

Discover Packages > Standard library > compress > gzip 

gzip




package


standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 8 |

Imported by: 45,704

Details

 Valid [go.mod](#) file   Redistributable license   Tagged version 

 Stable version 

[Learn more](#)

Repository

cs.opensource.google/go/go

Links

 [Report a Vulnerability](#)

 Documentation 

<> Documentation

Overview

Package gzip implements reading and writing of gzip format compressed files, as specified in [RFC 1952](#).

► [Example \(CompressingReader\)](#)

► [Example \(WriterReader\)](#)

Index

[Constants](#)

[Variables](#)

[type Header](#)

[type Reader](#)

[func NewReader\(r io.Reader\) \(*Reader, error\)](#)

[func \(z *Reader\) Close\(\) error](#)

[func \(z *Reader\) Multistream\(ok bool\)](#)

[func \(z *Reader\) Read\(p \[\]byte\) \(n int, err error\)](#)

[func \(z *Reader\) Reset\(r io.Reader\) error](#)

[type Writer](#)

[func NewWriter\(w io.Writer\) *Writer](#)

[func NewWriterLevel\(w io.Writer, level int\) \(*Writer, error\)](#)

[func \(z *Writer\) Close\(\) error](#)

```
func (z *Writer) Flush() error
func (z *Writer) Reset(w io.Writer)
func (z *Writer) Write(p []byte) (int, error)
```

Examples

[Package \(CompressingReader\)](#)

[Package \(WriterReader\)](#)

[Reader.Multistream](#)

Constants

[View Source](#)

```
const (
    NoCompression      = flate.NoCompression
    BestSpeed          = flate.BestSpeed
    BestCompression    = flate.BestCompression
    DefaultCompression = flate.DefaultCompression
    HuffmanOnly        = flate.HuffmanOnly
)
```

These constants are copied from the flate package, so that code that imports "compress/gzip" does not also have to import "compress/flate".

Variables

[View Source](#)

```
var (
    // ErrChecksum is returned when reading GZIP data that has an invalid checksum.
    ErrChecksum = errors.New("gzip: invalid checksum")
    // ErrHeader is returned when reading GZIP data that has an invalid header.
    ErrHeader = errors.New("gzip: invalid header")
)
```

Functions

This section is empty.

Types

type [Header](#)

```
type Header struct {
    Comment string    // comment
    Extra   []byte   // "extra data"
    ModTime time.Time // modification time
    Name    string   // file name
    OS      byte     // operating system type
}
```

The gzip file stores a header giving metadata about the compressed file. That header is exposed as the fields of the `Writer` and `Reader` structs.

Strings must be UTF-8 encoded and may only contain Unicode code points U+0001 through U+00FF, due to limitations of the GZIP file format.

type `Reader`

```
type Reader struct {  
    Header // valid after NewReader or Reader.Reset  
    // contains filtered or unexported fields  
}
```

A `Reader` is an `io.Reader` that can be read to retrieve uncompressed data from a gzip-format compressed file.

In general, a gzip file can be a concatenation of gzip files, each with its own header. Reads from the `Reader` return the concatenation of the uncompressed data of each. Only the first header is recorded in the `Reader` fields.

Gzip files store a length and checksum of the uncompressed data. The `Reader` will return an `ErrChecksum` when `Read` reaches the end of the uncompressed data if it does not have the expected length or checksum. Clients should treat data returned by `Read` as tentative until they receive the `io.EOF` marking the end of the data.

func `NewReader`

```
func NewReader(r io.Reader) (*Reader, error)
```

`NewReader` creates a new `Reader` reading the given reader. If `r` does not also implement `io.ByteReader`, the decompressor may read more data than necessary from `r`.

It is the caller's responsibility to call `Close` on the `Reader` when done.

The `Reader.Header` fields will be valid in the `Reader` returned.

func (`*Reader`) `Close`

```
func (z *Reader) Close() error
```

`Close` closes the `Reader`. It does not close the underlying `io.Reader`. In order for the GZIP checksum to be verified, the reader must be fully consumed until the `io.EOF`.

func (`*Reader`) `Multistream`

added in go1.4

```
func (z *Reader) Multistream(ok bool)
```

`Multistream` controls whether the reader supports multistream files.

If enabled (the default), the Reader expects the input to be a sequence of individually gzipped data streams, each with its own header and trailer, ending at EOF. The effect is that the concatenation of a sequence of gzipped files is treated as equivalent to the gzip of the concatenation of the sequence. This is standard behavior for gzip readers.

Calling `Multistream(false)` disables this behavior; disabling the behavior can be useful when reading file formats that distinguish individual gzip data streams or mix gzip data streams with other data streams. In this mode, when the Reader reaches the end of the data stream, `Read` returns `io.EOF`. The underlying reader must implement `io.ByteReader` in order to be left positioned just after the gzip stream. To start the next stream, call `z.Reset(r)` followed by `z.Multistream(false)`. If there is no next stream, `z.Reset(r)` will return `io.EOF`.

► Example

`func (*Reader) Read`

```
func (z *Reader) Read(p []byte) (n int, err error)
```

`Read` implements `io.Reader`, reading uncompressed bytes from its underlying Reader.

`func (*Reader) Reset`

added in go1.3

```
func (z *Reader) Reset(r io.Reader) error
```

`Reset` discards the Reader `z`'s state and makes it equivalent to the result of its original state from `NewReader`, but reading from `r` instead. This permits reusing a Reader rather than allocating a new one.

`type Writer`

```
type Writer struct {  
    Header // written at first call to Write, Flush, or Close  
    // contains filtered or unexported fields  
}
```

A Writer is an `io.WriteCloser`. Writes to a Writer are compressed and written to `w`.

`func NewWriter`

```
func NewWriter(w io.Writer) *Writer
```

`NewWriter` returns a new Writer. Writes to the returned writer are compressed and written to `w`.

It is the caller's responsibility to call `Close` on the Writer when done. Writes may be buffered and not flushed until `Close`.

Callers that wish to set the fields in `Writer.Header` must do so before the first call to `Write`, `Flush`, or `Close`.

func NewWriterLevel

```
func NewWriterLevel(w io.Writer, level int) (*Writer, error)
```

NewWriterLevel is like NewWriter but specifies the compression level instead of assuming DefaultCompression.

The compression level can be DefaultCompression, NoCompression, HuffmanOnly or any integer value between BestSpeed and BestCompression inclusive. The error returned will be nil if the level is valid.

func (*Writer) Close

```
func (z *Writer) Close() error
```

Close closes the Writer by flushing any unwritten data to the underlying io.Writer and writing the GZIP footer. It does not close the underlying io.Writer.

func (*Writer) Flush

added in go1.1

```
func (z *Writer) Flush() error
```

Flush flushes any pending compressed data to the underlying writer.

It is useful mainly in compressed network protocols, to ensure that a remote reader has enough data to reconstruct a packet. Flush does not return until the data has been written. If the underlying writer returns an error, Flush returns that error.

In the terminology of the zlib library, Flush is equivalent to Z_SYNC_FLUSH.

func (*Writer) Reset

added in go1.2

```
func (z *Writer) Reset(w io.Writer)
```

Reset discards the Writer z's state and makes it equivalent to the result of its original state from NewWriter or NewWriterLevel, but writing to w instead. This permits reusing a Writer rather than allocating a new one.

func (*Writer) Write

```
func (z *Writer) Write(p []byte) (int, error)
```

Write writes a compressed form of p to the underlying io.Writer. The compressed bytes are not necessarily flushed until the Writer is closed.



Source Files

[View all](#)

Why Go

[Use Cases](#)

[Case Studies](#)

Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

Packages

[Standard Library](#)

[About Go Packages](#)

About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Golang Weekly](#)

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)



Google