



Linux Ubuntu 26.04 Manual Pages on command 'zstd.1'

\$ man zstd.1

ZSTD(1) User Commands ZSTD(1)

NAME

zstd - zstd, zstdmt, unzstd, zstdcat - Compress or decompress .zst files

SYNOPSIS

zstdmt is equivalent to zstd -T0

unzstd is equivalent to zstd -d

zstdcat is equivalent to zstd -dcf

DESCRIPTION

zstd is a fast lossless compression algorithm and data compression tool, with command line syntax similar to gzip(1) and xz(1). It is based on the LZ77 family, with further FSE & huff0 entropy stages. zstd offers highly configurable compression speed, from fast modes at > 200 MB/s per core, to strong modes with excellent compression ratios. It also features a very fast decoder, with speeds > 500 MB/s per core, which remains roughly stable at all compression settings.

zstd command line syntax is generally similar to gzip, but features the following few differences:

- ? Source files are preserved by default. It's possible to remove them automatically by using the --rm command.
- ? When compressing a single file, zstd displays progress notifications and result summary by default. Use -q to turn them off.
- ? zstd displays a short help page when command line is an error. Use -q to turn it off.
- ? zstd does not accept input from console, though it does accept stdin when it's not the console.

? zstd does not store the input's filename or attributes, only its contents.

zstd processes each file according to the selected operation mode. If no files are given or file is -, zstd reads from standard input and writes the processed data to standard output.

zstd will refuse to write compressed data to standard output if it is a terminal: it will display an error message and skip the file. Similarly, zstd will refuse to read compressed data from standard input if it is a terminal.

Unless --stdout or -o is specified, files are written to a new file whose name is derived from the source file name:

? When compressing, the suffix .zst is appended to the source filename to get the target filename.

? When decompressing, the .zst suffix is removed from the source filename to get the tar? get filename

Concatenation with .zst Files

It is possible to concatenate multiple .zst files. zstd will decompress such agglomerated file as if it was a single .zst file.

OPTIONS

Integer Suffixes and Special Values

In most places where an integer argument is expected, an optional suffix is supported to easily indicate large integers. There must be no space between the integer and the suffix.

KiB Multiply the integer by 1,024 (2^{10}). Ki, K, and KB are accepted as synonyms for KiB.

MiB Multiply the integer by 1,048,576 (2^{20}). Mi, M, and MB are accepted as synonyms for MiB.

Operation Mode

If multiple operation mode options are given, the last one takes effect.

-z, --compress

Compress. This is the default operation mode when no operation mode option is specified and no other operation mode is implied from the command name (for example, unzstd implies --decompress).

-d, --decompress, --uncompress

Decompress.

-t, --test

Test the integrity of compressed files. This option is equivalent to --decompress

--stdout > /dev/null, decompressed data is discarded and checksummed for errors. No files are created or removed.

-b# Benchmark file(s) using compression level #. See BENCHMARK below for a description of this operation.

--train FILES

Use FILES as a training set to create a dictionary. The training set should contain a lot of small files (> 100). See DICTIONARY BUILDER below for a description of this operation.

-l, --list

Display information related to a zstd compressed file, such as size, ratio, and checksum. Some of these fields may not be available. This command's output can be augmented with the -v modifier.

Operation Modifiers

? -#: selects # compression level [1-19] (default: 3). Higher compression levels generally produce higher compression ratio at the expense of speed and memory. A rough rule of thumb is that compression speed is expected to be divided by 2 every 2 levels. Technically, each level is mapped to a set of advanced parameters (that can also be modified individually, see below). Because the compressor's behavior highly depends on the content to compress, there's no guarantee of a smooth progression from one level to another.

? --ultra: unlocks high compression levels 20+ (maximum 22), using a lot more memory. Note that decompression will also require more memory when using these levels.

? --fast[=#]: switch to ultra-fast compression levels. If =# is not present, it defaults to 1. The higher the value, the faster the compression speed, at the cost of some compression ratio. This setting overwrites compression level if one was set previously. Similarly, if a compression level is set after --fast, it overrides it.

? -T#, --threads=#: Compress using # working threads (default: 1). If # is 0, attempt to detect and use the number of physical CPU cores. In all cases, the number of threads is capped to ZSTDMT_NBWORKERS_MAX, which is either 64 in 32-bit mode, or 256 for 64-bit environments. This modifier does nothing if zstd is compiled without multithread support.

? --single-thread: Use a single thread for both I/O and compression. As compression is serialized with I/O, this can be slightly slower. Single-thread mode features significantly lower memory usage, which can be useful for systems with limited amount of memory.

ory, such as 32-bit systems.

Note 1: this mode is the only available one when multithread support is disabled.

Note 2: this mode is different from -T1, which spawns 1 compression thread in parallel with I/O. Final compressed result is also slightly different from -T1.

? --auto-threads={physical,logical} (default: physical): When using a default amount of threads via -T0, choose the default based on the number of detected physical or logical cores.

? --adapt[=min=#,max=#]: zstd will dynamically adapt compression level to perceived I/O conditions. Compression level adaptation can be observed live by using command -v. Adaptation can be constrained between supplied min and max levels. The feature works when combined with multi-threading and --long mode. It does not work with --single-thread. It sets window size to 8 MiB by default (can be changed manually, see wlog). Due to the chaotic nature of dynamic adaptation, compressed result is not reproducible.

Note: at the time of this writing, --adapt can remain stuck at low speed when combined with multiple worker threads (≥ 2).

? --long[=#]: enables long distance matching with # windowLog, if # is not present it defaults to 27. This increases the window size (windowLog) and memory usage for both the compressor and decompressor. This setting is designed to improve the compression ratio for files with long matches at a large distance.

Note: If windowLog is set to larger than 27, --long=windowLog or --memory=windowSize needs to be passed to the decompressor.

? -D DICT: use DICT as Dictionary to compress or decompress FILE(s)

? --patch-from FILE: Specify the file to be used as a reference point for zstd's diff engine. This is effectively dictionary compression with some convenient parameter selection, namely that windowSize > srcSize.

Note: cannot use both this and -D together.

Note: --long mode will be automatically activated if chainLog < fileLog (fileLog being the windowLog required to cover the whole file). You can also manually force it.

Note: up to level 15, you can use --patch-from in --single-thread mode to improve compression ratio marginally at the cost of speed. Using '--single-thread' above level 15 will lead to lower compression ratios.

Note: for level 19, you can get increased compression ratio at the cost of speed by specifying --zstd=targetLength= to be something large (i.e. 4096), and by setting a

large --zstd=chainLog=.

- ? --rsyncable: zstd will periodically synchronize the compression state to make the compressed file more rsync-friendly. There is a negligible impact to compression ratio, and a potential impact to compression speed, perceptible at higher speeds, for example when combining --rsyncable with many parallel worker threads. This feature does not work with --single-thread. You probably don't want to use it with long range mode, since it will decrease the effectiveness of the synchronization points, but your mileage may vary.
- ? -C, --[no-]check: add integrity check computed from uncompressed data (default: enabled)
- ? --[no-]content-size: enable / disable whether or not the original size of the file is placed in the header of the compressed file. The default option is --content-size (meaning that the original size will be placed in the header).
- ? --no-dictID: do not store dictionary ID within frame header (dictionary compression). The decoder will have to rely on implicit knowledge about which dictionary to use, it won't be able to check if it's correct.
- ? -M#, --memory=#: Set a memory usage limit. By default, zstd uses 128 MiB for decompression as the maximum amount of memory the decompressor is allowed to use, but you can override this manually if need be in either direction (i.e. you can increase or decrease it).
- This is also used during compression when using with --patch-from=. In this case, this parameter overrides that maximum size allowed for a dictionary. (128 MiB).
- Additionally, this can be used to limit memory for dictionary training. This parameter overrides the default limit of 2 GiB. zstd will load training samples up to the memory limit and ignore the rest.
- ? --stream-size=#: Sets the pledged source size of input coming from a stream. This value must be exact, as it will be included in the produced frame header. Incorrect stream sizes will cause an error. This information will be used to better optimize compression parameters, resulting in better and potentially faster compression, especially for smaller source sizes.
- ? --size-hint=#: When handling input from a stream, zstd must guess how large the source size will be when optimizing compression parameters. If the stream size is relatively small, this guess may be a poor one, resulting in a higher compression ratio than expected. This feature allows for controlling the guess when needed. Exact guesses result in better compression ratios. Overestimates result in slightly degraded compression ratio.

tios, while underestimates may result in significant degradation.

- ? --target-compressed-block-size=#: Attempt to produce compressed blocks of approximately this size. This will split larger blocks in order to approach this target. This feature is notably useful for improved latency, when the receiver can leverage receiving early incomplete data. This parameter defines a loose target: compressed blocks will target this size "on average", but individual blocks can still be larger or smaller. Enabling this feature can decrease compression speed by up to ~10% at level 1. Higher levels will see smaller relative speed regression, becoming invisible at higher settings.
- ? -f, --force: disable input and output checks. Allows overwriting existing files, input from console, output to stdout, operating on links, block devices, etc. During decompression and when the output destination is stdout, pass-through unrecognized formats as-is.
- ? -c, --stdout: write to standard output (even if it is the console); keep original files (disable --rm).
- ? -o FILE: save result into FILE. Note that this operation is in conflict with -c. If both operations are present on the command line, the last expressed one wins.
- ? --[no-]sparse: enable / disable sparse FS support, to make files with many zeroes smaller on disk. Creating sparse files may save disk space and speed up decompression by reducing the amount of disk I/O. default: enabled when output is into a file, and disabled when output is stdout. This setting overrides default and can force sparse mode over stdout.
- ? --[no-]pass-through enable / disable passing through uncompressed files as-is. During decompression when pass-through is enabled, unrecognized formats will be copied as-is from the input to the output. By default, pass-through will occur when the output destination is stdout and the force (-f) option is set.
- ? --rm: remove source file(s) after successful compression or decompression. This command is silently ignored if output is stdout. If used in combination with -o, triggers a confirmation prompt (which can be silenced with -f), as this is a destructive operation.
- ? -k, --keep: keep source file(s) after successful compression or decompression. This is the default behavior.
- ? -r: operate recursively on directories. It selects all files in the named directory and all its subdirectories. This can be useful both to reduce command line typing, and to circumvent shell expansion limitations, when there are a lot of files and naming breaks

the maximum size of a command line.

- ? --filelist FILE read a list of files to process as content from FILE. Format is compatible with ls output, with one file per line.
- ? --output-dir-flat DIR: resulting files are stored into target DIR directory, instead of same directory as origin file. Be aware that this command can introduce name collision issues, if multiple files, from different directories, end up having the same name. Collision resolution ensures first file with a given name will be present in DIR, while in combination with -f, the last file will be present instead.
- ? --output-dir-mirror DIR: similar to --output-dir-flat, the output files are stored underneath target DIR directory, but this option will replicate input directory hierarchy into output DIR.

If input directory contains ".", the files in this directory will be ignored. If input directory is an absolute directory (i.e. "/var/tmp/abc"), it will be stored into the "output-dir/var/tmp/abc". If there are multiple input files or directories, name collision resolution will follow the same rules as --output-dir-flat.
- ? --format=FORMAT: compress and decompress in other formats. If compiled with support, zstd can compress to or decompress from other compression algorithm formats. Possibly available options are zstd, gzip, xz, lzma, and lz4. If no such format is provided, zstd is the default.
- ? -h/-H, --help: display help/long help and exit
- ? -V, --version: display version number and immediately exit. note that, since it exits, flags specified after -V are effectively ignored. Advanced: -vV also displays supported formats. -vvV also displays POSIX support. -qV will only display the version number, suitable for machine reading.
- ? -v, --verbose: verbose mode, display more information
- ? -q, --quiet: suppress warnings, interactivity, and notifications. specify twice to suppress errors too.
- ? --no-progress: do not display the progress bar, but keep all other messages.
- ? --show-default-cparams: shows the default compression parameters that will be used for a particular input file, based on the provided compression level and the input size. If the provided file is not a regular file (e.g. a pipe), this flag will output the parameters used for inputs of unknown size.
- ? --exclude-compressed: only compress files that are not already compressed.

? --: All arguments after -- are treated as files

gzip Operation Modifiers

When invoked via a gzip symlink, zstd will support further options that intend to mimic the gzip behavior:

-n, --no-name

do not store the original filename and timestamps when compressing a file. This is the default behavior and hence a no-op.

--best alias to the option -9.

Environment Variables

Employing environment variables to set parameters has security implications. Therefore, this avenue is intentionally limited. Only ZSTD_CLEVEL and ZSTD_NBTHREADS are currently supported. They set the default compression level and number of threads to use during compression, respectively.

ZSTD_CLEVEL can be used to set the level between 1 and 19 (the "normal" range). If the value of ZSTD_CLEVEL is not a valid integer, it will be ignored with a warning message.

ZSTD_CLEVEL just replaces the default compression level (3).

ZSTD_NBTHREADS can be used to set the number of threads zstd will attempt to use during compression. If the value of ZSTD_NBTHREADS is not a valid unsigned integer, it will be ignored with a warning message. ZSTD_NBTHREADS has a default value of (1), and is capped at ZSTD_MAX_NBWORKERS_MAX==200. zstd must be compiled with multithread support for this variable to have any effect.

They can both be overridden by corresponding command line arguments: -# for compression level and -T# for number of compression threads.

ADVANCED COMPRESSION OPTIONS

zstd provides 22 predefined regular compression levels plus the fast levels. A compression level is translated internally into multiple advanced parameters that control the behavior of the compressor (one can observe the result of this translation with --show-default-cparams). These advanced parameters can be overridden using advanced compression options.

--zstd[=options]:

The options are provided as a comma-separated list. You may specify only the options you want to change and the rest will be taken from the selected or default compression level.

The list of available options:

strategy=strat, strat=strat

Specify a strategy used by a match finder.

There are 9 strategies numbered from 1 to 9, from fastest to strongest: 1=ZSTD_fast, 2=ZSTD_dfast, 3=ZSTD_greedy, 4=ZSTD_lazy, 5=ZSTD_lazy2, 6=ZSTD_btlazy2, 7=ZSTD_btopt, 8=ZSTD_btultra, 9=ZSTD_btultra2.

windowLog=wlog, wlog=wlog

Specify the maximum number of bits for a match distance.

The higher number of increases the chance to find a match which usually improves compression ratio. It also increases memory requirements for the compressor and decompressor. The minimum wlog is 10 (1 KiB) and the maximum is 30 (1 GiB) on 32-bit platforms and 31 (2 GiB) on 64-bit platforms.

Note: If windowLog is set to larger than 27, --long=windowLog or --memory=windowSize needs to be passed to the decompressor.

hashLog=hlog, hlog=hlog

Specify the maximum number of bits for a hash table.

Bigger hash tables cause fewer collisions which usually makes compression faster, but requires more memory during compression.

The minimum hlog is 6 (64 entries / 256 B) and the maximum is 30 (1B entries / 4 GiB).

chainLog=clog, clog=clog

Specify the maximum number of bits for the secondary search structure, whose form depends on the selected strategy.

Higher numbers of bits increases the chance to find a match which usually improves compression ratio. It also slows down compression speed and increases memory requirements for compression. This option is ignored for the ZSTD_fast strategy, which only has the primary hash table.

The minimum clog is 6 (64 entries / 256 B) and the maximum is 29 (512M entries / 2 GiB) on 32-bit platforms and 30 (1B entries / 4 GiB) on 64-bit platforms.

searchLog=slog, slog=slog

Specify the maximum number of searches in a hash chain or a binary tree using logarithmic scale.

More searches increases the chance to find a match which usually increases compression ratio but decreases compression speed.

The minimum slog is 1 and the maximum is 'windowLog' - 1.

minMatch=mml, mml=mml

Specify the minimum searched length of a match in a hash table.

Larger search lengths usually decrease compression ratio but improve decompression speed.

The minimum mml is 3 and the maximum is 7.

targetLength=tlen, tlen=tlen

The impact of this field vary depending on selected strategy.

For ZSTD_btopt, ZSTD_btultra and ZSTD_btultra2, it specifies the minimum match length that causes match finder to stop searching. A larger targetLength usually improves compression ratio but decreases compression speed.

For ZSTD_fast, it triggers ultra-fast mode when > 0. The value represents the amount of data skipped between match sampling. Impact is reversed: a larger targetLength increases compression speed but decreases compression ratio.

For all other strategies, this field has no impact.

The minimum tlen is 0 and the maximum is 128 KiB.

overlapLog=ovlog, ovlog=ovlog

Determine overlapSize, amount of data reloaded from previous job. This parameter is only available when multithreading is enabled. Reloading more data improves compression ratio, but decreases speed.

The minimum ovlog is 0, and the maximum is 9. 1 means "no overlap", hence completely independent jobs. 9 means "full overlap", meaning up to windowSize is reloaded from previous job. Reducing ovlog by 1 reduces the reloaded amount by a factor 2. For example, 8 means "windowSize/2", and 6 means "windowSize/8". Value 0 is special and means "default": ovlog is automatically determined by zstd. In which case, ovlog will range from 6 to 9, depending on selected strat.

ldmHashLog=lhlog, lhlog=lhlog

Specify the maximum size for a hash table used for long distance matching.

This option is ignored unless long distance matching is enabled.

Bigger hash tables usually improve compression ratio at the expense of more memory during compression and a decrease in compression speed.

The minimum lhlog is 6 and the maximum is 30 (default: 20).

ldmMinMatch=lmml, lmml=lmml

Specify the minimum searched length of a match for long distance matching.

This option is ignored unless long distance matching is enabled.

Larger/very small values usually decrease compression ratio.

The minimum `lmm1` is 4 and the maximum is 4096 (default: 64).

`ldmBucketSizeLog=lblog, lblog=lblog`

Specify the size of each bucket for the hash table used for long distance matching.

This option is ignored unless long distance matching is enabled.

Larger bucket sizes improve collision resolution but decrease compression speed.

The minimum `lblog` is 1 and the maximum is 8 (default: 3).

`ldmHashRateLog=lhrlog, lhrlog=lhrlog`

Specify the frequency of inserting entries into the long distance matching hash table.

This option is ignored unless long distance matching is enabled.

Larger values will improve compression speed. Deviating far from the default value will likely result in a decrease in compression ratio.

The default value is `wlog - lblog`.

Example

The following parameters sets advanced compression options to something similar to predefined level 19 for files bigger than 256 KB:

```
--zstd=wlog=23,clog=23,hlog=22,slog=6,mml=3,tlen=48,strat=6
```

`-B#`:

Specify the size of each compression job. This parameter is only available when multi-threading is enabled. Each compression job is run in parallel, so this value directly impacts the nb of active threads. Default job size varies depending on compression level (generally $4 * \text{windowSize}$). `-B#` makes it possible to manually select a custom size. Note that job size must respect a minimum value which is enforced transparently. This minimum is either 512 KB, or `overlapSize`, whichever is largest. Different job sizes will lead to non-identical compressed frames.

DICTIONARY BUILDER

`zstd` offers dictionary compression, which greatly improves efficiency on small files and messages. It's possible to train `zstd` with a set of samples, the result of which is saved into a file called a dictionary. Then, during compression and decompression, reference the same dictionary, using command `-D dictionaryFileName`. Compression of small files similar to

the sample set will be greatly improved.

`--train FILES`

Use FILES as training set to create a dictionary. The training set should ideally contain a lot of samples (> 100), and weight typically 100x the target dictionary size (for example, ~10 MB for a 100 KB dictionary). `--train` can be combined with `-r` to indicate a directory rather than listing all the files, which can be useful to circumvent shell expansion limits.

Since dictionary compression is mostly effective for small files, the expectation is that the training set will only contain small files. In the case where some samples happen to be large, only the first 128 KiB of these samples will be used for training.

`--train` supports multithreading if `zstd` is compiled with threading support (default).

Additional advanced parameters can be specified with `--train-fastcover`. The legacy dictionary builder can be accessed with `--train-legacy`. The slower cover dictionary builder can be accessed with `--train-cover`. Default `--train` is equivalent to `--train-fastcover=d=8,steps=4`.

`-o FILE`

Dictionary saved into FILE (default name: dictionary).

`--maxdict=#`

Limit dictionary to specified size (default: 112640 bytes). As usual, quantities are expressed in bytes by default, and it's possible to employ suffixes (like KB or MB) to specify larger values.

`-#` Use # compression level during training (optional). Will generate statistics more tuned for selected compression level, resulting in a small compression ratio improvement for this level.

`-B#` Split input files into blocks of size # (default: no split)

`-M#, --memory=#`

Limit the amount of sample data loaded for training (default: 2 GB). Note that the default (2 GB) is also the maximum. This parameter can be useful in situations where the training set size is not well controlled and could be potentially very large. Since speed of the training process is directly correlated to the size of the training sample set, a smaller sample set leads to faster training.

In situations where the training set is larger than maximum memory, the CLI will run?

domly select samples among the available ones, up to the maximum allowed memory bud? get. This is meant to improve dictionary relevance by mitigating the potential impact of clustering, such as selecting only files from the beginning of a list sorted by modification date, or sorted by alphabetical order. The randomization process is deterministic, so training of the same list of files with the same parameters will lead to the creation of the same dictionary.

`--dictID=#`

A dictionary ID is a locally unique ID. The decoder will use this value to verify it is using the right dictionary. By default, zstd will create a 4-bytes random number ID. It's possible to provide an explicit number ID instead. It's up to the dictionary manager to not assign twice the same ID to 2 different dictionaries. Note that short numbers have an advantage: an ID < 256 will only need 1 byte in the compressed frame header, and an ID < 65536 will only need 2 bytes. This compares favorably to 4 bytes default.

Note that RFC8878 reserves IDs less than 32768 and greater than or equal to 2^{31} , so they should not be used in public.

`--train-cover[=k#,d=#,steps=#,split=#,shrink[=#]]`

Select parameters for the default dictionary builder algorithm named cover. If d is not specified, then it tries d = 6 and d = 8. If k is not specified, then it tries steps values in the range [50, 2000]. If steps is not specified, then the default value of 40 is used. If split is not specified or split <= 0, then the default value of 100 is used. Requires that d <= k. If shrink flag is not used, then the default value for shrinkDict of 0 is used. If shrink is not specified, then the default value for shrinkDictMaxRegression of 1 is used.

Selects segments of size k with highest score to put in the dictionary. The score of a segment is computed by the sum of the frequencies of all the subsegments of size d.

Generally d should be in the range [6, 8], occasionally up to 16, but the algorithm will run faster with d <= 8. Good values for k vary widely based on the input data, but a safe range is [2 * d, 2000]. If split is 100, all input samples are used for both training and testing to find optimal d and k to build dictionary. Supports multithreading if zstd is compiled with threading support. Having shrink enabled takes a truncated dictionary of minimum size and doubles in size until compression ratio of the truncated dictionary is at most shrinkDictMaxRegression% worse than the compressed

sion ratio of the largest dictionary.

Examples:

```
zstd --train-cover FILES
```

```
zstd --train-cover=k=50,d=8 FILES
```

```
zstd --train-cover=d=8,steps=500 FILES
```

```
zstd --train-cover=k=50 FILES
```

```
zstd --train-cover=k=50,split=60 FILES
```

```
zstd --train-cover=shrink FILES
```

```
zstd --train-cover=shrink=2 FILES
```

```
--train-fastcover[=k#,d=#,f=#,steps=#,split=#,accel=#]
```

Same as cover but with extra parameters f and accel and different default value of

split. If split is not specified, then it tries split = 75. If f is not specified,

then it tries f = 20. Requires that $0 < f < 32$. If accel is not specified, then it

tries accel = 1. Requires that $0 < \text{accel} \leq 10$. Requires that d = 6 or d = 8.

f is log of size of array that keeps track of frequency of subsegments of size d. The

subsegment is hashed to an index in the range $[0, 2^f - 1]$. It is possible that 2 dif-

ferent subsegments are hashed to the same index, and they are considered as the same

subsegment when computing frequency. Using a higher f reduces collision but takes

longer.

Examples:

```
zstd --train-fastcover FILES
```

```
zstd --train-fastcover=d=8,f=15,accel=2 FILES
```

```
--train-legacy[=selectivity=#]
```

Use legacy dictionary builder algorithm with the given dictionary selectivity (de-

fault: 9). The smaller the selectivity value, the denser the dictionary, improving

its efficiency but reducing its achievable maximum size. --train-legacy=s=# is also

accepted.

Examples:

```
zstd --train-legacy FILES
```

```
zstd --train-legacy=selectivity=8 FILES
```

BENCHMARK

The zstd CLI provides a benchmarking mode that can be used to easily find suitable compres-

sion parameters, or alternatively to benchmark a computer's performance. zstd -b [FILE(s)]

will benchmark zstd for both compression and decompression using default compression level.

Note that results are very dependent on the content being compressed. It's possible to pass multiple files to the benchmark, and even a directory with -r DIRECTORY. When no FILE is provided, the benchmark will use a procedurally generated lorem ipsum text.

- ? -b#: benchmark file(s) using compression level #
- ? -e#: benchmark file(s) using multiple compression levels, from -b# to -e# (inclusive)
- ? -d: benchmark decompression speed only (requires providing a zstd-compressed content)
- ? -i#: minimum evaluation time, in seconds (default: 3s), benchmark mode only
- ? -B#, --block-size=#: cut file(s) into independent chunks of size # (default: no chunking)
- ? -S: output one benchmark result per input file (default: consolidated result)
- ? -D dictionary benchmark using dictionary
- ? --priority=rt: set process priority to real-time (Windows)

Beyond compression levels, benchmarking is also compatible with other parameters, such as number of threads (-T#), advanced compression parameters (--zstd=###), dictionary compression (-D dictionary), or even disabling checksum verification for example.

Output Format: CompressionLevel#Filename: InputSize -> OutputSize (CompressionRatio), CompressionSpeed, DecompressionSpeed

Methodology: For speed measurement, the entire input is compressed/decompressed in-memory to measure speed. A run lasts at least 1 sec, so when files are small, they are compressed/decompressed several times per run, in order to improve measurement accuracy.

SEE ALSO

zstdgrep(1), zstdless(1), gzip(1), xz(1)

The zstandard format is specified in Y. Collet, "Zstandard Compression and the 'application/zstd' Media Type", <https://www.ietf.org/rfc/rfc8878.txt>, Internet RFC 8878 (February 2021).

BUGS

Report bugs at: <https://github.com/facebook/zstd/issues>

AUTHOR

Yann Collet