NAME

xz, unxz, xzcat, lzma, unlzma, lzcat − Compress or decompress .xz and .lzma files

SYNOPSIS

xz [*option*]... [*file*]...

unxz is equivalent to **xz −−decompress**.
xzcat is equivalent to **xz −−decompress −−stdout**.
lzma is equivalent to **xz −−format=lzma**.
unlzma is equivalent to **xz −−format=lzma −−decompress**.
lzcat is equivalent to **xz −−format=lzma −−decompress −−stdout**.

When writing scripts that need to decompress files, it is recommended to always use the name **xz** with appropriate arguments (**xz −d** or **xz −dc**) instead of the names **unxz** and **xzcat.**

DESCRIPTION

xz is a general-purpose data compression tool with command line syntax similar to **gzip**(1) and **bzip2**(1). The native file format is the **.xz** format, but also the legacy **.lzma** format and raw compressed streams with no container format headers are supported.

xz compresses or decompresses each *file* according to the selected operation mode. If no *files* are given or *file* is −, **xz** reads from standard input and writes the processed data to standard output. **xz** will refuse (display an error and skip the *file*) to write compressed data to standard output if it is a terminal. Similarly, **xz** will refuse to read compressed data from standard input if it is a terminal.

Unless **−−stdout** is specified, *files* other than − are written to a new file whose name is derived from the source *file* name:

- When compressing, the suffix of the target file format (**.xz** or **.lzma**) is appended to the source filename to get the target filename.
- When decompressing, the **.xz** or **.lzma** suffix is removed from the filename to get the target filename. **xz** also recognizes the suffixes **.txz** and **.tlz**, and replaces them with the **.tar** suffix.

If the target file already exists, an error is displayed and the *file* is skipped.

Unless writing to standard output, **xz** will display a warning and skip the *file* if any of the following applies:

- *File* is not a regular file. Symbolic links are not followed, thus they are not considered to be regular files.
- *File* has more than one hard link.
- *File* has setuid, setgid, or sticky bit set.
- The operation mode is set to compress, and the *file* already has a suffix of the target file format (**.xz** or **.txz** when compressing to the **.xz** format, and **.lzma** or **.tlz** when compressing to the **.lzma** format).
- The operation mode is set to decompress, and the *file* doesn't have a suffix of any of the supported file formats (**.xz**, **.txz**, **.lzma**, or **.tlz**).

After successfully compressing or decompressing the *file*, **xz** copies the owner, group, permissions, access time, and modification time from the source *file* to the target file. If copying the group fails, the permissions are modified so that the target file doesn't become accessible to users who didn't have permission to access the source *file*. **xz** doesn't support copying other metadata like access control lists or extended attributes yet.

Once the target file has been successfully closed, the source *file* is removed unless **−−keep** was specified. The source *file* is never removed if the output is written to standard output.

Sending **SIGINFO** or **SIGUSR1** to the **xz** process makes it print progress information to standard error. This has only limited use since when standard error is a terminal, using **−−verbose** will display an automatically updating progress indicator.

Memory usage

The memory usage of **xz** varies from a few hundred kilobytes to several gigabytes depending on the compression settings. The settings used when compressing a file affect also the memory usage of the decompressor. Typically the decompressor needs only 5 % to 20 % of the amount of RAM that the compressor needed when creating the file. Still, the worst-case memory usage of the decompressor is several gigabytes.

To prevent uncomfortable surprises caused by huge memory usage, **xz** has a built-in memory usage limiter. While some operating systems provide ways to limit the memory usage of processes, relying on it wasn't deemed to be flexible enough. The default limit depends on the total amount of physical RAM:

- If 40 % of RAM is at least 80 MiB, 40 % of RAM is used as the limit.
- If 80 % of RAM is over 80 MiB, 80 MiB is used as the limit.
- Otherwise 80 % of RAM is used as the limit.

When compressing, if the selected compression settings exceed the memory usage limit, the settings are automatically adjusted downwards and a notice about this is displayed. As an exception, if the memory usage limit is exceeded when compressing with **−−format=raw**, an error is displayed and **xz** will exit with exit status **1**.

If source *file* cannot be decompressed without exceeding the memory usage limit, an error message is displayed and the file is skipped. Note that compressed files may contain many blocks, which may have been compressed with different settings. Typically all blocks will have roughly the same memory requirements, but it is possible that a block later in the file will exceed the memory usage limit, and an error about too low memory usage limit gets displayed after some data has already been decompressed.

The absolute value of the active memory usage limit can be seen with **−−info-memory** or near the bottom of the output of **−−long−help**. The default limit can be overridden with **−−memory=***limit*.

Concatenation and padding with .xz files

It is possible to concatenate **.xz** files as is. **xz** will decompress such files as if they were a single **.xz** file.

It is possible to insert padding between the concenated parts or after the last part. The padding must be null bytes and the size of the padding must be a multiple of four bytes. This can be useful if the .xz file is stored on a medium that stores file sizes e.g. as 512-byte blocks.

Concatenation and padding are not allowed with **.lzma** files or raw streams.

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 The integer is multiplied by 1,024 (2ˆ10). Also **Ki**, **k**, **kB**, **K**, and **KB** are accepted as synonyms for **KiB**.

MiB The integer is multiplied by 1,048,576 (2ˆ20). Also **Mi**, **m**, **M**, and **MB** are accepted as synonyms for **MiB**.

GiB The integer is multiplied by 1,073,741,824 (2ˆ30). Also **Gi**, **g**, **G**, and **GB** are accepted as synonyms for **GiB**.

A special value **max** can be used to indicate the maximum integer value supported by the option.

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, **unxz** implies **−−decompress**).

−d, **−−decompress**, **−−uncompress**
Decompress.

−t, **−−test**
Test the integrity of compressed *files*. No files are created or removed. This option is equivalent to **−−decompress −−stdout** except that the decompressed data is discarded instead of being written to standard output.

−l, **−−list**
List information about compressed *files*. No uncompressed output is produced, and no files are created or removed. In list mode, the program cannot read the compressed data from standard input or from other unseekable sources.

The default listing shows basic information about *files*, one file per line. To get more detailed information, use also the **−−verbose** option. For even more information, use **−−verbose** twice, but note that it may be slow, because getting all the extra information requires many seeks. The width of verbose output exceeds 80 characters, so piping the output to e.g. **less −S** may be convenient if the terminal isn't wide enough.

The exact output may vary between **xz** versions and different locales. To get machine-readable output, **−−robot −−list** should be used.

Operation modifiers

−k, **−−keep**
Keep (don't delete) the input files.

−f, **−−force**
This option has several effects:

- If the target file already exists, delete it before compressing or decompressing.
- Compress or decompress even if the input is a symbolic link to a regular file, has more than one hard link, or has setuid, setgid, or sticky bit set. The setuid, setgid, and sticky bits are not copied to the target file.
- If combined with **−−decompress −−stdout** and **xz** doesn't recognize the type of the source file, **xz** will copy the source file as is to standard output. This allows using **xzcat −-force** like **cat**(1) for files that have not been compressed with **xz**. Note that in future, **xz** might support new compressed file formats, which may make **xz** decompress more types of files instead of copying them as is to standard output. **−−format=** *format* can be used to restrict **xz** to decompress only a single file format.

−c, **−−stdout**, **−−to-stdout**
Write the compressed or decompressed data to standard output instead of a file. This implies **−−keep**.

−−no−sparse
Disable creation of sparse files. By default, if decompressing into a regular file, **xz** tries to make the file sparse if the decompressed data contains long sequences of binary zeros. It works also when writing to standard output as long as standard output is connected to a regular file, and certain additional conditions are met to make it safe. Creating sparse files may save disk space and speed up the decompression by reducing the amount of disk I/O.

−S *.suf*, **−−suffix=***.suf*
When compressing, use *.suf* as the suffix for the target file instead of **.xz** or **.lzma**. If not writing to standard output and the source file already has the suffix *.suf*, a warning is displayed and the file is skipped.

When decompressing, recognize also files with the suffix *.suf* in addition to files with the **.xz**, **.txz**, **.lzma**, or **.tlz** suffix. If the source file has the suffix *.suf*, the suffix is removed to get the target filename.

When compressing or decompressing raw streams (**−−format=raw**), the suffix must always be specified unless writing to standard output, because there is no default suffix for raw streams.

−−files[=*file*]
Read the filenames to process from *file*; if *file* is omitted, filenames are read from standard input. Filenames must be terminated with the newline character. A dash (**−**) is taken as a regular filename; it doesn't mean standard input. If filenames are given also as command line arguments, they are processed before the filenames read from *file*.

−−files0[=*file*]
This is identical to **−−files**[=*file*] except that the filenames must be terminated with the null character.

Basic file format and compression options

−F *format*, **−−format**=*format*
Specify the file format to compress or decompress:

- **auto**: This is the default. When compressing, **auto** is equivalent to **xz**. When decompressing, the format of the input file is automatically detected. Note that raw streams (created with **−−format=raw**) cannot be auto-detected.
- **xz**: Compress to the **.xz** file format, or accept only **.xz** files when decompressing.
- **lzma** or **alone**: Compress to the legacy **.lzma** file format, or accept only **.lzma** files when decompressing. The alternative name **alone** is provided for backwards compatibility with LZMA Utils.
- **raw**: Compress or uncompress a raw stream (no headers). This is meant for advanced users only. To decode raw streams, you need to set not only **−−format=raw** but also specify the filter chain, which would normally be stored in the container format headers.

−C *check*, **−−check**=*check*
Specify the type of the integrity check, which is calculated from the uncompressed data. This option has an effect only when compressing into the **.xz** format; the **.lzma** format doesn't support integrity checks. The integrity check (if any) is verified when the **.xz** file is decompressed.

Supported *check* types:

- **none**: Don't calculate an integrity check at all. This is usually a bad idea. This can be useful when integrity of the data is verified by other means anyway.
- **crc32**: Calculate CRC32 using the polynomial from IEEE-802.3 (Ethernet).
- **crc64**: Calculate CRC64 using the polynomial from ECMA-182. This is the default, since it is slightly better than CRC32 at detecting damaged files and the speed difference is negligible.
- **sha256**: Calculate SHA-256. This is somewhat slower than CRC32 and CRC64.

Integrity of the **.xz** headers is always verified with CRC32. It is not possible to change or disable it.

−0 ... **−9**
Select compression preset. If a preset level is specified multiple times, the last one takes effect.

The compression preset levels can be categorised roughly into three categories:

−0 ... **−2**
Fast presets with relatively low memory usage. **−1** and **−2** should give compression speed and ratios comparable to **bzip2 −1** and **bzip2 −9**, respectively. Currently **−0** is not very good (not much faster than **−1** but much worse compression). In future, **−0** may be indicate some fast algorithm instead of LZMA2.

−3 ... **−5**
Good compression ratio with low to medium memory usage. These are significantly slower than levels 0−2.

−6 ... **−9**

Excellent compression with medium to high memory usage. These are also slower than the lower preset levels. The default is **−6**. Unless you want to maximize the compression ratio, you probably don't want a higher preset level than **−7** due to speed and memory usage.

The exact compression settings (filter chain) used by each preset may vary between **xz** versions. The settings may also vary between files being compressed, if **xz** determines that modified settings will probably give better compression ratio without significantly affecting compression time or memory usage.

Because the settings may vary, the memory usage may vary too. The following table lists the maximum memory usage of each preset level, which won't be exceeded even in future versions of **xz**.

FIXME: The table below is just a rough idea.

Preset	Compression	Decompression
−0	6 MiB	1 MiB
−1	6 MiB	1 MiB
−2	10 MiB	1 MiB
−3	20 MiB	2 MiB
−4	30 MiB	3 MiB
−5	60 MiB	6 MiB
−6	100 MiB	10 MiB
−7	200 MiB	20 MiB
−8	400 MiB	40 MiB
−9	800 MiB	80 MiB

When compressing, **xz** automatically adjusts the compression settings downwards if the memory usage limit would be exceeded, so it is safe to specify a high preset level even on systems that don't have lots of RAM.

−−fast and **−−best**

These are somewhat misleading aliases for **−0** and **−9**, respectively. These are provided only for backwards compatibility with LZMA Utils. Avoid using these options.

Especially the name of **−−best** is misleading, because the definition of best depends on the input data, and that usually people don't want the very best compression ratio anyway, because it would be very slow.

−e, **−−extreme**

Modify the compression preset (**−0** ... **−9**) so that a little bit better compression ratio can be achieved without increasing memory usage of the compressor or decompressor (exception: compressor memory usage may increase a little with presets **−0** ... **−2**). The downside is that the compression time will increase dramatically (it can easily double).

−M *limit*, **−−memory=***limit*

Set the memory usage limit. If this option is specified multiple times, the last one takes effect. The *limit* can be specified in multiple ways:

- The *limit* can be an absolute value in bytes. Using an integer suffix like **MiB** can be useful. Example: **−−memory=80MiB**
- The *limit* can be specified as a percentage of physical RAM. Example: **−−memory=70%**
- The *limit* can be reset back to its default value by setting it to **0**. See the section **Memory usage** for how the default limit is defined.
- The memory usage limiting can be effectively disabled by setting *limit* to **max**. This isn't recommended. It's usually better to use, for example, **−−memory=90%**.

The current *limit* can be seen near the bottom of the output of the **−−long-help** option.

−T *threads*, **−−threads=***threads*

Specify the maximum number of worker threads to use. The default is the number of available CPU cores. You can see the current value of *threads* near the end of the output of the **−−long−help** option.

The actual number of worker threads can be less than *threads* if using more threads would exceed the memory usage limit. In addition to CPU-intensive worker threads, **xz** may use a few auxiliary threads, which don't use a lot of CPU time.

Multithreaded compression and decompression are not implemented yet, so this option has no effect for now.

Custom compressor filter chains

A custom filter chain allows specifying the compression settings in detail instead of relying on the settings associated to the preset levels. When a custom filter chain is specified, the compression preset level options (**−0** ... **−9** and **−−extreme**) are silently ignored.

A filter chain is comparable to piping on the UN*X command line. When compressing, the uncompressed input goes to the first filter, whose output goes to the next filter (if any). The output of the last filter gets written to the compressed file. The maximum number of filters in the chain is four, but typically a filter chain has only one or two filters.

Many filters have limitations where they can be in the filter chain: some filters can work only as the last filter in the chain, some only as a non-last filter, and some work in any position in the chain. Depending on the filter, this limitation is either inherent to the filter design or exists to prevent security issues.

A custom filter chain is specified by using one or more filter options in the order they are wanted in the filter chain. That is, the order of filter options is significant! When decoding raw streams (**−−format=raw**), the filter chain is specified in the same order as it was specified when compressing.

Filters take filter-specific *options* as a comma-separated list. Extra commas in *options* are ignored. Every option has a default value, so you need to specify only those you want to change.

−−lzma1[=*options*], **−−lzma2**[=*options*]

Add LZMA1 or LZMA2 filter to the filter chain. These filter can be used only as the last filter in the chain.

LZMA1 is a legacy filter, which is supported almost solely due to the legacy **.lzma** file format, which supports only LZMA1. LZMA2 is an updated version of LZMA1 to fix some practical issues of LZMA1. The **.xz** format uses LZMA2, and doesn't support LZMA1 at all. Compression speed and ratios of LZMA1 and LZMA2 are practically the same.

LZMA1 and LZMA2 share the same set of *options*:

preset=*preset*

Reset all LZMA1 or LZMA2 *options* to *preset*. *Preset* consist of an integer, which may be followed by single-letter preset modifiers. The integer can be from **0** to **9**, matching the command line options **−0** ... **−9**. The only supported modifier is currently **e**, which matches **−−extreme**.

The default *preset* is **6**, from which the default values for the rest of the LZMA1 or LZMA2 *options* are taken.

dict=*size*

Dictionary (history buffer) size indicates how many bytes of the recently processed uncompressed data is kept in memory. One method to reduce size of the uncompressed data is to store distance-length pairs, which indicate what data to repeat from the dictionary buffer. The bigger the dictionary, the better the compression ratio usually is, but dictionaries bigger than the uncompressed data are waste of RAM.

Typical dictionary size is from 64 KiB to 64 MiB. The minimum is 4 KiB. The maximum for compression is currently 1.5 GiB. The decompressor already supports dictionaries up to one byte less than 4 GiB, which is the maximum for LZMA1 and LZMA2

stream formats.

Dictionary size has the biggest effect on compression ratio. Dictionary size and match finder together determine the memory usage of the LZMA1 or LZMA2 encoder. The same dictionary size is required for decompressing that was used when compressing, thus the memory usage of the decoder is determined by the dictionary size used when compressing.

lc=*lc* Specify the number of literal context bits. The minimum is **0** and the maximum is **4**; the default is **3**. In addition, the sum of *lc* and *lp* must not exceed **4**.

lp=*lp* Specify the number of literal position bits. The minimum is **0** and the maximum is **4**; the default is **0**.

pb=*pb* Specify the number of position bits. The minimum is **0** and the maximum is **4**; the default is **2**.

mode=*mode*

Compression *mode* specifies the function used to analyze the data produced by the match finder. Supported *modes* are **fast** and **normal**. The default is **fast** for *presets* **0**−**2** and **normal** for *presets* **3**−**9**.

mf=*mf* Match finder has a major effect on encoder speed, memory usage, and compression ratio. Usually Hash Chain match finders are faster than Binary Tree match finders. Hash Chains are usually used together with **mode=fast** and Binary Trees with **mode=normal**. The memory usage formulas are only rough estimates, which are closest to reality when *dict* is a power of two.

hc3 Hash Chain with 2- and 3-byte hashing
Minimum value for *nice*: 3
Memory usage: *dict* * 7.5 (if *dict* <= 16 MiB);
dict * 5.5 + 64 MiB (if *dict* > 16 MiB)

hc4 Hash Chain with 2-, 3-, and 4-byte hashing
Minimum value for *nice*: 4
Memory usage: *dict* * 7.5

bt2 Binary Tree with 2-byte hashing
Minimum value for *nice*: 2
Memory usage: *dict* * 9.5

bt3 Binary Tree with 2- and 3-byte hashing
Minimum value for *nice*: 3
Memory usage: *dict* * 11.5 (if *dict* <= 16 MiB);
dict * 9.5 + 64 MiB (if *dict* > 16 MiB)

bt4 Binary Tree with 2-, 3-, and 4-byte hashing
Minimum value for *nice*: 4
Memory usage: *dict* * 11.5

nice=*nice*

Specify what is considered to be a nice length for a match. Once a match of at least *nice* bytes is found, the algorithm stops looking for possibly better matches.

nice can be 2−273 bytes. Higher values tend to give better compression ratio at expense of speed. The default depends on the *preset* level.

depth=*depth*

Specify the maximum search depth in the match finder. The default is the special value **0**, which makes the compressor determine a reasonable *depth* from *mf* and *nice*.

Using very high values for *depth* can make the encoder extremely slow with carefully crafted files. Avoid setting the *depth* over 1000 unless you are prepared to interrupt the

compression in case it is taking too long.

When decoding raw streams (**−−format=raw**), LZMA2 needs only the value of **dict**. LZMA1 needs also **lc**, **lp**, and **pb.**

−−x86[=*options*]

−−powerpc[=*options*]

−−ia64[=*options*]

−−arm[=*options*]

−−armthumb[=*options*]

−−sparc[=*options*]

Add a branch/call/jump (BCJ) filter to the filter chain. These filters can be used only as non-last filter in the filter chain.

A BCJ filter converts relative addresses in the machine code to their absolute counterparts. This doesn't change the size of the data, but it increases redundancy, which allows e.g. LZMA2 to get better compression ratio.

The BCJ filters are always reversible, so using a BCJ filter for wrong type of data doesn't cause any data loss. However, applying a BCJ filter for wrong type of data is a bad idea, because it tends to make the compression ratio worse.

Different instruction sets have have different alignment:

Filter	Alignment	Notes
x86	1	32-bit and 64-bit x86
PowerPC	4	Big endian only
ARM	4	Little endian only
ARM-Thumb	2	Little endian only
IA-64	16	Big or little endian
SPARC	4	Big or little endian

Since the BCJ-filtered data is usually compressed with LZMA2, the compression ratio may be improved slightly if the LZMA2 options are set to match the alignment of the selected BCJ filter. For example, with the IA-64 filter, it's good to set **pb=4** with LZMA2 (2^4=16). The x86 filter is an exception; it's usually good to stick to LZMA2's default four-byte alignment when compressing x86 executables.

All BCJ filters support the same *options*:

start=*offset*

Specify the start *offset* that is used when converting between relative and absolute addresses. The *offset* must be a multiple of the alignment of the filter (see the table above). The default is zero. In practice, the default is good; specifying a custom *offset* is almost never useful.

Specifying a non-zero start *offset* is probably useful only if the executable has multiple sections, and there are many cross-section jumps or calls. Applying a BCJ filter separately for each section with proper start offset and then compressing the result as a single chunk may give some improvement in compression ratio compared to applying the BCJ filter with the default *offset* for the whole executable.

−−delta[=*options*]

Add Delta filter to the filter chain. The Delta filter can be used only as non-last filter in the filter chain.

Currently only simple byte-wise delta calculation is supported. It can be useful when compressing e.g. uncompressed bitmap images or uncompressed PCM audio. However, special purpose algorithms may give significantly better results than Delta + LZMA2. This is true especially with audio, which compresses faster and better e.g. with FLAC.

Supported *options*:

dist=*distance*

Specify the *distance* of the delta calculation as bytes. *distance* must be 1−256. The default is 1.

For example, with **dist=2** and eight-byte input A1 B1 A2 B3 A3 B5 A4 B7, the output will be A1 B1 01 02 01 02 01 02.

Other options

−q, **−−quiet**

Suppress warnings and notices. Specify this twice to suppress errors too. This option has no effect on the exit status. That is, even if a warning was suppressed, the exit status to indicate a warning is still used.

−v, **−−verbose**

Be verbose. If standard error is connected to a terminal, **xz** will display a progress indicator. Specifying **−−verbose** twice will give even more verbose output (useful mostly for debugging).

The progress indicator shows the following information:

- Completion percentage is shown if the size of the input file is known. That is, percentage cannot be shown in pipes.
- Amount of compressed data produced (compressing) or consumed (decompressing).
- Amount of uncompressed data consumed (compressing) or produced (decompressing).
- Compression ratio, which is calculated by dividing the amount of compressed data processed so far by the amount of uncompressed data processed so far.
- Compression or decompression speed. This is measured as the amount of uncompressed data consumed (compression) or produced (decompression) per second. It is shown once a few seconds have passed since **xz** started processing the file.
- Elapsed time or estimated time remaining. Elapsed time is displayed in the format M:SS or H:MM:SS. The estimated remaining time is displayed in a less precise format which never has colons, for example, 2 min 30 s. The estimate can be shown only when the size of the input file is known and a couple of seconds have already passed since **xz** started processing the file.

When standard error is not a terminal, **−−verbose** will make **xz** print the filename, compressed size, uncompressed size, compression ratio, speed, and elapsed time on a single line to standard error after compressing or decompressing the file. If operating took at least a few seconds, also the speed and elapsed time are printed. If the operation didn't finish, for example due to user interruption, also the completion percentage is printed if the size of the input file is known.

−Q, **−−no−warn**

Don't set the exit status to **2** even if a condition worth a warning was detected. This option doesn't affect the verbosity level, thus both **−−quiet** and **−−no−warn** have to be used to not display warnings and to not alter the exit status.

−−robot

Print messages in a machine-parsable format. This is intended to ease writing frontends that want to use **xz** instead of liblzma, which may be the case with various scripts. The output with this option enabled is meant to be stable across **xz** releases. See the section **ROBOT MODE** for details.

−−info-memory

Display the current memory usage limit in human-readable format on a single line, and exit successfully. To see how much RAM **xz** thinks your system has, use **−−memory=100% −−info−memory**.

−h, **−−help**
Display a help message describing the most commonly used options, and exit successfully.

−H, **−−long−help**
Display a help message describing all features of **xz**, and exit successfully

−V, **−−version**
Display the version number of **xz** and liblzma in human readable format. To get machine-parsable output, specify **−−robot** before **−−version**.

ROBOT MODE

The robot mode is activated with the **−−robot** option. It makes the output of **xz** easier to parse by other programs. Currently **−−robot** is supported only together with **−−version**, **−−info-memory**, and **−−list**. It will be supported for normal compression and decompression in the future.

Version

xz −−robot −−version will print the version number of **xz** and liblzma in the following format:

XZ_VERSION=*XYYYZZZS*
LIBLZMA_VERSION=*XYYYZZZS*

X Major version.

YYY Minor version. Even numbers are stable. Odd numbers are alpha or beta versions.

ZZZ Patch level for stable releases or just a counter for development releases.

S Stability. **0** is alpha, **1** is beta, and **2** is stable. *S* should be always **2** when *YYY* is even.

XYYYZZZS are the same on both lines if **xz** and liblzma are from the same XZ Utils release.

Examples: 4.999.9beta is **49990091** and 5.0.0 is **50000002**.

Memory limit information

xz −−robot −−info-memory prints the current memory usage limit as bytes on a single line. To get the total amount of installed RAM, use **xz −−robot −−memory=100% −−info-memory**.

List mode

xz −−robot −−list uses tab-separated output. The first column of every line has a string that indicates the type of the information found on that line:

name This is always the first line when starting to list a file. The second column on the line is the filename.

file This line contains overall information about the **.xz** file. This line is always printed after the **name** line.

stream This line type is used only when **−−verbose** was specified. There are as many **stream** lines as there are streams in the **.xz** file.

block This line type is used only when **−−verbose** was specified. There are as many **block** lines as there are blocks in the **.xz** file. The **block** lines are shown after all the **stream** lines; different line types are not interleaved.

summary
This line type is used only when **−−verbose** was specified twice. This line is printed after all **block** lines. Like the **file** line, the **summary** line contains overall information about the **.xz** file.

totals This line is always the very last line of the list output. It shows the total counts and sizes.

The columns of the **file** lines:

2. Number of streams in the file
3. Total number of blocks in the stream(s)
4. Compressed size of the file

5. Uncompressed size of the file
6. Compression ratio, for example **0.123.** If ratio is over 9.999, three dashes (−−−) are displayed instead of the ratio.
7. Comma-separated list of integrity check names. The following strings are used for the known check types: **None**, **CRC32**, **CRC64**, and **SHA−256**. For unknown check types, **Unknown−***N* is used, where *N* is the Check ID as a decimal number (one or two digits).
8. Total size of stream padding in the file

The columns of the **stream** lines:

2. Stream number (the first stream is 1)
3. Number of blocks in the stream
4. Compressed start offset
5. Uncompressed start offset
6. Compressed size (does not include stream padding)
7. Uncompressed size
8. Compression ratio
9. Name of the integrity check
10. Size of stream padding

The columns of the **block** lines:

2. Number of the stream containing this block
3. Block number relative to the beginning of the stream (the first block is 1)
4. Block number relative to the beginning of the file
5. Compressed start offset relative to the beginning of the file
6. Uncompressed start offset relative to the beginning of the file
7. Total compressed size of the block (includes headers)
8. Uncompressed size
9. Compression ratio
10. Name of the integrity check

If **−−verbose** was specified twice, additional columns are included on the **block** lines. These are not displayed with a single **−−verbose**, because getting this information requires many seeks and can thus be slow:

11. Value of the integrity check in hexadecimal
12. Block header size
13. Block flags: **c** indicates that compressed size is present, and **u** indicates that uncompressed size is present. If the flag is not set, a dash (−) is shown instead to keep the string length fixed. New flags may be added to the end of the string in the future.
14. Size of the actual compressed data in the block (this excludes the block header, block padding, and check fields)
15. Amount of memory (as bytes) required to decompress this block with this **xz** version
16. Filter chain. Note that most of the options used at compression time cannot be known, because only the options that are needed for decompression are stored in the **.xz** headers.

The columns of the **totals** line:

2. Number of streams
3. Number of blocks
4. Compressed size
5. Uncompressed size
6. Average compression ratio
7. Comma-separated list of integrity check names that were present in the files
8. Stream padding size
9. Number of files. This is here to keep the order of the earlier columns the same as on **file** lines.

If **−−verbose** was specified twice, additional columns are included on the **totals** line:

10. Maximum amount of memory (as bytes) required to decompress the files with this **xz** version
11. **yes** or **no** indicating if all block headers have both compressed size and uncompressed size stored in them

Future versions may add new line types and new columns can be added to the existing line types, but the existing columns won't be changed.

EXIT STATUS

0 All is good.

1 An error occurred.

2 Something worth a warning occurred, but no actual errors occurred.

Notices (not warnings or errors) printed on standard error don't affect the exit status.

ENVIRONMENT

XZ_OPT

A space-separated list of options is parsed from **XZ_OPT** before parsing the options given on the command line. Note that only options are parsed from **XZ_OPT**; all non-options are silently ignored. Parsing is done with **getopt_long**(3) which is used also for the command line arguments.

LZMA UTILS COMPATIBILITY

The command line syntax of **xz** is practically a superset of **lzma**, **unlzma**, and **lzcat** as found from LZMA Utils 4.32.x. In most cases, it is possible to replace LZMA Utils with XZ Utils without breaking existing scripts. There are some incompatibilities though, which may sometimes cause problems.

Compression preset levels

The numbering of the compression level presets is not identical in **xz** and LZMA Utils. The most important difference is how dictionary sizes are mapped to different presets. Dictionary size is roughly equal to the decompressor memory usage.

Level	xz	LZMA Utils
−1	64 KiB	64 KiB
−2	512 KiB	1 MiB
−3	1 MiB	512 KiB
−4	2 MiB	1 MiB
−5	4 MiB	2 MiB
−6	8 MiB	4 MiB
−7	16 MiB	8 MiB
−8	32 MiB	16 MiB
−9	64 MiB	32 MiB

The dictionary size differences affect the compressor memory usage too, but there are some other differences between LZMA Utils and XZ Utils, which make the difference even bigger:

Level	xz	LZMA Utils 4.32.x

−1	2 MiB	2 MiB
−2	5 MiB	12 MiB
−3	13 MiB	12 MiB
−4	25 MiB	16 MiB
−5	48 MiB	26 MiB
−6	94 MiB	45 MiB
−7	186 MiB	83 MiB
−8	370 MiB	159 MiB
−9	674 MiB	311 MiB

The default preset level in LZMA Utils is **−7** while in XZ Utils it is **−6**, so both use 8 MiB dictionary by default.

Streamed vs. non-streamed .lzma files

Uncompressed size of the file can be stored in the **.lzma** header. LZMA Utils does that when compressing regular files. The alternative is to mark that uncompressed size is unknown and use end of payload marker to indicate where the decompressor should stop. LZMA Utils uses this method when uncompressed size isn't known, which is the case for example in pipes.

xz supports decompressing **.lzma** files with or without end of payload marker, but all **.lzma** files created by **xz** will use end of payload marker and have uncompressed size marked as unknown in the **.lzma** header. This may be a problem in some (uncommon) situations. For example, a **.lzma** decompressor in an embedded device might work only with files that have known uncompressed size. If you hit this problem, you need to use LZMA Utils or LZMA SDK to create **.lzma** files with known uncompressed size.

Unsupported .lzma files

The **.lzma** format allows *lc* values up to 8, and *lp* values up to 4. LZMA Utils can decompress files with any *lc* and *lp*, but always creates files with **lc=3** and **lp=0**. Creating files with other *lc* and *lp* is possible with **xz** and with LZMA SDK.

The implementation of the LZMA1 filter in liblzma requires that the sum of *lc* and *lp* must not exceed 4. Thus, **.lzma** files which exceed this limitation, cannot be decompressed with **xz**.

LZMA Utils creates only **.lzma** files which have dictionary size of 2^*n* (a power of 2), but accepts files with any dictionary size. liblzma accepts only **.lzma** files which have dictionary size of 2^*n* or 2^*n* + 2^(*n*−1). This is to decrease false positives when detecting **.lzma** files.

These limitations shouldn't be a problem in practice, since practically all **.lzma** files have been compressed with settings that liblzma will accept.

Trailing garbage

When decompressing, LZMA Utils silently ignore everything after the first **.lzma** stream. In most situations, this is a bug. This also means that LZMA Utils don't support decompressing concatenated **.lzma** files.

If there is data left after the first **.lzma** stream, **xz** considers the file to be corrupt. This may break obscure scripts which have assumed that trailing garbage is ignored.

NOTES

Compressed output may vary

The exact compressed output produced from the same uncompressed input file may vary between XZ Utils versions even if compression options are identical. This is because the encoder can be improved (faster or better compression) without affecting the file format. The output can vary even between different builds of the same XZ Utils version, if different build options are used.

The above means that implementing **−−rsyncable** to create rsyncable **.xz** files is not going to happen without freezing a part of the encoder implementation, which can then be used with **−−rsyncable**.

Embedded .xz decompressors

Embedded **.xz** decompressor implementations like XZ Embedded don't necessarily support files created with *check* types other than **none** and **crc32**. Since the default is **−−check=***crc64*, you must use **−−check=none** or **−−check=crc32** when creating files for embedded systems.

Outside embedded systems, all **.xz** format decompressors support all the *check* types, or at least are able to decompress the file without verifying the integrity check if the particular *check* is not supported.

XZ Embedded supports BCJ filters, but only with the default start offset.

EXAMPLES

Basics

A mix of compressed and uncompressed files can be decompressed to standard output with a single command:

xz -dcf a.txt b.txt.xz c.txt d.txt.xz > abcd.txt

Parallel compression of many files

On GNU and *BSD, **find**(1) and **xargs**(1) can be used to parallellize compression of many files:

find . −type f \! −name '*.xz' −print0 | xargs −0r −P4 −n16 xz

The **−P** option sets the number of parallel **xz** processes. The best value for the **−n** option depends on how many files there are to be compressed. If there are only a couple of files, the value should probably be **1**; with tens of thousands of files, **100** or even more may be appropriate to reduce the number of **xz** processes that **xargs**(1) will eventually create.

Robot mode examples

Calculating how many bytes have been saved in total after compressing multiple files:

xz --robot --list *.xz | awk '/^totals/{print $5−$4}'

SEE ALSO

xzdec(1), **gzip**(1), **bzip2**(1)

XZ Utils: <http://tukaani.org/xz/>
XZ Embedded: <http://tukaani.org/xz/embedded.html>
LZMA SDK: <http://7-zip.org/sdk.html>