WalkDir Documentation

Release 0.4.1.post1

Nick Coghlan

May 10, 2016

Contents

Module author: Nick Coghlan <ncoghlan@gmail.com>

The standard libary's `os.walk()` iterator provides a convenient way to process the contents of a filesystem directory. This module provides higher level tools based on the same interface that support filtering, depth limiting and handling of symlink loops. The module also offers tools that flatten the `os.walk()` API into a simple iteration over filesystem paths.

CHAPTER 1

Walk Iterables

In this module, `walk_iter` refers to any iterable that produces `path, subdirs, files` triples sufficiently compatible with those produced by `os.walk()`.

The module is designed so that all purely filtering operations *preserve* the output of the underlying iterable. This means that named tuples, tuples containing more than 3 values (such as those produced by `os.fwalk()`), and objects that aren't tuples at all but are still defined such that `x[0], x[1], x[2] => dirpath, subdirs, files`, can be filtered without being converted to ordinary 3-tuples.

Changed in version 0.3: Objects produced by underlying iterables are now preserved instead of being coerced to ordinary 3-tuples by filtering operations

CHAPTER 2

Path Iteration

Four iterators are provided for iteration over filesystem paths:

file_paths(*walk_iter*)
Iterate over the files in directories visited by the underlying walk

Directory contents are emitted in the order visited, so the underlying walk may be either top-down or bottom-up.

dir_paths(*walk_iter*)
Iterate over the directories visited by the underlying walk

Directories are emitted in the order visited, so the underlying walk may be either top-down or bottom-up.

all_dir_paths(*walk_iter*)
Iterate over all directories reachable through the underlying walk

This covers:

- all visited directories (similar to dir_paths)
- all reported subdirectories of visited directories (even if not otherwise visited)

Example cases where the output may differ from dir_paths:

- all_dir_paths always includes symlinks to directories even when the underlying iterator doesn't follow symlinks
- all_dir_paths will include subdirectories of directories at the maximum depth in a depth limited walk

This iterator expects new root directories to be emitted by the underlying walk before any of their contents, and hence requires a top-down traversal of the directory hierarchy.

New in version 0.4.

all_paths(*walk_iter*)
Iterate over all paths reachable through the underlying walk

This covers:

- all visited directories
- all files in visited directories
- all reported subdirectories of visited directories (even if not otherwise visited)

This iterator expects new root directories to be emitted by the underlying walk before any of their contents, and hence requires a top-down traversal of the directory hierarchy.

Changed in version 0.4: This function now combines the output of *file_paths()* with that of *all_dir_paths()* (previously it was the combination of *file_paths()* with *dir_paths()*)

Except when the underlying iterable switches to a new root directory, the last two functions yield subdirectory paths when visiting the parent directory, rather than when visiting the subdirectory.

For example, given the following directory tree:

```
>>> tree test
test
-- file1.txt
-- file2.txt
-- test2
|   -- file1.txt
|   -- file2.txt
|   -- test3
-- test4
    -- file1.txt
    -- test5
```

`all_paths` will produce:

```
>>> from walkdir import filtered_walk, all_paths
>>> paths = all_paths(filtered_walk('test'))
>>> print('\n'.join(paths))
test
test/file1.txt
test/file2.txt
test/test2
test/test4
test/test2/file1.txt
test/test2/file2.txt
test/test2/test3
test/test4/file1.txt
test/test4/test5
```

`all_dir_paths` will produce:

```
>>> from walkdir import filtered_walk, all_dir_paths
>>> paths = all_dir_paths(filtered_walk('test'))
>>> print('\n'.join(paths))
test
test/test2
test/test4
test/test2/test3
test/test4/test5
```

`dir_paths` will produce:

```
>>> from walkdir import filtered_walk, dir_paths
>>> paths = dir_paths(filtered_walk('test'))
>>> print('\n'.join(paths))
test
test/test2
test/test2/test3
test/test4
test/test4/test5
```

And `file_paths` will produce:

```
>>> from walkdir import filtered_walk, file_paths
>>> paths = file_paths(filtered_walk('test'))
>>> print('\n'.join(paths))
test/file1.txt
```

```
test/file2.txt
test/test2/file1.txt
test/test2/file2.txt
test/test4/file1.txt
```

Note: When used with *min_depth()* the output will be produced as multiple independent walks of each directory bigger than given *min_depth*.

Changed in version 0.4: Subdirectories are now emitted when visiting the parent directory, rather than when visiting the subdirectory itself. This means that subdirectories may now be emitted without being visited (e.g. subdirectories of directories visited by a depth-limited walk, symlinks to subdirectories when not following links), and all subdirectories of a given parent directory are emitted as a contiguous block, rather than being interleaved with their respective file listings.

CHAPTER 3

Directory Walking

A convenience API for walking directories with various options is provided:

filtered_walk(*top*, *included_files=None*, *included_dirs=None*, *excluded_files=None*, *excluded_dirs=None*, *depth=None*, *followlinks=False*, *min_depth=None*)

This is a wrapper around `os.walk()` and other filesystem traversal iterators, with these additional features:

- *top* may be either a string (which will be passed to `os.walk()`) or any iterable that produces sequences with `path, subdirs, files` as the first three elements in the sequence
- allows independent glob-style filters for filenames and subdirectories
- allows a recursion depth limit to be specified
- allows a minimum depth to be specified to report only subdirectory contents
- emits a message to stderr and skips the directory if a symlink loop is encountered when following links

Filtered walks created by passing in a string are always top down, as the subdirectory listings must be altered to provide a number of the above features.

include_files, *include_dirs*, *exclude_files* and *exclude_dirs* are used to apply the relevant filtering steps to the walk.

A *depth* of `None` (the default) disables depth limiting. Otherwise, *depth* must be at least zero and indicates how far to descend into the directory hierarchy. A depth of zero is useful to get separate filtered subdirectory and file listings for *top*.

Setting *min_depth* allows directories higher in the tree to be excluded from the walk (e.g. a *min_depth* of 1 excludes *top*, but any subdirectories will still be processed)

followlinks enables symbolic loop detection (when set to `True`) and is also passed to `os.walk()` when top is a string

The individual operations that support the convenience API are exposed using an `itertools` style iterator pipeline model:

include_dirs(*walk_iter*, **include_filters*)

Use `fnmatch.fnmatch()` patterns to select directories of interest

Inclusion filters are passed directly as arguments.

This filter works by modifying the subdirectory lists produced by the underlying iterator, and hence requires a top-down traversal of the directory hierarchy.

include_files(*walk_iter*, **include_filters*)

Use `fnmatch.fnmatch()` patterns to select files of interest

Inclusion filters are passed directly as arguments

This filter does not modify the subdirectory lists produced by the underlying iterator, and hence supports both top-down and bottom-up traversal of the directory hierarchy.

exclude_dirs(*walk_iter*, **exclude_filters*)
Use `fnmatch.fnmatch()` patterns to skip irrelevant directories

Exclusion filters are passed directly as arguments

This filter works by modifying the subdirectory lists produced by the underlying iterator, and hence requires a top-down traversal of the directory hierarchy.

exclude_files(*walk_iter*, **exclude_filters*)
Use `fnmatch.fnmatch()` patterns to skip irrelevant files

Exclusion filters are passed directly as arguments

This filter does not modify the subdirectory lists produced by the underlying iterator, and hence supports both top-down and bottom-up traversal of the directory hierarchy.

limit_depth(*walk_iter*, *depth*)
Limit the depth of recursion into subdirectories.

A *depth* of 0 limits the walk to the top level directory, a *depth* of 1 includes subdirectories, etc.

Path depth is calculated by counting directory separators, using the depth of the first path produced by the underlying iterator as a reference point.

This filter works by modifying the subdirectory lists produced by the underlying iterator, and hence requires a top-down traversal of the directory hierarchy.

min_depth(*walk_iter*, *depth*)
Only process subdirectories beyond a minimum depth

A *depth* of 1 omits the top level directory, a *depth* of 2 starts with subdirectories 2 levels down, etc.

Path depth is calculated by counting directory separators, using the depth of the first path produced by the underlying iterator as a reference point.

Note: Since this filter *doesn't yield* higher level directories, any subsequent directory filtering that relies on updating the subdirectory list will have no effect at the minimum depth. Accordingly, this filter should only be applied *after* any directory filtering operations.

Note: The result of using this filter is effectively the same as chaining multiple independent `os.walk()` iterators using `itertools.chain()`. For example, given the following directory tree:

```
>>> tree test
test
-- file1.txt
-- file2.txt
-- test2
|   -- file1.txt
|   -- file2.txt
|   -- test3
-- test4
    -- file1.txt
    -- test5
```

Then `min_depth(os.walk("test"), depth=1)` will produce the same output as `itertools.chain(os.walk("test/test2"), os.walk("test/test4"))`.

This filter works by modifying the subdirectory lists produced by the underlying iterator, and hence requires a top-down traversal of the directory hierarchy.

handle_symlink_loops(*walk_iter*, *onloop=None*)
Handle symlink loops when following symlinks during a walk

By default, prints a warning and then skips processing the directory a second time.

This can be overridden by providing the *onloop* callback, which accepts the offending symlink as a parameter. Returning a true value from this callback will mean that the directory is still processed, otherwise it will be skipped.

This filter skips processing subdirectories by modifying the subdirectory lists produced by the underlying iterator, and hence requires a top-down traversal of the directory hierarchy.

CHAPTER 4

Examples

Here are some examples of the module being used to explore the contents of its own source tree:

```
>>> from walkdir import filtered_walk, dir_paths, all_paths, file_paths
>>> files = file_paths(filtered_walk('.', depth=0,
...                   included_files=['*.py', '*.txt', '*.rst']))
>>> print '\n'.join(files)
./setup.py
./walkdir.py
./NEWS.rst
./test_walkdir.py
./LICENSE.txt
./VERSION.txt
./README.txt
>>> dirs = dir_paths(filtered_walk('.', depth=1, min_depth=1,
...                  excluded_dirs=['__pycache__', '.git']))
>>> print '\n'.join(dirs)
./docs
./dist
>>> paths = all_paths(filtered_walk('.', depth=1,
...                   included_files=['*.py', '*.txt', '*.rst'],
...                   excluded_dirs=['__pycache__', '.git']))
>>> print '\n'.join(paths)
.
./setup.py
./walkdir.py
./NEWS.rst
./test_walkdir.py
./LICENSE.txt
./VERSION.txt
./README.txt
./docs
./docs/index.rst
./docs/conf.py
./dist
```

CHAPTER 5

Obtaining the Module

This module can be installed directly from the Python Package Index with pip:

```
pip install walkdir
```

Alternatively, you can download and unpack it manually from the walkdir PyPI page.

There are no operating system or distribution specific versions of this module - it is a pure Python module that should work on all platforms.

Supported Python versions are 2.6, 2.7 and 3.1+.

5.1 Development and Support

WalkDir is developed and maintained on Gitub, with continuous integration services provided by Travis-CI.

Problems and suggested improvements can be posted to the issue tracker.

5.2 Release History

5.2.1 0.4.1 (2016-05-10)

- Include release date in release history

5.2.2 0.4 (2016-05-10)

- *SEMANTIC CHANGE*: to implement some of the fixes noted below, the `all_paths` iterator has been updated to emit paths in the following order for each directory produced by the underling iterator:
  - given directory if it appears to be a new root directory (i.e. it is not a subdirectory of the current root directory)
  - files in the given directory
  - subdirectories of the given directory

  Previously, directories were only emitted when walked by the underling iterator, which resulted in paths being missed in some cases.

- Thanks go to Aviv Palivoda for being the driving force behind this release, especially in addressing a variety of issues in the way directory filtering and symlinks to directories are handled.
- Issue #12: a new API, `all_dir_paths` has been added which, in addition to the directories visited by the underlying walk, also emits:
  - symlinks to directories when `followlinks` is disabled in the underlying iterator
  - subdirectories of leaf directories when the directory tree depth of the underlying iterator has been limited (for example, with the `limit_depth` filter)
- Issue #3: `all_paths` now correctly reports symlinks to directories as directory paths, even when `followlinks` is disabled in the underlying iterator (fix contributed by Aviv Palivoda)
- Issue #4: `all_paths` now correctly reports subdirectories at the maximum depth when the `limit_depth` filter is used to trim nested subdirectories (fix contributed by Aviv Palivoda)
- Issue #6: `min_depth`, `all_paths`, `dir_paths`, and `file_paths` all now work correctly with `os.fwalk` and other underlying iterators that produce a sequence with more than 3 elements for each directory (fix contributed by Aviv Palivoda)
- Issue #7: all filters now explicitly indicate in their documentation whether or not they support being used with bottom-up traversal of the underlying directory hierarchy
- A temporary generated filesystem is now used to test symlink loop handling and other behaviours that require a real filesystem (patch contributed by Aviv Palivoda)
- The correct error message is now emitted when an invalid maximum depth is passed to `limit_depth` on Python 2.6 (fix contributed by Aviv Palivoda)
- The correct error message is now emitted when an invalid minimum depth is passed to `min_depth` on Python 2.6 (fix contributed by Aviv Palivoda)
- development has migrated from BitBucket to GitHub

5.2.3 0.3 (2012-01-31)

- (BitBucket) Issue #7: filter functions now pass the tuples created by underlying iterators through without modification, using indexing rather than tuple unpacking to access values of interest. This means WalkDir now supports any underlying iterable that produces items where `x[0], x[1], x[2]` refers to `dirpath, subdirs, files`. For example, if the the iterable produces `collections.namedtuple` instances, those will be passed through to the output of a filtered walk.

5.2.4 0.2.1 (2012-01-17)

- Add MANIFEST.in so PyPI package contains all relevant files

5.2.5 0.2 (2012-01-04)

- (BitBucket) Issue #6: Added a `min_depth` option to `filtered_walk` and a new `min_depth` filter function to make it easier to produce a list of full subdirectory paths
- **(BitBucket) Issue #5: Renamed path iteration convenience APIs:**
  - `iter_paths` -> `all_paths`
  - `iter_dir_paths` -> `dir_paths`
  - `iter_file_paths` -> `file_paths`

- Moved version number to a VERSION.txt file (read by both docs and setup.py)
- Added NEWS.rst (and incorporated into documentation)

5.2.6 0.1 (2011-11-13)

- Initial release

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

W

A

D

E

F

H

I

L

M

W