ckolivas / **lrzip** Public

Code | Issues 3 | Pull requests 1 | Actions | Projects | Wiki | Security and quality | Insights

master | **5** Branches | **44** Tags | Go to file | Go to file | Code

ckolivas Trivial copyright date update. ✓ cc698c9 · 2 months ago

.github/workflows	Update github workflow to install libbz2	3 months ago
doc	Minor documentation error.	3 months ago
libzpaq	Implement write bounds checking within lib…	3 months ago
lzma	Silence dangling pointer warning in lzma en…	3 months ago
m4	fix warnings during autoconf (#254)	2 years ago
man	Update documentation.	4 years ago
util	explicitely require bash so that autogen.sh …	5 years ago
.gitignore	Add demo executables to .gitignore.	5 years ago
AUTHORS	Update changelogs.	15 years ago
BUGS	Add message about issue tracker in BUGS	12 years ago
COPYING	Update GPL license.	15 years ago
ChangeLog	Update documentation.	4 years ago
Dockerfile	Creating Dockerfile (#238)	2 years ago
INSTALL	Updated autotools/conf system courtesy of …	16 years ago
Makefile.am	Remove build requirement for deprecated h…	4 years ago
README-NOT-BACKWARD-COMP…	Update READMEs.	16 years ago
README.md	Update FAQ.	3 months ago
TODO	Update and bring back Assembler code for …	7 years ago
WHATS-NEW	Update WHATS NEW.	3 months ago
aes.c	remove trailing whitespace	15 years ago
aes.h	Better make length a long long int in aes_cr…	16 years ago
autogen.sh	fix-autogen	16 years ago
configure.ac	Reinstate mistaken removal of fatal_exit()s.	3 months ago
decompress_demo.c	Update copyright notices and style to 1TBS…	15 years ago
description-pak	Initial import	16 years ago
lrzip.c	Display what directory a temporary output fi…	3 months ago
lrzip_core.h	Implement write bounds checking within lib…	3 months ago
lrzip_private.h	Remove unused parameters from print out…	3 months ago
lrztar	Update copyright notices.	5 years ago
	Trivial copyright date update.	2 months ago

main.c		
md5.c	Update various copyright dates.	13 years ago
md5.h	Get rid of trailing whitespace	16 years ago
regressiontest.good	Added regressiontest.	10 years ago
regressiontest.sh	Added regressiontest.	10 years ago
runzip.c	Make fatal() check for printf attributes.	3 months ago
runzip.h	Update copyright dates for affected files.	4 years ago
rzip.c	Fix incorrect number of parameters being p…	3 months ago
rzip.h	Update copyright dates for affected files.	4 years ago
sha4.c	Remove unused functions from aes.c and s…	16 years ago
sha4.h	Remove unused sha4 exports from sha4.h.	15 years ago
stream.c	Implement write bounds checking within lib…	3 months ago
stream.h	Update more copyright notices	10 years ago
util.c	Fix incorrect number of parameters being p…	3 months ago
util.h	Reinstate mistaken removal of fatal_exit()s.	3 months ago

README GPL-2.0 license

lrzip - Long Range ZIP or LZMA RZIP

A compression utility that excels at compressing large files (usually > 10-50 MB). Larger files and/or more free RAM means that the utility will be able to more effectively compress your files (ie: faster / smaller size), especially if the filesize(s) exceed 100 MB. You can either choose to optimise for speed (fast compression / decompression) or size, but not both.

haneefmubarak's TL;DR for the long explanation:

Just change the word `directory` to the name of the directory you wish to compress.

Compression:

```
lrzdir=directory; tar cvf $lrzdir.tar $lrzdir; lrzip -Ubvvp `nproc` -S .bzip2-lrz -L 9 $lrzdir.tar; rm -fv $lrzdir.ta
```

`tar`s the directory, then maxes out all of the system's processor cores along with sliding window RAM to give the best **BZIP2** compression while being as fast as possible, enables max verbosity output, attaches the extension `.bzip2-lrz`, and finally gets rid of the temporary tarfile. Uses a tempvar `lrzdir` which is unset automatically.

Decompression for the kind of file from above:

```
lrzdir=directory; lrunzip -cdivvp `nproc` -o $lrzdir.tar $lrzdir.tar.bzip2-lrz; tar xvf $lrzdir.tar; rm -vf $lrzdir.t
```

Checks integrity, then decompresses the directory using all of the processor cores for max speed, enables max verbosity output, unarchives the resulting tarfile, and finally gets rid of the temporary tarfile. Uses the same kind of tempvar.

lrzip build/install guide:

A quick guide on building and installing.

What you will need

- gcc
- bash or zsh
- pthreads
- tar
- libc
- libm
- libz-dev
- libbz2-dev
- liblzo2-dev
- liblz4-dev
- coreutils
- Optional nasm
- git if you want a repo-fresh copy
- an OS with the usual *nix headers and libraries

Obtaining the source

Two different ways of doing this:

Stable: Packaged tarball that is known to work:

Go to https://github.com/ckolivas/lrzip/releases and download the `tar.gz` file from the top. `cd` to the directory you downloaded, and use `tar xvzf lrzip-X.X.tar.gz` to extract the files (don't forget to replace `X.X` with the correct version). Finally, cd into the directory you just extracted.

Latest: `git clone -v https://github.com/ckolivas/lrzip.git; cd lrzip`

Build

```
./autogen.sh
./configure
make -j `nproc` # maxes out all cores
```

Install

Simple 'n Easy™: `sudo make install`

lrzip 101:

Command	Result
`lrztar directory`	An archive `directory.tar.lrz` compressed with **LZMA**.
`lrzuntar directory.tar.lrz`	A directory extracted from a `lrztar` archive.
`lrzip filename`	An archive `filename.lrz` compressed with **LZMA**, meaning slow compression and fast decompression.
`lrzip -z filename`	An archive "filename.lrz" compressed with **ZPAQ** that can give extreme compression, but takes a bit longer than forever to compress and decompress.
`lrzip -l filename`	An archive lightly compressed with **LZO**, meaning really, really fast compression and decompression.
`lrunzip filename.lrz`	Decompress filename.lrz to filename.
`lrz filename`	As per lrzip above but with gzip compatible semantics (i.e. will be quiet and delete original file)
`lrz -d filename.lrz`	As per lrunzip above but with gzip compatible semantics (i.e. will be quiet and delete original file)

lrzip internals

lrzip uses an extended version of rzip which does a first pass long distance redundancy reduction. lrzip's modifications allow it to scale to accommodate various memory sizes.

Then, one of the following scenarios occurs:

- Compressed
- (default) **LZMA** gives excellent compression @ ~2x the speed of bzip2
- **ZPAQ** gives extreme compression while taking forever
- **LZO** gives insanely fast compression that can actually be faster than simply copying a large file
- **GZIP** gives compression almost as fast as LZO but with better compression
- **BZIP2** is a defacto linux standard and hacker favorite which usually gives quite good compression (ZPAQ>LZMA>BZIP2>GZIP>LZO) while staying fairly fast (LZO>GZIP>BZIP2>LZMA>ZPAQ); in other words, a good middle-ground and a good choice overall
- Uncompressed, in the words of the software's original author:

> Leaving it uncompressed and rzip prepared. This form improves substantially any compression performed on the resulting file in both size and speed (due to the nature of rzip preparation merging similar compressible blocks of data and creating a smaller file). By "improving" I mean it will either speed up the very slow compressors with minor detriment to compression, or greatly increase the compression of simple compression algorithms.
>
> (Con Kolivas, from the original lrzip README)

The only real disadvantages:

- The main program, lrzip, only works on single files, and therefore requires the use of an lrztar wrapper to fake a complete archiver.
- lrzip requires quite a bit of memory along with a modern processor to get the best performance in reasonable time. This usually means that it is somewhat unusable with less than 256 MB. However, decompression usually requires less RAM and can work on less powerful machines with much less RAM. On machines with less RAM, it may be a good idea to enable swap if you want to keep your operating system happy.
- Piping output to and/or from STDIN and/or STDOUT works fine with both compression and decompression, but larger files compressed this way will likely end up being compressed less efficiently. Decompression doesn't really have any issues with piping, though.

One of the more unique features of lrzip is that it will try to use all of the available RAM as best it can at all times to provide maximum benefit. This is the default operating method, where it will create and use the single largest memory window that will still fit in available memory without freezing up the system. It does this by `mmap`ing the small portions of the file that it is working on. However, it also has a unique "sliding `mmap`" feature, which allows it to use compression windows that far exceed the size of your RAM if the file you are compressing is large. It does this by using one large `mmap` along with a smaller moving `mmap` buffer to track the part of the file that is currently being examined. From a higher level, this can be seen as simply emulating a single, large `mmap` buffer. The unfortunate thing about this feature is that it can become extremely slow. The counter-argument to being slower is that it will usually give a better compression factor.

The file `doc/README.benchmarks` has some performance examples to show what kind of data lrzip is good with.

FAQ

> Q: What kind of encryption does lrzip use?

> A: lrzip uses SHA2-512 repetitive hashing of the password along with a salt to provide a key which is used by AES-128 to do block encryption. Each block has more random salts added to the block key. The amount of initial hashing increases as the timestamp goes forward, in direct relation to Moore's law, which means that the amount of time required to encrypt/decrypt the file stays the same on a contemporary computer. It is virtually guaranteed that the same file encrypted with the same password will never be the same twice. The weakest link in this encryption mode by far is the password chosen by the user. There is currently no known attack or backdoor for this encryption mechanism, and there is absolutely no way of retrieving your password should you forget it.

> Q: How do I make a static build?

> A: `./configure --enable-static-bin`

> Q: I want the absolute maximum compression I can possibly get, what do I do?

> A: Try the command line options "-Uzp 1 -L 9". This uses all available ram and ZPAQ compression, and even uses a compression window larger than you have ram. The -p 1 option disables multithreading which improves compression but at the expense of speed. Expect it to take many times longer.

> Q: I want the absolute fastest decent compression I can possibly get.

> A: Try the command line option -l. This will use the lzo backend compression, and level 7 compression (1 isn't much faster).

> Q: How much slower is the unlimited mode?

> A: It depends on 2 things. First, just how much larger than your ram the file is, as the bigger the difference, the slower it will be. The second is how much redundant data there is. The more there is, the slower, but ultimately the better the compression. Why isn't it on by default? If the compression window is a LOT larger than ram, with a lot of redundant information it can be drastically slower. I may revisit this possibility in the future if I can make it any faster.

> Q: Can I use your tool for even more compression than lzma offers?

> A: Yes, the rzip preparation of files makes them more compressible by most other compression technique I have tried. Using the -n option will generate a .lrz file smaller than the original which should be more compressible, and since it is smaller it will compress faster than it otherwise would have.

> Q: 32bit?

> A: 32bit machines have a limit of 2GB sized compression windows due to userspace limitations on mmap and malloc, so even if you have much more ram you will not be able to use compression windows larger than 2GB. Also you may be unable to decompress files compressed on 64bit machines which have used windows larger than 2GB.

> Q: How about 64bit?

> A: 64bit machines with their ability to address massive amounts of ram will excel with lrzip due to being able to use compression windows limited only in size by the amount of physical ram.

> Q: Other operating systems?

> A: The code is POSIXy with GNU extensions. Patches are welcome. Version 0.43+ should build on MacOSX 10.5+

> Q: Does it work on stdin/stdout?

> A: Yes it does. Compression and decompression work well to/from STDIN/STDOUT. However because lrzip does multiple passes on the data, it has to store a large amount in ram before it dumps it to STDOUT (and vice versa), thus it is unable to work with the massive compression windows regular operation provides. Thus the compression afforded on files larger than approximately 25% RAM size will be less efficient (though still benefiting compared to traditional compression formats).

> Q: I have another compression format that is even better than zpaq, can you use that?

> A: You can use it yourself on rzip prepared files (see above). Alternatively if the source code is compatible with the GPL license it can be added to the lrzip source code. Libraries with functions similar to compress() and decompress() functions of zlib would make the process most painless. Please tell me if you have such a library so I can include it :)

> Q: What's this "Starting lzma back end compression thread..." message?

> A: While I'm a big fan of progress percentage being visible, unfortunately lzma compression can't currently be tracked when handing over 100+MB chunks over to the lzma library. Therefore you'll see progress percentage until each chunk is handed over to the lzma library.

> Q: What's this "lz4 testing for incompressible data" message?

> A: Other compression is much slower, and lz4 is the fastest. To help speed up the process, lz4 compression is performed on the data first to test that the data is at all compressible. If a small block of data is not compressible, it tests progressively larger blocks until it has tested all the data (if it fails to compress at all). If no compressible data is found, then the subsequent compression is not even attempted. This can save a lot of time during the compression phase when there is incompressible data. Theoretically it may be possible that data is compressible by the other backend (zpaq, lzma etc) and not at all by lz4, but in practice such data achieves only minuscule amounts of compression which are not worth pursuing. Most of the time it is clear one way or the other that data is compressible or not. If you wish to disable this test and force it to try compressing it anyway, use -T.

> Q: I have truckloads of ram so I can compress files much better, but can my generated file be decompressed on machines with less ram?

> A: Yes. Ram requirements for decompression go up only by the -L compression option with lzma and are never anywhere near as large as the compression requirements. However if you're on 64bit and you use a compression window greater than 2GB, it might not be possible to decompress it on 32bit machines.

> Q: Why are you including bzip2 compression?

> A: To maintain a similar compression format to the original rzip (although the other modes are more useful).

> Q: What about multimedia?

> A: Most multimedia is already in a heavily compressed "lossy" format which by its very nature has very little redundancy. This means that there is not much that can actually be compressed. If your video/audio/picture is in a high bitrate, there will be more redundancy than a low bitrate one making it more suitable to compression. None of the compression techniques in lrzip are optimised for this sort of data. However, the nature of rzip preparation means that you'll still get better compression than most normal compression algorithms give you if you have very large files. ISO images of dvds for example are best compressed directly instead of individual .VOB files. ZPAQ is the only compression format that can do any significant compression of multimedia.

> Q: Is this multithreaded?

> A: As of version 0.540, it is HEAVILY multithreaded with the back end compression and decompression phase, and will continue to process the rzip pre-processing phase so when using one of the more CPU intensive backend compressions like lzma or zpaq, SMP machines will show massive speed improvements. Lrzip will detect the number of CPUs to use, but it can be overridden with the -p option if the slightly better compression is desired more than speed. -p 1 will give the best compression but also be the slowest.

> Q: This uses heaps of memory, can I make it use less?

> A: Well you can by setting -w to the lowest value (1) but the huge use of memory is what makes the compression better than ordinary compression programs so it defeats the point. You'll still derive benefit with -w 1 but not as much.

> Q: What CFLAGS should I use?

> A: With a recent enough compiler (gcc>4) setting both CFLAGS and CXXFLAGS to -O2 -march=native -fomit-frame-pointer

> Q: What compiler does this work with?

> A: It has been tested on gcc, ekopath and the intel compiler successfully previously. Whether the commercial compilers help or not, I could not tell you.

> Q: What codebase are you basing this on?

> A: rzip v2.1 and lzma sdk920, but it should be possible to stay in sync with each of these in the future.

> Q: Do we really need yet another compression format?

> A: It's not really a new one at all; simply a reimplementation of a few very good performing ones that will scale with memory and file size.

> Q: How do you use lrzip yourself?

> A: Three basic uses. I compress large files currently on my drive with the -l option since it is so quick to get a space saving. When archiving data for permanent storage I compress it with the default options. When compressing small files for distribution I use the -z option for the smallest possible size.

> Q: I found a file that compressed better with plain lzma. How can that be?

> A: When the file is more than 5 times the size of the compression window you have available, the efficiency of rzip preparation drops off as a means of getting better compression. Eventually when the file is large enough, plain lzma compression will get better ratios. The lrzip compression will be a lot faster though. The only way around this is to use as large compression windows as possible with -U option.

> Q: Can I use swapspace as ram for lrzip with a massive window?

> A: It will indirectly do this with -U (unlimited) mode enabled. This mode will make the compression window as big as the file itself no matter how big it is, but it will slow down proportionately more the bigger the file is than your ram.

> Q: Why do you nice it to +19 by default? Can I speed up the compression by changing the nice value?

> A: This is a common misconception about what nice values do. They only tell the cpu process scheduler how to prioritise workloads, and if your application is the *only* thing running it will be no faster at nice -20 nor will it be any slower at +19.

> Q: What is the LZ4 Testing option, -T?

> A: LZ4 testing is normally performed for the slower back-end compression of LZMA and ZPAQ. The reasoning is that if it is completely incompressible by LZ4 then it will also be incompressible by them. Thus if a block fails to be compressed by the very fast LZ4, lrzip will not attempt to compress that block with the slower compressor, thereby saving time. If this option is enabled, it will bypass the LZ4 testing and attempt to compress each block regardless.

> Q: Compression and decompression progress on large archives slows down and speeds up. There's also a jump in the percentage at the end?

> A: Yes, that's the nature of the compression/decompression mechanism. The jump is because the rzip preparation makes the amount of data much smaller than the compression backend (lzma) needs to compress.

> Q: Tell me about patented compression algorithms, GPL, lawyers and copyright.

> A: No

> Q: I receive an error "LZMA ERROR: 2. Try a smaller compression window." what does this mean?

> A: LZMA requests large amounts of memory. When a higher compression window is used, there may not be enough contiguous memory for LZMA: LZMA may request up to 25% of TOTAL ram depending on compression level. If contiguous blocks of memory are not free, LZMA will return an error. This is not a fatal error, and a backup mode of compression will be used.

> Q: Where can I get more information about the internals of LZMA?

> A: See http://www.7-zip.org and http://www.p7zip.org. Also, see the file ./lzma/C/lzmalib.h which explains the LZMA properties used and the LZMA memory requirements and computation.

> Q: This version is much slower than the old version?

> A: Make sure you have set CFLAGS and CXXFLAGS. An unoptimised build will be almost 3 times slower.

> Q: Why not update to the latest version of libzpaq?

> A: For reasons that are unclear the later versions of libzpaq create corrupt archives when included with lrzip

> Q: When limiting processors with the -p option, lrzip still spawns twice as many threads as specified with LZMA compression.

> A: LZMA compression uses a separate set of threads of its own. Should the number of threads be critical for your application, set it to half the amount you wish it to use.

> Q: Lrzip uses enormous amounts of resources.

> A: That is by design to obtain the maximum amount of compression in the minimum amount of time. It should not be used on a production server unless coupled with options to limit both its processor and ram resources.

LIMITATIONS

Due to mmap limitations the maximum size a window can be set to is currently 2GB on 32bit unless the -U option is specified. Files generated on 64 bit machines with windows >2GB in size might not be decompressible on 32bit machines. Large files might not decompress on machine with less RAM if SWAP is disabled.

BUGS:

Probably lots. https://github.com/ckolivas/lrzip/issues if you spot any :D

Any known ones should be documented in the file BUGS.

Backends:

rzip: http://rzip.samba.org/

lzo: http://www.oberhumer.com/opensource/lzo/

lzma: http://www.7-zip.org/

zpaq: http://mattmahoney.net/dc/

Thanks (CONTRIBUTORS)

Person(s)	Thanks for
`Andrew Tridgell`	`rzip`
`Markus Oberhumer`	`lzo`
`Igor Pavlov`	`lzma`
`Jean-Loup Gailly & Mark Adler`	`zlib`

Person(s)	Thanks for
Con Kolivas	***Original Code, binding all of this together, managing the project, original*** ***README***
Christian Leber	`lzma` compatibility layer
Michael J Cohen	Darwin/OSX support
Lasse Collin	fixes to `LZMALib.cpp` and `Makefile.in`
Everyone else who coded along the way (add yourself where appropriate if that's you)	Miscellaneous Coding
Peter Hyman	Most of the `0.19` to `0.24` changes
^^^^^^^^^^	Updating the multithreaded `lzma` lib
^^^^^^^^^^	All sorts of other features
René Rhéaume	Fixing executable stacks
Ed Avis	Various fixes
Matt Mahoney	`zpaq` integration code
Jukka Laurila	Additional Darwin/OSX support
George Makrydakis	`lrztar` wrapper
Ulrich Drepper	*special* implementation of md5
Michael Blumenkrantz	New config tools
^^^^^^^^^^^^^^^^^^	`liblrzip`
Authors of `PolarSSL`	Encryption code
Serge Belyshev	Extensive help, advice, and patches to implement secure encryption
Jari Aalto	Fixing typos, esp. in code
Carlo Alberto Ferraris	Code cleanup
Peter Hyman	Additional documentation
Haneef Mubarak	Cleanup, Rewrite, and GH Markdown of `README` --> `README.md`

Persons above are listed in chronological order of first contribution to **lrzip**. Person(s) with names in **bold** have multiple major contributions, person(s) with names in *italics* have made massive contributions, person(s) with names in ***both*** have made innumerable massive contributions.

README Authors

Con Kolivas (`ckolivas` on GitHub) kernel@kolivas.org Tuesday, 16 February 2021: README

Also documented by Peter Hyman pete@peterhyman.com Sun, 04 Jan 2009: README

Releases `2`

Version 0.651 Latest
on Mar 8, 2022

+ 1 release

Packages

No packages published

Contributors 28

+ 14 contributors

Languages

C 74.4% **C++** 21.0% **Shell** 1.7% **M4** 1.5% **Assembly** 0.7% **Makefile** 0.6% **Dockerfile** 0.1%