# 11.5. `filecmp` — File and Directory Comparisons

**Source code:** Lib/filecmp.py

The `filecmp` module defines functions to compare files and directories, with various optional time/correctness trade-offs. For comparing files, see also the `difflib` module.

The `filecmp` module defines the following functions:

`filecmp.`**`cmp`**(*f1*, *f2*, *shallow=True*)

Compare the files named *f1* and *f2*, returning `True` if they seem equal, `False` otherwise.

If *shallow* is true, files with identical `os.stat()` signatures are taken to be equal. Otherwise, the contents of the files are compared.

Note that no external programs are called from this function, giving it portability and efficiency.

This function uses a cache for past comparisons and the results, with cache entries invalidated if the `os.stat()` information for the file changes. The entire cache may be cleared using `clear_cache()`.

`filecmp.`**`cmpfiles`**(*dir1*, *dir2*, *common*, *shallow=True*)

Compare the files in the two directories *dir1* and *dir2* whose names are given by *common*.

Returns three lists of file names: *match*, *mismatch*, *errors*. *match* contains the list of files that match, *mismatch* contains the names of those that don't, and *errors* lists the names of files which could not be compared. Files are listed in *errors* if they don't exist in one of the directories, the user lacks permission to read them or if the comparison could not be done for some other reason.

The *shallow* parameter has the same meaning and default value as for `filecmp.cmp()`.

For example, `cmpfiles('a', 'b', ['c', 'd/e'])` will compare `a/c` with `b/c` and `a/d/e` with `b/d/e`. `'c'` and `'d/e'` will each be in one of the three returned lists.

`filecmp.`**`clear_cache`**()

Clear the filecmp cache. This may be useful if a file is compared so quickly after it is modified that it is within the mtime resolution of the underlying filesystem.

*New in version 3.4.*

## 11.5.1. The `dircmp` class

*class* `filecmp.`**`dircmp`**(*a*, *b*, *ignore=None*, *hide=None*)

Construct a new directory comparison object, to compare the directories *a* and *b*. *ignore* is a list of names to ignore, and defaults to `filecmp.DEFAULT_IGNORES`. *hide* is a list of names to hide, and defaults to `[os.curdir, os.pardir]`.

The `dircmp` class compares files by doing *shallow* comparisons as described for `filecmp.cmp()`.

The `dircmp` class provides the following methods:

**`report`**()

Print (to `sys.stdout`) a comparison between *a* and *b*.

**`report_partial_closure`**()

Print a comparison between *a* and *b* and common immediate subdirectories.

**`report_full_closure`**()

Print a comparison between *a* and *b* and common subdirectories (recursively).

The `dircmp` class offers a number of interesting attributes that may be used to get various bits of information about the directory trees being compared.

Note that via `__getattr__()` hooks, all attributes are computed lazily, so there is no speed penalty if only those attributes which are lightweight to compute are used.

**`left`**

The directory *a*.

**`right`**

The directory *b*.

**`left_list`**

Files and subdirectories in *a*, filtered by *hide* and *ignore*.

**right_list**

Files and subdirectories in *b*, filtered by *hide* and *ignore*.

**common**

Files and subdirectories in both *a* and *b*.

**left_only**

Files and subdirectories only in *a*.

**right_only**

Files and subdirectories only in *b*.

**common_dirs**

Subdirectories in both *a* and *b*.

**common_files**

Files in both *a* and *b*.

**common_funny**

Names in both *a* and *b*, such that the type differs between the directories, or names for which `os.stat()` reports an error.

**same_files**

Files which are identical in both *a* and *b*, using the class's file comparison operator.

**diff_files**

Files which are in both *a* and *b*, whose contents differ according to the class's file comparison operator.

**funny_files**

Files which are in both *a* and *b*, but could not be compared.

**subdirs**

A dictionary mapping names in `common_dirs` to `dircmp` objects.

`filecmp.`**DEFAULT_IGNORES**

*New in version 3.4.*

List of directories ignored by `dircmp` by default.

Here is a simplified example of using the `subdirs` attribute to search recursively through two directories to show common different files:

```
>>> from filecmp import dircmp
>>> def print_diff_files(dcmp):
...     for name in dcmp.diff_files:
...         print("diff_file %s found in %s and %s" % (name, dcmp.left
...               dcmp.right))
...     for sub_dcmp in dcmp.subdirs.values():
...         print_diff_files(sub_dcmp)
...
>>> dcmp = dircmp('dir1', 'dir2')
>>> print_diff_files(dcmp)
```