NAME

bzip2, bunzip2 - a block-sorting file compressor, v1.0.6
bzcat - decompresses files to stdout
bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS

bzip2 [**-cdfkqstvzVL123456789**] [*filenames ...*]
bzip2 [**-h|--help**]
bunzip2 [**-fkvsVL**] [*filenames ...*]
bunzip2 [**-h|--help**]
bzcat [**-s**] [*filenames ...*]
bzcat [**-h|--help**]
bzip2recover *filename*

DESCRIPTION

bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.

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

bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a compressed version of itself, with the name original_name.bz2. Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is naive in the sense that there is no mechanism for preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-DOS.

bzip2 and *bunzip2* will by default not overwrite existing files. If you want this to happen, specify the -f flag.

If no file names are specified, *bzip2* compresses from standard input to standard output. In this case, *bzip2* will decline to write compressed output to a terminal, as this would be entirely incomprehensible and therefore pointless.

bunzip2 (or *bzip2 -d)* decompresses all specified files. Files which were not created by *bzip2* will be detected and ignored, and a warning issued. *bzip2* attempts to guess the filename for the decompressed file from that of the compressed file as follows:

filename.bz2 becomes filename filename.bz becomes filename filename.tbz2 becomes filename.tar filename.tbz becomes filename.tar anyothername becomes anyothername.out

If the file does not end in one of the recognised endings, *.bz2, .bz, .tbz2* or *.tbz, bzip2* complains that it cannot guess the name of the original file, and uses the original name with *.out* appended.

As with compression, supplying no filenames causes decompression from standard input to standard output.

bunzip2 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 (-t) of concatenated compressed files is also supported.

You can also compress or decompress files to the standard output by giving the -c flag. Multiple files may be compressed and decompressed like this. The resulting outputs are fed sequentially to stdout. Compression of multiple files in this manner generates a stream containing multiple compressed file representations. Such a stream can be decompressed correctly only by *bzip2* version 0.9.0 or later. Earlier versions of *bzip2* will stop after decompressing the first file in the stream.

bzcat (or *bzip2 -dc)* decompresses all specified files to the standard output.

bzip2 will read arguments from the environment variables *BZIP2* and *BZIP,* in that order, and will process them before any arguments read from the command line. This gives a convenient way to supply default arguments.

Compression is always performed, even if the compressed file is slightly larger than the original. Files of less than about one hundred bytes tend to get larger, since the compression mechanism has a constant overhead in the region of 50 bytes. Random data (including the output of most file compressors) is coded at about 8.05 bits per byte, giving an expansion of around 0.5%.

As a self-check for your protection, *bzip2* 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 undetected bugs in *bzip2* (hopefully very 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 something is wrong. It can't help you recover the original uncompressed data. You can use *bzip2recover* to try to recover data from damaged files.

Return values: 0 for a normal exit, 1 for environmental problems (file not found, invalid flags, I/O errors, &c), 2 to indicate a corrupt compressed file, 3 for an internal consistency error (eg, bug) which caused *bzip2* to panic.

OPTIONS

-c --stdout
Compress or decompress to standard output.

-d --decompress
Force decompression. *bzip2, bunzip2* and *bzc at* are really the same program, and the decision about what actions to take is done on the basis of which name is used. This flag overrides that mechanism, and forces *bzip2* to decompress.

-z --compress
The complement to -d: forces compression, regardless of the invocation name.

-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.

-f --force
Force overwrite of output files. Normally, *bzip2* will not overwrite existing output files. Also forces *bzip2* to break hard links to files, which it otherwise wouldn't do.

bzip2 normally declines to decompress files which don't have the correct magic header bytes. If forced (-f), however, it will pass such files through unmodified. This is how GNU gzip behaves.

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

-s --small
Reduce memory usage, for compression, decompression and testing. Files are decompressed and tested using a modified algorithm which only requires 2.5 bytes per block byte. This means any file can be decompressed in 2300 k of memory, albeit at about half the normal speed.

During compression, -s selects a block size of 200 k, which limits memory use to around the same figure, at the expense of your compression ratio. In short, if your machine is low on memory (8 megabytes or less), use -s for everything. See MEMORY MANAGEMENT below.

-q --quiet
Suppress non-essential warning messages. Messages pertaining to I/O errors and other critical events will not be suppressed.

-v --verbose
Verbose mode -- show the compression ratio for each file processed. Further -v's increase the verbosity level, spewing out lots of information which is primarily of interest for diagnostic purposes.

-h --help
Print a help message and exit.

-L --license -V --version
Display the software version, license terms and conditions.

-1 (or --fast) to -9 (or --best)
Set the block size to 100 k, 200 k ... 900 k when compressing. Has no effect when decompressing. See MEMORY MANAGEMENT below. The --fast and --best aliases are primarily for GNU gzip compatibility. In particular, --fast doesn't make things significantly faster. And --best merely selects the default behaviour.

-- Treats all subsequent arguments as file names, even if they start with a dash. This is so you can handle files with names beginning with a dash, for example: bzip2 -- -myfilename.

--repetitive-fast --repetitive-best
These flags are redundant in versions 0.9.5 and above. They provided some coarse control over the behaviour of the sorting algorithm in earlier versions, which was sometimes useful. 0.9.5 and above have an improved algorithm which renders these flags irrelevant.

MEMORY MANAGEMENT

bzip2 compresses large files in blocks. The block size affects both the compression ratio achieved, and the amount of memory needed for compression and decompression. 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 header of the compressed file, and *bunzip2* then allocates itself just enough memory to decompress the file. Since block sizes are stored in compressed files, it follows that the flags -1 to -9 are irrelevant to and so ignored during decompression.

Compression and decompression requirements, in bytes, can be estimated as:

Compression: 400 k + (8 x block size)

Decompression: 100 k + (4 x block size), or 100 k + (2.5 x block size)

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 *bzip2* 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.

For files compressed with the default 900 k block size, *bunzip2* will require about 3700 kbytes to decompress. To support decompression of any file on a 4 megabyte machine, *bunzip2* has an option to decompress using approximately half this amount of memory, about 2300 kbytes. Decompression speed is also halved, so you should use this option only where necessary. The relevant flag is -s.

In general, try and use the largest block size memory constraints allow, since that maximises the compression achieved. Compression and decompression speed are virtually unaffected by block size.

Another significant point applies to files which fit in a single block -- that means most files you'd encounter using a large block size. The amount of real memory touched is proportional to the size of the file, since the file is smaller than a block. For example, compressing a file 20,000 bytes long

with the flag -9 will cause the compressor to allocate around 7600 k of memory, but only touch 400 k + 20000 * 8 = 560 kbytes of it. Similarly, the decompressor will allocate 3700 k but only touch 100 k + 20000 * 4 = 180 kbytes.

Here is a table which summarises the maximum memory usage for different block sizes. Also recorded is the total compressed size for 14 files of the Calgary Text Compression Corpus totalling 3,141,622 bytes. This column gives some feel for how compression varies with block size. These figures tend to understate the advantage of larger block sizes for larger files, since the Corpus is dominated by smaller files.

Compress Decompress Decompress Corpus Flag usage usage -s usage Size

-1 1200k 500k 350k 914704 -2 2000k 900k 600k 877703 -3 2800k 1300k 850k 860338 -4 3600k 1700k 1100k 846899 -5 4400k 2100k 1350k 845160 -6 5200k 2500k 1600k 838626 -7 6100k 2900k 1850k 834096 -8 6800k 3300k 2100k 828642 -9 7600k 3700k 2350k 828642

RECOVERING DATA FROM DAMAGED FILES

bzip2 compresses files in blocks, usually 900 kbytes long. Each block is handled independently. If a media or transmission error causes a multi-block .bz2 file to become damaged, it may be possible to recover data from the undamaged blocks in the file.

The compressed representation of each block is delimited by a 48-bit pattern, which makes it possible to find the block boundaries with reasonable certainty. Each block also carries its own 32-bit CRC, so damaged blocks can be distinguished from undamaged ones.

bzip2recover is a simple program whose purpose is to search for blocks in .bz2 files, and write each block out into its own .bz2 file. You can then use *bzip2* -t to test the integrity of the resulting files, and decompress those which are undamaged.

bzip2recover takes a single argument, the name of the damaged file, and writes a number of files rec00001file.bz2, rec00002file.bz2, etc., containing the extracted blocks. The output filenames are designed so that the use of wildcards in subsequent processing -- for example, bzip2 -dc rec*file.bz2 > recovered_data -- processes the files in the correct order.

bzip2recover should be of most use dealing with large .bz2 files, as these will contain many blocks. It is clearly futile to use it on damaged single-block files, since a damaged block cannot be recovered. If you wish to minimise any potential data loss through media or transmission errors, you might consider compressing with a smaller block size.

PERFORMANCE NOTES

The sorting phase of compression gathers together similar strings in the file. Because of this, files containing very long runs of repeated symbols, like aabaabaabaab ... (repeated several hundred times) may compress more slowly than normal. Versions 0.9.5 and above fare much better than previous versions in this respect. The ratio between worst-case and average-case compression time is in the region of 10:1. For previous versions, this figure was more like 100:1. You can use the -vvvv option to monitor progress in great detail, if you want.

Decompression speed is unaffected by these phenomena.

bzip2 usually allocates several megabytes of memory to operate in, and then charges all over it in a fairly random fashion. This means that performance, both for compressing and decompressing, is largely determined by the speed at which your machine can service cache misses. Because of this, small changes to the code to reduce the miss rate have been observed to give disproportionately large performance improvements. I imagine*bzip2* will perform best on mac hines with very large caches.

CAVEATS

I/O error messages are not as helpful as they could be. *bzip2* tries hard to detect I/O errors and exit cleanly, but the details of what the problem is sometimes seem rather misleading.

This manual page pertains to version 1.0.6 of *bzip2*. Compressed data created by this version is entirely forwards and backwards compatible with the previous public releases, versions 0.1pl2, 0.9.0, 0.9.5, 1.0.0, 1.0.1, 1.0.2 and above, but with the following exception: 0.9.0 and above can correctly decompress multiple concatenated compressed files. 0.1pl2 cannot do this; it will stop after decompressing just the first file in the stream.

bzip2recover versions prior to 1.0.2 used 32-bit integers to represent bit positions in compressed files, so they could not handle compressed files more than 512 megabytes long. Versions 1.0.2 and above use 64-bit ints on some platforms which support them (GNU supported targets, and Windows). To establish whether or not bzip2recover was built with such a limitation, run it without arguments. In any event you can build yourself an unlimited version if you can recompile it with MaybeUInt64 set to be an unsigned 64-bit integer.

AUTHOR

Julian Seward, jsewardbzip.org.

http://www.bzip.org

The ideas embodied in *bzip2* are due to (at least) the following people: Michael Burrows and David Wheeler (for the block sorting transformation), David Wheeler (again, for the Huffman coder), Peter Fenwick (for the structured coding model in the original *bzip,* and many refinements), and Alistair Moffat, Radford Neal and Ian Witten (for the arithmetic coder in the original *bzip).* I am much indebted for their help, support and advice. See the manual in the source distribution for pointers to sources of documentation. Christian von Roques encouraged me to look for faster sorting algorithms, so as to speed up compression. Bela Lubkin encouraged me to improve the worst-case compression performance. Donna Robinson XMLised the documentation. The bz* scripts are derived from those of GNU gzip. Many people sent patches, helped with portability problems, lent machines, gave advice and were generally helpful.