shutil — High-level File Operations

Purpose: High-level file operations.

The `shutil` module includes high-level file operations such as copying and archiving.

Copying Files

`copyfile()` copies the contents of the source to the destination and raises `IOError` if it does not have permission to write to the destination file.

```
# shutil_copyfile.py

import glob
import shutil

print('BEFORE:', glob.glob('shutil_copyfile.*'))

shutil.copyfile('shutil_copyfile.py', 'shutil_copyfile.py.copy')

print('AFTER:', glob.glob('shutil_copyfile.*'))
```

Because the function opens the input file for reading, regardless of its type, special files (such as Unix device nodes) cannot be copied as new special files with `copyfile()`.

```
$ python3 shutil_copyfile.py

BEFORE: ['shutil_copyfile.py']
AFTER: ['shutil_copyfile.py', 'shutil_copyfile.py.copy']
```

The implementation of `copyfile()` uses the lower-level function `copyfileobj()`. While the arguments to `copyfile()` are filenames, the arguments to `copyfileobj()` are open file handles. The optional third argument is a buffer length to use for reading in blocks.

```
# shutil_copyfileobj.py

import io
import os
import shutil
import sys


class VerboseStringIO(io.StringIO):

    def read(self, n=-1):
        next = io.StringIO.read(self, n)
        print('read({}) got {} bytes'.format(n, len(next)))
        return next


lorem_ipsum = '''Lorem ipsum dolor sit amet, consectetuer
adipiscing elit.  Vestibulum aliquam mollis dolor. Donec
vulputate nunc ut diam. Ut rutrum mi vel sem. Vestibulum
ante ipsum.'''

print('Default:')
input = VerboseStringIO(lorem_ipsum)
output = io.StringIO()
shutil.copyfileobj(input, output)

print()

print('All at once:')
```

```
print('All at once:')
input = VerboseStringIO(lorem_ipsum)
output = io.StringIO()
shutil.copyfileobj(input, output, -1)

print()

print('Blocks of 256:')
input = VerboseStringIO(lorem_ipsum)
output = io.StringIO()
shutil.copyfileobj(input, output, 256)
```

The default behavior is to read using large blocks. Use `-1` to read all of the input at one time or another positive integer to set a specific block size. This example uses several different block sizes to show the effect.

```
$ python3 shutil_copyfileobj.py

Default:
read(16384) got 166 bytes
read(16384) got 0 bytes

All at once:
read(-1) got 166 bytes
read(-1) got 0 bytes

Blocks of 256:
read(256) got 166 bytes
read(256) got 0 bytes
```

The `copy()` function interprets the output name like the Unix command line tool `cp`. If the named destination refers to a directory instead of a file, a new file is created in the directory using the base name of the source.

```
# shutil_copy.py

import glob
import os
import shutil

os.mkdir('example')
print('BEFORE:', glob.glob('example/*'))

shutil.copy('shutil_copy.py', 'example')

print('AFTER :', glob.glob('example/*'))
```

The permissions of the file are copied along with the contents.

```
$ python3 shutil_copy.py

BEFORE: []
AFTER : ['example/shutil_copy.py']
```

`copy2()` works like `copy()`, but includes the access and modification times in the metadata copied to the new file.

```
# shutil_copy2.py

import os
import shutil
import time


def show_file_info(filename):
    stat_info = os.stat(filename)
    print('  Mode    :', oct(stat_info.st_mode))
    print('  Created :', time.ctime(stat_info.st_ctime))
    print('  Accessed:', time.ctime(stat_info.st_atime))
    print('  Modified:', time.ctime(stat_info.st_mtime))


os.mkdir('example')
```

```
os.mkdir('example')
print('SOURCE:')
show_file_info('shutil_copy2.py')

shutil.copy2('shutil_copy2.py', 'example')

print('DEST:')
show_file_info('example/shutil_copy2.py')
```

The new file has all of the same characteristics as the old version.

```
$ python3 shutil_copy2.py

SOURCE:
  Mode    : 0o100644
  Created : Wed Dec 28 19:03:12 2016
  Accessed: Wed Dec 28 19:03:49 2016
  Modified: Wed Dec 28 19:03:12 2016
DEST:
  Mode    : 0o100644
  Created : Wed Dec 28 19:03:49 2016
  Accessed: Wed Dec 28 19:03:49 2016
  Modified: Wed Dec 28 19:03:12 2016
```

Copying File Metadata

By default when a new file is created under Unix, it receives permissions based on the umask of the current user. To copy the permissions from one file to another, use `copymode()`.

```
# shutil_copymode.py

import os
import shutil
import subprocess

with open('file_to_change.txt', 'wt') as f:
    f.write('content')
os.chmod('file_to_change.txt', 0o444)

print('BEFORE:', oct(os.stat('file_to_change.txt').st_mode))

shutil.copymode('shutil_copymode.py', 'file_to_change.txt')

print('AFTER :', oct(os.stat('file_to_change.txt').st_mode))
```

This example script creates a file to be modified, then uses `copymode()` to duplicate the permissions of the script to the example file.

```
$ python3 shutil_copymode.py

BEFORE: 0o100444
AFTER : 0o100644
```

To copy other metadata about the file use `copystat()`.

```
# shutil_copystat.py

import os
import shutil
import time


def show_file_info(filename):
    stat_info = os.stat(filename)
    print('  Mode    :', oct(stat_info.st_mode))
    print('  Created :', time.ctime(stat_info.st_ctime))
    print('  Accessed:', time.ctime(stat_info.st_atime))
    print('  Modified:', time.ctime(stat_info.st_mtime))
```

```
with open('file_to_change.txt', 'wt') as f:
    f.write('content')
os.chmod('file_to_change.txt', 0o444)

print('BEFORE:')
show_file_info('file_to_change.txt')

shutil.copystat('shutil_copystat.py', 'file_to_change.txt')

print('AFTER:')
show_file_info('file_to_change.txt')
```

Only the permissions and dates associated with the file are duplicated with `copystat()`.

```
$ python3 shutil_copystat.py

BEFORE:
  Mode    : 0o100444
  Created : Wed Dec 28 19:03:49 2016
  Accessed: Wed Dec 28 19:03:49 2016
  Modified: Wed Dec 28 19:03:49 2016
AFTER:
  Mode    : 0o100644
  Created : Wed Dec 28 19:03:49 2016
  Accessed: Wed Dec 28 19:03:49 2016
  Modified: Wed Dec 28 19:03:46 2016
```

Working With Directory Trees

`shutil` includes three functions for working with directory trees. To copy a directory from one place to another, use `copytree()`. It recurses through the source directory tree, copying files to the destination. The destination directory must not exist in advance.

```
# shutil_copytree.py
```

```
import glob
import pprint
import shutil

print('BEFORE:')
pprint.pprint(glob.glob('/tmp/example/*'))

shutil.copytree('../shutil', '/tmp/example')

print('\nAFTER:')
pprint.pprint(glob.glob('/tmp/example/*'))
```

The `symlinks` argument controls whether symbolic links are copied as links or as files. The default is to copy the contents to new files. If the option is true, new symlinks are created within the destination tree.

```
$ python3 shutil_copytree.py

BEFORE:
[]

AFTER:
['/tmp/example/example',
 '/tmp/example/example.out',
 '/tmp/example/file_to_change.txt',
 '/tmp/example/index.rst',
 '/tmp/example/shutil_copy.py',
 '/tmp/example/shutil_copy2.py',
 '/tmp/example/shutil_copyfile.py',
 '/tmp/example/shutil_copyfile.py.copy',
 '/tmp/example/shutil_copyfileobj.py',
 '/tmp/example/shutil_copymode.py',
 '/tmp/example/shutil_copystat.py',
 '/tmp/example/shutil_copytree.py',
```

```
 '/tmp/example/shutil_copytree_verbose.py',
 '/tmp/example/shutil_disk_usage.py',
 '/tmp/example/shutil_get_archive_formats.py',
 '/tmp/example/shutil_get_unpack_formats.py',
 '/tmp/example/shutil_make_archive.py',
 '/tmp/example/shutil_move.py',
 '/tmp/example/shutil_rmtree.py',
 '/tmp/example/shutil_unpack_archive.py',
 '/tmp/example/shutil_which.py',
 '/tmp/example/shutil_which_regular_file.py']
```

`copytree()` accepts two callable arguments to control its behavior. The `ignore` argument is called with the name of each directory or subdirectory being copied along with a list of the contents of the directory. It should return a list of items that should be copied. The `copy_function` argument is called to actually copy the file.

```
# shutil_copytree_verbose.py

import glob
import pprint
import shutil


def verbose_copy(src, dst):
    print('copying\n {!r}\n to {!r}'.format(src, dst))
    return shutil.copy2(src, dst)


print('BEFORE:')
pprint.pprint(glob.glob('/tmp/example/*'))
print()

shutil.copytree(
    '../shutil', '/tmp/example',
    copy_function=verbose_copy,
    ignore=shutil.ignore_patterns('*.py'),
)

print('\nAFTER:')
pprint.pprint(glob.glob('/tmp/example/*'))
```

In the example, `ignore_patterns()` is used to create an ignore function to skip copying Python source files. `verbose_copy()` prints the names of files as they are copied then uses `copy2()`, the default copy function, to make the copies.

```
$ python3 shutil_copytree_verbose.py

BEFORE:
[]

copying
 '../shutil/example.out'
 to '/tmp/example/example.out'
copying
 '../shutil/file_to_change.txt'
 to '/tmp/example/file_to_change.txt'
copying
 '../shutil/index.rst'
 to '/tmp/example/index.rst'

AFTER:
['/tmp/example/example',
 '/tmp/example/example.out',
 '/tmp/example/file_to_change.txt',
 '/tmp/example/index.rst']
```

To remove a directory and its contents, use `rmtree()`.

```
# shutil_rmtree.py

import glob
```

```
import pprint
import shutil

print('BEFORE:')
pprint.pprint(glob.glob('/tmp/example/*'))

shutil.rmtree('/tmp/example')

print('\nAFTER:')
pprint.pprint(glob.glob('/tmp/example/*'))
```

Errors are raised as exceptions by default, but can be ignored if the second argument is true, and a special error handler function can be provided in the third argument.

```
$ python3 shutil_rmtree.py

BEFORE:
['/tmp/example/example',
 '/tmp/example/example.out',
 '/tmp/example/file_to_change.txt',
 '/tmp/example/index.rst']

AFTER:
[]
```

To move a file or directory from one place to another, use `move()`.

```
# shutil_move.py

import glob
import shutil

with open('example.txt', 'wt') as f:
    f.write('contents')

print('BEFORE: ', glob.glob('example*'))

shutil.move('example.txt', 'example.out')

print('AFTER : ', glob.glob('example*'))
```

The semantics are similar to those of the Unix command `mv`. If the source and destination are within the same file system, the source is renamed. Otherwise the source is copied to the destination and then the source is removed.

```
$ python3 shutil_move.py

BEFORE:  ['example.txt']
AFTER :  ['example.out']
```

Finding Files

The `which()` function scans a search path looking for a named file. The typical use case is to find an executable program on the shell's search path defined in the environment variable `PATH`.

```
# shutil_which.py
```

```
import shutil

print(shutil.which('virtualenv'))
print(shutil.which('tox'))
print(shutil.which('no-such-program'))
```

If no file matching the search parameters can be found, `which()` returns `None`.

```
$ python3 shutil_which.py

/Users/dhellmann/Library/Python/3.5/bin/virtualenv
/Users/dhellmann/Library/Python/3.5/bin/tox
```

```
/usr/bin/shutil/lib/python3.5/bin/tox
None
```

`which()` takes arguments to filter based on the permissions the file has, and the search path to examine. The `path` argument defaults to `os.environ('PATH')`, but can be any string containing directory names separated by `os.pathsep`. The `mode` argument should be a bitmask matching the permissions of the file. By default the mask looks for executable files, but the following example uses a readable bitmask and an alternate search path to find a configuration file.

```
# shutil_which_regular_file.py

import os
import shutil

path = os.pathsep.join([
    '.',
    os.path.expanduser('~/pymotw'),
])

mode = os.F_OK | os.R_OK

filename = shutil.which(
    'config.ini',
    mode=mode,
    path=path,
)

print(filename)
```

There is still a race condition searching for readable files this way, because in the time between finding the file and actually trying to use it, the file can be deleted or its permissions can be changed.

```
$ touch config.ini
$ python3 shutil_which_regular_file.py

./config.ini
```

Archives

Python's standard library includes many modules for managing archive files such as `tarfile` and `zipfile`. There are also several higher-level functions for creating and extracting archives in `shutil`. `get_archive_formats()` returns a sequence of names and descriptions for formats supported on the current system.

```
# shutil_get_archive_formats.py

import shutil

for format, description in shutil.get_archive_formats():
    print('{:<5}: {}'.format(format, description))
```

The formats supported depend on which modules and underlying libraries are available, so the output for this example may change based on where it is run.

```
$ python3 shutil_get_archive_formats.py

bztar: bzip2'ed tar-file
gztar: gzip'ed tar-file
tar  : uncompressed tar file
xztar: xz'ed tar-file
zip  : ZIP file
```

Use `make_archive()` to create a new archive file. Its inputs are designed to best support archiving an entire directory and all of its contents, recursively. By default it uses the current working directory, so that all of the files and subdirectories appear at the top level of the archive. To change that behavior, use the `root_dir` argument to move to a new relative position on the filesystem and the `base_dir` argument to specify a directory to add to the archive.

```
# shutil_make_archive.py

import logging
```

```
import shutil
import sys
import tarfile

logging.basicConfig(
    format='%(message)s',
    stream=sys.stdout,
    level=logging.DEBUG,
)
logger = logging.getLogger('pymotw')

print('Creating archive:')
shutil.make_archive(
    'example', 'gztar',
    root_dir='..',
    base_dir='shutil',
    logger=logger,
)

print('\nArchive contents:')
with tarfile.open('example.tar.gz', 'r') as t:
    for n in t.getnames():
        print(n)
```

This example starts within the source directory for the examples for `shutil` and moves up one level in the file system, then adds the `shutil` directory to a tar archive compressed with gzip. The `logging` module is configured to show messages from `make_archive()` about what it is doing.

```
$ python3 shutil_make_archive.py

Creating archive:
changing into '..'
Creating tar archive
changing back to '...'

Archive contents:
shutil
shutil/config.ini
shutil/example.out
shutil/file_to_change.txt
shutil/index.rst
shutil/shutil_copy.py
shutil/shutil_copy2.py
shutil/shutil_copyfile.py
shutil/shutil_copyfileobj.py
shutil/shutil_copymode.py
shutil/shutil_copystat.py
shutil/shutil_copytree.py
shutil/shutil_copytree_verbose.py
shutil/shutil_disk_usage.py
shutil/shutil_get_archive_formats.py
shutil/shutil_get_unpack_formats.py
shutil/shutil_make_archive.py
shutil/shutil_move.py
shutil/shutil_rmtree.py
shutil/shutil_unpack_archive.py
shutil/shutil_which.py
shutil/shutil_which_regular_file.py
```

`shutil` maintains a registry of formats that can be unpacked on the current system, accessible via `get_unpack_formats()`.

```
# shutil_get_unpack_formats.py

import shutil

for format, exts, description in shutil.get_unpack_formats():
    print('{:<5}: {}, names ending in {}'.format(
        format, description, exts))
```

This registry is different from the registry for creating archives because it also includes common file extensions used for each

format so that the function for extracting an archive can guess which format to use based on the file extension.

```
$ python3 shutil_get_unpack_formats.py

bztar: bzip2'ed tar-file, names ending in ['.tar.bz2', '.tbz2']
gztar: gzip'ed tar-file, names ending in ['.tar.gz', '.tgz']
tar  : uncompressed tar file, names ending in ['.tar']
xztar: xz'ed tar-file, names ending in ['.tar.xz', '.txz']
zip  : ZIP file, names ending in ['.zip']
```

Extract the archive with `unpack_archive()`, passing the archive file name and optionally the directory where it should be extracted. If no directory is given, the current directory is used.

```
# shutil_unpack_archive.py

import pathlib
import shutil
import sys
import tempfile

with tempfile.TemporaryDirectory() as d:
    print('Unpacking archive:')
    shutil.unpack_archive(
        'example.tar.gz',
        extract_dir=d,
    )

    print('\nCreated:')
    prefix_len = len(d) + 1
    for extracted in pathlib.Path(d).rglob('*'):
        print(str(extracted)[prefix_len:])
```

In this example `unpack_archive()` is able to determine the format of the archive because the filename ends with `tar.gz`, and that value is associated with the `gztar` format in the unpack format registry.

```
$ python3 shutil_unpack_archive.py

Unpacking archive:

Created:
shutil
shutil/config.ini
shutil/example.out
shutil/file_to_change.txt
shutil/index.rst
shutil/shutil_copy.py
shutil/shutil_copy2.py
shutil/shutil_copyfile.py
shutil/shutil_copyfileobj.py
shutil/shutil_copymode.py
shutil/shutil_copystat.py
shutil/shutil_copytree.py
shutil/shutil_copytree_verbose.py
shutil/shutil_disk_usage.py
shutil/shutil_get_archive_formats.py
shutil/shutil_get_unpack_formats.py
shutil/shutil_make_archive.py
shutil/shutil_move.py
shutil/shutil_rmtree.py
shutil/shutil_unpack_archive.py
shutil/shutil_which.py
shutil/shutil_which_regular_file.py
```

File System Space

It can be useful to examine the local file system to see how much space is available before performing a long running operation that may exhaust that space. `disk_usage()` returns a tuple with the total space, the amount currently being used, and the amount remaining free.

```
# shutil_disk_usage.py

import shutil

total_b, used_b, free_b = shutil.disk_usage('.')

gib = 2 ** 30  # GiB == gibibyte
gb = 10 ** 9   # GB == gigabyte

print('Total: {:6.2f} GB  {:6.2f} GiB'.format(
    total_b / gb, total_b / gib))
print('Used : {:6.2f} GB  {:6.2f} GiB'.format(
    used_b / gb, used_b / gib))
print('Free : {:6.2f} GB  {:6.2f} GiB'.format(
    free_b / gb, free_b / gib))
```

The values returned by `disk_usage()` are the number of bytes, so the example program converts them to more readable units before printing them.

```
$ python3 shutil_disk_usage.py

Total: 499.42 GB  465.12 GiB
Used : 246.68 GB  229.73 GiB
Free : 252.48 GB  235.14 GiB
```

See also

- Standard library documentation for shutil
- Data Compression and Archiving – Modules for dealing with archive and compression formats.

tempfile — Temporary File System Objects

filecmp — Compare Files

Quick Links

Copying Files
Copying File Metadata
Working With Directory Trees
Finding Files
Archives
File System Space

This page was last updated 2016-12-28.

Navigation

tempfile — Temporary File System Objects
filecmp — Compare Files

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?