The GNU Binary Utilities

Version 2.8.1

May 1993

Roland H. Pesch
Jeffrey M. Osier
Cygnus Support

Cygnus Support
TEXinfo 2.185

Introduction

This brief manual contains preliminary documentation for the GNU binary utilities (collectively version 2.8.1):

`ar` Create, modify, and extract from archives

`nm` List symbols from object files

`objcopy` Copy and translate object files

`objdump` Display information from object files

`ranlib` Generate index to archive contents

`size` List file section sizes and total size

`strings` List printable strings from files

`strip` Discard symbols

`c++filt` Demangle encoded C++ symbols

`addr2line` Convert addresses into file names and line numbers

`nlmconv` Convert object code into a Netware Loadable Module

1 ar

```
ar [-]p[mod [relpos]] archive [member...]
ar -M [ <mri-script ]
```

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

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

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

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

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

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

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

1.1 Controlling ar on the command line

`ar [-]`*p*`[`*mod* `[`*relpos*`]]` *archive* `[`*member*...`]`

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

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

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

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

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

`d`
: *Delete* modules from the archive. Specify the names of modules to be deleted as *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.

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

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

  If no modifiers are used with `m`, any members you name in the *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.

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

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

`q`
: *Quick append*; 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.

`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.

`t` *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 *relpos* 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 *relpos* 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 *relpos* argument, before the *archive* specification. (same as '`b`').

`l` This modifier is accepted but not used.

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

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

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

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

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

1.2 Controlling ar with a script

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2 ld

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

3 nm

```
nm [ -a | --debug-syms ]  [ -g | --extern-only ]
   [ -B ]  [ -C | --demangle ] [ -D | --dynamic ]
   [ -s | --print-armap ]  [ -A | -o | --print-file-name ]
   [ -n | -v | --numeric-sort ]  [ -p | --no-sort ]
   [ -r | --reverse-sort ]  [ --size-sort ] [ -u | --undefined-only ]
   [ -t radix | --radix=radix ] [ -P | --portability ]
   [ --target=bfdname ] [ -f format | --format=format ]
   [ --defined-only ] [-l | --line-numbers ]
   [ --no-demangle ] [ -V | --version ]  [ --help ]  [ objfile... ]
```

GNU **nm** lists the symbols from object files *objfile*.... If no object files are listed as arguments, **nm** assumes '`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 section "Linker options" in *The GNU linker*.

`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 which 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.

`W` The symbol is weak. 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 section "Stabs Overview" in *The "stabs" debug format*.

? 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`
`-o`
`--print-file-name`
Precede each symbol by the name of the input file (or archive element) 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`
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. See Chapter 10 [c++filt], page 31, 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-armap`
When listing symbols from archive members, include the index: a mapping (stored in the archive by `ar` or `ranlib`) of which modules contain definitions for which names.

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

`--size-sort`
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. The size of the symbol is printed, rather than the value.

`-t` *radix*
`--radix=`*radix*
Use *radix* as the radix for printing the symbol values. It must be '`d`' for decimal, '`o`' for octal, or '`x`' for hexadecimal.

`--target=`*bfdname*
Specify an object code format other than your system's default format. See Section 13.1 [Target Selection], page 37, for more information.

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

`--defined-only`
Display only defined symbols for each object file.

`-V`
`--version`
Show the version number of `nm` and exit.

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

4 objcopy

```
objcopy [ -F bfdname | --target=bfdname ]
        [ -I bfdname | --input-target=bfdname ]
        [ -O bfdname | --output-target=bfdname ]
        [ -S | --strip-all ]  [ -g | --strip-debug ]
        [ -K symbolname | --keep-symbol=symbolname ]
        [ -N symbolname | --strip-symbol=symbolname ]
        [ -x | --discard-all ]  [ -X | --discard-locals ]
        [ -b byte | --byte=byte ]
        [ -i interleave | --interleave=interleave ]
        [ -R sectionname | --remove-section=sectionname ]
        [ -p | --preserve-dates ] [ --debugging ]
        [ --gap-fill=val ] [ --pad-to=address ]
        [ --set-start=val ] [ --adjust-start=incr ]
        [ --adjust-vma=incr ]
        [ --adjust-section-vma=section{=,+,-}val ]
        [ --adjust-warnings ] [ --no-adjust-warnings ]
        [ --set-section-flags=section=flags ]
        [ --add-section=sectionname=filename ]
        [ --change-leading-char ] [ --remove-leading-char ]
        [ --weaken ]
        [ -v | --verbose ] [ -V | --version ]  [ --help ]
        infile [outfile]
```

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

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

`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 which is not needed by the binary file.

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

`-I` *bfdname*
`--input-target=`*bfdname*
: Consider the source file's object format to be *bfdname*, rather than attempting to deduce it. See Section 13.1 [Target Selection], page 37, for more information.

`-O` *bfdname*
`--output-target=`*bfdname*
: Write the output file using the object format *bfdname*. See Section 13.1 [Target Selection], page 37, for more information.

`-F` *bfdname*
`--target=`*bfdname*
: Use *bfdname* as the object format for both the input and the output file; i.e., simply transfer data from source to destination with no translation. See Section 13.1 [Target Selection], page 37, for more information.

`-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*
`--strip-symbol=`*symbolname*
: Do not copy symbol *symbolname* from the source file. This option may be given more than once, and may be combined with strip options other than `-K`.

`-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*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 '`-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 debugging formats are supported, and the conversion process can be time consuming.

`--gap-fill` *val*
Fill gaps between sections with *val*. This 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 virtual 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 address of the new file to *val*. Not all object file formats support setting the start address.

`--adjust-start` *incr*
Adjust the start address by adding *incr*. Not all object file formats support setting the start address.

`--adjust-vma` *incr*
Adjust the address 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.

`--adjust-section-vma` *section*`{=,+,-}`*val*
Set or adjust the 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 '`--adjust-vma`', above. If *section* does not exist in the input file, a warning will be issued, unless '`--no-adjust-warnings`' is used.

`--adjust-warnings`
: If '`--adjust-section-vma`' is used, and the named section does not exist, issue a warning. This is the default.

`--no-adjust-warnings`
: Do not issue a warning if '`--adjust-section-vma`' is used, even if the named section does not exist.

`--set-section-flags` *section=flags*
: Set the flags for the named section. The *flags* argument is a comma separated string of flag names. The recognized names are '`alloc`', '`load`', '`readonly`', '`code`', '`data`', and '`rom`'. Not all flags are meaningful for all object file formats.

`--add-section` *sectionname=filename*
: Add a new section named *sectionname* while copying the file. The contents of the new section are taken from the file *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.

`--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 `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.

`--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.

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

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

`--help`
: Show a summary of the options to `objcopy`.

5 objdump

```
objdump [ -a | --archive-headers ]
        [ -b bfdname | --target=bfdname ] [ --debugging ]
        [ -C | --demangle ] [ -d | --disassemble ]
        [ -D | --disassemble-all ] [ --disassemble-zeroes ]
        [ -EB | -EL | --endian={big | little } ]
        [ -f | --file-headers ]
        [ -h | --section-headers | --headers ]  [ -i | --info ]
        [ -j section | --section=section ]
        [ -l | --line-numbers ] [ -S | --source ]
        [ -m machine | --architecture=machine ]
        [ -r | --reloc ] [ -R | --dynamic-reloc ]
        [ -s | --full-contents ]  [ --stabs ]
        [ -t | --syms ] [ -T | --dynamic-syms ] [ -x | --all-headers ]
        [ -w | --wide ] [ --start-address=address ]
        [ --stop-address=address ]
        [ --prefix-addresses] [ --[no-]show-raw-insn ]
        [ --adjust-vma=offset ]
        [ --version ]  [ --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 besides '`-l`' 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. See Section 13.1 [Target Selection], page 37, for more information.

`-C`
`--demangle`
: 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. See Chapter 10 [c++filt], page 31, for more information on demangling.

`--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.

`-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.

`--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.

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

`-h`
`--section-header`
`--header`
: 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.

`--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 disasembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the '-i' option.

`-r`
`--reloc` Print the relocation entries of the file. If used with '-d' or '-D', the relocations are printed interspersed 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.

`--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 section "Stabs Overview" in *The "stabs" debug format*.

`--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.

`--version`
: Print the version number of `objdump` and exit.

`-x`
`--all-header`
: 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.

6 ranlib

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

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

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

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

The GNU `ranlib` program is another form of GNU `ar`; running `ranlib` is completely equivalent to executing '`ar -s`'. See Chapter 1 [ar], page 3.

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

7 size

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

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

objfile... are the object files to be examined.

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.

`-d`
`-o`
`-x`
`--radix=`*number*

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

`--target=`*bfdname*

Specify that the object-code format for *objfile* is *bfdname*. This option may not be necessary; `size` can automatically recognize many formats. See Section 13.1 [Target Selection], page 37, for more information.

`-V`
`--version`

Display the version number of `size`.

8 strings

```
strings [-afov] [-min-len] [-n min-len] [-t radix] [-]
        [--all] [--print-file-name] [--bytes=min-len]
        [--radix=radix] [--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.

`--target=`*bfdname*
: Specify an object code format other than your system's default format. See Section 13.1 [Target Selection], page 37, for more information.

`-v`
`--version`
: Print the program version number on the standard output and exit.

9 strip

```
strip [ -F bfdname | --target=bfdname | --target=bfdname ]
      [ -I bfdname | --input-target=bfdname ]
      [ -O bfdname | --output-target=bfdname ]
      [ -s | --strip-all ] [ -S | -g | --strip-debug ]
      [ -K symbolname | --keep-symbol=symbolname ]
      [ -N symbolname | --strip-symbol=symbolname ]
      [ -x | --discard-all ] [ -X | --discard-locals ]
      [ -R sectionname | --remove-section=sectionname ]
      [ -o file ] [ -p | --preserve-dates ]
      [ -v | --verbose ]  [ -V | --version ]  [ --help ]
      objfile...
```

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. See Section 13.1 [Target Selection], page 37, for more information.

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

`-I` *bfdname*
`--input-target=`*bfdname*
: Treat the original *objfile* as a file with the object code format *bfdname*. See Section 13.1 [Target Selection], page 37, for more information.

`-O` *bfdname*
`--output-target=`*bfdname*
: Replace *objfile* with a file in the output format *bfdname*. See Section 13.1 [Target Selection], page 37, for more information.

`-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`
: Remove all symbols.

`-g`
`-S`
`--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.

10 c++filt

```
c++filt [ -_ | --strip-underscores ]
[ -n | --no-strip-underscores ]
        [ -s format | --format=format ]
        [ --help ]  [ --version ]  [ symbol... ]
```

The C++ language provides function overloading, which means that you can write many functions with the same name (providing each takes parameters of different types). All C++ 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.

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

`-s` *format*
`--format=`*format*
: GNU `nm` can decode three different methods of mangling, used by different C++ compilers. The argument to this option selects which method it uses:

  `gnu` the one used by the GNU compiler (the default method)

  `lucid` the one used by the Lucid compiler

  `arm` the one specified by the C++ Annotated Reference Manual

`--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
```

11 addr2line

```
addr2line [ -b bfdname | --target=bfdname ]
          [ -C | --demangle ]
          [ -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 '`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`
: 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. See Chapter 10 [c++filt], page 31, 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.

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 section "BFD" in *Using LD*, 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). See Section 13.1 [Target Selection], page 37, 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`'. See Section 13.1 [Target Selection], page 37, 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*
`--linker=`*linker*
: Use *linker* for any linking. *linker* can be an abosolute or a relative pathname.

`-h`
`--help`
: Prints a usage summary.

`-V`
`--version`
Prints the version number for `nlmconv`.

13 Selecting the target system

You can specify three aspects of the target system to the GNU binary file utilities, each in several ways:

- the target
- the architecture
- the linker emulation (which applies to the linker only)

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).

13.1 Target Selection

A *target* is an object file format. A given target may be supported for multiple architectures (see Section 13.2 [Architecture Selection], page 38). 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`'.

`objdump` Target

Ways to specify:

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

`objcopy` and `strip` Input Target

Ways to specify:

1. command line options: '`-I`' or '`--input-target`', or '`-F`' or '`--target`'
2. environment variable `GNUTARGET`
3. deduced from the input file

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

nm, size, and strings Target

Ways to specify:

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

Linker Input Target

Ways to specify:

1. command line option: '-b' or '--format' (see section "Options" in *Using LD*)
2. script command TARGET (see section "Option Commands" in *Using LD*)
3. environment variable GNUTARGET (see section "Environment" in *Using LD*)
4. the default target of the selected linker emulation (see Section 13.3 [Linker Emulation Selection], page 39)

Linker Output Target

Ways to specify:

1. command line option: '-oformat' (see section "Options" in *Using LD*)
2. script command OUTPUT_FORMAT (see section "Option Commands" in *Using LD*)
3. the linker input target (see "Linker Input Target" above)

13.2 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'.

objdump Architecture

Ways to specify:

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

`objcopy`, `nm`, `size`, `strings` Architecture

Ways to specify:

1. deduced from the input file

Linker Input Architecture

Ways to specify:

1. deduced from the input file

Linker Output Architecture

Ways to specify:

1. script command `OUTPUT_ARCH` (see section "Option Commands" in *Using LD*)
2. the default architecture from the linker output target (see Section 13.1 [Target Selection], page 37)

13.3 Linker emulation selection

A linker *emulation* is a "personality" of the linker, which gives the linker default values for the other aspects of the target system. In particular, it consists of

- the linker script
- the target
- several "hook" functions that are run at certain stages of the linking process to do special things that some targets require

The command to list valid linker emulation values is '`ld -V`'.

Sample values: '`hp300bsd`', '`mipslit`', '`sun4`'.

Ways to specify:

1. command line option: '`-m`' (see section "Options" in *Using LD*)
2. environment variable `LDEMULATION`
3. compiled-in `DEFAULT_EMULATION` from '`Makefile`', which comes from `EMUL` in '`config/`*target*`.mt`'

14 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.

14.1 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.

14.2 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-gnu-utils@prep.ai.mit.edu`'.

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?" Those bug reports are useless, and we urge everyone to *refuse to respond to them* except to chide the sender to report bugs properly.

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.

  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. Making them available for anonymous FTP is not as good, but may be the only reasonable choice for large object files.

  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 even 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.

Index

Table of Contents