## NAME

IO::Compress::Gzip - Write RFC 1952 files/buffers

## SYNOPSIS

```
use IO::Compress::Gzip qw(gzip $GzipError) ;

my $status = gzip $input => $output [,OPTS]
    or die "gzip failed: $GzipError\n";

my $z = new IO::Compress::Gzip $output [,OPTS]
    or die "gzip failed: $GzipError\n";

$z->print($string);
$z->printf($format, $string);
$z->write($string);
$z->syswrite($string [, $length, $offset]);
$z->flush();
$z->tell();
$z->eof();
$z->seek($position, $whence);
$z->binmode();
$z->fileno();
$z->opened();
$z->autoflush();
$z->input_line_number();
$z->newStream( [OPTS] );

$z->deflateParams();

$z->close() ;

$GzipError ;

# IO::File mode

print $z $string;
printf $z $format, $string;
tell $z
eof $z
seek $z, $position, $whence
binmode $z
fileno $z
close $z ;
```

## DESCRIPTION

This module provides a Perl interface that allows writing compressed data to files or buffer as defined in RFC 1952.

All the gzip headers defined in RFC 1952 can be created using this module.

For reading RFC 1952 files/buffers, see the companion module *IO::Uncompress::Gunzip*.

---

## Functional Interface

A top-level function, `gzip`, is provided to carry out "one-shot" compression between buffers and/or files. For finer control over the compression process, see the *OO Interface* section.

```
use IO::Compress::Gzip qw(gzip $GzipError) ;


gzip $input => $output [,OPTS]
    or die "gzip failed: $GzipError\n";
```

The functional interface needs Perl5.005 or better.

## gzip $input => $output [, OPTS]

`gzip` expects at least two parameters, `$input` and `$output`.

### The $input parameter

The parameter, `$input`, is used to define the source of the uncompressed data.

It can take one of the following forms:

A filename

> If the `$input` parameter is a simple scalar, it is assumed to be a filename. This file will be opened for reading and the input data will be read from it.

A filehandle

> If the `$input` parameter is a filehandle, the input data will be read from it. The string '-' can be used as an alias for standard input.

A scalar reference

> If `$input` is a scalar reference, the input data will be read from `$$input`.

An array reference

> If `$input` is an array reference, each element in the array must be a filename.

> The input data will be read from each file in turn.

> The complete array will be walked to ensure that it only contains valid filenames before any data is compressed.

An Input FileGlob string

> If `$input` is a string that is delimited by the characters "<" and ">" `gzip` will assume that it is an *input fileglob string*. The input is the list of files that match the fileglob.

> If the fileglob does not match any files ...

> See *File::GlobMapper* for more details.

If the `$input` parameter is any other type, `undef` will be returned.

In addition, if `$input` is a simple filename, the default values for the `Name` and `Time` options will be sourced from that file.

If you do not want to use these defaults they can be overridden by explicitly setting the `Name` and `Time` options or by setting the `Minimal` parameter.

### The $output parameter

The parameter `$output` is used to control the destination of the compressed data. This parameter can take one of these forms.

A filename

If the $output parameter is a simple scalar, it is assumed to be a filename. This file will be opened for writing and the compressed data will be written to it.

A filehandle

If the $output parameter is a filehandle, the compressed data will be written to it. The string '-' can be used as an alias for standard output.

A scalar reference

If $output is a scalar reference, the compressed data will be stored in $$output.

An Array Reference

If $output is an array reference, the compressed data will be pushed onto the array.

An Output FileGlob

If $output is a string that is delimited by the characters "<" and ">" gzip will assume that it is an *output fileglob string*. The output is the list of files that match the fileglob.

When $output is an fileglob string, $input must also be a fileglob string. Anything else is an error.

If the $output parameter is any other type, undef will be returned.

## Notes

When $input maps to multiple files/buffers and $output is a single file/buffer the input files/buffers will be stored in $output as a concatenated series of compressed data streams.

## Optional Parameters

Unless specified below, the optional parameters for gzip, OPTS, are the same as those used with the OO interface defined in the *Constructor Options* section below.

AutoClose => 0|1

This option applies to any input or output data streams to gzip that are filehandles.

If AutoClose is specified, and the value is true, it will result in all input and/or output filehandles being closed once gzip has completed.

This parameter defaults to 0.

BinModeIn => 0|1

When reading from a file or filehandle, set binmode before reading.

Defaults to 0.

Append => 0|1

TODO

## Examples

To read the contents of the file file1.txt and write the compressed data to the file file1.txt.gz.

```
use strict ;
use warnings ;
use IO::Compress::Gzip qw(gzip $GzipError) ;

my $input = "file1.txt";
gzip $input => "$input.gz"
    or die "gzip failed: $GzipError\n";
```

To read from an existing Perl filehandle, $input, and write the compressed data to a buffer,

$buffer.

```
    use strict ;
    use warnings ;
    use IO::Compress::Gzip qw(gzip $GzipError) ;
    use IO::File ;

    my $input = new IO::File "<file1.txt"
        or die "Cannot open 'file1.txt': $!\n" ;
    my $buffer ;
    gzip $input => \$buffer
        or die "gzip failed: $GzipError\n";
```

To compress all files in the directory "/my/home" that match "*.txt" and store the compressed data in the same directory

```
    use strict ;
    use warnings ;
    use IO::Compress::Gzip qw(gzip $GzipError) ;

    gzip '</my/home/*.txt>' => '<*.gz>'
        or die "gzip failed: $GzipError\n";
```

and if you want to compress each file one at a time, this will do the trick

```
    use strict ;
    use warnings ;
    use IO::Compress::Gzip qw(gzip $GzipError) ;

    for my $input ( glob "/my/home/*.txt" )
    {
        my $output = "$input.gz" ;
        gzip $input => $output
            or die "Error compressing '$input': $GzipError\n";
    }
```

## OO Interface
### Constructor

The format of the constructor for `IO::Compress::Gzip` is shown below

```
    my $z = new IO::Compress::Gzip $output [,OPTS]
        or die "IO::Compress::Gzip failed: $GzipError\n";
```

It returns an `IO::Compress::Gzip` object on success and undef on failure. The variable `$GzipError` will contain an error message on failure.

If you are running Perl 5.005 or better the object, `$z`, returned from IO::Compress::Gzip can be used exactly like an *IO::File* filehandle. This means that all normal output file operations can be carried out with `$z`. For example, to write to a compressed file/buffer you can use either of these forms

```
    $z->print("hello world\n");
    print $z "hello world\n";
```

The mandatory parameter `$output` is used to control the destination of the compressed data. This parameter can take one of these forms.

A filename

> If the `$output` parameter is a simple scalar, it is assumed to be a filename. This file will be opened for writing and the compressed data will be written to it.

A filehandle

> If the `$output` parameter is a filehandle, the compressed data will be written to it. The string '-' can be used as an alias for standard output.

A scalar reference

> If `$output` is a scalar reference, the compressed data will be stored in `$$output`.

If the `$output` parameter is any other type, `IO::Compress::Gzip::new` will return undef.

## Constructor Options

OPTS is any combination of the following options:

`AutoClose => 0|1`

> This option is only valid when the `$output` parameter is a filehandle. If specified, and the value is true, it will result in the `$output` being closed once either the `close` method is called or the `IO::Compress::Gzip` object is destroyed.
>
> This parameter defaults to 0.

`Append => 0|1`

> Opens `$output` in append mode.
>
> The behaviour of this option is dependent on the type of `$output`.
>
> * A Buffer
>
> > If `$output` is a buffer and `Append` is enabled, all compressed data will be append to the end if `$output`. Otherwise `$output` will be cleared before any data is written to it.
>
> * A Filename
>
> > If `$output` is a filename and `Append` is enabled, the file will be opened in append mode. Otherwise the contents of the file, if any, will be truncated before any compressed data is written to it.
>
> * A Filehandle
>
> > If `$output` is a filehandle, the file pointer will be positioned to the end of the file via a call to `seek` before any compressed data is written to it. Otherwise the file pointer will not be moved.
>
> This parameter defaults to 0.

`Merge => 0|1`

> This option is used to compress input data and append it to an existing compressed data stream in `$output`. The end result is a single compressed data stream stored in `$output`.
>
> It is a fatal error to attempt to use this option when `$output` is not an RFC 1952 data stream.
>
> There are a number of other limitations with the `Merge` option:
>
> 1    This module needs to have been built with zlib 1.2.1 or better to work. A fatal error will be thrown if `Merge` is used with an older version of zlib.
>
> 2    If `$output` is a file or a filehandle, it must be seekable.
>
> This parameter defaults to 0.

-Level

>Defines the compression level used by zlib. The value should either be a number between 0 and 9 (0 means no compression and 9 is maximum compression), or one of the symbolic constants defined below.

```
Z_NO_COMPRESSION
Z_BEST_SPEED
Z_BEST_COMPRESSION
Z_DEFAULT_COMPRESSION
```

>The default is Z_DEFAULT_COMPRESSION.

>Note, these constants are not imported by IO::Compress::Gzip by default.

```
use IO::Compress::Gzip qw(:strategy);
use IO::Compress::Gzip qw(:constants);
use IO::Compress::Gzip qw(:all);
```

-Strategy

>Defines the strategy used to tune the compression. Use one of the symbolic constants defined below.

```
Z_FILTERED
Z_HUFFMAN_ONLY
Z_RLE
Z_FIXED
Z_DEFAULT_STRATEGY
```

>The default is Z_DEFAULT_STRATEGY.

Minimal => 0|1

>If specified, this option will force the creation of the smallest possible compliant gzip header (which is exactly 10 bytes long) as defined in RFC 1952.

>See the section titled "Compliance" in RFC 1952 for a definition of the values used for the fields in the gzip header.

>All other parameters that control the content of the gzip header will be ignored if this parameter is set to 1.

>This parameter defaults to 0.

Comment => $comment

>Stores the contents of $comment in the COMMENT field in the gzip header. By default, no comment field is written to the gzip file.

>If the -Strict option is enabled, the comment can only consist of ISO 8859-1 characters plus line feed.

>If the -Strict option is disabled, the comment field can contain any character except NULL. If any null characters are present, the field will be truncated at the first NULL.

Name => $string

>Stores the contents of $string in the gzip NAME header field. If Name is not specified, no gzip NAME field will be created.

>If the -Strict option is enabled, $string can only consist of ISO 8859-1 characters.

>If -Strict is disabled, then $string can contain any character except NULL. If any null characters are present, the field will be truncated at the first NULL.

Time => $number

Sets the MTIME field in the gzip header to $number.

This field defaults to the time the `IO::Compress::Gzip` object was created if this option is not specified.

`TextFlag => 0|1`

> This parameter controls the setting of the FLG.FTEXT bit in the gzip header. It is used to signal that the data stored in the gzip file/buffer is probably text.
>
> The default is 0.

`HeaderCRC => 0|1`

> When true this parameter will set the FLG.FHCRC bit to 1 in the gzip header and set the CRC16 header field to the CRC of the complete gzip header except the CRC16 field itself.
>
> **Note** that gzip files created with the `HeaderCRC` flag set to 1 cannot be read by most, if not all, of the the standard gunzip utilities, most notably gzip version 1.2.4. You should therefore avoid using this option if you want to maximize the portability of your gzip files.
>
> This parameter defaults to 0.

`OS_Code => $value`

> Stores `$value` in the gzip OS header field. A number between 0 and 255 is valid.
>
> If not specified, this parameter defaults to the OS code of the Operating System this module was built on. The value 3 is used as a catch-all for all Unix variants and unknown Operating Systems.

`ExtraField => $data`

> This parameter allows additional metadata to be stored in the ExtraField in the gzip header. An RFC 1952 compliant ExtraField consists of zero or more subfields. Each subfield consists of a two byte header followed by the subfield data.
>
> The list of subfields can be supplied in any of the following formats
>
> ```
>     -ExtraField => [$id1, $data1,
>                     $id2, $data2,
>                      ...
>                    ]
>     -ExtraField => [ [$id1 => $data1],
>                      [$id2 => $data2],
>                      ...
>                    ]
>     -ExtraField => { $id1 => $data1,
>                      $id2 => $data2,
>                      ...
>                    }
> ```
>
> Where `$id1`, `$id2` are two byte subfield ID's. The second byte of the ID cannot be 0, unless the `Strict` option has been disabled.
>
> If you use the hash syntax, you have no control over the order in which the ExtraSubFields are stored, plus you cannot have SubFields with duplicate ID.
>
> Alternatively the list of subfields can by supplied as a scalar, thus
>
> ```
>     -ExtraField => $rawdata
> ```
>
> If you use the raw format, and the `Strict` option is enabled, `IO::Compress::Gzip` will check that `$rawdata` consists of zero or more conformant sub-fields. When `Strict` is disabled, `$rawdata` can consist of any arbitrary byte stream.
>
> The maximum size of the Extra Field 65535 bytes.

ExtraFlags => $value

>    Sets the XFL byte in the gzip header to `$value`.
>
>    If this option is not present, the value stored in XFL field will be determined by the setting of the `Level` option.
>
>    If `Level => Z_BEST_SPEED` has been specified then XFL is set to 2. If `Level => Z_BEST_COMPRESSION` has been specified then XFL is set to 4. Otherwise XFL is set to 0.

Strict => 0|1

>    `Strict` will optionally police the values supplied with other options to ensure they are compliant with RFC1952.
>
>    This option is enabled by default.
>
>    If `Strict` is enabled the following behaviour will be policed:
>
>    - The value supplied with the `Name` option can only contain ISO 8859-1 characters.
>    - The value supplied with the `Comment` option can only contain ISO 8859-1 characters plus line-feed.
>    - The values supplied with the `-Name` and `-Comment` options cannot contain multiple embedded nulls.
>    - If an `ExtraField` option is specified and it is a simple scalar, it must conform to the sub-field structure as defined in RFC 1952.
>    - If an `ExtraField` option is specified the second byte of the ID will be checked in each subfield to ensure that it does not contain the reserved value 0x00.
>
>    When `Strict` is disabled the following behaviour will be policed:
>
>    - The value supplied with `-Name` option can contain any character except NULL.
>    - The value supplied with `-Comment` option can contain any character except NULL.
>    - The values supplied with the `-Name` and `-Comment` options can contain multiple embedded nulls. The string written to the gzip header will consist of the characters up to, but not including, the first embedded NULL.
>    - If an `ExtraField` option is specified and it is a simple scalar, the structure will not be checked. The only error is if the length is too big.
>    - The ID header in an `ExtraField` sub-field can consist of any two bytes.

## Examples

TODO

## Methods

## print

>    Usage is
>
>    ```
>    $z->print($data)
>    print $z $data
>    ```
>
>    Compresses and outputs the contents of the `$data` parameter. This has the same behaviour as the `print` built-in.
>
>    Returns true if successful.

**printf**

Usage is

```
$z->printf($format, $data)
printf $z $format, $data
```

Compresses and outputs the contents of the $data parameter.

Returns true if successful.

**syswrite**

Usage is

```
$z->syswrite $data
$z->syswrite $data, $length
$z->syswrite $data, $length, $offset
```

Compresses and outputs the contents of the $data parameter.

Returns the number of uncompressed bytes written, or undef if unsuccessful.

**write**

Usage is

```
$z->write $data
$z->write $data, $length
$z->write $data, $length, $offset
```

Compresses and outputs the contents of the $data parameter.

Returns the number of uncompressed bytes written, or undef if unsuccessful.

**flush**

Usage is

```
$z->flush;
$z->flush($flush_type);
```

Flushes any pending compressed data to the output file/buffer.

This method takes an optional parameter, $flush_type, that controls how the flushing will be carried out. By default the $flush_type used is Z_FINISH. Other valid values for $flush_type are Z_NO_FLUSH, Z_SYNC_FLUSH, Z_FULL_FLUSH and Z_BLOCK. It is strongly recommended that you only set the flush_type parameter if you fully understand the implications of what it does - overuse of flush can seriously degrade the level of compression achieved. See the zlib documentation for details.

Returns true on success.

**tell**

Usage is

```
$z->tell()
tell $z
```

Returns the uncompressed file offset.

**eof**

Usage is

```
$z->eof();
eof($z);
```

Returns true if the `close` method has been called.

**seek**

```
$z->seek($position, $whence);
seek($z, $position, $whence);
```

Provides a sub-set of the `seek` functionality, with the restriction that it is only legal to seek forward in the output file/buffer. It is a fatal error to attempt to seek backward.

Empty parts of the file/buffer will have NULL (0x00) bytes written to them.

The `$whence` parameter takes one the usual values, namely SEEK_SET, SEEK_CUR or SEEK_END.

Returns 1 on success, 0 on failure.

**binmode**

Usage is

```
$z->binmode
binmode $z ;
```

This is a noop provided for completeness.

**opened**

```
$z->opened()
```

Returns true if the object currently refers to a opened file/buffer.

**autoflush**

```
my $prev = $z->autoflush()
my $prev = $z->autoflush(EXPR)
```

If the `$z` object is associated with a file or a filehandle, this method returns the current autoflush setting for the underlying filehandle. If EXPR is present, and is non-zero, it will enable flushing after every write/print operation.

If `$z` is associated with a buffer, this method has no effect and always returns `undef`.

**Note** that the special variable $ | **cannot** be used to set or retrieve the autoflush setting.

**input_line_number**

```
$z->input_line_number()
$z->input_line_number(EXPR)
```

This method always returns `undef` when compressing.

**fileno**

```
$z->fileno()
fileno($z)
```

If the `$z` object is associated with a file or a filehandle, `fileno` will return the underlying file descriptor. Once the `close` method is called `fileno` will return `undef`.

If the `$z` object is is associated with a buffer, this method will return `undef`.

## close

```
$z->close() ;
close $z ;
```

Flushes any pending compressed data and then closes the output file/buffer.

For most versions of Perl this method will be automatically invoked if the IO::Compress::Gzip object is destroyed (either explicitly or by the variable with the reference to the object going out of scope). The exceptions are Perl versions 5.005 through 5.00504 and 5.8.0. In these cases, the `close` method will be called automatically, but not until global destruction of all live objects when the program is terminating.

Therefore, if you want your scripts to be able to run on all versions of Perl, you should call `close` explicitly and not rely on automatic closing.

Returns true on success, otherwise 0.

If the `AutoClose` option has been enabled when the IO::Compress::Gzip object was created, and the object is associated with a file, the underlying file will also be closed.

## newStream([OPTS])

Usage is

```
$z->newStream( [OPTS] )
```

Closes the current compressed data stream and starts a new one.

OPTS consists of any of the the options that are available when creating the `$z` object.

See the *Constructor Options* section for more details.

## deflateParams

Usage is

```
$z->deflateParams
```

TODO

## Importing

A number of symbolic constants are required by some methods in `IO::Compress::Gzip`. None are imported by default.

:all

Imports `gzip`, `$GzipError` and all symbolic constants that can be used by `IO::Compress::Gzip`. Same as doing this

```
use IO::Compress::Gzip qw(gzip $GzipError :constants) ;
```

:constants

Import all symbolic constants. Same as doing this

```
use IO::Compress::Gzip qw(:flush :level :strategy) ;
```

:flush

These symbolic constants are used by the `flush` method.

```
Z_NO_FLUSH
Z_PARTIAL_FLUSH
Z_SYNC_FLUSH
Z_FULL_FLUSH
Z_FINISH
Z_BLOCK
```

:level

These symbolic constants are used by the `Level` option in the constructor.

```
Z_NO_COMPRESSION
Z_BEST_SPEED
Z_BEST_COMPRESSION
Z_DEFAULT_COMPRESSION
```

:strategy

These symbolic constants are used by the `Strategy` option in the constructor.

```
Z_FILTERED
Z_HUFFMAN_ONLY
Z_RLE
Z_FIXED
Z_DEFAULT_STRATEGY
```

# EXAMPLES
## Apache::GZip Revisited

See *IO::Compress::FAQ*

## Working with Net::FTP

See *IO::Compress::FAQ*

# SEE ALSO

*Compress::Zlib*, *IO::Uncompress::Gunzip*, *IO::Compress::Deflate*, *IO::Uncompress::Inflate*,
*IO::Compress::RawDeflate*, *IO::Uncompress::RawInflate*, *IO::Compress::Bzip2*,
*IO::Uncompress::Bunzip2*, *IO::Compress::Lzop*, *IO::Uncompress::UnLzop*, *IO::Compress::Lzf*,
*IO::Uncompress::UnLzf*, *IO::Uncompress::AnyInflate*, *IO::Uncompress::AnyUncompress*

*Compress::Zlib::FAQ*

*File::GlobMapper*, *Archive::Zip*, *Archive::Tar*, *IO::Zlib*

For RFC 1950, 1951 and 1952 see *http://www.faqs.org/rfcs/rfc1950.html*,
*http://www.faqs.org/rfcs/rfc1951.html* and *http://www.faqs.org/rfcs/rfc1952.html*

The *zlib* compression library was written by Jean-loup Gailly *gzip@prep.ai.mit.edu* and Mark Adler
*madler@alumni.caltech.edu*.

The primary site for the *zlib* compression library is *http://www.zlib.org*.

The primary site for gzip is *http://www.gzip.org*.

## AUTHOR

This module was written by Paul Marquess, *pmqs@cpan.org*.

## MODIFICATION HISTORY

See the Changes file.

## COPYRIGHT AND LICENSE

Copyright (c) 2005-2009 Paul Marquess. All rights reserved.

This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.