NAME

libarchive-formats — archive formats supported by the libarchive library

DESCRIPTION

The libarchive(3) library reads and writes a variety of streaming archive formats. Generally speaking, all of these archive formats consist of a series of "entries". Each entry stores a single file system object, such as a file, directory, or symbolic link.

The following provides a brief description of each format supported by libarchive, with some information about recognized extensions or limitations of the current library support. Note that just because a format is supported by libarchive does not imply that a program that uses libarchive will support that format. Applications that use libarchive specify which formats they wish to support, though many programs do use libarchive convenience functions to enable all supported formats.

Tar Formats

The libarchive(3) library can read most tar archives. However, it only writes POSIX-standard "ustar" and "pax interchange" formats.

All tar formats store each entry in one or more 512-byte records. The first record is used for file metadata, including filename, timestamp, and mode information, and the file data is stored in subsequent records. Later variants have extended this by either appropriating undefined areas of the header record, extending the header to multiple records, or by storing special entries that modify the interpretation of subsequent entries.

gnutar The libarchive(3) library can read GNU-format tar archives. It currently supports the most popular GNU extensions, including modern long filename and linkname support, as well as atime and ctime data. The libarchive library does not support multi-volume archives, nor the old GNU long filename format. It can read GNU sparse file entries, including the new POSIX-based formats, but cannot write GNU sparse file entries.

pax The libarchive(3) library can read and write POSIX-compliant pax interchange format archives. Pax interchange format archives are an extension of the older ustar format that adds a separate entry with additional attributes stored as key/value pairs immediately before each regular entry. The presence of these additional entries is the only difference between pax interchange format and the older ustar format. The extended attributes are of unlimited length and are stored as UTF-8 Unicode strings. Keywords defined in the standard are in all lowercase; vendors are allowed to define custom keys by preceding them with the vendor name in all uppercase. When writing pax archives, libarchive uses many of the SCHILY keys defined by Joerg Schilling's "star" archiver and a few LIBARCHIVE keys. The libarchive library can read most of the SCHILY keys and most of the GNU keys introduced by GNU tar. It silently ignores any keywords that it does not understand.

restricted pax
The libarchive library can also write pax archives in which it attempts to suppress the extended attributes entry whenever possible. The result will be identical to a ustar archive unless the extended attributes entry is required to store a long file name, long linkname, extended ACL, file flags, or if any of the standard ustar data (user name, group name, UID, GID, etc) cannot be fully represented in the ustar header. In all cases, the result can be dearchived by any program that can read POSIX-compliant pax interchange format archives. Programs that correctly read ustar format (see below) will also be able to read this format; any extended attributes will be extracted as separate files stored in PaxHeader directories.

ustar The libarchive library can both read and write this format. This format has the following limitations:

- Device major and minor numbers are limited to 21 bits. Nodes with larger numbers will not be added to the archive.
- Path names in the archive are limited to 255 bytes. (Shorter if there is no / character in exactly the right place.)
- Symbolic links and hard links are stored in the archive with the name of the referenced file. This name is limited to 100 bytes.
- Extended attributes, file flags, and other extended security information cannot be stored.
- Archive entries are limited to 8 gigabytes in size.

Note that the pax interchange format has none of these restrictions.

The libarchive library also reads a variety of commonly-used extensions to the basic tar format. These extensions are recognized automatically whenever they appear.

Numeric extensions.
: The POSIX standards require fixed-length numeric fields to be written with some character position reserved for terminators. Libarchive allows these fields to be written without terminator characters. This extends the allowable range; in particular, ustar archives with this extension can support entries up to 64 gigabytes in size. Libarchive also recognizes base-256 values in most numeric fields. This essentially removes all limitations on file size, modification time, and device numbers.

Solaris extensions
: Libarchive recognizes ACL and extended attribute records written by Solaris tar. Currently, libarchive only has support for old-style ACLs; the newer NFSv4 ACLs are recognized but discarded.

The first tar program appeared in Seventh Edition Unix in 1979. The first official standard for the tar file format was the “ustar” (Unix Standard Tar) format defined by POSIX in 1988. POSIX.1-2001 extended the ustar format to create the “pax interchange” format.

Cpio Formats

The libarchive library can read a number of common cpio variants and can write “odc” and “newc” format archives. A cpio archive stores each entry as a fixed-size header followed by a variable-length filename and variable-length data. Unlike the tar format, the cpio format does only minimal padding of the header or file data. There are several cpio variants, which differ primarily in how they store the initial header: some store the values as octal or hexadecimal numbers in ASCII, others as binary values of varying byte order and length.

`binary` The libarchive library transparently reads both big-endian and little-endian variants of the original binary cpio format. This format used 32-bit binary values for file size and mtime, and 16-bit binary values for the other fields.

`odc` The libarchive library can both read and write this POSIX-standard format, which is officially known as the “cpio interchange format” or the “octet-oriented cpio archive format” and sometimes unofficially referred to as the “old character format”. This format stores the header contents as octal values in ASCII. It is standard, portable, and immune from byte-order confusion. File sizes and mtime are limited to 33 bits (8GB file size), other fields are limited to 18 bits.

`SVR4` The libarchive library can read both CRC and non-CRC variants of this format. The SVR4 format uses eight-digit hexadecimal values for all header fields. This limits file size to 4GB, and also limits the mtime and other fields to 32 bits. The SVR4 format can optionally include a CRC of the file contents, although libarchive does not currently verify this CRC.

Cpio first appeared in PWB/UNIX 1.0, which was released within AT&T in 1977. PWB/UNIX 1.0 formed the basis of System III Unix, released outside of AT&T in 1981. This makes cpio older than tar, although cpio was not included in Version 7 AT&T Unix. As a result, the tar command became much better known in

universities and research groups that used Version 7. The combination of the **find** and **cpio** utilities provided very precise control over file selection. Unfortunately, the format has many limitations that make it unsuitable for widespread use. Only the POSIX format permits files over 4GB, and its 18-bit limit for most other fields makes it unsuitable for modern systems. In addition, cpio formats only store numeric UID/GID values (not usernames and group names), which can make it very difficult to correctly transfer archives across systems with dissimilar user numbering.

Shar Formats

A "shell archive" is a shell script that, when executed on a POSIX-compliant system, will recreate a collection of file system objects. The libarchive library can write two different kinds of shar archives:

shar The traditional shar format uses a limited set of POSIX commands, including echo(1), mkdir(1), and sed(1). It is suitable for portably archiving small collections of plain text files. However, it is not generally well-suited for large archives (many implementations of sh(1) have limits on the size of a script) nor should it be used with non-text files.

shardump

This format is similar to shar but encodes files using uuencode(1) so that the result will be a plain text file regardless of the file contents. It also includes additional shell commands that attempt to reproduce as many file attributes as possible, including owner, mode, and flags. The additional commands used to restore file attributes make shardump archives less portable than plain shar archives.

ISO9660 format

Libarchive can read and extract from files containing ISO9660-compliant CDROM images. In many cases, this can remove the need to burn a physical CDROM just in order to read the files contained in an ISO9660 image. It also avoids security and complexity issues that come with virtual mounts and loopback devices. Libarchive supports the most common Rockridge extensions and has partial support for Joliet extensions. If both extensions are present, the Joliet extensions will be used and the Rockridge extensions will be ignored. In particular, this can create problems with hardlinks and symlinks, which are supported by Rockridge but not by Joliet.

Zip format

Libarchive can read and write zip format archives that have uncompressed entries and entries compressed with the "deflate" algorithm. Older zip compression algorithms are not supported. It can extract jar archives, archives that use Zip64 extensions and many self-extracting zip archives. Libarchive reads Zip archives as they are being streamed, which allows it to read archives of arbitrary size. It currently does not use the central directory; this limits libarchive's ability to support some self-extracting archives and ones that have been modified in certain ways.

Archive (library) file format

The Unix archive format (commonly created by the ar(1) archiver) is a general-purpose format which is used almost exclusively for object files to be read by the link editor ld(1). The ar format has never been standardised. There are two common variants: the GNU format derived from SVR4, and the BSD format, which first appeared in 4.4BSD. The two differ primarily in their handling of filenames longer than 15 characters: the GNU/SVR4 variant writes a filename table at the beginning of the archive; the BSD format stores each long filename in an extension area adjacent to the entry. Libarchive can read both extensions, including archives that may include both types of long filenames. Programs using libarchive can write GNU/SVR4 format if they provide a filename table to be written into the archive before any of the entries. Any entries whose names are not in the filename table will be written using BSD-style long filenames. This can cause problems for programs such as GNU ld that do not support the BSD-style long filenames.

mtree

Libarchive can read and write files in `mtree`(5) format. This format is not a true archive format, but rather a textual description of a file hierarchy in which each line specifies the name of a file and provides specific metadata about that file. Libarchive can read all of the keywords supported by both the NetBSD and FreeBSD versions of `mtree`(1), although many of the keywords cannot currently be stored in an archive_entry object. When writing, libarchive supports use of the `archive_write_set_options`(3) interface to specify which keywords should be included in the output. If libarchive was compiled with access to suitable cryptographic libraries (such as the OpenSSL libraries), it can compute hash entries such as **`sha512`** or **`md5`** from file data being written to the mtree writer.

When reading an mtree file, libarchive will locate the corresponding files on disk using the **`contents`** keyword if present or the regular filename. If it can locate and open the file on disk, it will use that to fill in any metadata that is missing from the mtree file and will read the file contents and return those to the program using libarchive. If it cannot locate and open the file on disk, libarchive will return an error for any attempt to read the entry body.

SEE ALSO

`ar`(1), `cpio`(1), `mkisofs`(1), `shar`(1), `tar`(1), `zip`(1), `zlib`(3), `cpio`(5), `mtree`(5), `tar`(5)