NAME

lbzip2 − parallel bzip2 utility

SYNOPSIS

lbzip2|**bzip2** [**−n** *WTHRS*] [**−k**|**−c**|**−t**] [**−d**] [**−1** .. **−9**] [**−f**] [**−v**] [**−S**] [*FILE* ...]

lbunzip2|**bunzip2** [**−n** *WTHRS*] [**−k**|**−c**|**−t**] [**−z**] [**−f**] [**−v**] [**−S**] [*FILE* ...]

lbzcat|**bzcat** [**−n** *WTHRS*] [**−z**] [**−f**] [**−v**] [**−S**] [*FILE* ...]

lbzip2|**bzip2**|**lbunzip2**|**bunzip2**|**lbzcat**|**bzcat −h**

DESCRIPTION

Compress or decompress *FILE* operands or standard input to regular files or standard output using the Burrows-Wheeler block-sorting text compression algorithm. The **lbzip2** utility employs multiple threads and an input-bound splitter even when decompressing **.bz2** files created by standard bzip2.

Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and competitive with all but the best of the PPM family of statistical compressors.

Compression is always performed, even if the compressed file is slightly larger than the original. The worst case expansion is for files of zero length, which expand to fourteen bytes. Random data (including the output of most file compressors) is coded with asymptotic expansion of around 0.5%.

The command-line options are deliberately very similar to those of **bzip2** and **gzip**, but they are not identical.

INVOCATION

The default mode of operation is compression. If the utility is invoked as **lbunzip2** or **bunzip2**, the mode is switched to decompression. Calling the utility as **lbzcat** or **bzcat** selects decompression, with the decompressed byte-stream written to standard output.

OPTIONS

−n *WTHRS*

Set the number of (de)compressor threads to *WTHRS*. If this option is not specified, **lbzip2** tries to query the system for the number of online processors (if both the compilation environment and the execution environment support that), or exits with an error (if it's unable to determine the number of processors online).

−k, **−−keep**

Don't remove *FILE* operands after successful (de)compression. Open regular input files with more than one link.

−c, **−−stdout**

Write output to standard output, even when *FILE* operands are present. Implies **−k** and excludes **−t**.

−t, **−−test**
Test decompression; discard output instead of writing it to files or standard output. Implies **−k** and excludes **−c**. Roughly equivalent to passing **−c** and redirecting standard output to the bit bucket.

−d, **−−decompress**
Force decompression over the mode of operation selected by the invocation name.

−z, **−−compress**
Force compression over the mode of operation selected by the invocation name.

−1 .. **−9**
Set the compression block size to 100K .. 900K, in 100K increments. Ignored during decompression. See also the BLOCK SIZE section below.

−−fast Alias for **−1**.

−−best Alias for **−9**. This is the default.

−f, **−−force**
Open non-regular input files. Open input files with more than one link, breaking links when **−k** isn't specified in addition. Try to remove each output file before opening it. By default **lbzip2** will not overwrite existing files; if you want this to happen, you should specify **−f**.

−v, **−−verbose**
Be more verbose. Print more detailed information about (de)compression progress to standard error: before processing each file, print a message stating the names of input and output files; during (de)compression, print a rough percentage of completeness and estimated time of arrival (only if standard error is connected to a terminal); after processing each file print a message showing compression ratio, space savings, total compression time (wall time) and average (de)compression speed (bytes of plain data processed per second).

−S Print condition variable statistics to standard error for each completed (de)compression operation. Useful in profiling.

−s, **−−small**, **−q**, **−−quiet**, **−−repetitive−fast**, **−−repetitive−best**
Accepted for compatibility with **bzip2**, otherwise ignored.

−−exponential
Use an alternative block-sorting algorithm -- bucket-pointer refinement (BPR). See also **REPETITIVE DATA** below.

−h, **−−help**
Print help on command-line usage on standard output and exit successfully.

−L, **−−license**, **−V**, **−−version**
Print license and version information on standard output and exit successfully.

ENVIRONMENT

LBZIP2, *BZIP2*, *BZIP*

Before parsing the command line, lbzip2 inserts the contents of these variables, in the order specified, between the invocation name and the rest of the command line. Tokens are separated by spaces and tabs, which cannot be escaped.

OPERANDS

FILE Specify files to compress or decompress.

*FILE*s with **.bz2**, **.tbz**, **.tbz2** and **.tz2** name suffixes will be skipped when compressing. When decompressing, **.bz2** suffixes will be removed in output filenames; **.tbz**, **.tbz2** and **.tz2** suffixes will be replaced by **.tar**; other filenames will be suffixed with **.out**. If an **INT** or **TERM** signal is delivered to **lbzip2**, then it removes the regular output file currently open before exiting.

If no FILE is given, lbzip2 works as a filter, processing standard input to standard output. In this case, **lbzip2** will decline to write compressed output to a terminal (or read compressed input from a terminal), as this would be entirely incomprehensible and therefore pointless.

EXIT STATUS

0 if **lbzip2** finishes successfully. This presumes that whenever it tries, **lbzip2** never fails to write to standard error.

1 if **lbzip2** encounters a fatal error.

4 if **lbzip2** issues warnings without encountering a fatal error. This presumes that whenever it tries, **lbzip2** never fails to write to standard error.

SIGPIPE, **SIGXFSZ**

if **lbzip2** intends to exit with status **1** due to any fatal error, but any such signal with inherited **SIG_DFL** action was generated for **lbzip2** previously, then **lbzip2** terminates by way of one of said signals, after cleaning up any interrupted output file.

SIGABRT

if a runtime assertion fails (i.e. **lbzip2** detects a bug in itself). Hopefully whoever compiled your binary wasn't bold enough to **#define NDEBUG**.

SIGINT, **SIGTERM**

lbzip2 catches these signals so that it can remove an interrupted output file. In such cases, **lbzip2** exits by re-raising (one of) the received signal(s).

BLOCK SIZE

lbzip2 compresses large files in blocks. It can operate at various block sizes, ranging from 100k to 900k in 100k steps, and it allocates only as much memory as it needs to. The block size affects both the compression ratio achieved, and the amount of memory needed both for compression and decompression. Compression and decompression speed is virtually unaffected by block size, provided that the file being processed is large enough to be split among all worker threads.

The flags **−1** through **−9** specify the block size to be 100,000 bytes through 900,000 bytes (the default)

respectively. At decompression-time, the block size used for compression is read from the compressed file -- the flags **−1** to **−9** are irrelevant to and so ignored during decompression.

Larger block sizes give rapidly diminishing marginal returns; most of the compression comes from the first two or three hundred k of block size, a fact worth bearing in mind when using **lbzip2** on small machines. It is also important to appreciate that the decompression memory requirement is set at compression-time by the choice of block size. In general you should try and use the largest block size memory constraints allow.

Another significant point applies to small files. By design, only one of **lbzip2**'s worker threads can work on a single block. This means that if the number of blocks in the compressed file is less than the number of processors online, then some of worker threads will remain idle for the entire time. Compressing small files with smaller block sizes can therefore significantly increase both compression and decompression speed. The speed difference is more noticeable as the number of CPU cores grows.

REPETITIVE DATA

The core transformation of the compression algorithm is sorting all rotations of a single string (the algorithm is known as Burrows-Wheeler transformation). The primary factor of compression speed is therefore the ability to sort strings quickly.

The primary sorting algorithm of **lbzip2** is a highly optimized variation of Bentley-McIlroy three-way quicksort, which behaves well in the average case, but in the worst case it can run very slowly. **lbzip2** also implements an alternative block-sorting algorithm -- the bucket-pointer refinement (BPR in short).

Quicksort is usually faster than BPR, but sometimes, for very repetitive input data, it may be much slower. Normally, when **−−exponential** is not given, **lbzip2** first tries to use quicksort and then possibly switches to BPR if quicksort runs into difficulties. This allows **lbzip2** to run fast in average case while remaining reasonable speed in corner cases.

With the **−−exponential** option given, **lbzip2** will skip quicksorting and use BPR straight away. This option doesn't affect neither compression ratio nor decompression speed. Use of **−−exponential** makes sense if you know the input data is repetitive. Otherwise it's just waste of CPU cycles, but otherwise harmless.

ERROR HANDLING

Dealing with error conditions is the least satisfactory aspect of **lbzip2**. The policy is to try and leave the filesystem in a consistent state, then quit, even if it means not processing some of the files mentioned in the command line.

'A consistent state' means that a file exists either in its compressed or uncompressed form, but not both. This boils down to the rule 'delete the output file if an error condition occurs, leaving the input intact'. Input files are only deleted when we can be pretty sure the output file has been written and closed successfully.

RESOURCE ALLOCATION

lbzip2 needs various kinds of system resources to operate. Those include memory, threads, mutexes and condition variables. The policy is to simply give up if a resource allocation failure occurs.

Resource consumption grows linearly with number of worker threads. If **lbzip2** fails because of lack of some resources, decreasing number of worker threads may help. It would be possible for **lbzip2** to try to reduce number of worker threads (and hence the resource consumption), or to move on to subsequent files

in the hope that some might need less resources, but the complications for doing this seem more trouble than they are worth.

DAMAGED FILES

lbzip2 attempts to compress data by performing several non-trivial transformations on it. Every compression of a file implies an assumption that the compressed file can be decompressed to reproduce the original. Great efforts in design, coding and testing have been made to ensure that this program works correctly. However, the complexity of the algorithms, and, in particular, the presence of various special cases in the code which occur with very low but non-zero probability make it very difficult to rule out the possibility of bugs remaining in the program. That is not to say this program is inherently unreliable. Indeed, I very much hope the opposite is true -- **lbzip2** has been carefully constructed and extensively tested.

As a self-check for your protection, **lbzip2** uses 32-bit CRCs to make sure that the decompressed version of a file is identical to the original. This guards against corruption of the compressed data, and against undiscovered bugs in **lbzip2** (hopefully unlikely). The chances of data corruption going undetected is microscopic, about one chance in four billion for each file processed. Be aware, though, that the check occurs upon decompression, so it can only tell you that that something is wrong.

CRCs can only detect corrupted files, they can't help you recover the original, uncompressed data. However, because of the block nature of the compression algorithm, it may be possible to recover some parts of the damaged file, even if some blocks are destroyed.

BUGS

Separate input files don't share worker threads; at most one input file is worked on at any moment.

During decompression and testing highly compressed files large amounts of memory may be allocated; currently there is no way of limiting memory allocation other than limiting number of worker threads.

AUTHORS

lbzip2 was originally written by Laszlo Ersek <lacos@caesar.elte.hu>, http://lacos.hu/. Version 2.0 was written by Mikolaj Izdebski.

COPYRIGHT

THANKS

Adam Maulis at ELTE IIG; Julian Seward; Paul Sladen; Michael Thomas from Caltech HEP; Bryan Stillwell; Zsolt Bartos-Elekes; Imre Csatlos; Gabor Kovesdan; Paul Wise; Paolo Bonzini; Department of Electrical and Information Engineering at the University of Oulu.

SEE ALSO

bzip2(1)
http://www.bzip.org/

pbzip2(1)
http://compression.ca/pbzip2/

bzip2smp(1)
http://bzip2smp.sourceforge.net/

smpbzip2(1)
http://home.student.utwente.nl/n.werensteijn/smpbzip2/

dbzip2(1)
http://www.mediawiki.org/wiki/Dbzip2

p7zip(1)
http://p7zip.sourceforge.net/