Сжатие и архивирование файлов на Python

Стандартная библиотека Python предоставляет отличные модули и инструменты практически для любой задачи, о которой можно только подумать. Модули для работы со сжатыми файлами (архивами) не являются исключением. Будь то основные контейнеры, такие как tar и zip, или специальные инструменты или форматы, такие как gzip и bz2, или даже более экзотические форматы, такие как lzma, в Python есть все. Итак, чтобы ориентироваться во всех доступных вариантах, в этом материале рассмотрим модули, позволяющие работать с архивами, а так же узнаем, как сжимать, распаковывать, проверять, тестировать и защищать архивы всех видов форматов с помощью стандартной библиотеки Python.

Модули Python, позволяющие работать с архивами.

- модуль zlib в Python предоставляет собой код для работы с форматом сжатия и распаковки методом deflate, который используется такими форматами архивов, как zip, gzip и многими другими. Этот модуль, по сути, использует алгоритм сжатия, совместимый с gzip, только без удобной оболочки.
- модуль bz2 обеспечивает поддержку сжатия bzip2. Этот алгоритм, как правило, более эффективен, чем метод deflate, но может быть медленнее. Он работает только с отдельными файлами и поэтому не может создавать архивы директорий.
- модуль lzma представляет собой также название алгоритма. Он может обеспечить более высокую степень сжатия, чем некоторые старые методы, и является алгоритмом, лежащим в основе утилиты xz (точнее, LZMA2).
- модуль gzip использует уже упомянутый алгоритм сжатия zlib и служит интерфейсом, аналогичным утилитам gzip и gunzip.
- модуль shutil предоставляет служебные методы для работы с архивами и может быть удобным способом создания архивов tar, gztar, zip, bztar или xztar.
- модуль zipfile позволяет работать с zip-архивами в Python. Этот модуль предоставляет все ожидаемые методы для создания, чтения, записи или добавления в ZIP-файлы, а также классы и объекты для упрощения работы с такими файлами.
- модуль tarfile используется для работы с tar-архивами. Он может читать и записывать файлы или архивы gzip, bz2 и lzma. Он также поддерживает другие функции, которые знакомы из утилиты tar.

Низкоуровневый модуль zlib.

Модуль zlib довольно низкоуровневая библиотека, и следовательно она может использоваться не так часто, поэтому ра[Вверх]м базовое сжатие/распаковку всего файла сразу:

```python
import zlib, sys

filename_in = "data"
filename_out = "compressed_data"

with open(filename_in, mode="rb") as fin, open(filename_out, mode="wb") as fout:
    data = fin.read()
    compressed_data = zlib.compress(data, zlib.Z_BEST_COMPRESSION)
    print(f"Оригинальный размер: {sys.getsizeof(data)}")
    # Оригинальный размер: 1000033
    print(f"Сжатый размер: {sys.getsizeof(compressed_data)}")
    # Сжатый размер: 1024

    fout.write(compressed_data)

with open(filename_out, mode="rb") as fin:
    data = fin.read()
    compressed_data = zlib.decompress(data)
    print(f"Сжатый размер: {sys.getsizeof(data)}")
    # Сжатый размер: 1024
    print(f"Распакованный размер: {sys.getsizeof(compressed_data)}")
    # Распакованный размер: 1000033
```

В приведенном выше коде, используется входной файл, который был сгенерирован с данными `head -c 1MB /dev/zero > data`, что выдает 1 МБ нулей. Далее открываем и считываем этот файл в память, а затем используем функцию сжатия для создания сжатых данных. После эти данные записываются в выходной файл. Чтобы продемонстрировать восстановление данных, снова открываем сжатый файл и используем для него функцию распаковки. При выводе на печать видим, что размеры как сжатых, так и распакованных данных совпадают.

Низкоуровневый модуль bz2.

Следующий модуль bz2, также предоставляет одноименный формат сжатия bz2. Модуль используется очень похоже на описанный выше модуль zlib:

```python
import bz2, os, sys

filename_in = "data"
filename_out = "compressed_data.bz2"

with open(filename_in, mode="rb") as fin, bz2.open(filename_out, "wb") as fout:
    fout.write(fin.read())

print(f"Несжатый размер: {os.stat(filename_in).st_size}")
# Несжатый размер: 1000000
print(f"Сжатый размер: {os.stat(filename_out).st_size}")
# Сжатый размер: 48

with bz2.open(filename_out, "rb") as fout:
    data = fout.read()
    print(f"Несжатый размер: {sys.getsizeof(data)}")
    # Несжатый размер: 1000033
```

Интерфейс этих двух модулей в значительной степени идентичен, поэтому, в приведенном выше примере этап сжатия был сокращен практически до одной строки, а так же использовался os.stat для проверки размера файлов.

Низкоуровневый модуль lzma.

Последним из этих низкоуровневых модулей является lzma, и чтобы не показывать один и тот же код снова, выполним сжатие фрагментами, по 1024 байта. Такой способ может пригодится, если файл очень большой и целиком не помещается в оперативную память.

```python
import lzma, os
lzc = lzma.LZMACompressor()

# cat /usr/share/dict/words | sort -R | head -c 1MB > data
filename_in = "data"
filename_out = "compressed_data.xz"
```

Вверх

```
with open(filename_in, mode="r") as fin, open(filename_out, "wb") as fout:
    for chunk in fin.read(1024):
        compressed_chunk = lzc.compress(chunk.encode("ascii"))
        fout.write(compressed_chunk)
    fout.write(lzc.flush())

print(f"Несжатый размер: {os.stat(filename_in).st_size}")
# Несжатый размер: 972398
print(f"Сжатый размер: {os.stat(filename_out).st_size}")
# Сжатый размер: 736

with lzma.open(filename_out, "r") as fin:
    words = fin.read().decode("utf-8").split()
    print(words[:5])

# ['erroneous', 'dormice', 'bothers', "pediment's", 'Mendocino']
```

Начинаем с создания исходного файла, состоящего из слов, извлеченных из словаря, расположенного в /usr/share/dict/words. Это делается для того, чтобы потом можно было фактически подтвердить, что распакованные данные идентичны оригиналу.

Затем открываем исходный и выходной файлы, как в предыдущих примерах. На этот раз, читаем файл фрагментами по 1024 бита и сжимаем их с помощью метода LZMACompressor.compress(). Затем каждый сжатый фрагмент записывается (добавляется) в выходной файл. После того, как исходный файл прочитан до конца, необходимо вызвать метод lzc.flush(), чтобы завершить процесс сжатия и скинуть все оставшиеся данные в выходной файл.

Для проверки, открываем и распаковываем файл обычным способом и выводим первые 5 слов.

Использование модуль gzip совместно с shutil.

Переходим к модулям более высокого уровня и теперь будем использовать [модуль gzip] для тех же задач. Пример того, как применение модуля shutil может облегчить архивирования больших файлов.

```
import os, sys, shutil, gzip

filename_in = "data"
filename_out = "compressed_data.tar.gz"

# обратите внимание как открывается выходной файл `gzip.open()`
with open(filename_in, "rb") as fin, gzip.open(filename_out, "wb") as fout:
    # Читает файл по частям, экономя оперативную память
    shutil.copyfileobj(fin, fout)

print(f"Несжатый размер: {os.stat(filename_in).st_size}")
# Несжатый размер: 1000000
print(f"Сжатый размер: {os.stat(filename_out).st_size}")
# Сжатый размер: 1023

with gzip.open(filename_out, "rb") as fin:
    data = fin.read()
    print(f"Несжатый размер: {sys.getsizeof(data)}")
    # Несжатый размер: 1000033
```

В этом примере были объединены использование модулей gzip, и shutils. Может показаться, что здесь делается такое же сжатие, что и в примерах модулей zlib или bz2, но благодаря функции модуля [shutil.copyfileobj], получаем пошаговое инкрементное сжатие без необходимости перебирать данные в цикле, как это делалось с lzma.

Одним из преимуществ модуля gzip является то, что он предоставляет интерфейс командной строки, и это не gzip и gunzip в Linux, а интеграция с Python:

```
$ python3 -m gzip -h
# usage: gzip.py [-h] [--fast | --best | -d] [file [file ...]]
# ...
# optional arguments:
#   -h, --help        show this help message and exit
#   --fast            compress faster
#   --best            compress better
#   -d, --decompress  act like gunzip instead of gzip
$ python3 -m gzip --fast data
```

```
$ ls -l data*
# -rw-rw-r-- 1 main main 972398 янв 17 09:49 data
# -rw-rw-r-- 1 main main 463710 янв 17 10:12 data.gz
```

Высокоуровневый модуль zipfile.

Помимо основных операций сжатия/распаковки, модуль zipfile включают некоторые другие служебные методы, такие как проверка контрольных сумм, использование паролей или просмотр файлов в архивах.

```python
import zipfile

# shuf -n5 /usr/share/dict/words > words.txt
files = ["words1.txt", "words2.txt", "words3.txt", "words4.txt", "words5.txt"]
archive = "archive.zip"
password = b"secret"

with zipfile.ZipFile(archive, "w") as zf:
    for file in files:
        zf.write(file)

    zf.setpassword(password)

with zipfile.ZipFile(archive, "r") as zf:
    crc_test = zf.testzip()
    if crc_test is not None:
        print(f"Неверный CRC или заголовки файлов: {crc_test}")

    info = zf.infolist()  # also zf.namelist()
    print(info)
    # [ <ZipInfo filename='words1.txt' filemode='-rw-r--r--' file_size=37>,
    #   <ZipInfo filename='words2.txt' filemode='-rw-r--r--' file_size=47>,
    #   ... ]

    file = info[0]
    with zf.open(file) as f:
        print(f.read().decode())
        # Olav
        # teakettles
        # ...

    # так же, попробуйте zf.extractall()
    zf.extract(file, "/tmp", pwd=password)
```

Это довольно длинный фрагмент кода, но он охватывает все важные функции модуля zipfile. Код начинается с создания ZIP-архива при помощи контекстного менеджера ZipFile в режиме записи - 'w', а затем, в него добавляются файлы. Можно заметить, что при добавлении файлов в архив, последние не открываются на чтение, а просто передается имя файла в метод ZipFile.write().

После добавления файлов, устанавливаем пароль архива, используя метод ZipFile.setpassword.

Далее, для подтверждения того что сжатие работает, открываем архив. Перед чтением любых файлов, проверяем CRC и заголовки файлов, после чего извлекаем информацию обо всех файлах, присутствующих в архиве. В этом примере просто выведем список объектов zipfile.ZipInfo. Используя атрибуты этого объекта, можно получить CRC, размер, тип сжатия и т. д.

После проверки файлов, откроем и прочитаем один из них. Очевидно, что он имеет ожидаемое содержимое, поэтому продолжим извлекать его в файл, указанный по пути /tmp/.

Помимо создания архивов, модуль zipfile позволяет также добавлять файлы в существующие архивы. Для этого нужно изменить режим открытия архива на "*добавить*" - 'a':

```python
with zipfile.ZipFile(archive, "a") as zf:
    zf.write("words6.txt")
    print(zf.namelist())

# ['words1.txt', 'words2.txt', 'words3.txt', 'words4.txt', 'words5.txt', 'words6.txt']
```

Вверх

Как и в случае с модулем gzip, модуль zipfile в Python имеет интерфейс командной строки. Для выполнения базового архивирования и извлечения используйте следующее:

```
# создание архива
$python3 -m zipfile -c arch.zip words1.txt words2.txt
# тестирование архива
$python3 -m zipfile -t arch.zip
# Done testing

# Извлечение файлов из архива
python3 -m zipfile -e arch.zip /tmp
ls /tmp/words*
# /tmp/words1.txt  /tmp/words2.txt
```

Высокоуровневый модуль tarfile.

И последнее, но не менее важное: модуль tarfile. Этот модуль похож на zipfile, но также реализует некоторые дополнительные функции:

```
import tarfile

files = ["words1.txt", "words2.txt", "words3.txt", "words4.txt"]
archive = "archive.tar.gz"

with tarfile.open(archive, "w:gz") as tar:
    for file in files:
        tar.add(file)

    print(f"Архив содержит: {tar.getmembers()}")
    # [<TarInfo 'words1.txt' at 0x7f71ed74f8e0>,
    #  <TarInfo 'words2.txt' at 0x7f71ed74f9a8>
    #  ... ]

    info = tar.gettarinfo("words1.txt")
    print(f"{tar.name} содержит {info.name} с разрешениями {oct(info.mode)[-3:]}, \
         размер: {info.size} и владелец: {info.uid}:{info.gid}")
    # .../archive.tar contains words1.txt with permissions 644, size: 37 and owner: 500:500

    def change_permissions(tarinfo):
        tarinfo.mode = 0o100600  # -rw-------.
        return tarinfo

    tar.add("words5.txt", filter=change_permissions)

    tar.list()
    # -rw-r--r-- main/main   37 2021-08-23 09:01:56 words1.txt
    # -rw-r--r-- main/main   47 2021-08-23 09:02:06 words2.txt
    # ...
    # -rw------- main/main   42 2021-08-23 09:02:22 words5.txt
```

Код примера начинается с простого создания архива, но здесь используется режим доступа 'w:gz', который указывает, что необходимо использовать сжатие GZ. После этого добавляются все файлы в архив. С помощью модуля tarfile можно передавать, например, символические ссылки или целые каталоги, которые будут добавляться рекурсивно.

Далее, чтобы проверить, что все файлы действительно присутствуют в архиве, используется метод TarFile.getmembers(). Чтобы получить представление об отдельных файлах, можно использовать метод TarFile.gettarinfo(), который предоставит все атрибуты файлов Linux.

Модуль tarfile имеет одну интересную функцию, которой нет в других модулях, а именно возможность изменять атрибуты файлов, когда они добавляются в архив. В приведенном выше фрагменте кода меняется разрешение файла, предоставляя аргумент filter, который изменяет TarInfo.mode. Это значение должно быть представлено в виде восьмеричного числа, здесь 0o100600 устанавливает разрешения на 0600 или -rw-------.

Последнее, что нужно сделать с tar-архивом, это открыть его и распаковать. Для этого открываем его в режиме 'r:gz', извлекаем информационный объект (member) по имени файла, проверяем, действительно ли это файл, и извлекаем его в нужное ме

Вверх

```
with tarfile.open(archive, "r:gz") as tar:
    member = tar.getmember("words3.txt")
    if member.isfile():
        tar.extract(member, "/tmp/")
```