Using binutils, the Gnu Binary Utilities
1. Introduction
2. ar
   2.1. Controlling ar on the Command Line
   2.2. Controlling ar with a Script
3. nm
4. objcopy
5. objdump
6. ranlib
7. size
8. strings
9. strip
10. c++filt
11. addr2line
12. nlmconv
13. windres
14. Create files needed to build and use DLLs
15. readelf
16. Selecting the Target System
   16.1. Target Selection
      16.1.1. objdumpTarget
      16.1.2. objcopy and stripInput Target
      16.1.3. objcopy and stripOutput Target
      16.1.4. nm, size, and stringsTarget
   16.2. Architecture Selection
      16.2.1. objdumpArchitecture
      16.2.2. objcopy, nm, size, stringsArchitecture
17. Reporting Bugs
   17.1. Have You Found a Bug?
   17.2. How to Report Bugs
A. GNU Free Documentation License
   A.1. ADDENDUM: How to use this License for your documents
Index
This brief manual contains documentation for the gnu binary utilities (collectively version 2.14.90.0.4):

This document is distributed under the terms of the GNU Free Documentation License. A copy of the license is included in the section entitled "GNU Free Documentation License".
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 [-X32_64] [-]p[mod [relops] [count]] archive [member...]
Delete modules from the archive. Specify the names of modules to be deleted as member...; 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.

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 member 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.

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 member arguments, all the files in the archive are printed.

Quick append; Historically, add the files member... 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.

However, too many different systems assume quick append rebuilds the index, so gnu ar implements q as a synonym for r.

Insert the files member... 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 member... does not 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.

Display a table listing the contents of archive, or those of the files listed in member... 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 member, 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 `member`) 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 `member`, 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 `relnames` 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 `relnames` argument, before the `archive` specification. (same as `i`).

`c`

Create the archive. The specified `archive` is always created if it did not 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.

`f`

Truncate names in the archive. gnu `ar` will normally permit file names of any length. This will cause it to create archives which are not compatible with the native `ar` program on some systems. If this is a concern, the `f` modifier may be used to truncate file names when putting them in the archive.

`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 `relnames` argument, before the `archive` specification. (same as `b`).

`l`

This modifier is accepted but not used.

`N`

Uses the `count` parameter. This is used if there are multiple entries in the archive with the same name. Extract or delete instance `count` of the given name from the archive.

`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.

`P`

Use the full path name when matching names in the archive. gnu `ar` can not create an archive with a full path name (such archives are not POSIX complaint), but other archive creators can.
This option will cause gnu ar to match file names using a complete path name, which can be convenient when extracting a single file from an archive created by another tool.

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 \texttt{ar s} on an archive is equivalent to running \texttt{ranlib} on it.

Do not generate an archive symbol table. This can speed up building a large library in several steps. The resulting archive can not be used with the linker. In order to build a symbol table, you must omit the \texttt{s} modifier on the last execution of \texttt{ar}, or you must run \texttt{ranlib} on the archive.

Normally, \texttt{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 \texttt{u} modifier is allowed only for the operation \texttt{r} (replace). In particular, the combination \texttt{qu} is not allowed, since checking the timestamps would lose any speed advantage from the operation \texttt{q}.

This modifier requests the \texttt{verbose} version of an operation. Many operations display additional information, such as filenames processed, when the modifier \texttt{v} is appended.

This modifier shows the version number of \texttt{ar}.

\texttt{ar} ignores an initial option spelt \texttt{–X32_64}, for compatibility with AIX. The behaviour produced by this option is the default for gnu \texttt{ar}. \texttt{ar} does not support any of the other \texttt{–X} options; in particular, it does not support \texttt{–X32} which is the default for AIX \texttt{ar}.

### 2.2. Controlling \texttt{ar} with a Script

\texttt{ar \textasciitilde M [ <script ]}

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

The \texttt{ar} command language is \textit{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 \texttt{ar} for developers who already have scripts written for the MRI "librarian" program.

The syntax for the \texttt{ar} command language is straightforward:

- Commands are recognized in upper or lower case; for example, \texttt{LIST} is the same as \texttt{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** member, member, ... member

Add each named member as a module in the current archive.

Requires prior use of OPEN or CREATE.

**CLEAR**

Discard the contents of the current archive, canceling 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.
Exit from \texttt{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 \texttt{SAVE} command, those changes are lost.

\textbf{EXTRACT module, module, ... module}

Extract each named module from the current archive, writing them into the current directory as separate files. Equivalent to \texttt{ar -x archive module...}.

Requires prior use of \texttt{OPEN} or \texttt{CREATE}.

\textbf{LIST}

Display full contents of the current archive, in "verbose" style regardless of the state of \texttt{VERBOSE}. The effect is like \texttt{ar tv archive}. (This single command is a gnu \texttt{ar} enhancement, rather than present for MRI compatibility.)

Requires prior use of \texttt{OPEN} or \texttt{CREATE}.

\textbf{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 \texttt{archive} until you next use \texttt{SAVE}.

\textbf{REPLACE module, module, ... module}

In the current archive, replace each existing module (named in the \texttt{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 \texttt{OPEN} or \texttt{CREATE}.

\textbf{VERBOSE}

Toggle an internal flag governing the output from \texttt{DIRECTORY}. When the flag is on, \texttt{DIRECTORY} output matches output from \texttt{ar -tv} ....

\textbf{SAVE}

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

Requires prior use of \texttt{OPEN} or \texttt{CREATE}. 

gnu nm lists the symbols from object files objfile... If no object files are listed as arguments, nm assumes the file a.out.

For each symbol, nm shows:

- The symbol value, in the radix selected by options (see below), or hexadecimal by default.
- The symbol type. At least the following types are used; others are, as well, depending on the object file format. If lowercase, the symbol is local; if uppercase, the symbol is global (external).

A

The symbol’s value is absolute, and will not be changed by further linking.

B

The symbol is in the uninitialized data section (known as BSS).

C

The symbol is common. Common symbols are uninitialized data. When linking, multiple common symbols may appear with the same name. If the symbol is defined anywhere, the common symbols are treated as undefined references. For more details on common symbols, see the discussion of -warn-common in .

D

The symbol is in the initialized data section.

G

The symbol is in an initialized data section for small objects. Some object file formats permit more efficient access to small data objects, such as a global int variable as opposed to a large global array.

I

The symbol is an indirect reference to another symbol. This is a gnu extension to the a.out object file format that is rarely used.
N
The symbol is a debugging symbol.

R
The symbol is in a read only data section.

S
The symbol is in an uninitialized data section for small objects.

T
The symbol is in the text (code) section.

U
The symbol is undefined.

V
The symbol is a weak object. When a weak defined symbol is linked with a normal defined
symbol, the normal defined symbol is used with no error. When a weak undefined symbol is
linked and the symbol is not defined, the value of the weak symbol becomes zero with no error.

W
The symbol is a weak symbol that has not been specifically tagged as a weak object symbol. When
a weak defined symbol is linked with a normal defined symbol, the normal defined symbol is used with no error. When a weak undefined symbol is linked and the symbol is not defined, the value of the weak symbol becomes zero with no error.

-
The symbol is a stabs symbol in an a.out object file. In this case, the next values printed are
the stabs other field, the stabs desc field, and the stab type. Stabs symbols are used to hold
debugging information. For more information, see .

?
The symbol type is unknown, or object file format specific.

- The symbol name.

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

-A
-pre-print-file-name
Precede each symbol by the name of the input file (or archive member) in which it was found,
rather than identifying the input file once only, before all of its symbols.

-a
debug-syms
Display all symbols, even debugger-only symbols; normally these are not listed.

-B
The same as -format=bsd (for compatibility with the MIPS nm).
-C
-demangle[=style]

Decode (demangle) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler. Refer to Chapter 10 c++filt for more information on demangling.

-no-demangle

Do not demangle low-level symbol names. This is the default.

-D
-dynamic

Display the dynamic symbols rather than the normal symbols. This is only meaningful for dynamic objects, such as certain types of shared libraries.

-f format
-format=format

Use the output format format, which can be bsd, sysv, or posix. The default is bsd. Only the first character of format is significant; it can be either upper or lower case.

-g
-extern-only

Display only external symbols.

-l
-line-numbers

For each symbol, use debugging information to try to find a filename and line number. For a defined symbol, look for the line number of the address of the symbol. For an undefined symbol, look for the line number of a relocation entry which refers to the symbol. If line number information can be found, print it after the other symbol information.

-n
-v
-numeric-sort

Sort symbols numerically by their addresses, rather than alphabetically by their names.

-p
-no-sort

Do not bother to sort the symbols in any order; print them in the order encountered.

-P
-portability

Use the POSIX.2 standard output format instead of the default format. Equivalent to -f posix.

-S
-print-size

Print size, not the value, of defined symbols for the bsd output format.
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.

Reverse the order of the sort (whether numeric or alphabetic); let the last come first.

Sort symbols by size. The size is computed as the difference between the value of the symbol and the value of the symbol with the next higher value. If the `bsd` output format is used the size of the symbol is printed, rather than the value, and `-S` must be used in order both size and value to be printed.

Use `radix` as the radix for printing the symbol values. It must be `d` for decimal, `o` for octal, or `x` for hexadecimal.

Specify an object code format other than your system's default format. Refer to Section 16.1 `Target Selection` for more information.

Display only undefined symbols (those external to each object file).

Display only defined symbols for each object file.

Show the version number of `nm` and exit.

This option is ignored for compatibility with the AIX version of `nm`. It takes one parameter which must be the string `32_64`. The default mode of AIX `nm` corresponds to `-X 32`, which is not supported by gnu `nm`.

Show a summary of the options to `nm` and exit.
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. Note that `objcopy` should be able to copy a fully linked file between any two formats. However, copying a relocatable object file between any two formats may not work as expected.
**objcopy** creates temporary files to do its translations and deletes them afterward. **objcopy** uses bfd to do all its translation work; it has access to all the formats described in bfd and thus is able to recognize most formats without being told explicitly.

**objcopy** can be used to generate S-records by using an output target of srec (e.g., use `-O srec`). **objcopy** can be used to generate a raw binary file by using an output target of binary (e.g., use `-O binary`). When **objcopy** generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file. All symbols and relocation information will be discarded. The memory dump will start at the load address of the lowest section copied into the output file.

When generating an S-record or a raw binary file, it may be helpful to use `-S` to remove sections containing debugging information. In some cases `-R` will be useful to remove sections which contain information that is not needed by the binary file.

Note—**objcopy** is not able to change the endianness of its input files. If the input format has an endianness (some formats do not), **objcopy** can only copy the inputs into file formats that have the same endianness or which have no endianness (e.g., srec).

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

`-I bfdname`  
`-input-target=bfdname`  
Consider the source file’s object format to be `bfdname`, rather than attempting to deduce it. Refer to Section 16.1 Target Selection for more information.

`-O bfdname`  
`-output-target=bfdname`  
Write the output file using the object format `bfdname`. Refer to Section 16.1 Target Selection for more information.

`-F bfdname`  
`-target=bfdname`  
Use `bfdname` as the object format for both the input and the output file; that is, simply transfer data from source to destination with no translation. Refer to Section 16.1 Target Selection for more information.

`-B bfdarch`  
`-binary-architecture=bfdarch`  
Useful when transforming a raw binary input file into an object file. In this case the output architecture can be set to `bfdarch`. This option will be ignored if the input file has a known `bfdarch`. You can access this binary data inside a program by referencing the special symbols that are created by the conversion process. These symbols are called `_binary_objfile_start`, `_binary_objfile_end` and `_binary_objfile_size`. e.g. you can transform a picture file into an object file and then access it in your code using these symbols.

`-j sectionname`  
`-only-section=sectionname`  
Copy only the named section from the input file to the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable.
-R sectionname
-remove-section=sectionname

Remove any section named sectionname from the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable.

-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.

-strip-unneeded

Strip all symbols that are not needed for relocation processing.

-K symbolname
-keep-symbol=symbolname

Copy only symbol symbolname from the source file. This option may be given more than once.

-N symbolname
-keep-symbol=symbolname

Do not copy symbol symbolname from the source file. This option may be given more than once.

-G symbolname
-keep-global-symbol=symbolname

Keep only symbol symbolname global. Make all other symbols local to the file, so that they are not visible externally. This option may be given more than once.

-L symbolname
-locate-symbol=symbolname

Make symbol symbolname local to the file, so that it is not visible externally. This option may be given more than once.

-W symbolname
-weaken-symbol=symbolname

Make symbol symbolname weak. This option may be given more than once.

-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 ..)

-b byte
-byte=byte

Keep only every byte\text{th} byte of the input file (header data is not affected). byte can be in the range from 0 to interleave-1, where interleave is given by the \textit{-i or -interleave} option,
or the default of 4. This option is useful for creating files to program rom. It is typically used with
an srec output target.

-i interleave
-interleave=interleave

Only copy one out of every interleave bytes. Select which byte to copy with the -b or -byte
option. The default is 4. objcopy ignores this option if you do not specify either -b or -byte.

-p
-preserve-dates

Set the access and modification dates of the output file to be the same as those of the input file.

-debugging

Convert debugging information, if possible. This is not the default because only certain debug-
ing formats are supported, and the conversion process can be time consuming.

-gap-fill val

Fill gaps between sections with val. This operation applies to the load address (LMA) of the
sections. It is done by increasing the size of the section with the lower address, and filling in the
extra space created with val.

-pad-to address

Pad the output file up to the load address address. This is done by increasing the size of the last
section. The extra space is filled in with the value specified by -gap-fill (default zero).

-set-start val

Set the start address of the new file to val. Not all object file formats support setting the start
address.

-change-start incr
-adjust-start incr

Change the start address by adding incr. Not all object file formats support setting the start
address.

-change-addresses incr
-adjust-vma incr

Change the VMA and LMA addresses of all sections, as well as the start address, by adding
incr. Some object file formats do not permit section addresses to be changed arbitrarily. Note
that this does not relocate the sections; if the program expects sections to be loaded at a certain
address, and this option is used to change the sections such that they are loaded at a different
address, the program may fail.

-change-section-address section{=,+,-}val
-adjust-section-vma section{=,+,-}val

Set or change both the VMA address and the LMA address of the named section. If = is used,
the section address is set to val. Otherwise, val is added to or subtracted from the section
address. See the comments under -change-addresses, above. If section does not exist in
the input file, a warning will be issued, unless -no-change-warnings is used.

-change-section-lma section{=,+,-}val

Set or change the LMA address of the named section. The LMA address is the address where
the section will be loaded into memory at program load time. Normally this is the same as the
VMA address, which is the address of the section at program run time, but on some systems, especially those where a program is held in ROM, the two can be different. If \( = \) is used, the section address is set to \( \text{val} \). Otherwise, \( \text{val} \) is added to or subtracted from the section address. See the comments under \(-\text{change-addresses}\), above. If \section{} does not exist in the input file, a warning will be issued, unless \(-\text{no-change-warnings}\) is used.

\[-\text{change-section-vma} \section{}\{=,+,-\}\text{val}\]

Set or change the VMA address of the named \section{}. The VMA address is the address where the section will be located once the program has started executing. Normally this is the same as the LMA address, which is the address where the section will be loaded into memory, but on some systems, especially those where a program is held in ROM, the two can be different. If \( = \) is used, the section address is set to \( \text{val} \). Otherwise, \( \text{val} \) is added to or subtracted from the section address. See the comments under \(-\text{change-addresses}\), above. If \section{} does not exist in the input file, a warning will be issued, unless \(-\text{no-change-warnings}\) is used.

\[-\text{change-warnings}\]
\[-\text{adjust-warnings}\]

If \(-\text{change-section-address}\) or \(-\text{change-section-lma}\) or \(-\text{change-section-vma}\) is used, and the named section does not exist, issue a warning. This is the default.

\[-\text{no-change-warnings}\]
\[-\text{no-adjust-warnings}\]

Do not issue a warning if \(-\text{change-section-address}\) or \(-\text{adjust-section-lma}\) or \(-\text{adjust-section-vma}\) is used, even if the named section does not exist.

\[-\text{set-section-flags} \section{}\text{flags}\]

Set the flags for the named section. The \text{flags} argument is a comma separated string of flag names. The recognized names are \text{alloc, contents, load, noload, readonly, code, data, rom, share, and debug}. You can set the \text{contents} flag for a section which does not have contents, but it is not meaningful to clear the \text{contents} flag of a section which does have contents—just remove the section instead. Not all flags are meaningful for all object file formats.

\[-\text{add-section} \section{}\text{filename}\]

Add a new section named \section{} while copying the file. The contents of the new section are taken from the file \text{filename}. The size of the section will be the size of the file. This option only works on file formats which can support sections with arbitrary names.

\[-\text{rename-section} \text{oldname}\text{=newname[,\text{flags}]}\]

Rename a section from \text{oldname} to \text{newname}, optionally changing the section’s \text{flags} to \text{flags} in the process. This has the advantage over using a linker script to perform the rename in that the output stays as an object file and does not become a linked executable.

This option is particularly helpful when the input format is binary, since this will always create a section called .data. If for example, you wanted instead to create a section called .rodata containing binary data you could use the following command line to achieve it:

\[
\text{objcopy -I binary -O} \langle\text{output_format}\rangle \text{-B} \langle\text{architecture}\rangle \text{--rename-section .data=.rodata,alloc,load,readonly,code,contents} \text{input_binary_file} \langle\text{output_object_file}\rangle
\]

\[-\text{change-leading-char}\]

Some object file formats use special characters at the start of symbols. The most common such character is underscore, which compilers often add before every symbol. This option tells \text{objcopy} to change the leading character of every symbol when it converts between object file
formats. If the object file formats use the same leading character, this option has no effect. Otherwise, it will add a character, or remove a character, or change a character, as appropriate.

-remove-leading-char
If the first character of a global symbol is a special symbol leading character used by the object file format, remove the character. The most common symbol leading character is underscore. This option will remove a leading underscore from all global symbols. This can be useful if you want to link together objects of different file formats with different conventions for symbol names. This is different from -change-leading-char because it always changes the symbol name when appropriate, regardless of the object file format of the output file.

-srec-len=ival
Meaningful only for srec output. Set the maximum length of the Srecords being produced to ival. This length covers both address, data and crc fields.

-srec-forceS3
Meaningful only for srec output. Avoid generation of S1/S2 records, creating S3-only record format.

-redefine-sym old=new
Change the name of a symbol old, to new. This can be useful when one is trying link two things together for which you have no source, and there are name collisions.

-weaken
Change all global symbols in the file to be weak. This can be useful when building an object which will be linked against other objects using the -R option to the linker. This option is only effective when using an object file format which supports weak symbols.

-keep-symbols=filename
Apply -keep-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

-strip-symbols=filename
Apply -strip-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

-keep-global-symbols=filename
Apply -keep-global-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

-localize-symbols=filename
Apply -localize-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.

-weaken-symbols=filename
Apply -weaken-symbol option to each symbol listed in the file filename. filename is simply a flat file, with one symbol name per line. Line comments may be introduced by the hash character. This option may be given more than once.
Chapter 4. objcopy

-\texttt{-alt-machine-code=index}

If the output architecture has alternate machine codes, use the \texttt{index}th code instead of the default one. This is useful in case a machine is assigned an official code and the tool-chain adopts the new code, but other applications still depend on the original code being used.

-\texttt{-prefix-symbols=string}

Prefix all symbols in the output file with \texttt{string}.

-\texttt{-prefix-sections=string}

Prefix all section names in the output file with \texttt{string}.

-\texttt{-prefix-alloc-sections=string}

Prefix all the names of all allocated sections in the output file with \texttt{string}.

-\texttt{-V}
-\texttt{-version}

Show the version number of \texttt{objcopy}.

-\texttt{-v}
-\texttt{-verbose}

Verbose output: list all object files modified. In the case of archives, \texttt{objcopy -V} lists all members of the archive.

-\texttt{-help}

Show a summary of the options to \texttt{objcopy}.

-\texttt{-info}

Display a list showing all architectures and object formats available.
Chapter 5.

**objdump**

`objdump [-a|--archive-headers]
    [-b bfdname|--target=bfdname]
    [-C|--demangle[=style] ]
    [-d|--disassemble]
    [-D|--disassemble-all]
    [-z|--disassemble-zeroes]
    [-EB|EL|--endian={big | little }]
    [-f|--file-headers]
    [--file-start-context]
    [-g|--debugging]
    [-h|--section-headers|--headers]
    [-i|--info]
    [-j section|--section=section]
    [-l|--line-numbers]
    [-S|--source]
    [-m machine|--architecture=machine]
    [-M options|--disassembler-options=options]
    [-p|--private-headers]
    [-r|--reloc]
    [-R|--dynamic-reloc]
    [-s|--full-contents]
    [-G|--stabs]
    [-t|--syma]
    [-T|--dynamic-syms]
    [-x|--all-headers]
    [-w|--wide]
    [--start-address=address]
    [--stop-address=address]
    [--prefix-addresses]
    [--[no-]show-raw-insn]
    [--adjust-vma=offset]
    [-V|--version]
    [-H|--help]
    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.

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

The long and short forms of options, shown here as alternatives, are equivalent. At least one option from the list `-a, -d, -D, -f, -g, -G, -h, -H, -p, -r, -R, -S, -t, -T, -V, -x` must be given.

- `-a`, `--archive-header`  
  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.
-adjust-vma=offset

When dumping information, first add offset to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

-b bfdname
-tARGET=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. Refer to Section 16.1 Target Selection for more information.

-C
-demangle[=style]

Decode (demangle) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler. Refer to Chapter 10 c++filt for more information on demangling.

-g
-debugging

Display debugging information. This attempts to parse debugging information stored in the file and print it out using a C like syntax. Only certain types of debugging information have been implemented. Some other types are supported by readelf -w. Chapter 15 readelf.

-d
-disassemble

Display the assembler mnemonics for the machine instructions from objfile. This option only disassembles those sections which are expected to contain instructions.

-D
-disassemble-all

Like -d, but disassemble the contents of all sections, not just those expected to contain instructions.

-prefix-addresses

When disassembling, print the complete address on each line. This is the older disassembly format.

-EB
-EL
-endian={big|little}

Specify the endianness of the object files. This only affects disassembly. This can be useful when disassembling a file format which does not describe endianness information, such as S-records.
-f
-file-headers
Display summary information from the overall header of each of the objfile files.

-file-start-context
Specify that when displaying interleaved source code/disassembly (assumes -S) from a file that has not yet been displayed, extend the context to the start of the file.

-h
-section-headers
Headers
Display summary information from the section headers of the object file.
File segments may be relocated to nonstandard addresses, for example by using the -Ttext, -Tdata, or -Tbss options to ld. However, some object file formats, such as a.out, do not store the starting address of the file segments. In those situations, although ld relocates the sections correctly, using objdump -h to list the file section headers cannot show the correct addresses. Instead, it shows the usual addresses, which are implicit for the target.

-H
-help
Print a summary of the options to objdump and exit.

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

-j name
-section=name
Display information only for section name.

-l
-line-numbers
Label the display (using debugging information) with the filename and source line numbers corresponding to the object code or relocs shown. Only useful with -d, -D, or -r.

-m machine
-architecture=machine
Specify the architecture to use when disassembling object files. This can be useful when disassembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the -i option.

-M options
-disassembler-options=options
Pass target specific information to the disassembler. Only supported on some targets.
If the target is an ARM architecture then this switch can be used to select which register name set is used during disassembler. Specifying -M reg-name-std (the default) will select the register names as used in ARM's instruction set documentation, but with register 13 called 'sp', register 14 called 'lr' and register 15 called 'pc'. Specifying -M reg-names-apcs will select the name set used by the ARM Procedure Call Standard, whilst specifying -M reg-names-raw will just use r followed by the register number.
There are also two variants on the APCS register naming scheme enabled by `-M reg-names-atpcs` and `-M reg-names-special-atpcs` which use the ARM/Thumb Procedure Call Standard naming conventions. (Either with the normal register names or the special register names).

This option can also be used for ARM architectures to force the disassembler to interpret all instructions as Thumb instructions by using the switch `disassembler-options=force-thumb`. This can be useful when attempting to disassemble thumb code produced by other compilers.

For the x86, some of the options duplicate functions of the `-m` switch, but allow finer grained control. Multiple selections from the following may be specified as a comma separated string. `x86-64`, `i386` and `i8086` select disassembly for the given architecture. `intel` and `att` select betweenintel syntax mode and AT&T syntax mode. `addr32`, `addr16`, `data32` and `data16` specify the default address size and operand size. These four options will be overridden if `x86-64`, `i386` or `i8086` appear later in the option string. Lastly, `suffix`, when in AT&T mode, instructs the disassembler to print a mnemonic suffix even when the suffix could be inferred by the operands.

For PPC, `booke`, `booke32` and `booke64` select disassembly of BookE instructions. `32` and `64` select PowerPC and PowerPC64 disassembly, respectively.

For MIPS, this option controls the printing of register names in disassembled instructions. Multiple selections from the following may be specified as a comma separated string, and invalid options are ignored:

```plaintext
gpr-names=ABI
  Print GPR (general-purpose register) names as appropriate for the specified ABI. By default, GPR names are selected according to the ABI of the binary being disassembled.

fpr-names=ABI
  Print FPR (floating-point register) names as appropriate for the specified ABI. By default, FPR numbers are printed rather than names.

cp0-names=ARCH
  Print CP0 (system control coprocessor; coprocessor 0) register names as appropriate for the CPU or architecture specified by ARCH. By default, CP0 register names are selected according to the architecture and CPU of the binary being disassembled.

hwr-names=ARCH
  Print HWR (hardware register, used by the rdhwr instruction) names as appropriate for the CPU or architecture specified by ARCH. By default, HWR names are selected according to the architecture and CPU of the binary being disassembled.

reg-names=ABI
  Print GPR and FPR names as appropriate for the selected ABI.

reg-names=ARCH
  Print CPU-specific register names (CP0 register and HWR names) as appropriate for the selected CPU or architecture.
```

For any of the options listed above, `ABI` or `ARCH` may be specified as numeric to have numbers printed rather than names, for the selected types of registers. You can list the available values of ABI and ARCH using the `-help` option.
-P
-privates-headers

  Print information that is specific to the object file format. The exact information printed depends
  upon the object file format. For some object file formats, no additional information is printed.

-r
-reloc

  Print the relocation entries of the file. If used with -d or -D, the relocations are printed inter-
  spersed with the disassembly.

-R
-dynamic-reloc

  Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such
  as certain types of shared libraries.

-s
-full-contents

  Display the full contents of any sections requested.

-S
-source

  Display source code intermixed with disassembly, if possible. Implies -d.

-show-raw-insn

  When disassembling instructions, print the instruction in hex as well as in symbolic form. This
  is the default except when -prefix-addresses is used.

-no-show-raw-insn

  When disassembling instructions, do not print the instruction bytes. This is the default when
  -prefix-addresses is used.

-G
-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. For more information on stabs symbols, see .

-start-address=address

  Start displaying data at the specified address. This affects the output of the -d, -r and -s options.

-stop-address=address

  Stop displaying data at the specified address. This affects the output of the -d, -r and -s options.

-t
-syms

  Print the symbol table entries of the file. This is similar to the information provided by the nm
  program.
-T
-dynamic-syms
   Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the nm program when given the -D (-dynamic) option.

-V
-version
   Print the version number of objdump and exit.

-x
-all-headers
   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.

-w
-wide
   Format some lines for output devices that have more than 80 columns. Also do not truncate
   symbol names when they are displayed.

-z
-disassemble-zeroes
   Normally the disassembly output will skip blocks of zeroes. This option directs the disassembler
to disassemble those blocks, just like any other data.
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`. Chapter 2 `ar`.

`-v`
`-V`  
`-version`

Show the version number of `ranlib`. 

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.

objfile... are the object files to be examined. If none are specified, the file a.out will be used.

The command line options have the following meanings:

-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.
Using one of these options, you can control whether the size of each section is given in decimal (\(-d\), or \(-\text{radix}=10\)); octal (\(-o\), or \(-\text{radix}=8\)); or hexadecimal (\(-x\), or \(-\text{radix}=16\)). In \(-\text{radix}=\text{number}\), 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 \(-o\).

\(-t\)

\(-\text{totals}\)

Show totals of all objects listed (Berkeley format listing mode only).

\(-\text{target}=\text{bfdname}\)

Specify that the object-code format for \text{objfile} is \text{bfdname}. This option may not be necessary; \text{size} can automatically recognize many formats. Refer to Section 16.1 \textit{Target Selection} for more information.

\(-V\)

\(-\text{version}\)

Display the version number of \text{size}.
Chapter 8.  
strings

strings [-afov] [-min-len]  
[-n min-len] [--bytes=min-len]  
[-t radix] [--radix=radix]  
[-e encoding] [--encoding=encoding]  
[-] [--all] [--print-file-name]  
[--target=bfdname]  
[--help] [--version] file...

For each file given, gnu strings prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non-text files.

-a
-all

Do not scan only the initialized and loaded sections of object files; scan the whole files.

-f
-print-file-name

Print the name of the file before each string.

-help

Print a summary of the program usage on the standard output and exit.

-min-len
-n min-len
--bytes=min-len

Print sequences of characters that are at least min-len characters long, instead of the default 4.

-o

Like -t o. Some other versions of strings have -o act like -t d instead. Since we can not be compatible with both ways, we simply chose one.

-t radix
--radix=radix

Print the offset within the file before each string. The single character argument specifies the radix of the offset--o for octal, x for hexadecimal, or d for decimal.
-e encoding
-encoding=encoding

Select the character encoding of the strings that are to be found. Possible values for encoding are: s = single-7-bit-byte characters (ASCII, ISO 8859, etc., default), S = single-8-bit-byte characters, b = 16-bit big endian, l = 16-bit little endian, B = 32-bit big endian, L = 32-bit little endian. Useful for finding wide character strings.

-target=bfdname

Specify an object code format other than your system’s default format. Refer to Section 16.1 Target Selection for more information.

-v
-version

Print the program version number on the standard output and exit.
gnu strip discards all symbols from object files objfile. The list of object files may include archives. At least one object file must be given.

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

-F bfdname
-target=bfdname

Treat the original objfile as a file with the object code format bfdname, and rewrite it in the same format. Refer to Section 16.1 Target Selection for more information.

-help
Show a summary of the options to strip and exit.

-info
Display a list showing all architectures and object formats available.

-I bfdname
-input-target=bfdname

Treat the original objfile as a file with the object code format bfdname. Refer to Section 16.1 Target Selection for more information.

-O bfdname
-output-target=bfdname

Replace objfile with a file in the output format bfdname. Refer to Section 16.1 Target Selection for more information.

-R sectionname
-remove-section=sectionname

Remove any section named sectionname from the output file. This option can be given more than once. Note that using this option inappropriately may make the output file unusable.
-s
-strip-all
  Remove all symbols.

-g
-S
-d
-strip-debug
  Remove debugging symbols only.

-strip-unneeded
  Remove all symbols that are not needed for relocation processing.

-K symbolname
-keep-symbol=symbolname
  Keep only symbol symbolname from the source file. This option may be given more than once.

-N symbolname
-strip-symbol=symbolname
  Remove symbol symbolname from the source file. This option may be given more than once, and may be combined with strip options other than -K.

-o file
  Put the stripped output in file, rather than replacing the existing file. When this argument is used, only one objfile argument may be specified.

-p
-preserve-dates
  Preserve the access and modification dates of the file.

-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.
The C++ and Java languages provide function overloading, which means that you can write many functions with the same name (providing each takes parameters of different types). All C++ and Java function names are encoded into a low-level assembly label (this process is known as mangling). The `c++filt` program does the inverse mapping: it decodes (demangles) low-level names into user-level names so that the linker can keep these overloaded functions from clashing.

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 replaces the low-level name in the output.

You can use `c++filt` to decipher individual symbols:

```
c++filt symbol
```

If no `symbol` arguments are given, `c++filt` reads symbol names from the standard input and writes the demangled names to the standard output. All results are printed on the standard output.

```
-_- strip-underscores
    On some systems, both the C and C++ compilers put an underscore in front of every name. For example, the C name `foo` gets the low-level name `_foo`. This option removes the initial underscore. Whether `c++filt` removes the underscore by default is target dependent.

-j java
    Prints demangled names using Java syntax. The default is to use C++ syntax.

-n no-strip-underscores
    Do not remove the initial underscore.

-s format
    format=format
    `c++filt` can decode various methods of mangling, used by different compilers. The argument to this option selects which method it uses:

---

1. MS-DOS does not allow + characters in file names, so on MS-DOS this program is named `cxxfilt`. 

---
auto

Automatic selection based on executable (the default method)

gnu

the one used by the gnu C++ compiler (g++)

lucid

the one used by the Lucid compiler (lcc)

arm

the one specified by the C++ Annotated Reference Manual

hp

the one used by the HP compiler (aCC)

edg

the one used by the EDG compiler

gnu-v3

the one used by the gnu C++ compiler (g++) with the V3 ABI.

java

the one used by the gnu Java compiler (gcj)

gnat

the one used by the gnu Ada compiler (GNAT).

-help

Print a summary of the options to c++filt and exit.

-version

Print the version number of c++filt and exit.

Warning: c++filt is a new utility, and the details of its user interface are subject to change in future releases. In particular, a command-line option may be required in the the future to decode a name passed as an argument on the command line; in other words,

```
c++filt symbol
```

may in a future release become

```
c++filt option symbol
```
addr2line [-b bfdname|--target=bfdname]  
[-C|--demangle[=style]]  
[-e filename|--exe=filename]  
[-f|--functions] [-s|--basename]  
[-H|--help] [-V|--version]  
[addr addr ...]

addr2line translates program addresses into file names and line numbers. Given an address and an executable, it uses the debugging information in the executable to figure out which file name and line number are associated with a given address.

The executable to use is specified with the -e option. The default is the file a.out.

addr2line has two modes of operation.

In the first, hexadecimal addresses are specified on the command line, and addr2line displays the file name and line number for each address.

In the second, addr2line reads hexadecimal addresses from standard input, and prints the file name and line number for each address on standard output. In this mode, addr2line may be used in a pipe to convert dynamically chosen addresses.

The format of the output is FILENAME:LINENO. The file name and line number for each address is printed on a separate line. If the -f option is used, then each FILENAME:LINENO line is preceded by a FUNCTIONNAME line which is the name of the function containing the address.

If the file name or function name can not be determined, addr2line will print two question marks in their place. If the line number can not be determined, addr2line will print 0.

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

-b bfdname
-target=bfdname

Specify that the object-code format for the object files is bfdname.

-C
-demangle[=style]

Decode (demangle) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable. Different compilers have different mangling styles. The optional demangling style argument can be used to choose an appropriate demangling style for your compiler. Refer to Chapter 10 c++filt for more information on demangling.

-e filename
-exe=filename

Specify the name of the executable for which addresses should be translated. The default file is a.out.
-f
-functions

Display function names as well as file and line number information.

-s
-basenames

Display only the base of each file name.
Chapter 12.

nlmconv

nlmconv converts a relocatable object file into a NetWare Loadable Module.

Warning: nlmconv is not always built as part of the binary utilities, since it is only useful for NLM targets.

```
nlmconv [-I bfdname|--input-target=bfdname]
   [-O bfdname|--output-target=bfdname]
   [-T headerfile|--header-file=headerfile]
   [-d|--debug] [-l linker|--linker=linker]
   [-h|--help] [-V|--version]
   infile outfile
```

nlmconv converts the relocatable i386 object file infile into the NetWare Loadable Module outfile, optionally reading headerfile for NLM header information. For instructions on writing the NLM command file language used in header files, see the linkers section, NLMLINK in particular, of the [NLM Development and Tools Overview], which is part of the NLM Software Developer’s Kit (“NLM SDK”), available from Novell, Inc. nlmconv uses the gnu Binary File Descriptor library to read infile; see, for more information.

nlmconv can perform a link step. In other words, you can list more than one object file for input if you list them in the definitions file (rather than simply specifying one input file on the command line). In this case, nlmconv calls the linker for you.

- I bfdname
   -input-target=bfdname

   Object format of the input file. nlmconv can usually determine the format of a given file (so no default is necessary). Refer to Section 16.1 Target Selection for more information.

- O bfdname
   -output-target=bfdname

   Object format of the output file. nlmconv infers the output format based on the input format, e.g. for a i386 input file the output format is nlm32-i386. Refer to Section 16.1 Target Selection for more information.

- T headerfile
   -header-file=headerfile

   Reads headerfile for NLM header information. For instructions on writing the NLM command file language used in header files, see see the linkers section, of the [NLM Development and Tools Overview], which is part of the NLM Software Developer’s Kit, available from Novell, Inc.

- d
   -debug

   Displays (on standard error) the linker command line used by nlmconv.
-l linker
-l linker=linker

Use linker for any linking. linker can be an absolute or a relative pathname.

-h
-help

Prints a usage summary.

-V
-Version

Prints the version number for nlmconv.
windres may be used to manipulate Windows resources.

*Warning*: windres is not always built as part of the binary utilities, since it is only useful for Windows targets.

windres [options] [input-file] [output-file]

windres reads resources from an input file and copies them into an output file. Either file may be in one of three formats:

**rc**
A text format read by the Resource Compiler.

**res**
A binary format generated by the Resource Compiler.

**coff**
A COFF object or executable.

The exact description of these different formats is available in documentation from Microsoft.

When `windres` converts from the `rc` format to the `res` format, it is acting like the Windows Resource Compiler. When `windres` converts from the `res` format to the `coff` format, it is acting like the Windows CVTRES program.

When `windres` generates an `rc` file, the output is similar but not identical to the format expected for the input. When an input `rc` file refers to an external filename, an output `rc` file will instead include the file contents.

If the input or output format is not specified, `windres` will guess based on the file name, or, for the input file, the file contents. A file with an extension of `.rc` will be treated as an `rc` file, a file with an extension of `.res` will be treated as a `res` file, and a file with an extension of `.o` or `.exe` will be treated as a `coff` file.

If no output file is specified, `windres` will print the resources in `rc` format to standard output.

The normal use is for you to write an `rc` file, use `windres` to convert it to a COFF object file, and then link the COFF file into your application. This will make the resources described in the `rc` file available to Windows.

```
-i filename
-input filename
```

The name of the input file. If this option is not used, then `windres` will use the first non-option argument as the input file name. If there are no non-option arguments, then `windres` will read from standard input. `windres` can not read a COFF file from standard input.
-o filename
-output filename

The name of the output file. If this option is not used, then windres will use the first non-option argument, after any used for the input file name, as the output file name. If there is no non-option argument, then windres will write to standard output. windres can not write a COFF file to standard output. Note, for compatibility with rc the option -fo is also accepted, but its use is not recommended.

-J format
-input-format format

The input format to read. format may be res, rc, or coff. If no input format is specified, windres will guess, as described above.

-O format
-output-format format

The output format to generate. format may be res, rc, or coff. If no output format is specified, windres will guess, as described above.

-F target
-target target

Specify the BFD format to use for a COFF file as input or output. This is a BFD target name; you can use the -help option to see a list of supported targets. Normally windres will use the default format, which is the first one listed by the -help option. Section 16.1 Target Selection.

-preprocessor program

When windres reads an rc file, it runs it through the C preprocessor first. This option may be used to specify the preprocessor to use, including any leading arguments. The default preprocessor argument is gcc -E -xc-header -DRC_INVOKED.

-I directory
-include-dir directory

Specify an include directory to use when reading an rc file. windres will pass this to the preprocessor as an -I option. windres will also search this directory when looking for files named in the rc file. If the argument passed to this command matches any of the supported formats (as described in the -J option), it will issue a deprecation warning, and behave just like the -J option. New programs should not use this behaviour. If a directory happens to match a format, simple prefix it with ./ to disable the backward compatibility.

-D target
-define sym[=val]

Specify a -D option to pass to the preprocessor when reading an rc file.

-U target
-undefine sym

Specify a -U option to pass to the preprocessor when reading an rc file.

-r

Ignored for compatibility with rc.

-v

Enable verbose mode. This tells you what the preprocessor is if you didn’t specify one.
-l val
-language val

Specify the default language to use when reading an rc file. val should be a hexadecimal language code. The low eight bits are the language, and the high eight bits are the sublanguage.

-use-temp-file

Use a temporary file to instead of using popen to read the output of the preprocessor. Use this option if the popen implementation is buggy on the host (e.g., certain non-English language versions of Windows 95 and Windows 98 are known to have buggy popen where the output will instead go the console).

-no-use-temp-file

Use popen, not a temporary file, to read the output of the preprocessor. This is the default behaviour.

-h
-help

Prints a usage summary.

-v
-version

Prints the version number for windres.

-yydebug

If windres is compiled with YYDEBUG defined as 1, this will turn on parser debugging.
dlltool may be used to create the files needed to build and use dynamic link libraries (DLLs).

*Warning:* dlltool is not always built as part of the binary utilities, since it is only useful for those targets which support DLLs.

dlltool [-d|--input-def def-file-name]
[-b|--base-file base-file-name]
[-e|--output-exp exports-file-name]
[-l|--output-lib library-file-name]
[--export-all-symbols] [--no-export-all-symbols]
[--exclude-symbols list]
[--no-default-excludes]
[-S|--as path-to-assembler] [-f|--as-flags options]
[-D|--dllname name] [-m|--machine machine]
[-a|--add-indirect] [-U|--add-underscore] [-k|--kill-at]
[-A|--add-stdcall-alias]
[-x|--no-idata4] [-c|--no-idata5] [-i|--interwork]
[-n|--nodelete] [-v|--verbose]
[-h|--help] [-V|--version]
[object-file ...]

dlltool reads its inputs, which can come from the -d and -b options as well as object files specified on the command line. It then processes these inputs and if the -e option has been specified it creates a exports file. If the -l option has been specified it creates a library file and if the -z option has been specified it creates a def file. Any or all of the -e, -l and -z options can be present in one invocation of dlltool.

When creating a DLL, along with the source for the DLL, it is necessary to have three other files. dlltool can help with the creation of these files.

The first file is a .def file which specifies which functions are exported from the DLL, which functions the DLL imports, and so on. This is a text file and can be created by hand, or dlltool can be used to create it using the -z option. In this case dlltool will scan the object files specified on its command line looking for those functions which have been specially marked as being exported and put entries for them in the .def file it creates.

In order to mark a function as being exported from a DLL, it needs to have an -export:<name_of_function> entry in the .dectve section of the object file. This can be done in C by using the asm() operator:

```c
asm (".section .drectve");
asm (".ascii ":export:my_func"");

int my_func (void) { ... }
```

The second file needed for DLL creation is an exports file. This file is linked with the object files that make up the body of the DLL and it handles the interface between the DLL and the outside world.
This is a binary file and it can be created by giving the \texttt{-e} option to \texttt{dlltool} when it is creating or reading in a .def file.

The third file needed for DLL creation is the library file that programs will link with in order to access the functions in the DLL. This file can be created by giving the \texttt{-l} option to \texttt{dlltool} when it is creating or reading in a .def file.

\texttt{dlltool} builds the library file by hand, but it builds the exports file by creating temporary files containing assembler statements and then assembling these. The \texttt{-S} command line option can be used to specify the path to the assembler that dlltool will use, and the \texttt{-f} option can be used to pass specific flags to that assembler. The \texttt{-n} can be used to prevent dlltool from deleting these temporary assembler files when it is done, and if \texttt{-n} is specified twice then this will prevent dlltool from deleting the temporary object files it used to build the library.

Here is an example of creating a DLL from a source file \texttt{dll.c} and also creating a program (from an object file called \texttt{program.o}) that uses that DLL:

\begin{verbatim}
gcc -c dll.c
dlltool -e exports.o -l dll.lib dll.o
gcc dll.o exports.o -o dll.dll
gcc program.o dll.lib -o program
\end{verbatim}

The command line options have the following meanings:

- \texttt{-d filename}
  -input-def filename

  Specifies the name of a .def file to be read in and processed.

- \texttt{-b filename}
  -base-file filename

  Specifies the name of a base file to be read in and processed. The contents of this file will be added to the relocation section in the exports file generated by dlltool.

- \texttt{-e filename}
  -output-exp filename

  Specifies the name of the export file to be created by dlltool.

- \texttt{-z filename}
  -output-def filename

  Specifies the name of the .def file to be created by dlltool.

- \texttt{-l filename}
  -output-lib filename

  Specifies the name of the library file to be created by dlltool.

- \texttt{-export-all-symbols}

  Treat all global and weak defined symbols found in the input object files as symbols to be exported. There is a small list of symbols which are not exported by default; see the \texttt{-no-default-excludes} option. You may add to the list of symbols to not export by using the \texttt{-exclude-symbols} option.
-no-export-all-symbols

Only export symbols explicitly listed in an input .def file or in .drectve sections in the input object files. This is the default behaviour. The .drectve sections are created by dllexport attributes in the source code.

-exclude-symbols list

Do not export the symbols in list. This is a list of symbol names separated by comma or colon characters. The symbol names should not contain a leading underscore. This is only meaningful when -export-all-symbols is used.

-no-default-excludes

When -export-all-symbols is used, it will by default avoid exporting certain special symbols. The current list of symbols to avoid exporting is DllMain@12, DllEntryPoint@0, impure_ptr. You may use the -no-default-excludes option to go ahead and export these special symbols. This is only meaningful when -export-all-symbols is used.

-S path
-as path

Specifies the path, including the filename, of the assembler to be used to create the exports file.

-f options
-as-flags options

Specifies any specific command line options to be passed to the assembler when building the exports file. This option will work even if the -S option is not used. This option only takes one argument, and if it occurs more than once on the command line, then later occurrences will override earlier occurrences. So if it is necessary to pass multiple options to the assembler they should be enclosed in double quotes.

-D name
-dll-name name

Specifies the name to be stored in the .def file as the name of the DLL when the -e option is used. If this option is not present, then the filename given to the -e option will be used as the name of the DLL.

-m machine
-machine machine

Specifies the type of machine for which the library file should be built. dlltool has a built in default type, depending upon how it was created, but this option can be used to override that. This is normally only useful when creating DLLs for an ARM processor, when the contents of the DLL are actually encode using Thumb instructions.

-a
-add-indirect

Specifies that when dlltool is creating the exports file it should add a section which allows the exported functions to be referenced without using the import library. Whatever the hell that means!

-U
-add-underscore

Specifies that when dlltool is creating the exports file it should prepend an underscore to the names of the exported functions.
-k
-kill-at

Specifies that when \texttt{dlltool} is creating the exports file it should not append the string @ \texttt{<number>}. These numbers are called ordinal numbers and they represent another way of accessing the function in a DLL, other than by name.

-A
-add-stdcall-alias

Specifies that when \texttt{dlltool} is creating the exports file it should add aliases for stdcall symbols without @ \texttt{<number>} in addition to the symbols with @ \texttt{<number>}.

-x
-no-idata4

Specifies that when \texttt{dlltool} is creating the exports and library files it should omit the .idata4 section. This is for compatibility with certain operating systems.

-c
-no-idata5

Specifies that when \texttt{dlltool} is creating the exports and library files it should omit the .idata5 section. This is for compatibility with certain operating systems.

-i
-interwork

Specifies that \texttt{dlltool} should mark the objects in the library file and exports file that it produces as supporting interworking between ARM and Thumb code.

-n
-nodelete

Makes \texttt{dlltool} preserve the temporary assembler files it used to create the exports file. If this option is repeated then \texttt{dlltool} will also preserve the temporary object files it uses to create the library file.

-v
-verbose

Make \texttt{dlltool} describe what it is doing.

-h
-help

Displays a list of command line options and then exits.

-V
-version

Displays \texttt{dlltool}'s version number and then exits.
readelf [-a|--all]
   [-h|--file-header]
   [-l|--program-headers|--segments]
   [-S|--section-headers|--sections]
   [-e|--headers]
   [-s|--syms|--symbols]
   [-n|--notes]
   [-r|--relocs]
   [-u|--unwind]
   [-d|--dynamic]
   [-V|--version-info]
   [-A|--arch-specific]
   [-D|--use-dynamic]
   [-x <number>|--hex-dump=<number>]
   [-w[liaprmfso]]|--debug-dump=[line,=info,=abbrev,=pubnames,=ranges,=macro,=frames,=frames-interp,=str,=loc]
   [-I|--histogram]
   [-v|--version]
   [-W|--wide]
   [-H|--help]
   elffile...

readelf displays information about one or more ELF format object files. The options control what particular information to display.

elffile... are the object files to be examined. At the moment, readelf does not support examining archives, nor does it support examining 64 bit ELF files.

The long and short forms of options, shown here as alternatives, are equivalent. At least one option besides -v or -H must be given.

-a
 -all
   Equivalent to specifying -file-header, -program-headers, -sections, -symbols, -relocs, -dynamic, -notes and -version-info.

-h
 -file-header
   Displays the information contained in the ELF header at the start of the file.

-l
 -program-headers
 -segments
   Displays the information contained in the file’s segment headers, if it has any.

-S
 -sections
 -section-headers
   Displays the information contained in the file’s section headers, if it has any.
-s
-symbols
-syms

Displays the entries in symbol table section of the file, if it has one.

-e
-headers

Display all the headers in the file. Equivalent to -h -l -s.

-n
-notes

Displays the contents of the NOTE segment, if it exists.

-r
-relocs

Displays the contents of the file’s relocation section, if it has one.

-u
-unwind

Displays the contents of the file’s unwind section, if it has one. Only the unwind sections for IA64 ELF files are currently supported.

-d
-dynamic

Displays the contents of the file’s dynamic section, if it has one.

-V
-version-info

Displays the contents of the version sections in the file, if they exist.

-A
-arch-specific

Displays architecture-specific information in the file, if there is any.

-D
-use-dynamic

When displaying symbols, this option makes readelf use the symbol table in the file’s dynamic section, rather than the one in the symbols section.

-x <number>
-hex-dump=<number>

Displays the contents of the indicated section as a hexadecimal dump.

-w[liaprmfFso]
-debug-dump[=line,=info,=abbrev,=pubnames,=ranges,=macro,=frames,=frames-interp,=str]

Displays the contents of the debug sections in the file, if any are present. If one of the optional letters or words follows the switch then only data found in those specific sections will be dumped.
-I
- histogram

Display a histogram of bucket list lengths when displaying the contents of the symbol tables.

-v
- version

Display the version number of readelf.

-W
- wide

Don’t break output lines to fit into 80 columns. By default readelf breaks section header and segment listing lines for 64-bit ELF files, so that they fit into 80 columns. This option causes readelf to print each section header resp. each segment one a single line, which is far more readable on terminals wider than 80 columns.

-H
- help

Display the command line options understood by readelf.
You can specify two aspects of the target system to the GNU binary file utilities, each in several ways:

- the target
- the architecture

In the following summaries, the lists of ways to specify values are in order of decreasing precedence. The ways listed first override those listed later.

The commands to list valid values only list the values for which the programs you are running were configured. If they were configured with `--enable-targets=all`, the commands list most of the available values, but a few are left out; not all targets can be configured in at once because some of them can only be configured `native` (on hosts with the same type as the target system).

### 16.1. Target Selection

A target is an object file format. A given target may be supported for multiple architectures (Section 16.2 Architecture Selection). A target selection may also have variations for different operating systems or architectures.

The command to list valid target values is `objdump -i` (the first column of output contains the relevant information).

Some sample values are: `a.out-hp300bsd`, `ecoff-littlemips`, `a.out-sunos-big`.

You can also specify a target using a configuration triplet. This is the same sort of name that is passed to `configure` to specify a target. When you use a configuration triplet as an argument, it must be fully canonicalized. You can see the canonical version of a triplet by running the shell script `config.sub` which is included with the sources.

Some sample configuration triplets are: `m68k-hp-bsd`, `mips-dec-ultrix`, `sparc-sun-sunos`.

#### 16.1.1. objdump Target

Ways to specify:

1. command line option: `-b` or `-target`
2. environment variable `GNUTARGET`
3. deduced from the input file

#### 16.1.2. objcopy and strip Input Target

Ways to specify:

1. command line options: `--input-target`, or `-F` or `-target`
2. environment variable `GNUTARGET`
3. deduced from the input file
16.1.3. `objcopy` and `strip` Output Target

Ways to specify:

1. command line options: `-O` or `-output-target`, or `-F` or `-target`
2. the input target (see "`objcopy` and `strip` Input Target" above)
3. environment variable `GNUTARGET`
4. deduced from the input file

16.1.4. `nm`, `size`, and `strings` Target

Ways to specify:

1. command line option: `-target`
2. environment variable `GNUTARGET`
3. deduced from the input file

16.2. Architecture Selection

An architecture is a type of cpu on which an object file is to run. Its name may contain a colon, separating the name of the processor family from the name of the particular cpu.

The command to list valid architecture values is `objdump -i` (the second column contains the relevant information).

Sample values: `m68k:68020`, `mips:3000`, `sparc`.

16.2.1. `objdump` Architecture

Ways to specify:

1. command line option: `-m` or `-architecture`
2. deduced from the input file

16.2.2. `objcopy`, `nm`, `size`, `strings` Architecture

Ways to specify:

1. deduced from the input file
Chapter 17.

Reporting Bugs

Your bug reports play an essential role in making the binary utilities reliable. Reporting a bug may help you by bringing a solution to your problem, or it may not. But in any case the principal function of a bug report is to help the entire community by making the next version of the binary utilities work better. Bug reports are your contribution to their maintenance.

In order for a bug report to serve its purpose, you must include the information that enables us to fix the bug.

17.1. Have You Found a Bug?

If you are not sure whether you have found a bug, here are some guidelines:

- If a binary utility gets a fatal signal, for any input whatever, that is a bug. Reliable utilities never crash.
- If a binary utility produces an error message for valid input, that is a bug.
- If you are an experienced user of binary utilities, your suggestions for improvement are welcome in any case.

17.2. How to Report Bugs

A number of companies and individuals offer support for gnu products. If you obtained the binary utilities from a support organization, we recommend you contact that organization first.

You can find contact information for many support companies and individuals in the file etc/SERVICE in the gnu Emacs distribution.

In any event, we also recommend that you send bug reports for the binary utilities to bug-binutils@gnu.org.

The fundamental principle of reporting bugs usefully is this: report all the facts. If you are not sure whether to state a fact or leave it out, state it!

Often people omit facts because they think they know what causes the problem and assume that some details do not matter. Thus, you might assume that the name of a file you use in an example does not matter. Well, probably it does not, but one cannot be sure. Perhaps the bug is a stray memory reference which happens to fetch from the location where that pathname is stored in memory; perhaps, if the pathname were different, the contents of that location would fool the utility into doing the right thing despite the bug. Play it safe and give a specific, complete example. That is the easiest thing for you to do, and the most helpful.

Keep in mind that the purpose of a bug report is to enable us to fix the bug if it is new to us. Therefore, always write your bug reports on the assumption that the bug has not been reported previously.

Sometimes people give a few sketchy facts and ask, "Does this ring a bell?" This cannot help us fix a bug, so it is basically useless. We respond by asking for enough details to enable us to investigate. You might as well expedite matters by sending them to begin with.

To enable us to fix the bug, you should include all these things:

- The version of the utility. Each utility announces it if you start it with the -version argument.
Chapter 17. Reporting Bugs

Without this, we will not know whether there is any point in looking for the bug in the current version of the binary utilities.

• Any patches you may have applied to the source, including any patches made to the BFD library.

• The type of machine you are using, and the operating system name and version number.

• What compiler (and its version) was used to compile the utilities--e.g. "gcc-2.7".

• The command arguments you gave the utility to observe the bug. To guarantee you will not omit something important, list them all. A copy of the Makefile (or the output from make) is sufficient.

If we were to try to guess the arguments, we would probably guess wrong and then we might not encounter the bug.

• A complete input file, or set of input files, that will reproduce the bug. If the utility is reading an object file or files, then it is generally most helpful to send the actual object files, uuencoded if necessary to get them through the mail system. Note that bug-binutils@gnu.org is a mailing list, so you should avoid sending very large files to it. Making the files available for anonymous FTP is OK.

If the source files were produced exclusively using gnu programs (e.g., gcc, gas, and/or the gnu ld), then it may be OK to send the source files rather than the object files. In this case, be sure to say exactly what version of gcc, or whatever, was used to produce the object files. Also say how gcc, or whatever, was configured.

• A description of what behavior you observe that you believe is incorrect. For example, "It gets a fatal signal."

Of course, if the bug is that the utility gets a fatal signal, then we will certainly notice it. But if the bug is incorrect output, we might not notice unless it is glaringly wrong. You might as well not give us a chance to make a mistake.

Even if the problem you experience is a fatal signal, you should still say so explicitly. Suppose something strange is going on, such as your copy of the utility is out of synch, or you have encountered a bug in the C library on your system. (This has happened!) Your copy might crash and ours would not. If you told us to expect a crash, then when ours fails to crash, we would know that the bug was not happening for us. If you had not told us to expect a crash, then we would not be able to draw any conclusion from our observations.

• If you wish to suggest changes to the source, send us context diffs, as generated by diff with the -u, -c, or -p option. Always send diffs from the old file to the new file. If you wish to discuss something in the ld source, refer to it by context, not by line number.

The line numbers in our development sources will not match those in your sources. Your line numbers would convey no useful information to us.

Here are some things that are not necessary:

• A description of the envelope of the bug.

Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it.

This is often time consuming and not very useful, because the way we will find the bug is by running a single example under the debugger with breakpoints, not by pure deduction from a series of examples. We recommend that you save your time for something else.

Of course, if you can find a simpler example to report instead of the original one, that is a convenience for us. Errors in the output will be easier to spot, running under the debugger will take less time, and so on.

However, simplification is not vital; if you do not want to do this, report the bug anyway and send us the entire test case you used.
• A patch for the bug.

A patch for the bug does help us if it is a good one. But do not omit the necessary information, such as the test case, on the assumption that a patch is all we need. We might see problems with your patch and decide to fix the problem another way, or we might not understand it at all.

Sometimes with programs as complicated as the binary utilities it is very hard to construct an example that will make the program follow a certain path through the code. If you do not send us the example, we will not be able to construct one, so we will not be able to verify that the bug is fixed.

And if we cannot understand what bug you are trying to fix, or why your patch should be an improvement, we will not install it. A test case will help us to understand.

• A guess about what the bug is or what it depends on.

Such guesses are usually wrong. Even we cannot guess right about such things without first using the debugger to find the facts.
Appendix A.

GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you."

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.)

The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque."

Examples of suitable formats for Transparent copies include plain ASCII without markup, TeXinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated
version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sec-
tions 2 and 3 above, provided that you release the Modified Version under precisely this License,
with the Modified Version filling the role of the Document, thus licensing distribution and mod-
ification of the Modified Version to whoever possesses a copy of it. In addition, you must do
these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and
from those of previous versions (which should, if there were any, be listed in the History section
of the Document). You may use the same title as a previous version if the original publisher
of that version gives permission. B. List on the Title Page, as authors, one or more persons or
entities responsible for authorship of the modifications in the Modified Version, together with at
least five of the principal authors of the Document (all of its principal authors, if it has less than
five). C. State on the Title page the name of the publisher of the Modified Version, as the pub-
lisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright
notice for your modifications adjacent to the other copyright notices. F. Include, immediately
after the copyright notices, a license notice giving the public permission to use the Modified
Version under the terms of this License, in the form shown in the Addendum below. G. Preserve
in that license notice the full lists of Invariant Sections and required Cover Texts given in the
Document’s license notice. H. Include an unaltered copy of this License. I. Preserve the section
entitled "History", and its title, and add to it an item stating at least the title, year, new authors,
and publisher of the Modified Version as given on the Title Page. If there is no section entitled
"History" in the Document, create one stating the title, year, authors, and publisher of the Doc-
ument as given on its Title Page, then add an item describing the Modified Version as stated in
the previous sentence. J. Preserve the network location, if any, given in the Document for public
access to a Transparent copy of the Document, and likewise the network locations given in the
Document for previous versions it was based on. These may be placed in the “History” sec-
tion. You may omit a network location for a work that was published at least four years before
the Document itself, or if the original publisher of the version it refers to gives permission. K.
In any section entitled "Acknowledgements" or "Dedications", preserve the section’s title, and
preserve in the section all the substance and tone of each of the contributor acknowledgements
and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unal-
tered in their text and in their titles. Section numbers or the equivalent are not considered part
of the section titles. M. Delete any section entitled "Endorsements." Such a section may not be
included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or
to conflict in title with any Invariant Section. If the Modified Version includes new front-matter
sections or appendices that qualify as Secondary Sections and contain no material copied from
the Document, you may at your option designate some or all of these sections as invariant. To
do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice.
These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements
of your Modified Version by various parties—for example, statements of peer review or that the
text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25
words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only
one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through
arrangements made by) any one entity. If the Document already includes a cover text for the
same cover, previously added by you or by arrangement made by the same entity you are acting
on behalf of, you may not add another; but you may replace the old one, on explicit permission
from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use
their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS
You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications." You must delete all sections entitled "Endorsements."

7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11. FUTURE REVISIONS OF THIS LICENSE
The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being list their titles, with the Front-Cover Texts being list, and with the Back-Cover Texts being list. A copy of the license is included in the section entitled "GNU Free Documentation License."

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being list"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.
Index

.stab, see Chapter 5 objdump

A

addr2line, see Chapter 11 addr2line
address to file name and line number, see Chapter 11 addr2line
all header information, object file, see Chapter 5 objdump
ar, see Chapter 2 ar
ar compatibility, see Chapter 2 ar
architecture, see Chapter 5 objdump
architectures available, see Chapter 5 objdump
archive contents, see Chapter 6 ranlib
archive headers, see Chapter 5 objdump
archives, see Chapter 2 ar

B

base files, see Chapter 14 Create files needed to build and use DLLs
bug criteria, see Section 17.1 Have You Found a Bug?
bug reports, see Section 17.2 How to Report Bugs
bugs, see Chapter 17 Reporting Bugs
bugs, reporting, see Section 17.2 How to Report Bugs

c++filt, see Chapter 10 c++filt
changing object addresses, see Chapter 4 objccopy
changing section address, see Chapter 4 objccopy
changing section LMA, see Chapter 4 objccopy
changing section VMA, see Chapter 4 objccopy
changing start address, see Chapter 4 objccopy
collections of files, see Chapter 2 ar
compatibility, ar, see Chapter 2 ar
contents of archive, see Section 2.1 Controlling axon the Command Line

C

cxxfilt, see Chapter 10 c++filt
dates in archive, see Section 2.1 Controlling axon the Command Line
dynamic relocation entries, in object file, see Chapter 5 objdump
dynamic symbol table entries, printing, see Chapter 5 objdump
dynamic symbols, see Chapter 3 nm

D

dates in archive, see Section 2.1 Controlling axon the Command Line
demangling C++ symbols, see Chapter 10 c++filt
demangling in nm, see Chapter 3 nm
demangling in objdump, see Chapter 11 addr2line
demangling in objdump, see Chapter 5 objdump
disassembling object code, see Chapter 5 objdump
disassembling architecture, see Chapter 5 objdump
disassembly endianness, see Chapter 5 objdump
disassembly, with source, see Chapter 5 objdump
discarding symbols, see Chapter 9 strip
DLL, see Chapter 14 Create files needed to build and use DLLs
dlltool, see Chapter 14 Create files needed to build and use DLLs
dynamic relocation entries, in object file, see Chapter 5 objdump
dynamic symbol table entries, printing, see Chapter 5 objdump
dynamic symbols, see Chapter 3 nm

E

ELF core notes, see Chapter 15 readelf
ELF dynamic section information, see Chapter 15 readelf
ELF file header information, see Chapter 15 readelf
ELF file information, see Chapter 15 readelf
ELF object file format, see Chapter 5 objdump
ELF program header information, see Chapter 15 readelf
ELF reloc information, see Chapter 15 readelf
ELF section information, see Chapter 15 readelf
ELF segment information, see Chapter 15 readelf
ELF symbol table information, see Chapter 15 readelf
ELF version sections informations, see Chapter 15 readelf
endianness, see Chapter 5 objdump
error on valid input, see Section 17.1 Have You Found a Bug?
external symbols, see Chapter 3 nm
extract from archive, see Section 2.1 Controlling axon the Command Line

F

fatal signal, see Section 17.1 Have You Found a Bug?
file name, see Chapter 3 nm
H

header information, all, see Chapter 5 objdump

I

input .def file, see Chapter 14 Create files needed to build and use DLLs
input file name, see Chapter 3 nm

L

libraries, see Chapter 2 ar
listings strings, see Chapter 8 strings

M

machine instructions, see Chapter 5 objdump
moving in archive, see Section 2.1 Controlling ar on the Command Line
MRI compatibility, ar, see Section 2.2 Controlling ar with a Script

N

name duplication in archive, see Section 2.1 Controlling ar on the Command Line
name length, see Chapter 2 ar
nm, see Chapter 3 nm
nm compatibility, see Chapter 3 nm
nm format, see Chapter 3 nm
not writing archive index, see Section 2.1 Controlling ar on the Command Line

O

objdump, see Chapter 5 objdump
object code format, see Chapter 11 addr2line
object code format, see Chapter 8 strings
object code format, see Chapter 7 size
object code format, see Chapter 5 objdump
object code format, see Chapter 3 nm
object file header, see Chapter 5 objdump
object file information, see Chapter 5 objdump
object file sections, see Chapter 5 objdump
object formats available, see Chapter 5 objdump
operations on archive, see Section 2.1 Controlling ar on the Command Line

P

printing from archive, see Section 2.1 Controlling ar on the Command Line
printing strings, see Chapter 8 strings

Q

quick append to archive, see Section 2.1 Controlling ar on the Command Line

R

radix for section sizes, see Chapter 7 size
ranlib, see Chapter 6 ranlib
readelf, see Chapter 15 readelf
relative placement in archive, see Section 2.1 Controlling ar on the Command Line
relocation entries, in object file, see Chapter 5 objdump
removing symbols, see Chapter 9 strip
repeated names in archive, see Section 2.1 Controlling ar on the Command Line
replacement in archive, see Section 2.1 Controlling ar on the Command Line
reporting in archive, see Section 2.1 Controlling ar on the Command Line
reporting bugs, see Chapter 17 Reporting Bugs

S

scripts, ar, see Section 2.2 Controlling ar with a Script
section addresses in objdump, see Chapter 5 objdump
section headers, see Chapter 5 objdump
section information, see Chapter 5 objdump
section sizes, see Chapter 7 size
sections, full contents, see Chapter 5 objdump
size, see Chapter 7 size
size display format, see Chapter 7 size
size number format, see Chapter 7 size
sorting symbols, see Chapter 3 nm
source code context, see Chapter 5 objdump
source disassembly, see Chapter 5 objdump
source file name, see Chapter 3 nm
source filenames for object files, see Chapter 5 objdump
stab, see Chapter 5 objdump
start-address, see Chapter 5 objdump
stop-address, see Chapter 5 objdump
strings, see Chapter 8 strings
strings, printing, see Chapter 8 strings
strip, see Chapter 9 strip
symbol index, see Chapter 6 ranlib
symbol index, see Chapter 2 ar
symbol index, listing, see Chapter 3 nm
symbol line numbers, see Chapter 3 nm
symbol table entries, printing, see Chapter 5 \textit{objdump}
symbols, see Chapter 3 \textit{nm}
symbols, discarding, see Chapter 9 \textit{strip}

\textbf{U}

undefined symbols, see Chapter 3 \textit{nm}
Unix compatibility, \texttt{ar}, see Section 2.1 Controlling \texttt{ar} on the Command Line
unwind information, see Chapter 15 \texttt{readelf}
updating an archive, see Section 2.1 Controlling \texttt{ar} on the Command Line

\textbf{V}

version, see Chapter 1 \textit{Introduction}
VMA in \texttt{objdump}, see Chapter 5 \texttt{objdump}

\textbf{W}

wide output, printing, see Chapter 5 \texttt{objdump}
writing archive index, see Section 2.1 Controlling \texttt{ar} on the Command Line