The GNU Binary Utilities

Version 2.2

May 1993

Roland H. Pesch
Cygnus Support

Cygnus Support
Revision: 1.22
TEXinfo 2021-02-20.11

2 ar

```
ar [-]pmod [ membername ] archive file...
ar -M [ <mri-script ]
```

The GNU `ar` program creates, modifies, and extracts from archives. An *archive* is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called *members* of the archive).

The original files' contents, mode (permissions), timestamp, owner, and group are preserved in the archive, and can be restored on extraction.

GNU `ar` can maintain archives whose members have names of any length; however, depending on how `ar` is configured on your system, a limit on member-name length may be imposed for compatibility with archive formats maintained with other tools. If it exists, the limit is often 15 characters (typical of formats related to a.out) or 16 characters (typical of formats related to coff).

`ar` is considered a binary utility because archives of this sort are most often used as *libraries* holding commonly needed subroutines.

`ar` creates an index to the symbols defined in relocatable object modules in the archive when you specify the modifier '`s`'. Once created, this index is updated in the archive whenever `ar` makes a change to its contents (save for the '`q`' update operation). An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.

You may use '`nm -s`' or '`nm --print-armap`' to list this index table. If an archive lacks the table, another form of `ar` called `ranlib` can be used to add just the table.

GNU `ar` is designed to be compatible with two different facilities. You can control its activity using command-line options, like the different varieties of `ar` on Unix systems; or, if you specify the single command-line option '`-M`', you can control it with a script supplied via standard input, like the MRI "librarian" program.

2.1 Controlling ar on the command line

```
ar [-]pmod [ membername ] archive file...
```

When you use `ar` in the Unix style, `ar` insists on at least two arguments to execute: one keyletter specifying the *operation* (optionally accompanied by other keyletters specifying *modifiers*), and the archive name to act on.

Most operations can also accept further *file* arguments, specifying particular files to operate on.

GNU `ar` allows you to mix the operation code *p* and modifier flags *mod* in any order, within the first command-line argument.

If you wish, you may begin the first command-line argument with a dash.

The *p* keyletter specifies what operation to execute; it may be any of the following, but you must specify only one of them:

`d` *Delete* modules from the archive. Specify the names of modules to be deleted as *file*...; the archive is untouched if you specify no files to delete.

If you specify the '`v`' modifier, `ar` lists each module as it is deleted.

`m` Use this operation to *move* members in an archive.

The ordering of members in an archive can make a difference in how programs are linked using the library, if a symbol is defined in more than one member.

If no modifiers are used with `m`, any members you name in the *file* arguments are moved to the *end* of the archive; you can use the '`a`', '`b`', or '`i`' modifiers to move them to a specified place instead.

`p` *Print* the specified members of the archive, to the standard output file. If the '`v`' modifier is specified, show the member name before copying its contents to standard output.

If you specify no *file* arguments, all the files in the archive are printed.

`q` *Quick append*; add the files *file*... to the end of *archive*, without checking for replacement.

The modifiers '`a`', '`b`', and '`i`' do *not* affect this operation; new members are always placed at the end of the archive.

The modifier '`v`' makes `ar` list each file as it is appended.

Since the point of this operation is speed, the archive's symbol table index is not updated, even if it already existed; you can use '`ar s`' or `ranlib` explicitly to update the symbol table index.

`r` Insert the files *file*... into *archive* (with *replacement*). This operation differs from '`q`' in that any previously existing members are deleted if their names match those being added.

If one of the files named in *file*... doesn't exist, `ar` displays an error message, and leaves undisturbed any existing members of the archive matching that name.

By default, new members are added at the end of the file; but you may use one of the modifiers '`a`', '`b`', or '`i`' to request placement relative to some existing member.

The modifier '`v`' used with this operation elicits a line of output for each file inserted, along with one of the letters '`a`' or '`r`' to indicate whether the file was appended (no old member deleted) or replaced.

`t` Display a *table* listing the contents of *archive*, or those of the files listed in *file*... that are present in the archive. Normally only the member name is shown; if you also want to see the modes (permissions), timestamp, owner, group, and size, you can request that by also specifying the '`v`' modifier.

If you do not specify a *file*, all files in the archive are listed.

If there is more than one file with the same name (say, '`fie`') in an archive (say '`b.a`'), '`ar t b.a fie`' lists only the first instance; to see them all, you must ask for a complete listing—in our example, '`ar t b.a`'.

`x` *Extract* members (named *file*) from the archive. You can use the '`v`' modifier with this operation, to request that `ar` list each name as it extracts it.

If you do not specify a *file*, all files in the archive are extracted.

A number of modifiers (*mod*) may immediately follow the *p* keyletter, to specify variations on an operation's behavior:

`a` Add new files *after* an existing member of the archive. If you use the modifier '`a`', the name of an existing archive member must be present as the *membername* argument, before the *archive* specification.

`b` Add new files *before* an existing member of the archive. If you use the modifier '`b`', the name of an existing archive member must be present as the *membername* argument, before the *archive* specification. (same as '`i`').

`c` *Create* the archive. The specified *archive* is always created if it didn't exist, when you request an update. But a warning is issued unless you specify in advance that you expect to create it, by using this modifier.

`i` Insert new files *before* an existing member of the archive. If you use the modifier '`i`', the name of an existing archive member must be present as the *membername* argument, before the *archive* specification. (same as '`b`').

`l` This modifier is accepted but not used.

`o` Preserve the *original* dates of members when extracting them. If you do not specify this modifier, files extracted from the archive are stamped with the time of extraction.

`s` Write an object-file index into the archive, or update an existing one, even if no other change is made to the archive. You may use this modifier flag either with any operation, or alone. Running '`ar s`' on an archive is equivalent to running '`ranlib`' on it.

`u` Normally, '`ar r`'... inserts all files listed into the archive. If you would like to insert *only* those of the files you list that are newer than existing members of the same names, use this modifier. The '`u`' modifier is allowed only for the operation '`r`' (replace). In particular, the combination '`qu`' is not allowed, since checking the timestamps would lose any speed advantage from the operation '`q`'.

`v` This modifier requests the *verbose* version of an operation. Many operations display additional information, such as filenames processed, when the modifier '`v`' is appended.

`V` This modifier shows the version number of `ar`.

2.2 Controlling `ar` with a script

```
ar -M [ <script ]
```

If you use the single command-line option '`-M`' with `ar`, you can control its operation with a rudimentary command language. This form of `ar` operates interactively if standard input is coming directly from a terminal. During interactive use, `ar` prompts for input (the prompt is '`AR >`'), and continues executing even after errors. If you redirect standard input to a script file, no prompts are issued, and `ar` abandons execution (with a nonzero exit code) on any error.

The `ar` command language is *not* designed to be equivalent to the command-line options; in fact, it provides somewhat less control over archives. The only purpose of the command language is to ease the transition to GNU `ar` for developers who already have scripts written for the MRI "librarian" program.

The syntax for the `ar` command language is straightforward:

- commands are recognized in upper or lower case; for example, `LIST` is the same as `list`. In the following descriptions, commands are shown in upper case for clarity.
- a single command may appear on each line; it is the first word on the line.
- empty lines are allowed, and have no effect.
- comments are allowed; text after either of the characters '`*`' or '`;`' is ignored.
- Whenever you use a list of names as part of the argument to an `ar` command, you can separate the individual names with either commas or blanks. Commas are shown in the explanations below, for clarity.
- '`+`' is used as a line continuation character; if '`+`' appears at the end of a line, the text on the following line is considered part of the current command.

Here are the commands you can use in `ar` scripts, or when using `ar` interactively. Three of them have special significance:

`OPEN` or `CREATE` specify a *current archive*, which is a temporary file required for most of the other commands.

`SAVE` commits the changes so far specified by the script. Prior to `SAVE`, commands affect only the temporary copy of the current archive.

`ADDLIB` *archive*
`ADDLIB` *archive* (*module*, *module*, ... *module*)
: Add all the contents of *archive* (or, if specified, each named *module* from *archive*) to the current archive.

  Requires prior use of `OPEN` or `CREATE`.

`ADDMOD` *file*, *file*, ... *file*
: Add each named *file* as a module in the current archive.

  Requires prior use of `OPEN` or `CREATE`.

`CLEAR` Discard the contents of the current archive, cancelling the effect of any operations since the last `SAVE`. May be executed (with no effect) even if no current archive is specified.

`CREATE` *archive*
Creates an archive, and makes it the current archive (required for many other commands). The new archive is created with a temporary name; it is not actually saved as *archive* until you use `SAVE`. You can overwrite existing archives; similarly, the contents of any existing file named *archive* will not be destroyed until `SAVE`.

`DELETE` *module, module, ... module*
Delete each listed *module* from the current archive; equivalent to '`ar -d` *archive module ... module*'.

Requires prior use of `OPEN` or `CREATE`.

`DIRECTORY` *archive* (*module, ... module*)
`DIRECTORY` *archive* (*module, ... module*) *outputfile*
List each named *module* present in *archive*. The separate command `VERBOSE` specifies the form of the output: when verbose output is off, output is like that of '`ar -t` *archive module...*'. When verbose output is on, the listing is like '`ar -tv` *archive module...*'.

Output normally goes to the standard output stream; however, if you specify *outputfile* as a final argument, `ar` directs the output to that file.

`END` Exit from `ar`, with a `0` exit code to indicate successful completion. This command does not save the output file; if you have changed the current archive since the last `SAVE` command, those changes are lost.

`EXTRACT` *module, module, ... module*
Extract each named *module* from the current archive, writing them into the current directory as separate files. Equivalent to '`ar -x` *archive module...*'.

Requires prior use of `OPEN` or `CREATE`.

`LIST` Display full contents of the current archive, in "verbose" style regardless of the state of `VERBOSE`. The effect is like '`ar tv` *archive*'). (This single command is a GNU `ld` enhancement, rather than present for MRI compatibility.)

Requires prior use of `OPEN` or `CREATE`.

`OPEN` *archive*
Opens an existing archive for use as the current archive (required for many other commands). Any changes as the result of subsequent commands will not actually affect *archive* until you next use `SAVE`.

`REPLACE` *module, module, ... module*
In the current archive, replace each existing *module* (named in the `REPLACE` arguments) from files in the current working directory. To execute this command without errors, both the file, and the module in the current archive, must exist.

Requires prior use of `OPEN` or `CREATE`.

`VERBOSE` Toggle an internal flag governing the output from `DIRECTORY`. When the flag is on, `DIRECTORY` output matches output from '`ar -tv` '....

`SAVE` Commit your changes to the current archive, and actually save it as a file with the name specified in the last `CREATE` or `OPEN` command.

Requires prior use of `OPEN` or `CREATE`.

3 objcopy

```
objcopy [ -F format | --format=format ]
        [ -I format | --input-format=format ]
        [ -O format | --output-format=format ]
        [ -S | --strip-all ]  [ -g | --strip-debug ]
        [ -x | --discard-all ]  [ -X | --discard-locals ]
        [ -v | --verbose ]  [ -V | --version ]
        infile [outfile]
```

The GNU `objcopy` utility copies the contents of an object file to another. `objcopy` uses the GNU BFD Library to read and write the object files. It can write the destination object file in a format different from that of the source object file. The exact behavior of `objcopy` is controlled by command-line options.

`objcopy` creates temporary files to do its translations and deletes them afterward. `objcopy` uses BFD to do all its translation work; it knows about all the formats BFD knows about, and thus is able to recognize most formats without being told explicitly. See Section "BFD" in *Using LD*.

infile
outfile
: The source and output files respectively. If you do not specify *outfile*, `objcopy` creates a temporary file and destructively renames the result with the name of the input file.

`-I` *format*
`--input-format=`*format*
: Consider the source file's object format to be *format*, rather than attempting to deduce it.

`-O` *format*
`--output-format=`*format*
: Write the output file using the object format *format*.

`-F` *format*
`--format=`*format*
: Use *format* as the object format for both the input and the output file; i.e. simply transfer data from source to destination with no translation.

`-S`
`--strip-all`
: Do not copy relocation and symbol information from the source file.

`-g`
`--strip-debug`
: Do not copy debugging symbols from the source file.

`-x`
`--discard-all`
: Do not copy non-global symbols from the source file.

`-X`
`--discard-locals`
: Do not copy compiler-generated local symbols. (These usually start with '`L`' or '`.`'.)

`-V`
`--version`
: Show the version number of `objcopy`.

`-v`
`--verbose`
: Verbose output: list all object files modified. In the case of archives, '`objcopy -V`' lists all members of the archive.

4 ld

The GNU linker `ld` is now described in a separate manual. See Section "Overview" in *Using LD: the GNU linker*.

5 nm

```
nm [ -a | --debug-syms ]  [ -g | --extern-only ]
   [ -s | --print-armap ]  [ -o | --print-file-name ]
   [ -n | --numeric-sort ]  [ -p | --no-sort ]
   [ -r | --reverse-sort ]  [ -u | --undefined-only ]
   [ --target=bfdname ]
   [ objfile... ]
```

GNU `nm` lists the symbols from object files *objfile*....

The long and short forms of options, shown here as alternatives, are equivalent.

objfile...
: Object files whose symbols are to be listed. If no object files are listed as arguments, `nm` assumes '`a.out`'.

`-a`
`--debug-syms`
: Display debugger-only symbols; normally these are not listed.

`-g`
`--extern-only`
: Display only external symbols.

`-p`
`--no-sort`
: Don't bother to sort the symbols in any order; just print them in the order encountered.

`-n`
`--numeric-sort`
: Sort symbols numerically by their addresses, rather than alphabetically by their names.

`-s`
`--print-armap`
: When listing symbols from archive members, include the index: a mapping (stored in the archive by `ar` or `ranlib`) of which modules contain definitions for which names.

`-o`
`--print-file-name`
: Precede each symbol by the name of the input file where it was found, rather than identifying the input file once only before all of its symbols.

`-r`
`--reverse-sort`
: Reverse the order of the sort (whether numeric or alphabetic); let the last come first.

`--target=`*`bfdname`*
: Specify an object code format other than your system's default format. See Chapter 6 [objdump], page 13, for information on listing available formats.

`-u`
`--undefined-only`
: Display only undefined symbols (those external to each object file).

6 objdump

```
objdump [ -a ]  [ -b bfdname ]  [ -d ]  [ -f ]
        [ -h | --header ]  [ -i ]  [ -j section ]  [ -l ]
        [ -m machine ]  [ -r | --reloc ]  [ -s ]
        [ --stabs ]  [ -t | --syms ]  [ -x ]
        objfile...
```

`objdump` displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

The long and short forms of options, shown here as alternatives, are equivalent.

objfile...
: The object files to be examined. When you specify archives, `objdump` shows information on each of the member object files.

`-a`
: If any of the *objfile* files are archives, display the archive header information (in a format similar to '`ls -l`'). Besides the information you could list with '`ar tv`', '`objdump -a`' shows the object file format of each archive member.

`-b` *bfdname*
: Specify that the object-code format for the object files is *bfdname*. This option may not be necessary; *objdump* can automatically recognize many formats.

  For example,

  ```
  objdump -b oasys -m vax -h fu.o
  ```

  displays summary information from the section headers ('`-h`') of `fu.o`, which is explicitly identified ('`-m`') as a VAX object file in the format produced by Oasys compilers. You can list the formats available with the '`-i`' option.

`-d`
: Disassemble. Display the assembler mnemonics for the machine instructions from *objfile*.

`-f`
: File header. Display summary information from the overall header of each of the *objfile* files.

`-h`
`--header`
: Header. Display summary information from the section headers of the object file.

`-i`
: Display a list showing all architectures and object formats available for specification with '`-b`' or '`-m`'.

`-j` *name*
: Display information only for section *name*.

`-l`
: Label the display (using debugging information) with the source filename and line numbers corresponding to the object code shown.

`-m` *machine*
: Specify that the object files *objfile* are for architecture *machine*. You can list available architectures using the '`-i`' option.

`-r`
`--reloc`
: Relocation. Print the relocation entries of the file.

`-s` Display the full contents of any sections requested.

`--stabs` Display the full contents of any sections requested. Display the contents of the .stab and .stab.index and .stab.excl sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which `.stab` debugging symbol-table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the '`--syms`' output.

`-t`
`--syms` Symbol Table. Print the symbol table entries of the file. This is similar to the information provided by the '`nm`' program.

`-x` Display all available header information, including the symbol table and relocation entries. Using '`-x`' is equivalent to specifying all of '`-a -f -h -r -t`'.

7 ranlib

```
ranlib [-vV] archive
```

`ranlib` generates an index to the contents of an archive and stores it in the archive. The index lists each symbol defined by a member of an archive that is a relocatable object file.

You may use '`nm -s`' or '`nm --print-armap`' to list this index.

An archive with such an index speeds up linking to the library and allows routines in the library to call each other without regard to their placement in the archive.

The GNU `ranlib` program is another form of GNU `ar`; running `ranlib` is completely equivalent to executing '`ar -s`'. See Chapter 2 [ar], page 1.

`-v`
`-V` Show the version number of `ranlib`.

8 size

```
size [ -A | -B | --format=compatibility ]
     [ --help ]  [ -d | -o | -x | --radix=number ]
     [ --target=bfdname ]  [ -V | --version ]
     objfile...
```

The GNU `size` utility lists the section sizes—and the total size—for each of the object or archive files *objfile* in its argument list. By default, one line of output is generated for each object file or each module in an archive.

The command line options have the following meanings:

objfile...
: The object files to be examined.

`-A`
`-B`
`--format=`*compatibility*
: Using one of these options, you can choose whether the output from GNU `size` resembles output from System V `size` (using '`-A`', or '`--format=sysv`'), or Berkeley `size` (using '`-B`', or '`--format=berkeley`'). The default is the one-line format similar to Berkeley's.

  Here is an example of the Berkeley (default) format of output from `size`:

```
size --format Berkeley ranlib size
text    data    bss     dec     hex     filename
294880  81920   11592   388392  5ed28   ranlib
294880  81920   11888   388688  5ee50   size
```

  This is the same data, but displayed closer to System V conventions:

```
size --format SysV ranlib size
ranlib  :
section         size         addr
.text         294880         8192
.data          81920       303104
.bss           11592       385024
Total         388392


size  :
section         size         addr
.text         294880         8192
.data          81920       303104
.bss           11888       385024
Total         388688
```

`--help`
: Show a summary of acceptable arguments and options.

`-d`
`-o`
`-x`
`--radix=`*number*
: Using one of these options, you can control whether the size of each section is given in decimal ('`-d`', or '`--radix=10`'); octal ('`-o`', or '`--radix=8`'); or hexadecimal ('`-x`', or '`--radix=16`'). In '`--radix=`*number*', only the three values (8, 10, 16) are supported. The total size is always given in two radices; decimal

and hexadecimal for '`-d`' or '`-x`' output, or octal and hexadecimal if you're using '`-o`'.

`--target=`*`bfdname`*
: Specify that the object-code format for *objfile* is *bfdname*. This option may not be necessary; `size` can automatically recognize many formats. See Chapter 6 [objdump], page 13, for information on listing available formats.

`-V`
`--version`
: Display the version number of `size`.

9 strip

```
strip [ -F format | --format=format | --target=format ]
      [ -I format | --input-format=format ]
      [ -O format | --output-format=format ]
      [ -s | --strip-all ] [ -S | -g | --strip-debug ]
      [ -x | --discard-all ] [ -X | --discard-locals ]
      [ -v | --verbose ]  [ -V | --version ]
      objfile...
```

GNU `strip` discards all symbols from object files *objfile*. The list of object files may include archives.

`strip` will not execute unless at least one object file is listed.

`strip` modifies the files named in its argument, rather than writing modified copies under different names.

`-I format`
`--input-format=format`
: Treat the original *objfile* as a file with the object code format *format*.

`-O format`
`--output-format=format`
: Replace *objfile* with a file in the output format *format*.

`-F format`
`--format=format`
`--target=format`
: Treat the original *objfile* as a file with the object code format *format*, and rewrite it in the same format.

`-s`
`--strip-all`
: Remove all symbols.

`-g`
`-S`
`--strip-debug`
: Remove debugging symbols only.

`-x`
`--discard-all`
: Remove non-global symbols.

`-X`
`--discard-locals`
: Remove compiler-generated local symbols. (These usually start with ‘`L`’ or ‘`.`’.)

`-V`
`--version`
: Show the version number for `strip`.

`-v`
`--verbose`
: Verbose output: list all object files modified. In the case of archives, ‘`strip -v`’ lists all members of the archive.

10 c++filt

The C++ language provides function overloading, which means that you can write many function with the same name (but taking different kinds of parameters). So that the linker can keep these overloaded functions from clashing, all C++ function names are encoded ("mangled") into a funny-looking low-level assembly label. The `c++filt` program does the inverse mapping: It decodes ("demangles") low-level names into user-level names.

When you use `c++filt` as a filter (which is usually the case), it reads from standard input. Every alphanumeric word (consisting of letters, digits, underscores, dollars, or periods) seen in the input is a potential label. If the label decodes into a C++ name. the C++ name will replace the low-level name in the output.

A typical use of `c++filt` is to pipe the output of `nm` though it.

Note that on some systems, both the C and C++ compilers put an underscore in front of every name. (I.e. the C name `foo` gets the low-level name `_foo`.) On such systems, `c++filt` removes any initial underscore of a potential label.

Index

S

U

W

Table of Contents