`!py_compile` --- Compile Python source files

- Command-Line Interface

py_compile

Source code: `Lib/py_compile.py`

pair: file; byte-code

The `!py_compile` module provides a function to generate a byte-code file from a source file, and another function used when the module source file is invoked as a script.

Though not often needed, this function can be useful when installing modules for shared use, especially if some of the users may not have permission to write the byte-code cache files in the directory containing the source code.

PyCompileError

Exception raised when an error occurs while attempting to compile the file.

compile(file, cfile=None, dfile=None, doraise=False, optimize=-1, invalidation_mode=PycInvalidationMode.TIMESTAMP, quiet=0)

Compile a source file to byte-code and write out the byte-code cache file. The source code is loaded from the file named *file*. The byte-code is written to *cfile*, which defaults to the `3147`/`488` path, ending in `.pyc`. For example, if *file* is `/foo/bar/baz.py` *cfile* will default to `/foo/bar/__pycache__/baz.cpython-32.pyc` for Python 3.2. If *dfile* is specified, it is used instead of *file* as the name of the source file from which source lines are obtained for display in exception tracebacks. If *doraise* is true, a `PyCompileError` is raised when an error is encountered while compiling *file*. If *doraise* is false (the default), an error string is written to `sys.stderr`, but no exception is raised. This function returns the path to byte-compiled file, i.e. whatever *cfile* value was used.

The *doraise* and *quiet* arguments determine how errors are handled while compiling file. If *quiet* is 0 or 1, and *doraise* is false, the default behaviour is enabled: an error string is written to `sys.stderr`, and the function returns `None` instead of a path. If *doraise* is true, a `PyCompileError` is raised instead. However if *quiet* is 2, no message is written, and *doraise* has no effect.

If the path that *cfile* becomes (either explicitly specified or computed) is a symlink or non-regular file, `FileExistsError` will be raised. This is to act as a warning that import will turn those paths into regular files if it is allowed to write byte-compiled files to those paths. This is a side-effect of import using file renaming to place the final byte-compiled file into place to prevent concurrent file writing issues.

optimize controls the optimization level and is passed to the built-in `compile` function. The default of `-1` selects the optimization level of the current interpreter.

invalidation_mode should be a member of the `PycInvalidationMode` enum and controls how the generated bytecode cache is invalidated at runtime. The default is `PycInvalidationMode.CHECKED_HASH` if the `SOURCE_DATE_EPOCH` environment variable is set, otherwise the default is `PycInvalidationMode.TIMESTAMP`.

3.2 Changed default value of *cfile* to be PEP 3147-compliant. Previous default was *file* + `'c'` (`'o'` if optimization was enabled). Also added the *optimize* parameter.

3.4 Changed code to use `importlib` for the byte-code cache file writing. This means file creation/writing semantics now match what `importlib` does, e.g. permissions, write-and-move semantics, etc. Also added the caveat that `FileExistsError` is raised if *cfile* is a symlink or non-regular file.

3.7 The *invalidation_mode* parameter was added as specified in `552`. If the `SOURCE_DATE_EPOCH` environment variable is set, *invalidation_mode* will be forced to `PycInvalidationMode.CHECKED_HASH`.

3.7.2 The `SOURCE_DATE_EPOCH` environment variable no longer overrides the value of the *invalidation_mode* argument, and determines its default value instead.

3.8 The *quiet* parameter was added.

An enumeration of possible methods the interpreter can use to determine whether a bytecode file is up to date with a source file. The `.pyc` file indicates the desired invalidation mode in its header. See `pyc-invalidation` for more information on how Python invalidates `.pyc` files at runtime.

3.7

TIMESTAMP

The `.pyc` file includes the timestamp and size of the source file, which Python will compare against the metadata of the source file at runtime to determine if the `.pyc` file needs to be regenerated.

CHECKED_HASH

The `.pyc` file includes a hash of the source file content, which Python will compare against the source at runtime to determine if the `.pyc` file needs to be regenerated.

UNCHECKED_HASH

Like `CHECKED_HASH`, the `.pyc` file includes a hash of the source file content. However, Python will at runtime assume the `.pyc` file is up to date and not validate the `.pyc` against the source file at all.

This option is useful when the `.pycs` are kept up to date by some system external to Python like a build system.

Command-Line Interface

This module can be invoked as a script to compile several source files. The files named in *filenames* are compiled and the resulting bytecode is cached in the normal manner. This program does not search a directory structure to locate source files; it only compiles files named explicitly. The exit status is nonzero if one of the files could not be compiled.

python -m py_compile

<file> ... <fileN> -

Positional arguments are files to compile. If `-` is the only parameter, the list of files is taken from standard input.

`-q, --quiet`

Suppress errors output.

3.2 Added support for `-`.

3.10 Added support for `-q`.

Module `compileall`
Utilities to compile all Python source files in a directory tree.

□□□ #2
□ □□□ □□□ 2026-10-31 17:17:37 UTC

□ □□□ □□□ 2026-10-31 17:17:39 UTC