Help on module os:

NAME os - OS routines for NT or Posix depending on what system we're on.

MODULE REFERENCE https://docs.python.org/3.13/library/os.html

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION This exports: - all functions from posix or nt, e.g. unlink, stat, etc. - os.path is either posixpath or ntpath - os.name is either 'posix' or 'nt' - os.curdir is a string representing the current directory (always '.') - os.pardir is a string representing the parent directory (always '..') - os.sep is the (or a most common) pathname separator ('/' or '
') - os.extsep is the extension separator (always '.') - os.altsep is the alternate pathname separator (None or '/') - os.pathsep is the component separator used in $PATHetc-os.linesepisthelineseparatorintextfiles(''or''or'')-os.defpathisthedefaultsearchpathfor$

Programs that import and use 'os' stand a better chance of being portable between different platforms. Of course, they must then only use functions that are defined by all platforms (e.g., unlink and opendir), and leave all pathname manipulation to os.path (e.g., split and join).

CLASSES builtins.Exception(builtins.BaseException) builtins.OSError builtins.object posix.DirEntry builtins.tuple(builtins.object) stat$_resultstatvfs_resultterminal_sizeposix.times_resultposix.$

class DirEntry(builtins.object) — Methods defined here: — — $_fspath_{(self,/)|Returnsthepathfortheentry.||_repr_{(self}}$

error = class OSError(Exception) — Base class for I/O related errors. — — Method resolution order: — OSError — Exception — BaseException — object — — Built-in subclasses: — BlockingIOError — ChildProcessError — ConnectionError — FileExistsError — ... and 7 other subclasses — — Methods defined here: — — $_init_{(self,/,*args,**kwargs)|Initializeself.Seehelp(type(self))foraccuratesignature.||_reduce_{(self,/)|Helperforpickl}}$

class stat$_result(builtins.tuple)|stat_result(iterable=(),/)||stat_result:Resultfromstat,fstat,orlstat.|$ $Ifyourplatformsupportsst_blksize,st_blocks,st_rdev,|orst_flags,theyareavailableasattributesonly.||Seeos.s$ $|stat_result|builtins.tuple|builtins.object||Methodsdefinedhere:||_reduce_{(self,/)|Helperforpickle.||_replace_{(self,/,**ch}}$

class statvfs$_result(builtins.tuple)|statvfs_result(iterable=(),/)||statvfs_result:$ $Resultfromstatvfsorfstatvfs.||Thisobjectmaybeaccessedeitherasatupleof|(bsize,frsize,blocks,bfree,b$

$|statvfs_result|builtins.tuple|builtins.object||Methodsdefinedhere:||_{reduce_{(self,/)|Helperforpickle.||}}_{replace_{(self,/,}}$

class terminal$_size(builtins.tuple)|terminal_size(iterable=(),/)||Atupleof(columns,lines)forholdingte$
$|terminal_size|builtins.tuple|builtins.object||Methodsdefinedhere:||_{reduce_{(self,/)|Helperforpickle.||}}_{replace_{(self,/,*}}$

class times$_result(builtins.tuple)|times_result(iterable=(),/)||times_result:$
$Resultfromos.times().||Thisobjectmaybeaccessedeitherasatupleof|(user,system,children_user,children_s$
$|times_result|builtins.tuple|builtins.object||Methodsdefinedhere:||_{reduce_{(self,/)|Helperforpickle.||}}_{replace_{(self,/,**}}$

class uname$_result(builtins.tuple)|uname_result(iterable=(),/)||uname_result:$
$Resultfromos.uname().||Thisobjectmaybeaccessedeitherasatupleof|(sysname,nodename,release,versio$
$|uname_result|builtins.tuple|builtins.object||Methodsdefinedhere:||_{reduce_{(self,/)|Helperforpickle.||}}_{replace_{(self,/,*}}$

class waitid$_result(builtins.tuple)|waitid_result(iterable=(),/)||waitid_result:$
$Resultfromwaitid.||Thisobjectmaybeaccessedeitherasatupleof|(si_pid,si_uid,si_signo,si_status,si_code),|or$
$|waitid_result|builtins.tuple|builtins.object||Methodsdefinedhere:||_{reduce_{(self,/)|Helperforpickle.||}}_{replace_{(self,/,*}}$

FUNCTIONS WCOREDUMP(status, /) Return True if the process returning status was dumped to a core file.

WEXITSTATUS(status) Return the process return code from status.

WIFCONTINUED(status) Return True if a particular process was continued from a job control stop.

Return True if the process returning status was continued from a job control stop.

WIFEXITED(status) Return True if the process returning status exited via the exit() system call.

WIFSIGNALED(status) Return True if the process returning status was terminated by a signal.

WIFSTOPPED(status) Return True if the process returning status was stopped.

WSTOPSIG(status) Return the signal that stopped the process that provided the status value.

WTERMSIG(status) Return the signal that terminated the process that provided the status value.

$_{e}xit(status)Exittothesystemwithspecifiedstatus, withoutnormalexitprocessing.$

abort() Abort the interpreter immediately.

This function 'dumps core' or otherwise fails in the hardest way possible on the hosting operating system. This function never returns.

access(path, mode, *, dir$_{f}d = None, effective_{i}ds = False, follow_{s}ymlinks = True)Usetherealuid/gidtotestforaccesstoapath.$

path Path to be tested; can be string, bytes, or a path-like object. mode Operating-system mode bitfield. Can be F$_{O}Ktotestexistence, ortheinclusive-OR of R_{O}K, W_{O}K, andX_{O}K.dir_{f}dIfnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshould$

dir$_{f}d, effective_{i}ds, andfollow_{s}ymlinksmaynotbeimplementedonyourplatform.Iftheyareunavailable,$

Note that most operations will use the effective uid/gid, therefore this routine can be used in a suid/sgid environment to test if the invoking user has the specified access to the path.

chdir(path) Change the current working directory to the specified path.

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

chflags(path, flags, follow$_{s}ymlinks = True)Setfileflags.$

If follow$_{s}ymlinksisFalse, andthelastelementofthepathisasymboliclink, chflagswillchangeflagsonthe$

chmod(path, mode, *, dir$_{f}d = None, follow_{s}ymlinks = (os.name! =' nt'))Changetheaccesspermissionsofafile.$

path Path to be modified. May always be specified as a str, bytes, or a path-like object. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception. mode Operating-system mode bitfield. Be careful when using number literals for *mode*. The conventional UNIX notation for numeric modes uses an octal base, which needs to be indicated with a “0o“ prefix in Python. dir$_{f}dIfnotNone, itshouldbeafiledescriptoropent$

It is an error to use dir$_{f}dorfollow_{s}ymlinkswhenspecifyingpathasanopenfiledescriptor.dir_{f}dandfollo$

chown(path, uid, gid, *, dir$_{f}d = None, follow_{s}ymlinks = True)Changetheownerandgroupidofpathtot$

path Path to be examined; can be string, bytes, a path-like object, or open-file-descriptor int. dir$_{f}dIfnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative;$

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception. If dir$_{f}disnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelati$

chroot(path) Change root directory to path.

close(fd) Close a file descriptor.

closerange(fd$_{l}ow, fd_{h}igh, /)Closesallfiledescriptorsin[fd_{l}ow, fd_{h}igh), ignoringerrors.$

confstr(name, /) Return a string-valued system configuration variable.

cpu$_{c}ount()ReturnthenumberoflogicalCPUsinthesystem.$

Return None if indeterminable.

ctermid() Return the name of the controlling terminal for this process.

device$_{e}ncoding(fd)Returnastringdescribingtheencodingofaterminal'sfiledescriptor.$

The file descriptor must be attached to a terminal. If the device is not a terminal, return None.

dup(fd, /) Return a duplicate of a file descriptor.

dup2(fd, fd2, inheritable=True) Duplicate file descriptor.

execl(file, *args) execl(file, *args)

Execute the executable file with argument list args, replacing the current process.

execle(file, *args) execle(file, *args, env)

Execute the executable file with argument list args and environment env, replacing the current process.

execlp(file, *args) execlp(file, *args)

Execute the executable file (which is searched for along $PATH)withargumentlistargs, replacingthecurr$

execlpe(file, *args) execlpe(file, *args, env)

Execute the executable file (which is searched for along $PATH)withargumentlistargsandenvironmente$

execv(path, argv, /) Execute an executable path with arguments, replacing current process.

path Path of executable file. argv Tuple or list of strings.

execve(path, argv, env) Execute an executable path with arguments, replacing current process.

path Path of executable file. argv Tuple or list of strings. env Dictionary of strings mapping to strings.

execvp(file, args) execvp(file, args)

Execute the executable file (which is searched for along $PATH)withargumentlistargs, replacingthecurr$

execvpe(file, args, env) execvpe(file, args, env)

Execute the executable file (which is searched for along $PATH)withargumentlistargsandenvironmente$

fchdir(fd) Change to the directory of the given file descriptor.

fd must be opened on a directory, not a file. Equivalent to os.chdir(fd).

fchmod(fd, mode) Change the access permissions of the file given by file descriptor fd.

fd The file descriptor of the file to be modified. mode Operating-system mode bitfield. Be careful when using number literals for *mode*. The conventional UNIX notation for numeric modes uses an octal base, which needs to be indicated with a "0o" prefix in Python.

Equivalent to os.chmod(fd, mode).

fchown(fd, uid, gid) Change the owner and group id of the file specified by file descriptor.

Equivalent to os.chown(fd, uid, gid).

fdopen(fd, mode='r', buffering=-1, encoding=None, *args, **kwargs)

fork() Fork a child process.

Return 0 to child process and PID of child to parent process.

forkpty() Fork a new process with a new pseudo-terminal as controlling tty.

Returns a tuple of (pid, master$_{f}d).Likefork(), returnpidof0tothechildprocess, andpidofchildtothepare$ $terminal.$

fpathconf(fd, name, /) Return the configuration limit name for the file descriptor fd.

If there is no limit, return -1.

fsdecode(filename) Decode filename (an os.PathLike, bytes, or str) from the filesystem encoding with 'surrogateescape' error handler, return str unchanged. On Windows, use 'strict' error handler if the file system encoding is 'mbcs' (which is the default encoding).

fsencode(filename) Encode filename (an os.PathLike, bytes, or str) to the filesystem encoding with 'surrogateescape' error handler, return bytes unchanged. On Windows, use 'strict' error handler if the file system encoding is 'mbcs' (which is the default encoding).

fspath(path) Return the file system path representation of the object.

If the object is str or bytes, then allow it to pass through as-is. If the object defines $_{fspath_{(),thenreturntheresultofthatmethod.AllothertypesraiseaTypeError.}}$

fstat(fd) Perform a stat system call on the given file descriptor.

Like stat(), but for an open file descriptor. Equivalent to os.stat(fd).

fstatvfs(fd, /) Perform an fstatvfs system call on the given fd.

Equivalent to statvfs(fd).

fsync(fd) Force write of fd to disk.

ftruncate(fd, length, /) Truncate a file, specified by file descriptor, to a specific length.

fwalk(top='.', topdown=True, onerror=None, *, follow$_{s}ymlinks = False, dir_{f}d = None)Directorytreegenerator.$

This behaves exactly like walk(), except that it yields a 4-tuple

dirpath, dirnames, filenames, dirfd

'dirpath', 'dirnames' and 'filenames' are identical to walk() output, and 'dirfd' is a file descriptor referring to the directory 'dirpath'.

The advantage of fwalk() over walk() is that it's safe against symlink races (when follow$_{s}ymlinksisFalse$).

If dir$_{f}disnotNone, itshouldbeafiledescriptoropentoadirectory, andtopshouldberelative; topwillthenber$

Caution: Since fwalk() yields file descriptors, those are only valid until the next iteration step, so you should dup() them if you want to keep them for a longer period.

Example:

import os for root, dirs, files, rootfd in os.fwalk('python/Lib/email'): print(root, "consumes", end="") print(sum(os.stat(name, dir$_{f}d = rootfd).st_{s}izefornameinfiles), end =$

""$)print("bytesin", len(files), "non-directoryfiles")if'CVS'indirs : dirs.remove('CVS')don'tvisitCVS$

get$_blocking(fd, /)Gettheblockingmodeofthefiledescriptor.$

Return False if the O$_NONBLOCKflagisset, Trueiftheflagiscleared.$

get$_execpath(env = None)Returnsthesequenceofdirectoriesthatwillbesearchedforthenamedexecutable$

env must be an environment variable dict or None. If *env* is None, os.environ will be used.

get$_inheritable(fd, /)Gettheclose-on-exeflagofthespecifiedfiledescriptor.$

get$_terminal_size(fd =< unrepresentable >, /)Returnthesizeoftheterminalwindowas(columns, lines)$

The optional argument fd (default standard output) specifies which file descriptor should be queried.

If the file descriptor is not connected to a terminal, an OSError is thrown.

This function will only be defined if an implementation is available for this system.

shutil.get$_terminal_sizeisthehigh-levelfunctionwhichshouldnormallybeused, os.get_terminal_sizeisthel$ $levelimplementation.$

getcwd() Return a unicode string representing the current working directory.

getcwdb() Return a bytes string representing the current working directory.

getegid() Return the current process's effective group id.

getenv(key, default=None) Get an environment variable, return None if it doesn't exist. The optional second argument can specify an alternate default. key, default and the result are str.

getenvb(key, default=None) Get an environment variable, return None if it doesn't exist. The optional second argument can specify an alternate default. key, default and the result are bytes.

geteuid() Return the current process's effective user id.

getgid() Return the current process's group id.

getgrouplist(user, group, /) Returns a list of groups to which a user belongs.

user username to lookup group base group id of the user

getgroups() Return list of supplemental group IDs for the process.

getloadavg() Return average recent system load information.

Return the number of processes in the system run queue averaged over the last 1, 5, and 15 minutes as a tuple of three floats. Raises OSError if the load average was unobtainable.

getlogin() Return the actual login name.

getpgid(pid) Call the system call getpgid(), and return the result.

getpgrp() Return the current process group id.

getpid() Return the current process id.

getppid() Return the parent's process id.

If the parent process has already exited, Windows machines will still return its id; others systems will return the id of the 'init' process (1).

getpriority(which, who) Return program scheduling priority.

getsid(pid, /) Call the system call getsid(pid) and return the result.

getuid() Return the current process's user id.

grantpt(fd, /) Grant access to the slave pseudo-terminal device.

fd File descriptor of a master pseudo-terminal device.

Performs a grantpt() C function call.

initgroups(username, gid, /) Initialize the group access list.

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.

isatty(fd, /) Return True if the fd is connected to a terminal.

Return True if the file descriptor is an open file descriptor connected to the slave end of a terminal.

kill(pid, signal, /) Kill a process with a signal.

killpg(pgid, signal, /) Kill a process group with a signal.

lchflags(path, flags) Set file flags.

This function will not follow symbolic links. Equivalent to chflags(path, flags, follow$_symlinks = False$).

lchmod(path, mode) Change the access permissions of a file, without following symbolic links.

If path is a symlink, this affects the link itself rather than the target. Equivalent to chmod(path, mode, follow$_symlinks = False$)."

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. Equivalent to os.chown(path, uid, gid, follow$_symlinks = False$).

link(src, dst, *, src$_dir_fd = None, dst_dir_fd = None, follow_symlinks = True)Createahardlinktoafile.$

If either src$_dir_fdordst_dir_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andtherespectivepa$

listdir(path=None) Return a list containing the names of the files in the directory.

path can be specified as either str, bytes, or a path-like object. If path is bytes, the filenames returned will also be bytes; in all other circumstances the filenames returned will be str. If path is None, uses the path='.'. On some platforms, path may also be specified as an open file descriptor; the file descriptor must refer to a directory. If this functionality is unavailable, using it raises NotImplementedError.

The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

lockf(fd, command, length, /) Apply, test or remove a POSIX lock on an open file descriptor.

fd An open file descriptor. command One of F$_LOCK, F_TLOCK, F_ULOCKorF_TEST.lengthThenumber$

login$_tty(fd, /)Preparethettyofwhichfdisafiledescriptorforanewloginsession.$

Make the calling process a session leader; make the tty the controlling tty, the stdin, the stdout, and the stderr of the calling process; close fd.

lseek(fd, position, whence, /) Set the position of a file descriptor. Return the new position.

fd An open file descriptor, as returned by os.open(). position Position, interpreted relative to 'whence'. whence The relative position to seek from. Valid values are: - SEEK$_SET : seekfromthestartofthefile. - SEEK_CUR : seekfromthecurrentfileposition. - SEEK_END : seekfromtheendofthefile.$

The return value is the number of bytes relative to the beginning of the file.

lstat(path, *, dir$_fd = None)Performastatsystemcallonthegivenpath,withoutfollowingsymboliclinks$

Like stat(), but do not follow symbolic links. Equivalent to stat(path, follow$_symlinks = False$).

major(device, /) Extracts a device major number from a raw device number.

makedev(major, minor, /) Composes a raw device number from the major and minor device numbers.

makedirs(name, mode=511, exist$_ok = False)makedirs(name[,mode = 0o777][,exist_ok = False]$)

Super-mkdir; create a leaf directory and all intermediate ones. Works like mkdir, except that any intermediate path segment (not just the rightmost) will be created if it does not exist. If the target directory already exists, raise an OSError if exist$_okisFalse.Otherwisenoexceptionisraised.Thisisrecursive.$

minor(device, /) Extracts a device minor number from a raw device number.

mkdir(path, mode=511, *, dir$_fd = None)Createadirectory.$

If dir$_fdisnotNone,itshouldbeafiledescriptoropentoadirectory,andpathshouldberelative;pathwillthen$

The mode argument is ignored on Windows. Where it is used, the current umask value is first masked out.

mkfifo(path, mode=438, *, dir$_fd = None)Createa"fifo"(aPOSIXnamedpipe).$

If dir$_fdisnotNone,itshouldbeafiledescriptoropentoadirectory,andpathshouldberelative;pathwillthen$

mknod(path, mode=384, device=0, *, dir$_fd = None)Createanodeinthefilesystem.$

Create a node in the file system (file, device special file or named pipe) at path. mode specifies both the permissions to use and the type of node to be created, being combined (bitwise OR) with one of S$_IFREG,S_IFCHR,S_IFBLK,andS_IFIFO.IfS_IFCH$

If dir$_fdisnotNone,itshouldbeafiledescriptoropentoadirectory,andpathshouldberelative;pathwillthen$

nice(increment, /) Add increment to the priority of process and return the new priority.

open(path, flags, mode=511, *, dir$_fd = None)OpenafileforlowlevelIO.Returnsafiledescriptor(integ$

If dir$_fdisnotNone,itshouldbeafiledescriptoropentoadirectory,andpathshouldberelative;pathwillthen$

openpty() Open a pseudo-terminal.

Return a tuple of (master$_fd,slave_fd)containingopenfiledescriptorsforboththemasterandslaveends.$

pathconf(path, name) Return the configuration limit name for the file or directory path.

If there is no limit, return -1. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

pipe() Create a pipe.

Returns a tuple of two file descriptors: (read$_fd,write_fd$)

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

posix$_openpt(oflag,/)Openandreturnafiledescriptorforamasterpseudo-terminaldevice.$

Performs a posix$_openpt()Cfunctioncall.Theoflagargumentisusedtosetfilestatusflagsandfileaccessm$

posix$_spawn(path,argv,env,/,*,file_actions = (),setpgroup =< unrepresentable > ,resetids = False,setsid = False,setsigmask = (),setsigdef = (),scheduler =< unrepresentable >)Executetheprogramspecifiedbypathinanewprocess.$

path Path of executable file. argv Tuple or list of strings. env Dictionary of strings mapping to strings. file$_actionsAsequenceoffileactiontuples.setpgroupThepgrouptousewiththePOS$

posix$_{s}pawnp(path, argv, env, /, *, file_{a}ctions = (), setpgroup =< unrepresentable >, resetids = False, setsid = False, setsigmask = (), setsigdef = (), scheduler =< unrepresentable >)Executetheprogramspecifiedbypathinanewprocess.$

path Path of executable file. argv Tuple or list of strings. env Dictionary of strings mapping to strings. file$_{a}ctionsAsequenceoffileactiontuples.setpgroupThepgrouptousewiththePOS$

pread(fd, length, offset, /) Read a number of bytes from a file descriptor starting at a particular offset.

Read length bytes from file descriptor fd, starting at offset bytes from the beginning of the file. The file offset remains unchanged.

preadv(fd, buffers, offset, flags=0, /) Reads from a file descriptor into a number of mutable bytes-like objects.

Combines the functionality of readv() and pread(). As readv(), it will 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. Its fourth argument, specifies the file offset at which the input operation is to be performed. It will return the total number of bytes read (which can be less than the total capacity of all the objects).

The flags argument contains a bitwise OR of zero or more of the following flags:

- RWF$_{H}IPRI - RWF_{N}OWAIT$

Using non-zero flags requires Linux 4.6 or newer.

ptsname(fd, /) Return the name of the slave pseudo-terminal device.

fd File descriptor of a master pseudo-terminal device.

If the ptsname$_{r}()Cfunctionisavailable, itiscalled; otherwise, performsaptsname()Cfunctioncall.$

putenv(name, value, /) Change or add an environment variable.

pwrite(fd, buffer, offset, /) Write bytes to a file descriptor starting at a particular offset.

Write buffer to fd, starting at offset bytes from the beginning of the file. Returns the number of bytes written. Does not change the current file offset.

pwritev(fd, buffers, offset, flags=0, /) Writes the contents of bytes-like objects to a file descriptor at a given offset.

Combines the functionality of writev() and pwrite(). All buffers must be a sequence of bytes-like objects. Buffers are processed in array order. Entire contents of first buffer is written before proceeding to second, and so on. The operating system may set a limit (sysconf() value SC$_{I}OV_{M}AX)onthenumberofbuffersthatcanbeused.This$

The flags argument contains a bitwise OR of zero or more of the following flags:

- RWF$_{D}SYNC - RWF_{S}YNC - RWF_{A}PPEND$

Using non-zero flags requires Linux 4.7 or newer.

read(fd, length, /) Read from a file descriptor. Returns a bytes object.

readlink(path, *, dir$_{f}d = None)Returnastringrepresentingthepathtowhichthesymboliclinkpoints.$

If dir$_{f}disnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

dir$_{f}dmaynotbeimplementedonyourplatform.Ifitisunavailable, usingitwillraiseaNotImplementedEr$

readv(fd, buffers, /) Read from a file descriptor fd into an iterable of buffers.

The buffers should be mutable buffers accepting bytes. readv will 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.

readv returns the total number of bytes read, which may be less than the total capacity of all the buffers.

register$_at_fork(*, before =< unrepresentable >, after_in_child =< unrepresentable >$
$, after_in_parent =< unrepresentable >)Registercallablestobecalledwhenforkinganewprocess.$

before A callable to be called in the parent before the fork() syscall. after$_in_childAcallabletobecalledinthe$

'before' callbacks are called in reverse order. 'after$_in_child'and'after_in_parent'callbacksarecalledinorder.$

remove(path, *, dir$_fd = None)Removeafile(sameasunlink()).$

If dir$_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

removedirs(name) removedirs(name)

Super-rmdir; remove a leaf directory and all empty intermediate ones. Works like rmdir except that, if the leaf directory is successfully removed, directories corresponding to rightmost path segments will be pruned away until either the whole path is consumed or an error occurs. Errors during this latter phase are ignored – they generally mean that a directory was not empty.

rename(src, dst, *, src$_dir_fd = None, dst_dir_fd = None)Renameafileordirectory.$

If either src$_dir_fdordst_dir_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andtherespectivepa$

renames(old, new) renames(old, new)

Super-rename; create directories as necessary and delete any left empty. Works like 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 until either the whole path is consumed or a nonempty directory is found.

Note: this function can fail with the new directory structure made if you lack permissions needed to unlink the leaf directory or file.

replace(src, dst, *, src$_dir_fd = None, dst_dir_fd = None)Renameafileordirectory, overwritingthedestina$

If either src$_dir_fdordst_dir_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andtherespectivepa$

rmdir(path, *, dir$_fd = None)Removeadirectory.$

If dir$_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

scandir(path=None) Return an iterator of DirEntry objects for given path.

path can be specified as either str, bytes, or a path-like object. If path is bytes, the names of yielded DirEntry objects will also be bytes; in all other circumstances they will be str.

If path is None, uses the path='.'.

sched$_get_priority_max(policy)Getthemaximumschedulingpriorityforpolicy.$

sched$_get_priority_min(policy)Gettheminimumschedulingpriorityforpolicy.$

sched$_yield()VoluntarilyrelinquishtheCPU.$

sendfile(out$_fd, in_fd, offset, count, headers = (), trailers = (), flags = 0)Copycountbytesfromfiledesc$

set$_blocking(fd, blocking, /)Settheblockingmodeofthespecifiedfiledescriptor.$

Set the O$_NONBLOCKflagifblockingisFalse, cleartheO_NONBLOCKflagotherwise.$

set$_inheritable(fd, inheritable, /)Settheinheritableflagofthespecifiedfiledescriptor.$

setegid(egid, /) Set the current process's effective group id.

seteuid(euid, /) Set the current process's effective user id.

setgid(gid, /) Set the current process's group id.

setgroups(groups, /) Set the groups of the current process to list.

setpgid(pid, pgrp, /) Call the system call setpgid(pid, pgrp).

setpgrp() Make the current process the leader of its process group.

setpriority(which, who, priority) Set program scheduling priority.

setregid(rgid, egid, /) Set the current process's real and effective group ids.

setreuid(ruid, euid, /) Set the current process's real and effective user ids.

setsid() Call the system call setsid().

setuid(uid, /) Set the current process's user id.

spawnl(mode, file, *args) spawnl(mode, file, *args) -¿ integer

Execute file with arguments from args in a subprocess. If mode == P$_NOWAITreturnthepidoftheproces$ $P_WAITreturntheprocess'sexitcodeifitexitsnormally; otherwisereturn-SIG, whereSIGisthesignalthatk$

spawnle(mode, file, *args) spawnle(mode, file, *args, env) -¿ integer

Execute file with arguments from args in a subprocess with the supplied environment. If mode == P$_NOWAITreturnthepidoftheprocess.Ifmode ==$ $P_WAITreturntheprocess'sexitcodeifitexitsnormally; otherwisereturn-SIG, whereSIGisthesignalthatk$

spawnlp(mode, file, *args) spawnlp(mode, file, *args) -¿ integer

Execute file (which is looked for along $PATH)withargumentsfromargsinasubprocesswiththesuppliede$ $P_NOWAITreturnthepidoftheprocess.Ifmode == P_WAITreturntheprocess'sexitcodeifitexitsnormally$ $SIG, whereSIGisthesignalthatkilledit.$

spawnlpe(mode, file, *args) spawnlpe(mode, file, *args, env) -¿ integer

Execute file (which is looked for along $PATH)withargumentsfromargsinasubprocesswiththesuppliede$ $P_NOWAITreturnthepidoftheprocess.Ifmode == P_WAITreturntheprocess'sexitcodeifitexitsnormally$ $SIG, whereSIGisthesignalthatkilledit.$

spawnv(mode, file, args) spawnv(mode, file, args) -¿ integer

Execute file with arguments from args in a subprocess. If mode == P$_NOWAITreturnthepidoftheproces$ $P_WAITreturntheprocess'sexitcodeifitexitsnormally; otherwisereturn-SIG, whereSIGisthesignalthatk$

spawnve(mode, file, args, env) spawnve(mode, file, args, env) -¿ integer

Execute file with arguments from args in a subprocess with the specified environment. If mode == P$_NOWAITreturnthepidoftheprocess.Ifmode ==$ $P_WAITreturntheprocess'sexitcodeifitexitsnormally; otherwisereturn-SIG, whereSIGisthesignalthatk$

spawnvp(mode, file, args) spawnvp(mode, file, args) -¿ integer

Execute file (which is looked for along $PATH)withargumentsfromargsinasubprocess.Ifmode ==$ $P_NOWAITreturnthepidoftheprocess.Ifmode == P_WAITreturntheprocess'sexitcodeifitexitsnormally$ $SIG, whereSIGisthesignalthatkilledit.$

spawnvpe(mode, file, args, env) spawnvpe(mode, file, args, env) -¿ integer

Execute file (which is looked for along $PATH)withargumentsfromargsinasubprocesswiththesuppliede$ $P_NOWAITreturnthepidoftheprocess.Ifmode == P_WAITreturntheprocess'sexitcodeifitexitsnormally$ $SIG, whereSIGisthesignalthatkilledit.$

stat(path, *, dir$_fd = None, follow_symlinks = True)Performastatsystemcallonthegivenpath.$

path Path to be examined; can be string, bytes, a path-like object or open-file-descriptor int. dir$_fdIfnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldbearelative$

dir$_fdandfollow_symlinksmaynotbeimplementedonyourplatform.Iftheyareunavailable, usingthemwil$

It's an error to use dir$_fdorfollow_symlinkswhenspecifyingpathasanopenfiledescriptor.$

statvfs(path) Perform a statvfs system call on the given path.

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

strerror(code, /) Translate an error code to a message string.

symlink(src, dst, target$_is_directory = False, *, dir_fd = None)Createasymboliclinkpointingtosrcnamed$

target$_is_directoryisrequiredonWindowsifthetargetistobeinterpretedasadirectory.(OnWindows, symli$
$Windowsplatforms.$

If dir$_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

sync() Force write of everything to disk.

sysconf(name, /) Return an integer-valued system configuration variable.

system(command) Execute the command in a subshell.

tcgetpgrp(fd, /) Return the process group associated with the terminal specified by fd.

tcsetpgrp(fd, pgid, /) Set the process group associated with the terminal specified by fd.

times() Return a collection containing process timing information.

The object returned behaves like a named tuple with these fields: (utime, stime, cutime, cstime, elapsed$_time)Allfieldsarefloating-pointnumbers.$

truncate(path, length) Truncate a file, specified by path, to a specific length.

On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

ttyname(fd, /) Return the name of the terminal device connected to 'fd'.

fd Integer file descriptor handle.

umask(mask, /) Set the current numeric umask and return the previous umask.

uname() Return an object identifying the current operating system.

The object behaves like a named tuple with the following fields: (sysname, nodename, release, version, machine)

unlink(path, *, dir$_fd = None)Removeafile(sameasremove()).$

If dir$_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

unlockpt(fd, /) Unlock a pseudo-terminal master/slave pair.

fd File descriptor of a master pseudo-terminal device.

Performs an unlockpt() C function call.

unsetenv(name, /) Delete an environment variable.

urandom(size, /) Return a bytes object containing random bytes suitable for cryptographic use.

utime(path, times=None, *, ns=¡unrepresentable¿, dir$_fd = None, follow_symlinks =$
$True)Settheaccessandmodifiedtimeofpath.$

path may always be specified as a string. On some platforms, path may also be specified as an open file descriptor. If this functionality is unavailable, using it raises an exception.

If times is not None, it must be a tuple (atime, mtime); atime and mtime should be expressed as float seconds since the epoch. If ns is specified, it must be a tuple (atime$_ns, mtime_ns); atime_nsandmtime_nsshouldbeexpressedasintegernanosecondssincetheepoch.$

If dir$_fdisnotNone, itshouldbeafiledescriptoropentoadirectory, andpathshouldberelative; pathwillthen$

wait() Wait for completion of a child process.

Returns a tuple of information about the child process: (pid, status)

wait3(options) Wait for completion of a child process.

Returns a tuple of information about the child process: (pid, status, rusage)

wait4(pid, options) Wait for completion of a specific child process.

Returns a tuple of information about the child process: (pid, status, rusage)

waitid(idtype, id, options, /) Returns the result of waiting for a process or processes.

idtype Must be one of be P$_{P}ID, P_{P}GIDorP_{A}LL.idTheidtowaiton.optionsConstructedfromtheORingo$

Returns either waitid$_{r}esultorNoneifWNOHANGisspecifiedandtherearenochildreninawaitablestate$

waitpid(pid, options, /) Wait for completion of a given child process.

Returns a tuple of information regarding the child process: (pid, status)

The options argument is ignored on Windows.

waitstatus$_{t}o_{e}xitcode(status)Convertawaitstatustoanexitcode.$

On Unix:

* If WIFEXITED(status) is true, return WEXITSTATUS(status). * If WIFSIGNALED(status) is true, return -WTERMSIG(status). * Otherwise, raise a ValueError.

On Windows, return status shifted right by 8 bits.

On Unix, if the process is being traced or if waitpid() was called with WUNTRACED option, the caller must first check if WIFSTOPPED(status) is true. This function must not be called if WIFSTOPPED(status) is true.

walk(top, topdown=True, onerror=None, followlinks=False) Directory tree generator.

For each directory in the directory tree rooted at top (including top itself, but excluding '.' and '..'), yields a 3-tuple

dirpath, dirnames, filenames

dirpath is a string, the path to the directory. dirnames is a list of the names of the subdirectories in dirpath (including symlinks to directories, and excluding '.' and '..'). filenames is a list of the names of the non-directory files in dirpath. Note that the names in the lists are just names, with no path components. To get a full path (which begins with top) to a file or directory in dirpath, do os.path.join(dirpath, name).

If optional arg 'topdown' is true or not specified, the triple for a directory is generated before the triples for any of its subdirectories (directories are generated top down). If topdown is false, the triple for a directory is generated after the triples for all of its subdirectories (directories are generated bottom up).

When topdown is true, the caller can modify the dirnames list in-place (e.g., via del or slice assignment), and walk will only recurse into the subdirectories whose names remain in dirnames; this can be used to prune the search, or to impose a specific order of visiting. Modifying dirnames when topdown is false has no effect on the behavior of os.walk(), since the directories in dirnames have already been generated by the time dirnames itself is generated. No matter the value of topdown, the list of subdirectories is retrieved before the tuples for the directory and its subdirectories are generated.

By default errors from the os.scandir() call are ignored. If optional arg 'onerror' is specified, it should be a function; it will be called with one argument, an OSError instance. It can report the error to continue with the walk, or raise the exception to abort the walk. Note that the filename is available as the filename attribute of the exception object.

By default, os.walk does not follow symbolic links to subdirectories on systems that support them. In order to get this functionality, set the optional argument 'followlinks' to true.

Caution: if you pass a relative pathname for top, don't change the current working directory between resumptions of walk. walk never changes the current directory, and assumes that the client doesn't either.

Example:

import os from os.path import join, getsize for root, dirs, files in os.walk('python/Lib/email'): print(root, "consumes ") print(sum(getsize(join(root, name)) for name in files), end=" ") print("bytes in", len(files), "non-directory files") if 'CVS' in dirs: dirs.remove('CVS') don't visit CVS directories

write(fd, data, /) Write a bytes object to a file descriptor.

writev(fd, buffers, /) Iterate over buffers, and write the contents of each to a file descriptor.

Returns the total number of bytes written. buffers must be a sequence of bytes-like objects.

DATA CLD$_{C}ONTINUED = 6CLD_{D}UMPED = 3CLD_{E}XITED = 1CLD_{K}ILLED = 2CLD_{S}TOPPED = 5CLD_{T}RAPPED = 4EX_{C}ANTCREAT = 73EX_{C}ONFIG = 78EX_{D}ATAERR = 65EX_{I}OERR = 74EX_{N}OHOST = 68EX_{N}OINPUT = 66EX_{N}OPERM = 77EX_{N}OUSER = 67EX_{O}K = 0EX_{O}SERR = 71EX_{O}SFILE = 72EX_{P}ROTOCOL = 76EX_{S}OFTWARE = 70EX_{T}EMPFAIL = 75EX_{U}NAVAILABLE = 69EX_{U}SAGE = 64F_{L}OCK = 1F_{O}K = 0F_{T}EST = 3F_{T}LOCK = 2F_{U}LOCK = 0NGROUPS_{M}AX = 16O_{A}CCMODE = 3O_{A}PPEND = 8O_{A}SYNC = 64O_{C}LOEXEC = 16777216O_{C}REAT = 512O_{D}IRECTORY = 1048576O_{D}SYNC = 4194304O_{E}VTONLY = 32768O_{E}XCL = 2048O_{E}XEC = 1073741824O_{E}XLOCK = 32O_{F}SYNC = 128O_{N}DELAY = 4O_{N}OCTTY = 131072O_{N}OFOLLOW = 256O_{N}OFOLLOW_{A}NY = 536870912O_{N}ONBLOCK = 4O_{R}DONLY = 0O_{R}DWR = 2O_{S}EARCH = 1074790400O_{S}HLOCK = 16O_{S}YMLINK = 2097152O_{S}YNC = 128O_{T}RUNC = 1024O_{W}RONLY = 1POSIX_{S}PAWN_{C}LOSE = 1POSIX_{S}PAWN_{D}UP2 = 2POSIX_{S}PAWN_{O}PEN = 0PRIO_{D}ARWIN_{B}G = 4096PRIO_{D}ARWIN_{N}ONUI = 4097PRIO_{D}ARWIN_{P}ROCESS = 4PRIO_{D}ARWIN_{T}HREAD = 3PRIO_{P}GRP = 1PRIO_{P}ROCESS = 0PRIO_{U}SER = 2P_{A}LL = 0P_{N}OWAIT = 1P_{N}OWAITO = 1P_{P}GID = 2P_{P}ID = 1P_{W}AIT = 0RTLD_{G}LOBAL = 8RTLD_{L}AZY = 1RTLD_{L}OCAL = 4RTLD_{N}ODELETE = 128RTLD_{N}OLOAD = 16RTLD_{N}OW = 2R_{O}K = 4SCHED_{F}IFO = 4SCHED_{O}THER = 1SCHED_{R}R = 2SEEK_{C}UR = 1SEEK_{D}ATA = 4SEEK_{E}ND = 2SEEK_{H}OLE = 3SEEK_{S}ET = 0ST_{N}OSUID = 2ST_{R}DONLY = 1TMP_{M}AX = 308915776WCONTINUED = 16WEXITED = 4WNOHANG = 1WNOWAIT = 32WSTOPPED = 8WUNTRACED = 2W_{O}K = 2X_{O}K = 1_{all_{=['altsep','curdir','pardir','sep','pathsep','linesep',...altsep=Noneconfstr_{n}ames='CS_{P}ATH':1,'CS_{X}BS5_{I}L}}$

FILE /Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/os.py