to os.truncate(fd, length). Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.truncate with arguments fd, length.

os.**get\_blocking**(*fd*)

Get the blocking mode of the file descriptor: False if the [O\_NONBLOCK](https://docs.python.org/3/library/os.html#os.O_NONBLOCK) flag is set, True if the flag is cleared. See also [set\_blocking()](https://docs.python.org/3/library/os.html#os.set_blocking) and [socket.socket.setblocking()](https://docs.python.org/3/library/socket.html#socket.socket.setblocking).

os.**isatty**(*fd*)

Return True if the file descriptor *fd* is open and connected to a tty(-like) device, else False.

os.**lockf**(*fd*, *cmd*, *len*)

Apply, test or remove a POSIX lock on an open file descriptor. *fd* is an open file descriptor. *cmd* specifies the command to use - one of [F\_LOCK](https://docs.python.org/3/library/os.html#os.F_LOCK), [F\_TLOCK](https://docs.python.org/3/library/os.html#os.F_TLOCK), [F\_ULOCK](https://docs.python.org/3/library/os.html#os.F_ULOCK) or [F\_TEST](https://docs.python.org/3/library/os.html#os.F_TEST). *len* specifies the section of the file to lock. Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.lockf with arguments fd, cmd, len.

os.**F\_LOCK**

os.**F\_TLOCK**

os.**F\_ULOCK**

os.**F\_TEST**

Flags that specify what action [lockf()](https://docs.python.org/3/library/os.html#os.lockf) will take.

os.**lseek**(*fd*, *pos*, *how*)

Set the current position of file descriptor *fd* to position *pos*, modified by *how*: [SEEK\_SET](https://docs.python.org/3/library/os.html#os.SEEK_SET) or 0 to set the position relative to the beginning of the file; [SEEK\_CUR](https://docs.python.org/3/library/os.html#os.SEEK_CUR) or 1 to set it relative to the current position; [SEEK\_END](https://docs.python.org/3/library/os.html#os.SEEK_END) or 2 to set it relative to the end of the file. Return the new cursor position in bytes, starting from the beginning.

os.**open**(*path*, *flags*, *mode=0o777*, *\**, *dir\_fd=None*)

Open the file *path* and set various flags according to *flags* and possibly its mode according to *mode*. When computing *mode*, the current umask value is first masked out. Return the file descriptor for the newly opened file. The new file descriptor is [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance).

For a description of the flag and mode values, see the C run-time documentation; flag constants (like [O\_RDONLY](https://docs.python.org/3/library/os.html#os.O_RDONLY) and [O\_WRONLY](https://docs.python.org/3/library/os.html#os.O_WRONLY)) are defined in the [os](https://docs.python.org/3/library/os.html#module-os) module. In particular, on Windows adding [O\_BINARY](https://docs.python.org/3/library/os.html#os.O_BINARY) is needed to open files in binary mode.

This function can support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd) with the *dir\_fd* parameter.

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) open with arguments path, mode, flags.

*Changed in version 3.4:*The new file descriptor is now non-inheritable.

This function is intended for low-level I/O. For normal usage, use the built-in function [open()](https://docs.python.org/3/library/functions.html#open), which returns a [file object](https://docs.python.org/3/glossary.html#term-file-object) with read() and write() methods (and many more). To wrap a file descriptor in a file object, use [fdopen()](https://docs.python.org/3/library/os.html#os.fdopen).

os.**openpty**()

Open a new pseudo-terminal pair. Return a pair of file descriptors (master, slave) for the pty and the tty, respectively. The new file descriptors are [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance). For a (slightly) more portable approach, use the [pty](https://docs.python.org/3/library/pty.html#module-pty) module.

os.**pipe**()

Create a pipe. Return a pair of file descriptors (r, w) usable for reading and writing, respectively. The new file descriptor is [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance).

os.**pipe2**(*flags*)

Create a pipe with *flags* set atomically. *flags* can be constructed by ORing together one or more of these values: [O\_NONBLOCK](https://docs.python.org/3/library/os.html#os.O_NONBLOCK), [O\_CLOEXEC](https://docs.python.org/3/library/os.html#os.O_CLOEXEC). Return a pair of file descriptors (r, w) usable for reading and writing, respectively.

os.**posix\_fallocate**(*fd*, *offset*, *len*)

Ensures that enough disk space is allocated for the file specified by *fd* starting from *offset* and continuing for *len* bytes.

os.**posix\_fadvise**(*fd*, *offset*, *len*, *advice*)

Announces an intention to access data in a specific pattern thus allowing the kernel to make optimizations. The advice applies to the region of the file specified by *fd* starting at *offset* and continuing for *len* bytes. *advice* is one of [POSIX\_FADV\_NORMAL](https://docs.python.org/3/library/os.html#os.POSIX_FADV_NORMAL), [POSIX\_FADV\_SEQUENTIAL](https://docs.python.org/3/library/os.html#os.POSIX_FADV_SEQUENTIAL), [POSIX\_FADV\_RANDOM](https://docs.python.org/3/library/os.html#os.POSIX_FADV_RANDOM), [POSIX\_FADV\_NOREUSE](https://docs.python.org/3/library/os.html#os.POSIX_FADV_NOREUSE), [POSIX\_FADV\_WILLNEED](https://docs.python.org/3/library/os.html#os.POSIX_FADV_WILLNEED) or [POSIX\_FADV\_DONTNEED](https://docs.python.org/3/library/os.html#os.POSIX_FADV_DONTNEED).

os.**pread**(*fd*, *n*, *offset*)

Read at most *n* bytes from file descriptor *fd* at a position of *offset*, leaving the file offset unchanged.

Return a bytestring containing the bytes read. If the end of the file referred to by *fd* has been reached, an empty bytes object is returned.

os.**preadv**(*fd*, *buffers*, *offset*, *flags=0*)

Read from a file descriptor *fd* at a position of *offset* into mutable [bytes-like objects](https://docs.python.org/3/glossary.html#term-bytes-like-object) *buffers*, leaving the file offset unchanged. Transfer data into each buffer until it is full and then move on to the next buffer in the sequence to hold the rest of the data.

os.**RWF\_NOWAIT**

Do not wait for data which is not immediately available. If this flag is specified, the system call will return instantly if it would have to read data from the backing storage or wait for a lock.

If some data was successfully read, it will return the number of bytes read. If no bytes were read, it will return -1 and set errno to [errno.EAGAIN](https://docs.python.org/3/library/errno.html#errno.EAGAIN).

os.**RWF\_HIPRI**

High priority read/write. Allows block-based filesystems to use polling of the device, which provides lower latency, but may use additional resources.

Currently, on Linux, this feature is usable only on a file descriptor opened using the [O\_DIRECT](https://docs.python.org/3/library/os.html#os.O_DIRECT) flag.

os.**pwrite**(*fd*, *str*, *offset*)

Write the bytestring in *str* to file descriptor *fd* at position of *offset*, leaving the file offset unchanged.

Return the number of bytes actually written.

os.**pwritev**(*fd*, *buffers*, *offset*, *flags=0*)

Write the *buffers* contents to file descriptor *fd* at a offset *offset*, leaving the file offset unchanged. *buffers* must be a sequence of [bytes-like objects](https://docs.python.org/3/glossary.html#term-bytes-like-object). Buffers are processed in array order. Entire contents of the first buffer is written before proceeding to the second, and so on.

os.**RWF\_DSYNC**

Provide a per-write equivalent of the [O\_DSYNC](https://docs.python.org/3/library/os.html#os.O_DSYNC) [os.open()](https://docs.python.org/3/library/os.html#os.open) flag. This flag effect applies only to the data range written by the system call.

os.**RWF\_SYNC**

Provide a per-write equivalent of the [O\_SYNC](https://docs.python.org/3/library/os.html#os.O_SYNC) [os.open()](https://docs.python.org/3/library/os.html#os.open) flag. This flag effect applies only to the data range written by the system call.

os.**RWF\_APPEND**

Provide a per-write equivalent of the [O\_APPEND](https://docs.python.org/3/library/os.html#os.O_APPEND) [os.open()](https://docs.python.org/3/library/os.html#os.open) flag. This flag is meaningful only for [os.pwritev()](https://docs.python.org/3/library/os.html#os.pwritev), and its effect applies only to the data range written by the system call. The *offset* argument does not affect the write operation; the data is always appended to the end of the file. However, if the *offset* argument is -1, the current file *offset* is updated.

os.**read**(*fd*, *n*)

Read at most *n* bytes from file descriptor *fd*.

Return a bytestring containing the bytes read. If the end of the file referred to by *fd* has been reached, an empty bytes object is returned.

os.**sendfile**(*out\_fd*, *in\_fd*, *offset*, *count*, *headers=()*, *trailers=()*, *flags=0*)

Copy *count* bytes from file descriptor *in\_fd* to file descriptor *out\_fd* starting at *offset*. Return the number of bytes sent. When EOF is reached return 0.

The first function notation is supported by all platforms that define [sendfile()](https://docs.python.org/3/library/os.html#os.sendfile).

On Linux, if *offset* is given as None, the bytes are read from the current position of *in\_fd* and the position of *in\_fd* is updated.

The second case may be used on macOS and FreeBSD where *headers* and *trailers* are arbitrary sequences of buffers that are written before and after the data from *in\_fd* is written. It returns the same as the first case.

On macOS and FreeBSD, a value of 0 for *count* specifies to send until the end of *in\_fd* is reached.

All platforms support sockets as *out\_fd* file descriptor, and some platforms allow other types (e.g. regular file, pipe) as well.

Cross-platform applications should not use *headers*, *trailers* and *flags* arguments.

os.**set\_blocking**(*fd*, *blocking*)

Set the blocking mode of the specified file descriptor. Set the [O\_NONBLOCK](https://docs.python.org/3/library/os.html#os.O_NONBLOCK) flag if blocking is False, clear the flag otherwise. See also [get\_blocking()](https://docs.python.org/3/library/os.html#os.get_blocking) and [socket.socket.setblocking()](https://docs.python.org/3/library/socket.html#socket.socket.setblocking).

os.**splice**(*src*, *dst*, *count*, *offset\_src=None*, *offset\_dst=None*)

Transfer *count* bytes from file descriptor *src*, starting from offset *offset\_src*, to file descriptor *dst*, starting from offset *offset\_dst*. At least one of the file descriptors must refer to a pipe. If *offset\_src* is None, then *src* is read from the current position; respectively for *offset\_dst*. The offset associated to the file descriptor that refers to a pipe must be None. The files pointed by *src* and *dst* must reside in the same filesystem, otherwise an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised with [errno](https://docs.python.org/3/library/exceptions.html#OSError.errno) set to [errno.EXDEV](https://docs.python.org/3/library/errno.html#errno.EXDEV). This copy is done without the additional cost of transferring data from the kernel to user space and then back into the kernel. Additionally, some filesystems could implement extra optimizations. The copy is done as if both files are opened as binary. Upon successful completion, returns the number of bytes spliced to or from the pipe. A return value of 0 means end of input. If *src* refers to a pipe, then this means that there was no data to transfer, and it would not make sense to block because there are no writers connected to the write end of the pipe.

os.**readv**(*fd*, *buffers*)

Read from a file descriptor *fd* into a number of mutable [bytes-like objects](https://docs.python.org/3/glossary.html#term-bytes-like-object) *buffers*. Transfer data into each buffer until it is full and then move on to the next buffer in the sequence to hold the rest of the data. Return the total number of bytes actually read which can be less than the total capacity of all the objects. The operating system may set a limit ([sysconf()](https://docs.python.org/3/library/os.html#os.sysconf) value 'SC\_IOV\_MAX') on the number of buffers that can be used.

os.**tcgetpgrp**(*fd*)

Return the process group associated with the terminal given by *fd* (an open file descriptor as returned by [os.open()](https://docs.python.org/3/library/os.html#os.open)).

os.**tcsetpgrp**(*fd*, *pg*)

Set the process group associated with the terminal given by *fd* (an open file descriptor as returned by [os.open()](https://docs.python.org/3/library/os.html#os.open)) to *pg*.

os.**ttyname**(*fd*)

Return a string which specifies the terminal device associated with file descriptor *fd*. If *fd* is not associated with a terminal device, an exception is raised.

os.**write**(*fd*, *str*)

Write the bytestring in *str* to file descriptor *fd*. Return the number of bytes actually written. This function is intended for low-level I/O and must be applied to a file descriptor as returned by [os.open()](https://docs.python.org/3/library/os.html#os.open) or [pipe()](https://docs.python.org/3/library/os.html#os.pipe). To write a “file object” returned by the built-in function [open()](https://docs.python.org/3/library/functions.html#open) or by [popen()](https://docs.python.org/3/library/os.html#os.popen) or [fdopen()](https://docs.python.org/3/library/os.html#os.fdopen), or [sys.stdout](https://docs.python.org/3/library/sys.html#sys.stdout) or [sys.stderr](https://docs.python.org/3/library/sys.html#sys.stderr), use its write() method.

os.**writev**(*fd*, *buffers*)

Write the contents of *buffers* to file descriptor *fd*. *buffers* must be a sequence of [bytes-like objects](https://docs.python.org/3/glossary.html#term-bytes-like-object). Buffers are processed in array order. Entire contents of the first buffer is written before proceeding to the second, and so on. Returns the total number of bytes actually written.

The operating system may set a limit ([sysconf()](https://docs.python.org/3/library/os.html#os.sysconf) value 'SC\_IOV\_MAX') on the number of buffers that can be used.

os.**get\_terminal\_size**(*fd=STDOUT\_FILENO*)

Return the size of the terminal window as (columns, lines), tuple of type [terminal\_size](https://docs.python.org/3/library/os.html#os.terminal_size).

The optional argument fd (default STDOUT\_FILENO, or standard output) specifies which file descriptor should be queried.

If the file descriptor is not connected to a terminal, an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised.

[shutil.get\_terminal\_size()](https://docs.python.org/3/library/shutil.html#shutil.get_terminal_size) is the high-level function which should normally be used, os.get\_terminal\_size is the low-level implementation.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix, Windows.

*class*os.**terminal\_size**

A subclass of tuple, holding (columns, lines) of the terminal window size.

os.**get\_inheritable**(*fd*)

Get the “inheritable” flag of the specified file descriptor (a boolean).

os.**set\_inheritable**(*fd*, *inheritable*)

Set the “inheritable” flag of the specified file descriptor.

os.**get\_handle\_inheritable**(*handle*)

Get the “inheritable” flag of the specified handle (a boolean).

os.**set\_handle\_inheritable**(*handle*, *inheritable*)

Set the “inheritable” flag of the specified handle.

os.**chdir**(*path*)

Change the current working directory to *path*. This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd). The descriptor must refer to an opened directory, not an open file. This function can raise [OSError](https://docs.python.org/3/library/exceptions.html#OSError) and subclasses such as [FileNotFoundError](https://docs.python.org/3/library/exceptions.html#FileNotFoundError), [PermissionError](https://docs.python.org/3/library/exceptions.html#PermissionError), and [NotADirectoryError](https://docs.python.org/3/library/exceptions.html#NotADirectoryError).

os.**chflags**(*path*, *flags*, *\**, *follow\_symlinks=True*)

Set the flags of *path* to the numeric *flags*. *flags* may take a combination (bitwise OR) of the following values (as defined in the [stat](https://docs.python.org/3/library/stat.html#module-stat) module):

os.**chmod**(*path*, *mode*, *\**, *dir\_fd=None*, *follow\_symlinks=True*)

Change the mode of *path* to the numeric *mode*. *mode* may take one of the following values (as defined in the [stat](https://docs.python.org/3/library/stat.html#module-stat) module) or bitwise ORed combinations of them:

os.**chown**(*path*, *uid*, *gid*, *\**, *dir\_fd=None*, *follow\_symlinks=True*)

Change the owner and group id of *path* to the numeric *uid* and *gid*. To leave one of the ids unchanged, set it to -1. This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd), [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd) and [not following symlinks](https://docs.python.org/3/library/os.html#follow-symlinks).

See [shutil.chown()](https://docs.python.org/3/library/shutil.html#shutil.chown) for a higher-level function that accepts names in addition to numeric ids.

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.chown with arguments path, uid, gid, dir\_fd.

os.**chroot**(*path*)

Change the root directory of the current process to *path*.

os.**fchdir**(*fd*)

Change the current working directory to the directory represented by the file descriptor *fd*. The descriptor must refer to an opened directory, not an open file. As of Python 3.3, this is equivalent to os.chdir(fd).

os.**getcwd**()

Return a string representing the current working directory.

os.**getcwdb**()

Return a bytestring representing the current working directory.

os.**lchflags**(*path*, *flags*)

Set the flags of *path* to the numeric *flags*, like [chflags()](https://docs.python.org/3/library/os.html#os.chflags), but do not follow symbolic links. As of Python 3.3, this is equivalent to os.chflags(path, flags, follow\_symlinks=False).

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.chflags with arguments path, flags.

os.**lchmod**(*path*, *mode*)

Change the mode of *path* to the numeric *mode*. If path is a symlink, this affects the symlink rather than the target. See the docs for [chmod()](https://docs.python.org/3/library/os.html#os.chmod) for possible values of *mode*. As of Python 3.3, this is equivalent to os.chmod(path, mode, follow\_symlinks=False).

os.**lchown**(*path*, *uid*, *gid*)

Change the owner and group id of *path* to the numeric *uid* and *gid*. This function will not follow symbolic links. As of Python 3.3, this is equivalent to os.chown(path, uid, gid, follow\_symlinks=False).

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.chown with arguments path, uid, gid, dir\_fd.

os.**link**(*src*, *dst*, *\**, *src\_dir\_fd=None*, *dst\_dir\_fd=None*, *follow\_symlinks=True*)

Create a hard link pointing to *src* named *dst*.

This function can support specifying *src\_dir\_fd* and/or *dst\_dir\_fd* to supply [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd), and [not following symlinks](https://docs.python.org/3/library/os.html#follow-symlinks).

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.link with arguments src, dst, src\_dir\_fd, dst\_dir\_fd.

os.**listdir**(*path='.'*)

Return a list containing the names of the entries in the directory given by *path*. The list is in arbitrary order, and does not include the special entries '.' and '..' even if they are present in the directory. If a file is removed from or added to the directory during the call of this function, whether a name for that file be included is unspecified. *path* may be a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object). If *path* is of type bytes (directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface), the filenames returned will also be of type bytes; in all other circumstances, they will be of type str. This function can also support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd); the file descriptor must refer to a directory. Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.listdir with argument path.

os.**lstat**(*path*, *\**, *dir\_fd=None*)

Perform the equivalent of an lstat() system call on the given path. Similar to [stat()](https://docs.python.org/3/library/os.html#os.stat), but does not follow symbolic links. Return a [stat\_result](https://docs.python.org/3/library/os.html#os.stat_result) object.

On platforms that do not support symbolic links, this is an alias for [stat()](https://docs.python.org/3/library/os.html#os.stat).

As of Python 3.3, this is equivalent to os.stat(path, dir\_fd=dir\_fd, follow\_symlinks=False).

This function can also support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd).

os.**mkdir**(*path*, *mode=0o777*, *\**, *dir\_fd=None*)

Create a directory named *path* with numeric mode *mode*.

If the directory already exists, [FileExistsError](https://docs.python.org/3/library/exceptions.html#FileExistsError) is raised. If a parent directory in the path does not exist, [FileNotFoundError](https://docs.python.org/3/library/exceptions.html#FileNotFoundError) is raised. On some systems, *mode* is ignored. Where it is used, the current umask value is first masked out. If bits other than the last 9 (i.e. the last 3 digits of the octal representation of the *mode*) are set, their meaning is platform-dependent. On some platforms, they are ignored and you should call [chmod()](https://docs.python.org/3/library/os.html#os.chmod) explicitly to set them. This function can also support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd). It is also possible to create temporary directories; see the [tempfile](https://docs.python.org/3/library/tempfile.html#module-tempfile) module’s [tempfile.mkdtemp()](https://docs.python.org/3/library/tempfile.html#tempfile.mkdtemp) function. Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.mkdir with arguments path, mode, dir\_fd.

os.**makedirs**(*name*, *mode=0o777*, *exist\_ok=False*)

Recursive directory creation function. Like [mkdir()](https://docs.python.org/3/library/os.html#os.mkdir), but makes all intermediate-level directories needed to contain the leaf directory.

The *mode* parameter is passed to [mkdir()](https://docs.python.org/3/library/os.html#os.mkdir) for creating the leaf directory; see [the mkdir() description](https://docs.python.org/3/library/os.html#mkdir-modebits) for how it is interpreted. To set the file permission bits of any newly created parent directories you can set the umask before invoking [makedirs()](https://docs.python.org/3/library/os.html#os.makedirs). The file permission bits of existing parent directories are not changed. If *exist\_ok* is False (the default), an [FileExistsError](https://docs.python.org/3/library/exceptions.html#FileExistsError) is raised if the target directory already exists.

os.**mkfifo**(*path*, *mode=0o666*, *\**, *dir\_fd=None*)

Create a FIFO (a named pipe) named *path* with numeric mode *mode*. The current umask value is first masked out from the mode.

This function can also support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd).

FIFOs are pipes that can be accessed like regular files. FIFOs exist until they are deleted (for example with [os.unlink()](https://docs.python.org/3/library/os.html#os.unlink)). Generally, FIFOs are used as rendezvous between “client” and “server” type processes: the server opens the FIFO for reading, and the client opens it for writing. Note that [mkfifo()](https://docs.python.org/3/library/os.html#os.mkfifo) doesn’t open the FIFO — it just creates the rendezvous point.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.3:*The *dir\_fd* argument.

*Changed in version 3.6:*Accepts a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object).

os.**mknod**(*path*, *mode=0o600*, *device=0*, *\**, *dir\_fd=None*)

Create a filesystem node (file, device special file or named pipe) named *path*. *mode* specifies both the permissions to use and the type of node to be created, being combined (bitwise OR) with one of stat.S\_IFREG, stat.S\_IFCHR, stat.S\_IFBLK, and stat.S\_IFIFO (those constants are available in [stat](https://docs.python.org/3/library/stat.html#module-stat)). For stat.S\_IFCHR and stat.S\_IFBLK, *device* defines the newly created device special file (probably using [os.makedev()](https://docs.python.org/3/library/os.html#os.makedev)), otherwise it is ignored.

This function can also support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd).

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.3:*The *dir\_fd* argument.

*Changed in version 3.6:*Accepts a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object).

os.**major**(*device*)

Extract the device major number from a raw device number (usually the st\_dev or st\_rdev field from stat).

os.**minor**(*device*)

Extract the device minor number from a raw device number (usually the st\_dev or st\_rdev field from stat).

os.**makedev**(*major*, *minor*)

Compose a raw device number from the major and minor device numbers.

os.**pathconf**(*path*, *name*)

Return system configuration information relevant to a named file. *name* specifies the configuration value to retrieve; it may be a string which is the name of a defined system value; these names are specified in a number of standards (POSIX.1, Unix 95, Unix 98, and others). Some platforms define additional names as well. The names known to the host operating system are given in the pathconf\_names dictionary. For configuration variables not included in that mapping, passing an integer for *name* is also accepted. If *name* is a string and is not known, [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) is raised. If a specific value for *name* is not supported by the host system, even if it is included in pathconf\_names, an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised with [errno.EINVAL](https://docs.python.org/3/library/errno.html#errno.EINVAL) for the error number. This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd).

os.**pathconf\_names**

Dictionary mapping names accepted by [pathconf()](https://docs.python.org/3/library/os.html#os.pathconf) and [fpathconf()](https://docs.python.org/3/library/os.html#os.fpathconf) to the integer values defined for those names by the host operating system. This can be used to determine the set of names known to the system.

os.**readlink**(*path*, *\**, *dir\_fd=None*)

Return a string representing the path to which the symbolic link points. The result may be either an absolute or relative pathname; if it is relative, it may be converted to an absolute pathname using os.path.join(os.path.dirname(path), result).

If the *path* is a string object (directly or indirectly through a [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface), the result will also be a string object, and the call may raise a UnicodeDecodeError. If the *path* is a bytes object (direct or indirectly), the result will be a bytes object.

This function can also support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd).

When trying to resolve a path that may contain links, use [realpath()](https://docs.python.org/3/library/os.path.html#os.path.realpath) to properly handle recursion and platform differences.

os.**remove**(*path*, *\**, *dir\_fd=None*)

Remove (delete) the file *path*. If *path* is a directory, an [IsADirectoryError](https://docs.python.org/3/library/exceptions.html#IsADirectoryError) is raised. Use [rmdir()](https://docs.python.org/3/library/os.html#os.rmdir) to remove directories. If the file does not exist, a [FileNotFoundError](https://docs.python.org/3/library/exceptions.html#FileNotFoundError) is raised.

This function can support [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd).

On Windows, attempting to remove a file that is in use causes an exception to be raised; on Unix, the directory entry is removed but the storage allocated to the file is not made available until the original file is no longer in use.

This function is semantically identical to [unlink()](https://docs.python.org/3/library/os.html#os.unlink).

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.remove with arguments path, dir\_fd.

os.**removedirs**(*name*)

Remove directories recursively. Works like [rmdir()](https://docs.python.org/3/library/os.html#os.rmdir) except that, if the leaf directory is successfully removed, [removedirs()](https://docs.python.org/3/library/os.html#os.removedirs) tries to successively remove every parent directory mentioned in *path* until an error is raised (which is ignored, because it generally means that a parent directory is not empty). For example, os.removedirs('foo/bar/baz') will first remove the directory 'foo/bar/baz', and then remove 'foo/bar' and 'foo' if they are empty. Raises [OSError](https://docs.python.org/3/library/exceptions.html#OSError) if the leaf directory could not be successfully removed. Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.remove with arguments path, dir\_fd.

os.**rename**(*src*, *dst*, *\**, *src\_dir\_fd=None*, *dst\_dir\_fd=None*)

Rename the file or directory *src* to *dst*. If *dst* exists, the operation will fail with an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) subclass in a number of cases: On Windows, if *dst* exists a [FileExistsError](https://docs.python.org/3/library/exceptions.html#FileExistsError) is always raised.

os.**renames**(*old*, *new*)

Recursive directory or file renaming function. Works like [rename()](https://docs.python.org/3/library/os.html#os.rename), except creation of any intermediate directories needed to make the new pathname good is attempted first. After the rename, directories corresponding to rightmost path segments of the old name will be pruned away using [removedirs()](https://docs.python.org/3/library/os.html#os.removedirs). This function can fail with the new directory structure made if you lack permissions needed to remove the leaf directory or file. Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.rename with arguments src, dst, src\_dir\_fd, dst\_dir\_fd.

os.**replace**(*src*, *dst*, *\**, *src\_dir\_fd=None*, *dst\_dir\_fd=None*)

Rename the file or directory *src* to *dst*. If *dst* is a non-empty directory, [OSError](https://docs.python.org/3/library/exceptions.html#OSError) will be raised. If *dst* exists and is a file, it will be replaced silently if the user has permission. The operation may fail if *src* and *dst* are on different filesystems. If successful, the renaming will be an atomic operation (this is a POSIX requirement). This function can support specifying *src\_dir\_fd* and/or *dst\_dir\_fd* to supply [paths relative to directory descriptors](https://docs.python.org/3/library/os.html#dir-fd). Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) os.rename with arguments src, dst, src\_dir\_fd, dst\_dir\_fd.

os.**rmdir**(*path*, *\**, *dir\_fd=None*)

Remove (delete) the directory *path*. If the directory does not exist or is not empty, an [FileNotFoundError](https://docs.python.org/3/library/exceptions.html#FileNotFoundError) or an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised respectively. In order to remove whole directory trees, [shutil.rmtree()](https://docs.python.org/3/library/shutil.html#shutil.rmtree) can be used.

os.**scandir**(*path='.'*)

Return an iterator of [os.DirEntry](https://docs.python.org/3/library/os.html#os.DirEntry) objects corresponding to the entries in the directory given by *path*. The entries are yielded in arbitrary order, and the special entries '.' and '..' are not included. If a file is removed from or added to the directory after creating the iterator, whether an entry for that file be included is unspecified. Using [scandir()](https://docs.python.org/3/library/os.html#os.scandir) instead of [listdir()](https://docs.python.org/3/library/os.html#os.listdir) can significantly increase the performance of code that also needs file type or file attribute information, because [os.DirEntry](https://docs.python.org/3/library/os.html#os.DirEntry) objects expose this information if the operating system provides it when scanning a directory. All [os.DirEntry](https://docs.python.org/3/library/os.html#os.DirEntry) methods may perform a system call, but [is\_dir()](https://docs.python.org/3/library/os.html#os.DirEntry.is_dir) and [is\_file()](https://docs.python.org/3/library/os.html#os.DirEntry.is_file) usually only require a system call for symbolic links; [os.DirEntry.stat()](https://docs.python.org/3/library/os.html#os.DirEntry.stat) always requires a system call on Unix but only requires one for symbolic links on Windows. *path* may be a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object). If *path* is of type bytes (directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface), the type of the [name](https://docs.python.org/3/library/os.html#os.DirEntry.name) and [path](https://docs.python.org/3/library/os.html#os.DirEntry.path) attributes of each [os.DirEntry](https://docs.python.org/3/library/os.html#os.DirEntry) will be bytes; in all other circumstances, they will be of type str. This function can also support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd); the file descriptor must refer to a directory.

scandir.**close**()

Close the iterator and free acquired resources.

This is called automatically when the iterator is exhausted or garbage collected, or when an error happens during iterating. However it is advisable to call it explicitly or use the [with](https://docs.python.org/3/reference/compound_stmts.html#with) statement.

*class*os.**DirEntry**

Object yielded by [scandir()](https://docs.python.org/3/library/os.html#os.scandir) to expose the file path and other file attributes of a directory entry.

[scandir()](https://docs.python.org/3/library/os.html#os.scandir) will provide as much of this information as possible without making additional system calls. When a stat() or lstat() system call is made, the os.DirEntry object will cache the result.

os.DirEntry instances are not intended to be stored in long-lived data structures; if you know the file metadata has changed or if a long time has elapsed since calling [scandir()](https://docs.python.org/3/library/os.html#os.scandir), call os.stat(entry.path) to fetch up-to-date information. Because the os.DirEntry methods can make operating system calls, they may also raise [OSError](https://docs.python.org/3/library/exceptions.html#OSError). If you need very fine-grained control over errors, you can catch [OSError](https://docs.python.org/3/library/exceptions.html#OSError) when calling one of the os.DirEntry methods and handle as appropriate. To be directly usable as a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object), os.DirEntry implements the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface. Attributes and methods on a os.DirEntry instance are as follows:

os.**stat**(*path*, *\**, *dir\_fd=None*, *follow\_symlinks=True*)

Get the status of a file or a file descriptor. Perform the equivalent of a stat() system call on the given path. *path* may be specified as either a string or bytes – directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface – or as an open file descriptor. Return a [stat\_result](https://docs.python.org/3/library/os.html#os.stat_result) object. This function normally follows symlinks; to stat a symlink add the argument follow\_symlinks=False, or use [lstat()](https://docs.python.org/3/library/os.html#os.lstat). This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd) and [not following symlinks](https://docs.python.org/3/library/os.html#follow-symlinks).

On Windows, passing follow\_symlinks=False will disable following all name-surrogate reparse points, which includes symlinks and directory junctions. Other types of reparse points that do not resemble links or that the operating system is unable to follow will be opened directly. When following a chain of multiple links, this may result in the original link being returned instead of the non-link that prevented full traversal. To obtain stat results for the final path in this case, use the [os.path.realpath()](https://docs.python.org/3/library/os.path.html#os.path.realpath) function to resolve the path name as far as possible and call [lstat()](https://docs.python.org/3/library/os.html#os.lstat) on the result. This does not apply to dangling symlinks or junction points, which will raise the usual exceptions.

*class*os.**stat\_result**

Object whose attributes correspond roughly to the members of the stat structure. It is used for the result of [os.stat()](https://docs.python.org/3/library/os.html#os.stat), [os.fstat()](https://docs.python.org/3/library/os.html#os.fstat) and [os.lstat()](https://docs.python.org/3/library/os.html#os.lstat).

os.**statvfs**(*path*)

Perform a statvfs() system call on the given path. The return value is an object whose attributes describe the filesystem on the given path, and correspond to the members of the statvfs structure, namely: f\_bsize, f\_frsize, f\_blocks, f\_bfree, f\_bavail, f\_files, f\_ffree, f\_favail, f\_flag, f\_namemax, f\_fsid.

Two module-level constants are defined for the f\_flag attribute’s bit-flags: if ST\_RDONLY is set, the filesystem is mounted read-only, and if ST\_NOSUID is set, the semantics of setuid/setgid bits are disabled or not supported.

os.**supports\_dir\_fd**

A [set](https://docs.python.org/3/library/stdtypes.html#set) object indicating which functions in the [os](https://docs.python.org/3/library/os.html#module-os) module accept an open file descriptor for their *dir\_fd* parameter. Different platforms provide different features, and the underlying functionality Python uses to implement the *dir\_fd* parameter is not available on all platforms Python supports. For consistency’s sake, functions that may support *dir\_fd* always allow specifying the parameter, but will throw an exception if the functionality is used when it’s not locally available. (Specifying None for *dir\_fd* is always supported on all platforms.)

os.**supports\_effective\_ids**

A [set](https://docs.python.org/3/library/stdtypes.html#set) object indicating whether [os.access()](https://docs.python.org/3/library/os.html#os.access) permits specifying True for its *effective\_ids* parameter on the local platform. (Specifying False for *effective\_ids* is always supported on all platforms.) If the local platform supports it, the collection will contain [os.access()](https://docs.python.org/3/library/os.html#os.access); otherwise it will be empty.

os.**supports\_fd**

A [set](https://docs.python.org/3/library/stdtypes.html#set) object indicating which functions in the [os](https://docs.python.org/3/library/os.html#module-os) module permit specifying their *path* parameter as an open file descriptor on the local platform. Different platforms provide different features, and the underlying functionality Python uses to accept open file descriptors as *path* arguments is not available on all platforms Python supports.

os.**supports\_follow\_symlinks**

A [set](https://docs.python.org/3/library/stdtypes.html#set) object indicating which functions in the [os](https://docs.python.org/3/library/os.html#module-os) module accept False for their *follow\_symlinks* parameter on the local platform. Different platforms provide different features, and the underlying functionality Python uses to implement *follow\_symlinks* is not available on all platforms Python supports. For consistency’s sake, functions that may support *follow\_symlinks* always allow specifying the parameter, but will throw an exception if the functionality is used when it’s not locally available. (Specifying True for *follow\_symlinks* is always supported on all platforms.)

To check whether a particular function accepts False for its *follow\_symlinks* parameter, use the in operator on supports\_follow\_symlinks. As an example, this expression evaluates to True if you may specify follow\_symlinks=False when calling [os.stat()](https://docs.python.org/3/library/os.html#os.stat) on the local platform.

os.**symlink**(*src*, *dst*, *target\_is\_directory=False*, *\**, *dir\_fd=None*)

Create a symbolic link pointing to *src* named *dst*.

On Windows, a symlink represents either a file or a directory, and does not morph to the target dynamically. If the target is present, the type of the symlink will be created to match. Otherwise, the symlink will be created as a directory if *target\_is\_directory* is True or a file symlink (the default) otherwise. On non-Windows platforms, *target\_is\_directory* is ignored.

os.**sync**()

Force write of everything to disk.

os.**truncate**(*path*, *length*)

Truncate the file corresponding to *path*, so that it is at most *length* bytes in size.

os.**walk**(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)

Generate the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory *top* (including *top* itself), it yields a 3-tuple (dirpath, dirnames, filenames).

os.**fwalk**(*top='.'*, *topdown=True*, *onerror=None*, *\**, *follow\_symlinks=False*, *dir\_fd=None*)

This behaves exactly like [walk()](https://docs.python.org/3/library/os.html#os.walk), except that it yields a 4-tuple (dirpath, dirnames, filenames, dirfd), and it supports dir\_fd.

os.**memfd\_create**(*name*[, *flags=os.MFD\_CLOEXEC*])

Create an anonymous file and return a file descriptor that refers to it. *flags* must be one of the os.MFD\_\* constants available on the system (or a bitwise ORed combination of them). By default, the new file descriptor is [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance).

The name supplied in *name* is used as a filename and will be displayed as the target of the corresponding symbolic link in the directory /proc/self/fd/. The displayed name is always prefixed with memfd: and serves only for debugging purposes. Names do not affect the behavior of the file descriptor, and as such multiple files can have the same name without any side effects.

[Availability](https://docs.python.org/3/library/intro.html#availability): Linux 3.17 or newer with glibc 2.27 or newer.

os.**eventfd**(*initval*[, *flags=os.EFD\_CLOEXEC*])

Create and return an event file descriptor. The file descriptors supports raw [read()](https://docs.python.org/3/library/os.html#os.read) and [write()](https://docs.python.org/3/library/os.html#os.write) with a buffer size of 8, [select()](https://docs.python.org/3/library/select.html#select.select), [poll()](https://docs.python.org/3/library/select.html#select.poll) and similar. See man page [*eventfd(2)*](https://manpages.debian.org/eventfd(2)) for more information. By default, the new file descriptor is [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance). *initval* is the initial value of the event counter. The initial value must be an 32 bit unsigned integer. Please note that the initial value is limited to a 32 bit unsigned int although the event counter is an unsigned 64 bit integer with a maximum value of 264-2. *flags* can be constructed from [EFD\_CLOEXEC](https://docs.python.org/3/library/os.html#os.EFD_CLOEXEC), [EFD\_NONBLOCK](https://docs.python.org/3/library/os.html#os.EFD_NONBLOCK), and [EFD\_SEMAPHORE](https://docs.python.org/3/library/os.html#os.EFD_SEMAPHORE).

os.**eventfd\_read**(*fd*)

Read value from an [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd) file descriptor and return a 64 bit unsigned int. The function does not verify that *fd* is an [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd).

os.**eventfd\_write**(*fd*, *value*)

Add value to an [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd) file descriptor. *value* must be a 64 bit unsigned int. The function does not verify that *fd* is an [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd).

os.**EFD\_CLOEXEC**

Set close-on-exec flag for new [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd) file descriptor.

os.**EFD\_NONBLOCK**

Set [O\_NONBLOCK](https://docs.python.org/3/library/os.html#os.O_NONBLOCK) status flag for new [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd) file descriptor.

os.**EFD\_SEMAPHORE**

Provide semaphore-like semantics for reads from a [eventfd()](https://docs.python.org/3/library/os.html#os.eventfd) file descriptor. On read the internal counter is decremented by one.

os.**getxattr**(*path*, *attribute*, *\**, *follow\_symlinks=True*)

Return the value of the extended filesystem attribute *attribute* for *path*. *attribute* can be bytes or str (directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface). If it is str, it is encoded with the filesystem encoding. This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd) and [not following symlinks](https://docs.python.org/3/library/os.html#follow-symlinks).

os.**listxattr**(*path=None*, *\**, *follow\_symlinks=True*)

Return a list of the extended filesystem attributes on *path*. The attributes in the list are represented as strings decoded with the filesystem encoding. If *path* is None, [listxattr()](https://docs.python.org/3/library/os.html#os.listxattr) will examine the current directory.

os.**removexattr**(*path*, *attribute*, *\**, *follow\_symlinks=True*)

Removes the extended filesystem attribute *attribute* from *path*. *attribute* should be bytes or str (directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface). If it is a string, it is encoded with the [filesystem encoding and error handler](https://docs.python.org/3/glossary.html#term-filesystem-encoding-and-error-handler).

os.**setxattr**(*path*, *attribute*, *value*, *flags=0*, *\**, *follow\_symlinks=True*)

Set the extended filesystem attribute *attribute* on *path* to *value*. *attribute* must be a bytes or str with no embedded NULs (directly or indirectly through the [PathLike](https://docs.python.org/3/library/os.html#os.PathLike) interface). If it is a str, it is encoded with the [filesystem encoding and error handler](https://docs.python.org/3/glossary.html#term-filesystem-encoding-and-error-handler). *flags* may be [XATTR\_REPLACE](https://docs.python.org/3/library/os.html#os.XATTR_REPLACE) or [XATTR\_CREATE](https://docs.python.org/3/library/os.html#os.XATTR_CREATE). If [XATTR\_REPLACE](https://docs.python.org/3/library/os.html#os.XATTR_REPLACE) is given and the attribute does not exist, ENODATA will be raised. If [XATTR\_CREATE](https://docs.python.org/3/library/os.html#os.XATTR_CREATE) is given and the attribute already exists, the attribute will not be created and EEXISTS will be raised.

This function can support [specifying a file descriptor](https://docs.python.org/3/library/os.html#path-fd) and [not following symlinks](https://docs.python.org/3/library/os.html#follow-symlinks).

os.**XATTR\_SIZE\_MAX**

The maximum size the value of an extended attribute can be. Currently, this is 64 KiB on Linux.

os.**XATTR\_CREATE**

This is a possible value for the flags argument in [setxattr()](https://docs.python.org/3/library/os.html#os.setxattr). It indicates the operation must create an attribute.

os.**XATTR\_REPLACE**

This is a possible value for the flags argument in [setxattr()](https://docs.python.org/3/library/os.html#os.setxattr). It indicates the operation must replace an existing attribute.

os.**abort**()

Generate a SIGABRT signal to the current process. On Unix, the default behavior is to produce a core dump; on Windows, the process immediately returns an exit code of 3. Be aware that calling this function will not call the Python signal handler registered for SIGABRT with [signal.signal()](https://docs.python.org/3/library/signal.html#signal.signal).

os.**add\_dll\_directory**(*path*)

Add a path to the DLL search path.

os.**execl**(*path*, *arg0*, *arg1*, *...*)

os.**execle**(*path*, *arg0*, *arg1*, *...*, *env*)

os.**execlp**(*file*, *arg0*, *arg1*, *...*)

os.**execlpe**(*file*, *arg0*, *arg1*, *...*, *env*)

os.**execv**(*path*, *args*)

os.**execve**(*path*, *args*, *env*)

os.**execvp**(*file*, *args*)

os.**execvpe**(*file*, *args*, *env*)

These functions all execute a new program, replacing the current process; they do not return. On Unix, the new executable is loaded into the current process, and will have the same process id as the caller. Errors will be reported as [OSError](https://docs.python.org/3/library/exceptions.html#OSError) exceptions.

os.**\_exit**(*n*)

Exit the process with status *n*, without calling cleanup handlers, flushing stdio buffers, etc.

os.**kill**(*pid*, *sig*)

Send signal *sig* to the process *pid*. Constants for the specific signals available on the host platform are defined in the [signal](https://docs.python.org/3/library/signal.html#module-signal) module.

Windows: The [signal.CTRL\_C\_EVENT](https://docs.python.org/3/library/signal.html#signal.CTRL_C_EVENT) and [signal.CTRL\_BREAK\_EVENT](https://docs.python.org/3/library/signal.html#signal.CTRL_BREAK_EVENT) signals are special signals which can only be sent to console processes which share a common console window, e.g., some subprocesses. Any other value for *sig* will cause the process to be unconditionally killed by the TerminateProcess API, and the exit code will be set to *sig*. The Windows version of [kill()](https://docs.python.org/3/library/os.html#os.kill) additionally takes process handles to be killed.

os.**popen**(*cmd*, *mode='r'*, *buffering=- 1*)

Open a pipe to or from command *cmd*. The return value is an open file object connected to the pipe, which can be read or written depending on whether *mode* is 'r' (default) or 'w'. The *buffering* argument has the same meaning as the corresponding argument to the built-in [open()](https://docs.python.org/3/library/functions.html#open) function. The returned file object reads or writes text strings rather than bytes.

os.**posix\_spawn**(*path*, *argv*, *env*, *\**, *file\_actions=None*, *setpgroup=None*, *resetids=False*, *setsid=False*, *setsigmask=()*, *setsigdef=()*, *scheduler=None*)

Wraps the posix\_spawn() C library API for use from Python.

os.**spawnvpe**(*mode*, *file*, *args*, *env*)

Execute the program *path* in a new process.

os.**P\_NOWAIT**

os.**P\_NOWAITO**

Possible values for the *mode* parameter to the [spawn\*](https://docs.python.org/3/library/os.html#os.spawnl) family of functions. If either of these values is given, the spawn\*() functions will return as soon as the new process has been created, with the process id as the return value.

os.**P\_WAIT**

Possible value for the *mode* parameter to the [spawn\*](https://docs.python.org/3/library/os.html#os.spawnl) family of functions. If this is given as *mode*, the spawn\*() functions will not return until the new process has run to completion and will return the exit code of the process the run is successful, or -signal if a signal kills the process.

os.**P\_DETACH**

os.**P\_OVERLAY**

Possible values for the *mode* parameter to the [spawn\*](https://docs.python.org/3/library/os.html#os.spawnl) family of functions. These are less portable than those listed above. [P\_DETACH](https://docs.python.org/3/library/os.html#os.P_DETACH) is similar to [P\_NOWAIT](https://docs.python.org/3/library/os.html#os.P_NOWAIT), but the new process is detached from the console of the calling process. If [P\_OVERLAY](https://docs.python.org/3/library/os.html#os.P_OVERLAY) is used, the current process will be replaced; the [spawn\*](https://docs.python.org/3/library/os.html#os.spawnl) function will not return.

os.**startfile**(*path*[, *operation*][, *arguments*][, *cwd*][, *show\_cmd*])

Start a file with its associated application.

os.**system**(*command*)

Execute the command (a string) in a subshell. This is implemented by calling the Standard C function system(), and has the same limitations. Changes to [sys.stdin](https://docs.python.org/3/library/sys.html#sys.stdin), etc. are not reflected in the environment of the executed command. If *command* generates any output, it will be sent to the interpreter standard output stream. The C standard does not specify the meaning of the return value of the C function, so the return value of the Python function is system-dependent.

On Unix, the return value is the exit status of the process encoded in the format specified for [wait()](https://docs.python.org/3/library/os.html#os.wait).

os.**times**()

Returns the current global process times. The return value is an object with five attributes:

os.**wait**()

Wait for completion of a child process, and return a tuple containing its pid and exit status indication: a 16-bit number, whose low byte is the signal number that killed the process, and whose high byte is the exit status (if the signal number is zero); the high bit of the low byte is set if a core file was produced.

[waitstatus\_to\_exitcode()](https://docs.python.org/3/library/os.html#os.waitstatus_to_exitcode) can be used to convert the exit status into an exit code.

os.**waitpid**(*pid*, *options*)

The details of this function differ on Unix and Windows.

os.**cpu\_count**()

Return the number of CPUs in the system. Returns None if undetermined.

This number is not equivalent to the number of CPUs the current process can use. The number of usable CPUs can be obtained with len(os.sched\_getaffinity(0))

os.**curdir**

The constant string used by the operating system to refer to the current directory. This is '.' for Windows and POSIX. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**pardir**

The constant string used by the operating system to refer to the parent directory. This is '..' for Windows and POSIX. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**sep**

The character used by the operating system to separate pathname components. This is '/' for POSIX and '\\' for Windows. Note that knowing this is not sufficient to be able to parse or concatenate pathnames — use [os.path.split()](https://docs.python.org/3/library/os.path.html#os.path.split) and [os.path.join()](https://docs.python.org/3/library/os.path.html#os.path.join) — but it is occasionally useful. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**altsep**

An alternative character used by the operating system to separate pathname components, or None if only one separator character exists. This is set to '/' on Windows systems where sep is a backslash. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**extsep**

The character which separates the base filename from the extension; for example, the '.' in os.py. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**pathsep**

The character conventionally used by the operating system to separate search path components (as in PATH), such as ':' for POSIX or ';' for Windows. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**defpath**

The default search path used by [exec\*p\*](https://docs.python.org/3/library/os.html#os.execl) and [spawn\*p\*](https://docs.python.org/3/library/os.html#os.spawnl) if the environment doesn’t have a 'PATH' key. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**linesep**

The string used to separate (or, rather, terminate) lines on the current platform. This may be a single character, such as '\n' for POSIX, or multiple characters, for example, '\r\n' for Windows. Do not use *os.linesep* as a line terminator when writing files opened in text mode (the default); use a single '\n' instead, on all platforms.

os.**devnull**

The file path of the null device. For example: '/dev/null' for POSIX, 'nul' for Windows. Also available via [os.path](https://docs.python.org/3/library/os.path.html#module-os.path).

os.**getrandom**(*size*, *flags=0*)

Get up to *size* random bytes. The function can return less bytes than requested.

These bytes can be used to seed user-space random number generators or for cryptographic purposes.

getrandom() relies on entropy gathered from device drivers and other sources of environmental noise. Unnecessarily reading large quantities of data will have a negative impact on other users of the /dev/random and /dev/urandom devices.

The flags argument is a bit mask that can contain zero or more of the following values ORed together: [os.GRND\_RANDOM](https://docs.python.org/3/library/os.html#os.GRND_RANDOM) and [GRND\_NONBLOCK](https://docs.python.org/3/library/os.html#os.GRND_NONBLOCK).

os.**urandom**(*size*)

Return a bytestring of *size* random bytes suitable for cryptographic use.