tarfile — Tar Archive Access

Purpose: Tar archive access.

The `tarfile` module provides read and write access to Unix tar archives, including compressed files. In addition to the POSIX standards, several GNU tar extensions are supported. Unix special file types such as hard and soft links, and device nodes are also handled.

> **Note**
>
> Although `tarfile` implements a Unix format, it can be used to create and read tar archives under Microsoft Windows, too.

Testing Tar Files

The `is_tarfile()` function returns a boolean indicating whether or not the filename passed as an argument refers to a valid tar archive.

```
# tarfile_is_tarfile.py

import tarfile

for filename in ['README.txt', 'example.tar',
                 'bad_example.tar', 'notthere.tar']:
    try:
        print('{:>15}  {}'.format(filename, tarfile.is_tarfile(
            filename)))
    except IOError as err:
        print('{:>15}  {}'.format(filename, err))
```

If the file does not exist, `is_tarfile()` raises an `IOError`.

```
$ python3 tarfile_is_tarfile.py

     README.txt  False
    example.tar  True
bad_example.tar  False
   notthere.tar  [Errno 2] No such file or directory:
'notthere.tar'
```

Reading Metadata from an Archive

Use the `TarFile` class to work directly with a tar archive. It supports methods for reading data about existing archives as well as modifying the archives by adding additional files.

To read the names of the files in an existing archive, use `getnames()`.

```
# tarfile_getnames.py

import tarfile

with tarfile.open('example.tar', 'r') as t:
    print(t.getnames())
```

The return value is a list of strings with the names of the archive contents.

```
$ python3 tarfile_getnames.py

['index.rst', 'README.txt']
```

In addition to names, metadata about the archive members is available as instances of `TarInfo` objects.

```
# tarfile_getmembers.py

import tarfile
import time

with tarfile.open('example.tar', 'r') as t:
    for member_info in t.getmembers():
        print(member_info.name)
        print('  Modified:', time.ctime(member_info.mtime))
        print('  Mode    :', oct(member_info.mode))
        print('  Type    :', member_info.type)
        print('  Size    :', member_info.size, 'bytes')
        print()
```

Load the metadata via `getmembers()` and `getmember()`.

```
$ python3 tarfile_getmembers.py

index.rst
  Modified: Fri Aug 19 16:27:54 2016
  Mode    : 0o644
  Type    : b'0'
  Size    : 9878 bytes

README.txt
  Modified: Fri Aug 19 16:27:54 2016
  Mode    : 0o644
  Type    : b'0'
  Size    : 75 bytes
```

If the name of the archive member is known in advance, its `TarInfo` object can be retrieved with `getmember()`.

```
# tarfile_getmember.py

import tarfile
import time

with tarfile.open('example.tar', 'r') as t:
    for filename in ['README.txt', 'notthere.txt']:
        try:
            info = t.getmember(filename)
        except KeyError:
            print('ERROR: Did not find {} in tar archive'.format(
                filename))
        else:
            print('{} is {:d} bytes'.format(
                info.name, info.size))
```

If the archive member is not present, `getmember()` raises a `KeyError`.

```
$ python3 tarfile_getmember.py

README.txt is 75 bytes
ERROR: Did not find notthere.txt in tar archive
```

Extracting Files from an Archive

To access the data from an archive member within a program, use the `extractfile()` method, passing the member's name.

```
# tarfile_extractfile.py

import tarfile

with tarfile.open('example.tar', 'r') as t:
    for filename in ['README.txt', 'notthere.txt']:
```

```
        try:
            f = t.extractfile(filename)
        except KeyError:
            print('ERROR: Did not find {} in tar archive'.format(
                filename))
        else:
            print(filename, ':')
            print(f.read().decode('utf-8'))
```

The return value is a file-like object from which the contents of the archive member can be read.

```
$ python3 tarfile_extractfile.py

README.txt :
The examples for the tarfile module use this file and
example.tar as data.

ERROR: Did not find notthere.txt in tar archive
```

To unpack the archive and write the files to the file system, use `extract()` or `extractall()` instead.

```
# tarfile_extract.py

import tarfile
import os

os.mkdir('outdir')
with tarfile.open('example.tar', 'r') as t:
    t.extract('README.txt', 'outdir')
print(os.listdir('outdir'))
```

The member or members are read from the archive and written to the file system, starting in the directory named in the arguments.

```
$ python3 tarfile_extract.py

['README.txt']
```

The standard library documentation includes a note stating that `extractall()` is safer than `extract()`, especially for working with streaming data where rewinding to read an earlier part of the input is not possible, and it should be used in most cases.

```
# tarfile_extractall.py

import tarfile
import os

os.mkdir('outdir')
with tarfile.open('example.tar', 'r') as t:
    t.extractall('outdir')
print(os.listdir('outdir'))
```

With `extractall()`, the first argument is the name of the directory where the files should be written.

```
$ python3 tarfile_extractall.py

['README.txt', 'index.rst']
```

To extract specific files from the archive, pass their names or `TarInfo` metadata containers to `extractall()`.

```
# tarfile_extractall_members.py

import tarfile
import os

os.mkdir('outdir')
with tarfile.open('example.tar', 'r') as t:
    t.extractall('outdir',
                 members=[t.getmember('README.txt')],
```

```
                    )
print(os.listdir('outdir'))
```

When a `members` list is provided, only the named files are extracted.

```
$ python3 tarfile_extractall_members.py

['README.txt']
```

Creating New Archives

To create a new archive, open the `TarFile` with a mode of `'w'`.

```
# tarfile_add.py

import tarfile

print('creating archive')
with tarfile.open('tarfile_add.tar', mode='w') as out:
    print('adding README.txt')
    out.add('README.txt')

print()
print('Contents:')
with tarfile.open('tarfile_add.tar', mode='r') as t:
    for member_info in t.getmembers():
        print(member_info.name)
```

Any existing file is truncated and a new archive is started. To add files, use the `add()` method.

```
$ python3 tarfile_add.py

creating archive
adding README.txt

Contents:
README.txt
```

Using Alternate Archive Member Names

It is possible to add a file to an archive using a name other than the original filename by constructing a `TarInfo` object with an alternate `arcname` and passing it to `addfile()`.

```
# tarfile_addfile.py

import tarfile

print('creating archive')
with tarfile.open('tarfile_addfile.tar', mode='w') as out:
    print('adding README.txt as RENAMED.txt')
    info = out.gettarinfo('README.txt', arcname='RENAMED.txt')
    out.addfile(info)

print()
print('Contents:')
with tarfile.open('tarfile_addfile.tar', mode='r') as t:
    for member_info in t.getmembers():
        print(member_info.name)
```

The archive includes only the changed filename:

```
$ python3 tarfile_addfile.py

creating archive
adding README.txt as RENAMED.txt

Contents:
```

```
RENAMED.txt
```

Writing Data from Sources Other Than Files

Sometimes it is necessary to write data into an archive directly from memory. Rather than writing the data to a file, then adding that file to the archive, you can use `addfile()` to add data from an open file-like handle that returns bytes.

```
# tarfile_addfile_string.py

import io
import tarfile

text = 'This is the data to write to the archive.'
data = text.encode('utf-8')

with tarfile.open('addfile_string.tar', mode='w') as out:
    info = tarfile.TarInfo('made_up_file.txt')
    info.size = len(data)
    out.addfile(info, io.BytesIO(data))

print('Contents:')
with tarfile.open('addfile_string.tar', mode='r') as t:
    for member_info in t.getmembers():
        print(member_info.name)
        f = t.extractfile(member_info)
        print(f.read().decode('utf-8'))
```

By first constructing a `TarInfo` object, the archive member can be given any name desired. After setting the size, the data is written to the archive using `addfile()` and a `BytesIO` buffer as a source of the data.

```
$ python3 tarfile_addfile_string.py

Contents:
made_up_file.txt
This is the data to write to the archive.
```

Appending to Archives

In addition to creating new archives, it is possible to append to an existing file by using mode `'a'`.

```
# tarfile_append.py

import tarfile

print('creating archive')
with tarfile.open('tarfile_append.tar', mode='w') as out:
    out.add('README.txt')

print('contents:',)
with tarfile.open('tarfile_append.tar', mode='r') as t:
    print([m.name for m in t.getmembers()])

print('adding index.rst')
with tarfile.open('tarfile_append.tar', mode='a') as out:
    out.add('index.rst')

print('contents:',)
with tarfile.open('tarfile_append.tar', mode='r') as t:
    print([m.name for m in t.getmembers()])
```

The resulting archive ends up with two members:

```
$ python3 tarfile_append.py

creating archive
contents:
['README.txt']
```

```
adding index.rst
contents:
['README.txt', 'index.rst']
```

Working with Compressed Archives

Besides regular tar archive files, the `tarfile` module can work with archives compressed via the gzip or bzip2 protocols. To open a compressed archive, modify the mode string passed to `open()` to include `":gz"` or `":bz2"`, depending on the desired compression method.

```
# tarfile_compression.py

import tarfile
import os

fmt = '{:<30} {:<10}'
print(fmt.format('FILENAME', 'SIZE'))
print(fmt.format('README.txt', os.stat('README.txt').st_size))

FILES = [
    ('tarfile_compression.tar', 'w'),
    ('tarfile_compression.tar.gz', 'w:gz'),
    ('tarfile_compression.tar.bz2', 'w:bz2'),
]

for filename, write_mode in FILES:
    with tarfile.open(filename, mode=write_mode) as out:
        out.add('README.txt')

    print(fmt.format(filename, os.stat(filename).st_size),
          end=' ')
    print([
        m.name
        for m in tarfile.open(filename, 'r:*').getmembers()
    ])
```

When opening an existing archive for reading, specify `"r:*"` to have `tarfile` determine the compression method to use automatically.

```
$ python3 tarfile_compression.py

FILENAME                       SIZE
README.txt                     75
tarfile_compression.tar        10240      ['README.txt']
tarfile_compression.tar.gz     213        ['README.txt']
tarfile_compression.tar.bz2    199        ['README.txt']
```

See also

- Standard library documentation for tarfile
- GNU tar manual – Documentation of the tar format, including extensions.
- `zipfile` – Similar access for ZIP archives.
- `gzip` – GNU zip compression
- `bz2` – bzip2 compression

bz2 — bzip2 Compression

zipfile — ZIP Archive Access

This page was last updated 2016-12-29.

Navigation
bz2 — bzip2 Compression
zipfile — ZIP Archive Access

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?