Clzip

LZMA lossless data compressor
for Clzip version 1.8, 13 May 2016

by Antonio Diaz Diaz

Table of Contents

This manual is for Clzip (version 1.8, 13 May 2016).

Copyright © 2010-2016 Antonio Diaz Diaz.

This manual is free documentation: you have unlimited permission to copy, distribute and modify it.

1 Introduction

Clzip is a lossless data compressor with a user interface similar to the one of gzip or bzip2. Clzip is about as fast as gzip, compresses most files more than bzip2, and is better than both from a data recovery perspective.

Clzip uses the lzip file format; the files produced by clzip are fully compatible with lzip-1.4 or newer, and can be rescued with lziprecover. Clzip is in fact a C language version of lzip, intended for embedded devices or systems lacking a C++ compiler.

The lzip file format is designed for data sharing and long-term archiving, taking into account both data integrity and decoder availability:

- The lzip format provides very safe integrity checking and some data recovery means. The lziprecover program can repair bit-flip errors (one of the most common forms of data corruption) in lzip files, and provides data recovery capabilities, including error-checked merging of damaged copies of a file. See Section “Data safety” in `lziprecover`.
- The lzip format is as simple as possible (but not simpler). The lzip manual provides the code of a simple decompressor along with a detailed explanation of how it works, so that with the only help of the lzip manual it would be possible for a digital archaeologist to extract the data from a lzip file long after quantum computers eventually render LZMA obsolete.
- Additionally the lzip reference implementation is copylefted, which guarantees that it will remain free forever.

A nice feature of the lzip format is that a corrupt byte is easier to repair the nearer it is from the beginning of the file. Therefore, with the help of lziprecover, losing an entire archive just because of a corrupt byte near the beginning is a thing of the past.

The member trailer stores the 32-bit CRC of the original data, the size of the original data and the size of the member. These values, together with the end-of-stream marker, provide a 3 factor integrity checking which guarantees that the decompressed version of the data is identical to the original. This guards against corruption of the compressed data, and against undetected bugs in clzip (hopefully very unlikely). The chances of data corruption going undetected are microscopic. Be aware, though, that the check occurs upon decompression, so it can only tell you that something is wrong. It can’t help you recover the original uncompressed data.

Clzip uses the same well-defined exit status values used by lzip and bzip2, which makes it safer than compressors returning ambiguous warning values (like gzip) when it is used as a back end for other programs like tar or zutils.

Clzip will automatically use the smallest possible dictionary size for each file without exceeding the given limit. Keep in mind that the decompression memory requirement is affected at compression time by the choice of dictionary size limit.

The amount of memory required for compression is about 1 or 2 times the dictionary size limit (1 if input file size is less than dictionary size limit, else 2) plus 9 times the dictionary size really used. The option ‘`-0`’ is special and only requires about 1.5 MiB at most. The amount of memory required for decompression is about 46 kB larger than the dictionary size really used.

When compressing, clzip replaces every file given in the command line with a compressed version of itself, with the name "original_name.lz". When decompressing, clzip attempts to guess the name for the decompressed file from that of the compressed file as follows:

filename.lz	becomes	filename
filename.tlz	becomes	filename.tar
anyothername	becomes	anyothername.out

The DJGPP port of clzip checks if the used file system supports LFNs or only SFNs. If only SFN support is available, the file name and its extension must be truncated and modified to encode the compressor extension and the numbers to identify a multi volume archive. The program does not store the original file name in the header of the archive thus there is no chance to regenerate them when uncompressiong the archive in a SFN environment. Here the same naming schema is applied than the one used in the bzip2 and xz ports. The archiver extension is ".lz". On SFN systems, the ".lz" extension will consume one or two characters of the original file name extension.

For SFN systems the following rules apply when decompressing:

filename.exl	becomes	filename.ex
filename.elz	becomes	filename.e
filename.lz	becomes	filename
filename.tlz	becomes	filename.tar
anyothername.ext	becomes	anyothername.out

For SFN systems the following rules apply when compressing:

filename.exe	becomes	filename.exl
filename.ex	becomes	filename.exl
filename.e	becomes	filename.elz
filename	becomes	filename.lz
filename.tar	becomes	filename.tlz

If only SFN support is available, the DJGPP port of clzip will always consume the last 5 characters of the 8 possible name characters to encode the volume number. E.g.: when compressing a file with the file name '`filename.ext`' and splitting it into multiple volumes, the different volumes will become '`fil00001.exl`', '`fil00002.exl`', etc.

(De)compressing a file is much like copying or moving it; therefore clzip preserves the access and modification dates, permissions, and, when possible, ownership of the file just as "cp -p" does. (If the user ID or the group ID can't be duplicated, the file permission bits S_ISUID and S_ISGID are cleared).

Clzip is able to read from some types of non regular files if the '`--stdout`' option is specified.

If no file names are specified, clzip compresses (or decompresses) from standard input to standard output. In this case, clzip will decline to write compressed output to a terminal, as this would be entirely incomprehensible and therefore pointless.

Clzip will correctly decompress a file which is the concatenation of two or more compressed files. The result is the concatenation of the corresponding uncompressed files. Integrity testing of concatenated compressed files is also supported.

Clzip can produce multimember files and safely recover, with lziprecover, the undamaged members in case of file damage. Clzip can also split the compressed output in volumes of a given size, even when reading from standard input. This allows the direct creation of multivolume compressed tar archives.

Clzip is able to compress and decompress streams of unlimited size by automatically creating multimember output. The members so created are large, about 2 PiB each.

2 Invoking clzip

The format for running clzip is:

```
clzip [options] [files]
```

'`-`' used as a *file* argument means standard input. It can be mixed with other *files* and is read just once, the first time it appears in the command line.

Clzip supports the following options:

`-h`
`--help` Print an informative help message describing the options and exit.

`-V`
`--version`
Print the version number of clzip on the standard output and exit.

`-a`
`--trailing-error`
Exit with error status 2 if any remaining input is detected after decompressing the last member. Such remaining input is usually trailing garbage that can be safely ignored. See [concat-example], page 12.

`-b` *`bytes`*
`--member-size=`*`bytes`*
Set the member size limit to *bytes*. A small member size may degrade compression ratio, so use it only when needed. Valid values range from 100 kB to 2 PiB. Defaults to 2 PiB.

`-c`
`--stdout` Compress or decompress to standard output; keep input files unchanged. If compressing several files, each file is compressed independently. This option is needed when reading from a named pipe (fifo) or from a device. Use it also to recover as much of the uncompressed data as possible when decompressing a corrupt file.

`-d`
`--decompress`
Decompress the specified file(s). If a file does not exist or can't be opened, clzip continues decompressing the rest of the files. If a file fails to decompress, clzip exits immediately without decompressing the rest of the files.

`-f`
`--force` Force overwrite of output files.

`-F`
`--recompress`
Force re-compression of files whose name already has the '`.lz`' or '`.tlz`' suffix.

`-k`
`--keep` Keep (don't delete) input files during compression or decompression.

`-m` *`bytes`*
`--match-length=`*`bytes`*
Set the match length limit in bytes. After a match this long is found, the search is finished. Valid values range from 5 to 273. Larger values usually give better compression ratios but longer compression times.

`-o` *`file`*
`--output=`*`file`*
When reading from standard input and '`--stdout`' has not been specified, use '*`file`*' as the virtual name of the uncompressed file. This produces a file named '*`file`*' when decompressing, a file named '*`file`*`.lz`' when compressing, and several files named '*`file`*`00001.lz`', '*`file`*`00002.lz`', etc, when compressing and splitting the output in volumes.

`-q`
`--quiet` Quiet operation. Suppress all messages.

`-s` *`bytes`*
`--dictionary-size=`*`bytes`*
Set the dictionary size limit in bytes. Clzip will use the smallest possible dictionary size for each file without exceeding this limit. Valid values range from 4 KiB to 512 MiB. Values 12 to 29 are interpreted as powers of two, meaning 2^12 to 2^29 bytes. Note that dictionary sizes are quantized. If the specified size does not match one of the valid sizes, it will be rounded upwards by adding up to (*bytes* / 8) to it.

For maximum compression you should use a dictionary size limit as large as possible, but keep in mind that the decompression memory requirement is affected at compression time by the choice of dictionary size limit.

`-S` *`bytes`*
`--volume-size=`*`bytes`*
Split the compressed output into several volume files with names '`original_name00001.lz`', '`original_name00002.lz`', etc, and set the volume size limit to *bytes*. Each volume is a complete, maybe multimember, lzip file. A small volume size may degrade compression ratio, so use it only when needed. Valid values range from 100 kB to 4 EiB.

`-t`
`--test` Check integrity of the specified file(s), but don't decompress them. This really performs a trial decompression and throws away the result. Use it together with '`-v`' to see information about the file(s). If a file fails the test, clzip continues checking the rest of the files.

`-v`
`--verbose`
Verbose mode.
When compressing, show the compression ratio for each file processed. A second '`-v`' shows the progress of compression.
When decompressing or testing, further -v's (up to 4) increase the verbosity level, showing status, compression ratio, dictionary size, trailer contents (CRC, data size, member size), and up to 6 bytes of trailing data (if any).

`-0 .. -9` Set the compression parameters (dictionary size and match length limit) as shown in the table below. The default compression level is '`-6`'. Note that '`-9`' can be much slower than '`-0`'. These options have no effect when decompressing.

The bidimensional parameter space of LZMA can't be mapped to a linear scale optimal for all files. If your files are large, very repetitive, etc, you may need to use the '`--dictionary-size`' and '`--match-length`' options directly to achieve optimal performance.

Level	Dictionary size	Match length limit
-0	64 KiB	16 bytes
-1	1 MiB	5 bytes
-2	1.5 MiB	6 bytes
-3	2 MiB	8 bytes
-4	3 MiB	12 bytes
-5	4 MiB	20 bytes
-6	8 MiB	36 bytes
-7	16 MiB	68 bytes
-8	24 MiB	132 bytes
-9	32 MiB	273 bytes

`--fast`
`--best` Aliases for GNU gzip compatibility.

Numbers given as arguments to options may be followed by a multiplier and an optional '`B`' for "byte".

Table of SI and binary prefixes (unit multipliers):

Prefix	Value	Prefix	Value
k	kilobyte (10^3 = 1000)	Ki	kibibyte (2^10 = 1024)
M	megabyte (10^6)	Mi	mebibyte (2^20)
G	gigabyte (10^9)	Gi	gibibyte (2^30)
T	terabyte (10^12)	Ti	tebibyte (2^40)
P	petabyte (10^15)	Pi	pebibyte (2^50)
E	exabyte (10^18)	Ei	exbibyte (2^60)
Z	zettabyte (10^21)	Zi	zebibyte (2^70)
Y	yottabyte (10^24)	Yi	yobibyte (2^80)

Exit status: 0 for a normal exit, 1 for environmental problems (file not found, invalid flags, I/O errors, etc), 2 to indicate a corrupt or invalid input file, 3 for an internal consistency error (eg, bug) which caused clzip to panic.

3 File format

Perfection is reached, not when there is no longer anything to add, but when there is no longer anything to take away.
— Antoine de Saint-Exupery

In the diagram below, a box like this:

```
+---+
|   | <-- the vertical bars might be missing
+---+
```

represents one byte; a box like this:

```
+==============+
|              |
+==============+
```

represents a variable number of bytes.

A lzip file consists of a series of "members" (compressed data sets). The members simply appear one after another in the file, with no additional information before, between, or after them.

Each member has the following structure:

```
+--+--+--+--+----+----+=============+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ID string | VN | DS | LZMA stream | CRC32 |   Data size   |  Member size  |
+--+--+--+--+----+----+=============+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

All multibyte values are stored in little endian order.

‘`ID string (the "magic" bytes)`’
: A four byte string, identifying the lzip format, with the value "LZIP" (0x4C, 0x5A, 0x49, 0x50).

‘`VN (version number, 1 byte)`’
: Just in case something needs to be modified in the future. 1 for now.

‘`DS (coded dictionary size, 1 byte)`’
: The dictionary size is calculated by taking a power of 2 (the base size) and substracting from it a fraction between 0/16 and 7/16 of the base size.
Bits 4-0 contain the base 2 logarithm of the base size (12 to 29).
Bits 7-5 contain the numerator of the fraction (0 to 7) to substract from the base size to obtain the dictionary size.
Example: 0xD3 = 2^19 - 6 * 2^15 = 512 KiB - 6 * 32 KiB = 320 KiB
Valid values for dictionary size range from 4 KiB to 512 MiB.

‘`LZMA stream`’
: The LZMA stream, finished by an end of stream marker. Uses default values for encoder properties. See Section “Stream format” in `lzip`, for a complete description.

‘`CRC32 (4 bytes)`’
: CRC of the uncompressed original data.

‘`Data size (8 bytes)`’
: Size of the uncompressed original data.

‘`Member size (8 bytes)`’
: Total size of the member, including header and trailer. This field acts as a distributed index, allows the verification of stream integrity, and facilitates safe recovery of undamaged members from multimember files.

4 Algorithm

In spite of its name (Lempel-Ziv-Markov chain-Algorithm), LZMA is not a concrete algorithm; it is more like "any algorithm using the LZMA coding scheme". For example, the option '`-0`' of lzip uses the scheme in almost the simplest way possible; issuing the longest match it can find, or a literal byte if it can't find a match. Inversely, a much more elaborated way of finding coding sequences of minimum size than the one currently used by lzip could be developed, and the resulting sequence could also be coded using the LZMA coding scheme.

Clzip currently implements two variants of the LZMA algorithm; fast (used by option '`-0`') and normal (used by all other compression levels).

The high compression of LZMA comes from combining two basic, well-proven compression ideas: sliding dictionaries (LZ77/78) and markov models (the thing used by every compression algorithm that uses a range encoder or similar order-0 entropy coder as its last stage) with segregation of contexts according to what the bits are used for.

Clzip is a two stage compressor. The first stage is a Lempel-Ziv coder, which reduces redundancy by translating chunks of data to their corresponding distance-length pairs. The second stage is a range encoder that uses a different probability model for each type of data; distances, lengths, literal bytes, etc.

Here is how it works, step by step:

1) The member header is written to the output stream.

2) The first byte is coded literally, because there are no previous bytes to which the match finder can refer to.

3) The main encoder advances to the next byte in the input data and calls the match finder.

4) The match finder fills an array with the minimum distances before the current byte where a match of a given length can be found.

5) Go back to step 3 until a sequence (formed of pairs, repeated distances and literal bytes) of minimum price has been formed. Where the price represents the number of output bits produced.

6) The range encoder encodes the sequence produced by the main encoder and sends the produced bytes to the output stream.

7) Go back to step 3 until the input data are finished or until the member or volume size limits are reached.

8) The range encoder is flushed.

9) The member trailer is written to the output stream.

10) If there are more data to compress, go back to step 1.

The ideas embodied in clzip are due to (at least) the following people: Abraham Lempel and Jacob Ziv (for the LZ algorithm), Andrey Markov (for the definition of Markov chains), G.N.N. Martin (for the definition of range encoding), Igor Pavlov (for putting all the above together in LZMA), and Julian Seward (for bzip2's CLI).

5 Extra data appended to the file

Sometimes extra data is found appended to a lzip file after the last member. Such trailing data may be:

- Padding added to make the file size a multiple of some block size, for example when writing to a tape.
- Garbage added by some not totally successful copy operation.
- Useful data added by the user; a cryptographically secure hash, a description of file contents, etc.
- Malicious data added to the file in order to make its total size and hash value (for a chosen hash) coincide with those of another file.
- In very rare cases, trailing data could be the corrupt header of another member. In multimember or concatenated files the probability of corruption happening in the magic bytes is 5 times smaller than the probability of getting a false positive caused by the corruption of the integrity information itself. Therefore it can be considered to be below the noise level.

Trailing data can be safely ignored in most cases. In some cases, like that of user-added data, it is expected to be ignored. In those cases where a file containing trailing data must be rejected, the option '`--trailing-error`' can be used. See [–trailing-error], page 5.

6 A small tutorial with examples

WARNING! Even if clzip is bug-free, other causes may result in a corrupt compressed file (bugs in the system libraries, memory errors, etc). Therefore, if the data you are going to compress are important, give the '`--keep`' option to clzip and don't remove the original file until you verify the compressed file with a command like '`clzip -cd file.lz | cmp file -`'.

Example 1: Replace a regular file with its compressed version '`file.lz`' and show the compression ratio.

```
clzip -v file
```

Example 2: Like example 1 but the created '`file.lz`' is multimember with a member size of 1 MiB. The compression ratio is not shown.

```
clzip -b 1MiB file
```

Example 3: Restore a regular file from its compressed version '`file.lz`'. If the operation is successful, '`file.lz`' is removed.

```
clzip -d file.lz
```

Example 4: Verify the integrity of the compressed file '`file.lz`' and show status.

```
clzip -tv file.lz
```

Example 5: Compress a whole device in /dev/sdc and send the output to '`file.lz`'.

```
clzip -c /dev/sdc > file.lz
```

Example 6: The right way of concatenating compressed files. See Chapter 5 [Trailing data], page 11.

```
Don't do this
  cat file1.lz file2.lz file3.lz | clzip -d
Do this instead
  clzip -cd file1.lz file2.lz file3.lz
```

Example 7: Decompress '`file.lz`' partially until 10 KiB of decompressed data are produced.

```
clzip -cd file.lz | dd bs=1024 count=10
```

Example 8: Decompress '`file.lz`' partially from decompressed byte 10000 to decompressed byte 15000 (5000 bytes are produced).

```
clzip -cd file.lz | dd bs=1000 skip=10 count=5
```

Example 9: Create a multivolume compressed tar archive with a volume size of 1440 KiB.

```
tar -c some_directory | clzip -S 1440KiB -o volume_name
```

Example 10: Extract a multivolume compressed tar archive.

```
clzip -cd volume_name*.lz | tar -xf -
```

Example 11: Create a multivolume compressed backup of a large database file with a volume size of 650 MB, where each volume is a multimember file with a member size of 32 MiB.

```
clzip -b 32MiB -S 650MB big_db
```

7 Reporting bugs

There are probably bugs in clzip. There are certainly errors and omissions in this manual. If you report them, they will get fixed. If you don't, no one will ever know about them and they will remain unfixed for all eternity, if not longer.

If you find a bug in clzip, please send electronic mail to `lzip-bug@nongnu.org`. Include the version number, which you can find by running `clzip --version`.