filecmp — Compare Files

Purpose: Compare files and directories on the file system.

The `filecmp` module includes functions and a class for comparing files and directories on the file system.

Example Data

The examples in this discussion use a set of test files created by `filecmp_mkexamples.py`.

```
# filecmp_mkexamples.py

import os


def mkfile(filename, body=None):
    with open(filename, 'w') as f:
        f.write(body or filename)
    return


def make_example_dir(top):
    if not os.path.exists(top):
        os.mkdir(top)
    curdir = os.getcwd()
    os.chdir(top)

    os.mkdir('dir1')
    os.mkdir('dir2')

    mkfile('dir1/file_only_in_dir1')
    mkfile('dir2/file_only_in_dir2')

    os.mkdir('dir1/dir_only_in_dir1')
    os.mkdir('dir2/dir_only_in_dir2')

    os.mkdir('dir1/common_dir')
    os.mkdir('dir2/common_dir')

    mkfile('dir1/common_file', 'this file is the same')
    os.link('dir1/common_file', 'dir2/common_file')

    mkfile('dir1/contents_differ')
    mkfile('dir2/contents_differ')
    # Update the access and modification times so most of the stat
    # results will match.
    st = os.stat('dir1/contents_differ')
    os.utime('dir2/contents_differ', (st.st_atime, st.st_mtime))

    mkfile('dir1/file_in_dir1', 'This is a file in dir1')
    os.mkdir('dir2/file_in_dir1')

    os.chdir(curdir)
    return


if __name__ == '__main__':
    os.chdir(os.path.dirname(__file__) or os.getcwd())
    make_example_dir('example')
    make_example_dir('example/dir1/common_dir')
    make_example_dir('example/dir2/common_dir')
```

Running the script produces a tree of files under the directory `example`:

```
$ find example | sort

example
example/dir1
example/dir1/common_dir
example/dir1/common_dir/dir1
example/dir1/common_dir/dir1/common_dir
example/dir1/common_dir/dir1/common_file
example/dir1/common_dir/dir1/contents_differ
example/dir1/common_dir/dir1/dir_only_in_dir1
example/dir1/common_dir/dir1/file_in_dir1
example/dir1/common_dir/dir1/file_only_in_dir1
example/dir1/common_dir/dir2
example/dir1/common_dir/dir2/common_dir
example/dir1/common_dir/dir2/common_file
example/dir1/common_dir/dir2/contents_differ
example/dir1/common_dir/dir2/dir_only_in_dir2
example/dir1/common_dir/dir2/file_in_dir1
example/dir1/common_dir/dir2/file_only_in_dir2
example/dir1/common_file
example/dir1/contents_differ
example/dir1/dir_only_in_dir1
example/dir1/file_in_dir1
example/dir1/file_only_in_dir1
example/dir2
example/dir2/common_dir
example/dir2/common_dir/dir1
example/dir2/common_dir/dir1/common_dir
example/dir2/common_dir/dir1/common_file
example/dir2/common_dir/dir1/contents_differ
example/dir2/common_dir/dir1/dir_only_in_dir1
example/dir2/common_dir/dir1/file_in_dir1
example/dir2/common_dir/dir1/file_only_in_dir1
example/dir2/common_dir/dir2
example/dir2/common_dir/dir2/common_dir
example/dir2/common_dir/dir2/common_file
example/dir2/common_dir/dir2/contents_differ
example/dir2/common_dir/dir2/dir_only_in_dir2
example/dir2/common_dir/dir2/file_in_dir1
example/dir2/common_dir/dir2/file_only_in_dir2
example/dir2/common_file
example/dir2/contents_differ
example/dir2/dir_only_in_dir2
example/dir2/file_in_dir1
example/dir2/file_only_in_dir2
```

The same directory structure is repeated one time under the “common_dir” directories to give interesting recursive comparison options.

Comparing Files

`cmp()` compares two files on the file system.

```python
# filecmp_cmp.py

import filecmp

print('common_file    :', end=' ')
print(filecmp.cmp('example/dir1/common_file',
                  'example/dir2/common_file',
                  shallow=True),
      end=' ')
print(filecmp.cmp('example/dir1/common_file',
                  'example/dir2/common_file',
                  shallow=False))

print('contents_differ:', end=' ')
print(filecmp.cmp('example/dir1/contents_differ',
                  'example/dir2/contents_differ',
                  shallow=True),
      end=' ')
print(filecmp.cmp('example/dir1/contents_differ',
                  'example/dir2/contents_differ',
                  shallow=False))

print('identical      :', end=' ')
print(filecmp.cmp('example/dir1/file_only_in_dir1',
                  'example/dir1/file_only_in_dir1',
                  shallow=True),
      end=' ')
print(filecmp.cmp('example/dir1/file_only_in_dir1',
                  'example/dir1/file_only_in_dir1',
                  shallow=False))
```

The `shallow` argument tells `cmp()` whether to look at the contents of the file, in addition to its metadata. The default is to perform a shallow comparison using the information available from `os.stat()`. If the stat results are the same, the files are considered the same. Because the stat output includes the inode on Linux, separate files are not treated as the same even if all of their other metadata (size, creation time, etc.) match. In those cases, the file contents are compared. When `shallow` is `False`, the contents of the file are always compared.

```
$ python3 filecmp_cmp.py

common_file    : True True
contents_differ: False False
identical      : True True
```

To compare a set of files in two directories without recursing, use `cmpfiles()`. The arguments are the names of the directories and a list of files to be checked in the two locations. The list of common files passed in should contain only filenames (directories always result in a mismatch) and the files must be present in both locations. The next example shows a simple way to build the common list. The comparison also takes the `shallow` flag, just as with `cmp()`.

```
# filecmp_cmpfiles.py

import filecmp
import os

# Determine the items that exist in both directories
d1_contents = set(os.listdir('example/dir1'))
d2_contents = set(os.listdir('example/dir2'))
common = list(d1_contents & d2_contents)
common_files = [
    f
    for f in common
    if os.path.isfile(os.path.join('example/dir1', f))
]
print('Common files:', common_files)

# Compare the directories
match, mismatch, errors = filecmp.cmpfiles(
    'example/dir1',
    'example/dir2',
    common_files,
)
print('Match       :', match)
print('Mismatch    :', mismatch)
print('Errors      :', errors)
```

`cmpfiles()` returns three lists of filenames containing files that match, files that do not match, and files that could not be compared (due to permission problems or for any other reason).

```
$ python3 filecmp_cmpfiles.py

Common files: ['contents_differ', 'file_in_dir1', 'common_file']
Match       : ['common_file']
Mismatch    : ['contents_differ', 'file_in_dir1']
Errors      : []
```

Comparing Directories

The functions described earlier are suitable for relatively simple comparisons. For recursive comparison of large directory trees or for more complete analysis, the `dircmp` class is more useful. In its simplest use case, `report()` prints a report comparing two directories.

```
# filecmp_dircmp_report.py

import filecmp

dc = filecmp.dircmp('example/dir1', 'example/dir2')
dc.report()
```

The output is a plain-text report showing the results of just the contents of the directories given, without recursing.

```
$ python3 filecmp_dircmp_report.py

diff example/dir1 example/dir2
Only in example/dir1 : ['dir_only_in_dir1', 'file_only_in_dir1']
Only in example/dir2 : ['dir_only_in_dir2', 'file_only_in_dir2']
Identical files : ['common_file']
Differing files : ['contents_differ']
Common subdirectories : ['common_dir']
Common funny cases : ['file_in_dir1']
```

For more detail, and a recursive comparison, use `report_full_closure()`:

```
# filecmp_dircmp_report_full_closure.py

import filecmp

dc = filecmp.dircmp('example/dir1', 'example/dir2')
dc.report_full_closure()
```

The output includes comparisons of all parallel subdirectories.

```
$ python3 filecmp_dircmp_report_full_closure.py

diff example/dir1 example/dir2
Only in example/dir1 : ['dir_only_in_dir1', 'file_only_in_dir1']
Only in example/dir2 : ['dir_only_in_dir2', 'file_only_in_dir2']
Identical files : ['common_file']
Differing files : ['contents_differ']
Common subdirectories : ['common_dir']
Common funny cases : ['file_in_dir1']

diff example/dir1/common_dir example/dir2/common_dir
Common subdirectories : ['dir1', 'dir2']

diff example/dir1/common_dir/dir1 example/dir2/common_dir/dir1
Identical files : ['common_file', 'contents_differ',
'file_in_dir1', 'file_only_in_dir1']
Common subdirectories : ['common_dir', 'dir_only_in_dir1']

diff example/dir1/common_dir/dir1/common_dir
example/dir2/common_dir/dir1/common_dir

diff example/dir1/common_dir/dir1/dir_only_in_dir1
example/dir2/common_dir/dir1/dir_only_in_dir1

diff example/dir1/common_dir/dir2 example/dir2/common_dir/dir2
Identical files : ['common_file', 'contents_differ',
'file_only_in_dir2']
Common subdirectories : ['common_dir', 'dir_only_in_dir2',
'file_in_dir1']

diff example/dir1/common_dir/dir2/common_dir
example/dir2/common_dir/dir2/common_dir

diff example/dir1/common_dir/dir2/dir_only_in_dir2
example/dir2/common_dir/dir2/dir_only_in_dir2
```

```
example/dir2/common_dir/dir2/dir_only_in_dir2

diff example/dir1/common_dir/dir2/file_in_dir1
example/dir2/common_dir/dir2/file_in_dir1
```

Using Differences in a Program

Besides producing printed reports, `dircmp` calculates lists of files that can be used in programs directly. Each of the following attributes is calculated only when requested, so creating a `dircmp` instance does not incur overhead for unused data.

```
# filecmp_dircmp_list.py

import filecmp
import pprint

dc = filecmp.dircmp('example/dir1', 'example/dir2')
print('Left:')
pprint.pprint(dc.left_list)

print('\nRight:')
pprint.pprint(dc.right_list)
```

The files and subdirectories contained in the directories being compared are listed in `left_list` and `right_list`.

```
$ python3 filecmp_dircmp_list.py

Left:
['common_dir',
 'common_file',
 'contents_differ',
 'dir_only_in_dir1',
 'file_in_dir1',
 'file_only_in_dir1']

Right:
['common_dir',
 'common_file',
 'contents_differ',
 'dir_only_in_dir2',
 'file_in_dir1',
 'file_only_in_dir2']
```

The inputs can be filtered by passing a list of names to ignore to the constructor. By default the names RCS, CVS, and `tags` are ignored.

```
# filecmp_dircmp_list_filter.py

import filecmp
import pprint

dc = filecmp.dircmp('example/dir1', 'example/dir2',
                    ignore=['common_file'])

print('Left:')
pprint.pprint(dc.left_list)

print('\nRight:')
pprint.pprint(dc.right_list)
```

In this case, the “common_file” is left out of the list of files to be compared.

```
$ python3 filecmp_dircmp_list_filter.py

Left:
['common_dir',
 'contents_differ',
 'dir_only_in_dir1',
 'file_in_dir1',
 'file_only_in_dir1']
 'file_only_in_dir1']

Right:
['common_dir',
 'contents_differ',
 'dir_only_in_dir2',
 'file_in_dir1',
 'file_only_in_dir2']
```

The names of files common to both input directories are saved in `common`, and the files unique to each directory are listed in `left_only`, and `right_only`.

```python
# filecmp_dircmp_membership.py

import filecmp
import pprint

dc = filecmp.dircmp('example/dir1', 'example/dir2')
print('Common:')
pprint.pprint(dc.common)

print('\nLeft:')
pprint.pprint(dc.left_only)

print('\nRight:')
pprint.pprint(dc.right_only)
```

The “left” directory is the first argument to `dircmp()` and the “right” directory is the second.

```
$ python3 filecmp_dircmp_membership.py

Common:
['common_dir', 'common_file', 'contents_differ', 'file_in_dir1']

Left:
['dir_only_in_dir1', 'file_only_in_dir1']

Right:
['dir_only_in_dir2', 'file_only_in_dir2']
```

The common members can be further broken down into files, directories and “funny” items (anything that has a different type in the two directories or where there is an error from `os.stat()`).

```python
# filecmp_dircmp_common.py

import filecmp
import pprint

dc = filecmp.dircmp('example/dir1', 'example/dir2')
print('Common:')
pprint.pprint(dc.common)

print('\nDirectories:')
pprint.pprint(dc.common_dirs)

print('\nFiles:')
pprint.pprint(dc.common_files)

print('\nFunny:')
pprint.pprint(dc.common_funny)
```

In the example data, the item named “`file_in_dir1`” is a file in one directory and a subdirectory in the other, so it shows up in the funny list.

```
$ python3 filecmp_dircmp_common.py

Common:
['common_dir', 'common_file', 'contents_differ', 'file_in_dir1']

Directories:
```

```
Directories:
['common_dir']

Files:
['common_file', 'contents_differ']

Funny:
['file_in_dir1']
```

The differences between files are broken down similarly.

```
# filecmp_dircmp_diff.py

import filecmp

dc = filecmp.dircmp('example/dir1', 'example/dir2')
print('Same      :', dc.same_files)
print('Different :', dc.diff_files)
print('Funny     :', dc.funny_files)
```

The file `not_the_same` is only being compared via `os.stat()`, and the contents are not examined, so it is included in the `same_files` list.

```
$ python3 filecmp_dircmp_diff.py

Same      : ['common_file']
Different : ['contents_differ']
Funny     : []
```

Finally, the subdirectories are also saved to allow easy recursive comparison.

```
# filecmp_dircmp_subdirs.py

import filecmp

dc = filecmp.dircmp('example/dir1', 'example/dir2')
print('Subdirectories:')
print(dc.subdirs)
```

The attribute `subdirs` is a dictionary mapping the directory name to new `dircmp` objects.

```
$ python3 filecmp_dircmp_subdirs.py

Subdirectories:
{'common_dir': <filecmp.dircmp object at 0x1101fe710>}
```

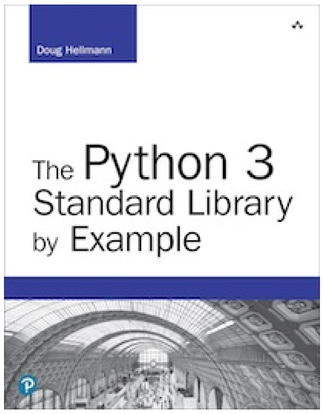
See also

- Standard library documentation for filecmp
- `difflib` – Computing the differences between two sequences.

shutil — High-level File Operations

mmap — Memory-map Files

This page was last updated 2018-12-09.



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?