gzip_static

Release 0.1.0

Ruben Vorderman

Nov 04, 2021

CONTENTS:

CHAPTER

ONE

INSTALLATION

gzip_static can be installed with `pip install gzip_static`. There are no dependencies by default.

The following packages can be installed to enhance the functionality of gzip_static.

- zopfli adds zopfli compression to gzip_static. Zopfli compressed files can be decompressed with any gzip-compatible tool and the compressed size is a few percent smaller than files compressed with gzip level 9. This comes with much increased compression time (~125x increase or thereabouts depending on the website). This works great for files that aren't changed much or at all but always downloaded like stylesheets.
- xxhash speeds up the checksumming process. This makes gzip-static about 28% faster when running on a website folder with all the gzip files up to date.
- isal speeds up the decompression of gzip files during the checksumming process. This makes gzip-static about 66% faster when running on a website folder with all the gzip files up to date. Isal is only available on 64-bit platforms.

Together xxhash and isal decrease the runtime of checksumming by about 60%, so it is about 2.5 times faster.

These dependencies are all optional and can be installed separately in the environment or with the optional dependency commands:

- `pip install gzip_static[zopfli]` installs gzip_static and zopfli.
- `pip install gzip_static[performance]` installs gzip_static, xxhash and isal.
- `pip install gzip_static[full]` installs gzip_static, zopfli, xxhash and isal.

CHAPTER

TWO

USAGE

2.1 gzip-static usage

```
usage: gzip-static [-h] [-e EXTENSIONS_FILE] [-l {6,9,11} | --zopfli] [-f]
                   [--remove-orphans] [-d]
                   directory
```

2.1.1 Positional Arguments

directory The directory containing the static site

2.1.2 Named Arguments

-e, --extensions-file A file with extensions to consider when compressing. Use one line per extension. Check the default for an example. DEFAULT: /home/docs/checkouts/readthedocs.org/user_builds/gzip-static/envs/latest/lib/python3.7/site-packages/gzip_static-0.1.0-py3.7.egg/gzip_static/extensions.txt

Default: /home/docs/checkouts/readthedocs.org/user_builds/gzip-static/envs/latest/lib/python3.7/site-packages/gzip_static-0.1.0-py3.7.egg/gzip_static/extensions.txt

-l, --compression-level Possible choices: 6, 9, 11

The compression level that will be used for the gzip compression. Use 11 for zopfli compression (if available). Default: 9

Default: 9

--zopfli Use zopfli for the compression. Alias for -l 11 or –compression-level 11.

-f, --force Force recompression of all earlier compressed files.

Default: False

--remove-orphans Remove gzip files for which the parent file is missing and for which the extension is in the extensions file. For example: page3.html.gz present but no page3.html is present. In that case page3.html.gz will be removed.

Default: False

-d, --debug Print debug information to stderr.

Default: False

2.2 gzip-static-find-orphans usage

```
usage: gzip-static-find-orphans [-h] [-e EXTENSIONS_FILE] directory
```

2.2.1 Positional Arguments

directory The directory containing the static site

2.2.2 Named Arguments

-e, --extensions-file A file with extensions to consider when compressing. Use one line per extension. Check the default for an example. DEFAULT: /home/docs/checkouts/readthedocs.org/user_builds/gzip-static/envs/latest/lib/python3.7/site-packages/gzip_static-0.1.0-py3.7.egg/gzip_static/extensions.txt

Default: /home/docs/checkouts/readthedocs.org/user_builds/gzip-static/envs/latest/lib/python3.7/site-packages/gzip_static-0.1.0-py3.7.egg/gzip_static/extensions.txt

CHAPTER

THREE

API DOCUMENTATION: GZIP_STATIC

Functions to compress a website's static files.

class gzip_static.**GzipStaticResult**(*created: int*, *updated: int*, *skipped: int*, *deleted: int*)

A class containing the results for the gzip_static function.

property created

Alias for field number 0

property deleted

Alias for field number 3

property skipped

Alias for field number 2

property updated

Alias for field number 1

gzip_static.**compress_idempotent**(*filepath: Union[str, os.PathLike]*, *compresslevel=9*, *hash_algorithm=<built-in function openssl_sha1>*, *force: bool = False*) → int

Only compress the file if no companion .gz is present that contains the correct contents.

This function ensures the mode, atime and mtime of the gzip file are inherited from the file to be compressed.

Parameters

- **filepath** – The path to the file.
- **compresslevel** – The compression level. Use 11 for zopfli.
- **hash_algorithm** – The hash_algorithm to check the contents with.
- **force** – Always create a new '.gz' file to overwrite the old one.

Returns An integer that stands for the action taken. Matches with the COMPRESSED, RECOMPRESSED and SKIPPED constants in this module.

gzip_static.**compress_path**(*filepath: Union[str, os.PathLike]*, *compresslevel: int = 9*, *block_size: int = 32768*) → None

Compress a file's contents and write them to a '.gz' file.

Similar to gzip -k <filepath>

Parameters

- **filepath** – The path to the file
- **compresslevel** – The gzip compression level to use. Use 11 for zopfli compression.
- **block_size** – The size of the chunks read from the file at once.

gzip_static.**find_orphaned_files**(*dir: Union[str, os.PathLike]*, *extensions: Container[str] = frozenset({'.css', '.htm', '.html', '.js', '.json', '.rss', '.svg', '.txt', '.xml', '.xsl'})*) → Generator[str, None, None]

Scan a directory recursively for '.gz' files that do not have a parent file with an extension in extensions.

For example `find_orphaned_files(my_dir, set(".html"))` will find `index.html.gz` if `index.html` is not present. It will not find `myhostedarchive.tar.gz` as `.tar` is not in the set of extensions.

Parameters

- **dir** – The directory to scan.
- **extensions** – Extensions of parents file to include.

Returns A generator of filepaths of orphaned '.gz' files.

gzip_static.**find_static_files**(*dir: Union[str, os.PathLike]*, *extensions: Container[str] = frozenset({'.css', '.htm', '.html', '.js', '.json', '.rss', '.svg', '.txt', '.xml', '.xsl'})*) → Generator[str, None, None]

Scan a directory recursively for files that have an extension in the set of extensions.

Parameters

- **dir** – The directory to scan.
- **extensions** – A set of extensions to scan for.

Returns A generator of filepaths that match the extensions.

gzip_static.**get_extension**(*filename: str*)

The filename's extension, if any.

This includes the leading period. For example: '.txt'

gzip_static.**gzip_static**(*dir: Union[str, os.PathLike], extensions: Container[str] = frozenset({'.css', '.htm', '.html', '.js', '.json', '.rss', '.svg', '.txt', '.xml', '.xsl'}), compresslevel: int = 9, hash_algorithm=<built-in function openssl_sha1>, force: bool = False, remove_orphans: bool = False*) → *gzip_static.GzipStaticResult*

Gzip all static files in a directory and its subdirectories in an idempotent manner.

Parameters

- **dir** – The directory to recurse through.
- **extensions** – Extensions which are static files.
- **compresslevel** – The compression level that is used when compressing.
- **hash_algorithm** – The hash algorithm is used when checking file contents.
- **force** – Recompress all files regardless if content has changed or not.
- **remove_orphans** – Remove '.gz' files where the parent static file is no longer present.

Returns A tuple with 4 entries. The number of compressed, recompressed, skipped and deleted gzip files.

gzip_static.**hash_file_contents**(*filepath: Union[str, os.PathLike], hash_algorithm=<built-in function openssl_sha1>, block_size: int = 32768*) → bytes

Read contents from a file and return the hash.

Parameters

- **filepath** – The path to the file. Paths ending in '.gz' will be automatically decompressed.
- **hash_algorithm** – The hash algorithm to use. Must be hashlib-compatible.

- **block_size** – The size of the chunks read from the file at once.

Returns A digest of the hash.

gzip_static.**read_extensions_file**(*filepath: Union[str, os.PathLike]*) → Set[str]
Read a file where there is an extension on each line

Parameters **filepath** – The extensions file

Returns a set of extensions.

CHAPTER

FOUR

TECHNICAL CONSIDERATIONS

4.1 Choosing a checksum

Different checksums were considered. MD5 is traditionally used for checksumming, but also SHA-1, SHA-256 and SHA-512 see use as a hashing algorithm nowadays. Traditionally, cyclic redundancy checks are performed. These are available in the Python `zlib` libraries as the `crc32` and `adler32` functions. A fast method called XXHash is also available nowadays for hashing. There are Python bindings available as a package on PyPI.

As highlighted in this answer on bleepcoder by the XXHash author cyclic redundancy checks have slightly worse collisions than modern hash algorithms.

The XXHash homepage has a list of algorithms and their speeds. The `SHA1` hash algorithm is the fastest algorithm available in `hashlib.algorithms_guaranteed`. (This was verified on two different PC's). Therefore it was chosen as default. The XXH3_128 algorithm is used when XXhash is installed.

4.2 Speedy hashing of small gzip files

Speedy hashing of normal files is quite easy. Open a file, read it in blocks, feed each block to the hasher and get a checksum in the end. Choose a decent block size to speed it up slightly. (32K was used here. 128K is used by `cat` so choosing more than Python's default of 8K is quite common).

Speedy hashing of gzip files presents a problem. We can simply use Python's `gzip.open` which returns a `GzipFile`, but that is slow. Just like normal `open` this creates an interface to read the file, but then it gets more complicated. This gets wrapped into a `_PaddedFile` object which is then wrapped into a `_GzipReader` object which is then wrapped by the `GzipFile`. All these layers solve two problems:

- A controlled number of bytes can be read from the compressed file. Since the compression ratio can differ along the file it is impossible to grab a certain number of bytes and exactly know the size of the output once decompressed. `_GzipReader.read` has mechanisms built-in to always output the desired numbers of bytes.
- Gzip allows for multiple members (each consisting of header, compressed body and trailer) to be concatenated together. After a member is decompressed the remaining bytes in the file must be checked for another gzip member.

This functionality creates a lot of overhead. Using Python's `zlib.decompress` with `wbits=31` solves this problem as it can compress an in-memory block in its entirety. It cannot read multiple members but since these gzip files are compressed by gzip_static itself we know they only contain one member.

However this presents another problem: files have to be read in memory entirely. This was solved by using a `zlib.decompressobj` instead and using the `decompress` method on that object. This works with streaming decompression. It is not a problem that we do not know before which number of bytes is returned by the function. This is typically in the 3-6 times the input bytes range. At best gzip can compress at ratios of ~1000x. (Tested with all zeroes binary, all ones binary, and a repetition of a single character). So if the input block size is 8k, we can expect at most 8M bytes be

read in memory. This is acceptable, and this way even large static files of several hundreds of MB can be checksummed in a streaming fashion.

The great advantage of this method is that most gzip's will be smaller than 8k. So only one decompress call is needed. This is almost as fast as in-memory decompression with `zlib.decompress` but allowing streaming.

For example on docs.python.org compressing the static files compresses 6374 static files with a combined size of 481 MB. The resulting gzip sizes are as follows.

- gzip 8K or below (one decompress call): 3516
- gzip 8K - 16K (two decompress calls): 1560
- gzip 16K -24K (three decompress calls): 565
- gzip 24K - 32K (four decompress calls): 308
- gzip 32k-64k (eight or less decompress calls): 356
- gzip larger than 64k: 69

In total 6305 (99%!) of the gzip files are smaller than 64K and can be decompressed with eight or less calls. Since the `gzip.GzipFile` overhead weighs in very heavy at these small file sizes using `zlib.decompressobj` creates a notable speed improvement, reducing decompression time by about ~30% for the docs.python.org website.

The speedup can be even greater when using python-isal. Using its `isal_zlib.decompressobj` reduces the decompression time with more than 50%.

4.3 No brotli support

Brotli is an excellent compression algorithm. Most browsers support it. There are several reasons why it is not supported by gzip_static.

- The ngx_brotli module is not provided as a package by either Debian, Ubuntu or CentOS.
- Supporting two formats simultaneously makes the code more complex.
- brotli_static does not work well with gzip_static

This project was made to work with nginx's gzip plugin to host my websites. The gzip plugin is builtin in even the simplest nginx package on Debian (`nginx-light`). Getting brotli to work however is much more work. It needs to be compiled, but it needs to compiled exactly with the right instructions. Brotli has been around since 2013 and has tremendous advantages, but ngx_brotli has not been packaged in Debian for 8 years. The last release of Debian (bullseye) had 11294 new packages but ngx_brotli is nowhere on the horizon.

Once a properly working ngx_brotli module is packaged in Debian, I am happy to add brotli support!

CHAPTER

FIVE

CHANGELOG

5.1 version 0.1.0

- Publish documentation on readthedocs.
- Make sure the gzip files inherit file attributes from the parent file.
- Add functionality to remove orphaned gzip files.
- Speed up the checksumming process with isal and xxhash.
- Add zopfli support.
- Create functions to compress a website's static assets idempotently.

PYTHON MODULE INDEX

INDEX