dkfileutils Documentation

Release 1.3.2

Dec 02, 2017

Contents

CHAPTER 1

dkfileutils

1.1 dkfileutils package

1.1.1 Submodules

1.1.2 dkfileutils.changed module

Check if contents of directory has changed.

class dkfileutils.changed.**Directory**
Bases: *dkfileutils.path.Path*

A path that is a directory.

changed (*filename='.md5'*, *glob=None*)
Are any of the files matched by glob changed?

dkfileutils.changed.**changed** (*dirname*, *filename='.md5'*, *args=None*, *glob=None*)
Has *glob* changed in *dirname*

Args: dirname: directory to measure filename: filename to store checksum

dkfileutils.changed.**digest** (*dirname*, *glob=None*)
Returns the md5 digest of all interesting files (or glob) in *dirname*.

dkfileutils.changed.**main** ()
Return exit code of zero iff directory is not changed.

1.1.3 dkfileutils.listfiles module

List interesting files.

dkfileutils.listfiles.**list_files** (*dirname='.'*, *digest=True*)
Yield (digest, fname) tuples for all interesting files in *dirname*.

dkfileutils.listfiles.**main**()
Print checksum and file name for all files in the directory.

dkfileutils.listfiles.**read_skipfile**(*dirname='.'*, *defaults=None*)
The .skipfile should contain one entry per line, listing files/directories that should be skipped by *list_files()*.

1.1.4 dkfileutils.path module

Poor man's pathlib.

(Path instances are subclasses of str, so interoperability with existing os.path code is greater than with Python 3's pathlib.)

class dkfileutils.path.**Path**
Bases: str

Poor man's pathlib.

absolute()
Return an absolute path.

abspath()
Return an absolute path.

access(*path*, *mode*) → True if granted, False otherwise
Use the real uid/gid to test for access to a path. 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. The mode argument can be F_OK to test existence, or the inclusive-OR of R_OK, W_OK, and X_OK.

append(*txt*, *mode='a'*)

basename()
Returns the final component of a pathname

cd(**args*, ***kwds*)

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

chmod(*path*, *mode*)
Change the access permissions of a file.

commonprefix(**args*)
Given a list of pathnames, returns the longest common leading component

contents()

classmethod curdir()
Initialize a Path object on the current directory.

dirname()
Returns the directory component of a pathname

drive()
Return the drive of *self*.

drivepath()
The path local to this drive (i.e. remove drive letter).

exists()
Test whether a path exists. Returns False for broken symbolic links

expanduser()
Expand ~ and ~user constructions. If user or $HOME is unknown, do nothing.

expandvars()
Expand shell variables of form $var and ${var}. Unknown variables are left unchanged.

ext

files()
Return all files in directory.

getatime()
Return the last access time of a file, reported by os.stat().

getctime()
Return the metadata change time of a file, reported by os.stat().

getmtime()
Return the last modification time of a file, reported by os.stat().

getsize()
Return the size of a file, reported by os.stat().

glob(*pat*)
pat can be an extended glob pattern, e.g. *'**/*.less'* This code handles negations similarly to node.js' minimatch, i.e. a leading *!* will negate the entire pattern.

isabs()
Test whether a path is absolute

isdir(**args*, ***kw*)
Return true if the pathname refers to an existing directory.

isfile()
Test whether a path is a regular file

islink()
Test whether a path is a symbolic link

ismount()
Test whether a path is a mount point

join(**args*)
Join two or more pathname components, inserting '/' as needed. If any component is an absolute path, all previous path components will be discarded. An empty last part will result in a path that ends with a separator.

lexists()
Test whether a path exists. Returns True for broken symbolic links

list(*filterfn=<function <lambda>>*)
Return all direct descendands of directory *self* for which *filterfn* returns True.

listdir(*path*) → list_of_strings
Return a list containing the names of the entries in the directory.

path: path of directory to list

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

lstat(*path*) → stat result
Like stat(path), but do not follow symbolic links.

makedirs(*path*[, *mode=0777*])
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. This is recursive.

mkdir(*path*[, *mode=0777*])
Create a directory.

normcase()
Normalize case of pathname. Has no effect under Posix

normpath()
Normalize path, eliminating double slashes, etc.

open(*mode='r'*)

parent

parent_iter()

parents

parts()

read(*mode='r'*)

realpath()
Return the canonical path of the specified filename, eliminating any symbolic links encountered in the path.

relpath(*other=''*)
Return a relative version of a path

remove(*path*)
Remove a file (same as unlink(path)).

removedirs(*path*)
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(*old*, *new*)
Rename a file or directory.

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.

rm(*fname=None*)
Remove a file, don't raise exception if file does not exist.

rmdir(*path*)
Remove a directory.

rmtree(*subdir=None*)
Recursively delete a directory tree.

If ignore_errors is set, errors are ignored; otherwise, if onerror is set, it is called to handle the error with arguments (func, path, exc_info) where func is os.listdir, os.remove, or os.rmdir; path is the argument to

that function that caused it to fail; and exc_info is a tuple returned by sys.exc_info(). If ignore_errors is false and onerror is None, an exception is raised.

split(*sep=None*, *maxsplit=-1*)
Split a pathname. Returns tuple "(head, tail)" where "tail" is everything after the final slash. Either part may be empty.

splitdrive()
Split a pathname into drive and path. On Posix, drive is always empty.

splitext()
Split the extension from a pathname.

Extension is everything from the last dot to the end, ignoring leading dots. Returns "(root, ext)"; ext may be empty.

stat(*path*) → stat result
Perform a stat system call on the given path.

subdirs()
Return all direct sub-directories.

touch(*mode=438*, *exist_ok=True*)
Create this file with the given access mode, if it doesn't exist. Based on:

https://github.com/python/cpython/blob/master/Lib/pathlib.py)

unlink(*path*)
Remove a file (same as remove(path)).

utime(*path*, *(atime*, *mtime)*)
utime(path, None)

Set the access and modified time of the file to the given values. If the second form is used, set the access and modified times to the current time.

write(*txt*, *mode='w'*)

dkfileutils.path.**cd**(**args*, ***kwds*)

dkfileutils.path.**doc**(*srcfn*)

1.1.5 dkfileutils.pfind module

CLI usage: `pfind path filename` will find the closest ancestor directory conataining filename (used for finding syncspec.txt and config files).

dkfileutils.pfind.**pfind**(*path*, **fnames*)
Find the first fname in the closest ancestor directory. For the purposes of this function, we are our own closest ancestor, i.e. given the structure:

```
/srv
|-- myapp
|   |-- __init__.py
|   `-- myapp.py
`-- setup.py
```

then both `pfind('/srv', 'setup.py')` and `pfind('/srv/myapp', 'setup.py')` will return `/srv/setup.py`

dkfileutils.pfind.**pfindall**(*path*, **fnames*)
Find all fnames in the closest ancestor directory. For the purposes of this function, we are our own closest ancestor. I.e. given the structure:

```
.
`-- a
    |-- b
    |   |-- c
    |   |   `-- x.txt
    |   `-- x.txt
    `-- y.txt
```

the call:

```
dict(pfindall('a/b/c', 'x.txt', 'y.txt'))
```

will return:

```
{
    'x.txt': 'a/b/c/x.txt',
    'y.txt': 'a/y.txt'
}
```

`a/b/x.txt` is not returned, since `a/b/c/x.txt` is the "closest" `x.txt` when starting from `a/b/c` (note: pfindall only looks "upwards", ie. towards the root).

1.1.6 dkfileutils.which module

Print where on the path an executable is located.

dkfileutils.which.**get_executable**(*name*)
Return the first executable on the path that matches *name*.

dkfileutils.which.**get_path_directories**()
Return a list of all the directories on the path.

dkfileutils.which.**is_executable**(*fname*)
Check if a file is executable.

dkfileutils.which.**which**(*filename*, *interactive=False*, *verbose=False*)
Yield all executable files on path that matches *filename*.

1.1.7 Module contents

dkfileutils - simple, common file utilities.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

O

P

R

S

T

U

W