NAME

llvm−ar − LLVM archiver

SYNOPSIS

llvm-ar [−]{dmpqrtx}[Rabfikouz] [relpos] [count] <archive> [files...]

DESCRIPTION

The **llvm-ar** command is similar to the common Unix utility, `ar`. It archives several files together into a single file. The intent for this is to produce archive libraries by LLVM bitcode that can be linked into an LLVM program. However, the archive can contain any kind of file. By default, **llvm-ar** generates a symbol table that makes linking faster because only the symbol table needs to be consulted, not each individual file member of the archive.

The **llvm-ar** command can be used to *read* both SVR4 and BSD style archive files. However, it cannot be used to write them. While the **llvm-ar** command produces files that are *almost* identical to the format used by other `ar` implementations, it has two significant departures in order to make the archive appropriate for LLVM. The first departure is that **llvm-ar** only uses BSD4.4 style long path names (stored immediately after the header) and never contains a string table for long names. The second departure is that the symbol table is formated for efficient construction of an in-memory data structure that permits rapid (red−black tree) lookups. Consequently, archives produced with **llvm-ar** usually won't be readable or editable with any `ar` implementation or useful for linking. Using the `f` modifier to flatten file names will make the archive readable by other `ar` implementations but not for linking because the symbol table format for LLVM is unique. If an SVR4 or BSD style archive is used with the `r` (replace) or `q` (quick update) operations, the archive will be reconstructed in LLVM format. This means that the string table will be dropped (in deference to BSD 4.4 long names) and an LLVM symbol table will be added (by default). The system symbol table will be retained.

Here's where **llvm-ar** departs from previous `ar` implementations:

Symbol Table

Since **llvm-ar** is intended to archive bitcode files, the symbol table won't make much sense to anything but LLVM. Consequently, the symbol table's format has been simplified. It consists simply of a sequence of pairs of a file member index number as an LSB 4byte integer and a null-terminated string.

Long Paths

Some `ar` implementations (SVR4) use a separate file member to record long path names (> 15 characters). **llvm-ar** takes the BSD 4.4 and Mac OS X approach which is to simply store the full path name immediately preceding the data for the file. The path name is null terminated and may contain the slash (/) character.

Compression

llvm-ar can compress the members of an archive to save space. The compression used depends on what's available on the platform and what choices the LLVM Compressor utility makes. It generally favors bzip2 but will select between "no compression" or bzip2 depending on what makes sense for the file's content.

Directory Recursion

Most `ar` implementations do not recurse through directories but simply ignore directories if they are presented to the program in the *files* option. **llvm-ar**, however, can recurse through directory structures and add all the files under a directory, if requested.

TOC Verbose Output

When **llvm-ar** prints out the verbose table of contents (`tv` option), it precedes the usual output with a character indicating the basic kind of content in the file. A blank means the file is a regular file. A 'Z' means the file is compressed. A 'B' means the file is an LLVM bitcode file. An 'S' means the file is the symbol table.

OPTIONS

The options to **llvm-ar** are compatible with other `ar` implementations. However, there are a few modifiers (*zR*) that are not found in other `ar`s. The options to **llvm-ar** specify a single basic operation to perform on the archive, a variety of modifiers for that operation, the name of the archive file, and an optional list of file names. These options are used to determine how **llvm-ar** should process the archive file.

The Operations and Modifiers are explained in the sections below. The minimal set of options is at least one operator and the name of the archive. Typically archive files end with a `.a` suffix, but this is not required. Following the *archive-name* comes a list of *files* that indicate the specific members of the archive to operate on. If the *files* option is not specified, it generally means either ''none'' or ''all'' members, depending on the operation.

Operations

d
: Delete files from the archive. No modifiers are applicable to this operation. The *files* options specify which members should be removed from the archive. It is not an error if a specified file does not appear in the archive. If no *files* are specified, the archive is not modified.

m[abi]
: Move files from one location in the archive to another. The *a*, *b*, and *i* modifiers apply to this operation. The *files* will all be moved to the location given by the modifiers. If no modifiers are used, the files will be moved to the end of the archive. If no *files* are specified, the archive is not modified.

p[k]
: Print files to the standard output. The *k* modifier applies to this operation. This operation simply prints the *files* indicated to the standard output. If no *files* are specified, the entire archive is printed. Printing bitcode files is ill-advised as they might confuse your terminal settings. The *p* operation never modifies the archive.

q[Rfz]
: Quickly append files to the end of the archive. The *R*, *f*, and *z* modifiers apply to this operation. This operation quickly adds the *files* to the archive without checking for duplicates that should be removed first. If no *files* are specified, the archive is not modified. Because of the way that **llvm-ar** constructs the archive file, its dubious whether the *q* operation is any faster than the *r* operation.

r[Rabfuz]
: Replace or insert file members. The *R*, *a*, *b*, *f*, *u*, and *z* modifiers apply to this operation. This operation will replace existing *files* or insert them at the end of the archive if they do not exist. If no *files* are specified, the archive is not modified.

t[v]
: Print the table of contents. Without any modifiers, this operation just prints the names of the members to the standard output. With the *v* modifier, **llvm-ar** also prints out the file type (B=bitcode, Z=compressed, S=symbol table, blank=regular file), the permission mode, the owner and group, the size, and the date. If any *files* are specified, the listing is only for those files. If no *files* are specified, the table of contents for the whole archive is printed.

x[oP]
: Extract archive members back to files. The *o* modifier applies to this operation. This operation retrieves the indicated *files* from the archive and writes them back to the operating system's file system. If no *files* are specified, the entire archive is extract.

Modifiers (operation specific)

The modifiers below are specific to certain operations. See the Operations section (above) to determine which modifiers are applicable to which operations.

[a]
: When inserting or moving member files, this option specifies the destination of the new files as being `a`fter the *relpos* member. If *relpos* is not found, the files are placed at the end of the archive.

[b]
: When inserting or moving member files, this option specifies the destination of the new files as being `b`efore the *relpos* member. If *relpos* is not found, the files are placed at the end of the archive. This modifier is identical to the the *i* modifier.

[f] Normally, **llvm-ar** stores the full path name to a file as presented to it on the command line. With this option, truncated (15 characters max) names are used. This ensures name compatibility with older versions of `ar` but may also thwart correct extraction of the files (duplicates may overwrite). If used with the *R* option, the directory recursion will be performed but the file names will all be `f`lattened to simple file names.

[i] A synonym for the *b* option.

[k] Normally, **llvm-ar** will not print the contents of bitcode files when the *p* operation is used. This modifier defeats the default and allows the bitcode members to be printed.

[N] This option is ignored by **llvm-ar** but provided for compatibility.

[o] When extracting files, this option will cause **llvm-ar** to preserve the original modification times of the files it writes.

[P] use full path names when matching

[R] This modifier instructions the *r* option to recursively process directories. Without *R*, directories are ignored and only those *files* that refer to files will be added to the archive. When *R* is used, any directories specified with *files* will be scanned (recursively) to find files to be added to the archive. Any file whose name begins with a dot will not be added.

[u] When replacing existing files in the archive, only replace those files that have a time stamp than the time stamp of the member in the archive.

[z] When inserting or replacing any file in the archive, compress the file first. This modifier is safe to use when (previously) compressed bitcode files are added to the archive; the compressed bitcode files will not be doubly compressed.

Modifiers (generic)

The modifiers below may be applied to any operation.

[c] For all operations, **llvm-ar** will always create the archive if it doesn't exist. Normally, **llvm-ar** will print a warning message indicating that the archive is being created. Using this modifier turns off that warning.

[s] This modifier requests that an archive index (or symbol table) be added to the archive. This is the default mode of operation. The symbol table will contain all the externally visible functions and global variables defined by all the bitcode files in the archive. Using this modifier is more efficient that using llvm-ranlib which also creates the symbol table.

[S] This modifier is the opposite of the *s* modifier. It instructs **llvm-ar** to not build the symbol table. If both *s* and *S* are used, the last modifier to occur in the options will prevail.

[v] This modifier instructs **llvm-ar** to be verbose about what it is doing. Each editing operation taken against the archive will produce a line of output saying what is being done.

STANDARDS

The **llvm-ar** utility is intended to provide a superset of the IEEE Std 1003.2 (POSIX.2) functionality for `ar`. **llvm-ar** can read both SVR4 and BSD4.4 (or Mac OS X) archives. If the `f` modifier is given to the `x` or `r` operations then **llvm-ar** will write SVR4 compatible archives. Without this modifier, **llvm-ar** will write BSD4.4 compatible archives that have long names immediately after the header and indicated using the "#1/ddd" notation for the name in the header.

FILE FORMAT

The file format for LLVM Archive files is similar to that of BSD 4.4 or Mac OSX archive files. In fact, except for the symbol table, the `ar` commands on those operating systems should be able to read LLVM archive files. The details of the file format follow.

Each archive begins with the archive magic number which is the eight printable characters "!<arch>\n" where \n represents the newline character (0x0A). Following the magic number, the file is composed of even length members that begin with an archive header and end with a \n padding character if necessary (to make the length even). Each file member is composed of a header (defined below), an optional newline-terminated ''long file name'' and the contents of the file.

The fields of the header are described in the items below. All fields of the header contain only ASCII characters, are left justified and are right padded with space characters.

name – char[16]
: This field of the header provides the name of the archive member. If the name is longer than 15 characters or contains a slash (/) character, then this field contains `#1/nnn` where `nnn` provides the length of the name and the `#1/` is literal. In this case, the actual name of the file is provided in the `nnn` bytes immediately following the header. If the name is 15 characters or less, it is contained directly in this field and terminated with a slash (/) character.

date – char[12]
: This field provides the date of modification of the file in the form of a decimal encoded number that provides the number of seconds since the epoch (since 00:00:00 Jan 1, 1970) per Posix specifications.

uid – char[6]
: This field provides the user id of the file encoded as a decimal ASCII string. This field might not make much sense on non-Unix systems. On Unix, it is the same value as the st_uid field of the stat structure returned by the *stat* (2) operating system call.

gid – char[6]
: This field provides the group id of the file encoded as a decimal ASCII string. This field might not make much sense on non-Unix systems. On Unix, it is the same value as the st_gid field of the stat structure returned by the *stat* (2) operating system call.

mode – char[8]
: This field provides the access mode of the file encoded as an octal ASCII string. This field might not make much sense on non-Unix systems. On Unix, it is the same value as the st_mode field of the stat structure returned by the *stat* (2) operating system call.

size – char[10]
: This field provides the size of the file, in bytes, encoded as a decimal ASCII string. If the size field is negative (starts with a minus sign, 0x02D), then the archive member is stored in compressed form. The first byte of the archive member's data indicates the compression type used. A value of 0 (0x30) indicates that no compression was used. A value of 2 (0x32) indicates that bzip2 compression was used.

fmag – char[2]
: This field is the archive file member magic number. Its content is always the two characters back tick (0x60) and newline (0x0A). This provides some measure utility in identifying archive files that have been corrupted.

The LLVM symbol table has the special name ''#_LLVM_SYM_TAB_#''. It is presumed that no regular archive member file will want this name. The LLVM symbol table is simply composed of a sequence of triplets: byte offset, length of symbol, and the symbol itself. Symbols are not null or newline terminated. Here are the details on each of these items:

offset – vbr encoded 32–bit integer
: The offset item provides the offset into the archive file where the bitcode member is stored that is associated with the symbol. The offset value is 0 based at the start of the first ''normal'' file member. To derive the actual file offset of the member, you must add the number of bytes occupied by the file signature (8 bytes) and the symbol tables. The value of this item is encoded using variable bit rate encoding to reduce the size of the symbol table. Variable bit rate encoding uses the high bit (0x80) of each byte to indicate if there are more bytes to follow. The remaining 7 bits in each byte carry bits from the value. The final byte does not have the high bit set.

length – vbr encoded 32–bit integer
: The length item provides the length of the symbol that follows. Like this *offset* item, the length is variable bit rate encoded.

symbol – character array

The symbol item provides the text of the symbol that is associated with the *offset*. The symbol is not terminated by any character. Its length is provided by the *length* field. Note that is allowed (but unwise) to use non-printing characters (even 0x00) in the symbol. This allows for multiple encodings of symbol names.

EXIT STATUS

If **llvm-ar** succeeds, it will exit with 0. A usage error, results in an exit code of 1. A hard (file system typically) error results in an exit code of 2. Miscellaneous or unknown errors result in an exit code of 3.

SEE ALSO

llvm-ranlib, *ar*(1)

AUTHORS

Maintained by the LLVM Team (<http://llvm.org/>).