gzip — Read and Write GNU zip Files

Purpose: Read and write gzip files.

The gzip module provides a file-like interface to GNU zip files, using zlib to compress and uncompress the data.

Writing Compressed Files

The module-level function open() creates an instance of the file-like class GzipFile. The usual methods for writing and reading bytes are provided.

```
# gzip_write.py

import gzip
import io
import os

outfilename = 'example.txt.gz'
with gzip.open(outfilename, 'wb') as output:
    with io.TextIOWrapper(output, encoding='utf-8') as enc:
        enc.write('Contents of the example file go here.\n')

print(outfilename, 'contains', os.stat(outfilename).st_size,
      'bytes')
os.system('file -b --mime {}'.format(outfilename))
```

To write data into a compressed file, open the file with mode 'wb'. This example wraps the GzipFile with a TextIOWrapper from the io module to encode Unicode text to bytes suitable for compression.

```
$ python3 gzip_write.py

application/x-gzip; charset=binary
example.txt.gz contains 75 bytes
```

Different amounts of compression can be used by passing a compresslevel argument. Valid values range from 0 to 9, inclusive. Lower values are faster and result in less compression. Higher values are slower and compress more, up to a point.

```
# gzip_compresslevel.py

import gzip
import io
import os
import hashlib


def get_hash(data):
    return hashlib.md5(data).hexdigest()


data = open('lorem.txt', 'r').read() * 1024
cksum = get_hash(data.encode('utf-8'))


print('Level  Size        Checksum')
print('-----  ----------  ---------------------------------')
print('data   {:>10}  {}'.format(len(data), cksum))

for i in range(0, 10):
    filename = 'compress-level-{}.gz'.format(i)
    with gzip.open(filename, 'wb', compresslevel=i) as output:
        with io.TextIOWrapper(output, encoding='utf-8') as enc:
            enc.write(data)
    size = os.stat(filename).st_size
    cksum = get_hash(open(filename, 'rb').read())
    print('{:>5d}  {:>10d}  {}'.format(i, size, cksum))
```

The center column of numbers in the output shows the size in bytes of the files produced by compressing the input. For this input data, the higher compression values do not necessarily pay off in decreased storage space. Results will vary, depending on the input data.

```
$ python3 gzip_compresslevel.py

Level  Size        Checksum
-----  ----------  ---------------------------------
data       754688  e4c0f9433723971563f08a458715119c
    0      754793  ced7189c324eb73a8388492a9024d391
    1        9846  5356d357f23e0d5b6d85e920929f0e43
    2        8267  8ce46bce238edc095e47e941cebad93d
    3        8227  91662517459db94a744671a6b4295b67
    4        4167  ad304e3aec585640de9f14306fb32083
    5        4167  4381a5d6dff4dd2746387f20411dcfcd
    6        4167  ef3a05112ea382abb53bc4a5bee3a52a
    7        4167  4723a253d1dc8ddecd4ff7b7adf0bc0b
    8        4167  0e1aeba7bdc39f0007039f130d9a28b2
    9        4167  eccf47c4c4f1cca3274e57a1b9b9ddd2
```

A GzipFile instance also includes a `writelines()` method that can be used to write a sequence of strings.

```
# gzip_writelines.py

import gzip
import io
import itertools
import os

with gzip.open('example_lines.txt.gz', 'wb') as output:
    with io.TextIOWrapper(output, encoding='utf-8') as enc:
        enc.writelines(
            itertools.repeat('The same line, over and over.\n',
                             10)
        )

os.system('gzcat example_lines.txt.gz')
```

As with a regular file, the input lines need to include a newline character.

```
$ python3 gzip_writelines.py

The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
The same line, over and over.
```

Reading Compressed Data

To read data back from previously compressed files, open the file with binary read mode (`'rb'`) so no text-based translation of line endings or Unicode decoding is performed.

```
# gzip_read.py

import gzip
import io

with gzip.open('example.txt.gz', 'rb') as input_file:
    with io.TextIOWrapper(input_file, encoding='utf-8') as dec:
        print(dec.read())
```

This example reads the file written by gzip_write.py from the previous section, using a TextIOWrapper to decode the text after it is decompressed.

```
$ python3 gzip_read.py

Contents of the example file go here.
```

While reading a file, it is also possible to seek and read only part of the data.

```
# gzip_seek.py

import gzip

with gzip.open('example.txt.gz', 'rb') as input_file:
    print('Entire file:')
    all_data = input_file.read()
    print(all_data)

    expected = all_data[5:15]

    # rewind to beginning
    input_file.seek(0)

    # move ahead 5 bytes
    input_file.seek(5)
    print('Starting at position 5 for 10 bytes:')
    partial = input_file.read(10)
    print(partial)

    print()
    print(expected == partial)
```

The seek() position is relative to the *uncompressed* data, so the caller does not need to know that the data file is compressed.

```
$ python3 gzip_seek.py

Entire file:
b'Contents of the example file go here.\n'
Starting at position 5 for 10 bytes:
b'nts of the'

True
```

Working with Streams

The GzipFile class can be used to wrap other types of data streams so they can use compression as well. This is useful when the data is being transmitted over a socket or an existing (already open) file handle. A BytesIO buffer can also be used.

```
# gzip_BytesIO.py

import gzip
from io import BytesIO
import binascii

uncompressed_data = b'The same line, over and over.\n' * 10
print('UNCOMPRESSED:', len(uncompressed_data))
print(uncompressed_data)

buf = BytesIO()
with gzip.GzipFile(mode='wb', fileobj=buf) as f:
    f.write(uncompressed_data)

compressed_data = buf.getvalue()
print('COMPRESSED:', len(compressed_data))
print(binascii.hexlify(compressed_data))
```

```
inbuffer = BytesIO(compressed_data)
with gzip.GzipFile(mode='rb', fileobj=inbuffer) as f:
    reread_data = f.read(len(uncompressed_data))

print('\nREREAD:', len(reread_data))
print(reread_data)
```

One benefit of using `GzipFile` over `zlib` is that it supports the file API. However, when re-reading the previously compressed data, an explicit length is passed to `read()`. Leaving the length off resulted in a CRC error, possibly because `BytesIO` returned an empty string before reporting EOF. When working with streams of compressed data, either prefix the data with an integer representing the actual amount of data to be read or use the incremental decompression API in `zlib`.

```
$ python3 gzip_BytesIO.py

UNCOMPRESSED: 300
b'The same line, over and over.\nThe same line, over and over.\nT
he same line, over and over.\nThe same line, over and over.\nThe
same line, over and over.\nThe same line, over and over.\nThe sam
e line, over and over.\nThe same line, over and over.\nThe same l
ine, over and over.\nThe same line, over and over.\n'
COMPRESSED: 51
b'1f8b080022caae5a02ff0bc94855284ecc4d55c8c9cc4bd551c82f4b2d5248c
c4b0133f4b8424665916401d3e717802c010000'

REREAD: 300
b'The same line, over and over.\nThe same line, over and over.\nT
he same line, over and over.\nThe same line, over and over.\nThe
same line, over and over.\nThe same line, over and over.\nThe sam
e line, over and over.\nThe same line, over and over.\nThe same l
ine, over and over.\nThe same line, over and over.\n'
```

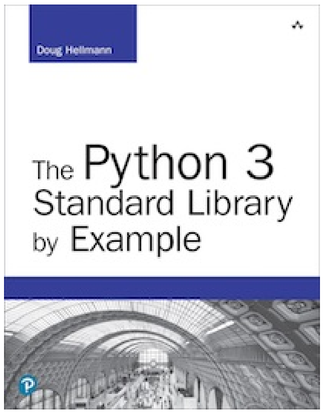
See also

- Standard library documentation for gzip
- `zlib` – The `zlib` module is a lower-level interface to gzip compression.
- `zipfile` – The `zipfile` module gives access to ZIP archives.
- `bz2` – The bz2 module uses the bzip2 compression format.
- `tarfile` – The `tarfile` module includes built-in support for reading compressed tar archives.
- `io` – Building-blocks for creating input and output pipelines.

zlib — GNU zlib Compression

bz2 — bzip2 Compression

This page was last updated 2018-03-18.



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?